Policy-based Vulnerability Assessment for Virtual Organisations

Jan Muhammad, Thomas Doherty, Sardar Hussain
National e-Science Centre,
University of Glasgow
University Avenue, G12 8QQ, UK
jan@dcs.gla.ac.uk

Richard Sinnott
Department of Computing and Information Systems,
University of Melbourne,
Melbourne, Victoria, 3010, Australia

Abstract: ***E-Infrastructures can be used to support e-science and e-research allowing different collaborators from disparate organisations, often from different disciplines and utilising heterogeneous software and hardware, to work together on common research problems. This is typically achieved through the formation of targeted Virtual Organisations (VO). Inter-organisational collaborations also bring challenges of security that must be overcome. There has been much work in e-Research-oriented security, i.e. at the middleware level, but far less on ensuring that middleware-oriented security is not made redundant through ensuring the robustness of the underlying hardware and software (fabric) upon which the e-Research middleware security is based, e.g. the operating systems, network configurations and core software required to support e-Research solutions. To tackle this, an integrated security framework is needed that is cognisant of VO requirements on e-Research middleware-oriented security and incorporates targeted fabric level security. In this paper we present an integrated architecture (ACVAS), which encompasses VO-specific fabric security including configuration-aware security monitoring (patch status monitoring) and vulnerability scanning and subsequent updating. We show how tool support can be used to pre-emptively identify and assess potential vulnerabilities in a VO, before they are potential exploited. We also outline how these vulnerabilities can be dynamically overcome to support the needs of the VO and associated e-Infrastructure to improve the overall VO security.***

Keywords: e-Infrastructure, Configuration Management, Monitoring, Security.

1. Introduction

The next generation of distributed computing such as Grids and clouds provide enabling environments (e-Infrastructures) that can be used by researchers from different organisations and research domains to undertake joint work in a collaborative and shared environment [1]. These e-infrastructures are often realised through seamless access to and sharing of compute resources (HPC clusters, SMP machines, ...); data resources (databases, storage and file systems, …); and other targeted resources, e.g. visualisation facilities shared by wider communities.

Such collaborations are based upon on shared access to and use of distributed hardware and software – the underlying *fabric*, with middleware and applications that sitting upon this fabric that make transpaent the problems inherent with distribution and heterogeneity. There has been a body of work undertaken in showing how middleware and applications-oriented security can be tackled through for example, middleware integration with advanced authorisation infrastructures [2-7]. However at the heart of these solutions are some fundamental assumptions. Two of the most dangerous assumptions are that the underlying fabric is secure and that sites take all steps necessary to ensure their overall security. These assumptions are naïve and can be dangerous, especially when e-Infrastructures are applied to more sensitive, security-oriented application domains such as e-Health.

In reality, an e-Infrastructure is as secure as the weakest link in the overall security of the system. Any vulnerability found in the fabric or the middleware can lead to a wide range of attacks leading to situations such as loss of service provision, lack of trust among collaborators and infrastructures providers, as well as many other negative consequences. This problem is not insurmountable, however it does require the harmonisation of e-Infrastructure security and the underlying fabric security.

According to CERT [8] most intrusions result from exploitation of known vulnerabilities, configuration errors, or virus attacks where countermeasures were already available. One of the major contributors to network attacks is poor configuration and maintenance of the underlying fabric. These can give rise to opportunities to those with malicious intent to penetrate into misconfigured sites and their underlying systems. A second factor in vulnerabilities is one of software flaws and bugs found in software systems. Even when security fixes for a particular vendor software (patch) is made available, there is typically a time window in the actual application of those patches. This situation is exacerbated in large scale systems such as Grids composed of many complex subsystems such as the basic hardware layer, the operating system (OS) and the middleware on which services are provided to end users, i.e. VO members [9]. To make sure that operations are successful at each layer and systems are not compromised, site administrators need to observe and monitor a variety of system parameters. From the temperature, voltage, and fan speed at the hardware layer; the disk and memory usage, CPU load at the OS layer; the number of jobs and typical resource utilisation, through to applications-oriented security. Each of these layers and the security information required to measure and assess the dangers of site compromise and overall end-end security of inter-organisational collaborations needs to be achieved in an encompassing security framework.

In this paper we show how VO-wide security can be augmented with configuration management including patch monitoring and vulnerability scanning in a VO-targeted manner. Key to this work is the requirement that local systems and organisations are always assumed to be autonomous. We show how the combination of

vulnerability scanning, configuration management and patch monitoring can be integrated to detect VO-specific vulnerabilities; monitor patch status across the VO and ultimately configure multiple resources across multiple nodes to improve the overall security of the VO.

2. Related Work

Installing and configuring middleware and application software across multi-domain Grid-aware computing facilities is a nontrivial process. In the existing literature we can find a number of reasons for these complexities: larger number of sites (often cross countries) at which software needs to deployed and maintained; different domains/ownership of resources and hence varying policies and configurations at sites, and the heterogeneous nature of hardware/software systems.

The EU project Enabling Grids for E-sciencE (EEGE) [10] security guidelines suggest two important aspects of large-scale system monitoring of Grids which should be adopted by local system administrators. Firstly, administrators are required to keep a secure, central log server to understand the cause of attacks and comprises. It is essential that this resource is protected so that would-be attackers are not able to access and change logs and obfuscate what they have done for example. Secondly, administrators are advised to perform local patches in *as timely manner as possible*. This timeliness can be a major issue for many collaborators.

Examples of attacks on un-patched systems are widespread. The recent '*Google Hack*' [11] is indicative of many. In this case, malicious intruders accessed Google's cloud services, despite the fact Microsoft had issued an emergency patch for its Internet Explorer flaw [12]. The question is why was this patch not applied in a timely (immediate!) manner? A major issue with patching is that it is complex in its nature and there are often operational reasons why organisations don't apply them immediately [13]. For example, users with the ability to install new software on their local machines may find that conflicts exist, e.g. with anti-virus or spyware software, which can result in the ineffectiveness of the protection software without the user ever knowing or even seeing a warning message. Studies have shown problems with traditional patching systems, e.g. software configured for manual update which never actually occurs; roaming users switching off their system earlier than the scheduled time for the update to take place or licenses which may expire causing updates not to happen at all [14].

Shibli et al [15] conducted a vulnerability and patch management evaluation using secure mobile agents. Their approach suits homogenous (closed) networks environments but does not address resources across multiple domains as typified by Grids. Similarly, the Systems Management Server (SMS) [16] is another popular patch management system from Microsoft but it too does not cover heterogeneous environments since it is only targeted to Windows-based systems. In e-Science and e-Research domains, heterogeneity is commonplace and must be overcome. This includes heterogeneity of the software and hardware as well as heterogeneity on the demands on security as a whole. The security demands for a VO that deals with medical records will be fundamentally different to the security demands of a VO supporting particle physicists for example.

To now, it has been a common practice that security has focused on *ad hoc* approaches where sites are trusted to set up and establish their own local security systems and common security policies put in place across these sites – albeit they typically accept a common authentication model (based around public key infrastructures). This model does not meet the demands of inter-organisational collaborations, where sites should have a common set of security policies aligned with the goals of the VO that is to be established. This includes software configuration, site monitoring and patch management policies as well as the more typical VO-specific policies, which tend to be structured around VO membership. Automated tools and processes that allow improving this process are thus essential for VOs, the communities that they support and the providers of resources associated with those VOs. Ultimately, security is about minimising the risk of dangers and if it can be identified that a given site is at risk or has not taken adequate measures to protect itself then this is a risk not just to that site but potentially to all other VO resources that are accessible through that site.

3. System Architecture

In order to tackle these problems discussed in previous section, an integrated and extensible architecture that can be repurposed to the needs of different VOs and their underlying fabrics is required. One such architecture Automated Configuration & Vulnerability System (ACVAS) is shown in Figure 1 and described below.

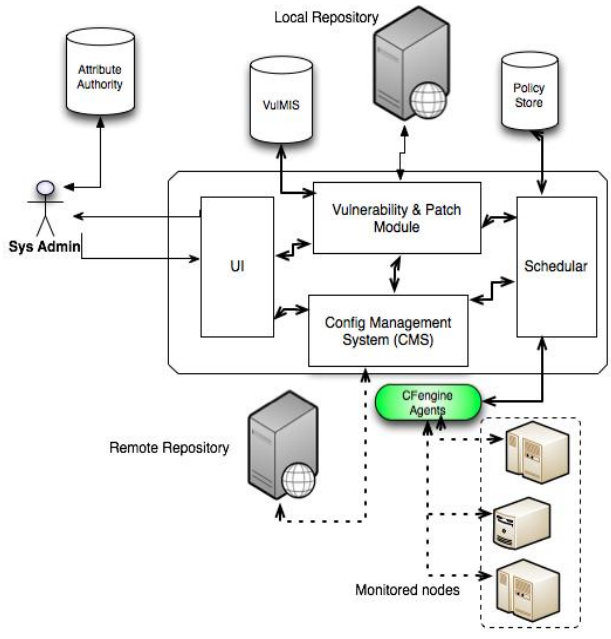

Fig. 1. An integrated approach for Vulnerability Management System

This architecture supports several key components and associated interfaces.

3.1 User Interface (UI)

The User Interface (UI) in Figure 1 provides system administrators with a web based interface to access and configure/use the underlying components present at the

different middleware and fabric layers. Specifically it allows access to a Vulnerability and Patch Module (VPM) - currently based on the open source Pakiti system [17], and a patch monitoring system and Configuration Management System (CMS). Through this interface *authorised* administrators are able to:

- run vulnerability scans (discovering vulnerable packages installed on local and *potentially* remote target nodes);
- download hashed versions of updated patches from trusted remote repositories, i.e. patches signed by trusted vendors and checked/validated by administrators at the local patch repository;
- query the organisational inventory system and policy repository.

Through this interface organisational policies for monitored nodes are configured to ensure that only those with suitable VO-specific privileges can access and use the VPM and CMS systems on their VO-resources. Given the nature of the information stored in the VPM, access to these reports is restricted using access control mechanisms specified in the configuration. The user interface allows site administrators to configure settings needed to process data received from monitored nodes. This includes the list of operating systems belonging to VO-specific domains to which remote nodes belong and basic statistics showing the number of packages which are up-to-date and importantly, not up-to-date and where security updates available.

3.2 Vulnerability and Patch Module

The VPM module provides patch status monitoring and vulnerability scanning services for VO-specific target nodes. Prior to any node becoming part of a particular collaboration, the VPM makes sure that no vulnerabilities exist on a given node that can subsequently be exploited by an attacker and hence endanger the overall collaboration. The VPM generates reports that outline the different potential vulnerabilities in a given VO resource that can subsequently be used to determine potential vulnerabilities and hence risks associated with a given resource. Access to this information needs to be protected to local administrators and those in the VO with sufficient privileges (signed VO-specific credentials used for VO-specific authorization decisions).

The VPM is based on a client/server model for checking the patch status of monitored systems. From the monitored nodes perspective, the client agent is responsible for collecting information about installed packages on local VO-nodes. Once securely collected this information is sent to the central VPM server. On the VPM service end, comparison of actual installed packages against the current list of possible updates is undertaken with results displayed as a comprehensive set of information about current status of nodes in the VO identifying those that are vulnerable or not. The algorithm for this process is shown in Figure 7.

The VPM module in some respects is similar to the patch and vulnerability status monitoring infrastructure currently used at the EGEE's Operational Security Coordination Team (OSCT) [18]. For example, when a severe vulnerability is found, the OSCT identifies and warns its collaborating sites by giving instructions to apply security updates on affected nodes if they are potentially affected. By adopting this approach, the VO and site administrators can keep track of sites vulnerable to specific security issues. However a key distinction of this work is that OSCT largely adopts a manual process and is push-oriented, with best effort vulnerability management and configuration management performed separately and in a manual fashion. The contribution of this work is to securely automate and hence expedite this process.

3.2.1 Vulnerability Management Information System

The Vulnerability Management Information System (VulMIS) is a key part of the ACVAS architecture. VulMIS keeps a formal record of the VO-specific resources and their configuration. In the ACVAS this is realised by a MySQL-based database containing information in an extensible range of VO-specific tables including the hardware, software and potentially other third party software (such as anti-virus and anti-spyware) across the VO nodes as shown in Figure 2.

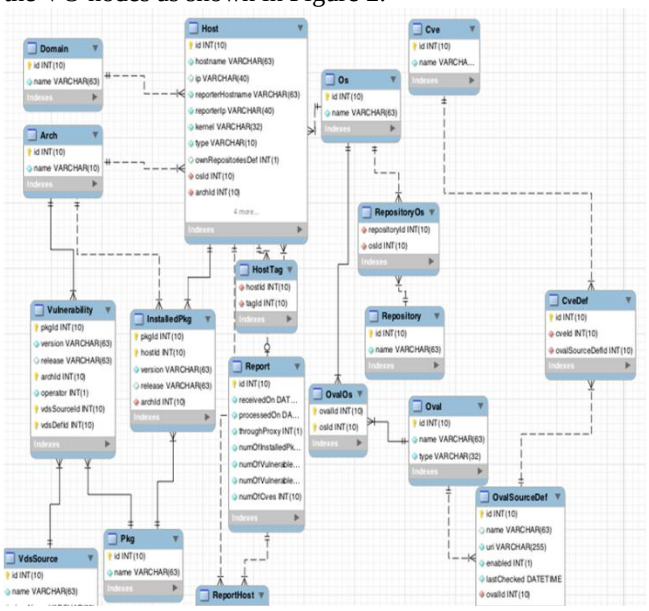

Fig. 2. The Vulnerability Database (VulMIS) schema

VulMIS is populated when clients send their data synchronously to the server to provide central monitoring capabilities. Access to the VulMIS is restricted to authorized VO-administrators using VO-specific credentials and VO-specific access control policies. It is worthwhile mentioning that VulMIS contains details of a multitude of packages installed via a variety of package management software systems used as local resources including yum [19], apt-get [20] etc.

VulMIS requires details of exactly what kind of information is required across the VO. The typical core set of information includes the name of packages and updates (e.g. RPMs) available and a listing of what OS each package was available for.

In many cases, software would only need to be compiled for one or two operating systems. For example, Red Hat Linux systems already have several packages that can be used directly for building other systems.

VulMIS also requires information on the versions of

software packages used for the specific VO and any information on the underlying system requirements, along with a list of software packages that should be installed on VO resources to support the VO as a whole. This includes versions of software that are required.

The key fields from VPM module are shown via an ER diagram (Figure 3), which gives an overview of the relationship between data items used in VulMIS. Inspired by [21], the VulMIS database has six main entities:

- *Vulnerability Entity* – a textual description of the vulnerability with associated (resolvable) identifiers such as CERT ID, BugTraq ID, CVE ID;
- *Vulnerability Specifications Entity* – including a description of the vulnerability consequences, hardware requirements, the name of vulnerable application/service;
- *Environment Specifications Entity* – including an enumeration of the user/application or service objects which may be exploitable;
- *Application/Services Entity* – including the name of application/service and its identity, the vulnerable versions, hardware requirements. The application/services entity can also have associated additional fields like protocols, port numbers, etc;
- *Operating System Entity* – similar to application entity but for the operating system;
- *Exploit Entity* – the actual exploit including the privileges needed to execute the exploit and the consequences of any exploitation.

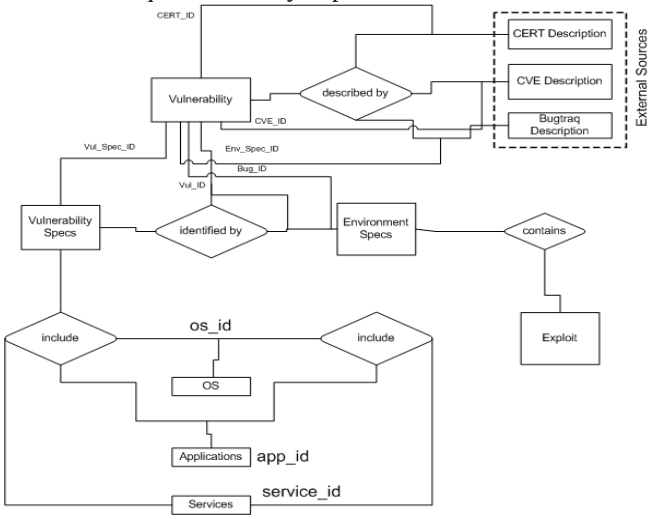

Fig. 3. VPM module Database Structure Inspired from [21]

3.3 Policy Store

The Policy Store (PS) contains the specification of the site specific and VO-level policies that are used to enforce access control across the sites that are relevant to the VO itself. Thus it might be identified that as part of the VO all non-essential services that are not required for the operation of that VO might be turned off, e.g. telnet. Alternatively it might be mandatory that all VO databases have strong passwords set. Individual sites/nodes are obliged to abide these policies as part of the VO membership itself.

3.4 Patch Repository

The Patch Repository (PR) shown in Figure 1 serves as a source of trusted patches (signed and verified) that can be automatically downloaded from remote repositories as required for a particular VO. Often a PR will only contain patches from recognized (trusted) software vendors and/or VO collaborators themselves. Currently this component supports remote repositories such as rpm-based packages from Redhat Linux and Debian, and for VO-based local repository based on yum. In both cases the PR needs to ensure that all packages are appropriately trusted (signed).

To ensure trustworthiness of patches the VO-system administrator can specify which vendor's package repositories will be monitored for updated package versions. Each package repository is assigned to the operating system group it represents. Checks can be preconfigured for repositories specific to monitored hosts, and information provided on misconfigured hosts.

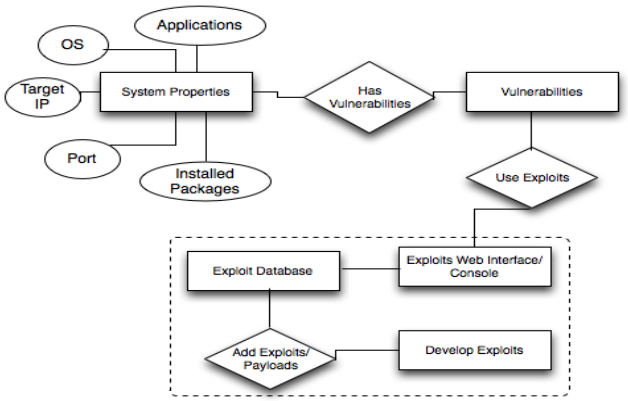

Fig. 4. VulMIS System Properties, adapted from [21]

As shown in Figure 4, VulMIS is depicted in the form of an ER-Diagram. The entity set system properties includes the operating system, the running applications along with their versioning information, installed packages, ports, target IPs as its main attributes. A distinct combination of these attributes maps to a distinct set of vulnerabilities from the vulnerabilities entity set. These vulnerabilities can be exploited by a certain set of exploits or be patched by certain set of available patches. Each exploit can potentially use a distinct set of payloads to compromise a target system and perform certain specific tasks. It is important to note that the vulnerabilities entity along with Patches, Exploits and Payloads are regularly updated from publically available databases such as (NVD and Bugtrack etc either in an automatic fashion (preferred) or manually depending on VO-specific needs.

3.5 Analyser

In the VPM module, the analyser shown in Figure 5 is used to monitor the patch status of target VO-nodes. Other functions include scheduling the scans for target VO-nodes, managing configuration inventories and storing information obtained from remote nodes in the inventory. The analyser gathers, converts and stores security knowledge from multiple sources such as OVAL [22] and CVE [23] in OVAL format. It then initiates a logic-based comparator to match the obtained information from remote repositories and those of target nodes stored in VulMIS. A significant advantage of this architecture is that the central

analyzer has a complete view of the configuration of every managed host in the VO, and can conduct various high-level security analysis on the configuration information.

3.6 Scheduler

A VO-wide scheduler periodically runs to makes sure that the VPM server is available to clients, i.e. target VO-nodes so that they can report any vulnerabilities or when updates are available to be applied. The scheduler also checks that any server daemons to support this process are running.

The collector unit (not shown in Figure 5) transfers the actual data from remote clients to a Pakiti module using the cURL protocol [24]. cURL is an open source program used for transferring data with http or https protocol.

As depicted in Figure 5, the scheduler provides a bridge to the upper layers to connect to remote nodes. It also enables VO-administrators to schedule vulnerability scanning and configuring affected nodes. The scheduler makes sure that the client-agent is running on the remote monitored VO-nodes and that the VPM module sends the remote node data to the centralised VulMIS.

At the heart of this system are targeted VO-specific Policy Decision and Enforcement Points (PDP/PEP). These gatekeepers allow/deny access to each of the individual components and sub-components in the architecture. For example in the current implementation, each PDP is configured to only allow remote sys-administrators with valid (digitally signed credentials) to install and upgrade application level patches and configure VO-specific databases, e.g. setting default VO-database passwords to strong ones.

3.7 Configuration Management System

The Configuration Management System (CMS) component is based upon the configuration management tool CFengine [25]. The CMS is used to patch install/update VO resources based on local (site-specific) and VO-wide policy. Typically this would require that the resource was a vulnerability-free node and that only trusted patches were installed and configured by known/trusted (authorised) sys-administrators. Once all constraints are met on targeted VO resources, e.g. those where patching may be necessary, the CMS is used to push patches (or optionally these can be pulled by configuration and patch management clients) to affected target node(s), where they are subsequently checked for local authorisation decisions before dynamic installation and configuration takes place.

3.8 ACVAS Data Flow

To understand how these components support the discovery and resolution of exploits across a VO, we describe the typical scenarios and interactions that are supported.

1. System admin authenticates (using X509-based PKI and not described here) and puts in a request to a VO-specific attribute authority for the credentials required to access the VPM server for their own specific VO.
2. The attribute authority returns the credentials of the authenticated system-administrator. This will include VO-specific attributes/privileges embedded in an X509 credential.
3. Using these VO-specific credentials, the system administrator puts in a request to check on monitored node/nodes.
4. The secure interface at the VPM module confirms (authorizes) the request by checking its local access policy for this particular request using the VO-specific credentials that are provided.
5. The client-agents provide a list of installed packages (patches.xml) to the VPM module. The scheduler also makes sure that the VO-client-agent is running on all monitored nodes.

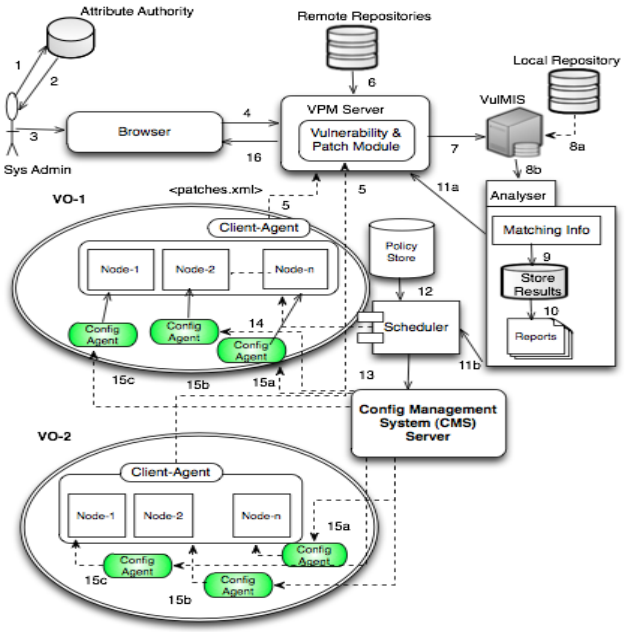

Fig. 5. Architecture & Data Flow for Multiple VOs

6. The VPM module receives updated hashes of the packages installed on monitored nodes from remote (trusted) repositories. The VPM makes sure that the trusted patches from vendors are available for the analyser to compare with the installed ones.
7. The VPM module saves the data received from remote nodes in its VulMIS database.

8(a). In step (8a), the local repository, which serves as a tested and trusted patch store for affected nodes, keeps a replica of remote repository patch updates.

9. In step (8b), the actual analysis phase starts (this algorithm is shown in algorithm 1.
10. Packages are matched, results stored and reports of existence of vulnerabilities generated.
11. In step (11a) reports are sent back to the system administrator via the VPM module.
12. In step (11b), if a vulnerability is found, reports are sent to the system administrator including the severity of the vulnerability and the scheduler is activated.
13. The CMS Policy store checks on the process that needs to be followed with regards to low/medium/high vulnerabilities.

14. The scheduler automatically triggers the CMS to address the vulnerability either by installing an update or fixing any misconfiguration.
15. The scheduler accesses the monitored nodes and affected client agents pull available configuration information from the CMS server.
16. Configuration-Agents (15a, 15b, 15c) installed on the target/monitored nodes pull the latest available configuration changes (subject to CMS policy for that particular node) using their VO-specific configuration-agents.
17. Finally updates on vulnerable nodes are sent back to the system administrator along with packages to resolve the vulnerability, and step 5 is repeated to make sure that the latest information is written to VulMIS.

Figure 6 shows a typical CFengine script for invoking the VPM instance on a target node to start the VPM client service that subsequently returns the patch status of the target node to the VO administrator (VO-admin role) via the UI.

```
control:
  url = ( http://www.example.com )
  actionsequence = ( "module:check_web ${url}" shellcommands )
  AddInstallable = ( RestartWeb )

shellcommands:
  Restart Pakiti::
    "/sbin/service pakiti start"
module:check_web
---------------
#!/bin/sh
url=$1
code=`curl -fsw %{http_code} $url |tail -1`
if [ "$[26]" != "200" ]; then
       echo +RestartWeb
fi
```

Fig. 6. CMS script invoking the VPM

```
1.  Get host ID
2.  Get hostname and ip address
3.  If host is behind proxy:
4.    then
5.     Is proxy is authorised to send data?
6.  else
7.  Connection denied
8.  elseIf
9.  Get host-information (new host)
10. data (host id, installed-packages,
    vulnerable-packages, versions)
11. Start Transaction
12. Connect to DB (VulMIS)
13. If host already (data in VulMIS)
14.    then
15.    Get other HTTP variables
16. (host, OS, architecture, kernel, site,
    version, type of package)

17. Parsing Package List
18. Set initial information about Vul-
    report
19. Get the host object(ID)
20. If host already in DB (VulMIS)
21. else
22. Create host object (ID)
23. Get Host Group
24. Get host tag

25. Call Handler
26. end
```

Algorithm 1. The analyser algorithm in VPM

4. Authorisation Rules for Policy Enforcement

Based on the functions (described in Table 1 at the end of paper) ACVAS supports various authorisation rules to define and enforce security policies. These rules are defined in the following section:

Rule 1 : *A has_account rule is of the form:*

$$has\_account(e, h, a) \leftarrow$$
$$exec\_code(e, h, r),$$
$$p \in permissions\ (r)$$

where e, r, h, a, p are entity, role, host, account and permission respectively.
This is an access control rule that says an entity that has an account with a role *r* on host *h* can execute code with permission *p*.

Rule 2 : *An accessFile rule can take of the form:*

accessFile (P, H, Access, Path) ←
execCode(P, H, User),
localFileProtection(H, User, Access, Path).

The above function specifies that principal *P* can *Access* and execute the files specified by *Path* on host *H*. Access can be either read, write, or execute.
With Rule 2, if an attacker P can execute arbitrary code on machine H with User's privilege, then he/she can have whatever access User has to files. The included predicate *localFileProtection* is a derived predicate specifying that the User on host H can have the specified Access to the file in the given Path.

Rule 3 : *hacl(Source, Destination, Protocol, DestPort).*

Rule 3 implies that the *Source* host can reach *DestPort* on *Destination* host through *Protoco*l. HACL is an abstraction function dependent on the physical topology, firewall rules, the configuration settings of other devices such as routers and switches, and so on. It is compatible with the high-level specification used in many automatic network management tools [22, 23, 5, 10]. Site administrator would typically use a range of tools to obtain such network information.

Rule 4 : *An netAccess rule is of the form:*

netAccess(P, N1, N2, Protocol, Port) ←
execCode(P, H1, User,'voadmin'),
hacl(H1, H2, Protocol, Port).

Rule 4 implies that if a principal *P* has local access on host *N1* as some *User* with *VO-administrator* role and the network allows *N1* to access *N2* through *Protocol* and *Port*, then the principal can access host *N2* through the given protocol and port. This rule allows for reasoning about multi-host attacks, where an attacker finds a weakest link a system, by first gaining access on one machine inside a network and launches further attacks from there.

The following rule specified in [27] uses predicate *principalCompromised(p1,p2)* to express that principal *p1*'s credential has been compromised by principal *p2*, i.e. the attacker. In the following we specify two rules which

state that under what condition a principal's credential has been identified as being compromised.

Rule 5: *A compromised rule is of the form:*

principalCompromised(Victim, Attacker) ←
hasAccount(Victim, H, User,site-admin),
execCode(Attacker, H, root, install root-kit),
malicious(Attacker).

Rule 4 implies that the principal '*Victim*' has an account *User* with privileges of site-administrator on host *H*. Another principle, the *Attacker* takes control of the host as *root* with privilege to install malicious *'root-kit'*. In this case Victim's credential can be assumed to be compromised by the *Attacker*, i.e. once an attacker compromises a machine, he/she can has access to the shadow file of the system, get the private key file of every user, or install key-stroke loggers or Trojan-horse SSH clients on the compromised

5. ACVAS Policies

ACVAS supports two main types of policy: site specific (local) policies and VO-wide policies which are similar to trust-contract policies discussed in [28]. These policies are triples of the form <*subjects, objects, actions*> with the option to contain <*obligations*>. Policies can relate to different stages of accessing resources. In this section we discuss three types of resources: credentials as resources; database and files as resources, and software/systems as resources. Policies can consist of multiple rules that can be combined to form a higher level '*PolicySe*t'.

5.1 Local Site-specific Policies

These types of policies are defined by resource providers and used to make access decisions for resources they provide. Normally sites define policies to protect the use of their resources including but not limited to data (files, database objects) or indeed compute cycles etc. Once defined these policies are stored in policy repositories and made available to policy decision points (PDP) used for making local decisions on access request to different local resources. For sites collaborating in a decentralised environment, it is often required to use a range of attribute authorities (AA) for managing security information, i.e. definition and management of privileges required for fine-grained access control.

The tabular representation for some of the local (site) policies is shown in Table 2. These cover various entities (subjects), resources (various database tables of VulMIS described in Figure 5, which can be accessed and the actions allowed to requesters. In the policy definition for site-specific or VO-specific resources exemplar tables from VulMIS database is presented. With the subject (requester) field, the possible attributes can be roles or user names or both.. In some cases, sites might include obligations, i.e. (conditions) as part of policies, which are to be enforced by policy enforcement points (PEP). Once defined, these policies are stored in the VO-specific Policy Store shown in Figure 5 of ACVAS architecture.

Table 2: Tabular view of local (site-specific) policies

Role	Resource	Action	Obligations
Jan : Gla.Siteadmin	*Gla:Installed-Packages*	*Select, Insert, Update*	
John:Gla.VOadmin	*Gla:Vulnerability*	*Select, Upgrade*	*Email site admin*
Gla.Secspecialist	*Gla:Host*	*Select Insert*	
Gla.Siteadmin/production	*Gla:Host*	*Select, Insert*	*Email VO admin*
Gla.VOadmin/developer	*ED:Vulnerability*	*Select*	*Email VO admin*
VOadmin	*Gla, Ed:Installed-Packages*	*Select*	
Gla. ACVAS/siteadmin	*Gla:Host*	*Select, Insert, Update*	

The site-specific policies indicate what roles are available and the relevant permissions associated with each role in the VO. In table 2, each of the rows is defined and subsequently enforced as a policy rule. The site level policies indicate what roles (principals) are available and their associated permissions on resources available on an organisational host. Each row of Table 2 can be enforced as a policy rule. For example:

has_account(Jan,Gla.SiteAdmin, node3, access) ←
exec_code(Jan.Gla, node3, compile-test.py),
InstalledPackages('execute') ∈ permissions (Gla.SiteAdmin)

This policy rule states that *Jan* (who has role of Glasgow *Site Administrator*) can access *node3* and compile code (test.py) file if allowed action on resource *'InstalledPackges'* is set to *'execute'*.

Similarly:

netAccess(VOadmin, VOnode1,VOnode2, ftp,23) ←
execCode(VOadmin,VOnode1, developer),
hacl(VOnode1, VOnode2, telnet, 22)

States that if a principal such as *VOadmin* has local access to *VOnode1* with *VOadmin* role and the local site allows *VOnode1* to access *VOnode2* through the FTP protocol running on port 23, then the principal can access VOnode1 as a *developer* and telnet from VOnode1 to VOnode2 through port 22. This rule allows for reasoning about multi-host attacks, where an attacker first gains access on one host inside a network and launches further attacks from there.

As demonstrated in Table 3, different VO-wide resources need different forms of protection. Keeping in view the sensitivity of many VO-wide resources, assignment of VO-specific roles to individual (known) and trusted VO-administrators is essential.

In following section we show an example policy, which is used to capture the needs of a VO-wide requirements and protect the resources identified in Table 3.

Here each subject in Table 3 is listed with privileges to have access to the *InstalledPackages* resource. Here the XACML rule composition algorithm is based on the "first-applicable" match. This means that the value of the first rule evaluated to be true is returned as the final result of the policy. The policy depicted in Figure 7 will return a permit result if the requester has a *SiteAdmin* role or a *Common Name* of *Jan Muhammad*, otherwise it returns a deny result. A sample of an XACML policy that allows a VOadministrator to select only records from InstalledPackages resource with obligations that email to be sent to the site administrator that a VOadmin role has access this resource is shown in Figure 7.

5.2 VO-specific Policies

VO-specific policies are similar to those of site-specific policies described in the previous section. However, these are designed for validating and enforcing agreement or trust contracts [29] between hosting sites and a particular VO.

Table 3: A VO specific policy protecting *Vulnerability* resource

Subject	Resource	Action	Obligation
Ed : Ed.Siteadmin,GLA.SA	*Gla:Vulnerability*	*Select, Insert, Update*	
Any : Ed.Siteadmin,GLA.SA	*Ed:Vulnerability*	*Select, Upgrade*	*Email site admin*
Gla.Secspecialist	*Gla:Vulnerability*	*Select Insert*	
Gla.Siteadmin/production	*Gla:Vulnerability*	*Select, Insert*	*Email VO admin*
Gla.VOadmin/developer	*Gla,Ed:Vulnerability*	*Select*	*Email VO admin*
Ed:VOadmin	*Ed,Gla:Installed-Packages*	*Select*	
Gla. ACVAS/siteadmin	*Gla:Host*	*Select, Insert, Update*	

VO-specific policies are used when a trusted, remote VO-administrator requests a service or resource from a particular site. VO-specific policies restrict access to VO-specific resources. For example the first row in table 2 implies that anyone from University of Edinburgh (ED) with a role of *VOdmin* role issued by Glasgow will be able to perform select, insert or update on resources identified with a *vulnerability*. The second row in the table 2 implies that *any one* that presents a *Site-administrator* role issued by Edinburgh will have partial privileges of a Site-administrator at Glasgow, i.e. they are able to perform certain operations such as select or insert on Glasgow resources identified with a *vulnerability*. The policy depicted in Figure 8 defines access control for *vulnerability* objects (resources). The rules defined in this policy contain the logic of who can access the vulnerability resource and what they can do with it.

6. Conclusion and Future Work

In this paper we have presented the architecture (ACVAS) of an integrated security framework that is cognisant of security and incorporates the underlying fabric level security. This encompasses site-specific and VO-specific policy specification for fabric security including security monitoring (patch status monitoring) and vulnerability scanning and subsequent updating with the needs of the VO itself. We believe that tool support can be used to identify and assess potential vulnerabilities in a VO, before they are exploited. However, we also note that there are several issues with regards to policy specification for vulnerability information and applying patches to affected software. For instance, should/will the collaborating sites disclose their vulnerability status of resources and/or patch status of target site resources in a VO environment with other collaborating sites? Sites are independent will typically not be dictated to with regard to joint policy specification on disclosure of any vulnerability information across the federation.

7. References

1. *JISC Virtual Research Environments programme*, http://www.jisc.ac.uk/whatwedo/programmes/vre1.aspx .
2. Chadwick, D.W. and A. Otenko, *The PERMIS X. 509 role based privilege management infrastructure.* Future Generation Computer Systems, 2003. **19**(2): p. 277-289.
3. Alfieri, R., et al. *VOMS, an authorization system for virtual organizations*. in *First European Across Grids Conference*. 2003.
4. Lorch, M., et al., *First experiences using XACML for access control in distributed systems*, in *Proceedings of the 2003 ACM workshop on XML security*. 2003, ACM: Fairfax, Virginia. p. 25-37.
5. Anderson, A. *SAML 2.0 profile of XACML*, . 2004.
6. *Internet2. Internet Shibboleth Technology*, http://shibboleth.internet2.edu/. 2009
7. Sinnott, R.O., et al., *Advanced security for virtual organizations: The pros and cons of centralized vs decentralized security models*. 2008. p. 106-113.
8. Power, R. and C.S. Institute, *2001 CSI/FBI Computer Crime and Security Survey*. 2001: Computer Security Institute.
9. *Grid Site Monitoring* 2005.
10. *Grid Security Monitoring*. 2008.
11. Muncaster, P. *Google hack-attack code hits the web*, http://www.securecomputing.net.au/News/164937,google-hack-attack-code-hits-the-web.aspx 2010 June 2012].
12. Kurtz, G. *Aurora Exploit in Google Attack Now Public*, http://blogs.mcafee.com/corporate/cto/dealing-with-operation-aurora-related-attacks. 2010 June 2012].
13. Prince, K., *Malicious Software Defense: Have We Moved Beyond Anti-Virus and Spyware Protection Software?* 2007, Perimeter eSecurity.
14. Shostack, A., *Quantifying Patch Management.* Secure Business Quarterly, 2003. **III**(2).
15. Stirparo, P., M.A. Shibli, and S. Muftic. *Vulnerability analysis and patches management using secure mobile agents*. in *Advanced Communication Technology, 2009. ICACT 2009. 11th International Conference on*. 2009.
16. *Microsoft SMS*, http://www.microsoft.com/smserver/default.mspx.
17. *Pakiti: A Patching Status Monitoring Tool*.
18. *EGEE Operational Security Coordination Team (OSCT)*, http://osct.web.cern.ch/osct/.
19. *Yum-Package Manager--*http://yum.baseurl.org/.
20. *apt-get --*http://www.apt-get.org/.
21. Laboratories, S., Temasek, R.H.C. Yap, and L. Zhong. *A Machine-Oriented Vulnerability Database for*

Automated Vulnerability Detection and Processing. in *Proceedings of the 18th USENIX conference on System administration*. 2004. Berkeley, CA, USA: USENIX Association.
22. Roberge, M.W., T. Bergeron, and Robert, *Introduction to OVAL: A new language to determine the presence of software vulnerabilities*. 2003.
23. *Common vulnerabilities and exposures list (CVE)*. 2011, http://cve.mitre.org/cve/.
24. *cURL: http://curl.haxx.se*.
25. *CFengine Web site: http://www.cfengine.org*.
26. Damianou, N., et al. *The Ponder Policy Specification Language*. in *Policy 2001: Workshop on Policies for Distributed Systems and Networks*. 2001. Bristol, UK: Springer-Verlag LNCS.
27. Ou, X., S. Govindavajhala, and A.W. Appel, *MulVAL: a logic-based network security analyzer*, in *Proceedings of the 14th conference on USENIX Security Symposium - Volume 14*. 2005, USENIX Association: Baltimore, MD.
28. Ajayi, O., R. Sinnott, and A. Stell, *Dynamic trust negotiation for flexible e-health collaborations*, in *Proceedings of the 15th ACM Mardi Gras conference: From lightweight mash-ups to lambda grids: Understanding the spectrum of distributed computing requirements, applications, tools, infrastructures, interoperability, and the incremental adoption of key capabilities*. 2008, ACM: Baton Rouge, Louisiana. p. 1-7.
29. Ajayi, O., *Dynamic Trust Negotiation for Decentralised e-Health Collaborations*. 2009, University of Glasgow.

```
<Policy PolicyId="site_specific_policy"
  xmlns="urn:oasis:names:tc:xacml:2.0:policy"  . . . . . .
  RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:first-applicable">
  <Target>
    <Resources>
      <Resource>
        <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">InstalledPackages
                </AttributeValue>
          <ResourceAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:
                resource:resource-id" DataType="http://www.w3.org/2001/XMLSchema#string" />
        </ResourceMatch>
      </Resource>
    </Resources>
  </Target>
  <Rule Effect="Permit" RuleId="SiteadminFullAccess">
    <Description>A Site Admin can access InstalledPackages</Description>
    <Condition>
      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only" />
        <SubjectAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:2.0:attribute:
                role" DataType="http://www.w3.org/2001/XMLSchema#string" />
      </Apply>
    </Condition>
  </Rule>
  <Rule Effect="Permit" RuleId="VOUserFullAccess">
    <Description>Jan can Access InstalledPackages</Description>
    <Condition> FunctionId="urn:oasis:names:tc:xacml:1.0:function:x500Name-match">
       <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:x500Name-one-and-only">
   <SubjectAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:subject:
                subject-id" DataType="urn:oasis:names:tc:xacml:1.0:data-type:x500Name" />
        </Apply>
        <AttributeValue DataType="urn:oasis:names:tc:xacml:1.0:data-type:x500Name">
                CN=Jan Muhammad</AttributeValue>
     </Condition>
  </Rule>
  <Rule Effect="Deny" RuleId="DenyOtherwise" />
</Policy>
```

Fig. 7. A site specific policy protecting '*InstalledPackages'* resource

Table 1. A site specific policy protecting '*InstalledPackages'*

Authorisation Function	**Description**
has_account (p, h, a)	implies that principal *p* has an account with the name of a user account *a* on host *h* that the principal can access.
accessFile(p,h, A,Path)	implies that entity *p* can *access* a file located on host *h* with a given path *A.Path*
hacl(S, Dest, Pro, DestPort).	implies that source host *S* can reach destination host *Dest* through protocol *Pro* via destination port *DestPort*
netAccess(p,src,des,Pro, port)	implies that entity *p* on source host *src* can send network packets from source port *port* to destination host *des* using protocol Pro
exec_code (p, h, r)	implies that principal *p* can execute code with privilege *r* on host *h*.
vulexist (h, id, p)	implies that a host *h* has vulnerability *id* caused by program *p*

```
<Policy PolicyId="vo_specific_policy"
  xmlns="urn:oasis:names:tc:xacml:2.0:policy"  . . . . . .
  RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:first-applicable">
  <Target>
    <Resources>
      <Resource>
        <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">vulnerability
               </AttributeValue>
          <ResourceAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:
               resource:resource-id" DataType="http://www.w3.org/2001/XMLSchema#string" />
        </ResourceMatch>
      </Resource>
    </Resources>
  </Target>
  <Rule Effect="Permit" RuleId="SiteadminFullAccess">
    <Description>A Site Admin can access vulnerability</Description>
    <Condition>
      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only" />
        <SubjectAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:2.0:attribute:
               role" DataType="http://www.w3.org/2001/XMLSchema#string" />
      </Apply>
    </Condition>
  </Rule>
  <Rule Effect="Permit" RuleId="VOUserFullAccess">
    <Description>John can Access Vulnerability</Description>
    <Condition> FunctionId="urn:oasis:names:tc:xacml:1.0:function:x500Name-match">
       <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:x500Name-one-and-only">
  <SubjectAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:subject:
               subject-id" DataType="urn:oasis:names:tc:xacml:1.0:data-type:x500Name" />
        </Apply>
       <AttributeValue DataType="urn:oasis:names:tc:xacml:1.0:data-type:x500Name">
               CN=John Boyd</AttributeValue>
     </Condition>
  </Rule>
  <Rule Effect="Deny" RuleId="DenyOtherwise" />
</Policy>
```

Fig. 8. A VO specific policy protecting '*Vulnerability'* resource