MQP CDR#GXS1002

Cloud Vulnerability Assessment

a Major Qualifying Project Report
submitted to the faculty of the
WORCESTER POLYTECHNIC INSTITUTE
in partial fulfillment of the requirements
for the Degree of Bachelor of Science by

Mika D. Ayenson

Andre Guerlain

April 26, 2012

Professor Gábor N. Sárközy, Major Advisor

Professor Stanley M. Selkow, Co-Advisor

Abstract

In 2009, the IBM T.J Watson Research Center investigated risks that administrators and users face while using cloud resources. In "*Managing Security of Virtual Machine Images in a Cloud Environment*", they propose a new system called "Mirage". This system was developed as a template that cloud users could use, to create a robust exploit mitigation system and provided security for virtual machines in the cloud. In this study, we reassess vulnerability assessment in the Cloud and develop the opensource "cvaFrame" framework. This framework reports to the cloud administrator, vulnerabilities and exploits discovered in virtual images, which are in the image repository. Our contribution to the cloud vulnerability assessment research community is research specifically targeting dormant public images. We build our framework on top of existing tools like Metasploit and OpenVAS and implement it in an existing cloud service "OpenNebula", to prove that our system works. We tested our software in a production cloud and it was able to find vulnerabilities and exploits hiding in dormant virtual machine images. At this date, we believe we are the first to develop an open-source framework that performs vulnerability assessment within the cloud on dormant images.

Acknowledgments

Gábor Sárközy, MQP Advisor

Sándor Ács, Project Advisor and SZTAKI contact

Márk Gergely, Co-Project Advisor and SZTAKI contact

Peter Kacsuk, SZTAKI Department Head

Stanley Selkow, MQP Co-Advisor

Róbert Lovas, SZTAKI Program Director

Péter Kotcauer, SZTAKI colleague

MTA-SZTAKI

Worcester Polytechnic Institute

Table of Contents

Abstract 2
Acknowledgments 3
1 Introduction 7
2 Background 9
2.1 Brief Overview 9
2.2 Cloud Computing 9
2.2.1 What is cloud computing? 9
2.2.2 Benefits of cloud computing 10
2.3 Vulnerabilities vs. Exploits 10
2.4 What is Malware? 11
2.5 What is a vulnerability scanner? 12
2.6 Hacking vs. Penetration Testing 12
2.7 Understanding Cloud Computing Vulnerabilities 12
2.8 Vulnerability Assessment Scanners 13
2.9 Vulnerability Scanning and Cloud Computing 14
2.10 Vulnerability Assessment with Application Security 15
2.11 Vulnerability Assessment Through the Cloud: A Case Study 16
2.12 Analysis on Cloud-Based Security Vulnerability Assessment 16
2.13 SecTools.Org: Top 125 Network Security Tools 17
2.13.1 Antimalware 17
2.13.2 Rootkit detectors 17
2.13.3 Vulnerability exploitation tools 17
2.13.4 Vulnerability scanners 18
2.14 Migrating Automatic Self-Testing To the Cloud 18
2.15 Vulnerability-assessment services on the rise 18
3 Related Research 20
3.1 CSAGUIDE 20
Figure 3.1.1 20
3.1.1 Interoperability and Portability 21
3.1.2 Virtualization 21
3.1.3 Security for Cloud Computing 21
3.1.4 Security as a Service 22
3.1.5 Information Management and Data Security 22
3.1.6 Application Security 23
3.1.7 Application Penetration Testing for the Cloud 24
3.2 A New Middleware Service for Making Grids Interoperable 24
3.3 OpenNebula 25
3.4 Managing Security of Virtual Machine Images in a Cloud Environment 26
4 Project Goals and Design 30
4.1 Assumptions 30
4.2 Experimental Design 30
4.2.1 Prerequisites 30
4.2.2 Cloud Vulnerability Assessment (CVA) 31

Figure 4.2.1 32
4.3 Methodology 32
4.3.1 Metasploit 34
Figure 4.3.1 37
4.3.2 OpenVAS 38
Figure 4.3.2.1 38
4.3.3 Reported Information Storage VM 42
4.3.4 Backtrack 47
4.3.5 Other Tools 48
4.4 Testing Environment 48
5 Results and Discussion 51
5.1 AutoVAS Module 51
5.2 Sploit Module 51
5.3 Both Modules 52
5.3.1 Test 1: Sploit then AutoVAS 52
5.3.2 Test 2: AutoVAS then Sploit 52
5.4 Results Graphs 53
6 Conclusion 55
6.1 Cloud Vulnerability Assessment 55
6.2 Further Work 55
6.2.1 Automatic Exploit Patches 56
6.2.2 More Modules 56
6.2.3 Encryption and Key Management 57
6.2.4 cvaFrame Privledges 57
References 58
Appendix 62
Example Database Output 62

List of Figures

Figure 3.1.1: Cloud Taxonomy.....20
Figure 4.2.1: CVA Framework Diagram.....32
Figure 4.3.1: Metasploit Module Diagram.....37
Figure 4.3.2.1: OpenVAS Framework Design.....38
Figure 4.3.3.2: OpenVAS Module Design.....41
Figure 4.3.3.1: Reported Information Storage VM Design.....44
Figure 4.3.3.2: CVA Framework Database Tables.....46
Figure 4.4.1: Chart of Virtual Machines Tested.....50
Figure 5.4.1: Sploit Functional Test.....53
Figure 5.4.2: AutoVAS Functional Test.....54
Figure 5.4.3: Performance Test.....54

1 Introduction

Companies and users alike continue to transition to cloud system resources. As more individuals migrate to the cloud, more personal data such as bank information, transaction reports, domain services, and even full imaged operating systems are increasingly targeted and vulnerable. This data is susceptible to attackers if continuous monitoring and maintenance is not conserved. Attackers can maliciously hack into a cloud system, steal this data, or even corrupt an entire system. This in turn leaves room for improvement in the cloud security model for researchers to explore, with hopes to mitigate these attacks. Previous research explains the history of in-house developed tools and security teams working non-stop to scan their systems and patch vulnerabilities [11]. A few common weaknesses with this system are

(i) it forces several security officers to work non-stop,

(ii) the in-house tools that are woven together are not effectively managed and,

(iii) tools used are not scalable and more importantly are not automated to work as one system.

In this study, we investigate cloud vulnerabilities and develop a system for security administrators to use that will report known vulnerabilities in their cloud system. Our open source CVA-framework is an effective solution because it can be built upon to fit any cloud system's needs. Inside our framework, we develop several modules that a security administrator can choose from. We designed each module to be automated in order to abstract each module configuration. This approach allows the security manager to interact only with the CVA. More importantly we compile all the data that we have collected from each module and place the output in a database that a security administrator can review.

To prove that our system works, we have chosen to implement our model in the open source cloud data center, OpenNebula Cloud Front End (CFE) [24]. We begin by developing two main modules, namely a Metasploit module and an OpenVAS module. The Metasploit module builds directly on top of the Metasploit framework [28] and the OpenVAS module similarly is developed on the OpenVAS framework [26]; both are described in more detail in the project goals and design subsections. We then test our modules on the image repository that resides in the CFE and report vulnerabilities and exploits. Later we extend our framework by discussing more modules that can be used as a security administrator.

The following paper is arranged as such: first we present necessary background information to the reader assuming little to no prior knowledge on the topic. We briefly discuss what a cloud is, what are vulnerabilities and exploits, vulnerability scanners, security tools, and pertinent articles and presentations. We also answer other minor questions such as, "What is the difference between hacking and penetration testing?" In the related research section, we take a step deeper towards our experiment to explain how to appropriately manage security of virtual machine images in a cloud environment. In our project goals and design section, we briefly express our project ideas, and later provide a design model in the experimental design section. Next is our methodology, which provides implementation of our framework, and then we provide our results and discussion section, which proves that our system works in a real cloud-computing environment. We show that this framework can be easily extended upon just as other open source frameworks. Last we conclude with our project constraints and future work.

2 Background

2.1 Brief Overview

In order to fully understand vulnerability assessment in the cloud, we must first present a few key aspects that tie a threat model in the cloud together. We first define cloud computing, what a cloud is, and the benefits of cloud computing. We then explain the difference between vulnerabilities and exploits, followed by how both of these can elude to other attack vectors such as malware, and rootkits. Next we present vulnerability scanners, and how they can be used to mitigate attacks. Before we begin our literature review we explain the difference between hacking and penetration testing. The following sections are then comprised of several academic articles and presentations that we felt were necessary to discuss, in order to provide a solid foundation on the nature of this topic. We introduce a simple understanding of cloud vulnerabilities, vulnerability assessment scanners, and then the notion of vulnerability assessment on the cloud. Next we discuss an analysis on cloud-based security vulnerability assessments, and provide a short list of important security and networking tools. We finally conclude this section by discussing how to conduct a vulnerability assessment, how to migrate automatic testing to the cloud, and what vulnerability assessment tools are on the rise.

2.2 Cloud Computing

2.2.1 What is cloud computing?

Cloud computing still lacks a clear definition as to what exactly it is, although it has been described as a service rather than a product [15]. The CSA Guide defines it as, “a model for

enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g. networks, server, storage, applications, and services) [20]." In history we have developed from mainframes to PC's to client servers and then to web services. Now we have reached this vague idea of a cloud, where users may pay for the resources that sit on the cloud. It essentially is a remote set of resources such as servers that other people connect to in order to host their data over the Internet. The cloud is not just one set of resources that everyone around the world uses. There are many cloud computing data centers with their own physical infrastructure, which may or may not benefit different customers.

2.2.2 Benefits of cloud computing

Some benefits of cloud computing are that anyone can access the cloud from anywhere as long as they have a network connection, it is scalable, and you only pay for the resources you use. All three of these benefits focus on easy cheap access to resources. This means that an individual can create an online company, host their data on the cloud resources, and not have to manage their own physical infrastructure.

2.3 Vulnerabilities vs. Exploits

Vulnerabilities and exploits have existed in computer systems since the development of the computer. As new machines and applications are developed, new vulnerabilities and exploits are found and patched. There is a cycle of discovering these holes in programs and creating a protection scheme to mitigate being attacked; the Metasploit section summarizes each. Vulnerability is "A security flaw or weakness in an application or system that enables an attacker to compromise the target system. A compromised system can result in privilege escalation, denial-of-service, unauthorized data access, stolen passwords, and buffer overflows." Whereas

an *Exploit* is "A program that takes advantage of a specific vulnerability and provides an attacker with access to the target system." In short a clear difference between the two is, vulnerability is typically a bug in software created by a developer and an exploit typically is an attacker taking advantage of that bug [29]. In our investigation, we analyze systems and discover vulnerabilities as well as exploits and report to the security administrator what areas need to be patched in order to secure each one.

2.4 What is Malware?

Another major problem with vulnerabilities is that malware can be installed without a user's knowledge. Malware can be Trojans, viruses, spyware, rootkits, worms or even adware. Each of these types of malware can take advantage of vulnerabilities and result in devastating effects as discussed in section 2.3. A virus requires human interaction; however it can exist inside an executable program such as a video game and be run once the video game is started. It will then start to spread to other programs and corrupt an entire system. Worms are similar to viruses, however they do not need human interaction to spread. Trojan horses appear as a useful program such as an Internet explorer icon and can also damage a system. Spyware and rootkits are similar in the sense that they watch a user's interaction while they use their computer and key log data such as passwords and other sensitive information. This logged data is then sent to the attacker. Lastly, adware is usually a program that swindles users by displaying many advertisements to them until they purchase an item. Due to the nature of vulnerabilities and exploits carrying payloads, we have investigated a malware scanner as a module in our study, which we discuss later in Section 4.3.2.

2.5 What is a vulnerability scanner?

A vulnerability scanner is a tool that can scan an entire system for known vulnerabilities. It typically works in conjunction with a database full of known vulnerabilities and cross checks the database with any exploits the scanner may find. The security implications behind using a scanner on a system are:

(i) they provide reports on vulnerabilities found,

(ii) and can be automated to scan an entire system.

One of the downsides to automated reports is that sometimes it reports false positives, also known as a false alarm. If there are many false alarms, the reliability of the vulnerability scanner diminishes, and security administrators must spend many hours checking each false positive. In our study, we use OpenVAS to cross check with reports that other modules produce.

2.6 Hacking vs. Penetration Testing

One simple difference between hacking and penetration testing is permission. Sometimes penetration testing is also referred to as ethical hacking, however if a user is not granted access to test exploits on a system, then they are by default hacking into the system. In our investigation, we use several penetration methods on a production OpenNebula cloud front end. We believe that in order to protect a system against attackers, you must first think as one. These methods are described below in our methodology, section 4.3.

2.7 Understanding Cloud Computing Vulnerabilities

As stated in the introduction, cloud computing introduces the idea of many virtual machines on a single physical infrastructure. "Understanding Cloud Computing Vulnerabilities" contained a

detailed description of the difficulties faced when dealing with cloud security rather than the security of a single machine [5]. These difficulties are known as *Control Challenges*. In other words, there are instances where security is more difficult to maintain once the unique attributes of a cloud are combined with more traditional security issues. A description of generic cloud infrastructure is presented, as it is a primary basis of the entire paper. The authors discussed many related topics, including the importance of determining the benefits and risks of cloud infrastructure in regards to security. The cloud is simply an environment in which the user is given specific privileges. The probability and implications of a user obtaining access without permission, is a crucial point of discussion when designing how to best defend the cloud. In addition, cloud-specific vulnerabilities are discussed as well as their possible solutions. Specific attacks are also discussed, including session riding attacks, virtual machine escape, and exploiting deprecated cryptography. This report concludes with the idea that since it is not possible to fully protect certain aspects of the cloud, perhaps adding a vulnerability assessment tool to the cloud infrastructure is both viable and necessary.

2.8 Vulnerability Assessment Scanners

Vulnerability assessment scanners are tools that scan networks and provide a diagnostic report on discovered vulnerabilities. The scanner cross checks with a database of known common security holes and tries to exploit them. Any detected security holes are classified, and mentioned to the administrator along with a set of recommendations that they should take to improve the security of their network. Chicago lab researchers in their investigation tested the top vulnerability scanners designed to test networks [6]. They tested Axent Technologies' NetRecon, BindView Corp.'s HackerShield, eEye Digital Security's Retina, Internet Security

Systems' Internet Scanner, Network Associates' CyberCop Scanner, and two open-source products: Nessus Security Scanner and Security Administrator's Research Assistant (SARA) on five platforms (Hewlett-Packard Co. HP-UX, Microsoft Windows NT, Novell NetWare, Red Hat Linux and Sun Microsystems Solaris). In their study, they tested these scanners with seventeen of the top known vulnerabilities and report that none of the tested scanners found all seventeen vulnerabilities. One interesting point made in this article described the notion of the 'Numbers Game'. They argue that a numbers game is not an accurate argument to determine what scanner is best. For example one scanner may have scanned the fastest, whereas another may have scanned slower and found more vulnerabilities. Or there were instances when one scanner scanned significantly more files than another scanner, but found fewer vulnerabilities. They conclude by proposing an ideal vulnerability scanner with a few key features. The database filled with vulnerabilities must remain up-to-date and accurate, it must be limited to the amount of false positives, and it should also store multiple scans for performance trends along with an easy method to fix discovered problems.

2.9 Vulnerability Scanning and Cloud Computing

The Internet has rapidly become vital to international commerce, and with commerce comes those who would attempt to exploit it. The importance of cyber security for businesses is increasing as companies and consumers increasingly rely on the Internet. Without secure servers, web applications, and infrastructure, businesses cannot expect to effectively secure their data. Scanning their resources is mandatory to stay ahead of potential attackers, but utilizing the cloud security presents a challenge to this necessity. In many cases, it is far more efficient for companies to utilize online resources rather than spend money bolstering and protecting their

own physical infrastructure. The problem is that although this service is very cost-effective, it comes with increased risk, because the companies can no longer manage their own security. This problem stems from the difficulty of differentiating between companies assessing their own resource vulnerability, and attackers attempting to breach security in order to steal information. Therefore, it is necessary for cloud administrators to provide a service to scan and test the resources of their clients for the benefit of both, despite the load on the cloud to perform the service. When discussing this topic the author, Craig Balding of CloudSecurity, suggests "Something like a 'ScanAuth' API call offered by cloud providers that a customer can call with parameters for conveying source IP address(es) that will perform the scanning, and optionally a subset of their Cloud hosted IP addresses, scan start time and/or duration [16]."

2.10 Vulnerability Assessment with Application Security

Web security is a quickly evolving subject because the web environment is constantly shifting with the introduction of new services. While the web offers an easier interface for consumers and companies, it also amplifies the potential for vulnerability exploit attacks. The dynamics of attacks over the web is morphing from a small number of targeted attacks on large institutions to many smaller attacks on a variety of smaller targets [8]. This change has caught many small companies and private individuals off guard, because they operated on the assumption that attacks on their resources were less likely. An increasing number of companies have started to protect themselves against this threat. In light of this, Geelan, et al. discussed the steps that smaller companies are taking in order to protect their assets. When discussing this topic, they emphasized that no security is perfect because the environment is constantly changing. They

argue that best protection is achieved by integrating multiple vulnerability scanners and exploit programs to get the best of each and therefore deter future attacks.

2.11 Vulnerability Assessment Through the Cloud: A Case Study

The chief information security officer at the Office of the Comptroller of the Currency (OCC) released a presentation, which focused on the cloud vulnerabilities that the OCC faces. The OCC contains extensive amounts of personal and uniquely identifiable information, which attackers try to steal [11]. Some of the types of attacks he reveals in his presentation are socially engineered e-mails, forged documents, or even SQL injections. Most of the vulnerabilities exist because of human error or other flaws that disregard the engineering code of ethics documentation. The OCC is just one of many major organizations that rely on cloud resources to maintain their data. Before cloud vulnerability assessment tools, administrators had to develop in-house systems that integrate several tools and forced several engineers to work full time managing their services. Now with vulnerabilities as a service (VaaS), these in-house systems are growing obsolete, especially if cloud-providing organizations want to compete with other cloud providers who have VaaS.

2.12 Analysis on Cloud-Based Security Vulnerability Assessment

Many different methods of cloud vulnerability assessment have been developed. In "Analysis on Cloud-Based Security Vulnerability Assessment" the authors focused on presenting a few of these methods, as well as drawing comparisons between them [10]. The key point was the authors' contention of having the target and the vulnerability assessment tool on the same

hardware. A much more complete scan can be taken, giving the owner the best possible assessment of risk to their cloud resources.

2.13 SecTools.Org: Top 125 Network Security Tools

SecTools is an organization, which monitors the security community, seeking out the best tools [23]. They regularly compile and update a list of the tools as well as foster the development of their own open-source projects. They provide links to other communities as well. The website features a host of different types of tools, many of which were related to our project goals.

2.13.1 Antimalware

Antimalware tools are specifically designed to detect and remove harmful viruses and Trojans from infected machines. Additionally, they help prevent machines from becoming infected by other malware as seen in section 2.4. Malware is used to either steal personal information from the owner or to take control of certain aspects of the machine and utilize the resources.

2.13.2 Rootkit detectors

Rootkit detectors are designed to find rootkits, which are programs, which by nature hide themselves from the operating system and by extension the user. Rootkits can intentionally leave openings through which malware can enter and corrupt the machine.

2.13.3 Vulnerability exploitation tools

Vulnerability exploitation tools are made to systematically seek out and attack vulnerabilities on a target machine. This means they will attempt to use potential vulnerabilities to take control of a machine and give control to an attacker.

2.13.4 Vulnerability scanners

Vulnerability scanners search machines for potentially harmful vulnerabilities where an attacker or program could potentially attack and exploit the machine. They are designed to alert the user as to where they need to patch in order to best protect their resources.

2.14 Migrating Automatic Self-Testing To the Cloud

The focus of this paper is to discuss the possibility of migrating the testing of personal resources to the cloud. Users are unable to assess their own resources because the host cannot ascertain whether the penetration testing and vulnerability scans are benign self-tests or intrusion attempts by an attacker[17]. For this reason, many cloud providers specifically ban any security assessment upon cloud resources by their owners. This is in neither party's best interest because both the end user and the cloud providers are more at risk in the long run. The only way to get around this issue is to move testing within the cloud infrastructure, which also makes it far more effective because access is easy. This paper goes on to discuss a specific design for implementing this type of cloud security using an automated system. In particular, it is noted that since the risk assessment of cloud images is only one at a time, the idea is practical because it will not require too many resources.

2.15 Vulnerability-assessment services on the rise

As cloud-computing demands have increased, security implications are now more vital than ever. New tools such as scanners are used to take an automated analytical approach that can determine that vulnerabilities not only exist, but also can effectively correct security flaws [4]. A few tools some online companies are using are Nessus, OpenVAS, Nexpose, or other vulnerability

scanning tools. These scanners over the years have developed from unreliable tools that reported false positives, lacked scalability, and other bugs that prevented the tools from effectively scanning the network for vulnerabilities. For example, they did not provide detailed information or control services. These major vulnerability assessment tools, despite their flaws, have provided an impetus for the development of new tools.

3 Related Research

3.1 CSAGUIDE

The Cloud Security Alliance provides extensive recommendations on reducing risk in cloud computing. The image below illustrates the differences between IaaS, cloud software, PaaS and software as a service [20].

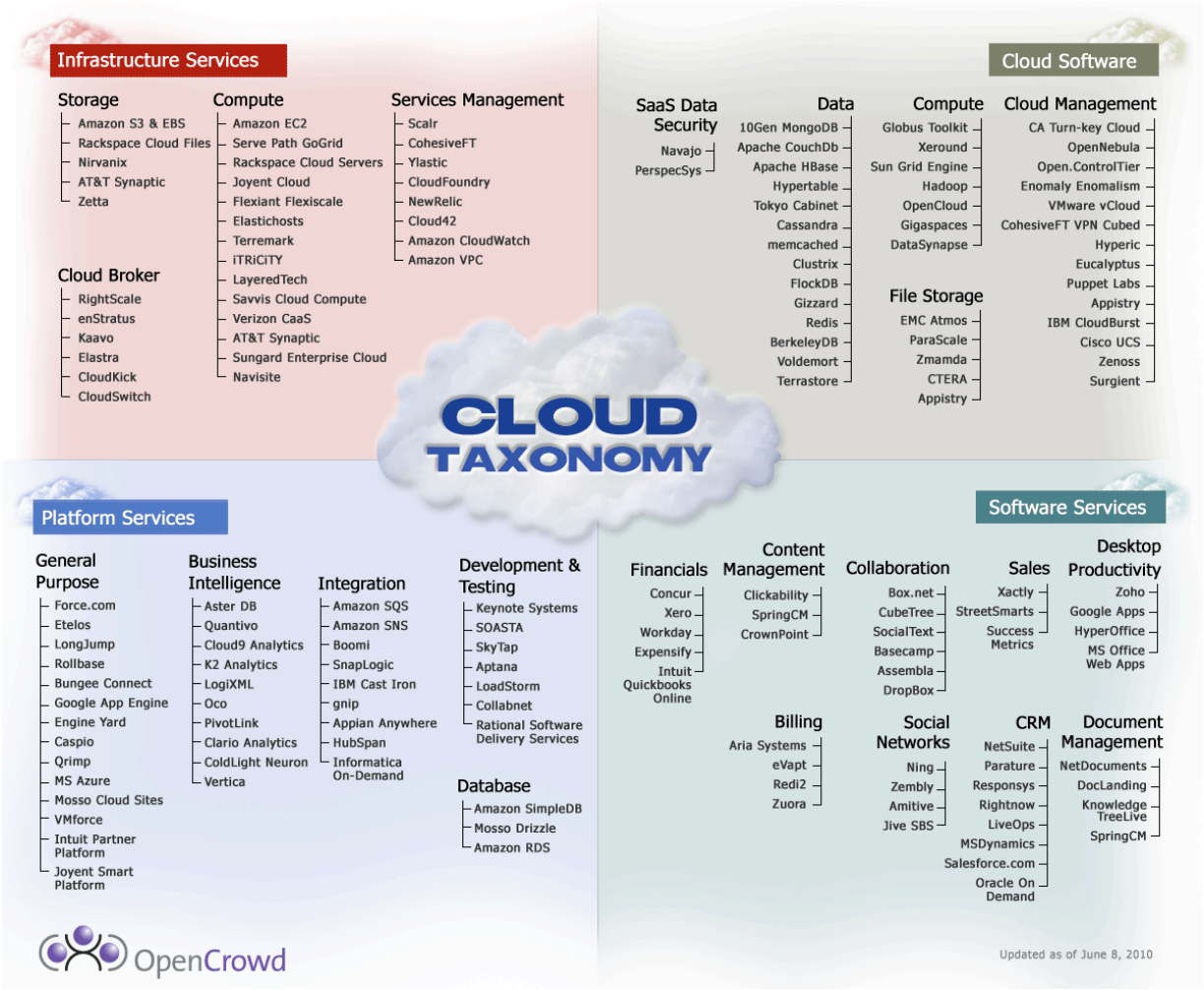

Figure 3.1.1

3.1.1 Interoperability and Portability

Interoperability and portability allow for rapid expansion of computing resources, because cloud services can quickly be purchased and appear to the user as a seamless part of a local computer infrastructure. Unfortunately, it is not identical in the sense that security is far harder to manage. To make the system seem unified, data must be transferred constantly to and from the local infrastructure as well as the cloud service. Fortunately these concepts are not new, having existed before cloud computing. However, without these two attributes, the cloud would never be a viable or useful idea.

3.1.2 Virtualization

Virtualization is vital to Infrastructure, Platform and Software-as-a-Service. Computing resources are most efficiently used when you can utilize multi-tenancy and operating with a common data center. However, many concerns are brought about when using VM's in addition to questions about the amount of resources it would take to properly monitor them. Using VM's create inter-VM attacks, guest users, and many more potential threats to the overall system. The shared data center is also more at risk along with the challenge of encrypting VM's in order to protect them when they are dormant within the image repository.

3.1.3 Security for Cloud Computing

Maintaining security within a cloud environment is very similar to managing more traditional security risks. However, the cloud's structure and underlying technology adds certain new elements of risk. To assess these risks, the authors of the CSA guide also note that there are four significant factors to note. Firstly, the resources, assets, and information must be taken into

account. It is important to know how they are being managed within the cloud. Another key question that needs to be considered is, "Who is managing the resources and how are they doing it?" Also, what security controls are there and how are they built into the system? It is important to figure this out quickly because this is the only way to differentiate whether the administrator or user is responsible for each security risk. Cloud users must be aware of what security risks are handled for them by their service provider in order to know which security precautions they must take to protect their assets.

3.1.4 Security as a Service

Security as a service is a potential way to secure the cloud resources. The idea is that there will be a central framework, which is designed to scan cloud resources and report back to the owners of these resources automatically. It is a service because the cloud administrators cannot differentiate between a user testing their own security and an attacker, and therefore must provide it as an internal service. Otherwise, there is no feasible way for a customer to know if their resources are secure, and the entire cloud is more likely to become compromised. In the CSA guide, the authors detail how this must be included in the contract between the service providers and the users.

3.1.5 Information Management and Data Security

The cloud environment is constantly changing, with new VM's and servers starting and stopping continuously. This makes data management a bit more complicated than when it was managed in static, non-virtualized resources. To adapt to this new environment, a *Data Security Lifecycle* is recommended. This lifecycle starts with the creation of any data. Once created, the data must be stored somewhere, possibly in some format for security purposes. Afterwards, someone can

access that data in order to use it. One of the possible uses is to then share that data with someone else, or with another computing resource. In addition, data can be archived for longer-term storage. Eventually, data will be destroyed either by digital or physical means. This lifecycle is not always completely used, as a file in not always used, or archived, or even destroyed for the purposes of examining data security in a cloud.

During this lifecycle, many factors must be accounted for by an organization's security administrator in order to maintain data security. It is important to classify all information to something like "trade secret" or "low priority", so that it is easy to monitor it all. Then, policies for each classification must be put in place to keep data secure. This has to take into account location. Limiting authorization to access data is important to restrict how many people can pose a risk to the data. Finally, ownership of the data and who is designated to manage it also needs to be accounted for.

3.1.6 Application Security

The size, variability and interactivity of a cloud challenge traditional notions about application security. The authors note that cloud applications need a similar level of security to those connected directly to the Internet because they are far more exposed than they would be in a stand alone system. Much like data security within the cloud, there is a life cycle, the *Software Development Life Cycle* that should be understood before designing an application for, or using an application on, the cloud. This life cycle details how an application should be protected as well as the states an application can enter. The developer must examine these states to account for possible security issues which may arise in each.

3.1.7 Application Penetration Testing for the Cloud

Within a cloud computing system, there can be many active VM's at once. Each of these VM's can have many different applications running which all may have vulnerabilities. Therefore, doing penetration testing on these applications is vital to cloud security. The testing would probably not be application specific and treat applications as a "black box", meaning that the penetration tester would be looking for more general and common programming mistakes which led to vulnerabilities. Other kinds of testing, which could potentially be run, are tests such as denial of service attacks, to see if the application is vulnerable to it. It is imperative to keep in mind that the attacker could be either inside or outside the cloud in order to best protect and assess the application.

3.2 A New Middleware Service for Making Grids Interoperable

Metabrokering is a method for managing grid infrastructure as well as making the grid interoperable. Researchers at SZTAKI developed this system in order to allow grid computing to move beyond the realm of research and into the commercial market. The idea behind grid computing is not new. Shared infrastructure was the method by which universities provided computing services starting in the 1960's until the microprocessor became cheap enough for personal computers. The idea was to provide computing as a service to a large group of users. This idea came about from the realization that a group of users uses a considerably more powerful computational resource far more efficiently than a single user with a personal machine [7]. The research topic of "A New Middleware Service for Making Grids Interoperable" was primarily to investigate and propose a new method to make grids interoperable rather than isolated [9]. The main challenge for researchers is that unlike single computers, each grid

infrastructure is generally a unique and complex set of computing resources. Since each grid contains a large, yet different set of resources, each grid also provides a corresponding set of services to their customers. If one grid can support many different users at once with a certain number of services and a certain degree of quality, it follows that two interoperable grids could provide an even better service to the users of both grids.

The most relevant issue discussed by the authors was the design of grids in general. The grid distinctly separates the user from the hardware, meaning that they will be fully unaware of how the grid is working, while fully understanding how to use the grid. This creates many new security issues, which the cloud administrator must deal with because the user no longer has the ability to manage their own security. This ties in well with OpenNebula, where in a similar fashion, the user does not need to know how the cloud operates, only understand how to utilize it.

3.3 OpenNebula

OpenNebula is open-source software designed to allow data center virtualization on local infrastructure [24]. A virtualized data center is essentially a collection of virtual resources. These resources can be used in order to store or process data using virtual machines and servers. It is designed to be interoperable, allowing it to communicate easily with other clouds. In addition, OpenNebula was designed with adaptability in mind, meaning it assimilates well with almost all local infrastructures. OpenNebula is designed to run as quickly and reliably as possible, with every type of virtual machine it supports, running any Operating System, whereas commercial cloud management systems are more inclined to favor one. As long as the hardware is robust enough to run a data center, OpenNebula will serve that purpose well.

OpenNebula is exceedingly popular in research areas because unlike commercially built systems, the code is open source, allowing researchers to pioneer new features and implementations [25]. They can also tweak OpenNebula in order to customize it to their research goals. OpenNebula's flexibility has motivated SZTAKI to run their datacenter in their experimental cloud. Implementing our new modules is fairly easy considering the overall size of the program. While OpenNebula is popular in research, it is also applied commercially because it is overall a very robust piece of software.

There are many benefits of using OpenNebula [24]. OpenNebula is capable of adding new hardware hosts to the cloud only when needed, can support local or remote cloud resources, and administrators can control all of the infrastructure from a centralized manager. This means the user has a very flexible environment in which to work. OpenNebula allows the user to run multiple machines with different and even incompatible required software, all while using the same shared hardware.

Our work is designed to be flexible, meaning when we apply it to OpenNebula, it will appear to the cloud administrator as part of the cloud front end. This way, the administrator can easily run each different security tool from the cloud front end, using CVA commands. Our work is not designed specifically with OpenNebula in mind, although we tested our CVA tool with this specific cloud infrastructure, as seen in section 4.4.

3.4 Managing Security of Virtual Machine Images in a Cloud Environment

Researchers from the IBM T.J. Watson Research Center investigated how to securely manage virtual machine images (VMimages) in the cloud [18]. They propose an image management system that incorporates several features such as filtering, scanning, user validation, and security

flaw repairs. They conclude by arguing that their "Mirage" system improves scalability and out-performs traditional vulnerability assessment tools in the cloud. Their research has provided a great foundation for further research in the cloud because they look at a variety of attack vectors. Due to the generality of their system to be implemented in any cloud system, they make some assumptions that will not function with every cloud system; these assumptions and implementation details will be flushed out in the following sections. For this reason, we have used the knowledge presented in their research and developed our own model that will provide vulnerability assessment in the cloud and aim to implement the system.

There are two key points that we took into consideration while developing our system. In VMimages, there must be high integrity and security because they determine the initial states of running images. The other key point made is that there are VMimages that are shared among unrelated users. For example, a user given named 'x1' may start an image provided by user 'x0'. User 'x0' may have initially created this image for malicious reasons such as leaving viruses that could access the host machine, embedding rootkits that could steal your personal login credentials, or any other type of surreptitious acts. Due to the nature of images on the cloud, any user can access publicly provided images to reap the benefits of the available resources, or face the consequences of there being a lack of security. By having a foundation of security within each of the virtual machine images, it improves protection of customer privacy and sensitive data by improving the security of the host machines.

Some security angles that can be looked at as described in this research are, publisher risks, customer risks, and administrator risks. The publisher is a user that creates a VMimage and posts it to the cloud, the customer accesses public images, and the administrator manages the security of the images hosted in their cloud. The publisher is concerned with confidentiality, the

customer faces safety or leakage problems, and the administrator tries to mitigate malware. The VMimages can have fully configured applications that affect all three types of users. For example, a publisher may have publicly published an image that has all of their bank information stored in their web browser. Later, an attacker, disguised as a customer, may visit that VMimage and access that personal information. Or another example is if an attacker disguised as a publisher posts an image with malware and rootkits, and a customer accesses an image they believe to be secure. Once the customer accesses that image, the attacker sees all of their keystrokes and actions. The IBM research team provides another probable example claiming that dormant images may have exploits that remain unpatched well after the exploit has been found. An attacker may later gain access into VMimage by exploiting a dormant vulnerability.

Although their proposed system "Mirage" does not provide complete security for images in the cloud, it does include several areas that must be looked at in developing an exploit mitigation system. We do not go into depth on any of their key features, however we do list them and briefly explain them. The first is that there must be access control that allows a user to share images. By having user identity authentication, images could be private, public, or protected images, which only grant access to those correct users. This can help protect customers from accidentally seeing other unrelated customers' information. Next, there must be filters that can automatically either hide or simply remove a user's personal data in an image. Next, there must be a regulation system that allows a user to upload or instantiate images. This could flag users who have previous history of posting malicious images, or keep records of who previously accessed an image and checked it in with malicious changes. The next feature is an image filter at publish and retrieval time that could scan an image for vulnerabilities and provide data to an administrator or publisher about the image. If the image has vulnerabilities, the

system could prevent the user from uploading or instantiating the image. The last key feature would be a set of repository maintenance services that automatically update and patch images as exploits are discovered as well as scan for malware.

These general ideas are a great starting point in developing a vulnerability assessment tool. They even provide benchmarks against a scanner clamAV, however we feel that this generalized cloud tool also includes assumptions that "Mirage" will work for every cloud service [19]. To remove generalities in this investigation, we have chosen a specific cloud system, OpenNebula. In section 3.2, we described the foundation of OpenNebula, metabrokering as a system, and in the following section we describe the implementation of our framework, namely *Cloud Vulnerability Assessment* (CVA).

4 Project Goals and Design

"If I had eight hours to chop down a tree, I'd spend the first six of them sharpening my axe." – Abraham Lincoln

4.1 Assumptions

The Cloud Vulnerability Assessment Tool operates upon several assumptions. First, the tool was designed specifically for OpenNebula in order to work within the existing cloud framework. However, the tool's CVA uses EC2 commands, meaning that the tool can be used with any cloud that supports the EC2 interface. This means that we assume the user has an EC2 account in order to use the commands. Our tool assumes several dependencies are met. Currently our tool works within the cloud front end. Also, our modules tool were developed and tested on Backtrack 5 R2, which is discussed in section 4.3.4. With the download, Backtrack OS contains both OpenVAS, discussed in section 4.3.2, and Metasploit, discussed in section 4.3.1. These two tools are absolutely necessary to operate the CVA. Therefore we assume that all dependencies are installed correctly. It is important that the security manager receives a report instantly via e-mail, therefore there must also be a mail server properly configured. In addition, we assume there is a properly installed persistent MySQL database to connect to on the cloud. In our methodology, we use a MySQL database, which will later be discussed in section 4.3.3. The final assumption is the existence of an image repository with images for the tool to scan.

4.2 Experimental Design

4.2.1 Prerequisites

We have developed a framework to automate our vulnerability assessment in the cloud. In figure 4.2.1, the diagram provides an abstract implementation of our framework. In this section, we

discuss the design of our Cloud Vulnerability Assessment framework. We have created a special user "SecurityAdmin" that has specific key pair encryption keys. The SecurityAdmin's public key is then placed into the module virtual machine, which allows the user to run a module from the Cloud Front End without user authentication. The Security manager also has special EC2 account credentials to run EC2 OpenNebula commands [31].

4.2.2 Cloud Vulnerability Assessment (CVA)

To start our framework, a SecurityAdmin only needs to specify a target address, which we pull from the instantiations of the images loaded, and at most, one other argument depending on the implementation on the different modules. Once the SecurityAdmin starts the framework, it first prompts the SecurityAdmin for a module to run as well as an image id of the module virtual machine. It then scans an image repository for all available targets and creates a list of target images. The next step is to load the module virtual machine and target machine 'y' to test. By having the public keys in the module VM, the SecurityAdmin has permission to send commands over the SSH tunnel. Once the target and module machines are running, the module begins scanning the IP address of the target machine with the specified module. After the module has completed its test, it dumps the data to a reported image storage virtual machine, and starts the next module on target machine 'y'. After each module on target 'y' has completed, target 'y' is subsequently shutdown, and the CVA framework loads target 'y + 1' for testing. The benefit of sending the vulnerability assessment output to a shared location, is that the information can be collaborated with other modules. This is later described in section 4.3.3 of the methodology section. It is important to mention that we heavily rely on a system like OpenNebula to develop and test our experimental design in order to produce real results from our implementation.

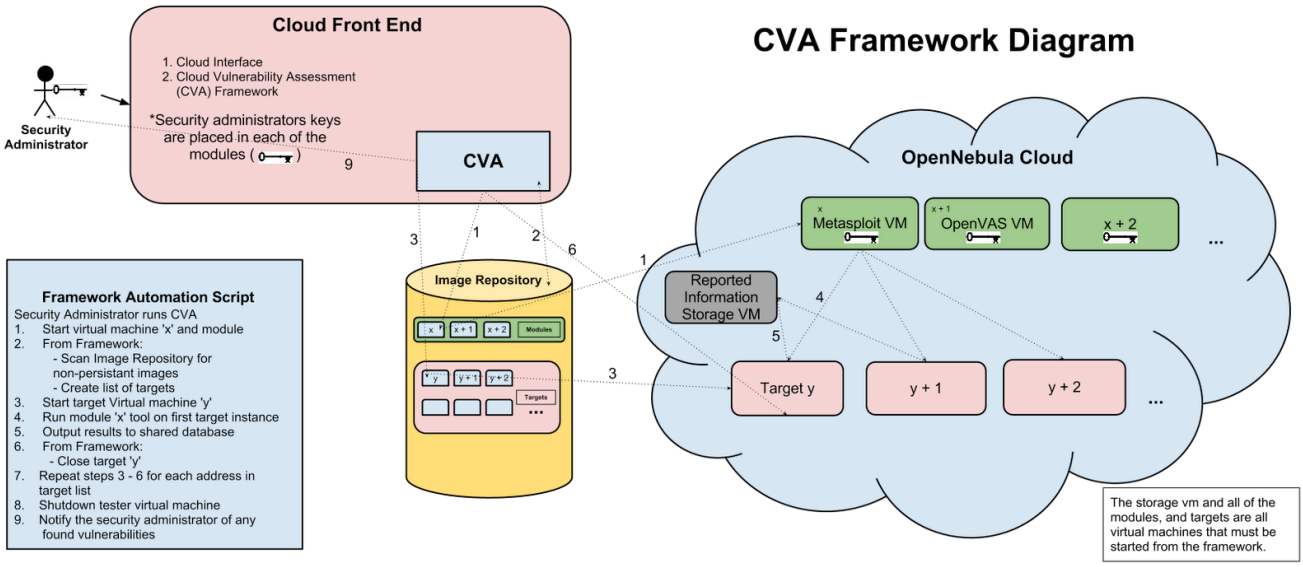

Figure 4.2.1

4.3 Methodology

Our experimental design let our modules in the cloud be very independent and flexible, meaning that we could choose essentially any combination of tools. Therefore, we were able to focus research on which tools were able to best maintain the security of images in the cloud rather than meet design constraints. Our primary objective was to include as many different tools as we could with the time allotted. Our goal was to discover different tools to evaluate in four separate categories, exploit tools, vulnerability assessment scanners, malware detectors and rootkit detectors. In the interest of implementing the best overall tool in the time available, we chose to determine the best exploit tool and vulnerability assessment scanner available. With the remaining time, we considered implementing the two other additional modules, namely malware and rootkit detectors. The Cloud Vulnerability Assessment tool and Reported Information

Storage VM were vital parts of the overall tool, leading us to allocate time specifically for their implementation as soon as the final design was set.

We selected Metasploit as our exploit tool. From our research, we determined that Metasploit was our best option, as penetration testers often use it for launching attacks. Metasploit exploits virtual machines within OpenNebula to best ensure they are secure from a wide variety of penetration attacks. This tool offered a wide array of functionality including scans to detect known vulnerabilities and various exploits to run on the target machine based upon the scan. We also chose OpenVAS in order to perform vulnerability assessment scans. There were many different types of vulnerability scanners available, but only a few were open source. We looked specifically at OpenVAS, but also considered NeXpose, an alternative vulnerability scanner. OpenVAS had an entirely separate development community from Metasploit and NeXpose, therefore utilizing it alongside Metasploit would result in the most ubiquitous results given two tools.

After selecting Metasploit and OpenVAS, we designed the framework as well as the necessary components as depicted in figure 4.2.1. The framework had to be modular, so that it could accommodate different types of tools and run them without conflicting with one another. Deciding how the framework would operate involved learning about the design of OpenNebula, which was one of the few constraints that we had to accommodate for. We determined that a virtual machine within the cloud was the best place to store and run our modules because our tools would have the most direct access to each target virtual machine. Due to this design choice, selecting an operating system on which to run our tool was necessary. Several different operating systems were considered, including a long-term support (LTS) version of Ubuntu Server and Backtrack OS. After considering both options, we chose Backtrack due to several

advantages it possessed. Backtrack is a Linux distribution specifically designed for security evaluation and comes with many different tools, including Metasploit and OpenVAS already installed. We will discuss Backtrack in further detail in section 4.3.4.

After running Metasploit and OpenVAS, reports are generated from the scans. The reports from these tools contain information from Metasploit about possible exploits and OpenVAS's potential vulnerabilities discovered.

A database was an ideal solution for storing the generated information. We decided that it was best to parse the information to eliminate useless text and consolidate the important results. MySQL was chosen primarily because of our team's prior knowledge of the syntax and language with regards to both building and using MySQL databases. Utilizing a database has the main advantage of allowing a user to query for reports on specific images within the cloud image repository, or by other useful sorting parameters. We will go into more detail about the database later in the Reported Information Storage VM, section 4.3.3.

4.3.1 Metasploit

Although Metasploit may appear as just a hacker tool, it is well recognized as an open source auditing tool that allows penetration testers to gather network information and test vulnerabilities in their network. One of the benefits of Metasploit is having penetration tools that quickly find targets and attack. Delving inside the entire Metasploit framework is out of the scope of this paper, however we will lightly gloss over its capabilities and go into detail about which aspects of Metaspoit we integrated into our system. Metasploit can gather information by port scanning, handle password sniffing, SNMP sweeps[3] that detail system specific information, consolidate gathered information in the Dradis framework[35], and even allows users to create their own scanners. Metasploit can do vulnerability scanning SMB login checks[2] that tell an attacker

where other valid passwords can be used, check for default passwords on VNC or Open_X11 servers, active WMAP web application scanning, and include vulnerability scans from third party frameworks like NeXpose[34] and Nessus[22]. The Metasploit framework can test for TFTP and IMAP fuzzers[32] against SQL Injection attacks, XSS, buffer overflows, and even directory traversal attacks. Metasploit includes documentation to easily extend the framework and create new exploits with useful API calls. These attacks along with many other client side exploits such as binary payloads, antivirus bypassers, java applet infections, or binary linux Trojans, are all available to a penetration tester. Once a tester has successfully opened a session from an exploit, they have more tools available to them that create privilege escalations, can screen capture sessions, packet sniff, create backdoors for re-explotation, and pivot to other machines in the network. Even more, Metasploit has extended usage that allows a user to automate browser attacks and target an untraditionally new operating system, MacOS X. Beyond there is more third party software such as Armitage [13] which provides a graphical user interface(GUI) for the Metasploit framework. This GUI is an alternative to the Metasploit msfconsole, or msfcli interfaces.

In previous versions of Metasploit, a service called db_autopwn existed, which automated the technical process of getting Metasploit working for attackers. The developers of Metasploit felt that the service did not belong in the core because it could launch many exploits at one target concurrently and crash them, instead of launching controlled attacks [12]. This service was the same module that fastrack, an extension of Metasploit used to automate penetration testing. This service has been removed from the latest versions of Metasploit as well as Backtrack 5 R1 and versions following. Metasploit developers claim that in the future, this automation process will be cleaner, well maintained and re-implemented back into the MSF core.

Due to this depreciation, we have used the Metasploit framework to develop our own module that incorporates the Metasploit framework service as a prototype until a new revision of the db_autopwn is developed and can be integrated into the CVA framework. Our framework uses the db_autopwn functionality provided as a upgraded plugin to the metasploit framework, maintained and provided by a user at github [14].

In figure 4.3.1, the Metasploit Module Diagram illustrates the design of the Metasploit Module that can be called from the CVA framework. In algorithm "sploit.py", it provides the implementation of the sploit module. The overall scheme of the module gathers information based upon the target address specified in the pullExploit() function, then tests the vulnerabilities in the testExploit() function, and last sends the information to the reported storage information virtual machine. The core of the module in the pullExploits() function first creates a resource file to pass to the msfconsole, to delete all previous host information. It then scans the target address to discover the operating system, and stores this computer information in the metasploit database. Next it runs the db_autopwn plugin to attempt to exploit the target by testing the machine against numerous known exploits that match the target's gathered information. The second time, a resource file is passed to the msfconsole in the testExploit() function. It uses the information stored from the db_autopwn to display the exploits that correlate with that target virtual machine. The sploit tool automates through each exploit that matches that target operating system, and will continue the same process for each target instance that is passed to the sploit module from the CVA framework. Once the tool has tested each exploit on a target, it will pass the output from the tool to the reported image storage virtual machine through a secure tunnel enabled by the key pair encryption keys.

To test this sploit module, Rapid7 has developed a "metasploitable" virtual machine that has several vulnerabilities [1]. We used this virtual image to test finding vulnerabilities and exploiting them. We opened sessions with each successful exploit and gracefully closed them. We verified each exploit against the known exploits in the provided metasploitable virtual image. To extend our testing of the sploit module, we tested this module on the cloud images, by creating instantiations of each public image and running the tool against different operating system types.

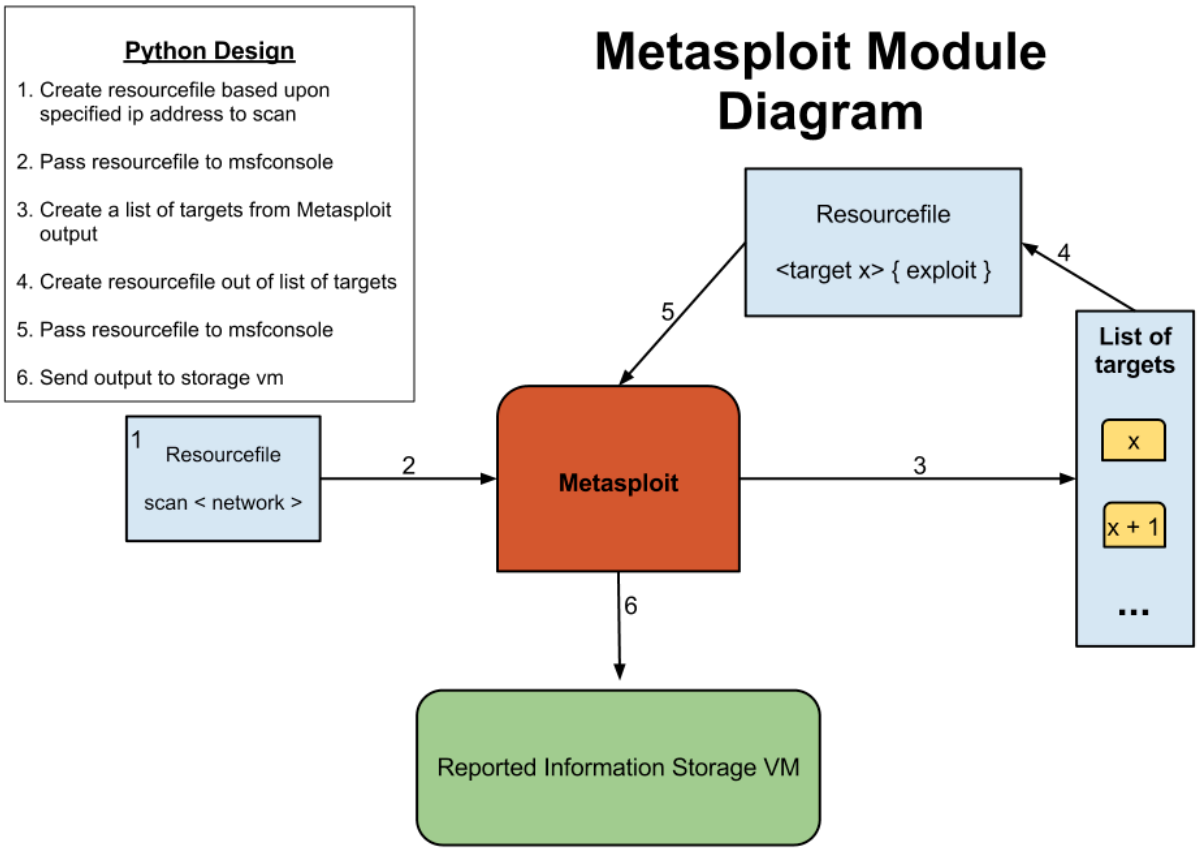

Figure 4.3.1

4.3.2 OpenVAS

OpenVAS is a versatile and powerful vulnerability scanner [27]. The tool was originally designed as an open-source vulnerability assessment tool called Nessus. At one point, Nessus no longer remained open-source at which point OpenVAS diverged from it in order to remain open source. OpenVAS has highly customizable scans, but comes with a set of preconfigured scans, which are adequate for most purposes. The version of OpenVAS used in our project is OpenVAS 4, as it is the latest stable release at this time. OpenVAS has many different modules, which work in conjunction to scan target machines.

OpenVAS [26]

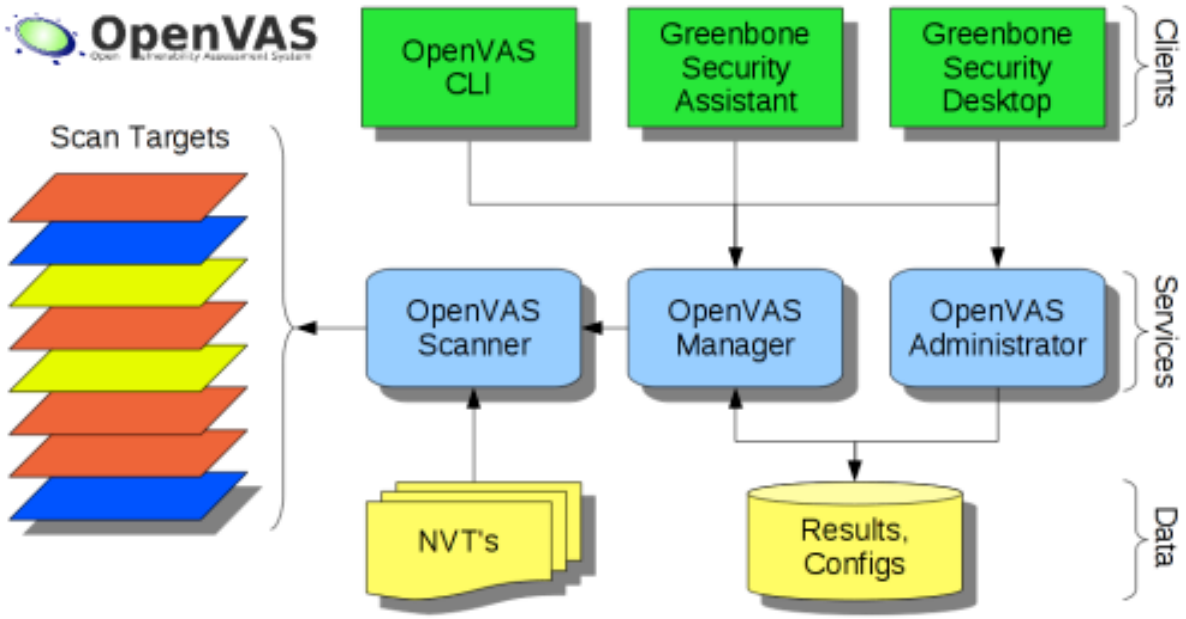

Figure 4.3.2.1

First there is the OpenVAS Scanner, which actually performs the scans. This scanner relies on Network Vulnerability Tests (NVT), or plugins. These plugins are loaded each time the scanner module is started. OpenVAS remains up to date via these plugins, each of which help

define how scans are performed. The scanner is capable of scanning just one target or entire IP ranges at once. The OpenVAS manager sits atop the scanner in the framework. The manager is more or less a black box module, which the user usually does not need to understand but does need to know how to use. The manager handles omp from the OpenVAS CLI as well as direct XML commands from all three of the available clients, the OpenVAS CLI, the Greenbone Security Assistant, and the Greenbone Security Desktop. The manager is the difference between a vulnerability scanner and a vulnerability manager because it adds both functionality and intelligence to the system. The manager directly runs the OpenVAS Scanner using the OpenVAS Transfer Protocol (OTP).

In addition there is the OpenVAS Administrator, which is primarily designed to be a command line tool using OpenVAS Administrator Protocol (OAP). The primary function is managing users, such as determining user privileges within the OpenVAS framework. A user must be created along with installing the tool, in order to make managing it possible. Both the manager and administrator connect with the database where preset configurations and old scan results are stored. The OpenVAS scanner, manager and administrator modules are created from code within the shared OpenVAS Libraries.

Integrating OpenVAS into our framework, and therefore into our experimental design, is a matter of properly scripting it. In order to do this, a good knowledge of the program is required. Little documentation pertaining to CLI automation existed. We first installed the program on a VM in order to test the functionality. This involved running commands provided on the OpenVAS website. Directing OpenVAS via XML rather than simply the short list of commands given by the help and man pages was essential to automating OpenVAS scans.

The first step in the script was to handle input parameters. OpenVAS operates by scanning based on IPs that the user provides as input. Therefore the first parameter to the script had to be a target IP to scan. As mentioned before, OpenVAS also can perform several different types of scans. The second input is an integer parameter representing different types of scans each of which has certain tradeoffs. Certain scans are more thorough, but take longer while others operate more quickly and may be more well suited for situations where cloud resources are limited or there is a high number of virtual machines. There is also an optional third parameter, which simply activates the verbose functionality, giving more details into what is happening when the script is run. This option may be particularly useful in testing situations, given that all of the outputs from the OMP commands are provided to the user for inspection. The final option is the help flag to guide the user.

The script starts by examining the input. If the script finds any sort of help flag, the help function runs and the execution terminates there. The input IP is parsed in order to determine whether or not it is valid. By splitting the possible range extension off, the function checks that each octet is a valid number itself before checking the extension. Another function checks that the scan value also acceptable. Finally if a third flag is given it is checked to see if it is the proper flag to run with verbose output. If so, extra information is automatically printed throughout the script.

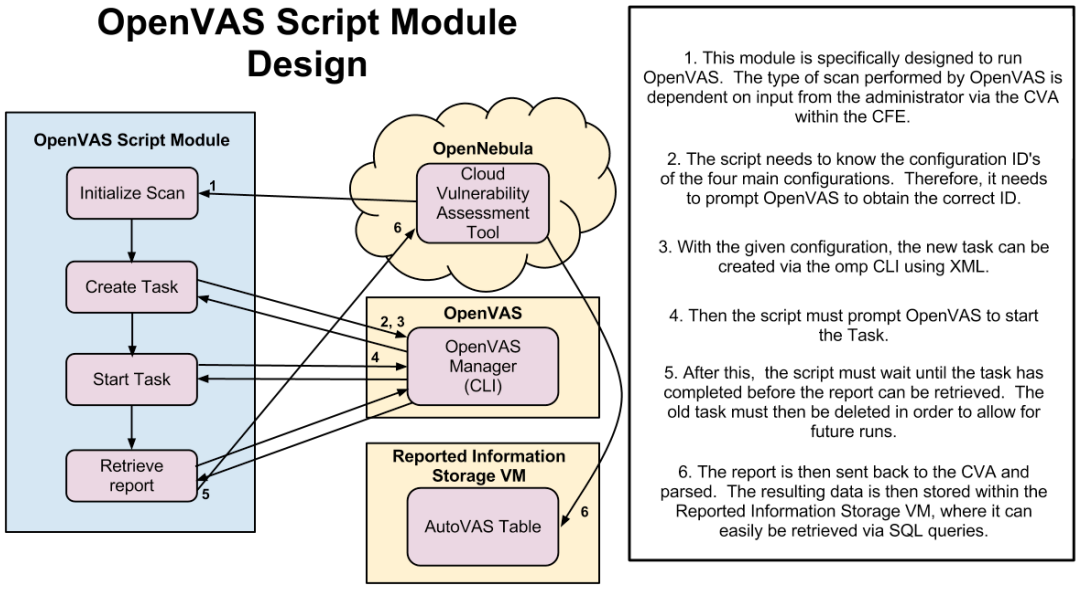

Figure 4.3.2.2

As seen in the diagram above, the script communicates with three separate other agents. First, the script initializes certain variables by confirming with the cloud. After that, the task is created and started via omp and the OpenVAS manager. After that the script prompts OpenVAS for a text report. Finally the report is parsed and the important results are stored into the Reported Information Storage VM. This problem involves a lot of interaction with the manager to work. One of the main challenges of automating OpenVAS was extracting information from the OpenVAS manager. The script has to consistently keep track of several different IDs in order to properly run. Furthermore, most of this information must be parsed out of the output using various string commands in Python. Then these stored values become input later in the script to perform various other functions.

Our OpenVAS module script was written in Python for two reasons. Firstly, Python offered a very easy way to manipulate strings. Secondly, our team decided that Python would be

the scripting language for all scripts we wrote in order to keep the entire tool easy to understand and reduce dependencies. The script interfaces indirectly with the OpenVAS manager using omp via bash. Since the script relies heavily on running bash commands from Python, we developed two different functions for this. The first function was created to run a single command, meaning that the command had to be parsed and executed using the Python subprocess module. Then the output had to be handled correctly in order to prevent deadlocks and the result returned. The second function for bash commands was designed to take two bash commands, piping the output of the first as the input to the second. This was especially helpful for using the grep command in order to sort output.

To initialize before running the primary parts of the script, the script has to actually prompt OpenVAS for the configuration ID given the input. Once that is done, the script can begin by creating a target with the given IP argument. Then a task must be created using the target along with the configuration the script already has. A task is essentially an instance of a scan, but it does not run. When you run the task you create a scan, which creates a unique report each time. One task can produce any number of reports over time. The script then proceeds to run the scan and wait for it to finish. The script must wait because the report is not fully generated until the scan has finished. Once it has, the script prompts for the report, parses it, and outputs the results to the database.

4.3.3 Reported Information Storage VM

We decided to choose a separate VM specifically to store vulnerability reports that the security manager could easily access. This model allows the security manager to separately access the reports without having to run the specific module. Much like we delegated different roles within the framework to each of the other modules, this VM fulfills a very specific functionality within

the CVA. Having a separate VM for the database also allows for it to be persistent, unlike our module and target VM's. This persistence allows the CVA to maintain a collection of past reports and allows the user to examine vulnerability trends. With a non-persistent database VM, all data would be lost every time the VM every time an image was instantiated and shut down.

Our Reported Information Storage VM runs a MySQL Version 14.14 Distribution 5.1.61 for Debian-Linux-gnu (x86_64). This server runs within Debian 6.0.4, an open source Linux distribution. At this time, both are the latest versions available.

We chose MySQL because it is a well-developed database language. MySQL allows us to execute simple queries in order to retrieve data on previous reports. In addition, it allowed us to divide our data within the desired schema. We initially stored the report information as files back to the CVA, but having a separate database offers many advantages, such as:

(i) organize data by creating tables, specifying their columns and types,
(ii) very inexpensive and cost effective operations, and
(iii) joins to easily retrieve specific preexisting data.

By comparison, files are difficult to manage and often store redundant data. In addition files are difficult to navigate, which adds to the time to assess the reported vulnerabilities.

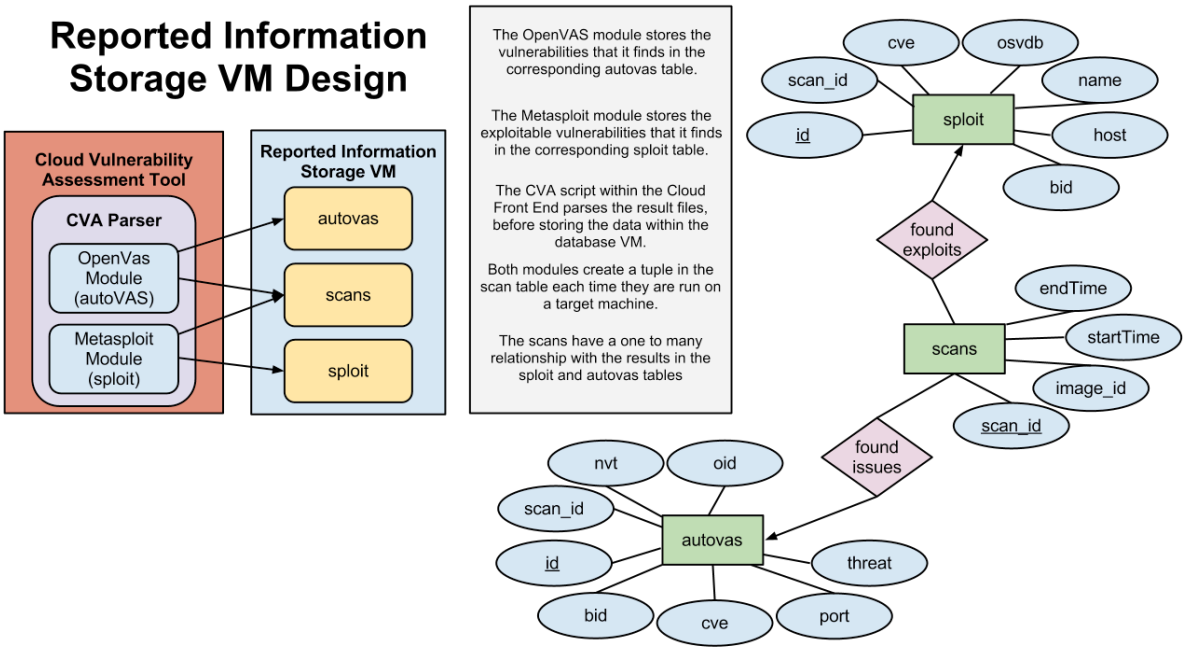

Figure 4.3.3.1

The figure above, Figure 4.3.3.1, illustrates our database design. The individual modules do not directly interface with the database VM. Whenever the CVA script runs a module, the module produces a results file. The CVA retrieves this file and parses it in order to extract important information. The CVA inserts this information into the proper tables within the database VM. The CVA script also ensures that the different tables correspond to one another in order to make joins easier for the user. In the ERD model above, a more in depth depiction of the database is provided. All three tables contain a scan_id field, meaning that the results in both the sploit and autovas tables can be joined to their respective scans. The database design consists of separate tables for each module, as seen in figure 4.3.3.2. The database is configured this way because the two modules produce different outputs upon completion of their scans. By having a separate scans table, the database becomes modular, by allowing the addition of new tables for new tools provided they contain the scan_id attribute.

The tables in the diagram below are built specifically for each module. The scan table contains general information in regards to a single scan. The attributes contained in the scan table are pieces of information, such as the virtual image id, which can be associated with a scan run by any module. The sploit table contains different fields relating to exploit information provided by the sploit module. This module produces information such as, 'id', 'host', 'cve', 'osvdb', 'bid', 'name', and 'scan_id'. Similarly, the autovas table contains the necessary attributes to accurately summarize an OpenVAS scan. It contains 'id', 'scan_id', 'nvt', 'oid', 'threat', 'port', 'cve', and 'bid'. The 'cve' attribute is a common vulnerability and exposure. The 'nvt' field stands for network vulnerability tool, while 'osvdb' stands for open source vulnerability database. The 'oid' and 'bid' fields are both standard references to vulnerabilities and exploits. The 'cve' field is in both module tables, meaning it is the best way to draw comparisons between the two tools. Both of the module tables use their id field as their primary attribute, while the scan table uses the scan_id field.

```
mysql> show tables;
+----------------+
| Tables_in_CVAdb |
+----------------+
| autovas        |
| scans          |
| sploit         |
+----------------+
3 rows in set (0.00 sec)

mysql> describe scans;
+-----------+-------------+------+-----+---------+----------------+
| Field     | Type        | Null | Key | Default | Extra          |
+-----------+-------------+------+-----+---------+----------------+
| scan_id   | int(11)     | NO   | PRI | NULL    | auto_increment |
| image_id  | varchar(20) | YES  |     | NULL    |                |
| startTime | varchar(30) | YES  |     | NULL    |                |
| endTime   | varchar(30) | YES  |     | NULL    |                |
+-----------+-------------+------+-----+---------+----------------+
4 rows in set (0.01 sec)

mysql> describe sploit;
+---------+--------------+------+-----+---------+----------------+
| Field   | Type         | Null | Key | Default | Extra          |
+---------+--------------+------+-----+---------+----------------+
| id      | int(11)      | NO   | PRI | NULL    | auto_increment |
| host    | varchar(20)  | YES  |     | NULL    |                |
| cve     | varchar(40)  | YES  |     | NULL    |                |
| osvdb   | varchar(40)  | YES  |     | NULL    |                |
| bid     | varchar(40)  | YES  |     | NULL    |                |
| name    | varchar(100) | YES  |     | NULL    |                |
| scan_id | int(11)      | YES  |     | NULL    |                |
+---------+--------------+------+-----+---------+----------------+
7 rows in set (0.00 sec)

mysql> describe autovas;
+---------+--------------+------+-----+---------+----------------+
| Field   | Type         | Null | Key | Default | Extra          |
+---------+--------------+------+-----+---------+----------------+
| id      | int(11)      | NO   | PRI | NULL    | auto_increment |
| scan_id | int(11)      | YES  |     | NULL    |                |
| nvt     | varchar(150) | YES  |     | NULL    |                |
| oid     | varchar(60)  | YES  |     | NULL    |                |
| threat  | varchar(40)  | YES  |     | NULL    |                |
| port    | varchar(40)  | YES  |     | NULL    |                |
| cve     | varchar(40)  | YES  |     | NULL    |                |
| bid     | varchar(40)  | YES  |     | NULL    |                |
+---------+--------------+------+-----+---------+----------------+
8 rows in set (0.00 sec)

mysql>
```

Figure 4.3.3.2

4.3.4 Backtrack

"The quieter you are, the more you are able to hear" - Backtrack

As a part of our experimental design, we load the module VM that contains our modules such as sploit and autoVAS. In our methodology, we can easily create these tools based upon existing frameworks such as Metasploit and OpenVAS. Developing this framework has been simplified again for us, by choosing the right operating system (OS) to develop and test on. We now take a step even further by introducing the operating system Backtrack [30] which contains all of our modules, why we have decided to use this OS, and how it is an ideal penetration testing tool.

Backtrack is an open source Linux based operating system designed specifically for penetration testing funded by Offensive Security [36]. It was developed to be the perfect tool for security professionals, and so it contains numerous scripts, penetration testing programs, and configurations to simplify offensive security. Some of the important tools they include in the OS are Metasploit and OpenVAS. In Backtrack 5 R2, both the latest versions of Metasploit and OpenVAS are install, and are already preconfigured with the right specifications to quickly get up and running. This has dramatically cut our developed process, because it allowed us to test the tools in order to decide which ones we wanted to use and tie it into our framework. As opposed to manually installing each tool, we use a more generic maintained OS, which contains the tools. OpenVAS and Metasploit have documentation provided online, however the level of abstraction Backtrack has provided by automatically configuring these tools almost eliminates the setup process. We then simply had to use the tool and examine each tool that we wanted to incorporate into our system. By using Backtrack, it also allows others to easily duplicate our work by installing one version of Backtrack, instead of one version of multiple tools.

We initially looked at another operating system, Ubuntu Server [33], in hopes of developing a lightweight OS that would merely have our tools and their dependencies, but the preconfigured tools on Backtrack outweighed this idea. We consider this as a future work, and will later discuss this in our conclusion.

4.3.5 Other Tools

In addition to OpenVAS and Metasploit, several other tools were considered to be part of the CVA. Specifically two types of tools were considered important. A virus scanner was one of the potential modules, which would augment the vulnerability scanner and exploit tool. In particular, we considered ClamAV [19], an open source antivirus software. Also, a rootkit detector is vital to maintaining security and several were considered. These tools are of particular importance because if a machine is already compromised, then doing vulnerability scanning and running exploits is futile. Given that time did not permit the implementation of these modules, this topic will be further discussed in the Further Work section 6.1.

4.4 Testing Environment

To test the CVA framework, we have implemented our experimental design directly into the cloud front end of SZTAKI's OpenNebula system. In the image repository, there are several different images with different operating systems; so we scan each public image as an instance and report the module feedback. The systems we tested are detailed in figure 4.4.1. While testing, we complete two types of test, a functional test, and a performance test as our primary control. Functional test: we put a vulnerable image into the Image Repository and the tool can find all of its known vulnerabilities. Performance test: time cost for the assessment (VM -> avg,

min, max; whole Image Repo). Our variable is the modules that are run during both tests. The test sequence is as follows:

1. Functional test: Sploit Module
2. Functional test: AutoVAS Module
3. Functional test: Sploit and AutoVAS Module
4. Performance test: Sploit Module
5. Performance test: AutoVAS Module
6. Performance test: Sploit and AutoVAS Module

To simulate a real working environment, other instances of images that are run by other users are not shut down. To prevent contamination and duplicating test reports on one image, we have removed all running instances of images under the security administrator's account prior to testing.

Chart of Virtual Machines Tested

ID : 169 NAME : debian 6.0 TYPE : OS	ID : 168 NAME : CVA_DB TYPE : OS	ID : 166 NAME : edgi-testwms TYPE : OS
ID : 165 NAME : edgi-testvoms TYPE : OS	ID : 164 NAME : edgi-testui TYPE : OS	ID : 163 NAME : edgi-testboinc TYPE : OS
ID : 146 NAME : Metasploitable TYPE : OS	ID : 143 NAME : BackTrack 5r2 TYPE : OS	ID : 141 NAME : Volatile-Datablock-16GB TYPE : DATABLOCK
ID : 136 NAME : WinXP-Base TYPE : OS	ID : 130 NAME : SPEQULOS_BOINC TYPE : OS	ID : 99 NAME : bridge_w_metajob TYPE : OS
ID : 111 NAME : SPEQULOS TYPE : OS	ID : 66 NAME : M3S TYPE : OS	ID : 65 NAME : SALMon TYPE : OS
ID : 64 NAME : SL5_EMIUI_gUSE3.4_VirtIO TYPE : OS	ID : 62 NAME : salmon_v2 TYPE : OS	ID : 61 NAME : 3.4 gUSE (SL5+gLite3.2+VirtIO) TYPE : OS
ID : 58 NAME : g34HG1-backup TYPE : OS	ID : 53 NAME : 3.4 gUSE (SL5+gLite3.2) TYPE : OS	ID : 52 NAME : SL5_EMIUI_gUSE3.4 TYPE : OS
ID : 45 NAME : Base-SL5 TYPE : OS	ID : 44 NAME : DGSG3GBR_v5 TYPE : DATABLOCK	ID : 37 NAME : SL5-Base TYPE : OS
ID : 26 NAME : gUSE3.3.3-TEST TYPE : OS	ID : 24 NAME : debian_6_tud_nap_2 TYPE : OS	ID : 22 NAME : debian_6_tud_nap TYPE : OS
ID : 20 NAME : EMI-UI TYPE : OS	ID : 19 NAME : ec2-9b74ef20-c272-012e-e2f4-001636d2ed43 TYPE : OS	ID : 17 NAME : debian_6_robinak TYPE : OS
ID : 16 NAME : debian_6 TYPE : OS		

Figure 4.4.1

5 Results and Discussion

5.1 AutoVAS Module

The AutoVAS module automatically runs OpenVAS on target machines. For test purposes, we ran AutoVAS on all of the available public images. At the time we ran the test there were 32 publically available images. The test took 4:17:02 to scan all 32 of the accessible public images. Out of all the scans, the longest lasted 20:51, while the shortest lasted only 54 seconds. Overall, the average scan time was 8:02. This module found 560 issues and stored them into the database. These issues are classified into four different categories based upon the threat posed by the vulnerabilities, high, medium, low, and log. A majority of the issues found are classified as log, accounting for 316 of the stored results. Log issues are largely unimportant, but occasionally do have a CVE or BID numbers describing the issue further in depth. In addition, there were 184 results, which were classified as low, 36 as medium, and 24 as high. Often, medium and high vulnerabilities also had corresponding CVE and BID numbers, which describe the issue along with possible fixes. These are the most important for the security administrator to examine and address. Some scans revealed no results, while one found 94 possible vulnerabilities.

5.2 Sploit Module

The Sploit module attempts to find exploits in a target machine and open sessions to expose weaknesses. Therefore, this module should return far less results because it is much harder to open sessions in remote machines than to find smaller vulnerabilities. Our test took 2:08:26 to complete, with an average scan time of 4:49. Out of all 32 public images scanned, only one

session was opened, via a vulnerability allowing injection of PHP code. This result is important because it becomes much more difficult for an administrator to defend other VM's within the cloud when an attacker uses a hijacked machine as an attack vector.

5.3 Both Modules

Running both modules at once allows for comparisons of the data to be made, as well as ensure the implementation for running multiple modules is correct. Two tests were conducted in order to assess the functionality and performance,

1. running the Sploit module followed by the AutoVAS module on each target, and
2. running AutoVAS followed by Sploit on each target.

5.3.1 Test 1: Sploit then AutoVAS

This test took a total time of 7:21:13 to complete, with an average of 6:54 per scan. The Sploit module opened 2 sessions, one more than the prior test where it was run without AutoVAS. The AutoVAS module found 719 vulnerability issues, meaning that it was far more effective when run in conjunction with the Sploit module. In particular, the AutoVAS module found more results from each of the four classifications, high, medium, low and log. The AutoVAS scans were much more effective in terms of results than during the individual test, but also took more time to complete.

5.3.2 Test 2: AutoVAS then Sploit

The second test took 6:43:04 to run, averaging 6:18 per scan. As with the first test utilizing both modules, the Sploit module found two exploits opening a session with both. The AutoVAS module found a total of 648 vulnerabilities, less than when it was run after Sploit but still more

than when run alone. Similarly to the prior test, the AutoVAS module took more time to complete.

5.4 Results Graphs

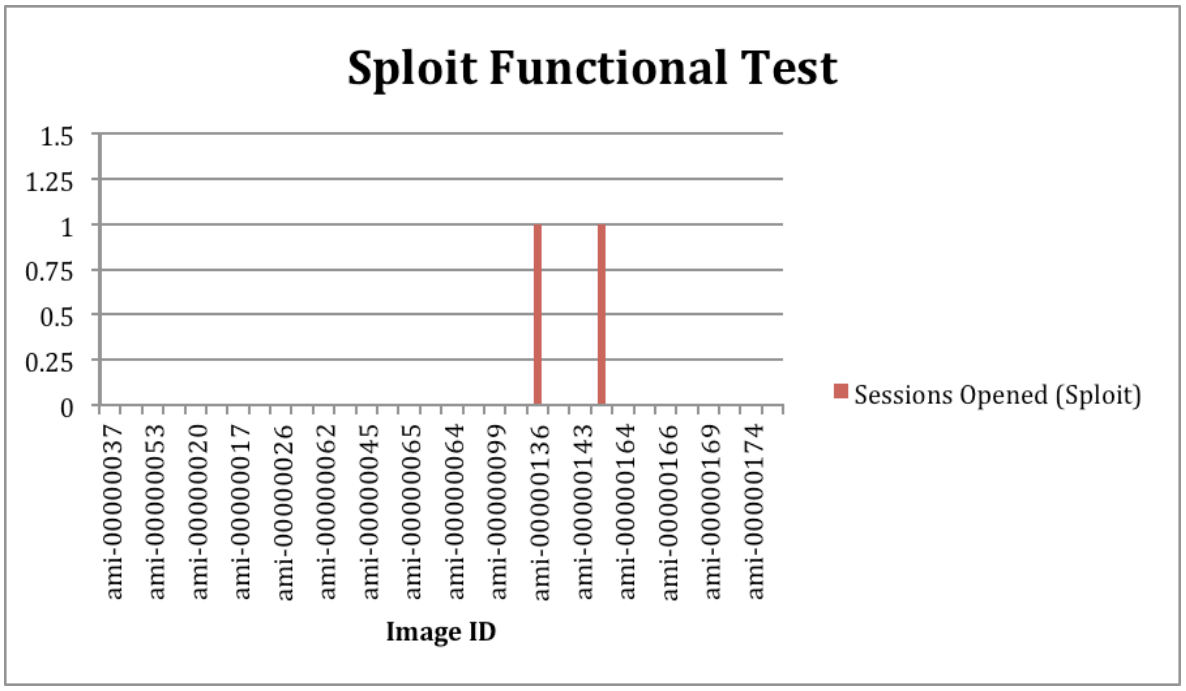

Figure 5.4.1

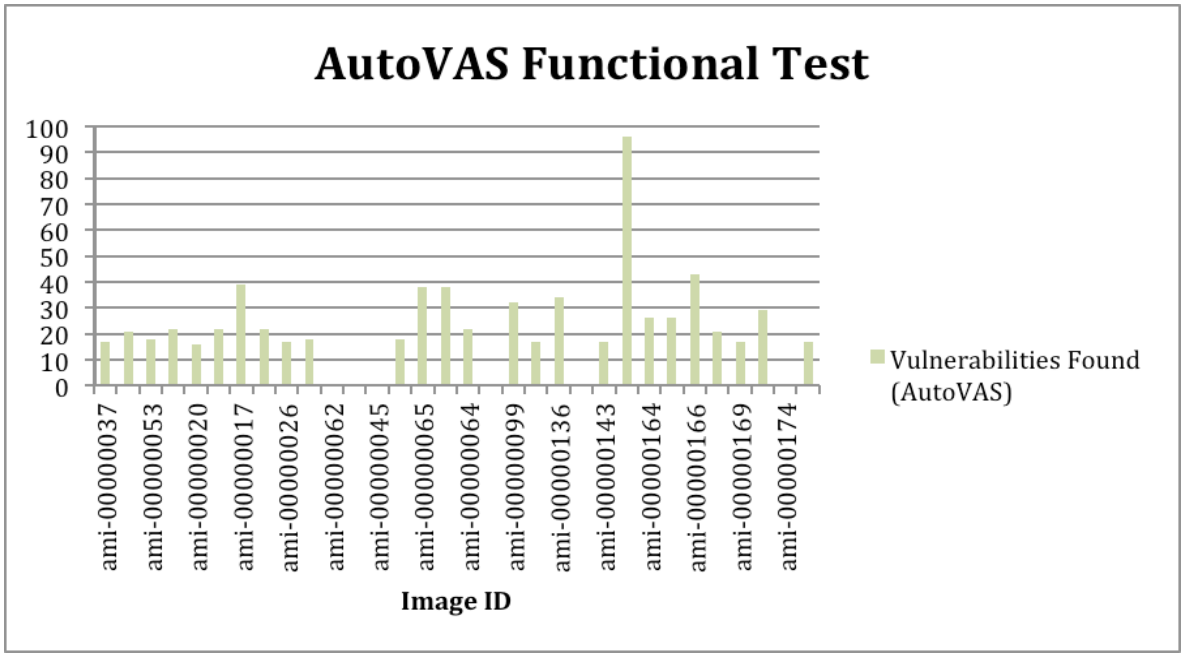

Figure 5.4.2

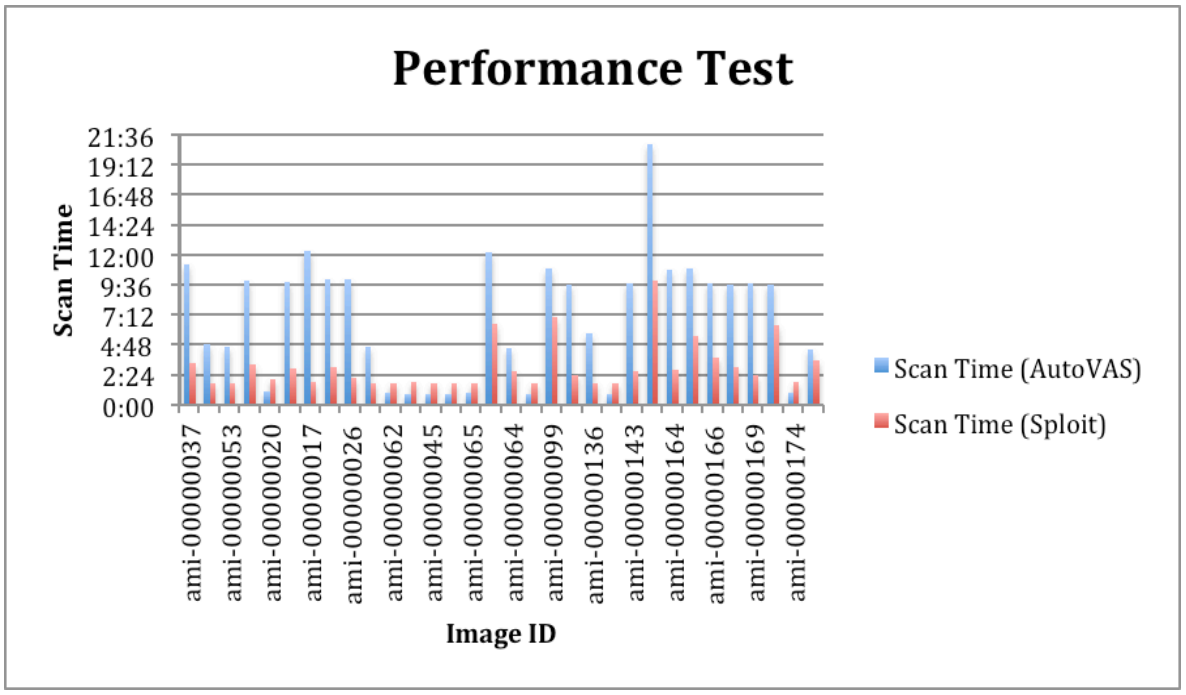

Figure 5.4.3

6 Conclusion

6.1 Cloud Vulnerability Assessment

We have developed the opensource 'cvaFrame' framework as a tool that cloud administrators can use to discover vulnerabilities and exploits among their virtual images. Our intention in developing this framework was to prove that such a system can work in the cloud. Similar to the "Mirage" system proposed by the IBM T.J Watson Research team, our framework can be managed by an administrator in the cloud and provide protection to both parties. An advantage to the 'cvaFrame' framework is that it is more than merely a proposal. Our opensource project is a great starting point for other researchers, users, cloud administrators, and many others to protect their cloud system. Each user has full capabilities in extending this framework, or manipulating it towards there needs. At the time of this research, there were no known tools that provided vulnerability assessment in the cloud and could specifically test dormant public images in the image repository. We believe we are the first to investigate cloud vulnerability assessment in the cloud and also the first to develop an opensource framework that discovers these vulnerabilities and exploits among dormant images. In section 6.2, our future work sections details how we can extend this framework even more, which can in turn provide a more robust vulnerability assessment report.

6.2 Further Work

Each topic listed in this section are all key features that can extend the ' cvaFrame' framework. These features exist in the future work section due to our 6 week research limitation. Each extension to our framework provides an equal level of contribution to the framework so we do

not mention these extensions in any particular order. We feel that each of these extensions should exist in the core of the framework, and other proposed extensions such as the features mentioned in the "Mirage" system would exist as plugins to this framework. We highly recommend using this framework as a stepping stone towards automating vulnerability image assessment in the cloud.

6.2.1 Automatic Exploit Patches

Currently our framework is expected to send a report to the security manager assessing the cloud; however we believe that our system could be extended to automatically patch vulnerabilities and exploits discovered by other modules. Each module was designed to share the same shared storage virtual machine so this module, namely "patch". It could update dormant images based upon information in our database if there are no default update tools on the image, or by running the default patch system that exists on the operating. It would load the image, update the machine, and store over the persistent image the updated image so that users who accessed the public image would use an updated instance.

6.2.2 More Modules

Two specific modules we felt were necessary to include into the 'cvaFrame' core are the 'rootkit' and 'patch' modules. The 'rootkit' module would detect rootkits that are surreptitiously installed on the machine. We looked at a few available rootkit detectors, namely Sysinternals, Tripwire, DumpSec, AIDE, and HijackThis. Unfortunately the only tool that fits our framework is AIDE (Advanced Intrusion Detection Environment), because it is opensource and works natively on all platforms. Sectools claims that AIDE is a free replacement for Tripwire which makes cryptographic hashes of important system files and stores them in a database [23]. Based

upon information stored in the database, it can detect rootkits by analyzing changes in those important files. The 'patch' module as seen in section 6.2.1 would likewise be a module that managed updates.

6.2.3 Encryption and Key Management

The csa guide provides several recommendations in regards to encryption and key management [20]. The most relevant to this research are,

(i) alternative approaches to encryption,
(ii) cryptography in cloud deployments,
(iii) encryption in cloud databases,
(iv) key management in the cloud, and
(v) safe-guarding keys.

In our framework we use a public key infrastructure (PKI) to allow only the security administrator to access the modules, however to secure a data collection, we feel it is necessary to explore options available concerning the five noted csa guide suggestions.

6.2.4 cvaFrame Privledges

Throughout our experiment, we assume that our framework is run from the security administrator viewpoint. We make this assumption because our framework has potential to reveal security issues that can only a security administrator should know. If this information were to be leaked to the wrong user, an entire system can be directly compromised, undermining the purpose of this framework. We therefore propose extending this framework to account for roles. There can be a security administrator role which has full access, and a user role which has limited access. This way, a user may have a limited number of scans allotted to test all of their images for vulnerabilities and exploits instead of waiting for the security manager to report to the image owner.

References

[1] Aharoni, Mati, William Coppola, and et al. "Metasploitable." Metasploit Unleashed. Rapid7, n.d. Web. 10 Apr 2012. <http://www.offensive-security.com/metasploit-unleashed/Metasploitable>.

[2] Aharoni, Mati, William Coppola, and et al. "SMB Login Check." Metasploit Unleashed. Offensve-Security, n.d. Web. 10 Apr 2012. <http://www.offensive-security.com/metasploit-unleashed/SMB_Login_Check>.

[3] Aharoni, Mati, William Coppola, and et al. "SNMP Sweeping." Metasploit Unleashed. Offensive-Security, n.d. Web. 10 Apr 2012. <http://www.offensive-security.com/metasploit-unleashed/SNMP_Sweeping>.

[4] Andress, Mandy (2002). Vulnerability-assessment services on the rise. Retrieved 31 March 2012 from http://www.networkworld.com/reviews/2002/0204bgside.html

[5] B., Grobauer,, T., Walloschek, and E., Stocker, (2010). Understanding Cloud Computing Vulnerabilities. Retrieved 31 March 2012 from IEEE: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5487489&tag=1

[6] Forristal, Jeff and Shipley, Greg (2001). Vulnerability Assessment Scanners. Retrieved 31 March 2012 from http://www.networkcomputing.com/1201/1201f1b1.html

[7] Frankston, Robert M. (1974). THE COMPUTER UTILITY AS A MARKETPLACE FOR COMPUTER SERVICES. Retrieved 31 March 2012 from http://www.frankston.com/public/?name=TR128

[8] Geelan, Jeremy, Rutsky, Ken, White, Elizabeth, Llorente, Ignacio M. and O'Gara, Maureen (2012). Vulnerability Assessment with Application Security. Retrieved 31 March 2012 fromhttp://cloudcomputing.sys-con.com/node/2149564

[9] Kertész, Attila and Kacsuk, Péter (July 17, 2009). GMBS: A new middleware service for making grids interoperable. Retrieved 1 April 2012 from http://www.sciencedirect.com/science/article/pii/S0167739X09001575

[10] Li, Huan-Chung, Liang, Po-Huei, Yang, Jiann-Min and Chen, Shiang-Jiun (2011). Analysis on Cloud-Based Security Vulnerability Assessment. Retrieved 31 March 2009 from

IEEE:http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=5704304&abstractAccess=no&userType=inst

[11] Mahach, Roger (). Vulnerability Assessment through the Cloud: A Case Study. Retrieved 31 March 2012 from Office of the Comptroller of the
Currency: http://www.gtra.org/component/attachments/download/763

[12] Moore, HD. "Six Ways to Automate Metasploit." Security Street. Rapid7, n.d. Web. 10 Apr 2012. <https://community.rapid7.com/community/metasploit/blog/2011/12/08/six-ways-to-automate-metasploit>.

[13] Mudge, Raphael. "Armitage." Cyber Attack Management for Metasploit. N.p., n.d. Web. 10 Apr 2012. <http://www.fastandeasyhacking.com/>.

[14] Neinwechter, . "db_autopwn plugin." . Rapid7, n.d. Web. 10 Apr 2012. <https://github.com/neinwechter/metasploit-framework>.

[15] Soll, David F. (2011). Cloud Computing A General State of the Union. Retrieved 31 March 2012 from Omicron Consulting: http://princetonacm.acm.org/downloads/Cloud_Computing.pdf

[16] Subramanian, Krishnan (2009). Vulnerability Scanning And Cloud Computing. Retrieved 31 March
2012 from http://www.cloudave.com/1917/vulnerability-scanning-and-cloud-computing/

[17] T.M., King, and A.S., Ganti, (2010). Migrating Autonomic Self-Testing to the Cloud. Retrieved 31 March 2012 from
IEEE: http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=5463688

[18] Wei, Jinpeng, Zhang, Xiaolan, Ammons, Glenn, Bala, Vasanth and Ning, Peng (). Managing Security of Virtual Machine Images in a Cloud Environment. Retrieved 31 March 2012 from http://users.cis.fiu.edu/~weijp/Jinpeng_Homepage_files/ccsw09.pdf

[19] ClamAV. About ClamAV. Retrieved 10 April 2012 from Sourcefire Inc.:http://www.clamav.net/lang/en/

[20] Cloud Security Alliance ed.. Security Guidance For Critical Areas Of Focus In Cloud Computing V3.0. CSA, 2011. Web 2 April 2012. <https://cloudsecurityalliance.org/guidance/csaguide.v3.0.pdf>

[21] Cloud Taxonomy. Retrieved 1 April 2012
from http://www.opencrowd.com/assets/images/views/views_cloud-tax-lrg.png

[22] Nessus. Tenable Network Security, n.d. Web. 10 Apr 2012.
<http://www.tenable.com/products/nessus>.

[23] Nmap Security Scanner Project (). SecTools.Org: Top Network Security Tools. Retrieved 31 March 2012 from Nmap Security Scanner Project: http://www.sectools.org

[24] OpenNebula (). About the OpenNebula.org Project. Retrieved 31 March 2012 from OpenNebula:http://opennebula.org/about:about

[25] OpenNebula (). Why OpenNebula? Retrieved 2 April 2012 from OpenNebula:http://opennebula.org/about:why

[26] OpenVAS (). OpenVAS Framework. Retrieved 31 March 1990 from OpenVAS:http://www.openvas.org/about.html

[27] OpenVAS (). About OpenVAS Software. Retrieved 31 March 2012 from OpenVAS:http://www.openvas.org/software.html

[28] Rapid 7 (). Metasploit Framework. Retrieved 31 March 2012 from Rapid 7:http://www.metasploit.com/

[29] Rapid 7 (2012). Metasploit Framework User Guide. Retrieved 31 March 2012 from Rapid 7:https://community.rapid7.com/docs/DOC-1751

[30] "Backtrack Linux-Penetration Testing Distribution.". Backtrack-Linux, n.d. Web. 10 Apr 2012. <http://www.backtrack-linux.org/>.

[31] "Scaling out Computing Clusters to EC2." . OpenNebula, n.d. Web. 10 Apr 2012.
<http://opennebula.org/documentation:uc3>.

[32] "The Open Web Application Security Project." Fuzzing. OWASP, n.d. Web. 10 Apr 2012.
<https://www.owasp.org/index.php/Fuzzing>.

[33] "Ubuntu For You." . Canonical Ltd., n.d. Web. 10 Apr 2012.
<http://www.ubuntu.com/ubuntu>.

[34] "Vulnerability Management." Rapid7, n.d. Web. 10 Apr 2012. <http://www.rapid7.com/products/vulnerability-management.jsp>.

[35] "What is Dradis?." Dradis Framework. Security Roots, n.d. Web. 10 Apr 2012. <http://dradisframework.org/>.

[36] "World Class Information Training and Penetration Testing." Offensive Security Training and Services. Offensive-Security, n.d. Web. 10 Apr 2012. <http://www.offensive-security.com/>.

Appendix

Example Database Output

Sploit Table

id	host	cve	osvdb	bid	name	scan_id
1	host=192.168.143.178	CVE-2008-4250	OSVDB-49243	MSB-MS08-067	name=exploit/windows/smb/ms08_067_netapi	41
2	host=192.168.143.182	CVE-2007-5423	OSVDB-40478	BID-26006	name=exploit/unix/webapp/tikiwiki_graph_formula_exec	47

Scan Table

scan_id	image_id	startTime	endTime
1	ami-00000037	20120419132854	20120419133223
2	ami-00000037	20120419132854	20120419134359
3	ami-00000052	20120419134516	20120419134757
4	ami-00000052	20120419134516	20120419135743
5	ami-00000053	20120419135822	20120419140111
6	ami-00000053	20120419135822	20120419141047
7	ami-00000022	20120419141154	20120419141519
8	ami-00000022	20120419141154	20120419142504
9	ami-00000020	20120419142548	20120419142819
10	ami-00000020	20120419142548	20120419143805
11	ami-00000016	20120419143904	20120419144220
12	ami-00000016	20120419143904	20120419145155
13	ami-00000017	20120419145258	20120419150034
14	ami-00000017	20120419145258	20120419151243
15	ami-00000024	20120419151333	20120419151705
16	ami-00000024	20120419151333	20120419152651
17	ami-00000026	20120419152847	20120419153108
18	ami-00000026	20120419152847	20120419154054
19	ami-00000058	20120419154149	20120419154358
20	ami-00000058	20120419154149	20120419155344
21	ami-00000062	20120419155533	20120419155731
22	ami-00000062	20120419155533	20120419155836
23	ami-00000044	20120419160134	20120419160329
24	ami-00000044	20120419160134	20120419160435
25	ami-00000045	20120419160506	20120419160700
26	ami-00000045	20120419160506	20120419160754
27	ami-00000061	20120419160845	20120419161122
28	ami-00000061	20120419160845	20120419162107
29	ami-00000065	20120419162202	20120419162858
30	ami-00000065	20120419162202	20120419164109
31	ami-00000066	20120419164158	20120419164844
32	ami-00000066	20120419164158	20120419170103
33	ami-00000064	20120419170146	20120419170432
34	ami-00000064	20120419170146	20120419171417
35	ami-00000111	20120419171438	20120419171634
36	ami-00000111	20120419171438	20120419171738
37	ami-00000099	20120419171831	20120419172533
38	ami-00000099	20120419171831	20120419173632
39	ami-00000130	20120419173700	20120419173934
40	ami-00000130	20120419173700	20120419174909
41	ami-00000136	20120419175018	20120419175535
42	ami-00000136	20120419175018	20120419180137
43	ami-00000141	20120419180442	20120419180643
44	ami-00000141	20120419180442	20120419180748
45	ami-00000143	20120419180906	20120419181114
46	ami-00000143	20120419180906	20120419182100
47	ami-00000146	20120419182130	20120419183141
48	ami-00000146	20120419182130	20120419185253
49	ami-00000164	20120419185343	20120419185643
50	ami-00000164	20120419185343	20120419190730
51	ami-00000165	20120419190808	20120419191422
52	ami-00000165	20120419190808	20120419192407
53	ami-00000166	20120419192436	20120419193255
54	ami-00000166	20120419192436	20120419194342
55	ami-00000168	20120419194518	20120419194828
56	ami-00000168	20120419194518	20120419195803
57	ami-00000169	20120419195834	20120419200100
58	ami-00000169	20120419195834	20120419201035
59	ami-00000163	20120419201106	20120419201728
60	ami-00000163	20120419201106	20120419202826
61	ami-00000174	20120419202918	20120419203115
62	ami-00000174	20120419202918	20120419203220
63	ami-00000173	20120419203420	20120419203619
64	ami-00000173	20120419203420	20120419204605

AutoVAS Table

id	scan_id	nvt	oid	threat	port	cve	bid
530	50	NTP 'ntpd' Autokey Stack Overflow Vulnerability	1.3.6.1.4.1.25623.1.0.900652	High (CVSS: 6.8)	ntp (123/udp)	CVE-2009-1252	35017
531	50	NTP Stack Buffer Overflow Vulnerability	1.3.6.1.4.1.25623.1.0.900623	High (CVSS: 6.8)	ntp (123/udp)	CVE-2009-0159	34481
532	50	Sendmail NULL Character CA SSL Certificate Validation Security Bypass Vulnerability	1.3.6.1.4.1.25623.1.0.100415	High (CVSS: 7.5)	smtp (25/tcp)	CVE-2009-4565	37543
533	50	TCP Sequence Number Approximation Reset Denial of Service Vulnerability	1.3.6.1.4.1.25623.1.0.902815	Medium	ssh (22/tcp)	CVE-2004-0230	10183
534	50	Record route	1.3.6.1.4.1.25623.1.0.12264	Low	general/icmp		
535	50	TCP timestamps	1.3.6.1.4.1.25623.1.0.80091	Low	general/tcp		
536	50	Sendmail Version Detection	1.3.6.1.4.1.25623.1.0.800608	Low	general/tcp		
537	50	Traceroute	1.3.6.1.4.1.25623.1.0.51662	Low	general/tcp		
538	50	NTP read variables	1.3.6.1.4.1.25623.1.0.10884	Low	ntp (123/udp)		
539	50	Services	1.3.6.1.4.1.25623.1.0.10330	Low	smtp (25/tcp)		
540	50	SMTP Server type and version	1.3.6.1.4.1.25623.1.0.10263	Low	smtp (25/tcp)		
541	50	SMTP antivirus scanner DoS	1.3.6.1.4.1.25623.1.0.11036	Low	smtp (25/tcp)		3027
542	50	Services	1.3.6.1.4.1.25623.1.0.10330	Low	ssh (22/tcp)		
543	50	CPE Inventory	1.3.6.1.4.1.25623.1.0.810002	Log (CVSS: 0.0)	general/CPE-T		
544	50	Host Summary	1.3.6.1.4.1.25623.1.0.810003	Log (CVSS: 0.0)	general/HOST-T		
545	50	ICMP Timestamp Detection	1.3.6.1.4.1.25623.1.0.103190	Log (CVSS: 0.0)	general/icmp		
546	50	OS fingerprinting	1.3.6.1.4.1.25623.1.0.102002	Log (CVSS: 0.0)	general/tcp	CVE-1999-0524	
547	50	Checks for open tcp ports	1.3.6.1.4.1.25623.1.0.900239	Log (CVSS: 0.0)	general/tcp		
548	50	Checks for open udp ports	1.3.6.1.4.1.25623.1.0.103978	Log (CVSS: 0.0)	general/tcp		
549	50	3com switch2hub	1.3.6.1.4.1.25623.1.0.80103	Log	general/tcp		
550	50	Information about the scan	1.3.6.1.4.1.25623.1.0.19506	Log (CVSS: 0.0)	general/tcp		
551	50		0	Log	smtp (25/tcp)		
552	50		0	Log	smtp (22/tcp)		
553	50	SSH Authorization	1.3.6.1.4.1.25623.1.0.90022	Log (CVSS: 0.0)	ssh (22/tcp)		
554	50	SSH Server type and version	1.3.6.1.4.1.25623.1.0.10267	Log (CVSS: 0.0)	ssh (22/tcp)		
555	50	SSH Protocol Versions Supported	1.3.6.1.4.1.25623.1.0.100259	Log (CVSS: 0.0)	ssh (22/tcp)		