Exploring the Automatic Identification and Resolution of Software Vulnerabilities in Grid-based Environments

Jan Muhammad

Submitted in fulfilment of the requirements for the
Degree of Doctor of Philosophy
School of Computing Science
College of Science and Engineering
University of Glasgow

June 2013

Abstract

Security breaches occur due to system vulnerabilities with numerous reasons including; erroneous design (human errors), management or implementation errors. Vulnerabilities are the weaknesses that allow an attacker to violate the integrity of a system. To address this, system administrators and security professionals typically employ tools to determine the existence of vulerabilities. Security breaches can be dealt with through reactive or proactive methods. Reactive approaches are passive, in which when a breach occurs, site administrators respond to provide damage control, tracking down how the attacker got in, resolving the vulnerability and fixing the system. On the other hand, proactive approaches preemptively discover and fix vulnerabilities in their systems and networks before attacks can occur. For many research and business areas, organizations need to collaborate with peers by sharing their resources (storage servers, clusters, databases etc). This is often achieved through formation of Virtual Organisations (VO). For successful operation of such endeavors, security is a key issue and system configuration is vital. A faulty or incomplete configuration of a given site can cause hinderances to their normal operation and indeed be a threat to the whole VO. Management of such infrastructures is complex since they should ideally address the overall configuration and management of a dynamic set of VO-specific resources across multiple sites, as well as configuration and management of the underlying infrastructure upon which the VO exists - referred to in this thesis as the fabric.

This thesis investigates the feasibility of using a proactive approach towards detecting vulnerabilities across VO resources. First, it investigates whether vulnerability assessment tools can preemptively help in detecting fabric level weaknesses. Then it explores how the combination of advanced authorisation infrastructures with configuration management tools can allow distributed site administrators to address the challenges associated with vulnerabilities. The primary contribution of this work is a novel approach for vulnerability management which addresses the specific challenges facing VO-wide security and incorporation of fabric management security considerations.

Acknowledgements

I would like to express my deep gratitude to my supervisors, Prof. Richard O. Sinnott and Prof. Chris W. Johnson for their support and guidance throughout this research. Their suggestions, criticisms and frequent encouragement have been pivotal in improving this dissertation. My utmost gratitude also goes to Prof. Sinnott who has been my main supervisor for his major feedback into this research for the crucial three years before he left the University of Glasgow, and for his exceptional support, inspiring guidance and constant encouragement. His constructive comments and insightful reviews have enabled me to complete this thesis. I have been fortunate enough to learn from the vast experiences he has for conducting real life research in multiple inter-disciplinary fields of computer science and e-Science and e-Research application areas.

I am indebted to the Higher Education Commission (HEC), Government of Pakistan, through the Faculty Development Program for granting me a PhD scholarship and to my home university Balochistan University of Information Technology, Engineering and Management Sciences (BUITEMS) for the study fellowship that allowed me to pursue my further studies at doctoral level.

I must also express my sincere gratitude to my colleagues and staff members in the School of Computing Science at the University of Glasgow, in particular the Glasgow Security (GLASS) research group for their constructive and insightful comments during my presentations. I would particularly like to thank Dr Femi Ajayi, Dr. John Watt, Mr. Chris Bayliss, Sardar Hussain and Sulman Sarwar for their help in setting up my experiment requirements and their office light and witty chats.

Thanks to my friend Mr. Khalid Mengal at the University of Bremen, Germany, with whom I have collaborated and co-authored research papers; for his initial help on using Xen and other simulation softwares, and the constant sharing of experiences in using the simulators and associated research discussion. I am also grateful to my caring friends here in the UK and back home for their friendship and encouragement during my time at Glasgow University.

Finally, I am eternally grateful to my parents and family whose love, continuous support and encouragement from a distant land were key motivating factors for the completion of this work. My dearest gratitude also goes to my wife Naz and children (Aazain, Humza and Haris) for their unconditional love, support, motivation and patience. I also like to thank my family friends (Karamat Bhutta and Aapi Tasneem) for their unconditional love and support to my family while I was busy in my research.

Dedications

I dedicate this thesis to my elder son Muhammad Humza.

Contents

List of Figures

List of Tables

Glossary

AA	Attribute Authority
ABAC	Attribute-based access-control
ACL	Access Control List
ACVAS	Automated Configuration & Vulnerability Accessment System
API	Application Programming Interface
BCFG	Bee-ConFig
CA	Certificate Authority
CE	Computer Element
CERT/CC	Computer Emergency Response Team/Coordination Center
CM	Configuration Management
CMS	Configuration Management System
CVE	Common Vulnerabilities Exposure
DHCP	Dynamic Host Configuration Protocol
DN	Distinguished Name
EGEE	Enabling Grids for E-SciencE
EGI	European Grid Infrastructure
GSSVA	Grid Site Software Vulnerability Analyzer
GT	Globus Toolkit
HaaS	Hardware as a Service
IaaS	Infrastructure as a Service
ISS	Internet Security Scanner
MPI	Message Passing Interface
MulVAL	Multi-host,Multi-stage Vulnerability Assessment Language
NIST	National Institute of Standards and Technology
NGS	National Grid Service
NVD	National Vulnerability Database
OSG	Open Science Grid
OSVDB	Open Source Vulnerability Database
OVAL	Open Vulnerability Assessment Language
PaaS	Platform as a Service
PDP	Policy Decision Point
PEP	Policy Enforcement Point
SaaS	Software as a Service
SOAP	Simple Object Access Protocol
VO	Virtual Organisation
VOBOX	Virtual Box Service
VOMS	Virtual Organisation Membership Service
VPM	Vulnerability and Patch Module
VulMIS	Vulnerability Management Information System
WSDL	Web Services Description Language
WSUS	Windows System Update Server
XML	eXtended Markup Language
XACML	eXtended Access Control Markup Language

Chapter 1

Introduction

In order to meet business challenges for service provision, organisations add more and more hosts, both physical and virtual to meet consumer demand. As a result system administrators must spend increasing time ensuring all hosts comply with security policy. Whilst a 'normal' user with a consumer-grade personal computer, needs to configure it prior to use by providing a user name, a computer name and often automatic patch options for Microsoft Windows products; the job of a network/system administrator is much more complex and often demands setup of different configuration and security options in a organisational network for a increasingly heterogenous collection of hardware, software systems and the networks they utilise. The past decade has seen organisations around the world becoming increasingly interconnected with the amount of critical and sensitive information being stored on networks and systems increasing rapidly. The profileration of wireless, handheld devices and notebooks (mobiles, PDAs, iPads etc) have also encouraged users to store critical information on them for convenience and use them to access centralised organisational systems. Hence it is vital that systems, networks and devices that store information are configured as securely as possible to reduce the likelihood of any compromises.

In this 'networked-world', distributed organisations, humans, computer applications and devices often need to be interconnected to undertake joint collaborations that would be impossible without contributions and resources of multiple collaborators.

A recent survey by the AT&T corporation [12] presents an analysis and projection showing the increase in the number of business collaborators expected in the near future. Other similar surveys [13, 14, 15] back this trend and predict a significant increase in virtual organisations (VOs) and eCollaborations in the forthcoming years.

In such distributed environments security and system configuration play a vital role. This is especially so given that any inappropriate or faulty configuration can cause hindrances to normal operation and delivery of services existing on collaborating infrastructures. Network misconfigurations can potentially cause reachability failure, security violations and introduction of a range of vulnerabilities. In a study on network misconfigurations [16], Alimi et al found that more than 62% of network failures are due to configuration errors. Oppenheimer et al. [17] also found that *"inadequate and inaccurate understanding of system configuration may cause service unavailability expected from these services"*.

However, it is not always the case that these failures are caused only by configuration inaccuracies. For example it is often the case that failures arise when an operator/system administrator wants to add/remove a user/node from a network, establish a new configuration, shift a particular service from one server to another, or even to change/delete a particular file. Similarly, there may be cases when several developers/collaborators working on a joint project need to make system configuration modifications. If a particular system administrator and/or collaborator deletes/changes a file perceiving it as unnecessary this may cause the whole collaboration to fail due to unforeseen side-effects caused by dependencies on this file. Tackling this requires novel solutions that support dynamic and often ad-hoc collaborations in a secure and adaptable manner.

Vulnerability management is another area closely related to configuration management. Possible security-threats often arise due to insufficient safeguarding mechanisms or flaws in technology, allowing an attacker to find a situation to gain unauthorised access to a system and to misuse it. Vulnerability management is the process of monitoring, quantifying and responding to those flaws. VA is a proactive methodology to keep organisational security teams alert of any possible danger prior to the occurence of an attack. Ideally VA and configuration management should be closely coupled to improve the overall security of VOs.

e-Science and e-Research is about supporting collaborations especially those that cross administrative boundaries. One way that this has been addressed is through the notion of virtual organisations (VOs). Foster et. al defines a VO as: "*A set of individuals and/or institutions defined by such sharing rules from what we call a virtual organization (VO)*" [18]. VOs come into existence when one or more institutions and their research personnel wish to collaborate and share resources. VOs are typically realized through the application of Grid technologies and middleware. For large-scale Grids involving multi-site collaborations, the installation and configuration of the software infrastructure needs for particular VOs becomes increasingly demanding and complex for local system administrators at given sites since software targeted to the needs of specific VOs often comes from a variety of sources including remote collaborators themselves. Detailed knowledge of these software tools and their often prototypic nature mean that local administrators are left with a variety of potentially conflicting software and software configuration requirements. This is especially the case when site resources are shared across multiple VOs.

Whilst software vendors such as Microsoft and Sophos generally release patches to fix vulnerabilities in their software. Patching heterogeneous software from multiple providers where dependencies can exist between the code bases is much harder to achieve. Patches, if applied correctly can remove vulnerabilities from systems, however in some cases patches may break existing systems if not properly analysed and tested before applying. Furthermore, if applying patches is seen as the 'cure-all' to software vulnerabilities, the question arises as to why organisations do not apply them as soon as they are made available? The problem is that patch management is multifaceted, and there are often operational reasons why organisations do not apply patches immediately. 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. Other problems with traditional patching systems include software set for manual updates which never actually occur; 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 etc. Many of these issues are discussed in [19].

As one example, in November 2008, a worm (known as Conficker or Downadup) swept through millions of servers and workstations in corporate, educational and public computer networks around the world [20]. The Conficker worm exploited a vulnerability in Microsoft Windows, by discovering network passwords and hand-carried user gadgets like USB keys. Conficker was the worst infection since the Slammer worm reported in January 2003 infecting millions of personal computers around the world. Figure 1.1 shows an illustration of the situation that gave rise to the conficker worm [1]; but this model is also common to many other vulnerabilities.

Worms like Conficker can spread across the Internet at rapid speed creating alliances of infected computers often called *botnets* which can then accept programming instructions from their undisclosed controllers. Since the majority of novice computer users may not notice that their machines have been infected, the impact and threat of these compromised machines will only be discovered when attacks occur. As such, the severity of such worms is very high as they may operate in the background, using infected computers to send spam or infect other computers, and perform unauthorised actions such as stealing a user's personal information. Although, vendors like Microsoft rushed an emergency patch to defend the Windows operating systems against this vulnerability in October, the worm continued to spread even as the level of warnings grew. In 2009, French airforce fighter planes were put down by a similar computer worm [21]. Similarly, it is estimated that about 30 percent of Windows-based

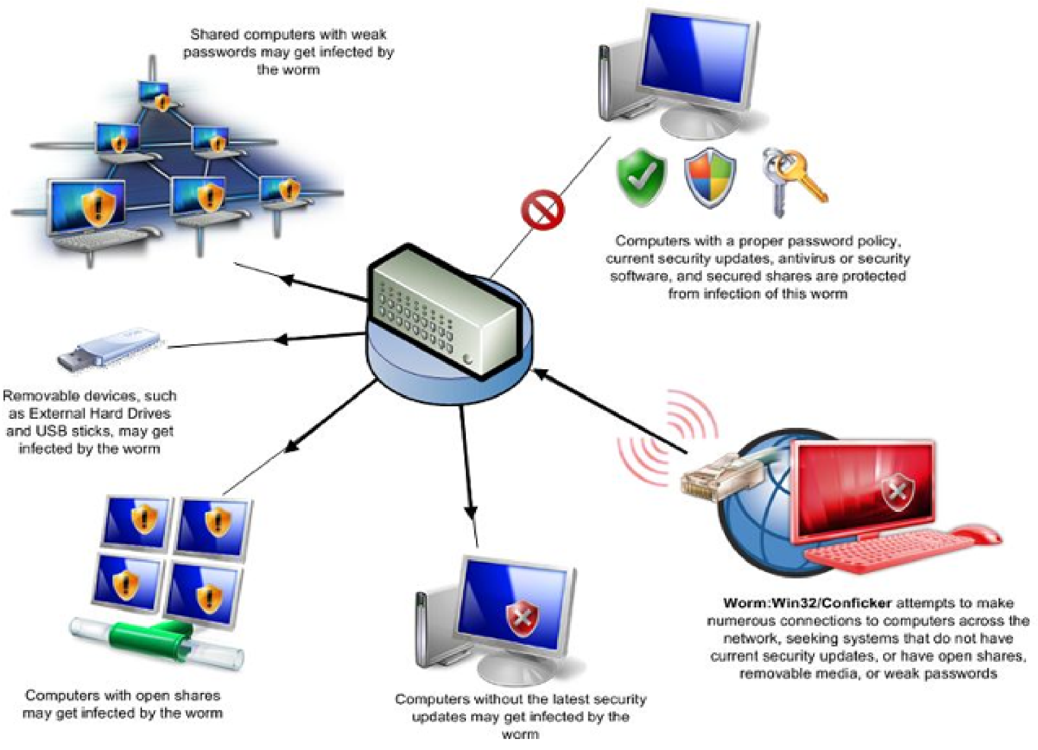

Shared computers with weak passwords may get infected by the worm
Computers with a proper password policy, current security updates, antivirus or security software, and secured shares are protected from infection of this worm
Removable devices, such as External Hard Drives and USB sticks, may get infected by the worm
Worm:Win32/Conficker attempts to make numerous connections to computers across the network, seeking systems that do not have current security updates, or have open shares, removable media, or weak passwords
Computers with open shares may get infected by the worm
Computers without the latest security updates may get infected by the worm

Figure 1.1: How conficker works [1]

computers attached to the Internet still remain vulnerable to infection as they have not been updated with the appropriate patches, despite the fact that these patches have been made available. As such the prevalence of Conficker was due in part to lax security practices by both companies and individuals in not immediately installing updates. It is thus clear that many systems are simply left un-patched for months, even years.

According to the Computer Emergency Response Team / Coordination Center (CERT/CC) [22], around 95 percent of security breaches could be prevented by keeping systems up-to-date with appropriate patches [23]. This lax security process is simply not tenable for many security-oriented VOs. This problem is exacerbated where a single inadequately configured VO resource can endanger all other sites within a given VO if it does not take all appropriate measures to ensure its own configuration and security reflecting both its own specific (autonomous) site policy and the agreements set in place for the VO itself. Whilst advanced authorization infrastructures can protect access to given resources, these security mechanisms will be made redundant if the basic underlying fabric upon which those services exist are themselves inadequately protected. This might be from numerous perspectives: firewalls that have been left open or incorrectly configured; out of date anti-virus software; operating systems or middleware itself that have not been patched with latest updates to name but a few examples.

Ensuring that all nodes used across a VO have the necessary operating system, middleware and that the underlying fabrics are properly patched with the most up to date antivirus software protection is something that up until now has been left to the individual VO sites. In many cases however this is something that cannot be left to chance. It needs to be managed at the VO level. For example, a VO dealing with medical records or any other highly sensitive data sets demands that resources are protected as far as possible. Middleware solutions up to now have been primarily targeted at isolated aspects of security, e.g. defining and enforcing advanced authorization infrastructures which can be used to protect access to particular services or data. In this thesis it is argued that a proactive and more integrated solution is needed that deals with end-end VO-wide security including VO-wide security policies and their enforcement which are aligned with local security policies and supporting the security of the underlying fabric upon which that VO is based. To address this requires an understanding of local and global VO policies and the ability to securely deliver and configure software patches and perform local site configurations. One way in which such updates and configuration can be supported is through configuration management tools. However these have to now primarily focused on a single domain and not tackled inter-organisational collaborations. Thus a typical assumption is that the person using the configuration management tool is the local systems administrator. This is often not the case in inter-organisational collaborations, where remote collaborators may want or need to install patches on a given remote VO resource, especially if it has been identified as compromised (subject to the local security policy allowing remote administrators to install/patch their local resource).

1.1 Thesis Statement

Configuration and vulnerability management of resources across potentially geographically far-flung administrative boundaries is difficult to achieve. This is especially so with inter-organisational collaborations. I assert that it is possible to realise and establish a configuration aware, security-oriented vulnerability assessment model (through trusted credentials) in a decentralised multi-domain collaborative environment to support improved end-to-end security. This type of model supports decentralised access control for various autonomous and heterogeneous organisations and helps to keep track of system/site policies for configuration; for monitoring of resources; for dynamic updates and patches of VO-specific operating systems or third party software including antivirus software. I also assert that by adopting an integrated approach to policy-driven vulnerability management that allows seamless and transparent establishment of trusted virtual collaborations it is possible to improve the overall end-end security of such endeavours.

1.2 Key Research Question

Can we proactively identify and resolve security vulnerabilities and system mis-configurations in an automated and generic way in a Grid-based environment?

In an attempt to answer the above question several sub-questions were raised:

- ***How do we identify the version of installed software in a multi-node Grid environment?***

We want to validate the software that is deployed on a Grid resource before vulnerabilities arise. Increasingly important is the configuration of the software. Information about the existence of potential vulnerability is required for site administrators to able to resolve them in a timely manner. For large-scale Grids involving multi-site collaborations, identifying software vulnerabilities in the fabric upon which services are provisioned is increasingly demanding and complex for local system administrators since the software targeted to the needs of specific VOs often comes from a variety of sources including remote collaborators themselves and might contain vulnerabilities.

- ***How do we verify that the version information is reliable?***

Detailed knowledge of installed software along with their versioning information and proper configuration is vital to successful operation of collaborations based on Grid technologies. Ideally it should be possible to authenticate the signatures on software before it is deployed to ensure that it comes from a reliable source and/or has been verified by a VO administrator prior.

- ***How do we automatically verify if versions are vulnerable?***

Ideally it should be possible to apply "controlled"/authorised use of vulnerability scanners by a VO admin to see if a given site has a vulnerability and use configuration/patch management tools to address the vulnerability, e.g. securely patch all nodes affected etc. The problem of vulnerabilities is exacerbated in a Grid environment due to their dynamic nature.

- ***How do we ensure only trustworthy (authentic) patches are applied?***

An authentic patch implies a verifiable and un-tampered patch. Typically this is achieved through digitally signing patches and using their signature as a basis for determining their trustworthiness. Trust-based patch delivery and installation is successfully realised when a system or VO administrator successfully installs authentic patches across inter-organisational VO resources.

- ***How do we automate VO-wide vulnerability assessment without compromising VO nodes/sites?***

An authorised Vulnerability Assessment process implies that only authorised remote system administrators are able to invoke VO-specific vulnerability assessment tools. It is essential that only trusted remote system administrators are able to conduct security (automated vulnerability assessments), deploy patches and configure resources. This demands that they are in possession of VO-specific authentic and valid credentials that have been agreed as part of the establishment of the VO since all VOs are implicitly autonomous. This in turn implies that sites have security policies in place that define and enforce the VO-specific vulnerability assessment and resolution processes.

1.3 Motivation

In large-scale multi-domain collaborations typified by Grids such as the European Grid Infrastructure (EGI) [24] (formerly known as Enabling Grids for E-sciencE (EGEE) [25]) and Open Science Grid (OSG) [26], middleware and other application software are typically installed by local site/system administrators - hereafter called (site administrators) via customised and site-specific configuration management tools. These include technologies such as CFengine [27] and Quattor [28]. These configuration management tools help site administrators manage site resources. This typically entails, installation and removal and configuration of, Grid-based (VO-specific) software. However in many collaborative projects, installation and configuration of multiple site resources is required. VO managers can be designated with privileges to manage software for a specific VO on a per VO/site basis, however the actual local installation and configuration of such software largely remains the task of local site administrators. This model has a variety of issues as outlined – especially where the software installed and configured at a given site requires patching due to an identified vulnerability which can threaten the whole multi-site collaboration.

Due to the complex nature of multi-domain collaborations, a variety of challenges exist in tackling this issue: researchers may want to execute their own domain specific (VO-specific) application codes across the VO-specific nodes and these applications can be of a variety of forms and/or be unknown in advance. For example, it is quite possible that the application is self-contained and of a modest size and without external dependencies, and that it can be shipped for execution request to a collaborating site's computing resource and removed once it has terminated. In other cases software dependencies might generate problems due to requirements on external packages/libraries and their specific system configuration, e.g. support for environment variables, Message Passing Interface (MPI), compiler versions, etc.

It is equally possible that the execution environment required for a particular application to run successfully can be specified within the job description so that the Grid resource brokers can find the appropriate resources. However, especially in dynamic Grid environments, the schema describing either the resource or the application requirements might not be sufficient to allow the resource broker to correctly match those requirements and the availability of a particular site resource. In this case, sites cannot guarantee the correct execution environment for incoming jobs and applications. In order to tackle these issues, collaborating institutions and resources form VOs. These domains can have varying/conflicting policies/configurations and require special resources to be able to run VO-specific applications. Ideally an appropriate execution environment should be guaranteed to VO users, either by ensuring an appropriate system environment exists, or by making available pre-installed VO specific software applications [29].

1.4 Research Contribution

This thesis introduces a model for an Automated Configuration and Vulnerability Assessment System (ACVAS) that can be used to address vulnerability assessment and patch and configuration management issues that arise in heterogeneous and autonomous VO collaborations. ACVAS proactively detects vulnerabilities and where appropriate provides automated remedies to them either by applying relevant patches and/or triggering configuration management tools to proactively enforce VO-specific security policies. We introduce an architecture, process model and variety of metrics for defining, enforcing and subsequently measuring VO security. We show how this model supports trusted intermediary parties and components which can provide multiple decisions exploiting delegations of rights to help achieve trust-based patch management between collaborating institutions. We also show how the model protects services and access policies and reduces the overall risk whilst resolution of vulnerabilities takes place.

ACVAS delivers patches that are authentic, valid and where appropriate uses intermediary parties and components to satisfy access policies defined by remote sites. ACVAS establish trust routes to push patches between two or more nodes in a VO. Trust-based patch delivery and installation is successfully realised when a system or VO administrator successfully installs patches or delegates his/her digital credentials to trusted intermediary entities who can subsequently push patches across inter-organisational VO resources.

In ACVAS, patches can be installed on remote nodes on the basis of: their digital signatures; local VO policies; VO-wide policies and credentials associated with the collaboration. ACVAS addresses the heterogeneity and autonomy of VO resources and is specifically suited to multi-domain environments typified by e-Science Grids. The model is based on integration of vulnerability assessment and configuration management tools, VO-specific patch management and fine-grained security. ACVAS has been applied on a simulation test-bed based on the Xen virtualization technology where multiple Virtual Machines (VMs) were used to model VO nodes that had vulnerabilities identified and patched based on VO-specific security requirements.

1.5 Supporting Publications

1. J. Muhammad, R.O. Sinnott, Policy-based Vulnerability Assessment for Virtual Organisations, 5th International Conference on Frontiers of Information Technology (CSS 2012), Melbourne, Australia, December 2012.

2. J. Muhammad, R.O. Sinnott, Policy-Driven Patch Management for Distributed Environments, International Conference on Network and System Security (NSS 2009), Gold Coast, Australia, October 2009.

3. R.O. Sinnott, J. Muhammad, Supporting the End-End Security of Virtual Organisations through Policy-based Configuration Management, UK e-Science All Hands Meeting conference, Edinburgh, September 2008.

4. R.O. Sinnott, J. Muhammad, Y. Wu, Deployment of Grids through Integrated Configuration Management, Proceedings of 26th International Conference on Parallel and Distributed Computing and Networks (PDCN), Innsbruck, Austria, February 2008.

1.6 Assumptions

In this research it is assumed that remote system administrators are authenticated by default. That is, our model predominantly focuses on authorization aspects of collaborations, e.g.

where related decisions for assessing vulnerabilities, adding/removing patch updates are defined and enforced. Many models for authentication exist and are well defined in the literature, e.g. Public Key Infrastructures [30] or use of federated access management such as the Internet2 Shibboleth technology [31].

1.7 Thesis Outline

The rest of this thesis is organised as follows. Chapter 2 presents a literature review which provides the foundation necessary for the understanding of the subsequent chapters and to set the context for the work as a whole. It starts with an overview of configuration and patch management problems and describes well-known configuration and patch management approaches that have been proposed for distributed environments. We describe the state of the art in collaborative security that have typified current e-Science efforts. We also present the method of study used and justification for the use of virtualization-based simulation as a means of evaluating the proposed methodology. Chapter 3 highlights the general threats and vulnerabilities related to dynamic distributed environments. It also highlights some of the key use case scenarios used to evaluate the work as a whole. Chapter 4 presents the proposed designs and architecture of the ACVAS framework and outlines its key capabilities. This chapter also introduces the service-oriented architecture (SOA) based vulnerability and patch management process in dynamic distributed environments and investigates the pros and cons of four mainstream approaches to a wide range of security oriented patch management processes. Moreover, this chapter also discusses a hybrid model (i.e. an integrated model for proactive vulnerability assessment and delivery of trusted patches) more suited for trusted patch management. Chapter 5 conducts an analysis of VO specific vulnerability assessment policy definition and enforcement mechanisms. It investigates security oriented vulnerability analysis and patch management issues including signed patches, RPM Package Manager (RPMS) and inter-dependency issues that must often be addressed for trusted patch management. Chapter 6 presents the implementation of the ACVAS framework with the help of a case study and use case scenarios. In Chapter 7, the theoretical evaluation and the pratical validation of the prototype implementation of ACVAS is presented. Finally, Chapter 8 concludes the thesis by summarising the lesson learnt during the course of this research and its contribution as a whole. The chapter finally outlines the conclusion and potential directions for further research work.

Chapter 2

Literature Review and Related Work

This chapter presents a review of background literature in the area of configuration, vulnerability and patch management. It provides an overview of related work associated with vulnerabilities, configuration and patch problems, focusing in particular on large-scale distributed systems and e-Research collaborations. The discussion focuses on VO-specific vulnerability and patch management issues such as identification and resolution of vulnerabilities and attack trends in the near past. It describes the challenges of trust, software dependencies and challenges of existing live-update systems. This chapter lays the ground work for the security-oriented vulnerability management contributions of this thesis. The chapter concludes with reviews of other work or projects that have tried to address vulnerability management problems in the context of collaborative environments such as e-Science and e-Research.

2.1 Configuration Management & Distributed Environments

Bellovin et al [32] identified several problems with regards to challenges for system administrators when an organisational system size grows. Firstly, since many computer users are novice to system management, this results in poorly configured local systems. Secondly, in many organisational setups, configurations are centrally defined and individuals allowed to configure their own local systems, e.g. their choice of software. This can result in misconfiguration with open questions like–did every individual install mandatory organisational applications? If so did they set up automatic update and patching? Thirdly, two different configurations that might be individually correct, may potentially interact in a dangerous way. For example, Java and firewall behavior for the FTP protocol can combine to produce security vulnerabilities [33].

Configuration management can involve reconfiguring every aspect of an entire site. This makes configuration tools an attractive target for malicious attackers. Such tools are themselves designed to be generic. That is, configuration management tools have evolved to allow for the flexible installation and configuration of software across a given set of resources. Typically these work in a closed domain, e.g. a single organisation but more frequently inter-organistional security and configuration management is needed.

2.1.1 Changes within the lifecycle of a Computer

As indicated by Evard in [2], the following represent the machine lifecycle. In Figure 2.1, a computer moves between these states:

- New. A new computer.
- Clean. A computer with the OS installed, but not configured to work in the environment.
- Configured. A computer that is configured correctly according to the requirements of the computing environment.
- Unknown. A computer that has been misconfigured, or has got out of date.
- Off. All done.
- Retired.

As discussed in [2], the interesting part of the Figure 2.1 are the changes that a computer goes through as it moves between states. These are the "processes," and consist of:

Build. In the build process, the operating system is installed on target machine.

Initialise. This process occurs directly after build, and at many sites, it is thought to be part of the build process. The initialisation process consists of modifying the OS image that is required to have the systems operate in the environment. Intialisation processes typically include network configuration, installing OS patches and other changes. At the completion of initialisation process, the system is (theoretically) functional and ready to be used in the computing environment.

Update. After the initialization process, at some point the system might have to be modified. For instance the network configuration might have changed, or a user needs to be added, or an OS security patch needs to be applied, or the computer needs some kind of new

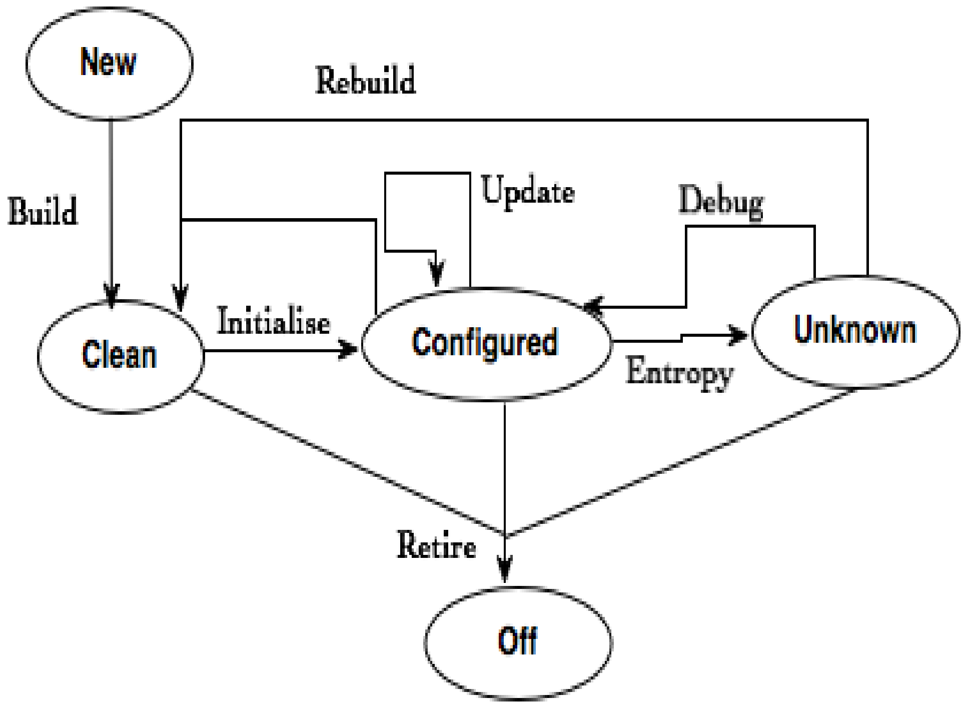


Figure 2.1: The lifecycle of a machine [2]

functionality. Whatever the cause, the computer needs to be updated in order to address the new requirements. In most cases, this will happen continually for the lifetime of the computer.

Entropy. The entropy process refers to the gradual process of change returning from the unknown to a configured state. There could be numerous reasons for this: undisciplined changes made to the machine, major changes in the environment, or other unexplained problems.

Debug. The debug process refers to debugging an "unknown" state of the computer, and getting it back into a known state. This process is usually an intensive (manual activity). In some cases debugging can often involve updating the systems as well.

Rebuild. The rebuild process might be needed in certain cases, either because of some kind of problem or because the changes to be made are so drastric that simple updates are useless and make no sense. A possible scenario would be, when a rebuild is typically done when upgrading from one major revision of an OS to the next one. In that particular case, the rebuild process will be possible by simply reapplying the former two processes, i.e. the build and initialisation processes to the computer.

Retire. This process would mean the end of the lifecycle of the computer. For some sites, there is an official process for doing this, while in others, it would merely involve just turning the system off and/or forgetting it had existed.

This work predominantly focuses on the initialise and update process. However, rather than deal with a simple machine the focus is multiple machines across multiple sites directed through Grids. Integration of configuration management tools with Grid technologies is needed to support large-scale Grids, where VO administrators are able to install and configure VO-specific software across multi-site resources. This is not without challenges however. Perhaps the greatest challenge to overcome is one of security and especially trust. Before any large scale deployment across multiple domains is attempted, it is essential to show that Grid technologies and a variety of configuration management software can be integrated since different sites will likely have their own policies and software for configuration management. This is complicated in that the process of system configuration often takes place in an uncertain environment. For example, one scenario might be when the available hosts or network connections have completely failed, partially failed or are sporadically failing. This requires the configuration management tool to reconfigure and take remedial actions in the overall system to counter such types of failure. These requirements and challenges demand a robust configuration management process able to tackle the above mentioned uncertainties and variety of scenarios. Additionally, it is often difficult to know and/or monitor the configuration changes that may occur during such an uncertain condition—especially when dealing with highly distributed resources.

Similarly as a part of configuration management in Grids, the deployment process is often itself a tedious/laborious and time consuming job. Sometimes configurations set for a particular site/host may not be suitable for another. The vision of the Grid is to coordinate/share resources for a common goal. Refining this goal and ensuring that the VO is set-up to meet the requirements is key to successful configuration management. This can be ensuring that the right software/data is deployed across the right resources, but might also dealing with data privacy and data integrity. In this context, users and organisations may be concerned with the vulnerabilities of collaborating site (nodes) which may be a potential vehicle/threat to themselves.

2.1.2 Configuration Management and Security

Due to the exponential growth of internet usage among the masses such as home users and many modern devices such as iPhones, iPADs and BlackBerrys etc, which all require secure and reliable Internet systems to function correctly. Whilst the research community and private business sector have put forward numerous efforts on developing defence techniques to counter security threats, efficient management and configuration of systems has been largely overlooked [34]. In [32] Bellovin et al highlight three requirements for centralized configuration management vital for successful secure organisational operations: security, database-driven and robustness. Security here implies that the relevant configuration management databases must be protected against unauthorized access or modification; the communication among the managed hosts must be authenticated, integrity-protected, and ideally encrypted. The database-driven aspect of configuration management covers the individual node-level asset inventory details by clearly stating the requirements, e.g. what nodes need what software depending on user or site-wide needs. Robustness covers aspects of individual nodes configuration failures during attempted updates including scenarios when vendor supplied patches may fail to install. If configuration changes are security-related; this could have serious consequences for the overall organisational security. Teo and Zheng [35] view system configuration through five main categories: meta information; hardware; operating system; software and services. Meta information includes information on what the target host actually is e.g. a server, a workstation or a handheld device. Hardware information also plays a vital role in keeping track of configuration updates such as device drivers and hardware (firmware) bugs. Software information helps to categorise particular nodes with vendor-specific list of software and their categories/versions. Operating systems are an extremely crucial part of an individual node hence are treated differently from software. Finally, widely adopted services provided by hosts to external entities such as HTTP or SMTP are recognised as essential when considering configuration management.

Security-sensitive configuration of appliances such as firewalls and filtering routers are important as well since these help in denying access to any 'external enemy' from an organisational network. One of the hardest problems in an organistional environment is configuration mechanics and dealing with network topology and security policies especially in inter-organisational relationships. Complex firewall configurations are known to be error-prone [36], even if fully distributed firewalls are used [37] topology issues remain.

2.1.3 State-of-the-art Configuration Management

The major body of work in system configuration is mainly targeted to the end-user needs/experiences and has largely ignored the tools required by system operators/administrators needed for large-scale system configuration, monitoring, diagnosis and ultimately repair of systems. Whilst configuration management (CM) tools can allow for deployment and management and, of course configuration, of multiple software environments on heterogeneous infrastructures, they typically focus on the installation and configuration of software in a given domain and across a closed set of resources and not with inter-organisational configuration management. Some of the leading configuration management tool sets include: Configuration Engine

(CFengine) [27], SmartFrog [38] and OGSAconfig [39]. Through these tools it is possible to deploy a variety of Grid software, other software (such as Java or any other software) and indeed support data management activities that are commonly required for a particular VO. However they focus on a single administrator and not on VO-specific administrator(s) usage to address VO-specific issues.

Anderson [40] compares system configuration problems with the early days of computer programming. For example, due to incompatible hardware, changing platforms often requires significant investment of time and resources and there is often no possibility to share code between machines/systems without considerable rewriting/porting. Many early programs were created in an unstructured way. However with the advent of high-level languages, with their underlying theory and portable compilers, this situation has now improved. We now have the same situation in the system configuration field, with a real need for new tools, based on sound theory and targeted at a much higher level of configuration description. As programming language development require new generation of specialists and approaches, it seems likely that real progress in configuration management tools is not possible without a comparable development, requiring specialists with a fair understanding of theory, software development and system administration practice. As was the case with development of programming languages, new approaches for system configuration—especially crossing local and remote IT systems, will demand a significant change in approach from system administrators [41]. Anderson [40] also identified a number of complicating factors in finding a suitable configuration: managing relationships, distributed systems, usability autonomics, uncertainty and last but not least security. By global adoption of configuration management tools there can be dramatic reduction/change on the time spent by performing repetitive system administration tasks [42].

With efficient configuration tools in hand, more time can be spent on improving system/network management capabilities to help to improve service provision to clients/users. The Bfcg2 [43] model has been used to replace the typical scripting method of fundamental unit of automation for system administration tasks since the scripting a multi-node environment is fault-prone and results in difficult error handling. Furthermore, due to heterogeneous local topologies, information is often hardcoded thus preventing scripts being ported across multiple sites. Problems also arise when operators perform tasks incorrectly, e.g. if they are unaware of the existing system configuration. While diagnosing a problem, an operator must often understand both the existing system configuration and the history of system configuration before the problem began [17].

After arguing the importance of understanding configuration in maintaining system availability, [17] also suggest that building operator tools that interface with existing systems provides an incremental path to improving maintainability. It has also been argued that such tools should not be mere wrappers of command-line installations and configuration utilities as typically used in system administration tools. Tools supporting current and past configuration information and predicting the global impact of changes in component configurations are needed. Because as large-scale VO collaborations are operated by distributed teams of site and VO administrators and depend on coordination among those teams with the VO end users, frameworks are needed for VO-specific policy definition and enforcement. In establishing a given configuration or diagnosing a problem, many distinct stakeholders may be involved. Operators can benefit for example from tools that track configuration history and the VO-wide system health over time.

Desai et. al. [42] count three factors that affect an administrator's ability to maintain pace with the continuous demand for configuration changes: efficient deployment of changes; control over how and when new changes are used and deployed, and the ability to understand patterns in configuration changes and their propagation. Desai et. al. used Bfcg2 [43] to cope with frequent changes in several environments in two key steps: tight integration with a repository under revision control and a feedback mechanism which allows configuration management tools to detect entry of clients into particular states.

Several widely adopted configuration management tools now exist. As noted, CFengine [27] is an open source tool widely used for policy host configuration systems. It includes a high level policy language and a low level logic engine. At start-up, CFengine retrieves a policy definition from a central place (server) and then tries to adapt the local system to that policy. The *cfenvd* daemon which is part of CFengine collects local system statistics in order to detect abnormal behaviour.

The Local ConFiGuration system (LCFG) [44] was originally designed by Paul Anderson developed around the same time as CFengine and has provided a test-bed for research into system configuration challenges scaling to over 1,000 diverse machines. LCFG compared to CFengine is difficult to adopt and less portable, consequently requiring heavy initial investment by sites. Similarly, the configuration management process involves scalability issues when the number of nodes increases with the number of system administrators with varying roles and of course the discrete nature of configuration. To address these issues, Desai et. al. [45] used BCFG as a symbolic configuration management tool for heterogeneous cluster environments. Although the BCFG solution helps in detecting client configuration changes, it lacks several important features including detecting changes when there are several collaborating sites (nodes) with varying policies based on their specific VO involvements.

Anderson et. al [46] explored an integrated framework consisting of LCFG and SmartFrog for automatic reconfiguration of larger infrastructures. The LCFG architecture has the particular advantage of having a single source of configuration information by making sure the availability of complete configuration information which allows complete sites to be reconstructed from scratch. The dynamic elements of configuration are provided by SmartFrog framework.

There are several other tools available which provide dynamic reconfiguration according to central policy; for instance Dynamic Host Configuration Protocol (DHCP) helps in dynamically configuring IP network addresses within a range of IPs according to the policy embedded in the server configuration in the network. The IETF Zeroconf tool [47] aims to provide various dynamic configuration network parameters including allocating IP addresses without the need of a DHCP server, translating names/IPs without DNS server etc. However, large-scale infrastructures need to define a common policy within which several of these autonomous tools can operate. The report on the GridWeaver [48] project includes a thorough survey of existing system configuration tools. The comprehensive list of these tools shows that very few of these tools support a clear description of the desired configuration of the underlying fabric and none of these have the ability to specify a higher-level policy about the overall configuration and enforcement mechanisms.

2.2 Vulnerability Management

Vulnerability management is an important foundational component of any organisation's security program. The overall process of network scanning and discovery, identification of vulnerabilities, assessment, patching (in case of missing updates) and mitigation is referred to as vulnerability management. For securing organisational network assets, conventional system administrators and security professionals use firewalls, vulnerability scanners and intrusion detection systems. Each of these systems can be used independently or in conjunction with other tools. However it is the case that these have not proved to be a fool-proof mechanism to protect network assets during normal operations of organisations [49, 50]. Though devices like firewalls provide shielding to networks by implementing rules to restrict network connectivity, they often suffer from the inherent weakness of not understanding the data going through legitimate connections, which can potentially exploit vulnerabilities on the corresponding servers [49].

Based on the discussion in the previous section, due to the complex nature of software, vulnerabilities have and are always likely to exist, despite efforts being put forward by industry and the research community. Often these vulnerabilities result from the mistakes of human programmers.

In this section we provide an analysis of the trends and discuss statistics of these vulnerabilities. The normal response to vulnerabilities is the application of patches, hence patching trends follow cycles of vulnerability. Understanding vulnerabilities by security teams allows better decisions about when and how patches should be deployed. The consequences of these vulnerability trends, e.g. a recent worm attacks such as Stuxnet [51, 52] or Flame [53, 54] usually appear in the introduction to most of the papers on vulnerability and patch management. Examples of this are major worm attacks which have the effect of motivating the security community to action. For instance, the CERT/CC was formed in response to the Morris worm in 1988. Much of the other work into patch management emerged after the worm outbreaks of 2001 [55].

Peltier defines vulnerability as a state of a missing or ineffectively administered safeguard or control that allows a threat to occur with a greater impact or frequency or both [56]. Arbaugh defines it as a flaw or defect in a technology or its deployment that produces an exploitable weakness in a system [57] and makes it susceptible to attack (Wahlström 2005). Based on these definitions, vulnerability can be defined as "a situation that someone or something may exploit to unlawfully access a system and potentially misuse it".

Seacord et.al [3] define vulnerability as: "A security flaw is a defect in a software application or component that when combined with the necessary conditions, can lead to a vulnerability". Hence a vulnerability is a set of preconditions which when they become true can allow an explicit or implicit violation of security policies. There are many sources of vulnerabilities: mistakes during software coding; incomplete system logic or incorrect use of a function or command [58]. Figure 2.2 shows the relationship among software vulnerabilities from Seacord-Householder [3].

Vulnerability management is the process of identifying, monitoring, and responding to vulnerabilities. This is especially challenging since vulnerabilities in a released software product are often not managed risks that the product manufacturer has envisaged or even has a clear understanding of. Indeed the research community is now agreed on the fact that they (vulnerabilities) are unknowns, and the liability for managing these risks often falls to the customer. It is thus the customers' responsibility to identify the vulnerabilities affecting them, in order that better risk management decisions can be made. It is the case that vulnerabilities are increasing; are being exploited more often; and the time to exploit vulnerabilities is shrinking.

Vulnerability Assessment (VA) is the audit process launched against organisational environments (systems, devices and applications). VA compares the current security state of the organsational security level and reveals potential issues and feeds the results into security compliance systems such as security policy and patch management for final evaluation of security procedures already in use.

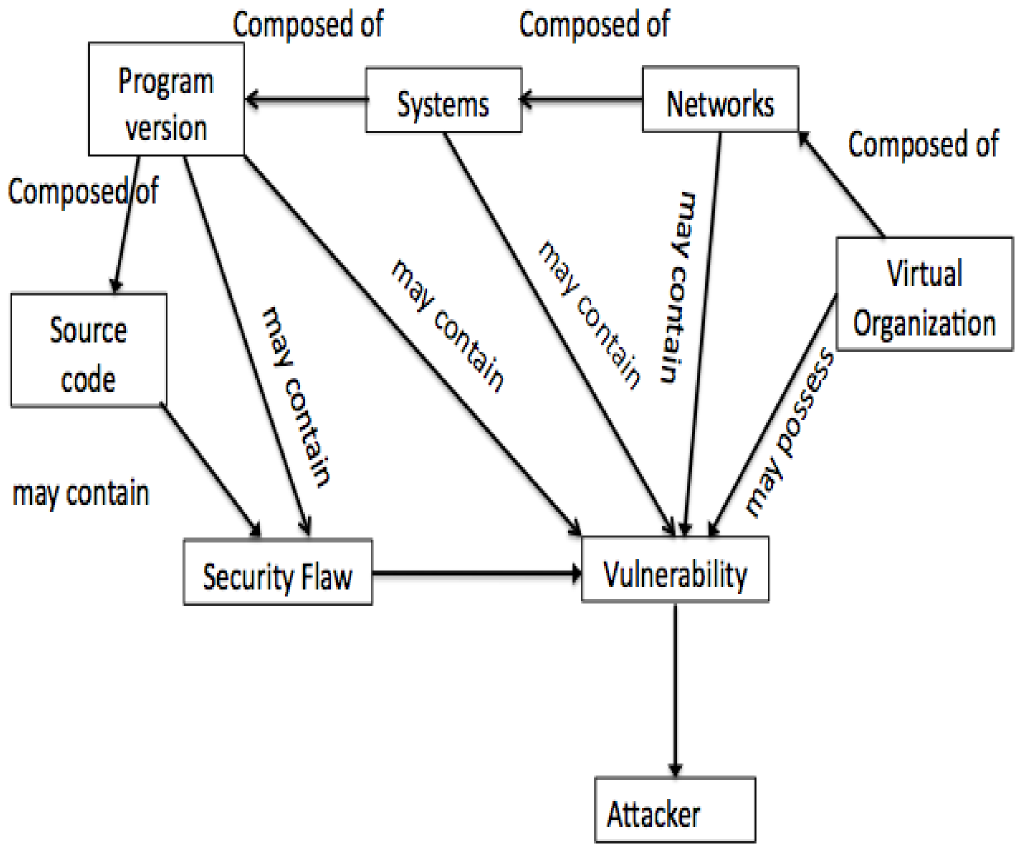


Figure 2.2: Objects and Relationships of Vulnerability [3]

Patch management is the process of correct and timely application of software patches to minimize downtime and remove vulnerabilities from a system. As patches are released in response to a vulnerability, their broader aspect cannot always be predicted. Quantifying patch management is thus a difficult and complex task. These complexities can prevent patches being deployed in a timely manner to vulnerable systems - in some cases patches are only deployed months later. For the few administrators diligently applying patches, the task is still non-trivial.

2.3 Vulnerabilities, Malware and Exploitation Trends

One crucial aspect in organisational security is the numerous variables (e.g. increased number of vulnerabilities, high number of patches, third party software and relevant updates), which are all potentially moving targets. Many issues need to be considered when situating any patch management discussion in the reality of the security landscape: malware exploiting holes due to vulnerabilities in software; escalated number of attacks, and the shrinking exploit window are some of these. Other attack vectors such as e-mail phishing, instant messaging, evolving security root-kits and other confidence tricks fall outside of the scope of this thesis. In [59] Dominic has identified following key aspects related to vulnerabilities and threat exploitation within organisational structures: growing number of vulnerabilities and attacks, and reduced time taken to exploit a vulnerability. We consider these in turn.

2.3.1 Growing number of vulnerabilities

Today in the era of the Internet most security incidents are caused by vulnerabilities. The statistics from the CERT/CC reveal that the number of reported vulnerabilities has increased dramatically over the years. Other studies [60, 61] on malware and vulnerabilities suggest that there is an increase in the number of vulnerabilities. In July 2010 Microsoft rushed out an emergency patch for all its supported versions of Windows to cover a security flaw for all its supported versions (XP, Vista, Windows 7, Windows Server 2008 and Windows Server 2008). This addressed the way shortcuts are displayed by these operating systems [5]. This vulnerability could allow remote code execution if the icon of a specially crafted shortcut was displayed. An attacker who could successfully exploit this vulnerability could gain the same user rights as the local user. As shown in Figure 2.3, IBM monitored 13 billion real-time security events every day for more than 4,000 clients, i.e. about 3 million events/client per day or as a whole, 150,000 events per second. They found that that there were 8,000 new web vulnerabilities during 2010; 27% higher than those discovered in 2009 [4]. The most prevalent reasons for this trend is the increased use of the Internet by businesses and the public at large; the emerging Web 2.0 technologies and the increasing complexity of software and the increase in the number of software projects [23]. Moreover, Debian 5.0 has 323 million lines of code as compared to RedHat Linux 7.1 has 30 million lines of code from 170,000 lines in early Linux distributions. It is estimated that the number of software bugs ranges from 5-20 per 1000 lines of code [62]. It has been reported that the code size of Mitre's reference implementation of the Open Vulnerability and Assessment Language (OVAL) [63] scanner developed in C++ has approximately 35,000 lines of code [64]. An application of such large size makes it hard for developers to keep it totally flawless, and it is hard if not impossible to verify that there are no vulnerabilities in such applications [65]. Based on these estimates it can be seen that the number of potential bugs and the potential resulting vulnerabilities has grown immensely. Nevertheless, not all of these bugs will result in direct security flaws, and it is difficult to make precise extrapolations.

Additionally, this increasingly complex software is interacting and interworking with other increasingly complex software systems. The current trend in social networking applications, with several third party softwares running on a single machine, combined with the low cost of communication over the Internet suggests this will become an increasingly ubiquitous problem [66]. If vendors do provide users the ability to lock down their software, it is often not distributed in a secured state but left to the end users themselves. Other factors contributing to the vulnerability landscape are: lack of security knowledge among end users, understaffed systems administrators for large organisations, and the rise of preventable configuration errors becomes clear.

Common Vulnerabilities Exposure (CVE) [67] is the main source of these statistics which assigns a common name to each vulnerability discovered in given software. Symantec reported the total number 4,989 of new vulnerabilities in 2011. This figure works out to approximately 95 new vulnerabilities a week. However, compared 2010 which this number was 6,253, it represents a decrease of 20% from that of 2010. At a first impression, this may seem like positive news, but it must be viewed in the context of a longer time window. For example if we look at the trend over the longer term, we can see that the overall pattern is still on an upward trajectory. For instance the number of vulnerabilities reported in January 2012, amounts to 488 and is already well ahead of the numbers reported in the same month in 2011 [68]. The analysis of vulnerability threat trends [69] suggests that only one out of the top 10 vendors managed to reduce vulnerability disclosures in the year 2012 compared to average disclosures of the past ten preceeding years.

There is little reason to believe this increase will not continue. Highlighting the seriousness of these vulnerabilities, Eschelbeck [70] pointed out two hypothesis. Firstly, there is a constant

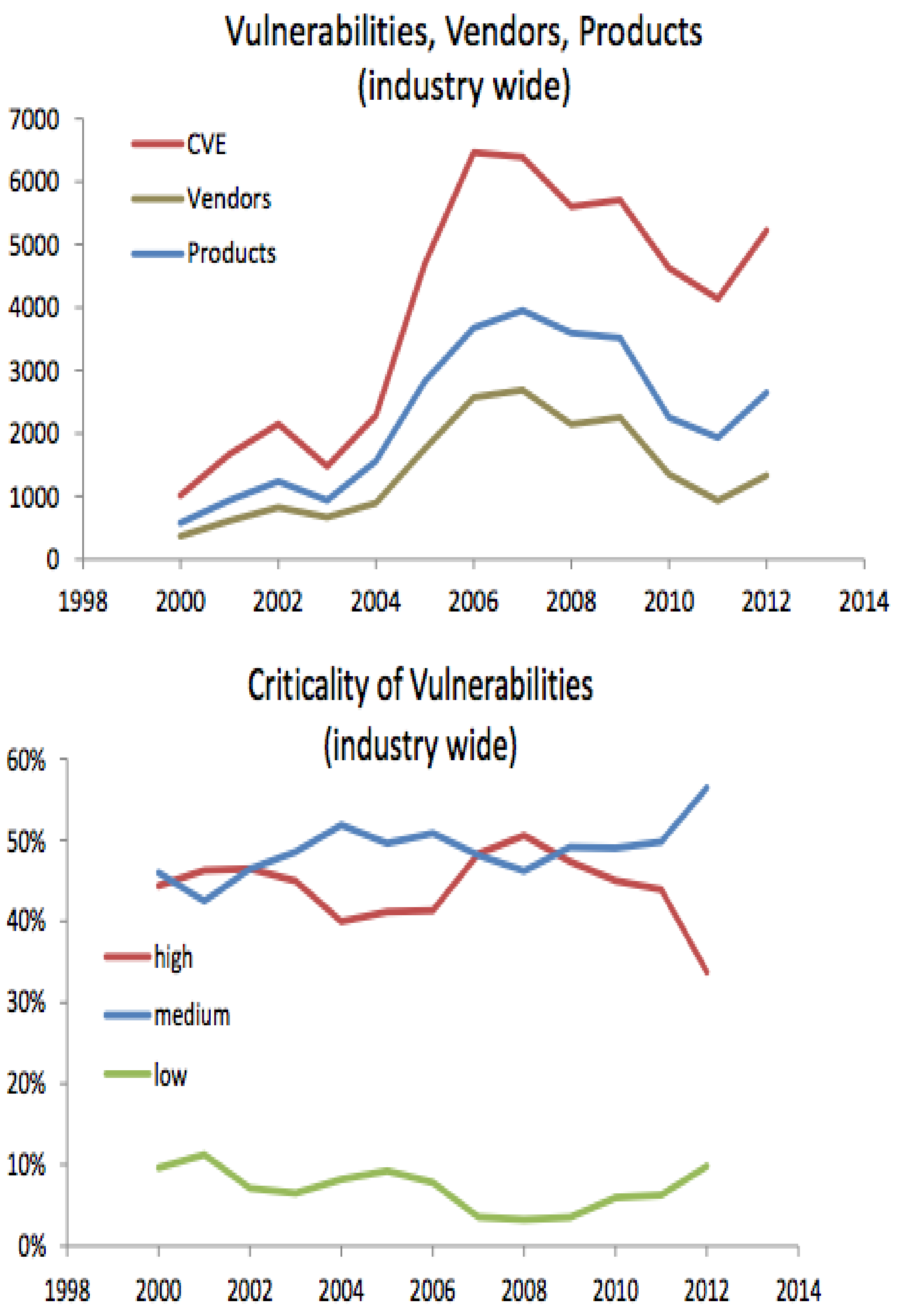


Figure 2.3: Growing number of vulnerabilities[4]

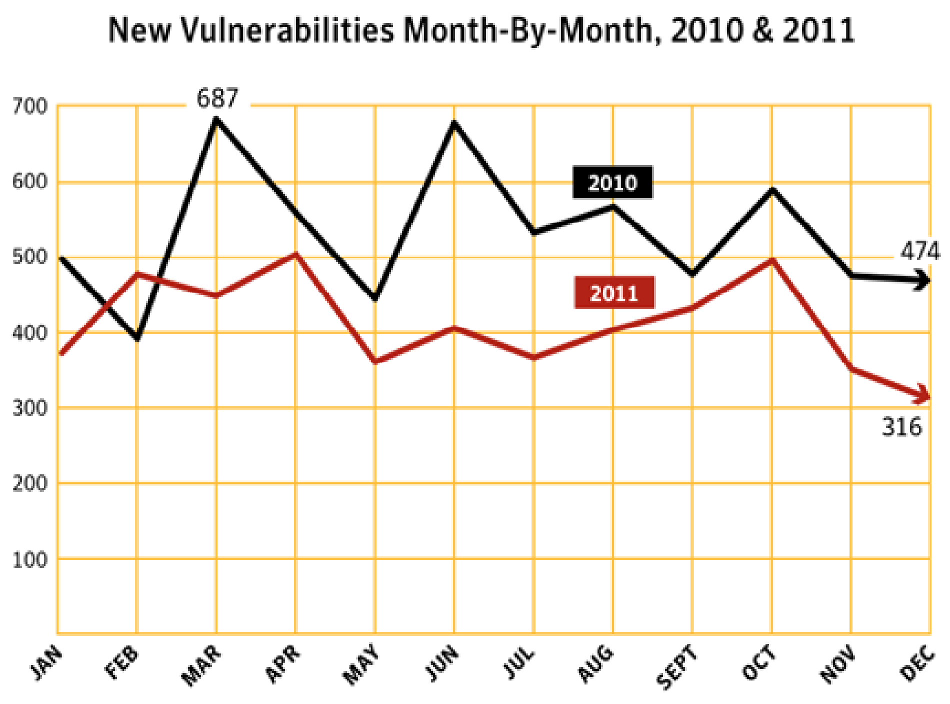


Figure 2.4: New Vulnerabilities month-by-month 2010 &2011

discovery of new critical vulnerabilities, which leads to a situation where half of the most common and critical vulnerabilities are replaced every year; and secondly due to continual deployment and redeployment of unpatched software on machines, these vulnerabilities often have an extended lifespan. Hence these vulnerabilities have a cumulative effect, whereby the marginal discovery of new vulnerabilities is increasing and the total number of critical vulnerabilities is increasing. Other factors include the failure of system administrators and users to cope with previous vulnerabilities, which could have been successfully mitigated.

2.3.2 Escalated number of attacks

The growing number of vulnerabilities have resulted in an increase in attacks on organisational and individual assets. This is a complicated statistic to measure however. Dominic [59] gives several reasons for this. For monitoring the attack trends, we need to collect statistics, which could potentially help indicate the number of attacks. However, many attacks are not detected, and in some cases if they are detected they are not reported. The problem is further exacerbated due to the non-trivial nature of the statistics. Firstly, if an attack goes undetected, then it cannot be counted. Secondly, reporting is driven by the victim and many organisations who are attacked are either reluctant to report and share them for publicity reasons [71], or administrators have dealt with the vulnerabilities and do not wish to disclose this [72]. It

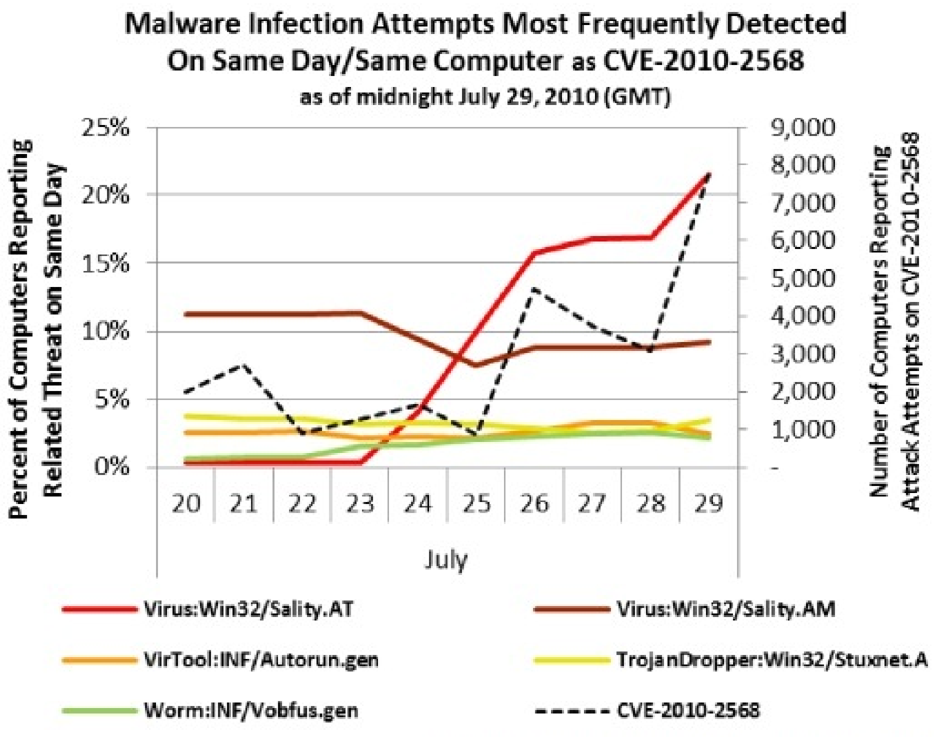


Figure 2.5: Malware Infection attempts most frequently detected [5]

is also difficult to get organisational consent to allow third parties to monitor organisational systems [71]. An alternate solution is needed in this situation, as suggested in a study conducted by Dominic [59], e.g. to conduct penetration tests and note the number of reported intrusions. However, this approach is not without issues, where on the one hand it would be illegal to attack remote sites without their prior consent, on the other hand, negative results could influence a site's reporting rate.

Due to recent advancements in Web 2.0 technologies, the situation is exacerbated, where attackers are now moving away from using mass compromises and focusing on more complex targeted Trojans, successfully exploiting hosts on-the-fly through a mix of popular rootkit installs, which provide more manageable results and help to evade detection [73]. The Hacker Defender anti-detection service provides a service where a semi-unique version of their Hacker Defender rootkit can be bought and used in a "pointed attack" specifically designed to avoid detection by anti-virus software by reducing the chances of anti-virus researchers crafting a general detection signature for the rootkit by providing many unique versions [74].

There are two main possibilities for estimating attack activity. In the first approach, taken by the CERT/CC, sites were asked to confidentially report incidents. The Internal Revenue Service (IRS) who established the Computer Security Incident Response Center (CSIRC), in the year 2010, the IRS detected 2,768 computer security incidents and threats, which represent a 22 percent increase over each of the past two years.The alternative approach taken by the

SANS Internet Storm Center and their DSHIELD [75] project is to receive submissions on network activity from distributed sites and perform central analysis of the data. This allows the number of attacks to be better modelled, however it does not provide information as to how many of those attacks are successful, unless a successful attack displays some obvious behaviour (this is often true of worm activity, but not of human exploitation). Some of the resulting noise from attacks can be used to perform a back-scatter analysis, which is particularly effective for Denial of Service attacks [76]. However difficult it is to model attack trends, research tends to agree that the number of attacks and incidents is increasing every year [77, 78].

Given the increasing number of vulnerabilities it is obvious that this will lead to a higher number of successful attacks. This is borne out by the continuing success of automated self-propagating malware (worms) and their continued activity even after a patch has been available for several months [70]. This is further corroborated by DSHIELD which saw the average time between attacks drop below five minutes in September 2005. This time to live or survivability statistic gives an unpatched machine less time until it is compromised than it would take to download and deploy the necessary patches [79].

2.3.3 Shrinking Vulnerability & Exploit Window

As shown in Figure 2.6, an exploit window is the time between the release or announcement of a vulnerability and the release of the public exploit code. Recent literature shows the evidence of this window decreasing [80, 81, 82, 83, 84, 61, 85, 86, 87]. The exploitable vulnerabilities provide an attacker with the opportunity to compromise organzational assets before the software developer or vendor becomes aware of the flaw. The term “Zero-day” is used to define the age of the exploit, when an attacker is able to exploit a vulnerability. As soon as the system administrator/developer becomes aware of the flaw, the race begins to close the security hole prior to attackers’ misuse of the vulnerability. Figure 2.6 shows the typical timeline of various aspects from reporting vulnerabilities until a vulnerable resource is exploited [88].

Indeed, the evidence from the research community confirms the shrinking window in terms of vulnerability and exploit. For instance, the Nimda worm appeared a year after the vulnerability had been announced, the SQL Slammer worm appeared after six months, Slapper took six weeks, Blaster halved that to three weeks, Sasser took two weeks, Zotob appeared after five days, and the fastest vulnerability-to-worm cycle to date has been the Witty worm, which appeared 36 hours after the vulnerability was announced [70, 89].

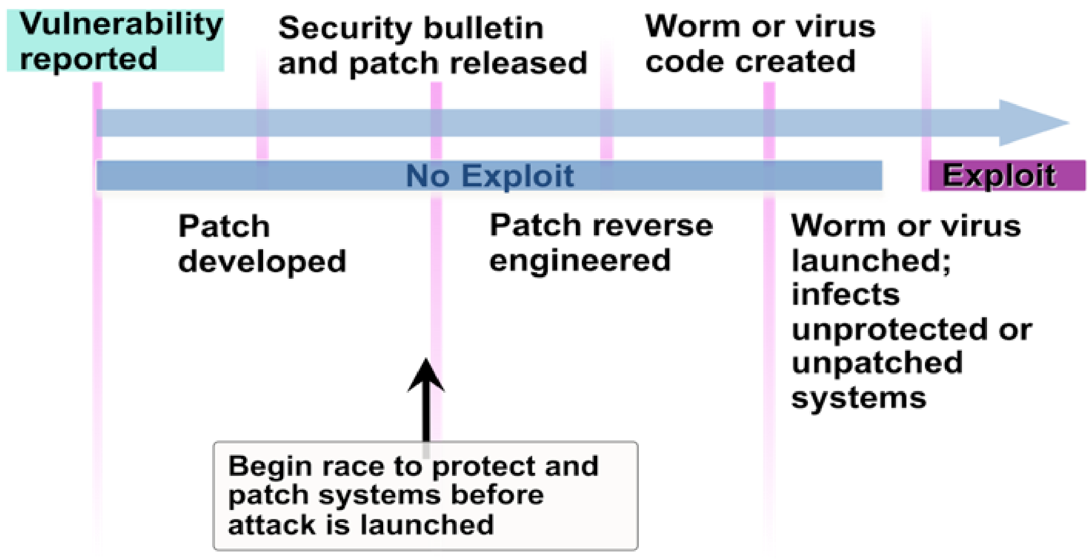

Vulnerability reported
Security bulletin and patch released
Worm or virus code created
No Exploit
Exploit
Patch developed
Patch reverse engineered
Worm or virus launched; infects unprotected or unpatched systems
Begin race to protect and patch systems before attack is launched

Figure 2.6: Exploit timeline for "Vulnerability-window"

The vulnerability disclosure time until the release of a scripted exploit is the most significant indicator showing when a vulnerability has progressed from a theoretical discussion to both a "likely to occur" and "likely to be successful" attack. The previous trend from worm attacks shows that the exploit window is shrinking. This hypothesis is confirmed by several sources [70, 71, 90]. In addition, the exploit window appears to be shrinking faster than the remediation window. A powerful example of this reduction is the emergence and growth of 0-Day(Zero Day) exploits. As noted the term 'Zero Day' traditionally refers to an exploit for an undisclosed vulnerability, but is increasingly used to refer to scripted exploits released on the same day as the vulnerability was disclosed. In both situations the exploit window is but a few hours.

In [71] CERT/CC hypothesised that malicious groups could have privately hoarding exploit tools which could be made public immediately when a vulnerability is released, thus skewing the time from public disclosure of a vulnerability until public disclosure of an exploit. However, the most likely reason for the shrinking exploit window as suggested by [91] there is an increase in the sophistication of exploit development tools [71, 92].

2.3.4 Vulnerability Management Process

A vulnerability management process normally comprises starting with systems which do not have known vulnerabilities. Tian et al. [93] identify several vulnerability management processes:

Firstly, maintaining system information (inventory) including what type of hardware, operating system version, running services and third-party application is installed on organisational resources. However, this can be achieved through manual or automated approaches. There are proprietary software solutions such as Tivoli Inventory [94] open source technologies such as Pakiti [95]. Other similar surveys such as SecurityFocus [96] systems follow a similar pattern whereby site administrators spend an average of 2.1 hours per day hunting (via bulletins/email lists) for security information relevant to their own site security issues.

Secondly, getting reliable, exhaustive, and up-to-date vulnerability information related to target systems is a prerequisite for effective VM. Common sources of vulnerability information such as Bugtraq [97], CERT [22], and ICAT Metabase [98] available free of cost. However commercial software systems are available which provide vulnerability information including Security Intelligence Alert from SecurityFocus [96], SecurityTracker [99]. Currently there are no automated tools for administrators to customise vulnerability information because the vulnerability advisories often come from various sources and have ambiguous text-based description in different formats and using different terminologies.

Thirdly, assessing and evaluating vulnerabilities in organisational resources helps to determine what vulnerabilities exist in the target systems to calculate the risks associated. There are a number of free and commercial security vulnerability assessment tools such as Nessus [100] and ISS [101]. However these tools do not provide the detail of security circumstance of the target system such as critical configurations and deployed versions of software. This leads to an incomplete picture of the overall organisational systems under potential threat.

2.3.5 Proactive Vulnerability Scanning

In an ideal scenario, vulnerability scanners would not be required due to vigilance of individuals at an organisational level to maintain patches and tested systems, routers, gateways, workstations and servers. However, this is rarely the case in the real world. It is human nature to forget things such as installing updates, patching systems and configuring them properly [102]. A proactive approach to vulnerability management is the prevention of any malicious intent activities towards organisational assets. In most cases, organisations adopt the strategy of proactive patching in parallel with vulnerability scanning to mitigate problems that can occur which have been identified, and where patches have already been released to address them. As shown in Figure 2.7, a typical vulnerability scanner generally comprises a scan engine, a vulnerability database (Vul DB) with latest information (plug-ins) which is updated

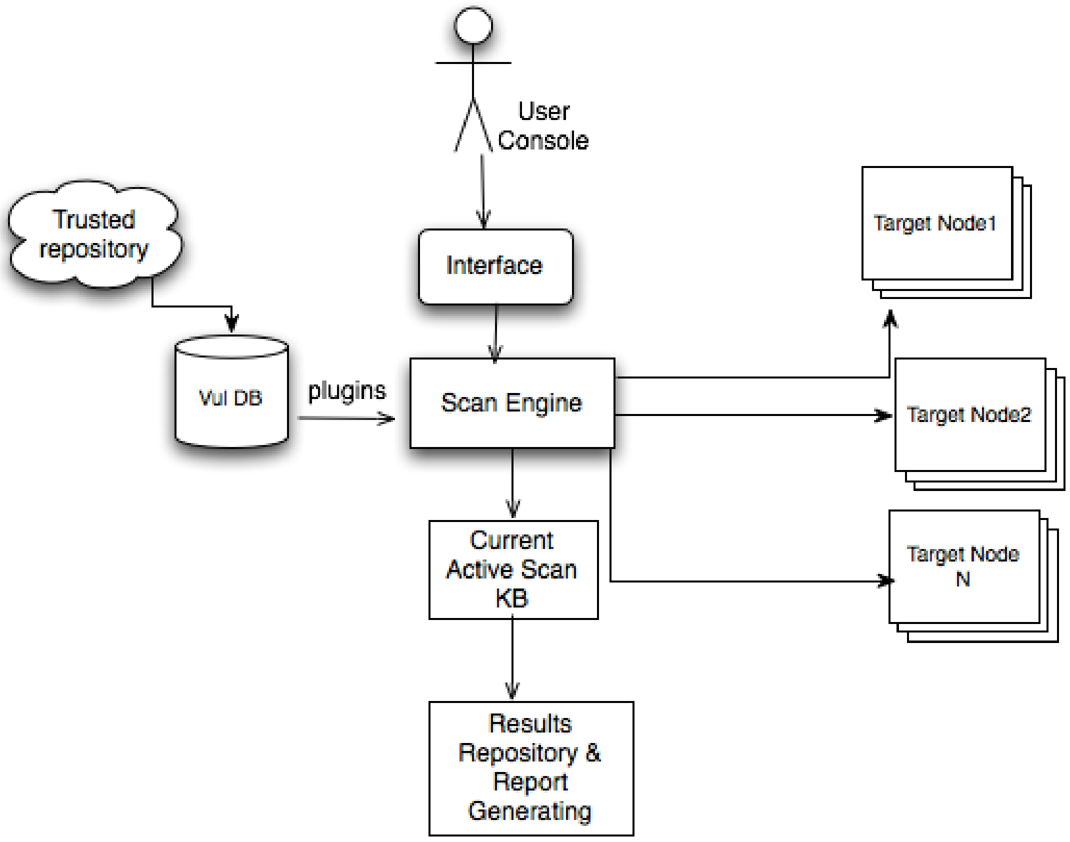


Figure 2.7: A Typical Vulnerability Scan Engine

regularly from a trusted repository; a results database, and an administrative/user console. Such scanners allow the installation of these components on one host, and databases to store scan results. This type of combined architecture gives site administrators the advantage of an easy installation. Scanner technologies such as NetRecon [103] and Internet Scanner [101] can be installed and run in minutes but such products can also create administrative hurdles in large organisations that need to distribute the product across many networks yet maintain central control [104].

The basic assumption behind proactive vulnerability management and patching is that the systems are functioning normally. Therefore, why apply a patch if the system is functioning normally, since any change could potentially involve risk and downtime? In the real world, all systems functioning normally, can potentially have underlying problems waiting to happen as identified by [105]: memory corruption that has not yet caused a problem; data corruption, which is typically unnoticed until the data is re-read, and latent security issues.

Most security issues tend to be latent issues, i.e. they exist in the system, but are not causing any potential issues yet. Thus, it is important that site administrators take proactive action to prevent such security vulnerabilities from being exploited by remote hackers. In comparison to reactive patching, proactive patching generally implies more change, and additional planning, for regularly scheduled maintenance windows and testing. [106] suggests that proactive patching is recommended for : reducing unplanned downtime; preventing systems from experiencing known issues: providing the ability to plan ahead and do appropriate testing before deployment. The benefits of this planned downtime are that for proactive maintenance is usually much less protracted than unplanned downtime when addressing issues reactively.

2.4 Vulnerability Scanning tools and their Weaknesses

Currently there exist several "security audit" packages, both free or as commercial products. This review has concentrated on non-commercial packages due to their accessibility and ample reference material. It is also generally true to say that the commercial products are sometimes derived from, directly or indirectly, the same source as the public domain products. In reviewing security audit tools, Howard [107] identifies five basic areas in which weaknesses can be shown to exist in security tools: generality; maintenance; relevance; trust, and timeliness. We discuss these here.

Generality: many security tools are written in a general way that is independent of the hardware platform and operating system on which they will run.

Maintenance: many security packages, whilst being well written and able to perform a useful task, are not actively maintained. This is often the case when they were created via academic research projects.

Relevance: when a tool has been created with an internal database of vulnerabilities often it becomes out of date from the moment it is released. As identified the number of exploits currently being discovered and publicised is of the order of 20 a month. Any tool not able to detect these new problems declines in relevance as time goes by. For example the work of the MITRE Corporation [108] suggests that by separating the "tool" from the "database" of vulnerabilities, the same tool that is currently being used will be able to leverage new developments in vulnerability databases because the tool will use information loaded at run-time rather then information available at creation time.

Trust: typically, many audit tools require access to the entire system (root priviliges) in order to completely assess for the existence for vulnerabilities. This typically results in running the audit tool with “superuser” permissions on target node(s). This type of scenario demands that the system administrator is able to trust the tool in terms of its operation, interaction with the system(s) on which it runs, and the results it may produce.

Timeliness: to protect systems from latest vulnerabilities, the tool should be able to access current vulnerability information in a timely fashion. If it cannot either by it's design or because of practical implications, then it doesn't exhibit the desirable quality of "timeliness".

Not all tools exhibit all five weaknesses, but all tools appear to suffer from some of them. The following public domain tools are presented in brief and then related to the five weaknesses outlined earlier in the section.

2.4.1 Discussion

Vulnerability scanners have advantages and disadvantages, which site administrators need to consider prior to joining a VO and any vulnerability scans that may be expected. Advantages and disadvantages of vulnerability scanners are described in [109]. Firstly, a particular vulnerability scanner can only assess a "snapshot of time" of a typical system. Ideally a regular scanning process should be conducted to improve the security.

Secondly, human judgment is often vital to finding reliable results from scanners, since vulnerability scanners only report vulnerabilities according to the plug-ins installed in the scan database. They cannot always determine whether the response (result) from a scan is a false negative, i.e. a failure to recognise an existence of a flaw on a target nodes, or a false positive, i.e. the incorrect determination of the presence of a vulnerability. Hence human judgment is always required for analyzing results of the scanning process.

Thirdly, since the sole purpose of designing a vulnerability scanner is to discover known vulnerabilities they cannot identify other security threats, such as those related to physical, configuration, operational or procedural issues. In addition, many vulnerability scanners rely on "plug-ins" to determine potential vulnerabilities in a target resource. However, these plug-ins are part of the knowledge database (or scan database) of vulnerabilities that these scanners are capable of detecting. With the speed of malicious codes being produced the limited number of plug-ins is a drawback with largely static vulnerability scanners, i.e. where the plug-ins do not change rapidly with evolving exploits.

Fourthly, a vulnerability scan itself can pose risks to organizational assets (systems). This might include overloading vulnerable servers if all "*plug-ins*", including high-risk ones such as Denial of Service scans are enabled. Risk assessment and careful planning are necessary before scanning takes place. Whilst, it might be acceptable to enable all plug-ins including high-risk ones as part of an initial VO setup process, for ongoing continual scans on a particular VO node, administrators may consider disabling certain high-risk plug-ins. Moreover, whilst conducting VO-wide resource scanning using a network-based scanner, a large amount of system requests and network traffic can be generated. Therefore system administrators should note any deterioration in the systems and network performance of the target resources in the VO during scanning.

Fifthly, for successful VO operations, participating sites, will absolutely wish to keep the vulnerability information in a safe place. The leakage of scanning results containing system vulnerability information, would be an extremely dangerous resource to VO and provide an opportunity to attackers wishing to exploit them. Hence it is vitally important to safeguard this information by keeping it in a safe place, using encryption and VO-specific access control to prevent unauthorised access. In some cases, sites might have an external party employed for the vulnerability assessment process. In this case, sites should ensure that any party involved is trustworthy, and that both findings (scan results) and proprietary information is kept secure. Alternatively, VOs may designate a partner site responsible for maintaining the security of this information.

Finally, the improper use of vulnerability scanning tools could pose an enormous risk and cause tremendous harm to VO-wide resources and other information systems. Therefore, policies and procedures should be in place which clearly specify to whom, how and when vulnerability assessment tools are allowed to be used. These policies may potentially include the kind of prior VO formation (contract negotiation) process, management approval and/or legal clearances that are required before a scanning takes place.

2.5 Patch Management

The patch management focus of this thesis builds upon an intersection of two related fields: vulnerability management and configuration or change management [110]. As discussed in the previous section, a vulnerability is a flaw in an information technology product (software or hardware) or configuration of a product, that can allow an attacker to access and use resources that should not ordinarily be allowed. As noted once identified by vendors, patches are then developed to address these vulnerabilities. It is often the case that system and network administrators have to keep an up-to-date patch management system with regular testing and installation procedures for applying patches. However it can be the case that such patches that are regularly introduced into systems can themselves potentially cause failures. This situation creates the "*patch paradox*" for system managers, where, without a patch an asset (system) is vulnerable to attacks, and with a patch the asset is potentially vulnerable to failures [59].

Some security practitioners specifically argue over the use of the term 'threat'. Bejtlich claims that security professionals are deliberately "mixing and matching the terms threat and vulnerability and risk to suit their fancy" [111]. This might be a valid argument, since few seem to agree on the subject. Additionally, there are several other resources such as the Office of Cyber Security & Critical Infrastructure Coordination [112], the US military Information Assurance division [113], the National Institute of Standards (NIST) Special Document 800-30 [114] and Microsoft's Security Risk Management Guide [115] each have their own definitions. The international standard ISO/IEC 17799 unfortunately does not provide a formal definition of such terms. The National Institute of Standards Special Publication 800-40 [116] defines, incorrectly, a threat as a synonym for malware [17]. In general terms,

A vulnerability is defined as a weakness in a resource (asset) which could be potentially exploited by a threat.

A vulnerability usually refers to "flaws or mis-configurations that cause a weakness in the security of a system" [114]. In [117], a vulnerability is defined as the intersection of three elements: a system's susceptibility or flaw, attacker's access to the flaw, and an attacker's capability to exploit the flaw. The International Electrotechnical Commission (IEC)/International organisation for Standardization (ISO) 13335-1 [117] defines a threat as "a potential cause of an unwanted impact to a system or organisation". In other words, a threat is an entity which has the potential to exploit a known vulnerability in an asset. Other researchers also define the source of threat, i.e. the actual entity with "capabilities, intentions, and attack methods of adversaries to deliberately damage, exploit, or alter information or an information system" [59]. In this thesis we adopt this definition.

We define an exploit as either an individual/group or tool(s) that are used to attack a vulnerability in an asset. That is, it is the action of an attacker to target unpatched assets (with missing patches or incorrectly configured patches) often using different tactics to exploit this vulnerability, and hence realising the threat against that asset. In the recent arena of Web technologies, attack methods such as malwares in the form of viruses, Trojan horses, root-kits and other worms are commonly use exploits. Example scenarios to identify exploits might include phishing to exploit human trust.

Figure 2.8 shows the high-level relationships between the components of threats, vulnerabilities and control mechanisms (safeguards). A threat exploits an existing vulnerability and therefore represents a potential danger to organisational assets and is initiated by a threat agent. To pose a risk to organisational assets, a threat has to exploit a vulnerability or potentially give rise to follow-up threats, via a physical, technical or administrative weakness, and cause damage to defined assets. Safeguards (access control mechanisms) have to be installed, and applied respectively, to mitigate an identified vulnerability and to protect the corresponding assets by either preventive, corrective or detective measures [6].

A patch is typically a piece of code used to update a software product [118] that corrects vul-

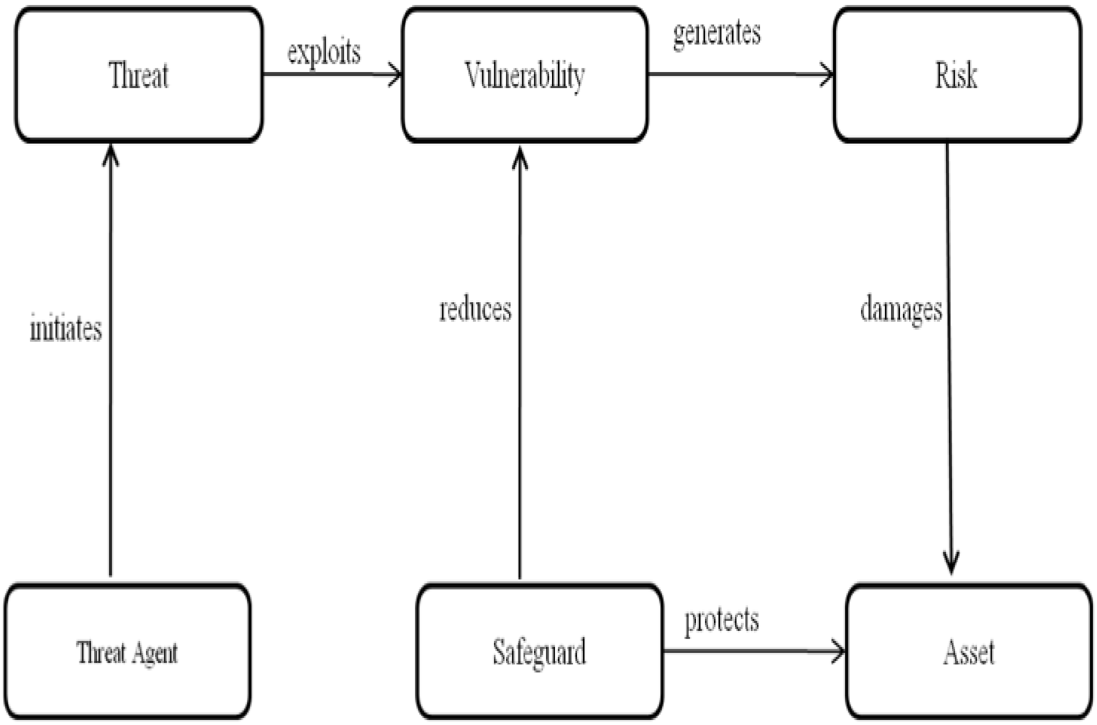


Figure 2.8: Security relationships [6]

nerabilities. The corrective action performed by the patch will not only prevent an attacker from successful exploitation but also remove or mitigate a threat's capability to exploit a specific vulnerability in an asset. In general, vendors and software providers issue a patch which can be used to correct a flaw that might not be security related, e.g. it may address performance issues, or could add a new functionality to an asset. In general, a patch usually consists of packaged pieces of software used to replace existing flawed code. Vendors normally release a patch in one of the three ways [59]: as a patch to the source code of a program; as a patch to the compiled binary code, or a complete file(s) replacement. Typically patch management tasks include:

- Maintaining current knowledge of available patches;
- Decisions on what patches are appropriate for a particular system;
- Ensuring that patches are properly installed;
- Testing systems after patch installation;
- Documenting all associated procedures.

2.6 Patch Management for Virtual Organisations

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 often bring with them challenges of security. For large-scale infrastructures such as those based on Grid technologies involving multi-site collaborations, the overall installation and configuration of the software infrastructure needs for particular VOs is increasingly demanding and complex for local system administrators at given sites since software targeted to the needs of specific VOs often comes from a variety of sources including from remote collaborators themselves. Management of such dynamic infrastructures is made more complex since they should ideally address the overall configuration and management of VO-specific resources across multiple sites, as well as configuration and management of the underlying fabric upon which the VO exists. Since an insecure fabric can undermine the security of collaborating sites and any threat (perceived or real) can impede the operation of the whole VO and confidence in use of shared resources it is essential that the "overall" security is tackled. In this context, patch management plays a vital role in supporting the overall security of such collaborations.

Patches, if applied correctly, can remove vulnerabilities from systems, however as noted in some cases patches may break existing systems if not properly analysed and tested before applying. Furthermore, if patches are seen as the '*cure-all*' to software vulnerabilities, then question arises why organisations don't apply them as soon as they are made available? The problem is that patch management is multifaceted, and there are often operational reasons why organisations don't apply them immediately. For example, users may experience conflicts with the installed anti-virus or spyware software, though having the ability to install new software on their local machines. These situations can result in the ineffectiveness of the softwares' protection mechanism without the user's knowledge or even any warning message to the user. Other problems with traditional patching systems include software set to be configured for manual updates that 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 etc [119]. Due to the above mentioned factors, many systems are left unpatched for months, even years [120]. According to the Computer Emergency Response Team / Coordination Center (CERT/CC) [66], around 95 percent of security breaches could be prevented by keeping systems up-to-date with appropriate patches [121]. The Slammer virus, which swept the Internet in January 2003, caused network outages all around the world, affecting 911 call centers, airlines, ATMs. However Microsoft released the patch fixing the vulnerability that Slammer exploited six months before the incident. Similarly, Conficker wreaked havoc on those companies that were not current with their software patch updates [1].

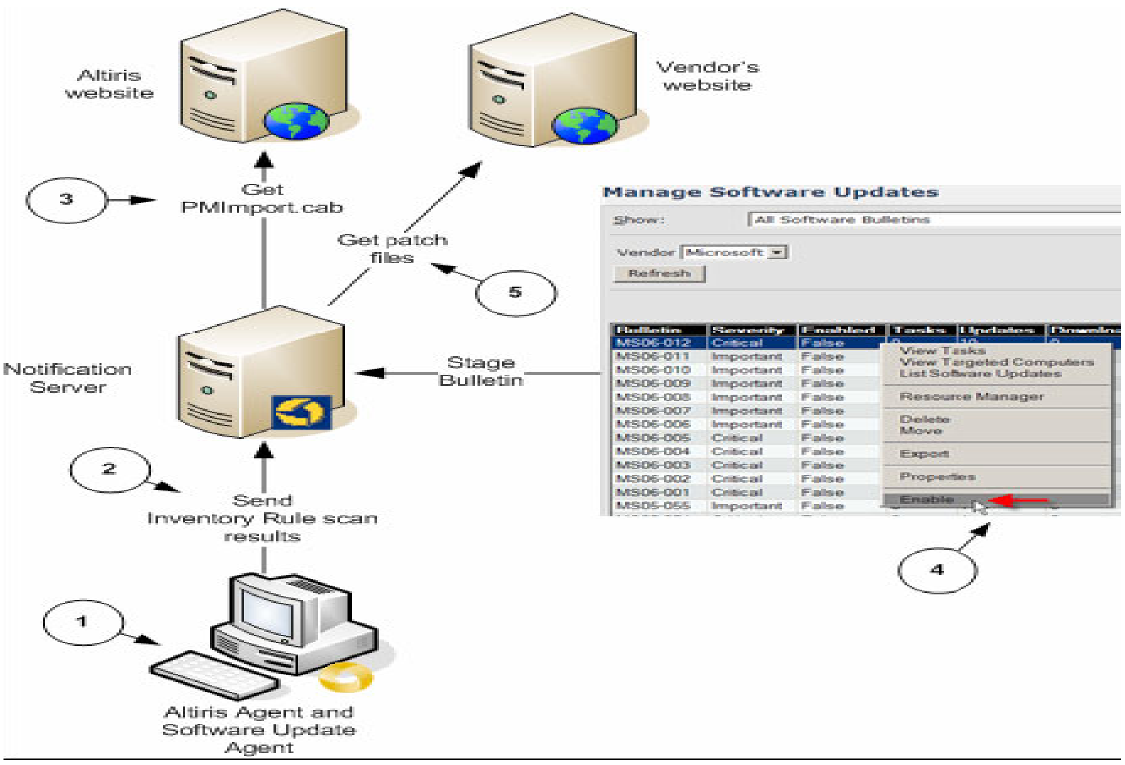
Huseyin et al. [122] point out several reasons for not applying patches. Firstly, there are often too many vulnerabilities to patch. For example in an average week, vendors and security organisations announce around one hundred and fifty vulnerabilities along with information on how to fix them [123]. Secondly, patches cannot always be trusted without testing [124]. Prior to applying them in production environments, each patch must be tested to make sure that it is working properly and does not conflict with other existing applications in the system potentially worsening the overall usability/security. In some cases, sites need to reconfigure the system and/or recode applications so that the patch can work without causing any new problems. Thirdly, distribution of patches is not standard. Some patches are available on vendor web sites e.g. Microsoft Windows Update (http://www.update.microsoft.com/windowsupdate) whilst some patches are not. Fourthly, every patch requires testing after installation and for many installed patches restarting (rebooting) of resources is often necessary. To summarize it can be stated that applying patches to mitigate the challenge/danger of unpatched systems in the face of attacks is expensive. On the other hand, the consequences of not applying updates immediately with available patches can be severe too.

This problem is exacerbated in dynamic e-Research domains where a single inadequately configured VO resource can endanger all other sites within a given VO if it does not take all appropriate measures to ensure its own configuration and security. Whilst advanced authorization infrastructures can protect access to given resources, these security mechanisms will be made redundant if the basic underlying fabric upon which those services exist are themselves inadequately protected. This might be from numerous perspectives: firewalls that have been left open or incorrectly configured; out of date anti-virus software; operating systems or middleware itself that have not been patched with latest updates to name but a few examples. Ensuring that all nodes used across a VO have the necessary operating system, middleware and that the underlying fabrics are properly patched and the most up to date antivirus software protection is something that up until now has been left to the individual VO sites. This is something that cannot be left to chance however. It needs to be managed at the VO level. For example, a VO dealing with medical records or any other highly sensitive data sets demands that resources are protected as far as possible. It is the case that the middleware solutions up to now have been primarily targeted at isolated aspects of security, e.g. defining and enforcing advanced authorization infrastructures which can be used to protect access to particular services or data. This thesis argues that a more integrated solution is needed that deals with end-end VO-wide security including VO-wide security policies and their enforcement which are aligned with local security policies and supporting the security of the underlying fabric upon which that VO is based. This requires an understanding of local and global VO policies and the ability to securely deliver and configure any software patches or local site configurations. One way in which such updates and configurations can be supported is through configuration management tools. However these have to now primarily focused on a single domain and not tackled inter-organisational collaborations. It is often assumed that the person using the configuration management tool is the local systems administrator. This is often not the case in inter-organisational collaborations, where remote collaborators may want or need to install patches on a given remote resource if it has been identified as compromised (subject to the remote security policy allowing remote administrators to install/patch their local resource).

Although configuration management tools help to address some aspects of the above problem they lack important and desirable features. They currently do not address inter-organisational (VO-wide) policies; they do not address heterogeneous platform dependencies; they do not typically support VO-wide patch management where resources may exist in shared jurisdictions, nor do they address VO-wide interdependency of patches and patching of live systems where for example jobs may be running and resources used by several VOs concurrently.

2.7 Patch Management Challenges in Distributed Systems

It has been proposed that distributed infrastructures such as the Grid should be accessible not only to ‘an elite few’, but also to the ‘general public/masses’ [125]. Ideally, individuals, even those not belonging to a “member” organisation, should be able to access or contribute resources, and collaborate with fellow peers in a Grid-like environment. Recently this paradigm has moved to Cloud-computing whereby the collaborative model is based on on-demand computing, e.g. software as a service (SaaS) [126], Infrastructure as a Service (IaaS) [127] and Hardware as a Service (HaaS) [128]. In these paradigms individual user(s) typically have to pay for the usage (Pay-as-you-go) of their infrastructure/service use. These collaborations may be short-lived and/or may not be client/server based. An individual with spare computing resources/power should be allowed to join a Grid as a service provider, and/or as a service consumer. In the case of a Cloud, a user does not even need to own/operate any infrastructure (hardware or software). These Grid and Cloud types of collaborations and endeavors can involve different VO members and collaborating organisations/entities. In such cases we can no longer assume the same level of trust is given to all protagonists. There is therefore a need for a dynamic and adaptive security framework that incrementally builds trust among participating entities. Furthermore in a trusted collaboration of several VOs the users need to know if they are interacting with the right piece of code/software or human and that their software



Bulletin	Severity	Enabled	Tasks	Updates	Downloa
MS06-012	Critical	False	0	10	0
MS06-011	Important	False			
MS06-010	Important	False			
MS06-009	Important	False			
MS06-008	Important	False			
MS06-007	Important	False			
MS06-006	Important	False			
MS06-005	Critical	False			
MS06-004	Critical	False			
MS06-003	Critical	False			
MS06-002	Critical	False			
MS06-001	Critical	False			
MS05-055	Important	False			

Figure 2.9: The Altiris Patch Management Solution [7]

interactions (messages/data) will not be modified or stolen as they traverse the VO. In short, users must trust the Grid/Cloud software infrastructure providers to sufficiently prevent malicious activities as far as possible. Similarly, it is reasonable to expect that the user must be proactive in this regard and share part of the burden of VO-security.

2.7.1 Patch Management Process

Many patch management processes follow a similar recipe. Figure 2.9 and 2.10 show patch management best practice being followed by industry patch management solution provider Altiris (now part of Symantec Inc). At a high level, the Patch Management Solution process works as follows:

1. On each of the managed nodes, the necessary Altiris agents are deployed;

2. In a repeating schedule, the installed agents gather and upload current vulnerability data to the central server;

3. On a scheduled interval, the central server retrieves updated patch management data from Altiris website;

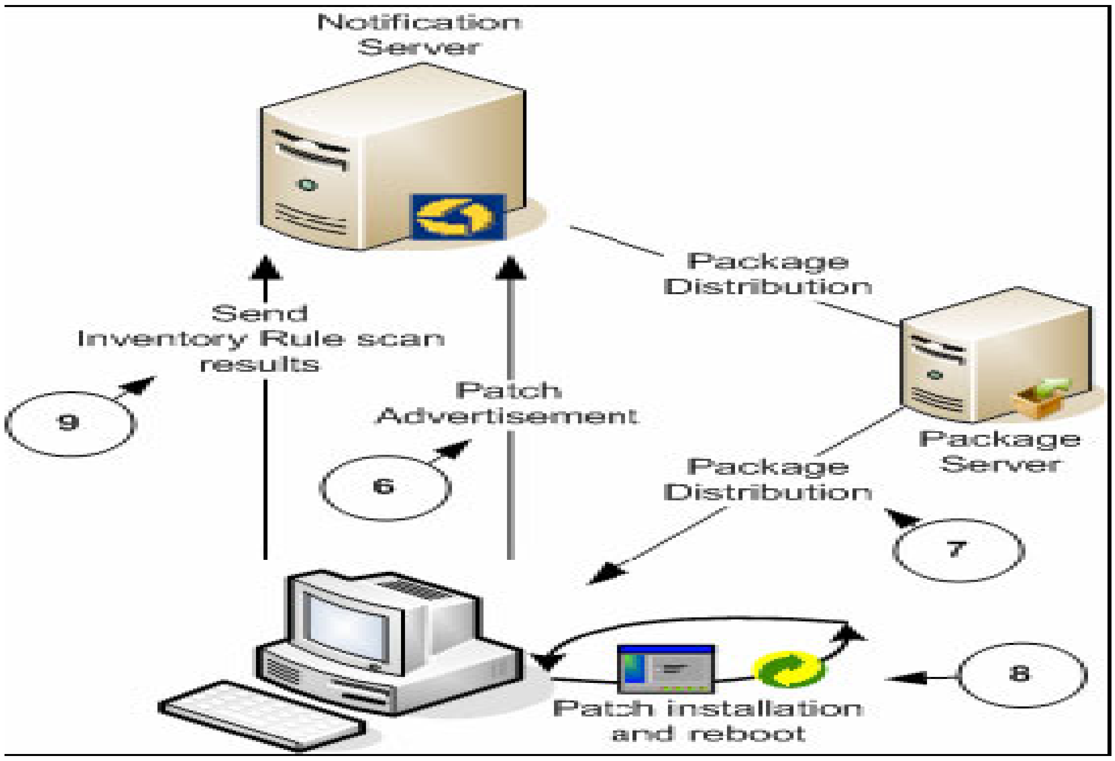


Figure 2.10: Altiris PM Solution part-II [7]

4. On enabling a “patch bulletin” to a central server by a system administrator, staging can take place;

5. The notification server then retrieves the related patch installation files from the vendor’s web site;

6. To avoid network conjestion, the central server advertises the patches to the managed nodes through an intermediary package distribution server;

7. The managed nodes periodically check in with the central server and download the patch from the package server;

8. For managed nodes, the agents install the patches at a predetermined time and reboot the nodes if required;

9. Depending on the schedule, agents continuously gather and upload current vulnerability data to the central server.

This model is very effective for a given centrally controlled set of resources but does not address the dynamic, often ad-hoc set of resources that need to be administered as arises with Grid-based VOs.

2.8 Patch Management Systems Research

A body of work has been undertaken by the research and commercial community in patch management. Whilst patching solutions such as Microsoft-Windows Software Update Services (WSUS) [129] provide patching for Windows platform and MS proprietary products, these do not meet the wider research demands for integrated patch management, e.g. for eResearch VOs. In typical virtual collaborations it is not always possible to restrict users' particular application installed. VOs are typically driven by their unique software requirements. Windows Software Update Services has a ''blackbox update model'' which allows easy and seemingly automatic patch update, yet the non-transparency of the system leads to the following issues identified in [10]:

- privacy and trust issues: for example it is difficult to determine if the scanner is performing the correct actions while preserving a local system security policy. As the update system is hosted on the remote server (vendor's site), there is no guarantee that local information will not leak to an external source thus breaching local system privacy.
- Solutions such as WSUS behaves more as a vendor patch update mechanism rather than standardized vulnerability entry checking. Little information is given back to users in terms of standard vulnerability report information. This might be too limiting for system administrators who, for example, want to ensure that his systems are up-to-date against recent vulnerability reports regardless of whether patches for the vulnerability are available or not.
- as there is no control over the scanner, users need to trust that Windows Update works as it is supposed to. This issue of blind trust is highlighted in the incident of Swen-style Trojan horse which posed as a legitimate update [130]. Although this is a social-engineering style attack example, it illustrates the fact that the Windows Update mechanism can itself be a vulnerability.

Other related concerns of the Windows Update mechanism are discussed [131]. In this thesis, we argue that any automatic update/alert processing mechanism for patches should be based on an open model which can be independently verified. Moreover, the administrator/user should be able to determine the consequences of a patch or alert on his/her system.

Patching techniques themselves can take numerous forms. They can be based on pull-models (agent-based) or push models (scan-based) [120]. Scan-based architectures typically require that the communication is initiated by the server querying each node/computer directly. In this model the desired scanned system (clients) have to be connected to the network otherwise these will be missed/remain unpatched when offline. This can often be the case in VOs and on-demand computing scenarios, i.e. one cannot guarantee that the clients are always online. As a result scan-based architectures are best suited to static networks. Scan-based architectures also have performance issues associated with them. For example they require that the scanning system sequence through every individual node to determine their characteristics and subsequently retrieve a significant amount of information from each node to determine the relevant patches to be installed. In comparison with scan-based approaches/architectures, agent-based approaches install agent software on each client node which in itself can be a disadvantage of these systems. These architectures are best suited to situations when the nodes are roaming and periodically disconnected or inactive. These systems scale well in comparison to scan-based approaches as in the former case, the agent is responsible for performing most of the work. Despite the extra burden of installing agents on all nodes in the network, in terms of accuracy and speed, agent-based systems are typically superior to scan-based architectures.

Jung et al. [132] conducted a survey of patch management products and found that only 4 out of 10 products could provide protection against attacks such as man-in-the middle, forgery and replay. Dominic and Barry [133] propose an automated patch management architecture claiming to be a complete patch management system with important features of initial vulnerability detection, thorough to testing, deployment, reporting and maintenance. This was an ambitious project and is currently a work-in-progress so verification of the system working in true multi-platform environments has yet to be demonstrated. In [134], Tian et al. propose an automated vulnerability management system using web services and agent-based systems which scan for vulnerabilities in the operating system version, services and third-party applications. The system sends resource profile information to a web service ‘scanner’ and returns all possible vulnerabilities. Although, this system works fine in a standalone or static environments, it has not been tested in a multiplatform and heterogeneous environment where several underlying operating systems and a wide range of policies may co-exist.

Chang et al. [135] present a patch management model and architectural design for large-scale heterogeneous environments. This architecture is based on five layers, however it does not address environments when there are several collaborators with a wide range of site/system policies as is typical in inter-organisational VOs, and where policy conflict detection and resolution are required. Dominic [133] in an attempt to enhance his previous work [136] identified that a complete framework based on open source community tools is needed to tackle the patch management process in a holistic approach rather than simply patching the holes. Although this effort brings up several ideas for the research community as whole in developing complete patch management solutions, the extension for heterogeneous and multi-site collaborative environments is not addressed. Furthermore the proposed architecture has not been rigorously tested in a real environment which makes the author's claims unsubstantiated.

The major configuration management vendors and tools providers like Microsoft, Altiris, Computer Associates International, IBM Tivoli and LANDesk do not have embedded knowledge of patch interrelationships, nor patch analysis capabilities when dealing with heterogeneous software environments comprising multiple software from multiple providers. Thus these tools are not optimized for inter-organisational and heterogeneous patch management, and their use is labor intensive [137]. In August 2008 RedHat confirmed that its update server normally used for signing Fedora software packages was compromised [138].

The modern operating systems have a variety of tools available for obtaining and applying patches, either the vendor-provided tools or third-party. There are, however, large differences in how the tools function. For example, how the patches are obtained and how the patches are applied. Patches are usually obtained using one of three methods [139]:

- Obtaining most patches from a set of common "repositories", maintained by the vendor or community, with only unusually esoteric or site-specific software excluded. In this method, one software application is generally used to download and apply the patches.
- Obtaining all patches from one source[1], such as a vendor-supplied firmware for hardware or appliances.
- Obtaining operating system and OS vendor software patches from one source, and non-OS vendor software patches from the respective software vendor or distributor.

Within these methods, there is also some difference in terms of patch selection. Many operating systems provide automated tools that list available patches and allow the administrator to apply them, but some tools (generally the ones designed for non-technical end-users) provide less granularity in package selection or automatically select all available packages.

There are three primary methods of patching software: patching the source code and recompiling (common in Unix-based solutions); installing an "update" or "delta" package, i.e. only include the files that were changed or patched since the last major release of the package, or completely re-installing a software package.

Historically most patches have been distributed as files containing specific changes to be made to existing files (either source code which would then be recompiled or directly as binary code) by a patching utility,which replaced the file contents at the line or byte-level [139]. Most modern operating systems now manage software as distinct packages of related programs and

[1]Though the common practice is to have third-party firmware available, such as DD-WRT and OpenWRT firmware for consumer-grade routers.

files (such as an application or a specific web server) which greatly eases software management. However, according to Jason [139], packaging systems are a double-edged sword when it comes to patching; software managed through the package management system does not need to be manually patched, instead making use of update or delta packages.

DNS poisoning tested by Kaminsky [140] who revealed that it was easy to fool systems and people into thinking their intended update server was reached when in actual fact they had connected to a maliciously crafted server. Once clients connected to such a bogus server, the users would unknowingly download whatever type of malware the attacker wished such as Trojans, spyware, keystroke logger and bots etc. For most software auto-updates there is no easy way to know whether the patch was legitimate or not, hence many sites require checking MD5 signatures manually after downloading updates. On the other hand, many software vendors use proxy servers to simplify and speedup software delivery mechanisms. This approach might be convenient in some cases but it also means that an attacker can compromise the proxy server by placing their own attack code on the proxy. On top of these issues is the fact that there is no standard update mechanism used across vendors, which means users can easily fall victim to these types of attacks, especially if the software vendors are using non-secure channels.

In the long term, end users need to keep pushing and pressurizing software makers to secure the update delivery mechanisms by demanding use of digital signature on those updates. In the short term, users themselves are often left with downloading updates manually and verifying checksums on the downloaded packages to ensure that they actually match with those published on vendor's web sites. By using vulnerability scanners and tools such as *ISR-evilgrade* [141] to keep an eye on running software an overall organisation might itself be vulnerable. It might also be useful to potentially turn off auto-update features applications and manually install updates. Though there is no "one-size-fits-all" solution, these precautions are worthwhile especially considering all the attention given to attacking software auto-update features.

Whilst, patching VO-wide systems can be a good first step to mitigating vulnerability risks, simply installing the latest patches may fix only a subset of all vulnerabilities. Some mistake patch management as an adequate substitution for vulnerability management. Patches typically address only a portion of the vulnerabilities for a technology. NIST supports these findings in its Special Publication 800-40, "Procedures for Handling Security Patches," [142] which states: "Not all vulnerabilities have related patches; thus, system administrators must not only be aware of vulnerabilities and patches, but also mitigate 'unpatched' vulnerabilities through other methods (for example, firewalls and router access control lists). It is a common mistake among system administrators to monitor only patches and not vulnerabilities. Although this omission is understandable given the time pressures many system administrators face, it can be dangerous because the system administrator's chief adversary, the attacker, spends considerable time monitoring and exploiting vulnerabilities..."

Patching is a subset of the overall vulnerability management process. Deploying patches will only reduce a subset of an organisation's vulnerability risks. Some of the vulnerabilities require the IT organisations to address exposures by changing system settings or implementing workaround solutions before the vendor distributes a patch. Not every patch is considered critical or high-risk to an enterprise's security posture — so why spend resources and time on non-critical patches? Comprehensive vulnerability management requires an organisation to assess its system's vulnerabilities, apply system patches, make changes to the asset's configuration so that it will operate more securely. All of these problems are magnified with VOs and distributed, heterogeneous software environments.

2.8.1 Limitations of Current Patch Management Systems

A careful analysis of the similarities and differences between both the default (vendor or operating system-supplied) patch management tools and third party tools for the operating systems reveals that there are a number of severe limitations [139]. For instance, the method of obtaining notification of patches is inconsistent among vendors, and can vary from mailing lists and direct email to specific web sites or even alerts built-in to the software. Often, many vendors publish updates and patch announcements as RSS feeds, this disparity makes the task of collecting patch information for multiple operating systems and many applications extremely difficult. Moreover, the use of various tools for applying patches to different operating systems not only adds confusion to site administrators (forcing the administrator to remember syntax of many similar commands) but complicates automated management as details of tool use may change between versions, and vary greatly even within a given class of operating system, e.g. Unix or Linux.

2.9 Summary

In this chapter a detailed review of background literature has been presented in the area of configuration, vulnerability and patch management. The focus of the chapter was on related research work associated with vulnerabilities, configuration and patch problems, focusing in particular on large-scale distributed systems and e-Research collaborations. In the first section a discussion on configuration management in distributed environments was presented. The second section highlighted the trends towards vulnerabilities and malware in the recent past with a discussion on vulnerability management and vulnerabilities and malware threats organisational assets may suffer from. The third section provided an in-depth discussion on patch management challenges and the demand for distributed environments. The chapter provided a review of other work that have tried to address the vulnerability and patch management problem in the context of collaborative environments. Finally the chapter concluded with a summary of limitations of current patch management systems being implemented for Grid based infrastructures mainly used by the e-Science community.

Chapter 3

General Threats and Vulnerabilities– Use Case Scenarios

This chapter motivates the need for an authorization-oriented framework for security-oriented vulnerability management. It describes different deployment and configuration software models that are used across Virtual Organizations (VOs) that exist today and how they can be integrated with security-oriented authorization approaches to improve the security of collaborations. In particular, this chapter focuses on a variety of software deployment scenarios and their utilization of centralised and decentralised security authorization models. We highlight the challenges and research questions that exist based on these models and establish that it should indeed be possible to apply "controlled" (authorised) use of vulnerability scanners by local and remote VO administrators to improve the overall VO security including patching of vulnerable nodes. This chapter thus provides the basis for the work of this thesis.

As we have discussed in Chapter 2, section 2.4.1, that a single vulnerable node in a VO can endanger overall operation of the whole collaboration and confidence in use of shared resources. Since Grid-based applications are typically targeted to solve a particular scientific or research problem, which result in the customised portions of the software without quality assurance process. In contrast, the more mature and longer-lived software products undergo a rigorous process of testing and evaluation. Combined with the large attack surface, these Grid applications may contain vulnerabilities at the time of deployment which will result in substantial exposure for the Grid. Gehani et. al [143] pointed out several reasons which pose a substantial risk for Grid infrastructures which are discussed in the following section.

3.1 Risk Factors to Grid Infrastructure

3.1.1 Vulnerable code

The Grid community aimed at designing VO-solutions to be applicable for an extended period of time with an implicit demand that the software developed would be secure. Grid infrastructures such as the UK eScience National Grid Service (NGS), Open Science Grid, Scotgrid to name some of them are typically funded and managed by consortia comprising universities, companies, and government bodies that can afford to make the necessary commitment of resources to achieve security. On the other hand, the applications developed to run over the Grid are often produced by small groups of researchers, scientists, and engineers. Often such user/groups may not be trained in secure software engineering practices, nor are they provided with a budget that could afford it. Instead, they develop a piece of code to answer a focused research question and may not reuse the code in the future.

Another factor in this area is of secure software development practices overheads, especially if their programs are expected to have a short life cycle. Boehm [144] identifies that Grid applications are often likely to contain bugs at the time of deployment. These inherited flaws potentially allow an adversary to exploit some fraction of these bugs to breach security.

3.1.2 Exposed services

Grid computing projects are often structured as collections of Web services [145], each of which operates independently from the others and provides distinct functionality. To facilitate interactions between components, many projects use standardized data representations, typically in eXtended Markup Language (XML) [146], with available resources described in a language such as Web Services Description Language (WSDL) [147], and use communication protocols like Simple Object Access Protocol (SOAP) [148] for remote invocations. Based on SOAP, this service-oriented architecture lets legacy systems be seamlessly integrated while simultaneously allowing them to be dynamically replaced at a later point [149]. In so doing it is possible to utilise explicit service contracts, abstracting utilities, support loose coupling to minimize dependencies, promoting code reuse, facilitate composition of applications, and allowing the discovery of new functionality. As mentioned 2.8.1, these Grid based applications can be dispatched to arbitrary VO nodes in the wider Grid from where they may potentially initiate connections to and receive callbacks from external nodes (e.g. application-specific database servers) over the Internet. As identified by Gehani et. al. [143], the disadvantage of using such an open architecture provides an opportunity to a large attack surface for external adversaries to target.

3.1.3 Automatic privilege escalation

Grid technologies such as Globus incorporate functionality to transparently delegate rights from one node to another – so called delegation of authority. But this delegation demands a user to furnish their identity (e.g. certificate or password) on individual nodes running the job. As discussed in [143],the Grid infrastructures farm subtasks out to a large number of nodes, requires to configure the underlying infrastructure to utilise the delegation mechanisms. This in turn, facilitates the migration of user code along with its permissions to arbitrary nodes in the Grid without manual intervention. Nevertheless, this convenience comes at a cost by introducing system-wide risk in the event that a malicious user gains access to a node in the system. Users can leverage predefined trust relationships to potentially execute code at many other nodes in the Grid infrastructure. Thus, an attacker who is able to compromise a part of an application running on a single node will be able to rapidly escalate access to a larger aspect of the Grid.

3.1.4 Attractive attack platform

Grid nodes form an attractive target to attackers for a variety of reasons. Each node is usually well provisioned with fast processors, large amounts of memory and disk storage, and high network bandwidth. Dedicated nodes (such as those used in TeraGrid [150] and Open Science Grid [26]) are provisioned with hardware needed for high throughput interactive computing. Further, such Grid resources are deployed in decentralised configurations which makes it difficult to detect and interfere with an attacker's activity. The absence of interactive users who may note anomalous application behavior, the lack of egress filters deployed by Internet service providers to curtail spam originating from their customers' infected computers, and the dearth of intervening institutional firewalls make such Grid nodes particularly suitable for use by attackers.

3.2 Dynamic and Complex VO Requirements

In addition to the risk factors highlighted in previous section, it is essential to guarantee that the VO-specific software provided is well configured and validated, i.e. it runs as expected and produces the expected results. Whilst it is possible for a collaborating site to satisfy specific network or system configuration requirements imposed by a VO, if a site supports numerous VOs, it is almost impossible for a single site administrator to install, configure and validate VO specific software while keeping up to date with all software releases and installation requirements required for each VO. In such dynamic environments, many releases of the same package containing (libraries, object and source files, and other auxiliary files) need to coexist.

Site administrators might not be familiar with all VO specific software requirements and release procedures, not the dynamic and changing communities of end users. Consequently, a known set of "privileged" users within a VO need to be identified prior to the VO establishment process and, especially those tasked with managing the application software throughout the lifecycle of the VO collaboration.

3.2.1 VO Administrators Roles and Responsibilities

A VO administrator requires adequate tools that allow for triggering software installation and configuration on target VO resources (sites). In addition to VO-specific software management, a VO administrator may also have responsibilities including managing disk space and software versions; planning for software upgrades and removal (including detecting vulnerabilities and applying remedial patches); publishing site and software status related information amongst other things. Based on the sensitivity of these activities, several concerns for site administrators exist, especially in terms of site autonomy and security where local policies need to be respected and enforced. Thus a VO requirement for major software upgrades may not be realistic based on ongoing site usage. For a successful VO operation solutions must be provided for (at least) as described by Santinelli et al. [29]: establishing mechanisms that allow for scheduling of software installation processes when appropriate; ensuring adequate disk and space management; failure and conflict resilience; resolving software dependency issues; handling concurrent installation processes, and satisfying any required pre-requisites before the application software installation and configuration process is triggered.

3.3 Use Case Scenarios

There are a multitude of challenges in how Grid software deployment across VO-resources can be achieved. To distill these, a focus is made on case studies representative of these challenges, which provides the basis for the work in later chapters.

3.3.1 Installing Software on-the-fly Through a Job Description

In this model a privileged user, e.g. a VO-administrator sends a job that installs software on a file system that is shared between all VO resources, e.g. worker nodes, across the site. Alternatively unprivileged users, e.g. normal VO members, may only have read access to this file system. A precondition to this scenario is that VO participants and sites contributing resources agree on user roles and privileges required in advance. The Virtual Organisation Management Service (VOMS) [151] offers one model for the delineation and assignment of such privileges.

3.3.2 Deploying VO Software as a Virtual Machine

The aim of this model is to remove the need for the installation of the VO software and to minimise the number of platforms, e.g. operating system combinations, on which VO-specific software needs to be supported and tested. This is increasingly important since the installation and maintenance of tools is a time consuming task that may require a range of complex software systems to be configured and provisioned. This is further exacerbated when new software releases are to be installed and when previous releases kept to ensure backwards compatibility. Whilst overcoming dependency and configuration issues, the downside of this model is that larger image sizes can also increase the network overhead caused by transfering the image to the host systems and longer start-up time of the virtual machines on which they are to run. The CERN Virtual Machine File System (CernVM-FS) [152] is one example of such a system. CernVM-FS is a baseline virtual software appliance for the participants of European Organization for Nuclear Research (CERN) at the Large Hadron Collider (LHC) experiments [153]. The appliance represents a complete, portable and easy to configure user environment for developing and running LHC data analysis locally and on the Grid, independently of operating system software and hardware platform.

As depicted in Figure 3.1, the CernVM-FS operates by provisioning nearly all software analysis versions required at all main experiments for the LHC by providing a smaller baseline image size of about 1 GB. CernVM-FS realises this by decoupling the underlying operating system from the experiment-specific software. With the help of tools such as rBuilder [154], a minimal operating system is sufficient to run a given application. CernVM-FS exploits the fact, that a typical analysis job only requires a small fraction of files from a specific release. From a VO user perspective, all files appear to be stored in the VO-specific virtual machine. However, in reality the required files are downloaded on demand from a proxy using HTTP and cached in the virtual machine.

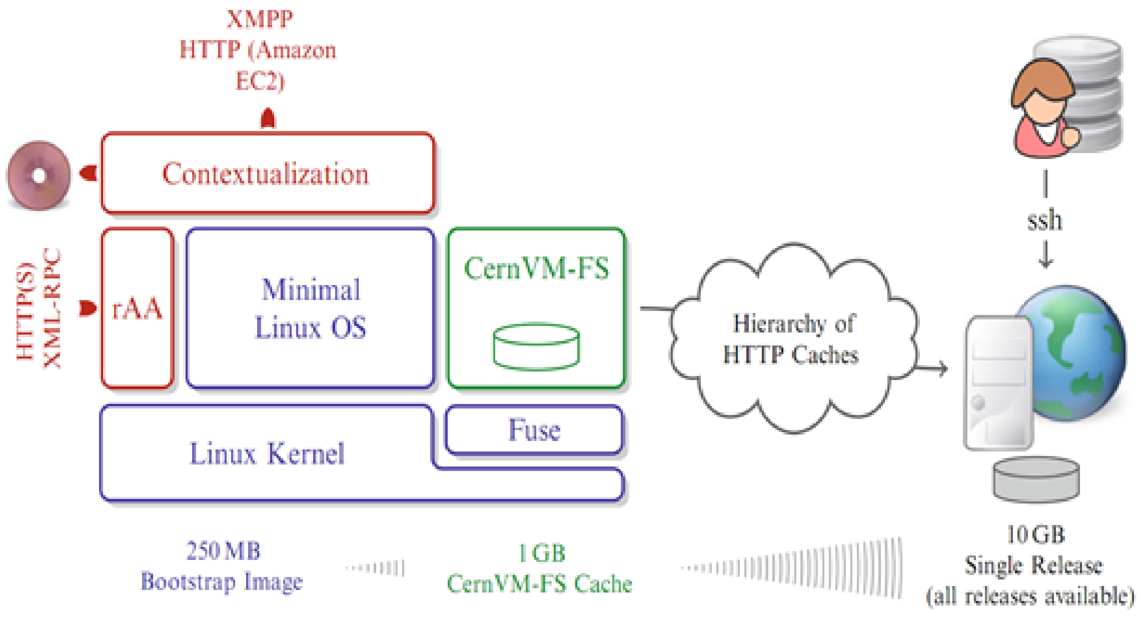


Figure 3.1: The key building blocks of a CernVM-FS image [8]

In this model, a site may let its worker nodes mount a CernVM-FS containing all software (and calibration files) that may be needed by any job for the VO at hand. This mechanism is used in particular by the ATLAS (A Toroidal LHC Apparatus) [155] and LHCb [156] VOs, at sites that support CernVM-FS based solutions. The Compact Muon Solenoid (CMS) [157] VO is also moving towards this approach for its software deployment. A VO-administrator can also potentially allow users to install software on-the-fly along with the job submitted. The A Large Ion Collider Experiment (ALICE) [158] VO is moving its sites to this method, making use of a BitTorrent server at CERN as the seeder, while each worker node then serves other worker nodes at the same site.

3.3.3 Using a Virtual Box Service

A site may have a Virtual Box Service (VOBOX) [159] that also has a mounted shared area accessible via GSISSH [160]. A VOBOX is typically used to provide additional services not provided as the core of the VO, e.g. a data subscription service that allows a site to subscribe to receive a subset of the data produced at another site as soon as it becomes available. A VOBOX can be established so that it is only accessible for privileged users, e.g. the VO-administrators of the supported VOs. VOBOX services (depicted in Figure 3.1) can either be targeted by remote software installations or run a daemon that takes care of access and usage. The latter is the classic ALICE [158] VO model.

To show the applicability of ACVAS, this section outlines a typical example. More advanced

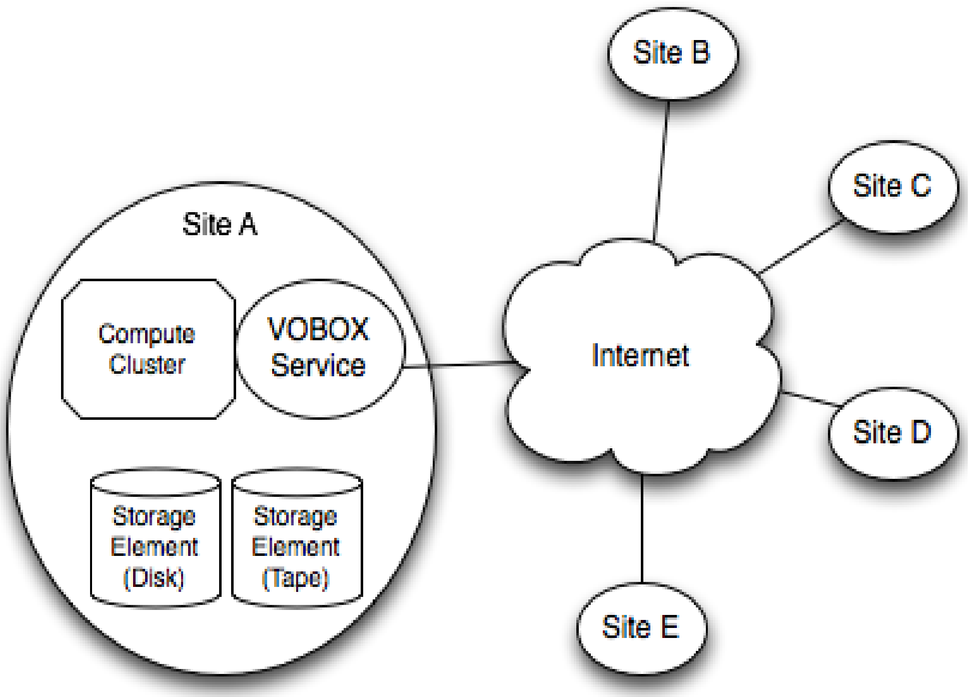


Figure 3.2: The VOBox Service

scenarios are evaluated in more detail in Chapter 7.

Example

Bob is a system administrator at the University of Edinburgh with the responsibility for checking for vulnerabilities on different nodes connected to a Grid facility located at Edinburgh, having a head node which is running Globus toolkit (a web service container for Grid services). Bob is aware that exploiting the vulnerability of the head node provides an opportunity for a remote attacker to submit thousands of roge jobs and subsequently prohibit legitimate users to submit jobs for processing. Bob runs a vulnerability check on a shared server (running globus job manager on GT) located at the Grid facility University of Manchester. Prior to running any vulnerability checks on a shared resource, the following idealises the vulnerability remediation process in the above case. Remedying a globus head node vulnerability Bob needs to identify himself at the Grid site located at Manchester. Since Bob does not have a local ID at Manchester Grid and cannot authenticate him locally. But both the Universities at Manchester and Edinburgh are part of the existing collaboration. As shown in Chapter 4, Figure 4.7, the VulMIS helps Bob with following five main functional areas:

- Whether a given host or range of hosts are currently active

- The operating system installed on the targe node
- Currently active ports on monitored hosts
- Services running on these ports
- List of installed packages on target hosts

Once Bob has gathered the remote nodes properties including(operating system, installed packages, running applications and open ports) using VPM module in ACVAS, he can proceed to find out if a specific target or set of target nodes are at risk due to attacks (considering a missing and/or vulnerable package installed or any port has been left open).

Request Types

Assigning a role to Bob

Running a vulnerability scan on a target node

Accessing central VulMIS results

Update VulMIS with newly discovered vulnerable nodes

Configure the affected node (in case there's high vulnerability found)

Is there any risk involved?

Any dependency?

A typical vulnerability scan request is of the form:

Allow Bob who is having credentials issued by site located at Edinburgh that asserts(assures) Bob has an ***"admin"*** role with the ***"vo admin"*** local role granted with permission P as {*select action on hosts table*}.

We can write this request as a predicate as follows:

authorise((credential issued by Edinburgh

that asserts entity has admin role), voadmin,hosttable(select))

The specification of these predicates are described in detail in Chapter 5.

3.4 Open Issues and Challenges

One of the basic requirements for collaborative platforms is how a VO administrator makes available VO-specific software across multi-site VO-resources. This in turn raises the question of how such software is known and advertised in the information systems or indeed made available to administrators more generally? A privileged user may be able to publish VO-specific software information in VO-wide information systems. Such information can subsequently be used to allow Compute Elements (CEs) to be selected for jobs that need particular versions of particular software available on the worker nodes across a VO. If CernVM-FS is used for a given VO at a given site, that site just needs to have CernVM-FS advertised for that experiment, since it means that any software that is relevant for the experiment will be available.

Another issue is how a VO-administrator can install, validate or remove VO-specific software on a particular VO-site? To support finer grained access control and privilege management, in the EGEE/LHC domain privileges are expressed by a targeted VOMS roles. Such roles are used when creating the VOMS proxy credentials (X509 credentials with VOMS attribute certificates embedded). This privileged VOMS role is typically mapped to a privileged local account that has write access to the shared area that is available to the given experiment (VO).

In other domains, e.g. health, the access to and use of resources across a VO can demand that much finer grained access control systems are in place reflecting the sensitivities and information governance concerns of the resources and stakeholders involved. Even in this domain however it is essential that the underlying fabric is made as secure as possible. In scenarios such as those, key challenges include how to define and enforce VO-specific policies to achieve the VO-specific goals keeping in view that sites are autonomous and one cannot dictate such issues. Other scenarios also exist where a particular VO might demand that participating site(s) provide a *'clean site'* tag (making sure that there aren't any security issues) prior to establishment of the target VO collaboration.

3.5 EGEE-based Vulnerability Assessment

To understand vulnerability assessment practices in collaborative environments, the example of the Enabling Grids for E-sciencE (EGEE) project is given. In this model permissions are assigned to appropriate security personnel, e.g. VO and/or site security admininstrators that enable them to conduct security audits consisting of vulnerability assessment and configuration. This might comprise limited penetration tests against a given sites VO-specific resources.

EGEE example Site-specific policy[1]

- A site-administrator should:
  - provide and maintain a central repository containing all information for resources available on the site;
  - be responsible for monitoring licensed software violations at the site;
  - keep track of successful (e.g. through penetration tests) and unsuccessful access attempts (log information) from authorized and unauthorized users;
- Sites themselves are authorized to examine any files residing on any servers/workstations owned by the site
- Site admins may run intrusion detection software and other monitoring tools to assess any threats and security vulnerabilities and investigate any possible security incidents to connected servers, head-nodes and worker-nodes. They should also ensure that vulnerability information of resources is accessible to suitably privileged individuals and ensure that stored vulnerability information is protected from modifications from unauthorized individuals.
- A site may reserve the right to terminate any scan jobs and associated user jobs that appear to be operating beyond their authorisation limits and/or are not in compliance with this policy.

[1]The detailed policies are discussed in Chapter 5, section 5.5

EGEE VO-Specific Policy[2]

- Before submitting jobs (scanning, configuring and applying a patch) at a resource belong to a site, the VO admin must:
  - have approval from the target site;
  - confirm that the target site(s) share their vulnerability status reports with VO administrators (particularly with regards to VO-specific software);
- Each scan job is the responsibility of one of a limited number of authorised and registered members of the VO, e.g. the VO-administrator;
- The VO is responsible for implementing a process for authorising jobs to VO members and ensuring that they adhere to the conditions laid down by the participating sites;
- The scan job owner and the VO on behalf of whom the job is submitted, are held responsible by the site for the safe and secure operation of the scan job and its associated VO member job(s);
- The scan job must only execute user jobs belonging to registered and authorised members of the VO;
- The scan job must use the approved system utility to map the application and data files to the actual owner of the workload and interface to local site authorization, audit and accounting services;
- The scan job must respect the result of any local authorisation and/or policy decisions, e.g. blocking the running of the user scan job;
- The scan job must not attempt to circumvent job accounting or limits placed on system resources, for example the execution of more parallel jobs than is allowed;
- When fetching a user specified job and credentials into a worker node, the scan job must use means at least as secure as the original scan job submission process;
- The sites reserves the right to terminate any scan jobs and associated user jobs that appear to be operating beyond their authorisation limits and/or are not in compliance with the site policy. Other possible consequences include blacklisting of users or the VO as a whole.
- The VO and/or scan job owner must produce and keep audit logs and assist any Grid Security Operations in security incident responses.

[2]VO-specific policies are described in Chapter 5, section 5.5.2

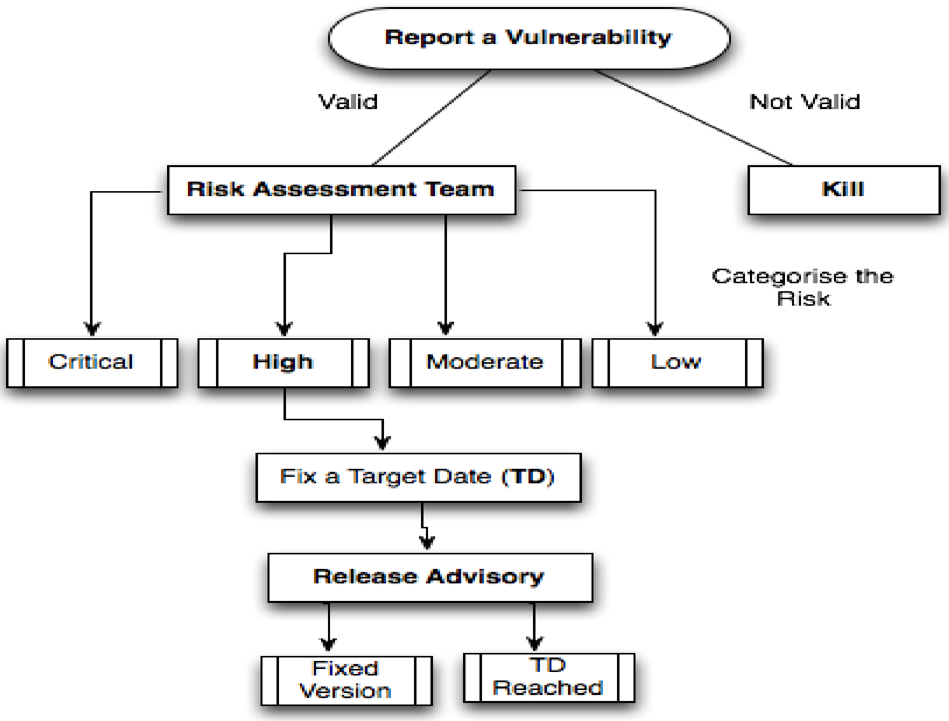


Figure 3.3: The EGI mechanism for software vulnerability reporting/handling

The basic process reporting and addressing vulnerabilities at CERN is depicted in Figure 3.3 and is explained as follows:

- Anyone VO participant may report a vulnerability by e-mail to the EGI vulnerability group;
- The Risk Assessment Team (RAT) along with the reporter and developer investigate the issue to see if it is indeed valid;
- If a reported issue is found to be valid, the RAT places the issue in one of four risk categories – Critical, High, Moderate, or Low;
- According to the risk category, a fixed target date for fixing this vulnerability is set;
- The site administrators and/or VO-developers should try and fix the issue by the target date;
- An advisory is released when a fixed version of the software is released, or on the target date, whichever is the sooner.

The CERN document on software vulnerability analysis [161] describes this process in more detail, and defines the responsibilities from the point of view of the various parties involved, i.e. the reporter of the issue, the software vulnerability group, the software providers, the EGI middleware support unit, security operations teams and the VO sites involved.

3.6 SUMMARY

To prove that Grid technologies and different configuration management tools could be integrated, we have explored in this chapter some representative scenarios. As highlighted in [162], due to deficiencies in authorisation systems currently used in the Grid environments, many researchers and organisations are dissuaded from adopting such technologies. The Grid as a whole has to be as secure as the security of any given site, yet be manageable for administrators and end users alike. The process for applying for and acquiring X.509 certificates and converting them into an appropriate format is itself a hurdle that a large swathe of the non-Grid research community will not overcome. Shibboleth offers one possibility through which the usability factor for end users can however be addressed.

Another important issue that needs to be considered is fabric management, i.e. the layer upon which VO is established. Grid-based collaborations will be considered a threat if the security of the underlying fabric is ignored. For VO-based collaborations to be truly secure, blindly trusting participating sites to take necessary steps in their own local security and patch management is naïve. Tools are needed to assess the contributed software and security infrastructures of participating sites before VOs are established. This demands an integrated approach towards the security and integrity of Grid applications, the middleware and the underlying operating systems which host the logical layer of VOs. Questions such as, will a given site want to collaborate in a particular VO with another site if they allow telnet access, or they are using an older version of software with known security holes? Should a given site share their vulnerability scan reports with other sites prior to establishment of VO collaborations? How can a site measure the "*cleanliness*" of a given site prior to committing to collaboration?

This necessitates the use of vulnerability assessment and configuration management tools to be brought to the fore in understanding the establishment, management and monitoring of VO fabrics. These should ideally be used in a collaborative workbench through which vulnerability reports and recording the usage patterns of the users in a given VO is achieved as well as systematically measuring the overall security of the VO and its associated resources. Exploratory work covering this area and some of the basic scenarios were presented by Sinnott et al. in[163] and [164].

Chapter 4

Design and Architecture of Automated Configuration and Vulnerability Assessment System(ACVAS)

4.1 Objectives and Design Overview

This chapter presents the design overview and objectives of the Automated Configuration and Vulnerability Assessment System (ACVAS). The design is divided into three main components which are described in detail along with their justification. In this chapter we discuss a representative scenario whereby a particular VO admin finds a vulnerability in a resource and proactively applies the ACVAS approach to mitigate the vulnerability. The chapter concludes with a discussion on other similar approaches towards patch and vulnerability management; justifying the need for an integrated approach suitable for distributed collaborative domains.

4.1.1 Design Objectives

Correct configuration and management of hardware and software is important for any collaboration but it is so specially in e-Research domains such as e-health and other similar security-oriented domains. For further growth and adaption of these collaborations it is necessary to ensure that the underlying resources (fabric) are properly configured so that an attacker does not exploit any weakness to undermine the VO and/or the spirit of such collaborations. To address these issues, the ACVAS framework has been designed with the following objectives in mind:

- **To support a configuration-enabled and proactive approach towards vulnerability management:** to tackle research challenges in Chapter 1, the primary goal of ACVAS is to enable VO system administrators to proactively identify site weaknesses (vulnerabilities) in an automated way and to remedy any configuration problem.

- **To support vulnerability discovery in VO-resources or the underlying fabric:** to know whether a particular VO resource is in vulnerable state or has any misconfiguration is vital for organisational security. A key goal of ACVAS is to enable site administrators to discover vulnerabilities and misconfigurations more easily.

- **To ensure that vulnerabilities found are addressed properly across the whole VO:** to ensure that vulnerabilities and misconfigurations found in organisational resources (assets) are addressed properly, ACVAS supports existing security vulnerability and configuration standards such as Open Vulnerability and Assessment Language (OVAL) [63] and Common Vulnerabilities and Exposures (CVE) [165].

- **To ensure that only trusted and authentic patches are used that adhere to local and VO-wide policies:** in a multi-node large scale network environment involving several collaborators (individuals and organisations) and site administrators, it is important that patches are checked for their authenticity, e.g. checking the digital signatures are valid prior to their deployment on target nodes. State-of-the-art solutions for configuration and management of large scale infrastructure such as Puppet [166], Quattor [28] and Cfengine [27] do not focus on patch monitoring and vulnerability discovery. In contrast to these, ACVAS not only provides monitoring of patch status updates and applying patches, but also helps in finding vulnerabilities on target nodes.

- **Where appropriate allow remote system administrators to install patches on a local site:** due to the autonomous nature of VOs, once vulnerabilities are identified and relevant patches are identified and verified, it should be possible for VO resources to be configured and patched by remote VO administrators holding the right credentials and where local policy allows to deploy patches and configure resources. This objective provides aims to address to research question 5, whereby only trusted remote system administrators are able to conduct automated vulnerability assessments, deploy patches and configure VO-wide resources.

As discussed in Chapter 2 (section 2.2) it is essential for site administrators to ensure that all resources in the infrastructure offer a sufficient level of security. However, offering a homogeneous level of security across multiple, often heterogeneous, resources can be challenging. It is also equally important that applying software security updates does not result in significant service downtime and that a sufficient technical expertise and coordination exist. This cannot be always assured. For example the Enabling Grids for E-sciencE (EEGE) [167] project indicated that a number of critical updates were not applied for the following reasons given by Prochazka and Wartel [168]:

1. **Technical issues:** Ineffective mitigation techniques for handling risks e.g. missing dependencies such as hardware vendor did not provided the kernel-level driver critical for a system, leaving them vulnerable due to unavailability of patches;
2. **Communication/Infrastructure related issues:** Faulty information flow, e.g. incorrect roles and responsibilities defined where the staff did not understand or agree with the implications of the risk. Infrastructure related issues also arose, e.g. where systems were partially upgraded or once upgraded services/systems were not restarted;
3. **Staff-related matters:** where insufficient staff expertise existed to identify and/or resolve exploits.

In order to resolve the above issues, ACVAS aims to improve the overall security of VOs and deal with issues of distributed software heterogeneity recognising the autonomy of individual system administators/organisations and their locally defined access/usage and configuration policies. In the following subsection we present a layered representation of a VO as discussed in [9] justifying the need to protect the underlying fabric upon which such a collaboration is established.

4.1.2 Layered representation of a VO

Gould et. al. [9] highlighted the layered representation of a VO shown in Figure 4.1, it comprises of the following layers:

Network layer: The network layer consists of computing and connectivity resources. For VOs to be established, there should be no specialist hardware needed since the "Grid" should be built on top of existing protocols and require no additional or specialist equipment

Resource (Grid layer): The Grid layer is constructed using the Grid specific middleware, e.g. Globus.

Application Layer: This layer is intended to provide the participants both within and outside the organisations with software tools needed for collaborating in one or more "virtual organisations".

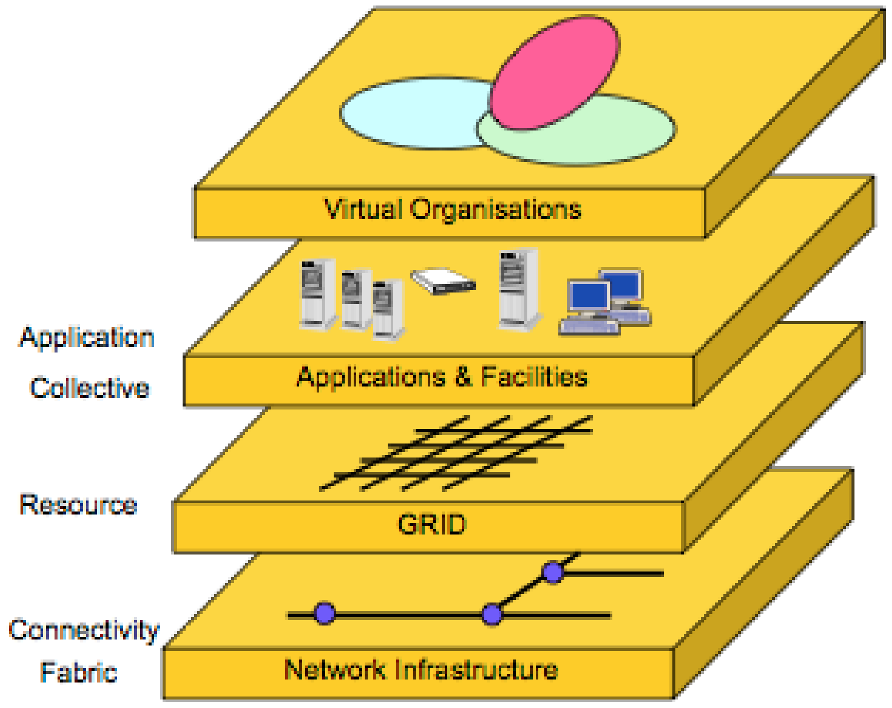


Figure 4.1: Layered representation of a VO [9]

Virtual Organisation Layer: There is body of work which suggest that VO's are constituted around a specific technical goal. This goal is owned by one or more of the VO participants, and only resources, organisations and people that are engaged in meeting that goal are included. This goal-oriented approach helps in defining and controlling the duration of the VO and makes deployment and management of shared resources easier. These goal based organisations are known as Virtual Organisations formed by Goal-Oriented Networks (VOGONs) [9].

As shown in Figure 4.1, it is necessary that the underlying (connectivity fabric) upon which VO collaboration is formed, needs to be protected from malicious activities.

4.2 Architectural Overview

In this section we provide an overview of the ACVAS architecture. In Figure 4.2 a decentralised view of the ACVAS architecture is presented, where the bottom layer resembles the fabric connectivity discussed in Figure 4.1; Configuration Management System (CMS) works at the Resource level (Grid level/ Middleware level), and the Vulnerability and Patch (VPM) layer works at a higher level equivalent to application and facilities layer of Figure 4.1.

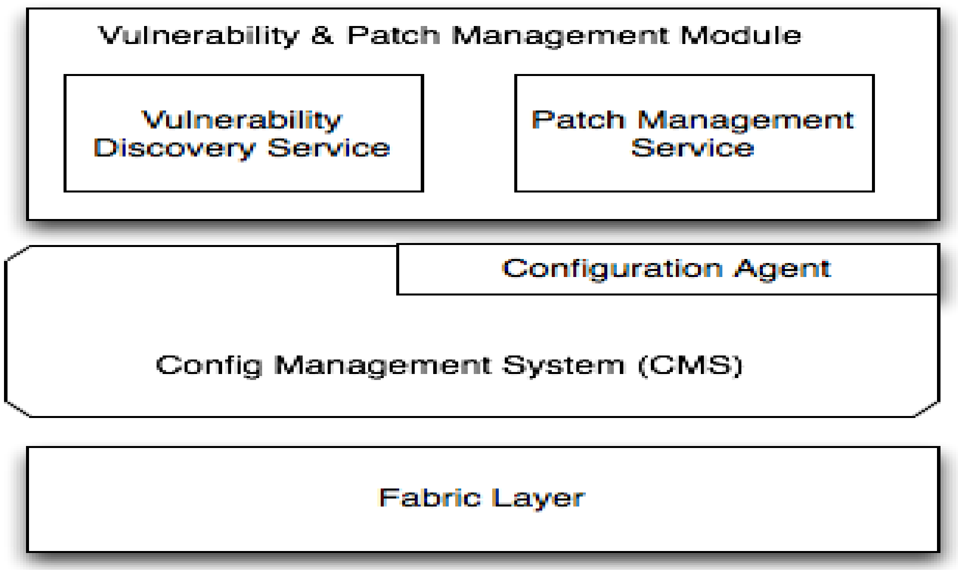

Vulnerability & Patch Management Module
Vulnerability Discovery Service
Patch Management Service
Configuration Agent
Config Management System (CMS)
Fabric Layer

Figure 4.2: Decentralised View of ACVAS Architecture

The VPM module itself is sub-divided into two components: vulnerability discovery and patch monitoring. The top-down view of ACVAS architecture is shown in Figure 4.3, it is made up of three layers. The upper layer is the user interface; the middle tier consists of three modules: CMS, VPM and a scheduler (as depicted in Figure 4.3). The lower layer consists of the remote target nodes (monitored nodes). The User Interface provides two protocols: the first one is for vulnerability discovery and patch status monitoring which helps in finding vulnerable packages installed on target VO nodes. The second protocol deals with configuration management and is involved with identifying and resolving misconfigured hosts (having open ports, appropriate firewall rules not set etc).

In addition to vulnerability discovery and application of trustworthy patches, these protocols also ensure that VPM and CMS modules are operated by authorised system administrators i.e. system administrators in possession of valid credentials. The authorisation based rules and policies are discussed in more detail in Chapter 5.

Since the vulnerability discovery system only identifies that vulnerable packages according to the OVAL [63] definition exist, the actual action of resolving the vulnerability/misconfiguration is the responsibility of the configuration management system.

The implementation of the vulnerability scanning, patch delivery, configuration management system is achieved through the application of Web/Grid services. The lower layer of architecture Figure 4.2 is multifaceted and comprises several components that operate seamlessly and securely so that trusted patch delivery is achievable. ACVAS aims at finding vulnerabilities on a proactive basis, delivering patches that are authentic, valid and able to satisfy an access

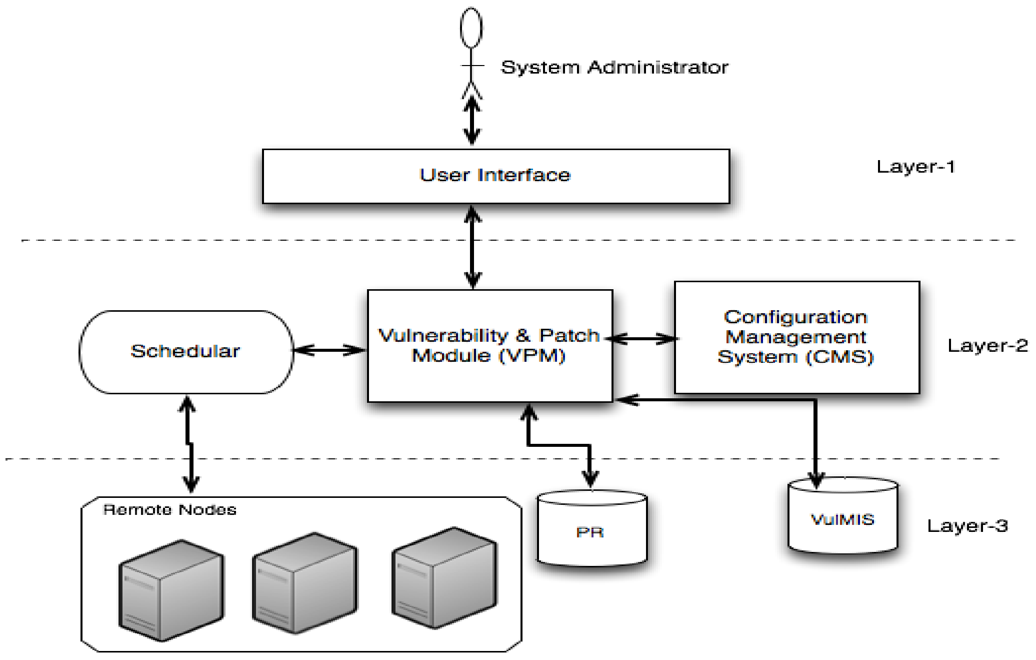


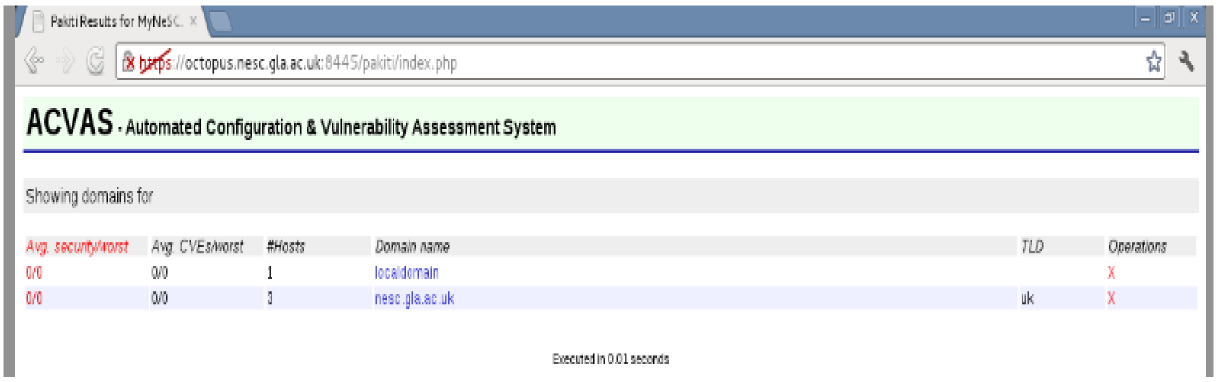
Figure 4.3: ACVAS Layered approach for Integrated Vulnerability Management

control policy defined by remote sites with respect to their configurable policies. To understand the ACVAS components and architecture as a whole, we describe the interplay of the various components of the architecture in the following sections.

4.2.1 ACVAS User Interface

ACVAS provides system administrators with a web based interface (depicted in Figure 4.4) to access and configure/use the underlying components present at the lower layers. It offers a medium to access the VPM (currently based on the open source Pakiti [95] patch monitoring system) and Configuration Engine. Through this interface authorised system administrators are able to:

- run vulnerability scanning (discovering vulnerable packages installed on target remote nodes);
- download the hashes 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.


Pakiti Results for MyNeSC
https://octopus.nesc.gla.ac.uk:8445/pakiti/index.php

ACVAS - Automated Configuration & Vulnerability Assessment System

Showing domains for

Avg. security/worst	Avg. CVEs/worst	#Hosts	Domain name	TLD	Operations
0/0	0/0	1	localdomain		X
0/0	0/0	3	nesc.gla.ac.uk	uk	X

Executed in 0.01 seconds

Figure 4.4: ACVAS User Interface

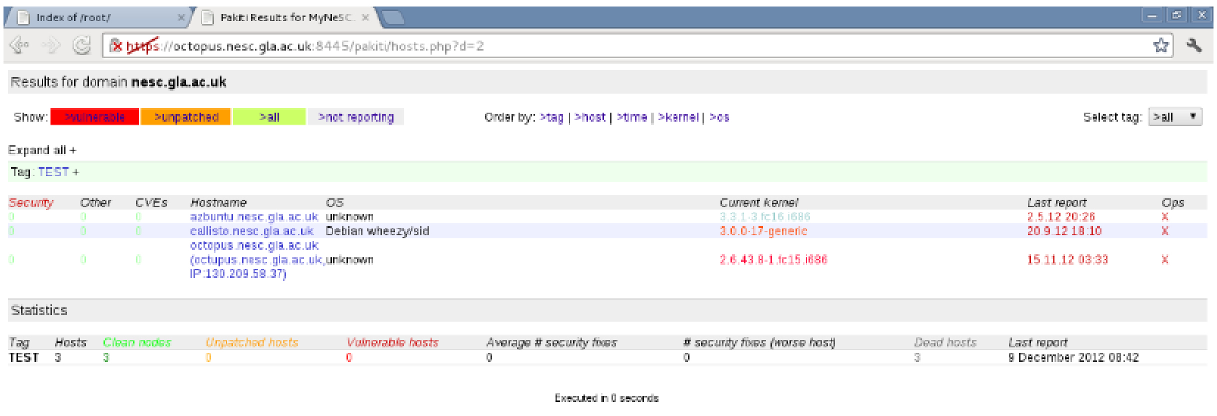
Through this interface organisational policies for monitored nodes are configured to ensure that VPM and CMS are operated by those with suitable VO-specific privileges. ACVAS provides system administrator with a simple web-based interface to monitor the patch status of the target nodes and access the results, i.e. if they are vulnerable or not. Given the nature of the information access to these reports is restricted using access control lists specified in the configuration. The user interface allows site administrators to configure the ACVAS server with settings needed to process the data received from the monitored nodes. The ACVAS server stores all information in a relational database, currently using MySQL. The representative schema is shown in Annex-1 A.

The user interface also provides a list of VO-specific domains to which remote nodes belong and are grouped together with basic statistics showing number of packages which are not up-to-date and which have security updates available.

4.2.2 Vulnerability and Patch Module (VPM)

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 Vulnerability and Patch Module (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. This is achieved by restricting access to these with VO-specific credentials.

VPM is based on a client/server model of checking the patching status of monitored systems. From the monitored nodes perspective, the client agent is responsible for collecting informa-


Index of /root/ | Pakiti Results for MyNeSC

https://octopus.nesc.gla.ac.uk:8445/pakiti/hosts.php?d=2

Results for domain **nesc.gla.ac.uk**

Show: >vulnerable >unpatched >all >not reporting   Order by: >tag | >host | >time | >kernel | >os   Select tag: >all

Expand all +

Tag: TEST +

Security	Other	CVEs	Hostname	OS	Current kernel	Last report	Ops
0	0	0	azbuntu.nesc.gla.ac.uk	unknown	3.3.1-3.fc16.i686	2.5.12 20:26	X
0	0	0	callisto.nesc.gla.ac.uk	Debian wheezy/sid	3.0.0-17-generic	20.9.12 18:10	X
0	0	0	octopus.nesc.gla.ac.uk (octupus.nesc.gla.ac.uk, IP:130.209.58.37)	unknown	2.6.43.8-1.fc15.i686	15.11.12 03:33	X

Statistics

Tag	Hosts	Clean nodes	Unpatched hosts	Vulnerable hosts	Average # security fixes	# security fixes (worse host)	Dead hosts	Last report
TEST	3	3	0	0	0	0	3	9 December 2012 08:42

Executed in 0 seconds

Figure 4.5: VPM monitored nodes patch status

tion about installed packages on local VO-nodes. The information is collected and sent to the central VPM server using secure protocols. On the VPM service end, comparison of actual installed packages against the current list of possible updates takes place, with results displayed as a comprehensive set of information about current status (either vulnerable or not) of all target nodes in the VO. The algorithm for this process is shown in Algorithm 4.1.

The VPM module in some represent is similar to the patch and vulnerability status monitoring infrastructure currently used at the EGEE's Operational Security Coordination Team (OSCT) [169]. 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 in a vulnerable state. By adopting this approach, the VO and site administrators can keep track of sites vulnerable to specific security issues. It is important to note 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.

4.2.2.1 Vulnerability Management Information System (VulMIS)

The heart of ACVAS is the VulMIS depicted at layer three in Figure 4.3 with the internal structure of VulMIS described in Figure 4.4. VulMIS serves as the main repository for all information such as *system properties* and their corresponding *vulnerabilities*, *their potential exploits* and their available *patches*. VulMIS includes three data stores (shaded) and six processes:

data sources include i) *system characteristics*, ii) *vulnerability management information system* (VulMIS) and iii) *system vulnerabilities* containing information about the installed packages and their vulnerability status. The data store *system characteristics* is populated by *installed packages, scan ports*, *scan results of misconfiguration* (firewall or default password

Algorithm 4.1 The analyser algorithm in 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. If (new host)
9. then Get host-information data
10. (host id, installed-packages, vulnerable-packages, versions)
11. Start Transaction
12. Connect to DB (VulMIS)
13. If host already have 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
21. host already in DB (VulMIS)
22. else
23. Create host object (ID)
24. Get Host Group Get host tag
25. Call Handler end
```

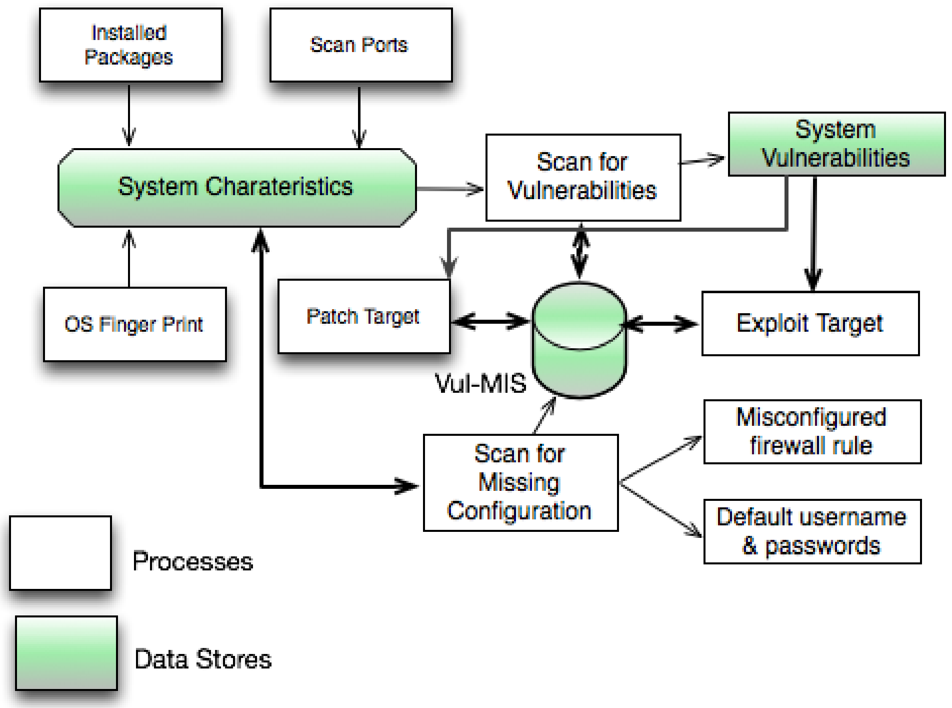


Figure 4.6: Internal structure of the Vulnerability Management Information System

issues), and *OS fingerprints*. OS fingerprinting might be some potential attacker to collect information about a particular host prior to launching the attack.

The *System characteristics* also contains information about specific target nodes such as the *domain name*, *IP address*, *open ports*, *running services* on these ports along with their *versioning* information, *operating system* type and their *version*, *user access privileges*, etc. One of the processes connected directly to *VulMIS* i.e. *Scan for vulnerabilities* takes system attributes as input from the data store *System characteristics*, matches it with the corresponding data from VulMIS and appropriately populates the data store *System Vulnerabilities*. The process *exploit target* takes the vulnerabilities existing in a particular node for a given set of system properties and matches it with *VulMIS* to select a suitable exploit to compromise the target node. The process *Patch Target* selects appropriate patches available in the *VulMIS* and vulnerability information obtained from the VO-system based on the information about existing vulnerabilities in VO resources.

Figure 4.7 depicts VulMIS in the form of an ER-Diagram. Firstly, the entity set *system properties* has *operating system*, running *applications* along with their versioning information, *installed packages*, *ports*, *target IPs* as its main attributes. A distinct combination of these attributes map 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 possibly use a distinct set of payloads to compromise a

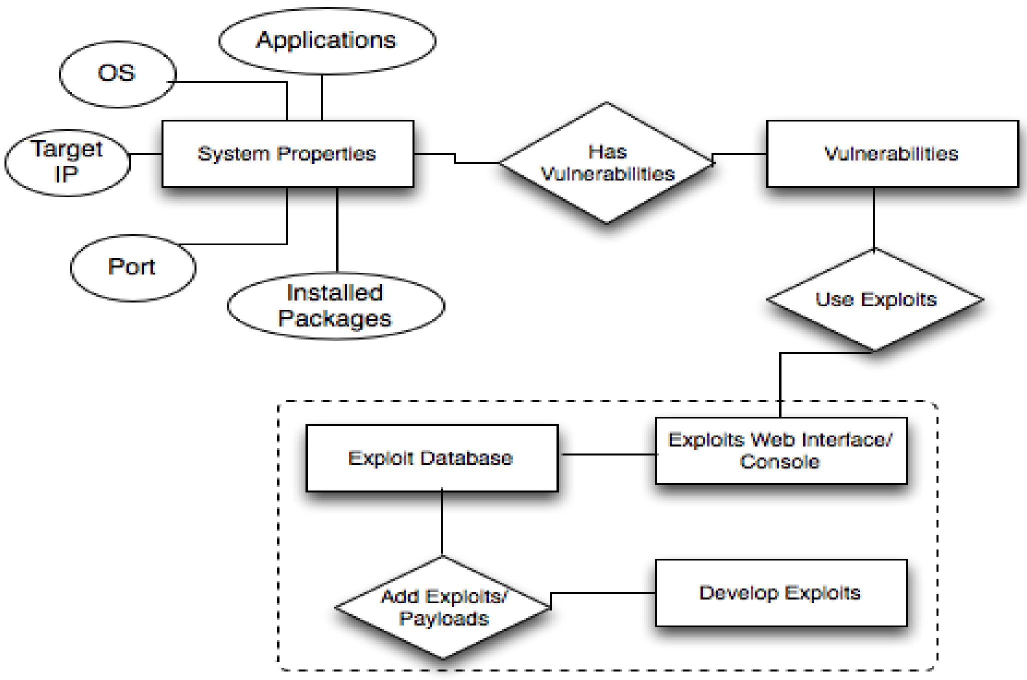


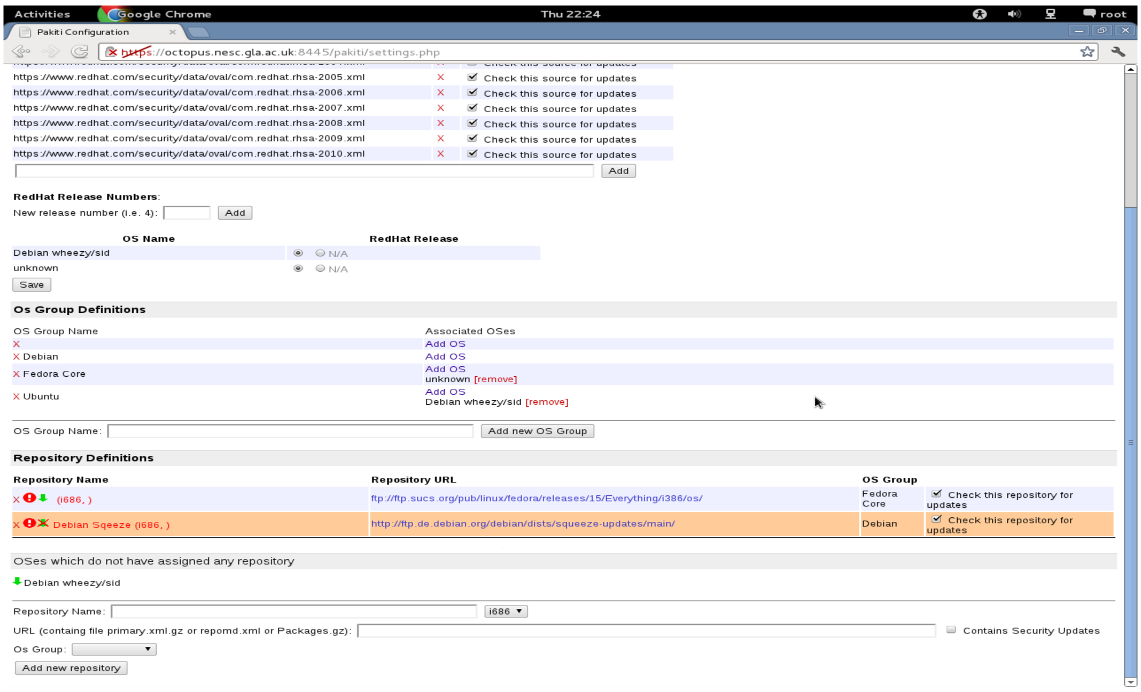
Figure 4.7: VulMIS System Properties, adapted from [10]

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 on manaully depending on VO-specific needs.

4.2.3 Specifying Patch Repositories

As discussed in section 4.2, the Patch Repository (PR) shown in Figure 4.3 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 has been implemented to support remote repositories such as *rpms* [170] based packages from Redhat Linux and Debian and for VO based local repository based on *yum* [171] to make sure these packages are trusted and signed.

To ensure trustworthiness of patches ACVAS VO-system administrator can specify which vendor's package repositories will be observed for up-to-date package versions. As depicted in Figure 4.8, each package repository is assigned to the operating system group it represents. As the ACVAS server checks preconfigured repositories (Figure 4.8), which should be used


Pakiti Configuration

https://octopus.nesc.gla.ac.uk:8445/pakiti/settings.php

https://www.redhat.com/security/data/oval/com.redhat.rhsa-2005.xml	X	✓ Check this source for updates
https://www.redhat.com/security/data/oval/com.redhat.rhsa-2006.xml	X	✓ Check this source for updates
https://www.redhat.com/security/data/oval/com.redhat.rhsa-2007.xml	X	✓ Check this source for updates
https://www.redhat.com/security/data/oval/com.redhat.rhsa-2008.xml	X	✓ Check this source for updates
https://www.redhat.com/security/data/oval/com.redhat.rhsa-2009.xml	X	✓ Check this source for updates
https://www.redhat.com/security/data/oval/com.redhat.rhsa-2010.xml	X	✓ Check this source for updates

RedHat Release Numbers:

New release number (i.e. 4):

OS Name	RedHat Release
Debian wheezy/sid	● ○ N/A
unknown	● ○ N/A

Os Group Definitions

OS Group Name	Associated OSes
X	Add OS
X Debian	Add OS
X Fedora Core	Add OS unknown [remove]
X Ubuntu	Add OS Debian wheezy/sid [remove]

OS Group Name:

Repository Definitions

Repository Name	Repository URL	OS Group	
X (i686,)	ftp://ftp.sucs.org/pub/linux/fedora/releases/15/Everything/i386/os/	Fedora Core	✓ Check this repository for updates
X Debian Sqeeze (i686,)	http://ftp.de.debian.org/debian/dists/squeeze-updates/main/	Debian	✓ Check this repository for updates

OSes which do not have assigned any repository

Debian wheezy/sid

Repository Name: i686

URL (containg file primary.xml.gz or repomd.xml or Packages.gz): Contains Security Updates

Os Group:

Figure 4.8: Specifying patch repositories

on monitored hosts, it can provide information about misconfigured hosts.

4.2.4 Configuration Management System (CMS)

The configuration management system (CMS) module is based upon the configuration management tool CFengine [27]. The CMS is used to install and/or update VO resources based on local and VO-wide patch management policy. For a given VO this would typically require that the resource was vulnerability-free and that only trusted patches were installed and configured by known/trusted (authorised) system 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 to affected target nodes, where they are subsequently checked (authorised) before dynamic installation and configuration takes place. Alternatively, these patches can be pulled by configuration and management clients to the effected target nodes.

As discussed in Chapter 2 (section 2.1.1), the lifecyle of machine [2] ,the primary principle of on which CMS module (CFengine) is based i.e. automatic convergence from Clean or Unknown states back to Configured. As Cfengine is idempotent[1] in its operation, which means that “multiple applications of the operation do not change the result.” Therefore we can write:

[1]It is the property of certain operations in mathematics and computer science, that they can be applied multiple times without changing the result beyond the initial application.

Cfengine(incorrect state) correct state

Cfengine(correct state) correct state

Cfengine runs regularly, bringing the system (monitored nodes) back to the correct state, or keeping it there. For example, making sure a crontab entry is present:

If it is, Cfengine does nothing. If it's not, Cfengine adds it.

Result in either case: crontab entry is present.

4.2.4.1 Declarative Language

Cfengine has its own declarative language. This language is used to write configuration "*policy rules*" or "*promises*" of how the target system should be configured. How to get to the *promised state* is inherently or built-in to Cfengine.

Cfengine can be considered as an "automatic system administrator" that makes the changes necessary to a given system to a "*correct state*". The real system administrator's job is to configure the "automatic system administrator" (ASA) and monitor it's function.

4.2.4.2 Why Cfengine?

CFengine has several features that make it suitable for ACVAS:

Resilience

Since hosts monitored with Cfengine pull their configuration from a policy server, or if the policy server is unavailable or has not been configured, they can be fetched from local cache. In other words, it can run standalone, as as part of a distributed system. Therefore Cfengine will keep running even if there is a network outage or a policy server outage. It will use its cached copy of the last known policy.

Scalability

In a particular environment comprising multiple heterogenous operating systems and applications where each host needs to be responsible for carrying out checks and maintenance on itself, based on its local copy of policy. Cfengine supports this model [172].

Lightweight

Cfengine is small and lightweight in its process size (cfagent is 1.9 MB) and used on everything from supercomputers to embedded devices. As such there is no disadvantage of running heavy agent on VO-nodes.

4.2.4.3 Key CFengine Concepts used in CMS Module

CFengine has various concepts that are key to ACVAS and that underpin this work more generally.

Promises

In the current version of Cfengine key concepts included Promises and Patterns of Promises.

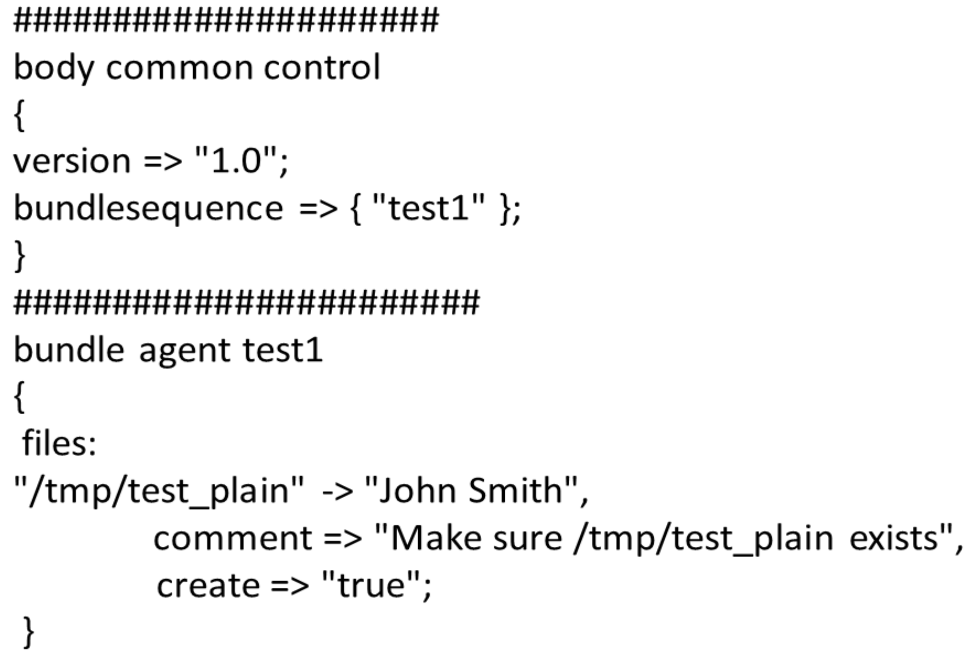
A promise is a declaration about the state we desire to maintain for a particular host, e.g. the permissions or contents of a file, the de(installation) of package or presence/absence of a service. A promise is a Cfengine policy statement. For example, that *"etcshadow"* is only readable by root and it implies Cfengine will endeavor to keep that promise.

Pattern

A configuration is a design arrangement made with system resources. CFengine makes it easy to describe and implement patterns using tools like *lists*, *bundles* and *regular expressions*. While promises are things that are kept, the efficiencies of configuration come from how the promises form simple re-usable patterns.

A promise example is described in Figure 4.9.

In Figure 4.9, the first part, is the mandatory control section, where certain values can be set which promise bundles to run. The next section is a promise bundle comprised of a group of one or more promises. The bundle type "agent" means it will result in an action on the part of CFengine, e.g. executed by the CFengine agent. Each promise bundle has a name, and that name is referenced in the bundlesequence.



```
######################
body common control
{
version => "1.0";
bundlesequence => { "test1" };
}
########################
bundle agent test1
{
 files:
"/tmp/test_plain" -> "John Smith",
          comment => "Make sure /tmp/test_plain exists",
          create => "true";
 }
```

Figure 4.9: A CFengine promise example

4.2.5 Configuration Agents

Target nodes (managed/monitored clients) are connected in a client/server model where configuration-agents are deployed on each monitored node. These agents gather and upload current installed packages in XML format to a central server. On a scheduled interval, the central server (CMS) retrieves updated hashes of available updates from remote vendor sites into a local VO-specific repository. Administrators can request a "patch status" for managed nodes. The central server advertises the available patches and configuration changes to the managed nodes through its scheduler which runs continously to make sure all VO-clients pull necessary changes from the central VO-server. The managed nodes connect to the central server and download the patches from the appropriate location (based on their own local policy). The CFengine configuration-agent installs the patches and rectifies any misconfiguration at a predetermined time and re-boots (if required) the node (if necessary).

4.3 ACVAS Architecture

The focus of this chapter is on the detailed design and realisation of the various components of the ACVAS architecture. The ACVAS is driven by fine grained security and recognition

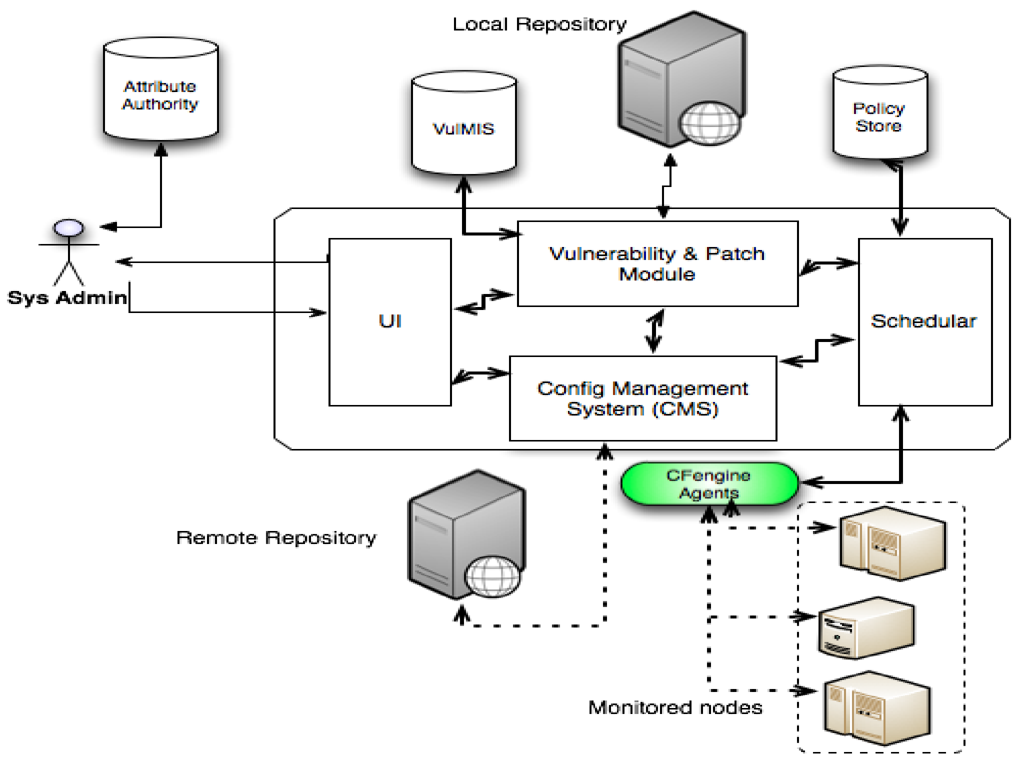


Figure 4.10: ACVAS System Architecture

of local and VO-wide policy. The chapter reviews VO-specific security policy needs and outlines how the ACVAS has been adapted from the CFengine and Pakiti frameworks to provide integrated and trusted delivery of patches to target VO-nodes.

4.4 Configuration Management System

ACVAS provides an infrastructure where Open Vulnerability and Assessment Language (OVAL) based vulnerability information can be securely obtained from monitored hosts and subsequently translated into policy rules interpretable by secure configuration management systems. In doing so ACVAS architecture provides autonomic maintenance mechanisms for collaborating VO organisations and their nodes/resources.

Internally the configuration management system consists of set of components responsible for manipulating vulnerability assessment and performing various configuration processes. These include:

• *cfservd*: the CFengine daemon which must be running on all hosts;

• *cfagent*: which performs the actual changes, e.g. pushes patches to remote hosts;

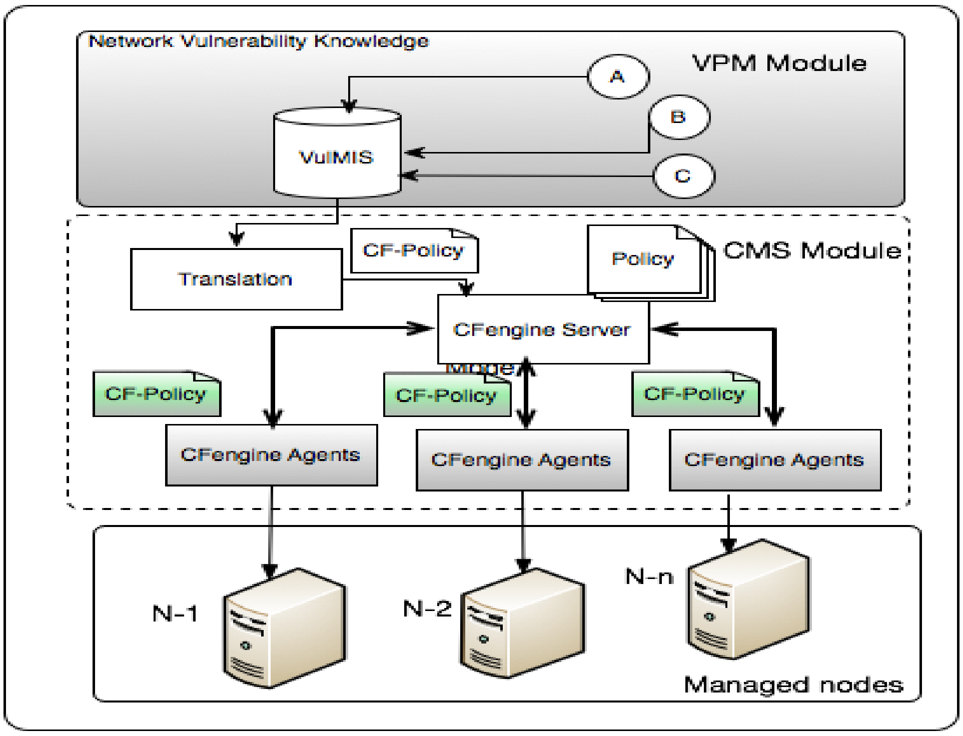


Figure 4.11: CMS Internal mechanism

• *cfexecd*: the CFengine scheduler, which is used to capture *cfagent* output and send notifications

• *cfrun*: pushes updates to clients using cfagent. cfrun is a tool for executing one or more remote agents; it contacts the cfservd daemon running on a remote host, which (in turn) starts an authorized agent.

• *cfenvd*: records system activity in a database which can be viewed using the cfenvgraph program.

4.4.1 Principles of CMS security

As with CFengine [27], the CMS adheres to the following design principles:

1. It shall be, by design, impossible to send policy-altering data to a config agent. Every target host (client) shall retain its right to veto policy suggestions at all times. This is called the *Voluntary Cooperation Model*, Cfengine allows local control of policy in anticipation of consensus building amongst human administrators. By voluntary cooperation we mean that the monitored clients always pull policy from a server and there is never an overwriting push from the server. A policy push is often indistinguishable

from attack. It is worth noting that CFengine(discussed in detail in section 4.3.2) has had just 3 security vulnerabilities in 17 years due to this principle [173].

2. CFengine supports the encryption of data transmitted over the network by default;
3. Each host functions as far as possible, without the need for communication with other hosts;
4. CFengine uses a lightweight peer model for key trust. No centralized certificate authority shall be used. SSL and TLS shall not be used;
5. CFengine provides safe defaults, that grant no access to other hosts.

In Figure 4.11, CMS internally follows a three tiered approach. At the top layer are various sources of vulnerabilities such as Bugtraq, CVE and CERT. These are checked for the latest vulnerability present in different vendor products. The monitored hosts package information is stored in VulMIS as discussed earlier in section 4.2.2. Once a vulnerability report is generated, by comparison with the installed packages and newly received vulnerability information, the translation module generates appropriate policies for the CMS. CM-agents subsequently pull their required configuration from the central CMS server.

In CMS a simple, private protocol that is based on (but not identical to) that used by OpenSSH [174]. It is based on mutual, bi-directional challenge-response using an autonomous Public Key Infrastructure. When communicating with the centralised server, CM-agents use the mandatory Public Key for authentication. However, the data transmitted from monitored hosts to server is encrypted.

4.4.2 The connection sequence (From agent to server)

1. A client attempts to connect to port 5308
2. Server examines IP address of connection and applies rules from
   - allowconnects
   - allowallconnects
   - denyconnects
3. If host is allowed to connect, read max 2048 bytes to look for valid hail
4. Client-agent sends its hostname, username and public key to server

5. Server checks whether public key is known

   If known, host and user are confirmed, go to access control

   If unknown, use trustkeysfrom rules to check whether we should accept the client's asserted identity

6. If not in trustkeysfrom list, break connection

7. If willing to trust, go to further checks

8. If skipverify is set, ignore reverse DNS lookup checks else check asserted identity by reverse DNS lookup

9. If fails break off

10. Check user ID is in allowusers

11. If fails break off

12. Go to file access control

13. Process admit first then deny

14. Mapping of root privilege on server is governed by maproot. If this is false, only resources owned by the authenticated user name may be transmitted.

15. If *ifencrypted* is set, access is denied to non-encrypted connections.

16. Symbolic links to files are not honoured by the server when computing access.

17. Access control is evaluated by the rules:

    (a) First admit rule that matches wins
    (b) All other admit rules are ignored
    (c) No admit rule means you're denied!
    (d) Then look at deny rules (overrides admit)
    (e) First deny rule that matches wins
    (f) All other deny rules are ignored
    (g) No deny rule means you're admitted

4.4.3 Authenticating CM-Agents

Authentication is about making sure users are who they claim they are. Traditionally, there have been two approaches to authentication. Firstly, where a trusted third party (arbiter of the truth) decides whether two individuals who want to authenticate should trust each other. Much web-based authentication follows this approach.

Secondly, the challenge-response approach where each individual is allowed to decide whether to trust the other. This is the approach used in CMS (as adopted from CFEngine design) and is based on *OpenSSH*.

During the key exchange between a CMS client and server, the server has to decide if it will trust the client by using the *"trustkeysfrom"* directive. The *trustkeysfrom* directive allows the server to accept keys from one or more communicating nodes (monitored hosts).

On the monitored nodes end (client-side), the client also has to specify if it will trust keys from the server by using the *"trustkey"* directive. This trustkey directive allows a client to decide whether to accept keys from the server. The CMS authentication model is based on the ssh scheme, however unlike ssh, the authentication process in CMS is not interactive and the keys are generated by a *cf-key* program instead of '*ssh key-gen*' program. Similarly, the key acceptance is itself accomplished using a *trust-key* method.

The authentication mechanism between CM-agents and server is implemented by using RSA 2048 public key encryption for authentication. This in turn uses a 128 bit random *Blowfish* encryption key for data transmission is used with the challenge response verified by an MD5 hash.

4.4.4 Configuration Management System trust model

The trust model used in CMS (described in Section 4.3.2) is to some extent at odds with the external firewall concept. For example the CMS entity says: "I am my own boss. I don't trust anyone to push me data." In reply, the firewall says: "I only trust things that are behind me." The firewall thinks it is being secure if it pushes data from behind itself to the DMZ. A CMS thinks it is secure if it makes the decision to pull the data autonomously, without any orders from some potentially unknown host. One of these mechanisms has to give if firewalls are to co-exist with CMS.

From a firewall's point of view, push and pull are two different techniques: where a push requires only an outgoing connection from a trusted source to an untrusted destination. On the other hand, a pull necessarily requires an untrusted connection being opened to a trusted server within the secure area. For some firewall administrators, the latter is simply unacceptable (because they are conditioned to trust their firewall). But it is important to evaluate the actual risk. We have a few observations about the latter to offer at this point:

As noted by Aleksey [173]it is not the aim to advocate any one method (push or pull) of update. It would depend on organisational needs. However, evaluating security implication is required. For example exporting data from the secure area to the DMZ automatically downgrades the privacy of the information being transmitted. In the CMS, the assumed security model is about the security of every host to be taken seriously in an organisational context. Nevertheless, a firewall should never be used as a substitute for host security. It would be ludicrous to suggest that an arbitrary employee's machine is more secure than an inaccessible host in the DMZ.

4.5 Vulnerability Management Information System(VulMIS)

The Vulnerability Management Information System (VulMIS) is the most important part of the ACVAS architecture that 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 18 different tables including the hardware, software and potentially other third party software (such as anti-virus and anti-spyware) across the VO nodes. The VulMIS is populated when clients send their data synchronously to the server to provide central monitoring. However access to the VulMIS is restricted using access control lists based on VO-specific attribute. Access to and use of this resource is strictly authorised as determined by each VO itself. In VulMIS packages are installed with the help of operating system specific package manager tools such as *yum* [171], *apt-get* [175] and *up2date* [176]. It is worthwhile mentioning that VulMIS contains those packages installed via multiple package management software including yum, apt-get etc.

VulMIS requires details of exactly what kind of information is required across the VO. ACVAS has been designed to be flexible in this regard. The typical core set of information includes:

• The name of packages (RPM) available;

• A listing of what OS each package was available for. Noting that some software would only need to be compiled for one or two operating systems. For example, Red Hat Linux systems already have installed out of the box several packages which can be used for building other systems ;

• Versions of software packages used for the VO and information on the underlying system requirements (and exceptions);

• A list of every software package that should be installed on every node for the VO;

• A field in each node's profile that allows specification of versions of software packages and whether this differs from the currently deployed software.

In our proof-of-concept design inspired from [10], the database has seven main entities namely:

An entity relationship diagram which gives an overview of the relationship between the data items used in VulMIS is given in Figure 4.12. Some of the key fields from the VPM module include:

• Vulnerability Entity – is a textual description for the vulnerability, identifiers such as CERT ID, BugTraq ID, CVE ID and also other keys corresponding to other tables.

• Vulnerability Specifications Entity – vulnerability consequences, hardware requirements, name of vulnerable application/service. A service could be a daemon.

• Environment Specifications Entity – existence of required user/application or service objects which may be exploitable.

• Application/Services Entity – name of application/ service, application/service ID, vulnerable versions, hardware requirements. Services can have additional fields like protocols, port numbers, etc.

• Operating System Entity – similar to application entity but for the operating system.

• Exploit Entity – actual exploit including the privileges needed to execute exploit and the consequences of using the exploit.

In Figure 4.13, the VPM module processes the gathered OVAL information from target monitored hosts and stores the results in the VulMIS database. The Grid middleware module connects the VPM to the Gird infrastructure where it can submit jobs and collect outputs. The collector module transfers the job outputs to the VPM module. The scheduler module is responsible for synchronising the VPM and CMS.

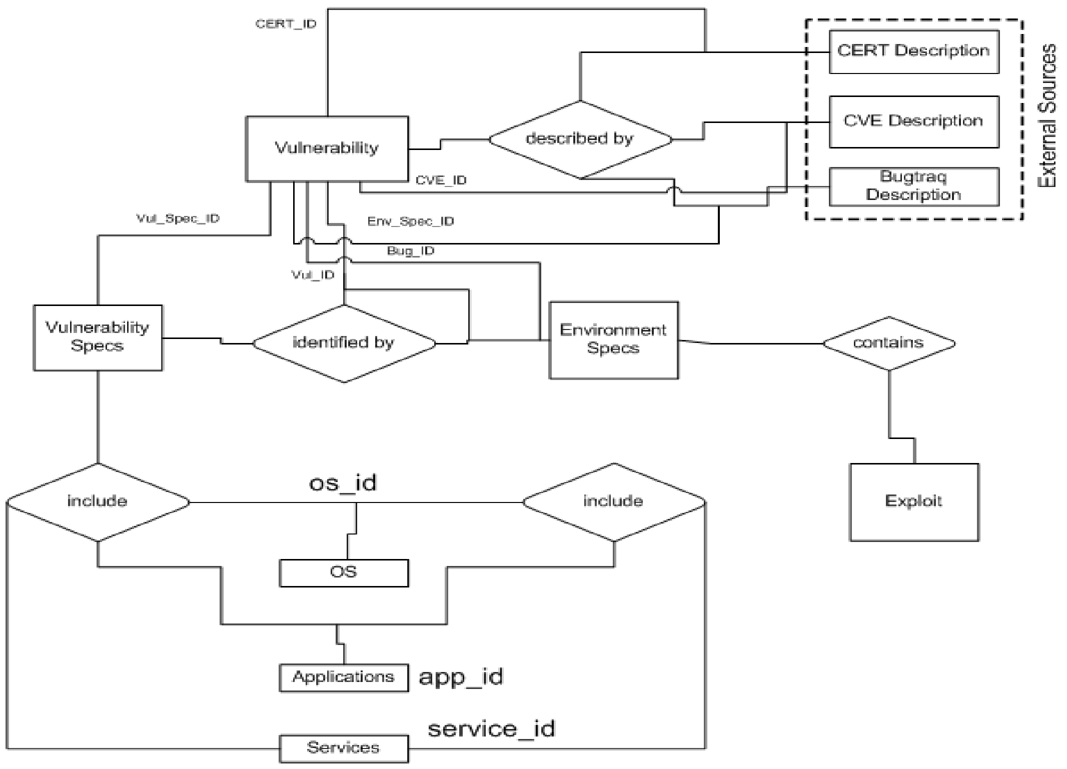


Figure 4.12: VPM module database structure inspired from [10]

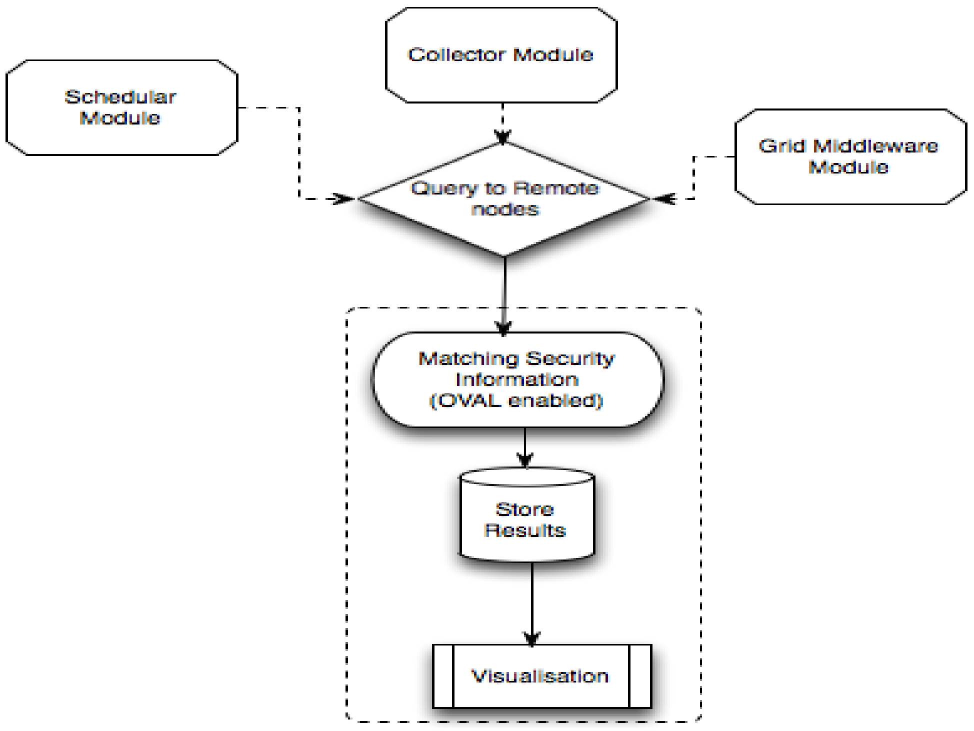


Figure 4.13: Flowchart of VPM Component Interactions

4.5.1 Analyser

In the VPM module, the analyser is used to monitor the patches status of target VO-nodes. Its functions include scheduling the scans for target VO-nodes, managing configuration inventory and storing information obtained from remote nodes in the inventory. The analyser gathers, converts and stores security knowledge from multiple sources such as OVAL and CVE in OVAL format. It then initiates the logic based comparator to match the obtained information from remote repositories and those of target nodes stored in VulMIS. A significant advantage of the ACVAS 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.

4.5.2 Scheduler

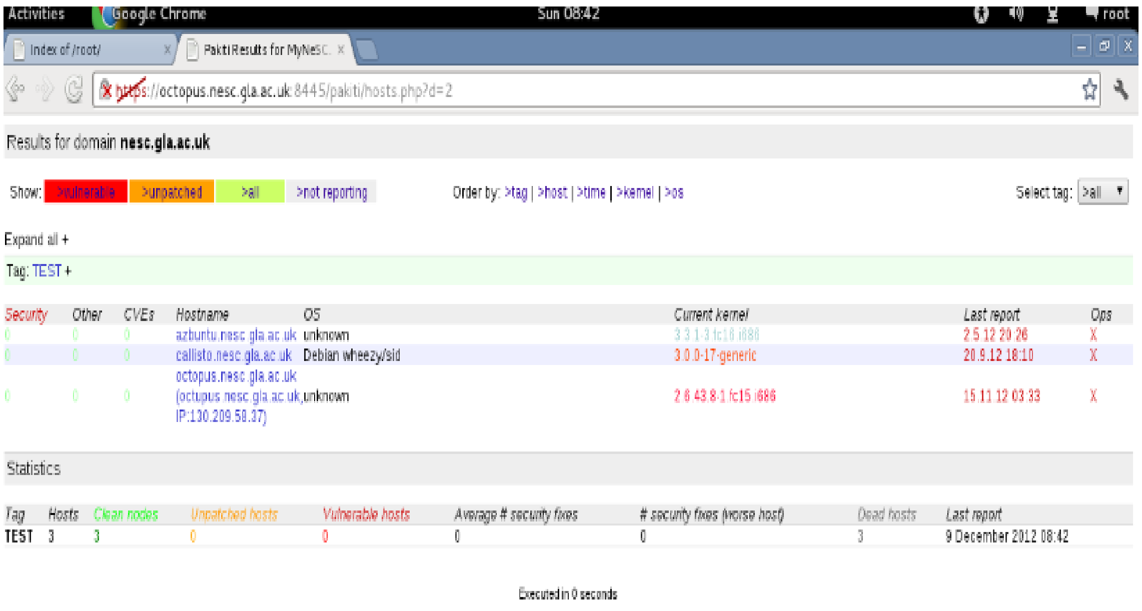
A VO-wide operating system's built-in scheduler periodically runs to makes sure that the VPM server is available to clients so that they can report any vulnerabilities or when updates are available. The scheduler also checks that the server daemon running all the times.

The collector unit transfers the actual data from remote clients to a Pakiti module using the cURL protocol. cURL is an open source program used for transferring data with http or https protocol. The VPM module logs every important event and information and then matches security data from OVAL and then writes the data to MySQL based database.

As depicted in Figure 4.3, the scheduler provides a glue (bridge) to the upper layers to connect the 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 nodes and VPM module sending the remote nodes data to the centralised VulMIS.

4.5.3 Patch Repository

ACVAS supports RedHat OVAL definitions because other vendors (e.g., SuSE) do not follow the OVAL standard, which makes the OVAL processing impossible at the moment. A package/host is considered vulnerable if a package version is lower than one in a package repository containing security updates assigned to the host's operating system or the package with the current version is listed in some Common Vulnerabilities and Exposures (CVE) definition. CVEs in the RedHat OVAL definitions are categorized by their severity. ACVAS uses



Activities Google Chrome Sun 08:42 root

Index of /root/ | Pakti Results for MyNeSC

https://octopus.nesc.gla.ac.uk:8445/pakiti/hosts.php?d=2

Results for domain **nesc.gla.ac.uk**

Show: >vulnerable >unpatched >all >not reporting Order by: >tag | >host | >time | >kernel | >os Select tag: >all

Expand all +

Tag: TEST +

Security	Other	CVEs	Hostname	OS	Current kernel	Last report	Ops
0	0	0	azbuntu.nesc.gla.ac.uk	unknown	3.3.1-3.fc16.i686	2.5.12 20:26	X
0	0	0	callisto.nesc.gla.ac.uk	Debian wheezy/sid	3.0.0-17-generic	20.9.12 18:10	X
0	0	0	octopus.nesc.gla.ac.uk (octupus.nesc.gla.ac.uk, IP:130.209.58.37)	unknown	2.6.43.8-1.fc15.i686	15.11.12 03:33	X

Statistics

Tag	Hosts	Clean nodes	Unpatched hosts	Vulnerable hosts	Average # security fixes	# security fixes (worse host)	Dead hosts	Last report
TEST	3	3	0	0	0	0	3	9 December 2012 08:42

Executed in 0 seconds

Figure 4.14: ACVAS colors to show the severity of vulnerable packages

different colors to show the severity of CVEs. It should be noted that the severity ranking used by RedHat (and others) is mainly focused towards desktop systems and thus unsuitable for our purposes. For example, escalations to root privileges in Linux kernels are often not considered critical, while rather minor browser vulnerabilities score much higher. The ACVAS addition of CVE tagging allows an administrator to tag the severity of a CVE. An administrator can define any number of tags and assign them to CVEs. Hosts found to have packages related to a marked CVE can be viewed on a single page in the GUI. Using the tagging mechanism it is directly possible to detect machines exposing vulnerabilities deemed critical and not rely solely on the severity score assigned by an operating system vendor.

ACVAS also supports severity of packages compiled manually and installed from a package, which cannot be check directly with any repository or OVAL. If a package is vulnerable and a local administrator locally fixes the vulnerability, they will usually add some additional string to the version of the new package leaving the original version number to distinguish the package from the vendor's one.

ACVAS is able to show all package versions installed on monitored hosts according to the particular CVE, making it easy to select versions that should not show up in the reports.

4.5.4 Processing Data from Remote Nodes

As shown in Figure 4.16, the processing of installed packages start with inputing the configuration file to VPM server. Once data has been received by VulMIS, it extracts the parameters

```
# Monitoring System for CRITICAL status
body common control
{ bundlesequence  => { "check_listening_ports"  };
  inputs  => { "Cfengine_stdlib.cf"  }; }
##################################################
bundle agent check_listening_ports
{
vars:
"listening_ports_and_processes_ideal_scene"
string =>
"22 sshd 80 httpd 443 httpd 5308 cf-server";
# this is our expected configuration

vars:
"listening_ports_and_processes" string =>
execresult("/usr/sbin/lsof -i -n -P | \
/bin/grep LISTEN | \
/bin/sed -e 's#*:##' | \
/bin/grep -v 127.0.0.1 | \
/bin/grep -v ::1 | \
/bin/awk '{print $8,$1}' | \
/bin/sort | \
/usr/bin/uniq | \
/bin/sort -n | \
/usr/bin/xargs echo", "useshell");  # this is our
# actual configuration.
# we tell Cfengine to use a shell with "useshell"
# to do a command pipeline.
classes: "reality_does_not_match_ideal_scene" not =>
regcmp (
"$(listening_ports_and_processes)",

"$(listening_ports_and_processes_ideal_scene)"    );
# check whether expected configuration matches actual.
reports:
reality_does_not_match_ideal_scene::
"
DANGER!!!
DANGER!!!  Expected open ports and processes:
DANGER!!!  $(listening_ports_and_processes_ideal_scene)
DANGER!!!
DANGER!!!  Actual open ports and processes:
DANGER!!!  $(listening_ports_and_processes)
DANGER!!!
"; # and yell loudly if it does not match.
   # Note:  A "commands" promise could be used in
   # addition to "reports" to send a text message
   # to a sysadmin cell phone, or to feed
   # CRITICAL status to a monitoring system.
}
```

Figure 4.15: A CFengine policy bundle for monitoring system in CRITICAL status [11]

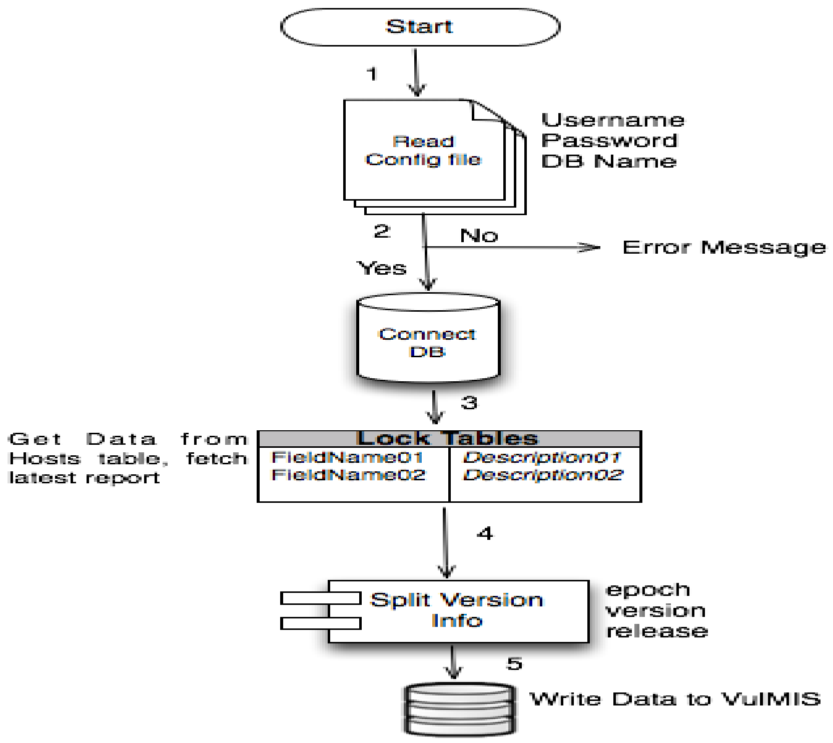


Figure 4.16: Processing data from remote monitored nodes

e.g. (username, password and database name) prior to actual processing. As a first step it begins with extracting and parsing configuration files containing parameters. In case of incorrect parameters an error message is returned.

Assuming that the parameters are correct (step 2), the request is connected to the DB. To match the data from remote nodes the hosts table is fetched via a SQL query and tables are locked so that any other parallel request/action do not over write actual processes. Once the latest host report data is fetched from the local repository and compared with the hashes of packages received from remote repositories, the packages are split into version and release levels. The verified packages (free from vulnerabilities) are written back to the VulMIS database and updates occurs.

4.6 ACVAS Data Flow

In Figure 4.17 a detailed view of the data flow between ACVAS components is shown for a single VO. In this section we enumerate the interactions between these components.

1. System admin puts a request to the attribute authority for credentials required for access to the VPM server for their own specific VO.

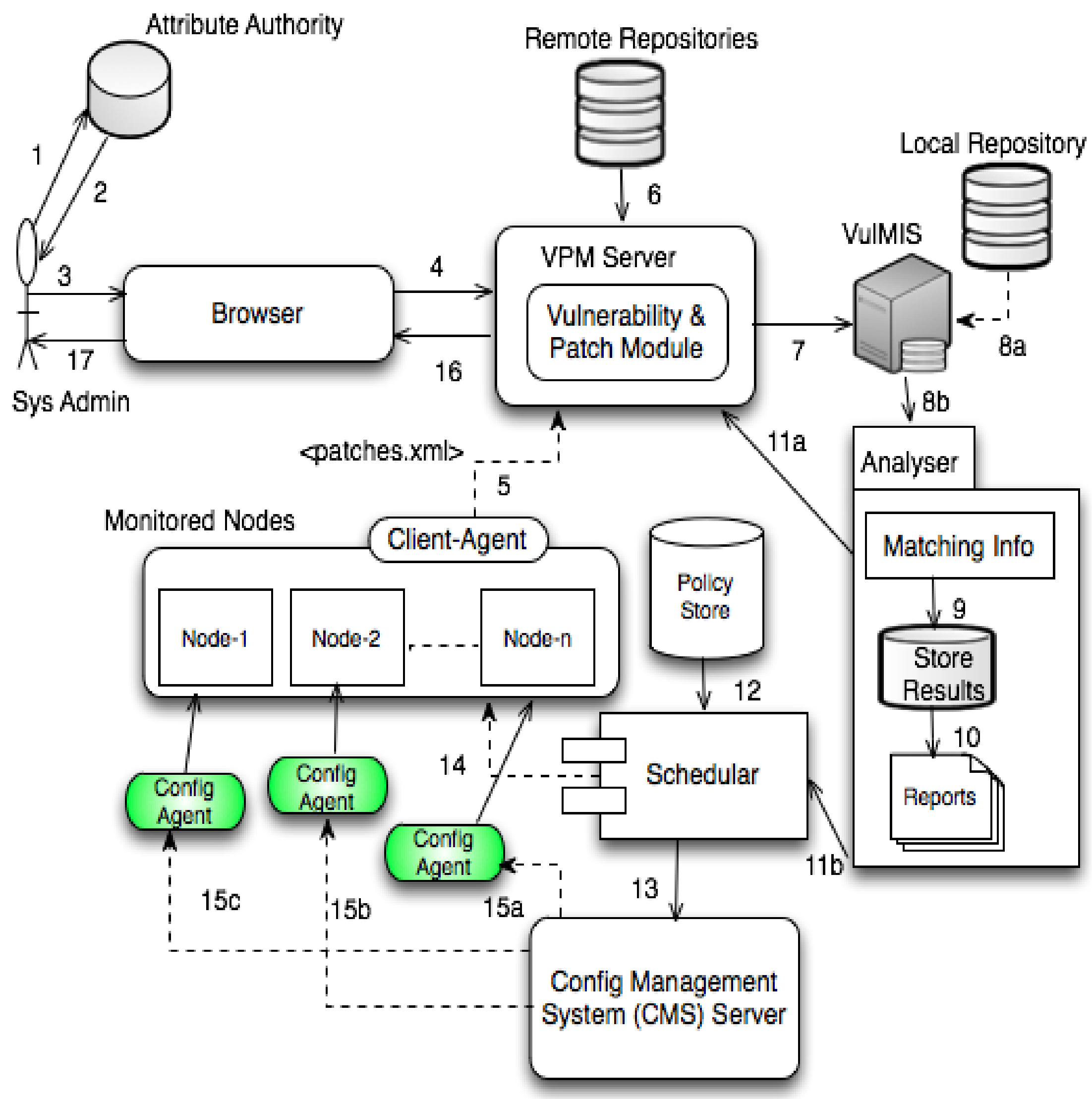


Figure 4.17: The ACVAS Data Flow for a single VO

2. **Passing Credentials**: The attribute authority pushes credentials of the authenticated sys-admin. This will include VO-specific attributes/privileges.

3. **Requesting Information:** System administrator puts a request to check a monitored node/nodes.

4. **Attribute Verification**: The secured interface at the VPM module confirms the request by checking its local acess policy for this particular request.

5. **Background process (via VO Client-Agents)**: The client-agent provides a list of installed packages (*patches.xml*) to the VPM module. The scheduler makes sure that the VO-client-agent is running on all monitored nodes.

6. **Getting Updated (Hashed) Data from Remote Repositories**: The VPM module receives updated hashes of the packages installed on monitored nodes by remote repositories. The VPM makes sure that the trusted patches from vendors are available for the analyser to compare with the installed ones.

7. **Writing data (patches.xml) to Vul-MIS**: The VPM module saves the data received from remote nodes in its database.

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

9. **Analyser** (**8b**): The actual analysis phase starts from this step onwards (the Algorithm is shown in depiction 4.16).

10. In steps (**9 & 10**) packages are matched, results stored and reports of existence of vulnerabilities are generated.

11. In step (**11a**) reports are sent back to system admin via the VPM module.

12. In step (**11b**), in case a vulnerability is found, reports are sent to system admin and the scheduler is activated.

13. The CMS Policy store (**12**) makes sure that high vulnerability reports are comply with organisational policy.

14. The scheduler automatically triggers (**13**) the CMS to address the vulnerability either by installing an update or fixing any misconfiguration.

15. **Scheduler-to-nodes:** The scheduler accesses (**14**) the monitored nodes via client-agents running on them to make sure the affected clients pull available configuration from the CMS server.

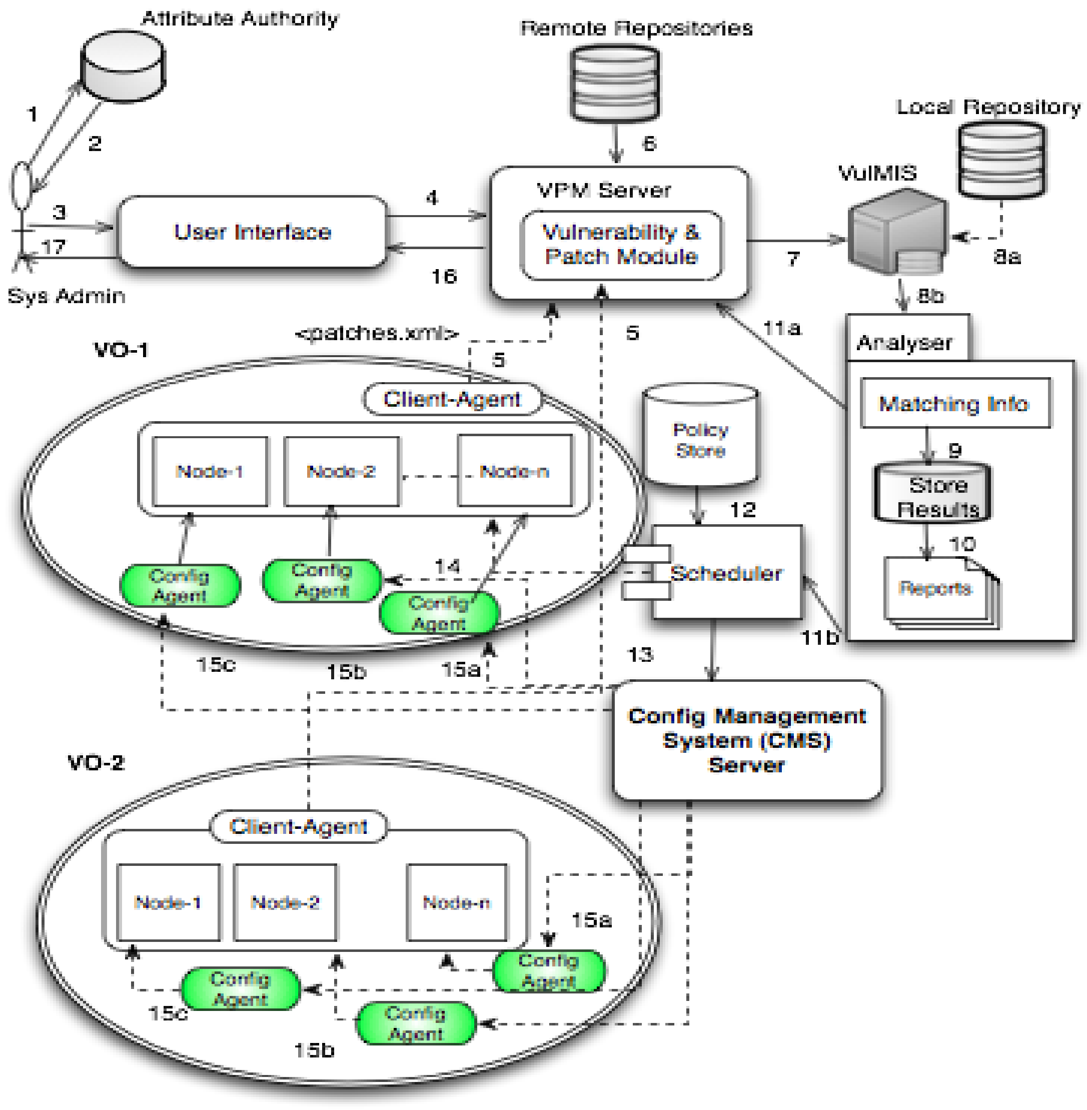


Figure 4.18: The ACVAS Data Flow for a multiple VOs

16. The 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 config-agents.

17. Finally updates on vulnerable nodes (either vulnerable package exists or not) are sent back to the system administrator and step 5 is repeated to make sure that the latest information is written to VulMIS.

4.7 Discussion

In this section we present a comparitive analysis of other work similar and/or very close to the design of ACVAS.

Shibli et. al. [177] conducted a vulnerability and patch management evaluation using secure mobile agents. Their approach suits homogenous (closed) networks environments and does not address resources across multiple domains as typified by Grids. Similarly, the Systems Management Server (SMS) [178] 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.

There are a range of vulnerability scanner programs such as Nessus [100], MBSA [179] and Pakiti [95]. Nessus is a network based vulnerability scanner requiring update plugins to detect vulnerabilities. Nessus does not guarantee against false-positives while scanning systems. Also due to heavy traffic generated during the scan process, the Nessus server can be bottlenecked and in worst cases crash. As discussed in Section 6.6, by adopting a centralised data management for each participating site can potentially be a bottleneck if the size of the organisation grows and/or become a single point of failure. However, this issue can be resolved if multiple vulnerability databases are used on a per site/VO basis.

MSBA is an analyser for windows-based systems and is not suitable for hetergenous domains. Pakiti is an open source solution that is easy to analyze and modify. Pakiti collects information about the security status of resources and matches the gathered information with security data. Pakiti uses secure protocols to communicate. It also provides a graphical user interface. The program has several disadvantages in Grid environments however. For example, Grid administrators have to install the client software on every machine and configure firewall settings. User management is also insufficient (just one admin role is available) and there is no option to maintain the history of monitored nodes. In ACVAS, we overcome the client issues by using CFengine to automatically install and manage client nodes. We also introduced multiple VO-specific roles that can be used to monitor and manage different VO specific resources.

Sandor et. al. [180] designed and implemented Grid Site Software Vulnerability Analyzer (GSSVA) which can automatically explore the installed Linux packages on the Grid based nodes. Their approach uses a modified status monitoring system based on Pakiti [95]. GSSVA collects important information about the status of the nodes and analyses the information gathered from a security point of view. During this analysis it compares the results with an external information repository to find security problems of resources. The application offers a graphical user interface to visualize the information for Grid administrators. In GSSVA, administrators do not need to install client side software or configure firewalls so a patching status monitoring system can easily be set up. GSSVA is similar in some respects to ACVAS, however it doesn't address misconfiguration and configuration management issues nor multiple security-level capabilities.

A key challenge in distributed domains is when local system administrators find vulnerability in their resources, how do (or should) they share that with collaborators? Although there are many efforts being carried out to address vulnerabilities, the NIST's SP 800-40 primarily recommends the implementation of a systematic, accountable and documented process for managing exposure to vulnerabilities through the timely deployment of patches.

However, there is a need for organisations to understand the deficiencies in the NIST model before addressing the changing threat and vulnerability landscape caused by increasingly complex distributed, dynamic and heterogeneous environments such as e-Research and e-Science domains. Such a gradual and systematic approach is often not tenable. In a proactive approach, as a first step prior to deploying patches, other security approaches and technologies may be required such as network and host-based intrusion prevention systems; network and host-based firewalls, and networking devices such as routers configured to prevent attacks. These steps can provide an effective pre-emptive approach to vulnerabilities before any patching is required.

Furthermore, organisations turning to NIST 800-40 for guidance on patch management should ideally look to expand their vulnerability management activities to include networking devices, database applications, and internally developed web applications since many of these cannot be patched in the same manner as mainstream servers and desktops. As noted, the downside of this approach is in potential system disruption [181].

4.8 Summary

In this chapter the overarching objectives of ACVAS were presented. The ACVAS objectives and design was presented and discussed. This included the key ACVAS components and their justification. The chapter described the function and purpose of these components and their associated issues. A discussion on the solutions of ACVAS was presented indicating the benefits and features of Cfengine and Pakiti; a comparison with other similar approaches was given. It was presented how ACVAS meets many of the needs and challenges identified in Chapter 2 and 3. This chapter has provided a detailed architecture for proactively finding vulnerabilities in VO resources and showing how those weaknesses can be preemptlively resolved. The chapter elaborated the internal working mechanisms of ACVAS and outlined how an integrated approach could support continuous security monitoring. It descirbed the CMS components, which included the basic principal CMS security, the connection mechanism from an agent to CMS server, authentication of agents and the trust model adopted in CMS. It also described the components that make up vulnerability assessment module (VPM), which included the schedular, patch repository, collect and analyser. The chapter presented how the processing of data collected from remote nodes is achieved and then stored for permanent storage in VulMIS.

The chapter concluded with a detailed overview of the ACVAS architecture showing the flow of data between various components during vulnerability assessment process for single and multiple VOs.

Chapter 5

Security-oriented Policies for ACVAS

The previous chapter introduced the design and architecture for ACVAS that uses vulnerability assessment and configuration management systems to proactively find weaknesses in VO resources and consequently resolve those vulnerabilities. This approach demands restricted access to authorized individuals. This chapter specifies the syntax and semantics of the policy language used in ACVAS. The basic concepts of authentication, authorisation and roles used in ACVAS are presented and the policy language itself. The use of credentials for exchanging authorisation information between various components of ACVAS is described. The chapter concludes with a discussion on resolution of conflicts between VO and site level policies and how ACVAS can be used to manage this.

5.1 Authentication, Authorisation and Roles

Ensuring the overall security of VOs through tool supported vulnerability scans (assessments) and patch management demands that access to these tools is restricted to those entitled to do so. As discussed in Chapter 2 this entitlement typically comes through authentication and authorization systems.

Authentication is about ensuring that different users/processes in a communication requesting access to resources are who they claim to be (or certified to be who they claim to be). In distributed paradigms such as Grids, authentication processes are based around public key infrastructures and use of certification/registration authorities. This model is well understood and accepted by the community and resource providers alike [182, 183].

However it is not practical to describe authorization permissions or restrictions for every single user. Instead many authorization systems use the concept of "`roles`" which can be assigned to individuals/groups of users. These roles can be used to define actual permissions (allow/deny) for individuals with the role. Roles can be assigned to site-administrators, VO-administrators, developers and production roles as deemed appropriate for the particular collaboration.

A user can be a member of several groups and be assigned multiple roles according to different roles he/she might take as part of the VOs they are involved in. These roles can be assigned by site administrators; by VO-administrators, or others (if delegated authority is supported).

System administrators are typically responsible for assessing site resources for any potential vulnerability and other remedial actions. On the other hand, VO administrators are responsible VO-wide resources. As such, a VO-admin might potentially be authorized to deploy VO-specific software on remote nodes located at different sites, but any other activity, i.e. an unauthorized activity such as running port scanning on resources beyond the VO jurisdiction may be considered as a 'hacking' activity and should be blocked. The ability to specify what is allowed and what is not allowed is thus essential.

As defined in Chapter 2, many role-based authorization systems use role hierarchies to manage the roles and their relationships. In ACVAS, site administrators generally have a superior role to VO administrators when it comes to accessing vulnerability information of managed nodes and/or configuring nodes, i.e. nodes are deemed autonomous. However in exceptional circumstances, over-riding this hierarchy may be required, e.g. if the node is a major threat to other resources and the local system administrator is unavailable.

The Vulnerability Management Information System (VulMIS) of ACVAS discussed in Chapter 4 potentially contains very sensitive data resources, hence it should be protected by limiting access to only selected (authorized) individuals, e.g. only site administrators or VO-administrators can perform certain operations.

Through delegation of authority, software systems can also be assigned privileges, i.e. not just individuals. Thus configuration management software may use site/VO administrators delegated credentials to configure and apply patches to remote VO-nodes. However, certain operations may depend on the mode of operation of the configuration management e.g. urgent patch install, security update.

In all of these scenarios, the ability to define roles/privileges and associate them with tasks/activities and then use this information to define and enforce local and VO-wide policy checks is essential.

5.2 ACAVAS Policy Language

To understand ACVAS security policies it is essential to understand the language used to specify and describe them. ACVAS adopts the rule-based framework described in [184, 185] as the basis for its policy specification. The rule-based framework offers a declarative language based on logic to define rules. The policy language used to describe policies for ACVAS are based on the XACML policy specification language [186].

A clause i.e. (rule) consists of a head and body and takes the form as follows:

$$Rh \longleftarrow Rb \cdots\cdots\cdots (1)$$

In equation 1 above, ‘Rh’ on the left-hand side is called the rule head and the right-hand side ‘Rb’ is the rule body. In the model of equation (1), when Rb is satisfied then it makes Rh true. Typically a rule body contains conditions and when these conditions are satisfied (evaluated as true), then the rule will be enforced and might trigger some actions. If a rule contains an empty body such as Rh . then it is called a fact and is always true, in other words this rule has no condition and is always true.

Using this model, it is possible to encode an authorisation policy in the rule body and the rule head itself can be used for encoding the authorisation decision. For example in equation (2), we can define a policy rule which can decide which subject s can access a resource r:

$$isOwner(s, r) \longleftarrow canAccess(subject(s), resource(r)) \cdots\cdots\cdots (2)$$

In the above example, the function *isOwner (s,r)* evaluates to true, if the subject *s* can access the resource *r*, then access is granted to *s*. In case of a false evaluation, then access is denied.

5.3 Policy Language Rules for Vulnerabilities and Exploits

The policy rules language used in ACVAS are based on Xinming et. al [184] and Becker et. al. [186]. ACVAS policy rules are designed to support and use credentials for vulnerability management. In ACVAS, a credential is considered to be a constrained predicate asserted by an entity called an *issuer* with the entity in possession of the issued credential called a *holder*. A credential with no issuer is considered invalid. In a typical rule, valid predicates are tagged with both the holder and issuer. For example, *AliceGla.p(x)* is a rule which states Alice holds

Policy rule		::=	*(Head, Body Pred, Constraint)*
Head	head	::=	E1◊E2.pred
Body Pred	P	::=	e1◊e2.pred
Credential	cred	::=	head ← c
Rule	rule	::=	cred
		|	head ← P1, ..., Pn, c
Constraint	c	∈	C

Figure 5.1: Formal Framework for ACVAS Policies

a predicate *p(x)* asserted (issued) by Glasgow (*Gla*). In this example the holder (*Alice*) is postfixed with the symbol "◊" and the issuer by "."

The prefix used in the Figure 6.1 for the issuer is similar to the prefix used for roles in Xinming[184] and Becker [186]. The ACVAS framework assumes that a degree of mutual trust exists among the site administrators and VO administrators as part of the VO establishment process. Thus a common authentication model and signing authorities are assumed.

The holder prefix used here is similar to the location prefix used in [186]. In ACVAS it is important to factor in constraints as part of security policy specification. Thus policies in MulVAL [184, 65], specify only the principal who can access what resources and there is no mention of constraints. ACVAS considers each principal and resource as well as the constraints that may apply, e.g. the condition in which access is allowed. This is given a symbolic name (c), which is mapped to a concrete entity by the binding information. A policy statement is of the form allow(`Principal, Access, Resource`).

The typing rules for a policy rule in ACVAS as specified in [183] is as follows:

$$\frac{\Gamma \vdash e : \text{entity} \qquad \Gamma \vdash e' : \text{entity} \quad \Gamma \vdash \text{pred}}{\Gamma \vdash e \lozenge e'.\text{pred}} \quad (1)$$

$$\frac{\Gamma \vdash \text{head} \qquad \Gamma \vdash c}{\Gamma \vdash \text{head} \leftarrow c} \quad (2)$$

$$\frac{\Gamma \vdash \text{head} \quad \Gamma \vdash p \ldots\ldots \; \Gamma \vdash P_n \; \Gamma \vdash c}{\Gamma \vdash e \lozenge e'.\text{pred}} \quad (3)$$

The first policy rule is the rule head, which can be described as: the proposition e ◊ e'.pred is

Authorisation Function	**Description**
has_account (p, h, a)	*principal p has an account a on host h that he/she can access.*
accessFile(p,h, A.Path)	*entity p can access a file located on host h on given path A.Path*
hacl(S, Dest, Pro, DestPort)	*source host S can reach destination Dest using port DestPort*
netAccess(p,src,des,Pro, port)	*entity p can send network packets from source to destination host*
exec_code (p, h, r)	*principal p can execute code with privilege r on host h.*
read (p, h, r)	*principal p can have read privilege on host h.*
write (p, h, w)	*principal p have write w privileges on host h*
vulexist (h, id, p)	*host h has vulnerability id caused by program p*

Table 5.1: ACVAS Authorisation Functions

part of the propositions of a rule head if e, e' and pred are from the same proposition where e, e' are entities.

ACVAS implements policy rules using predicates (functions). A predicate can be defined as a Boolean function. Typically these functions have set of arguments and a return value. The resulting value is true if the conditions for the predicate are satisfied or false otherwise. In ACVAS two categories of policy functions exist which are based on [187]: specification and authorisation functions. For vulnerability assessment policies, ACVAS uses functions as defined in [184]:

We briefly explain each of the functions in Table 5.1 with the help of example.

$$canAccess(p, h) \longleftarrow has\_account(subject(p), host(h), account(a)) \cdots\cdots\cdots (3)$$

In the example given in equation (3), the function *canAccess (p,h)* evaluates true, i.e. if subject *p* has an account *a* on host *h*, then access is granted and *p* can access the resource, i.e. host *h*. In case of a false evaluation, then access is denied.

$$isAllowed(p, h, path) \longleftarrow accessFile(p, h, A.Path) \cdots\cdots\cdots (4)$$

The above equation (4) indicates that *isAllowed (p, h, path)* is true if entity *p* can access a file located on host *h* with a given path *A.Path*. The access will be denied if the path is incorrect.

$$multihost\_Access(p, h, pro, destport) \longleftarrow hacl(S, Dest, Pro, DestPort) \cdots\cdots\cdots (5)$$

Equation (5) states that *multihost_Access (p, h, pro, destport)* evaluates to true if source host *S* can reach destination host *Dest* through protocol *Pro* via destination port *DestPort*. If no path/connection exists between *S* and *Dest*, then *hacl* results in a deny decision.

$$canSend(p, h, des, pro, port) \longleftarrow netAccess(p, src, des, Pro, port) \cdots\cdots\cdots (6)$$

Equation (6) states that the function *canSend (p, src, des, pro, port)* results to true if entity *p* on source host *src* can send network packets from source port *port* to destination host *des* using protocol *Pro*. In case of missing paramter then *canSend* results in a deny decision.

$$canRead(p, h, r) \longleftarrow exec\_code(p, h, r, voadmin) \cdots\cdots\cdots (7)$$

Formula (7) states that the function *canRead* evaluates to true if an entity *p* has read *r* privileges on host *h* and provides role *voadmin* with the read request. If there are no read permissions or role *voadmin* is not presented, then the above rule will result in a deny decision.

$$read(p, h, r, developer) \longleftarrow$$
$$has\_account(subject(p), host(h), accessFile(p, h, A.path) \cdots\cdots\cdots (8)$$

Equation (8) states that the function *read* will evaluate to true if a principal *p* has an account on host *h* and can access a given path where software is installed. In case of any other or different paths the function will result a deny decision.

$$write(p, h, w, voadmin) \longleftarrow$$
$$has\_account(subject(p), accessFile(p, h, A.path), exec\_code(p, h, r, voadmin) \cdots\cdots\cdots (9)$$

Equation (9) states that the function *write* evaluates to true if principal *p* has account on host *h,* can access a file in a given path by presenting a role *voadmin* in the request. In other cases, e.g. where the role is not presented or different or the path is not specified, then the function effects in a deny decision.

$$vulExist(h, prog, range, consequence) \longleftarrow$$
$$bugHype(h, prog, range, consequence) \cdots\cdots\cdots (10)$$

Equation (10) states that a vulnerability exists when the hypothetical bug *bugHype* on host *h* has a vulnerable installed/running program *prog* with a given exploit *range* and some known *consequence*. The *range* in equation (10) can have several possible values, e.g. it might be a *local exploit* or *remote exploit* [184]. The consequences given in (10) can include loss of *confidentiality* and *integrity*; *privilege escalation* and/or potential for *DOS* (Denial of Service). The function will give a deny decision when there is no bug present in a program or in case a patch has been applied to rectify the bug causing the vulnerability.

5.4 Authorisation Rules for Policy Enforcement

Based on these functions ACVAS supports various authorisation rules to define and enforce security policies. These rules are defined in the following section and realized as functions:

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

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

where *e, r, h, a, p* are *entity, role, host, accoun*t 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:*

$accessFile(P, H, Access, Path) \longleftarrow$
$execCode(P, H, User), localfileProtection(H, User, Access, Path)$

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. The *execCode* function is discussed separately in rule 5.

Rule 3 : *A hacl rule is of the following form:*

$hacl(Source, Destination, Protocol, DestPort) \longleftarrow fireWall(s, d, port)$

Rule 3 implies that the *Source* host can reach *Destination* thorugh *DestPort* on host through *Protocol*. HACL is an abstraction function dependent on the physical topology, firewall rules(for example source *s* is allowed to access destination *d* with an open *port*), the configuration settings of other devices such as routers and switches, and so on. Site administrator would typically use a range of tools to obtain such network information.

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

$netAccess(P, H1, H2, Protocol, Port) \longleftarrow execCode(P, H1, User), hacl(H1, H2, Protocol, Port)$

Rule 4 implies that if a principal *P* has local access on machine *H1* as some *User* with *VO-administrator* role and the network allows *H1* to access *H2* through *Protocol* and *Port*, then the principal can access host *H2* 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 [184] 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* `local-privilege-escalation` *rule (*`execCode`*) is of the form:*

As defined in [184] a vulnerability rule can be defined as follows:

$$execCode(VOsite1, Host, User) \longleftarrow malicious(Attacker), vulExist(Host, Prog, localExploit, privilegeEscalation)$$

Rule 5 is vulnerability related which implies that, if a malicious *attacker* with a vague DN or fake email address can first compromise an account (AnyUser) on a target host in the *VOsite1*, and there is a locally exploitable privilege-escalation bug in a program for example 'setuid' which is owned by user 'john', then the attacker can gain the privilege of user 'john' to attack the whole VO.

Moreover, in [184] Ximing defines *execCode* as a server-side exploit and have associated client-side exploits. In many client-server systems, client-side programs establish communications with a server, e.g. a web browser, or an email client etc. For an attacker to exploit a bug in these client-based programs, the precondition is that the victim must initiate a connection to a server that may subsequently allow malicious inputs from the attacker. Example exploit scenarios could be when a compromised web page contains Java programs that exploit the Java Virtual Machine (JVM); when a mail client such as Outlook Express containing worms exploiting vulnerabilities through the mails a server may deliver. For all of the preconditions of these exploits to occur, the victim needs to do something first, e.g. browse a compromised webpage, or open/click a link in a malicious email. The execCode function will thus have a multitude of implementations whose preconditions need to be evaluated carefully to determine potential exploit opportunities.

Rule 6: *A compromised rule is of the form:*

$$principalCompromised(victim, attacker) \longleftarrow \\ execCode(VOsite1, h, code), execCode(VOsite2, h, code), execCode(VOsite3, h, code)$$

This rule implies that the principal *'Victim'* has an account *User* on host *H*. Another principle, the *Attacker* takes control of the host as *root*. In this case VOsite1 allows to run code but the other two sites i.e. VOsite2 and VOsite3 do not allow to execute the arbitary code to run in the VO as "code" being a potential threat and might provide the *Attacker* to compomise the overall VO. For example 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 VO nodes.

5.5 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 [188]. 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

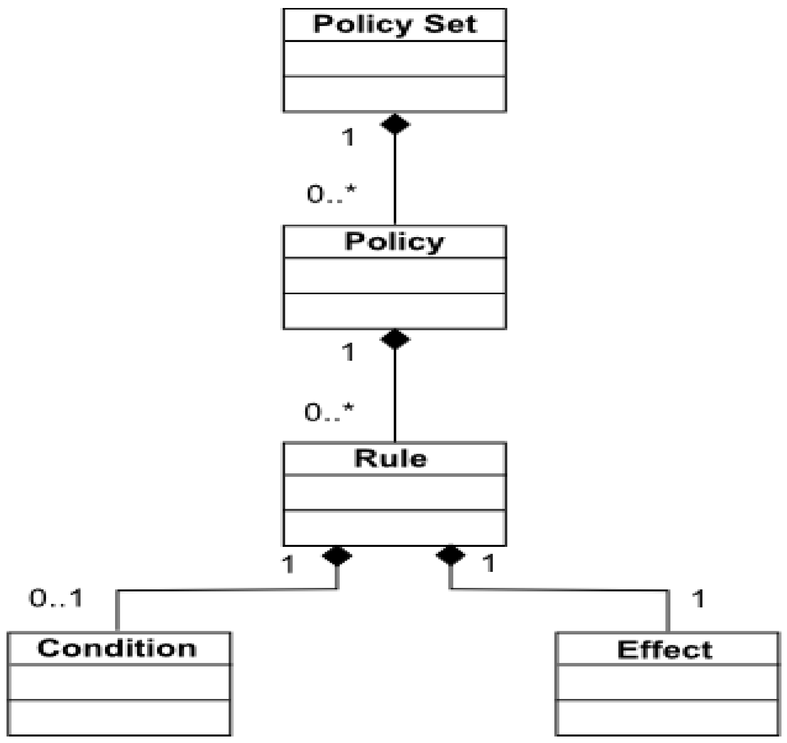


Figure 5.2: XACML policy set

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 *'PolicySet'*.

5.5.1 Local Site-specific Policies

These types of policies put in place by resource providers (nodes) to make access decisions on resources they provide. Once local policies are stored in local policy repositories and made available to local policy decision points (PDP) when making decisions on access request to different protected resources. Sites collaborating in a decentralised environment can also delegate authority to various Attribute Authorities (AA) for managing security policies, and management of privileges.

The tabular representation for some of the local (site) policies is shown in Table 5.2. These cover various entities (subjects), resources (e.g. database tables of VulMIS described in section 4.5 of Chapter 4), which can be accessed subject to privileges. The VulMIS schema is depicted in Figure A.1 in the Annex. With the subject (requester) field, the possible attributes can be roles or user names (distinguished names) or a combination of both. In some cases sites might include obligations, i.e. conditions that must be enforced, as part of their policies.

Subject	**Resource**	**Action**	**Obligation**
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*	

Table 5.2: A tabular view of local (site-specific) policies

Site-specific policies indicate what roles are available and the relevant permissions associated with each role in an organisation. In table 5.2, each of the rows represents a policy rule. Thus the subject *Jan* who has role of site administrator in Glasgow is allowed to select, insert or update actions on *installed packages* in Glasgow.

Site level policies indicate what roles (principals) are available and their associated permissions in an organisational host. Each row of table 5.2 can be enforced as a policy rule. We can describe each of these rules as described in Section 5.4 in pseudocode as mentioned in Figure 5.1 (XML is shown in Annex A2 Figure A.1):

```
Input:  Site-specific Policy
Output: decision (allow or deny)
Resource Match and Match ID = 'Installed Packages'
Rule 1
If {
subject == site-admin
Resource == Installed-Packages
Action Allowed == Full Access
}
Condition {
Attribute Value (subject Attribute(role))
Attribute Values (Resource Attribute(installed packages)}
}
Then Decision == Permit
Rule 2
If {
subject == site-admin
Resource == Installed-Packages
Action Allowed == Full Access
}
Condition {
Attribute Value (subject Attribute(X500 DN))
Attribute Values (Resource Attribute(Jan Muhammad)}
}
Then Decision == Permit
Rule 3
Other than rule 1 and 2
Deny All Other Requests
```

Figure 5.1: Pseudocode for Site specific policy

The above three rules in Figure 5.1 state that if a principal such as siteadmin has local access on VO node1 as siteadmin role and the local site allows then the principal can access node2 through the given protocol and port. These rules may be implemented in XACML policies - normally stored as PolicySets. XACML also uses policy-combining algorithms to aggregate results of individual policies decisions to make final access control decisions. Whatever final authorisation decision is reached and wherever applicable, the obligations with a matching XACML fulfilOn option set are also included when the PDP sends a response to a PEP.

With ACVAS, every rule in a policy is evaluated and its corresponding results are combined to reach a final policy decision. If the evaluated condition is true then the rule outcome can either be Permit or Deny. In situations when a rule cannot be evaluated or if a rule evaluated to false, the overall result of the rule can be Indeterminate or `NotApplicable` respectively. For example, Figure 5.1 defines access control for the Gla:InstalledPackages object (resource). The rules defined in this policy contain the logic of who can access the InstalledPackages resource at Glasgow. The first two rules shown in Figure 5.1 have permit as the value of their "`Rule Effect`". The third rule has a deny value for its "`Rule Effect`" and since the rule has no condition, it is evaluated to be true.

Each subject in Table 5.2 listed to have access to the Glasgow `InstalledPackages` resource can be defined as a rule in the policy. The rule combining algorithm used in the policy (Figure 5.1) was the XACML first-applicable algorithm, i.e. the value of the first rule evaluated to be true is returned as the final result of the policy. Thus Figure 5.1 will return a permit result if the requester has a Site-administrator role or a Common Name equal to Jan Muhammad otherwise it returns a deny result. A sample of an XACML policy that allows a VO-administrator to select only records from the Glasgow `InstalledPackages` table with obligations that an email to be sent to the site admin that a VO-administrator role has accessed this resource is shown in Extract 1 of Appendix A.

5.5.2 VO-specific Policies

VO-specific policies are similarly structured to site-specific policies described in the previous section. However, these are designed for validating and enforcing agreement or trust contract [188] between hosting sites and a particular VO.

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 and services. For example the first row in table 5.3 implies that anyone from the University of Edinburgh (ED) with role of *VO-administrator* issued by and recognized by Glasgow will be able to perform select, insert or update on resources identified with a *vulnerability*. The second row in the table 5.3 implies that *anyone* with 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 pseudocode for VO-specifc policy is depicted in Figure 5.2 defines access control for a *Vulnerable Packages* (resource). The rules defined in this policy contain the logic of who can access the vulnerability resource and what they can do with it. Such policies can be used to define and enforce rules regarding default database passwords, use of vulnerability assessment tools, e.g. to check software versions and identify known security holes, or indeed to allow a

VO-administrator in a multi-site collaboration to provide a '*clean site*' tag. The XML code for VO specific policies is depicted in Annex A2 (Figure A4).

```
Input:  VO-specific Policy
Output: decision (allow or deny)
Resource Match and Match ID = 'Vulnerable Packages'
Rule 1
If {
subject == VO-admin
Resource == Vulnerable-Packages
Action Allowed== Full Access
}
Condition {
Attribute Value (subject Attribute(role))
Attribute Values (Resource Attribute(vulnerable packages)}
}
Then Decision == Permit
Rule 2
If {
subject == VO-admin
Resource == vulnerable-Packages
Action Allowed == Full Access
}
Condition {
Attribute Value (subject Attribute(X500 DN))
Attribute Values (Resoiurce Attribute(John Boyd)}
}
Then Decision == Permit
Rule 3
Other than rule 1 and 2
Deny All Other Requests
```

Figure 5.2: Pseudocode for VO specific policy

5.6 DISCUSSION

However it should be recognized that a range of issues related to description and enforcement of policies for controlling access to and use of dynamic Grid resources remain. These policies

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
VOadmin	Ed:Installed-Packages	Select	
Gla. ACVAS/siteadmin	Gla:Host	Select, Insert, Update	

Table 5.3: A tabular view of VO specific policy protecting vulnerability resource

can be in the shape of permissions or desired usage scenarios allowed by service/resource providers, by collaborating VO members, or even governing bodies for multi-organisational VO collaborations. As noted, many contemporary Grid collaborations have either “in-spirit” usage policies with no actual enforcement, e.g., all resource providers are assumed to contribute in kind [13], or the policy enforcement on resource/service usage is defined/enforced by ad-hoc mechanisms. Collaborations that do define VOs with resource usage policies typically consider only CPU resources, neglecting Grid resources/services such as disk, bandwidth, software and the underlying fabric. As a result of this deficiency, many collaborations are stifled by fear of resources/services being misused and/or overrun [189].

5.7 Summary

In service-oriented and collaborative environments, secure infrastructures are essential. This chapter has described the formal framework that underpins the ACVAS architecture and outlined how it has been implemented using XACML.

In this chapter we have specified the syntax and semantics of the policy language used in ACVAS. The rule-based policy language that underpins ACVAS was discussed. It was shown how a rule consists of a rule head and rule body. A rule body then contains conditions and if these conditions are satisfied, the rule will be enforced/implemented. With this model, a rule body can be used to encode the parts of an authorisation policy and the rule head for encoding what should be enforced by the policy. Based on the specification, policy language functions/predicates were defined for ACVAS. These functions express the various authorisation rules that can used to enforce ACVAS policies. The chapter described how policies defined for ACVAS can include: site-specific and VO-specific policies.

Keeping in view the very sensitive nature of the resources being protected in ACVAS, limiting access to only selected (authorised) individuals is essential. The ACVAS policy language and associated functions can be used to ensure only site administrators or VO-administrators can perform certain operations across particular VOs.

The policies defined for ACVAS are different than those defined in MulVAL [184, 65], as MulVAL specifies that only the principal can access resources and there is no mention of constraints on access and usage. ACVAS considers each principal and resource as well as the constraints that may apply, e.g. the condition in which access is allowed. Moreover, ACVAS differs from Pakiti [95] and Grid Site Software Vulnerability Analyzer (GSSVA) [190] in that it introduces the application of trusted patch delivery and definition of security policies for site/VO administrators. The ACVAS framework provides multiple rules, which protect organisational assets by using trusted credentials and access policies in collaborative environments.

In this chapter we have assumed that there is a minimal level of trust exists between collaborating sites, which enabled us to specify site-specific and VO-specific policies. However there are also several issues with regards to policy specification for vulnerability information and applying patches to affected software. Thus in a VO collaboration the participating sites are autonomous and cannot be dictated to regarding policy specification or indeed disclosure of vulnerability information. Other important questions not covered in ACVAS framework and are considered as future work, include issues arising prior to VO formation. For example, the idea that collaborating sites should agree to disclose their vulnerability information and/or the patch status of target nodes to collaborating sites prior to any collaboration/trust being in place? Moreover, while collaboration is taking place, at what point should vulnerability assessments happen and patching take place? These issues are non-trivial to overcome and comprise both technical challenges as well as process-oriented challenges in how organisations can trust one another more generally. It may well be the case that ACVAS is only applied where it builds upon known and trusted collaborations that have already taken place (and where established trust frameworks are in place). This goes somewhat against the flexible (ad hoc) model of Grid-based VOs where resources and individuals are allow to come and go in a dynamic manner.

Chapter 6

Implementation

This chapter describes the implementation of the ACVAS architecture described in the chapter 4. The implementation comprises several architectural components: a Vulnerability Management Information System (VulMIS) is realized as a database that contains VO-wide resource vulnerability and patch status information and offers a patch status monitoring system based on Pakiti [95, 168] for collecting VO-specific resource information. The chapter is structured as follow. Section 6.1 describes the internal structure of VulMIS database. Section 6.2 describes the prototype implementation of ACVAS architecture discussed in Chapters 4. Section 6.3 describes the authorisation mechanisms used for ACVAS services with section 6.4 highlighting a typical policy-based vulnerability assessment scenario using ACVAS. Section 6.5 highlights some of the current limitations of ACVAS. Finally the chapter concludes with a discussion on the implementation work as a whole in Section 6.6. Throughout the chapter the experiments and experiences in implementing and utilizing ACVAS for detecting vulnerabilities and the associated security policies for remedial measures are discussed.

6.1 VulMIS Internal Structure

As discussed in Section 2.2.14, successful vulnerability management requires [191]:

1. Maintaining system information (inventory) including what type of hardware, operating system version, running services and third-party application is installed on organisational resources. Numerous proprietary software solutions such as Tivoli Inventory [94] and open source technologies such as Pakiti exist [95].

S.no	Host_id	pkg_id	version	Release	Arch	Act_version
1	1	3	1:3.0.4	12.fc15	i686	0
2	2	6	0:2.52	12.fc15	i686	0
3	3	13	2:5.110.22.p23	12.fc16	x64	1

Table 6.1: Entries from the installed_pkgs table

2. Getting reliable, exhaustive, and up-to-date vulnerability information related to target systems is a prerequisite for effective vulnerability management.
3. Assessing and evaluating vulnerabilities in and across organisational resources to determine what vulnerabilities exist in target systems to be used a basis for risk identification and management.

A rich variety of free and commercial security vulnerability assessment tools such as Nessus [100] and Internet Security Scanner (ISS) [192] also exist. However these tools do not provide the details of security circumstance of target systems such as critical configurations and deployed versions of software. This leads to an incomplete picture of the overall organisational systems under potential threat. The VulMIS system seeks to meet these challenges directly. The core schema for the VulMIS internal database structure is shown in Figure A.1 (Annex-A).

At its heart, VulMIS consists of an extensible database comprised of a range of tables. Some of these are highlighted here that are particularly relevant for policy decisions. For example, looking at the entries given in Table 6.1, we can see that second column shows the host (remote VO node) identification e.g. (server1.nesc.gla.ac.uk) we use numbers instead of actual hostnames; pkg_id identifies the installed packages and versioning information. The release column indicates the vendor specific operating system releases, e.g. Fedora core 15. The architecture here implies that the target VO nodes have a 32 or 64 bit architecture; and the last column suggest that these packages last updated (1) or not (0).

The second important VulMIS table holds information about the target VO node including the node id, the installed OS, the kernel version, the IP address (report_ip e.g. 130.209.*.*), the site_id (the site that the node belongs to) and the timestamp when the packages were last modified. The last column in table 6.2 shows the type of packages installed on the target node, which is typically in rpm format but could potentially be in many other formats.

Table 6.3 shows the VulMIS information specifying which software packages are installed on particular target nodes. For each entry in this table, one column specifies the package ID of the packages installed with the second column specifying the name of the packages that are installed on that particular node.

id	Host_id	Kernel	OS	report_host	report_ip	site_id	pkgs_change_timestamp	type
1	1	3	1:3.0.4	12.fc15	i686	0	2012-04-13 17:44:49	rpm
2	2	6	0:2.52	12.fc15	i686	0	2012-04-13 15:32:45	rpm
3	3	13	2:5.110.22.p23	12.fc16	x64	1	2012-04-13 15:32:45	rpm

Table 6.2: Entries from the host table

id	name
1	setools-libs
2	tcpdump
3	httpd
4	make

Table 6.3: Entries from the packages table

pkg_id	operator	severity	version	cve_id	release	vendor
1	1	3	1:3.0.4	12.fc15	0	RedHat
2	2	4	2:0.5	14.fc16	1	Fedora
3	3	2	3:1.4.6	12.deb14	2	Debian

Table 6.4: Entries from the CVE table

Table 6.4 shows the list of nodes and packages vulnerable to the CVEs along with their severity level used in VulMIS.

Finally table 6.5 shows the site ID, the name of the sites and the number of hosts (nodes) on a particular site contributed as part of a given VO collaboration that are key to VulMIS.

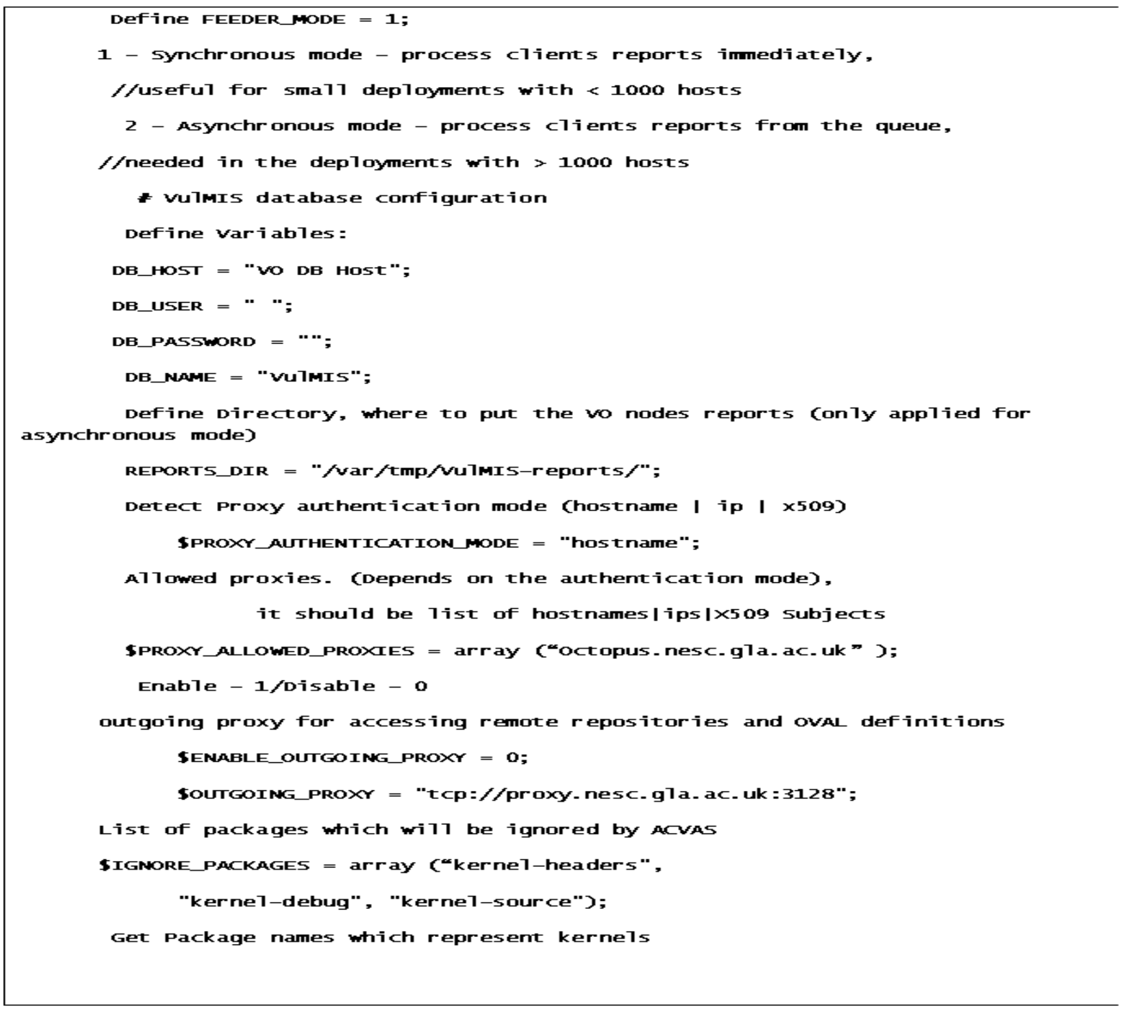
VulMIS data gathering mechanism is depicted in Figure 6.2; it shows reports received from the target hosts (VO nodes). At initialization, the remote node's report incorporates the agreed information to be shared at establishment of the VO, e.g. how many nodes and the required OS etc. This is periodically checked for updates deemed necessary by the VO and the local data providers.

id	name	numhosts
1	gla.nesc	30
4	man.clus	65

Table 6.5: Entries from the site table

```
*/ ! /bin/bash
*/ Default Values

      configuration =
      server =
      serverURL =
      Method =
      cURL-bin =
     openSSL-bin =

If cURL is used

     Then
     Method = cURL
     else
     if method = openssl
     else
     Quit (No transport method available)

Check For Proxy

     If Proxy = enabled
     then
     Parse lines containing (host, architecture, OS)
     Get Packages
     Get System Parameters
     Host, kernel, architecture, packages, type, site

Get List of Packages by Method

     For RPM based or Debian
     rpm -qa
     dpkg -query
      Detect OS
        If OS =  'unknown'
      Try to get OS from Configuration file
     /etc/debian-version
     fedora
     suse
     redhat

Sending Data to ACVAS server

         POST-DATA
     Use HTTP-Post
```

Figure 6.1: Remote node data gathering mechanism for VulMIS



```
        Define FEEDER_MODE = 1;
       1 - Synchronous mode - process clients reports immediately,
        //useful for small deployments with < 1000 hosts
         2 - Asynchronous mode - process clients reports from the queue,
       //needed in the deployments with > 1000 hosts
          # VulMIS database configuration
         Define Variables:
        DB_HOST = "VO DB Host";
        DB_USER = " ";
        DB_PASSWORD = "";
         DB_NAME = "VulMIS";
         Define Directory, where to put the VO nodes reports (only applied for
asynchronous mode)
         REPORTS_DIR = "/var/tmp/VulMIS-reports/";
         Detect Proxy authentication mode (hostname | ip | x509)
             $PROXY_AUTHENTICATION_MODE = "hostname";
         Allowed proxies. (Depends on the authentication mode),
                   it should be list of hostnames|ips|X509 Subjects
         $PROXY_ALLOWED_PROXIES = array ("octopus.nesc.gla.ac.uk" );
          Enable - 1/Disable - 0
       outgoing proxy for accessing remote repositories and OVAL definitions
             $ENABLE_OUTGOING_PROXY = 0;
             $OUTGOING_PROXY = "tcp://proxy.nesc.gla.ac.uk:3128";
       List of packages which will be ignored by ACVAS
       $IGNORE_PACKAGES = array ("kernel-headers",
             "kernel-debug", "kernel-source");
        Get Package names which represent kernels
```

Figure 6.1: Initialising the VulMIS database

6.1.1 Initialising the VulMIS

The VulMIS initialization depicted in Figure 6.2 shows reports received from the target hosts (VO nodes). At initialization, the remote node's report incorporates the agreed information to be shared at establishment of the VO, e.g. how many nodes and the required OS etc. This is periodically checked for updates deemed necessary by the VO and the local data providers. It should be noted that the VulMIS initialization works in both synchronously and asynchronously. Updates can be delivered in a synchronous fashion when multiple resources need to be checked for vulnerabilities, or come asynchronously (periodically) from nodes themselves.

6.1.2 Database (VulMIS)

After initialisation, it is necessary to establish and populate a database storing all of the software information and patch information associated with particular VOs. The VulMIS database includes:

- The name and versions of software packages to be used;
- A listing of the OS packages (RPMs) available. In many cases, software may need to be compiled for VO operating systems resources. For example, Red Hat Linux systems already have installed out of the box, several packages for building other systems. Thus it is important to differentiate between the packages shipped with a given OS and those that can be built specifically for a VO.
- The default versions of software packages applicable to each operating system across the VO resources.
- A VO-specific itinerary listing software packages to be installed on VO resources and importantly, an enumeration of those packages identified as core, e.g. those used for installation and configuration or building of other software systems.

6.1.3 Component Analysis

ACVAS can be configured to execute when registered nodes in a given VO are ready. When this is the case, nodes are immediately assessed for whether potential updates (patches) are required. When this is the case, i.e. versioning or other anomalies are detected, the corresponding updates are downloaded, configured and automatically applied (subject to the local site administrator's permission and local site policy). In undertaking this, each of the VO nodes are classified into groups based on their configuration properties with respect to OS/architecture that distinguishes them.

6.2 ACVAS Implementation Environment

All of the experiments in using the ACVAS system were performed using an AMD Athlon 64 bit processor with a 2.4 GHZ CPU, running Fedora Core 6 operating systems with 16 GB RAM. Nessus 4.2 and Pakiti 2.1 were used as the VPM module and Cfengine 3.2 used as the configuration engine. VulMIS utilized a MySQL database. The test environment itself comprised four different virtual machines (representing collaboration nodes using in different VOs). In the following section the implementation and use of the technical components of ACVAS are presented. The ACVAS VPM consists of server and client-side elements.

6.2.1 VPM Server-side Implementation

An ACVAS VPM Feeder module is used to receive reports from target hosts (VO nodes). This component checks whether the node reports contain any changes from previous reports with those stored in the database. The Feeder module allows reports to be processed immediately with nodes optionally receiving the results directly, or it allows reports to be generated and processed afterwards.

An analysis module (analyser) is used for report generation. This module gathers statistical data and provides reports on the state and vulnerabilities across the VO resources. This resource is augmented with a Vulnerability Definition Source (VDS) which synchronises the internal vulnerability database (VulMIS) with the external sources such as vendor package repositories or OVAL definitions.

The actual server-side implementation utilises **Java** - for interface design and grid service implementation; **mySQL** for the backend database storing user-role and user-delegation information. An **LDAP** server is used to store details about additional attributes of users and resources in the organization. **XACML** is used to express Access Control Policies and Enforcement points.

Several web services are available including a vulnerability service (***VulService***) and a scanning service (***ScanService***). These support the following operations (discussed in Chapter 6):

view - for viewing vulnerabilities that may exist;

scan – for scanning target nodes for presence of vulnerabilities;

insert – to enable site administrators to include new host data into VulMIS;

update – to update the host information stored in VulMIS.

6.2.2 Service-side Communication

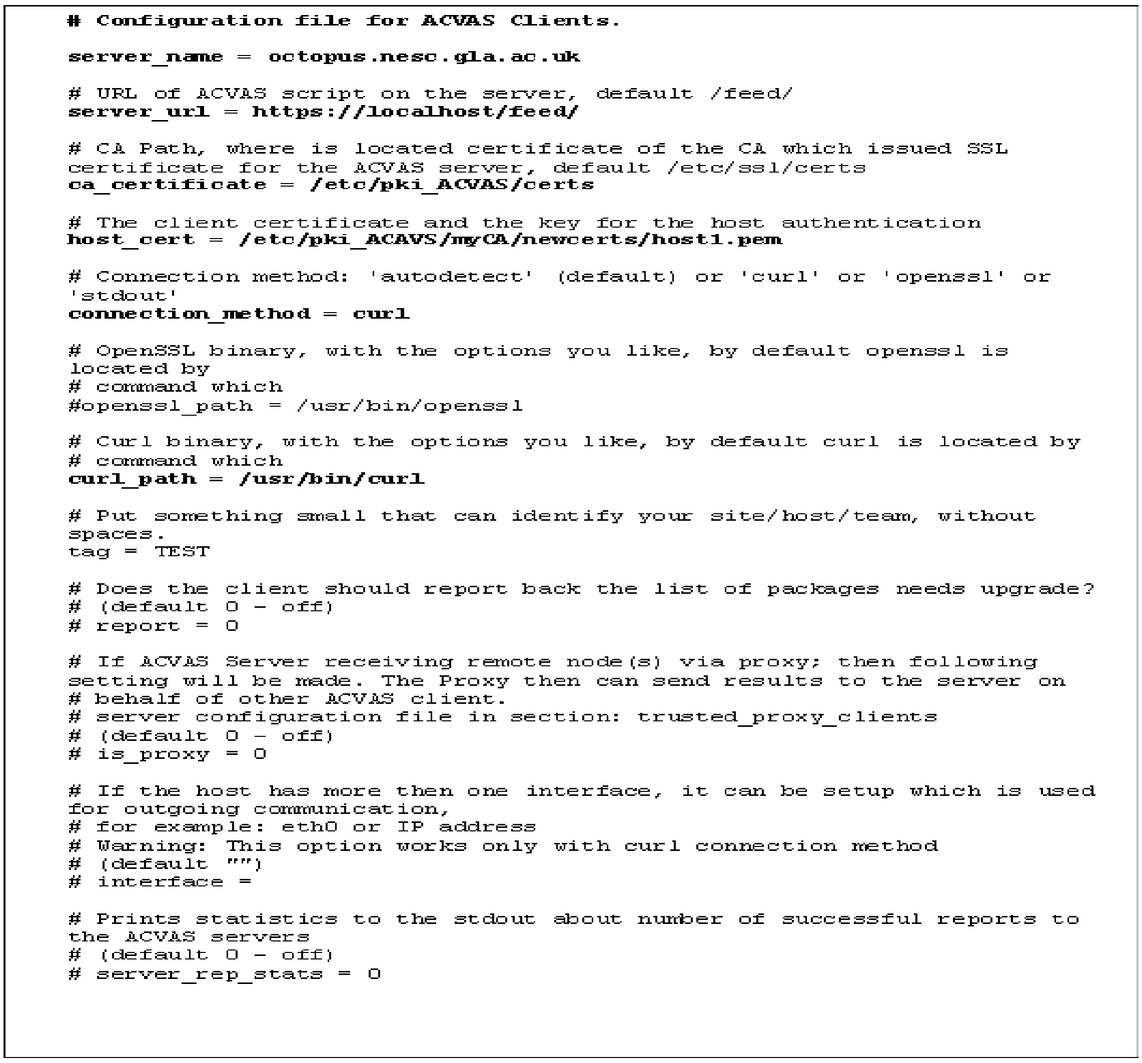
Typically, in a VO environment a central server is used for collecting data from different nodes. The server hostname is given in the client configuration and verified during the SS-L/TLS handshaking process in order to reduce the risk of leaking potentially sensitive information. To ensure confidentiality, the data transfer from the client (nodes) to the server is realised using HTTPS. The server runs as a web application in an Apache HTTP server. The server is divided into two parts: the data processor and presentation layer. The data processing part collects and processes data sent by the client. In particular, upon receiving a report, the list of the packages stored in the database is compared to versions from vendor repositories including OVAL and CVE. The presentation layer provides several (secure) views of the data stored in the ACVAS database.

6.2.3 Target Client-side Module

The client side implementation can be achieved in many ways including web service based clients of scripts (shown in extract 1). Based on the monitored node, the client side script runs common distribution binaries such as ***dpkg-query*** or ***rpm -qa*** to obtain the list of installed packages including operating system version and release, kernel version and hostname. The collected information is subsequently sent to the ACVAS server using POST over HTTPS. It is important to note that the ACVAS clients use the host certificate to achieve mutual authentication with the server.

Client jobs can be periodically triggered, e.g. by *cron* jobs, or by obtaining data directly from target nodes using probes launched by a site monitoring tool such as Nagios. As noted, data can be processed either synchronously or asynchronously, based on the configuration directives. Using the synchronous mode, the server processes data from the client immediately and the results are sent back to the client that is waiting for the response. In this case the client will receive a list of packages that could/should be upgraded (in case a vulnerability is found). In the asynchronous mode, the client nodes do not wait for results and typical close the connection after data is provided. The ACVAS server typically processes data of all target hosts on a regular basis. For large-scale deployments with thousands of monitored hosts and where the clients are broadly distributed the asynchronous form of processing is likely to be the preferred model.

A typical client script is shown in Figure 6.3. This consists of five important directives (shown in bold). The **server_name** shows the name of the ACVAS server where the clients send their reports. The **server_URL** provides the user interface to which site administrators can access



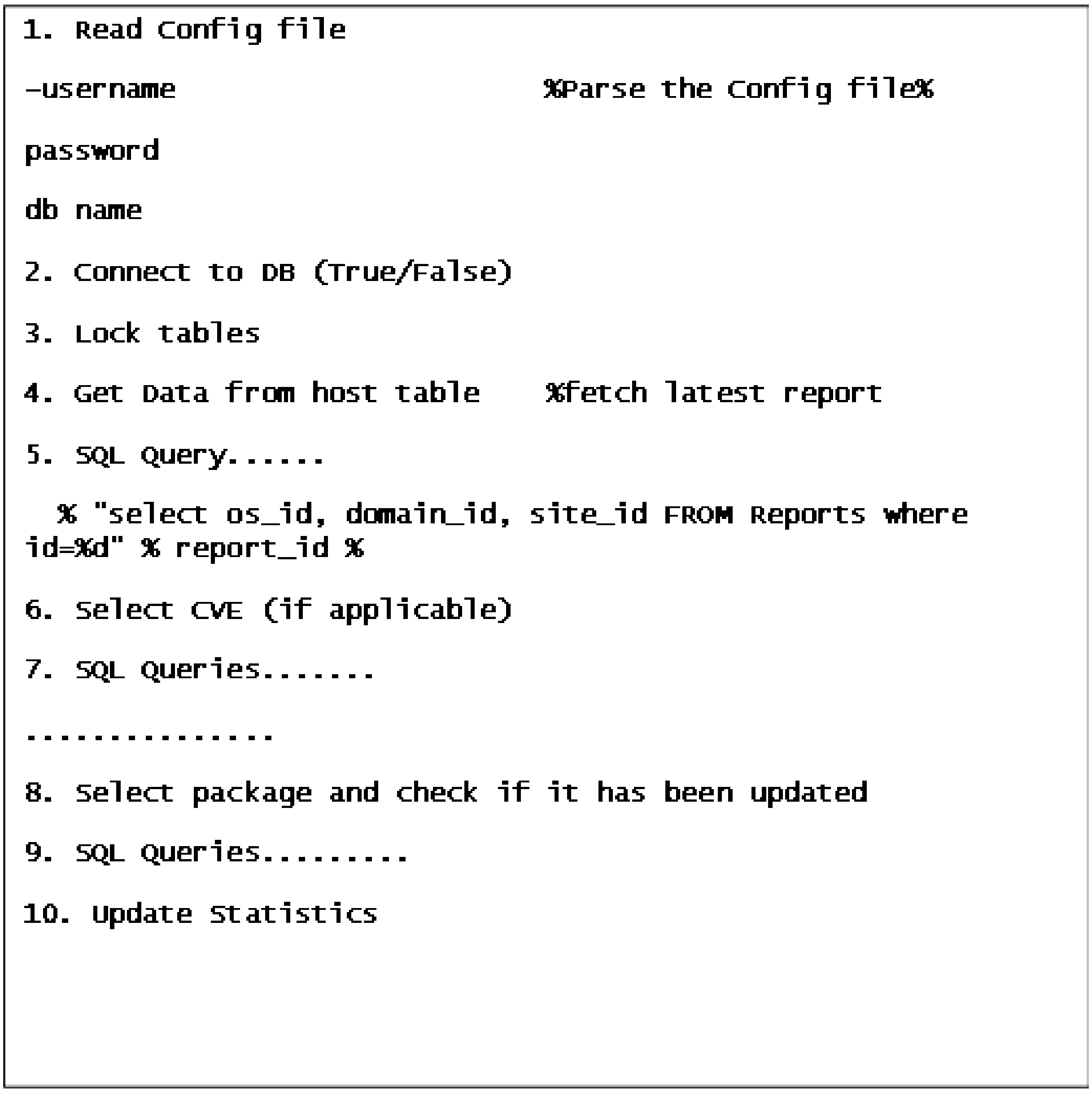
```
# Configuration file for ACVAS Clients.

server_name = octopus.nesc.gla.ac.uk

# URL of ACVAS script on the server, default /feed/
server_url = https://localhost/feed/

# CA Path, where is located certificate of the CA which issued SSL
certificate for the ACVAS server, default /etc/ssl/certs
ca_certificate = /etc/pki_ACVAS/certs

# The client certificate and the key for the host authentication
host_cert = /etc/pki_ACAVS/myCA/newcerts/host1.pem

# Connection method: 'autodetect' (default) or 'curl' or 'openssl' or
'stdout'
connection_method = curl

# OpenSSL binary, with the options you like, by default openssl is
located by
# command which
#openssl_path = /usr/bin/openssl

# Curl binary, with the options you like, by default curl is located by
# command which
curl_path = /usr/bin/curl

# Put something small that can identify your site/host/team, without
spaces.
tag = TEST

# Does the client should report back the list of packages needs upgrade?
# (default 0 - off)
# report = 0

# If ACVAS Server receiving remote node(s) via proxy; then following
setting will be made. The Proxy then can send results to the server on
# behalf of other ACVAS client.
# server configuration file in section: trusted_proxy_clients
# (default 0 - off)
# is_proxy = 0

# If the host has more then one interface, it can be setup which is used
for outgoing communication,
# for example: eth0 or IP address
# Warning: This option works only with curl connection method
# (default "")
# interface =

# Prints statistics to the stdout about number of successful reports to
the ACVAS servers
# (default 0 - off)
# server_rep_stats = 0
```

Figure 6.2: The target node configuration for ACVAS communication

and see through the client data. The **ca_certificate** shows the location of Certification Authority (CA) certificate that is recognized (trusted) by the server and vice versa. The **host_cert** is used for the host's own certificate. Lastly, the **connection_method** shows the mechanism of communication on which clients can send their data to the ACVAS sever, e.g. *cURL* or *openssl*.

6.2.4 VulMIS Data Processing

A typical scenario for data processing for a target node to VulMIS is shown in Figure 6.4. Firstly, the file is read and the user name and password are checked. If these are correct then a database connection is established. Following this, the VulMIS database tables are locked and the report from remote target nodes fetched. Various queries are executed to check for CVE entries and comparisons. Where required new patches are found and the VulMIS database is



```
1. Read Config file

-username                        %Parse the Config file%

password

db name

2. Connect to DB (True/False)

3. Lock tables

4. Get Data from host table    %fetch latest report

5. SQL Query......

  % "select os_id, domain_id, site_id FROM Reports where
id=%d" % report_id %

6. Select CVE (if applicable)

7. SQL Queries.......

...............

8. Select package and check if it has been updated

9. SQL Queries.........

10. Update Statistics
```

Figure 6.3: Algorithm for processing client data for VulMIS

updated.

6.3 VulService Authorisation

Ensuring authorised access to *VulService* is essential. To this end, authorisation decisions have been implemented based both upon centralised and decentralised authorisation infrastructures utilising VOMS and Shibboleth attribute delivery models respectively. XACML decision engines for subsequent policy enforcement use these attributes.

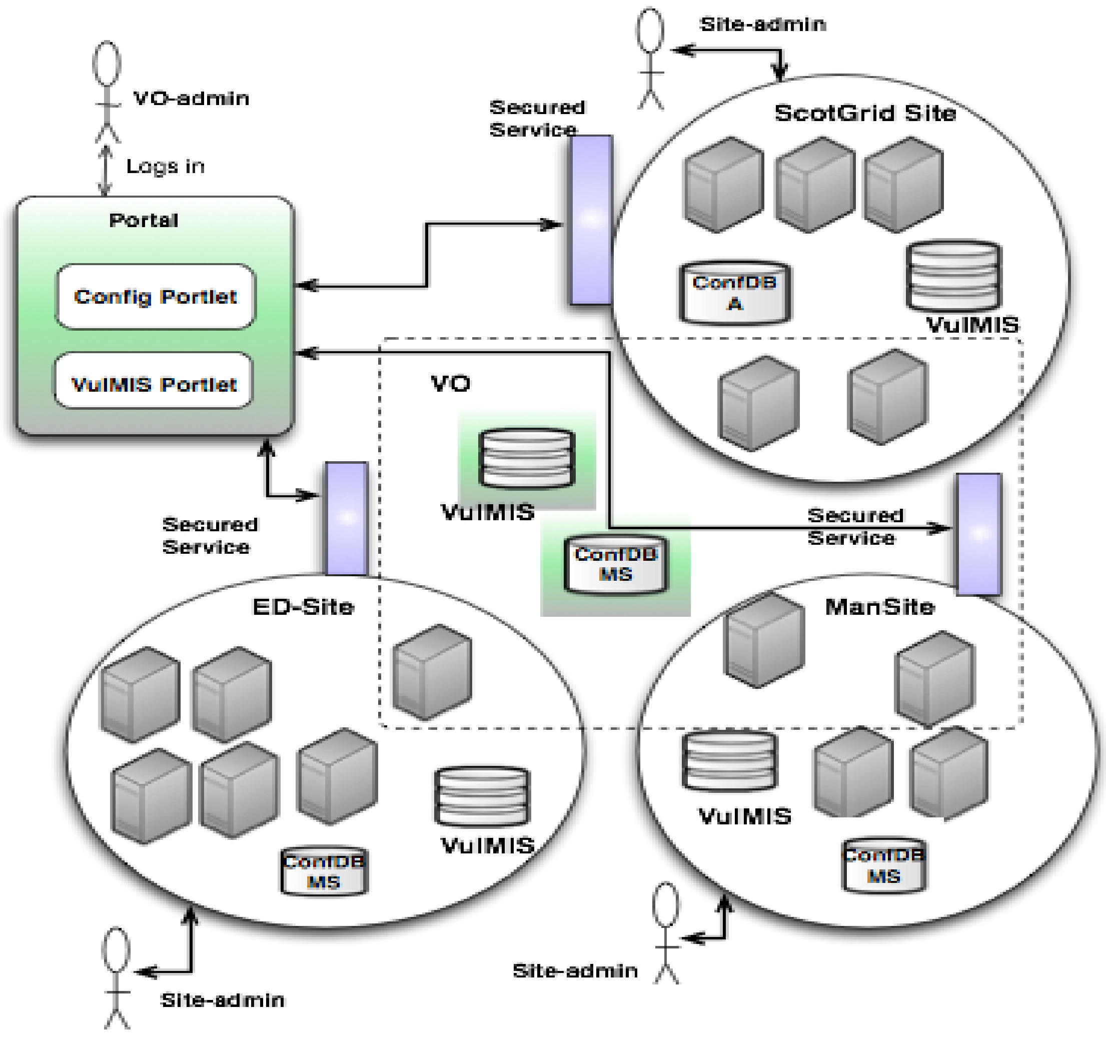


Figure 6.4: ACVAS Implementation for a multi-site collaboration

In Figure 6.5, the ACVAS implementation consists of a user interface (based upon the Liferay Portal framework). This is accessible to site and VO administrators. Once respective users (site/VO administrators) log in, depending on the credentials presented (discussed in Section 6.2.1), their relevant workspace is loaded and the services to which they are authorised are visible. This mechanism builds on the work undertaken as part of the SPAM-GP project [193], whereby authentication at the portal is achieved through logging into a recognised (trusted) Identity Providers. However, the trust is enforced through the SPAM-GP Scoped Attribute Management Portlet (SCAMP). In the SCAMP approach, the portlet filters the identity providers from the UK Federation to allow only those that are recognise as having privileges to access the portal (in our case VO administrators and site administrator). This scoping of credentials is achieved through filtering the attribute acceptance policy which defines the metadata for the UK federation. Once a VO administrator is authenticated, they have access to the portal, the roles and privileges that are released from their identity provider are used to configure and personalise the contents of the portal itself. The ACVAS server hosts services on behalf of sites and provides a bridge between site-specific vulnerability management information systems (VulMIS) and configuration databases (ConfDB). It should be noted that the portal provides authentication-oriented access, whilst finer-grained authorisation is addressed at the ACVAS server level.

As depicted in Figure 6.5 and 6.6, the two sites A and B use a centralised VulMIS and configuration databases to monitor the versions of software packages installed on their site(s). The XML code for the same request is given in appendix B Figure A.3

```
Begin
Request
If Subject (requestor) = Subject ID
Then
Compare Attribute 'John' as Role ID
If subject = John and role = VO-admin
Then Decision = allow
Resource = 'Installed-Packages'
else
Allowed Action
actionType = 'string' where value = 'view'
End Request
```

Figure 6.5: The Pseudo code for XACML based Request to access resource at VulMIS

The example request described in Figure 6.5 corresponds to the local policy specification in Chapter 5, section 5.5.1. Figure 6.6 contains several important aspects when it comes to defining XACML based policies: the subject, resource and actions allowed. In the above

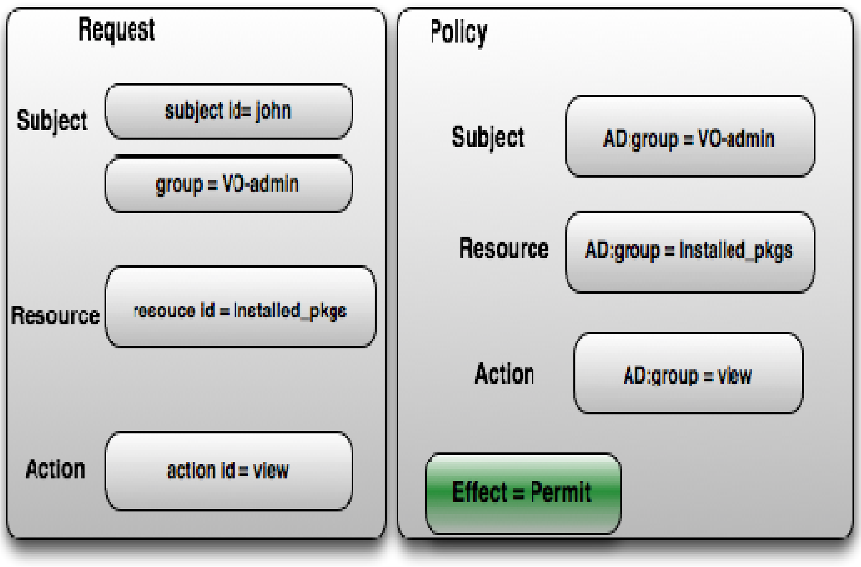


Figure 6.5: XACML Request and Policy matching

example, the Subject has ***subjectID*** "John", and group as VO-administrator. The resource ID is ***1234*** and attributes value is "*installed_pkgs*" i.e. the actual resource name.

```
Begin
Response
If Resource = 'Installed-Packages' and
role = VO-admin
then Decision = Permit
and Status = OK
else
Decision = Deny
End Response
```

Figure 6.6: The Pseudo code for XACML Response allowing access to resource at VulMIS

The response to the request presented in Figure 6.6 is depicted in Figure 6.7 which shows the "Permit" decision. Moreover the Request and Policy inter-related entities such as Subject, Resource and Action are depicted in Figure 6.8.

6.3.1 Policy Enforcement and Decision Points

As depicted in Figure 6.2, the key responsibility of the Policy Enforcement Point (PEP) is to protect resources (services) and allow only legitimate and authorised access requests. The PEP receives incoming messages from the Protocol Handler (discussed in section 6.2.2). The

PEP supports three types of requests.

- authorization request: the service invocation request comes as an "isAllowed" request to the PEP and in a successful response, an authorization token is served;
- access request with authorization credentials:
- in the case of a successful access by the PDP, the ACVAS server invokes the relevant service.

As the interaction between the portal and ACVAS server utilises SOAP based communications, the PEP identifies the first type of authorization request from an "*isAllowed*" SAML assertion attached within the header of the incoming SOAP access request. However, in the second case an authorization context is prepared for the PDP and used to establish an authorization response through a targeted security context handler. A context handler is used to map security attributes from various sources (pushed from the user through the PEP or pulled from various attribute authorities by a policy information point (PIP)). These may use different naming and structural details to the native format used by the PDP. For instance, if the PERMIS PDP is deployed then X.509 attributes certificates [8] are used, and if XACML PDP is used then SAML attributes format [9] and/or XACML attribute format are used [10].

The PDP uses the request context to retrieve the security policy protecting the services and make an authorization decision. If the policy exists then either a permit or deny decision is returned to PEP using the PERMIS PDP [11]. However with a XACML implementation, if the submitted request context to the PDP doesn't associate with any of the security policies then an intermediate result is returned. The PEP also decides whether the authorization request from the ACVAS server was for access to the services. If the access request was a normal request to get access to the service then the request will be allowed or denied according to the authorization decision from the PDP.

6.4 Case Study

To demonstrate the ACVAS framework we consider a representative scenario where three sites decide to form a VO collaboration (Edinburgh, Manchester and ScotGrid). Prior to the formation of this VO, the sites agree on the research data sets to be shared, the computational services and resources to be offered and the associated roles defined across the virtualized environment. As part of this process an agreement on vulnerability and configuration information for VO-specific purposes is agreed upon. The 'secured services' shown in Figure 6.9,

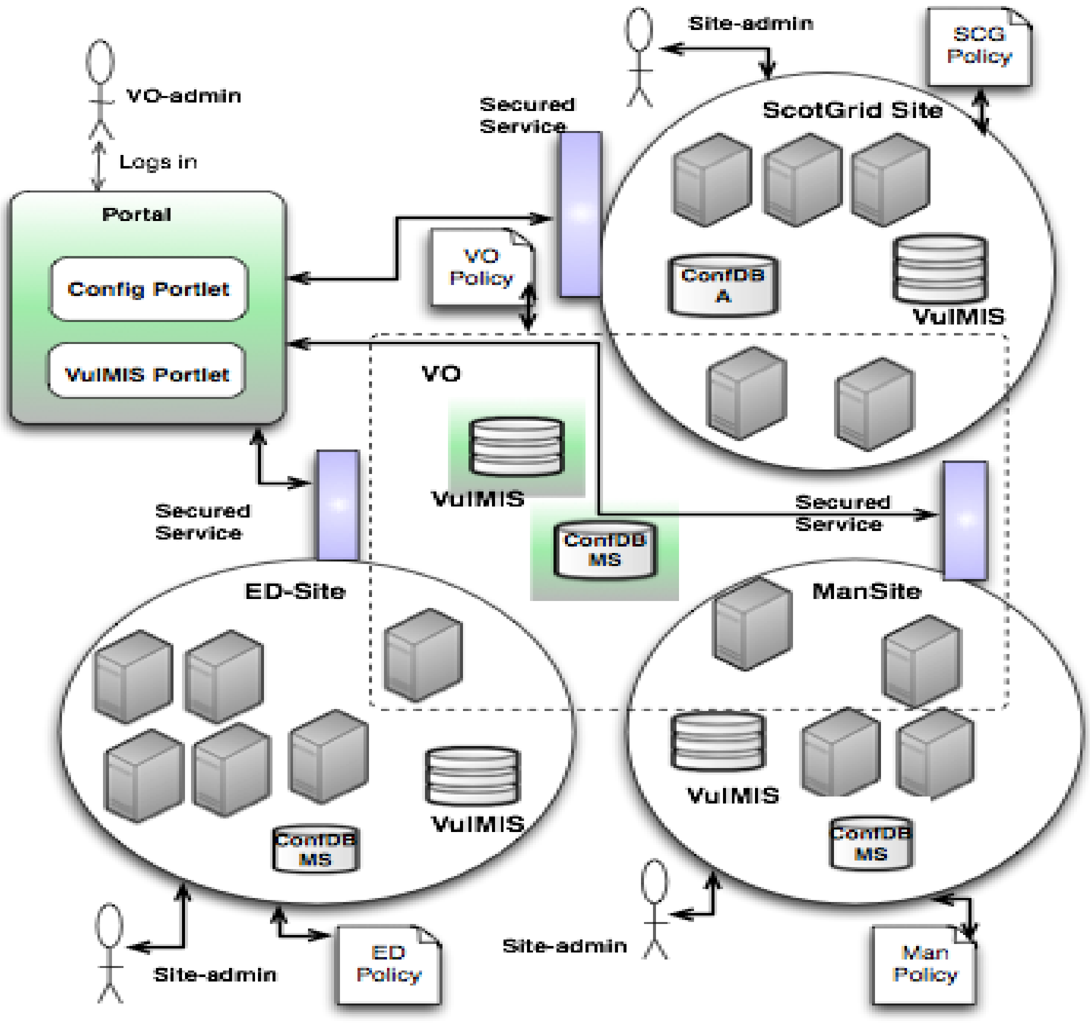


Figure 6.6: Secure-based VO collaboration for vulnerability assessment

provide an interface to access the data and services provided by sites. As discussed in section 3.4 for secure collaborations, it is often necessary to establish a clean bill of health, i.e. where all sites are fully patched and securely configured. The three collaborating sites (see Figure 6.9) provide their relevant VO-specific data to authorised VO administrators through secured services utilizing authorization solutions, e.g. role based access control (RBAC).

As depicted in Figure 6.9, sites are in disparate domains and have their own policies (e.g. ED, Man and SCG policies) for accessing their own local resources including the configuration information and patch status of their own contributed resources (nodes). A VO admin logs in to the portal using authorisation credentials agreed upon by the three respective sites prior to VO formation. Through this portal, vulnerability (VulMIS) and configuration information (confDB) from the three sites is made available to authorized administrators using VO-specific credentials. The VO specific policy of Figure 6.9 on which the three sites have agreed is depicted in Figure 6.10.

```
Input:  VO-specific Policy
Output: allow or deny decision
Resource Match and Match ID = 'Vulnerable Packages'
Rule 1
If {
subject == VO-admin
Resource== Vulnerable-Packages
Action Allowed== Full Access
}
Condition {
Attribute Value (subject Attribute(role))
Attribute Values (Resoiurce Attribute(vulnerable packages)}
}
Then Decision == Permit
Rule 2
If {
subject == VO-admin
Resource== vulnerable-Packages
Action Allowed== Full Access
}
Condition {
Attribute Value (subject Attribute(X500 DN))
Attribute Values (Resoiurce Attribute(John Boyd)}
}
Then Decision == Permit
Rule 3
Other than rule 1 and 2
Deny All Other Requests
```

Figure 6.7: Pseudocode for VO Specific Policy

6.4.1 Request Format

The VO-specific policy given in Figure 6.10, shows how a VO-administrator role with Common Name (CN) "*John*" is authorized to view the vulnerability information available through collaborating sites shown in Figure 6.11.

```
Begin
Request
If Subject (requestor) = Subject ID
Then
Compare Attribute 'VOadmin' as Role ID
If subject = group and role = VOadmin
Then Decision = allow
Resource = 'Installed-Packages'
else
Acces Denied
Allowed Action
actionType = 'string' where value = 'view'
Resource = 'voadmins-guide.html'
Decision = Allow
End Request
```

Figure 6.8: VO-specific access request from VO admin

6.4.2 Response Format

The successful response related to the request shown in Figure 6.11 is shown below in Figure 6.12.

```
Begin
Response
If Resource = 'voadmin-guide.html' and
role = VO-admin
then Decision = Permit
and Status = OK
else
Decision = Deny
End Response
```

Figure 6.9: VO-specific response to access request in Figure 6.10

6.5 Scenarios

With the previous environment established, several key scenarios are illustrated to highlight how the ACVAS framework can be used to improve end-to-end security across the VO.

For many VO collaborations, default passwords and default settings more generally, can be a potential danger for collaborating sites. Weak credentials used in inter-organisational collaborations can also potentially be exploited by remote attackers.

6.5.1 Securing Unprotected User Accounts

In this scenario we consider user accounts and their access privileges during the installation process in a typical MySQL database – including user privileges on table permissions. For some installations, accounts can have a default user name (root) which is the superuser account that has all privileges and can do anything. This root account often has no password unless specifically set. If this remains the case, a malicious attacker can potentially connect to the MySQL database as root without a password and be granted all privileges. It is noted that this phenomenon is also not uncommon in operating systems such as Windows, where root accounts can be created that permit connections from the local host only. In this case, remote connections can be made by specifying the host name as localhost (typically the IP address 127.0.0.1). In this case, if the user selects the enable root access from remote machines option during installation, the Windows installer creates another root account that permits connections from any host. Moreover, some MySQL database accounts are for anonymous users and thus have an empty user name/password. In this case they are directly vulnerable to attackers.

6.5.2 Scanning for credentials

In vulnerability scanners such as Nessus there are many plugins which can be used to check for default and common credentials such as those beginning with "account_*". Indeed many attackers try to exploit these for login via telnet and/or SSH. These plugins test for credentials that are known to be associated with a particular device or application. In VulService (discussed in Chapter 6, Section 6.3) a built-in default and weak credential policy is supported. The VulService utilizes a filter feature in the policy setup where searches for the occurrence of both “account” and “default” are supported. This filtering feature allows identification of weak or known default username/password combinations and to flag these when discovered. This information can subsequently be fed to local administrators for resolution.

6.5.3 Filtering ICMP Protocol Vulnerability

Many attacks are caused by time-based vulnerabilities. Thus a remote host can use ICMP time stamping on a local machine to overcome time-based authentication protocols. The work around for such security warnings is to filter the icmp timestamp requests and the outgoing icmp timestamp replies and flag them to VO administrators, who may subsequently need to monitor them prior to any serious attacks/outages.

```
bundle monitor watch_breakin_attempts
{
measurements:
"/var/log/auth.log"
# This is likely what you'll see when a script kiddie probes # your
system
handle => "ssh_username_probe",
stream_type => "file",
data_type => "counter",
match_value => scan_log(".*sshd\[.*Invalid user.*"),
history_type => "log",
action => sample_rate("0"); "/var/log/auth.log"
# As scary as this looks, it may just be because someone's DNS
# records are misconfigured - but you should double check!
handle => "ssh_reverse_map_problem",
stream_type => "file", data_type => "counter",
match_value => scan_log(".*sshd\[.*POSSIBLE BREAK-IN ATTEMPT!.*"),
history_type => "log",
action => sample_rate("0"); "/var/log/auth.log"
# Someone is trying to log in to an account that is locked
# out in the sshd config file handle => "ssh_denygroups",
stream_type => "file", data_type => "counter",
match_value => scan_log(".*sshd\[.*group is listed in
DenyGroups.*"),
history_type => "log",
action => sample_rate("0");
"/var/log/auth.log"
# This is more a configuration error in /etc/passwd than a
# breakin attempt...
handle => "ssh_no_shell",
stream_type => "file",
data_type => "counter",
match_value => scan_log(".*sshd\[.*because shell \S+ does not
exist.*"),
history_type => "log",
action => sample_rate("0");
```

Figure 6.10: A CMS illustration for monitoring break-in attempts

It is good practice that site administrators and VO admins remain vigilant by scanning syslog for break-in attempts. It might be the case that ‘script-kiddies’ probe remote sites for vulnerabilities, such as using attacks based on dictionaries of account/password combinations, looking for unguarded accounts or accounts with default passwords. However, most of these scans are considered harmless, because a well-maintained site will not use the default passwords that these hackers seek to exploit. However, knowing that a site is being scanned is a good thing, and the CMS module can help site administrators to find that out. For example, ‘sshd’ logs it’s message through ‘syslog’, one can again filter lines based on the service name. In the ACVAS implementation, authorization messages are routed to /var/log/auth.log, and monitored using directives depicted in Figure 6.13.

6.5.4 Secure Software Deployment

In this scenario, we consider the case where a VO administrator wants to deploy software across a set of known Grid sites for a particular VO-specific application to be run. In this case we focused in particular upon a scenario where a certain version of Java was needed (Java 1.6) across the three nodes dedicated for a VO (see Figure 6.9), whereby on our testbed for a simple Java 1.6 based application to execute. The preconditions associated with this scenario include firstly that the target node has the appropriate version of Java installed and that VO-administrator have the privilege to install software on the other nodes. We assume that the successful post-condition for this scenario is that Java 1.6 is installed across the three nodes of the test-bed and that the application could be run. With CMS all that we check is that the desired packages are installed correctly. This can be used to simply report on the correctness of the target node, without attempting to fix anything.

```
bundle agent verify_packages
{
vars:
"allpkgoutput string => execresult("/usr/bin/rpm -qa --queryformat
\"%{name}\n\"");
"allpkgs" slist => splitstring ("$ (allpkgoutput)", "\s+", 999999);
packages:
"$(allpkgs)"
package_policy => "verify",
package_method => rpm,
classes => if_notkept("incorrect_$(allpkgs)");
reports:
"Problem:  package $(allpkgs) is not installed correctly."
ifvarclass => "incorrect_$(allpkgs)";
}
```

Figure 6.14: A CMS directive for installing a package

The directive in Figure 6.14 starts by getting the listing of all packages by running an external command using the *execresult()* function, and storing it in the *$(allpkgoutput)* string, which then gets split by the *splitstring()* function into the *@(allpkgs)* list. It then iterates over this list verifying each package in turn. If the promise is not kept (e.g. if the package does not get verified correctly), the packages bundle defines an *incorrect_packagename* class. In the reports: section, it iterates again over *@(allpkgs)*, printing a message for the packages whose *incorrect_packagename* class is defined. This can be used as a general "sanity check" of a system, for example to produce a report of its current state or to trigger automatic corrective actions.

Furthermore, the interactive nature of software installation requires users to agree to the terms and conditions of the licenses needed during actual installation and configuration. However, to overcome this issue scripts can be used to automate the process, however in this case situations can lead to automation where site administrators are unaware of the agreements and license models of the software being installed. Thus it may not always be desirable to automate this process as outlined in [12].

6.5.5 Middleware Software Deployment

As an extension to the scenario outlined above in 6.5.4, AVCAS was used in an example scenario for automating the installation and configuration of Grid middleware [12], e.g. Globus Toolkit (GT). In these particular scenarios it was shown how a typical Grid-based middleware environment could be installed across remote VO nodes. The precondition associated with this scenario was that the client (target node) had the various GT4 software components needed for installation and configuration of the GT4 environment. The ACVAS CMS directive used for the middleware software deployment is depicted in Figure 6.15.

```
control:
actionsequence = (copy shellcommands)
domain = (nesc.gla.ac.uk)
policyhost = (octopus.nesc.gla.ac.uk)
master_cfinput = ( /usr/cfinput)
== server input directory
workdir = (/home/jan)
==deploy grid service ==
shellcommands:
any::
"mkdir -p /opt/globus/globus-4..0.1"
chown globus.globus /opt/globus-4.0.1
ls -la /opt | grep globus
su -globus
cd
tar -jxf gt4.0.1-all-source-installer.tar.bz2
export JAVA_HOME = /opt/jdk1.5.0_06
export PATH = $JAVA_HOME/bin:  $PATH
export JAVA_HOME = /opt/jdk 1.0.5_06
export PATH = $ JAVA_HOME/bin:  $PATH
cd gt4.0.1-all-source--installer
export GLOBUS_LOCATION = /opt/globus-4.0.1
./configure --prefix = $GLOBUS_LOCATION --disable-rls
make
make install
```

Figure 6.15: A CMS directive for deploying a middleware software

6.5.6 Grid Service Deployment

In multi-domain collaborations, for successful VO operations and due to reasons of reliability and/or fault tolerance it is often the case that services need to be replicated across other hosts. In this scenario ACVAS was used to show how configuration management tools could be used for automatic transfer of Grid services (e.g. given as Grid archive files) and the copying, extraction, installation and subsequent activation (deployment) of Grid services. The basic precondition for this scenario to be successful was that section 6.5.5 was satisfactorily completed. [12] shows how a GT4 client interacting with a GT4 Grid service (installed on a given node) could work with the same service once migrated to another node through dynamic deployment via ACVAS. In particular a Grid archive (gar) file was moved to a remote VO host and extracted through a configuration management script as depicted in Figure 6.16. Soon after the extraction of the gar file, the remote Grid container was stopped and restarted. The newly added service deployment was confirmed when it was available/visible for interaction to the client.

```
control:
actionsequence = (copy shellcommands)
domain = (nesc.gla.ac.uk)
policyhost = (octopus.nesc.gla.ac.uk)
master_cfinput = ( /usr/cfinput)
== server input directory
workdir = (/home/jan)
==deploy grid service ==
shellcommands:
any::
"/usr/local/gt4.0/bin/globus-deploy-gar
$(workdir)/org_globus_examples_services_core_first.gar"
```

Figure 6.10: A CMS directive for deploying a grid service

6.6 ACVAS Limitations

During the course of implementing the ACVAS framework several limitations have been identified. Firstly, adopting a centralised data management model for each site can potentially be a bottleneck if the size of the organisation grows since it can become a single point of failure. This can be resolved by having multiple vulnerability databases, e.g. on a per site/VO basis. Secondly, in ACVAS it is assumed that the collaborating sites have some sort of trust and are willing to 'share' their 'vulnerability information' across the collaboration. However in many real world cases, this may be untenable. The challenge of the 'patch paradox' also needs to be tackled, i.e. where a VO-specific resource is found to be 'vulnerable' and need application of an emergency patch but there are running jobs or other circumstances that make the application of the patch impossible at that time.

6.7 Summary/Discussion

This chapter has presented the ACVAS infrastructure for automated configuration and vulnerability assessment and described its implementation. The chapter has dealt with two main problems: identifying vulnerable software across VO-wide resources and the automated application of security-oriented policies to affected resources. The ACVAS system allows secure verification of the patch status information of target hosts that subsequently enables VO-administrators to specify and download secure (hashed) updates of patches from remote repositories. The user interface of ACVAS allows site/VO administrators to obtain required patches and deploy them on-the-fly over the target affected VO-resources.

Sample policies for vulnerability assessment and configuration issues have been presented and implemented. Whilst showing the proof of concept, it is important to emphasize that these are, as stated, a proof of concept and a multitude of other vulnerability and patch-specific scenarios exist. It may be the case that the complete automation of patch management for other systems is an unattainable goal, e.g. where complex interdependencies of software system updates exist. ACVAS has however been designed to be generic and support a range of scenarios.

Chapter 7

ACVAS Evaluation

This chapter describes use case scenarios for our implementation discussed in Chapter 6. The chapter is structured as follow. Section 7.1 describes the theoretical evaluation of ACVAS and overview of Grid domain. Section 7.2 describes the pratical validation of our prototype implementation of ACVAS in Chapter 7 and the architecture discussed in Chapters 4 and 5. The chapter concludes with a discussion on different aspects of these scenarios in section 7.3.

7.1 Theoretical Evaluation

To consider the challenges of vulnerability assessment and configuration aspects and provide the context in which ACVAS is explored, several key theoritical scenarios are outlined and the requirements are disccused in context of secure collaborations. The first scenario presents a representative sequence of interactions demonstrating how default database passwords which are based on default settings can be potential danger. The second and third are overview scenarios for checking weak credentials used in inter-organisation collaboration and making sure the remote attacker are unable to defeat all time based authentication protocols of the affected node used in collaboration respectively. In this section we briefly highlight the above mentioned scenarios for which we tested ACVAS's capabilities to check for vulnerabilities due to configuration errors and possible missing patches.

7.1.1 Securing un-passworded Accounts

In this scenario we consider the MySQL user accounts and their access privileges during installation process in the database (mysql.user) that contains the grant tables permissions.

Among the accounts some have the default user name root which is the superuser account that has all privileges and can do anything. The initial root account passwords are empty by default as well, so anyone (malicious intent attacker) can connect to the MySQL server as root without a password and be granted all privileges. Particularly, such weaknesses are common in operating systems such as Windows, where root accounts are created that permit connections from the local host only. Remote connections can be made by specifying the host name as localhost or the IP address 127.0.0.1. If the user selects the Enable root access from remote machines option during installation, the Windows installer creates another root account that permits connections from any host. Moreover, some accounts in MySQL are for anonymous users and have an empty user name. The anonymous accounts have no password, so any attacker can use them to connect to the MySQL server.

7.1.2 Scanning for credentials

In vulnerability scanners such as Nessus there are currently seventy plugins which check for default and common credentials such as those beginning with "account_*" (an attacker try to exploit these) for login via telnet and/or SSH. These plugins test for generic common credentials or credentials that are known to be associated with a particular device or application. In VulService (discussed in Chapter 6, Section 6.3) we have used the built-in policy named "Default & Weak Credential Check". Moreover, we have configured the policies within VulService and used the “Filter” feature in the policy setup and searched for occurrence of both "account" and "default". This filtering feature allowed us to go through each family in each of the search results and manually enabled the plugins that tested for either weak or known default username/password combinations.

7.1.3 ICMP Protocol Vulnerability

In this case of vulnerability the security warning from VulService (is looks like: “The remote host answers to an ICMP timestamp request. This allows an attacker to know the date which is set on your machine”. This vulnerability would mean that this scenario if unaddressed may help a remote attacker to defeat all time based authentication protocols of the affected node. The work around for such security warnings is to filter out the icmp timestamp requests and the outgoing icmp timestamp replies. These vulnerabilities mentioned above can easily be used to break into organizational networks. VO administrators need to have a close look at them and correct them as soon as possible prior to any serious outage to happen. In the literature we can find a quite few approaches which are used to vulnerability modeling for performing analysis of attacks from malicious intent attackers on computing system and networks. The field of safety critical systems examines hazard analysis process, when it comes to computer systems and modern day communication infrastructure, vulnerabilities can be perceived as hazardous. There are several techniques being applied to analyse hazards including; checklists, event tree analysis, fault tree analysis and cause-consequence analysis. In [1] Vidalis and Jones discuss several pros and cons of these techniques, we briefly discuss those here. Firstly, the checklists technique cannot demonstrate the relationships between the vulnerabilities since they are static. Moreover, this technique also lacks to determine the inter-relationship between two vulnerabilities to found out how and why these vulnerabilities can trigger each other. Secondly, event tree analysis is based a Boolean approach for examining vulnerabilities and failures. Due to complex nature of computing systems there are several type of the vulnerabilities which cannot be expressed with Boolean values. This technique might work very well for hardware vulnerabilities, but there are six other vulnerability types i.e. (physical, natural, human, hardware/software, media and communication), that cannot be addressed effectively [2-7] Thirdly, fault trees just present a chronological ordering of events over time and hence are not adequate to visualize and model the different types of vulnerability relationships. At each level these merely show the same thing in more detail. Fourthly, Cause-Consequence Analysis (CCA) follows a top-down or backward technique to determine the causes of an event. Advantage of using CCA is that once can model both time dependencies and casual relationships among the events. However, the disadvantage of CCAs is the size of the diagrams, their complexity and the fact that they cannot accept data from other diagrams. Nevertheless, the use of history attack data for producing patterns and attack trees. By using this technique one predict the path that a particular threat agent will follow by analysing the exploits that might be used. While completing an attack tree, each path represents a unique attack on the organisational systems. Attack trees are not without any disadvantage; since they cannot analyse complex systems or even larger size organisational networks mainly due to their complexity [8]. As Vidalis et. al [1] mentions, a different number of exploits might be used for attacking more than one vulnerabilities, and the same exploits can be used for attacking different vulnerabilities. In dynamic system, using exploits as nodes to produce attack trees is not efficient.

7.2 Practical validation of ACVAS Prototype

In our example virtual collaboration, we consider three sites to share their resources and agree to form a VO including (Edinburgh, Manchester and ScottGrid). However, prior to formation of this VO, the sites agree on the data set to be shared and roles to be defined across the virtualized environment and the relevant data (anonymised data of target nodes) in the VO. By anonymised data here we mean that the respective site have produced and make available the vulnerability and configuration information for VO-specific purposes. The ‘secured services’ shown in Figure 7.1, are providing an interface to access to those logical data sets provided by sites. As discussed in section 3.4, Chapter 3 that in a secured based collaborative scenarios where due demands from a particular VO from the site(s) to provide ‘clean-site’ tags to make sure there are not any security issues to the establishment of the target VO collaboration. Since one of the weakest link in the VO chain will provide the opportunity to the attacker to impersonate the overall sprit of the collaboration. Moreover, the three collaborating sites mentioned in Figure 7.1 will provide their relevant VO-specific data to authorised VO administrator through secured services. We also note that these services are based on fine-grained level (granular) authorisation mechanism backed up by policy. Approaches based on typical role based access control (RBAC) themselves are not enough to support such endeavors; for example a plain <Role, Target, Action) i.e. (Role x Target x Action) have no meaning and authorisation mechanism in those complex requirements. For example in case where a patch is required to address a vulnerability, under what circumstances a particular VO administrator is allowed to access vulnerability information made available through joint collaborative shared environment.

As depicted in Figure 7.1, three sites share resources to form a VO through dedicating compute nodes located at each collaborating sites namely ScotGrid, Manchester and Edinburgh. Since sites are in disparate domains and potentially have their own policies for accessing their resources (configuration information and patch status of contributed resources (nodes). VO admin logs in via the given interface (portal based) based on credentials agreed upon at three respective sites prior to VO formation. Supposing authentication mechanism has been successful and now it needs access to VO specific vulnerability information provided they are in possession of correct credentials. The annoymised data from the three sites is available at VO-level including configuration (confDB) and vulnerability/patch status (VulMIS). The VO specific policy on which the three sites are agreed is depicted in Figure 7.2.

7.2.1 Site administrator’s view of ACVAS

As discussed in Chapter 5, section 5.5.1, the site-specific policy allows a site-administrator role with a common name (CN) “Jan” to view the vulnerability information available through collaborating sites shown in condensed form Figure 7.5. Thus the subject *Jan* who has role of site administrator in Glasgow is allowed to select, insert or update actions on *installed packages* in Glasgow. However, as mentioned in section 7.1, the information available to site administrator is not anonymised data contributed through three sites for the collaboration. Access to site specific information enables site administrators to make sure that the data contributed

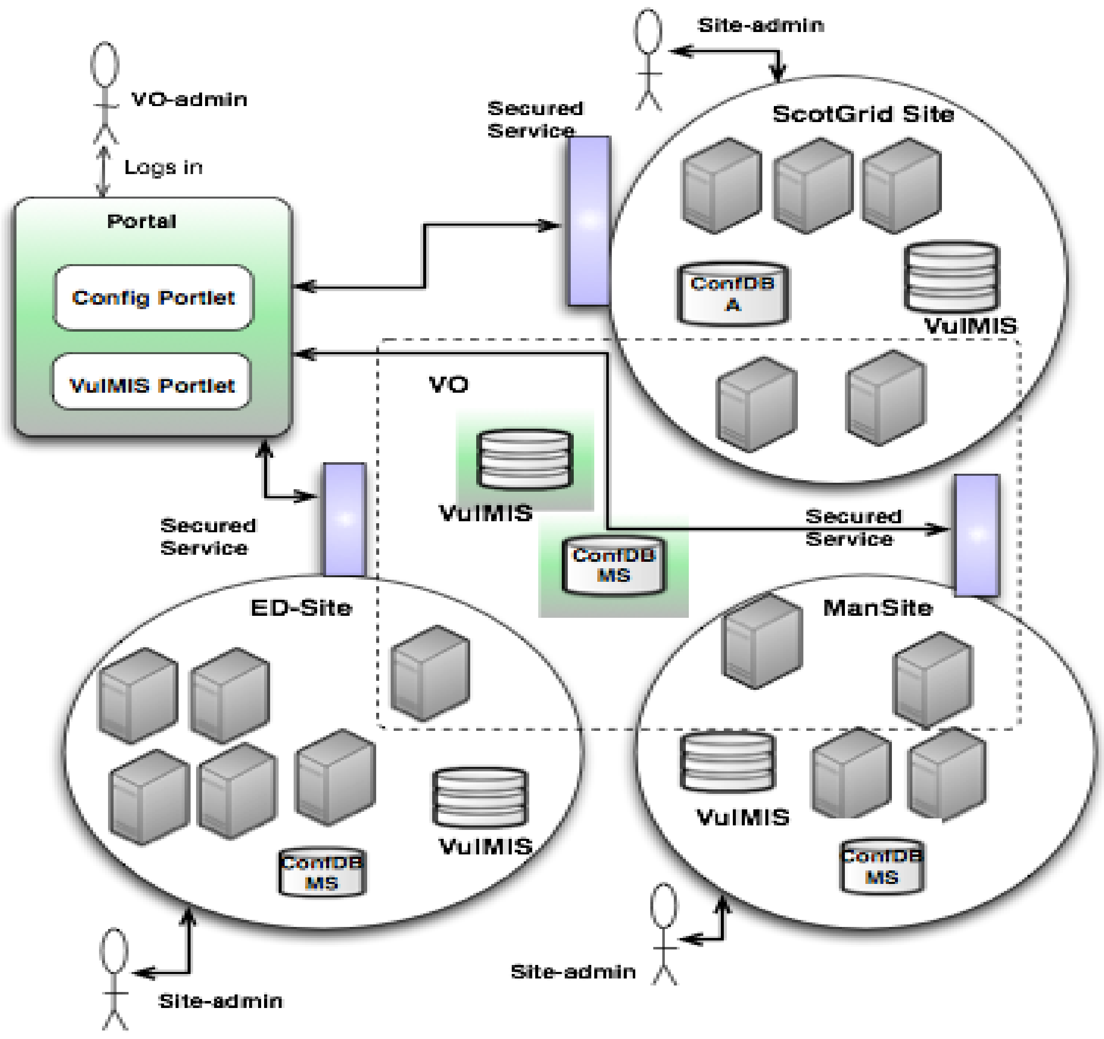


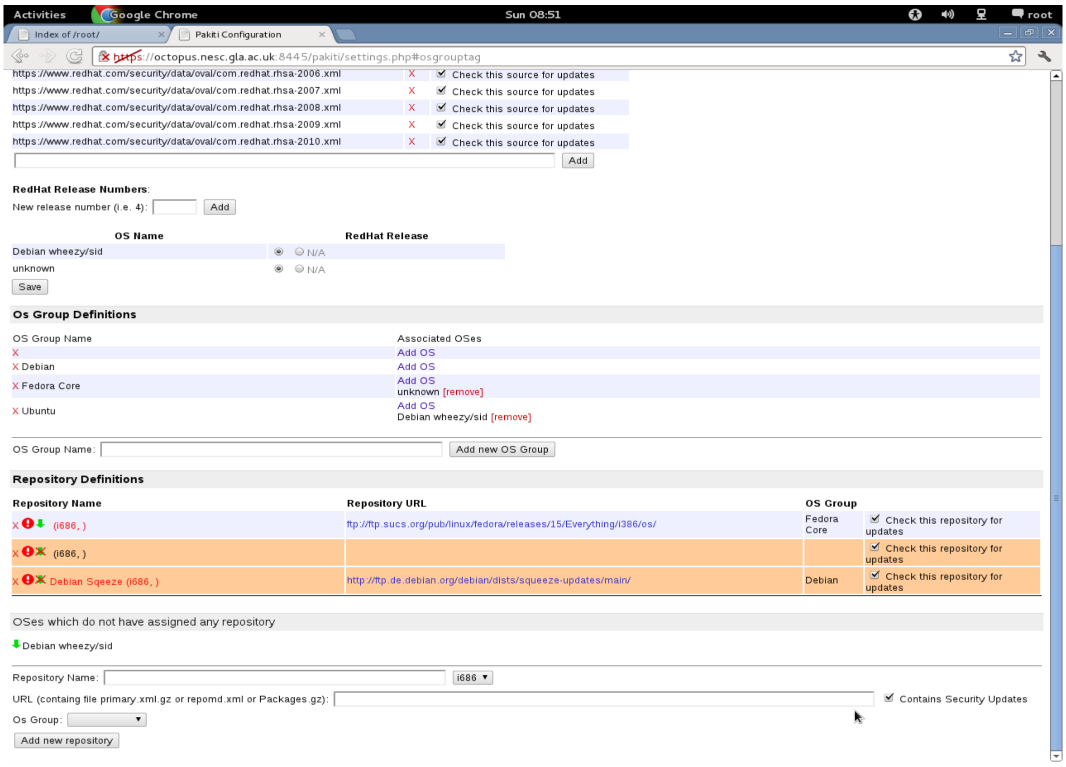
Figure 7.1: ACVAS Implementation for a multi-site collaboration

by individual sites is clean and do not pose any security risk to individual site involved in the collaboration.

Apart from specifyng the trusted patch repositories for site specific OS, site administrators are allowed to define the configure OVAL based sources for the target nodes to get the latest vulnerablility information and download updates as depicted in Figure 7.6.

7.2.2 VO adminisrator's view of ACVAS

The VO-specific policy discussed in Chapter 5, section 5.5.2 allows a VO-administrator role with a common name (CN) "John" to view the vulnerability information available through collaborating sites shown in condensed form Figure 7.2. However, as mentioned in section 7.1, the information available to VO administrator is anonymised data contributed through



https://www.redhat.com/security/data/oval/com.redhat.rhsa-2006.xml	X	✓ Check this source for updates
https://www.redhat.com/security/data/oval/com.redhat.rhsa-2007.xml	X	✓ Check this source for updates
https://www.redhat.com/security/data/oval/com.redhat.rhsa-2008.xml	X	✓ Check this source for updates
https://www.redhat.com/security/data/oval/com.redhat.rhsa-2009.xml	X	✓ Check this source for updates
https://www.redhat.com/security/data/oval/com.redhat.rhsa-2010.xml	X	✓ Check this source for updates

Add

RedHat Release Numbers:

New release number (i.e. 4): Add

OS Name	RedHat Release
Debian wheezy/sid	● ○ N/A
unknown	● ○ N/A

Save

Os Group Definitions

OS Group Name	Associated OSes
X	Add OS
X Debian	Add OS
X Fedora Core	Add OS unknown [remove]
X Ubuntu	Add OS Debian wheezy/sid [remove]

OS Group Name: Add new OS Group

Repository Definitions

Repository Name	Repository URL	OS Group	
X (i686,)	ftp://ftp.sucs.org/pub/linux/fedora/releases/15/Everything/i386/os/	Fedora Core	✓ Check this repository for updates
X (i686,)			✓ Check this repository for updates
X Debian Sqeeze (i686,)	http://ftp.de.debian.org/debian/dists/squeeze-updates/main/	Debian	✓ Check this repository for updates

OSes which do not have assigned any repository

Debian wheezy/sid

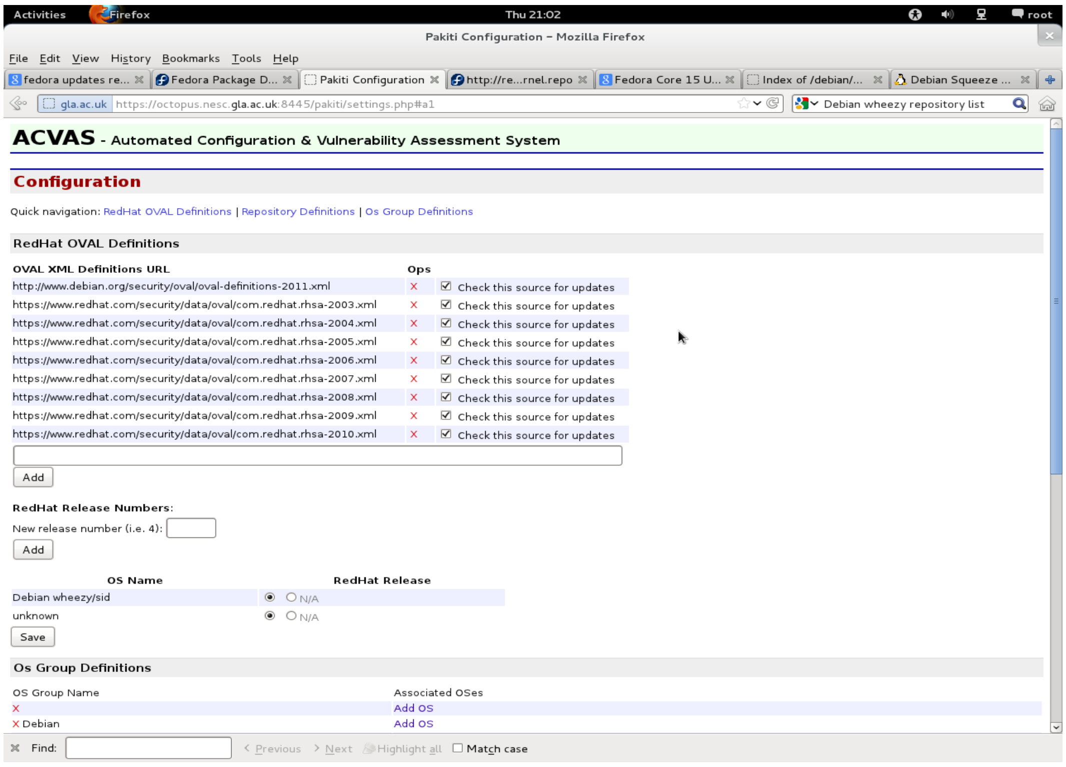
Repository Name: i686

URL (containg file primary.xml.gz or repomd.xml or Packages.gz): ✓ Contains Security Updates

Os Group:

Add new repository

Figure 7.2: Site administrators' view of ACVAS setting OS definition



ACVAS - Automated Configuration & Vulnerability Assessment System

Configuration

Quick navigation: RedHat OVAL Definitions | Repository Definitions | Os Group Definitions

RedHat OVAL Definitions

OVAL XML Definitions URL	Ops	
http://www.debian.org/security/oval/oval-definitions-2011.xml	X	✓ Check this source for updates
https://www.redhat.com/security/data/oval/com.redhat.rhsa-2003.xml	X	✓ Check this source for updates
https://www.redhat.com/security/data/oval/com.redhat.rhsa-2004.xml	X	✓ Check this source for updates
https://www.redhat.com/security/data/oval/com.redhat.rhsa-2005.xml	X	✓ Check this source for updates
https://www.redhat.com/security/data/oval/com.redhat.rhsa-2006.xml	X	✓ Check this source for updates
https://www.redhat.com/security/data/oval/com.redhat.rhsa-2007.xml	X	✓ Check this source for updates
https://www.redhat.com/security/data/oval/com.redhat.rhsa-2008.xml	X	✓ Check this source for updates
https://www.redhat.com/security/data/oval/com.redhat.rhsa-2009.xml	X	✓ Check this source for updates
https://www.redhat.com/security/data/oval/com.redhat.rhsa-2010.xml	X	✓ Check this source for updates

Add

RedHat Release Numbers:

New release number (i.e. 4):

Add

OS Name	RedHat Release
Debian wheezy/sid	● ○ N/A
unknown	● ○ N/A

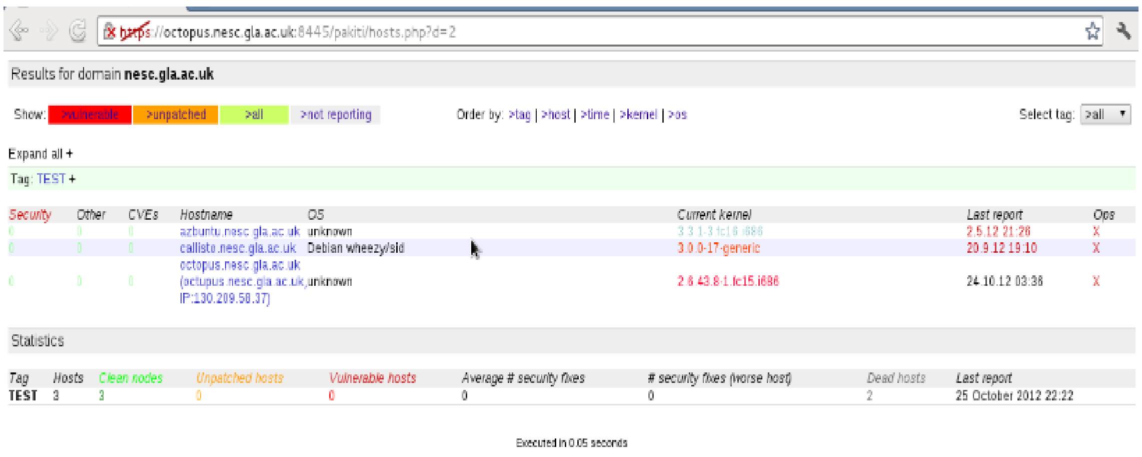
Save

Os Group Definitions

OS Group Name	Associated OSes
X	Add OS
X Debian	Add OS

Find: Previous Next Highlight all ☐ Match case

Figure 7.3: OVAL definition of trusted vendor sources for getting latest patch/update



~~https~~://octopus.nesc.gla.ac.uk:8445/pakiti/hosts.php?d=2

Results for domain **nesc.gla.ac.uk**

Show: >vulnerable >unpatched >all >not reporting

Order by: >tag | >host | >time | >kernel | >os

Select tag: >all

Expand all +

Tag: TEST +

Security	Other	CVEs	Hostname	OS	Current kernel	Last report	Ops
0	0	0	azbuntu.nesc.gla.ac.uk	unknown	3.3.1-3.fc16.i686	2.5.12 21:26	X
0	0	0	callisto.nesc.gla.ac.uk	Debian wheezy/sid	3.0.0-17-generic	20.9.12 19:10	X
0	0	0	octopus.nesc.gla.ac.uk (octupus.nesc.gla.ac.uk, IP:130.209.58.37)	unknown	2.6.43.8-1.fc15.i686	24.10.12 03:38	X

Statistics

Tag	Hosts	Clean nodes	Unpatched hosts	Vulnerable hosts	Average # security fixes	# security fixes (worse host)	Dead hosts	Last report
TEST	3	3	0	0	0	0	2	25 October 2012 22:22

Executed in 0.05 seconds

Figure 7.4: VO administrator's view of nodes in condensed form

three sites for the collaboration. Access to such information enables VO administrators to make sure that the data contributed by individual cross domain sites i.e. (Scotgrid, Manchester and Edinburgh).

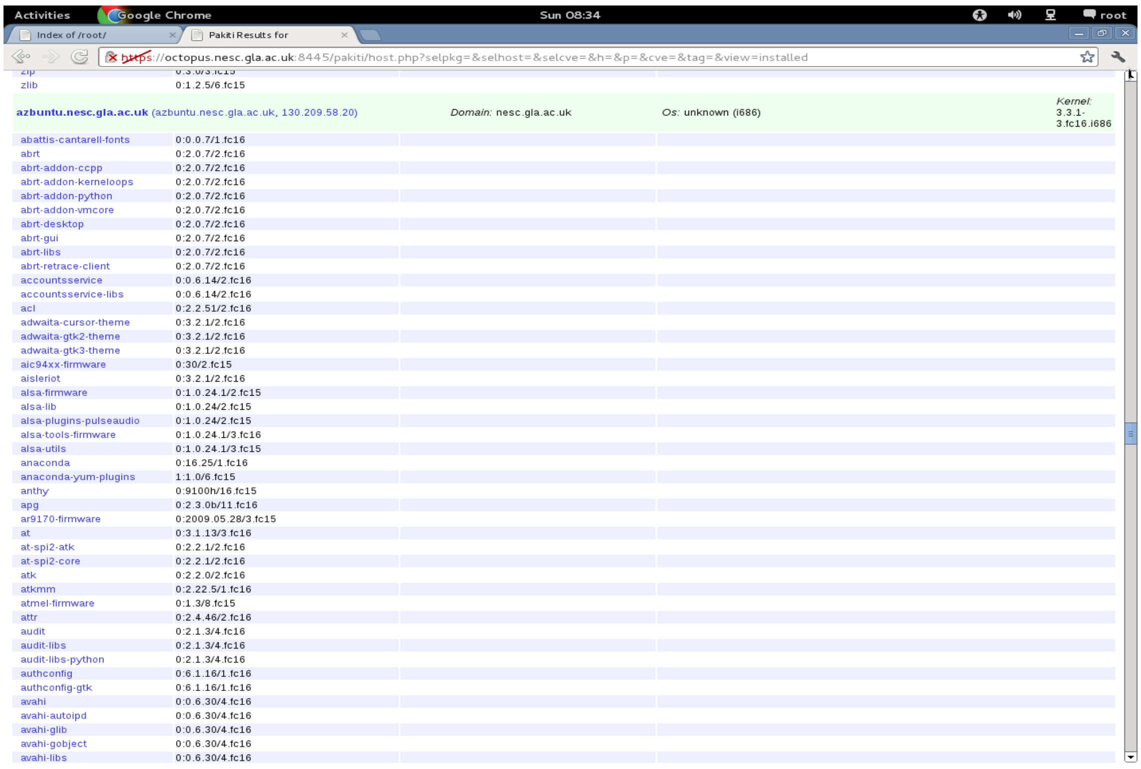
The detailed information about the installed packages on each single node of a VO is shown in Figure 7.3 in detailed form.

7.3 Conclusions

In this chapter we have outlined some of the supporting scenarios for ACVAS infrastructure for automated configuration & vulnerability. These scenarios support our implementation of ACVAS discussed in chapter 7 which basically deals with two main problems: identification of vulnerable software across VO-wide resources and applying security-oriented policies to control and monitor access to those important organisational asset contributed by several multi-tenant.

This chapter has presented the ACVAS infrastructure for automated configuration and vulnerability assessment and described its implementation. The chapter has dealt with two main problems: identifying vulnerable software across VO-wide resources and the automated application of security-oriented policies to affected resources. The ACVAS system allows secure verification of the patch status information of target hosts that subsequently enables VO-administrators to specify and download secure (hashed) updates of patches from remote repositories. The user interface of ACVAS allows site/VO administrators to obtain required patches and deploy them on-the-fly over the target affected VO-resources.

Sample policies for vulnerability assessment and configuration issues have been presented and



Activities Google Chrome Sun 08:34 root

Index of /root/ | Pakiti Results for

https://octopus.nesc.gla.ac.uk:8445/pakiti/host.php?selpkg=&selhost=&selcve=&h=&p=&cve=&tag=&view=installed

zlib	0:1.2.5/6.fc15

azbuntu.nesc.gla.ac.uk (azbuntu.nesc.gla.ac.uk, 130.209.58.20) *Domain:* nesc.gla.ac.uk *Os:* unknown (i686) *Kernel:* 3.3.1-3.fc16.i686

abattis-cantarell-fonts	0:0.0.7/1.fc16
abrt	0:2.0.7/2.fc16
abrt-addon-ccpp	0:2.0.7/2.fc16
abrt-addon-kerneloops	0:2.0.7/2.fc16
abrt-addon-python	0:2.0.7/2.fc16
abrt-addon-vmcore	0:2.0.7/2.fc16
abrt-desktop	0:2.0.7/2.fc16
abrt-gui	0:2.0.7/2.fc16
abrt-libs	0:2.0.7/2.fc16
abrt-retrace-client	0:2.0.7/2.fc16
accountsservice	0:0.6.14/2.fc16
accountsservice-libs	0:0.6.14/2.fc16
acl	0:2.2.51/2.fc16
adwaita-cursor-theme	0:3.2.1/2.fc16
adwaita-gtk2-theme	0:3.2.1/2.fc16
adwaita-gtk3-theme	0:3.2.1/2.fc16
aic94xx-firmware	0:30/2.fc15
aisleriot	0:3.2.1/2.fc16
alsa-firmware	0:1.0.24.1/2.fc15
alsa-lib	0:1.0.24/2.fc15
alsa-plugins-pulseaudio	0:1.0.24/2.fc15
alsa-tools-firmware	0:1.0.24.1/3.fc16
alsa-utils	0:1.0.24.1/3.fc15
anaconda	0:16.25/1.fc16
anaconda-yum-plugins	1:1.0/6.fc15
anthy	0:9100h/16.fc15
apg	0:2.3.0b/11.fc16
ar9170-firmware	0:2009.05.28/3.fc15
at	0:3.1.13/3.fc16
at-spi2-atk	0:2.2.1/2.fc16
at-spi2-core	0:2.2.1/2.fc16
atk	0:2.2.0/2.fc16
atkmm	0:2.22.5/1.fc16
atmel-firmware	0:1.3/8.fc15
attr	0:2.4.46/2.fc16
audit	0:2.1.3/4.fc16
audit-libs	0:2.1.3/4.fc16
audit-libs-python	0:2.1.3/4.fc16
authconfig	0:6.1.16/1.fc16
authconfig-gtk	0:6.1.16/1.fc16
avahi	0:0.6.30/4.fc16
avahi-autoipd	0:0.6.30/4.fc16
avahi-glib	0:0.6.30/4.fc16
avahi-gobject	0:0.6.30/4.fc16
avahi-libs	0:0.6.30/4.fc16

Figure 7.5: VO administrator's view of nodes in detailed form

implemented. Whilst showing the proof of concept, it is important to emphasize that these are, as stated, a proof of concept and a multitude of other vulnerability and patch-specific scenarios exist. It may be the case that the complete automation of patch management for other systems is an unattainable goal, e.g. where complex interdependencies of software system updates exist. ACVAS has however been designed to be generic and support a range of scenarios.

Chapter 8

Conclusion and Future Work

This chapter provides an overview of the work described and research questions addressed in this thesis. In the first part, it presents the assumptions on which the work has been based, and then draws conclusions and discussion of the ACVAS prototype. Finally it describes the potential areas for future work.

8.1 Assumptions

This work is based on several assumptions. Firstly, it assumes that a means of authentication exists for site/VO administrators across collaborating domains. This can be achieved by several well-established mechanisms, e.g. federated approaches such as through Shibboleth or centralised authentication models. One of the key success factors in any security model is support for single sign-on. Secondly, this work is based on the assumption that a limited trust relationship exists between site and VO administrators with regards to identification of vulnerabilities and the application of trustworthy (authentic) patches across organisation-wide affected resources (nodes). This implies that site administrators are able to identify vulnerabilities on site-specific resources and communicate with VO administrators on the vulnerability information and its resolution. The focus of this work is predominantly in the area of identification of vulnerable VO-specific resources and securely automating the application of trustworthy patches through targeted tool support for site/VO administrators. Thirdly, this work assumes that sites have security policies in place, e.g. regarding vulnerability scans and automatic patching of resources. This point is especially challenging since the vulnerabilities a site may have need to be highly protected from unauthorized access and use (exploitation).

8.2 Observations

As discussed in Chapter 2, the patch management component of ACVAS builds upon an intersection of two related fields: vulnerability management and configuration/change management. As noted in section 2.4, once vendors have identified vulnerabilities, patches are typically developed to address those vulnerabilities. However it is often the case that system/network administrators have to keep an up-to-date patch management system that includes regular testing and installation procedures for applying patches. This is especially important since patches that are introduced into systems can themselves potentially cause failures and further vulnerabilities, e.g. system conflicts that result in security vulnerabilities arising. This situation can create a “patch paradox” for system managers, where, without a patch an asset (system) is vulnerable to attacks, and with a patch the asset is potentially vulnerable to failures [59]. The process of patching and consequences of patching thus needs to be closely monitored and tested repeatedly across VO resources.

During the course of this research, we have observed that configuration and vulnerability management of resources across potentially geographically far-flung administrative boundaries is difficult to achieve. To address this challenge our main research question was (Chapter 1, section 1.2):

Can we proactively identify and resolve security vulnerabilities and system mis-configurations in an automated and generic way in a Grid-based environment?

Through the design and implementation of ACVAS prototype in (Chapters 4 & 5), we have proved that identification of security vulnerabilities and finding misconfigured nodes in an autonomous inter-organisation collaboration (VO) is possible to achieve. To further reinforce the ACVAS support, we have divided the main research question into five sub questions elaborated in next section (8.3).

8.2.1 Vulnerability Management Process

As identified in [93], the vulnerability management process normally commences with systems that do not have known vulnerabilities. ACVAS follows a vulnerability management processes which involves:

- Establishing and maintaining system information (inventory) including what type of hardware, operating system versions, running services and third-party applications are installed on organisational resources. This can be achieved either through manual or automated approaches.

- Obtaining reliable, exhaustive, and up-to-date vulnerability information related to VO resources. To support this, there are numerous open source and commercial software systems available that provide vulnerability information as discussed in Chapter 2. However at present, there are no automated tools for administrators to customise this vulnerability information for inter-organisational collaborations. This is especially challenging since vulnerability information can often come from various sources and have ambiguous text-based descriptions in different formats and using different terminologies.

- Assessing and evaluating vulnerabilities in organisational resources helps to determine the risks associated with collaborations a priori, i.e. before VOs are established. There are a number of free and commercial security vulnerability assessment tools that can help site administrators to find out vulnerabilities. However these tools typically do not support the adaptation to particular VO-specific needs and requirements. As a result, existing approaches result in an incomplete picture of the overall organisational systems and the potential threats they may give rise to.

8.3 Contribution

The primary contribution of this thesis is in the design and implementation of ACVAS and how it supports the integration of vulnerability assessment and application of patches across decentralised (autonomous) inter-organisational collaborations. Specifically the thesis contribution can be summarised as follows:

- **Identification of vulnerable software in heterogeneous environments:** ACVAS enables site/VO administrators to validate software to be deployed on local resource before vulnerabilities can arise. Information about the existence of potential vulnerabilities is essential site administrators to able to resolve them in a timely manner. For large-scale multi-site collaborations, identifying distributed and heterogeneous software vulnerabilities is increasingly demanding since VO-specific software often comes from a variety of sources including remote collaborators themselves.

- **Automatic verification of vulnerable software:** It is often a tedious and laborious activity to identify vulnerable software across multi-node environments. ACVAS facilitates the collection of detailed knowledge (inventory) of installed software across organisational nodes along with their versioning information; and supports the installation of patches and configuration of organisational nodes. The seamless integration of all of these activities is vital to successful operation of collaborations.

- **Application of trustworthy patches:** An authentic and trustworthy patch implies a verifiable and untampered patch. In ACVAS this is achieved through making sure that patches are digitally signed (by trusted vendors) and using their signature as a basis for determining their trustworthiness. Trust-based patch delivery and installation is successfully realised when a site or VO administrator successfully installs authentic patches across inter-organisational VO-specific resources. ACVAS automates this process and targets it to particular needs and requirements of VOs.

- **Automation of VO-wide vulnerability assessment:** In ACVAS an authorised vulnerability assessment request implies that only authorised remote system/VO administrators are able to invoke VO-specific vulnerability assessment and patch management tools. ACVAS allows remote site/VO administrators to conduct security assessments (automated vulnerability assessments), deploy patches and configure organisational resources in a seamless framework. ACVAS realises this through exploitation of VO-specific authentication and authorisation credentials that have been agreed as part of the establishment of the VO. ACVAS has at its heart the autonomy of sites and their local expression and enforcement of security policies on access to resources and importantly access to vulnerability and patching information.

8.4 Achievements of the ACVAS prototype

Through the design and development of the Automated Configuration and Vulnerability Assessment (ACVAS) framework, it has been shown how it is possible to address the problems of discovery of vulnerable software installed on organisation nodes across collaborating sites and the application of trusted patch management in a decentralised authorization-driven collaborative environment.

ACVAS differs from Pakiti [95] and Grid Site Software Vulnerability Analyzer (GSSVA) [190] in that it introduces the application of trusted patch delivery and definition of security policies for site/VO administrators. ACVAS provides multiple rules, which protect organisational assets by using trusted credentials and access policies in collaborative environments.

8.5 Suggestions and potential future work

Grid infrastructures such as the UK based National e-Infrastructure Service (NES) [194], Open Science Grid (OSG) [195] and Scotgrid [196] etc are typically funded and managed by consortia comprising universities, companies, and government bodies that can afford to make the necessary commitment of resources to achieve an adequate level of security. While implementing ACVAS prototype, we have covered several aspects of secure collaborations (VO) in previous chapters. These areas along with potential future ones are summerised below.

1. Security policies are an essential component of ACVAS prototype. In ACVAS we have focused on identification of vulnerabilities and application of security policies enabling site and/or VO administrators to potentially address particular VO security vulnerabilities. As discussed in Chapter 2, the NIST document [116] on patch management discusses patch management and the dilemma of patch management and other complexities. However for larger scale collaborations the assumption of "in-sprit" trust and certification of 'clean-sites' is largely still based on blind trust.

2. Through the literature survey (Chapter 2), we also argued that rather than blindly trusting collaborators to do the right thing, individuals and VO collaborations should be able to systematically assess the security of the overall VO systems and the associated underlying fabric. Specifically ACVAS allows to assess core questions identified in section 1.2, Chapter 1:

   a) How we can identify software versions across heterogeneous VOs?

   b) How do we verify that the information is reliable?

   c) How can we establish that a particular software version and/or configuration is vulnerable?

   d) How can we dynamically apply relevant patches to those affected nodes that may be a threat to the VO as a whole?

   e) How can we automate the vulnerability assessment process and allow relevant stakeholders to install and configure relevant remedial patches across distributed VO resources reflecting the local autonomous policies of sites involved?

3. Applications developed to run over the Grid are often produced by small groups of researchers, scientists, and engineers. Fenz et. al.[197] identified that Grid applications are often likely to contain bugs at the time of deployment. Often such user/groups are not trained in secure software engineering practices. Instead, they develop software to answer a focused research question that is expected to have a short life cycle. Tackling

this demands that software developers are cognisant of best practice in secure systems development – including development and delivery of patches. The process of certifying such systems also requires a step change in how software systems and developed and hardened.

4. There are also several issues with regards to policy specification for vulnerability information and applying patches to affected software. It is essential that sites are independent and cannot be dictated to regarding policy specification on disclosure of vulnerability information. However, prior to VO formation, questions such as "will/should the collaborating sites disclose their vulnerability information for their resources and/or the patch status of target nodes to collaborating sites?" must be answered. The trust relationship required to agree to such policies is extremely complex and typically goes against the grain of traditional site security approaches.

5. In building the ACVAS prototype, it became evident that until now insufficient attention has been paid to issues related to description and enforcement of policies on access to and use of dynamic Grid resources. Policies can be permissions, prohibitions or obligations or indeed desired usage scenario(s) expected by service/resource providers, collaborating VO members, or even the VO as a whole. Contemporary Grid collaborations have adopted "in-spirit" usage policies with no actual fine-grained enforcement, e.g. all resources are assumed to be contributed in kind [198], or where the policy enforcement on resource/service usage is at best ad-hoc. Collaborations that do define VO-wide policies typically only focus on resource usage policies, e.g. focused on target shares of CPU resources, and neglect the Grid resources/services such as disk and bandwidth. Due to insufficient resource/service usage policies and the relevant mechanisms for their enforcement, users, resources and service providers more generally are increasingly reluctant to participate in large-scale dynamic Grid-environments out of fear that their resources/services will be misused and/or overrun [199].

6. Further challenges on VOs establishment also exist. For example, how VO administrators cope with situations where a particular resource has been found to be 'vulnerable' and/or where a site administrator decides to apply a patch onto VO-specific resources with application dependencies or running jobs. At what point should vulnerability assessments happen and patching take place? Many major research infrastructure providers are obliged to be available as much as possible.

7. Contemporary cloud-based solutions such as cloud Inspect [200] and McAfee GetSusp [201] now allow customers to pay an extra amount to scan the target Cloud infrastructure prior to deploying their applications to make sure that the underlying fabric does not pose any security threat.

8.6 The future of Vulnerability Assessment

As we argued in various parts of this thesis that due to exponential growth in IT infrastruture, there are complexcities in software used. Despite of the efforts from aoftware industry the number of discovered vulnerabilities is increasing. Moreover, the site administrators and security staff are unable to cope with the vulnerability problem. This demands a and of comple of Vulnerability assessment tools have advantages and disadvantages that site administrators need to consider prior to joining a VO. These include:

- A particular vulnerability scanner can only assess a "snapshot of time" of a typical system. Ideally a regular scanning process should be conducted to improve the security. Human judgment is often vital to establishing reliable results from vulnerability scanners, since scanners only report vulnerabilities according to the information (plug-ins) associated with the scanner database. Due to the potential undetected nature of results from a vulnerability scan, i.e. the failure to recognise an existing flaw on a target resource, or indeed a false positive, i.e. the incorrect determination of the presence of vulnerability where none exists, human judgment is often required for analysing results of the scanning process.
- Since vulnerability scanners are purposely designed to discover known vulnerabilities, they cannot identify other security threats, e.g. the physical, configuration, operational or procedural issues and/or not yet identified/published vulnerabilities. In many cases, vulnerability scanners rely on software “plug-ins” to determine potential vulnerabilities in a target resource. With software complexities and the speed of malicious codes now being produced the need to dynamically update such software plug-in database information is a drawback of largely static vulnerability scanners.
- Vulnerability scanners can themselves pose major risks to organisational assets. As well as being the prime source of information that could lead to direct exploitations, repeated vulnerability scans can cause system, e.g. be used for denial of service scan attacks. To tackle this, risk assessment and planning is typically required prior to scan policies being put into place and actual scanning taking place.
- For successful VO operations, participating sites will clearly wish to keep their site-specific vulnerability information in a safe place. Leakage of scanning results containing information of system vulnerabilities would be an extremely dangerous if they fall into the wrong hands. ACVAS utilizes VO-specific policy specification and enforcement to ensure that only recognized (trusted) individuals have access to such information. This model is perhaps the most challenging aspect of ACVAS, i.e. would a site allow access to such information to VO collaborators. In this thesis, it is argued that a

weakness in any site is a potential threat to the VO as a whole. The risk analysis is thus an essential process in the application of ACVAS.

- When a degree of pre-existing trust exists among organisations, a common external party may be employed to help in the vulnerability assessment process. This model is being more commonplace with Cloud-based usage scenarios. Alternatively, VOs may designate a partner site responsible for undertaking vulnerability assessment (white/-grey vulnerability assessments) and potentially for maintaining the security of this information.

- Last but not least, the improper use of vulnerability scanning tools can obviously pose an enormous risk and cause tremendous harm to VO-wide resources and other information systems when misused and/or used beyond the agreed terms and conditions. Therefore, policies and procedures should be in place which clearly specify to whom, how and when vulnerability assessment tools can be used as part of the process of establishing a particular VO. These policies may potentially include the kind of prior VO formation (contract negotiation) process, management approval and/or legal clearances that are required before particular scans takes place.

Appendix A

A.1 Section

A.1 Scenario

A.1.1 Configure a node as webserver in a VO

In the first scenario a simple CMS configuration (based on CFengine) is used to configure a VO node as a webserver. FigureA.2 depicts the initial site.cf file for this scenario. In the example (Figure A.2), the CMS class grants permission to users 'thomas' and 'gordon' to remotely activate classes matching the regular expression 'extraordinary' when using the cf-runagent to activate CMS module.

In this way a Role Based Access Control (RBAC) system is supported for unprivileged users, i.e. users that do not have privileged access (root access) on the target host directly. However, the implicit assumption here is that the user identity is based on trusted CMS keys created by the user and exchanged with the server (as discussed in Section 5.1.2).

The webservers.cf file (see Figure A.3) contains a list of declarations that have to be fulfilled to add functionality to that server. The declaration shows a CFengine based declaration for the CMS module with access control rules for VO administrators to install and configure an Apache web server. First, the VO administrators allowed to install Apache packages are defined. Secondly, three additional promises are given for deploying the Apache configuration on the webservers.

A reload_apache class (promise) is set when one of these promises is executed. This means that when a VO admin changes the configuration files, CMS deploys the new configuration to the webservers Apache's configuration where it is subsequently reloaded. A further promise

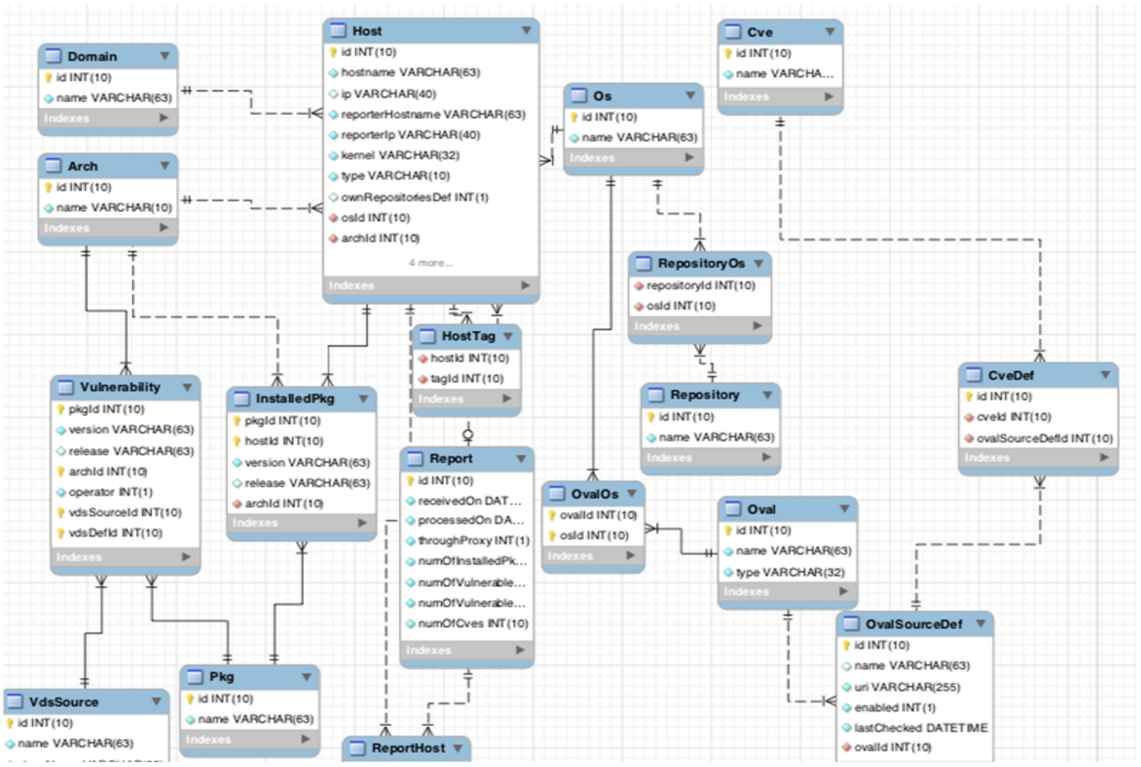


Figure A.1: VO-Inventory (VulMIS) schema

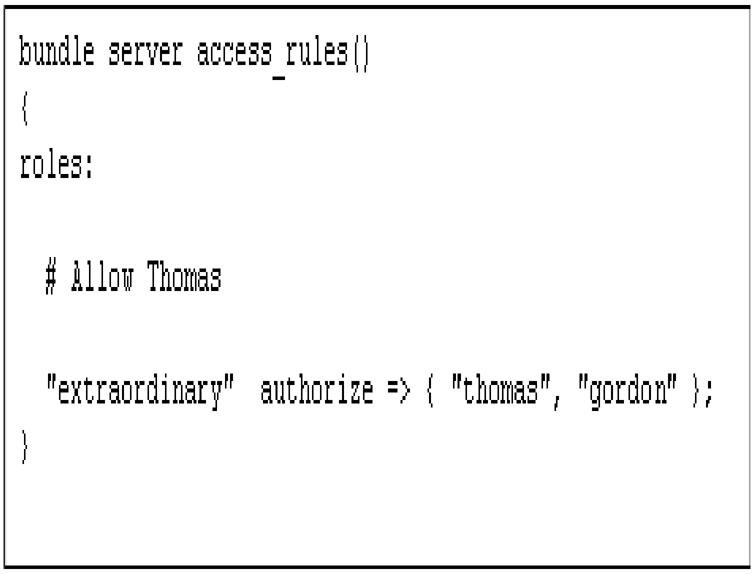
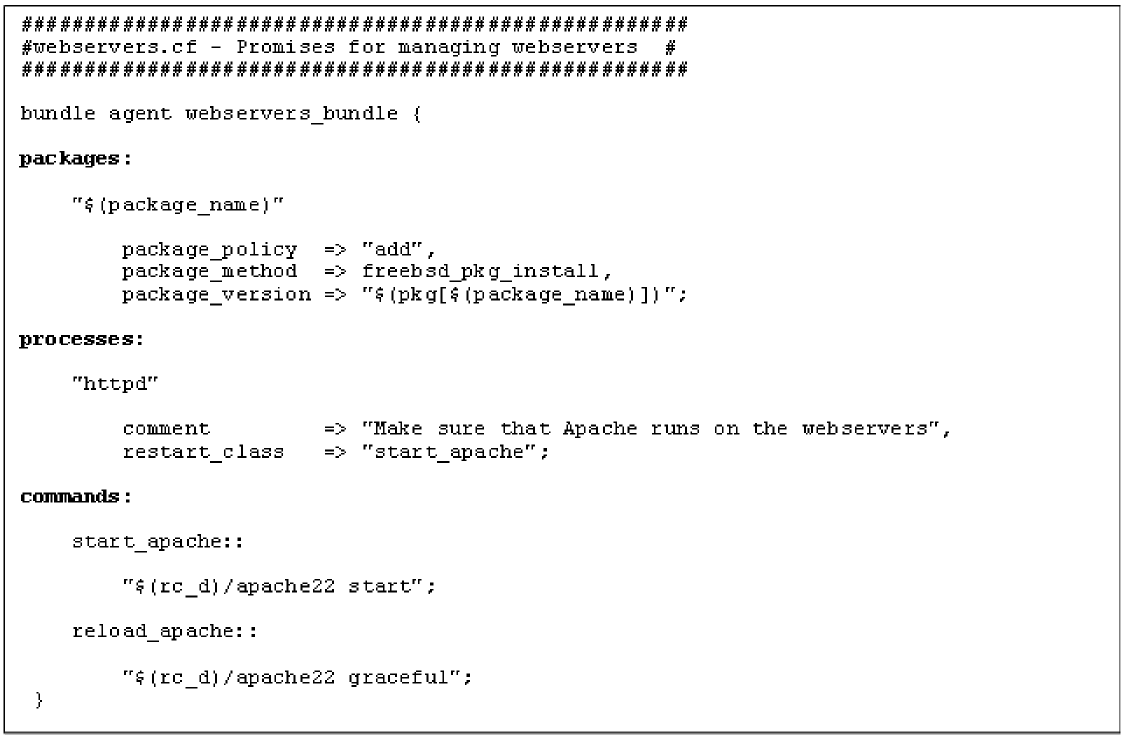

```
bundle server access_rules()
{
roles:

  # Allow Thomas

  "extraordinary"  authorize => { "thomas", "gordon" };
}
```

Figure A.2: The CMS promises to grant permission to users Thomas and Gordon



```
#######################################################
#webservers.cf - Promises for managing webservers  #
#######################################################

bundle agent webservers_bundle {

packages:

    "$(package_name)"

        package_policy  => "add",
        package_method  => freebsd_pkg_install,
        package_version => "$(pkg[$(package_name)])";

processes:

    "httpd"

        comment         => "Make sure that Apache runs on the webservers",
        restart_class   => "start_apache";

commands:

    start_apache::

        "$(rc_d)/apache22 start";

    reload_apache::

        "$(rc_d)/apache22 graceful";
 }
```

Figure A.3: The webserver.cf declaration of installing and running apache web server

checks if the Apache server is running in the processes: section. If Apache is not running it will be restarted.

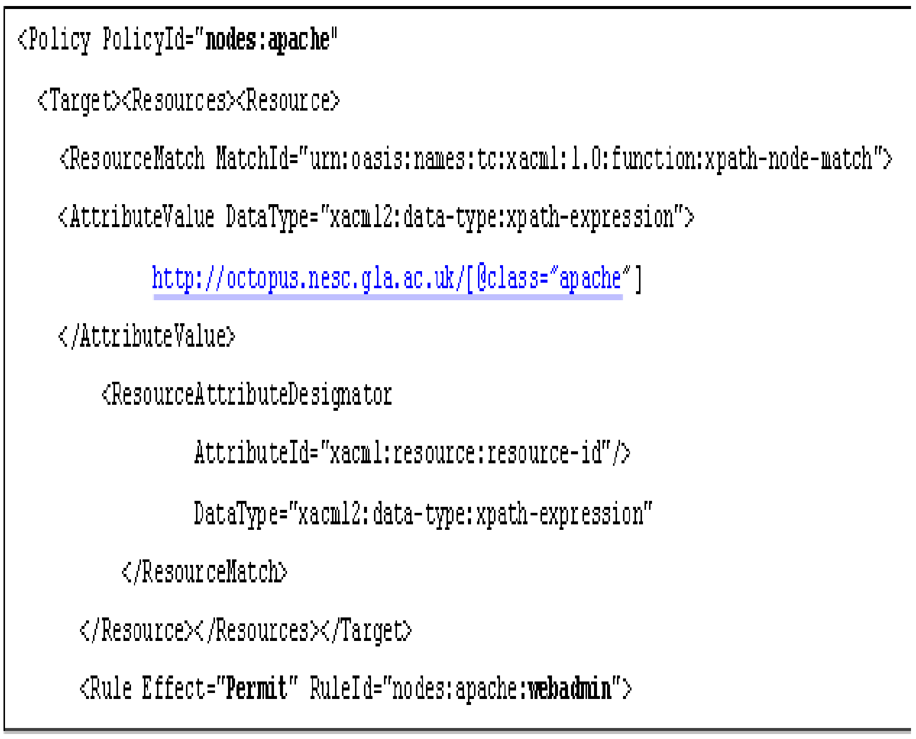
As depicted in Figure A.4, the XACML policy for this scenario allows a VO administrator from the webadmin group to include Apache classes in the configuration of a node. This provides fine-grained access control to the statements in the CMS declarations of Figure A.3.

The request format for XACML policy of Figure A.3 is depicted in Figure A.4 showing the add action.

The response related to request based in Figure A.5 is shown below in Figure A.6 with a permit decision to voadmin role based on the request shown in Figure A.5.

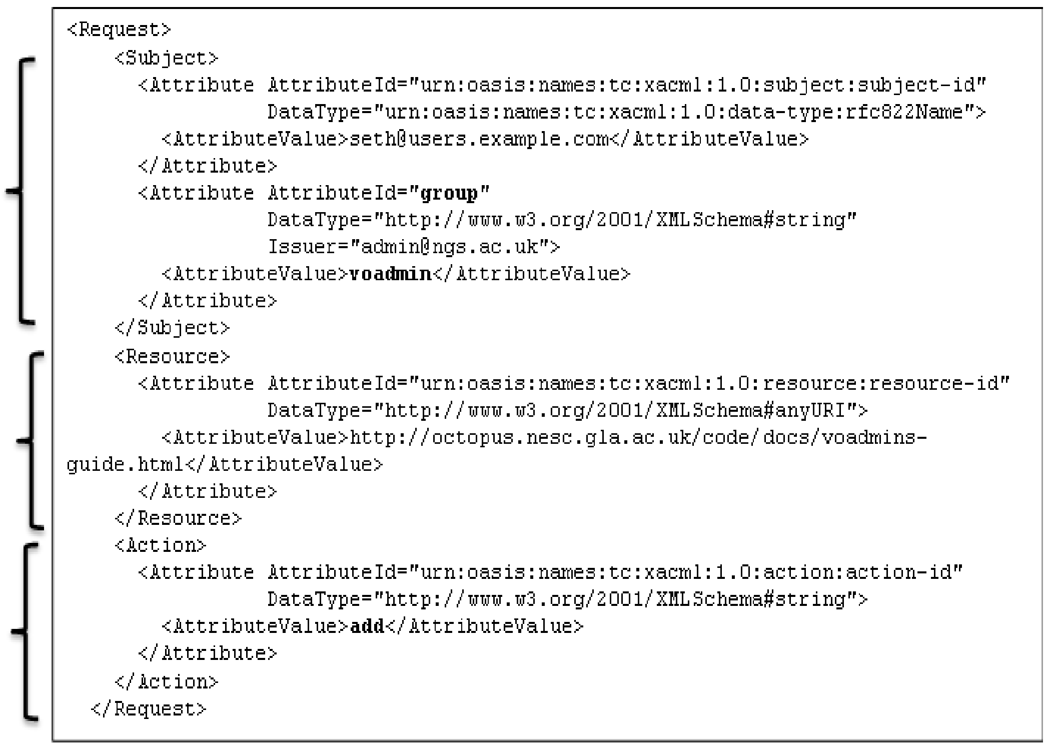
A.1.2 Adding virtual hosts to VO nodes

The second scenario used for evaluation uses the same Apache webserver setup (described in section A.2.1), and allows VO-administrators from the webuser group to add virtual hosts to the apache configuration. VO-admins can only add virtual hosts to the configuration when the documentroot is located in their own home directory. The document root parameter controls the directory that contains the files that a webserver should serve to the VO users of a particular



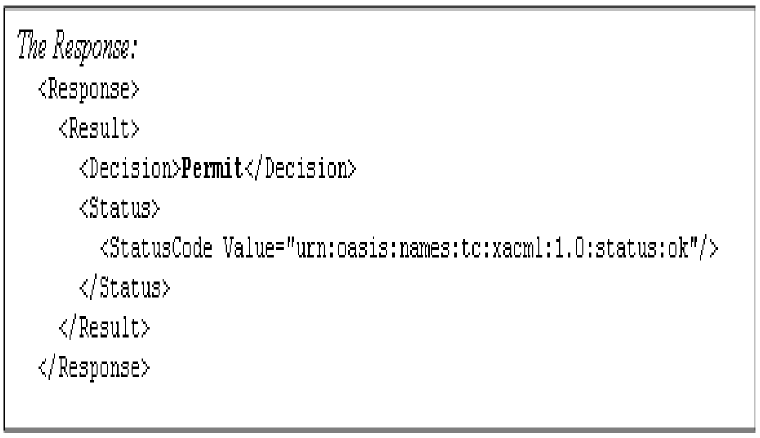
```
<Policy PolicyId="nodes:apache"
 <Target><Resources><Resource>
   <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:xpath-node-match">
   <AttributeValue DataType="xacml2:data-type:xpath-expression">
           http://octopus.nesc.gla.ac.uk/[@class="apache"]
   </AttributeValue>
       <ResourceAttributeDesignator
               AttributeId="xacml:resource:resource-id"/>
               DataType="xacml2:data-type:xpath-expression"
         </ResourceMatch>
     </Resource></Resources></Target>
     <Rule Effect="Permit" RuleId="nodes:apache:webadmin">
```

Figure A.4: The XACML-based policy to allow VO-admins group to add the apache class to sites



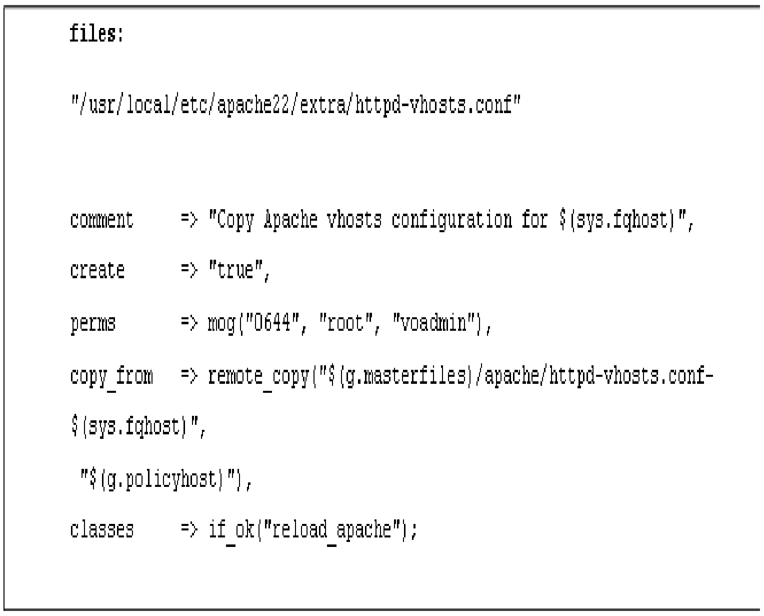
```
<Request>
    <Subject>
      <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
                 DataType="urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name">
        <AttributeValue>seth@users.example.com</AttributeValue>
      </Attribute>
      <Attribute AttributeId="group"
                 DataType="http://www.w3.org/2001/XMLSchema#string"
                 Issuer="admin@ngs.ac.uk">
        <AttributeValue>voadmin</AttributeValue>
      </Attribute>
    </Subject>
    <Resource>
      <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
                 DataType="http://www.w3.org/2001/XMLSchema#anyURI">
        <AttributeValue>http://octopus.nesc.gla.ac.uk/code/docs/voadmins-
guide.html</AttributeValue>
      </Attribute>
    </Resource>
    <Action>
      <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
                 DataType="http://www.w3.org/2001/XMLSchema#string">
        <AttributeValue>add</AttributeValue>
      </Attribute>
    </Action>
  </Request>
```

Figure A.5: VO-specific access request from VO admin



The Response:
```
<Response>
  <Result>
    <Decision>Permit</Decision>
    <Status>
      <StatusCode Value="urn:oasis:names:tc:xacml:1.0:status:ok"/>
    </Status>
  </Result>
</Response>
```

Figure A.6: VO-specific response to access request



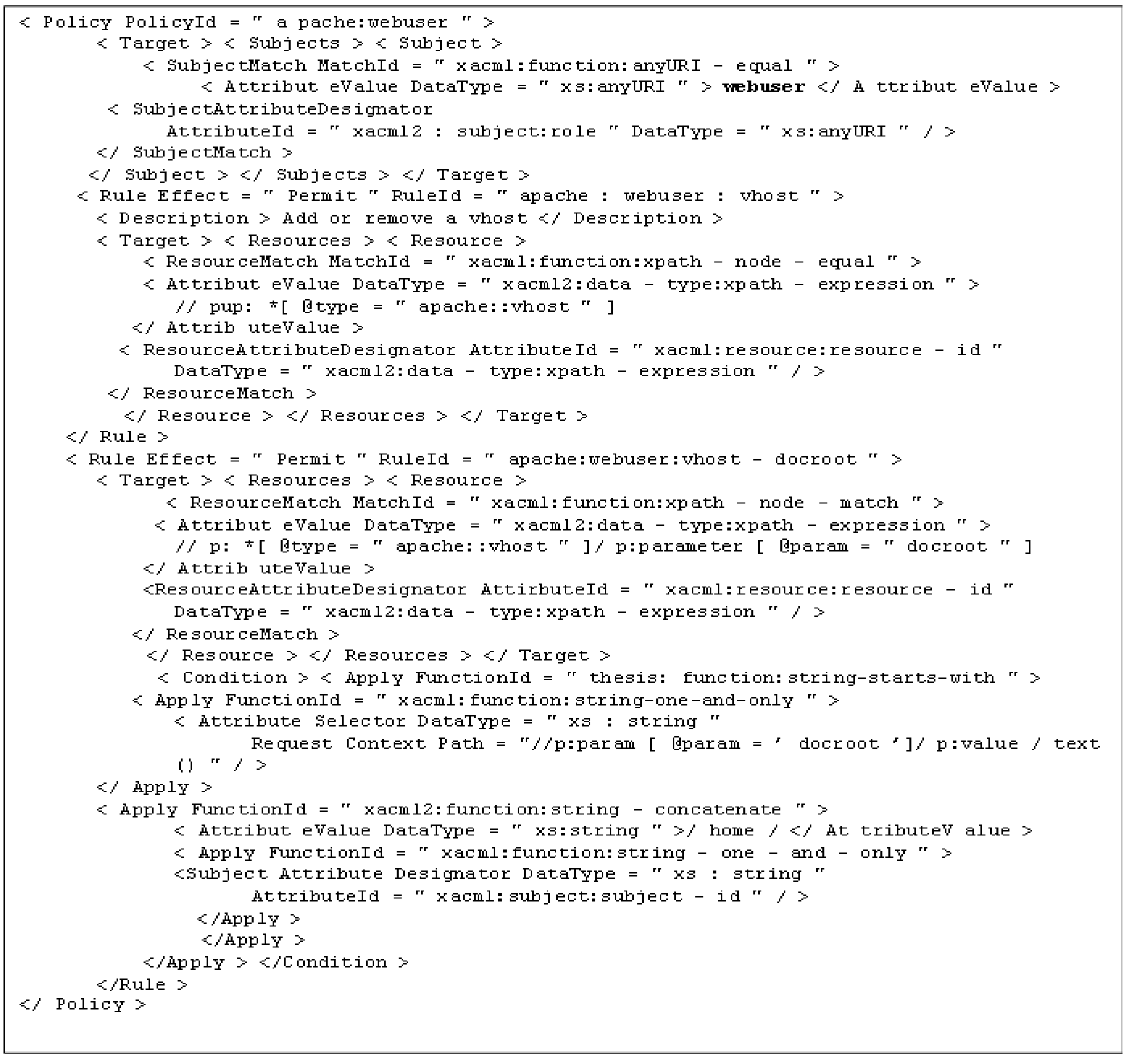
```
files:

"/usr/local/etc/apache22/extra/httpd-vhosts.conf"


comment     => "Copy Apache vhosts configuration for $(sys.fqhost)",

create      => "true",

perms       => mog("0644", "root", "voadmin"),

copy_from   => remote_copy("$(g.masterfiles)/apache/httpd-vhosts.conf-

$(sys.fqhost)",

 "$(g.policyhost)"),

classes     => if_ok("reload_apache");
```

Figure A.7: A VO-specific declaration to copy virtual host settings

domain. The CMS declaration shown in Figure A.7 shows a manifest from the Apache module that configures the virtual hosts in the system.

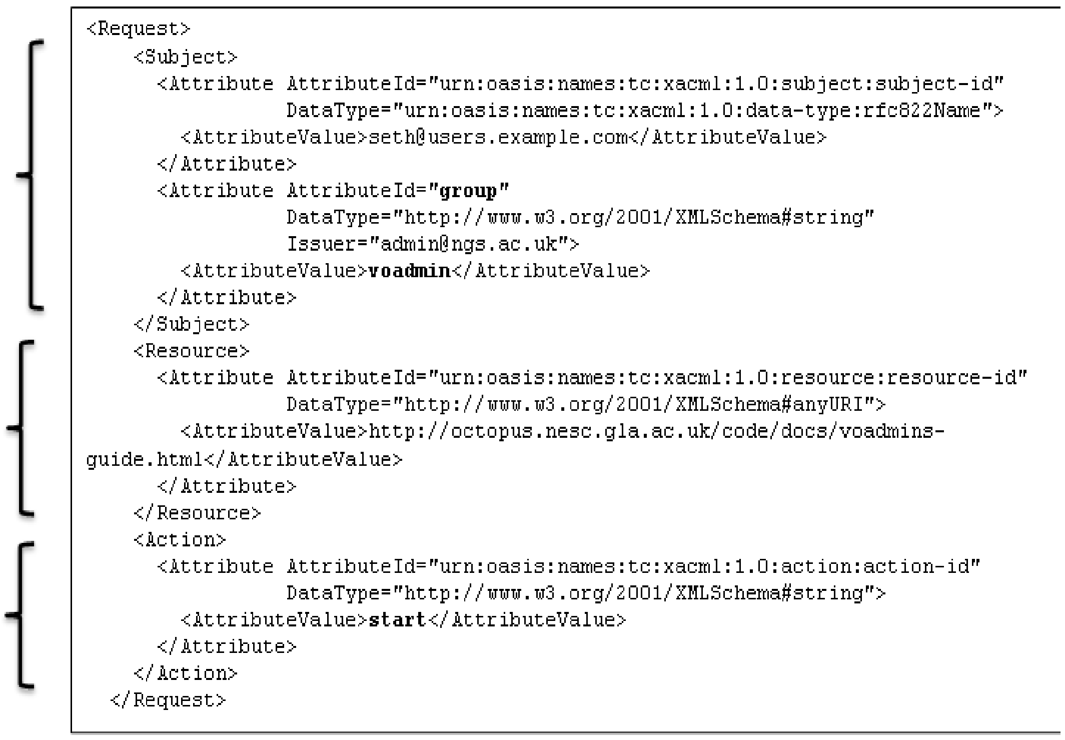
The XACML policy in Figure A.8 is used to enforce access control based on the contents of the change and not based on the file location. It enforces access control on all occurrences of virtual host resources in any CMS declaration file included in the repository. The policy builds the home directory of the user by concatenating /home/with the voadministrator. The value of the documentroot parameters of the virtual host resource start with the home directory policy being created.

The request format for the XACML policy of Figure A.8 is depicted in Figure A.9.



```
< Policy PolicyId = " a pache:webuser " >
      < Target > < Subjects > < Subject >
          < SubjectMatch MatchId = " xacml:function:anyURI - equal " >
               < Attribut eValue DataType = " xs:anyURI " > webuser </ A ttribut eValue >
        < SubjectAttributeDesignator
             AttributeId = " xacml2 : subject:role " DataType = " xs:anyURI " / >
       </ SubjectMatch >
      </ Subject > </ Subjects > </ Target >
     < Rule Effect = " Permit " RuleId = " apache : webuser : vhost " >
       < Description > Add or remove a vhost </ Description >
       < Target > < Resources > < Resource >
           < ResourceMatch MatchId = " xacml:function:xpath - node - equal " >
           < Attribut eValue DataType = " xacml2:data - type:xpath - expression " >
             // pup: *[ @type = " apache::vhost " ]
          </ Attrib uteValue >
         < ResourceAttributeDesignator AttributeId = " xacml:resource:resource - id "
             DataType = " xacml2:data - type:xpath - expression " / >
        </ ResourceMatch >
         </ Resource > </ Resources > </ Target >
    </ Rule >
    < Rule Effect = " Permit " RuleId = " apache:webuser:vhost - docroot " >
       < Target > < Resources > < Resource >
             < ResourceMatch MatchId = " xacml:function:xpath - node - match " >
            < Attribut eValue DataType = " xacml2:data - type:xpath - expression " >
             // p: *[ @type = " apache::vhost " ]/ p:parameter [ @param = " docroot " ]
           </ Attrib uteValue >
           <ResourceAttributeDesignator AttirbuteId = " xacml:resource:resource - id "
             DataType = " xacml2:data - type:xpath - expression " / >
          </ ResourceMatch >
           </ Resource > </ Resources > </ Target >
            < Condition > < Apply FunctionId = " thesis: function:string-starts-with " >
          < Apply FunctionId = " xacml:function:string-one-and-only " >
              < Attribute Selector DataType = " xs : string "
                    Request Context Path = "//p:param [ @param = ' docroot ']/ p:value / text
              () " / >
       </ Apply >
       < Apply FunctionId = " xacml2:function:string - concatenate " >
              < Attribut eValue DataType = " xs:string " >/ home / </ At tributeV alue >
              < Apply FunctionId = " xacml:function:string - one - and - only " >
              <Subject Attribute Designator DataType = " xs : string "
                    AttributeId = " xacml:subject:subject - id " / >
                </Apply >
                </Apply >
           </Apply > </Condition >
       </Rule >
</ Policy >
```

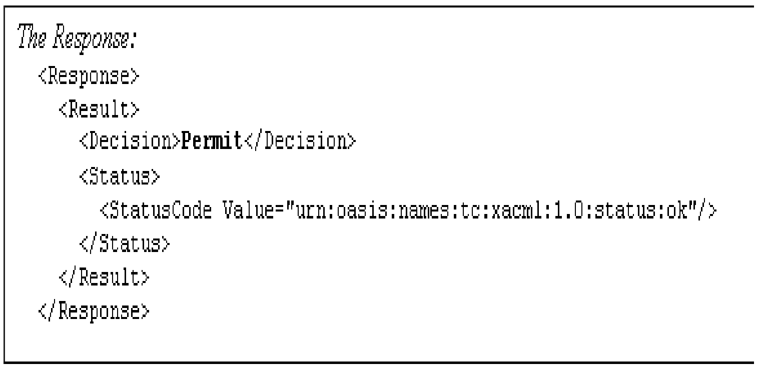
Figure A.8: XACML policy to allows VO-admins from webuser group to add vhosts root in their homedirectory



```
<Request>
    <Subject>
      <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
                 DataType="urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name">
        <AttributeValue>seth@users.example.com</AttributeValue>
      </Attribute>
      <Attribute AttributeId="group"
                 DataType="http://www.w3.org/2001/XMLSchema#string"
                 Issuer="admin@ngs.ac.uk">
        <AttributeValue>voadmin</AttributeValue>
      </Attribute>
    </Subject>
    <Resource>
      <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
                 DataType="http://www.w3.org/2001/XMLSchema#anyURI">
        <AttributeValue>http://octopus.nesc.gla.ac.uk/code/docs/voadmins-
guide.html</AttributeValue>
      </Attribute>
    </Resource>
    <Action>
      <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
                 DataType="http://www.w3.org/2001/XMLSchema#string">
        <AttributeValue>start</AttributeValue>
      </Attribute>
    </Action>
  </Request>
```

Figure A.9: VO-specific to access request according to policy

The response related to the request shown in Figure A.9 is shown in Figure A.10.



The Response:

```
<Response>
  <Result>
    <Decision>Permit</Decision>
    <Status>
      <StatusCode Value="urn:oasis:names:tc:xacml:1.0:status:ok"/>
    </Status>
  </Result>
</Response>
```

Figure A.10: VO-specific response to access request

Appendix A

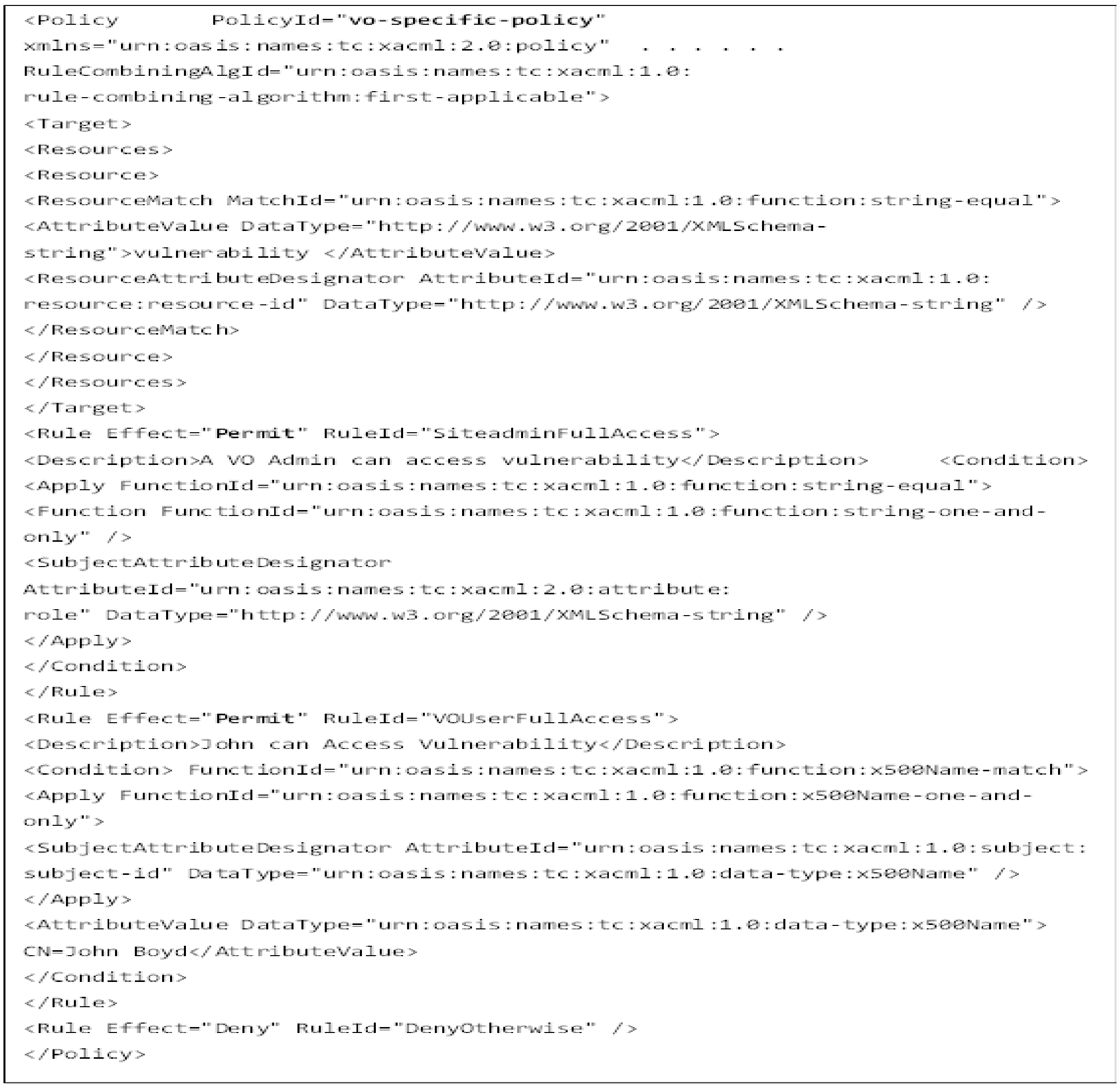
B.2 XACML Policies

The site-specific policy discussed in Chapter 5 is given in XML format in Figure 5.1 below.

The VO specific policy is shown in Figure B.2 below.

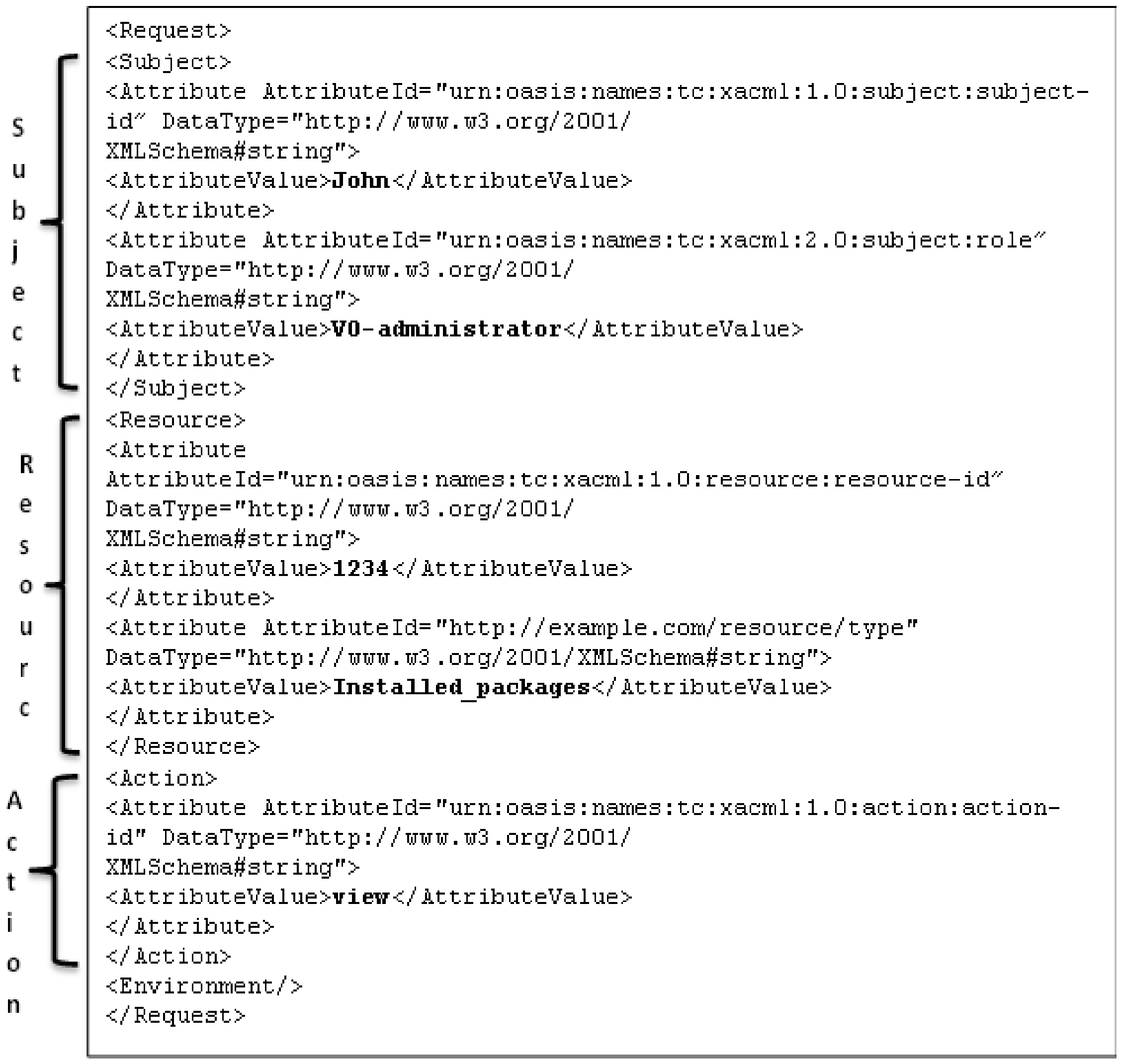
```
<Policy PolicyId="site_specific_policy"
xmlns="urn:oasis:names:tc:xacml:2.0:policy"  . . . . . .
RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-
algorithm:first-applicable">    <Target>
<Resources>
<Resource>
<ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
<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>
```

Figure A.1: A site-specific policy protecting ‘InstalledPackages’ resource



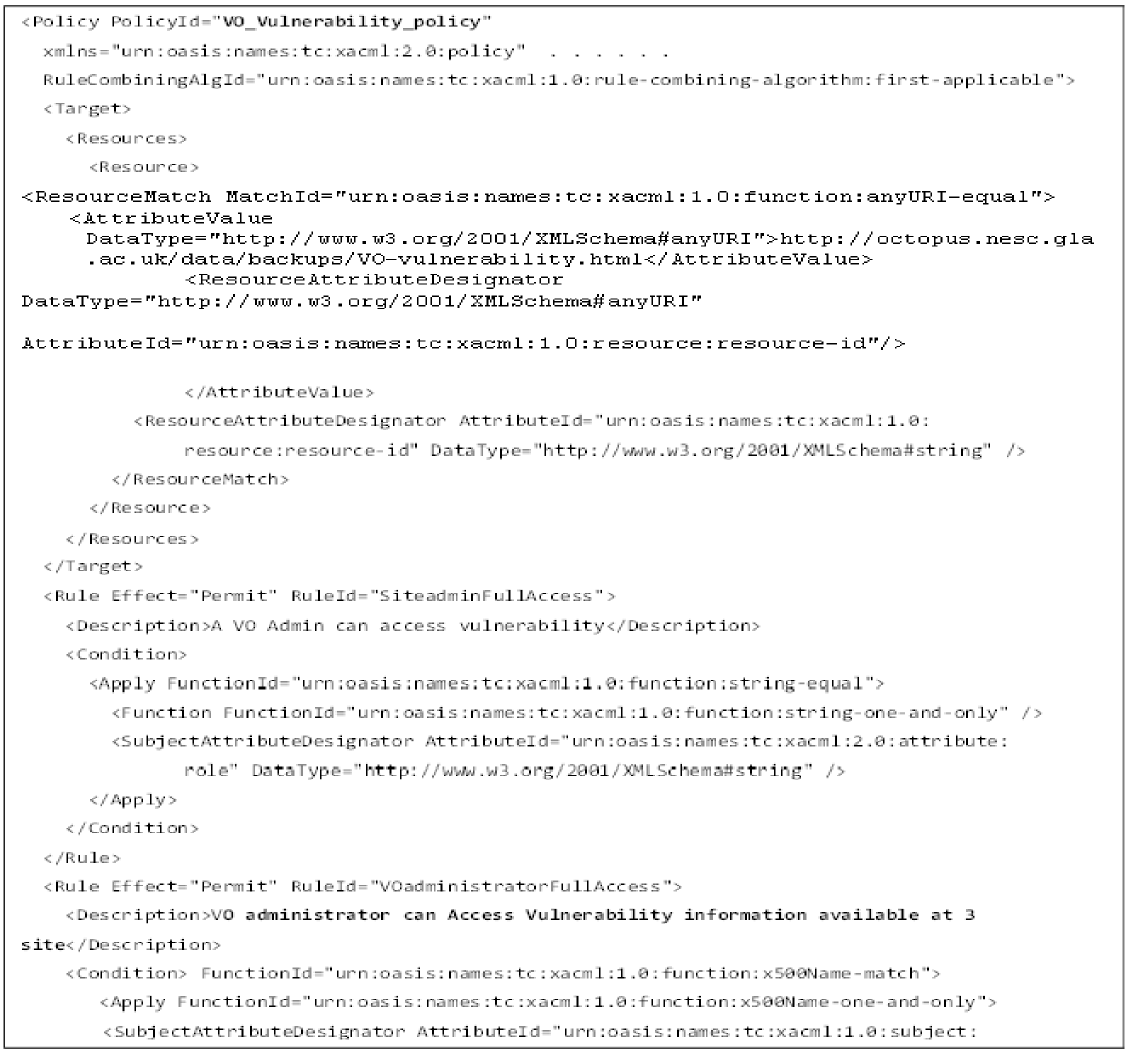
```
<Policy       PolicyId="vo-specific-policy"
xmlns="urn:oasis:names:tc:xacml:2.0:policy"  . . . . . .
RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:
rule-combining-algorithm:first-applicable">
<Target>
<Resources>
<Resource>
<ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
<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 VO 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>
```

Figure A.2: A VO specific policy protecting ‘Vulnerability’ resource


Subject
Resourc
Action

```
<Request>
<Subject>
<Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-
id" DataType="http://www.w3.org/2001/
XMLSchema#string">
<AttributeValue>John</AttributeValue>
</Attribute>
<Attribute AttributeId="urn:oasis:names:tc:xacml:2.0:subject:role"
DataType="http://www.w3.org/2001/
XMLSchema#string">
<AttributeValue>VO-administrator</AttributeValue>
</Attribute>
</Subject>
<Resource>
<Attribute
AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
DataType="http://www.w3.org/2001/
XMLSchema#string">
<AttributeValue>1234</AttributeValue>
</Attribute>
<Attribute AttributeId="http://example.com/resource/type"
DataType="http://www.w3.org/2001/XMLSchema#string">
<AttributeValue>Installed_packages</AttributeValue>
</Attribute>
</Resource>
<Action>
<Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-
id" DataType="http://www.w3.org/2001/
XMLSchema#string">
<AttributeValue>view</AttributeValue>
</Attribute>
</Action>
<Environment/>
</Request>
```

Figure A.3: XACML Request format to access resource at VulMIS



```
<Policy PolicyId="VO_Vulnerability_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:anyURI-equal">
    <AttributeValue
      DataType="http://www.w3.org/2001/XMLSchema#anyURI">http://octopus.nesc.gla
      .ac.uk/data/backups/VO-vulnerability.html</AttributeValue>
              <ResourceAttributeDesignator
DataType="http://www.w3.org/2001/XMLSchema#anyURI"

AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"/>

              </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 VO 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="VOadministratorFullAccess">
    <Description>VO administrator can Access Vulnerability information available at 3
site</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:
```

Figure A.4: VO-specific policy for accessing vulnerability data for sites in Fig 6.9

Bibliography

[1] "Protect yourself from the conficker worm virus." `http://www.microsoft.com/security/pc-security/conficker.aspx`, 2008.

[2] R. Evard, "An analysis of unix system configuration," in *Proceedings of the 11th USENIX conference on System administration*, LISA '97, (Berkeley, CA, USA), pp. 179–194, USENIX Association, 1997.

[3] R. C. Seacord and A. Householder, "A structured approach to classifying security vulnerabilities.," tech. rep., Carnegie Mellon Software Engineering Institute, 2005.

[4] J. Kaskade, "Future of security." `http://jameskaskade.com/?p=1340`, 2010.

[5] R. Naraine, "As attacks escalate, microsoft ships emergency windows patch." http://www.zdnet.com/blog/security/as-attacks-escalate-microsoft-ships-emergency-windows-patch/7027, August 2010.

[6] S. Fenz and A. Ekelhart, "Formalizing information security knowledge," in *Proceedings of the 4th International Symposium on Information, Computer, and Communications Security*, ASIACCS '09, (New York, NY, USA), pp. 183–194, ACM, 2009.

[7] "Altiris patch management solution." `http://www.altiris.com/upload/multimedia/patch/index.html`, 2010.

[8] "A practical approach to virtualization in hep," *The European Physical Journal Plus*, vol. 126, pp. 1–8, 2011. 10.1140/epjp/i2011-11013-1.

[9] A. Gould, S. Barker, E. Carver, D. Golby, and M. Turner, "Baegrid: From e-science to e-engineering- a progress report on grids in the aerospace industry," in *Proceedings of the e-Science All-Hands meeting*, 2003.

[10] T. Laboratories, Sufatrio, 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*, (Berkeley, CA, USA), pp. 47–58, USENIX Association, 2004.

[11] A. Tsalolikhin, "Monitoring system for crtical status– https://github.com/cfengine/design-center/blob/master/examples/verticalsysadmin-training-examples/220-1840-security.-check-open-ports.cf." February 2011.

[12] AT and T, "Collaboratin across borders: survey and white paper in cooperation with the economist intelligence unit." `http://www.corp.att.com/emea/docs/s5-collaboration-eng.pdf`, 2008.

[13] B. Iyer, J. Freedman, M. Gaynor, and G. Wyner, "Web services: Enabling dynamic business networks," in *Communications of the Association for Information Systems*, vol. 11, 2003.

[14] T. Ed, "Gartner research: Gartner says worldwide web conference and team collaboration software markets will reach usd 2.8 billon in 2010." http://www.gartner.com/it/page.jsp?id=507717, 2007.

[15] E. v. Heck and P. Vervest, "Smart business networks: how the network wins," *Commun. ACM*, vol. 50, pp. 28–37, June 2007.

[16] R. Alimi, Y. Wang, and Y. R. Yang, "Shadow configuration as a network management primitive," *SIGCOMM Comput. Commun. Rev.*, vol. 38, pp. 111–122, August 2008.

[17] O. David, G. Archana, and P. D. A., "Why do internet services fail, and what can be done about it?," in *Proceedings of the 4th conference on USENIX Symposium on Internet Technologies and Systems - Volume 4*, USITS'03, (Berkeley, CA, USA), pp. 1–1, USENIX Association, 2003.

[18] I. Foster, C. Kesselman, and S. Tuecke, "The anatomy of the grid: Enabling scalable virtual organizations," *International Journal of High Performance Computing Applications*, vol. 15, no. 3, pp. 200–222, 2001.

[19] K. Prince, "Malicious software defense: Have we moved beyond anti-virus and spyware protection software?," white paper, Perimeter eSecurity, 2007.

[20] J. Markoff, "Worm infects millions of computers worldwide; http://www.nytimes.com/2009/01/23/technology/internet/23worm.html," January 2009.

[21] K. Willsher, "French fighter planes grounded by computer virus, the daily telegraph." `http://www.telegraph.co.uk/news/worldnews/europe/france/4547649/French-fighter-planes-grounded-by-computer-virus.html`, 2009.

[22] "Computer emergency response team/coordination center (cert/cc)." `http://www.cert.org/certcc.html`, 2010.

[23] R. F. Dacey, "Effective patch management is critical to mitigating software vulnerabilities.," Technical Report (GA)-03-1138T, United States General Accounting Office, September 2003.

[24] "European grid infrastructure (egi)." `http://www.egi.eu/`.

[25] "Enabling grids for e-science." `http://www.eu-egee.org/`.

[26] "The open science grid (osg)." `https://www.opensciencegrid.org/bin/view`.

[27] "Cfengine–http://www.cfengine.org." http://www.cfengine.org.

[28] R. A. G. Leiva, M. B. Lopez, G. C. Meliá, B. C. Marco, L. Cons, P. Poznanski, A. Washbrook, E. Ferro, and A. Holt, "Quattor: Tools and techniques for the configuration, installation and management of large-scale grid computing fabrics," *J. Grid Computing*, vol. 2, no. 4, pp. 313–322, 2004.

[29] R. Santinelli and F. Donno, "Installing and configuring application software on the lhc computing grid," in *Proceedings of the First International Conference on e-Science and Grid Computing*, E-SCIENCE '05, (Washington, DC, USA), pp. 369–376, IEEE Computer Society, 2005.

[30] R. Housley and T. Polk, *Planning for PKI: Best Practices Guide for Deploying Public Key Infrastructure*. Willey Computer Publishing, 2001.

[31] "Internet2 shibboleth technology." `http://shibboleth.internet2.edu`, 2009.

[32] S. Bellovin and R. Bush, "Configuration management and security," *Selected Areas in Communications, IEEE Journal on*, vol. 27, no. 3, pp. 268 –274, 2009.

[33] J. Martin, D.M., S. Rajagopalan, and A. Rubin, "Blocking java applets at the firewall," in *Network and Distributed System Security, 1997. Proceedings., 1997 Symposium on*, pp. 16 –26, Feb. 1997.

[34] E. Al-Shaer, C. Kalmanek, and F. Wu, "Journal of network and systems management; automated security configuration management," vol. 16, no. 3, pp. 231–233, 2008.

[35] L. Teo and Y. Zheng, "Secure and automated software updates across organizational boundaries," in *Workshop on Information Assurance and Security*, vol. 1, pp. 30–37, Citeseer, IEEE Computer Society, 2002.

[36] "A quantitative study of firewall configuration errors," *Computer*, vol. 37, pp. 62–67, June 2004.

[37] S. Bellovin, "Distributed firewalls," *Journal of Login*, vol. 24, no. 5, pp. 37–39, 1999.

[38] "Smartfrog web site:." `http://www.smartfrog.org`.

[39] "Ogsa config web site:." `http://groups.inf.ed.ac.uk/ogsaconfig`.

[40] P. Anderson, *System Configuration*, vol. 14 of *Short Topics in System Administration*. SAGE, 2006.

[41] P. Anderson, "Why configuration management is crucial," *Login*, vol. 31, no. 1, pp. 5–8, 2006.

[42] D. Narayan, R. Bradshaw, and J. Hagedorn, "System management methodologies with bcfg2," *Login*, vol. 3, no. 1, pp. 11–18, 2006.

[43] D. Narayan, B. Rick, and L. Cory, "Directing change using bcfg2," in *Proceedings of the 20th conference on Large Installation System Administration*, (Berkeley, CA, USA), pp. 17–17, USENIX Association, 2006.

[44] P. Anderson and A. Scobie, "LCFG: The Next Generation," in *UKUUG Winter Conference*, UKUUG, 2002.

[45] N. Desai, A. Lusk, R. Bradshaw, and Evard, "Bcfg: A configuration management tool for heterogeneous environments," 2003.

[46] A. Paul, G. Patrick, and P. Jim, "Smartfrog meets lcfg: Autonomous reconfiguration with central policy control," in *Proceedings of the 17th USENIX conference on System administration*, (Berkeley, CA, USA), pp. 213–222, USENIX Association, 2003.

[47] "Zero configuration networking (zeroconf)." `http://www.zeroconf.org`.

[48] G. Beckett, K. Kavoussanakis, G. Mecheneau, P. Toft, P. Goldsack, P. Anderson, J. Paterson, and C. Edwards, "Gridweaver: automatic, adaptive, large-scale fabric configuration for grid computing," 2003.

[49] T. Escamilla, *Intrusion Detection: Network security behind firewall*. New York, NY, USA: John Wiley & Sons, Inc., 1998.

[50] C. Phillips and L. P. Swiler, "A graph-based system for network-vulnerability analysis," in *Proceedings of the 1998 workshop on New security paradigms*, NSPW '98, (New York, NY, USA), pp. 71–79, ACM, 1998.

[51] S. M. Sean Collins, "Stuxnet: the emergence of new cyber weapon and its implications," *Journal of Policing, Intelligence and Counter Terrorism*, vol. 7, pp. 80–91, April 2012.

[52] T. M. Chen and S. Abu-Nimeh, "Lessons from stuxnet," *IEEE Computer*, vol. 44, no. 4, pp. 91–93, 2011.

[53] J. Aron, "Flame virus used world-class cryptographic attack– http://www.newscientist.com/blogs/onepercent/2012/06/flame-used-world-class-cryptog.html."

[54] M. Kelly, "Flame virus offshoot burns high-profile victims– http://venturebeat.com/2012/10/15/miniflame-malware/."

[55] M. Dekker, "Security of the internet." `http://www.cert.org/encyc_article/tocencyc.html`, 1997.

[56] T. R. Peltier, J. Peltier, and J. A. Blackley, *Managing a Network Vulnerability Assessment*. Auerbach Publishers Inc., 2003.

[57] A. W. A., F. W. L., and M. John, "Windows of vulnerability: A case study analysis," *IEEE Computer Society*, vol. 33, pp. 52–59, December 2000.

[58] R. A. Martin, "Managing vulnerabilities in networked systems," *IEEE Computer Society*, vol. 34, pp. 32–38, November 2001.

[59] D. S. D. White, "Limiting vulnerability exposure through effective patch management: threat mitigation through vulnerability remediation," Master's thesis, Department of Computer Science, Rhodes University, January 2006.

[60] "Global report reveals security concerns hinder adoption of web 2.0 and social networking in business," McAfee, 2010.

[61] "Network vulnerabilities a continued problem in 2011," Secunia, 2011.

[62] P. Ray., "Human error website. research website." `http://panko.shidler.hawaii.edu/HumanErr/`, 2005.

[63] M. W. B. Roberge, "Introduction to oval: A new language to determine the presence of software vulnerabilities." `http://www.robergewriting.com/documents/OVALIntroductoryWhitePaper-November2003_000.pdf`, Novemeber 2003.

[64] "Oval (open vulnerabilities and assessment language)." http://oval.mitre.org/.

[65] A. Rakshit and X. Ou, "A host-based security assessment architecture for industrial control systems," in *Resilient Control Systems, 2009. ISRCS '09. 2nd International Symposium on*, pp. 13 –18, aug. 2009.

[66] C. S. 2005-2010. `http://www.cert.org/stats/`, January 2011.

[67] "Common vulnerabilities and exposures list (cve)." `http://cve.mitre.org/cve/`, May 2011.

[68] "Total number of vulnerabilities." `http://http://www.symantec.com/threatreport/topic.jsp?id=vulnerabilitytrends&aid=totalnumberofvulnerabilities`.

[69] S. Frei, "Vulnerability threat trends–a decade in review, transition on the way (http://aroundcyber.files.wordpress.com/2013/02/2013-ab-vulnerability-threat-trends.pdf)."

[70] G. Eschelbeck, "The laws of vulnerabilities: Which security vulnerabilities really matter?," *Inf. Secur. Tech. Rep.*, vol. 10, pp. 213–219, January 2005.

[71] H. Kevin and W. George., "Trends in denial of service attack technology." `http://www.cert.org/archive/pdf/DoS_trends.pdf`.

[72] H. K. Browne, W. A. Arbaugh, J. McHugh, and W. L. Fithen, "A trend analysis of exploitations," in *Proceedings of the 2001 IEEE Symposium on Security and Privacy*, (Washington, DC, USA), pp. 214–, IEEE Computer Society, 2001.

[73] "Sophos security threat management report 2005. technical report." `http://www.sophos.com/virusinfo/whitepapers/SophosSecurity2005-mmuk.pdf`, 2005.

[74] "Hacker defender antidetection service.." `http://hxdef.czweb.org/about.php`.

[75] "Distributed intrusion detection system, the internet's early warning system and internet security community site." `http://www.dshield.org/`.

[76] D. Moore, G. M. Voelker, and S. Savage, "Inferring internet denial-of-service activity," in *Proceedings of the 10th conference on USENIX Security Symposium - Volume 10*, SSYM'01, (Berkeley, CA, USA), pp. 2–2, USENIX Association, 2001.

[77] J. D. Howard, "An analysis of security incidents on the internet." `http://www.cert.org/research/JHThesis/Chapter12.html`, 1997.

[78] V. Yegneswaran, P. Barford, and J. Ullrich, "Internet intrusions: global characteristics and prevalence," in *Proceedings of the 2003 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, SIGMETRICS '03, (New York, NY, USA), pp. 138–147, ACM, 2003.

[79] "Internet storm center: Survival time." `http://isc.sans.edu/survivaltime.html`, 2005.

[80] M. Barrère, R. Badonnel, and O. Festor, "Towards vulnerability prevention in autonomic networks and systems," 2011.

[81] M. Barrere, R. Badonnel, and O. Festor, "Supporting Vulnerability Awareness in Autonomic Networks and Systems with OVAL," in *Network and Service Management - CNSM'11*, (Paris, France), p. N/A, Oct. 2011.

[82] M. Barrère, R. Badonnel, and O. Festor, "Towards vulnerability prevention in autonomic networks and systems," 2011.

[83] D. Jones, "Defending against client-side attacks through a vulnerability management program." http://www.rwsp.org/projects/127-defending-against-client-side-attacks-through-a-vulnerability-management-program.html, May 2011.

[84] A. Horan, "The ghosts of microsoft: Patch, present and future," *SC Magazine*, December 2012.

[85] Y.-C. W. K. A.-B. C. Tanguma, "An exploratory study of the security management practices of hispanic students," *International Journal of Security*, vol. 5, pp. 13–21, May 2011.

[86] *Studying SoftwareVulnerabilities*, vol. 23, CROSSTALK, September/October 2010.

[87] Y. Wu, H. Siy, and R. Gandhi, "Empirical results on the study of software vulnerabilities: Nier track," in *Proceeding of the 33rd international conference on Software engineering*, ICSE '11, (New York, NY, USA), pp. 964–967, ACM, 2011.

[88] "When will hp supply patches before they are required? usenet." `http://groups.google.com/group/comp.sys.hp/browse_thread/thread/e065debcf70b5ec0/5cd814ab642863ce`, 1992.

[89] D. Sancho, "The future of bot worms: What we can expect from worm authors in the coming months.," technical report, Trend Micro, 2005.

[90] T. Dean, E. Stephen, F. Oliver, A. David, H. Daniel, F. Marc, G. Sarah, S. Peter, C. Eric, C. David, M. Dylan, and B. Brad, "Symantec internet security threat report: Trends for july 04-december 04.," Tech. Rep. Volume VII, Symantec, 2005.

[91] D. S. D. White, "Limiting vulnerability exposure through effective patch management: threat mitigation through vulnerability remediation," Master's thesis, Department of Computer Science, Rhodes University, January 2006.

[92] "Overview of attack trends (cert/cc)." `http://www.cert.org/archive/pdf/attack/trends.pdf`, December 2005.

[93] H. T. Tian, L. S. Huang, Z. Zhou, and Y. L. Luo, "Arm up administrators: Automated vulnerability management," in *International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN'04)*, 2004.

[94] "Tivoli web site:." `http://www.tivoli.com`.

[95] "Pakiti: A patching status monitoring tool." `http://pakiti.sourceforge.net/`.

[96] "Securityfocus." `http://www.securityfocus.com`.

[97] "Bugtrack." `http://www.bugtrack.net/`.

[98] "Icat-cve vulnerability search engineweb site:." `http://www.sadikhov.com/forum/index.php?/topic/644-icat-cve-vulnerability-search-engine/`.

[99] "Securitytracker." `http://securitytracker.com/`.

[100] "Nessus–the network vulnerability scanner." `http://www.nessus.org/nessus/intro.php`, 2010.

[101] "Ibm internet scanner." `http://www.proventiaworks.com/Internet-Scanner.asp`.

[102] C. Ardagna, C. Braghin, and M. Cremonini, "What is vulnerability assessment?," in *Computer And Information Security Handbook* (J. Vacca, ed.), Morgan Kaufmann, 2009.

[103] "Netrecon." `http://www.securityinnovation.com/security-report/october/tools/NetRecon.htm`.

[104] T. Iwanski, "Network vulnerability scanners." `http://www.windowsitpro.com/article/product-review/network-vulnerability-scanners`, February 2002.

[105] Mark, "Patch management:." `http://kohi10.wordpress.com/2009/12/27/patch-management/`, December 2009.

[106] E. O'Conner, "Patch management best practices," *BigAdmin System Administration Portal*, April 2008.

[107] J. Howard, "A general computer vulnerability checker," Master's thesis, Australian Defence Force Academy, 1999.

[108] D. E. Mann and S. M. Christey, "Towards a common enumeration of vulnerabilities." `http://cve.mitre.org/docs/docs-2000/cerias.html`, Jan. 1999.

[109] "An overview of vulnerability scanners." `http://www.infosec.gov.hk/english/technical/articles.html`, February, 2008.

[110] R. J. Colville, R. Wagner, and M. Nicolett, "Patch management benefits, challenges and prerequisites," 2002.

[111] K. Poulsen, "Nachi worm infected diebold atms." `http://www.securityfocus.com/news/7517`, November 2003.

[112] L. Harding, "Court hears how teenage introvert created devastating computer virus in his bedroom. the guardian newspaper," July 2005.

[113] "Us army information assurance division. army regulation 25-2. glossary." `https://ia.signal.army.mil/default.asp`.

[114] S. Gary, G. Alice, and F. Alexis., "Risk management guide for in- formation technology systems.," Special Publication 800-30 MD 20899-8930, National Institute of Standards (NIST), Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology, Gaithersburg, July 2002.

[115] "The security risk management guide," technical report, Microsoft, October 2004.

[116] P. Mell, T. Bergeron, and D. Henning, "Creating a patch and vulnerability management program," Special Publication 800-40 Version 2.0, National Institute of Standards and Technology (NIST), 2005.

[117] "Guideline for management of it security part 1: Concepts and models for it security," technical report, ISO IEC, 1996.

[118] "Definition: Patch. the jargon file.." http://www.wordiq.com/definition/Patch

[119] G. Moses, L. Chunmei, and N. Washington, "A comparative analysis of anti-malware software, patch management, and host-based firewalls in preventing malware infections on client computers," in *Proceedings of the Fifth International Conference on Information Technology: New Generations*, (Washington, DC, USA), pp. 628–632, IEEE Computer Society, 2008.

[120] D. Voldal, “A practical methodology for implementing a patch management process,” tech. rep., SANS Institute, 2003.

[121] “Security tools-microsoft technet.” `http://technet.microsoft.com/en-us/security/cc297183`, 2005.

[122] Workshop on the Economics of Information Security (WEIS 2006), *Economics of Security Patch Management*, Robinson College, University of Cambridge, England, 2006.

[123] L. McGhie, “Software patch management - the new frontier.,” *Secure Business Quarterly*, 2003.

[124] D. M., “Patch management - bits, bad guys, and bucks!,” tech. rep., Secure Business Quarterly, 2003.

[125] J. Feng, G. Wasson, and M. Humphrey, “Resource usage policy expression and enforcement in grid computing,” in *Proceedings of the 8th IEEE/ACM International Conference on Grid Computing*, GRID ’07, (Washington, DC, USA), pp. 66–73, IEEE Computer Society, 2007.

[126] “The truth about software as a service (saas).” `http://www.cio.com/article/109706/TheTruthAboutSoftwareasaServiceSaaS`, May 2007.

[127] J. Staten, O. King, J. Belissent, C. Wang, Z. Reiss-Davis, and L. E. Nelson, “Infrastructure-as-a-service (iaas) clouds are local and so are their implications.” `http://www.forrester.com/rb/Research/infrastructure-as-a-service_iaas_clouds_are_local_and_so/q/id/55304/t/2`.

[128] L. Walsh, “Hardware-as-a-service.” `http://mspondemand.blogspot.com/2009/09/hardware-as-service-vs-lease.html`, September 2009.

[129] “Microsoft software update services.” http://technet.microsoft.com/en-us/wsus/bb466190.

[130] G. Keizer, “Trojan horse poses as windows xp update.” http://www.informationweek.com/news/17300290, 2004.

[131] D. Berlind, “Why windows update desperately needs an update.” http://www.zdnet.com/news/why-windows-update-desperately-needs-an-update/299080, August 2003.

[132] J.-T. Seo, Y.-j. Kim, E.-K. Park, S.-w. Lee, T. Shon, and J. Moon, "Design and implementation of a patch management system to remove security vulnerability in multi-platforms," in *Fuzzy Systems and Knowledge Discovery* (W. Lipo, J. Licheng, S. Guanming, L. Xue, and L. Jing, eds.), vol. 4223 of *Lecture Notes in Computer Science*, pp. 716–724, Springer Berlin / Heidelberg, 2006. 10.1007/11881599-87.

[133] D. White and B. Irwin, "A unified patch management architecture," tech. rep., 2004.

[134] H. Tian, L. Huang, J. Shan, and G. Chen, "Automated vulnerability management through web services," in *Grid and Cooperative Computing* (M. Li, X.-H. Sun, Q. Deng, and J. Ni, eds.), vol. 3032 of *Lecture Notes in Computer Science*, pp. 1067–1070, Springer Berlin / Heidelberg, 2004. 10.1007/978-3-540-24679-4-182.

[135] C.-W. Chang, D.-R. Tsai, and J.-M. Tsai, "A cross-site patch management model and architecture design for large scale heterogeneous environment," in *Security Technology, 2005. CCST '05. 39th Annual 2005 International Carnahan Conference on*, pp. 41 – 46, 2005.

[136] D. White and B. Irwin, "A unified architecture for automatic software updates," *Proceedings of Information Security South Africa 2004*, 2004.

[137] M. Nicolett and R. Colville, "Robust patch management requires specific capabilities," *Research Note T-19-4570, Gartner*, 2003.

[138] E. Mills, "Red hat, fedora servers compromised." http://news.cnet.com/8301-1009-3-10023565-83.html, August 2008.

[139] J. Antman, "Patch management: An overview." `http://rutgerswork.jasonantman.com/antman-patchManagement.pdf`, December 2008.

[140] D. Kaminsky, "Black ops 2008: It's the end of the cache as we know it," *Black Hat USA*, 2008.

[141] F. Amato, "Infobyte security research (isr)-evilgrade." `http://www.infobyte.com.ar/down/isr-evilgrade-Readme.txt`, 2008.

[142] P. Mell and M. Tracy, "Procedures for handling security patches," *NIST Special Publication*, vol. 800, p. 40, 2002.

[143] A. Gehani, B. Baig, S. Mahmood, D. Tariq, and F. Zaffar, "Fine-grained tracking of grid infections," in *GRID*, pp. 73–80, 2010.

[144] B. W. Boehm, "Software engineering economics." `http://userfs.cec.wustl.edu/~cse528/Boehm-SE-Economics.pdf`, 1984.

[145] S. S. G. D. P. B. Y. N. P. F. D. K. C. Z. Steve Graham, Doug Davis, *Building Web Services with Java: Making Sense of XML, SOAP, WSDL, and UDDI*. Sams Publishing, 2005.

[146] "Extensible markup language (xml)–http://www.w3.org/xml/." `http://www.w3.org/XML/`.

[147] G. M. S. W. Erik Christensen, Francisco Curbera, "Web services description language (wsdl)." `http://www.w3.org/TR/wsdl`, 2001.

[148] "Simple object access protocol (soap)." `http://www.w3.org/TR/soap/`, 2007.

[149] I. Foster, C. Kesselman, J. Nick, and S. Tuecke, "Grid services for distributed system integration," *Computer*, vol. 35, pp. 37 –46, jun 2002.

[150] "Teragrid: https://www.teragrid.org/."

[151] R. Alfieri, R. Cecchini, V. Ciaschini, L. Dell'Agnello, A. Frohner, A. Gianoli, K. Lorentey, and F. Spataro, "VOMS, an authorization system for virtual organizations," in *Grid Computing*, pp. 33–40, Springer, 2004.

[152] "Cern virtual marchine." `http://cernvm.cern.ch/portal/`.

[153] "Lhc computing grid project– http://www.cern.ch/lcg/." `http://www.cern.ch/lcg/`.

[154] "rpath rbuilder tools for virtual appliance development." `http://docs.rpath.com`.

[155] "A toroidal lhc apparatus (atlas)." `http://public.web.cern.ch/public/en/lhc/ATLAS-en.html`.

[156] "Large hadron collider beauty (lhcb)– http://public.web.cern.ch/public/en/lhc/lhcb-en.html." `http://public.web.cern.ch/public/en/lhc/LHCb-en.html`.

[157] "Compact muon solenoid (cms)." `http://public.web.cern.ch/public/en/lhc/CMS-en.html`.

[158] "A large ion collider experiment (alice)." `http://aliceinfo.cern.ch/`.

[159] "The vobox service– http://service-vobox.web.cern.ch/service-vobox/." `http://service-vobox.web.cern.ch/service-VOBOX/`, 2009.

[160] "The gsi-enabled ssh." `http://grid.ncsa.illinois.edu/ssh/`.

[161] "Software vulnerability group– http://www.egi.eu/about/policy/groups/software-vulnerability-group-svg.html."

[162] R. Sinnott, *Grid Security, in Grid Computing: Infrastructure, Service, and Applications.* CRC Press, Taylor and Francis Group LLC, U.S.A., 2009.

[163] J. Muhammad and R. Sinnott, "Deployment of grids through integrated configuration management," Parallel and Distributed Computing and Networks(PDCN 2008), IASTED, 2008.

[164] J. Muhammad and R. Sinnott, "Policy-driven patch management for distributed environments," in *Third International Conference on Network and System Security*, pp. 158–163, IEEE, 2009.

[165] D. W. B. S. M. C. W. H. H. D. E. Mann, "The development of a common enumeration of vulnerabilities and exposures." Workshop Proceedings, September 1999.

[166] L. Kanies, "Puppet: Next-generation configuration management," vol. 31, pp. ??–??, Feb. 2006.

[167] "Enabling grids for e-science (eege)." http://www.eu-egee.org/.

[168] M. Prochazka and R. Wartel, "Pakiti: A patching status monitoring tool." `http://pos.sissa.it/archive/conferences/133/031/ISGC%202011%20&%20OGF%2031_031.pdf`, 2009.

[169] "Egee operational security coordination team." `http://osct.web.cern.ch/osct/`.

[170] "Rpm package manager–http://rpm.org/." `http://rpm.org/`.

[171] "Yellowdog updater modified–http://yum.baseurl.org." `http://yum.baseurl.org/`.

[172] M. Burgess, "Cfengine knowledge management–cfengine technology insight," tech. rep., 2009.

[173] A. Tsalolikhin, "The state of open source system automation." http://www.linux-mag.com/id/7841/, August 2010.

[174] "Open ssh." http://www.openssh.com/.

[175] "Advanced packaging tool (apt)." https://help.ubuntu.com/8.04/serverguide/C/apt-get.html, 2011.

[176] "Using the update agent (up2date)." http://www.redhat.com/advice/tips/up2date.html, 2003.

[177] P. Stirparo, M. Shibli, and S. Muftic, "Vulnerability analysis and patches management using secure mobile agents," in *Advanced Communication Technology, 2009. ICACT 2009. 11th International Conference on*, vol. 02, pp. 1054 –1058, 2009.

[178] "Microsoft sms." http://www.microsoft.com/smserver/default.mspx.

[179] "Microsoft baseline security analyzer." `http://technet.microsoft.com/en-us/security/cc184924`.

[180] Z. B. S. Acs, M. Kozlovszky, "Automation of security analysis for service grid systems." Proceedings of the First International Conference on Parallel, Distributed and Grid Computing for Engineering", Civil-Comp Press, Stirlingshire, UK, 2009.

[181] W. Eazel, "Nist guidelines not adequate, warns gartner." http://www.scmagazineus.com/nist-guidelines-not-adequate-warns-gartner/article/33102/, February 2006.

[182] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli, "Proposed nist standard for role-based access control," vol. 4, no. 3, pp. 224–274, 2001.

[183] S. R. Gysin, A. D. Petrov, P. Charrue, P. Gajewski, V. Kain, K. Kostro, G. Kruk, S. T. Page, and M. P. Pery, "Role-based access control for the accelerator control system at cern," in *ICALEPCS07*, (Knoxville, Tennessee, USA), pp. 90–92–, 2007.

[184] X. Ou, 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*, SSYM'05, (Berkeley, CA, USA), pp. 8–8, USENIX Association, 2005.

[185] S. Abiteboul and S. Grumbach, "A rule-based language with functions and sets," *ACM Trans. Database Syst.*, vol. 16, pp. 1–30, Mar. 1991.

[186] M. Becker and P. Sewell, "Cassandra: flexible trust management, applied to electronic health records," in *Computer Security Foundations Workshop, 2004. Proceedings. 17th IEEE*, pp. 139 – 154, june 2004.

[187] L. Zhang, G.-J. Ahn, and B.-T. Chu, "A rule-based framework for role-based delegation and revocation," vol. 6, no. 3, pp. 404–441–, 2003.

[188] O. Ajayi, R. Sinnott, and A. Stell, "Dynamic trust negotiation for flexible e-health collaborations," in *Proceedings of the 15th ACM Mardi Gras conference: From light-weight mash-ups to lambda grids: Understanding the spectrum of distributed com-*

puting requirements, applications, tools, infrastructures, interoperability, and the incremental adoption of key capabilities, MG '08, (New York, NY, USA), pp. 8:1–8:7, ACM, 2008.

[189] "Grids and basic research programmes," consesus report, Organisation for Economic Co-operation and Development Global Science Forum, September 2005.

[190] Z. B. S. Acs, M. Kozlovszky, "Automation of security analysis for service grid systems," in *Proceedings of the First International Conference on Parallel, Distributed and Grid Computing for Engineering*, 2009.

[191] H. Tian, L. Huang, Z. Zhou, and Y. Luo, "Arm up administrators: Automated vulnerability management," in *7th International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN'04)*, pp. 587–593, IEEE Computer Society, 2004.

[192] "Ibm internet security systems." `http://www.iss.net/`.

[193] "The spam-gp project." `http://www.nesc.ac.uk/hub/projects/spam-gp`.

[194] "The national e-infrastructure service (nes)–http://www.ngs.ac.uk/."

[195] "The open science grid (osg)–https://www.opensciencegrid.org/."

[196] "Scotgrid enabling scientific collaboration–http://www.scotgrid.ac.uk/."

[197] S. Fenz and A. Ekelhart, "Formalizing information security knowledge," in *Proceedings of the 4th International Symposium on Information, Computer, and Communications Security*, ASIACCS '09, (New York, NY, USA), pp. 183–194, ACM, 2009.

[198] J. Feng, G. Wasson, and M. Humphrey, "Resource usage policy expression and enforcement in grid computing," in *Proceedings of the 8th IEEE/ACM International Conference on Grid Computing*, GRID '07, (Washington, DC, USA), pp. 66–73, IEEE Computer Society, 2007.

[199] "Grids and basic research programmes," tech. rep., Organisation for Economic Co-operation and Development, Global Science Forum, 2005.

[200] "Core cloudinspect." `https://www.corecloudinspect.com/microsite/index.html`, Aug. 2011.

[201] S. Ghosh, "Mcafee getsusp portable cloud scanner." `http://www.insightsintechnology.com/2012/05/mcafee-getsusp-is-portable-cloud.html`, May 2012.