Digital Forensics: Increasing the Evidential Weight of System Activity Logs

Atif Ahmad

Submitted in total fulfillment of the requirements of the degree of Doctor of Philosophy

April 2007

Department of Information Systems
The University of Melbourne
Abstract

The application of investigative techniques within digital environments has lead to the emergence of a new field of specialization that may be termed ‘digital forensics’. Perhaps the primary challenge concerning digital forensic investigations is how to preserve evidence of system activity given the volatility of digital environments and the delay between the time of the incident and the start of the forensic investigation.

This thesis hypothesizes that system activity logs present in modern operating systems may be used for digital forensic evidence collection. This is particularly true in modern organizations where there is growing recognition that forensic readiness may have considerable benefits in case of future litigation.

An investigation into the weighting of evidence produced by system activity logs present in modern operating systems takes place in this thesis. The term ‘evidential weight’ is used loosely as a measure of the suitability of system activity logs to digital forensic investigations. This investigation is approached from an analytical perspective. The first contribution of this thesis is to determine the evidence collection capability of system activity logs by a simple model of the logging mechanism. The second contribution is the development of evidential weighting criteria that can be applied to system activity logs. A unique and critical role for system activity logs by which they establish the reliability of other kinds of computer-derived evidence from hard disk media is also identified.

The primary contribution of this thesis is the identification of a comprehensive range of forensic weighting issues arising from the use of log evidence that concern investigators and legal authorities. This contribution is made in a comprehensive analytical discussion utilizing both the logging model and the evidential weighting criteria. The practical usefulness of the resulting evidential weighting framework is demonstrated by rigorous and systematic application to a real-world logging system.
Declaration

This is to certify that

(i) the thesis comprises only my original work towards the PhD except where indicated in the Preface,

(ii) due acknowledgement has been made in the text to all other material used,

(iii) the thesis is less than 100,000 words in length, exclusive of tables, maps, bibliographies and appendices.

Atif Ahmad
Preface

The detection/selection/description model in chapter 4 is derived from a paper published at ISOOneWorld authored by myself and A.B. (Tobias) Ruighaver (Ahmad & Ruighaver, 2003b). In this paper, a distinction is made between the ‘Gathering’ and ‘Managing’ phases of system activity logs. This distinction is reflected in figure 5.1 that differentiates between a phase where incident-related evidence is written to the log, and a subsequent phase where event log evidence is managed in the digital environment. In the same paper (figure 3), I present the selection/detection model that forms the basis of chapter 4.

Chapter 5 is based on a paper submitted to the Australian Computer, Network, Information and Forensics Conference in 2004 (Ahmad & Ruighaver, 2004). This paper develops the evidential weighting criteria for system activity logs from the existing human-directed framework found in literature. It also identifies ‘utility’ as a major contribution of system activity logs towards evidential weighting.

Various other models utilized in this research have also been published.

The management of log evidence in terms of storage, reduction, and retention has been discussed in a paper titled ‘Top-Down Approach Towards Translating Organizational Security Policy Directives to System Audit Configuration’. This paper was published in the 2002 proceedings of IFIP (TC 11) Security held in Cairo, Egypt (Ahmad & Ruighaver, 2002).

The potentially critical role played by network-access events in forensic investigations and the fact that they are not currently loggable by system activity logs is covered in a paper presented in the proceedings of the 1st Australian Computer Network, Information and Forensics Conference (Ahmad & Ruighaver, 2003a).

The exploratory/deductive phases in forensic investigation were introduced in a paper presented in Japan titled ‘The Forensic Chain-of-Evidence Model: Improving the Process of Evidence Collection in Incident Handling Procedures’ (Ahmad 2002).
Note that a paper titled ‘Modeling System Activity Logging for Evidence Collection’ was published in the Journal of Information Warfare. In 2013, more than five years after the thesis passed examination.
Table of Contents

ABSTRACT ................................................................................................................. II
DECLARATION ............................................................................................................... III
PREFACE ..................................................................................................................... IV
ACKNOWLEDGEMENTS ................................................................................................ ERROR! BOOKMARK NOT DEFINED.
TABLE OF CONTENTS ................................................................................................ VI

CHAPTER 1: INTRODUCTION ...................................................................................... 1
  1.1 BRIEF BACKGROUND ............................................................................................... 1
  1.2 THESIS APPROACH ................................................................................................. 5
  1.3 RESEARCH METHODOLOGY ................................................................................... 6
  1.4 CONTRIBUTIONS ..................................................................................................... 8
  1.5 CRITICAL EVALUATION OF THIS INVESTIGATION .................................................. 9
  1.6 THESIS SCOPE ..................................................................................................... 12
  1.7 THESIS OVERVIEW ............................................................................................... 13

CHAPTER 2: AN OVERVIEW OF DIGITAL FORENSICS RESEARCH ............................. 16
  2.1 AN OVERVIEW OF DIGITAL FORENSICS LITERATURE .......................................... 17
  2.2 AN INTRODUCTION TO DIGITAL FORENSIC INVESTIGATIONS ............................ 20
      2.2.1 Terminology ...................................................................................................... 20
      2.2.2 Motivation ......................................................................................................... 23
      2.2.3 Parallels between Physical and Digital Forensic Investigations ...................... 25
      2.2.4 Process Models for Digital Forensic Investigations ......................................... 28
  2.3 THE EVIDENCE COLLECTION PROBLEM ............................................................. 32
      2.3.1 Sources of Evidence in Computing Systems ..................................................... 33
      2.3.2 Evidence Management in a Forensic-Ready Organization ................................ 35
      2.3.3 Preserving Evidence of System Activity ............................................................ 40
      2.3.4 Identifying Evidence Collection Requirements for Organizations ................. 41
  2.4 ADMISSIBILITY AND WEIGHT OF ELECTRONIC EVIDENCE ............................... 43
      2.4.1 General Tests for Submission of Evidence to Court .......................................... 44
      2.4.2 Overlap between Admissibility and Weight ...................................................... 45
      2.4.3 Admissibility of Evidence .................................................................................. 46
      2.4.4 Weight of Evidence ........................................................................................... 49
      2.4.5 Admissibility of Computer Log Files ................................................................. 49
      2.4.6 Desirable Features of Computer-Derived Evidence ......................................... 50
  2.5 SUMMARY ............................................................................................................. 53

CHAPTER 3: AN OVERVIEW OF EVENT LOG RESEARCH ........................................... 54
  3.1 EVENT LOGGING: A CHRONOLOGICAL RECORD OF SYSTEM ACTIVITY ............. 55
      3.1.1 Terminology ...................................................................................................... 55
      3.1.2 Simple Event Logging Models .......................................................................... 57
  3.2 AN OVERVIEW OF EVENT LOGGING LITERATURE ............................................... 59
  3.3 EVENT LOG MANAGEMENT ................................................................................... 61
      3.3.1 Event Selection .................................................................................................. 61
      3.3.2 Storage, Reduction and Retention ..................................................................... 62
      3.3.3 Log File Security ............................................................................................... 64
      3.3.4 Log File Integration ......................................................................................... 65
  3.4 INADEQUACIES IN SYSTEM ACTIVITY LOGGING ............................................. 66
      3.4.1 Inability to Log Some Types of System Activity ................................................. 67
      3.4.2 Inability to Log Specific Events to the Exclusion of Others ............................... 68
      3.4.3 Inadequate Characterization of Events .............................................................. 68
  3.5 SECURITY-MOTIVATED LOGGING ......................................................................... 69
      3.5.1 Security-Motivated Event Data Analysis ............................................................ 71
      3.5.2 Intrusion Detection ............................................................................................ 72
CHAPTER 7: MAXIMIZING THE EVIDENTIAL WEIGHT OF SYSTEM ACTIVITY LOGS

7.1 A PRIORITY MODEL FOR EVIDENTIAL WEIGHTING .......................................................168
7.2 PRIORITIZING EVIDENTIAL WEIGHTING ISSUES REGARDING SYSTEM ACTIVITY LOGS ......................................................171
7.2.1 Level 1: Establishing Relevance of Computer-derived Evidence ...................................171
7.2.2 Level 2: Evidential Reliability Tests ..................................................................................174
7.3 CONCLUSIONS ..............................................................................................................178

CHAPTER 8: APPLYING THE EVIDENTIAL WEIGHTING FRAMEWORK TO MICROSOFT-WINDOWS LOGGING .........................................................................................180
8.1 CONNECTING DIGITAL ACTORS TO REAL-WORLD ACTORS ............................................187
8.1.1 User Actor Identification in Individual Windows Log Entries ..........................................187
8.1.2 Application Actor Identification in Individual Windows Log Entries ................................193
8.1.3 File Actor Identification in Individual Windows Log Entries ..........................................196
8.1.4 User Activity Profiles for Improved Actor Identification .................................................199
8.1.5 Profiling Application Activity for Improved Actor Identification ....................................201
8.1.6 Profiling File Activity for Improved Actor Identification ...............................................204
8.1.7 Grouping Related Log Entries for Actor Profiling ...........................................................208
8.1.8 Complete Coverage of System Activity for Actor Identification ......................................210
8.1.9 Consistency of Evidence of Actor Identification .............................................................215
8.1.10 Reconstruction of Evidence of Actor Identification .......................................................218
8.1.11 Control Over the Generation of Evidence of Actor Identification ..................................221
8.1.12 Guidance on Selection of Evidence of Actor Identification ...........................................225
8.2 CONNECTING SYSTEM ACTIVITY TO REAL-WORLD ACTS ..........................................228
8.2.1 Accuracy of Action Identification in Individual Log Entries .............................................229
8.2.2 Accuracy of System Operation Logging .........................................................................232
8.2.3 Grouping Related Log Entries to Identify System Operations .........................................236
8.2.4 Complete Coverage of System Activity .........................................................................238
8.2.5 Consistency of Log Evidence Across Multiple Log Entries .............................................243
8.2.6 Complete Reconstruction of System Activity .................................................................245
8.2.7 Control over Generation of Evidence of System Activity ................................................247
8.2.8 Guidance on Selection of Evidence of Activity Identification .........................................249
8.3 CONNECTING TIMING OF SYSTEM ACTIVITY TO REAL-WORLD TIME ........................................250
8.3.1 Accuracy of Individual Log Entry Timestamp .................................................................252
8.3.2 Accurate Identification of a Sequence of Operations .......................................................257
8.3.3 Accurate Identification of an Incident Timeframe ............................................................262
8.3.4 Grouping Related Log Entries to Identify System Operations .........................................264
8.3.5 Complete Coverage of System Activity .........................................................................265
8.3.6 Consistency of Log Evidence Across Multiple Log Entries .............................................267
8.3.7 Complete Reconstruction of System Activity .................................................................268
8.3.8 Control Over the Generation of Timing Information Regarding System Activity ................270
8.3.9 Guidance on the Selection of Timing Information ...........................................................271
8.4 LOG MANAGEMENT .....................................................................................................271
8.4.1 Security of Windows Event Logging ................................................................................271
8.4.2 Retention Policies of Windows Logging ..........................................................................276
8.5 FINAL ASSESSMENT ......................................................................................................277
8.5.1 Usefulness of Weighting Criteria ....................................................................................278
8.5.2 Recommendations on Windows for Systems Administrators and OS Designers .................280
8.5.3 Useful Characteristics of this Framework .........................................................................281

CHAPTER 9: CONCLUSION ....................................................................................................283
9.1 Contributions .................................................................................................................284
9.2 Limitations and Future Work .........................................................................................287

REFERENCES .......................................................................................................................290

APPENDICES A-1 TO A-4 ....................................................................................................298
Chapter 1

Introduction

1.1 Brief Background

Given the widespread availability of digital technology and its usage in almost every facet of contemporary society, it is unsurprising that criminal, civil and even organizational forensic investigations are increasingly involving computing systems. The application of investigative techniques within digital environments has lead to the emergence of a new and relatively immature field of specialization. Evidence of the immaturity of this field can be seen from existing debate into its most fundamental concepts such as basic terminology, motivation and scope of application. After much consideration, the term ‘digital forensics’ has been chosen to refer to this field (see section 2.2.1 for a discussion on this topic).

It is important to recognize that digital forensics is a new science rooted in investigative practice. This discipline has evolved from adhoc procedures rather than classical scientific development. This is reflected in almost all definitions in literature that define this field as a series of practices or procedures aimed at determining potential legal evidence.

The rapidly evolving nature of modern digital technology and the constantly changing face of computing implementations and their manner of utilization have posed legal as well as technical challenges primarily for the courts and forensic investigators but also for organizations intending to launch a forensic investigation.

Perhaps the primary challenge facing digital forensic investigations is how to preserve evidence of system activity given the volatility of digital environments and the long delay between the time of the ‘incident’ and the start of the investigation. Since digital forensic investigations are typically launched in response to an ‘incident’, most of the digital evidence contained in those parts of the computing system other than the hard disk is rarely preserved. Therefore, investigators rely on examining evidence from less transient storage devices such as hard disks, as they are typically the only remaining source of forensic information (Ahmad 2002).
Chapter 1: Introduction

Unfortunately, the rewards of examining hard disk media and the subsequent extraction of potential evidence (commonly known as ‘media analysis’ in academic circles) are typically limited to information contained within existing user data files and intelligence that may be drawn from the raw filesystem. This source of forensic evidence is useful where the investigation is concerned with possession of particular kinds of information. However, where the focus is instead related to an incident that has taken place in the digital environment, evidence of system activity is frequently more critical. Given the volatility of the digital environment, evidence of incidents occurring even in the recent past is unlikely to be recoverable without explicit logging of system activity.

This thesis hypothesizes that system activity logs (or event logs), that were originally designed for troubleshooting and performance monitoring of the digital environment, are functionally suited to forensic evidence collection. Most operating systems provide logging facilities that are able to generate a history file containing information describing system activity. Event logs preserve information about system activity onto hard disks in real-time. This information may be recovered at a later date for analysis.

System activity logs will not be enabled in all cases, indeed, where acts are committed for which there is no desire to keep records, logs are unlikely to be actively collecting evidence. For example, where hackers undertake illegal activity, it is unreasonable to expect them to enable logging and configure the system to yield an evidential account of their activities.

However, this is not the case in managed environments such as in modern organizations. At least in academic literature, there is growing recognition that forensic readiness may have considerable benefits to organizations in case an investigation were to be mounted at a later date (Tan 2001; Yasinsac & Manzano, 2001; Carrier & Spafford, 2003). Experts advise that there are many pieces of evidence that may be actively collected to the benefit of the organization. Among these are emails, backup disks, and particularly log files (Rowlingson 2004).
Chapter 1: Introduction

The provision of an historical account of system activity appears to greatly simplify the problem of digital forensic investigation in managed environments. However, currently activity logging has not been designed for forensics. Therefore, it is not surprising that a closer look at literature in the area of system activity logs reveals a number of criticisms have been made regarding the logging of system activity (Sommer 1998):

- Event log entries capturing evidence of an incident may not contain enough detail to describe the system activity on its own.
- Intense examination may demonstrate the event log to be vague or subject to interpretation.
- Event logs may not be able to provide a sufficiently complete picture to be of use. This is especially the case where the incident in question takes place over multiple platforms.
- Event logs on each platform may provide a fragment, however when put together they may not provide a complete picture of what happened.
- References to the user may not be sufficient for identification purposes (in this sense, external parties cannot be related to events taking place inside the computing system).

There are three observations in particular that concern the design of the logging mechanism that are of particular concern to forensic evidence collection:

- Event logs have not been designed to log certain kinds of system activity that are frequently the focus of forensic investigations (Price 1997).
- The design of event selection facilities tends to constrain the collection of particular event data to the exclusion of others (Axelsson et al., 1999).
- The design of individual log entries typically results in the lack of sufficient detail to characterize events (Price 1997).

Interestingly, this criticism of the design of system activity logs suggests that they may not be a reliable source of evidence for digital forensic investigations. This is despite the fact that purely from a functional standpoint system activity logs may be a
useful source of evidence for digital forensic investigations. Further, experts indicate that logfiles should be proactively collected as a source of evidence (Rowlingson 2004). More troubling however, is that evidence of routine business activity collected in logfiles is generally admissible in court (Casey 2000). Perhaps the apparent contradiction can be explained by the reluctance of legal experts to challenge the technical basis of information systems evidence (Allinson 2003). In summary, although system activity logs may be functionally suited to collect evidence of system activity, their reliability as a source of evidence and general ‘suitability’ to digital forensic investigations has not been previously investigated.

The primary aim of this thesis is to investigate the suitability of log evidence to digital forensic investigations. The term ‘evidential weight’ is used as a measure of the overall ‘suitability’ of system activity logs towards forensic investigations. The term ‘evidential weight’ encompasses the potential contribution system event logs can make to digital forensic investigations.

Therefore, the primary research question of this thesis is:

**How can the evidential weight of system event logs be maximized?**

The term ‘maximized’ is used within the context of organizations where there exist competing priorities such as that of system performance. Therefore, the research question does not investigate how the evidential weight of system event logs can reach an ‘absolute maximum’, but rather how event logs can be used ‘to their best advantage’. The logging capability of modern computing systems is ever increasing due to advancements in CPU speed and media storage capacity. Therefore, ‘maximized’ evidential weight refers to using event logs to their best advantage given the ever increasing logging capabilities of modern computing systems.

Two relevant sub-questions follow from the primary research question:

1. **How can the evidence collection capability of event logs be determined?**
2. **How can the weight of event log evidence be measured?**
Chapter 1: Introduction

1.2 Thesis Approach

This thesis adopts an analytical approach to answering the research question. The aim of this analytical approach is the identification of a comprehensive range of weighting issues specific to event log evidence. Any approach towards investigating a comprehensive range of issues will identify obvious concerns along with those that require more analytical effort. In this context, identification of obvious concerns is an indication of the completeness of the discussion and should be understood as evidence of the systematic nature of the argumentation employed (the advantages and disadvantages of this analytical approach is discussed in ‘research methodology’ - section 1.3).

The primary contribution of this thesis is the development of an evidential weighting framework that includes a comprehensive range of weighting issues specific to event log evidence. To answer the first research sub-question, a generic ‘technology-independent’ model of an event reporting service is developed. The aim is to identify features of logging mechanisms that are typical of real-world operating systems and influence their ability to capture evidence of system activity. Three architectural features of logging mechanisms are identified. The term ‘event detection’ describes those events made available by systems architects for logging purposes. The term ‘event selection’ represents the ability to use the interface to the logging mechanism to nominate events for recording purposes in a single logging session. Finally, ‘event description’ is the ability to record particular information related to an event into a single log entry for later reference. The constraints imposed on evidence collection due to these factors are investigated.

To measure the weight of event log evidence (sub-question two), new evidential weighting criteria are developed. The basis for these new criteria is an existing evidential reliability criteria designed specifically for the more typical case of human-directed forensic investigations. These existing reliability criteria measure the accuracy, completeness and authenticity of the process with which forensic investigators preserve and ultimately present computer-derived evidence to a forensic authority. A parallel is drawn between the existing evidence acquisition process of the
forensic investigator and the evidence acquisition process of system activity logs. Using this parallel as a basis, the evidential principles of accuracy, completeness and authenticity are analyzed, interpreted and defined in the context of system activity logs.

The results of the two research questions are two independent models. The first model identifies those factors that determine the evidence collection capability of event logs. The second model identifies criteria that measure the weighting of log evidence. In the analysis section of this thesis these two models are used to conduct a comprehensive and systematic investigation. The aim of the investigation is to identify evidential weighting issues arising from the design and management of system activity logs.

The framework of evidential weighting issues is prioritized according to the rules of precedence by which legal tests are applied on evidence in courts of law. The outcome of this discussion is an understanding of the key strengths and contributions of system activity logs towards digital forensic investigations. Finally, the practical usefulness of the evidential weighting criteria (that was first developed and then prioritized) is demonstrated by application to real-world operating systems.

1.3 Research Methodology

This thesis adopts an analytical approach to the investigation of evidential weight. Instead of adopting a narrow focus by investigating a few reliability issues concerning system activity logging, this thesis takes a breadth-wise approach that seeks to identify and investigate as many relevant issues to the evidential-weight of logs as possible. Therefore, rather than implementing an event log or part of an event log, this thesis builds a comprehensive theoretical framework and then tests it by applying it on a real-world activity logging system.

As will be pointed out in section 1.4, existing literature addressing the problem of logging system activity has taken a narrow focus. The main advantage of adopting a breadth-wise analytical approach is that the outcome paints a ‘big picture’ of the issues relevant to the research topic. Further, the outcome of this type of analytical
study is unlikely to be influenced by minor changes in technology. There is a disadvantage to a breadth-wise approach as well. The number of issues is large, and due to limited resources available to this study, decisions have to be made on which issues should be explored in greater depth.

A key feature of a comprehensive analytical study is the systematic process by which the outcome is reached. Chapter 6 is the core of this thesis, where a systematic and comprehensive investigation of issues arising from the application of two independent models is conducted in the context of evidential weighting.

In general, the purpose of this research is to use knowledge of digital forensics and knowledge of system activity logging to build a comprehensive framework of evidential weighting issues concerning system activity logs and then to demonstrate its validity by evaluating real-world logging systems. For example, mainstream logging systems such as those employed by Windows and UNIX based operating systems as well as ERP systems such as SAP and even dedicated application environments.

This evaluation serves as an ‘empirical confrontation with reality’. This is explained in the following justification for this methodology taken from a seminal paper on research methodology (Weick, 1984 cites McGuire, 1983):

“...empirical confrontation is not a test of whether a theory is correct; rather, it is a discovery process: to make clear what the theory means, disclose its hidden assumptions, and clarify the conditions under which it is true or false. Through a series of studies, one seeks to establish the theory's pattern of adequacy, that is, to articulate the settings in which its inevitable misrepresentations are tolerable or intolerable.”
1.4 Contributions
This dissertation makes several important contributions to operating systems design and management and digital forensics.

With regard to operating systems, little research has been done on the role of logs in the digital environment and the adequacy with which it reports system activity. This thesis looks at the design and management of event logs and the reliability with which they report system activity. The models and discussion on architectural features that influence the design of the logging mechanism and its usage are contributions to this field.

Further, research could not be found into the reliability of evidence from automated evidence collection and management services operating within a digital environment such as system activity logs. The evidential weighting criteria developed for system-directed evidence collection and management is a contribution to digital forensics research.

The primary contribution of this dissertation is the identification of a range of evidential weighting issues arising from the design of system activity logs. These issues are a contribution to the field of operating systems as they describe the influence of the log design on evidential weight. At the same time these issues are a contribution to digital forensics as they describe the weighting of evidence from system activity logs. The immediate beneficiaries of this assessment include the forensic investigator and the forensic authority as they gain a crucial technical appreciation of the standing of system activity logs in relation to computer-derived evidence in general.

Another major contribution is the identification of the unique attribute of ‘utility’ – the role log evidence can play in establishing the reliability of other computer-derived evidence and potential use in cross-examination of forensic investigator testimony. This contribution is significant because it highlights the unique and critical role
system activity logs can play in establishing the authenticity of computer-derived evidence in general.

The precedence model recognizes that not all evidential weighting issues are of equal importance from the point of view of the court. This model can be applied to any process by which computer-derived evidence is produced to determine which evidential weighting issues will have most impact on the admissibility and subsequent weighting of evidence. This model is a further contribution of the thesis.

The penultimate chapter of this thesis assesses the evidential weight of a real-world activity logging system. The outcome of this chapter is a contribution to research into operating systems especially event logging and digital forensics.

1.5 Critical Evaluation of this Investigation

The contributions of this thesis are around a comprehensive framework that has two useful characteristics.

The first important characteristic is that this research problem is addressed in the context of key stakeholders involved in the evidentiary, technical and legal aspects of computer-derived evidence. This approach yields a more comprehensive outcome than those conducted previously.

The problem of digital forensic evidence collection involves at least four main stakeholders. Operating systems architects design the digital environment in general and facilities within operating systems such as system activity logs. Systems administrators configure and manage the digital environment and have a significant impact on what evidence is available from computing systems under their care. These two stakeholders can be seen as ‘producers’ of evidence. Forensic investigators are involved in the identification, collection, analysis and presentation of evidence to a legal authority. The legal authority sets rules and guidelines that determine the admissibility and weighting of evidence. The latter two stakeholders can be seen as ‘consumers’ of evidence.
Previous discussions on system activity logs have assumed a comparatively narrow focus. For example, Price (1997) discusses system activity logs from an intrusion detection point of view. Price points out a number of attributes of system activity logs that makes them unsuitable for misuse detection purposes. However, the role of system architects and systems administrators in the collection of information regarding system activity is not pursued. Therefore, much of the discussion surrounding system activity logs is in the context of system security and intrusion detection and does not address operating system design and systems administration.

Similarly, Sommer (1998) discusses digital evidence with a focus on systems legal issues and on existing computing technologies that may be potential sources of evidence. However, Sommer does not discuss the role of systems administrators and systems architects that are responsible for designing and managing the digital environment from where evidence is being collected.

Compared to the discussions in Price (1997) and Sommer (1998), this thesis discusses the implications of the issues identified in the framework for the individual stakeholders. For example, one such discussion focuses on how system activity logs can assist forensic investigators to achieve their evidentiary objectives. The legal process by which computer-derived evidence is submitted to court is discussed and commentary on the admissibility of system activity logs compared to other computer-derived evidence is presented. Further, the impact of existing practices by systems architects on the weight of evidence generated by digital environments in general and system activity logs in particular is examined.

The second important characteristic of this framework is the comprehensiveness with which weighting issues (both obvious issues as well as not-so-obvious issues) are identified. There are three significant aspects to this attribute. Firstly, this framework identifies an exhaustive range of issues relating to the research problem. Secondly, each individual issue is explored in sufficient detail to identify the practices of specific stakeholders that have contributed. Thirdly, the issues of highest priority are identified and the implications of the priorities are then re-examined from the perspective of a real-world logging system.
Chapter 1: Introduction

There have been few discussions on the suitability of system activity logs towards any specific goal. Of the existing discussions such as Price (1997), Axelsson et al. (1999), Schaan and McKenney (1991), and Sommer (1998), only Schaan and Price have attempted to identify a comprehensive range of issues. The scope of Schaan’s investigation encompasses network environments and the author approaches the discussion in terms of high-level management practice. As a result, the ensuing discussion is frequently vague and general. If the comprehensiveness of Schaan’s investigation were to be examined within the scope of this thesis then it can be shown that some key concerns regarding system activity logs have not been identified. For example, a major discussion in this thesis surrounds the accuracy of timing information. This discussion has significant implications for the reliability of system activity logs independent of forensic use. Some of these implications include the timings of significant events and the correctness of the chronological sequence of log entries. These issues are not identified in Schaan’s discussion and neither is the obviously important management practice of maintaining the precision of system clocks used to date event log entries.

Price’s discussion on the needs of misuse detection systems results in a comprehensive identification of the inadequacies of event data collection in existing operating systems. However, Price’s discussion is limited to identifying the range of system activities that are not available from system activity logs and does not pursue issues with the accuracy of timing or even availability of timing information regarding significant events in logs.

In general, previous discussions on the inadequacies of system activity logs such as those conducted by Schaan and Sommer remain at a high-level. Unlike in this thesis, they are not explored until the specific practice or design decision contributing to the concern is identified. As an example, two design issues that contribute to the accuracy of timing information in individual log entries have been identified. These are atomicity with which the request to write a log entry is executed and the precision of the source from where timing information is collected. By identifying the particular design issues that contribute to the precision of timing information, the contributing
roles of systems architects and systems administrators in the accuracy of timing information can be identified.

1.6 Thesis Scope
This thesis is about evidence collection by system activity logs in the digital environment. Some communities of experts have limited digital forensic investigations to particular types of motivations such as where an identifiable crime occurs. This thesis argues that digital forensic investigations may be motivated by civil, contractual and organizational purposes as well. Note that in the case of forensic investigations conducted within organizations, evidence may be collected for presentation to an organizational authority as opposed to a legal one. Therefore, discussion of evidential weighting occurs in the most general sense of the term. Chapter 7 is a notable exception to this rule. A framework of precedence is constructed that prioritizes evidential weighting issues based on the legal tests applied by courts to determine evidence admissibility prior to further weighting considerations.

Digital forensic investigations may be part of a general forensic investigation with a significant physical forensic component. Part of this combined forensic investigation involves the management of digital evidence once it has been extracted from the digital environment. The scope of this thesis has been limited to event log evidence while it remains in the digital environment (prior to extraction by forensic investigators).

The majority of operating systems have not been designed to log low-level system activity. Most discussion on the design of event logs stems from the more conventional type of event logging mechanism present in mainstream operating systems. However, only a few operating systems are capable of logging low-level system activity such as system calls. Therefore, when discussing low-level system activity logging, references are being made in the context of the reporting features of logs in two kinds of operating systems. These are the UNIX-based Solaris with the BSM module and Windows-based (NT/XP).
Chapter 1: Introduction

In this thesis, ‘system activity logging’ refers only to those facilities provided by operating systems for logging activity within the digital environment. Although there is potential for additional logging (for example at network and application levels) the facilities to provide additional logging are not examined and is deemed outside of the scope of this investigation.

A further assumption is that the event log is operating in a digital environment within a network-capable host and that there are no special add-ons (e.g. specialized hardware) or customizations applied to the system that enhance a system’s capability to preserve evidence of system activity.

1.7 Thesis overview

Chapter 1 introduces the thesis and presents the research questions.

Chapter 2 presents a brief overview of digital forensic literature. Fundamental concepts in digital forensic investigations in terms of terminology, process, motivation and application are introduced. Background is provided to the thesis in a discussion on the evidence collection problem. Legal challenges and key rulings related to admissibility and weight of computer-derived evidence in general and system activity logs in particular are summarized.

Chapter 3 reviews literature in the area of event logging. The purpose and function of event logging is discussed and references to simple models that describe the process by which event data is collected are presented. A major section of the chapter is devoted to discussing the inadequacies of system activity logging pointed out in literature. Further, discussion on security-motivated event logging and debate on whether intrusion detection systems can be used for evidence collection purposes also takes place.

Chapter 4 begins with an extensive and detailed discussion of the architectural features of a practical operating system’s event logging mechanism. It explores the mechanics of the logging process employed by the operating system. The aim is to construct a generic high-level model upon which evidential weighting criteria can be
Chapter 1: Introduction

applied. This model is constructed from three key features of the event logging mechanism that influence the evidence collection capability of event logs.

The focus of Chapter 5 is the development of criteria for the evidential weighting of event logs. A comparison is made between the role of the event log and that of the forensic investigator. Similarities and differences between the two roles are identified to show applicability. These are then used to analyze, interpret and define the meaning of the evidential principles in the context of event logs. This chapter discusses the contributions made by event logs to computer-derived evidence in general. This attribute of system activity logs is named ‘utility’.

Chapter 6 conducts a comprehensive and systematic investigation of the evidential weighting issues of system activity logs arising from their design and management in operating systems. The generic high-level model developed in chapter 4 and the evidential weighting criteria identified in chapter 5 are used to conduct this investigation.

Chapter 7 addresses the primary research question by identifying strategies towards increasing the weighting of log evidence towards forensic investigations. To this end a simple model is developed based on the legal process of evidence submission.

Chapter 8 demonstrates the practical usefulness of the evidential-weighting criteria developed in chapter 6 by applying it systematically and rigorously on Microsoft Windows’ system activity logs. In particular, this chapter assesses the capability of Windows to collect and manage evidence that is both relevant and reliable in a court of law.

Chapter 9 concludes this thesis by answering the research questions and discussing the contribution of this thesis. Given the prioritized significance of evidential weighting issues, the primary forensic challenges facing the producers of event log evidence – systems administrators and systems architects are identified. Strategies designed to increase the evidential weight of evidence produced by event logs are discussed and some reflections on future directions take place.
Chapter 1: Introduction

Chapter 2: An Overview of Digital Forensics Related Research

Chapter 3: An Overview of System Activity Log Related Research

Chapter 4: Modeling the Event Logging Mechanism

Chapter 5: Evidential Weighting Criteria for Event Logs

Chapter 6: Investigating the Evidential Weighting of Logs

Chapter 7: Maximizing the Evidential Weight of System Activity Logs

Chapter 8: Applying the Evidential Weight Framework to Microsoft-Windows Logging

Chapter 9: Conclusion

Figure 1.1: Logical Flow of Chapters in this Thesis
Chapter 2

An Overview of Digital Forensics Research

The purpose of this chapter is to provide background to the thesis in the area of evidence collection in the digital environment.

This chapter:

• Presents a brief overview of digital forensic literature
• Introduces digital forensic investigations
• Describes the digital evidence collection problem
• Examines legal issues relating to the weighting of digital evidence

Section 2.1 points out that the majority of literature in digital forensics discusses the internals of digital technologies in the interest of recovering potential evidence. Although there is some related research to tools and techniques and standards of education, these discussions are largely centered on the practical aspects of digital forensic investigations (DFI).

Section 2.2 introduces the fundamentals of digital forensic investigations in terms of process, motivation, and application. The author’s preference for ‘digital forensics’ over other frequently used terminology is justified. Of particular importance is a critical discussion on the relevance of digital forensic investigations to a range of incident types and the precise boundaries that separate the digital environment from the physical environment. The particular phases involved in a digital forensic investigation will be reviewed, as will the terminology frequently used to describe them in the literature. A relatively less known mental model based on inductive/deductive reasoning is also presented to underline the difference between intelligence gathering and evidence gathering in digital forensic investigations.

Section 2.3 discusses some of the key themes related to evidence collection to provide background to the thesis problem. Sources of evidence useful in forensic investigations are discussed and the suitability of log files to evidence preservation is
highlighted.

Finally, in section 2.4 the legal challenges and key legal rulings relating to admissibility and weight of digital evidence are discussed. In particular, the discussion focuses on two key obstacles to the admissibility of digital evidence – the Best Evidence Rule and the Hearsay Rule. The chapter ends with a discussion of the criteria for the weighting of computer-derived evidence.

2.1 An Overview of Digital Forensics Literature

Digital forensics is a relatively new field of research with roots in the practice of digital forensic investigations. The lack of maturity in digital forensic research can be seen from continuing disagreement on whether ‘digital forensics’ or a number of similar sounding terms should be used when referring to the topic area (see section 2.2). A number of papers have discussed this issue (Yasinsac et al., 2003; Carrier & Spafford, 2003; Reith 2002).

A study of the arguments reveals fundamental agreement on two issues. Firstly, forensic investigations typically involve the collection of evidence related to a prior incident. Secondly, ‘digital’ refers to the environment where such investigative practices are applied. It is, therefore, not surprising that the majority of literature relates to the inner workings of operating systems, networks, and applications.

Early digital forensics cases in the 80s involved the confiscation of digital equipment followed by analysis of storage media after which potential evidence was identified and ultimately presented in court (Yasinsac et al., 2003). Literature frequently refers to this activity as ‘media analysis’.

As digital equipment pervaded the workplace, home and almost every aspect of modern life, more traditional (physical) forensic investigations in both criminal and civil domains involved digital equipment (Sommer 1992). Hence, the motivation for conducting digital forensic investigations has greatly expanded. Such investigations can be launched for a variety of reasons including establishing possession of illicit material, commission of a criminal act within a digital environment, breach of
contract, and even simply root cause analysis (forensic investigation into the causes of an incident) (Stucki, 2002). In more contemporary times, forensic investigations may be conducted within an organizational context where systems are actively managed and evidence collection processes can be designed to assist forensic investigations (Rowlingson, 2004; Tan, 2001).

The primary focus of research in digital forensics has remained tied to the largely reactive practice of analyzing storage media (media analysis). The majority of literature discusses physical structure and operation of digital media as well as logical structure and composition of stored data (McKemmish 1999). For example, a number of research papers investigate filesystem formats and file structures in various operating systems and digital devices in order to recover deleted information or deduce facts about past activity.

Alvarez (2004) describes how extended file information file headers (EXIF) can be used to determine if a digital picture is authentic. The inadequacy of formal descriptions of filesystems and I/O methodology is discussed by Gerber & Leeson (2002). Willassen (2003) writes about evidence extraction from GSM mobile phone technology including the SIM and core network. De Vel (2004) reports on a study on the automatic learning of file classification using byte sub-stream kernels that capture low-level file structures while Pemble (2004) discusses the preparation of tools and staff for forensic investigations of mainframes and other very large computing systems. Finally, Leimkueller (1995) considers the problem of recovering data from files with unknown formats or have been stored on damaged magnetic media. The author investigates the technical problem of making magnetically stored information on filesystems as digital data.

One particular focus of this segment of literature is timing information. Weil (2002) determines the actual time and date of a system by correlating times and dates contained within a file to the modified, accessed and created times of the file. Hosmer (1998, 2002) demonstrates how to prove the integrity of timing information in digital evidence by binding the identity of the party involved in creating a digital record with the digital record itself and the time when the record was created. Boyd and Forster (2004) point out how the improper handling of evidence can affect timing
information. Stevens (2004) uses a clock model to adjust time stamps by compensating for known clock errors. This technique can help forensic investigators to merge diverse sets of digital data using different timing sources. These papers focus on analysis of the contents of pervasive media long after the events of interest took place.

Forensic aspects of network technology have also been discussed from a technical point of view. Forte (2002) analyzes onion routing traffic algorithms (used to anonymize network traffic) with a view to incorporating forensic back-tracing and logging. Casey (2004) discusses the strengths and weaknesses of existing forensic tools in the context of the stages of the digital investigation process. Daniels (2004) stores observations about network traffic to addressing the problem of determining the point of entry of a network attacker. Corey et al. (2002) discuss the role and functions of Network Forensic Analysis Tools (NFAT).

Research into media analysis has lead to the emergence of related topics. One such topic is the development of purpose-built tools for digital forensic investigation or the adaptation of old tools for forensic purposes (chain of evidence, traceback, etc.) Adelstein (2003) describes the Mobile Forensic Platform (MFP), a tool for performing remote network forensics which is especially useful when forensic investigators need to gather evidence and run analyses on machines that are geographically distributed. Arthur and Venter (2004) conduct an investigation into forensic tools that extract and analyze data from storage devices to identify similarities, differences and potential improvements. Stephenson (2004) highlights the importance of identifying an appropriate forensic tool for a particular task and a suitable forensic investigator for the venue of an investigation.

As forensic investigations became more widespread, new research emerged describing measures that can be taken in systems to assist forensic investigation. Schneier and Kelsey (1998) demonstrate how tamperproof logging facilities can increase confidence in the integrity of logs.

Further research suggests organizations in general can play a role in assisting in the design of electronic records for evidence purposes.
Tan (2001) uses the term ‘forensic readiness’ to describe the maximization of an environment’s ability to collect credible digital evidence whilst minimizing the costs associated with incident response. HB 171 (2003) points out that in an era where organizations are more reliant on information technology and information itself for business, they must produce records and other information that can be readily used as evidence if required. Rowlingson (2004) argues that there are numerous circumstances where an organization may benefit from an ability to gather and preserve digital evidence before an incident occurs. Rowlingson presents a ten step process for an organization to implement forensic readiness (see 2.3.2 for more detail on all three references).

There is ongoing discussion on the moral, legal, ethical and other such overarching issues of digital forensic investigations (Talleur 2002). A significant section of literature discusses legal aspects of typical forensic processes such as preservation and retrieval of evidence (see section 2.2.4). Comparisons of generalized rules and principles of admissibility of computer-derived evidence in relation to traditional legal precedents in non-digital scenarios are also key topics (Sommer, 1992).

In addition, some researchers have identified the lack of standardization in the education and training of personnel involved in forensic investigations. Suggestions range from explicit curricula devised to address this problem to informal ‘lessons learned’ repositories (Yasinsac et al., 2003; Harrison & Aucsmith, 2002).

2.2 An Introduction to Digital Forensic Investigations

The purpose of this section is to discuss the most fundamental concepts of DFI such as basic terminology, motivation and scope of application. Further, this section analyzes, synthesizes and hypothesizes about the concepts of digital forensics.

2.2.1 Terminology

When discussing ‘digital forensic investigations’ or ‘computer forensic investigations’ researchers frequently refer to ‘digital forensics’ or ‘computer forensics’ as a scientific discipline in its own right. In essence these are really just the application of investigative technique, as Judd Robbins explains (Robbins n.d.):
“Computer forensics is simply the application of computer investigation and analysis techniques in the interests of determining potential legal evidence”.

It is easy to forget that this discipline is rooted in investigative practice and to think that it is independent in its own right. It is the practice of digital forensic investigations from which the definition of digital forensics is derived and not the other way around.

Digital forensics literature reveals a large number of terms in circulation such as forensic computing, IT forensics, computer forensics and digital forensics. These terms are related in meaning but are not precisely the same, despite the fact that they are frequently used interchangeably in the literature.

Historically, perhaps the first term to receive widespread exposure is ‘computer forensics’. Its use originated in the 1980s as a term for the examination of stand-alone computers for digital evidence of a crime – as previously mentioned, this activity has been termed ‘media analysis’ today by some experts (Yasinsac et al, 2003).

Yasinsac et al. (2003) references other experts in favour of using ‘forensic computing’ rather than ‘computer forensics’. These experts argue that ‘computer forensics’ limits the scope to computer equipment only and does not encompass analysis of all media where such evidence can be found, especially within the context of the more contemporary network-centric environment.

McKemmish (1999) however, goes further explicitly stating forensic computing is:

“...the process of identifying, preserving, analysing and presenting digital evidence in a manner that is legally acceptable.”
Furthermore, he suggests forensic computing is derived from three activities:

1. Media and electronic analysis: examination of physical and logical structures of various kinds of storage media
2. Data communication analysis: network intrusion or misuse and data interception
3. Research and development: constant research and development to keep up with changing technology

Carrier and Spafford (2003) and Reith (2002) approach the issue from a different angle. They dispense with both ‘computer forensics’ and ‘forensic computing’ in favor of ‘digital forensics’.

Reith (2002) uses a similar argument to Yasinsac et al. (2003) that computer forensics is limited to a single platform or environment unlike digital forensics:

“While computer forensics tends to focus on specific methods for extracting evidence from a particular platform, digital forensics must be modeled such that it can encompass all types of digital devices, including future digital technologies.”

Carrier and Spafford (2003) present a more technical reason. The authors argue that computers should be treated as a point of entry into the digital environment which constitutes an entirely separate ‘crime scene’ created by the cooperative workings of the hardware and software. The paper focuses on the difference between the physical and digital worlds rather than particular devices (discussed in detail later in this section).

Forensic computing has been described in terms of a process, implying it is not limited to any particular technology. However, like computer forensics, it is still predominantly focused on media analysis, albeit on a broader spectrum of technology than just computing systems. A similar situation exists with digital forensics. Although Carrier and Spafford (2003) present entirely logical reasoning behind the choice of the term, their process model (see section 2.2.4) suggests the primary activity remains examination of the physical and logical structure of storage media. This common limitation may be in part due to the unavailability of more useful
content (such as in the form of logs) as most computing systems confiscated by forensic investigators in criminal investigations are not managed in an organizational environment.

The term ‘computer forensics’ may be justifiably used in the title of this thesis as it deals primarily with event logs that are featured in computing systems. However, this thesis uses ‘digital forensics’ for the following reasons:

1. ‘Computer Forensics’ has been closely associated with ‘media analysis’ while the scope of this thesis is outside this traditional area
2. ‘Digital forensics’ better describes the general area of investigations into the digital environment for the reasons pointed out by (Carrier & Spafford, 2003)

2.2.2 Motivation
The motivation for launching a digital forensic investigation is another frequently discussed item. A few experts have limited the scope of digital forensic investigations to particular types of incidents and in particular circumstances. The majority, however, point out that as more societal functions are involving the use of computing technologies the scope of digital forensic investigations is expanding.

 Experts such as Yasinsac and Manzano (2001) have referred to computer forensics as being post-intrusion or generally related to IT security:

“post-incident analysis of computers victimized by intrusion or malicious code”.

Broucek (2002) clarifies the confusion between IT security and digital forensics by outlining three broad distinctions between these two terms. IT security is largely about protecting computers whereas computer forensics is about investigating incidents that may or may not have anything to do with the security of the platform where the incident took place. Broucek and Turner (2001) point to Patel and Ciardhuain (2000) which describes a forensic case centered on the distribution of child pornography on the Net and also suggest a scenario involving the misuse of email to send life threatening messages to a student. In both cases, there was no breach of system security (although in the author’s opinion, in many organizations these cases would constitute a breach of organizational security guidelines). In
dispelling the notion that forensics is involved only in security incidents, Broucek and Turner (2001), have cited two examples where crimes were committed. In this discussion, they explicitly cite “inappropriate, criminal or other illegal behaviour” in the use of digital technologies as the primary motivation for a forensic investigation.

Forensic investigations, however, can be conducted where no crime has been committed and no security violation has occurred either. A contractual dispute where litigation involves digital environments is one such example (HB 171 2003). Forensic investigations can also be initiated where there is no litigation involved. Organizations may perform root-cause investigations on digital environment related incidents. This has also been identified as a legitimate reason for launching a forensic investigation (Stucki, 2002; Sommer, 1992). The authors also claim organizations may undertake root cause analysis without intending to go to court. According to HB 171 (2003) besides criminal, civil, and other administrative litigation, forensic investigations are conducted as part of due diligence or due to interest of third party authorities.


“facilitation or furthering the reconstruction of events found to be criminal, or helping to anticipate unauthorized actions shown to be disruptive to planned operations”.

However, the fundamental outcome of a forensic investigation must be information of some kind for presentation to an authority. As Kruse and Heiser (2002), Stucki (2002) and Sommer (1992) point out, this information is not necessarily collected for presentation in court but may be for use within the organizational context. Again, it is important to note that incidents that warrant investigation do not have to be a crime or violation of a policy or procedure. A contractual dispute is sufficient to warrant a forensic investigation as pointed out by the relevant Australian Standards Handbook (HB 171 2003). In the case of a root-cause investigation, anomalous system activity may lead to a forensic investigation.
Therefore, in principle, any activity in the digital environment may become the subject of a digital forensic investigation. However, at least within the organizational context, it is likely that incidents will involve the utilization of applications such as word processing, email and browsing, as this is the primary reason for modern use of computing systems. This observation is reinforced by a survey of the various categories of digital forensic evidence typically presented in court (Sommer 1997):

**Criminal Domain**
- Infringed copyright materials offered in the course of a business
- Evidence of fraudulent offers to deliver goods
- Evidence of fraudulent offers to provide services
- Evidence of fraudulent or non-compliant investment offers
- Holding or offering pornographic files and images
- Incitements to racial hatred, terrorism and other offences
- Conspiracies
- Computer misuse

**Civil Domain**
- Evidence of the existence and terms of a contract which is alleged to have been breached
- Evidence whether a document or e-mail or file was sent or received as dated
- Breach of copyright
- Defamation
- Evidence of negligence

### 2.2.3 Parallels between Physical and Digital Forensic Investigations
There are a number of dimensions of physical forensic evidence as discussed in classic works such as Lee (1995) and Saferstein (1998). Most of these are specific to evidence in the physical world and have no obvious parallels with the digital environment. For example, handwriting analysis and bloodstain pattern analysis (BPA) are particular to evidence that does not exist in the digital environment.
However, at a more abstract level, certain generalized parallels can be made between
digital and physical evidence. Whereas physical forensic investigations operate within
an environment that is governed by the laws of nature, digital forensic investigations
operate within a very different environment that is constructed by the execution of
code written by operating system engineers. Sommer (1997) identifies a number of
parallels between digital and physical evidence that is relevant to this discussion.

Firstly, digital evidence retains a dynamic quality that makes it difficult to establish
content and precise dating. For example, digital evidence in virtual memory and even
on hard disks is being constantly accessed and modified both by the operating
system’s own management processes and also by users and applications. This state of
flux makes it difficult to establish the precise state of a digital exhibit. If a forensic
investigation refers to a particular file it may be able to establish the precise filename
and location on the hard disk, but its content and date of creation, modification and so
forth will be more difficult to determine.

Secondly, digital data can be modified leaving no obvious signs of tampering. Unlike
in the physical world where the blotting out of an entry in a ledger book leaves
obvious signs of tampering, deletion of digital information does not leave
immediately visible traces of modification in the same way. However, Casey (2000)
points out that although tampering with digital evidence does not leave immediately
visible signs of modification, at the same time digital evidence is relatively difficult to
destroy as the widespread success of data recovery processes have demonstrated.

Thirdly, the computer environment exhibits a kind of fragile fluidity that is not only
dynamic (constantly executing background tasks that continually change the state of
the system) but is significantly impacted by apparently simple tasks - for example the
opening or closing of a file. Attempts at collecting computer-derived evidence are
prone to contaminate computer evidence.

Fourthly, computer-derived evidence is not immediately understandable by the
layperson. Digital forensic experts prepare evidence for presentation to a largely
computer-illiterate audience. Digital evidence will rarely be presented in its native
format, but is instead interpreted, analyzed and then visually illustrated to the court audience. This point has become the focus of a debate around the need for specialist juries to deal with hi-tech crime cases (British Computer Society, n.d.). One side argues that jury members must be technically competent enough to understand computing-related issues, as many cases hinge on technical concerns. The other side argues that the very premise of the jury system is that common sense is the only fundamental requirement in the pursuit of justice and that limiting jury members to a small percentage of the population in legal proceedings of any kind is not advisable.

Casey (2000, p. 4) also points out two further distinctions. Digital evidence differs from physical evidence in that it can be duplicated exactly, allowing investigations to be safely conducted on the copy without risking contamination of the original. In addition, modification and tampering of digital evidence can be easily detected with the right tools if the original is available.

In terms of purpose, motivation, and process there are similarities between physical and digital forensic investigations. For example, some of the fundamental laws governing the physical world have parallels in digital environments. One of the most well-known rules in the physical forensic domain is the Locard Exchange Principle explained by Chisum and Turvey (2000) in the following passage:

“Dr. Locard’s belief and assertion that when any person comes into contact with an object or another person, a cross-transfer of physical evidence occurs. By recognizing, documenting, and examining the nature and extent of this evidentiary exchange, Locard observed that criminals could be associated with particular locations, items of evidence and victims. The detection of the exchanged materials is interpreted to mean that the two objects were in contact. This is the cause and effect principle reversed; the effect is observed and the cause is concluded.”

A similar effect is seen in the digital environment. Carrier and Spafford (2003) point out that software execution may leave traces such as temporary files, memory contents and deleted file fragments on the hard disk. They also rightly point out that the designers and implementers of the code (operating system or application) control whatever evidence is left behind. Both physical and digital forensic investigations
Chapter 2: An Overview of Digital Forensics Research

frequently aim to reconstruct activity surrounding the central incident. Incident reconstruction is frequently needed to answer the specific questions directed towards the forensic investigator (Casey 2000).

“The ultimate goal of a crime reconstruction is to establish what happened and when. It is not enough to know that someone was injured or that a computer was broken into. Reconstructing the details surrounding the injury or break-in are often essential to understanding what happened, who caused the events when, where, how and why. Without an accurate reconstruction it is difficult to determine what the intent was, and what actually transpired.” (p.64)

However, some confusion has arisen about the boundaries of digital ‘crime’ scenes as they must be located within a physical ‘crime’ scene. Carrier and Spafford (2003) point out that the digital crime scene should be considered a secondary scene, i.e., a door through which investigators can move from the physical scene to the digital scene and that the two scenes are separate and distinct. This opinion differs from previous authors who have either suggested that any scene involving computers should be considered a ‘computer crime scene’ or have recognized the distinction between the physical and digital crime scenes but have neglected to argue that physical forensic techniques should be applied to the physical scene outside the digital environment.

The following difficulties related to the characteristics of digital evidence pose a challenge for forensic investigations:

- Establishing digital evidence content and precise dating
- Establishing deliberate tampering or contamination
- Identifying influence of routine system activity on digital evidence
- Immediate understanding of digital evidence for laypersons

2.2.4 Process Models for Digital Forensic Investigations

The process model of a digital forensic investigation has been the subject of numerous papers. Each of the models proposed varies in abstraction and detail, and uses overlapping terminology that presents a fragmented picture of the processes involved. Four steps are common to all process models. McKemmish (1999) outlines these
Chapter 2: An Overview of Digital Forensics Research

fundamentals:

1. Identification: knowing what evidence exists, where it is stored and how it is stored.
2. Preservation: examination carried out in the ‘least intrusive manner’; least amount of change and alteration (if cannot be avoided) must be accounted for and justified
3. Analysis: ‘extraction, processing and interpretation’ of digital data
4. Presentation: a combination of manner, expertise and qualifications of presenter and credibility of processes

Reith (2002) studies a number of digital forensic models and proposes nine steps. Among these steps is a new recognition phase that attempts to determine the type of incident, a preparation phase allowing tools, warrants, monitoring authorizations and management support to be obtained before the investigation begins, and an explicit approach strategy to be formulated before the evidence is identified. Chronologically these three phases occur before the first identification phase outlined in McKemmish (1999).


Carrier and Spafford (2003) presents an integrated digital-physical investigation process featuring 17 different phases grouped into five categories:

1. Readiness phases: Addresses need for operations and infrastructure support for the investigation
2. Deployment phases: Covers incident detection and notification of appropriate personnel followed by authorization to begin the investigation
3. Physical crime scene investigation phases: Collect and analyze the physical evidence and reconstruct what transpired during the incident
4. Digital crime scene investigation phases: Six phases detailed below
5. Review phase: reviews the investigation with the aim of identifying areas of improvement

This digital crime scene investigation model details a different approach from previous models. Instead of beginning with the identification of potential evidence, it focuses on preserving the entire digital environment before potential evidence is identified. This phase is called ‘preservation of digital evidence’. This approach may be feasible for digital devices contained within a physical crime scene, for example, in the case of a raid on the home of a hacker. However, in large organizations with numerous computers it is impossible to preserve all computers before beginning to look for evidence.

The second phase is similar to the first step described by McKemmish (1999). Carrier and Spafford (2003) name it ‘survey’ and describe it as aimed at identifying potential evidence from the preserved digital scene. The documentation phase records the digital evidence when it is found. This phase is implicit in the ‘preservation’ step (McKemmish 1999).

Carrier and Spafford (2003) advocate a second analysis stage termed ‘Search and collection’. This involves the use of various analysis techniques that focus on an exhaustive study of low-level aspects of the digital environment. The reconstruction phase is similar to the analysis phase in McKemmish (1999) where accumulated digital evidence is analyzed. The final presentation phase is similar to his presentation step.

The above descriptions of computer forensic processes focus on discrete (technical) steps undertaken by forensic investigators when undertaking an investigation. This thesis is not concerned with the high-level methodology of investigations. Instead, it is interested in the contributions of digital forensic evidence towards forensic investigations. More useful is some understanding of how forensic investigators analyze digital evidence.

In this regard, digital forensic analysis can be described in terms of a two-phase mental model or mind-set (Ahmad 2002). The first phase, known as the exploratory
phase, is an attempt by the investigator to identify the nature of the problem at hand and to define what s/he thinks transpired at the scene of the incident. For example, in a hacking case the investigator may need to pinpoint the source of the break in. In a corporation with hundreds of computers and thousands of entry points, this may well be a daunting task. Another possible scenario may involve a network server crash. Such incidents involve numerous unknown factors that need to be examined by the investigator before a plausible explanation can be formulated. Identifying possible suspects may be even more challenging.

Once the investigator has determined what s/he thinks took place the induction ends and the deduction, i.e. the evidence phase, begins. The evidence phase revolves around the accumulation of proof admissible in court that deductively proves the conclusion of the forensic investigator made by way of induction.

The exploratory or inductive phase of the investigation tests the investigator’s ability to detect patterns. Each scenario consists of recurring patterns that define a commonly occurring “normal” sequence of events, e.g. users following their usual patterns of computer/network usage, backups taking place according to their predetermined schedule. The patterns that form this “normal” sequence of events when identified by the investigator allow him/her to visualize any disruptions or anomalous events that may have taken place. The most promising avenues of investigation tend to lie in these anomalous occurrences that should be marked for careful scrutiny at a later date.

Too often, forensic investigators confuse their inductive hypothesis with the necessary deductive reasoning (Chisum & Turvey, 2000):

“Induction is a type of inference that proceeds from a set of specific observations to a generalization, called a premise. This premise is a working assumption, but it may not always be valid. A deduction, on the other hand, proceeds from a generalization to a specific case, and that is generally what happens in forensic practice. Providing that the premise is valid, the deduction will be valid. But knowing whether the premise is valid is the name of the game here; it is not difficult to be fooled into thinking that one’s premises are valid when they are not.
Forensic scientists have, for the most part, treated induction and deduction rather casually. They have failed to recognize that induction, not deduction, is the counterpart of hypothesis testing and theory revision. Too often a hypothesis is declared as a deductive conclusion, when in fact it is a statement awaiting verification through testing.”

Event logs can be used in the exploratory phase for intelligence gathering purposes and in the deductive phase for evidence gathering.

A major part of this section was devoted to digital forensic investigation process models. This thesis is concerned with the collection of evidence by computing systems. It is concerned with ‘how’ the evidence will be used to support a forensic investigations rather than ‘when’ (or in which phase) it will be extracted from the computing environment by the human investigator. Therefore, for the purposes of this thesis a ‘whole of lifecycle’ mental model of forensic analysis is more useful in identifying requirements for digital evidence.

2.3 The Evidence Collection Problem

This section identifies some of the key themes related to evidence collection to provide background to the thesis problem. Sources of evidence useful in forensic investigations are discussed. In particular, the potential suitability of log files to evidence preservation is highlighted. This is especially the case where the forensic investigation hinges on specific details of an incident that occurred a long time ago.

Forensic investigators typically do not have control over the configuration and management of computing environments prior to an investigation. However, within the organizational environment such a scenario is more likely. Organizations are beginning to recognize the value of implementing an evidence management plan as part of overall forensic readiness. Australian guidelines like HB171-2003 have been drafted for the design of evidence from computing environments (HB 171). Within this environment, evidence collection of system activity can be proactively addressed prior to incidents occurring.
2.3.1 Sources of Evidence in Computing Systems

Ideally, forensic investigations would benefit from a comprehensive reconstruction of the events leading up to and including the incident itself. The ultimate aim of a reconstruction in digital environments is similar to that of a reconstruction in physical environments. Essentially, without a reconstruction it is not possible to discount many events that may impact the outcome of an investigation.

Reconstruction of an incident within a digital environment includes the virtual environment itself, which is created by specific hardware and an operating system. But it also includes the applications and the (dynamic) system activity in the general time frame when the incident took place (Carrier & Spafford, 2003).

Apart from event logs, Ahmad (2002) points out three further categories of evidence that are crucial to reconstruction of the incident scenario. The first is data files present on the hard drive. The second source of evidence is data found within filesystem structures like slack space. Low-level bit-by-bit copies are necessary to preserve a relatively precise image of the hard disk that includes slack space. The last category is the most difficult to capture – it is a type of evidence that is dynamic and exists in the less pervasive memory space of the systems involved such as RAM. An image dump of memory is valuable in identifying active processes that may have been involved in the incident.

Recall the discussion on terminology in section 2.2. Terminology such as ‘computer forensics’, ‘forensic computing’ and ‘digital forensics’ maintains a ‘media analysis’ mindset where the physical and logical structure of storage devices is the predominant focus of evidence collection. As this discussion on the categories of evidence points out, analysis of storage media must not be limited to structure and associated patterns emerging from usage but must include data describing the system environment that may be intentionally stored for future use.

Although trace evidence from hard disks is useful, the most critical type of evidence
from a reconstruction perspective is that which is closest to the incident itself - the system activity that occurred during the incident. Unfortunately, as forensic investigations typically take place long after the incident, unless immediate steps were taken to preserve the state of the system, all three categories of evidence would have been partially or completely lost (Ahmad 2002). Forensic investigators are typically limited to recovering evidence like data files or conducting an exhaustive low-level analysis of hard disks, as they are typically the only remaining source of evidence. Event logging is uniquely suited to the task of creating evidence regarding system activity.

In the opinion of the author, the majority of cases that heavily rely on technical evidence can be successfully refuted using the Trojan Horse Defense (THD) argument (Brenner et al, 2004). This argument suggests that the illicit activities that form the subject of the legal investigation may have been conducted by a Trojan – a program that secretly orchestrates activities ‘behind the scenes’. Such a defense has been successfully mounted in both England and the United States (e.g. Regina v Greene, Regina v Caffrey) (Carney & Rogers, 2004).

The THD argument was used by the defendant Aaron Caffrey (Regina v Caffrey) to argue that a crime (Port Houston computers were hacked causing them to shut down) had been committed from the defendant’s laptop but that the defendant was framed by other hackers that had installed Trojan horse programs. Although the prosecution pointed out that the laptop had no traces of Trojan horse programs, the defense countered by claiming the Trojan horses were of the ‘self-erasing’ kind such that no trace remained. The jury acquitted Caffrey thereby setting a precedent where defendants may claim they are not responsible for any action committed by their computer.

Given the scope and range of motivations for digital forensic investigations mentioned in section 2.2 (especially in the civil domain), there are a number of scenarios where data files extracted from the hard disk or other intelligence gained from file system structures are not likely to offer useful evidence. Consider a simplified example where a civil investigation regarding a multi-party contract may attempt to establish whether an application service unexpectedly failed at a particular
time in the past (which may be many years ago). In such cases media analysis is unlikely to provide any significant evidence especially after such a long period has elapsed. In this scenario, the event log service is uniquely capable of maintaining records relevant to the investigation.

In this section, the various sources of evidence available from a computing system are explored. It was pointed out that at least in terms of investigations into past system activity, the most useful source of evidence is also the most difficult to capture. Event logging is uniquely suited to the task of creating evidence of system activity.

2.3.2 Evidence Management in a Forensic-Ready Organization

As discussed in the previous section, in the majority of forensic cases, it is highly unlikely that logging would even be enabled, let alone configured and managed towards assisting a potential investigation. However, within the organizational environment there is a distinct motive for instituting an evidence management plan as part of a policy of forensic readiness.

Tan (2001) suggests organizations may want to remain forensically ready to maintain an advantage on internal incidents or to use the evidence in court. The author suggests forensic readiness has two key objectives:

1. Maximizing an environment’s ability to collect credible digital evidence and;
2. Minimizing the cost of forensics in an incident response

Proactive collection of potential evidence improves the organization’s ability to use such evidence in future litigation (Rowlingson, 2004; Yasinsac & Manzano, 2001; HB 171, 2003).

Rowlingson (2004) points to a number of reasons why organizations would be motivated as such:

• Digital evidence could help manage the impact of some important business risks
• Digital evidence can support a legal defence
• Digital evidence could support a claim to Intellectual Property Rights
• It could show due care (or due diligence) was taken in a particular process
• It could verify the terms of a commercial transaction
• It could lend support to disciplinary actions
• It could assist with a dispute or information security event
• It could serve as a deterrent aimed at internal staff

HB 171 (2003) states that fiduciary obligations of company directors include the implementation of financial and other controls to protect the rights and property of the company and its stakeholders. In this context digital evidence is:

“a tool to confirm or deny the reality of a given set of purported facts and.. it allows organizations to protect themselves…”

The author cites three reasons why organizations may want to manage digital evidence:

1. To take action against those causing or facilitating damage (i.e. litigate)
2. To refer such action to the relevant authorities or
3. To protect themselves from litigation

Yasinsac and Manzano (2001) illustrate the importance of systematically storing potential evidence by presenting a diagram of the typical evidence-gathering behaviour of organizations that are not forensically ready.

In the first figure, evidence collection begins after the incident is detected by personnel. In this scenario a substantial portion of the evidence regarding events taking place leading up to and during the incident is already lost.
In the figure below, Yasinsac and Manzano (2001) present an improved model where the gathering of potential forensic evidence is occurring constantly, thereby allowing the preservation of critical evidence in the critical phases of the incident. This information is immediately ready for analysis.

**Figure 2.1: Evidence Gathering After Incident Detection**

**Figure 2.2: Evidence Gathering Before Incident Detection**
Chapter 2: An Overview of Digital Forensics Research

HB 171 (2003) proposes six steps towards the management of digital evidence:

1. Design for evidence
   In this stage the handbook suggests that evidentially significant records are identified along with their authors. Important information such as their date/time of creation or alteration, authenticity of the records, and reliability of the computer programs are also established.

2. Produce records
   The handbook goes on to stipulate conditions for the production of records such as the name of the human author that created the record, the time of creation and any computer programs that were involved in the process.

3. Collect evidence
   The objective of this stage is to collect all relevant evidence and secure the original electronic records so that nothing is altered.

4. Analyze evidence
   In this stage, the handbook suggests that the IT evidentiary records are analyzed to extract facts and to deduce opinions related to those facts. Finally to identify any evidence that may assist the enquiry that has not been collected.

5. Reporting and presentation
   The objective here is to make a convincing case of the facts extracted from the IT evidence. At this point the handbook suggests that assistance may be needed in formatting the evidence for presentation and/or obtaining legal assistance with various aspects of the case such as code of conduct for expert witnesses.

6. Determine weight of evidence
   The handbook points out the need for records to be relevant to the case, authentic and produced by a computer program that was correctly operating. However, it also points out that a discussion on the rules of evidence is out of scope of the handbook and must be left to legal authorities.
HB 171 (2003) presents an internal forensic investigation-type life cycle that begins with the explicit aim to design computer systems in order to maximize evidential weighting of electronic records and incorporates a continual assessment of the strength of the evidence culminating in a final assessment by a legal authority. The handbook represents a benchmark that explicitly states what an organization must do to manage its digital evidence in a manner consistent with global industry best practice and national legal guidelines. Each step in the life cycle addresses the precise requirements of electronic records from an evidence point of view. Implicit in the description are the policies and procedures relating to personnel and capabilities the organization must acquire to meet these standards.

Rowlingson (2004) focuses on ten specific actions an organization must take to achieve forensic readiness. Many of these are organizational guidelines in the form of policies and procedures. Rowlingson advocates a business risk assessment is first undertaken to identify where digital evidence must be required. He then suggests the organization identify sources of potential evidence that may be utilized. The evidence collection requirement is the defined by deciding which potential evidence sources can assist in addressing the threats identified. The author then advocates the organization establish a capability for gathering the evidence needed and then establish a policy for secure storage and handling of potential evidence. He also suggests the evidence itself is monitored for threats. Further, circumstances must be specified when escalation to a full formal investigation is required. Staff must be trained to understand their role in the digital evidence process. A policy must be produced that describes how an evidence-based case should be assembled. Finally, legal review must be requested to review the case from a legal standpoint.

However, both models suggest the following:

1. Organizations adopt a risk-driven process to identify the evidence collection requirement.
2. At various points in the process of forensic readiness the strength of the evidence must be weighed. HB 171 (2003) suggests this may be done by a legally competent independent third party, or senior organizational management.
3. Policies and guidelines that address collection, analysis, reporting and presentation must be drafted and tested and personnel trained to carry them out.
Following the suggestion that organizations utilize a risk-based approach, Rowlingson (2004) identifies five such scenarios where businesses may determine if the return on investment may justify forensic evidence collection.

His next step is to identify sources of potential evidence available within the organization (and other evidence that can be generated) that may be useful for collection purposes. Evidence from organizations are categorized into background evidence which is typically gathered and stored as part of business operations and foreground evidence which commonly known as ‘monitoring’. He then suggests an analysis of the first two steps be conducted to determine which of the sources identified assists the gathering of evidence related to scenarios the business is concerned with. The author names this stage ‘the evidence collection requirement’.

Much of this section simply reinforces the need for organizations to collect digital forensic evidence. However, there are certain points that are significant from an evidence collection point of view:

1. Organizations aim to maximize an environment’s ability to collect credible digital evidence
2. Organizations will need to periodically determine evidential weight for best results
3. Organizations determine evidential requirements by adopting a risk-driven process.

### 2.3.3 Preserving Evidence of System Activity

The capturing of potential evidence from a dynamic computing system poses a new challenge for forensic investigations in an organizational environment. From past discussions, it is apparent that the most critical source of evidence for an accurate reconstruction of system activity during an incident is most likely to be available during the very short duration after the incident occurred. This information is typically overwritten and eventually lost long before the forensic investigation is able to begin.
Sommer (1997) suggests mechanisms of evidence collection including capturing terminal input, using screen dump captures or software-based screen cameras such as Lotuscam. He suggests video recording as well. Other techniques described in literature include using a hardware expansion card to copy volatile memory to an external storage device (Carrier & Grand, 2004).

All the above suggestions involve an add-on to existing systems. Audit trails, however, have actually been used for evidential purposes. The term ‘audit trail’ is frequently used to refer to a general category of text capture facilities such as system and database logs (also known as journals) (Allinson 2001). The author acknowledges their use for evidential purposes:

“..audit has now become a process where a record is maintained of a particular series of events in order to provide evidence in the case of a dispute, to ensure compliance with certain rules and regulations, to check on the effectiveness of control systems, and to provide evidence in the case of criminal activity. These records are commonly known as audit trails or audit logs.”

Event logging facilities are particularly useful to organizations determined to become ‘forensic ready’ as they are featured in almost all computing systems. Although these logging facilities were not originally designed for forensic purposes (Murray 1998), they can generate a history file containing information descriptions of system activity. Essentially, event logs preserve information about system activity on hard disks as the activity is unfolding. A log therefore is a series of descriptions of past system activity written to pervasive media which is useful for forensic investigations. A more comprehensive discussion on event logs is the topic of chapter 3.

2.3.4 Identifying Evidence Collection Requirements for Organizations

Evidence collection from organizational information infrastructure represents a significant challenge (Ahmad & Ruighaver, 2002). Following the risk assessment and determination of evidence collection requirements (as recommended by Rowlingson (2004)), the organization is likely to be left with somewhat standard requirements regarding evidence collection of potential incidents. These requirements should be formalized as part of an organization-wide high-level logging policy that incorporates
data collection for reasons other than evidence collection, such as intrusion detection and performance monitoring. The aim of this policy would be to provide administrators with a standard that can be applied to all systems thereby maintaining consistency across all domains.

However, as Ahmad and Ruighaver (2002) point out, the translation of these evidence requirements is frequently difficult, as systems administrators will find it difficult to translate the high-level evidence collection requirements to the log management interface that tends to mirror the low-level workings of an operating system. For example, a system administrator desiring to log all activity related to a single application cannot instruct the operating systems by executing a single command. In Windows for example, the administrator must instruct the operating system to begin detailed tracking of processes and to log a very large number object accesses to generate the desired evidence. In addition, the specific objects that were being monitored would also have to be selected before the relevant log entries would be generated.
2.4 Admissibility and Weight of Electronic Evidence

This section begins by pointing out that evidence submitted to court is tested for admissibility before weight is considered. The unique nature of digital evidence presents two key obstacles relating to its admissibility, the Best Evidence Rule and the Hearsay rule is also discussed. The Best Evidence rule relates to the concept of an ‘original’ document. In the case of digital evidence, identifying an ‘original’ analogous to the physical world presents a challenge. Similarly, the principle of hearsay evidence tests the truth of a document as it relates to activities that have taken place outside the court. In the case of computer printouts, this presents another challenge. Further, topics of discussion include the particular case of the admissibility of computer log files and criteria for the weighting of digital evidence of computer-derived evidence: authenticity, accuracy and completeness.
2.4.1 General Tests for Submission of Evidence to Court

Hallen (1988) summarizes the context in which evidence is submitted to court (references to particular cases have been omitted):

“Where a case is heard by a judge without a jury, as the majority of cases are, the judge is the sole arbiter of law and fact.

Where a case is heard by judge and jury, the functions of each must be distinguished. The judge decides all questions of law and of practice and procedure and determines whether or not evidence is relevant, and if relevant, whether it is admissible. This may involve the determination of a fact before a decision is made whether or not the evidence should be admitted.”

And towards clarifying the distinction between relevance and admissibility (note that this rule refers to the admissibility of the evidence in the first instance and does not preclude the evidence from being thrown out at a later stage, for example due to evidence contamination):

“...the basic rule governing the question of admissibility is that evidence must be relevant.”

And referring to the weight of evidence:

“After the evidence is found to be admissible (a matter of law), what is ultimately considered by the tribunal of fact is the cogency of that evidence.”

Sommer (1998) explains admissibility as the requirement to ‘conform to certain legal rules which are applied by a judge’. The rules of admissibility are rooted in the legal history of the specific jurisdictions and can be found in the Rules of Evidence. In general, the adversarial system has developed a complex set of rules governing what constitutes evidence from a legal point of view. Weight implies the evidence “must be understood by, and be sufficiently convincing to the court” (Sommer 1997).
From this discussion emerges an obvious general principle that admissibility precedes weighting of evidence. Furthermore, a reasonable distinction between admissibility and weight can be deduced. Admissibility places conditions on evidence designed to prevent its outright presentation to court whereas weighting concerns the ability of the evidence to convince the court one way or the other.

2.4.2 Overlap between Admissibility and Weight

Sommer suggests various legal systems differ in the way they treat issues of admissibility and weight of evidence in general. He refers to two cases (not related to the digital environment) - R v Doheny (1996) in the UK and Daubert V Merrel Dow in the US (1993).

In the UK case, the expert presenting DNA evidence neglected to mention the basis of the empirical data and did not inform the jury of the frequency with which matching DNA may be found in the population at large. The expectation was that it was up to the jury to decide whether the DNA evidence was conclusive. The case was subsequently overturned because the expert was considered to have overstepped the boundary by preventing the jury from consciously making the decision to accept the scientific basis or to reject it. In this scenario, the jury decided whether to accept or reject the scientific basis of the evidence, in effect judging both its admissibility and its compelling nature.

In the US, however, the judge has more control over evidence admissibility. Once the judge decides the evidence is admissible, the jury must consider its implications. In this scenario, admissibility is separated from weight and the jury considers the implication of the evidence only after the judge decides to admit the evidence.

In Daubert vs Merrel Dow a number of factors were considered in determining whether the method used to derive particular evidence had a sound scientific basis (termed the Daubert test) after which the actual evidence was presented to the jury.
The Daubert tests are:

- Whether the theory or technique can be (and has been) tested
- The error rate associated with the method
- Publication in a peer-reviewed journal
- Whether the technique has gained widespread acceptance

The two cases illustrate that at times there is no clear boundary between the conditions of admissibility and weighting. In these unique circumstances, the decision to accept the evidence can be considered an admissibility problem or a weighting problem depending on the legal process adopted.

### 2.4.3 Admissibility of Evidence

Ryan and Shpantzer (2000) suggest admissibility of digital evidence must satisfy two conditions. Firstly, that the evidence is relevant to the case at hand, and secondly that the evidence must have been derived scientifically and with appropriate validation.

This suggestion is reinforced by Krause and Tipton (1998). They point out that computer evidence tends to be treated with suspicion possibly due to the ease it can be tampered with and difficulty in detecting contamination. Krause and Tipton (1998) relate two conditions of admissibility in this regard:

1. Relevance of evidence
2. Reliability of evidence

Although relevance can be defined lexically, in general it is best to leave this test to the logic of the court (Hallen 1988). The author further comments that in practice evidence is generally considered relevant as long as the purpose is clear and the subject matter considered is logically relevant.

Krause and Tipton (1998) consider reliability to apply to both the process by which evidence is produced as well as the evidence itself.

In the case of digital environments there are two difficulties in meeting the condition of admissibility, namely the Best Evidence rule and the Hearsay rule (Clough 2002).
Chapter 2: An Overview of Digital Forensics Research

**Best Evidence Rule**

In general, this rule stipulates that documentary evidence submitted to court must be the best evidence available to the party. In terms of physical documentation the best evidence rule implies only original copies of documents will be admissible if available to the party.

As digital evidence is intangible and cannot be used as evidence in its native form, the Best Evidence Rule allows information stored in computers to be held in physical containers (paper) and considered to be ‘original’.

As Clough (2002) points out, it can be argued that the concept of ‘original’ does not exist in the digital domain. If the original is the copy that was being created as a user was typing the content in, then that was destroyed when the document was committed to disk and the copy in RAM was overwritten. Although intellectually the argument may be controversial, in legal practice proceedings have been straightforward. The author cites the position taken by the High Court in Butera:

> ‘The best evidence rule is not applicable to exclude evidence derived from tapes which are mechanically or electronically copied from an original tape. Provided the provenance of the original tape, the accuracy of the copying process and the provenance of the copy tape are satisfactorily proved’.

The Federal Rules of Evidence in the USA : 1001(3) asserts the following regarding the Best Evidence rule (Ryan & Shpantzer, 2000)

> “..if data are stored in a computer or similar device, any printout readable by sight, shown to reflect the data accurately, is an ‘original’”.

Both Clough (2002) and Ryan and Shpantzer, (2000) relate that the respective jurisdictions still require oral testimony regarding the document’s reliability as evidence.
Hearsay Evidence Rule
One of the relevant rules that pertain to computing investigations is “hearsay” – traditionally this refers to the situation where a witness can only claim to be party to a conversation of interest and cannot produce any evidence of substance to that effect.

Hoey (1996 as cited in Casey, 2000) states:

Evidence is hearsay where a statement in court repeats a statement made out of court in order to prove the truth of the content of the out of court statement. Similarly, evidence contained in a document is hearsay if the document is produced to prove that statements made in court are true. The evidence is excluded because the crucial aspect of the evidence, the truth of the out of court statement (oral or documentary), cannot be tested by cross-examination. (p.46)

From this point of view, all computer-generated evidence may be perceived to be hearsay. Casey (2000) in the context of US Law, identifies relevant exceptions to this case stated in the US Federal Rules of Evidence. The Rules of Evidence explicitly identify “memorandum, report, record, or data compilation, in any form, or acts, events, conditions, opinions or diagnoses” that have been “kept in the course of regularly conducted business activity, and if it was the regular practice of that business activity”. Such records are exempt from the “hearsay” rule in general.

Given this line of argument Casey (2000) suggests event log files would be admissible and relates from experience that computer-generated information has been accepted as tangible evidence towards proof of a fact (direct evidence):

“For instance, computer log files are created routinely and contain information about acts and events made at specific times by, or from information transmitted by, a person with knowledge. In fact, some computer-generated information has been seen as so reliable that it has been accepted as direct evidence. Direct evidence is usually something tangible that is presented to prove a fact. Under certain circumstances, a computer log file might be accepted as direct evidence.”
Chapter 2: An Overview of Digital Forensics Research

Casey inserts a footnote at the end of this passage that is relevant to this discussion:

“Existing rules of evidence are not entirely clear about when digital evidence is hearsay and when it is direct evidence. Therefore, the courts are ultimately responsible for deciding if evidence is admissible.”

2.4.4 Weight of Evidence

As previously mentioned, the weight of evidence is only considered after it has been deemed admissible in court. Although the distinction between admissibility and weighting is at times unclear, in general, weighting relates to the ability of the evidence to convince. The weighting of evidence depends to some extent on its bearing on the case. As such there are few general rules relating to the determination of weight (Hallen 1988).

Hallen presents four possible factors relating to testimonial evidence:

1. Demeanour of person, i.e., believability of the account
2. Whether witness can be qualified as an expert
3. Whether or not the witness is cross-examined about some aspect of his evidence which is in issue
4. Whether or not the evidence is corroborated (eg. by documentary evidence)

2.4.5 Admissibility of Computer Log Files


'logs files are admissible as the evidence if they are collected in the regular course of the business'.
Chapter 2: An Overview of Digital Forensics Research

They, however, raise concerns about the security of the systems upon which this crucial evidence is collected:

- These systems may not even be able to collect the data that they are designed to collect
- Where data collection occurs, these systems may only provide a partial data set
- This data set may itself be flawed, erroneous or have already been tampered with

Note that the authors do not mention that digital evidence presented in court still requires oral testimony. Sommer points out that the health of the system must be established by a forensic expert. It is not enough to point out that systems are vulnerable to attack in order to throw out all evidence. The onus is on the opposition to show that there are grounds to investigate the health of the system.

2.4.6 Desirable Features of Computer-Derived Evidence

In this section Sommer’s desirable features for non-testimonial evidence are presented whilst recognizing it was developed in the context of human-directed evidence preservation. These attributes of evidence, and the subsequent series of tests produced by Sommer to meet the requirements of these attributes, will be examined in the course of constructing evidential weighting criteria in chapter 5.

Sommer identifies three main desirable attributes of evidence – authenticity, accuracy and completeness  Miller (1992 as cited in Sommer, 1997):

(1) Accurate: free from any reasonable doubt about the quality of procedures used to collect the material, analyse the material if that is appropriate and necessary and finally to introduce it into court – and produced by someone who can explain what has been done.

(2) Complete: tells within its own terms a complete story of a particular set of circumstances or events

(3) Authentic: specifically linked to the circumstances and persons alleged
Reed (1990 as cited in Sommer, 1997) explains authentication as:

“Authentication means satisfying the court that (a) the contents of the record have remained unchanged, (b) that the information in the record does in fact originate from its purported source, whether human or machine, and (c) that extraneous information such as the apparent date of the record is accurate. As with paper records, the necessary degree of authentication may be proved through oral and circumstantial evidence, if available, or via technological features in the system or the record”.

The author expands these attributes for more technical types of evidence and presents a series of tests that can be used to meet the above requirements of accuracy, completeness and authenticity. In addition to these tests, Sommer notes that the quality of forensic presentation is also a desirable feature of computer-derived evidence.

1. Computer’s Correct Working Test
Sommer (1997) argues that the computer must be shown to be behaving “correctly” or “normally”. In cases where the computer acts as an information store then such a requirement may be easy to satisfy. However, if the computer is providing a service such as a database query function and given the investigation is related to precisely that function then it must be tested and shown to be “correct” or “normal”.

2. Provenance of Computer Source Test
The evidence collected that is deemed relevant to the investigation must be proven to be taken from the specific computer and from nowhere else.

3. Content/Party Authentication Test
The evidence collected must be relevant i.e. linked to the incident or parties accused in the investigation.

4. Evidence Acquisition Test
The information evidence must have been gathered accurately, must be free from contamination, and must be complete (note this refers back to the three main attributes of non-testimonial evidence).

5. Continuity of Evidence/Chain of Custody Test
A full account of what happened to the retrieved evidence after it was extracted must be provided. Frequently, all of the individuals involved in the collection and transportation of evidence may be requested to testify in court. Thus, to avoid confusion and to retain complete control of the evidence at all times, the chain of custody should be kept to a minimum (Saferstein, 1998, cited in Casey, 2000).

Unlike the previous criteria that were developed in the context of media analysis, Allinson (2003) presents second criteria relating to event logs in general which is closer to the context of this thesis. The author comments on the analysis of audit trails in relation to several legal cases involving computers. She makes two key observations about evidence regarding the circumstances surrounding these cases. Firstly, that the analysis must determine whether all the relevant events were recorded:

“Audit trail content requires both analysis of what is recorded within each event/activity record and whether or not all event/activity records are recorded in the audit trail.”

because:

“Without full recording of all event/activity it is not possible to state or prove with a degree of certainty that particular actions were or were not performed. Therefore, there is a requirement for a high level of assurance that all activity is recorded.”

And secondly, that the “reliability” of each event must be analyzed in particular:

“The validity, relevance and reliability of each piece of data is the second area of scrutiny.”

The second criteria applies ‘authenticity, accuracy and completeness’ to the information content within the event log as opposed to the process by which evidence is extracted from media.
2.5 Summary

Digital forensics is a relatively new field of research with roots in the practice of digital forensic investigations. In recent years, the motivation for conducting digital forensic investigations has expanded to include organizational context where systems are actively managed and evidence collection processes can be designed to assist forensic investigations. Research into forensics has diversified from being focused on technical aspects of computing and network technology to addressing ‘forensic readiness’ and evidence management in organizations.

Although any activity in the digital environment may become the subject of a digital forensic investigation, however at least within the organizational context, it is likely that incidents will involve the utilization of applications such as word processing, email and browsing, as this is the primary reason for using computing systems.

Unlike traditional forms of computer-derived evidence such as data files and filesystem structures, event logs record system activity. Fortunately, logs are widely believed to be suitable for evidentiary purposes.

There are three desirable attributes of evidence that contribute to reliability - authenticity, accuracy and completeness. Sommer presented a series of tests to be undertaken by a forensic investigator to demonstrate that a certain computer-derived exhibit meets these three attributes of evidence.

This chapter has provided a background discussion to digital forensics - one of the primary areas of research in this thesis. The following chapter provides background to the second primary area of research – system event logging. Chapter 5 develops weighting criteria for system activity logging using Sommer’s tests for evidential reliability.
Chapter 3

An Overview of Event Log Research

The purpose of this chapter is to provide background to the thesis in the area of event logging in the digital environment.

This chapter:

• Provides a review of literature relating to event logs
• Introduces the purpose and function of event logging
• Discusses the inadequacies of system activity logging
• Discusses security motivated event logging
• Examines intrusion detection systems as evidence collectors

Section 3.1 identifies and describes commonly used terminology such as ‘event logging’, ‘system activity logging’ and ‘auditing’. Simple logging models are presented to describe the process by which event data is collected. The structure and composition of various log files are also touched upon.

Section 3.2 points out that the little research conducted into event logging features primarily in the areas of computer security and intrusion detection. The context of this research has related to collection of data regarding system activity and associated issues of management and administration such as event log storage, retention, integration and reduction.

Section 3.3 summarizes key research issues in event log management. Advice given to systems administrators on issues such as event selection, contemporary log storage and retention strategies are presented as well as security concerns such as confidentiality, integrity and availability of log files. Integration issues regarding event data correlation and lack of compatibility of log structures across operating systems is also discussed.
Chapter 3: An Overview of Event Log Research

Section 3.4 discusses inadequacies in the logging of system activity. Three particular issues identified in literature are the inability to log particular system activity, the inability to log specific events to the exclusion of others, and the inadequate characterization of events.

Section 3.5 presents a brief overview of the literature relating to security event logging, particularly the aims of logging for security purposes. Various kinds of security auditors used to conduct event log analysis are also discussed.

Finally, section 3.6 summarizes the main issues in an ongoing debate as to whether intrusion detection systems can be used for evidence collection purposes. In particular, the advantages of using intrusion detection systems for evidence collection are discussed. A number of reasons why intrusion detection systems are unlikely to be appropriate for forensic use are also suggested.

3.1 Event Logging: A Chronological Record of System Activity
The purpose of this section is to discuss fundamental concepts of event logging such as basic terminology and existing models representing the logging mechanism.

3.1.1 Terminology
According to Murray (1998), an event is:
“..any type of detectable change that occurs in a system”

In the context of computing systems, a typical dictionary definition of an event log is (Picket & editors, 2000):

“A record, as of the performance of a machine or the progress of an undertaking: a computer log."

The phrase “system activity logging” or just “activity logging” has been defined more usefully as ‘the passive recording of events for subsequent review” (Bonyun 1981).
Bonyun (1981) makes two important points about logging:

1. Logging involves the recording of events
2. The purpose of logging is subsequent review

Another term that has a similar meaning to “system activity logging” is “event logging”. The following is a definition of event logging (Murray, 1998):

“Event logging is a facility used by computer systems to record the occurrence of significant events.”

The author proceeds to qualify this statement by explaining that the significance of events is subjective and that since any event is a candidate for logging, therefore it is important for a system administrator to determine which events must be logged:

“Because a computer system experiences literally hundreds or thousands of events each second, it is important to distinguish between which events require the immediate attention of the system administrators, which event warrants a mention with an entry written to a log, and which events can be safely ignored. Any type of event that may be detected by a process running on a computer system is fair game to be reported and logged.”

Note that in this thesis, the terms ‘event logging’ and ‘system activity logging’ are used interchangeably.

When logging is motivated by security concerns the term ‘auditing’ has been frequently used. Bishop (1995) makes a distinction between logging and auditing by suggesting that logging refers to the process of making the record only, whereas auditing is a security-motivated analysis of that record. Therefore, Bonyun’s second point can be replaced to define auditing as – ‘the purpose of logging is security analysis’.
Price (1997) cites three definitions of audit:

1. To examine a system for security problems and vulnerabilities (Russell & Gangemi, 1991)
2. To record and analyze system activity for security problems and vulnerabilities (Russell & Gangemi, 1991)
3. To analyze system activity for security problems (Bishop 1990).

Russell and Gangemi (1991) consider the process of recording and analysis collectively as ‘auditing’ and therefore define the set of event log entries regarding system activity as an ‘audit trail’ (but make no mention of a security motivation):

*Audit Trail (1): A chronological set of records of system activity*

This section discusses two categories of terms. The first category simply describes a function independent of motivation. This category includes terms like “logging”, “system activity logging”, “activity logging” and “event logging”. In this thesis no distinction is made between these terms.

The second category of terms ascribes a motivation to the function. The term ‘audit’ refers to security-motivated logging. There is, however, no similar term for forensic-motivated logging. Therefore, in this thesis the first category of terms is used to refer to the act of logging. The term “audit” is not used except in reference to papers that use this term.

### 3.1.2 Simple Event Logging Models

Price (1997) presents a high-level model for an auditing system. The following illustration is an adaptation of figure 1.2 (in Price’s thesis) entitled “Simple Model of an Auditing System”.

![Figure 3.1: A Simple Model of An Event Logging System](image)

57
The term ‘audit data’ has been replaced with ‘event data’ to make the diagram independent of motivation, therefore simpler to analyze in the context of forensic evidence collection. This diagram indicates event data may be collected from a computing system and analyzed, after which a report may be generated.

Price (1997) points out that the simple model of data collection and analysis becomes more complicated when the collection encompasses a distributed computing system. In such a case, more than one logger will be involved and may require more than one analyzer as well. The following is adapted from this model of an auditing system in a distributed environment:

![Figure 3.2: Model of an Event Logging System in a Distributed Environment](image)

Conventional operating systems typically feature a single event logging facility. However, multiple event logs can be maintained on the same host focusing on collection of event data from different parts of the operating system.

Sommer (1998) describes various kinds of logs:

1. System logs
2. Audit logs
3. Application logs
4. Network management logs
5. Network traffic capture
6. Contemporaneous manual entries
Without providing a definitive description of each of these categories, Sommer (1998) refers to all of the above as ‘logs’. Note that in Windows, system and application logs refer to two separate event (or audit) logs. The former is a repository of messages relating to Windows system components such as the operating system or hardware subsystems whereas the latter is commonly used to report errors or convey some information on the state of a process (a more detailed discussion of Windows logs is in section 4.1.2). Network packets (eg. tcp/ip) traveling ‘over the wire’ may be preserved as a source of evidence. This act requires specialized tools and is typically known as ‘network traffic capture’ (Casey 2004). Network management logs may simply record information such as network alarms (Haggerty & Seetharaman, 1998).

It is interesting to note that the above logs are not all structured the same way. Operating system supplied logs typically feature multiple entries in chronological order, each describing a separate event (e.g. system logs, application logs, network management log) whereas others record data streams (network traffic capture).

The model of the ‘event log’ in this thesis is based on those featured in operating systems that meet a few criteria. Firstly, the operating system must have the ability to generate event data related to system activity. Secondly, the operating systems must have entered mainstream use. Thirdly, each event logging facility must have available and comprehensive documentation. Finally, the logging system must be able to log some low-level activity (such as using system calls or object-accesses). Therefore, this thesis refers to the features of logs in two kinds of operating systems. They are the UNIX-based Solaris with the BSM module and Windows-based (NT/XP).

3.2 An Overview of Event Logging Literature

Although the logging of system activity may be undertaken for a variety of purposes, prior work on this topic has been predominantly motivated by security analysis needs in general and intrusion detection in particular.
Chapter 3: An Overview of Event Log Research

Event logging has been used to accumulate data regarding system activity for subsequent analysis. However, due to the varying architectural design and implementation of operating systems, a few papers have focused on issues relating to the inadequacies of event logging. Some of this discussion has focused on difficulties in correlating and ensuring interoperability between event logs maintained on different systems (Bishop 1995). Other discussion has questioned whether system event logging in general is insufficient for misuse detection systems (Price 1997).

Although research involving event logs primarily concerns intrusion detection, general administrative and operational issues of event logging have also been touched. A number of experts have raised concerns regarding the collection, storage, retention, reduction, security, and integration of event logs (Schaen & McKenney, 1991; Axelsson et al., 1999; Sommer 1997).

However, little work has been done on the evidential aspects of event logs. One researcher has criticized conventional implementations of event logging as having proven to be largely inadequate for forensic purposes (Sommer 1998).

An exception to the use of event logs for security analysis is research into fault-tolerant computing. Research in this field tends to use event logs as input data, from which observations can be drawn regarding system workload and system configuration to improve the availability of computing systems. For example, Tang et al. (1990) used event logs to evaluate system dependability characteristics. Lal and Choi (1998) use UNIX server event logs to calculate Mean Time Between Failures (MTBF) and availability. Simache et al (2002) conducts measurement-based availability assessment on Windows machines producing statistics on uptime/downtime and reboot rates. Simache and Kaaniche (2005) conduct the same tests as Simache et al (2002) but on SunOS/Solaris machines. Interestingly, although all the above-mentioned papers base their conclusions regarding the reliability of the computing system on event logs, they do not discuss the reliability of the event logging mechanism itself.
3.3 Event Log Management

3.3.1 Event Selection

Event selection advice to systems administrators has been consistently focused on detecting suspicious activity and/or identifying high-level system events. Event failures are high on the list of logging needs. However, more recently logging with forensics in mind has been advised as well (Computer Emergency Response Team, n.d.).

Event Selection Advice on UNIX-based systems

In general, UNIX systems administrators are asked to log suspicious behavior or high-level activity (Adkins 2002). For example, in the UNIX environment ‘su’ and failed login attempts are recommended for logging purposes. Systems administrators are encouraged to use logging as an alarm or ‘tripwire’ mechanism. This is understandable as logging ‘everything’ is likely to be impractical. The tripwire mentality encourages systems administrators to identify particular events that are of concern. Once such an event occurs, the systems administrator is notified. Adkins (2002) states the following (quote follows):

"Ensure that the following items are being logged:

- SU attempts
- Failed login attempts
- Last command – who logged in, when and from where
- System events"

Event Selection Advice on Windows-based Systems

In the Windows NT environment login/logout attempts, changes to logging policy and other such high-level events are recommended for logging (Computer Emergency Response Team n.d.). Low-level logging at process tracking and object-access levels are typically used for troubleshooting for specific cases only.

Event selection at such a high-level of abstraction implies the Windows event selection model is largely consistent with the UNIX strategy. Forensic considerations have become visible, however they are limited to high level events as well. The
following quote from Computer Emergency Response Team (n.d.) prescribes event selection strategy:

“The Audit Policy configuration for a Windows NT 4.0 Workstation allows you to have several types of success and failure events recorded in corresponding system, security and application log files. The types of events for which recording in logs may be enabled include:

- User and process logon and logoff
- Access to data or devices associated with the system
- Use of access rights by users and processes on their behalf
- Additions, changes or removal of user accounts and groups
- Changes of access rights to system data and resources
- Shutdown or restart of the system, registering of trusted logon processes, or other activities that affect system security
- Execution of processes

Auditing of events should be enabled for all deployed Windows NT systems. Begin with auditing enabled to track failure events for all types as these should not occur very often, and are probably worth examining when they do occur.

For data helpful in examining security-related problems, track success events associated with logons and logoffs, changes to user accounts and groups, security policy changes, and system restarts and shutdowns.

The purpose of logging success events is to keep a record of normal, expected activity and to help you identify events that should not have succeeded, and which affect security. Data of this nature may be critical for investigating security intrusions, and for evidence required in legal proceedings.”

3.3.2 Storage, Reduction and Retention

From previous discussion, it is evident that event logs are data files that are physically located on a storage medium. However, a number of storage strategies can be implemented and each has implications for log management (as well as forensics).
Schaen and McKenney (1991) discuss various log storage strategies in a network scenario. Event data can be stored in one of three locations:

a. in the component where it was originally collected
b. in a central component where it must be forwarded
c. in both (a) and (b) at the same time

Schaen and McKenney (1991) suggest that sharing a portion of the event data in the central component may facilitate network event analysis, even if mechanisms for the forwarding of event data to a centralized component are needed. In addition, if the event logging components on the network retained varying levels of security then access control would have to be carefully managed to prevent confidentiality breaches. If encryption were used then this would prevent event data consolidation. The latter two arguments are reasons to consider maintaining copies of event data locally.

Event data retention is crucial to the maintenance of the event context. Event data pertaining to a single event may be stored in different locations. If part of the event’s description is destroyed due to poorly synchronized retention periods then analysis will be impaired.

Although this is much less of an issue, at the time of publishing this thesis storage space was still a key consideration, given the enormous amount of data generated by event logs. Compression is suggested but careful selection of events to log is considered the primary means of reducing storage.

Schaen and McKenney (1991) discuss a number of strategies that can be used to filter out events during collection, archival or analysis stages of the auditing cycle. Note that it may be difficult to prove the neutrality of such filtration processes (Sommer 1998). Guidelines are necessary to prevent unauthorized reduction that may affect integrity and therefore reliability of event data in court.
Chapter 3: An Overview of Event Log Research

Schae and McKenney (1991) also note that connectionless protocols are harder to log as each packet is independent. Schae and McKenney (1991) suggest there may be some justification for treating each packet as a separate event.

In some cases event entries do not provide sufficient context to describe the circumstance. In the case of the contents of a file, if events describe the opening and saving of a file the entry will not be able to relate what information may have been added or deleted in the file.

3.3.3 Log File Security
Event data residing on computing systems, whether host-based or network-based, poses a number of security issues. Schae and McKenney (1991) discusses the security of event data within a host and across a network from confidentiality, integrity and availability attacks - intentional or accidental. Hashii and Wee (2000) identify unauthorized alteration of the system state and malicious attack on the logging system as threats to the security of event logs. Sommer (1998) also discusses the possibility that logs may be compromised at any stage of the evidence process (forensic investigation).

Schae and McKenney (1991) points to time-stamping or chained encryption as useful techniques and suggests ‘authentication’ techniques and trust in the audit processes will be necessary.

Confidentiality and integrity of event data written prior to the hijacking of the operating system can be protected by use of a tamperproof event logging technique. The use of such technology prevents the attacker from reading prior log entries and from undetectably modifying or destroying event log entries (Schneier & Kelsey, 1998). Once event data is written to a file then secure digital timestamping can improve the integrity of the log evidence as well as reinforce the chain of custody (Hosmer, 2002).

Access to event data from other organizations is a complex problem as it may not be clear which components may have event data relevant to a particular event. Read
access to event data should be granted on a need-to-know basis only (Schaen & McKenney, 1991).

Hashii and Wee (2000) studied event log countermeasures and suggests that event logs are vulnerable primarily because the separation of duties regarding systems administration privileges (and event logs) is not enforced. Hashii and Wee (2000) outlines two particular goals of an attacker, namely to access protected objects without being logged and to access protected objects in such a way that inaccurate information is logged. Hashii and Wee (2000) describes techniques for achieving these goals across both UNIX and NT-based machines.

Murray (1998) points out that the event logging facility is usually a primary target of intruders. Hackers typically aim to disable the logging service or attack the log itself to remove a record of illegal activity. Therefore, the logging service and the event log itself must be protected from unauthorized modification including total destruction.

### 3.3.4 Log File Integration

Schaen and McKenney (1991) identify two categories of problems with integration. The first is the correlation of log entries recorded in different components that describe the same event. The second is variation in the adequacy of the logging capability of components and differences in the amount of detail, event type and format of logs. Correlation and content variation issues apply to both network and host based scenarios. Logging can take place in various components within the same system, for example system and network stacks may log entries for the same and different events.

A heterogeneous network of systems featuring differing event log formats does not lend itself to correlation and interoperability of event information. Three different approaches appear in literature. Bishop (1995) defines a standard log format with two fundamental properties, portability and extensibility. According to the author, portability would allow logs to be processed on any system. Bishop (1995) proposes using ASCII characters to avoid byte ordering, character representation, and floating point format problems. He suggests not using a fixed record length to avoid problems caused by precision. In addition, the author allows user-defined fields and removes
Chapter 3: An Overview of Event Log Research

the need for a fixed number of fields and designated names. These two measures reflect the extensibility property. Another means of addressing correlation and interoperability is to use the Normalized Audit Data Format (NADF). This format is structured as a sequential file of NADF records. Audit trails can be converted to NADF format. NADF feeds into the ASAX misuse detection system to allow it to be somewhat independent of a particular operating system (Mounji, 1995, 1997). Finally, Chen, et al. (2003) develops a consolidated repository of correlated event data that can be queried for forensic purposes.

3.4 Inadequacies in system activity logging

A number of authors suggest event logging of system activity has proven inadequate for security purposes. Interestingly, the flaws that make event logs inadequate for security reasons also make them inadequate for forensic purposes.

Sommer (1998) adopts the point of view of a forensic investigator in pointing out a number of commonly occurring circumstances where event logs have been collected but are unable to convince a third party.

1. Event logs may lack sufficient detail

Event log entries capturing an incident may not contain enough detail to describe the incident on its own. Under intense examination, event log entries may be shown to be vague or subject to interpretation.

2. Event logs may be incomplete for the relevant period

Event logs may not be able to provide a sufficiently complete picture to be of use. This is especially the case where the incident in question takes place over multiple platforms. Event logs on each platform may provide a fragment, however when put together they may remain unable to provide a complete picture as to what happened.

3. Event logs may be unable to identify the subject in a useful way

References to the user may not be sufficient for identification purposes. In this sense, external parties cannot be related to events taking place inside the computing system.
From an operating system point of view, literature describes three key limitations in the logging of system activity. These are the inability to log certain kinds of activity, inability to log specific events to the exclusion of others, and inadequate characterization of events.

### 3.4.1 Inability to Log Some Types of System Activity

In an attempt to determine whether event logs fulfill the data collection needs of misuse detection systems, Price (1997) identifies certain kinds of system activity that cannot be logged by conventional operating systems.

Price (1997) surveys a number of operating systems (HP-UX, OpenVMS, Solaris (BSM), UNICOS, Windows NT) and identifies shortcomings. The author found that event-logging systems are inadequate in “tracking users in a distributed system, in providing object domain and content information, and in providing information on network activity”. Further, many operating systems omit user-level activity (commands etc.) and resource utilization (user CPU time, system CPU time, etc.) information from event logs. In addition, I/O activity cannot be deduced from event logs.

Price (1997) also notes that computing systems do not execute subject actions on objects atomically. She provides the example of a write to file action. In this case a “write to file” operation may consist of multiple system events like “request of a write operation”, “initiation of a write operation”, more than one “status of a write operation”, and “completion of a write operation”. As the author points out, only one of these events triggers an event log entry that represents the complete write operation.

Since event logs engage in internal monitoring of system activity as opposed to external monitoring, they are forced to use an internal system event as a trigger (e.g. execution of some piece of code) indicating that a particular operation is taking place.
She points out in her example, a trigger like the “request for a write operation” can be used to indicate a write operation has taken place.

However, from a forensic point of view these system events do not constitute evidence that the operation took place but rather that a system event took place that triggered a log entry.

3.4.2 Inability to Log Specific Events to the Exclusion of Others
Event logs provide system administration with a facility to tag particular events (from the set of events made available for logging) such that, when they subsequently occur in the system environment, they trigger the logging subsystem to record a description. Unfortunately system management consoles tend to follow an “all or nothing” approach where logging is either on (and overwhelming in volume) or off.

Axelsson et al. (1999) point out that the collection of particular event data to the exclusion of others may be difficult, as audit mechanisms typically do not incorporate such functionality. The authors attempt to record every invocation of the exec(2) command on SunOS BSM and notes that for every exec(2) call, 15 system calls in total must be logged.

3.4.3 Inadequate Characterization of Events
Conventional operating systems tend to characterize an event in terms of a subject, object and action where a subject is the user, or process acting on behalf of the user (Price 1997).

Subjects are typically referenced using identifiers linked to the user name, group membership, process acting on behalf of the user and terminal. Objects are referenced using unique identifiers and locations. Event log entries are therefore not self-contained but must be cross-referenced with system information to determine identification.

Most computing systems do not feature user-level information in event logs. In particular, user-level commands tend not to be preserved. Resource utilization
information is also typically absent in event logs. Distinctions between shell scripts and program files are not made. Object references that are aliases are not noted either.

However, characterizing a generic event, independent of whether it occurred in a physical or digital domain can be summed up in the fundamental who, what, where, when and why questions that are common to incident reporting. According to O’Connor (n.d.), a reconstruction of an event aims to present:
“A plausible theory of who, what, when, where and why the crime happened.”

Yasinsac and Manzano (2001) points to a digital forensic reconstruction of events as needing to answer the “who, what, when and where” questions only.

3.5 Security-Motivated Logging

Although event logs were first designed for system accounting purposes, it was soon recognized that the same information collection tool would be useful in monitoring suspicious system activity (Price 1997).

When used for security purposes, event logs have five primary aims (National Computer Security Center 1988). The event log must:

1. Evaluate the effectiveness of protection mechanisms of the system and allow for the review of patterns of access to objects, access histories of processes and the use of protection mechanisms to be carried out
2. Capture attempts by users to bypass protection mechanisms
3. Capture the use of privileges by users assuming functionality greater than their own
4. Function as a deterrent against typical attempts to bypass protection mechanisms. For the deterrent to be effective it must be known that such a function exists and is utilized
5. Provide post-incident facility to record and capture breaches of security

Although more than one of these aims is generally of interest to a forensic investigation, only the fifth aim refers to the use of an event log in an explicit forensic capacity, albeit with respect to security breaches only.
Chapter 3: An Overview of Event Log Research

Bonyun (1981), Piciotto (1987) and Schaen and McKenney (1991), (cited in Price, 1997) explain the uses of event logs from a computer security perspective:

- **Maintaining Individual Accountability**: An individual's actions are tracked in an audit trail allowing users to be personally accountable for their actions. Knowing their activity is tracked leads to users being less likely to circumvent security policy, and if an incident does occur, individual accountability can be maintained.

- **Reconstructing Events**: Audit trails can be used to reconstruct the events leading up to an incident, exposing vulnerabilities in the system. The detection and removal of vulnerabilities is important to the defense of the system.

- **Assessing Damage**: Audit trails can be analyzed to determine the amount of damage that occurred with an incident. Audit data can reveal what information was disclosed or corrupted or who gained unauthorized access to information or the system.

- **Problem Monitoring**: Audit trails can be used to uncover problems and perform system health monitoring. Real-time monitoring and analysis of the status of the system allows detection of problems, such as disk failures or network outages, as they arise.

- **Deterring Computer Crime**: Belief that an effective auditing system exists and there is a significant risk of detection is a deterrent to computer crime.

Although the term ‘forensic’ is not explicitly used, reconstruction of events leading up to an incident is a forensic activity. In addition, the tracking of user actions is also frequently part of a forensic investigation.
However, the above uses of event log data are limited to security incidents only, whereas the scope of forensic investigations into prior incidents is wider than just security-related incidents (see section 2.2 for discussion on motivations for forensic investigation).

In the past, event log content was typically analyzed by system administration. The administrators were looking for security (and other) warning signs. Given security in general is a preventative activity, human-directed event log analysis (that typically takes place after-the-fact) was rapidly recognized as inadequate. In addition, human analysis of raw event data output was becoming impractical due to the ever-growing volume of information generated by event logs.

### 3.5.1 Security-Motivated Event Data Analysis

Bishop (1990) presents a general and formal model of auditing based on four stages – logging, reduction, analysis, and notification.

In the above diagram, the commands are the user and program behavior on the system. According to the command, the system changes state. Depending upon the type of logger, it records either the state or the command causing the state transition. Reduction then removes the irrelevant entries and produces a final log that is then analyzed in the analysis stage and then used to issue an audit report. Notification simply disseminates the messages to the intended recipients.

---

**Figure 3.3: A Formal Auditing Model**
Bishop (1990) classifies auditors into three types:

1. Static auditor: audit the state of the system with respect to a known configuration or policy (eg. Hobgoblin (Leadly et al, 1991), COPS (Farmer & Venema, n.d.))
2. Dynamic auditor: audit the state with respect to previous states
3. Quasi-dynamic auditor: the transitions between states of the system with respect to a model or policy (eg. Tripwire (Kim & Spafford, 1994)).

### 3.5.2 Intrusion Detection

As most of the data in event logs tends to describe legitimate access and only a small percentage related to security violations, Anderson (1980) suggests reducing the event log into a sequence of records related to security activity. This automated practice has been used by real-time intrusion detection systems to identify system attacks launched by insiders or external agents. Intrusion detection research documents two approaches to addressing this problem.

The first approach involved characterizing the ‘normal’ behavior of a system and using it as a template to identify ‘abnormal’ behavior. This technique is known as ‘anomaly detection’ and was implemented using statistical intrusion detection systems such as Haystack (Smaha, 1988, cited in Price, 1997, p.37).

The second approach was developed because intrusion techniques that existed within the ‘normal’ range of system behavior could not be detected by the anomaly detection model (Price 1997). This approach hypothesized that intrusions exhibited a defined signature or pattern that could be used for detection purposes. This technique became known as ‘misuse detection’ and was implemented by a number of rule-based expert systems like DIDS (Snapp et al, 1991) and Wisdom & Sense (Vaccaro & Liepins, 1989).
3.6 Intrusion Detection Systems as Evidence Collectors

Some discussion has surrounded the question of using intrusion detection systems for the purpose of evidence collection (Stephenson 2000; Broucek & Turner, 2002a; Sommer, 1998). Sommer (1998) argues that intrusion detection systems are generally not suited to collecting forensic evidence. However, Stephenson (2000) hypothesizes that intrusion detection systems could be deployed to play the roles of intrusion detection at the same time as evidence collection. Stephenson (2000) points to a claim by Yuill et al. (1999) that attack profiles can be generated from IDS systems and used for forensic profiling.

The following are arguments as to why IDS systems are not suitable for evidence collection (Sommer 1998):

1. The effectiveness of the IDS system is based on its ability to provide the earliest possible warning of an attack. This principle inherently conflicts with the need to collect as much evidence as possible regarding the usage of the computing system for use in profiling external parties. In short, the earlier the warning, the shorter the duration of evidence collection possible.

2. The recognition by an IDS system of the attacking pattern/attacker is not necessarily admissible in court without the raw event data from which the conclusions were made and/or an expert testifying to the correctness of the mechanics of the system and its particular configuration and operation.

3. The IDS system alone, without multiple streams of corroborating evidence is unlikely to be sufficiently convincing in a court of law.

4. There needs to be a chain of custody from the source to the court.

Stephenson (2000) relies on the claim by Yuill et al. (1999) that an IDS can collect significant evidence during an ongoing attack to profile an attacker to present an open question:
“Is it practical and appropriate to combine intrusion detection and response with forensic management of collected data within a single IDS in today’s networks?”

Stephenson (2000) describes three difficulties arising from this question:

1. IDS requirements vary considerably among organizations as they tend to be determined by organizational security policy whereas forensic requirements (Stephenson uses ‘forensic’ to mean ‘for legal proceedings’) tend to be rigid
2. IDS systems do not manage their information such as forensic systems do making it difficult to adhere to forensic requirements like maintaining a chain of custody
3. To achieve both aims of detection and evidence collection requires a particular standard architecture which is capable of both tasks

Stephenson (2000) readily accepts little work has been done in this area and proceeds to hypothesize that IDS systems can be designed for intrusion detection and forensic purposes. However, unfortunately this hypothesis was not subsequently tested as per an email from P. Stephenson on 7 February 2005.

In the light of the arguments leveled by Stephenson (2000) and Sommer (1998), prior work in IDS systems seems less relevant for a number of reasons. Firstly, there is evidently little existing research into forensic aspects of intrusion detection systems. Secondly, although intrusion detection systems may store forensically useful information regarding system activity, they cannot be a dependable source of forensic information. This is because selection of system activity for logging is not based on forensic goals. This implies there is nothing special about IDS information collection as opposed to any other application conducting system activity related information collection. Further, the high number of false alarms that IDS systems are known to generate indicate the inadequacy of the evidence collected in uniquely identifying system activity (this relates to Sommer’s first point as to why IDS systems are not suitable for evidence collection).

From a forensic point of view it may be argued that the IDS collect data from the system as part of a routine (and the collected data is therefore admissible). According
to the author’s analysis, if the IDS system does not preserve the raw event data and utilizes automated analysis techniques to generate security event records, then in principle the intrusion detection system produces evidence that is contaminated.

3.7 Summary

Prior research into system activity logging has been largely motivated by the needs of security analysis in general and intrusion detection in particular. Most discussions of system activity logs have been limited to data collection for security analysis. However, a few papers have referred to various management and administration issues such as log storage, retention, integration and reduction.

In terms of data collection, there are three primary reasons why existing event log facilities have proven inadequate for security purposes. These are the inability to log certain kinds of system activity, inability to log specific events to the exclusion of others and inadequate characterization of events.

Some researchers have discussed the possible forensic use of intrusion detection systems. However, the fundamental principles underlying effective intrusion detection systems conflict with forensic requirements. Further, no substantive work has been carried out in this area. Therefore, there are no grounds for the belief that intrusion detection systems are more capable of storing forensically useful information than any event data collection application. Further, IDS systems use analysis techniques on the original data that are likely to render any evidence contaminated. Therefore, prior research conducted in the context of intrusion detection systems is unlikely to be relevant to this thesis.

Chapters 2 and 3 provided the background context to the two research sub-questions presented in the introduction to this thesis.

Chapter 2 discussed the background to the evidence collection problem and the role system activity logs may play in collecting digital forensic evidence within organizational environments. Chapter 5 will build upon this discussion by constructing weighting criteria that can be applied to log evidence.
Chapter 3 discussed the motivations behind prior research related to event logging. Inadequacies in design architecture that may influence data collection of any kind including that of evidence collection were identified.

The following chapter explores the mechanics of the logging process in more depth using a real-world operating system. A simple high-level representation of the logging mechanism will be constructed based on key architectural constraints of the real world operating system that influence evidence collection capability.
Chapter 4

Modeling the Event Logging Mechanism

The purpose of this chapter is to investigate the capability of operating systems’ logging to create an historical record of system activity in digital environments. This chapter develops a generic ‘technology-independent’ model of an event reporting service based on high-level architectural features of mainstream operating systems. For the purpose of demonstration in this chapter, the Windows operating system model has been used.

Section 4.1 introduces the Windows event logging system. An architectural overview of Windows lays a foundation for further discussion on the logging of system activity. The three main activity logs found in Windows operating systems, namely, the security, system and application logs are described and the merits of the Security log are highlighted. The aims of Windows security logging are presented followed by a detailed description of Windows’ security logging capabilities and log file structure.

Section 4.2 describes how real-world operating systems use low-level instructions to report high-level system activity. This description lays the foundation for a generalized model of an event log. The model is particularly useful in highlighting the difference between evidence of activity in an event log and the reporting of activity by a logging system. A generalized model of an event log is constructed based on this discussion.

Section 4.3 extends the simple event logging model described in section 3.2. This model is based on three key features that determine the data collection capability of logging mechanisms – ‘Event detection’, ‘Event selection’ and ‘Event description’. Event detection is the ability to log events using particular system events as triggers. Common limitations of event detection such as the inability to log events regarding network, user-level, and application activity are discussed. Event selection is the ability to nominate events for logging prior to their occurrence in the system environment. This discussion focuses on issues of control over selecting individual events to the exclusion of others, lack of guidance provided to administrators in selecting useful sets of event data and the potential impact of
selection facilities on system performance. Event description is the ability to record an information-based description of an event. Forensic reporting guidelines that have universal applicability are presented and deficiencies in event information frequently recorded by operating systems are pointed out.

4.1 Windows System Activity Logging

The event logging service in Windows, as in all mainstream operating systems, is a passive security auditing mechanism (Murray 1998). It does not search out activity to log but rather operates in the background waiting for reports of events from processes running on the system. Each event is recorded as an entry in one of three logs.

Although the following sub-sections focus on the Windows event logging function, related information such as the fundamental architecture of Windows is also presented.

4.1.1 Fundamental Windows System Architecture

According to Silberschatz (2002), the purpose of an operating system is to ‘provide an environment in which a user can execute programs in a convenient and efficient manner’. Silberschatz indicates that a primary aim of an operating system is to ensure correct operation of the computing system as a whole and prevent user programs from interfering with the processes responsible for managing the computing environment.

The Microsoft Windows family of operating systems includes Windows 2000, Windows XP, and Windows Server 2003 (Windows Vista had not been released in time for consideration in this thesis). The architecture of these operating systems is designed to be ‘modular’. Performance-sensitive parts of the operating system are moved into kernel mode where the overhead incurred by high frequency context switching can be avoided. Code from these parts of the operating system is prohibited from accessing each other unless passing information across defined interfaces.

In the classical model of an operating system, interference from applications is prevented by the use of ‘modes’. Applications run in a restricted ‘user mode’ where they have limited access to system resources. The operating system runs in an unrestricted ‘kernel mode’ where hardware and other resources can be directly accessed and the integrity of essential functions
and data structures of the operating system can be maintained. User programs can access privileged functions by calling a system service. The result is a context switch and the system changes from user mode to kernel mode in order to carry out the desired request. Once the request is completed, the system returns to the user mode allowing the calling program to continue functioning.

Modern operating systems like Windows are primarily used as a platform for applications that perform a range of functions. These applications run in user mode and utilize system resources by calling upon the operating system to act on their behalf. To protect the integrity of the core operating system, Windows separates ‘user mode’ from ‘kernel mode’. Applications and those parts of the operating system running in user mode have only restricted access to system resources while kernel mode typically features unrestricted access to system memory and external devices.

The base services offered by Windows run in kernel mode and are known as the ‘Executive’. In Windows NT, the Executive includes the Object Manager, the I/O Manager, the Security Reference Monitor (SRM), the Virtual Memory Manager, the Process Manager, Plug and Play (PnP) Manager, and the Local Procedure Call Facility. Other significant services running in kernel mode (besides the kernel) are the Window Manager and the Graphics Device Interface.

The Object Manager is a resource management service that acts as the gatekeeper for access to all Windows resources including physical objects (eg external media drive) and logical objects (e.g. logical file, registry key, printer). The SRM enforces security checks where resources are requested for utilization. The I/O manager manages communication with external devices such as the hard disk and the network card. It is responsible for file systems, device drivers and network drivers. The Virtual Memory Manager controls paging memory in and out of physical memory and the Process Manager handles process and thread creation and termination. The PnP manager supports device detection and installation at boot time and starts/stops services on demand.

The display system has two components – the Window manager draws windows and menus and handles input events from the keyboard and mouse, and the GDI draws lines and curves, renders fonts and handles palettes.
4.1.2 Activity Logs in Windows Operating Systems

The large number of services offered by Windows for the purpose of managing the computing environment and fulfilling application requests gives an indication of the range of system activities taking place in the digital environment.

For the purpose of documenting system activity, Windows provides three separate logs – the application log, the system log and the security log:
Chapter 4: Modeling the Event Logging Mechanism

Application log
The application log does not record activity inside the application environment. Instead, it is designed to be used by applications to report errors or convey some information on the state of a process. Examples of such events would be ‘failure to update virus software’ and ‘failure to allocate memory’. Although Windows provides an application log, it doesn’t monitor how applications use that facility. It is common to see the application filled with many instances of a few types of entries as few applications try to record events.

System log
As its namesake suggests, the system log is a non-configurable, repository of messages relating to Windows system components such as the operating system or hardware subsystems. Operating systems engineers use the system log to diagnose problematic hardware or software. The kinds of events likely to be seen in the system log are (Murray, 1998):

- Device drivers failing to load
- Devices causing I/O port or Interrupt Request conflicts
- Installed components found missing from a system configuration or not responding properly
- Low-level system or network data transport errors

Security log
The security log records events such as valid and invalid logon attempts, application startup and shutdown, as well as events related to object usage that are needed to create, open, or delete files. An administrator can specify which events are recorded in the security log. For example, if logon auditing has been enabled, attempts to log on to the system are recorded in the security log (an in-depth description of the security log follows in section 4.1.4).

The Windows operating system does not regulate or in any way control, the generation of events in the application log. Although the application and system logs share the same Windows logging infrastructure as the security log, they rely entirely on application and system component designers respectively to decide what ‘event’ warrants an entry, when to log this entry and what information to pass to the log. Perhaps as a result of the lack of
control exerted on application and systems logging, there is almost no documentation to be found that describes either of the two logs and the significance of log entries.

Like the application and system logs, the security log is also a centralized repository of information generated by various components in the operating system. However, systems administrators control whether events are generated at all and what kinds of events are generated in what circumstances. Windows provides a switchboard type selection interface by which administrators can choose which events to log and the resulting volume of events (see ‘event generation using selection interface’ - section 4.1.6).

Microsoft has designed security logging in a systematic manner as it aimed to make Windows ‘C2 – security level’ compliant, according to the NSA’s Trusted Computer System Evaluation Criteria (TCSEC). A primary requirement to achieve the C2 security rating is a ‘security auditing’ capability. However, designing a logging service for security does not necessarily imply that the same service will provide an optimal forensic solution. This will be demonstrated in the evaluation section of this thesis.

4.1.3 Windows Security Activity Logging Goals

C2 security rating requires operating systems to implement a secure logon facility, discretionary access control, object reuse protection and auditing (Russinovich & Solomon, 2004). To logon to a C2 compliant system, users must present a unique identifier and corresponding password, after which an authentication process must be performed prior to access being granted. Discretionary access control permits users to determine which access restrictions may be applied to resources owned. Object-reuse protection prevents users from accessing data that has been previously used, and then later released for reuse by the operating system.

Pertaining to auditing, the TCSEC states (note that TCB is Trusted Computer Base – the part of the computing system that enforces security policy of a secure operating system and an ADP system is an Automatic Data Processing system) (TCSEC 1985):
“The TCB shall be able to create, maintain, and protect from modification or unauthorized access or destruction an audit trail of accesses to the objects it protects. The audit data shall be protected by the TCB so that read access to it is limited to those who are authorized for audit data. The TCB shall be able to record the following types of events: use of identification and authentication mechanisms, introduction of objects into a user's address space (e.g., file open, program initiation), deletion of objects, and actions taken by computer operators and system administrators and/or system security officers, and other security relevant events. For each recorded event, the audit record shall identify: date and time of the event, user, type of event, and success or failure of the event. For identification/authentication events the origin of request (e.g., terminal ID) shall be included in the audit record. For events that introduce an object into a user's address space and for object deletion events the audit record shall include the name of the object. The ADP system administrator shall be able to selectively audit the actions of any one or more users based on individual identity. “

C2 security rating requires the following types of activity to be logged:

- Use of identification and authentication mechanisms
- Introduction of objects into a user’s address space (e.g. file open, program initiation)
- Deletion of objects
- Actions taken by computer operators and system administrators and/or security officers
- Other relevant events

In the case of authentication events, the source of the request must be identifiable from the audit record. For object-introduction events, the object actor must be identifiable from the record. Finally, the system administrator must be able to selectively audit actions of one or more users based on their identity.

The most influential C2 security requirement on system activity is the auditing of the introduction of objects into a user’s space. Since all resources in Windows is an object (e.g. files, printers, devices, registry), this requirement can be roughly termed as the need to track resource usage. Since operating systems largely manage the interaction between processes
and resources, therefore auditing of this category is likely to provide the greatest insight into system activity in general.

On the one hand, one may argue that C2 security requirements serve forensic interests by citing the requirement to track resource usage. However, on the other hand, limiting the use of authentication to the start of a computing session rather than at key events where the responsibility for event initiation is critical is an example of how C2 requirements do not incorporate forensic goals.

4.1.4 Windows NT Security Audit Policies
Windows NT security logging can be divided into two categories. The first category is focused on tracking resource usage that generates evidence of system activity occurring in the digital environment. Windows allows only three kinds of system activities to be logged (described below). The second category tracks the use of a particular privilege or right such as changing the system time or adding a computer to a domain.

Event Categories related to tracking resource usage:

**Account Logon/Logoff**: These events audit (log) each instance of a user logging on to or logging off from a computer. The security log records whether the logon request was made from the terminal (interactive) or from the network etc.

This feature allows evidence of all user login sessions to be preserved in the log. Therefore, users’ identity can potentially be matched with login sessions to determine when, where and how long a login session had been.

*Account Logon/Logoff Events:
Successful logon, Unknown username or bad password, Account currently disabled, Logon type restricted, Password expired, Unsuccessful logon, User logoff
(See Appendix A-1 for sample log entries)*

**Process Tracking**: Events in this category record significant process activity from which specifics regarding application activity can be extracted.
From this category evidence identifying which applications are running, when they started up and exited, and so forth can be extracted.

Process Tracking Events:
New process has been created, Process has expired, Handle duplicated, Indirect access to object
(See Appendix A-2 for sample log entries)

Object Access: Logging accesses to objects is a two stage process under Windows. Object-access must first be enabled and the administrator must manually tag particular objects for events (regarding that object) to be generated.

From this category, process requests including access to files, printers, and registry keys can be logged. The particular application that initiated the request and the object that was the target of the request can be identified. Also, the length of time that the object was available for accessing can be determined.
Object Access Events:
Object Open, Handle Closed
(See Appendix A-3 for sample log entries)

Event Categories related to Environment Management:
Policy Change: Registers change in administrator choice of which events will generate log entries. This security setting determines whether to audit every incident of a change to user rights assignment policies, audit policies, or trust policies.
Policy Change Events:
User right assigned, User right removed, Audit policy change

Privilege Use: Determines whether to audit (log) each instance of a user exercising a user right. In this context Microsoft has defined a pool of specific rights such as ‘Access this computer from the network’ and ‘Add workstations to domain’. For most user rights, Windows logs a ‘Privilege Use’ event when a user exercises the right. Some of these rights can be assigned to users to elevate their privileges (for that right only) to administrator level.
Privilege Use Events:
Special Privilege Assigned, Privileged Service Called, Privileged Object Operation

**System Events:** Logs particular system events such as whether to audit (log) when a user restarts or shuts down the computer or when an event occurs that affects either the system security or the security log. Note that Microsoft has attached a new meaning to ‘system events’ in Windows. Traditionally, a system event is any significant occurrence in terms of system activity. Windows uses the same term to mean any significant occurrence in terms of the computing system as a whole. Wherever ‘system event’ is used in this thesis, the latter definition is implied unless otherwise stated.

*System Events (Events):*
- System Restart
- System Shutdown
- Authentication Package Load
- Logon Process Registered
- Some Audit Event Records Discarded
- Audit Log Cleared

**Account Management:** Events in this category can be used to monitor changes to user accounts and groups.

*Account Management Events:*
- User Account Created
- User Account Changed
- User Account Deleted
- Global Group Member Added
- Global Group Member Removed
- Local Group Created
- Local Group Member Added
- Local Group Member Removed
- Local Group Changed
- Local Group Deleted

**4.1.5 Windows Security Log Structure**

Each entry in the security log has a header and a description. The header is a set of standard fields that are available for every type of event. These fields record the event ID, date and time the event took place, the outcome of the event (success/failure) and the event’s source and category (see event header).

<table>
<thead>
<tr>
<th>Event Header Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date: 10/31/2006</td>
</tr>
<tr>
<td>Time: 2:30:13 PM</td>
</tr>
<tr>
<td>Type: Failure Audit</td>
</tr>
<tr>
<td>User: XXX &lt;NT Authority\SYSTEM&gt;</td>
</tr>
<tr>
<td>Computer: CALADAN</td>
</tr>
<tr>
<td>Source: Security</td>
</tr>
<tr>
<td>Category: Account Logon</td>
</tr>
<tr>
<td>Event ID: 680</td>
</tr>
</tbody>
</table>
Table 4-1: Event Header Fields

<table>
<thead>
<tr>
<th>Information</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date</td>
<td>The date the event occurred. The date and time of an event are stored in Universal Time Coordinate (UTC) but always display in the viewer's locale.</td>
</tr>
<tr>
<td>Time</td>
<td>The time the event occurred. The date and time of an event are stored in UTC but always display in the viewer's locale. It is not clear whether the time recorded is when the log entry was generated or when the event was reported.</td>
</tr>
<tr>
<td>Type</td>
<td>A classification of the event severity: Error, Information, or Warning in the system and application logs; Success Audit or Failure Audit in the security log. In the Event Viewer normal list view, these are represented by a symbol.</td>
</tr>
<tr>
<td>User</td>
<td>The name of the user on whose behalf the event occurred. This name is the client ID if the event was actually caused by a server process, or the primary ID if impersonation is not taking place. Impersonation occurs when the server allows one process to take on the security attributes of another. Where applicable, a security log entry contains both the primary and impersonation IDs.</td>
</tr>
<tr>
<td>Computer</td>
<td>The name of the computer where the event occurred. This is usually the name of the local computer, unless an event log from another computer is being viewed.</td>
</tr>
<tr>
<td>Source</td>
<td>The software that logged the event, which can be either a program name such as SQL Server, or a component of the system (such as a driver name) or of a large program. For example, &quot;Elnkii&quot; indicates an EtherLink II driver. The Source always remains in its original language.</td>
</tr>
<tr>
<td>Category</td>
<td>A classification of the event by the event source. This information is primarily used in the security log. For example, for security audits, this corresponds to one of the event types for which success or failure auditing can be enabled in Group Policy by a member of the Administrators group.</td>
</tr>
<tr>
<td>Event</td>
<td>A number identifying the particular event type for this source. The first line of the description usually contains the name of the event type. For example, 6005 is the ID of the event that occurs when the Event log service is started. The first line of the description of such an event is &quot;The Event log service was started.&quot; Using the values of Source and Event together, product support representatives can troubleshoot system problems.</td>
</tr>
</tbody>
</table>

Table 4-2: Meanings of Event Header Fields

The Windows description can also be broken down into two types of information. The first is a static template with defined placeholders into which relevant information (dynamic strings) regarding the event is inserted. The second is the dynamic strings themselves (see figure below). The outcome is the merged description that constitutes a single event log entry (see ‘merged description’ on right hand side of figure below) [3]. Note that the merged description
includes a variable-length “Description” field that lists more detail regarding the event. Some of this detail is not available in the static part of the event record. When an event is generated, this information on is collected by the event log service and stored in a separate ‘message’ file. The message file is also merged with the event record when displaying the complete event log in the Event Viewer.

<table>
<thead>
<tr>
<th>Static Description</th>
<th>Dynamic Strings</th>
<th>Merged Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logon Failure</td>
<td>Administrator</td>
<td>Event Type:</td>
</tr>
<tr>
<td>Reason: Unknown</td>
<td>STUDENT01</td>
<td>Failure Audit</td>
</tr>
<tr>
<td>user name or bad</td>
<td>3</td>
<td>Event Source:</td>
</tr>
<tr>
<td>password</td>
<td>NtlmSpr</td>
<td>Security</td>
</tr>
<tr>
<td>User Name: %1</td>
<td>MICROSOFT_</td>
<td>Event Category:</td>
</tr>
<tr>
<td>Domain: %2</td>
<td>AUTHENTICATION_</td>
<td>Logon/Logoff</td>
</tr>
<tr>
<td>Logon Type: %3</td>
<td>PACKAGE_V1_0</td>
<td>Event ID: 529</td>
</tr>
<tr>
<td>Logon Process: %4</td>
<td>STUDENT01</td>
<td>Date: 10/31/2005</td>
</tr>
<tr>
<td>Authentication</td>
<td></td>
<td>Time: 2:51:58 PM</td>
</tr>
<tr>
<td>Package: %5</td>
<td></td>
<td>User: NT AUTHORITY\SYSTEM</td>
</tr>
<tr>
<td>Workstation Name:</td>
<td></td>
<td>Computer: CALADAN</td>
</tr>
<tr>
<td>%6</td>
<td></td>
<td>Description:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Logon Failure:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Reason: Unknown user name or bad password</td>
</tr>
<tr>
<td></td>
<td></td>
<td>User Name: Administrator</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Domain: STUDENT01</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Logon Type: 3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Logon Process: NtlmSpr</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Authentication Package: MICROSOFT_AUTHENT...</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Workstation Name: STUDENT01</td>
</tr>
</tbody>
</table>

**Table 4-3: Event Description**

In the above case, string 1 (%) determines the user who failed to log on. String 3 (%) indicates how the user tried to logon, in this case it was via the network (logon type 3).

Note that an entry in the Windows Security Log is either of type ‘Success audit’ or ‘failure audit’ [5]:

<table>
<thead>
<tr>
<th>Event type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success Audit</td>
<td>Any audited security event that succeeds. For example, a user's successful attempt to log on the system will be logged as a Success Audit event.</td>
</tr>
<tr>
<td>Failure Audit</td>
<td>Any audited security event that fails. For example, if a user tries to access a network drive and fails, the attempt will be logged as a Failure Audit event.</td>
</tr>
</tbody>
</table>

**Table 4-4: Meaning of Success/Failure Audit**
4.1.6 Event Generation Using the Selection Interface

Administrators use a GUI interface to configure security event log settings. The Security Settings \ Local Policies \ Audit Policy directory in the Local Security Policy console shows the following view (figure 4.2). By selecting one or more audit policies, administrators configure the operating system to generate entries upon the occurrence of particular events.

![Figure 4.2: List of Audit Policies](image)

Windows features an ‘all or nothing’ selection mechanism for logon/logoff auditing. This pair of events triggers log entries for all users logging on to a Windows system or none at all. System administrators are faced with the same choice regarding the creation/termination of processes (the figure below shows audit settings for process tracking – a similar choice exists for logon/logoff).
Windows features a two-step selection mechanism for object-access logging. Once Object Access is turned on, administrators must choose to manually tag individual objects before log entries will be generated. Figure 4.3 shows the auditing settings for a file object. The window on the right hand side appears when right clicking on an individual object. Windows allows usernames or groups to be entered so that when a process running on behalf of a listed username or group requests access to this object a log entry will be generated. The window on the left hand side lists the types of accesses permissible on the object. Note that each audit setting may list different settings for individual users or groups allowing fine granularity of selection on object accesses.

It is relevant to note that Windows logging in the Executive is focused primarily on the Object Manager’s functions whereas other key components such as the Window Manager (user interface events) and I/O Manager are not covered.
Chapter 4: Modeling the Event Logging Mechanism

Figure 4.4a: Object Access Selection

Figure 4.4b: Object Access Selection
4.1.7 Event Log Service

In the Windows environment, the event logging function is implemented as a Windows service. A Windows service is a process running in the background and must be differentiated from ‘application service’. The former is a Windows technical term whereas the latter is a general term used to describe a running application.

The event logging service controls access to all three event log files such that no other process may have direct interaction. Instead, access to the log files is gained by requesting the event log service conduct the operations on behalf of external processes.

The event logging service has a number of components. These are the three event logs, the message files, the registry, the Application Programmer Interface (API), and the Event Viewer.

The three event logs are stored in three separate files. They are formatted in a proprietary binary record format (.EVT) and are not readable by most word processing / text editors (they are not encrypted however).

The event log itself does not contain the description information visible in Table 4-3. Log entries visible in the Event Viewer display information coming from two sources. The static part of the log entry comes from the event log whereas the description detail comes from message files. Message files are separate from the log files because they are designed to hold ‘localized’ data. In Windows, all language-dependent information is referred to as ‘localized’ data. When the event log reads a single record from the event log it looks up the registry for the location of the localized strings. A match is made between the event sources that are recorded in the entry record in the event log with the event sources in the registry. A match leads to the location in the message file where the localized data is found. This data is then merged with the existing record in the event log for a complete event report viewable in the Event Viewer. Note, other than registering a map of event sources, the registry also stores all configuration and policy information regarding event logging.

The splitting of the event log into three components (static, message and registry) is unique to Windows. Although it makes some sense to implement this system from a ‘localization’ perspective, it dramatically increases the complexity of the security problem (especially the
integrity of forensic evidence) as it requires all three components and their integration to be secured from unauthorized influence.

4.1.8 Security and Retention
Windows protects the event logging service by accepting it as an internal component of the operating system. The event logging service may be configured to start on boot-up by the Service Control Manager (SCM) or manually started by the systems administrator. However, once started, the event log service cannot be manually stopped by a user (or super-user) on the system.

As previously mentioned, the integrity of the event log files is protected by disallowing applications and other processes to directly interact with event log files. Instead, requests can be made through the API to read/write to event log files. Note that more recent models of Windows, such as Windows XP, do not allow third parties to write to the security log at all.

In general, the security of any event logging system is as strong as the security of the computing system. However, Windows has implemented a number of measures by default to make it difficult for an external party to compromise the event logging service.

Event log files are opened when the service is started and cannot be shared with any other process other than SERVICES.EXE. Only one process may read or write to the event log files at one time and the log files are never closed until the service shuts down.

The log files cannot be directly copied from the system as the SCM changes the format of the file slightly so it appears to be corrupt to other processes. However, event log files can be backed up using the BackupEventLog function in the API.

The event logging service has been known to hang under extreme loading conditions (see Knowledge Base Q164938 - referenced in Murray p.34). Also, note that if the system is rebooted under MS-DOS, the event log files are no longer protected by the above measures.

The purpose of a retention policy is to determine what Windows should do when an event log reaches its maximum allowed size. Windows provides three retention policies for its event logs. The default policy is to overwrite the oldest events by newer events. A variation on this
policy is to set a certain time period after which overwriting on events older than that point can proceed. A third possibility is to drop any events arriving after the log is full (until the log is cleared).

### 4.1.9 Viewing Logs in Windows

Windows provides only one tool to view the binary file format of event logs. The event viewer allows either the application, system or security log to be viewed at one time. It displays the event log in a record format, allows the administrator to set the size of the log and retention policy, clears and saves the log, and provides rudimentary filtering facilities.

![Figure 4.5: Event Viewer](image)

The view window lists the three kinds of logs on the left pane and the log entries on the right (figure 4.5). The right pane only shows a summarized view of each log entry. To see a
complete log entry, an event’s properties must be viewed (see figure ‘merged description’ part of figure 4.3).

4.2 A Simple Model for Logging System Activity

In modern operating systems writing a log-entry can be achieved by making a single function call. However, system architects exercise considerable influence on what information ultimately appears in the log. This influence stems from three avenues. Firstly, system architects decide how many events can be logged in an arbitrary piece of code and precisely from between which lines of code can log entries be reported. Secondly, system architects construct facilities that systems administrators use to decide which events should generate a log in an arbitrary logging session. Finally, system architects decide what information is written in a log entry.

4.2.1 Reporting High-level System Activity Using Low-level Instructions

Modern computing systems typically execute a very large number of instructions in a short period of time. A rough calculation of the numbers involved is useful to illustrate the point. For example, a modern computing system running a Windows operating system is likely to be measured in kilo MIPS or in excess of one thousand – million instructions per second.

In principle, even a small group of instructions that performs a significant system activity is potentially an event, although it is obvious that not all events are worthy of logging and neither is that possible if acceptable system performance levels are to be maintained. The decision as to what system activity may be logged is left to system architects. From a systems perspective, this decision relates to the desired outcome or purpose of the logging facility.

Windows auditing was designed for troubleshooting and performance monitoring. Lately, considerable effort has been invested into making the logging facility C2-compliant. As a result, Microsoft system architects have provided a number of security events of which three categories are particularly related to system activity. These are logon/logoff, process tracking and object-access. A closer look at one of these, process-tracking, serves to highlight an important concept relating to levels of abstraction.
According to Russinovich and Solomon (2004), there are six stages in the creation of a process:

1. Open the image file (.exe) to be executed inside the process.
2. Create the Windows executive process object.
3. Create the initial thread (stack, context, and Windows executive thread object).
4. Notify the Windows subsystem of the new process so that it can set up for the new process and thread.
5. Start execution of the initial thread (unless the CREATE_SUSPENDED flag was specified).
6. In the context of the new process and thread, complete the initialization of the address space (such as load required DLLs) and begin execution of the program.
Every action in figure 4.6 may be considered a system event. However, Windows architects chose to represent the entire system operation with a single system event (Russinovich & Solomon, 2004), Microsoft system architects decided to position this event-reporting request close to the end of stage 2, just before the primary thread of the new process begins to execute). This one-to-many relationship, where a single system event of a system operation featuring multiple events is logged, is important from a forensic perspective as it illustrates the point that the Windows operating systems does not log evidence of every event executed, but instead logs one event in a train of events. This self-imposed constraint of limiting logging of an operation to a single system event is true of all operations logged by Windows (an exception would be operation-based logging introduced in Windows XP – see section 8.1.6). Later chapters will show that this (inadequate) representation of system operations due to the choice of system event and the limit of a single event per system operation has significant influence on the evidential weight of Windows activity logs.

The following fragment of pseudo code is a set of instructions that ends in a logical decision that makes the ‘event’ available for logging:

```
Instruction 1;
Instruction 2
.
.
Instruction m;
If (flag==SELECTED)
  WRITE_EVENT(parameter1, parameter2, ....parameter_n);
Instruction m +1;
.
.
Instruction m + n;
```

Table 4-5a: Code for a System Operation

The purpose of the logical decision is to give administrators the choice whether to log or not to log the event. Without the logical decision, every time the code ran it would generate a log entry regardless of whether it was required or not. If the code fragment was executed a large number of times in a short period there may be significant performance degradation. The
WRITE_EVENT function passes a series of parameters (typically in the form of a data structure that has been previously populated) to be recorded into the log entry. Notice that by employing more sophisticated logic at the decision-making point, system architects can give administrators more options. For example, the if-then statement may allow the event to be generated if a particular user was logged in. There is a trade-off here, as this would increase the amount of control given to the administrator but incur the processing cost of resolving the decision every time the code fragment was executed.

4.2.2 Reporting Real-World Activity Using System Operations

The previous section identified a one-to-many relationship between a group of system events executed and a representative log entry recorded. In this thesis, the occurrence of a group of instructions performing a significant system act is called a 'system event' whereas an event at a higher level of abstraction such as ‘create process’ and ‘logon’ is called a ‘system operation’.

Forensic investigations typically examine log evidence in the context of events of significance to the real world, such as the usage of email, web browsing, and document management (section 2.2.2). Events in the real world are usually described at a higher level of abstraction than events represented by individual log entries. Therefore, another one-to-many relationship comes to light. A real world event such as ‘send email’ may involve more than one system operation of the type ‘create process’, ‘connect to network’ and ‘end process’.

The same feature can be seen in UNIX operating systems. Section 3.4.1 referred to an example from (Price 1997). Price pointed out that although computing systems are frequently seen as fundamentally executing subject actions on objects, these are not executed atomically. She presented the example of a write-to-file action in the context of UNIX operating systems. In this case a “write-to-file” operation may be broken down to multiple system events. Typically, only one of these events triggers an event log entry that represents the complete write operation.

Therefore, in the discussion so far three different levels of abstraction have been identified at which system activity can be described:
1. Real world act/event: a high-level action undertaken by a subject; e.g. ‘send an email’

2. System operation: executed in multiples in response to a real world act; e.g. ‘create process’,

3. System event: typically executed in multiples as part of a system operation; e.g. ‘create stack’, ‘create thread’ (note: these may be broken down into even lower levels of abstraction, however this does not serve any further purpose from a forensic perspective).

The first of these abstractions describes a real-world view of activity taking place on the system. ‘Sending an email’, ‘browsing a website’ or ‘reading a document’ or ‘writing a document’ are examples of real-world actions. At this level of abstraction, the focus is on activities taking place in the digital environment but is meaningful in a real world sense.

The second level of abstraction describes system behaviour in response to a real-world action. For example, the sending of an email will include multiple system actions like ‘opening a file’, ‘writing a file’, ‘connecting to a mail server’ and so forth. I call these system operations. Although activities at the second level describe system behaviour, these are not atomic. A single system operation such as a ‘write to a file’ may consist of multiple system events as described in point 3 above (‘request to write’, ‘initiate write’ etc.). These system events reflect what the system is doing at its most fundamental (atomic) level. The number of system events that typically take place even in a small period of time is very large.
4.2.3 A Formalized Event Reporting Model

System activity can be abstractly described in terms of actors, actions and timing information (section 3.4). This approach can be described using the simple notation below:

\[ \text{Real_world_activity} = \langle \text{system_operation}_1, \text{system_operation}_2, \ldots, \text{system_operation}_l \rangle \]

where

\[ \text{system_operation} = \langle \text{system_event}_1, \text{system_event}_2, \ldots, \text{system_event}_m \rangle \]

and:

\[ \text{system_event}_1 = \langle \text{actors}_1, \text{action}_1, \text{timing}_1, \text{location}_1 \rangle \]
\[ \text{system_event}_2 = \langle \text{actors}_2, \text{action}_2, \text{timing}_2, \text{location}_2 \rangle \]
\[ \vdots \]
\[ \text{system_event}_n = \langle \text{actors}_n, \text{action}_n, \text{timing}_n, \text{location}_n \rangle \]

Table 4-5b: System Activity in High-level Notation

To formalize this model even further, the environment where system events are taking place can be described. Take an arbitrary computing system. There exists a set such that each element of this set is a system event that may potentially take place in the computing system’s environment. Let us aggregate all system events that can potentially take place in a computing system into a single (universal) set \( E \). Note that the potential set of system events for a computing system changes depending upon what is running in the digital environment at the time. A formal method of stating this follows.

Let \( e \) be a system event that can take place in a computing system. Therefore \( E \) is a set of system events \( \{e_1, \ldots, e_n\} \) denoting all such system events that can potentially take place in a computing system.

Each such system event can be described by a set of attributes. These attributes describe the system event but do not constitute the event themselves. There is a one-to-one relation
between the occurrence of a system event and the entry describing the event. In other words when a system event occurs, only one event entry may be written to the log.

Let \( A_1, \ldots, A_n \) be sets of attributes for describing system events; where each \( A_i = \{a_{i1}, \ldots, a_{im_i}\} \)

In other words \( A_1 \) can be a set of actors, \( A_2 \) can be a set of Actions and so forth. Each set may not have the same number of elements which is useful, as the possible subjects will not be the same number as the possible actions or the possible times and so forth.

A series of attributes describing an event as an event entry may be defined. Each attribute \( x_i \) is a member of the corresponding set of attributes \( A_i \).

An event entry is a tuple \( (y_1, \ldots, y_n) \in A_1 \times A_n \).

Note: each \( y_i \in A_i \)

EE is defined as an ordered set of event entries representing an event log. Note that due to changes in configuration of the system (for instance, the installation and removal of key components of the operating system and applications) \( E \) does not actually remain static.

### 4.3 Design Factors that Influence Evidence Collection Capability

The aim of this section is to identify the generic design factors that influence evidence (event data) collection. To identify these design factors, a high-level model must be developed as no models have been suggested by previous research (section 3.2). Windows will be used as a basis for developing this model.

Section 3.2 reproduced the following event log model taken from (Price, 1997):

![Figure 4.7: A Simple Event Log Model](image)

**Figure 4.7: A Simple Event Log Model**
Evidence of system activity can be collected in Windows using a two-stage process. In the first stage, the selection interface must be used to ‘tag’ certain events for evidence collection. For example, the Windows selection interface allows evidence of a successful logon followed by a successful logoff to be preserved by ticking the logon/logoff checkbox. The second stage occurs when the tagged event takes place in the digital environment. In this case a log entry is written to the log.

From the above two stages, it is apparent that amount and type of evidence generated in a Windows log is dictated by three features.

**Factor 1: Event Reporting Calls**
The design of event reporting in the operating system influences the kind of system activity that is reported. Although much system activity takes place in a short period of time, a very small subset of this activity is available for logging purposes. Windows allows logon/logoff, process tracking and object-access to be logged, however a range of other activities such as application interactions with the network cannot be logged. Further, within each system operation such as process tracking, there is no choice of which system events can be logged to represent the system operation. This affects the amount of evidence that may be generated in a log.

**Factor 2: Selection Policy**
The amount of evidence generated in the log regarding system activity is influenced by the selection policy adopted by a systems administrator. For example, selecting logon/logoff but not selecting process tracking implies evidence of every logon by Windows users will be available in the log but process creation/termination will not be logged.

**Factor 3: Log Entry Write**
The set of evidence (information) written to individual log entries is the third factor that influences the amount of evidence generated in a log.

Therefore, the design of these three features influences the capability of the logging mechanism to collect event data. Labels are now applied to each feature, and they are described more formally in terms of the capability of the logging mechanism to log the generic ‘event’.
Chapter 4: Modeling the Event Logging Mechanism

1. Event Detection: The ability to log particular events as opposed to others
2. Event Selection: The ability to ‘tag’ or nominate events for logging prior to their occurrence in the system environment.
3. Event Description: The ability to record a set of information describing the event that occurred.

The following sections explore each of these features in more depth.

4.3.1 Event Detection
In terms of the limitations of event detection, section 3.4 presents two key observations that are summarized as follows. Firstly, conventional operating systems in general neglect to log a broad range of system activity. Secondly, events are not detected at an atomic level, allowing inaccurate recognition of system activity. This means system architects do not report events for every atomic action. Instead, particular system events considered representative of higher-level system activity are made available for logging.

With regard to the first observation, any passive event logging service that functions on the same system within which events are being logged will have limitations. For example, there will be certain system activity that occurs at bootup time or at shut down time that cannot be logged. This is logical because an operating system shut down will involve a series of activities. One of these (not the last) will be terminating the logging service. Any activities following the termination of the logging service will not be logged. The same is true for boot-up. Certain activities must occur before the logging service can be started. All such activities cannot be logged. To take this logic even further, each act of logging is in itself a candidate for further logging. Allowing an event log to accept logging of its own activities will create a circular process.

The following section, adapted from Russinovich and Solomon (2004), gives a brief overview of the windows bootup process. Note that the last modules to load up into the Windows system prior to the logon screen appearing are Session Manager Subsystem (SMSS) and Winlogon. The fact that these two modules are primary players in event logging, implies that the majority of the instructions involved in the boot up process, including hardware checks and services, cannot be currently logged by Windows.
Chapter 4: Modeling the Event Logging Mechanism

The Windows boot up process begins by performing a series of tests to ensure the hardware components are working properly. These components include the power supply, the memory, keyboard and BIOS chip. After that, other devices such as video cards are initialized. When the Power On Self Test (POST) is completed the system begins to boot off the Master Boot Record (MBR). The MBR calls NTldr (NT Loader) which opens up resources on the system such as the RAM banks and the file system. Boot.ini is next in line to be loaded by the NT loader which initializes the boot screen. NT loader then runs Ntdetect which gathers information about the hardware comparing it to a database of known hardware stored in the registry. After the hardware checks are completed the NT loader loads a Ntoskrnl (NT Operating System kernel). This program loads the HAL (Hardware Abstraction Layer) which keeps the kernel separate from the hardware. Ntldr finally loads drivers for the boot drive and hands over control to Ntoskrnl.

The kernel goes through a two-phase initialization procedure. In the first phase Windows assumes control of the hardware, in the second phase the HAL configures the computer to accept data from the hardware. This process includes starting up the Memory Manager, Cache Manager, I/O Manager and other services.

The I/O Manager loads all the other drivers for hardware installed on the system. Another program known as the Session Manager Subsystem (SMSS) prepares the graphics systems for use. The Win32.sys switches the PC into graphics mode. The SMSS then loads all Windows services configured to start automatically at startup, and then two further programs – Winlogon and LSASS. The login screen then loads up and welcomes the user ending the boot up process.

From a forensic point of view the integrity of the system is particularly vulnerable during the boot up phase as operating system instruction code is being executed to create the digital environment. In principle, the laws that govern the digital environment are particularly vulnerable to external influence during this period.
Chapter 4: Modeling the Event Logging Mechanism

For example, a user with physical access to the computing system can change the boot up settings of the computing system to boot off an external source of media thereby making a number of changes to the Windows operating system while the event logging system is not running.

Like in UNIX, events in Windows are also not detected at an atomic level. A discussion earlier in this chapter (section 4.2.1) regarding process creation pointed out that a large number of instructions were executed before and after the only request to log the start of the process. This implies that the majority of system events were not validated to be in order before the log entry was made.

This section demonstrates that a substantial amount of system activity is unavailable for logging by the Windows operating system. It is suggested given previous discussion in section 3.4.1 and given examples of Windows in this section, that this fact applies to operating systems in general and is not limited to Windows or UNIX.

Constraints arising from Event Detection
The following section uses formal notation to describe the previous discussion.

In any particular computing system, there is a set of ‘loggable system events’ designated by system designers and made available through a selection interface. We label this set $LE_s$. Regarding these events we can say the loggable system events are those elements of $E$ for which the system creates event log entries.

The loggable system events of system $S$ is $LE_s \subseteq E$

And therefore it follows by logical extension that the set of non-loggable system events is $NLE_s = E \setminus LE_s$

The set of non-loggable system events of a typical computing system will include application events, network access events and so forth as previously discussed.
The design of a computing system may lead to two different possibilities regarding loggable events. Logically, either all events that may potentially occur are available for logging or only a subset is available. Therefore $\text{LE}_S = \text{E}$; i.e. any event that occurred in the system could be logged by the event log infrastructure. The other possibility is $\text{LE}_S < \text{E}$; i.e. the event log infrastructure can log a subset of E but not all E.

### 4.3.2 Event Selection

When configuring an event log for logging purposes, the administrator must first identify the anticipated events for which a record must be preserved. The list of anticipated incidents is largely contingent upon the characteristic usage of the system and its purpose and functionality in the organization.

The usage patterns exhibited by systems in an organizational setting are likely to indicate that the most common activities taking place are word processing tasks and web browsing. Therefore, possible incidents that may attract an investigation may include downloading illicit content and use of resources to generate content that violates policy (section 2.2.2).

Given this (short) list of anticipated incidents, the next step towards configuration of event logs is to determine in the case of each anticipated incident, precisely which system-level events must be tagged for logging purposes. The event logging selection interface has one primary function. To allow the administrator to select a set of events for which evidence is to be generated. However, the design of the user interface influences the configuration of the event logging mechanism through to the final set of event data collected.
Chapter 4: Modeling the Event Logging Mechanism

The management interface affects event logging in two primary ways.

If \( Z \) is the set of events that must be logged for coverage of a particular incident, then the logging subsystem defines how much control (granularity) the administrator has in selecting a particular event for logging.

This can be seen in Windows’ approach to selection, as it does not provide a single interface for selecting all events. As the introduction to event selection (section 4.1.6) describes, Windows enforces an ‘all or nothing’ approach to generating evidence of logon/logoff and process creation/termination whereas object-access logging features a more fine-grained approach.

The second kind of influence of an interface on the model presented to the administrator, through which the administrator makes choices, is on how to use the interface to enable logging. These facilities are typically not designed to efficiently map (high-level) real world events to entries in the event log. Instead, administrators are presented with a collection of switches representing operating system actions upon operating system objects. Hence administrators wanting to log high-level events are forced to translate their requirements according to the complex and mechanical view of an operating system. Arising from the operating system view is a distinctly different set of questions such as what subjects, objects and actions must be audited?

The Windows selection interface does not allow an administrator to log particular activities such as the usage of a particular application by a single user efficiently. To achieve this goal using Windows, an administrator must first generate evidence of all logons and logoffs as particular users cannot be tracked to the exclusion of other users. Secondly, evidence of the invocation and termination of all application sessions must be generated for the same reasons. Finally, if the administrator wants to identify any accesses to particular files and printers, then each file and printer must be separately tagged for logging. Windows does not provide any guidance on how to capture all activity of a single user. The administrator must determine what system activity must be logged in order to utilize the selection interface.
Therefore, collection of event data rests on two main issues:

- Control over the selection of the event data set
- Guidance given to administrator to assist in collecting sets of useful forensic event data

Although administrators are allowed to select or tag from a limited choice of system events using a management console, frequently systems apply further constraints by preventing the selection of individual system events to the exclusion of others.

**Constraints arising from Event Selection**

The set of selected events SE is defined as those events tagged by the administrator for logging purposes. It follows that \( SE_a \subseteq LE_s \); i.e. the set of selected events by the administrator is a subset of the set of loggable events designated by the system. Therefore the following may be presented:

![Figure 4.9: The Set of Selected Events](image)

As previously mentioned, the selection facilities in conventional computing systems typically do not allow selection of individual events. Some computing systems only allow logging to be turned on or off (SE=LE) effectively forcing all loggable events to be selected. Others allow smaller pre-determined subsets of events to be designated for logging (SE<=LE). In both cases there is a large number of events logged that are not of particular interest to the administrator. Since event logging consumes system resources, the size of SE incurs a performance penalty that typically results in one of the three following scenarios:
1. Event logging degrades system performance to a point where logging is no longer feasible
2. Event logging degrades system performance to a point, forcing logging to be limited to short intervals only
3. Event logging does not degrade system performance, allowing uninterrupted logging

4.3.3 Event Description

The occurrence of an event, that is both loggable and selected, results in the event log writing an entry by preserving the set of descriptive attributes that describe the event from the system environment.

Previous discussion of audit policies in Windows presents sample log entries for logon/logoff, process tracking and object-access (section 4.1.4).

The significance of the information recorded in log entries may be analysed using the standard guidelines for reporting any event (forensic guidelines are used although they are generic). These are summed up in the fundamental who, what, where, when and why questions.

According to O’Connor (n.d.), a (physical) forensic science reconstruction of an event aims to present:
“A plausible theory of who, what, when, where and why the crime happened."

Yasinsac points to a digital forensic reconstruction of events as needing to answer the “who, what, when and where” questions only.

Obviously the ‘why’ question is not for historical records of system activity to address, whereas the ‘where’ is understood to be the digital environment where the system activity is taking place. These questions (also known as the 4Ws) can be translated into information description requirements that can be used to assess the adequacy of an information description of the generic event:
Chapter 4: Modeling the Event Logging Mechanism

1. Who was involved in the event? (Actors)
This question aims to determine the identity of all subjects involved in the event. A brief examination of the Windows log entries will show that system activity in the digital environment involves interaction between three entities - processes, objects and users.

2. What happened? (Action)
This question is focused on identifying the action that constituted the event. Windows applies event identifiers to events. For example, logon is a type of action that is different to other actions such as logoff. They have different event identifiers allocated to them. Similarly, the opening of a file, creation of a process and so forth have allocated event identifier numbers.

3. When did it happen? (Timing)
The timing of the event is of obvious importance to the forensic investigation. Timing information is essential to establishing the sequence of the events in the reconstruction. All Windows log entries have an associated timestamp that dates the action in the ‘big picture’ of session activity.

In general, any event of forensic interest will contain information that can be generally represented by the following attribute categories:

\[
Actors_i = \{actors^i_1, ..., actors^i_m \} \\
Action_i = \{action^i_1, ..., action^i_m \} \\
Time_i = \{time^i_1, ..., time^i_m \}
\]

Therefore \( F \) is a tuple representing an event log entry with the above attributes:

\[
F = \{v_{actors}, v_{action}, v_{time}\}
\]

Where \( v_{actors} \in Actors_i \) and so forth.
Chapter 4: Modeling the Event Logging Mechanism

Constraints arising from Event Description
Recall from section 3.3, a single event entry format is typically used for all events (subject, object, action, etc.). There are a number of limitations including lack of self-contained identification of subjects and objects, lack of information on user-level commands, lack of resource utilization information and inability to distinguish between shell scripts and programs in some operating systems (see criticism of event logging in literature in section 3.4). General incident reporting technique attempts to find the ‘who, what, when, where and why’ of events.

Information preserved for descriptive purposes can be divided into two sets, the first being the information relevant to the system event that exists in the digital environment. The second is the set of information that was recorded in the event entry (see figure 4.5).

![Figure 4.10: The Set of Event Description Information](image)

The occurrence of an event that is both loggable and selected, results in the event log writing a log entry by preserving particular details that describe the event from the system environment.

Given a particular selected event has occurred in a particular computing system, there is a set of information that is logged in an event entry. Given WI is the written information contained in a log entry and RI is the descriptive information identified using the 4W model that is available from the computing system at the point of event occurrence then $WI \subseteq RI$. Incidentally $RI \subseteq I$, where I is all of the information available from the computing system at the time of event occurrence.
4.4 Conclusions

This chapter develops a generic model of an event logging mechanism. The first section presents background on Windows event logging from an architectural and implementation perspective. Three high-level design factors that influence Windows’ logging mechanism’s data collection capability were identified and their limitations investigated after which they were used to form a technology-independent model of event reporting. These are Event Detection, Event Selection and Event Description. Together these three factors determine the data collection capability of system activity logs.

Although this model explains the capability of a logging service to report system activity, it does not account for the difference in abstraction between real world incidents and system level activity. Forensically, this difference is crucial as it models the different levels at which users and computing systems perceive significant events. Systems architects have taken these different levels into account when designing the logging service as they consciously report high-level system activity using system events. The basis of this model is that system activity can be identified at three different levels. These are Real World Acts, System Operations and System Events.

In chapter 6, the event reporting model and the event abstraction model developed in this chapter will be used to conduct a comprehensive analysis of the evidence collection capability of real world operating systems. The following chapter will construct evidential weighting criteria that can be used to assess the reporting capability of logs from a forensic perspective.
Chapter 5: Evidential Weighting Criteria for Event Logs

This chapter develops evidential weighting criteria for event logs based on the universal principles of evidential reliability, namely accuracy, completeness and authenticity. A significant aspect of this contribution is recognition of the ‘utility’ of event logs as a contributing factor to evidential weight. Utility is the usefulness of event logs in complementing the testimony of forensic investigators by establishing the reliability of all computer-derived evidence such as user files from the filesystem.

Unfortunately there is no evidential framework or model for event logs. The existing evidential reliability framework for computer-derived exhibits is developed specifically for human-directed forensic investigation. As the only relevant framework for this discussion was developed specifically for human-directed evidence collection, section 5.1 compares the role of the event log to that of the forensic investigator. The aim of this comparison is to identify similarities and differences between the two roles so that the meaning of the evidential principles of accuracy, completeness and authenticity can be analyzed, interpreted and defined in the context of event logs.

Section 5.2 also focuses on the role of event logs in establishing the authenticity of computer-derived evidence. This section uses tests designed to meet the three evidential reliability principles to develop criteria for the authenticity of computer-derived evidence. An important limitation of the event log capability, is that the authenticity of evidence must be in part established outside the digital environment and after the computing system is suspended for investigation purposes.

A desirable attribute of log evidence is the ability to collect evidence that may complement the testimony of forensic investigators regarding the reliability of all computer-derived evidence derived from the host computing environment. This attribute is investigated in section 5.4.
5.1 The Role of Event Logs in Evidence Collection

Event logs are engaged in passive recording of system activity for the purpose of later review (section 3.2). Event logs are history files containing information descriptions of system activity taking place in the computing environment. They preserve information about volatile system activity onto hard disks that can be later retrieved by a forensic investigator. Although in the past, the logging of system activity has taken place for a variety of purposes such as troubleshooting and performance monitoring, the generation of a history file of past system activity is a particularly useful source of evidence for forensic investigations.

Figure 5.1 identifies three key phases relating to system activity logs in the forensic investigation timeline. Phase A is where information relating to system activity is written to an event log file. In phase B the evidence contained within the event log file remains in the digital environment where it may be subjected to management processes. Phase B ends when the forensic investigator extracts the log evidence from the digital environment to the physical environment. Phase C ends when the evidence has played its role in the forensic investigation. It is important to recognize that these phases are not strictly chronological. Activities in phase A are likely to continue while activities in phase B are taking place. For example, event log entries will be continuously written even while the log file is being managed in the digital environment.

![Figure 5.1: Forensic Investigation Timeline](image)

Existing literature provides an evidential reliability framework for evidence extracted from a computing environment under human direction. Underpinning this framework are three universal principles of evidential reliability that apply to computer-derived exhibits –
accuracy, completeness and authenticity (see section 2.4 for definitions of each principle). Sommer (1997) suggests that a series of six tests can be applied to meet these principles (see section 2.4.6 for a discussion of these tests).

However, the existing framework does not adequately cover events logs, as they are an output of a computer-directed (as opposed to human-directed) evidence acquisition facility. Although event logs work within the confines of the digital environment, they are a distinctly separate mechanism whose provenance and reliability must be established separately by the forensic investigator (Sommer 1997).

There is no existing basis for a discussion on the evidential weight of event logs in literature. The existing framework has been developed specifically for human-directed forensic investigation only. To lay the foundation for a discussion on event logs, a comparison must be made between the role of the event log and that of the forensic investigator. The aim is to identify similarities and differences that can then be used to analyze and then interpret the meaning of the evidential principles of accuracy, completeness and authenticity in the context of event logs.

There are a number of obvious differences between traditional forensic preservation and the logging of events:

1. Forensic investigation is human-directed whereas event logging is computer-directed.
2. Event logging takes place in the digital environment created by the cooperative workings of hardware and software whereas forensic investigation is not confined to the digital environment.
3. Forensic investigation typically begins long after the incident has taken place and critical evidence is lost. Event logging is a continuous background process running prior to the incident, as the incident is unfolding, and post-incident.
4. The forensic investigator preserves existing evidence from the target computer whereas event logs produce new evidence regarding system activity (event logs produce ‘new’ evidence by making such transient evidence available to forensic investigators by writing to long-term media).
However, one key similarity illustrates the applicability of the principles of evidential reliability that were developed for human-directed forensic investigation to computer-directed event logging.

Both the human-directed and computer-directed cases feature a similar process as they copy to and from digital media. The forensic investigator is copying from a static data source (hard disk) whereas event logs are copying from a dynamic data source (transient memory). Therefore, in terms of evidence acquisition there is a parallel between the two. The weighting criteria of ‘accuracy’ and ‘completeness’ applied to the process of copying has meaning in both circumstances.

Of importance this discussion is the distinct difference in the inherent complexity of the processes. The forensic investigator simply copies digital evidence from a relatively stable environment (the source is suspended) whereas the event log captures evidence from a dynamic environment (note the event log does not capture all dynamic evidence). This difference in complexity must be considered in the weighting criteria.

The role of the human-directed and computer-directed cases is starkly different when it comes to establishing ‘authenticity’. The forensic investigator essentially testifies regarding the authenticity of evidence derived from a particular computer by addressing issues like provenance of source, correct working, and link between parties described in evidence and parties identified in the legal proceedings. This is entirely understandable as the forensic investigator is trying to relate the computer-derived evidence to the forensic investigation proceeding in the domain of the court.

The role of the event log is limited to the acquisition of evidence of system activity and perhaps its management in the digital environment. Therefore, of the three conditions laid out by Reed (1990 as cited Sommer, 1997), ensuring the ‘continuity of evidence’ falls within the responsibility of event log management whereas the other two do not.

However, this is not to say that the event log cannot make valuable contributions to the remaining tests outlined by Sommer (1997). It is not the responsibility of event logs to meet these tests although any information collected that complements forensic investigator testimony underlines the ‘usefulness’ of system event logs towards forensic investigations.
Chapter 5: Evidential Weighting Criteria for Event Logs

Therefore, a more general term, namely ‘evidential weight’ is preferred over ‘evidential reliability’ as the latter term focuses on evidence of system activity only and does not adequately encompass the contribution of event logs towards establishing the authenticity of evidence outside the event log. Although both terms may seem to be similar and in some cases may be used interchangeably, in practice evidential weight is a broad term which goes further than the reliability of evidence and includes the content of files and inferences that may be drawn from them (Sommer 1997). However, this thesis uses the term ‘evidential weight’ to represent the collective contribution of an event log towards establishing the reliability of all computer-derived evidence in a forensic investigation.

Therefore, there are two significant roles for event logs in terms of digital evidence collection. Firstly, event logs create information records of system activity thereby acting as a computer-directed evidence acquisition mechanism. This role is particularly critical during the time period when system activity related to the incident is occurring (see (A) in figure). There is a parallel between the role of event logs in logging system activity and that of forensic investigators preserving evidence from pervasive media. I focus on this role in section 5.3.

Secondly, event logs play a unique role in authenticating all computer-derived evidence. Although the event log is able to collect evidence regarding the correct working nature of the host environment at any time, there is a particular period of time (spanning B) where authenticity can play a role in relating system activity to the incident. I discuss this role in section 5.4.

5.2 Developing Criteria for Evidential Weight of Event Logs

5.2.1 Accuracy
From an evidential weighting point of view (in the human-directed case) a clear distinction can be made between the reliability of the contents of a document and the reliability of the process by which it is preserved. Therefore, even if a document has been reliably preserved, it does not imply the information within the document is reliable as well (eg. a document containing inaccurate and misleading information can be reliably preserved).
However, this is not necessarily the case with event logs. Since event entries are authored by the computer through an automated process, there is a direct relationship between the reliability of the computer and the reliability of the information content. For example, if the computer is demonstrably working and maintained correctly then information in the entry such as date, event category and event type, must be correct (assuming the log has been correctly preserved).

According to Sommer (1997), accuracy refers to the correctness of the preservation process. For example, accuracy in the preservation of a static piece of data can be demonstrated if the process of preservation is correct; i.e., the preserved copy is identical to the original. However, in the case of event logs accuracy refers to the quality of the procedure by which evidence of system activity is created. Accuracy can only be demonstrated by establishing that the evidence identifies the logged system activity. This is an important statement as it identifies an additional dimension to accuracy in the context of event logs. The role of the forensic investigator is simply to make an identical copy of trace evidence whereas the role of the logging system is to create an information record of the event taking place. The difference here can be highlighted by contrasting ‘accuracy of information’ with ‘accurate identification’. Although functionally, log entries are created by passing a time-stamped set of values, the intention of creating a log entry is to identify an event rather than just to record information. Accurate identification of system activity implicitly assumes accuracy of information whereas the converse is not true.

It is also important to recognize that event logs are computer-directed and therefore the correctness of the event log facility is related to the correctness of the computing system. Further, in principle this discussion only relates to ‘incident-related’ evidence. If evidence that is not related to the forensic investigation is determined to be inaccurate then this finding may not affect the accuracy of relevant evidence. Of course, depending upon the reasons for the inaccuracy, there may be concerns raised that are relevant to the evidence directly linked to the forensic investigation. However, for the sake of simplicity, subsequently developed definitions do not explicitly limit the scope to ‘incident-related evidence’ although this is implied.

A process-centric definition may be accuracy is the extent to which the computer-directed event log mechanism correctly logs system activity. However an evidence-centric definition
would be *accuracy is the extent to which the evidence in the event log correctly identifies system activity*. The latter definition is more useful towards the concept of evidential weight.

When it comes to erroneous event log evidence, non-contamination is a special case. Inaccuracy can be caused in two cases – firstly, where errors are not instigated maliciously and occur due to the quality of event log processes and secondly, if the errors in event log evidence are due to malicious attack. The latter case is also categorized as an accuracy problem, however, more specifically it is a case of contamination. The impacts of a malicious attack on an event log are possibly two – contamination of existing information (this includes deletion) and introduction of new information. In the security arena the technical terms for these attacks are modification and fabrication.

As previously mentioned, in the case of event logs there is a link between the correctness of the computer’s logging mechanism and the correctness of the information content. It can be reasoned that an event log mechanism that is demonstrably correct will produce a correct event log. Therefore from an information content point of view, accuracy refers to the extent to which the information description of system activity is correct.

This discussion can be further reinforced with supporting observations from a recent study of audit trails (Allinson 2003). The author makes a series of observations regarding the analysis of audit trails in relation to several legal cases involving computers and as a result, states that control over audit trails presents a challenge. The author identifies a series of areas that must be considered as part of this challenge. They include audit capture, audit information retrieval, central storage for audit trails, backup, analysis, and security.

These audit trails are stored on digital computing systems but describe events that have taken place in the physical world as opposed to the digital world. Although this discussion has been focused on events taking place within the digital environment, the observations of Allinson (2003) from a physical world perspective prove useful in further establishing observations about the desirable qualities of event log preservation functionality.

The author makes the following statement about audit trail content.

*The validity, relevance and reliability of each piece of data is the second area of scrutiny*
Chapter 5: Evidential Weighting Criteria for Event Logs

Unfortunately the use of terminology across literature in this area tends not to be strictly defined and consistent. However, in particular the term ‘validity’ can be construed to refer to ‘correctness’ in the context of audit trail content. Reliability in the context of data appears to be synonymous with trust and dependability. This concept is related to whether the data was contaminated or not (addressed later in this chapter). The third term ‘relevance’ is not related to the problem of reliability of evidence preservation process as discussed in this thesis, but it is addressed in the admissibility process early on in the investigation process.

A secondary aspect of the above quote is reference to ‘each piece of data’. Recall that an event log is a collection of entries relating to past events. Although event logs have been treated as discrete entities, in reality each log is a set of multiple entries where each entry is a record of a past event. Therefore accuracy refers to the correctness of each piece of data in the event log.

Despite the above discussion, the following example demonstrates the need to extend the correctness property for analog values. Consider an event log entry describing a logon event:

Date: 24/06/2004
Time: 11:32
User: Adam
Type: Success
Source: Security
Event ID: 576

Querying the accuracy of this entry leads to an assessment of correctness. Only in the case of the time field is correctness ambiguous. The time may be considered correct but due to its analog properties correctness may not be sufficient to demonstrate accuracy. In this case of analog values there is an additional need to be ‘precise’. In the case of a digital environment several events may occur within the space of a minute affecting the reconstructed sequence of events. However, the extent of precision necessary to remain accurate is relative to the situation.
Therefore, an evidence-centric definition of accuracy is presented.

Accuracy is the extent to which the evidence in the event log correctly identifies system activity and is not wholly or partially modified or fabricated and the timing information is relatively precise.

The following figure charts the argument used to construct this definition:

![Figure 5.2: Accuracy – Argument Construction](image)

It is important to note here that the scope of both Accuracy and Completeness is limited to phase ‘A’ as identified in figure 5.1. Therefore, modification and fabrication of evidence in the event log can occur in phase ‘A’ and phase ‘B’. For example in phase ‘A’ flaws in the design of the event logging mechanism may lead to incorrect information appearing in event log entries. In phase ‘B’ external attacks on event log files can also result in modification and fabrication of information in logs. Note that I deal with phase ‘B’ under the ‘authenticity’ weighting criterion (see sub-criteria ‘a’ in the definition of ‘authenticity’).

5.2.2 Completeness
Traditionally, completeness refers to the extent to which an investigator preserves all the relevant evidence related to the incident. Although this definition is easily stated in a theoretical sense, practically absolute completeness is a somewhat unattainable ideal.

Completeness of evidence related to a forensic investigation requires forensic investigators to preserve all the evidence that may be relevant to an investigation. Practically, preserving and then presenting every single piece of evidence that may be relevant is difficult. Even though hard disks can be preserved with a very high-level of completeness (eg. a bit stream copy), identifying every piece of evidence on a hard disk that may be relevant remains difficult.

The problem of preserving evidence becomes significantly more complex when incident-related system activity is considered. Creating an evidential record of all the system activity related to the incident is a difficult problem.

At the time of the incident a considerable number of related events are likely to occur. This activity will take place in a very short amount of time making it very difficult to generate sufficient log entries without seriously degrading the performance of the computing system.

In addition, a number of other factors concerning hardware, operating system, system configuration and so forth play a critical role in determining what system activity is to be logged (note that the evaluation chapters present examples where some sample baseline logging requirements are distilled that are meaningful in those contexts).

Preserving evidence of system activity with absolute completeness is a practical impossibility. However, given completeness is not a binary criterion, there are two thresholds of completeness that make sense from a weighting perspective.

The first threshold relates accuracy to completeness. For log evidence to correctly identify system activity there must be a certain level of completeness in the evidence. The evidence must provide enough information to the forensic investigator to uniquely identify the activity that generated the evidence.

A second and significantly higher threshold of completeness applies to reconstruction. To achieve reconstruction the evidence must provide an exhaustive amount of information to the
forensic investigator such that any questions regarding the particular activity can be conclusively resolved.

Another aspect of completeness is having multiple corroborating pieces of evidence that reinforce evidence of system activity. The weighting of evidence of activity can increase if there are independently separate sources that can be used to demonstrate particular system activity occurred.

However, the definition of completeness from an evidential weighting perspective does not have to factor in the two practical thresholds of completeness. Strictly from a weighting point of view, defining completeness in the theoretical context of event logs is relatively straightforward. *Completeness is the extent to which the evidence in the event log describes all system activity.*

This is reinforced by the following statements from Allinson (2003):

“*Audit trail content requires both analysis of what is recorded within each event/activity record and whether or not all event/activity records are recorded in the audit trail.*”

because:

“*Without full recording of all event/activity it is not possible to state or prove with a degree of certainty that particular actions were or were not performed. Therefore, there is a requirement for a high level of assurance that all activity is recorded.*”
5.2.3 Authenticity

According to Reed (1990 as cited in Sommer, 1997), authenticity of computer-derived evidence is derived from the following:

1. Link between the evidence and the circumstances of the retrieval of the evidence - such as date and time of retrieval
2. Continuity or unchanging state of evidential record
3. Link between the evidence and the source (i.e. the evidence was taken from the particular computer identified in the proceedings)

According to the role definition of event logs (section 5.1), only point number 2 falls within their direct responsibility. This is because retrieval of the evidence is a human-directed act for which computing systems cannot take responsibility. Further, although the link between the evidence and the source may be argued to be the responsibility of the event log, this does not fall within its primary role, which is recording evidence of system activity and its subsequent management. Instead, such extraneous services are better placed under ‘utility’ (see section 5.3). Therefore, authenticity in terms of event logs is:

*Continuity or unchanging state of evidential record*
Determination of the evidential weight of an event log is conducted at the time the forensic investigator makes the final presentation. Figure 5.1 shows a timeline of a forensic investigation. Although the event log records evidence of system activity at A, the evidential weight is measured at C. Since authenticity (particularly continuity) of evidence is a factor in evidential weight, the weight may vary at points A, B and C.

However, the evidential weight at point B is particularly important since all computer-derived evidence is extracted there and the digital environment does not influence event logs past that point. Subsequent discussion in this section considers the weight of event log evidence at point B. It is possible that the event log may be archived before being retrieved by the forensic investigator. In this case each time the log evidence is moved from one location to another each of the three authenticity requirements must be established.

**Continuity of Evidence**

In the general case, continuity of evidence or chain of custody refers to the time period after the evidence was acquired by the investigator until the evidence was presented to the designated authority. The forensic investigator must explain what happened to the evidence during this time period. Although the chain of custody is typically understood to begin after the evidence leaves the digital environment there is an obvious extension to this principle that applies to event logs.

As previously mentioned, there is a particular time period that begins when the evidence relevant to the investigation was written to the event log and ends with the extraction of the evidence from the digital environment (point B in fig 5.2). An account of what happened to the event log evidence during this time period must be provided.

Authenticity, in summary, can be defined as follows:

**Authenticity:** The extent to which evidence in the event log has remained free from fabrications and unchanged until extracted from the digital environment.
5.3 Utility: Establishing Reliability of Computer-derived Evidence in General

Event logs are uniquely positioned to collect evidence about the digital environment that complements the role of the forensic investigator in establishing the reliability of computer-derived evidence. As previously mentioned, forensic investigators perform a series of tests to demonstrate the extent to which computer-derived evidence is reliable. Although the forensic investigator directs the tests and testifies in court on the reliability of evidence, event logs may also contain useful information about the kinds of issues covered by the tests that assists the establishment of reliability.

To establish the accuracy, completeness and authenticity of computer-derived evidence in general, Sommer suggested the following five tests could be performed:

1. Computer's Correct Working Test
2. Provenance of Computer Source Test
3. Content/Party Authentication Test
4. Evidence Acquisition Test
5. Continuity of Evidence/Chain of Custody Test

The forensic investigator conducts these five tests to establish the reliability of any digital evidence acquired from the computing environment including the evidence contained within event logs.

This function can also be conducted in part by computer-directed evidence logging processes. In addition to preserving evidence related to the forensic investigation, event logs can assist investigators to establish the reliability of evidence derived from the host computing environment.

Utility is therefore a desirable attribute of log evidence that assists the process of forensic investigation. Although Sommer’s criteria is directed towards the forensic investigator, the possibility of collecting information that is not necessarily related to system activity but can be used to complement the testimony of the forensic investigator must be investigated:
Chapter 5: Evidential Weighting Criteria for Event Logs

Computer’s Correct Working Test
The first criteria tests whether the digital environment was behaving “correctly” or “normally”. This coincides with the primary reason why event logs were originally designed – for troubleshooting purposes. Event logs can collect information from the environment that can be used to ascertain whether the system was working “correctly” or “normally”.

Provenance of Computer Source Test
The second criteria tests whether the evidence used by the forensic investigation is taken from the digital environment where it claims to be from. The burden of proof here is largely on the forensic investigator to show the traceable link from the extraction of the evidence to the presentation of evidence in court. However, event logs may provide some assistance in this case. One such scenario may involve the event log acting as information collector. The log may record information identifying the host environment where it has been collected. Event logs have not been designed to record host identification information in each event entry. Adding such a feature would reduce the burden of proof on forensic investigators using evidence from event logs. This is not considered an optimal solution from a performance point of view. The problem of traceability can be solved in the larger context of a network environment where management activities like log centralization can be made responsible for tracing the origins of event information.

Content/Party Authentication Test
Event logs cannot directly address the relevance of information to the incident and/or parties. This is because the nature of the incident and the identity of the parties is not known at the time of information collection. However, the identification of system actors, actions and timings may assist forensic investigators to related evidence of system activity to evidence outside of this activity.

Evidence Acquisition Test
The evidence acquisition test examines the actions of the forensic investigator in extracting evidence from the digital environment. Since forensic investigators typically do not work on original and live environments, therefore event logs are unlikely to be operational and are unable to log the actions of the forensic investigator. This implies the evidence acquisition test is not within the scope of the event log and is limited primarily to the testimony of the forensic investigator.
Continuity of Evidence / Chain of Custody Test
Chain of custody examines what happened to the evidence after it was extracted from the digital environment. Event logs are unable to contribute to the chain of custody as they cannot track evidence once it has been extracted from the digital environment.

In the case of traditional forensic investigations the authenticity of evidence must be established from the time of retrieval from the source to the time of presentation. In the case of evidence derived from event logs there are two particular times that are of importance (see figure 5.1 section B)

The first time is when the last piece of evidence is written to the event log and the second time is when the event log is extracted from the computing environment by the forensic investigator.

I present ‘utility’ as a desirable quality of event logs that makes the evidence more compelling:

Utility: The extent to which the information in the event log:
(a) testifies to the correct working nature of the relevant system or systems
(b) identifies the host system in as much detail as possible
(c) identifies parties and resources in as much detail as possible
(d) identifies all systems that have hosted the event log or any information contained in the event log

5.4 Conclusions
There are two significant roles for event logs in terms of digital evidence collection. Firstly, event logs create information records of system activity thereby acting as a computer-directed evidence acquisition mechanism. Secondly, event logs can play a unique role in authenticating all computer-derived evidence from digital environments.

Based on this contention, the term ‘evidential weight’ is preferred over ‘evidential reliability’. The latter term focuses on evidence of system activity and does not adequately encompass the contribution of event logs towards establishing the authenticity of evidence
outside the event log. Unfortunately there is no evidential framework or model for event logs. The existing evidential reliability framework for computer-derived exhibits is developed specifically for human-directed forensic investigation.

The primary contribution of this chapter is the development of criteria for the evidential weight of event logs based on the principles of accuracy, completeness and authenticity. A significant aspect of this contribution is recognition of the utility of event logs as a contributing factor to evidential weight.

The outcome of this contribution is the following criteria:

**Accuracy:** The extent to which the evidence in the event log correctly identifies system activity and is not wholly or partially modified or fabricated and the timing information is relatively precise.

**Completeness:** The extent to which the evidence in the event log describes all system activity.

**Authenticity:** The extent to which evidence in the event log has remained unchanged until extracted from the digital environment.

**Utility:** The extent to which the information in the event log:
(e) testifies to the correct working nature of the relevant system or systems
(f) identifies the host system in as much detail as possible
(g) identifies parties and resources in as much detail as possible
(h) identifies all systems that have hosted the event log or any information contained in the event log

This chapter developed evidential weighting criteria that can be applied to event logging systems. the previous chapter constructed a technology-independent model of the event logging process of operating systems. The next chapter applies the evidential weighting criteria to the model of the event logging process to investigate evidential weighting issues that apply to event logs.
Chapter 6

Investigating the Evidential Weighting of Logs

The purpose of this chapter is to identify evidential weighting issues arising from the design and management of system activity logs. The event log model developed in Chapter 4 and the evidential weighting criteria developed in Chapter 5 are used to conduct this analysis.

Recall the primary research question of this thesis is:

“How can the evidential weight of system event logs be maximized?”

Two related sub-questions identified in section 1.1 are:

1. How can the evidence collection capability of event logs be determined?
2. How can the weight of event log evidence be measured?

A model for the design of system event logs was developed in chapter 5 to answer sub-question 1. A model for the weighting of incident-related evidence in an event log was used to answer sub-question 2. This chapter investigates the weighting of log evidence by using these two models.

Section 6.1 discusses and justifies the method by which the evidential weighting model and the event log model will be used to analyze the influence of event log design on evidential weight.

Section 6.2 highlights issues related to the identification of actors, actions and timing information within individual log entries.

Section 6.3 investigates the ability of log evidence to accurately identify and describe system activity. In particular, the focus of this discussion is on the precision of timing information and its impact on the incident timeframe, sequence of events and certainty with which a particular event can be tied down to a period of time. Further, the impact of the absence of critical system activity such as that of network access, and application activity is also discussed.
Section 6.4 examines the selection interface to activity logs. Two issues of interest are the impact of selection control on system performance and the guidance given to administrators that frequently results in forensic investigators finding insufficient and inadequate evidence.

Section 6.5 discusses the impact of log management techniques on continuity. Discussion focuses on potential contamination arising from administrative practices like reduction, storage and retention strategies on log files. Contamination from attacks on the security of the log file and on the continuity of the evidential record, are also addressed.

6.1 Method of Analysis

The particular method of analysis adopted in this thesis is extremely important. The method must be based on a systematic approach. Therefore, this section discusses the various approaches that may have been pursued and justification for the choice of method adopted.

6.1.1 Scope of Analysis

The scope of analysis is the time interval where log evidence remains in the digital environment. Section 5.3.3 notes that the weight of all computer-derived evidence is measured at the time it is presented to the intended authority. In figure 5.1, this point lies at the end of interval ‘C’. Since the primary point of interest is the investigation of the weight of log evidence for the period of time it remains in the digital environment, therefore evidential weight is measured at the end of interval B rather than at the end of interval C. This implies the scope of the analysis spans interval A and interval B:

- Interval A is the evidence acquisition phase involving the logging mechanism
- Interval B is the evidence management phase where the log file is managed by the digital environment.

The management phase continues until the evidence is extracted from the digital environment for the purposes of presentation.
6.1.2 Interval A: Evidence Acquisition Phase
To conduct the analysis in the first interval, the two models from chapters 4 and 5 are utilized. The event log model applies to interval A as it was constructed to model the evidence acquisition process. In terms of the evidential weighting model, the accuracy and completeness criteria apply to the evidence acquisition process while authenticity applies to management.

The investigation of the evidence acquisition phase can be approached using two different methods:

Investigation Method 1
The first method involves conducting three separate analyses. The first analysis would investigate the influence of event detection on the weight of evidence. This involves examining the accuracy and completeness of evidence as a result of the design of event detection. The second analysis would investigate the influence of event selection on the weight of evidence in a similar fashion. Finally, the last analysis would investigate the influence of event characterization on the weight of evidence using the same process.

Investigation Method 2
The second method involves conducting only two analyses. However, these are considerably more complicated as it would examine event detection, event selection and event characterization from an accuracy perspective first. Then the procedure would be repeated from a completeness perspective.

Investigation Method Chosen
In principle both approaches should yield the same results. However, since the complexity of the investigation lies in examining the detail of the design (architecture) factors, the first method was adopted. This approach concentrates on examining a single architectural factor before moving to the next. Therefore, it is possible to examine the complexities of event detection using the weighting criteria before moving on to selection. By adopting this approach both the process of analysis and its written presentation are simplified.
The chosen approach focuses on answering the following questions:

1. How does the design of event log detection influence accuracy and completeness of incident evidence?
2. How does the design of event log selection influence accuracy and completeness of incident evidence?
3. How does the design of event log description influence accuracy and completeness of incident evidence?

Presentation of Results

After much deliberation, event description was placed before event detection, therefore the results of the investigation are in the following order:

1. Influence of event description on weight of log evidence
2. Influence of event detection on weight of log evidence
3. Influence of event selection on weight of log evidence

The primary reason for this change in order of presentation is the necessity to examine evidence at the level of a single log entry before attempting to describe the ‘bigger picture’ issues associated with multiple log entries. Further, since event description and event detection make up the two parts of the logging mechanism, therefore it is reasonable to complete the discussion of the logging mechanism before moving on to event selection infrastructure.

Further, the results are presented in the simplest and most intuitive manner by consolidating issues as much as possible. For example, the accuracy of timing information in individual log entries raises two weighting concerns. One concern is whether the logging mechanism records entries atomically, the second is the accuracy of the system clock providing the timing information. Although the two issues were identified separately, they have been presented together to avoid unnecessary fragmentation of this discussion. Further, discussion on event description frequently refers to event detection. When discussing issues such as actor and action identification it is difficult to separate the two. Therefore, these issues have been raised regarding event detection in the section discussing event description.
6.1.3 Interval B: Evidence Management Phase
The analysis of the evidence management phase is conducted by applying the authenticity criterion to the various management processes. The second phase of analysis seeks to address the following question:

- How does the continuity of evidential record influence authenticity of incident evidence?

This question involves the examination of log file management processes that may be involved in handling evidence. These processes were identified in section 3.3 – security, retention, storage, and reduction.

6.2 Influence of event description on weight of log evidence
Recall that event detection determines the potential logging capability of a computing system and event selection determines how the logging capability can be used to create evidence of system activity during a particular session of operation. Event description is the choice of information recorded as evidence of a single system event. In most conventional operating systems only a pre-selected set of information is passed by the event reporting function to a log entry. Other operating systems divide their event records into two sections - a generic section that records the same information for all events and a ‘message’ block of text that contains event-specific information (e.g. in the case of Windows).

There are three categories of information passed to the logging service responsible for writing the new log entry. These are the identification of the actors, the time the event took place, and description of the action that occurred.

Traditionally, the evidential weighting criterion of accuracy refers to the correctness of the process by which the forensic investigator extracts digital evidence from a computing system. When applied to the logging mechanism within a digital environment, accuracy refers to the correctness with which the computer-directed process conducts its task of delivering the right information to the log.
The operating system employs a process by which the correct timing, action and actor information is inserted into the log entry. If the computing system is working correctly and has not been subject to a malicious attack, then the information in a single log entry is likely to have been recorded accurately. Although this process may be deemed accurate for the operating system, from the forensic investigator’s perspective the information recorded in the log entry may not qualify as ‘accurate’ according to forensic needs.

6.2.1 Accuracy of Individual Log Entry Timestamp

The accuracy of the timing information recorded in individual log entries is perhaps the most intuitive to address. The central forensic issue in this case is the following:

*Is the time recorded in the log entry precisely the same as the correct (absolute) time when the event described in the log entry occurred?*

To investigate the weighting of evidence generated by system activity logs in this case, the event log model will be applied. Two key issues were identified by considering the role of event detection and event description in producing the timestamp in individual log entries. The role of event detection is to implement the time stamping mechanism. The source of timing information is related to event description. Therefore the accuracy of the log entry timestamp depends on two distinct factors. First, whether the reporting of the log entry introduced a delay that may impact the accuracy of the timestamp and second, whether the source of the information used was accurate in the first place.

The evidential weight in this case depends on the following major issues:

1. *Is the source of the timing information used by the logging mechanism accurate?*
2. *Is the reporting of the event occurring atomically?*

Typically, the time-stamping process itself is an automated feature of the logging mechanism (see discussion of Windows log entry in section 4.1.5). When a log entry is written, the logging mechanism records the time using the computing system’s clock as the source of information.
Since the system clock is set by the system’s owner, therefore it is not possible to make a conclusive assessment of the evidential weight of system activity logs with respect to accuracy of timing information.

This scenario highlights an important problem. Given accuracy of timing information has a significant impact on the evidential weight of system activity logs, every effort must be made to ensure system clocks are providing accurate timing information.

Timing information in log entries might reflect when events were reported as opposed to when they took place. As will be discussed in 6.4.2, the system architect’s decision as to where to position requests to write a log entry determines the precision with which log entries report events. Further, the single timestamp does not represent the period of time when the event was active.

Therefore, evidence of multiple system operations executing simultaneously may mislead forensic investigators into believing the timeframe of the incident and sequence of major events can be deduced with absolute certainty. It may appear that the impact is insignificant because the difference between the time of occurrence of an event and its reporting is relatively small. However, it must be noted that even small margins of error in precision may have considerable impact on the relevance of evidence gathered from multiple computing systems.

It is interesting to note that in terms of evidential weight, it is difficult to speculate on the impact of inaccuracy of timing information. However, given a large number of events typically take place in a short amount of time within digital environments, the impact of timing inaccuracies can be amplified when events are correlated across multiple computing systems.

Therefore, there may be some merit in forensic investigators correlating timing information across multiple computing environments as a means of demonstrating confidence in the accuracy of the source of timing information on individual systems.

In addition to the two key factors mentioned in this section, there are some additional factors related to timing that may also affect evidential weight. Firstly, however remote, there is a
possibility that external actors may directly manipulate the source of timing information. Secondly, there may be a time gap between when the event was executed and when the source of timing information was queried.

### 6.2.2 Accuracy of Actor Identification in Individual Log Entries

It is primarily the responsibility of the forensic investigator to link identity references in log evidence to real-world actors. Section 2.4.8 pointed out that a forensic investigator must establish the authenticity of the evidence regarding parties/content to the evidence exhibits (Sommer’s evidence test #3).

In this context, the term ‘actors’ refers to applications, file, processes, and objects in the digital environment. The term ‘real-world actors’ refers to users, document exhibits and the like that exist in the physical world rather than in the digital world.

The influence of event detection on evidential weight stems from the system architect’s decisions as to where in the code to make a request to log an entry. Depending upon where in the code a request is made to log an entry, different actors may be passed to the log for writing. Although the position of this request may affect which actors are identified in a log entry, positioning is not related to the accuracy of the information passed on to individual log entries. This implies that event detection influences which actors are identified rather than the accuracy of the identifying information. The latter issue is addressed by event description.

The following question may be asked:

*What evidential weighting concerns apply to identification of actors in the operating system environment?*

Evidential weighting concerns with actor identification vary depending upon the categories of actors that exist in digital environments. In the digital environment, real-world actors such as documents are frequently identified by their filename, applications are identified by the image file (also a filename) and a user identifier (a representative number) may be used to represent a real world user account (which in turn may be linked to a real world user).
Identification of files and applications is a particularly difficult problem. In the case of a file, the content of a file is frequently changing and yet its identification remains the same. This situation implies that in order to identify a file it is not enough to know its digital identifier. In fact the file may be identified by two means in the digital environment. Firstly, by a handle that is assigned as a reference to an opened file - this is obviously not very useful as a means of forensic identification. Secondly, by its filename – this too may not be very useful as well for several reasons. The filename might not be unique on the system as there may be several files with the same filename in different directories. At this point it may be reasoned that the precise path may be an effective solution to the problem of correct identification. However, the file may be moved, and/or the file contents might change. In terms of log entries the correct path to the file including its filename can be preserved within a log entry (timing information can be deduced from the entry anyway).

Identification of applications is also challenging because the forensic identity of an application is related to its functional behavior whereas the operating system identifies the application by the filename of its image. Further, an application service may not result from the execution of a single (exe) file, but a number of files that may be loaded into memory at once or at different times thereby varying the functional capability of the application service. Application services will nominate a single executable file as the starting point for execution. It is this file that Windows associates with the process that is the application service. Windows identifies this file when it logs the creation of the process that becomes the application service.

To illustrate the problem with identifying files and applications consider the example below that takes place in a UNIX-type environment. It is taken from (Denning, 1987).

Smith copies an executable GAME file into the <Library> directory. The result of this action is a write violation since he does not have permission to that directory:

(Smith, execute, <Library>COPY.EXE, 0, CPU=00002, 11058521678)
(Smith, read, <Smith>GAME.EXE, 0, RECORDS=0, 11058521679)
(Smith, write, <Library>GAME.EXE, write-viol, RECORDS=0, 11058521680)
where each entry is in the following format:

(Subject, Action, Object, Exception-condition, Resource-usage, Time-stamp)

*and the values are explained as following:*

- **Subject:** Initiator of activity on the system
- **Object:** Resources managed by the system
- **Action:** Operation executed by the subject on the object
- **Exception-condition:** Exception outcome of the action
- **Resource-usage:** Quantitative elements used
- **Time-stamp:** Time/date when action took place

The copy operation has been broken down into three single-object events mirroring the mechanics of a system and simplifying potentially complex system interaction for examination.

From a forensic point of view, the information gathered about the single-object information covers the requisite who/what/when. However, the individual elements gathered like identity of the user, timestamp, and identification of object are relative references and will not withstand hostile criticism.

Although the system operation indicates ‘Smith’ copied GAME.EXE, the inaccuracy of the event log characterization function is readily visible. Assuming the event detection function correctly identified the operation and actors, there may be more than one copy of GAME.EXE on the system. As the log entry does not specify path, the precise location of the file cannot be determined, leaving the possibility that the particular file recovered (based on the evidence) is the wrong file.

To establish the identity of this actor, the path of the program must be preserved (path may change too), with the knowledge that the name GAME.EXE is the name of a particular object and may have little to do with the object’s true nature, which can only be determined if a copy of the object was preserved along with the event log record.
Therefore, within the scope of an individual log entry, the evidential weighting concern revolves around the extent to which the specific actor can be identified on the filesystem. A relevant question to the forensic investigator is the following:

*Is the filename identifying the actor sufficiently unique to identify its existence on the filesystem?*

Further, timing information from the log entry can be used to increase the accuracy of actor identification. Instead of relying on just a filename as identification of a document, forensic investigators can refer to a filename referenced at an approximate time. Forensically, this is very useful information. Since the contents of a file may vary over the period that it exists on a hard disk. Knowing the time at which a file was accessed helps forensic investigators to be more definitive about the precise exhibit that they are interested in. Rather than claiming the exhibit is ‘file X’, the claim can be the exhibit is ‘the contents of file X at the time Y’.

Unfortunately in the case of real world users, log evidence is limited to providing user identifiers. In the example above, ‘Smith’ is the only actor identification available to a forensic investigator. Approximate timing information helps forensic investigators to be slightly more specific. Forensic investigators can refer to a user active at a particular time. This is particularly useful in the case of common actor names such as ‘Smith’, as it is likely to be assigned to another user. This changes the pairing of digital actor to real world actor potentially causing more confusion for forensic investigators.

Operating systems employ a system of digital labeling of actors to manage internal resource interaction. In general, forensic investigators will find identification of real world actors from information collected within the digital environment a difficult problem.

This is primarily because the contents of documents or the behavior of an application is frequently changing, whereas the digital identifier used in log entries remains the same. Identifying users is also difficult as the external and real physical presence of users is connected to the digital environment through digital identifiers. Given system activity logs use the same scheme of digital identifiers to reference actors, forensic investigators will find it difficult to connect digital identifiers to external users through information recorded in log entries.
Chapter 6: Investigating the Evidential Weighting of Logs

It is important to point out that although the scheme of digital labeling implemented by operating systems is appropriate for routine processes, it is not sufficiently rigorous so that a forensic investigator can readily relate digital actors to their real world counterparts. Therefore, in terms of reliability of actor identification, the weight of evidence from the digital environment is negatively impacted.

6.2.3 Accuracy of Action Identification in Individual Log Entries

One of the fundamental aims of a forensic investigator is to identify system activity from digital evidence such as individual log entries. In this case, the evidence in question is action identification information that has been designed by system architects for logging purposes. For example, in Windows operating systems, a unique event id is assigned to represent activity in a log entry. In other operating systems, the precise action may be implicit if the log file is specific to a particular activity in the operating system. This is common where there is a need to log a single activity, or category of activities, as facilities are not provided native to the operating system.

An investigation of the weight of this evidence using the event log model reveals that event detection does not influence the correctness of this information. Event description, however, raises an important question about the ability of a forensic investigator to correctly identify the action undertaken by the system event that resulted in the generation of the particular log entry.

Forensic investigators rely on operating system architects to identify what system activity is being described in a log entry. Frequently, some of the detail contained in individual log entries will not be understandable without assistance from system documentation or event architects themselves (system documentation is notoriously unhelpful in these circumstances). This is especially the case where log entries are generated during low-level operating systems management such as object accesses. In these circumstances forensic investigators may find it difficult to recognize the significance of the action (note: a practical example involving object accesses is described in section 8.2.1).
System architects will typically log an important event that has taken place by virtue of the execution of a block of code as illustrated in figure 6.1. This unique event will be assigned an ‘event id’ which will be recorded in the log entry recorded. Forensic investigators may find it difficult to determine what this ‘event id’ represents. For example, Windows records the username of a person logged in for every entry generated by object-access events (560/562). The presence of the username implies that the user initiated the object-access requests even though the request is made by an application. The operating system records the username of the user logged in when the request was made but only does this for processes spawned from the user’s logon process (see section 8.2.2 for more examples). Frequently an obscure (technical) description is provided in system documentation with no assurances that the action identification is correct or comprehensive.

6.2.4 Actor Identification Across Multiple Log Entries

Section 6.2.2 discussed the accuracy of information identifying actors. It was noted that the forensic requirements of identification are unlikely to be met with the method of identification used by operating systems. Individually, a log entry may simply record identification of an actor involved in a system event. However, the availability of multiple log entries that reference the same actor presents a unique opportunity for the construction of a behavioral profile that may provide crucial insights into the real world identity of that actor.
Chapter 6: Investigating the Evidential Weighting of Logs

There is a direct relationship between the design of event detection and the ability of a forensic investigator to construct a viable profile of an actor from the log. The design of event detection dictates the amount of log entries and type of information used to describe actor behavior. User profiling may offer further assistance in identifying the real-world user. This provision can be used to identify a consistent pattern of user instruction that is characteristic of the real-world user to rule out evidence of third party involvement.

Actors such as files and applications are also relevant to this discussion. The content of a file and the function of an application are not necessarily reflected in the digital identifiers used in log entries. In both cases, forensic investigators can gain significant insight into the real world identities of these categories of actors by constructing a profile of their behavior in the digital environment. For example, if a known application (e.g. MSWORD.exe) were to exhibit a pattern of opening an unknown file then there is reason to suspect that the file is of Word format. If the application were more specialized, then that would narrow candidates even further. Similarly, if an unknown application was referenced in a log entry then the kinds of files it accessed may help to identify its function.

*Therefore, a key strength of system activity logs is the ability to accumulate a vast quantity of information, much of which describes the behavior of actors.* A system activity profile of an actor can be constructed and serve as a source of information on the identity of actors. A profile of the system activity of an actor is a more concrete basis for actor identification than what is typically available from hard disks.

6.2.5 Consistency of Log Evidence Across Multiple Log Entries

The logging of multiple events as a result of a system operation, or part of a system operation, allows for inaccuracies to be identified through information consistency checks.

Although it is unlikely, flaws in operating system design may result in fundamental mistakes in reporting system events to log entries. For example, if a file is opened in the digital environment and then subsequently closed (given each event is logged), it is reasonable to assume that the log entries reporting the ‘open’ and ‘close’ events will use the same file identifier in both log. If the identifier cited in the log entries were for some reason different
then this would be an indication of unreliable operation by the computing system. As a result, the weighting of log evidence would consequently be questioned.

A far more likely scenario with a similar outcome may occur in the case of external attacks on the log file (note although this discussion would normally be placed in section 6.5, it has been put here to prevent fragmenting the argument). Deliberate deletion of one or more log entries can also be detected by checking the consistency of information across the log.

In both scenarios outlined above, inconsistencies can be identified in terms of actors, action and timing information.

Inconsistency in timing information is most likely to be caused by modification attacks to the log. For example, if a distinct variation in the frequency of entries logged occurs where there is no change in selection policy, then this indicates either events are occurring but are not being logged or entries have been removed post-logging. This scenario can take the form of a significant reduction in the number of logged entries or the wholesale absence of entries over a period of time.

Inconsistencies in action information can be detected with the help of basic knowledge about the system activity that is taking place. For example, some operating systems open files with a log entry that contains the filename and a ‘handle’. The handle is a temporary identifier that may subsequently appear in a close file log entry. To identify the pair of entries indicating the open/close events, the handle must be used. Based on this example, it is logical that for every object close handle there must have been an object open handle.

In general, any identified inconsistencies are a weighting issue. Event detection plays a significant role in influencing the evidential weight in this case. If the design of event detection does not consider consistency of information then the overall evidential weight of the outcome of the log maybe negatively affected. The system architect decides how many log entries should be made available for logging for any particular system operation. Theoretically, it is possible that the system architect may omit one or more requests for log entries that may be needed to adequately describe the system operation. In principle, such a scenario would adversely impact the weight of evidence generated from the computing system.
6.3 Influence of Event Detection on Weight of Log Evidence

A key concern for forensic investigators is to identify high-level system activity from the set of log entries preserved by the log. The central issue for forensic investigators is whether system operations can be accurately identified from the log evidence. To investigate the weighting of this evidence, the event log model developed in chapter 4 is applied.

6.3.1 Grouping Related Log Entries to Identify System Operations

High-level system activities typically consist of multiple system operations. Each system operation may consist of a large number of system events. Each system event may result in an entry written to a log file.

Before forensic investigators can identify system operations (and ultimately incident activity), log entries must be grouped together into sets that represent evidence of related activity. These sets form the basis upon which forensic investigators will attempt to identify system activity.

From an event detection perspective, it is the number of possible logging requests and their positioning which determines how many log entries may be generated. These log entries represent the potential evidence that may be generated by a system operation. Although event detection determines which log entries will be generated by a system operation, it is the choice of information passed on to the individual log entries (event description) that influences the ability of the forensic investigator to group them together.

However, it may be difficult to separate evidence of user involvement (that can be used to establish intention) from evidence of routine operating system activity. This is particularly the case when identifying those entries that are related to a single user session. The event log will not only describe user behavior during the session but will also include those events related to managing the user’s operating system and application environment.

A complication relates to log entries that describe actions and actors that are common to many system operations and therefore cannot be uniquely associated with any of them. This is possible if certain key system events or code fragments are an integral part of more than
one system operation and the operating system does not log a unique identifier that can be used to link them.

Further, two different operations may require the same piece of code to be executed resulting in two similar system events (precisely the same action and actors but different timing) being executed. Deducing which system event was generated by which system operation may be difficult unless an identifier is used to group entries together.

Even if the incident is well known and can be replicated on different computers in order to isolate the set of log entries related to a particular incident there may be no easy way of identifying incident related log entries. As the precise configuration and build of computing systems is typically unique, the same incident activity on two different machines may generate a different set of log entries. In fact incident activity on the same machine executed at different times may also produce a varying set of log entries as they are logged under different circumstances.

*Given this discussion, it is certainly conceivable that forensic investigators may not be able to successfully group all the log entries related to an incident in all circumstances. This scenario will be a serious concern for evidential weighting. The level of confidence a forensic investigator may have in the accurate identification of system activity may be severely diminished if there is no reliable method of identifying log-entries generated by the same system operation.*

### 6.3.2 Uniquely Identifying a System Operation

An obvious extension to the discussion in the last section is that there is a strong likelihood that even if all log entries generated by the same system operation were successfully identified, there may not be enough evidence to uniquely identify the system operation. From an event detection perspective, many system operations may not generate enough evidence in logs such that forensic investigators will be able to uniquely identify them from log evidence.

For example, a series of code fragments that are executed to achieve a high-level system operation may generate a pattern of log entries that is not unique. This scenario may occur where a large number of entries are related to object accesses or similar resource management activity. Such activities are likely to be invoked by many different system
operations and are likely to contain references to actors and actions that cannot be easily traced to particular system operations.

As an aside, there is an interesting parallel between this scenario and one discussed in section 2.4.2. Recall that in R vs Doheny (1996), an expert presented DNA evidence to the jury and neglected to mention the reliability of the identification technique. In this case the issue was the frequency with which matching DNA can be found in the population at large. In the case of identification of system operations from log evidence, the reliability of the identification technique has a similar concern.

The forensic investigator must be aware that there may be more than one explanation for the existence of a particular log-entry or a pattern of log-entries. It is possible for many different system operations to involve generation of the same log-entry. Therefore, it may be difficult to prove that a particular log entry was generated by a particular system event that is inextricably linked to particular system activity.

6.3.3 Identifying Key Activities in a System Operation

An important distinction must be made between uniquely identifying a system operation from log evidence, and ensuring all key activities are identified. In both cases, the system architect’s decision as to how many entries to log and where to position the log-entry requests is important.

Again, it is important to note that a forensic approach to logging requires all identifiable activities to be logged. This is different from the traditional approach adopted by the operating system architect where the log entry simply reports the most relevant issues from a systems perspective.

In terms of event detection, accurate identification of a system operation may depend on whether log entries were written at appropriate points in the execution of the system operation. Recall the example presented in section 3.4. A ‘write to file’ operation may consist of multiple events such as ‘request of a write operation’, ‘initiation of a write operation’, more than one ‘status of a write operation’, and ‘completion of a write operation’. In this case, the outcome of the system operation may remain unclear until the very last system event is undertaken. Consider the following system operation code:
Chapter 6: Investigating the Evidential Weighting of Logs

Figure 6.2: Sample System Operation Code

Depending upon where in the code calls to log an entry are made, it may be difficult to determine whether the system operation was successful or not. A system operation may start with a log entry being generated by the initiating system event or one that is close to it (for eg. Request_Write_File()). It is possible that a subsequent system event may return a fail condition if the system operation was unsuccessful. If a log entry is not generated at this point, then it may not be possible to deduce whether the system operation failed from the log. Further, in the absence of a clear indication that the system operation failed, the forensic investigator may be lead to believe the operation was actually successful. Where the system operation branches out into two or more paths, log entries must be potentially generated for each path so that all key activities can be identified from the log file.

This problem is particularly important in non-object oriented systems where operations may consist of a sequence of events beginning with a calling system event and ending with a receiving system event. At the time of calling, the identity of the receiving event is unknown. Therefore it is not possible to determine whether the system operation was completed successfully without both events.

In object-oriented systems, there is a centralized entity called the object manager. The object manager catches the initiating request, approves it and forwards it for completion. In this case logging should potentially occur at all three points within the process: at the initiating request, at the object manager approving the request, and the completion of the system event.
An important weighting concern has been raised in this section. *Even if log entries are successfully grouped and system operations are uniquely identified, there remains the possibility that not all the activity executed by the system operation is identifiable. In addition, the outcome of the system operation may not be known either.*

### 6.3.4 Accurate Identification of a Sequence of Operations

Forensic investigators typically seek to establish a timeframe within which system activity related to the incident must be identified. Part of this exercise is to establish a timeline of key events within that timeframe and to ensure the sequence of these events is accurate. In this case the weighting of log evidence depends on the relative positioning of logging requests to the beginning/end of the system operation.

Consider the influence of event detection where there is only one place in the code (figure 6.1) and an event entry may be written to the log. Assume the code represents an entire system operation rather than just a small fragment. This scenario is common in operating systems where a single log-entry is frequently accepted as representative of a system operation (see example of process startup in section 4.2.1).

Execution of this code will generate a single entry in the event log. A forensic investigator seeking information about this system operation will have only one log-entry with a single timestamp available upon which to make deductions. Obviously, the only logical deduction to be made is that the system operation occurred. However, a number of pertinent observations related to timing can be made.

Firstly, if the log records a single timing value related to the system operation, then obviously this value must have been written at some point in the code (one of the write-entry requests). This implies that the entry timestamp recorded in the event log represents a single instance when the system operation was known to be in progress. Further, it is unknown how close the entry timestamp is to the beginning or end of the system operation (in terms of lines of code between beginning of operation and call to report the log entry).

This scenario may be further complicated when multiple system operations are progressing simultaneously. Consider a second example to illustrate this problem. System operation...
starts at time $t_1$ and ends at time $t_2$ whereas system\_operation\_y starts at time $t_3$ and ends at time $t_4$. And $t_1 < t_3$ and $t_2 > t_3$. Therefore in terms of timing, these two system operations overlap (system\_operation\_x starts before system\_operation\_y and ends after system\_operation\_y has started but before it has finished). The following diagram illustrates this example graphically:

![Diagram showing multiple system operations running simultaneously](image)

**Figure 6.3a: Multiple System Operations Running Simultaneously**

The following is one possible outcome:
If an entry is written at $t_3$ for system\_operation\_y and at $t_2$ for system\_operation\_x then since $t_3 < t_2$, an entry for system\_operation\_y will appear first in the log.

The following is the other possible outcome:
If an entry was written at $t_1$ for system\_operation\_x and $t_3$ for system\_operation\_y then the entry for system\_operation\_x will appear before that of system\_operation\_y since $t_1 < t_3$.

In this scenario, the chronological sequence of log entries appearing in the log file is not predictable. As a consequence, forensic investigators may not identify the correct sequence of events as they took place during the incident. This may have significant consequences for the ‘bigger picture’.

This discussion highlights another weighting concern regarding log evidence. *Forensic investigators attempting to establish the sequence of events in an incident must be aware simultaneously executing system operations may not necessarily generate log-entries in a predictable pattern.* The chronological sequence of log-entries does not necessarily reflect the order in which overlapping incident events occurred. Further, log-entries related to an
individual system operation may be generated at any time while the operation is executing. These entries may not give an accurate representation of the execution period of the system operation.

6.3.5 Accurate Identification of Incident Timeframe

From the point of view of the forensic investigator, determining the beginning and end of the incident is crucial as it establishes the temporal relevance of all other activity in the computing system to the incident. Based solely on the event log, an approximation of the incident timeframe can be made once the first and last log entries generated by the first and last system operations (respectively) known to be associated with the incident activity are identified.

In the exploratory phase the forensic investigator is likely to identify key activities that may have occurred during the incident. These are a good starting point to begin grouping existing log entries and linking them to the known system activities. If the grouping exercise is inaccurate then there is a strong likelihood that the first and last entries associated with the incident will be misleading. The previous section established that even if the grouping is successful, the timing information in these entries may not give an accurate indication of the incident timeframe. The chronological sequence of log entries may not reflect the true sequence of events as they occurred in the system. Note, this inaccuracy applies to simultaneously occurring events since they may not write an entry to the log in any predictable sequence.

Consider figure 6.2 once again, this time in the context of the beginning/end of a system operation. Forensic investigators are typically unable to determine whether the sequence of log entries reflects the true chronological sequence of events (unless they have additional information to the log). Therefore, once again, the inaccuracy of the timeframe deduced reflects the difference in the temporal proximity of the first log entry to the true beginning of the incident and the last log entry to the true end of the incident.

If the forensic investigator knew the true chronological sequence of events then perhaps a more accurate approximation of the incident timeframe could be made. However, investigators are typically limited to determining the first and last known log entries
associated with the incident regardless of whether they were generated by the first/last events in true chronological order.

At this point it is useful to highlight an important trade-off faced by forensic investigators. The forensic investigator will use the incident timeframe to establish the relevance of all log entries from all relevant computing systems and all events that take place outside the digital environment to the incident. Due to the uncertainty of the timing information provided by the log entries forensic investigators may be tempted to widen the incident timeframe to compensate. However, investigators must recognize that over inflating the timeframe may result in the need to investigate a large number of log entries unrelated to the incident. At the same time, being too restrictive may result in relevant events being ignored.

Based on the discussion in the last section, it is interesting to note that the general sequence of operations remains preserved even if the system clock is not precise. However, this is not the case with the incident timeframe. From the point of view of the forensic investigator, the beginning and end of the incident is crucial as it establishes the relevance of all incident related activity whether inside the computing system where the incident occurred or in another digital environment or even in the external physical environment. One or more out-of-sync clocks has considerable impact on the ability of forensic investigators to determine the relevance of activities in the different frames of reference.

A primary cause for concern is variation in timing between digital environments. Digital environments typically experience a large number of events in a small time frame. Therefore, even a small variation in the timing between computing systems may inhibit the identification of potential evidence.

6.3.6 Complete Reconstruction of System Activity in an Incident
As mentioned in the previous section, frequently forensic investigations hinge on a particular point in time where a definitive action has occurred. For example, it is not uncommon for an investigation to hinge on establishing whether a particular person knowingly committed a particular action. In such cases a full reconstruction of the scenario would greatly assist the forensic investigator.
Ideally, forensic investigators would like to reconstruct all system activity and identify all the actors that participated at the time of the incident. Total reconstruction helps to eliminate discussion of hypothetical scenarios and restricts discussion to verifiable facts.

In principle, a logging mechanism that can create a record of even the highest frequency system event would make incident reconstruction possible. However, the design of event detection and event description and the fact that the logging mechanism shares the same resources as the environment it is monitoring, makes such a task impractical. Operating systems architects cannot report a log entry for every event that occurs in the operating system as there may be simply too many events occurring in a short period of time. This would result in the denial of service of the availability of system resources. However, a case can be made for reconstruction in limited timeframes.

*Forensic investigators may still partially reconstruct system activity given certain crucial pieces of information and evidence of certain types of system activity. The task of partial reconstruction is to determine what incident-related system activity could have taken place given existing log evidence.* This is different from definitively describing what did take place. The strategy here is to limit hypothetical scenarios based on intelligence gathered about the system configuration, incident circumstances and existing evidence recorded by the log.

Simple logic can be applied to eliminate the possibility that certain system activities took place on the basis of information available to forensic investigators. For example, if investigators are able to determine the precise number and identities of applications that were running at the time the incident took place, then potential system events can be limited to those known to be characteristic of those applications (and those that may be generated by the precise model and build of the installed operating system). This is because the invocation of an application introduces new code to the system and therefore the potential for more events to be logged. Note that the complete set of code from which events can be logged consists of a combination of the operating system and live applications.
6.3.7 Complete Coverage of System Activity

Existing operating systems do not generate log-entries representing system activity from all parts of the digital environment. Forensic Investigators must not assume that if evidence of a certain kind of system activity does not appear in the log, then it never took place in the digital environment. Rather, forensic investigators must identify which kinds of system activities will not be preserved in the log and factor that into the weight of log evidence.

The following sections identify some of the kinds of system activity that is not or cannot be logged:

Limitations of a Self-Logging Service

Practically, it is not possible to create a logging service that monitors all system activity on its own host system. Logically there will be certain kinds of system activity that cannot be logged with a service running in the same environment where the events are taking place.

For example, most operating systems maintain a boot-strap sequence which loads system code into memory after which various parts of the operating systems are brought online in a specific order. The logging service is not the first component of the system to be fully functional and is therefore unable to detect events occurring prior to the point in time where it becomes functional.

Further, the act of creating a log entry involves the execution of one or more events. If those events could generate log entries (not that we would want them to anyway) then we would have an untenable situation. Conceptually, a logging service cannot monitor the ‘act of logging’ itself. This would create a scenario where the logging of an entry would result in further log entries, which in turn would log further entries and so forth.

Logging I/O activity

Most operating systems do not generate log entries for input/output events (Price, 1997). Since the overwhelming majority of corporate usage of computing systems is related to application environments (in particular document generation and management), input/output related activities are likely to be crucial to forensic investigations. More recent operating systems have improved upon this however the complete I/O picture of activity across all devices is not loggable.
Logging network connectivity
Events related to activity at the interface between a host computing system and network is also not loggable by conventional operating systems (Ahmad, 2003). This implies that network traffic may not be linked to applications that are connecting to network ports to send or received information. Forensic investigations into network activities involving email, web browsing, or any other application services frequently focus on tracing network traffic back to individuals that can be held accountable. This ability seriously undermines forensic investigations into incidents where network access plays a key role.

Logging events inside application environments
Of all the previously discussed constraints on logging system activity, the most significant is the wholesale absence of log entries related to application activity.

Even though most operating systems provide the mechanism for logging application events, however most application services do not take advantage of this interface. Operating systems have not been designed to allow administrators to force applications to log events.

In general, operating systems see applications as a black box in that very little known about what is happening inside the environment. When this circumstance is combined with the absence of information about user activity (in particular user interaction events - which will be discussed in the next section), then forensic investigators will find it difficult to rule out a number of possibilities such as the possibility of a Trojan horse (see section 2.3.1 for a detailed discussion of the Trojan horse defense).

Logging user interaction
Evidence of user interaction through the operating system cannot be found in system activity logs. Logging user interaction is a difficult problem. In some operating systems applications implement their own user interface with minimal participation from the operating system. In other cases, the operating system acts as an intermediary vehicle by passing on user interface events to applications. Even if the user interface events were passed by the operating system, the volume of events is likely to be very high compared to other system activity in the operating system. Forensic investigators are typically interested in establishing the intention of a user. In particular, they are interested in establishing whether an instruction issued in the digital environment can be reliably attributed to the user as opposed to another application or
Chapter 6: Investigating the Evidential Weighting of Logs

Trojan. Even if user interface activity could be logged, user interface interrupts can come from anywhere especially in transparent operating systems.

6.4 Influence of event selection on weight of log evidence

Where event detection determines the logging capability of a computing system, event selection determines how the logging capability may be used to write events during an arbitrary session of operation. Selection does not decide how an individual event will be described; rather its role is limited to allowing or disallowing an event from being logged.

Section 4.2.2 identified two primary aspects of selection that affect the capability of operating systems to produce evidence of system activity. They were

- Control of the characteristics of the event data set
- Guidance given to administrator to assist in collecting sets of useful forensic event data

Therefore, the reasons for the absence of evidence of a particular system activity are two. Either the activity could not generate events as designed by the operating system or the system administrator decided not to generate events if that activity occurred.

Event selection raises some similar issues to event detection, such as ‘uniquely identifying a system operation’ and ‘evidence of key activities in a system operation’. This is expected, as the recording of individual log entries depends upon both event detection capability and event selection configuration. Again, event detection defines the set of events for which log entries can be potentially generated, event selection acts as a filter that determines which events will generate entries for a particular logging session. Therefore, the absence of evidence for particular logging activity can be due to the inability of the system to generate entries or the selection policy enforced at that point in time.

6.4.1 Control Over Evidence Generation

Since event selection influences which log entries are generated, this indirectly impacts whether system operations can be identified. Section 4.3.2 describes three scenarios where constraints arising from the design of event selection may influence the evidence written to the event log:
1. Event logging degrades system performance to a point where logging becomes infeasible
2. Event logging degrades system performance to a point, forcing logging to be limited to short intervals only
3. Event logging does not degrade system performance, allowing uninterrupted logging

Although administrators are allowed to select or tag from a limited choice of system events using a management console, frequently systems apply further constraints by preventing the selection of individual system events to the exclusion of others. An “all or nothing” approach frequently results in too much data being logged or too little. For example, where a piece of code is executed with high frequency, systems administrators will either create an event entry every time it runs or will not create any entries at all. Ideally, if an entry can be created given particular circumstances, then administrators have more control over the characteristics of the event data set generated.

*Where selection infrastructure imposes coarse granularity, systems administrators will have little control. On the other hand, fine granularity allows a high degree of control. Here the administrator has more flexibility in the choice of what may be logged.*

Events such as object accesses are typically generated very often. These events are reported from the same piece of code. *If there is no further selection control provided other than ‘on/off’, logging high-frequency events such as object accesses will result in a significant performance penalty.*

In this case, the wholesale absence of evidence of high frequency system activity implies the completeness of log evidence will suffer significantly. Incident reconstruction becomes largely impractical. Further, even though existing conventional operating systems do not log user interface events and network access events, there is an in-principle difficulty with allowing logging from these sources as they generate events with high frequency. This implies that evidence from these events are unlikely to make a common appearance even if system activity from those sources were to be made available for logging. Accurate identification of system activity that may be assisted by evidence of low-level events will also suffer.
Low-frequency events are unlikely to result in severe performance penalties. Therefore, if logging a low-frequency event will be useful then it is likely the event will be selected for logging. This certainly coincides with current security advice given to systems administrators on event selection (section 3.3.1).

**Uniquely identifying a system operation**

The design of event selection may affect the potential for unique identification of system operations. Section 6.3.2 argued that due to the design of event detection, system operations may not generate enough log entries to allow forensic investigators to uniquely identify them.

In a similar vein, the design of event selection may result in a performance penalty for the logging of certain crucial events needed to identify a system operation. For example, it is common for events associated with object accesses to generate large numbers of entries. Since all processes use the same piece of code that enables object accesses, when logging is enabled it results in a significant performance penalty. This design inevitably leads systems administrators to use logging of high-frequency events sparingly. Without evidence of activities such as object accesses, forensic investigators may not be able to identify a system operation or key activities conducted by the system operation. This is especially true when the pattern of object accesses for specific system operations is both unique and consistent. Such ‘signatures’ may not be detected if such system behavior is not visible from log evidence.

*In general, where the design of event selection imposes an ‘all or nothing’ choice, there may be a scenario preventing one or more events from being logged. A possible consequence may be a significant reduction in the number of log entries recorded resulting in insufficient evidence to differentiate between two or more system operations.*

**Evidence of Key Activities in a System Operation**

Selection may have an impact on the ability of a forensic investigator to identify key activities in a system operation from log evidence. This is possible in circumstances where the granularity of selection is fine enough to select one event to the exclusion of another.
For example, consider the system operation ‘writing to file’. This operation may consist of multiple events such as ‘request of a write operation’, more than one ‘status of a write operation’ and a ‘completion of a write operation’. If only the first event is selected for logging purposes, then evidence of the failure of this operation will not be preserved.

In this case, there is a similarity between event selection and event detection. *The impact of not selecting an event on the availability of log evidence is similar to not making an event available for logging.*

**Accurately Identifying a Sequence of Operations**

Section 6.4.4 presented an example that demonstrated how event detection influences the accuracy of timing information in event log evidence so as to potentially affect the chronological sequence of system operations. A timeline was used to show how the sequence of two system operations can be confused depending on when events are reported to the log (these system operations must be executing simultaneously for at least a small period of time for the uncertainty to exist).

In principle, in an operating system where selection has sufficiently fine granularity, the sequence of operations can be influenced by the choice of which events are logged and which ones are not. Consider the time-line example. This time take two events per system operation rather than one.

System operation \(_x\) starts at time \(t_1\) and ends at time \(t_2\) whereas system operation \(_y\) starts at time \(t_3\) and ends at time \(t_4\). However there are two further events occurring at \(t_5\) and \(t_6\).

And \(t_1 < t_3\) and \(t_2 > t_3\). Therefore system operation \(_x\) starts before system operation \(_y\) and ends after system operation \(_y\) has started but before it has finished. The following figure illustrates this example graphically:
In this example depending upon which events are selected for logging we may have a different sequence of log entries.

The following is one possible outcome:
If the event occurring at \( t_5 \) is selected and the event occurring at \( t_1 \) is unselected for system_operation_\( x \) and similarly events occurring at \( t_3 \) is selected and \( t_6 \) is unselected for system_operation_\( y \) then since \( t_3 < t_5 \) therefore an entry for system_operation_\( y \) will appear first in the log.

The following is the other possible outcome:
If events occurring at \( t_1 \) is selected for system_operation_\( x \) and \( t_3 \) is selected for system_operation_\( y \) then the entry for system_operation_\( x \) will appear before that of system_operation_\( y \) since \( t_1 < t_3 \)

**Accurate Identification of Incident Timeframe**
The previous section suggested event selection may influence evidence of the chronological sequence of system operations. This argument can be extended to suggest the design of event selection may result in the inaccurate identification of incident timeframe.
6.4.2 Guidance on Evidence Selection

Whereas ‘Control over evidence generation’ looks at which events can be selected for evidence generation purposes, ‘Guidance on evidence selection’ looks at how event selection should be exercised in different cases.

System administrators use a selection interface (built into the operating system) to define the set of events they want to log. Note that this selected set of events is a subset of the larger set that represents the logging capability of the computing system (i.e. event detection). Unfortunately, most operating systems feature selection interfaces that are modeled on system events rather than real-world events (Ahmad & Ruighaver, 2002).

In general, event selection interfaces do not efficiently map real world events to event logging configurations. Instead, selection interfaces tend to reflect system architecture. For example, to log user activity taking place during a Windows logon session, administrators will, at a minimum, tag logon/logoff and process tracking to capture evidence of a logon session and applications running during that session. To log accesses to objects such as files, each individual file must be tagged, specifying which type of access should trigger an event and in the case of which user or group.

Administrators interested in collecting evidence of real-world activity must identify which events must be tagged to generate sufficient evidence to meet evidence goals. Often, this will involve creating a mental model of the real-world activity and breaking it down into its individual system actions that can be logged separately. Windows’ selection interface mirrors its own architecture in that the event interface is not defined in terms of real-world events but Windows events (see snapshots and description in section 4.1.6).

As previously discussed in section 6.4.1, lack of flexibility in the control over evidence generation may make it difficult to select the evidence required without logging unnecessary information that may ultimately degrade system performance.

*Therefore, generating evidence of real-world events (such as the usage of a particular application service) is difficult as it requires precise actions, subjects, objects and so forth to be identified before any evidence can be generated.*
<table>
<thead>
<tr>
<th></th>
<th>Accuracy</th>
<th>Completeness</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy of Individual Log Entry Timestamp</td>
<td>√</td>
<td></td>
</tr>
<tr>
<td>Accuracy of Actor Identification in Individual Log Entries</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>Accuracy of Action Identification in Individual Log Entries</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>Actor Identification Across Multiple Log Entries</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>Consistency of Log Evidence Across Multiple Log Entries</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>Uniquely Identifying a System Operation</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>Identifying Key Activities in a System Operation</td>
<td></td>
<td>√</td>
</tr>
<tr>
<td>Accurate Identification of a Sequence of Operations</td>
<td>√</td>
<td></td>
</tr>
<tr>
<td>Accurate Identification of Incident Timeframe</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>Complete Reconstruction of System Activity</td>
<td></td>
<td>√</td>
</tr>
<tr>
<td>Complete Coverage of System Activity</td>
<td></td>
<td>√</td>
</tr>
<tr>
<td>Control Over Evidence Generation</td>
<td></td>
<td>√</td>
</tr>
<tr>
<td>Guidance on Evidence Selection</td>
<td></td>
<td>√</td>
</tr>
</tbody>
</table>

Table 6.1: Issues Related to Accuracy and Completeness of Log Evidence

6.5 Influence of Log Management on Weight of Log Evidence

Continuity of the evidential record applies to the interval illustrated in figure 5.1 as ‘B’ + ‘C’. (for simplicity this interval begins when the last incident-related event has been logged). As discussed in section 5.2, the continuity of evidence begins within the digital environment but extends outside into the physical world once extracted. As this thesis is only concerned with the period where the evidential record remains within the digital environment, the focus on the interval designated ‘B’ only.

The second and third conditions of authenticity relate to the forensic investigator’s extraction of the evidence from the digital environment. Log management activities do not directly influence these two authenticity conditions. The forensic investigator will testify to the procedures used when extracting evidence including the date/time when the extraction took
place and circumstances of the extraction including the identity of the host. The event log may play a role in establishing authenticity as a contribution to its evidential weight.

6.5.1 Security of evidential record

Security attacks on data have been classified into three categories. They are attacks on confidentiality (C), integrity (I), and availability (A) (Stallings 1995). Figure 6.4 is a data flow diagram that charts the lifecycle of a log entry from the point of creation to the point of retirement. The types of attacks have been identified as well as the precise points where they may be leveled during this lifecycle.

Step 1: A process captures data in the system environment and writes it to a log file
Step 2: A backup process writes the event log to a storage medium
Step 3: The event log file is accessed by multiple processes reading from, or writing to, the event log file
Step 4: A restore process may retrieve the event log from backup and write it to an active file
Step 5: A retirement process will destroy the event log from its existing medium

In this lifecycle there are multiple opportunities for attacks. Attacks are either directed towards a file or a process. Confidentiality, integrity and availability attacks can be mounted on files, Integrity and Availability attacks on processes. From an evidential weighting perspective, attacks on confidentiality are not significant since they do not affect the authenticity of the evidence. However, an integrity attack that changes all or part of the
Chapter 6: Investigating the Evidential Weighting of Logs

evidence does affect authenticity. In addition long-term unavailability of the evidence that prevents the forensic investigator from acquiring the evidence also affects authenticity.

<table>
<thead>
<tr>
<th>Files</th>
<th>Processes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event Log Files (local or centralized)</td>
<td>Event Log Environment processes responsible for audit management</td>
</tr>
<tr>
<td>Backup Media</td>
<td>Backup process</td>
</tr>
<tr>
<td></td>
<td>Restore process</td>
</tr>
<tr>
<td></td>
<td>Authorized processes writing to/reading from event log</td>
</tr>
</tbody>
</table>

Table 6-2: Entities Against Which Security Attacks May be Directed

Note that existing operating systems natively offer steps 1 and 3 only.

Internally, event log environments divide security privileges into two main categories, user-level control and kernel-level control. Kernel level control implies trusted super user status; once acquired, all security controls applied by the operating systems on processes or files are no longer effective. In addition the integrity of OS processes like those of the audit subsystem are no longer trustworthy.

If the integrity of kernel-level control remains intact, then operating systems rely almost entirely on a single mechanism for protecting assets like event logs - discretionary access control. This regime relies on denying unauthorized processes rights that lead to a breach of C, I, and A. A legitimate user attempting an unauthorized access to an object will be denied by the operating system based on directives arising from the discretionary access control properties of that object.

The authority to access kernel-level control is typically validated by use of ‘authentication by knowledge’ mechanisms (in this case password entry). As a first line of defence, the security of the password management system itself must be reviewed for vulnerabilities.

If a renegade process gains kernel-level control to an OS, then event log generation past that point in time cannot be trusted. However, event logs that have been written to a log file, and log files that are no longer open for writing can be protected from attack.
6.5.2 Consistency of Log File with Backups
An important consideration for forensic investigators is to determine whether the event log file has been maliciously contaminated. If the log file has been backed up then examining the consistency of information across backups and ‘live’ log files is a means of testing authenticity of log evidence.

6.5.3 Contamination from Reduction Processes
Since a very large number of log entries may be generated, there may be an argument for running automated processes that reduce the size of the logs using some predetermined algorithm. Such an algorithm may be simply discarding log entries deemed to be irrelevant or uninteresting. Another possibility may be to summarize multiple log entries into a single entry that describes activity of a higher abstraction.

From a forensic point of view, modifying the original evidence is essentially contaminating it. Logically, for a forensic authority to accept evidence that has been subject to a reduction process, it must be proven that the reduction process does not influence the weighting of evidence.

6.5.4 Impact of Storage Location on Continuity
Administrators must decide whether to store event logs locally on the system where they were gathered or centrally on another system. This decision has no direct effect on the administrator’s ability to collect particular event data. However, it does have an effect on the forensic requirements that must be met to present data that has moved outside of its native host.

The main reason for storing event data off-site is increased security whereas the best reason to store data centrally is better correlation. A system compromise would render event data vulnerable. Therefore locating evidence off-site forces the culprit to compromise a second system to cover tracks on the first. A chain reaction can be implemented such that the attacker is always forced to break into another machine to cover tracks on the previous machine. In addition, event data should be consolidated to allow for analysis of events across more than a single computer (network attacks DDoS etc.)
However, every time the event log entries are moved the chain of custody becomes longer and the conditions of authenticity must be iteratively applied. This requires the forensic investigator to identify the systems where part or all of the evidence was hosted, the time frames during which the evidence was hosted and the state of the evidence in each such interval.

6.5.5 Contamination from Log Retention Processes

Various retention facilities are offered by conventional operating systems to manage the volume of data held by the event log. Among them are ‘stop upon size limit’ where the administrator designates the maximum log size. If the system approaches the limit the entire digital environment shuts down. Another policy is ‘cyclic retention’ where event logs are limited by maximum size or by operating time limit after which the event log overwrites itself. Finally, ‘cycle across multiple files’ allows event logs to skip to a new file after the first one reaches a designated limit.

Any policy allowing event log entries to be overwritten risks loss or contamination of evidence. The last of the three policies is most suitable from an evidential weighting point of view, overwriting is the least acceptable option whether triggered by time or size limit. However, given the cost of disk space has decreased to a point where it is no longer perceived as a constraint, organizations should no longer be forced to overwrite logs.

6.6 Conclusions

The primary contribution of this chapter is the comprehensive identification of the range of weighting issues that apply to event logs. Recognizing these issues is critical to forensic investigators planning to use log evidence in forensic investigations.

Chapter 1 suggested that although system activity logs are functionally suited to evidence collection, they have not played a useful role in digital forensic investigations in the past. Literature in the area makes three relevant observations here. Event logs were not designed to log system activity frequently of interest to forensic investigations. Further, event selection facilities prevent exclusive targeting of particular system activities. And finally, that log entries may not contain enough information to characterize events. Although these observations were made from a system security point-of-view, they identify fundamental
inadequacies of system activity logs in their primary function (collecting information about system activity) that are relevant to evidence collection.

This chapter investigated the weighting of log evidence from a forensic point-of-view. The concerns identified in the literature were also identified in this analysis. However, this forensic-motivated analysis has identified a more extensive range of issues that apply specifically to log evidence.

Important questions are raised about the accuracy of timing information recorded in logs and its impact on the chronological sequence of events, and estimation of incident timeframe. In addition, there is a discussion on the coverage of system activities available for logging and the potential for administrators to have a dramatic impact on the evidential ‘big picture’ arising from the design of the selection infrastructure. The reliable identification of system activity from log evidence is another important point of discussion.

Some unexpected but extremely useful functions of event logs were also identified. Event logs can improve actor identification through profiling their behavior. Partial reconstruction of system activity can be achieved through a combination of logging the right events and examining the digital environment. Further, the log collects information about its environment that may be useful in establishing the authenticity of evidence collected.

The next chapter discusses the evidential weight of system activity logs given evidence admission priorities.
Chapter 7

Maximizing the Evidential Weight of System Activity Logs

This chapter answers the primary research question by prioritizing the weighting issues discussed in chapter 6 according to the rules of evidence applied in courts of law.

Section 7.1 revisits the fundamental concepts of admissibility and weight. In particular, the discussion focuses on the rules of precedence by which legal tests are applied on evidence in courts of law. Based on these rules of precedence, a two-level model for prioritizing weighting issues regarding computer-derived evidence is developed.

Section 7.2 discusses the precedence of legal tests on system activity logs. This is done by identifying the relevant issues in chapter 6, and then examining the strengths and weaknesses of log evidence in the context of the individual tests.

7.1 A Priority Model for Evidential Weighting

Up to this point there has been no discussion on whether one issue regarding evidential weight is more significant than another. This section proposes a two-level model for the prioritization of weighting aspects of computer-derived evidence in general. In this model, relevance is the highest priority weighting consideration (1st level) followed by the series of Sommer’s tests discussed in section 5.1.2 (2nd level).

There are some general rules of law that place more importance on certain aspects of weighting over others. Section 2.4 pointed out that evidence is tested for admissibility before weighting is considered. This distinction is captured in figure 7.1 (below). Computer-derived evidence submitted to a court of law is subjected to admissibility tests before further weighting takes place. The first level of figure 7.1 reflects this separation. There are two kinds of admissibility tests. Relevance is a binary test. If evidence fails this test, then further tests are never applied, as the evidence is immediately deemed inadmissible. However, if the
test for relevance is passed then the test for reliability is applied. Again, the reliability testing that is applied under the rules of admissibility can be differentiated from the reliability testing that is applied as part of weighting. The difference between the two is that in the first case admissibility places conditions on evidence designed to prevent its outright presentation. In the second case weighting concerns the ability of the evidence to convince the court. Note that the reliability considerations categorized under admissibility are concerned with the scientific derivation and appropriate validation with which the evidence has been produced.

**Figure 7.1: Precedence of Admissibility Tests over Further Weighting Considerations**

Evidence is typically not admissible unless found to be logically relevant to the forensic investigation. Relevance must be established before further issues regarding reliability of the process by which evidence is produced are considered. Relevance, in this context, refers to a connection between the evidence submitted to the court and the forensic investigation. According to the discussion in section 2.4.3, relevance is defined lexically and is best left to the logic of the court (Hallen 1998).

Therefore, in practice, forensic investigators need only establish a connection between the evidence submitted and the forensic investigation in a logical sense. If a forensic investigator were to demonstrate that the evidence submitted related to real-world actors in the forensic investigation, then that would be grounds for the evidence to be admitted. Also, if the forensic investigator could demonstrate that the evidence of system activity occurred during the same timeframe, as that of the incident being investigated, then that could also be grounds for admissibility. Finally, if the evidence could be shown to be relevant to a particular action,
or sequence of actions, then that would be a third reason for admission of evidence. In summary where relevance is concerned there are three possible connections – relevance by actor, relevance by timing, and relevance by action (note actor, action and timing correspond to the reporting model described in section 4.2.3).

The discussion on relevance provides the basis for a useful model that can be used to recognize the significance of evidential weighting issues to evidence producers (systems administrators and systems architects). This model suggests that establishing relevance takes precedence over reliability testing in general. If evidence cannot be deemed relevant to the investigation then its reliability is simply never a concern since the evidence will immediately fail at the admissibility test.

Recall admissibility can be divided into two tests – relevance and reliability. Reliability, in the context of admissibility, refers to the question of scientific derivation and appropriate validation. This test plays an important role where evidence is taken from new and emerging technologies. The historic role played by these tests was discussed in section 2.4. In the case of system activity logs, the primary challenges posed by the scientific derivation and validation tests have already been addressed in the ‘best evidence rule’ and the ‘hearsay evidence rule’. Further, some recognized experts have found system activity logs to be generally admissible.

In terms of prioritization of evidential weighting concerns, a conclusion can be drawn from this discussion. Relevance of evidence to the forensic investigation must be demonstrated before further evidential weighting can occur. However, based on this conclusion, the priority is to connect computer-derived evidence to the forensic investigation. This can be done in three ways – by connecting digital actors to real world entities, by connecting system activity to real world acts, and by relating timing information to the real world reference time frame.

Further insight into significant evidential weighting issues can be drawn from Sommer’s reliability tests. These tests are applied to the human-directed process by which computer-derived evidence is extracted from the digital environment for the purpose of presentation to a legal authority.
Chapter 7: Maximizing the Evidential Weight of System Activity Logs

Since any legal authority will apply the tests identified by Sommer in one form or another, they constitute known parameters against which processes involving computer-derived evidence will be tested.

7.2 Prioritizing evidential weighting issues regarding system activity logs
The following section uses the two-level model to prioritize the evidential weighting issues identified in chapter 6. This priority reflects the evidential weighting of system activity logs as a conceptual function in modern operating systems.

The aim of this section is to discuss the evidential weighting of system activity logs within the prioritization framework constructed in the previous section. A primary motivation behind this discussion is the comparison between system activity logs in particular and other kinds of computer-derived evidence available to forensic investigators.

The primary research question of this thesis asks the following:

“How can the evidential weight of system event logs be maximized?”

This section identifies the strengths and weaknesses of system activity logs and their significance for evidential weighting.

7.2.1 Level 1: Establishing Relevance of Computer-derived Evidence
Connecting Digital Actors to Real World Actors
The previous section stated that the relevance of computer-derived evidence to a forensic investigation could be established in three ways. One of these ways is to link actors operating in the digital environment to actors operating in the real world. Recall that real-world actors are users or document exhibits that exist in the physical world whereas digital actors include objects, processes, applications and digitized files.

Section 6.2.2, pointed out that the scheme of digital labeling implemented within operating systems is not sufficiently rigorous such that forensic investigators can readily relate digital
actors to their real world counterparts. System activity logs refer to actors within the digital domain using temporary identifiers created by an authentication scheme implemented within the operating system. Therefore, in the case of actor identification, the weighting of evidence within system activity logs is no different to any other form of computer-derived evidence.

However, section 6.2.4 pointed out that system activity logs are capable of accumulating a vast quantity of information that describes the behavior of real world actors within the digital domain. From this information, a profile can be potentially developed that provides forensic investigators with a more viable means of linking real world actors to evidence of related system activity.

Although applications are not real world actors, they typically play a key role in forensic investigations. They are used by real-world actors to achieve an objective (e.g. attacking a system), or they tend to be the site of the incident itself (e.g. manipulation of an important document). In either case, positively identifying the application may become important. This is especially true where unknown applications are identified on a computing system. A profile of application activity will assist in identifying the functionality of the application and the role it may have played (see practical discussion of profiling in sections 8.1.4 to 8.1.6).

The ability to develop profiles of system activity that can be used to identify actors is a key strength of system activity logs. Given the highest priority in terms of weighting is establishing the relevance of evidence, this capability of system activity logs is especially significant when compared to other computer-derived evidence.

Further, once evidence from system activity logs is rendered admissible, other computer-derived evidence that complements the log will be easier to accept. In this sense, logs can play a role in improving the likelihood that other kinds of computer-derived evidence are admitted.

Connecting System Activity To Real World Acts
Continuing from the opening arguments in the previous section, a second way of demonstrating relevance is to link system activity to real world actions that have been recognized by forensic investigators.
Chapter 7: Maximizing the Evidential Weight of System Activity Logs

Unfortunately there are a number of constraints inherent in the design of system activity logs that impede the generation of an accurate, complete and authentic evidential account of system activity (and positive identification of system activity from the evidence). Therefore, relating log evidence of system activity to a real world incident becomes difficult.

Forensic investigators rely on system documentation for an explanation of log entry jargon. At times this documentation may be obscure and difficult to understand and explain to a legal authority (see section 8.2.1 for example on object-access logging). It is conceivable that forensic investigators may not be able to completely and reliably explain the patterns of entries generated in the log due difficulties in grouping log entries and the absence of critically important types of system activity such as application events. Due to all these reasons, establishing that a particular log or section of a log describes system activity that is relevant to the investigation may be difficult.

Aside from issues of loggability, in many circumstances the lack of sufficiently fine granularity in selection control makes it difficult for a particular set of events to be logged within acceptable performance parameters (see sections 8.1.11, 8.2.7 and 8.3.8 for practical discussion of selection). Note that the selection interface also makes it difficult to identify all the events that must be selected such that sufficient evidence of key activities would be necessarily generated.

However, at the same time, system activity logs provide a rich source of evidence on past system activity that does not exist elsewhere in digital environment. This is despite the fact that there exist a number of concerns regarding the generation and identification of evidence of system activity within existing operating systems.

Further, relevance can be established with small pieces of information. For example, consider a case involving a particular application or certain kinds of system or network activity that are typical to a particular kind of application. A forensic investigator may argue that an application capable of generating the kind of system activity or network activity responsible for the incident resides on the computing system where the incident took place. Compare this argument to one where the forensic investigator is backed by a log entry stating the same particular network application was initiated prior to the incident occurring by the defendant. The latter argument is a stronger basis for establishing relevance than the former as it links
Chapter 7: Maximizing the Evidential Weight of System Activity Logs

the real world act by the defendant to the digital environment. Note that the use of such log entries depends on whether the log entry is considered reliable evidence. If the log entry is deemed unreliable then it may not be used to support the forensic investigator’s argument.

Relating Timing Information from the Digital Environment to Real World Time

A third way of demonstrating relevance is to link timing information within system activity logs to the incident (time) frame of reference identified by the forensic investigation.

In forensic investigations, the time frame of the incident is the frame of reference upon which the relevance of events is decided. Forensic investigators will typically look more carefully at events that occurred during the timeframe of the incident than those that occurred outside the timeframe.

System activity logs bind timing information to log entries reporting actors and actions that occurred on the system. This provision adds three new pieces of valuable information to the forensic investigator. Events of interest to the forensic investigator can be dated. In most cases, the chronological sequence of events can be determined from the log and a time frame of relevant digital events can be approximated from the log as well.

Section 6.3.1 pointed out that the accuracy of the timing information in system activity logs depends primarily on precision of the time source used by the system administrator. There are other design issues that affect the precision of timing information such as atomicity of event reporting, the relative positioning of log entry requests, and the possibility of external (malicious) interference. However, where relevance of evidence is concerned, the provision of binding time to an historic account of system activity renders the evidential weight of system activity logs greater than other forms of computer-derived evidence.

7.2.2 Level 2: Evidential Reliability Tests

The second level of the precedence model refers to a series of tests that demonstrate the extent to which computer-derived evidence is reliable. Although the forensic investigator directs the tests and testifies in court on the reliability of evidence, event logs may also contain useful information about the kinds of issues covered by the tests that assists the establishment of reliability.
Event logs are uniquely positioned to collect evidence about the digital environment that complements the testimony of the forensic investigator in establishing the reliability of computer-derived evidence. Recall this particularly useful characteristic of system activity logs is named ‘utility’ in section 5.3.3. This section further explores the role of utility in the evidential weighting of system activity logs.

**Computer’s Correct Working Test**

A key test of evidential reliability is that the computer is working ‘correctly’ or ‘normally’ (see 5.1.2). This is significant as a flaw in the design or operation of a computer may produce unreliable evidence.

Section 6.3.5 pointed out that any identified inconsistencies in log evidence are potentially a weighting concern. Inconsistencies may occur in actor, action and timing information. These inconsistencies are indications that the computer may not be working correctly and/or the logging mechanism may not be working correctly (external attack is also possible). It is important for forensic investigators to recognize that although system activity logging is an automated function, it is ultimately a programmed service that may contain flaws. Further, at this time event log services have not been built for forensic purposes. Therefore interpretation of log evidence must be carried out with caution.

System activity logs may actively record information regarding the workings of the computer over a long period of time. In this way logs provide a source of information on the correct workings of the computer that can be used to complement the forensic investigator’s testimony on the same topic. This was pointed out in section 5.3.3.

Compared to other computer-derived evidence, system activity logs are capable of providing valuable information on the correct workings of the computer. This makes their evidential weight greater than other computer-derived evidence, such as hard disk files, in this aspect.

**Provenance of Computer Source Test**

Another test of evidential reliability is that the evidence presented by the forensic investigator to the legal authority is demonstrably from the purported source and from nowhere else. The onus is on the forensic investigator to link the particular host to the evidence extracted.
Some event logging systems bind information identifying the host computer to each individual log entry. This allows the entry to be linked to the host where it was created. However, with a single field only, there may be no way of knowing from the log entry how many different computing systems hosted a particular entry once it was recorded. Further, the information recorded in the identification field may not identify the host computer in any useful way. Computer names that identify the user of the computer or a sequence number designed by the systems administrator are common in organizations and may not assist forensic investigators to conveniently and reliably identify the host system.

Before logs are removed from the digital domain by the forensic investigator, they may have been hosted in a number of environments. These may include backup tapes and more than one computing systems (some may have functioned as centralized archives). If the chain of evidence is not maintained, there may be no reliable information on the provenance of source.

**Content/Party Authentication Test**

The content/party authentication test seeks to test whether the evidence presented can be demonstrably linked to the subject of the investigation and/or the parties involved. This implies the identifiers within the evidence would have to be linked with the real world actors or subject of the investigation.

The content/party authentication test is essentially about relevance as previously discussed (see comprehensive treatment of this topic in section 7.2.1).

**Evidence Acquisition Test**

This test is focused on the forensic investigator’s process of evidence acquisition. Evidence must have been gathered from the digital environment accurately, completely and without contamination. As this process is conducted by the forensic investigator and requires the digital environment to be suspended, there is typically no active role for computer-derived evidence to play.

However, an exception to this rule is where the accuracy and completeness of the evidence (in its original form) that is extracted from the environment must be established. Of course this is likely to be the case if the legal authority requires the forensic basis for evidence from
Chapter 7: Maximizing the Evidential Weight of System Activity Logs

system activity logs to be established. Sections 2.4.5 and 3.4 cited numerous concerns regarding the evidential weight of the log evidence service provided by operating systems. The majority of chapter 6 was devoted to identifying a range of issues with the weight of system activity log evidence. The evidence acquisition test is general. Therefore, depending upon the particular forensic investigation, it is difficult to say what impact the characteristics of the system activity log service will have on the overall evidential weighting of computer-derived evidence.

Continuity of Evidence/Chain of Custody Test
The larger concept of ‘continuity of evidence/chain of custody’ is originally focused on the forensic investigator’s custody of evidence. However, the chain of custody extends into the digital environment as the log may move from its host of origin to secondary sites such as a centralized log server. Therefore, this test contributes to the evidential weighting of computer-derived evidence for the period that the evidence remains in the digital environment.

All computer-derived evidence is vulnerable to contamination while it remains in the digital environment. Log evidence is typically in one of two states. Either it is being gathered through a dynamic process of evidence acquisition or it resides statically in a log file. Section 6.5.1 identified a range of vulnerabilities inherent in both states. The integrity and availability of the evidence acquisition process depends largely on the design of the logging mechanism and the security of environment within which the logging mechanism is working (provided by the operating system). Once evidence is written to a file, evidence can be corrupted as a result of external interference as well as log management practices such as reduction and retention strategies. Further, where logs are stored off-site from where they were created, forensic investigators may need to establish the chain of custody. Note that existing logging systems may at best record the identity of the initial host where entries were being first recorded. Subsequent hosts will not be identifiable within each individual log entry.

At the same time it must be noted that log evidence may verify when the computing system was suspended by the forensic investigator for evidence extraction. This can be demonstrated by identifying the last known log entry recorded in the log and comparing the timing information with the forensic investigator’s own testimony on custody issues.
Relative to computer-derived evidence such as data files residing on a hard disk, system activity log files may have less evidential weight. This is primarily because log files are more susceptible to contamination and require more tests to be conducted before weighting can be established.

### 7.3 Conclusions

This chapter answers the primary research question by prioritizing the weighting issues discussed in chapter 6, according to the rules of evidence applied in courts of law. The prioritization takes the form of a two-level model. The first level identifies three fundamental tests to be used when establishing relevance whereas the second level consolidates all remaining issues of reliability and utility.

The aim of chapter 8 is to demonstrate the practical usefulness of the evidential weighting framework by evaluating a real-world event logging system. In order to apply the framework in a systematic manner, the framework must be used to answer specific questions about the operating system’s event logging function. Chapter 8 will argue that the questions of who/what/when are fundamental to any forensic investigation and must be answered in almost all circumstances. Further, these three questions are at the heart of both relevance and reliability hurdles. Therefore, the evidential weighting framework will be used rigorously and systematically to answer these three tests – ‘Connecting Digital Actors to Real-World Actors’, ‘Connecting System Activity to Real World Acts’ and ‘Relating Timing of System Activity to Real World Time’. These three tests form level 1 of Table 7.1.
Chapter 7: Maximizing the Evidential Weight of System Activity Logs

### Evidential Weighting

<table>
<thead>
<tr>
<th>Evidential Weighting Test</th>
<th>Contributions of Activity Logging to Evidential Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Level 1: Establishing Relevance of Computer-derived Evidence</strong></td>
<td></td>
</tr>
<tr>
<td>Connecting Digital Actors to Real World Actors</td>
<td>Profiling digital actors can assist forensic investigators to identify real world actors</td>
</tr>
<tr>
<td>Connecting System Activity to Real World Acts</td>
<td>Although logs are unlikely to record a complete account of system activity in an arbitrary period of time, logs can preserve evidence of various system activities of which some may be relevant to the forensic investigation</td>
</tr>
<tr>
<td>Relating Timing of System Activity to Real World Time</td>
<td>Logs bind timing information to evidence of system activity including actor behavior thereby time-stamping important events, recording chronological sequence and identifying timeframe of important activities</td>
</tr>
<tr>
<td><strong>Level 2: Evidential Reliability Tests</strong></td>
<td></td>
</tr>
<tr>
<td>Computer’s Correct Working Test</td>
<td>Logs can provide an historical account of the workings of a computing system</td>
</tr>
<tr>
<td>Provenance of Computer Source Test</td>
<td>Logs can bind information identifying the host computer to each individual log entry</td>
</tr>
<tr>
<td>Content/Party Authentication Test</td>
<td>See relevance. Note that evidential reliability testing invites more rigorous scrutiny of the evidence compared to evidential relevance testing.</td>
</tr>
<tr>
<td>Evidence Acquisition Test</td>
<td>This depends upon the subject of the forensic investigation (chapter 6 discusses a comprehensive range of issues that may arise in a particular investigation)</td>
</tr>
<tr>
<td>Continuity of Evidence / Chain of Custody Test</td>
<td>Custody inside digital environment: log evidence is highly susceptible to contamination from attack and from internal management processes like reduction and retention strategies; Security: The integrity and availability of log evidence is depends on the security of the logging mechanism and the environment within which the logging service is operating</td>
</tr>
</tbody>
</table>

Table 7-1: Relevance and Reliability Tests
Chapter 8

Applying the Evidential Weighting Framework to Microsoft-Windows Logging

The objective of this chapter is to demonstrate the practical usefulness of the evidential-weighting framework developed in Chapter 6. This is achieved by using the framework to assess the capability of Microsoft Windows to collect log evidence that is both relevant and reliable in a court of law. Relevance of log evidence can be established by demonstrating that a connection exists between the log evidence, on the one hand, and real-world actors, activity and timeframes, on the other (level 1 – table 7.1). Reliability, in its most general sense, seeks to answer the same who/what/when questions that relevance addresses. The difference being that relevance seeks to attain admissibility (on logical grounds), whereas reliability seeks to convince and therefore invites rigorous scrutiny.

Chapter 7 pointed out that the who/what/when ‘tests’ are necessary to demonstrate relevance. As a matter of general practice, an operating system’s logging capability must allow a forensic investigator to answer the following who/what/when questions as they relate to past events:

1. The ‘who’ test: ‘Connecting digital actors to real-world entities’
2. The ‘what’ test: ‘Connecting system activity to real-world acts’
3. The ‘when’ test: ‘Relating timing information to the real-world time frame’

The event log must answer these questions in almost all circumstances, although it must be acknowledged that this may not be sufficient in certain circumstances (e.g. where specific evidence relating to specific system events may be required). Therefore, this chapter investigates the capability of Windows operating systems to answer the fundamental questions of who/what/when by applying the evidential weighting criteria from chapter 6 in a systematic manner:

To apply the evidential weighting criteria systematically, it must be first determined which criteria are relevant to each of the three tests of who/what/when. Not all evidential-weighting criteria apply to all three of these tests. For example, ‘actor identification’ criteria (6.3.2 and
6.3.4) apply exclusively to the ‘who’ test and ‘accuracy of individual log entry timestamp’ apply to the ‘when’ test. Some criteria apply to all three tests. For example, ‘complete coverage of system activity’ and ‘complete reconstruction of system activity’ play a major role in all three tests. Table 8-0a is arranged according to the applicability of each criterion to a test.

An assessment of log management requires the direct application of the log management criteria on Windows. Since Windows only provides facilities for security and retention of event log evidence, these are the only log management facilities that will be assessed. Table 8.0b is therefore arranged differently to Table 8-0a.

The structure of this chapter is as follows. The three fundamental tests are discussed in the first three main sections of this chapter, Sections 8.1 to 8.3. The logical structure of each section is similar and will be illustrated now by using section 8.1 as an example. The forensic investigator’s task in answering the ‘who’ question, is to establish a link between a real-world actor and log evidence. Table 8-0a identifies two criteria that are directly and exclusively linked to the ‘who’ test. These criteria can be identified by the presence of the words ‘actor identification’, i.e., ‘accuracy of actor identification in individual log entries’ (6.2.2) and ‘actor identification across multiple log entries’ (6.2.4). Section 8.1 shows how application of these two criteria gives the investigator logical grounds for assessing the reliability of evidence that connects digital actors to real-world entities. Table 8-0a also identifies some criteria that are applicable on all tests. For example, the operating system must facilitate the grouping of multiple log entries related to a specified actor (criterion 6.3.1) to allow profiles to be constructed. Further, to construct activity profiles, guidance must be provided to assist selection of evidence as well as allowing the evidence to be generated given performance limitations (criteria 6.4.1 to 6.4.2). An indicator of the usefulness of an actor profile is the completeness of actor activity (criterion 6.3.7). Inconsistencies in the integrity of evidence of actor identification must be investigated. Finally, the potential for identifying evidence that can be used to reconstruct system activity must also be explored. Thus the full set of criteria useful in making Fundamental Assessment 1 are 6.2.2, 6.2.4, 6.3.1, 6.3.7, 6.2.5, 6.3.6, 6.4.1 and 6.4.2.

Each of Sections 8.1 through 8.3 is divided into a number of sub-sections, with each sub-section divided into two parts. The first part presents the findings of the investigation after
applying the relevant criteria. The second section explains how the criteria were applied and (where necessary) presents technical background to the findings (figure 8.1 is a template for sub-sections in chapter 8).

At the end of each sub-section, entries are made in Table 8-0a to provide a summary of the criteria from Chapter 6 shown to be necessary to assess the weight of the evidence discussed in that sub-section. For example, sub-section 8.1.1 on “user actor identification” uses criterion 6.2.2 for evaluating the usefulness of Windows log evidence. At the end of section 8.1.1, “8.1.1” is placed in the row for criterion 6.2.2 in the column headed Fundamental Assessment 1 in Table 8-0a. At the end of section 8.1.2 on “application actor identification”, “8.1.2” is placed in this same cell, because the same criterion (6.2.2) was also needed for evaluating the evidence in Table 8.2. At the end of section 8.1.3 on “file actor identification”, “8.1.3” is placed in this same cell, because the same criterion (6.2.2) was also needed for evaluating the evidence in Table 8.3.

The result of this process, which continues through each subsection of Sections 8.1 through 8.3, is that by the end of Section 8.3, Table 8-0a summarizes all the criteria needed for conducting each of the three Fundamental Assessments in this chapter. The non-blank cells in the three columns on the right indicate which criteria from Chapter 6 were useful in making each of the three Fundamental Assessments discussed in this chapter (which, as indicated earlier, are useful for establishing the reliability of Windows log evidence in a court of law). Moreover, the “non-blank” entries in the cells identify the sub-sections where application of the criterion is discussed. For example, as shown in Table 8-0a, criterion 6.3.7, Complete
Coverage of System Activity, was necessary for evaluating the evidence in Windows logs relating to Fundamental Assessment 1 in sub-section 8.1.8, and to Fundamental Assessment 2 in sub-section 8.2.4.

Depending on the conclusions drawn from the application of evidential weighting criteria, each cell is given a representative color. The basis for the color assigned to a cell is purely subjective. It represents the author’s opinion as to the severity of the concerns arising from the discussion. Green indicates measures do exist to address a weighting concern whereas Red indicates the concern remains unmitigated thereby constituting a significant issue. Orange means although measures do exist, some concerns remain outstanding. As the colors may not be visible on black-and-white print, a ‘G’ has been placed where the color is green, an ‘R’ has been placed when the color is red and an ‘O’ has been placed where the color is orange. Table 8-0a thus acts as the summarizing device for the assessment of evidence collection capability. In a similar way, Table 8.0b acts as a summarizing device for the assessment of log management capability that takes place in section 8.4.

Finally, to conclude the chapter, section 8.5 discusses the usefulness of the weighting criteria in the evaluation, and presents recommendations for Windows’ administrators and engineers on increasing evidential weight.

Due to the large number of issues discussed in this chapter, a table has been included as a summary reference. The issues most likely to be of interest to the reader are colored in red (some issues in the orange category are also interesting). The section numbers associated with a particular test and applied criteria allow parts of the evaluation to be accessible without having to read the entire chapter.
## Chapter 8: Applying the Evidential Weighting Framework to Microsoft-Windows Logging

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>6.2.2</td>
<td>Accuracy of Actor Identification in Individual Log Entries</td>
<td>[O] 8.1.1, 8.1.2, 8.1.3 (log evidence does not provide reliable link between actors and real-world entities)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6.2.4</td>
<td>Actor Identification Across Multiple Log Entries</td>
<td>[R] 8.1.4, 8.1.5, 8.1.6 (little actor behavior captured from available log evidence; profiles cover few types of activities)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6.2.3</td>
<td>Accuracy of Action Identification in Individual Log Entries</td>
<td></td>
<td>[O] 8.2.1 (object-access evidence is misleading; logon and process tracking evidence is accurate)</td>
<td></td>
</tr>
<tr>
<td>6.3.2</td>
<td>Uniquely Identifying a System Operation</td>
<td></td>
<td>[R] 8.2.2 (object-access operations exercising a single right multiple times cannot be distinguished from each other)</td>
<td></td>
</tr>
<tr>
<td>6.3.3</td>
<td>Identifying Key Activities in a System Operation</td>
<td></td>
<td>[R] 8.2.2 (process call event for object-access is not recorded including most actual accesses)</td>
<td></td>
</tr>
<tr>
<td>6.2.1</td>
<td>Accuracy of Individual Log Entry Timestamp</td>
<td></td>
<td></td>
<td>[R] 8.3.1 (accuracy of system clock)</td>
</tr>
<tr>
<td>Section</td>
<td>Description</td>
<td>Evidence Codes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
<td>----------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6.3.4</td>
<td>Accurate Identification of a Sequence of Operations</td>
<td>[R] 8.3.2 (lack of complete timing allows for inaccurate identification of sequence of operations; timing resolution may be too coarse for identifying sequence of events within system operations)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6.3.5</td>
<td>Accurate Identification of Incident Timeframe</td>
<td>[O] 8.3.3 (if the timeframe is less than one second then accurate identification is difficult; cannot identify start time for system operations)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6.3.1</td>
<td>Grouping Related Log Entries to Identify System Operations</td>
<td>[G] 8.1.7 (all activity sessions can be grouped for profiling) [G] 8.2.3 (groups operations well but cannot group threads belonging to the same process) [G] 8.3.4 (Windows relates log evidence that takes place in one second; Grouping activity at sub-second level is not possible since timing resolution is limited to one second)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6.3.7</td>
<td>Complete Coverage of System Activity</td>
<td>[R] 8.1.8 (bad for identifying users and applications, good for files but still unhelpful) [R] 8.2.4 (many important system activities are not available for logging) [O] 8.3.5 (Windows typically logs exit conditions leaving out starting conditions; not all timings of object reads and writes are available)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6.2.5</td>
<td>Consistency of Log Evidence Across Multiple Log Entries</td>
<td>[G] 8.1.9 (consistency checks can be applied on user, application and file identification) [G] 8.2.5 (consistency of logon sessions assisted by evidence of app sessions, same relation btw app and object-access,) [G] 8.3.6 (consistency of timing information regarding logon sessions assisted by evidence of app sessions, same relation between app and object-access)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table 8-0a/b: Assessment of Windows Evidence Collection and Management Capabilities

<table>
<thead>
<tr>
<th>Section</th>
<th>Topic</th>
<th>Assessment</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.3.6</td>
<td>Complete Reconstruction of System Activity</td>
<td>[O] 8.1.10 (selection policy useful; other avenues of questionable value)</td>
</tr>
<tr>
<td>6.4.1</td>
<td>Control over Evidence Generation</td>
<td>[O] 8.1.11 (fine/coarse granularity of control; file activity profile efficient; degrades performance for app activity profiles)</td>
</tr>
<tr>
<td>6.4.2</td>
<td>Guidance on Evidence Selection</td>
<td>[R] 8.1.12 (profiles of users/apps requires multiple policies to be selected; profiles of objects only one policy)</td>
</tr>
<tr>
<td>6.5.1</td>
<td>Security of Evidential Record</td>
<td>[G] 8.4.1 (measures taken to protect log files, service, and architecture; other measures discussed in research not implemented)</td>
</tr>
<tr>
<td>6.5.2</td>
<td>Consistency of Log File with Backups</td>
<td></td>
</tr>
<tr>
<td>6.5.3</td>
<td>Contamination from Reduction Processes</td>
<td></td>
</tr>
<tr>
<td>6.5.4</td>
<td>Impact of Storage Location on Continuity</td>
<td></td>
</tr>
<tr>
<td>6.5.5</td>
<td>Contamination from Log Retention Processes</td>
<td>[R] 8.4.2 (default strategy will result in loss of evidence; )</td>
</tr>
</tbody>
</table>
8.1 Connecting Digital Actors to Real-World Actors

Common real-world actors that frequently become the subject of a forensic investigation are users and documents (e.g. incidents involving document drafting and editing) and application services (e.g. incidents involving usage of network services like email and web browsing and involving document management like word processing) (section 2.2.2).

To determine the reliability of the link between log evidence and real-world actors, a number of criteria will be applied. Evidential weighting criteria directly and exclusively related to actor identification are - ‘accuracy of actor identification in individual log entries’ (criterion 6.2.2) and ‘actor identification across multiple log entries’ (criterion 6.2.4). The first criterion will be applied to determine how conclusive the link is between individual log entries in Windows and real-world actors. The second criterion will be used to assess the reliability of actor profiles generated in Windows log evidence. As Windows handles users, processes and files differently, each actor will be investigated in separate sections. Generic weighting criteria will be subsequently applied to further investigate Windows’ actor identification capabilities. For example, ‘Complete coverage of system activity’ (criterion 6.3.7) will be applied to determine the potential range of activities in Windows’ actor profiles. ‘Complete reconstruction of system activity’ (criterion 6.3.6) will be used to determine how much evidence of actor identification can be constructed from available log evidence. Windows’ ability to generate evidence of actor identification will be assessed using evidential weighting criteria developed for selection interfaces. ‘Control over evidence generation’ (criterion 6.4.1) will be used to assess Windows’ capability to produce evidence of actor identification and ‘Guidance on evidence selection’ (criterion 6.4.2) will be used to assess the usefulness of Windows’ advice on how to take advantage of its evidence generation capabilities.

8.1.1 User Actor Identification in Individual Windows Log Entries

The presence of a username may appear to be sufficient grounds to link Windows log evidence to a real-world user. For example, a forensic investigator may argue that the Windows user-account name ‘adam’ may be related to real-world person ‘Adam Smith’ because the log evidence is making a direct reference to the real-world actor. Of course this association is incorrect since account names cannot be directly linked to a real-world user.
The logic of the court may require the forensic investigator to demonstrate (using deduction rather than induction) that the log evidence relates to ‘Adam Smith’ and nobody else. This demonstration may become necessary if ‘Adam Smith’ claims that he did logon, but that at some time in the future he left the terminal and is therefore not responsible for any subsequent system activity.

In general, positive identification (by physical features) of a real-world user is not possible given the only input into the digital environment is user input. Instead, Windows simply logs process activity, but records the real-world user account name because it is associated with the process conducting the activity being logged. Therefore, forensic investigators will find that in Windows a username such as ‘adam’ cannot be conclusively linked to ‘Adam Smith’ (the real-world user).

User profiling may offer further assistance in identifying the real-world user. This provision can be used to identify a consistent pattern of user input that is characteristic of the real-world user to rule out evidence of third party involvement. Windows’ capabilities in this regard are assessed in the section on ‘Profiling User Activity for Improved Actor Identification’ (section 8.1.5).

**Application of Evidential Weighting Criterion 6.2.2**

<table>
<thead>
<tr>
<th>Test</th>
<th>Criterion Applied</th>
<th>Conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connect User Actors in Log Evidence to Real World User?</td>
<td>Apply ‘Actor Identification in Individual Log Entries’ (6.3.2)</td>
<td>No conclusive link between ‘username’ and real-world actor; Must profile user activity to investigate reliability of link between Windows and real-world actor</td>
</tr>
</tbody>
</table>

**Figure 8.1: User Actor Identification in Individual Windows Log Entries**

To test the reliability of user identification in individual log entries, a forensic investigator will find the criterion ‘accuracy of actor identification in individual log entries’ (6.2.2) to be useful. Two key points were raised in chapter 6 that are relevant to this discussion. Firstly, typical schemes of digital labeling rely on a static identifier that is not sufficiently rigorous such that a forensic investigator can conclusively link digital actors to their real-world
counterparts. Secondly, timing information can be used to be more specific about when a user was active on a system thereby increasing the reliability of the link.

User actor identification aims to identify a real-world user from evidence of his/her presence in the digital environment. Therefore, forensic investigators will be testing the reliability of the link between log evidence and real-world users. Windows recognizes the authority of real-world users to logon by means of a username / password combination. Establishing the credentials of a user creates a link between a digital identifier and a real-world user. From this point onwards Windows does not attempt to re-authenticate the user. Therefore, Windows is effectively assuming that the same user that was authenticated at logon will continue to issue all user-initiated inputs until an explicit logoff takes place. Log evidence of user activity refers to the real-world user by the username only.

The user account name appears in a large number of log entries during any logon session. The presence of the user account name gives the impression that the log entry is generated as a result of a user action. This is not the case. Windows is actually logging process activity rather than user activity. The user account name appears because the process is conducting the particular activity on the user’s behalf.

Technically, the process acting on behalf of the user (explorer.exe) spawns another process that executes the instructions in “IEXPLORE.EXE”. Although the user plays a key role in this operation, at least from a process-level view this contribution is not visible. Therefore, at the process-level, the key activities are:

1. Explorer.exe spawns a process
2. New process runs application ‘IEXPLORE.EXE’
Table 8-1: Process-tree entries Leading to Internet Explorer Startup

Windows generates two log entries from which both steps can be clearly identified. Note that the log entries in Table 8-1 can be used to reconstruct the process tree. The log entries are in reverse chronological order (match the creator process id of the first entry to the process id of the second to form a tree):

This sequence of log entries was generated when a user started Internet Explorer. However, precisely the same log entries can be generated in an alternate set of circumstances. Windows produces the same sequence of log entries if the application is started up automatically without a user’s explicit involvement. For example, using a batch file or script rather than an
interactive command. Forensically, the distinction between these two scenarios is significant because it goes to show intent. An argument could be made that if the real-world user is shown not to be involved directly with the startup of an application then it may follow that the real-world user was not involved in subsequent activities as well. This is an opportunity to use the Trojan horse defense argument (the Trojan horse defense argument is explained in chapter 2 - section 2.3.1).

This process tree demonstrates that Windows log evidence does not record distinct evidence that captures the difference between an application starting up as part of a login script or an application starting interactively by a user through the console. Note that process ‘userinit’ runs programs when a user logs in (Userinit, n.d.). In this case it runs the environment manager explorer.exe.

Therefore, in general, forensic investigators must realize that although in most cases the user logged in will be responsible for starting applications, there are exceptions to this rule. Other than the login script, which doesn’t show explicitly in log evidence, other batch files or sources configured for automatic startup may also start applications without the need for explicit user intervention. Process identifiers may be used to identify parent processes and backtrack all the way to the root of the process tree. This investigation will ultimately lead to the top-level explorer.exe that initiates automatically on logon.

Another issue identified using this criterion is that Windows does not actually store the name of the user in the event log record. Instead, a ‘user source identifier’ (USID) is stored and used by the event viewer to reference the username in the SAM. If the security event log contains event records associated with a user whose account name has been changed then the user field will change accordingly. If an event record is associated with an account that has been deleted and therefore removed from the SAM, then the event viewer will display a blank user field. If the event is not associated with a specific user account then the Event Viewer displays a N/A in the User column (Murray page 44-45).

Regarding the second point brought up in chapter 6, the time of logon is useful as it indicates when the user underwent authentication. This information makes two contributions. Firstly, it assists forensic investigators to argue that someone (a real-world user) was definitely sitting at the console at a particular time. Secondly, this information may assist forensic
investigators to establish the identity of the real-world user through other sources of evidence such as CCTV (security surveillance cameras).

<table>
<thead>
<tr>
<th>Event ID</th>
<th>Type</th>
<th>Description</th>
<th>Time</th>
<th>Computer</th>
<th>User Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>560</td>
<td>Success</td>
<td>Object Open</td>
<td>25/10/2006 5:09:39 PM</td>
<td>DIS-WS0002062</td>
<td>adam</td>
</tr>
</tbody>
</table>

Object Name: C:\Program Files\Microsoft Office\OFFICE11\1033\ID_011.DPC
Handle ID: 556
Image File Name: C:\Program Files\Microsoft Office\OFFICE11\WINWORD.EXE
Accesses:
READ_CONTROL
SYNCHRONIZE
ReadData (or ListDirectory)
ReadEA
ReadAttributes

<table>
<thead>
<tr>
<th>562</th>
<th>Success</th>
<th>Handle Close</th>
<th>25/10/2006 5:09:39 PM</th>
<th>DIS-WS0002062</th>
<th>adam</th>
</tr>
</thead>
</table>

Handle ID: 556
Process ID: 2864
Image File Name: C:\Program Files\Microsoft Office\OFFICE11\WINWORD.EXE

Table 8-2: Microsoft Word Accesses a Binary File

Looking at log evidence, if the evidence shows that an application decides to read a particular file, it is not clear from the evidence if the user instructed the application to read the file or the application initiated the action itself. This cannot be established from the log. In the log entries in Table 8-2 it appears as if the log evidence is suggesting that ‘adam’ instructed Microsoft Word to open ID_011.DPC. In fact, the application is reading a binary source file as part of its routine loading process (the full listing is available from Appendix A-4).
Without sources of information on actor identification from outside the computing environment, forensic investigators will have no other recourse than to resort to user profiling. This technique can be used to show a pattern of user input to the Windows environment over numerous sessions of computer use. A largely recurring pattern may indicate there was no evidence of third party involvement. Evidential weighting criteria ‘actor identification across multiple log entries’ (criterion 6.3.4) will be used to assess Windows’ ability to create user profiles for identification purposes (see section 8.1.5).

The logic underlying ‘Accuracy of actor identification in Individual Log Entries’ (criterion 6.2.2) has shown that there is no conclusive link between a ‘username’ appearing in individual log entries and a real-world actor. Therefore, the sub-section number for this section (8.1.1) appears in the Fundamental Assessment column ‘Connecting Digital Actors to Real-World Actors’ of Table 8-0a for the aforementioned criterion.

8.1.2 Application Actor Identification in Individual Windows Log Entries
In the context of a forensic investigation, an application is seen as an agent capable of performing real-world services. Therefore, applications such as Microsoft Exchange and Internet Explorer may be described as an email service and browsing service respectively. Windows identifies applications by the filename (Windows NT) or path (Windows XP) of the application image file. This evidence may be used to argue that an important application involved in a forensic investigation is identified in log evidence. However, the validity of this argument rests on whether the court will accept the relationship between the filename/path and the application service itself to be a logical link. In fact, from an evidential perspective, there is no connection between the filename or path of an application and its real-world function. Therefore, this argument is fundamentally incorrect. Unfortunately, on the surface of the argument, this relationship may appear to be convincing if the application and its corresponding filename are well known (eg. MSWORD.exe or WINWORD.exe).

Timing information may be useful in arguing that a particular application was operational during a specific period of time. Further, other sources of evidence, such as evidence of network activity occurring during the timeframe when the application was operational may be used to infer the functional behavior of the application. Forensic investigators must profile application activity to identify application functionality. Windows’ capabilities are assessed
in the section titled ‘Profiling application activity for Improved Actor Identification’ (section 8.1.6).

**Application of Evidential Weighting Criterion 6.2.2**

<table>
<thead>
<tr>
<th>Test</th>
<th>Criteria Applied</th>
<th>Conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connect Application Actors in Log Evidence to Real World Services?</td>
<td>Apply ‘Actor Identification in Individual Log Entries’ (6.2.2)</td>
<td>No conclusive link between filename/path and application service; Must profile process activity to establish functional behavior of application service</td>
</tr>
</tbody>
</table>

**Figure 8.2: Application Actor Identification in Individual Windows Log Entries**

Once again, to test the reliability of application identification in individual log entries, a forensic investigator will find the criterion ‘accuracy of actor identification in individual log entries’ (6.2.2) to be useful. The same two points raised in chapter 6 are useful in this discussion (see 8.1.1).

Although both real-world users and application services are actors in the digital environment, there is a distinct difference in the role that they play in the digital environment. Unlike real-world users that are limited to issuing commands and input data into the system, application services carry out a range of functions within the digital environment. Actor identification, as it pertains to a real-world user, aims to link user input in the digital environment to the real-world user. Actor identification, as it pertains to an application service, aims to link the range of input that the application is capable of performing to the real-world application service as understood by the court.

Windows logs only one pair of entries tracking application activity. The following table shows an application process starting and then subsequently exiting (see Appendix A-2 for full entry details). The log entries show that the (application service) process 3500 was created on 12/11/2006 9:35:47 AM from the WINWORD.EXE file that is located in a uniquely identified directory (see ‘Image File Name’ fields in Table 8-3).
Chapter 8: Applying the Evidential Weighting Framework to Microsoft-Windows Logging

Since it is well known that MSWORD.EXE and WINWORD.EXE are common filenames of the Microsoft Word Application executable, forensic investigators may try to argue that this log evidence is relevant to a forensic investigation where a word processing service was used (of course the timeframe of the incident and the identity of the particular computer would have to be consistent with this argument).

However, this connection is not reliable as the filename of an application bears no direct relationship to its functional behavior (see discussion in section 6.2.2). Therefore, conclusively identifying the function of an application is not possible from the filename or path. This is certainly the case for Windows. In Windows NT, log entries link only the filename of the application image, whereas with Windows XP (as above) the full pathname is available. Both operating systems use the name/location of the file on the filesystem to identify the application. Note that Windows NT does not even provide the precise location of the application image file.

Further, Windows applications are broken down into multiple files of which one is an executable and the rest may be dynamically linked libraries (dlls). In this case, an application’s functional behavior may be influenced by libraries that are loaded during the execution of the process. For example, Windows application services, like Microsoft Word,
load instruction sets from dynamically linked libraries (dlls) (note log entry linking application process to dll in Table 8-2). In this case, the functional behavior of an application changes depending upon which libraries are linked at what time. Therefore, in a fundamental sense, a static identifier is not a reliable means of identifying an application in operating systems.

The usefulness of timing information in log evidence can be used once again to strengthen the reliability of the actor identification link. This information may be useful in arguing that a particular application was operational during a specific period of time. Further, other sources of evidence such as evidence of network activity occurring during the timeframe when the application was operational may be used to infer the functional behavior of the application.

In general, profiling application behavior is well suited to actor identification (section 6.2.4). This is because an actor activity profile is essentially a record of the range of actor functionality, which is how applications are identified. Windows’ capability to profile applications is discussed in section 8.1.5.

The logic underlying ‘Accuracy of actor identification in individual log entries’ (criterion 6.2.2) is useful in establishing that there is no conclusive link between a filename/path of an application and the service it provides in the digital environment. Therefore, the sub-section number for this section (8.1.2) appears in the Fundamental Assessment column ‘Connecting Digital Actors to Real-World Actors’ of Table 8-0a for the aforementioned criterion (6.2.2).

8.1.3 File Actor Identification in Individual Windows Log Entries
Real-world documents are identified on the basis of content, whereas in Windows log evidence, documents are identified by a filename or path depending on the version of Windows. In principle, it is impossible to positively identify the contents of a document from a static identifier. Further, the content of a document may change whereas the identifier remains the same.

Although object-access entries may identify a unique file, this evidence is only useful if the file contents (at the time that the log entry was created) can be established. For example, a backup of the file system created at the time the log was written may be used to identify the file. However, since the static identifier has no relation with the content of the file, there is no
way of knowing whether the file retrieved is precisely the same file that was identified in the log (given there is no file hash available). Trying to establish a link between a static identifier and the contents of a real-world document is difficult, as forensic investigators must argue that a particular document with particular real-world contents is identified in log evidence despite the (frequently poorly named) label. Windows’ capabilities regarding file identification are further discussed in ‘Profiling File Activity for Improved Actor Identification’ (section 8.1.7).

Application of Evidential Weighting Criterion 6.2.2

<table>
<thead>
<tr>
<th>Test</th>
<th>Criteria Applied</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connect File Actors in Log Evidence to Real World Documents?</td>
<td>No conclusive link between filename/path and document content; Object-access profiling may be useful in determining real-world content of files</td>
</tr>
<tr>
<td>Apply ‘Actor Identification in Individual Log Entries’ (6.2.2)</td>
<td></td>
</tr>
</tbody>
</table>

Figure 8.3: File Actor Identification in Individual Windows Log Entries

The criterion ‘Accuracy of actor identification in individual log entries’ (6.2.2) is applied once again. This time the criterion is applied on the accuracy of file identification. The same two points raised in chapter 6 are useful in this discussion (see 8.1.1).

The aim of file actor identification is to identify the real-world content of a document from log evidence. In this sense, it is different from both real-world user identification where the focus is on connecting user input to real-world users, and application service identification, where the focus is on linking the range of application functionality to a real-world application service.

There are two types of actions that are performed on documents in the Windows environment. File management involves reading/modification of document environment attributes such as filename, location in the filesystem and access permissions. Reading and writing of document content is the second of the two categories. The latter is forensically more important as it relates directly to the content of the document.
Chapter 8: Applying the Evidential Weighting Framework to Microsoft-Windows Logging

<table>
<thead>
<tr>
<th>Event ID</th>
<th>Type</th>
<th>Description</th>
<th>Time</th>
<th>Computer</th>
<th>User Name</th>
</tr>
</thead>
</table>

Type: File
Object Name: C:\Documents and Settings\adam\My Documents\bluehills.pdf
Handle ID: 272
Image File Name: C:\Program Files\Adobe\Acrobat 7.0\Reader\AcroRd32.exe
Accesses:
READ_CONTROL
SYNCHRONIZE
ReadData (or ListDirectory)
ReadEA
ReadAttributes

<table>
<thead>
<tr>
<th>Event ID</th>
<th>Type</th>
<th>Handle</th>
<th>Time</th>
<th>Computer</th>
<th>User Name</th>
</tr>
</thead>
</table>

Handle ID: 272
Image File Name: C:\Program Files\Adobe\Acrobat 7.0\Reader\AcroRd32.exe

Table 8-4: Object-Access Log Entries

Windows records evidence of file activity only on NTFS volumes (file activity cannot be logged on FAT and FAT-32) under the ‘object-access’ category of logging. The log entries in Table 8-4 were recorded when Adobe Acrobat was used to open ‘bluehills.pdf’ in Windows XP. Note that operation-based logging (event id 567) associated with this action will be discussed in ‘Profiling File Activity’ in section 8.1.8 (full listing of this table is in Appendix A-3).

The discussion surrounding ‘accuracy of actor identification in individual log entries’ (criterion 6.2.2) in this chapter, points to the disconnection between filename and the content. Applying this criterion to Windows results in confirmation of the above observation. In this case, there is no direct relationship between ‘bluehills.pdf’ and the contents of the file. The file contains an image of a scenic landscape.
Windows XP identifies files within log entries by their full path. Windows NT uses only the name of a file (not the path) as the only means of identification. The latter case raises the possibility that there may be more than one file with the same name on a single system. Note that Windows systems have been using the full path to identify files since Windows Server 2003/Windows XP.

Forensic investigators may use the filename or path to retrieve a copy of the real-world document from the hard disk. However, the reliability of the link between the real-world document and the one subject to forensic investigation can be easily challenged. The discussion of the weighting criterion in chapter 6 points out that although the filename/path of a document may not change, the content may have changed a number of times. Therefore, to identify the real-world document, forensic investigators have to find a backup that was created at precisely the same time as that specified in the object-access log entries. However, theoretically, since the static identifier has no relation with the content of the file, there is no way of knowing whether the file retrieved is precisely the same file that was identified in the log (given there is no file hash available). File access profiling may still be useful in detailing file version history. From this history, the most accurate version of a file may be identified. See section 8.1.7 for the application of ‘Accuracy of actor identification across multiple log entries’ on Windows log evidence.

The logic underlying ‘Accuracy of actor identification in individual log entries’ (criterion 6.2.2) is useful in establishing that there is no conclusive link between a filename/path of a document and its real-world content. Therefore, the sub-section number for this section (8.1.3) appears in the Fundamental Assessment column ‘Connecting Digital Actors to Real-World Actors’ of Table 8-0a for the aforementioned criterion.

8.1.4 User Activity Profiles for Improved Actor Identification
A user-activity profile documents the activity taking place within multiple logon sessions that may be attributed to a single user. Windows features three types of log entries that describe activity that may be undertaken by real-world users. These are user logon/logoff, process creation/exit and object-access.
User actor identification is achieved by making a link between a real-world user and input/data issued to the computing system. Previous discussion on user actor identification (section 8.1.1) concluded that individual Windows log entries do not conclusively link username to real-world actor. This section applies the evidential-weighting criterion ‘Accuracy of actor identification across multiple log entries’ (6.2.4) to Windows log evidence. Discussion of this criterion in chapter 6 points out that multiple log entries referring to the same actor can be used to construct a behavioral profile. This profile can be used to provide crucial insights into the real-world identity of actors.

A user activity profile documents the activity taking place within multiple logon sessions that may be attributed to a single user. Windows features three types of log entries that describe activity that may be undertaken by real-world users. These are user logon/logoff (sample entries in Table 8-10), process creation/exit (sample entries in Table 8-1), and object-access (sample entries in Table 8-4).

Connecting logon/logoff entries from the same user is straightforward. Table 8-10 shows a logon (event id 528) and logoff (event id 538) entry pair that identifies the user account name. Therefore, to construct a profile of a particular user’s logons and logoffs, all events with identifier 528 and 538 that identify the particular user can be collected. Similarly, incorporating application and file activity into a user activity profile requires process creation/termination events and object-access events to be linked respectively (see sections 8.1.5 and 8.1.6). Therefore, over a period of time, logon/logoff and process startup/shutdown entries form a pattern that is characteristic to the particular user and can be used to determine the likelihood that a user is responsible for a specific action.
Chapter 8: Applying the Evidential Weighting Framework to Microsoft-Windows Logging

Note that since Windows log evidence collection only takes place in the process environment, application profiling is limited to the logging of process creation/termination and requests for object-access only. Crucial evidence of activity within application environments and relating to network access are unavailable (see section 8.1.5 for a more comprehensive discussion).

Windows allows all successful instances of logon/logoff for all users to be logged by taking a single action. Similarly, all successful instances of process creation/termination may also be logged by taking a single action. However, although object-access can be turned on using a similar method, each individual object must be separately configured for logging purposes. As a result, it is more likely that logon/logoff and process creation/termination will be logged by system administrators in comparison with object-access logging.

Note, that on a stand-alone system, the only evidence of direct interaction by the user is interactive user logon/logoff. Neither process creation/exit nor object-access is necessarily generated by explicit user involvement. Although they frequently act upon user input, it is difficult to prove that a particular action was initiated by a user and from nowhere else (see sections 8.2.1 and 8.2.2 for justification). Therefore, they are not reliable indicators of user behavior that can be used for identification purposes.

Further, given the discussion above, the three categories of log evidence featured in Windows’ log evidence consist of a fraction of the range of activities undertaken by Windows’ users. Despite this fact, the logic underlying ‘Actor identification across multiple log entries’ (criterion 6.2.4) is still useful in identifying Windows’ evidence that can be used to construct a user activity profile. Therefore, the sub-section number for this section (8.1.4) appears in the Fundamental Assessment column ‘Connecting Digital Actors to Real-World Actors’ of Table 8-0a for the aforementioned criterion.

8.1.5 Profiling Application Activity for Improved Actor Identification

Windows provides facilities for applications to log internal activity. However, most application developers, including Microsoft, do not use such facilities. Therefore, Windows’ log evidence collection only takes place in the process environment. As a result, application profiling is limited to the logging of process creation / termination and requests for object-access only. Although some useful information can be extracted from process and object
interactions, crucial evidence of activity within application environments and relating to network access are unavailable. Given forensic investigations are likely to focus on the usage of application services such as Word Processing, Email and Browsing, Windows application profiling is unlikely to be useful when identifying patterns of activity within the application environment and when connecting to the network interface (unless applications begin to take advantage of Windows logging facilities). However, even with the limited evidence collection ability, application activity profiles can be used to monitor the process creation/exit and object-access request behavior of both trusted and un-trusted applications.

### Application of Evidential Weighting Criteria 6.2.4

<table>
<thead>
<tr>
<th>Test</th>
<th>Criteria Applied</th>
<th>Conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Profile Application Activity to Identify Range of Application Functions</td>
<td><strong>Apply ‘Actor Identification in Multiple Log Entries’ (6.2.4)</strong></td>
<td>Application profiles can be made from process tracking and object access logging. They can monitor process creation/exit and object-access request behavior of both trusted and un-trusted applications.</td>
</tr>
</tbody>
</table>

**Figure 8.5: Profiling Application Activity for Improved Actor Identification**

Applications are identified by the range of functions they are capable of performing. Previous discussion on application actor identification (section 8.1.2) concluded that individual Windows log entries do not conclusively link the filename/path of an application image file to the range of application functions. This section applies the weighting criterion ‘actor identification across multiple log entries’ to identify evidence that can be used to connect application actors to their activity profiles.

There are two kinds of Windows log entries that characterize application process behavior. These are process creation/exit (sample entries in Table 8-1) and process requests for object-access (sample entries in Table 8-3). From these types of entries, forensic investigators may extract some useful information. For example, patterns of applications starting/exiting, objects of interest to applications, the approximate time when objects were requested, and the account name of the user officially logged in when the application was running. In the case of Windows Server 2003 / XP, some evidence of the different types of actions executed by the
application on specific objects is also available (the significance of this type of log entry is discussed in section 8.1.6).

Many patterns of object-access displayed by applications such as Microsoft Word and Internet Explorer are likely to be unique to the particular application. This is particularly the case with the loading sequence of applications. For example, a sample log entry is presented in Table 8-7, where Microsoft Word accesses a dll when loading its instruction set. Constructing a profile of activity of the same application requires more work compared to the case of the user (see Table 8-3). At the heart of the problem is the fact that applications are identified by a range of functions rather than a process identifier. This range of functions is linked with a particular filename (note that filenames are not reliably linked with program instruction content). In principle, a unique filename can be used to filter the log to produce a list of log entries with process identifiers that have been related to the application image file. These log entries form an application activity profile.

Application activity profiles can be used for different purposes. In the case of applications that are trusted by a system administrator, application activity profiling can be used to check if changes have been made that may compromise that trusted status. In the case where an application is un-trusted, activity profiles can help to identify the range of functions exercised by the un-trusted application. Where applications are trusted and user input is unavailable, an activity profile may be used to demonstrate consistency in the known and trusted behavior of the application. In this case, it may be easier to demonstrate that any anomalous activity is likely to be the result of user intervention. Where applications are untrusted, activity profiles can assist in identifying application functionality based on exhibited behavior. For example, Windows’ application activity profile may be useful in both identifying and/or confirming that an application is WINWORD.exe, based on which dlls are accessed and with what frequency. In the case of newly installed applications, this activity profiling is unlikely to be useful for identifying unknown applications and especially Trojan horses. This is because objects that are involved in application profiles must be previously selected for logging purposes or else they will not appear in log evidence (unless new objects inherit audit settings). In addition, objects introduced after the selection policy is implemented (and do not inherit elements of the audit policy) will not appear in the log. Further, rogue applications are unlikely to utilize existing (audited) objects so they may function without generating log entries.
This discussion raises an interesting issue regarding the use of activity profiling for the investigation of un-trusted applications in Windows. Changes to the content of an application image file and/or its libraries will affect the integrity of its activity profile. Since the instructions within an application image file defines the range of functions available to an application process, therefore a change of content may result in a change of functions; this change will effect a new activity profile. If forensic investigators do not recognize this change, any interpretation of the activity profile will be misleading.

Without an application activity profile, forensic investigators are only able to identify a Windows application based on its filename or path. With the assistance of the application profile, an investigator may show that Windows log evidence refers specifically to a filename/path that is typical for a particular Windows application, and that the application process accessed a set of files that are exclusively associated with that application. And further, that the pattern of access was consistent with the behavior of that application. However, this method is limited to applications where object-access patterns are unique and can be shown to be consistent.

Therefore, the logic underlying ‘Actor identification across multiple log entries’ (criterion 6.2.4) is useful in establishing that the evidence of application activity in Windows event logs consists of characteristic object-access patterns that can be used to demonstrate consistency between a particular application process and a known application service. Therefore, the sub-section number for this section (8.1.5) appears in the Fundamental Assessment column ‘Connecting Digital Actors to Real-World Actors’ of Table 8-0a for the aforementioned criterion.

**8.1.6 Profiling File Activity for Improved Actor Identification**

According to the object-access audit policy, Windows is capable of logging the complete range of actions that any process may potentially execute on files. This would imply that Windows’ object-access profiling is comprehensive as the intentions of a process regarding file execution, read content, change content, query attributes, modify attributes, delete object, take ownership, and change permissions can be logged separately.
However, Windows NT only logs requests for access to objects rather than the accesses themselves. Windows XP logs actual accesses although not all of them. Further, Windows XP’s capability to produce comprehensive profiles of file activity, these profiles do not provide forensic investigators with direct evidence identifying the content of real-world documents. However, file access profiling is useful as it allows a list of application processes that have accessed a particular file to be produced. This list may assist in identifying the format and structure of the file.

### Application of Evidential Weighting Criteria 6.2.4

<table>
<thead>
<tr>
<th>Test</th>
<th>Criteria Applied</th>
<th>Conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Profile File Activity to Identify Real-World Content</td>
<td>Apply ‘Actor Identification in Multiple Log Entries’ (6.2.4)</td>
<td>Windows logs many object-access requests and some actual accesses but these do not identify real world content; File activity profiling can give clues regarding file format and structure;</td>
</tr>
</tbody>
</table>

**Figure 8.6: Profiling File Activity for Improved Actor Identification**

Real-world documents are identified by their content, unlike in the digital environment, where they are referenced by filename or path. Previous discussion on file actor identification (section 8.1.3) stated that the content of documents in the digital environment could not be conclusively identified from filename or path. This section applies the weighting criterion ‘actor identification across multiple log entries’ (criterion 6.2.4) to identify document actors through file activity profiles.

Since Windows preserves process-object interaction using process creation/exit (sample entries in Table 8-1) and object-access log entries (sample entries in Table 8-4), these can be used to profile both application process behavior as well as file activity behavior. Using this profiling capability, forensic investigators may extract different types of object-access requests as well as a list of application processes along with timing information. In the case of Windows Server 2003 / XP, a partial record of the different types of actions executed by the application on specific objects is also available.
A list of object-access types that may appear in a file activity profile appears below:

<table>
<thead>
<tr>
<th>Object Access Permission</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute File</td>
<td>Script or .exe executed</td>
</tr>
<tr>
<td>Read Data</td>
<td>Content of file read</td>
</tr>
<tr>
<td>Read Attributes</td>
<td>Attributes of file (eg. Read Only, Archive, System, Hidden) queried by Explorer</td>
</tr>
<tr>
<td>Read Extended Attributes</td>
<td>Extended attributes (as defined by applications) queried by Explorer</td>
</tr>
<tr>
<td>Write Data</td>
<td>Content of file changed</td>
</tr>
<tr>
<td>Append Data</td>
<td>Content appended to end of file</td>
</tr>
<tr>
<td>Write Attributes</td>
<td>Attributes of file (eg. Read Only, Archive, System, Hidden) modified by Explorer</td>
</tr>
<tr>
<td>Write Extended Attributes</td>
<td>Extended attributes (as defined by applications) modified by Explorer</td>
</tr>
<tr>
<td>Delete</td>
<td>Object Deleted; allows/denies deleting subfolders and files</td>
</tr>
<tr>
<td>Read Permissions</td>
<td>Object’s ACL queried</td>
</tr>
<tr>
<td>Change Permissions</td>
<td>Object’s ACL modified</td>
</tr>
<tr>
<td>Take Ownership</td>
<td>Owner of object changed</td>
</tr>
</tbody>
</table>

Table 8-5: Object Access Auditing Policies

The above table shows the comprehensiveness of Windows’ file access profiling. In fact, the table also states that any requests to read or change the permissions of a file or take ownership of a file can also be logged.

Constructing a file activity profile from log entries is straightforward where the path of the file is uniquely identified (see Table 8-4). In this case, object-access log entries identifying the same path can be collected to form a complete activity profile. However, where only the filename rather than the complete path is used (such as in Windows NT), the file activity profile may be referring to different files with exactly the same name.

A more fundamental problem exists with the way object-access is logged. When a process requests access to an object, Windows NT generates an object-access log entry. However, this entry is generated at the point where the security check is made (the check is made to decide whether the process should be given access to the object) before the object is accessed. This implies that Windows NT ‘object-access’ does not log the accessing of an object at all. Rather, it logs the security check conducted when a process requests access to an object.
Chap
Chapter 8: Applying the Evidential Weighting Framework to Microsoft-Windows Logging

object (see application of ‘Identifying key activities in a system operation’ - section 8.2.2 for a more comprehensive discussion of this topic). Note that it is likely that most well-behaved processes requesting to use an object will subsequently complete the action. However, from a forensic point of view, Windows NT does not log evidence of the action being completed. It is relevant to note that in later versions of Windows (Windows Server 2003 and Windows XP), Microsoft introduced operation-based auditing which allows some direct process-to-object accesses to be logged.

Table 8-6: Operation-based Logging Entries

Windows XP logged the entries below as part of the object-access listings represented in Tables 8-4 and 8-2. Note that in 567 or ‘object access attempt’ entries, Windows records evidence of an application reading a particular object. The identity of the object is determined by matching the Handle ID in the 567 entry with the corresponding field in the 560 ‘Object Open’ entry where the filename/path of the object can be found.

Without a profile of file activity, a forensic investigator is only able to identify an application based on its filename/path. With a file activity profile, there is significant information about processes that are interested in the object. From this information, a list of application
processes that have accessed a particular file may be produced. This list may assist in identifying the format and structure of the file. For example, if a file has only been accessed by Microsoft Word, then it may be ‘a Word file’. Although Windows demonstrates a comprehensive capability of logging object accesses, this profile relates to file management not file content. Therefore, Windows’ file activity profiles do not directly assist forensic investigators to identify real-world content.

The logic underlying ‘Actor identification across multiple log entries’ (criterion 6.2.4) is useful in establishing that although there is no direct evidence in Windows event logs from which file content can be derived, some inferences may be made about file structure and content from a history of file activity. Therefore, the sub-section number for this section (8.1.6) appears in the Fundamental Assessment column ‘Connecting Digital Actors to Real-World Actors’ of Table 8-0a for the aforementioned criterion.

8.1.7 Grouping Related Log Entries for Actor Profiling
Windows uses unique identifiers within log entries to allow user logon sessions, process activity sessions, and file activity sessions to be identified from log evidence. By collecting these sessions for particular actors, log evidence can be used to generate actor profiles.

Applying Evidential Weighting Criterion 6.3.1

<table>
<thead>
<tr>
<th>Test</th>
<th>Criteria Applied</th>
<th>Conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connect Log Evidence to Real-World Actors?</td>
<td>Apply ‘Grouping Related Log Entries’ (6.3.1)</td>
<td>Windows provides link identifiers that can be used to identify logon sessions, process sessions and object sessions. All such sessions belonging to a particular actor can be collected for profiling purposes.</td>
</tr>
</tbody>
</table>

**Figure 8.7: Grouping Related Log Entries for Actor Profiling**

To conduct actor profiling on Windows, log entries identifying the same user/application/file must be related by ‘grouping’ techniques. The Windows security-logging model revolves around the concept of ‘sessions’ (Tables 8-1, 8-3 and 8-4 are examples of sessions). A session is a period of time where a resource is being utilized, whether that is a user logon session, an object-access session, or a process activity session. Windows allows these activity
sessions to be identified from the log by matching a log entry generated at the beginning of the session to the log entry generated at the end.

Regarding user profiling, previous discussion has established that the only indication of user interaction with Windows is user logon/logoff. This logon to logoff period is known as a ‘logon session’. Therefore, Windows logging creates a single entry-pair to reflect the beginning of a transaction and the end of a transaction (see Table 8-10). To profile a single user’s logon/logoff behavior, all log entries related to logon/logoff (eg. 528/538) with the same username can be collected.

During a single system boot session Windows generates unique identifiers for each spawned process. When a system process or user opens an executable, Windows logs an entry with event ID 592. This log entry identifies the executable with its full path and the account that initiated the process. When the process terminates, another log entry is written, this time with event ID 593. All security events involving this process will include a process ID field so that those events can also be linked (see Table 8-2). To profile an application actor, all events with id 592 (create process) with the particular application image file must be collected. Once the process identifiers are determined, all event log entries with those process ids can be collected to create an application actor profile. Note in Table 8-1 Windows also links the parent process to the daughter process by including both process IDs at the point where the new process is spawned. This allows the spawning sequence of processes to be identified so that back-tracking can occur which introduces a measure of accountability. This is particularly important, as system operations can be identified from low-level log entries. The ‘Process Tracking’ category through event IDs 592 and 593 potentially provide a complete history of all windows programs executed on a system.

Object-access events can be used to track attempts to access files and other Windows objects. Files, folders, services, registry keys and printer objects can be tracked using this facility. Further, Windows assigns different event IDs for a session that begins when an object is opened and ends when it is closed (Table 8-3 shows ID 560 for open and ID 562 for close). To create an activity profile of an object, all events with ID 560 (object open) with the particular path (note filename is not a unique source of identification) must be collected. For each object-open (entry with ID 560), the corresponding object-close (entry with ID 562)
must also be collected. This can be achieved by linking the ‘Handle ID’ that is unique to the pair (object-open/object-close).

Logon events can be matched with logoff events to determine the exact duration of the user’s logon session. Further, process logging can be used to connect the running of any programs to the particular user logon session. In fact, the approximate duration that an application has been running can be determined from the log entry generated when the application started up and the log entry generated when the application shut down. Further, if object-access is enabled for those entities that are accessed during the logon session, then the object-access patterns of each and every application started up by the user can also be identified (recall object-access entry patterns reflect application behavior not user behavior). Therefore, log entries related to a particular user logon session can be identified and entries related to an application’s behavior could be identified as well. Also, access patterns relating to a particular object can also be identified.

Other than the session based logging model, it is also relevant to note that the first log entry in the pair is more comprehensively detailed than the second. It documents the security transaction that allows the session to begin whereas the second entry is sparsely documented as it only allows the event viewer to match the second entry with the corresponding first one to determine the duration of the session.

Therefore, the logic underlying ‘Grouping Related Log Entries’ (criterion 6.3.1) is useful in determining if log evidence can be grouped so that actor profiling can be used to connect log evidence to real-world actors. Therefore, the sub-section number for this section (8.1.7) appears in the Fundamental Assessment column ‘Connecting Digital Actors to Real-World Actors’ of Table 8-0a for the aforementioned criterion (6.3.1).

8.1.8 Complete Coverage of System Activity for Actor Identification

A brief look at the architecture of Windows reveals that user interaction with applications first passes through an interface provided by Windows before it is passed on. This implies that Windows has access to raw user input and therefore, in principle, can make the information available for logging. Windows’ application profiles are limited to patterns of application process creation/exit and interaction with objects. Therefore, the wholesale absence of key evidence of activity taking place inside application environments as well as
connections to the network significantly reduces the usefulness of Windows’ application profiling. Previous discussion of file activity logging (section 8.1.6) has established that Windows logs all accesses to files, however these file management accesses do not constitute evidence of file content.

### Application of Evidential Weighting Criterion

<table>
<thead>
<tr>
<th>Test</th>
<th>Criteria Applied</th>
<th>Conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connect Actors in Log Evidence to Real-World Actors?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Apply ‘Complete Coverage of System Activity’ (6.4.7) to user, application and file identification</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Key activity within application environments and network access patterns are unavailable for logging</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Windows logs all file accesses but these are not (direct) evidence of file content</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 8.8: Complete Coverage of System Activity for Actor Identification**

This section applies ‘Complete coverage of system activity’ (criterion 6.4.7) to the investigation of user, application and file actor identification in the Windows environment. Previous sections have identified system activity that can be logged for actor identification purposes. This section investigates the potential for logging additional evidence in Windows to increase the weighting of actor identification.

**Complete Coverage of User Activity**

Applying this criterion to Windows shows that Windows has the potential of creating a comprehensive user profile. An examination of the internal architecture of Windows indicates that although the operating system cannot reach inside an application environment to capture user input, it still retains control over raw user input (see section 4.1.1) The Window manager inside the Executive is responsible for passing user input to the application. This component of the operating system may be subjected to some (limited) logging to capture keystrokes. Although it may be argued that logging mouse clicks is likely to have a negative impact on system performance, the impact is likely the same or at least comparable
to some existing provisions such as the logging of some potentially high-frequency object accesses such as folder traversal and file attribute reading (see section 8.1.7). A clever OS designer may be able to minimize or avoid performance penalties, for example, by appending raw user input to other (unrelated) log entries – i.e. a type of log-entry piggybacking.

**Complete Coverage of Application Activity**

Regarding application activity, forensic investigators are more likely to be interested in network access patterns and activity inside application environments especially since forensic investigations are likely to focus on email and/or browsing services. Although Windows does have a dedicated application log, the logging of application events is left to application designers and there is no regulation of when application events are generated and what they describe. Application developers have used this facility to log casual messages regarding significant events related to the application process’ interaction with other parts of the operating system. Therefore, events related to what is happening inside the application environment are not available. The following error entry is a typical example of an application log entry, the source is the VirusScan application. The variable field suggests the entry was generated when an update procedure failed:

<table>
<thead>
<tr>
<th>Event ID</th>
<th>Type</th>
<th>Source</th>
<th>Time</th>
<th>Computer</th>
<th>User Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>257</td>
<td>ERROR</td>
<td>Alert Manager Event Interface</td>
<td>14/10/2006 5:35:55 PM</td>
<td>DIS-WS0002062</td>
<td>N/A</td>
</tr>
</tbody>
</table>

VirusScan Enterprise: The update failed; see event log,(from DIS-WS0002062 IP 127.0.0.1 user SYSTEM running VirusScan Ent. 8.0.0 UPD)

**Table 8-7: Typical Entry in an Application Log**

Windows has the capability to log a large amount of evidence regarding application interaction with the operating system from which significant inferences can be drawn about what might be happening inside the application environment. In general, Windows provides an environment within which applications can utilize system resources in an orderly manner. The operating system wraps itself around applications, passes user input via the Window Manager, and resolves requests for all services including interaction with external media and network access through the I/O manager. Windows’ existing inability to log the above activity implies that network traffic may not be linked to applications that are connecting to
network ports to send or receive information. Forensic investigations into network activities involving email, web browsing, or any other application services may require network traffic to be traced back to individuals that can be held accountable. This inability undermines forensic investigations into incidents where network access plays a key role.

For example, in any legal proceeding involving application activity, a challenge can be made that a Windows application is actually a Trojan Horse and that the application was initiating a number of tasks of its own accord such as connecting to the network and sending out confidential information. In this case, a profile of the application’s activities from log evidence would not be able to identify any network activity undertaken by the application.

If Windows was able to log the use of operating system and network services, then an additional source of timing information related to application activity would also become available. The timing of key events taking place elsewhere on the system could be correlated with the timing of application activity. Identifying applications that were connecting to the network at the same time that malicious traffic was generated from the system would be an effective means of inferring Trojan horse behavior in application processes.

In line with its security goals, Windows XP has also introduced a single event id (861) that can be logged when an application process attempts to listen for incoming traffic on a host system. This is a failure event so it only shows up if failure events are logged for processes.
### Table 8-8: Failure Event for Application Logging

Further, an additional tool that is not enabled by default and does not use the Windows event logging interface is a forensic utility known as PortReporter (Availability and Description of the Port Reporter tool, n.d.). This utility creates three separate text files containing information regarding activity at the network interface. PortReporter can be used to track application connections to network ports. It details process ID, timestamp, port, local IP, state of the process (LISTENING, TIME WAIT, ESTABLISHED) for every running process connected to a port.

The existence of adhoc tools like PortReporter is evidence of the fact that operating system engineers are capable of providing additional logging that may serve forensic interests. The fact that such a tool is not integrated into the existing Windows logging mechanism and is not enabled by default reinforces the argument that forensic goals are not being considered in addition to security concerns.
Complete Coverage of File Activity
The problem of identifying real-world content of files cannot be directly addressed through event logging. This is because event logging looks at system activity not information content. Therefore, there is no logic in applying ‘complete coverage of system activity’ in this instance. Of course, it can be argued that information about the content of documents can be preserved from parts of memory or slackspace. However, the preservation of evidence in slackspace does not occur systematically, but is largely influenced by the circumstances in the live system prior to the system being frozen and the low-level construction of the OS itself, especially the filesystem. Therefore, the outcome of this method is not predictable as compared to systematic activity logging.

Therefore, the logic underlying ‘Complete coverage of system activity’ (criterion 6.3.7) is useful in identifying potential system activity that may be useful in connecting log evidence to real-world actors. Therefore, the sub-section number for this section (8.1.8) appears in the Fundamental Assessment column ‘Connecting Digital Actors to Real-World Actors’ of Table 8-0a for the aforementioned criterion (6.3.7).

8.1.9 Consistency of Evidence of Actor Identification
Windows provides evidence upon which consistency checks can be applied in all three cases of actor identification. For example, in the case of user actor identification, consistency checks can be applied on logon/logoff, process creation/termination entries and object-access entries. In this case, if a process creation or termination entry occurs after a logoff entry, then there is a clear inconsistency in the integrity of evidence of user activity. Since object-accesses tend to occur more frequently than process creation/termination events, they are even more likely to be a useful indication of inconsistency. In a similar fashion to the previous example, object-access events occurring after a logoff entry signals inconsistency in the integrity of evidence.
### Applying Evidential Weighting Criterion 6.2.5

<table>
<thead>
<tr>
<th>Test</th>
<th>Criteria Applied</th>
<th>Conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Connect Actors in Log Evidence to Real-World Actors?</strong></td>
<td><strong>Apply ‘Consistency of Log Evidence Across Multiple Log Entries’ (6.2.5) to user, application and file identification</strong></td>
<td>Consistency checks can be applied on logon, process and object access entries for user identification</td>
</tr>
<tr>
<td></td>
<td><strong>Consistency checks can be applied on process and object-access entries for application identification</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Consistency checks can be applied on object-access entries for file identification</strong></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 8.9: Consistency of Evidence of Actor Identification**

This section applies ‘Consistency of log evidence across multiple log entries’ (criterion 6.2.5) to the investigation of user, application and file actor identification in the Windows environment. In chapter 6, this criterion pointed out that the logging of multiple events (redundancy) allows for inaccuracies to be identified through information consistency checks.

User actor identification relies on logon/logoff, process creation/termination and object-access entries. To check the integrity of the evidence of user actor identification, these three types of entries featured in user profiles can be examined.

According to EventReporter (an on-line knowledge base on event reporting), Windows frequently does not report logoffs when the system is shutting down (Event ID 538 Explained, n.d.). This is due to a phenomenon known as ‘token leak’. In Windows, logon sessions are uniquely linked to a token. During the logon session, a number of applications and system components will request access to this token which will result in a reference count being incremented accordingly. As long as the reference counter is not zero, the system will assume that some application or system component is active and therefore the token will not be destroyed and the logon session will not be terminated. Token leaks occur when an application requests the token but then loses the handle resulting in an incorrect reference count, which ultimately prevents an event entry representing logoff from being generated.
Although token leaks may prevent logoff from being recorded, a system shutdown event can be used to mark the end of the logon session. Process creation/termination event entries provide further evidence upon which consistency checks can be applied. For example, if a process creation or termination event entry occurs after a logoff entry, then there is a clear inconsistency in the integrity of evidence of user activity. Since object-accesses tend to occur more frequently than process creation/termination events, they are even more likely to be a useful indication of inconsistency. In a similar fashion to the previous example, object-access events occurring after a logoff entry signals inconsistency in the integrity of evidence.

Application actor identification relies on evidence of process creation/termination and object interaction. Consistency checks can be applied to determine if a process termination entry occurs after a process creation entry and object-accesses related to the particular process appear between the creation and termination entries. However, it is important to remember that object-accesses may not be enabled leaving only evidence of process activity available for application identification purposes. In this case, only the evidence of a complete pair of creation/termination entries may be checked for consistency.

In terms of file actor identification, Windows NT provides the starting and ending points to a session of object accesses by logging the session request and session closing. Windows XP adds operation-based logging that produces evidence of actual accesses that must occur within an object-access session. Therefore, consistency checks can be applied to object-access sessions to determine if the sequence of operation-based accesses follow a session request and are followed by a session-close.

This discussion suggests that consistency checks can be applied on Windows log evidence used in actor profiling. The logic underlying ‘Consistency of Log Evidence Across Multiple Log Entries’ (criterion 6.2.5) is useful in determining if the evidence of actor identification is consistent. Therefore, the sub-section number for this section (8.1.9) appears in the Fundamental Assessment column ‘Connecting Digital Actors to Real-World Actors’ of Table 8-0a for the aforementioned criterion.
8.1.10 Reconstruction of Evidence of Actor Identification

Complete reconstruction of system activity is aimed at identifying crucial pieces of evidence that may assist forensic investigators to reconstruct as much of the system activity (that cannot be logged) as possible. The selection policy is one such piece of information which can assist forensic investigators to determine which events definitely did not take place and which events may or may not have taken place. This information benefits actor identification for users, applications and files.

Since Windows does not log user interface input, not much can be inferred from evidence that can be logged regarding user behavior. However, a partial reconstruction of activity taking place within an application environment may be possible if object-access was enabled for particular application libraries featuring known functionality. Also, in terms of real-world content of files, the list of application processes that have accessed a particular file may give forensic investigators insight into the structure and format of the document. More importantly, the reliability of a backed-up copy of a real-world document may depend on whether the file was at all modified, deleted or replaced. Since file activity profiles may document the history of management changes, forensic investigators may be able to conclusively demonstrate the link between a digital file and a real-world document.

Applying Evidential Weighting Criterion 6.3.6

<table>
<thead>
<tr>
<th>Test</th>
<th>Criteria Applied</th>
<th>Conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connect Actors in Log Evidence to Real-World Actors?</td>
<td>Apply ‘Complete Reconstruction of System Activity’ (6.3.6) to user, application and file</td>
<td>In general, selection policy can identify which events did not happen and which events may or may not have happened</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Activity inside an application environment may be reconstructed if object-access was enabled on application libraries</td>
</tr>
<tr>
<td></td>
<td></td>
<td>File activity profile may be used as version history when used with backups</td>
</tr>
</tbody>
</table>

Figure 8.10: Reconstruction of Evidence of Actor Identification
This section applies ‘Complete reconstruction of system activity’ (criterion 6.3.6) to the investigation of user, application and file actor identification in the Windows environment. Previous discussion regarding actor identification in individual and multiple log entries was aimed at identifying all possible evidence logged by Windows that may be useful. ‘Complete coverage of system activity’ was aimed at identifying evidence that was not available for logging but available in the OS environment. ‘Complete reconstruction of system activity’ is aimed at identifying crucial pieces of logged evidence that may assist forensic investigators to deduce what system activity may have occurred.

Previous discussion has established that evidence of user logon only indicates that a real-world user was present at the console at the time of logon. The timing of logon may be useful given external evidence that may assist in making a positive identification. From a reconstruction perspective, it is important for forensic investigators to determine if a user logon may have taken place but was not logged. This may be the case if the system administrator did not select logon/logoff auditing when the act of logging on took place.

In general, an important piece of evidence for reconstruction is the Windows selection policy. Two things can be deduced from the selection policy. Firstly, those events that were not logged and were selected in the policy definitely did not take place. Secondly, those events that were not selected may or may not have taken place. Therefore, in the latter case, no conclusions can be derived from the absence of evidence. Fortunately, Windows logs the complete set of high-level audit policy settings every time they are changed (the event log entry does not record the changes, but rather the complete audit settings). Note that the audit policy preserved does not include audit settings for individual object-type entities (e.g., settings for individual files and registry keys). Therefore, the log must be sufficiently complete such that the last change to the selection policy was logged. Perhaps a better alternative, from a forensic perspective, would have been for Windows to log the selection policy on startup.
The link between log evidence and an application service is based on the process creation/exit entries and object-access patterns as discussed in ‘profiling application activity for improved actor identification’ (section 8.1.5). For reconstruction purposes, the selection policy is crucial, as it will convey when the log began to track process creation/exit events that may have taken place in the digital environment (since this policy is either on or off, the first log entry of this type will indicate the policy has been activated). Note that in Windows 2000, Microsoft set all audit policies by default to ‘no auditing’ (Microsoft Windows 2000 Security Configuration Guide, n.d.).

A partial reconstruction of activity taking place within an application environment may be possible if particular application libraries featuring known functions have object-access enabled. Since Windows implements a fine granularity selection mechanism that allows audit policies to differ from one object to another, there is no way of determining which objects were tagged for logging and according to what policy. However, if all object accesses could be enabled, then object-access patterns could be established and from this information it may be possible to determine what functions were being exercised by processes active inside an application environment. For example, ‘msafd.dll’ is a winsock wrapper that is called upon for network access events. By logging the execution of this library, evidence of an application’s intention to access the network can be preserved.
Reconstructing the link between a file and its content is difficult. Windows logs all file access activity anyway and this does not give any indication of content. However, two pieces of information in Windows logs may be indirectly useful. Firstly, the list of application processes that have accessed the file may give forensic investigators insight into the structure and format of the document (as pointed out in section 8.1.6). Note that if the file was an executable, then activity profiles become even more important as they track the function of the file in the digital environment. Secondly, and more importantly, the reliability of a backed-up copy of the real-world document may depend on whether the file was at all modified, deleted or replaced. Since file activity profiles may document the history of management changes. Forensic investigators may be able to conclusively demonstrate the link between the digital file and the real-world document. This history allows forensic investigators to determine what the content of the file was at times in the past. From this information, an historical record of the file content can be constructed from the backup archive (of course this also depends on the frequency and completeness of the backup archive).

Therefore, the logic underlying ‘Complete Reconstruction of System Activity’ (criterion 6.4.6) is useful in identifying key issues related to logging selection policy and object activity towards reconstructing as much of actor identification as possible. Therefore, the sub-section number for this section (8.1.9) appears in the Fundamental Assessment column ‘Connecting Digital Actors to Real-World Actors’ of Table 8-0a for the aforementioned criterion.

8.1.11 Control Over the Generation of Evidence of Actor Identification

A key factor related to an operating system’s capability to profile actors is the influence of the selection interface. The flexibility with which a selection interface allows events to be tagged for logging has a direct impact on the volume of evidence generated that ultimately affects system performance. The outcome is that logging may become entirely infeasible, limited to short intervals, or have no impact on system performance (section 6.5.1).

Windows negotiates this tradeoff carefully by employing coarse-grained selection for low-frequency events and fine-grained selection for events that may reach higher frequencies. Although the volume of evidence remains under control when each is employed separately, performance penalties are likely to be high when they are used collectively to generate actor identification profiles.
Application profiling requires all log entries related to the creation/exit of the particular application and all related object accesses to be logged. Since one process cannot be logged to the exclusion of another, and object-access logging can only be restricted on a user basis, every object-access made by every user must be logged in order to construct a single application’s profile. This setting is likely to have an extremely negative effect on system performance.

Applying Evidential Weighting Criterion 6.4.1

<table>
<thead>
<tr>
<th>Test</th>
<th>Criteria Applied</th>
<th>Conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connect Actors in Log Evidence to Real-World Actors?</td>
<td>Apply ‘Control over Evidence Generation’ (6.4.1) to user, application and file identification</td>
<td>Although a single user logon/logoff cannot be singled out for logging, system performance is unlikely to be affected.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Constructing a single application’s profile requires all accesses to all objects by all users to be logged which is likely to affect.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Comprehensive profiling of a single file’s activity can be achieved with minimal affect to system performance.</td>
</tr>
</tbody>
</table>

**Figure 8.11: Control Over the Generation of Evidence of Actor Identification**

This section investigates Windows’ ability to generate evidence for actor identification purposes given system performance limitations. Therefore, Windows’ ability to generate the actor profiles of users, applications and files will be assessed given the influence of the Windows selection interface. Previous sections identified Windows log evidence useful for identification of users, applications and documents. The three kinds of log entries discussed are listed below:

<table>
<thead>
<tr>
<th></th>
<th>Users</th>
<th>Applications</th>
<th>Documents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows’ Evidence</td>
<td>Logon/Logoff, process creation/termination and object-access</td>
<td>Process creation / termination and object-access</td>
<td>Object access and Process creation / termination</td>
</tr>
</tbody>
</table>

**Table 8-9: Evidence Collection Requirements for Profiling**
A key factor related to an operating system’s capability to collect evidence is the influence of the selection interface. The discussion of this criterion in chapter 6 states that the flexibility provided by the selection interface is important, as there is a tradeoff between coarse-grained and fine-grained control over evidence generation. As the criterion points out, coarse-grained control over high-frequency events typically results in a significant performance penalty. However, using fine-grained control requires choices to be made on how to generate specific evidence.

In terms of evidence of user identification, the Windows logon/logoff process uses a coarse-grained selection control. The only choice available in the selection interface is to generate evidence of logon sequences or not. System performance is not significantly impacted by this selection method as the piece of code executed for logon/logoff is infrequently executed. When it is executed, it generates only one log entry on login and one on logoff. Since forensic evidence collection typically incorporates accountability for users, logon/logoff auditing should be turned on by default.

A system administrator may require a comprehensive activity profile of an untrusted application to determine the kinds of activity it is conducting in the digital environment, whereas, for a trusted application, the administrator may only want to know if the executable or the libraries have changed. Therefore, there may be a need to log more information about the activity of one application than another. Application profiling requires all log entries related to the creation/exit of the particular application and all related object accesses to be logged. Unfortunately, according to the way the Windows architecture is structured, it is not possible to design a different profile of activity for one application compared to another application.

There are two reasons for this. Firstly, the selection control on process creation is coarse-grained. Therefore, either all process creation/exit is logged or none. This results in a log entry generated every time any application starts or exits.
It is relevant to note that the evidential weighting criterion on high-frequency evidence states that in most cases, a coarse-grained approach to process logging typically results in a significant performance penalty. This is because some operating systems might adopt a model where application processes spawn temporary processes to allow multiple tasks to be conducted in parallel. This scenario would result in lots of short-lived (temporary) processes starting up and exiting consequently generating a large number of log entries.

Logging all process startup/shutdown under Windows does not (typically) lead to the system grinding to a halt because of the way Windows structures its generic ‘process’. The term ‘process’ in the context of Windows is simply a set of resources reserved for one or more agents of execution known as ‘threads’. This is a simple way of gathering all of the agents executing different streams of instructions on behalf of a single application under one conceptual entity. Therefore, relating log entries to processes is simplified by Windows as it logs the process as the agent of execution rather than individual threads.

It is also not possible to create a complete activity profile of one application to the exclusion of other applications because selection control on object-access logging is user-centric. This means that evidence of object-access is only generated after a series of conditions are met. In
Windows, object-access entries are generated based on which user is logged in, rather than which application is requesting access. The unfortunate result of the combination of the process and object selection models is that object-access must be set to ‘everyone’ to capture every user’s use of one application. Therefore, since one process cannot be logged to the exclusion of another, every object-access made by every application process must be logged (regardless of which user is logged on). This setting will likely have an extremely negative effect on system performance. The same is not true for profiling all accesses to a single file. The fundamental difference is that the high volume of events in the previous scenario is caused by the large number of objects being accessed by a few processes. These processes are constantly requiring access to object resources, hence the high frequency of events. Logging all accesses to a single file by a small number of processes is much less likely to impact system performance.

Therefore, the logic underlying ‘Control over generation of evidence (criterion 6.4.1) is useful in assessing the flexibility with which a selection interface allows evidence to be generated and its impact on system performance. Therefore, the sub-section number for this section (8.1.11) appears in the Fundamental Assessment column ‘Connecting Digital Actors to Real-World Actors’ of Table 8-0a for the aforementioned criterion.

8.1.12 Guidance on Selection of Evidence of Actor Identification
The Windows’ selection interface is modeled on the operating system view of system activity as it only provides a direct interface to the three types of activity logging offered by the operating system, namely, logon/logoff, process tracking and object-access.

For the purposes of guiding selection for user identification, the logging interface must indicate that the minimum is to activate the logon/logoff and process tracking options. Since object-access logging utilizes fine granularity selection, there must be some guidance on which objects must be tagged for user activity profiling, especially given greater logging may be required for high-privilege and un-trusted accounts. For application identification purposes, guidance on object-access selection is required since process tracking alone is not sufficient.

File activity profiling is relatively simple, as logging all accesses by any process is simple to activate given Windows features an interface with fine granularity.
Figure 8.12: Guidance on Selection of Evidence of Actor Identification

The discussion of this criterion in chapter 6 suggested that, in general, event selection interfaces do not efficiently map real world events to event logging configurations. Instead, selection interfaces tend to reflect system architecture. Therefore, administrators must identify which events must be tagged to generate sufficient evidence to meet real-world goals. Often, this will involve creating a mental model of the real-world activity and breaking it down into its individual system actions that can be logged separately. Therefore, generating evidence of real-world events (such as the usage of a particular application service) is difficult, as it requires precise actions, subjects, objects and so forth to be identified before any evidence can be generated. Windows’ selection interface mirrors its own architecture, in that the event interface is not defined in terms of real-world events but Windows events (see snapshots and description in section 4.1.6).

Although an obvious starting point towards identifying a user is ensuring logon/logoff is recorded, the selection interface should indicate that logon/logoff does not clearly identify users given the standard password exchange. More reliable forms of authentication can be useful in this regard, especially if operating systems require that they be used repeatedly during a single logon session. Given the limitations of user identification through
logon/logoff, the logging interface must also recommend the use of actor profiling for identification purposes. For this case, process tracking must be activated as a minimum, in addition to logon/logoff, to capture evidence of user logon and application activity. Unlike with logon/logoff and process tracking, where an ‘all-or-nothing’ form of coarse granularity is implemented, object-access can be controlled with fine-granularity. Windows does not provide guidance on object-access logging in general. Therefore, there needs to be more guidance on which objects must be tagged for user activity profiling. Since object-access logging cannot be logged for all users, some distinction must be made between those user accounts that must attract greater logging (e.g. high-privilege accounts and un-trusted accounts) and those that do not. For the former case, object-access logging may have to be used to create a comprehensive activity profile. For example, files associated with applications (e.g. dlls and data files) may be useful in determining if an application has changed or its pattern of activity has changed thereby influencing the consistency of the evidence of user actor identification.

Application identification begins with enabling process tracking as the image filename can be related to the live process running in the digital environment. As is the case with user actor identification, the filename alone is not sufficient to identify an application. Section 8.1.5 indicated that an application activity profile benefits from recording the pattern of object-accesses made by an application as well as identifying changes in an application’s libraries. Guidance must be provided to administrators so they may tag objects for logging.

File activity profiling requires all accesses involving a specific file to be logged. Section 8.1.6 showed this was useful as the identity of processes interested in the object may be established. These processes may provide useful clues as to what the format and structure of the file may be. In this case, guidance must be provided to assist systems administrators to identify which object-accesses must be tagged for profiling the activity of particular files.

Therefore, the logic underlying ‘Guidance on Evidence Selection’ (criterion 6.4.2) is useful in assessing the guidance provided by selection interfaces and its impact on evidence generation. Therefore, the sub-section number for this section (8.1.12) appears in the Fundamental Assessment column ‘Connecting Digital Actors to Real-World Actors’ of Table 8-0a for the aforementioned criterion (6.4.2).
8.2 Connecting System Activity to Real-World Acts

Logon, process-tracking and object-access are three activities for which Windows allows evidence preservation. These activities are the focus of the early part of this section where activity-specific weighting criteria are used to evaluate Windows. Evidential weighting criterion ‘accuracy of action identification in individual log entries’ (criterion 6.2.3) assesses Windows’ ability to accurately identify events in log entries.

Applying all weighting criteria associated with the identification of all actions requires extensive and laborious investigations into the internals of logon, process activity and object activity. In section 8.2.1, all three are considered from a fundamental ‘action identification’ perspective. However, when applying more in-depth criteria such as ‘Uniquely identifying a system operation’ (6.3.2) and ‘Identifying key activities in a system operation’ (6.3.3), only object-access is chosen. This is because object-access provides the most interesting discussion regarding evidential weighting.

General weighting criteria are applied to further investigate Windows’ activity logging capabilities. Forensic investigators may not be able to relate real-world activity to log evidence without a means of relating log entries to construct a ‘big picture’. The criterion ‘Grouping related log entries to identify system operations’ (criterion 6.3.1) assesses the ability of Windows to link log entries together at varying levels of abstraction such as process-level and thread-level activity. Complete coverage of system activity (criterion 6.3.7) will be applied to determine the potential range of system activity that can be logged in Windows such as events taking place during boot-up prior to the startup of the event logging service. ‘Consistency of log evidence across multiple log entries’ (criterion 6.3.5) will be applied to determine the potential impact of inaccuracies in the logging of system activity. For example, the presence of object-access entries may reduce the risk of misplacing a process activity entry. ‘Complete reconstruction of system activity’ (criterion 6.3.6) will be used to determine how much evidence of system activity, that is not available for logging, can be determined from evidence that is available. An example of this case is the Windows selection policy that specifies the configuration of event logging under which evidence collection is taking place.
Chapter 8: Applying the Evidential Weighting Framework to Microsoft-Windows Logging

Windows’ ability to generate evidence of system activity will be assessed using evidential weighting criteria developed for selection interfaces. ‘Control over evidence generation’ (criterion 6.4.1) will be used to assess Windows’ flexibility in generating evidence of one activity to the exclusion of another. ‘Guidance on evidence selection’ (criterion 6.4.2) will be used to assess the usefulness of Windows’ advice on the generation of evidence of system activity given the selection interface.

### 8.2.1 Accuracy of Action Identification in Individual Log Entries

Windows features three categories of actions, namely, logon, process tracking and object-access. Of the three categories, object-access turns out to be the most misleading since an object-access event is generated upon request for access as opposed to the access itself. The action identification information associated with the remaining logon and process tracking events are less likely to be easily misinterpreted.

Applying Evidential Weighting Criteria 6.2.3

<table>
<thead>
<tr>
<th>Test</th>
<th>Criteria Applied</th>
<th>Conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connecting Log Evidence to Email and Web-browsing Activity</td>
<td>Apply ‘Accuracy of Action Identification in Individual Log Entries’ (Criterion 6.2.3) to logon/logoff</td>
<td>Most logon actions are correctly identified</td>
</tr>
<tr>
<td></td>
<td>Apply ‘Accuracy of Action Identification in Individual Log Entries’ (Criterion 6.2.3) to process tracking</td>
<td>Process activity entries correctly identify creation and termination</td>
</tr>
<tr>
<td></td>
<td>Apply ‘Accuracy of Action Identification in Individual Log Entries’ (Criterion 6.2.3) to object-access</td>
<td>Object-access action identification is misleading</td>
</tr>
</tbody>
</table>

**Figure 8.13: Accuracy of Action Identification in Individual Log Entries**

The discussion of criterion ‘Accuracy of action identification in individual log entries’ (6.2.3) in chapter 6 pointed out that the description of an event recorded in a log entry may not be correct or comprehensive. In the case of logon/logoff (see sample log entries below), Windows logs a separate event for each action-type (i.e. a separate log entry for a logon and another one for logoff). The logon entry clearly identifies the action as being successful. Note
that the Logon ID is used to match the two log entries to identify evidence of a completed logon session.

<table>
<thead>
<tr>
<th>Event ID</th>
<th>Type</th>
<th>Description</th>
<th>Time</th>
<th>Computer</th>
<th>User Name</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Logon ID: (0x0,0xa7c60)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

|          |            | Logon ID: (0x0,0xa7c60)   |                 |             |           |

**Table 8-10: Logon/Logoff Entries**

Further, Windows provides a series of additional event ids for log entries that are not successful (see below). Most of the actions that lead to the logging of such events are unambiguously identified with the exception of event id 537.

<table>
<thead>
<tr>
<th>Event ID</th>
<th>Logon Failure Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>529</td>
<td>Unknown user name or bad password</td>
</tr>
<tr>
<td>530</td>
<td>Account logon time restriction violation</td>
</tr>
<tr>
<td>531</td>
<td>Account currently disabled</td>
</tr>
<tr>
<td>532</td>
<td>The specified user account has expired</td>
</tr>
<tr>
<td>533</td>
<td>User not allowed to logon at this computer</td>
</tr>
<tr>
<td>534</td>
<td>The user has not been granted the requested logon type at this machine</td>
</tr>
<tr>
<td>535</td>
<td>The specified account's password has expired</td>
</tr>
<tr>
<td>537</td>
<td>An unexpected error occurred during logon</td>
</tr>
</tbody>
</table>

**Table 8-11: Logon Failure Reasons**

Regarding process activity tracking, log entries identify process creation and termination events clearly in the description fields (see table 8-3). Note that there is no process failure event id, which raises the question as to what happens if any instruction in the complex startup process fails to complete (see section 4.2.1 for a description of Windows process startup).
Chapter 8: Applying the Evidential Weighting Framework to Microsoft-Windows Logging

However, as previously described in the discussion of file activity profiling (section 8.1.6), object-access action identification is misleading. The log entry identifies the action as ‘object open’ which is an ambiguous description frequently misinterpreted to imply the object in question was accessed. Essentially, in the case of Windows NT, an object-access event is generated upon request for access as opposed to the access itself. It may be argued that an application process requesting an object will only logically proceed to perform the intended activities with the object. This may be true in the case of well-behaved applications. However, from a forensic perspective, there is no evidence of the application process interacting with the object directly.

Microsoft recognized some of these shortcomings - see quote from Microsoft developer in charge of Event Logging Project below (Windows Security Logging and Other Esoterica, n.d.). Microsoft introduced a new feature in Windows 2003 known as ‘operation-based logging’ (see Table 8-6). The log entry identifies this action as ‘object access attempt’. This feature allows administrators to generate a separate log entry documenting the first time a process exercises a permitted action on an object. Subsequent usage of the same permission is not loggable.

“A lot of people are unhappy with object access auditing on Windows, because what they want to know is "who touched the object and what did that person do", but what Windows auditing tells you is actually "who touched the object and what did they ask for permission to do". The distinction is subtle, but if you are interpreting object access events as recording what changes were made to objects, then you're probably misunderstanding what the log is saying.”

Note that the Microsoft developer believes the object-access entry does constitute evidence of a process actually touching an object. Although the process requesting access is likely to touch the object immediately after permission is given, the log entry is not generated as a result of the ‘touching’ but rather the security check. Despite this observation, Windows 2003 is a significant improvement from a forensic point of view because it allows accurate identification of some important actions to be documented as opposed to just permission to exercise an action.
However, there is still not enough information logged about system operations to determine the full extent of process activity. For example, if a process executes more than one read on a file or more than one write on a file, this cannot be identified from the log. Therefore, a process reading a complete file cannot be distinguished from a process reading half a file. The same is true for the exercising of other object-access rights.

The logic underlying ‘Accuracy of Action Identification in Individual Log Entries (criterion 6.2.3), is useful in determining the link between an action described in a log entry and the real-world activity such as logon/logoff, process creation/termination and object-access. Therefore, the sub-section number for this section (8.2.1) appears in the Fundamental Assessment column ‘Connecting System Activity to Real-World Acts’ of Table 8-0a for the aforementioned criterion (6.2.3).

8.2.2 Accuracy of System Operation Logging

Windows NT does not log two important activities in an object-access, these are the call executed by a process when intending to access an object and evidence of the access itself. Windows XP has improved in this situation in that it logs many, but not all, actual accesses to the object. A system operation consisting of multiple reads executed by the same process cannot be distinguished from another system operation executing only a single read as there is not enough information in the log entries recorded. In general, wherever an object-access right is exercised more than once in an object-access session, the ability to uniquely identify the system operation is lost.
Applying Evidential Weighting Criteria 6.3.2 and 6.3.3

<table>
<thead>
<tr>
<th>Test</th>
<th>Criteria Applied</th>
<th>Conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connect Action in Log Evidence to Real-World Actions?</td>
<td>Apply ‘Identifying Key Activities in a system operation’ (6.3.3) to object-access</td>
<td>Windows does not log the calling event and in many cases the actual access to an object</td>
</tr>
<tr>
<td></td>
<td>Apply ‘Uniquely identifying a system operation’ (6.3.2) to object-access</td>
<td>Object-access operations exercising a single right multiple times cannot be distinguished from each other</td>
</tr>
</tbody>
</table>

**Figure 8.14: Accuracy of Object-Access Activity Logging**

To further test the accuracy of object-access activity logging, two further criteria can be applied. In chapter 6, ‘Identifying key activities in a system operation’ (criterion 6.3.3) suggested that each key activity in a system operation must generate a separate log entry for evidence of the system operation to be complete. Whereas, ‘Uniquely identifying a system operation’ (criterion 6.3.2) pointed out that there may be more than one explanation for the existence of a particular log entry or pattern of log entries.

The criterion ‘Identifying key activities in a system operation’ (6.3.3) is useful in determining if each key activity in the reading of the file can be deduced from the log evidence. Consider the following diagram:

**Figure 8.14b: Object Access Operation**
In the above figure, process P1 (e.g. a word processing service) requests access to object O1 (e.g. a document). Windows requires that P1 specify precisely what types of actions it intends to execute on object O1. The Windows Object Manager (not shown in the diagram) calls the Security Reference Monitor (SRM) and sends it the set of desired rights. The SRM runs a series of security checks that take into account the security identity of the process (taken from the user logged in), the desired access, and the object’s security settings. The outcome is that access is either granted or it is not. If access is granted then the SRM returns a set of rights that the process is allowed to exercise. The Object Manager stores these rights into an object handle and returns this to the process. The key entity in this figure is the LSASS or Local Security Authority Subsystem. It enforces the security policy and sends security audit messages to the event log. More importantly, the LSASS takes a partially formed record with the information required by the SRM and complements it with other such as process identification information. These entries are sent to the event log via the Event Logger.

The evidential-weighting criterion ‘Identifying key activities in a system operation’ (6.4.3) suggests that each key activity in a system operation must generate a separate log entry for evidence of the system operation to be complete. Note in this case the system operation is ‘process P1 accesses object O1’.

From a forensic perspective, there are three key activities in this transaction are:

1. The calling process requesting object-access
2. The security check on the transaction requested by the calling process and the outcome of the request (approved/rejected – in Windows terms success/failure)
3. The specific action exercised on the object itself and the outcome of the action (success / failure)

Therefore, the following information would make evidence complete (relevant information at each event is in parentheses):

Log Entry 1: (process id of caller P1, time the call was made, target object O1, ACL)
Log Entry 2: (process id of caller P1, time the request was processed, target object O1, success/failure)
Log Entry 3: (object id O1, time the action was executed, action executed)
However, when comparing the proposed set of three log entries with object logging provisions in NT and XP, the following observations can be made:

Firstly, The beginning of the system operation is marked by the calling process P1 requesting permission to access the target object O1. This system event (Log Entry 1) is not available for logging in Windows. Secondly, Windows logs the second system event (Log Entry 2) in the form of a pair of object-access entries that represent the application process requesting permission to commence a session within which the data file would be accessed. The first entry is generated when the process requesting access to the object is processed by the security enforcement part of the operating system. The second entry is generated when the object-access is closed (sample entries in Table 8-4). Note that although the calling process identifier was supplied by the LSASS, the process call system event (Log Entry 1) is not logged explicitly. The forensic investigator will have an identifier based on the result of a process lookup. As Windows only logs the second system event, the time when the process call was made is not available. Thirdly, and most importantly, Windows NT does not log the system event where the action was exercised on the object (Log Entry 3) although the permissions requested by the process is recorded. Therefore there is no evidence of any action being exercised on the object. As previously mentioned, Windows XP logs an event the first time a requested right is exercised. Subsequent exercising of the same right is not logged. Therefore, an event is logged only the first time a process reads a file, writes to a file, and so forth.

Given the above discussion, applying ‘Uniquely identifying a system operation’ (criterion 6.3.2) is straight-forward. Even where operation-based logging is implemented (eg Windows XP), as mentioned at the conclusion of the previous section, a system operation consisting of multiple reads executed by the same process cannot be distinguished from another system operation executing only a single read as there is not enough information in the log entries recorded. In general, wherever an object-access right is exercised more than once in an object-access session, the ability to uniquely identify the system operation is lost. However, a relevant aside to this discussion is that the evidence produced from an object-access session is influenced by an application’s pattern of object-access activity. To explain, some background on handles is necessary. Essentially, Windows grants access to an object using handles. A handle is an index into a table where the necessary information resides to facilitate object-access activity. Once the application closes off the object-access session, the handle is
closed. Some applications open and close handles very often, whereas others keep the handle open for longer periods of time. If an application exercises its object-access rights only once before requesting a new handle then there is a higher likelihood that object-access system operations can be uniquely identified.

Therefore, ‘Uniquely Identifying a System Operation’ (criterion 6.3.2) and ‘Identifying Key Activities in a System Operation’ (criterion 6.3.3) are useful in investigating the link between Windows log evidence and real-world activities such as object-access activity. Therefore, the sub-section number for this section (8.2.2) appears in the Fundamental Assessment column ‘Connecting System Activity to Real-World Acts’ of Table 8-0a for the aforementioned criterion (6.3.2, 6.3.3).

8.2.3 Grouping Related Log Entries to Identify System Operations
Windows provides an effective mechanism through the use of link identifiers. Using this method, log entries referring to the same system operation can be related. Windows assigns unique identifiers to processes and objects and inserts these identifiers in log entries. All object accesses can be related to corresponding processes and all processes can be traced to their parent processes. Therefore, process flow-trees can be easily constructed and all requests to access specific objects can be correspondingly linked.

However, processes frequently interleave between tasks such as user input, writing to temporary files, and connecting to the network stack. These actions are carried out by threads that may generate multiple object-access entries. Unfortunately, these entries cannot be grouped together, based on whether they were conducted in parallel or sequence to each other, as there are no thread identifiers.
Applying Evidential Weighting Criterion 6.3.2

<table>
<thead>
<tr>
<th>Test</th>
<th>Criteria Applied</th>
<th>Conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connect System Activity to Real-World Acts?</td>
<td>Apply ‘Grouping Related Log Entries’ (6.3.2)</td>
<td>All log entries can be related to system operations by process identifiers. Logon, process and object access sessions can be easily related using special link identifiers. However, entries related to a single process cannot be grouped based on which thread they belonged to.</td>
</tr>
</tbody>
</table>

**Figure 8.15: Grouping Related Log Entries to Identify System Operations**

Log evidence of real-world activities revolve around two kinds of entries. These are application process creation/termination and file access. Discussion of ‘Grouping related log entries to identify system operations’ (criterion 6.4.1), identified three distinct issues in operating systems that might influence grouping capabilities. These are the ability to separate evidence of routine operating system management from user activity, the ability to group evidence of system activity in sessions and linking evidence of high-frequency events (common to many system operations) to the particular system operation that generated it.

The underlying concern fueling the discussion of the above criterion was the possibility that log entries generated from the same process’ activities could not be grouped according to an investigator’s needs. This was possible if some operating systems adopted a model where application processes spawn temporary (other) processes to allow multiple tasks to be conducted in parallel. This scenario would result in lots of log entries generated from the same activity that may not be grouped together because of the lack of a parent process identifier.

The term ‘process’, in the context of Windows, is simply a set of resources reserved for one or more agents of execution known as ‘threads’. This is a simple way of gathering all of the agents executing different streams of instructions on behalf of a single application under one conceptual entity. Therefore, on the face of it, Windows simplifies relating log entries to processes as it logs the process as the agent of execution rather than individual threads. However, real world activities involving email and web browsing applications feature a series of actions in parallel. For example, user input, writing to temporary files, and connecting to
the network stack may be multitasked. To support this functionality, Windows uses threads. Each thread may generate object-accesses along the way, therefore, a significant insight can be gained into the inner workings of an application with multiple threads if object-accesses generated by the same thread could be linked by some kind of thread identifier (i.e. grouping based on threads). Unfortunately, Windows hides this kind of information so object-access entries can only be linked at a process-level rather than a thread-level. Forensic investigators cannot determine which object accesses may have occurred in parallel and which ones were generated in strict sequence. This does not allow connections to be made successfully between real-world activity and log evidence. In this sense, operating systems that generate new processes rather than threads have an advantage from a forensic perspective.

The third issue is therefore simple to resolve in Windows. Since all log entries related to the same system operation must have the same process identifier, all log entries can be linked to their respective system operations. Regarding the first issue, evidence of routine management is not straightforward even though ‘user’ is marked ‘N/A’ or ‘System’ in certain cases. In principle, any entries generated by the activity of a process descended from the user logon process will be marked by the user’s name. As section 8.1.1 demonstrated, user-initiated activity cannot be distinguished from other activity in the process subsystem. The second issue regarding grouping system activity into sessions was discussed in section 8.1.7.

Therefore, the logic underlying ‘Grouping related log entries (criterion 6.3.1) is useful in determining the extent to which log evidence can be grouped so that evidence of system activity can be connected to real-world activity. Therefore, the sub-section number for this section (8.2.3) appears in the Fundamental Assessment column ‘Connecting System Activity to Real-World Acts’ of Table 8-0a for the aforementioned criterion.

8.2.4 Complete Coverage of System Activity

There are a number of areas of system activity that are not available for logging in the Windows environment. System activity during boot-up prior to the loading of the Service Control Manager that is responsible for starting the Event Logging Service is not preserved. Although Windows can detect the introduction of a new file resulting from an I/O operation, evidence may only be preserved if the directory and its child objects were previously selected for logging purposes. Further, key activities related to web browsing, email usage and
document management such as network access, application environment events and user input is not available for logging either.

**Applying Evidential Weighting Criterion 6.4.7**

<table>
<thead>
<tr>
<th>Test</th>
<th>Criteria Applied</th>
<th>Conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connect Log Evidence to Real-World Activity?</td>
<td>Windows system activity prior to starting the Service Control Manager is not logged</td>
<td></td>
</tr>
<tr>
<td>Apply ‘Complete Coverage of System Activity’ (6.4.7) to system</td>
<td>Windows can detect the introduction of a new file only if the file is created in a directory that has object-access logging enabled beforehand</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Windows does not make network access, application environment and user interaction events available for logging</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 8.16: Complete Coverage of System Activity**

This section applies the weighting criterion ‘Complete coverage of system activity’ (6.4.7) to Windows log evidence. Discussion of this criterion identifies five different kinds of system activity that are typically unavailable for logging. These are activity during bootup and shutdown, I/O, network connectivity, application environment and raw user interaction. Each of these areas is considered separately.

The Windows boot up process (described in section 4.3.1) tests the hardware, then initializes the boot screen and then compares a hardware inventory with the registry. More software is then loaded which enforces the separation between kernel and hardware after which essential components of Windows such as the Memory Manager and the I/O Manager, are loaded. The graphics subsystem is prepared for use, the computer is switched to graphics mode and then Winlogon and LSASS are initialized. Winlogon then starts up the Service Control Manager (SCM) which in turn starts the event logging service. Note that none of these actions are logged by Windows at the startup of the event logging service.

Regarding logging input/output from a computing system, Windows can detect the introduction of a new file only if the file is created in a directory that has object-access
logging enabled beforehand. The log entries below show the series of object-access operations involved in the creation of a file. However, even though the creation of a file is logged, there is no indication as to which input device was the source of the file.

<table>
<thead>
<tr>
<th>Event ID</th>
<th>Type</th>
<th>Description</th>
<th>Time</th>
<th>Computer</th>
<th>User Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>560</td>
<td>Success</td>
<td>Object Open</td>
<td>28/12/2006 3:19:05 PM</td>
<td>DIS-WS0002062</td>
<td>adam</td>
</tr>
</tbody>
</table>

Object Open:
Object Server: Security
Object Type: File
Object Name: C:\Documents and Settings\adam\My Documents\Track01.cda
Handle ID: 1612
Operation ID: {0,366986}
Process ID: 432
Accesses: DELETE
READ_CONTROL
SYNCHRONIZE
WriteData (or AddFile)
AppendData (or AddSubdirectory or CreatePipeInstance)
WriteEA
ReadAttributes
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Success Object Access Attempt</td>
<td>ReadAttributes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>28/12/2006 3:19:05 PM</td>
<td>DIS-WS0002062</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>28/12/2006 3:19:05 PM</td>
<td>DIS-WS0002062</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>28/12/2006 3:19:05 PM</td>
<td>DIS-WS0002062</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Success Object Access Attempt</td>
<td>WriteData (or AddFile)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>28/12/2006 3:19:05 PM</td>
<td>DIS-WS0002062</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Success Object Access Attempt</td>
<td>WriteAttributes</td>
</tr>
</tbody>
</table>
Table 8-12: Input/Output Log Entries

Previous discussion on the complete coverage of system activity in section 8.1.7 points out that Windows does not make network access events, application environment events and raw user interaction events available for logging. Given the above discussion, considerable evidence of system activity pertaining to web browsing, email usage and document management such as the logging of network access events, application environment events and user interface events is not available for logging.

The logic underlying ‘Complete coverage of system activity’ (criterion 6.3.7) exercised in this section and in section 8.1.8, is useful in identifying potential system activity that may be connect log evidence to real-world activity. Therefore, the sub-section number for this section (8.2.4) appears in the Fundamental Assessment column ‘Connecting Digital Actors to Real-World Actors’ of Table 8-0a for the aforementioned criterion.
8.2.5 Consistency of Log Evidence Across Multiple Log Entries

Inconsistencies in the evidence of logon sessions can be minimized if there exists sufficient evidence of application sessions between logon/logoff events. Evidence of a steady pattern of activity within a logon session indicates the logoff entry is unlikely to be misplaced. Object-access entries can play the same role for application sessions. If an application session makes numerous and consistent file accesses, then the potential impact of an inaccurate placing of an application creation/termination entry can be reduced. Therefore, object-access evidence allows consistency checks to be applied on applications, and application sessions allow consistency checks to be applied on logon sessions. Consistency checks can be applied on object-accesses themselves as they reflect the interaction between the application and Windows, resulting in the generation of large numbers of entries.

Applying Evidential Weighting Criterion 6.2.5

<table>
<thead>
<tr>
<th>Test</th>
<th>Criteria Applied</th>
<th>Conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Consistency of logon sessions can be demonstrated given sufficient evidence of</td>
<td></td>
</tr>
<tr>
<td></td>
<td>application sessions</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Consistency of application sessions can be demonstrated given sufficient evidence</td>
<td></td>
</tr>
<tr>
<td></td>
<td>of object accesses</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Consistency checks can be applied on object-access sessions given sufficient object-</td>
<td></td>
</tr>
<tr>
<td></td>
<td>access patterns</td>
<td></td>
</tr>
</tbody>
</table>

Figure 8.17: Consistency of Log Evidence Across Multiple Log Entries

This section applies ‘Consistency of log evidence across multiple log entries’ (criterion 6.2.5) to the logon, application and object-access activities in the Windows environment. This criterion points out that the logging of multiple events (redundancy) as a result of a system operation or part of a system operation allows for inaccuracies to be identified through information consistency checks.
Regarding the consistency of all three types of sessions (logon, application and object-access), logon/logoff is reported to have significant issues in the Windows environment. In particular, Windows frequently does not record logoffs when the system is shutting down due to token leaks (see section 8.1.9 for a discussion on token leaks). This is a significant concern from a consistency perspective. It is possible for a logoff sequence to be initiated and then to hang indefinitely thereby allowing a third party to begin issuing commands to the computing system. Since Windows does not log the start of a logoff sequence but the end, this inconsistency cannot be identified. However, inaccuracies regarding logoff due to a token leak may be reduced if process tracking is enabled and evidence of multiple application sessions is preserved in the Windows log. The presence of log entries describing application startup and termination linked to the logon session indicate the logon session is still active and has not been terminated. Therefore, if a logoff entry appears before a process activity entry, then this is an indication of inconsistency in the log evidence. Object-access entries can play the same role for application sessions. If an application session makes numerous and consistent file accesses, then the potential impact of an inaccurate application creation/termination entry can be reduced. Therefore, object-access evidence allows consistency checks to be applied on applications, and application sessions allow consistency checks to be applied on logon sessions.

Given this train of logic, it can be argued that applying consistency checks on object accesses is likely to be difficult as there is no fourth category of events that may drill down further during an object-access session. However, in Windows this is only partially true. From a real-world perspective, an object-access is a single action. For example, the opening of a file requires an object to be accessed. Consequently, it can be expected that a single ‘object open’ and ‘object close’ entry would be generated in a Windows log. When a process opens a single file, there may be multiple open/close entries. This is because object-access events reflect the interaction between Windows and applications rather than users and applications. Therefore, if Microsoft Word opens and closes a file multiple times while reading from it or writing to it, then each such action will generate separate object-access entries. Even single atomic actions such as listing a folder or deleting a file will also generate open and close entries. Given attempts at opening a file are likely to result in many pairs of object-access entries, impact resulting from the loss of a single pair may be reduced in this case. In general, having multiple entries for object-access requests facilitates the application of information consistency checks.
Therefore, the logic underlying ‘Consistency of Log Evidence Across Multiple Log Entries’ (criterion 6.2.5) is useful in determining if system activity can be consistently identified. Therefore, the sub-section number for this section (8.2.5) appears in the Fundamental Assessment column ‘Connecting Digital Actors to Real-World Actors’ of Table 8-0a for the aforementioned criterion (6.2.5).

8.2.6 Complete Reconstruction of System Activity

Complete reconstruction of system activity is applied to identify crucial pieces of evidence that may assist forensic investigators to construct as much of the system activity as possible. The selection policy is one such piece of information which can assist forensic investigators to determine which events definitely did not take place and which events may or may not have taken place. This information benefits activity identification in general. Boot-up activity (and shut-down activity) taking place while the event log service is not operational is difficult to reconstruct. In addition, object-access can be enabled for application and system libraries allowing activity within applications and utilization of system services to be partially reconstructed.

Applying Evidential Weighing Criterion 6.4.6

<table>
<thead>
<tr>
<th>Test</th>
<th>Criteria Applied</th>
<th>Conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connect Activity in Log Evidence to Real-World Activity?</td>
<td>Apply ‘Complete Reconstruction of System Activity’ (6.4.6) to system activity</td>
<td>In general, selection policy can identify which events did not happen and which events may or may not have happened</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Activity inside an application environment may be partially reconstructed if object-access was enabled on application libraries</td>
</tr>
<tr>
<td></td>
<td></td>
<td>There does not seem to be any means of reconstructing boot-up activity</td>
</tr>
</tbody>
</table>

**Figure 8.18: Complete Reconstruction of System Activity**

This section applies ‘Complete reconstruction of system activity (criterion 6.4.6) to the investigation of system activity in the Windows environment. Previous discussion regarding
system activity was aimed at identifying all possible evidence logged by Windows that may be useful. In particular, ‘Complete coverage of system activity’ was applied to identify evidence that was not available for logging but available in the OS environment.

Previous discussion on complete coverage of system activity established that system activity in the boot-up phase prior to the start-up of the event logging service is not logged. Unfortunately, there doesn’t appear to be a means of reconstructing any of this activity from evidence that can be logged. However, as previously discussed in section 8.1.9, preserving the selection policy is important as it can help to identify which events did not happen and which events may or may not have happened. In addition, object-access can be enabled for application and system libraries allowing activity within applications and utilization of system services to be partially reconstructed. In Windows NT, activities inside an application environment cannot be reconstructed apart from some impression of the high-level intentions of an application. These can determined when the application requests permission to execute a function that utilizes services provided by the operating system or some external library. In Windows XP, partial reconstruction may be achieved, as some evidence may exist regarding the actual utilization of external services through operation-based logging. Note that both cases assume that object-access logging is strategically applied on files that would reveal the application’s intentions. Since such object-access logging cannot be applied across all files, careful selection must be exercised (for example, object-access can be enabled for only high-capability user accounts or suspect user accounts). Without object-access, the extent of reconstruction becomes very limited.

Therefore, the logic underlying ‘Complete reconstruction of system activity’ (criterion 6.3.6) is useful in identifying evidence that may assist forensic investigators to reconstruct as much of the system activity as possible. Therefore, the sub-section number for this section (8.2.6) appears in the Fundamental Assessment column ‘Connecting Digital Actors to Real-World Actors’ of Table 8-0a for the aforementioned criterion.
8.2.7 Control over Generation of Evidence of System Activity

In general, for any forensic analysis using event logs, both logon/logoff and application process creation / termination events are a necessary minimum. However, generating a more comprehensive profile of particular activities in the computing system, such as the events related to a logon sequence or an application’s activities, require associated evidence of object-access activity.

Windows features fine granularity when it comes to singling out an object-access type or selecting all object-access types for logging. However, these can only be generated if a particular user or group directly accesses the object. Therefore, generating all object-accesses occurring during a high-level activity such as a logon sequence of application startup is not possible without configuring each individual object’s audit setting separately according to which user would initiate access and what type of access would be initiated.

Applying Evidential Weighting Criteria

<table>
<thead>
<tr>
<th>Test</th>
<th>Criteria Applied</th>
<th>Conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connect Activity in Log Evidence to Real-World Activity?</td>
<td>Apply ‘Control over Generation of Evidence’ (6.3.6) to system activity</td>
<td>Logon and process activity logging is a necessary minimum for forensic analysis</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The ability to limit object-access logging to particular high-level activities such as logon is not available</td>
</tr>
</tbody>
</table>

**Figure 8.19: Control Over Generation of Evidence of System Activity**

The flexibility provided by the selection interface is important, as there is a tradeoff between coarse-grained and fine-grained control over evidence generation. Coarse-grained control over high-frequency events typically results in a significant performance penalty, whereas fine-grained control requires choices to be made on how to generate specific evidence. Depending on the implementation of the selection interface, logging may become entirely infeasible due to performance degradation, may have no impact on performance at all, or may have enough impact to limit evidence generation to short intervals only.
Chapter 8: Applying the Evidential Weighting Framework to Microsoft-Windows Logging

In order to get a high-level view of user activity in a computing system, logon/logoff and process tracking are a necessary minimum. The evidence generated from this selection policy will identify user accounts that have been active on the system and all the processes that ran (including applications). Previous discussion on the generation of profiles using the Windows selection interface (section 8.1.8) established that user logon/logoff and process tracking creates a relatively small number of entries thereby having little impact on system performance. Fortunately, from a control perspective, configuring the log to generate this evidence is simple. Both categories of events feature coarse-grained selection, i.e. either all such events are logged or none at all.

However, generating a more comprehensive profile of particular activities in the computing system, such as the events related to a logon sequence or an application’s activities, require associated evidence of object-access activity. Logging any activity featuring a large number of object-accesses is likely to grind the computing system to a halt, therefore careful selection of which object-accesses to log is necessary. On the positive side, the event interface allows all object-accesses by a user, or a group of any size, to a single file to be selected without affecting logging on other files. Further, Windows allows fine-grained control over object-accesses such as reads, writes, and so forth, to be exercised on a single file. However, object-accesses occurring during particular activities, such as during a logon sequence, or associated with a single application, cannot be selected. This is because object-accesses can only be generated when a particular user or group directly accesses the object regardless of what high-level activity is ongoing. As previously discussed in section 8.1.11, it is not possible to profile one application to the exclusion of other applications because selection control on object-access logging is user-centric, i.e., object-access entries are generated based on which user is logged in, rather than which application is requesting access. The same is true for a logon sequence that is likely to ‘touch’ many objects in the process that can only be captured if all objects are tagged for logging. What may be required is conditional logging of objects based on the occurrence of a particular high-level activity such as logon or application start-up.

The logic underlying ‘Control over generation of evidence (criterion 6.4.1) is useful in assessing the flexibility with which a selection interface allows evidence to be generated and its impact on system performance. Therefore, the sub-section number for this section (8.2.7)
appears in the Fundamental Assessment column ‘Connecting Digital Actors to Real-World Actors’ of Table 8-0a for the aforementioned criterion.

### 8.2.8 Guidance on Selection of Evidence of Activity Identification

The Windows’ selection interface is modeled on the operating system view of system activity rather than that of the forensic investigator. Although logon and process tracking events must be logged as a minimum, considerable guidance is needed with selecting object-access events. Windows does not provide guidance on which events are appropriate for different circumstances. Further, the selection interface is structured in a way that contradicts forensic practice. For example, the success and failure of process tracking activity is separated such that both have to be selected if all exit points from a system operation is to be logged.

**Applying Evidential Weighting Criterion 6.4.6**

<table>
<thead>
<tr>
<th>Test</th>
<th>Criteria Applied</th>
<th>Conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connect Activity in Log Evidence to Real-World Activity?</td>
<td>Apply ‘Guidance on Evidence Generation’ (6.4.6) to system activity</td>
<td>No guidance on object-access settings for comprehensive logging of user, application and/or file activity</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Success and failure events are separated by the interface allowing incomplete evidence of an activity to be collected</td>
</tr>
</tbody>
</table>

**Figure 8.20: Guidance on Selection of Evidence of Activity Identification**

Event selection interfaces do not efficiently map real-world events to event logging configurations. Instead, selection interfaces tend to reflect system architecture. Therefore, administrators must identify which events must be tagged to generate sufficient evidence to meet real-world goals. Windows’ selection interface mirrors its own architecture in that the event interface is not defined in terms of real world events but Windows events (see snapshots and description in section 4.1.6).

The previous section suggested that a minimum set of forensic evidence must include logon/logoff and process tracking activity. In addition, depending on the aims of the forensic investigation, particular sets of object-access evidence will be critical. For example, to collect
evidence of a single application’s activities, associated libraries of dlls must be logged. These must first be identified by administrators and then manually tagged for object-access logging. Windows does not provide any guidance on object-access settings that may be useful for administrators attempting to comprehensively log particular actions related to user, application and/or file activity. In general, there is no guidance on which user accounts (e.g. untrusted and high-capability accounts), which applications (e.g. unknown or critical), and which files (e.g. confidential or highly available) may need special logging settings. Further, the selection interface separates the success and failure outcomes of system activities thereby allowing one to be logged without evidence of the other. This can mislead forensic investigators, as not all the exit points of a process are logged. Therefore, if a process is only selected for ‘success’ and it fails, there may be no evidence of the event having occurred. The converse of this scenario is true.

The logic underlying ‘Guidance on Evidence Generation’ (criterion 6.4.2) is useful in assessing the guidance provided by selection interfaces and its impact on evidence generation. Therefore, the sub-section number for this section (8.2.8) appears in the Fundamental Assessment column ‘Connecting Digital Actors to Real-World Actors’ of Table 8-0a for the aforementioned criterion (6.4.2).

8.3 Connecting Timing of System Activity to Real-World Time

The backbone of any forensic investigation is the timeline upon which significant events are dated. Timing plays a key role in establishing a connection between log evidence and real-world events. For example, if log evidence correctly describes a range of real world activities but places some of them inaccurately on a timeline (e.g. one hour in advance or one hour in the past) then the conclusions drawn from the evidence are likely to be misleading. Timing information may be instrumental in actor identification as well. When a user logs in to a computing system, the timing of the act of logging on may be correlated with other evidence such as pictures from a real-world camera to establish actor identity and answer the ‘who’ question.

In almost all forensic investigations involving Windows logging, certain ‘big picture’ timing information will be important. These are likely to include the time when a user logon session began and when it concluded. In the case of network applications that enable email and
browsing activities, the precise beginning and ending times of sessions will also be important. This information assists forensic investigators to identify timeframes as they provide useful bounds within which critical events are known to have occurred. These timeframes can be correlated across computers (time synchronization among computers must also be considered). For example, if malicious network traffic is traced to a particular computer, it may be determined from the log that only one application was active at that time (i.e. only one application process was executing object-accesses). Therefore, it may be argued that the particular application may be responsible for the traffic.

In addition to ‘big picture’ timing information, investigations are likely to focus on specific actions that answer the ‘who’ question and therefore establish intent. These may include the act of sending an email or instructing a web browser to visit a particular web site. In these cases, the specific timings of such actions are of interest. As previously mentioned, these timings can be used as a central reference point upon which user presence is queried as well as network activity.

Two criteria identified in chapter 6 are useful in making an assessment of timing information recorded in Windows log evidence. Evidential weighting criteria ‘Accuracy of individual log entry timestamp’ (criterion 6.3.1), ‘Accurate identification of a sequence of operations’ (criterion 6.3.4) and ‘Accurate identification of incident timeframe’ (criterion 6.4.5) are relevant to this discussion. Criterion 6.3.5 extends the discussion in criterion 6.4.4 that concerns the potential impact of timing on the sequence of entries in a log.

General weighting criteria may be applied to further investigate Windows’ capabilities in recording timing information. The criterion ‘Grouping related log entries to identify system operations’ (criterion 6.3.1) will be used to determine the influence of timing resolution on the ability to differentiate activity taking place at the same time. Complete coverage of system activity (criterion 6.3.7) will be applied to determine the range of timings associated with system activity that can be logged in Windows. Consistency of log evidence across multiple log entries (criterion 6.2.5) will be applied to determine the potential impact of inaccuracies in the logging of system activity. ‘Complete reconstruction of system activity’ (criterion 6.3.6) will be used to determine how important times such as the start of a system operation can be deduced by logging object-accesses taking place at the targeted time period. ‘Consistency of log evidence across multiple log entries’ (6.2.5) will be applied to determine
Chapter 8: Applying the Evidential Weighting Framework to Microsoft-Windows Logging

the potential influence of inaccuracies on the logging of timing information. Windows’ selection interface will be assessed from a timing perspective. ‘Control over evidence generation’ (criterion 6.4.1) will be used to assess Windows’ capability to produce evidence of timing information and ‘Guidance on evidence selection’ (criterion 6.4.2) will be used to assess the usefulness of Windows’ guidance on the generation of timing information regarding system activity.

8.3.1 Accuracy of Individual Log Entry Timestamp
Forensic investigators will be concerned with three aspects of timing accuracy. These are the accuracy of the system clock, if the logged time reflected when the event was generated or when the report was issued and the atomicity of the event reporting mechanism. Generally, in all operating systems including Windows, the accuracy of the system clock used in event logging cannot be determined from the log itself. This is because logs typically include only timestamps and do not reference the accuracy of the system clock used to obtain the timestamps.

Windows’ timestamps are recorded to a 1-second resolution. This is adequate when correlating system activity with real-world physical activity and for comparing high-level system activity. However, there are some cases such as thread activity within the same process that requires finer timing resolution.

Microsoft documentation indicates that Windows recognizes a difference between the time of event generation and the time of event writing. Documentation sources differ on which of the two values is displayed in the Event viewer. The fact that the times of event generation and writing are almost always the same indicates the granularity of the timing information reported is not fine enough to make a useful distinction between these two events. However, with access to both of these timings, it is no longer necessary to investigate atomicity.
### Applying Evidential Weighting Criterion 6.3.1

<table>
<thead>
<tr>
<th>Test</th>
<th>Criteria Applied</th>
<th>Conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relating Timing of System Activity to Real-World Time</td>
<td>Apply ‘Accuracy of Individual Log Entry Timestamp’</td>
<td>Windows recognizes a difference between ‘time of event generation’ and ‘time of event writing’; it is unclear which one is used. In practice, the values are almost always the same indicating the timing resolution is relatively coarse</td>
</tr>
<tr>
<td>The accuracy of the system clock cannot be determined from log evidence</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Figure 8.21: Accuracy of Individual Log Entry Timestamp

Discussion of ‘Accuracy of individual log entry timestamp’ (criterion 6.3.1) identified three key issues relating to timing information in individual log entries. These were the accuracy of the source of timing information used by the log (typically the accuracy of the system clock), whether the logged time reflected when the event was generated or when the report was issued, and the atomicity of the event reporting mechanism. The significance of the first issue is relatively obvious. If the system clock is set to the wrong time then all timing information in log entries will be inaccurate. The second issue is significant in terms of the granularity of the timing information logged, as event generation and event writing register the same time for almost all log event entries. The third issue may be significant depending upon the relative loss of timing precision from delays introduced by the logging mechanism.

Regarding the first concern, there is no information in Windows log entries that indicates the accuracy of the system clock used to determine timestamps. This is a fundamental problem with all stand-alone computing systems, since timing information can only be established through correlation with an independent and authoritative timing source. The event logging service timestamps each event record before writing it to the log. The timestamp is recorded as the number of seconds elapsed since January 1, 1970 according to the system clock (Time, _Time32, _Time64, n.d.).
Although there is some confusion regarding the resolution of timing information in documentation, Windows’ timestamps are recorded to a 1-second resolution rather than a 1ms or 10ms resolution as per an email from the lead Microsoft technical manager on logging systems, E.Fitzgerald on 15 November 2006 (see extract below). For correlating log evidence with external events such as timestamps associated with CCTV, or with most internal computing system type activity, a 1-second resolution is adequate. However, given the potentially high number of events taking place on a Windows system, the timing resolution of 1 second seems coarse (this point will be reinforced with the discussion of TIMEGENERATED and TIMEWRITTEN later in this section) especially where thread activity within a single process must be differentiated.

The accuracy of the timestamp in the event log is one second. The order of events in any given event log accurately represents the order in which they were received by the log.

Regarding manual change of system clock times, Windows logs the change of system time (see log entry below). Entries written prior to the change of time retain their old timestamps, while entries written after the change of time derive their value from the new programmed time. If the manual change sets the system clock to an earlier or later time, the event viewer will not re-sort but maintain the sequence of log entries regardless of the change of time. Therefore, an inspection of the event viewer will show new log entries to be out of time sequence with previous entries. Note that the stored event log will always retain the correct sequence of entries because it uses an internal record index number that maintains chronological order regardless of timestamp values.

The change of system clock time is logged in the security log (event id 520):
Chapter 8: Applying the Evidential Weighting Framework to Microsoft-Windows Logging

### Tale 8-13: Change of System Time Log Entry

It is important to acknowledge that networked computing systems can be configured to allow system time to be synchronized with an external authentic time source. Windows XP/2003 feature W32Time, a service that can maintain an accurate system clock by synchronizing to a reliable time service on the Internet or even a locally-connected hardware source of timing information. The primary use for this service is to act as an authentic time source for the Windows Kerberos authentication service that imposes time limits on access to services (to prevent replay attacks). The loading of Kerberos authentication is logged at startup of a system. This can be correlated with the presence of information messages in the system log to determine if the system clock was synchronized using an authentic time source or not (Windows’ system log indicates system time is not synchronized by logging event id 36 from source W32Time). Note that the purpose of using an authentic time source is focused on getting multiple computers to use common time rather than to get the timing synchronized with the real-world. Therefore, the fact that timing information is synchronized across multiple computers does not necessarily imply that the system clocks are precise in absolute terms.

The second concern pointed out by ‘Accuracy of individual log entry timestamp’ is that there is an inherent delay between requesting an event is reported and writing an event to the log.
Although the event viewer only shows a single timestamp for each event log entry, in reality the stored log (in proprietary binary format) records two timestamps for each record. The first is TIMEGENERATED, which records the time when an event is generated and TIMEWRITTEN, which records the time when an event is written to the log. According to Murray (1998), the latter of the two is shown by the Event Viewer, whereas the Microsoft Developer Network (MSDN) gives a strong hint that since the time of event generation is the more significant of the two, the Event Viewer is showing the former rather than the latter (EventLogEntry.TimeGenerated Property, n.d.). Murray (1998) comments that the two values are almost always the same. Therefore, this indicates that the granularity of the timing information reported in the log is not fine enough to make use of the distinction between these two entries. Murray does go on to claim that in conditions of extreme loading, it is theoretically possible to have a difference of more than a few seconds.

Regarding atomicity of Windows event reporting, given the fact that Windows records TIMEGENERATED and TIMEWRITTEN, there is no need to determine if the audit function is executed atomically as the precise timings of the start of the process and the end of the process are already available. However, since the values of both fields tend to be almost always the same, there is a significant concern only if the forensic investigator must differentiate between timings of events that are less than one second apart (for example, thread activity within the same process). In this case, there is likely to be a problem as the order in which calls to the event log are completed are not necessarily consistent with the order of the events taking place. This is confirmed by the following correspondence from Microsoft (as per an email from E.Fitzgerald on 15 November, 2006):

*It is possible, due to the way threads are scheduled, that events in the Windows security log are out of order from the actual operations that they record. This most commonly happens on multi-processor machines where many activities are occurring simultaneously (example: authentication on a multi-proc DC), or when a longer audited operation is composed of multiple audited operations (example: account creation sets default password, adds user to default group, etc.). The reason this occurs is because of the order in which the threads are serviced; the first thread to get enough time to complete its audit call determines the first event to arrive at the log. Once events arrive at the event log, order is preserved.*
Note that the above observations are predicated on the assumption that Windows is actually querying the system clock separately for TIMEGENERATED and TIMEWRITTEN and not using the same timestamp for both fields.

The logic underlying ‘Accuracy of individual log entry timestamp’ (criterion 6.3.1) is useful in assessing the influence of individual log entry timestamps on evidential weight. Therefore, the sub-section number for this section (8.3.1) appears in the Fundamental Assessment column ‘Relating Timing of System Activity to Real-World Time’ of Table 8-0a for the aforementioned criterion (6.3.1).

8.3.2 Accurate Identification of a Sequence of Operations

The ability to correctly log a sequence of operations depends on the availability of the starting time, significant events and ending time of each operation from log evidence. Windows does not aim to log the precise interval within which a system operation is taking place. As a result, Windows’ ability to accurately identify a sequence of operations in this regard depends on which timing information can be extracted from the kinds of evidence logged by Windows.

In the case of object-access logging, the lack of complete timing information on the read and write sequence makes it difficult to determine if a read from one document precedes the write to another (and vice versa). An interesting concern applies to the logging of individual system operations. A single system operation generating multiple log entries can have the same timestamp (to the second) indicating the resolution of timing information in Windows is too coarse. Therefore, with a timing resolution of 1 second, a high volume of events may impact the chronological sequence of log entries. These issues call into question either the suitability of the timing resolution of Windows and/or the process by which it queries the clock for use in system operations. The outcome in this scenario is uncertainty regarding the sequence of events within a system operation.
Applying Evidential Weighting Criterion 6.3.4

<table>
<thead>
<tr>
<th>Test</th>
<th>Criteria Applied</th>
<th>Conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relating Timing of System Activity to Real-World Time</td>
<td>Apply ‘Accurate Identification of System Operations’ (6.3.4)</td>
<td>Lack of complete timing information on object-access makes it difficult to identify a sequence of operations Multiple entries relating to the same operation can have precisely the same time indicating timing resolution is too coarse</td>
</tr>
</tbody>
</table>

Figure 8.22: Accurate Identification of a Sequence of Operations

The evidential weighting criterion ‘Accurate identification of a sequence of operations’ (6.3.4) discussed the possibility that two system operations running in parallel may execute a series of system events in a different sequence to what the evidence in the log entries conveyed.

Logon can be used to demonstrate the impact of this criterion on the accurate identification of system operations. For example, suppose a user logs onto a network-enabled computing system. Since Windows does not log the starting point of the ‘act of logon’ but rather the exit condition (whether that endpoint be a successful logon or a failure logon), it is possible that network activity taking place prior to or during the logon may be recorded in the event log before the evidence of the logon itself is preserved. The same is true for logoff, since the starting point for a logoff is not documented, the act could hang allowing a different user take control of the console.

In the case of process activity, Windows logs the creation of all processes at the same point in execution (just before the primary thread of the process begins to execute). Therefore, the chronological sequence of log entries documenting the starting sequence of two overlapping processes is likely to be preserved (theoretically, a context switch during the interval between log generation and thread execution may influence accuracy of the record).

In the case of object activity logging there are significant concerns. Suppose a forensic investigation is being conducted into the following scenario. A user opens a document using Microsoft Word and an email message (there are two simultaneously active processes). The
Word document contains sensitive information whereas the email message is empty. The investigation is conducted into the possibility that confidential information was copied using a copy-and-paste process from one file to another. For this example, assume the source file is ‘source_file’ and the destination file is ‘destination_file’.

In the case of Windows NT, object-access logging may produce the following scenario:

**Figure 8.23a: Windows NT object-access logging evidence**

Potential evidence in Windows NT is limited to documenting that both files were open at the same time, and that the source file was open for reading and the destination file was open for writing. There is no evidence that any reading or writing actually took place.

The situation is significantly improved with Windows XP.

**Figure 8.23b: Read before Write**

In the above scenario, the likelihood that information was copied from the source to the destination file is significantly higher because of the operation-level read and write accesses. This is evidence that the source file was read and that the destination file was written to, and that the read preceded the write.
Conversely, Windows XP may log sufficient evidence to rule out the possibility that information was transferred between source and destination files. In figure 8.3.2c, the first read occurs after the destination file was no longer available for writing.

However, a third scenario raises a concern regarding the sequence of system operations:

In this diagram, the first read operation occurs before the destination file is closed. Since Windows XP only logs the first write operation, it is unclear if the destination file was written to after the first read of the source file. Therefore, at a system operation level, a forensic investigation may suggest that there is evidence that the source file was read before the destination file was written. In this scenario, incomplete information about the number and timing of writes to the destination-file has resulted in an unknown sequence of system operations.

Another interesting scenario relates to the sequence of events in a single object-access operation. Referring to the analysis of an object-access operation in section 8.2.2, note that the three log entries identified during the forensic analysis should have their own individual times:
L1: The time the process requested access to the target object
L2: The time the security subsystem processed the request
L3: The time the process executed the intended action on the target object

Forensically, the time of object-access is of critical importance because it establishes when a relevant action took place. Prior to Windows Server 2003 / XP, the time the object was accessed is not available. Therefore, in this case, the beginning point and ending point of the object-access session is the best approximation of when the actual access might have taken place (remember it is unknown if the object was actually accessed). It is relevant to note that some applications exhibit peculiar behavior when accessing objects. They exercise only one permission in an object-access session. As a consequence, the event log registers many pairs of start/end entries for object-access sessions as the application is constantly opening and closing files. In this case, the approximation of when an actual right was exercised on an object is more precise than where an application exercises its permissions multiple times in a single object-access session. Of course the precision of timing information in this case will not be greater than 1 second since the log does not support timing information with greater precision. Therefore, for greater precision of timing information related to object-accesses in this case, the resolution must be finer than 1 second.

Windows XP is able to be more precise for most, but not all, object accesses. Only in the case where operation-based auditing does log an entry representing an actual object-access (not just a security check) is the precise time available. Tables 8-3, 8-4 and 8-6 reveal a new issue related to the accuracy of timing information in Windows log entries. The time of the security check and the actual access are precisely the same. In the case of Table 8-2, the handle is closed at the same time as well. This implies one of two possibilities. Either some of these activities have taken place in less than 1 second (which would again suggest that the timing resolution of 1 second is inadequate for system activity logging), or Windows is reusing the same timestamp for these events.

Recall that the discussion of ‘Accurate identification of a sequence of system operations’ (criterion 6.3.4) presents a scenario where multiple system operations are progressing simultaneously and the starting sequence (operation1 starts before operation2) is not preserved (log_entry_for_operation2 appears log_entry_for_operation1).
According to Microsoft, log entries may still be generated in very close proximity to one another either deliberately or due to a very high volume of events being generated. If the events are generated within 10ms of each other there may be an unpredictable sequence of log entries recorded (EVT_CHANNEL_CLOCK_TYPE, n.d.):

“Note that if the volume of events is high, the resolution for system time may not be fine enough to determine the sequence of events.”

Therefore, the logic underlying ‘Accurate Identification of a Sequence of Operations’ (criterion 6.3.4) is useful in assessing the reliability of the chronological sequence of log entries related to a system operation. Therefore, the sub-section number for this section (8.3.2) appears in the Fundamental Assessment column ‘Relating Timing of System Activity to Real-World Time’ of Table 8-0a for the aforementioned criterion.

8.3.3 Accurate Identification of an Incident Timeframe
To accurately identify an incident timeframe, Windows must log events sufficiently close to the beginning and end of the time interval as well as provide suitable timing resolution to differentiate between the times at endpoints of the incident. Since the granularity of the timing resolution in Windows is one second, the ability of Windows to accurately identify an incident timeframe depends if the timeframe of the incident is greater or less than one second. In the case of a logon operation (the act of logging on), Windows does not log the starting point thereby making it difficult to estimate the incident timeframe.
Applying Evidential Weighting Criterion 6.3.5

<table>
<thead>
<tr>
<th>Test</th>
<th>Criteria Applied</th>
<th>Conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relating Timing of System Activity to Real-World Time</td>
<td>Apply ‘Accurate Identification of Incident Timeframe’ (6.4.5)</td>
<td>Accurate identification of an incident timeframe depends partly if the timeframe of the incident is greater or less than one second</td>
</tr>
<tr>
<td></td>
<td></td>
<td>In the case of logon and process startup operations, Windows does not log the beginning point of a system operation making estimation of the incident timeframe difficult</td>
</tr>
</tbody>
</table>

**Figure 8.24: Accurate Identification of an Incident Timeframe**

The main argument of the criterion ‘Accurate identification of an incident timeframe’ (criterion 6.3.5) was that the precision with which log evidence identified the beginning and end of an incident timeframe influenced the quantity of evidence that had to be investigated. If a large number of log entries were being produced (e.g. object-accesses were enabled) then even slight inaccuracies in the length of a time interval may render a large number of events either inside the timeframe (which increased the cost of investigation if these should have been designated ‘irrelevant’) or outside the timeframe (which may result in crucial evidence being ignored).

Since the granularity of the timing resolution in Windows is one second, the ability of Windows to accurately identify an incident timeframe depends if the timeframe of the incident is greater or less than one second. If the incident concerns events that have taken place within the space of a single second, such as events taking place within the context of a single process, then obviously Windows will not be able to accurately identify the incident timeframe with any greater precision than a single second. Although the activities of a specific process may become the focus of an investigation, the entire incident’s timeframe is unlikely to be less than one second.

Another issue with identifying the timeframe of an incident is the ability of Windows to log events sufficiently close to the beginning and end of the incident. For example, take a system logon action (not the logon–logoff session but just the act of logging on to the system). This act might be important where a significant concern has been identified regarding other
activities taking place in the computing system when the user was logging on (these could be
related to the activities of a remotely logged-in user taking place at the same time when the
user sitting at the console decided to log in). In this case it may be necessary to determine
precisely when the various scripts and so forth were executed to setup the system
environment for the new user logon session. Windows only logs the end-point of the system
operation (success or failure of the logon) and not the starting point – i.e. when the user
finished entering his/her credentials and submitted the information for authentication.
Therefore, Windows does not accurately identify the timeframe of the logon operation. This
is a concern since the logon system operation frequently takes many seconds, i.e. the incident
interval is meaningful in the context of a forensic investigation.

The logic underlying ‘Accurate Identification of Incident Timeframe (criterion 6.3.5) is
useful in assessing the reliability of the chronological sequence of log entries as they relate to
the start and end of incident timeframes. Therefore, the sub-section number for this section
(8.3.4) appears in the Fundamental Assessment column ‘Relating Timing of System Activity to
Real-World Time’ of Table 8-0a for the aforementioned criterion.

**8.3.4 Grouping Related Log Entries to Identify System Operations**

Windows relates log entries representing activity that takes place during a particular second.
This is obvious as a quick glance at a log will show that all entries describing activities that
have taken place during the same second will appear together (since the log is typically sorted
in chronological order). Grouping activity at a sub-second level, significant when logging
object-access events, is not possible in Windows since the timing resolution is limited to one
second.

**Applying Evidential Weighting Criterion 6.3.1**

<table>
<thead>
<tr>
<th>Test</th>
<th>Criteria Applied</th>
<th>Conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relating Timing of System Activity to Real-World Time</td>
<td>Apply ‘Grouping Related Log Entries’ (6.3.2)</td>
<td>Windows relates log entries representing system activity on a per-second basis. Grouping activity taking place at a sub-second level is not possible.</td>
</tr>
</tbody>
</table>

**Figure 8.25: Grouping Related Log Entries to Identify System Operations**
Discussion of grouping in sections 8.1.7 and 8.2.3 has already demonstrated that Windows’ grouping relates log entries to a logon session or a process session using link identifiers. Profiles of particular actors can also be constructed by filtering logs by actor identifiers. It follows from previous discussions that where system operations and activity profiles can be constructed, timing information can be extracted and grouped to get, for example, when a session took place and when an actor conducted important activities.

In general, from a grouping perspective, Windows relates log entries representing activity that takes place during a particular second. This is obvious as logs will show that all entries describing activities that have taken place during the same second will appear together (since the log is typically sorted in chronological order). It is obvious that differentiating activity at a sub-second level is not possible in Windows since the timing resolution is limited to one second. The kinds of log entries that are likely to appear in large numbers and bunched together during short periods of time are object-access entries. In fact, it is frequently the case that a logon session extends over the period of a whole day or even multiple days. Similarly, applications may be open for lengthy periods of time. In this case, grouping object-accesses together is the only way to tell if an application is ‘active’ as opposed to ‘running’ since when an application is active, it will access resources such as the filesystem and I/O devices. Therefore, clusters of object-accesses may generate log entries. In this scenario, grouping from a timing perspective is useful as it defines points of activity in an application’s running history.

Therefore, the logic underlying ‘Grouping related log entries (criterion 6.3.1) is useful in determining if timing information extracted from logs can be grouped so that connections can be made to real-world activity. Therefore, the sub-section number for this section (8.2.3) appears in the Fundamental Assessment column ‘Relating timing of system activity to real-world timing’ of Table 8-0a for the aforementioned criterion (6.3.1).

**8.3.5 Complete Coverage of System Activity**

Applying ‘Complete coverage of system activity’ implies that all significant times relating to any incident must be loggable by Windows, i.e., at the very least, for every system operation involved in the investigation there must be a start/end time. Windows logs the exit conditions of logon/logoff and process startup/shutdown sessions so the end times of each high-level
operation is known. However, the times at which the logon was initiated and process was initiated are not typically known.

### Application of Evidential Weighting Criterion 6.3.7

<table>
<thead>
<tr>
<th>Test</th>
<th>Criteria Applied</th>
<th>Conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relating Timing of System Activity to Real-World Time</td>
<td>Apply ‘Complete Coverage of System Activity’ (6.3.5)</td>
<td>All significant times involved in an incident are not loggable by Windows.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Windows typically logs the exit conditions of an operation leaving out the starting conditions</td>
</tr>
</tbody>
</table>

**Figure 8.26: Complete Coverage of System Activity**

This section applies ‘Complete coverage of system activity (criterion 6.3.7) in order to relate timing of system activity to real-world time. Applying this criterion indicates that all significant times involved in a system operation must be loggable by Windows, i.e., for every real world activity, there must be at least a start/end time. For example, Windows logs two system operations in a logon session – the act of logging on and the act of logging off. Applying this criterion implies that the timeframe during which the act of logging on to a system occurred must be determinable and the timeframe during which the logoff took place must also be explicit. The same applies for application process startup/shutdown. In addition, for every object-access there must be an identifiable time interval during which the access took place (e.g. read and write).

Windows logs the exit conditions of logon/logoff and process startup/shutdown sessions so the end times of each high-level operation is known. However, the times at which the logon was initiated and process was initiated, are not known. In the former case, there is a particular concern as was discussed in the previous section. Further, in the case of Windows NT, there is no timing information regarding actual object accesses except for the beginning and end of the object-access session that can be determined by the open/close of object handles. Although Windows XP logs the first time permissions are exercised, this identifies the start time of the object-access (e.g. the start of the read or the start of the write) but not the end time that can only be determined through logging the last of the series of accesses.
Therefore, the logic underlying ‘Complete coverage of system activity’ (criterion 6.4.7) is useful in identifying the range of timing information that may be useful in connecting log evidence to real-world times. Therefore, the sub-section number for this section (8.3.5) appears in the Fundamental Assessment column ‘Relating Timing of System Activity to Real-World Time’ of Table 8-0a for the aforementioned criterion.

### 8.3.6 Consistency of Log Evidence Across Multiple Log Entries

Possible inconsistencies in the evidence of a logon session are reduced if timing information can be extracted from evidence of multiple application sessions that have took place. Further, timing information extracted from the evidence of object activity reduces the likelihood of inconsistencies related to process activity sessions (and so forth). The more object activity takes place, the more timing information is available regarding object-accesses thereby reducing the likelihood of inconsistencies occurring.

#### Applying Evidential Weighting Criterion 6.2.5

<table>
<thead>
<tr>
<th>Test</th>
<th>Criteria Applied</th>
<th>Conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relating Timing of System Activity to Real-World Time</td>
<td>Apply ‘Consistency of Log Evidence Across Multiple Log Entries’ (6.4.5)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Consistency of timing of logon sessions can be demonstrated given sufficient timing of application sessions</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Consistency of timing of application sessions can be demonstrated given sufficient timing of object accesses</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Consistency checks can be applied given sufficient timing information about object access patterns</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 8.27: Consistency of Log Evidence Across Multiple Log Entries**

This section applies ‘Consistency of log evidence across multiple log entries’ (criterion 6.3.5) to the logon, application and object-access activities in the Windows environment. This criterion points out that the logging of multiple events (redundancy) as a result of a system
operation or part of a system operation allows for inaccuracies to be identified through information consistency checks.

To connect log evidence to a real-world event from a timing perspective, a time interval identifying the starting and ending time within which the real world event occurred is desirable. From a consistency perspective, the more events occur during this interval (each event provides another timestamp), the more information exists upon which information consistency checks can be applied.

Section 8.2.5 pointed out that inconsistencies in the evidence of a logon session are reduced if evidence of multiple application sessions exists during the logon session. Further, evidence of object activity reduces the likelihood of inconsistencies related to process activity sessions (and so forth). From a timing perspective, the same logic applies. Multiple application sessions provides more information on timing within a logon session. Multiple object accesses provide more information on timing within a process activity session. The more object activity takes place, the more timing information is available regarding object-accesses thereby reducing the likelihood of inconsistencies occurring.

Therefore, the logic underlying ‘Consistency of log evidence across multiple log entries’ (criterion 6.2.5) is useful in determining if inconsistencies in timing information can be identified. Therefore, the sub-section number for this section (8.3.6) appears in the Fundamental Assessment column ‘Relating Timing of System Activity to Real-World Time’ of Table 8-0a for the aforementioned criterion (6.2.5).

**8.3.7 Complete Reconstruction of System Activity**

From a reconstruction perspective, beginning times of system operations such as logon can be obtained by applying object-access logging on Windows events that take place at the targeted point in time.
Applying Evidential Weighting Criterion 6.3.6

<table>
<thead>
<tr>
<th>Test</th>
<th>Criteria Applied</th>
<th>Conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connect Activity in Log Evidence to Real-World Activity?</td>
<td>Apply ‘Complete Reconstruction of System Activity’ (6.3.6) to timing information</td>
<td>Beginning times of system operations such as logon can be obtained by applying object-access logging on Windows events that take place at the targeted point in time</td>
</tr>
</tbody>
</table>

**Figure: 8.28: Complete Reconstruction of System Activity**

This section applies ‘Complete reconstruction of system activity’ (criterion 6.3.6) in order to relate timing of system activity to real-world time. Previous discussion regarding timing information was aimed at identifying all possible times logged by Windows that may be useful. Complete coverage of system activity was aimed at identifying timing evidence that was not available for logging but available in the OS environment. Complete reconstruction of system activity is aimed at identifying crucial pieces of evidence that may assist forensic investigators to reconstruct as much of the timing information as possible.

The discussion of complete coverage of system activity in section 8.3.7 suggested that the precise interval during which a logon or logoff took place cannot be determined since the operating system does not provide this facility. From a reconstruction perspective, the timing information required could be obtained by applying object-access logging on Windows events that take place at the targeted point in time. For example, to identify the precise time interval during which a user logged on to a computing system, the user logon event should be logged (both success and failure to cover both exit points) and then object-access logging can be applied to those Windows registry keys and files involved in the earliest possible stages of the logon process. Simply applying a ‘read-access’ audit should generate the required log entries from which timing information can be extracted to estimate the beginning of the system operation’s running time interval.

Therefore, the logic underlying ‘Complete reconstruction of system activity’ (criterion 6.3.6) is useful in identifying evidence that may assist forensic investigators to reconstruct timing...
information. Therefore, the sub-section number for this section (8.3.7) appears in the Fundamental Assessment column ‘Relating Timing of System Activity to Real-World Time’ of Table 8-0a for the aforementioned criterion (6.3.6).

8.3.8 Control Over the Generation of Timing Information Regarding System Activity
The Windows selection interface does not allow the logging of the starting and ending points of timing intervals within which system operations occur. Instead, Windows selects a system event (in the case of the ‘act of logging on’, this event may be an exit condition) not on the basis of timing significance. The proximity of the time of the system event to the beginning or end of the timing interval is unknown.

Applying Evidential Weighting Criterion

<table>
<thead>
<tr>
<th>Test</th>
<th>Criteria Applied</th>
<th>Conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relating Timing of System Activity to Real-World Time</td>
<td>Apply ‘Control over Evidence Generation’ (6.4.1)</td>
<td>The Windows selection interface does not allow the logging of the starting and ending points of timing intervals within which system operations occur</td>
</tr>
</tbody>
</table>

Figure 8.29: Control Over the Generation of Timing Information Regarding System Activity

The Windows selection interface does not allow the logging of the starting and ending points of timing intervals within which system operations occur. When logging a system operation, Windows logs an event that is representative of the entire operation. In the case of process startup, this event takes place just before the first thread begins execution. Although this piece of timing information may prove to be important in a forensic investigation, the timing of the beginning of the process startup is also useful, but not made available. Without this beginning time the only conclusion regarding the timing information that is recorded in log evidence, is that it is a single point in time when the system operation was executing (not much is known about the proximity of this time to the beginning time or the end time of the operation). Some times such events are at exit points – i.e. the success / failure of the system operation may be determined from this event. This suggests that the chosen event is close to the endpoint of the timing interval (however there may be further actions - clean up of used
memory space - yet to be completed). However, the choice of which system event to use to log a system operation is not selected due to forensic considerations. Therefore, much timing information regarding system operations is unavailable. The previous section discussed the start of the logon action as one such system operation.

Therefore, the logic underlying ‘Control over evidence generation’ (criterion 6.4.1) is useful in assessing the flexibility with which a selection interface allows evidence of timing information to be generated and its impact on system performance. Therefore, the sub-section number for this section (8.3.8) appears in the Fundamental Assessment column ‘Relating Timing of System Activity to Real-World Time’ of Table 8-0a for the aforementioned criterion.

8.3.9 Guidance on the Selection of Timing Information
Since Windows’ selection interface does not allow events to be selected based on timing, therefore, no guidance is provided on when events should be logged.

8.4 Log Management
The Windows model of log management revolves around the standalone computing system. For example, many security measures are aimed at preventing the logfile on the standalone system from being corrupted or moved off-site. Related to other management issues, there is no support for log centralization. Although the log management part of the evidential weighting criteria is also designed to assess a single system, it must be noted that enterprise logging will almost certainly require activity logs to be moved off-site. Windows does not provide support for centralized logging in general.

8.4.1 Security of Windows Event Logging
Windows has implemented a number of measures by default to make it difficult for an external party to compromise the event logging service. These involve protecting the SCM and LSASS from unauthorized access. Although Windows implements significant protective measures to safeguard event log evidence, there may be ways to augment security by moving some user-mode components of the event logging service such as LSASS into kernel mode. In addition, the path from the source of the event data to the event log entry may be encrypted to prevent malicious tampering. However, it must be acknowledged that the processing
capacity consumed by such a technique may result in a denial of service on the event logging service.

Windows protects event log files by requiring administrator privileges to clear/delete logs and preventing them from being directly copied off the system. However, there are further methods that have been researched that have not been implemented. The integrity of evidence that has been written to an event log can be protected by use of a tamperproof event logging technique. This technique prevents modification or destruction of event log entries from going undetected (Schneier & Kelsey, 1998).

**Applying Evidential Weighting Criterion 6.5.1**

<table>
<thead>
<tr>
<th>Test</th>
<th>Criteria Applied</th>
<th>Conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assessing Evidential Weight of Log Management Strategies</td>
<td>Apply ‘Security of Evidential Record (6.5.1)’</td>
<td>Windows implements security measures to protect the event logging service from unauthorized access. Other measures such as moving the LSASS may further augment process security.</td>
</tr>
<tr>
<td>Windows protects event log files by requiring admin privileges for log management and making copying difficult. Other measures such as tamper-proofing have not been implemented</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 8.30 Security of Windows Event Logging**

Discussion of ‘Security of evidential record’ points out that information security attacks can be classified into three categories – attacks on confidentiality, integrity and availability. Windows manages event log evidence from the time a request is made for an event entry to be written, to the time when evidence is extracted from the digital environment. Therefore, evidence stored in log files is vulnerable to attack from unauthorized accesses. In addition, processes involved in the generation and reporting of events log entries are vulnerable to attack as well as they may be manipulated in order to influence the logs. Further, the log file must be protected from self-contamination arising from Windows’ management processes such as the log retention policy.
Regarding process security
In general, the security of any event logging system is as strong as the security of the computing system. Windows has implemented a number of measures by default to make it difficult for an external party to compromise the event logging service.

Windows does not allow third-party software to write to the security log. However, the same is not true for system and application logs. The separation of the security log from the application and system logs works as a security measure by compartmentalizing relatively trusted evidence from less trustworthy evidence. Note that previously, Windows NT allowed third party processes from writing log entries to the security log.

Although Windows protects the event logging service by making it an internal component of the operating system, the integrity of the operating system itself (including all protection measures) relies on the trust afforded to the super-user account. Windows, like most operating systems, uses a single line of defense to protect the super-user account – a username/password combination (of course, exploiting vulnerabilities in the design of Windows may allow access to super-user privileges).

The Windows auditing subsystem consists of a number of process-level components. Some of them function in user-mode whereas others function in kernel-mode. The LSASS writes entries directly to the event log. It receives event records from the SRM but also generates log entries itself and from the SAM.
Figure 8.31a: LSASS Architecture

See the flow of event log records below:

Figure 8.31b: Flow of Control When Generating Event Log Records
Windows takes a number of measures to protect the event logging service. Firstly, the event logging service may be configured to start on boot-up by the Service Control Manager (SCM) or manually started by the systems administrator. However, once started, the event log service cannot be manually stopped by a user (or super-user) on the system (section 4.1.8). Secondly, Windows protects the LSASS from being contacted by a masquerading process attempting to write to the Security Log. During system initialization, the SRM connects to the LSASS by creating a private communication port. The LSASS reciprocates by creating another private communication port to which the SRM connects. Once these connections are made, either process does not attempt to listen for connection requests preventing third party processes from successfully connecting to the two key components in the event logging service. Thirdly, Microsoft had allowed processes to write to the Security Log using an Application Programmer Interface (API) in Windows NT. This would allow only pre-programmed and tested routines to access the security log. However, processes are no longer allowed to access the Security log using any means in Windows XP. Note, the event logging service has been known to hang under extreme loading conditions (Murray 1998).

The security of the log generation infrastructure depends on processes operating in user-mode as well as those operating in kernel-mode. Note that the process responsible for writing a log entry (LSASS) sits in the user mode part of OS architecture and other key components such as the SRM that enforces security checks and generates requests for logging sits in the kernel mode. A comprehensive evaluation of the security of the event logging service is outside the scope of this thesis. Such an evaluation must consider the dual levels of protection instituted by Windows for processes running in the user-mode and kernel-mode.

**Regarding log file security**

Windows also protects event log files by requiring admin privileges to clear/delete logs and preventing them from being directly copied off the system. This is achieved by having the SCM change the format of the file slightly so it appears to be corrupt to unauthorized processes. However, note that event log files can be backed up using the BackupEventLog function in the API. Therefore, there does exist means of getting around this protective measure. Further, once the event logging service starts, the log files are opened and are not closed until the service shuts down. If any other process attempts to write directly to the log while it is open, a sharing violation will result. It is important to remember that physical
access to a system implies that logical protective measures can be over-ridden. For example, if the system is rebooted under MS-DOS, the event log files are no longer protected by the above measures.

Although Windows has introduced a range of measures to protect the event logging service, there are further methods that have been researched that have not been implemented. The integrity of evidence that has been written to an event log can be protected by use of a tamperproof event logging technique. This technique prevents modification or destruction of event log entries from going undetected (Schneier & Kelsey, 1998).

Therefore, the logic underlying ‘Security of evidential record’ (criterion 6.5.1) is useful in assessing the impact of security attacks on the evidential weight of event logs. Therefore, the sub-section number for this section (8.4.1) appears in the appropriate cell in Log Management Table 8.0b for the aforementioned criterion.

### 8.4.2 Retention Policies of Windows Logging

The loss of evidence, as a strategy, is not acceptable whether it is from overwriting or dropping events. As very large files may be difficult to manage, new log files may be created on reaching a maximum to keep file size manageable. However, the option of overwriting evidence as a retention strategy must be discouraged perhaps (by disabling all such strategies by default).

**Applying Evidential Weighting Criterion**

<table>
<thead>
<tr>
<th>Test</th>
<th>Criteria Applied</th>
<th>Conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assessing Evidential Weight of Log Management</td>
<td>Apply ‘Contamination from log retention processes’ (6.4.5)</td>
<td>A deliberate strategy to drop evidence is not forensically acceptable. Retention strategies resulting in the loss of evidence must not be enabled by default</td>
</tr>
</tbody>
</table>
The discussion surrounding ‘Contamination from log retention policies’ pointed out that allowing event log entries to be overwritten results in the risk of loss or contamination of evidence. The purpose of a retention policy is to manage the amount of disk space consumed by event log files.

Windows allows administrators to specify the maximum size of an event log and what to do if the log reaches this maximum. The default policy is to overwrite the oldest events by newer events. A variation on this policy is to set a certain time period after which overwriting on events older than that point can proceed. A third possibility is to drop any events arriving after the log is full (until the log is cleared).

The loss of evidence, as a strategy, is not acceptable whether it is from overwriting or dropping events. As very large files may be difficult to manage, new log files may be created on reaching a maximum to keep file size manageable. However, the option of overwriting evidence as a retention strategy must be discouraged perhaps by disabling all such strategies by default.

Therefore, the logic underlying ‘Contamination from log retention processes’ (criterion 6.5.5) is useful in assessing the influence of retention strategies on log evidence. Therefore, the subsection number for this section (8.4.2) appears in the Log Management Table 8-0b for the aforementioned criterion.

8.5 Final Assessment

The final assessment of this evaluation chapter consists of two sections. The first section identifies those weighting criteria that have identified the most significant concerns regarding Windows evidence collection and management. The second section outlines recommendations for system administrators and operating system engineers from an evidential weighting perspective.
8.5.1 Usefulness of Weighting Criteria

This evaluation applied 16 evidential weighting criteria to three fundamental tests (who/what/when) as they relate to past events. Not all the evidential weighting criteria were applied for each of the three tests. Some were exclusively linked to one test whereas others were relevant to all three tests. This section discusses criteria that, when applied, have identified characteristics of Windows that negatively impact evidential weighting.

‘Actor identification across multiple log entries’ plays a significant role in assessing Windows’ capabilities in identifying real-world users responsible for instigating particular system activities. This issue is a common and critical objective of forensic investigators. Unfortunately, Windows does not collect user-centric evidence, opting instead to pass raw user input to third party applications without regulating their behavior or providing logging facilities to preserve such evidence. The precise user input issued to an application takes effect within the application environment that is outside of the scope of Windows logging. Therefore, it is not possible to determine if the user explicitly instructed the application to carry out a function or whether the application carried out the function for other reasons. This discussion inevitably makes its way to the identity of the application itself, specifically, an investigation into the application and its usual range of functionality. Unfortunately, Windows limits the logging of application activity to the creation/termination of the application process and interactivity with objects, whereas the kinds of activities that are typically the subject of forensic investigations are document management and usage of network applications, such as email and web browsing. Windows does not offer logging facilities for these kinds of activities and usage of system services (e.g. executing dlls) in general. A consequence of the lack of coverage of system activity by Windows logging is the ineffectiveness of powerful profiling paradigms. Since evidence of user input and key application activity is largely unavailable, identifying users and applications is difficult.

Applying ‘Identifying key activities in a system operation’ resulted in the discovery that Windows does not actually log evidence of object-accesses in Windows NT. Instead it logs the request for an object-access. This key finding is significant from an evidential point of view as it implies there is no basis for claiming that any file or object was touched, read or modified on that type of system. Microsoft has introduced a new category of logging, known as operation-based logging, in Windows XP. However, the new logging facilities only preserves the first time a particular type of object-access permission is exercised in an object-
access session. Therefore, only the first of many writes will be logged, the same is true for reads and so forth.

‘Complete reconstruction of system activity’ was used to identify a set of evidence that can be logged for the purpose of making inferences regarding system activity that cannot be explicitly logged. The Windows audit selection policy can be used to identify system activity that did not take place and system activity that may or may not have taken place. Further, this criterion gives an investigator timing information not available from Windows log evidence. An important piece of information is the time a user initiated a logon sequence. This piece of information can be determined by enabling object-access on registry keys or files that are accessed at the early stages of a logon sequence.

‘Accuracy of individual log entry timestamp’ revealed a number of concerns regarding timing information in Windows log evidence. Firstly, that the accuracy of the system clock cannot be determined from log evidence as there is no information to qualify the accuracy of the timing source. Secondly, although the one-second timing resolution may be appropriate for correlating events in the physical world to system activity, this timing resolution is not suitable when differentiating thread activity within the same process. Thirdly, log entries related to object- accesses frequently cite precisely the same time for more than one (distinct) event. For example, an object-open, operation-based access, and object-close may have precisely the same timestamp. This indicates either the timestamp was queried only once in three events, or the timestamp is not sufficiently fine to preserve the sequence of events in the record.

‘Control over evidence generation’ identified two paradigms for controlling evidence generation through the selection interface. The first is a coarse-grained mechanism that is used for user logon/logoff and process tracking. Here, the system operation is tagged itself, so that a log entry is created every time it runs regardless of other circumstances. The second is a fine-grained mechanism like that of object-access. Here, certain object accesses can be logged while others are not. This choice is based on the identity of the user/group logged on. Coarse granularity logging is excellent for admissibility when used in isolation as it ensures a ‘big picture’ of user logon sessions and process flow will be preserved. The latter is excellent for profiling as it allows all possible accesses to a single object to be recorded to the exclusion of other objects. However, forensic evidence collection places a different set of
demands on the selection interface. Although some of them, such as logging all activity involving a single file-object, can be implemented without difficulty. Others, such as logging all the activity of a single application is more difficult to achieve as it requires all process interaction with all objects to be logged to generate the required evidence.

8.5.2 Recommendations on Windows for Systems Administrators and OS Designers

A number of recommendations can be extracted from the evaluation of Windows in this chapter. As a minimum, system administrators must log process activity as well as logon/logoff (some object-accesses may need to be logged to support the minimum). These two categories of evidence can be used to identify user account names, application image filenames (possibly including locations) and some important timing information such as when the logon and logoff were completed, when the application completed its startup sequence and its exit (note Windows logs exit conditions). In general, whenever logging an event, both the success and failure of the event must be logged. If a process is only selected for ‘success’ and it fails, there may be no evidence of the event having occurred. The same is true if the event is selected for ‘failure’ and it is successful. System administrators are advised to identify user accounts and applications of interest before configuring event selection. All object-accesses executed by high-capability and untrustworthy user accounts should be logged. At the same time, applications of interest should have their binaries tagged for logging to identify if they have been modified in any way. To reconstruct important timing information regarding system operations like logon and logoff (the beginning time of the logon sequence and the logoff sequence cannot be readily identified from logon/logoff auditing), appropriate registry keys and object-accesses can be identified for logging purposes.

Operating system engineers will find this chapter to be a particularly rich source of advice on design for forensic objectives. In general, evidence of user activity must be targeted for improved logging. In particular, activity at the user-interface and application usage of system services is of interest. Coverage of system activity that is common in forensic investigations, such as network access, must be integrated into the existing security logging infrastructure. For each system operation, the beginning and ending events must be loggable as well as all significant events in the system operation. All object-accesses in an object-access session must be loggable, not just the first read, write, execute and so forth. The control interface must be more flexible so that object-accesses associated with a system operation such as
logon can be logged without having to first identify the specific objects that might be accessed while the system operation is being executed. Further, the control interface must allow object-accesses to be logged based on which application (not which user) is doing the accessing.

### 8.5.3 Useful Characteristics of this Framework

There are two important characteristics of this framework that distinguish it from previous investigations of system activity logs. Firstly, the framework incorporates forensic, systems and legal perspectives thereby making contributions to research in forensic science and computer science. Secondly, the framework takes a comprehensive view of the system event log function thereby identifying an exhaustive range of issues within the designated scope. Both Price (1997) and Schaen and McKenney (1991) investigate event logs from a systems perspective. Price takes a systems view of logging focusing on the availability of information from the audit function of multiple operating systems and queries if this information is suitable for misused detection systems. Schaen and McKenney (1991) attempts to identify a comprehensive range of issues related to the network audit function from a systems perspective. Sommer’s framework is intended for the same purpose as the framework developed in this thesis – i.e. to determine the weighting of evidence collected by a human-directed forensic investigation.

Price identifies some key concerns regarding the reliability of the audit function, however Price’s primary contributions that are relative to this thesis, is identification of various system activities that are not loggable by mainstream operating systems. Compared to Price, this framework identifies a more comprehensive range of issues pertaining to the availability of important system activity (see discussion of Complete Coverage of System Activity in sections 8.1.8, 8.2.4, 8.3.5). This discussion is just one weighting criterion (6.3.7). A range of other criteria looks at other characteristics of the audit function such as consistency of reporting, ability to group entries together, accuracy of data recorded (such as actions and timestamps) and selection concerns, among others.

In the investigation conducted by Schaen and McKenney, host-based audit issues are also covered however these are identified at a high-level of abstraction. An example is the issue “Auditable Events Must be Enabled or Disabled”. Within the brief discussion of this issue, Schaen and Mckenney notes that a tradeoff must be made between the usefulness of audit
data and the amount of storage capacity required to hold the data. The design of the audit function can be more comprehensively investigated using ‘Control over Evidence Generation’ and Guidance on Evidence Selection’. These criteria investigate the granularity of event selection and the resulting control over evidence generation that can be applied to various kinds of evidence produced by operating systems. For example, these criteria are applied to evidence of logon, process and object activity in this chapter.
Chapter 9

Conclusion

This dissertation began with a critical analysis of diverse and frequently contradicting views from practitioners and academics on the fundamentals of Digital Forensics. Two key issues critical to Digital Forensics were identified in chapter 1 by asking the question, ‘how can the evidential weight of system activity logs be maximized?’ These were, firstly, that evidence collection from digital environments needs to be improved. Secondly, that event logs may play a significant role in achieving this objective.

Chapter 3 points out that event logs in current operating systems were not designed for the purposes of evidence collection. Event logs have been discussed in the context of security-motivated event data collection. Security literature identifies several deficiencies in the general capability of the logging mechanism to collect event data. These include the inability to characterize events and to provide information on network and user-level activity (e.g. commands).

Interestingly, these same deficiencies influence the forensic evidence collection capability as well. This is not entirely surprising since these issues relate to event data collection in general, irrespective of motivation. However, despite the existing discussion on the logging capability of operating systems, little research has looked at improving the ability of the logging mechanism to collect event data in general and forensic evidence in particular. In this thesis, the logging mechanism’s capability to collect forensic evidence is investigated from a weighting perspective, and an evidential weighting framework is developed and then applied on a real-world operating system. The models used to develop the framework and the framework itself is contributions to research in event logging and digital forensics.
9.1 Contributions

To address the primary research question, chapter 1 posed the following sub-questions:

1. How can the evidence collection capability of event logs be determined?
2. How can the weight of event log evidence be measured?

The major contributions of this thesis revolve around these questions presented at the introduction to this thesis. The concept of ‘evidential weight’ is in itself a contribution. The term ‘evidential weight’ is used as a qualitative measure of log evidence in its role of collection of evidence of system activity and evidence management as well as authentication of computer-derived evidence in general.

To answer the first research sub-question, a generic ‘technology-independent’ logging model was abstracted from a real-world family of operating systems (Microsoft Windows) as existing literature did not present a suitable equivalent. This model was constructed to determine the evidence collection capability of event logs and was based on three factors - event detection, event selection and event description.

The role of event logs in evidence collection was investigated to answer the second sub-question. In the course of this investigation the following question was posed – ‘given the forensic investigator is ultimately tasked with evidence collection from the digital environment, what role can event logs play in assisting the forensic investigator to collect evidence?’ This is a useful question because it implicitly recognizes that event logs may play more than one role. In fact, this research identified that event logs can play at least two roles. One role was already envisaged, i.e. the collection of evidence of system activity and its management until the time it is recovered by the forensic investigator. Another role that is critical from both an evidence admissibility and weighting point of view was called ‘utility’.

A major contribution of this thesis is the development of evidential weighting criteria based on these two roles of event logs. To construct this framework, a parallel was drawn between the forensic investigator’s extraction process and the event log’s own extraction process. The outcome of this exercise was an analysis, interpretation and finally a definition of the three attributes of event log evidence namely authenticity, accuracy and completeness - in terms of what they mean for log evidence. In addition to the three attributes, a new attribute termed
‘utility’ was also identified. Utility is an attribute of event logs whereby they can collect information about the reliability of the digital environment within which logs are functioning. Utility can establish the authenticity of all evidence derived from that computing environment such as hard disk evidence and even the log itself. Traditionally, the testimony of the forensic investigator has been the sole means of establishing the reliability of evidence from digital environments. Although event logs may never replace the forensic investigator’s testimony, they represent a second authority on the reliability of the digital environment.

In addition to the major contribution being the development of the evidential weighting criteria discussed in the previous paragraph, the primary contribution of this thesis is the identification of a comprehensive range of weighting issues specific to event log evidence (see table 6.1). As indicated in section 1.3, an analytical approach is adopted that employs a breadth-wise approach towards identifying as many relevant issues related to the evidential-weight of logs as possible. This contribution utilizes both the logging model and the evidential weighting framework in chapter 6. To answer the primary research question, the significance of the various evidential weighting tests is discussed in the context of the legal domain in chapter 7. The range of weighting considerations previously outlined in the analytical discussion in chapter 6 is prioritized according to the legal distinctions between admissibility and further weighting considerations.

The practical usefulness of the evidential weighting framework is demonstrated by rigorous and systematic application of the issues identified in chapter 6 to Windows log evidence collection and management. This investigation assesses the capability of Windows (log evidence) to answer the fundamental questions that are common to legal relevance and reliability considerations. These are the who/what/when questions (as they relate to past events) that an event log must answer in almost all circumstances. In general, the evaluation of Windows identified a range of concerns (see Table 8-0a/b), many of which could have been addressed by Microsoft had forensic evidence collection and management been a design goal. For example, on the one hand the ‘actor identification’ criteria (6.2.2 and 6.2.4) point out that Windows does refer to digital actors within evidence of system activity. However, these criteria also demonstrate that these references are not reliable and that profiling needs to be used to better identify actors, by constructing patterns of activity for actor identification. However, Windows does not provide adequate support for actor profiling. This discussion leads to a more significant finding – i.e. Windows’ approach to evidence collection is not
user-centric but process-centric. As a result, collecting direct evidence of user involvement in system activity is difficult, as it must be traced through process activity. Another example is the accuracy of log entry timestamps. Windows’ log evidence binds timing information to system activity, thereby providing forensic investigators with a rough timeline of many significant events from which an approximate incident timeframe and a chronological sequence of high-level system operations can be determined. However, the one-second timing resolution in Windows is not suitable for differentiating events taking place in less than 1 second of each other. By means of demonstration, events such as object-open, operation-based access and object-close may frequently have precisely the same timestamp. Further, the accuracy of the system clock used to produce timestamps cannot be determined from log evidence.

Given the design constraints of the Windows logging system, practical advice on forensic evidence collection and management can be extracted for systems administrators and design engineers. For example, systems administrators are advised to ensure that logon and process activity logging is enabled. This is to collect a minimum amount of evidence that can describe how many users logged into a system, what account names were used, when each session started and stopped, and which applications and other processes were executed (along with approximate session timings). At the same time, administrators are advised to select both success and failure events so that evidence of the event occurring is guaranteed to be logged regardless of the process’ exit conditions.

Further, discussion on the tactical use of object-access logging identifies a range of uses such as activity reconstruction and monitoring changes to actor source files that may influence actor identification. Regarding object-accesses, there also need to be guidelines on minimal logging keeping in mind tactical objectives.

Significant weighting concerns arising from the design of operating systems can also be identified from the evaluation of Windows. Crucial issues include the lack of evidence of direct user input into Windows, the inability to log both events taking place inside an application environment and an application’s use of system resources such as network access. Further, evidence describing a system operation is typically limited to exit conditions whereas evidence of key activities as well as the start and end of an operation should be
Chapter 9: Conclusion

preserved (see section 8.5.2 for more recommendations for systems administrators and operating systems engineers).

9.2 Limitations and Future Work

This thesis is based on an analysis of mainstream system activity logs. However, the contributions of this thesis may apply to system activity logs in general (even though other kinds of logs like those produced by customized hardware and add-ons that capture data from live operating systems were not explicitly investigated). Further, the scope of the universal principles of accuracy, completeness and authenticity were applied on logs while they remained in the digital environment. Therefore, other issues that apply to log evidence from the time it is extracted from the digital environment to the time it is assessed by the court are not considered (e.g. chain of custody).

The event logging model developed in chapter 4 was based on a network-enabled yet standalone computing system. In the real-world, event logs are collected from a number of computers that are networked. To assess the weight of log evidence from a networked environment requires a more sophisticated logging model. Further, the event logging model used in this thesis was developed from high-level features of mainstream operating systems. In this thesis, Windows was used as the choice of operating system used to construct this model (although not much was said about UNIX, the features used to develop the model in chapter 4 are common to both operating systems).

Future research may seek to apply the evidential weighting framework to other logging systems such as ERP systems (eg. SAP) and even dedicated application environments. A number of lessons have been learned from exercising the framework on Windows that will be useful in further applications. First, forensic investigations will frequently focus on collecting user-centric evidence that ties the role of the real-world user to system activity. Therefore, logging systems that do not capture input from the real-world will be deficient in actor identification. Second, logging systems that preserve a wide-range of (significant) activities will generate more useful actor profiles than those that log a relatively limited range of activity. Third, enough log entries must be generated per system operation to uniquely identify it and to have enough redundancy to test system reliability and malicious tampering. Fourth, generating event entries at the beginning and end of a system operation results is
important for accurate accounting of time intervals. Fifth, any log retention strategy that results in the loss of evidence is not forensically acceptable.

In a more theoretical vein, the evidential weighting framework could be applied to theoretical operating system designs to determine their forensic suitability. In general, operating systems research may benefit from extracting design principles for forensic-friendly operating systems. The same is true for research into various aspects of computing systems that may have a bearing on forensics and evidence collection. For example, this thesis considers user interface design purely with the aim of gauging its influence on evidence collection and management. Future research into user interface design may look at the issue of forensic guidance incorporated in a selection interface from a usability perspective.

This thesis adopted an analytical approach, focusing on the issues of evidential weighting in a breadth-wise manner. The evidential weighting framework was not applied to a specific situation in an organization. Therefore, there are no practical results or data that may indicate what evidential weighting is achievable in real-world situations. The topic of evidential weighting of activity logs may lend itself to action research. Without the need for any implementation, system activity logs in organizations can be configured for evidence collection and the evidential weighting framework can be used to assess weighting. Of course, this thesis identifies a number of potential improvements that can be made to logging in operating systems. Although any of these can be implemented, the real benefits from pursuing this line of research will only be realized when the implementation is deployed in a live situation and evidence is gathered and then assessed using the evidential weighting framework.

The evidential weighting principles used to construct the framework were not specific to a particular legal jurisdiction. Although the fundamental principles are likely to be more or less the same in Western legal systems, every legal jurisdiction tends to have (some) different rules and interpretations. These were not considered in this thesis. Further, there are many interesting avenues of legal research that arise from this thesis. For example, the literature review did not find substantial research into how logs have been used in legal proceedings and how their weight is being assessed.
Chapter 9: Conclusion

Of particular interest would be the competence of forensic investigators in assessing evidential weight given the high likelihood of misunderstandings that can be traced back to inadequate and misleading documentation. The evidential weighting framework in this thesis may be applied to evidence previously assessed in court in order to make comparisons. A series of comparisons may reveal some common factors that affect the way log evidence is perceived in the legal profession.
References


References


Broucek, V. & Turner, P. (2002a) "Risks and Solutions Arising from Illegal or Inappropriate On-Line Behaviours: Two Core Debates within Forensic Computing", In: EICAR.


References


Clough, J. (2002) *The Admissibility of Digital Evidence*


References


References


Robbins, J. *An Explanation of Computer Forensics*,

http://www.computerforensics.net/forensics.htm


References

Userinit (n.d.). Retrieved March 6, 2007 from

In: Proceedings of the IEEE Symposium on Security and Privacy


McFarlan, F. W. (Ed.), The Information Systems Research Challenge (pp. 111-132).


Windows Security Logging and Other Esoterica (n.d.). Retrieved March 6, 2007 from

Workshop, D.F. A Road Map for Digital Forensics Research www.dfrws.org

In: Workshop on Information Assurance and Security


Appendices A-1 to A-4

Appendix A-1: Logon/Logoff Entries in Windows XP In Reverse Chronological Order

[Entry Number Here] Security
Type: AUDIT SUCCESS
Computer: DIS-WS0002062
User: DIS-WS0002062\adam
User initiated logoff:

User Name: adam
Domain: DIS-WS0002062
Logon ID: (0x0,0xa7c60)

[Entry Number Here] Security
Type: AUDIT SUCCESS
Computer: DIS-WS0002062
User: DIS-WS0002062\adam
Successful Logon:

User Name: adam
Domain: DIS-WS0002062
Logon ID: (0x0,0xA7C60)
Logon Type: 2
Logon Process: User32
Authentication Package: Negotiate
Workstation Name: DIS-WS0002062
Logon GUID: {00000000-0000-0000-0000-000000000000}
Appendix A-2: Process Creation/Exit Entries in Windows XP
In Reverse Chronological Order

[Entry Number Here] Security
Type: AUDIT SUCCESS
Computer: DIS-WS0002062
User: DIS-WS0002062\adam
A process has exited:

   Process ID: 3500
   Image File Name: C:\Program Files\Microsoft Office\OFFICE11\WINWORD.EXE
   User Name: adam
   Domain: DIS-WS0002062
   Logon ID: (0x0,0x8C82F)

[Entry Number Here] Security
Type: AUDIT SUCCESS
Computer: DIS-WS0002062
User: DIS-WS0002062\adam
A new process has been created:

   New Process ID: 3500
   Image File Name: C:\Program Files\Microsoft Office\OFFICE11\WINWORD.EXE
   Creator Process ID: 3300
   User Name: adam
   Domain: DIS-WS0002062
   Logon ID: (0x0,0x8C82F)
## Appendix A-3: Object Access Entries in Windows XP

### In Reverse Chronological Order

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Handle Closed:</td>
<td></td>
<td></td>
<td></td>
<td>Object Server: Security</td>
</tr>
<tr>
<td></td>
<td>Handle ID: 272</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Process ID: 1784</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Image File Name: C:\Program Files\Adobe\Acrobat 7.0\Reader\AcroRd32.exe</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Object Access Attempt:</td>
<td></td>
<td></td>
<td></td>
<td>Object Server: Security</td>
</tr>
<tr>
<td></td>
<td>Handle ID: 272</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Object Type: File</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Process ID: 1784</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Image File Name: C:\Program Files\Adobe\Acrobat 7.0\Reader\AcroRd32.exe</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Access Mask: ReadData (or ListDirectory)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
[Entry Number Here] Security
Type: AUDIT SUCCESS
Computer: DIS-WS0002062
User: DIS-WS0002062\adam

Object Open:

Object Server: Security
Object Type: File

Object Name: C:\Documents and Settings\adam\My Documents\bluehills.pdf

Handle ID: 272
Operation ID: {0,830609}
Process ID: 1784

Image File Name: C:\Program Files\Adobe\Acrobat 7.0\Reader\AcroRd32.exe
Primary User Name: adam
Primary Domain: DIS-WS0002062
Primary Logon ID: (0x0,0x8C82F)
Client User Name: -
Client Domain: -
Client Logon ID: -

Accesses: READ_CONTROL
SYNCHRONIZE
ReadData (or ListDirectory)
ReadEA
ReadAttributes

Privileges: -

Restricted Sid Count: 0
Appendix A-4: Application Process Accesses DLLs Without User Consent

In Reverse Chronological Order

[Entry Number Here] Security
Type: AUDIT SUCCESS
Computer: DIS-WS0002062
Time: 25/10/2006 5:09:39 PM ID: 562
User: DIS-WS0002062\adam
Handle Closed:

Object Server: Security
Handle ID: 556
Process ID: 2864

Image File Name: C:\Program Files\Microsoft Office\OFFICE11\WINWORD.EXE

[Entry Number Here] Security
Type: AUDIT SUCCESS
Computer: DIS-WS0002062
Time: 25/10/2006 5:09:39 PM ID: 567
User: DIS-WS0002062\adam
Object Access Attempt:

Object Server: Security
Handle ID: 556
Object Type: File
Process ID: 2864

Image File Name: C:\Program Files\Microsoft Office\OFFICE11\WINWORD.EXE
Access Mask: ReadData (or ListDirectory)
Appendices A-1 to A-4

[Entry Number Here] Security

Type: AUDIT SUCCESS
Computer: DIS-WS0002062
Time: 25/10/2006 5:09:39 PM  ID: 560
User: DIS-WS0002062\adam

Object Open:

    Object Server: Security
    Object Type: File

    Object Name: \Program Files\Microsoft Office\OFFICE11\1033\ID_011.DPC

    Handle ID: 556
    Operation ID: {0,529532}
    Process ID: 2864

    Image File Name: \Program Files\Microsoft Office\OFFICE11\WINWORD.EXE

    Primary User Name: adam
    Primary Domain: DIS-WS0002062
    Primary Logon ID: (0x0,0x12300)

    Client User Name: -
    Client Domain: -
    Client Logon ID: -

    Accesses: READ_CONTROL
             SYNCHRONIZE
             ReadData (or ListDirectory)
             ReadEA
             ReadAttributes

    Privileges: -

    Restricted Sid Count: 0
Appendices A-1 to A-4

[Entry Number Here] Security
Type: AUDIT SUCCESS
Computer: DIS-WS0002062
Time: 25/10/2006 5:09:39 PM   ID: 562
User: DIS-WS0002062\adam
Handle Closed:

Object Server: Security
Handle ID: 516
Process ID: 2864

Image File Name: C:\Program Files\Microsoft Office\OFFICE11\WINWORD.EXE

[Entry Number Here] Security
Type: AUDIT SUCCESS
Computer: DIS-WS0002062
Time: 25/10/2006 5:09:39 PM   ID: 567
User: DIS-WS0002062\adam
Object Access Attempt:

Object Server: Security
Handle ID: 516
Object Type: File
Process ID: 2864

Image File Name: C:\Program Files\Microsoft Office\OFFICE11\WINWORD.EXE

Access Mask: Execute/ Traverse
Appendices A-1 to A-4

[Entry Number Here] Security
Type: AUDIT SUCCESS
Computer: DIS-WS0002062
Time: 25/10/2006 5:09:39 PM   ID: 560
User: DIS-WS0002062\adam
Object Open:

Object Server: Security
Object Type: File

Object Name: C:\Program Files\Microsoft Office\OFFICE11\1033\SRINTL.DLL

Handle ID: 516
Operation ID: {0,527261}
Process ID: 2864

Image File Name: C:\Program Files\Microsoft Office\OFFICE11\WINWORD.EXE

Primary User Name: adam
Primary Domain: DIS-WS0002062
Primary Logon ID: (0x0,0x12300)
Client User Name: -
Client Domain: -
Client Logon ID: -
Accesses: SYNCHRONIZE
          Execute/Traverse

Privileges: -
Restricted Sid Count: 0
Author/s: AHMAD, ATIF

Title: Digital forensics: increasing the evidential weight of system activity logs

Date: 2007

Persistent Link: http://hdl.handle.net/11343/42195

File Description: Digital Forensics: Increasing the Evidential Weight of System Activity Logs