A HOST-BASED SECURITY ASSESSMENT ARCHITECTURE FOR EFFECTIVE LEVERAGING OF SHARED KNOWLEDGE

by

ABHISHEK RAKSHIT

B.E., Bharati Vidyapeeth Deemed University, India, 2006

A REPORT

submitted in partial fulfillment of the
requirements for the degree

MASTER OF SCIENCE

Department of Computing and Information Sciences
College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas
2009

Approved by:

Major Professor
Dr. Xinming Ou

Abstract

Security scanning performed on computer systems is an important step to identify and assess potential vulnerabilities in an enterprise network, before they are exploited by malicious intruders. An effective vulnerability assessment architecture should assimilate knowledge from multiple security knowledge sources to discover all the security problems present on a host. Legitimate concerns arise since host-based security scanners typically need to run at administrative privileges, and takes input from external knowledge sources for the analysis. Intentionally or otherwise, ill-formed input may compromise the scanner and the whole system if the scanner is susceptible to, or carries one or more vulnerability itself. It is not easy to incorporate new security analysis tools and/or various security knowlege-bases in the conventional approach, since this would entail installing new agents on every host in the enterprise network. This report presents an architecture where a host-based security scanner's code base can be minimized to an extent where its correctness can be verified by adequate vetting. At the same time, the architecture also allows for leveraging third-party security knowledge more efficiently and makes it easier to incorporate new security tools. In our work, we implemented the scanning architecture in the context of an enterprise-level security analyzer. The analyzer finds security vulnerabilities present on a host according to the third-party security knowledge specified in Open Vulnerability Assessment Language(OVAL). We empirically show that the proposed architecture is potent in its ability to comprehensively leverage third-party security knowledge, and is flexible to support various higher-level security analysis.

Table of Contents

List of Figures

List of Tables

Acknowledgments

All through my research and my graduate program many people have played instrumental role in my success and achievements. I wish to show my gratitude to all of them.

Dr. Xinming Ou, assistant professor with the department of Computing and Information Sciences at Kansas State University and major adviser for my report, has always helped me to look besides the obvious. I wish to acknowledge his contribution in the work presented in this report, and thank him for guiding me in times of failure and success.

Dr. Gurdip Singh, professor with the department of Computing and Information Sciences at Kansas State University and member of my supervisory committee, has taught me the nuances of computer networks, which helped me greatly in my research. I wish to thank him for his guidance and support all through my graduate program.

I am also thankful to Dr. Mitchell L. Neilsen, associate professor with the department of Computing and Information Sciences at Kansas State University and member of my supervisory committee, for his valuable advices and time to enhance my work.

Last but not least, I wish to thank my friends Avinash, Sinu and Vikas for their invaluable inputs and support.

Dedication

To,

my father Mr. Rupendra Rakshit, my strength.

my mother Mrs. Shikha Rakshit, my spirit.

my sister Noopur, my inspiration.

Chapter 1

Introduction

With rapid growth in both the number and sophistication of cyber attacks, it has become imperative that cyber defenders be equipped with highly effective tools that identify security vulnerabilities before they are exploited. A vulnerability can be defined as a set of conditions which if true, can leave a system open for intrusion, unauthorized access, denied availability of services running on the system or in any way violate the security policies of the system [2].

While breaches happen at every corner of an enterprise network, often the security of the end hosts is the most brittle line of defense. A breach of security occurs when a stated organizational policy or legal requirement regarding information security, has been contravened. However every incident which suggests that the confidentiality, integrity and availability of the information has been inappropriately changed, can be considered a "security incident". Every security breach is always initiated via a security incident, only if confirmed does it become a security breach.[3]

The rise in security breaches is alarming. According to the Computer Crime and Security Survey 2007(CSI) [4]; there is an increase of 29% in the number of organizations reporting security incidents. The need for approaches that can secure end hosts more effectively is imperative. "Vulnerability assessment scanners" are one such advancement. Vulnerability Assessment(VA) scanners are tools that scan a host system or an enterprise network to check for the presence of security vulnerabilities. The term host/target used throughout, refers to the machine which is being scanned for vulnerabilities. VA scanners can be broadly

classified into two categories.

1. Network Based

2. Host Based

A network-based scanner (*e.g.* Nessus [5]) probes a machine remotely to find vulnerabilities. A host-based scanner on the other hand is installed on the host system itself. The host-based scanners have an edge over their counterpart network scanners as they directly access the configuration information of the host and various services running on the same. However, conventional host-based scanners require regular installation/updates for the agents installed on every machine on the enterprise network. This becomes a daunting task as the network scales up to incorporate a large number of systems, when compared against the network-based scanners. Incidents when the agents even crashed the host machine during scanning have also been reported.

The conventional host-based scanners send the security knowledge to the target host. The term "security knowledge" used throughout the report refers to the definitions, which determine the conditions for a known vulnerability to be true or false on a target/host system. The scanner gathers the host's configuration information and performs various types of analysis based on the received security knowledge. Since the knowledge(which might be provided by a third party) is consumed on the end hosts, any security vulnerability in the agent could render the end host vulnerable. This has become an obstacle in convincing a system administrator to experiment with a new, less well-known security scanner. Specifically the administrators are skeptical to trust the agent code (with tens or even hundreds of lines of code), not to harm the host itself. Moreover, consuming the knowledge on each end host introduces a great amount of replicated effort in leveraging the security knowledge, and makes it hard to combine knowledge from various sources and conduct a global security analysis at the enterprise level.

We propose an architecture for host-based security scanning, which not only improves the host-based scanners by overcoming the shortcomings mentioned above, but also enables

incorporation of security knowledge from various sources and use it efficiently to provide a comprehensive security analysis of the enterprise network. We accomplish this by separating the process of gathering host configuration information and the analysis of the same. The proposed system is designed as a bi-component architecture for host-based vulnerability scanning. First component is the scanner (or the agent as we often call it) that needs to be installed on the host. The *agent* serves the sole purpose of gathering information from the host and does not perform any analysis. The second component is the *analyzer* which resides on a server or a cluster of servers depending on the size of the network. The sole responsibility of the analyzer(or the manager) is to perform security analysis on the information. The analyzer gathers security knowledge from various sources and co-relates it with the information furnished by the agents in the enterprise network. It then produces a comprehensive security report for every host on the network, leveraging third-party security knowledge efficiently. This comprehensive report can become the basis for network administrator's decisions to safeguard the host (and the network) against reported vulnerabilities, or can be the input to a higher-level global security analysis tool.

The proposed architecture has the following advantages over the conventional design.

- To begin with, it remarkably reduces the size of the code-base on the end host. This makes the installation and configuration of the agent easy and the agent is less likely to disrupt services running on the host. In addition, code-base being small reduces the likelihood of introduction of programming flaws/bugs in it.

- The small code base makes it possible for the administrator(or some other trusted party) to check for flaws and malicious content, restoring the trust in the code as well.

- Any change or addition in the security knowledge only needs to be made available to the analyzer running on a centralized location, unlike conventional scanners where these changes/additions have to be made available to every agent residing on the hosts in the network.

- This architecture also supports other high level security analysis on the data collected from all the hosts on the network.

Our research is conducted within the context of the MulVAL(Multi-host, multi-stage Vulnerability Analysis) [6] project. MulVAL is an enterprise-level security analyzer that can automatically compute all possible multi-step, multi-host attack paths in an enterprise network based on security vulnerabilities discovered on end hosts. The input to MulVAL is the result of host-based vulnerability assessment performed on each and every managed machine in the enterprise network. The original MulVAL work used an adapted OVAL (Open Vulnerability Assessment Language) [7] interpreter released by the MITRE corporation to analyze each end host. The external security knowledge is specified in the OVAL language and needs to be sent to all the end hosts. When we tried to deploy the MulVAL tool on some enterprise networks, the first concern from the system administrators was always the trustworthiness of the OVAL interpreter. The current release of MITRE's OVAL reference interpreter (with limited capability) contains around 35,000 lines of code. The OVAL repository, which is a third-party security knowledgebase, has 1615 [1] entries for the Windows platform and the size of the XML file input to the interpreter is 6.2MB. It is a legitimate request that the system administrators be convinced that an application of this size will not adversely affect servers and workstations in any way. This motivated us to develop a scanning architecture, where the scanner that needs to be run on a host has a minimized code base, and thus its correctness can be verified through adequate code vetting. At the same time, our architecture also allows for more efficient leveraging of third-party security knowledge at an enterprise level. This is possible because the architecture allows a knowledgebase maintained on the Analyzer to be a compilation of security knowledge from many different sources.

Our goal is to design an architecture for host-based security scanning, which enables more efficient and flexible usage of external knowledge in enterprise network security management,

[1] as on May 6th'08

and supports a range of enterprise-level security analysis based on information provided by host-based security scanning. We focus on building an agent that runs on a host which is minimized to tens of lines of code, to support thorough code vetting. We have implemented an OVAL vulnerability scanner based on the proposed architecture, and have empirically demonstrated the advantages enlisted above.

The report is organized as follows: Chapter 2 provides a thorough information about vulnerability analyzers and their types. A more insightful view of our proposed architecture and how it effectively counters the issues in the conventional architecture is provided in Chapter 3. Chapter 4 describes the technical details of implementation of our assessment tool. Chapter 5 shows the results obtained from our security analyzer. A discussion of the future and related work can be found in Chapter 6. We conclude our work in Chapter 7

Chapter 2

Background

With the rapid development of more complex systems, the chance of introduction of errors, faults and failures increases in many stages of software development life-cycle[8]. This class of system failures is commonly termed as "software vulnerabilities". These security vulnerabilities violate security policies and can cause the system to be compromised leading to loss of information[9]. Vulnerabilities can be introduced in a host system in different ways; via errors in the code of installed software, mis-configurations of the software settings that leave systems less secure than they should be (improperly secured accounts, running of unneeded services, etc) [10].

Vulnerability analysis, also known as vulnerability assessment, is a process that defines, identifies, and classifies the vulnerabilities (security holes) in a computer, network, or communications infrastructure [11]. Vulnerability analysis can be used to predict the effectiveness of the proposed countermeasures and evaluate them after they are put into use. Vulnerability analysis begins with gathering, defining and classifying network or system resources. The resources can be classified according to their level of importance in the network. Next comes the identification of potential threats to those resources. This stage of identification of threats can be performed by probing the network or system to discover potential weak points.

A vulnerability assessment tool (or scanner) can be defined as a utility that can be used to test the capability of a systems or networks security and discover their points of

weakness[2]. These tools themselves do not provide any kind of security or protection to the system, rather they gather and report information, which can be used to instate a different tool, policy or mechanism to secure the system. Vulnerability assessment tools can be broadly classified into network based and host based analyzers as described in the following Sections 2.1 and 2.2 respectively.

2.1 Network Based Analyzers

Network based vulnerability assessment gathers information of the system and services attached to the network and, identifies weakness and vulnerabilities exploitable in the network. These vulnerabilities could be related to services, such as HTTP, FTP and SMTP protocol, running on the given network. A network-based scanning assessment may also detect extremely critical vulnerabilities such as mis-configured firewalls or vulnerable web servers in a De-Militarized Zone(DMZ), which could provide a security hole to an intruder, allowing them to compromise an organizations security[1]. Network assessment tools gather information and may also have network mapping and port scanning abilities.

A typical network based scanner architecture is shown in Figure 2.1. Generally, a network based scanner consists of a number of components. It has a vulnerability database which contains all the vulnerability definitions and information about how to detect these vulnerabilities. This database has to kept updated, for new vulnerabilities are discovered very frequently. In addition it also has an interactive console, which helps the administrator to schedule vulnerability assessments on different targets on the network. Furthermore the scanning engine is the main component of the network based scanner. It performs the assessment as instructed by the interactive console by sending specially constructed packets for the test. Results repository is the final component, which holds all the scan results received and is also used for report generation for the system administrators [12]. The strengths of network based scanners lie in the fields described below:

1. Network scanners provide a comprehensive view of all the services running on the

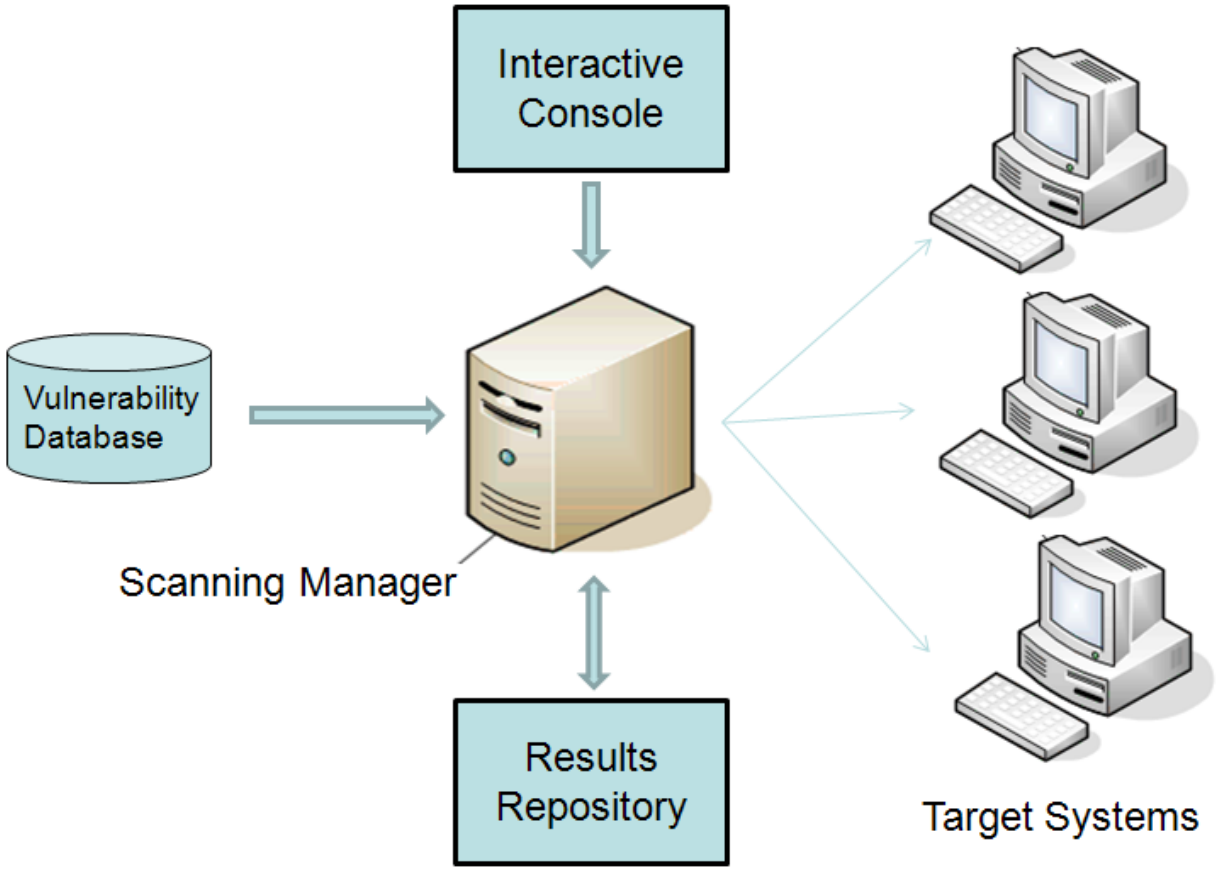

Figure 2.1: *Architecture of a typical Network-Based Analyzer [1]*

network. It can also discover unknown devices on the network and determine if your network has unknown perimeters.

2. Network based scanners are inherently non-intrusive to the systems under scan. This is due to the fact that there is no need of any installation of agents on the target hosts. Also, because of the same reason network scanners can be setup quickly without any deployment or lot of planning.

3. There are many vulnerabilities which are analyzed more effectively by a network scan. For instance vulnerable services or daemons running on operating systems and low level protocol weakness. Moreover, advanced network attacks like protocol spoofing can be tested only by a network scan.

4. Devices which do not support host-based scanning like the routers, switches, remote access servers and firewalls, can only be scanned by network based scanners.

However network-based scanners also have the following demerits:-

1. A network-based scanner does not have direct access to the target's file system. Thus, it is not able to check for file permission. Vulnerabilities can be present in form of rogue programs with incorrect "SetUID" and "SetGID" bits enabled.

2. Another problem which network based scanners face is their inability to scan targets behind a firewall. Complicated measures have be taken to let scanners get to these target systems.

3. Network based scanners may also face the problem of network congestion due to a lot to-and-fro data transfer.

2.2 Host-Based Analyzers

Host based analyzers also scan the system for vulnerabilities like the network scanners, however they are able to scan much more due to the fact that they have a service/agent residing on the target system. They can easily identify system-level vulnerabilities such as file permissions, user account properties and registry settings. A typical host-based architecture is shown in Figure 2.2. The host-based analyzer is installed on a network by first installing a scanning manager on the network. Agents are then installed on all the target systems to be scanned. The working of these agents is controlled by the manager. In some systems, there may be a separate user console to interact with the scanning manager, which is merged with the manager otherwise. When the manager wants to initiate a scan, it sends the necessary information like scanning policy to the agent on the host. The scanning policy consists of the different vulnerability checks. The agent on the host scans accordingly and reports back the results of the scan. As new vulnerabilities are discovered frequently the security definitions have to be regularly updated on each agent.

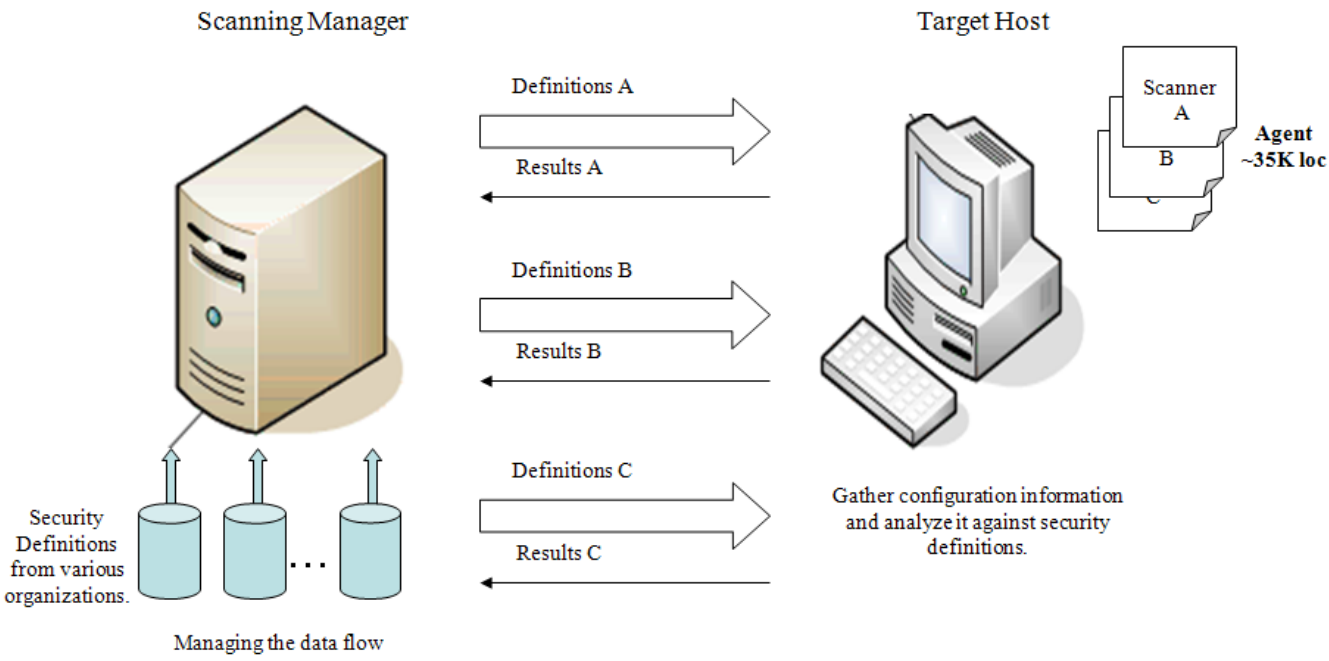

Figure 2.2: *Architecture of Conventional Host-Based Analyzer*

The main strength of host-based scanners lies in the fact that they have direct access to configuration details and services of the target system. The merits of host-based scanners are described below.

1. Host based scanners are adept at checking for malicious user behavior that violates the security policies. They can an check for behavior like using weak passwords or sharing hard drives on the network.

2. Host based scanners can detect devices which can initiate unauthorized remote access servers (e.g modem). They can also check for "remote control applications" which can be used to access resources from unknown network perimeter.

3. A big advantage that host-based scanners have over the network-based scanners is that they run the checks on the target system locally and thus network traffic is reduced considerably.

4. These scanners can detect if the system has already been infiltrated by intruders by suspicious filenames, program files found in unexpected places, unexpected SUID/SGID privileged programs, sniffer programs etc.

5. They are ideal for performing resource intensive full system scans which are not possible through the network.

However, host-based scanners do not come without weaknesses. They are described below:-

1. The scanning agents have to be installed on new systems and updated regularly on the old ones to keep the assessment fool-proof.

2. System administrators are reluctant to install unknown agents on the production systems.

3. Agents residing on the target host utilize its resources, sometimes interfering in the normal functioning of the host.

4. As the enterprise network increases in size, managing and updating agents on all the hosts becomes an issue.

This chapter describes the advantages and disadvantages of both network and host-based systems and so most of the state of art system employ both the network and host-based scanners for vulnerability assessment. In our work, we have addressed the problems faced by the host-based scanners by proposing a new architecture for the same. Chapter 3 is devoted to the complete description of the new architecture and how it rectifies the weaknesses described above.

Chapter 3

Proposed Architecture

The motivation for our research is, the need for an architecture that will provide an effective way for leveraging third-party knowledge and provide a way for global analysis and a comprehensive view of Enterprise Security. This chapter will describe the high level details of our architecture and its strong points.

3.1 Centralized Host Based Architecture

As described in Chapter 2, in conventional host-based scanner, the scanning manager looks after two information flows; sending security definitions to each end host, and getting the report back after the analysis is performed. The centralized architecture takes a totally different approach when it comes to the use security knowledge for vulnerability analysis. As the name suggests the architecture has a centralized approach when analyzing a host with regards to the security definitions from the knowledge base.

Security definitions are different from virus definitions used in virus scanners. Virus signatures typically refer to file name and contents, whereas conditions to determine the existence of a security vulnerability i.e. security definitions refer to configuration parameters with arbitrary logical relationships. For example, the OVAL [7] language for the Windows platform could be used to specify conditions on any Windows registry entry, on any file's attributes or on any process running on a machine, etc. It also has a number of logical connectors and attributes which can specify the full propositional relations as well as limited

first-order logic semantics. Thus checking the existence of a security vulnerability is more "ad-hoc" than checking the existence of malware using virus definitions. Moreover, a host-based security scanner often has a large code-base, due to the need to support various kinds of checking and analysis tasks. For example, the code size of MITRE's reference implementation of an OVAL scanner [13] developed in C++ programming language has, around 35,000 lines of code. One can imagine than an application of this size, makes it very hard for the developers to keep it totally flawless, and it is hard if not impossible, to verify that there is no security vulnerabilities in the same.

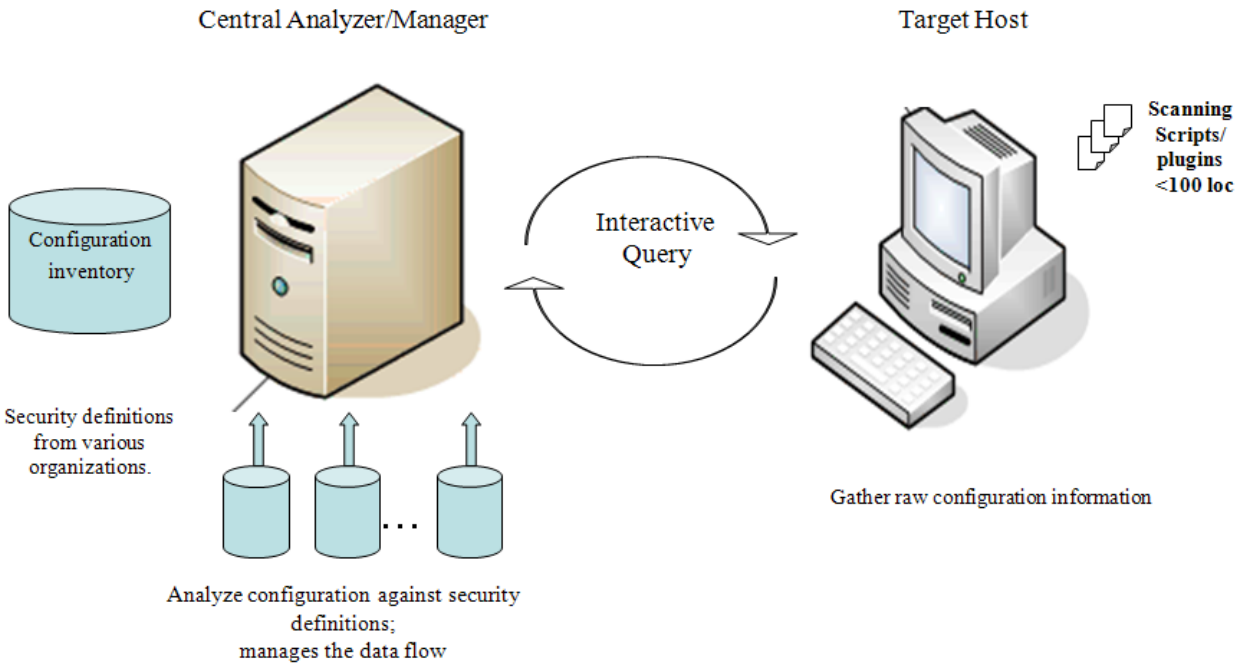

Figure 3.1: *Proposed Architecture*

Figure 3.1 shows that the proposed architecture consists of central analyzer, a configuration inventory and a very stripped-down agent on the target host. The configuration inventory holds all the configuration information obtained from the target systems on the enterprise network. Unlike the conventional architecture, the new architecture employs an agent which performs few basic functionalities with regards to the configuration gathering, such as dumping the whole Windows registry, querying for a file's attribute and getting

a process's status. Most of these functionalities have readily available system commands or application programming interfaces(APIs) and thus the amount of code needed to accomplish this is minimal. Our agent consists of a shell and a visual basic script. It is less than hundred lines of code therefore significantly reducing the possibilities of programming flaws/bugs in the code. The small code base can even be rigorously vetted for any kind of flaws therefore increasing the trustworthiness of the agent. These light weight scripts also overcome the problem of resource consumption on the host by adding no significant burden on the hosts resources. The agent in the new architecture reports end-host configuration information to the central analyzer for the analysis.

The Central Analyzer is the most important part of the architecture. Its functionalities as a *scanning manager* include scheduling the scan for all the target hosts on the network, managing the configuration inventory and storing all the information obtained from the target hosts in the inventory. As the *analyzer* it has to first gather, convert and store security knowledge from various sources in datalog format acceptable the logic-based comparator. Secondly, it has to initiate the logic-based comparator by providing it with both the configuration information of the target as well as the security knowledge to get the results of the vulnerability analysis for the respective host. The purpose of the analyzer is similar to the central manager in terms of conventional architecture, alongside the added functionality of performing vulnerability analysis based on updated security definitions from various sources and maintaining a configuration inventory of all the hosts in the enterprise network.

3.2 Advantages of Centralized Architecture

A significant advantage that the new architecture brings over the conventional architecture is that the central analyzer has a complete view of the configuration of every managed host in the enterprise network, and can conduct various high level security analysis on the configuration information.

Moreover, since all hosts' configuration is centrally stored at one place, the application of

security knowledge is more efficient than conventional host-based scanners. This is because we have to process the raw security knowledge from multiple sources just once and it is used for all the machines on the network. This cost of pre-processing of the knowledge is amortized over all the machines under scan. In our implementation, we compile the knowledge into executable code, so that the compilation is done only once and the code can be directly applied to all the hosts' configuration data to efficiently perform the analysis. The proposed architecture suits especially well to the recent trend of sharing security knowledge in an open and standard format. Apart from aiding to a more wider and comprehensive analysis for security vulnerabilities, our architecture also paves a platform for more diverse research and development in the area of Information Security.

Open Vulnerability Assessment Language (OVAL) is one such effort to enable a "community approach" to security management of enterprise systems. The effort has echoed well in the IT security management industry, with vendors like GFI LANguard [14] and SofCheck [15] which already "OVAL-compatible" products. This is a significant departure from the conventional business model where the vendors of vulnerability assessment tools provide security definitions in their own proprietary format. With the rapid growth in cyber security threats, it is evident that no single organization can provide a holistic solution to all security problems faced by the enterprise network systems. The ability to share security knowledge efficiently is the key to win the "Cyber War" against the Internet miscreants.

The proposed security-scanning architecture presented in this report facilitates knowledge sharing since it separates the two distinct phases in security analysis, configuration information gathering and vulnerability analysis. In the conventional architecture, the two phases are merged together in a single agent. As a result, addition of new knowledge inevitably entails installing/updating new agents on every machine. The conventional architecture pushes the knowledge to the end host, not only duplicating data and using a lot of network resources, but also increasing the resource consumption on the host systems which can cause interruptions in the functioning of the host. In the architecture presented in this

report, the two phases are implemented as two separate components. If the new knowledge or analysis tools need the same set of configuration information from the end hosts, there is no need to update the agent at all. This separated architecture avoids re-inventing the wheels in security analysis, facilitates knowledge sharing, and thus, maximize the benefit from all efforts involved in security management. The fact that our architecture holds configuration inventory also opens up avenues for its diverse usage in fields like "Enterprise Inventory Management" and "Troubleshooting".

To summarize, the strong points of our architecture are listed below:-

1. It provides a platform for higher level Global Security Research and Analysis.

2. The architecture effectively leverages shared knowledge from multiple sources providing a comprehensive vulnerability report.

3. Minimal code base on the target host allows thorough code vetting, making the code reliable and hence making the agent acceptable to the system administrators.

4. The agents are less resource intensive and a small code base also ensures easy installation and management of the agents.

In this report we have implemented an OVAL vulnerability scanner based on the architecture described above. Chapter 4 provides the details of the same.

Chapter 4

Implementation of an OVAL scanner based on the new architecture

This chapter begins with shedding some light on the security knowledge base which is a very significant part of the architecture. Our host-based VA architecture is discussed in the section 4.2.

4.1 The OVAL Language

Open Vulnerability and Assessment Language (OVAL) [7] is an international, information security, community standard to promote open and publicly available security content, and to standardize the transfer of this information across the entire spectrum of security tools and services. OVAL includes a language used to encode system details, and an assortment of content repositories held throughout the community. The language standardizes the three main steps of the assessment process: representing configuration information of systems for testing; analyzing the system for the presence of the specified machine state (vulnerability, configuration, patch state, etc.); and reporting the results of this assessment. It is an XML-based language and specifies vulnerable machine configuration for almost all types of platforms such as Windows, Linux, HP-UX, Cisco IOS and Sun Solaris.

There are many security knowledge bases such as the Common Vulnerabilities and Exposures (CVE) [16] by MITRE, the National Vulnerability Database(NVD) [17] by National

Institute of Standards and Technology, etc. The OVAL repository is a collection of security knowledge conforming to CVE standards which deals with configuration checking. Configuration checking deals with analyzing low level configuration details of a computer system. It consists of collecting information about all the installed softwares and the services running on that system. The definitions provided by OVAL stipulate the conditions that, when satisfied by the host, confirm the presence(or absence) of vulnerabilities on the same.

```
Title: Microsoft Outlook Advanced Find Vulnerability
Definition Id: oval:org.mitre.oval:def:153
CVE ID: 2007-0034

Definition Synopsis:
   Outlook 2000
      Outlook 2000 is installed
      AND the version Outllib.dll is less than 9.0.0.8954
   OR Outlook 2002
      Outlook 2002 is installed
      AND the version of Outllib.dll is less than 10.0.6822.0
   OR Outlook 2003
      Outlook 2003 is installed
      AND the version of Outllib.dll is greater than 11.0.8118.0
```

Figure 4.1: *OVAL vulnerability definition example*

A natural-language description of an OVAL security definition, for the "buffer overflow" vulnerability in the "Advanced Search (Finder.exe)" (Microsoft Outlook 2000, 2002, and 2003) is shown in Figure 4.1. The XML definition for the same, can be found in Appendix A. Even though the OVAL definition provides a comprehensive detail about the vulnerability but, in the definition presented in Appendix A we have only shown the information needed by our analyzer without any loss of generality. The vulnerability definition has information about MS outlook installation on the host and the file version number for the respective "Outlib.dll" file. For each versions of Outlook the definition specifies conditions in separate test criterions. The test criteria have logical conditional "and/or" constraints. All criterions related with an "and" clause should be true for the particular vulnerability to be valid and one or more true criterions in a group of criterions coupled with an "or" clause will render the vulnerability true. That said, it is evident in the example shown in Figure 4.1 that

the host will have the corresponding "buffer overflow" vulnerability, if one or more of the criteria mentioned hold true since all three of them are woven into a single "or" clause.

Moreover, in some cases validity of a definition can be a prerequisite for other definitions. For example, the definition to determine a particular operating system platform on a host will be a prerequisite for any definition which checks for installation of a particular software for that operating system platform. [1].

4.2 Architecture

As described in Chapter 3 the architecture is divided into a data collection part and the analysis part. The data collection part is done by the host resident agent described in the section 4.2.1 and the analysis is done by the analyzer described in the section 4.2.2.

4.2.1 Configuration data collection

The agent consists of a set of scripts, and resides on the host machine. The agent is dedicated to gather information from the target/host. The information needed by the OVAL definitions is gathered from the registry dump, and the version numbers for specific files, obtained by checking file attributes of the same. In our implementation the residing agent collects registry entry information and file version numbers using windows system commands.

The host information has to be gathered on a periodic basis and is sent to the central analyzer for further processing. The analyzer can query the scanner for specific information if needed. This may bring up the question of high network usage, but this problem can be resolved by querying a small set of machines at one time, as opposed to querying the entire network. The amount of data that needs to be transferred from a single host for a complete OVAL checking is less than 38 megabytes (MB) for a machine running Windows XP operating system and less than 26 megabytes for the Windows 2000 Server operating system.

[1]The complete documentation of the OVAL language can be found at *http://oval.mitre.org/*

As all the system configurations and other information resides on the central analyzer, it also helps in keeping a check on what changes have been made on the machine from the time of the previous scan. This information is valuable for security forensics, as well as can be used by system administrators to track down configuration changes on machines for troubleshooting.

More scripts can always be added to the agent, to get more information with respect to network, port and file system status without incurring a lot of addition to its code weight. This in turn opens up new avenues for contributors to participate in the development of the agent, to gather knowledge specific to other applications such as inventory tracking and troubleshooting [18]. It also supports the notion of effective sharing of knowledge.

The agent with the above described functionalities has minimal code weight. In the current implementation the agent consists of three files; two batch scripts to get the registry information and a visual basic script to get the file attributes. Altogether the agent is less than 30 lines of code. Thus, the agent in the architecture proposed in the report not only provides a trustworthy small code base which is a practical contender for thorough code vetting, it also uses minimal host resources causing no interruptions in the normal functioning of the host.

4.2.2 The Analyzer

The Analyzer is a centralized server connected to all the hosts on the enteprise network. It is responsible for performing a comprehensive security analysis of the hosts and reporting all the vulnerabilities present on them. The analyzer accomplishes this task by analyzing the information received from the agents (residing on the host), against the preprocessed security knowledge obtained from the information gathered from various security knowledge bases. The analyzer is further divided into two parts: *knowledge convertor* and *logical comparator*.

Knowledge Convertor

The knowledge convertor converts the updated security definitions(XML format) and raw host configuration(ascii text format) into datalog format. An example of host configuration information in datalog format which is taken as input by the comparator is shown below:-

```
win_Reg_Entry(Path,Name,Value)
```

In the example shown above, "Path" is the *hive* and *key* values from a registry entry of the host machine. The "Path+Name"/ "Value" pair is matched against the specified knowledge of the security definitions by the comparator. The value part is either a string or a numerical value. An example of a converted registry entry to datalog format is shown in the Figure 4.2.

```
win_Reg_Entry('hkey_current_user\software\microsoft','DefaultFormat','RadioQuality').

win_Reg_Entry('hkey_current_user\software\microsoft','CorpPC2Phone',dword('00000000')).
```

Figure 4.2: *Datalog format registry entry example*

The convertor's next task is to processing security knowledge. Figure 4.1 shows one of the definition from the collection of all the definition provided by OVAL for the analysis of systems running on windows operating systems. Appendix A shows a highlighted section containing a particular criteria block shown in Figure 4.3. Figure 4.4 shows the conversion of the criteria shown in Figure 4.3.

```
<criteria comment="Outlook 2000" operator="AND">
  <criterion negate="false" test_ref="oval:org.mitre.oval:tst:895"/>
  <criterion negate="false" test_ref="oval:org.mitre.oval:tst:162"/>
</criteria>
```

Figure 4.3: *Conditional criteria in XML format.*

Appendix B shows a highly stripped down version of the converted definition from the one shown in Appendix A.

```
criteria(unique_id1,'Outlook 2000'):-
   criterion('oval:org.mitre.oval:tst:162','false'),
   criterion('oval:org.mitre.oval:tst:895','false').
```

Figure 4.4: *Conditional criteria converted to datalog format.*

The OVAL repository is updated regularly and our architecture is well suited to adapt the constantly changing knowledge. Once we have both the host information and the security definitions in the required format, these are provided to the logical comparator for analysis.

Logical Comparator

The logical comparator accepts both the processed host information and the security knowledge and compares them using the "XSB" engine. XSB is a Logic Programming and Deductive database system for Unix and Windows. The comparator analyzes the information from the host with respect to the criterions from the security definitions to report all the vulnerabilities present on that host.

The analysis can be performed in a number of different ways. The reason for using a logic-based approach is that, datalog is both a declarative specification and an executable program. Thus, the security knowledge can be compiled into executables, during which significant optimization can be done to speed up the analysis process. This optimization cost is only paid once in converting the knowledge into datalog byte-code, and can be amortized over the repeated application of the knowledge to a large number of machines over a period of time. As mentioned in Chapter 1 our work is the underlying research for MulVAL [6] project, which is a datalog-based framework for modeling the interaction of software bugs with system and network configurations. It is thus, a natural choice for us to use the same logical language for individual host's vulnerability assessment. Moreover, the saving gained through amortizing knowledge preprocessing cost (compilation in this case) applies to other internal knowledge representation as well.

Chapter 5

Results

The primary goal of our work is to investigate the effectiveness of central host-based architecture described in the Chapter 4 when compared with the MITRE's reference scanner [13]. The results indicate that the centralized scanner is not only effective in vulnerability assessment but also is more efficient with respect to time taken to perform comprehensive analysis when compared with the reference scanner.

The test bed for our experiments consists of non trivial network consisting of three machines running *Microsoft Windows XP* operating system and two running *Microsoft Windows 2000 Server* operating system. The end hosts were left un-patched with vulnerabilities present for testing purposes. Table 5.1 shows the system architecture of the machines on the network. The analyzer is a dedicated Linux server with three dual opteron processors. The analyzer communicates with the host using the secure shell (SSH) protocol and the file transfer between the two takes place using the Session Control Protocol (SCP).

During the analysis, the files transmitted to the analyzer from the agent residing on the

Table 5.1: *Test Bed*

Architecture	Analyzer	Host	
Operating System	Linux	Win_XP	Win_2000
Processor	Dual Opteron 2.2ghz(x3)	2.2ghz AMD Opteron	2.2ghz AMD opteron
Memory(GB)	16	2	2

Table 5.2: *Host Data (megabytes)*

Files	Win_XP	Win_2000
Registry	38	26
Version Numbers	.01	.01

Table 5.3: *Vulnerability Performance Comparison*

Operating System	Reference scanner	Centralized Scanner
Win_XP	223	224
Win_2000	260	260

host, correspond to registry information and version numbers of the files specified in the security definitions for the host. Table 5.2 shows the average size of both the files from the respective host machines.

Table 5.3 compares the vulnerabilities detected by the reference scanner with those detected by our analyzer. As seen in the table, vulnerability detection capabilities of the central host-based analyzer are at par with the Mitre's reference implementation of OVAL scanner. Both the scanners reported 260 vulnerabilities for the host with Windows 2000 Server operating system. For hosts with Windows XP operating system, the reference scanner reported 223 vulnerabilities. The centralized scanner has reported 224 vulnerabilities, including one vulnerability that was missed by the reference scanner but manually verified to exist on the host.

The centralized scanner came better when compared against the time taken to analyze a host machine by the reference OVAL scanner. Table 5.4 and Figure 5.1 shows average time (in minutes) to complete the vulnerability analysis of the hosts with Windows 2000 Server and Windows XP operating system. A point worth mentioning here is that in the case of the reference scanner, the time reported reflects the time it executed on the host machine engaging the host's resources. On the contrary, in our scanner, only a fraction of the total analysis time shown is the time when resources on the host were used to get its

Table 5.4: *Analysis Time Comparison(minutes)*

Operating System	Reference scanner	Centralized Scanner
Win_XP	3:11	2:48
Win_2000	3:16	2:22

configuration information. Rest of the time reflects the analysis part which is performed by our dedicated analyzer.

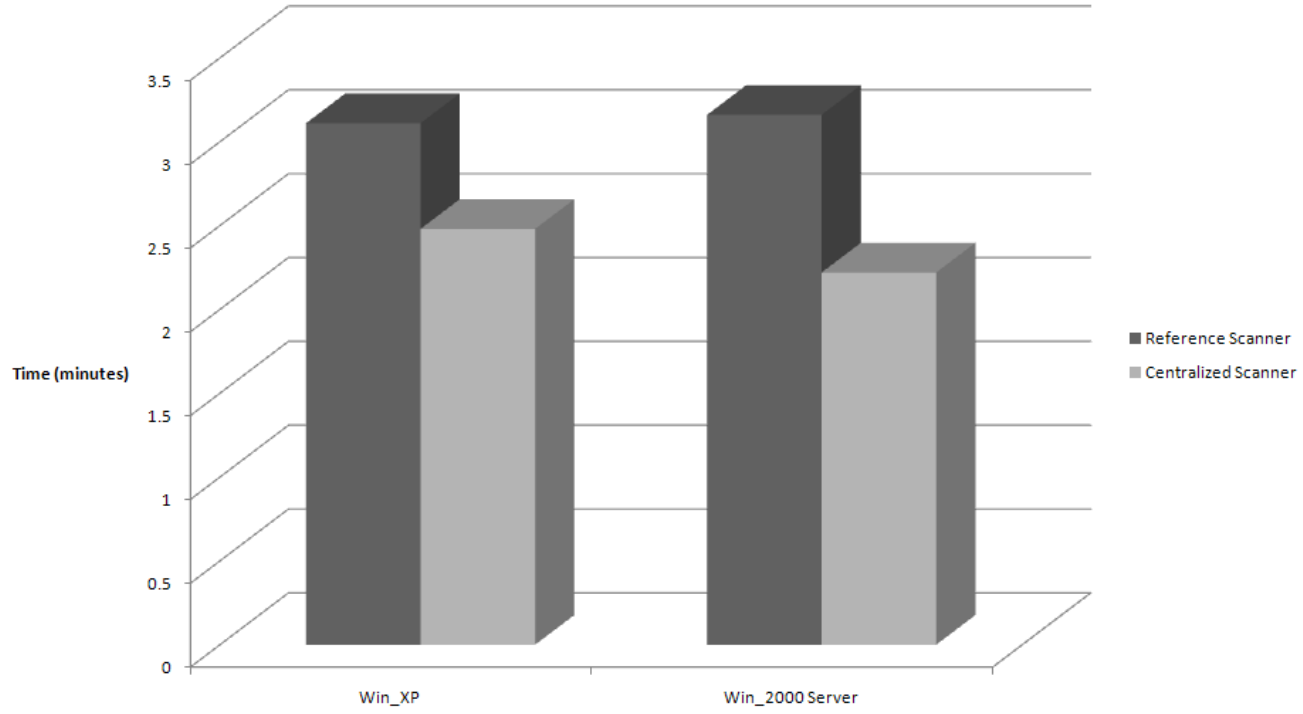

Figure 5.1: *Analysis Time Comparison*

The security definitions are updated regularly by the OVAL community. We pre-compile and store the converted OVAL Prolog bytecode whenever a new OVAL definition file is released. The compilation (with optimization) for an OVAL Windows definition takes 102 seconds and this time is not included in the above data. This overhead is a one-time investment and the cost can be amortized when the system is running on a large enterprise network because the Prolog bytecode will be used for all the machines on the network. In

addition one may also incrementally compile, the newly added OVAL definitions, which can further reduce the knowledge pre-processing time.

Chapter 6

Related and Future Work

This Chapter provides a review of the current research related to host-based vulnerability assessment followed by avenues for further research.

Initial developments in the field of host-based vulnerability assessment field are captured in tools like COPS [19] and Tiger [20]. Both these tools were initially unix based and focused on improper access permissions to a system and weak passwords. This initial effort was acknowledged by the security community and also promoted further research in the field of vulnerability analysis. Although these tools were designed to audit security of the host using configuration analysis, they were not built in a way to support convenient knowledge update and had to be tailored for other operating systems. To overcome the shortcomings of the original host-based analyzers, recent years have observed trends towards formalizing security definitions to be used in automatic host-based security scanning. Most notably the OVAL [7] language formalizes security knowledge used in host configuration analysis so that a host-based security scanner can be constantly updated with the most recent knowledge in the OVAL language format.

Projects like "Ferret" [21] utilize this trend and proposes modular vulnerability assessment tools which use plugins for each vulnerability to be checked on the host system. In their architecture new plug-ins have to be updated for every new vulnerability discovered. Being modular in its design, Ferret effectively supports the knowledge updates in the system. However, since the updates have to be done on each host, it causes the security knowledge

to be replicated over all the hosts.

Sufatrio *et al.* [22] have proposed - "A Machine Oriented Integrated Vulnerability Database for Automated Vulnerability Detection and Processing" combining many different security databases into a machine adaptable format. They propose architecture where the database created can be stored on the local machine or remotely. In their approach, the scanning robot queries the database after scanning the system properties to check for vulnerabilities. Even if the database is present on a remote system still in their approach, the analysis part is under the domain of the scanning robot apart from the task of querying the database and reporting the vulnerabilities. Due to the fact the analysis has to be done on the system will not only make the administrator skeptical towards trusting the code but this approach also suffers from the problem of resource consumption on the host.

The architecture proposed in this report supports advanced security analysis such as attack-graph tools (e.g MulVAL) [23–25]. Attack graph tools are high-level enterprise network security analyzers that take as input the baseline vulnerability information provided by the VA scanners. To determine the impact of the discovered vulnerabilities, the attack-graph tools are equipped with high-level knowledge on reasoning about security interactions in an enterprise network. Application of this knowledge often needs additional configuration information beyond what the VA scanner can provide. Our proposed architecture can easily accommodate this needs since the scripts that run on the target machine can collect whatever configuration information that is accessible from within a host. In the conventional architecture, another scanning agent would have to be installed for the attack-graph analysis tools.

Further the centralized architecture of our system assures access to all the hosts on the network from the server. Thus the server is also capable of hosting a network-based scanner to harness the advantages of the same. This would complement the host-based scanner and provide a extensive security analysis for the enterprise network.

The proposed scanning architecture can also be applied beyond security applications.

A likely candidate is "Inventory Management". The agents provide all the configuration information which is stored on the centralized server and this information can be used for inventory management purposes to keep track of all the resources on the hosts throughout the enterprise network.

Trouble-shooting configuration problems in an enterprise network is another application which can be applied to the proposed architecture. During trouble-shooting it is often needed to collect a range of configuration parameters and send them to a centralized place for analysis [18]. Since the scanning agent that needs to run on the target machine has a small code base, it is easier to guarantee that it will not further disrupt service while attempting to fix an existing problem. Moreover configuration changes on a particular host can be tracked due to access to the configuration information before and after the problem occurred. The changes made in machine configuration is a logical starting point when troubleshooting the respective host.

Thus, the proposed architecture provides a novel approach to host-based vulnerability analysis along with a lot of potential for further research and development in many other fields including the field of vulnerability analysis.

Chapter 7

Conclusion

Security threats and breaches in an organization's network infrastructure can cause critical disruption of business processes and lead to information and capital losses. A potent security system is imperative for an enterprise networks and vulnerability assessment is an important element for the same. A host-based vulnerability scanning system informs us about the vulnerabilities that the respective host carries.

The conventional host-based vulnerability analysis approach, though effective for individual systems does not support the trend of knowledge sharing and global perspective when it comes to security analysis. The conventional architecture also suffers from the problems of knowledge replication and need to regularly update agents residing on all the hosts. Apart from the above, the complexity of the scanning agent and its nature of utilizing host resources does not help the system administrator's concern towards installing an unknown software on the network. The current security scenario demands an approach which focus not only on being able to assimilate data from many different knowledge sources but which would also become the foundation for advanced security analysis.

The centralized host-based security scanning architecture proposed in the report is one such approach. It supports knowledge assimilation from various sources providing a comprehensive security analysis. The centralized architecture overcomes the issue of knowledge replication and eradicates the need to regularly update the client due to its separation of analysis from the data collection part and further performing all the analysis on the cen-

tralized server. The agent residing on the host uses minimal resources and is simple enough to install and maintain. The reduced code base of the agent makes it less susceptible to programming flaws and also allows rigorous code vetting to gain the administrators' confidence. We empirically show that the centralized vulnerability analyzer to be at par with Mitre's reference scanner in terms of vulnerability detection and comes out superior in terms of time taken to do the assessment.

This work has contributed to the evolving trend of knowledge sharing in security analysis by providing a centralized host-based architecture which apart from providing a comprehensive security assessment, supports advanced security analysis and research

Bibliography

[1] Network and Host-based Vulnerability Assessment, http://documents.iss.net/whitepapers/nva.pdf.

[2] Introduction to vulnerability assessment tools, NISCC Technical Note, 2004.

[3] Information Security Glossary, http://www.yourwindow.to/information-security/gl_securitybreach.htm.

[4] R. Richardson, CSI:Computer Crime and Security Survey, http://i.cmpnet.com/v2.gocsi.com/pdf/CSISurvey2007.pdf, 2007, Web page fetched on April 19, 2008.

[5] Nessus, http://www.nessus.org/nessus.

[6] X. Ou, S. Govindavajhala, and A. W. Appel, MulVAL: A logic-based network security analyzer, in *14th USENIX Security Symposium*, pages 113–128, Baltimore, 2005.

[7] OVAL (Open Vulnerabilities and Assessment Language), http://oval.mitre.org/.

[8] B. Marick, *The craft of software testing*, Prentice Hall, 1995.

[9] I. V. Krsul, *Software Vulnerability analysis*, PhD thesis, Purdue University, 1998.

[10] Vulnerability Assessment Scanning, http://www.sunbeltsoftware.com/documents/snsi_whitepaper.pdf, 2004, Web page fetched on November 11, 2008.

[11] Searchsecurity, http://searchsecurity.techtarget.com/sDefinition/0,,sid14_gci1176511,00.html.

[12] R. Fussell, Vulnerability assessment: Network based versus host based, Technical report, SANS Institute, 2002.

[13] OVAL Interpreter, http://oval.mitre.org/language/download/interpreter/index.html.

[14] GFILANguard, http://www.gfi.com/lannetscan/.

[15] SofCheck, http://www.sofcheck.com/.

[16] Common Vulnerabilities and Exposures, http://cve.mitre.org/.

[17] National Vulnerability Database, http://nvd.nist.gov/.

[18] H. Huang et al., PDA: A tool for automated problem determination, in *21st Large Installation System Administration Conference, (LISA)*, pages 153–166, Dallas, 2007.

[19] D. Farmer and E. H. Spafford, The COPS security checker system, in *Summer Usenix Conference*, pages 165–170, Berkley, 1990.

[20] Tiger Analytical Research Assistant, http://www-arc.com/tara/index.shtml, Web page fetched on May 07, 2008.

[21] A. Sharma, J. R. Martin, N. Anand, M. Cukier, and W. H. Sanders, Ferret: A host vulnerability checking tool, in *10th IEEE Pacific Rim International Symposium on Dependable Computing*, 2004.

[22] Sufatrio, R. H. C., and L. Zhong, A machine-oriented integrated vulnerability database for automated vulnerability detection and processing, in *18th Large Installation System Administration Conference, (LISA)*, Atlanta, 2004.

[23] S. Jajodia, S. Noel, and B. O'Berry, Topological analysis of network attack vulnerability, in *Managing Cyber Threats: Issues, Approaches and Challanges*, edited by V. Kumar, J. Srivastava, and A. Lazarevic, chapter 5, Kluwer Academic Publisher, 2003.

[24] R. Lippmann et al., Evaluating and strengthening enterprise network security using attack graphs, Technical Report ESC-TR-2005-064, MIT Lincoln Laboratory, 2005.

[25] X. Ou, W. F. Boyer, and M. A. McQueen, A scalable approach to attack graph generation, in *13th ACM Conference on Computer and Communications Security, (CCS)*, pages 336–345, 2006.

Appendix A

OVAL Definition in XML Format

```
<oval_definitions>
   <definition id="oval:org.mitre.oval:def:153" class="vulnerability" version="1">
    <criteria operator="OR">
    ------------------------------------------------------------------------
   | <criteria comment="Outlook 2000" operator="AND">                       |
   |    <criterion negate="false" test_ref="oval:org.mitre.oval:tst:895" />  |
   |    <criterion negate="false" test_ref="oval:org.mitre.oval:tst:162" />  |
   |  </criteria>                                                           |
    ------------------------------------------------------------------------
    </criteria>
   <definition>
   <tests>
    <registry_test id="oval:org.mitre.oval:tst:895"
check_existence="at_least_one_exists" check="at least one">
     <object object_ref="oval:org.mitre.oval:obj:670" />
     <state state_ref="oval:org.mitre.oval:ste:804" />
    </registry_test>
    <file_test id="oval:org.mitre.oval:tst:162"
   check="at least one" check_existence="at_least_one_exists">
```

```
  <object object_ref="oval:org.mitre.oval:obj:97" />
  <state state_ref="oval:org.mitre.oval:ste:160" />
 </file_test>
</tests>
<objects>
 <registry_object id="oval:org.mitre.oval:obj:670">
  <hive>HKEY_LOCAL_MACHINE</hive>
  <key>SOFTWARE\Microsoft\Office\9.0\Outlook\InstallRoot</key>
  <name>Path</name>
 </registry_object>
 <file_object id="oval:org.mitre.oval:obj:97">
  <path var_ref="oval:org.mitre.oval:var:728" />
  <filename>Outllib.dll</filename>
 </file_object>
</objects>
<states>
 <registry_state id="oval:org.mitre.oval:ste:804">
  <value operation="pattern match">.*\\[Oo][Ff][Ff][Ii][Cc][Ee][\\9].*</value>
 </registry_state>
 <file_state id="oval:org.mitre.oval:ste:160">
  <version datatype="version" operation="less than">9.0.0.8954</version>
 </file_state>
</states>
<variables>
 <local_variable id="oval:org.mitre.oval:var:728">
  <object_component item_field="value" object_ref="oval:org.mitre.oval:obj:120" />
 </local_variable>
```

```
    </variables>
</oval_definitions>
```

Appendix B

Converted OVAL Definition in Prolog Format

```
definition('oval:org.mitre.oval:def:153'):-
criteria(unique_id0).

criteria(unique_id0):-
criteria(unique_id1,'Outlook 2000').

criteria(unique_id1,'Outlook 2000'):-
criterion('oval:org.mitre.oval:tst:162','false'),
criterion('oval:org.mitre.oval:tst:895','false').

criterion('oval:org.mitre.oval:tst:162'):-
oval_test('oval:org.mitre.oval:obj:97','oval:org.mitre.oval:ste:160','
                at least one','at_least_one_exists').

criterion('oval:org.mitre.oval:tst:895'):-
oval_test('oval:org.mitre.oval:obj:670','oval:org.mitre.oval:ste:804','
                at least one','at_least_one_exists').
```

```
object_ref('oval:org.mitre.oval:obj:97',filepath_ref(X,'Outllib.dll')):-
var_ref('oval:org.mitre.oval:var:728',X).

state_ref('oval:org.mitre.oval:ste:160',version('less than','9.0.0.8954')).

object_ref('oval:org.mitre.oval:obj:670',winReg_ref('hkey_local_machine\\software\\
            microsoft\\office\\9.0\\outlook\\installroot','Path')).

state_ref('oval:org.mitre.oval:ste:804',val('pattern match','.*\\[Oo][Ff][Ff][Ii][Cc]
            [Ee][\\9].*')).

var_ref('oval:org.mitre.oval:var:728',X):-
object_ref('oval:org.mitre.oval:obj:120',Obj),
object_val(Obj,X).
```