

# CHALMERS



**D.E.V.A.S**

**Distributed Enterprise Vulnerability Assessment Scanner**

*Master of Science Thesis*

**AMIR BAZINE**

**YILDIRIM ZAYNAL**

Department of Computer Science & Engineering  
Division of Networks and Systems  
CHALMERS UNIVERSITY OF TECHNOLOGY  
Göteborg, Sweden, 2009

The Author grants to Chalmers University of Technology and University of Gothenburg the non-exclusive right to publish the Work electronically and in a non-commercial purpose make it accessible on the Internet.

The Author warrants that he/she is the author to the Work, and warrants that the Work does not contain text, pictures or other material that violates copyright law.

The Author shall, when transferring the rights of the Work to a third party (for example a publisher or a company), acknowledge the third party about this agreement. If the Author has signed a copyright agreement with a third party regarding the Work, the Author warrants hereby that he/she has obtained any necessary permission from this third party to let Chalmers University of Technology and University of Gothenburg store the Work electronically and make it accessible on the Internet.

D.E.V.A.S

Distributed Vulnerability Assessment Scanner

Amir Bazine

Yildirim Zaynal

© Amir Bazine, February 2009.

© Yildirim Zaynal, February 2009.

Examiner: Tomas Olovsson

Department of Computer Science and Engineering

Chalmers University of Technology

SE-412 96 Göteborg

Sweden

Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering

Göteborg, Sweden February 2009

# Abstract

The Distributed Enterprise Vulnerability Assessment Scanner is a service requested by the European institution we are currently working for. The request was triggered by the Information Security Team, who wished to have an opinion regarding the security level of their current network and aimed to improve any possible holes and security flaws. Our job within the Information Security department of this organization was to create a vulnerability scanner that would make weekly scans of their networks and report the results to the IT-administrators. The scanner will not only be used as a tool to help us identify weaknesses, but also as a proof to convince other departments within the organization and Board Members that Information Security is a vital part of the organization well being. This would give the organization a better view of what kind of attacks that is possible from inside and outside the organization. The project has been a success. It is currently running in production in this major European institution.

# Abstrakt

Den distribuerade företags-sårbarhets-utvärderings-skannern är en tjänst som begärdes av den europeiska institutionen vi jobbar för. Denna begäran kom till liv när informationssäkerhetsavdelningen bestämde sig för att de behövde en uppfattning av säkerhetsnivån i deras nuvarande nätverk och för att förbättra möjliga säkerhetshål och sårbarheter. Vårt jobb inom informationssäkerhetsavdelningen var att skapa en sårbarhetsskanner som skulle kunna göra sårbarhetsutvärderingar varje vecka i deras nätverk och som sedan rapporterade resultatet till IT-administratörerna. Detta verktyg kommer att göra det möjligt att identifiera svagheter, men även som ett bevis för att kunna övertyga andra avdelningar inom organisationen och styrelseledamöter om att informationssäkerhet är en livsviktig del av organisationens välmående. Sårbarhetsskannern ska kunna ge organisationen en bättre syn på vilka attacker som är möjliga från insidan och utsidan av organisationen. Projektet avslutades framgångsrikt. Den används idag i produktion av den europeiska institutionen.

# Acknowledgments

*I would like to thank my family for their endless support. I would also like to thank my girlfriend Nélia and my friends for all help and patience you had with me. Lastly, many thanks to my examiner Tomas Olovsson and the anonymous organization for providing me with the opportunity for doing this master thesis.*

-Amir

*I would like to thank my family, especially my older brother Savash and his wife Meral for being there for me at all times. I would also like to thank my sweet niece Neslihan for bringing laughter on hard times. My appreciation goes also to my examiner Tomas Olovsson, my close friends and of course to my colleagues of the anonymous organization making this thesis project a reality.*

-Yildirim

# Table of Contents

---

<b>1. Introduction.....</b>	<b>5</b>
1.1 Background .....	5
1.2 Problem .....	5
1.3 Purpose.....	5
1.4 Limits .....	5
1.5 Time Frame .....	6
1.6 Chapter Overview.....	6
<b>2. Information Security Introduction .....</b>	<b>7</b>
2.1 CIA .....	7
2.2 Risk Management.....	7
2.3 Access Control .....	8
2.4 Defense in Depth .....	9
<b>3. TCP/IP .....</b>	<b>10</b>
3.1 Introduction .....	10
3.2 Background .....	11
3.3 TCP/IP in Detail.....	12
3.3.1 Internet Model and OSI Model .....	12
3.3.2 Internet Protocol.....	14
3.3.3 Classless Inter Domain Routing Scheme (CIDR) .....	14
3.3.4 Internet Control Message Protocol (ICMP) .....	15
3.3.5 3-Way Handshaking.....	15
3.4 TCP State Diagram .....	17
<b>4. Port Scanning .....</b>	<b>19</b>
4.1 Introduction .....	19
4.1.1 Network Scanning .....	19
4.1.2 Ports.....	19
4.2 Network Scanning Techniques .....	21
4.2.1 Host Discovery .....	21
4.2.2 Port Scanning .....	21
4.3 Operating System Detection .....	22
<b>5. NMAP.....</b>	<b>23</b>
5.1 Introduction .....	23
5.1.1 Nmap's Features .....	24
5.1.2 Nmap's User Interface .....	24

<b>5.2 Nmap in Enterprises .....</b>	<b>25</b>
5.2.1 Compliance Testing .....	25
<b>5.3 Optimizations .....</b>	<b>26</b>
<b>5.4 Examples .....</b>	<b>27</b>
5.4.1 TCP Connect Scan (-sT) .....	27
5.4.2 TCP SYN Scan (-sS) .....	28
5.4.3 TCP NULL, FIN, and Xmas Scans (-sN; -sF; -sX) .....	29
5.4.4 Other Scans .....	29
<b>6. Nessus .....</b>	<b>30</b>
<b>6.1 Nessus Scanning Stages .....</b>	<b>32</b>
6.1.1 Host Detection .....	32
6.1.2 Service Detection .....	32
6.1.3 Information Gathering .....	32
6.1.4 Vulnerability Fingerprinting .....	33
6.1.5 Denial of Service Testing .....	33
6.1.6 Report Generation .....	33
<b>7. Enterprise Scanning .....</b>	<b>35</b>
7.1 Planning and Deployment .....	36
7.2 Network Topology .....	36
<b>8. The Implementation of D.E.V.A.S .....</b>	<b>39</b>
<b>8.1 Introduction .....</b>	<b>39</b>
<b>8.2 D.E.V.A.S Configuration .....</b>	<b>39</b>
8.2.1 Cron .....	39
8.2.2 Nessus Configuration .....	40
8.2.3 Plugins .....	40
8.2.4 Credentials .....	41
8.2.5 Scan Options .....	41
8.2.6 Prefs .....	42
8.2.7 KB .....	42
<b>8.3 Application Overview .....</b>	<b>43</b>
8.3.1 Update Plugins .....	43
8.3.2 Blacklisting .....	45
8.3.3 Configuration Checking .....	46
8.3.4 Generate Target List .....	47
8.3.5 Vulnerability Scanning .....	48
8.3.6 Logging .....	50
8.3.7 Health Report .....	51
<b>8.4 Distributed Scanning .....</b>	<b>53</b>
8.4.1 Update and Distribute New Plugins to Agents .....	54
8.4.2 Generation Of Alive Target List .....	54
8.4.3 Generation Of Alive Agent List .....	54
8.4.4 Sharing Targets Between Agents .....	55
8.4.5 Start, Collect and Merge Agent Result Files .....	56

<b>8.5 Data Analysis and Correlation .....</b>	<b>58</b>
8.5.1 Analysis from a Single Result File .....	58
<b>8.6 Correlation Between Two Result Files .....</b>	<b>59</b>
<b>9. Future improvements.....</b>	<b>64</b>
9.1 Upgrade to Nessus 3.x .....	64
9.2 Rewrite D.E.V.A.S .....	64
9.3 Save Scanning Results into a Database .....	64
9.4 Web Frontend .....	65
9.5 Integration with Intrusion Detection Systems .....	66
<b>10. Conclusions.....</b>	<b>68</b>
<b>11. Appendix .....</b>	<b>69</b>
<b>Installation guides .....</b>	<b>69</b>
Fast installation for single host scan .....	69
Install packages.....	69
Checkout from the SVN repository.....	69
Nessus.....	69
Crontab .....	70
Access to files .....	70
Server installation for distributed scanning .....	71
Install packages.....	71
Checkout from the SVN repository.....	71
SSH.....	71
Nessus.....	71
Crontab .....	72
Access to files .....	72
Client installation for distributed scanning.....	73
Install packages.....	73
Checkout from the SVN repository.....	73
SSH.....	73
Nessus.....	73
Crontab .....	74
Access to files .....	74
<b>Directory layout .....</b>	<b>75</b>
The directory layout.....	75
BACKUP .....	75
conf .....	75
logs.....	76
results .....	76
scripts.....	76
Scripts/debugScripts .....	76
Scripts/nbeScripts .....	77
Scripts/lib .....	77
<b>12. Glossary.....</b>	<b>78</b>



<b>13. Bibliography .....</b>	<b>80</b>
<b>14. Figures Index .....</b>	<b>84</b>

# 1. Introduction

---

## 1.1 Background

Information security has an important role today's business and cyber community. Network scanning is a key component to maintaining secure networks and systems. Proactive management is an important aspect and can help to find vulnerabilities before they can be found and exploited by attackers.

We introduce some network scanning tools and potential uses of these security tools. As usually happens with public security tools used by administrators and security professionals, attackers misuse these tools to find vulnerabilities. It is therefore very important to detect these vulnerabilities and have the opportunity to correct them before they are misused.

The security assessment scanner consists of the security tools Nmap, Nessus, SinFP and is combined with various scripting languages.

## 1.2 Problem

The idea for this master thesis came during our traineeship in the IT security department of an anonymous international organization. The IT security administrators searched for a way to map their network topology and vulnerabilities by using open source software. There were not any security tools that fitted the organization's requirements and needs regarding time limitations, scalability, network topology, automatic periodic running and report generation.

## 1.3 Purpose

The purpose of this master thesis was to create a distributed enterprise vulnerability assessment scanner using available open source tools.

The vulnerability scanner can be divided into two parts consisting of a management server and multiple agents. The server initiates the scan and divides the workload between the agents. The agents perform the scan given by the server and send back the results to the server, where the results are combined and a report is generated and sent to the IT administrators via email.

By deploying and using this vulnerability scanner, the IT security administrators have identified multiple vulnerabilities and found ways to improve the security of their network. The ease of use and the report generation from our solution have created a comfortable and time saving method for identifying vulnerabilities in a large scale.

## 1.4 Limits

The design and architecture of the application was decided within a short period of time, but the implementation of the final distributed scanner went beyond the initial time frame. Due to time constraints, we had to use fast prototyped scripting languages and finished with an implementation consisting of multiple programming languages.

## 1.5 Time Frame

The initial time frame we set was not fully accomplished. The estimated time frame was 1 month of research, 2 months of implementation and 1-2 months for finalizing the report.

The research was done within 2 weeks but the implementation took more time than initially expected. The total time spent with implementation was 4-5 months and the preparation for the report was done in parallel with the implementation.

## 1.6 Chapter Overview

Chapter 2 provides a general overview of information security. It explains core principles and defines general terms used in the information security field.

Chapter 3 describes the TCP/IP protocol suite. This chapter is the basis for understanding the coming chapters.

Chapter 4 provides a general overview of port scanning techniques.

Chapter 5 describes the port scanner Nmap, which is used in our project.

Chapter 6 covers the application Nessus, the vulnerability assessment tool we are using in our project.

Chapter 7 provides an insight in the art of enterprise scanning. It also presents different architectures of how to implementing an enterprise scanner.

Chapter 8 presents the implementation of D.E.V.A.S and how the different parts explained in previous chapters are used to construct the final product. This chapter presents our results.

Chapter 9 introduces potential future improvements of the project.

Chapter 10 presents the conclusions of the thesis.

## 2. Information Security Introduction

---

During the World War II, information security evolved considerably. Many of today's terms and inventions arose from that time, especially in physical security. In the end of the 20<sup>th</sup> century, where great innovation and development was made in telecommunication and computers systems, information- and computer security evolved. During the last ten years, E-businesses have pushed the demand for more secure systems. The PCI DSS (Payment Card Industry Data Security Standard) standard is one of the responses that will hopefully make payment systems more secure.

### 2.1 CIA

The core principles in information security have long been the CIA triad:

- Confidentiality
- Integrity
- Availability

Confidentiality is about preventing disclosure of information to unauthorized parties. Nowadays in the E-commerce era, where people are sending credit card numbers over Internet for buying goods, it is a necessity to maintain privacy. This is normally achieved by using cryptography, but it can also include having better physical security for our computer systems. It is very crucial in some areas such as governments, financial institutes and military, that confidentiality is maintained; otherwise people's private information could be leaked out or companies can go bankrupt.

Integrity means that data cannot be modified by unauthorized parties. This includes staff changing their own salary, deleting files or computer viruses compromising computer systems. To be able to verify and trust input data, there has to be some type of integrity checks to prevent or detect malicious or accidental changes.

The purpose of availability is to have available information when it is needed. To succeed with this property, expensive high availability systems are implemented to be able to recover if one of the components in the work process does not work. Its aim is to work all times, even in cases of hardware failures or power cuts. There exist many types of denial of service attacks with the goal of doing a service unavailable.

### 2.2 Risk Management

Wikipedia defines risk management as: *"Risk management is the process of identifying vulnerabilities and threats to the information resources used by an organization in achieving business objectives, and deciding what countermeasures, if any, to take in reducing risk to an acceptable level, based on the value of the information resource to the organization."* It is an ongoing iterative process as the environment is constantly changing and new vulnerabilities are being published every day. This security process should start from the creation of the organization until the end of it.

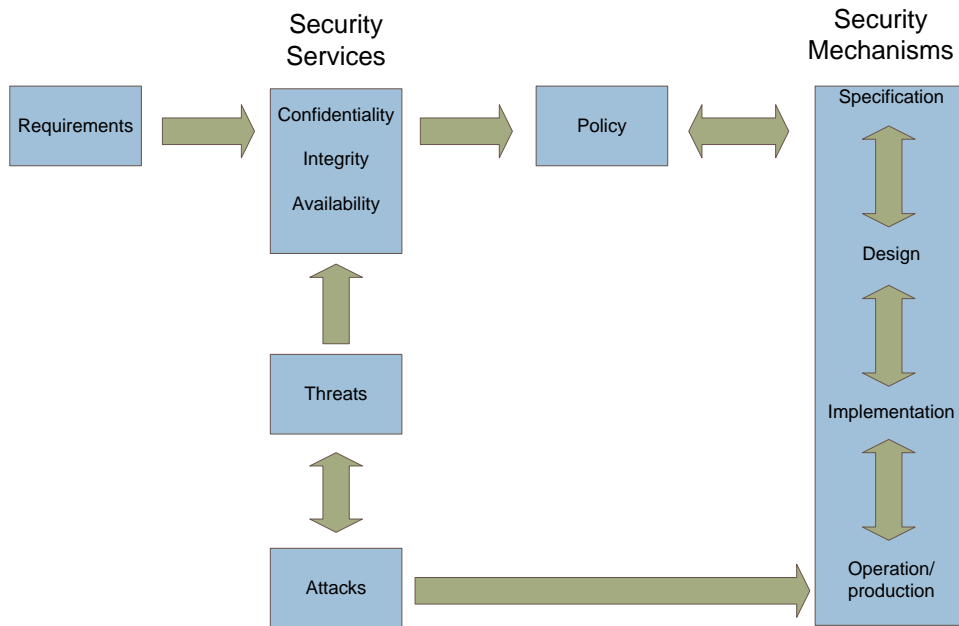


Figure 1 Ongoing security process

There are some commonly used terms in risk management and in security that should be defined:

- **Vulnerability** is a weakness that could be used by an unauthorized party to cause harm to an asset.
- **Threat** is anything that can potentially cause harm.
- **Impact** is when a threat uses a vulnerability that causes harm.
- **Risk** is a measure of the likelihood that something bad will happen that causes losses to the organization. A common way to calculate the risk is:  $\text{threat} * \text{vulnerability} * \text{value} = \text{risk}$ .

The impact in a security standpoint is a loss in confidentiality, integrity or availability. It is up to the management of the organization to accept the risk, mitigate the risk or deny the risk. Two common reasons for accepting the risk are the low likelihood of an impact or the low value of the asset that has a vulnerability.

In the case management of the organization is mitigating the risk, appropriate control measures should be implemented to reduce the risk. The control measure could be administrative, technical or physical. Administrative control measures are about policies, procedures, standards and guidelines. Administrative controls tell people how the business should be run, which they later need to follow. Technical control measures use software to control data and monitor. It includes for example firewalls, access control lists and encryption. Physical control measures monitors and controls the access to the physical environment and to the computing facilities. Common physical control measures are fences, locks and security guards.

## 2.3 Access Control

Access control is an important part of security as it tells who is authorized to access information and resources. The more sensitive or valuable assets to protect the more fine grained and stronger control mechanisms need to be implemented. Access control is split into three steps; identification, authentication and authorization.

The identification step tells who you are; the actual verification to assert the claim of the identity is in the authentication step.

The authentication step is the process of verifying an identity. The authentication step could be compared to the verification step in the airport where the airport staff verifies you with your passport photo. There are three different types of information that can be used for authentication: something you know, something you have and something you are. Something you know could be a password that you remember; something that you have could be a physical security token that you possess; something that you are could be your fingerprint that is unique. It is called strong authentication or two-factor authentication if two of these three authentication types are used. Two factor authentication is very common in e-banking services nowadays.

Username is the most common form of identification and the password as authentication. After the identification and authentication step has been completed, there is a verification step to determine what type of information resources you can access and what operation you can perform. This step is called authorization.

## **2.4 Defense in Depth**

Defense in depth is an important security principle. It is an act of having overlapping security measures. If one security measure should fail, there will be additional security measures to protect the sensitive resource. This is especially important in complex systems where sensitive information travels through and is stored in many systems. The more complexity involved, the more potential for vulnerabilities exists. Defense in depth is as strong as its weakest link.

Security is an ongoing process. As a review of the current system and its vulnerability state can just determine how vulnerable an organization is at a particular time, it is important that this is a recurrent process because the environment is changing as fast as ever before.

## 3. TCP/IP

### 3.1 Introduction

The TCP/IP protocol suite is used for communication between applications and operating systems on Internet and other networks. It can be used for video, voice or data communication.

Intranets, which are widely used by organizations all around the world are composed by internal networks and websites. These websites are only available within the organization's network and cannot usually be accessed from outside. Parts of the intranet that can be accessed from the public Internet are called extranet. The name is derived from 'External Network'. Among public services available to everybody, there are subscribed services on the extranet that are only available for subscribers (paying customers). One example is Reuters, which is a British news agency that provides reports to newspapers and broadcasters. Different agencies and newspaper organizations can subscribe to Reuter's service through the Internet and get real time data and information.

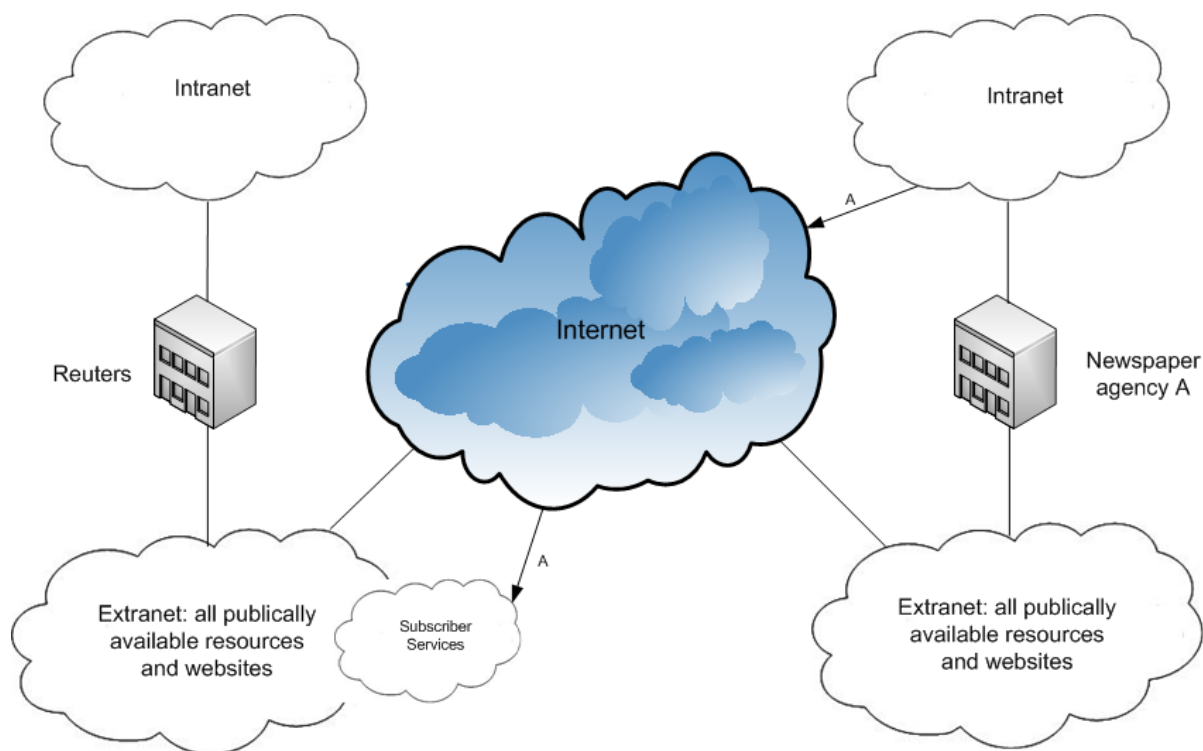


Figure 2 Intranet, Extranet and Internet

TCP/IP is also extremely extensible, broad and available. With TCP/IP, one can communicate and reach other TCP/IP networks all around the world from any point. This can be done very from the user's point of view. TCP/IP has been successful in being deployed as a worldwide protocol for communications. One example of its extensibility is the ongoing switch from IPV4 to IPV6. Since the

massive increase of computers around the world connected to Internet, the need for unique IP addresses cannot longer be covered by IPV4 and there is a need to change to IPV6, which has a much larger IP address space. IPV4 is the fourth revision of IP which utilizes 32bit addressing. This limits IPV4 to  $2^{32}$  unique addresses. Some of these addresses are reserved for private networks and other purposes which reduces the amount of available addresses.

## 3.2 Background

The birth of TCP started within DARPA (Defense Advanced Research Projects Agency), an agency of the United States department of defense responsible for the development of new technology used by the military. ARPANET (Advanced Research Projects Agency Network) was developed by DARPA and was used by the military. Their main goal with ARPANET was to let different computers be able to communicate with each other as if they were just one computer.

The shape of TCP/IP as it is today started from the fact that all hardware and software architecture were closed systems. They needed to create it in a way that it would not require significant modifications or alterations of the operating system or the hardware.

ARPANET began operating in 1969 and the developers were using the Network Control Protocol (NCP). TCP was later created by Kahn and Cerf and started to replace NCP. TCP was preferable for many reasons and became quickly very popular. The TCP protocol was faster and easier, not to mention less expensive to implement. The experts decided to adopt a single networking protocol for the best and ease of use practices, since many other networks started to connect to ARPANET like radio, satellites, USENET, BITNET etc. One of the major things that helped the rising of TCP was that DARPA allowed the protocol suite to become part of Berkeley's UNIX system. As DARPA began to work on Internet Technology, their researchers developed the Internet protocol, which was used to route packets through Internet.

Later, Berkeley's UNIX system with the TCP/IP protocol suite became the standard choice for many universities and was used in research environments. DARPA made a requirement that all computers that were connected to ARPANET should use the TCP/IP protocol.

During this time, many other networks started to be formed; some examples were CSNET, BITNET and USENET. All were based on TCP/IP and were interconnected using ARPANET as a backbone. Even Local Area Networks started to take place in universities and Ethernet based routers were created by companies to enable hosts or terminals to be interconnected. It was just a matter of time until different intranets started to get interconnected such as universities to share scientific data. In the end, the original ARPANET was expensive to run and maintain within ARPA. The project was later closed and in its place Internet emerged.



## 3.3 TCP/IP in Detail

### 3.3.1 Internet Model and OSI Model

A traditional way of describing the design of the Internet model, is to describe the OSI model and make comparison with the Internet model. The Open Systems Interconnection Basic Reference Model (OSI Model) is a more general description for network protocol design and the purpose of the OSI model is to standardize network communication. By using the OSI model we can show the differences and characteristics of the TCP/IP protocol suite by making simple comparisons of the layers.

The Internet model has fewer layers compared to the OSI model. The model consists of 4 different layers, in difference to the OSI model which has 7 layers. Both the Internet model and the OSI model are based on packet switching networks.

Looking at Figure 3, it is possible to see the similarities and differences between the different models. The various layers of these models are described below.

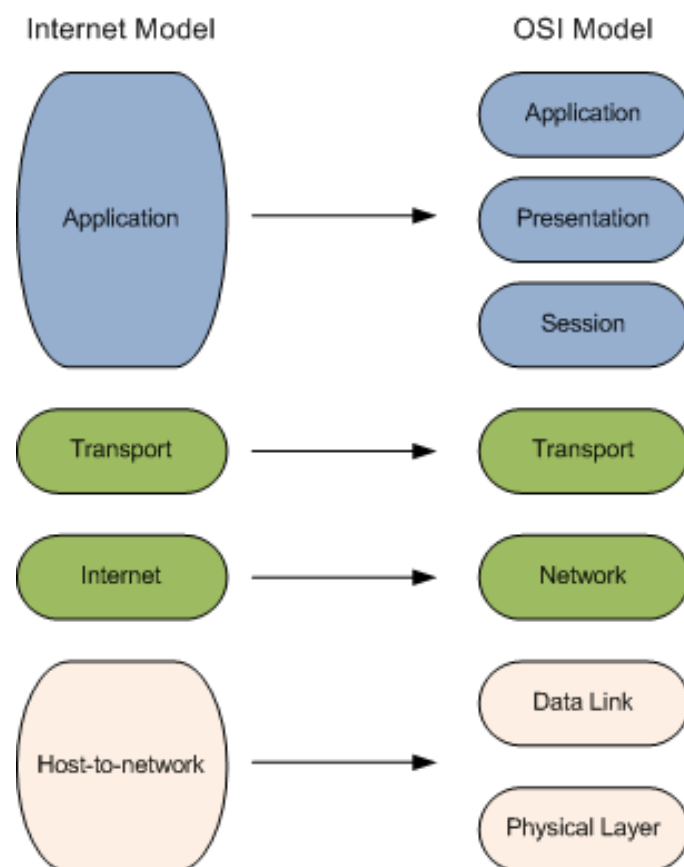


Figure 3 OSI Model

**Application layer:** In the OSI model, the Internet model application layer consists of three different layers; Application, presentation and session. These three different layers deal with encoding (Application), dialog control issues (Session) and presentation (Presentation).

These three layers are only concerned with software application issues and not related to the underlying implementation of networking. The session Layer's function is to help applications to create sessions with each other. A session is a persisting established link of two software applications. Sessions can be compared to a telephone call between 2 persons. These functions or tools are provided to the applications by Application Program Interfaces (APIs). Some examples of APIs are NetBIOS, TCP/IP Sockets and Remote Procedure Calls (RPCs). These APIs are used by programmers when they create software for doing intercommunication.

The presentation layer takes care of the presentation of data. This includes translation between different operating systems. Since networks can connect different types of operating systems, such as Windows XP, Windows 2000, GNU/Linux and Solaris, the presentation layer helps to translate the data between these different systems. Compression and decompression, Secure Sockets Layer (SSL) encryption and decryption are done at this layer too. Some of the encryption is even done at a lower level than the presentation layer, like the security protocol IPSec (Internet Protocol Security). This technology will not be covered by this report since it is not a part of the thesis.

The application layer in the OSI model is used by network applications. For example, a web browser is an application that uses the Hyper Text Transfer Protocol (HTTP), which operates at the application layer level. But not only applications make use of the application layer; operating systems use the application layer services as well. Some examples of application layer protocols are HTTP, FTP, SMTP, DHCP, NFS, Telnet, SNMP, POP3 and IRC.

**Transport:** The transport layer in the Internet model provides similar services with its equivalent transport layer in the OSI model. Its responsibilities include application data segmentation, transmission reliability, flow and error control.

The transport layer is the 'bridge' between the application and the lower level layers. Its purpose is to provide the required functions so that applications on different computers can communicate with each other.

**Internet:** The Internet layer in Internet model provides the same services as the OSI Network layer. Their purpose is to help packets get across networks, to route the packet to the correct device compared to the Data Link Layer which only routes to neighboring devices.

**Host-to-network:** The Internet model's host-to-network layer corresponds to OSI model's physical and data link layer. An example of the physical layer is the Ethernet protocol. It is the lowest layer in the OSI model.

The data link layer's purpose is to provide error free data transmission. By using the Cyclic Redundancy Check (CRC), error detection is performed. It is the layer where many wired or wireless local area networks function.

### 3.3.2 Internet Protocol

The Internet Protocol (IP) is located in the Internet layer of the Internet model. It transmits blocks of data called datagrams. These datagrams are transmitted and received to and from source and destination hosts. The IP protocol does not establish any kind of session and it offers a best effort connectionless delivery service between the source and the destination. This layer is also responsible for IP protocol addressing. In order to have multiple IP networks sending data, there must be a flow between different addressed systems. A router device is used to route the data between different IP addressed networks. A router can be explained as a traffic cop, you tell the traffic cop where you want to go and he will point you to that direction.

The major components of the Internet Protocol are:

- **IP Identification (IPID).** Used to identify IP datagrams when they are fragmented, so that they are assembled correctly on the receiving end.
- **Time-to-live (TTL).** When TTL reaches zero, the datagram is dropped. This is to prevent datagrams from routing infinitely in circles.
- **Source IP Address.** The IP address of the sender.
- **Destination IP Address.** The IP address of the receiver.

IP addresses are usually written in dot decimal notation, an example is 192.168.1.1. They can also be written in decimal, octal, hexadecimal, dotted octal, and in the dotted hexadecimal format. The dot decimal notation consists of 4 groups of numbers divided by dots. These groups are called octets and each octet can be a number of 2<sup>8</sup> size.

The first IP address allocation schema was done in such way that the first octet was assigned to the network ID and the three last octets to the host ID. This was changed as it would limit the amount of networks to 256. This limit was bypassed by defining different classes of networks and that created the birth of classful networking. The classful networking had five different classes; they were called class A, B, C, D and E. There were limitations in such ways that some networks had many host addresses while others did not. For this reason in 1993 the Classless Inter Domain Routing schema replaced the classful networking schema.

### 3.3.3 Classless Inter Domain Routing Scheme (CIDR)

In contrast to the classful networking schema, CIDR does not have limitations and can rearrange the number of addresses that may be allocated to entities such as Internet Service Providers (ISPs) or Local Area Networks. With CIDR, the IP address format changes from <IP> to <IP / CIDR Prefix>.

Class A networks have prefix /8.

Class B networks have prefix /16.

Class C networks have prefix /24.

The class names presented above may introduce confusion as it is supposed to be a classless schema, but these class names are only used to define the size of the 3 most common networks. A class C network is a network with maximum 254 hosts. This means that any network with CIDR prefix ranging from 24 – 32 is a class C network regarding the amount of hosts.

The prefix can range from /0 to /32. The prefix value indicates the number of bits starting from the left that will be used for the network. The remaining bits are for the host addressing. The prefix is not affected by network classes and can be assigned to any network regardless the class.

A host assigned an address in a subnet cannot directly communicate with another host assigned an address in a different subnet, if they are on the same physical network without any type of router. Some CIDR address blocks are assigned for special reasons or functions. An important example is the private network address blocks. These are:

10.0.0.0/8,  
172.16.0.0/12,  
192.168.0.0/16.

An example to clarify the CIDR notation:

Network: 192.168.003.001/24

The first 24 bits are used to define the network and each octet is 8 bit. The 24 bits equals the first three octets: 192.168.003. The last octet is then used for host addressing.

### 3.3.4 Internet Control Message Protocol (ICMP)

The Internet Control Message Protocol is used for error reporting and managing errors for IP networks. Some common ICMP messages are:

- **Echo Request (Type 8).** Used by programs such as ping to determine if a host is alive.
- **Destination Unreachable (Type 3).** This message is sent to the source IP address when the destination IP address cannot be found. This happens when a host or the network is down.
- **Time Exceeded (Type 11).** Occurs when a packet's TTL reaches 0.

### 3.3.5 3-Way Handshaking

The three way handshaking is the method used by the TCP protocol to establish a connection. A connection will only be established after the sender has exchanged a few control packets to the receiver. This has the function of synchronizing each endpoint with a sequence number and an acknowledgment number. This is a mechanism to prevent loss or duplication of protocol data units. The first packet sent from Host A to Host B is a connection request message. This is done by sending a data unit with the SYN flag set and a sequence number x. The sequence number can start at any number and is chosen by Host A. Host B will receive the data unit and understand that it is a connection request because of the SYN flag. If Host B decides to accept the connection then it will build a new data unit to send back to Host A to notify that the connection request has been accepted. This data unit has the ACK(ACKNOWLEDGE) and SYN flag set with a ACK number of x +1 and with a sequence number y set by Host B. Y stands for the initial sequence number chosen by host B. Lastly, host A will send a packet with the ACK flag set, with an acknowledgement number of y + 1. Figure 4 below displays the whole procedure for starting a connection. It also shows how data is sent and how the termination of a connection is performed.

For closing a connection, Host A will request the connection to be closed. This is done by sending a segment with the FIN flag set. Host B will accept this request and sends a request that the

connection on host A's side to be closed as well. From the sequence numbers and the acknowledgement values, we can see how much data that has been received by each host.

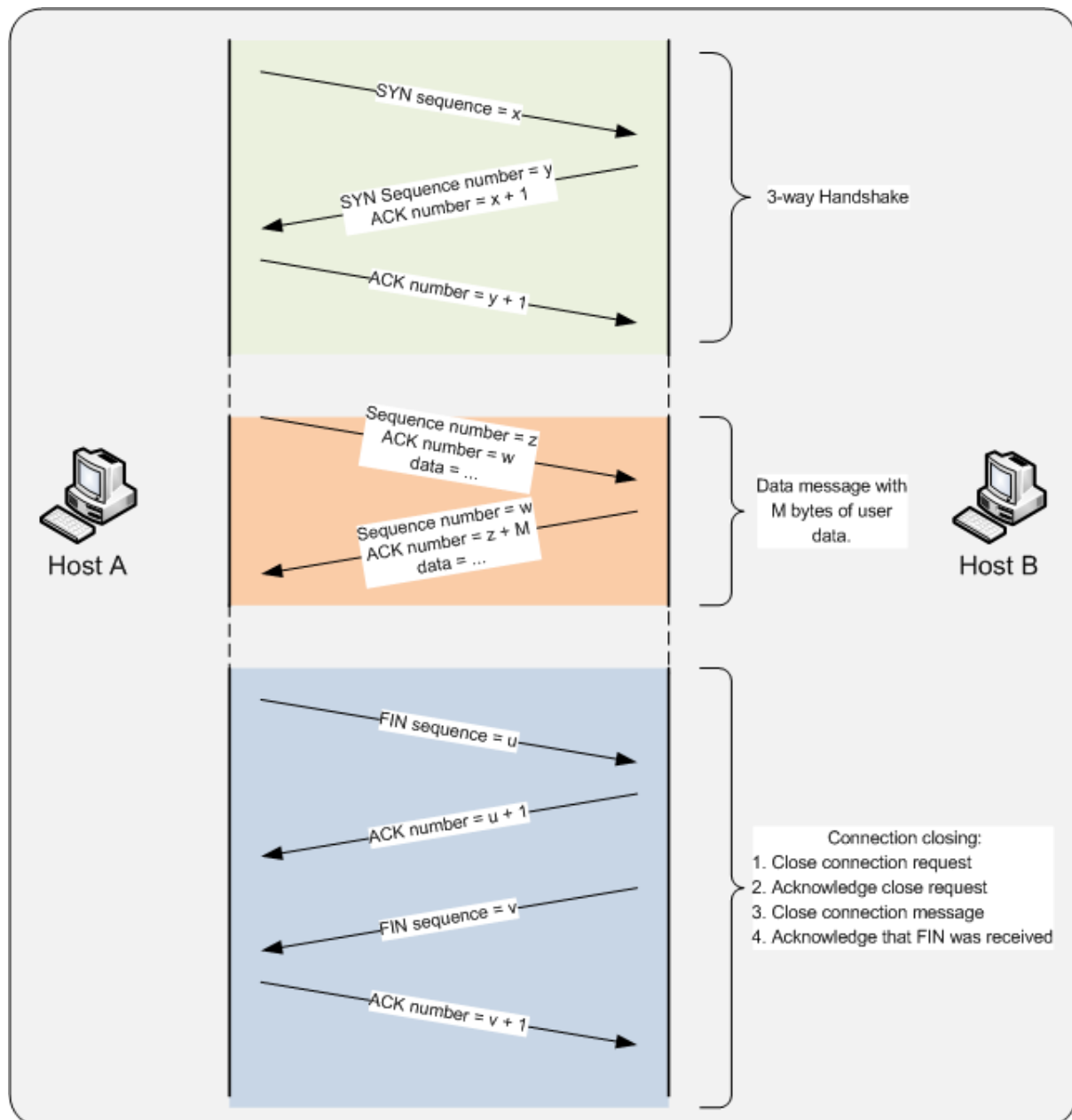


Figure 4 Complete TCP session

### 3.4 TCP State Diagram

All possible states that the TCP protocol can enter can be visualized with a TCP state diagram. The TCP state diagram shows all possible states and the path to a particular state.

We will go through and explain the 3-WAY-HANDSHAKE process using the TCP state diagram.

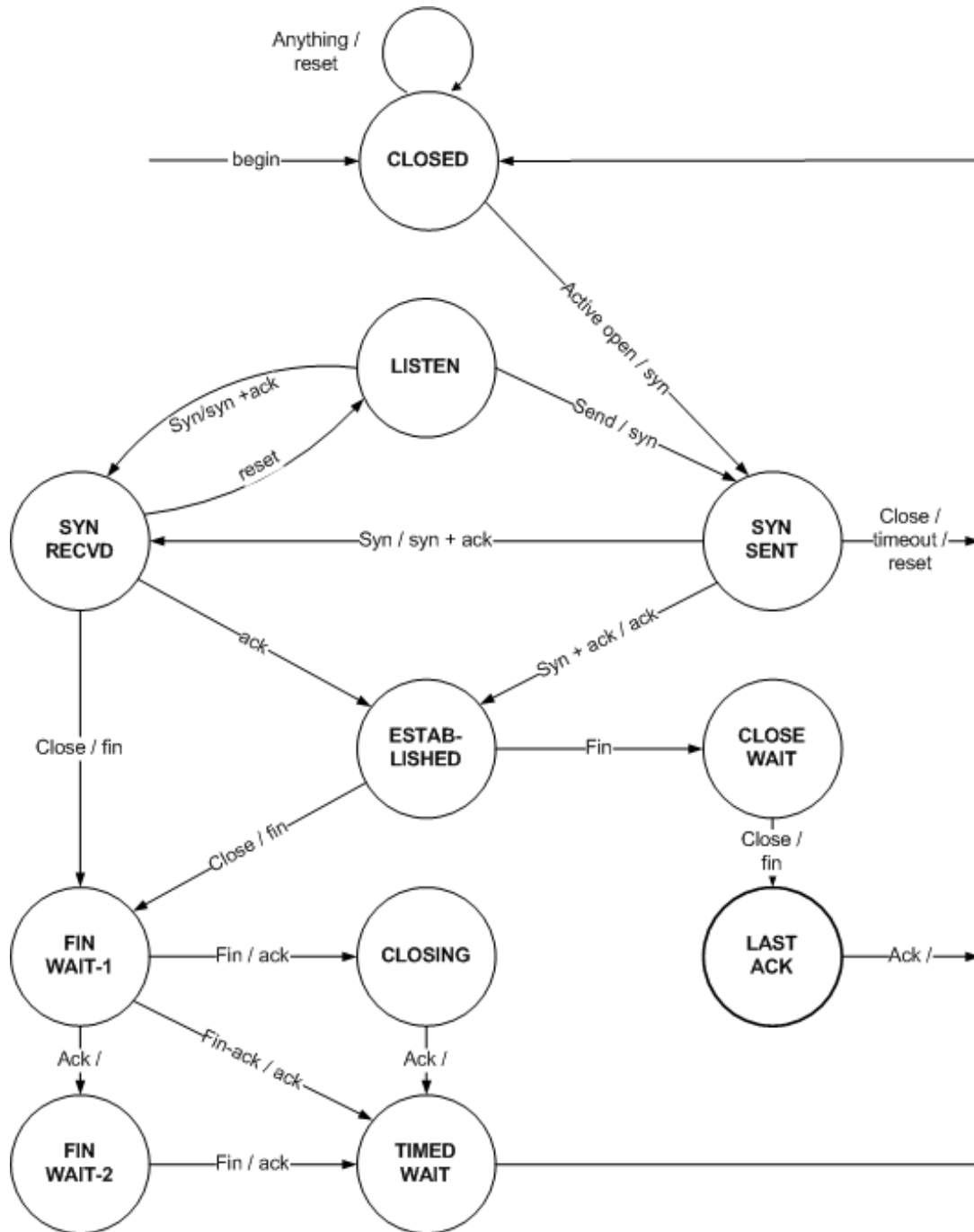


Figure 5 TCP state diagram (RFC 793)

The labels on the transitions are what TCP receives in order to make the transition and also what it replies with. As seen from the diagram, the TCP software on both ends starts in a closed state.

Opening of a connection can be seen from the two transitions coming from SYN RCVD and SYN SENT. Data transfer is progressed in the ESTABLISHED state.

Receiving a SYN segment when the connection is in the LISTEN state, the transition goes to the SYN\_RCVD state and replies back with an ACK+SYN segment. As seen in the diagram, the transition from the LISTEN state to the SYN RCVD state occurs when the connection receives a SYN segment, replying with SYN + ACK segment. The client will make the transition to the SYN\_SENT state when it does an active open by sending a SYN segment to the server. The client continues the transition to the ESTABLISHED state when it receives the SYN + ACK segment and responds with an ACK. Following the diagram we can easily trace that the server connection state changes to the ESTABLISHED state when it receives the SYN segment when it is in the SYN RCVD state. This is the complete trace of the THREE-WAY HANDSHAKE.

# 4. Port Scanning

---

## 4.1 Introduction

In the world of information security, port scanning is one of the base steps an intruder or a penetration tester needs to take in order to understand and see possibilities of weaknesses or network structures.

A parable in non technical terms to understand what the purpose of port scanning would be the thief-in-neighborhood example. A thief would go through the neighborhood and check for each house if the entrance is locked or not. Each house can have many different entrance points, such as doors, windows, etc.

If we map the thief example terms into technical terms:

- |   |            |
|---|------------|
| • Thief                                     | - Intruder |
| • Neighborhood                              | - Network  |
| • House                                     | - Computer |
| • Doors/windows                             | - Ports    |
| • Act of checking for house entrance points | - Scanning |

Port scanning as the name implies comes from the act of scanning ports of a computer. Ports are the entrance points to the services provided by a computer.

### 4.1.1 Network Scanning

Network scanning is a combination of different processes related to port scanning. It can be divided into four main parts:

- **Network Mapping** sending messages to a host and see if it will generate any response and determining by the response if the host is alive or not.
- **Port Scanning** sending messages to a specific port on a host to see if the port is open.
- **Service detection** sending specially crafted messages to generate responses that will indicate the service name and the service version that is running behind the port.
- **OS detection** is also performed by sending specially crafted messages. By using statistical data and information from the responses, it is possible to determine the operating system.

### 4.1.2 Ports

The most popular transport protocols TCP and UDP use 16 bit port addressing, i.e. the protocol is able to have totally 65536 different ports. These ports are used for endpoint-to-endpoint communication and can be divided into three different ranges:

- **0 - 1023** These ports are the well known ports. These are assigned by the Internet Assigned Numbers Authority (IANA) and are considered static. Binding a communication socket in this range requires administrator privileges and some of them are used by the operating system.
- **1024 - 49151** Ports in this range are called registered ports. They are registered by the Internet Corporation for Assigned Names and Numbers (ICANN) for a certain use.



- **49152 – 65535** These are the dynamic/private ports and are not permanently assigned for any specific use or any public application.

A computer might have many different network applications running during the same time. The function of ports helps the operating system and application to determine which incoming data belongs to which service. Applications such as an email client, web browser, FTP server and others are assigned to different port numbers. This prevents data from a web server going to an ftp client by mistake. Below is a table of the most common ports and their descriptions.

Internet Protocol Ports	Protocol(s)	Description
80	TCP	Hypertext Transfer Protocol (HTTP), commonly used for web servers
443	TCP	HTTP secure sockets (HTTPS) for secure web communication
53	UDP and TCP	Domain Name Service (DNS) for resolving names to IP addresses
25	TCP	Simple Mail Transfer Protocol (SMTP) for sending email
22	TCP	Secure Shell (SSH) protocol for encrypted communication
23	TCP	Telnet, a plaintext administration protocol
20 and 21	TCP	File Transfer Protocol (FTP) for transferring data between systems
135 – 139 and 445	TCP and UDP	Windows file sharing, login and Remote Procedure Call (RPC)
500	UDP	Internet Security Association and Key Management Protocol (ISAKMP) for secure Internet Protocol (IPSec)
5060	UDP	Session Initiation Protocol (SIP) for some Voice over IP (VOIP) uses
123	UDP	Network Time Protocol (NTP) for network time synchronization

Figure 6 Common ports and descriptions

## 4.2 Network Scanning Techniques

The most common network scanning techniques are host discovery, port scanning and operating system detection, which are described in more detail below.

### 4.2.1 Host Discovery

Host discovery implies finding alive hosts in a network. This is achieved by trying to get a response from a host. Network scanners use various techniques or methods to retrieve a response from the target and the most common methods are listed below:

- **ICMP ECHO REQUEST.** More commonly known as ping and is an ICMP type 8 message. The valid response of an active host is an ICMP ECHO REPLY (type 0) message. Sending ICMP ECHO REQUEST messages to multiple addresses is known as *ping sweep*.
- **ICMP TIMESTAMP.** An ICMP type 13 message is sent from the scanner. If the target is alive, a respond with the current time (ICMP type 14) will be sent back.
- **ICMP Address Mask Request.** An ICMP type 17 message is sent from the scanner. If the target is alive it will respond with its netmask (ICMP type 18).
- **TCP Ping.** This is a method of simulating a TCP 3-way-handshake procedure. By sending TCP SYN or TCP ACK segments to specific ports (such as 22, 80 and/or 445) on the target system, the scanner can determine if the target is alive by waiting for any TCP segment response. This is of course also depending on firewall settings and the type of response will vary depending on the operating system.
- **UDP Ping.** A UDP datagram is sent to a specific port on the target host. If the port is closed, the target may reply with an *ICMP Port Unreachable* message which will show that the host is alive. Since the UDP protocol is connectionless, no response from target will also indicate that the target is alive.

These techniques are not 100% accurate due to the fact that different operating systems may not comply with the requests and simply drop packets. Other reasons of failures can be because of a firewall or a router that is present that drops these packets.

Inverse mapping is another method to find alive hosts on a network. A firewall or router that drops ping packet requests will not respond with an ICMP echo reply message if the host is alive, but most firewalls and routers reply with an ICMP host unreachable packet if the target is not alive. This means that all routers that do not send an ICMP host unreachable message are showing that a host may be present on the network and that the router or firewall is dropping all echo requests.

### 4.2.2 Port Scanning

The second step in the network scanning process is to use the list of active hosts and try to determine which ports are open and which services they are running. This is known as port scanning. It can be compared to a burglar checking all possible entrance points of the targeted house. The most commonly used port scanning techniques are:

- **Connect Scan.** This method performs a full 3-way-handshake and tries to connect to an open port of the target by establishing a connection. These scans are easily detected and logged. As soon as a connection is established, the scanner tears down the connection in a graceful way (using FIN packets).
- **Half Open Scan.** This scan does not complete the 3-way-handshake of the TCP connection. The method is also known as the **SYN scan** and is the most common port scan. When the scanner receives the SYN/ACK segment response from a listening port, it will respond back with a RST segment and tear down the connection before the 3-way-handshake is completed. Since the connection is not established, it is not logged by the host's endpoint application, therefore it is considered as a stealthier scan. There are still intrusion detection systems and firewalls that can detect these types of port scans.
- **Stealth Scan.** Stealth scans use a variety of different scan settings to evade logging or detection. These scans utilize different TCP flag settings and IP fragmentation. Some of these scans are mentioned in more detail in the next chapter.

### 4.3 Operating System Detection

Knowing the operating systems of the computers that are being scanned is an important step in the vulnerability scanning process. After having successfully pinpointed what operating system the computer is running, it is possible to filter out from this information possible attacks or vulnerabilities that the system may be vulnerable to.

Operating system detection is also called operating system fingerprinting. Sometimes, one has the need to scan for specific machines such as routers, wireless access points, telephone PBXs, game consoles, Windows desktops or UNIX servers, other times performing fine grained scans to distinguish between specific operating system versions. This can be accomplished by using different techniques and its duration can vary. Fingerprinting can be done passively or actively. In the active operating system detection process, the scanner sends several packets to the target with various packet settings. These settings usually involve various types of flags set in TCP packets or flags in ICMP messages. Operating systems have different characteristics implemented in their TCP/IP stacks and can give different responses to different requests. Some of these characteristics are differences in their TCP window size. These responses are analyzed and compared to a database of known responses, also called signatures. If an exact signature is found in the database, the found operating system name will be returned to the user; otherwise a list of probable operating system names will be returned if the signature partly matches.

The passive operating system fingerprinting tools do not send any data to the target but only monitor the communication. This is a stealthy method of operating system fingerprinting and it is very difficult to detect its presence, but it does not provide as accurate results as the active procedure. The active operating system fingerprinting procedure interacts with the operating system in a way that is not a common user behavior that identifies the operating system more accurately.

# 5. NMAP

## 5.1 Introduction

Nmap, originally written by Gordon “Fyodor” Lyon, is one of the most popular network scanning tools available. The debut release of Nmap was in September 1997, when Fyodor posted an article in the Phrack magazine. The article created an avalanche affect where many security enthusiasts got an interest in the security tool. Nmap’s popularity was proved by its use in one of the most innovative movies of all time, Matrix Reloaded.

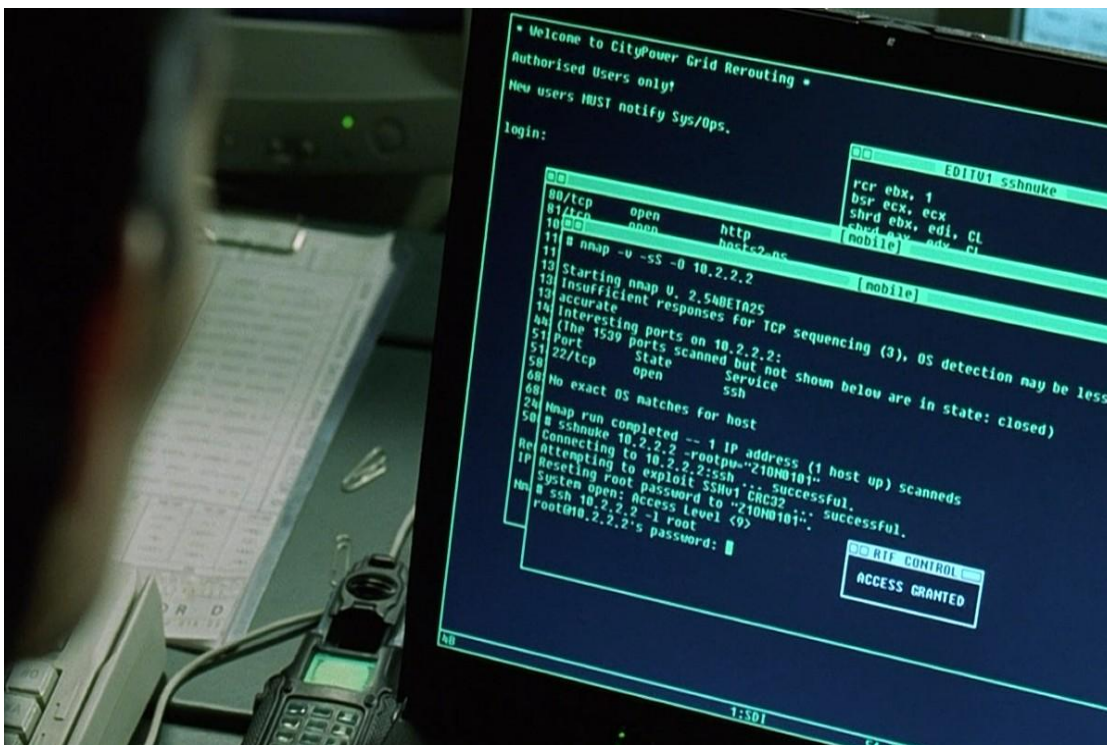


Figure 7 Image from movie Matrix Reloaded.

The name Nmap, derived from “Network Mapper”, can be used to discover machines connected to a network and thus creating a “map” of the network. It is an open source tool, freely distributed under the GNU Public License and available for all major operating systems. Nmap has the ability to make OS detections, scan whole networks or specific hosts, perform service detection, port scanning and also conduct various evasion techniques to prevent detection. System administrators and security specialists use the tool for many reasons, among some listed below:

- Network inventory
- Managing service upgrade schedules
- Monitoring host or service uptime

### 5.1.1 Nmap's Features

Nmap has many features which makes it an attractive choice for anybody who needs a network scanner. It has the capacity to perform simple scans such as ICMP ping scans to detect alive hosts, to more complex scans doing fine-grained OS detection. It gives the possibility to scan a huge IP address space and save the results to different formats.

### 5.1.2 Nmap's User Interface

Nmap is mostly used as a command line tool, this is due to the possibility for UNIX users to conduct scans remotely, create scripts and utilize the power of Nmap in combination with different scripting languages. Although having the command line basis, Nmap offers a graphical user interface called Zenmap. The graphical user interface is popular nowadays as governments, enterprises and other businesses are forced to map their networks to find security issues. Nmap's GUI popularity was also highly influenced by internet worms; administrators needed a way to easily localize infected computers.

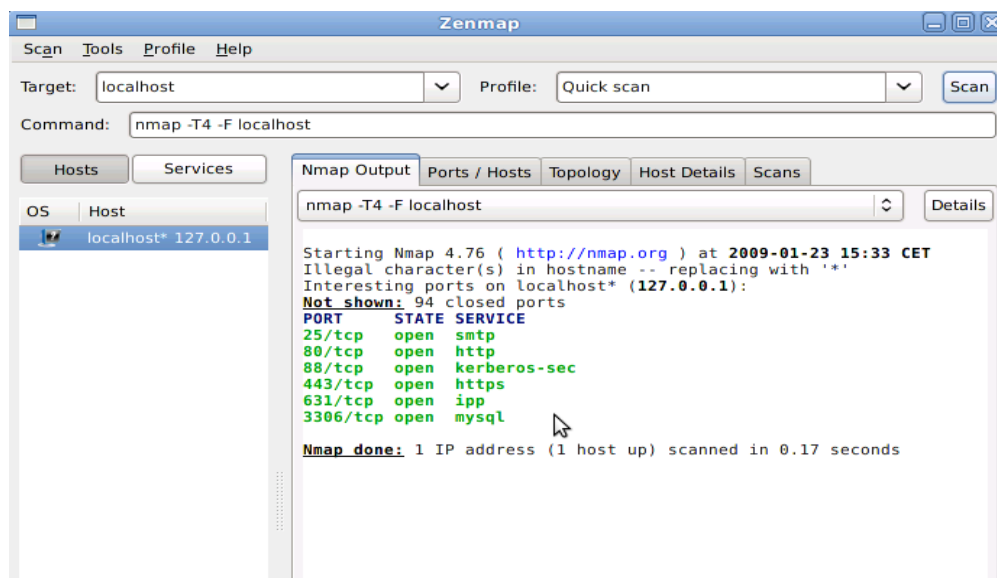


Figure 8 Zenmap

## 5.2 Nmap in Enterprises

Many other factors arise when network scanning is conducted in larger enterprises. Nmap's attributes - being an open source, free and well known tool - are a good reason for IT administrators to select it, but in many enterprises the administrators need special approval for using such a tool. What attracts enterprises are vendor support, maintenance agreements, how well it produces results and also a sense of assurance that the tool itself can be trusted. Nmap being a well used tool and having gone through extensive testing are usually enough reasons for most enterprises to utilize the tool.

### 5.2.1 Compliance Testing

One of the most important aspects in an organization's infrastructure is compliance testing of their applications and network. Compliance testing can be explained as various tests against a system or network of systems to make sure that all applications and operating systems within this network have certain configurations and policy settings set, according to specific rules and regulations enforced by the enterprise. These rules and regulations within security terms are called security policies. A simple example would be that no FTP servers are allowed to run within the intranet. To make sure that these rules are followed, a compliance test is done by using a network scanner such as Nmap that scans the entire network for FTP services. If no FTP services are found on the network, the compliance test is passed and it is proven that the security policy is protected.

There are many different types of compliance tests and some examples are:

- Port scanning on the interface of a firewall to find open ports.
- Scans against workstations to determine unauthorized applications.
- Finding systems with file sharing ports open.
- Locating unauthorized operating systems and printers.
- Other rules specific to the organization.

An inventory map can be constructed by scanning for servers or workstations and making sure that they are running. This can be used for troubleshooting remote servers to make sure that specific services on specific servers are really running. Nmap can perform scans against a whole subnet of IP addresses and within minutes determine which hosts on the network are alive.

The common steps used to do a more in depth scanning with Nmap against a range of IP addresses is to first determine which hosts are alive, then utilize the more advanced scans such as OS fingerprinting, i.e. to determine which operating systems these hosts are running. This will reduce the total scanning time by eliminating a set of IP addresses on the network which are not allocated by any machine.

Below is an example of a host discovery scan to detect if the host is alive.

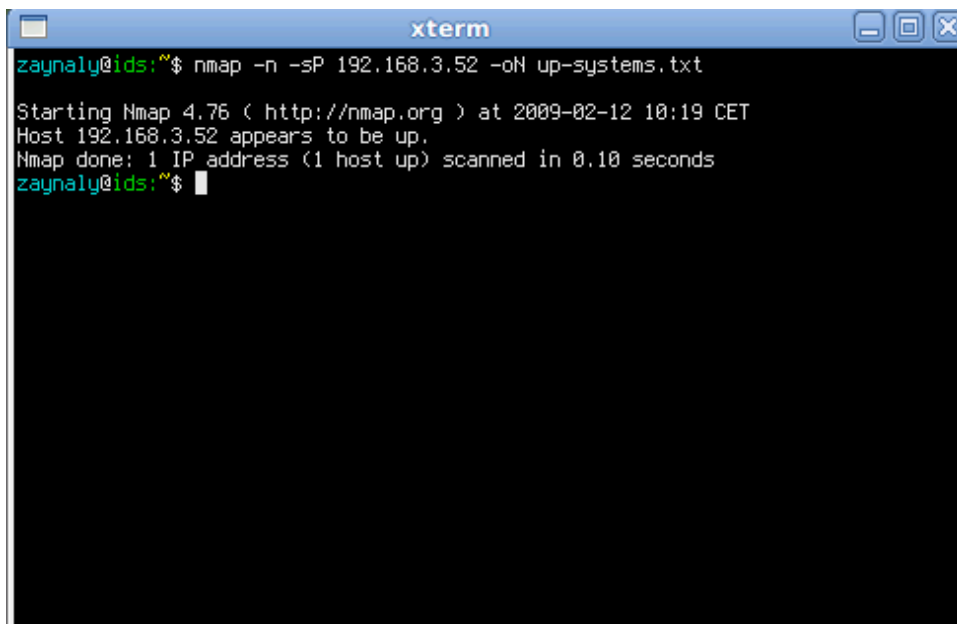
A screenshot of an xterm window titled 'xterm'. The terminal shows a user prompt 'zaynaly@ids:~\$' followed by the command 'nmap -n -sP 192.168.3.52 -oN up-systems.txt'. The output of the command is displayed in green text: 'Starting Nmap 4.76 ( http://nmap.org ) at 2009-02-12 10:19 CET', 'Host 192.168.3.52 appears to be up.', and 'Nmap done: 1 IP address (1 host up) scanned in 0.10 seconds'. The prompt 'zaynaly@ids:~\$' is shown again at the bottom of the output.

Figure 9 Nmap `-sP` scan

The Nmap flag `-n` informs Nmap to skip any DNS lookups of the hostname, it will decrease the total scanning time as the hostnames are not relevant in this test. The flag `-sP` stands for pings scan and will simply ping the host to find out if it is alive or not. The flag `-oN` tells nmap to output the information to a normal text file called `up-systems.txt`.

To be able to use this text file of active hosts again with Nmap, it would be easier if the output were more structured. Nmap will require a list of IP addresses where each line contains an IP address, or a list of IP addresses where each address is separated by a space character. This can be achieved by first outputting the scan result to a text file using the `-oG` flag, which is the greppable format. By using the greppable format, it becomes much easier to manipulate the text with UNIX tools such as `grep`, `cut`, `awk` or `sed`. The formatted text file can later be used by Nmap as a target list with the `-iL` flag.

Writing these types of scripts and automating the whole procedure is very common among administrators and has many benefits. The scan can be performed remotely and without any human intervention.

## 5.3 Optimizations

Optimizations are very important regarding network scanning. A network can consist of 10 – 100 000 network devices; this means that a scan can take anything from a couple of minutes to days depending on the network and the scan parameters.

Optimizations in the scanning process should follow a simple logic and the scan can be divided into many scans instead of trying to do everything in one massive scan. Redundant information should be disabled and only necessary scan settings should be enabled. Any information regarding the network should be used if available. An example follows:

A network of moderate size with different operating systems exists. We want to locate and perform a port scan only on the workstations on the network. We know in advance that all workstations in this network are running the operating system Windows XP. Using this information we start to create our steps for the scan. Usually not all IPs are allocated on the network and the first step would involve finding all alive hosts and create a target list. Hosts running Windows XP have by default the firewall enabled and it would block any ping attempts to determine if the host is alive. We also know that all hosts running Windows 2000 server, Windows 2003 and Windows XP have TCP port 445 (SMB over TCP) open. Using this information we can create our nmap scan with optimal parameters:

```
nmap -sS -n -PN -T5 -p 445 --open -oG outputfile.gr <networks to scan>
```

With the `-T` flag, different values between 1-5 can be given and the value 5 is the fastest scan possible. This flag is a template for different timing settings in Nmap. The template names that exist are `paranoid` (0), `sneaky` (1), `polite` (2), `normal` (3), `aggressive` (4), and `insane` (5). The Sneaky and Polite templates (0 and 1) are normally used for IDS/Firewall evasion. What really interest us here are the aggressive or insane scans. These aggressive modes require having a fast and reliable network. Though, with the insane template it can be expected to sacrifice some accuracy for speed.

1. Extract and create a target list from Nmap's output file containing only IP addresses of alive hosts with TCP port 445 open. Name the file `target.txt`.
2. Make a new Nmap scan to find open ports from the alive target list:

```
nmap -sS -PN -n -T5 --open -iL target.txt -oG results.txt
```

In the above example, default ports are chosen, specific ports can be chosen instead to reduce the scanning time even more. The Stealth scan flag `-sS` will be explained in more detail later in this chapter.

Nmap has numerous parameters and options, the perfect settings are unique to each network and its characteristics. Nmap's manual is a great resource for finding different optimization settings.

## 5.4 Examples

Inexperienced users and script kiddies try to solve every scanning problem with the standard syn scan, but experts choose the most appropriate scan to reach their goals. That is to try to achieve the most correct results and save time. Some of these scans require root privileges since Nmap needs to send and receive raw packets, which requires root privileges in UNIX systems. Some of the most common scans will be explained briefly:

### 5.4.1 TCP Connect Scan (`-sT`)

This is the default scan in Nmap when the user does not have root privileges. Nmap will use the operating system's "connect" system call to try to connect to the target host instead of using raw packets. This scan is not time efficient compared to when Nmap creates its own raw packets. It is slower because Nmap is trying to perform a full three-way handshaking process for each port.



Further suspicion is also increased as a full connection is more likely to be logged by the target's application.

### 5.4.2 TCP SYN Scan (-sS)

When the user has root privileges the default scan is automatically set to the TCP SYN scan. The TCP SYN scan uses a faster method for detecting open, closed and filtered ports. This scan is also the most popular scan as it is fast and not blocked by stateful firewalls compared to a TCP ACK scan. The speed of this scan comes from the fact that it does not complete a full TCP connection, hence that is why it is also referred as the half-open scan. The scanning method works by sending a SYN packet, if a SYN/ACK response is observed from the host then the port is registered as open, if a RST response is observed then the port is closed and if no response is observed after multiple SYN packets are sent, the port is registered as filtered.



```
zaynaly@ids:~$ nmap -sS 192.168.3.0/25
You requested a scan type which requires root privileges.
QUITTING!
zaynaly@ids:~$ sudo nmap -sS 192.168.3.0/25
[sudo] password for zaynaly:

Starting Nmap 4.76 ( http://nmap.org ) at 2009-01-23 15:48 CET
Interesting ports on 192.168.3.20:
Not shown: 996 closed ports
PORT      STATE SERVICE
22/tcp    open  ssh
23/tcp    open  telnet
80/tcp    open  http
443/tcp   open  https
MAC Address: 00:08:02:0F:40:C8 (Hewlett Packard)

Interesting ports on 192.168.3.21:
Not shown: 999 closed ports
PORT      STATE SERVICE
22/tcp    open  ssh
MAC Address: 00:90:27:BE:E0:F6 (Intel)

Interesting ports on 192.168.3.22:
Not shown: 999 closed ports
PORT      STATE SERVICE
22/tcp    open  ssh
MAC Address: 00:90:27:7D:99:E4 (Intel)

Interesting ports on 192.168.3.51:
Not shown: 987 closed ports
PORT      STATE SERVICE
22/tcp    open  ssh
80/tcp    open  http
3000/tcp  open  ppp
MAC Address: 00:80:9D:96:F6:03 (Hewlett Packard)

Interesting ports on 192.168.3.52:
Not shown: 997 closed ports
PORT      STATE SERVICE
80/tcp    open  http
88/tcp    open  kerberos-sec
443/tcp   open  https

Interesting ports on 192.168.3.53:
Not shown: 996 filtered ports
PORT      STATE SERVICE
135/tcp   open  msrpc
139/tcp   open  netbios-ssn
445/tcp   open  microsoft-ds
5357/tcp  open  unknown
MAC Address: 00:0F:FE:7C:93:B9 (G-pro Computer)

All 1000 scanned ports on 192.168.3.54 are filtered
MAC Address: 00:17:A4:44:B3:A3 (Hewlett Packard)

Nmap done: 128 IP addresses (7 hosts up) scanned in 41.59 seconds
zaynaly@ids:~$
```

Figure 10 Nmap's TCP SYN scan

### 5.4.3 TCP NULL, FIN, and Xmas Scans (-sN; -sF; -sX)

According to the TCP RFC 793 (Request for comments) any packets not containing the SYN, ACK or RST flag will result in a RST response if the port is closed and the target will not provide with any response if the port is open. These three scans take advantage of these specific characteristics of the TCP stack.

- TCP NULL (-sN)  
No flags are set in the TCP header.
- TCP FIN (-sF)  
The FIN flag is set in the TCP header.
- TCP Xmas (-sX)  
PSH, URG and FIN are set in the TCP header.

These scans are considered stealthy and may pass firewalls and other security controls such as packet filtering routers.

### 5.4.4 Other Scans

Other scans that are available, but not covered by this thesis report are:

- -sU (UDP Scan)
- -sA (TCP ACK scan)
- -sW (TCP Window scan)
- -sM (TCP Maimon scan)
- -sO (IP protocol scan)
- -sI *<zombie host>*[:*<probeport>*] (idle scan)
- --scanflags (Custom TCP scan)

## 6. Nessus

---

Nessus is a free vulnerability assessment tool from Tenable security. Its purpose is to find potential vulnerabilities in computer systems. A vulnerability is either a programming error or a misconfiguration that could lead to control of the computer system or access to sensitive information; that also includes weak passwords or unpatched programming flaws.

The reason for performing a vulnerability assessment is to discover which systems are vulnerable and start a patching process. The cycle of doing a security assessment, patching and verification is an ongoing process in many organizations.

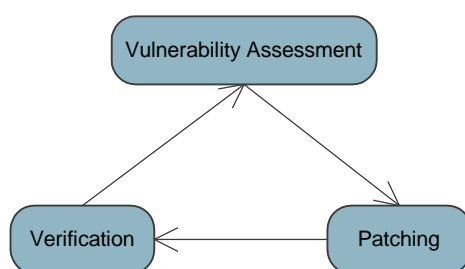


Figure 11 Ongoing security process

Nessus was created 1998 by Renaud Deraison. Before the creation of Nessus, the only open-source vulnerability scanner that were available was SATAN (Security Administrator Tool for Analyzing Networks), which had outdated security checks. There were several commercial vulnerability scanning products on the market though, but they were very expensive. Nessus was licensed in GPL in the beginning and was totally open and free.

In October 2005, Tenable security announced that their license for their new Nessus 3 engine would change from GPL to a new proprietary license. The new Nessus 3 engine is still free for non commercial use and can receive weekly updates. If you are using the new version of Nessus for commercial use, new updates will be sent to you as soon as they are released.

Nessus major strengths are:

1. Nessus client/server architecture
2. Nessus plugins
3. Nessus Knowledge Base

Nessus is based on a client/server architecture, which makes it very flexible to use. Its architecture makes scans scalable, manageable and more precise. Before its release, the vulnerability scanners on the market were based on a single client where all the security checks were done from. It was important to find a good location in the network from which all hosts were reachable with a good bandwidth. The scan was limited from a single point in the network and was further limited based on processing power. The more processing power and memory that exists, the less time it would take to conduct the vulnerability scan.

Scanning from a single point introduces problems like bandwidth spikes in specific areas on the network. Being located behind a NAT (Network Address Translation) could be problematic as the vulnerability scanner initiates many connections to its targets, which could fill the address translation table and making it unusable for other computers that needs to go through the same NAT gateway. The same problem applies for firewalls that could block all the security checks for the vulnerability scanner. By deploying a Nessus server that performs the scanning located outside the firewall, we can circumvent that problem.

Security scanners need an updating feature to be able to make tests against the newest vulnerabilities. Nessus security checks are handled by its plugins. The plugins are written in Nessus own scripting language, called Nessus Attack Scripting Language (NASL). NASL makes it easy to create customized security checks. The scripting language is similar to the C programming language but is less complex and it provides you with a lot of pre-built functionality.

All results that are discovered by the plugins are saved in Nessus internal database, called Knowledge Base. The Knowledge Base is a place for storing information found by the plugins, so that subsequent plugins can reuse information. For instance, if a plugin finds a specific version of an OpenSSH server on a target, a subsequent plugin that makes security checks against OpenSSH do not need to find out again if the target is running SSH and what application and version is running behind the service. This feature increases the scanning speed and reduces the amount of traffic generated on the network. Figure 12 shows how an interaction between Nessus plugins and the Knowledge Base could be carried out.

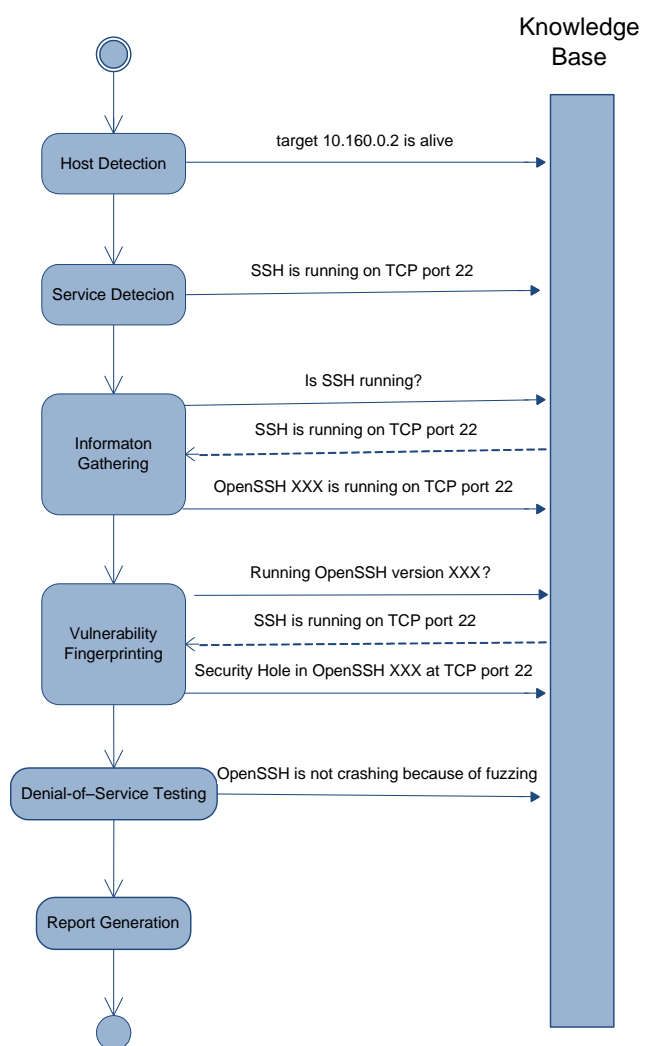


Figure 12 Nessus plugins interaction with the KB

## 6.1 Nessus Scanning Stages

Nessus divides its process for detecting vulnerabilities into different stages. Each stage is dependent on the success from the previous stage. By having this layered and modular approach (Figure 13), it extends the functionality for having more different types of plugins per stage that are built on plugins from previous stages. Each step in the scanning process will be described below.

### 6.1.1 Host Detection

In the first stage, Nessus tries to identify targets that are “alive”. There is no reason to scan targets which are “dead”. There are three ways to tune the host detection plugin to identify if the target is alive or not. The three techniques are:

1. ICMP echo request messages
2. TCP SYN segments
3. None, treat the target as always alive

The famous ping tool is using the first technique to determine if a host is alive or dead. When Nessus is using this method, it sends an ICMP echo request message to the target and if the target responds with any type of message,

Nessus will treat the host as alive.

Unfortunately, many firewalls drops ICMP messages (does not return any packet back to the user) which makes it sometimes impractical to use. Sending a TCP segment with the SYN flag set to specific ports mitigates that problem by tricking the firewall that a normal connection will be established. This technique is a more accurate way to determine if the host is alive, but it requires that you can guess if a port is open in advance.

### 6.1.2 Service Detection

In the service detection stage, Nessus tries to identify what services are running on the target. Nessus has the possibility to use several port scanners. From the well know Nmap port scanner to the built-in Nessus port scanner. All ports that are found by the port scanner are saved in the Knowledge Base to be used of subsequent Nessus plugins. When the complete list of ports are found, Nessus launches its service detection plugins to try to determine what services are running behind the ports.

### 6.1.3 Information Gathering

During the third stage, Nessus is using the information based on the previous stages to conduct information gathering about each host and its services. The techniques used here are service querying, application fingerprinting and general banner grabbing. Even though this stage is called information gathering, it can still find vulnerabilities. The only difference is that all the plugins in this category are not intrusive and cannot cause damage to the target that is being scanned.

Some plugins in this category are used to retrieve information to conduct other vulnerability checks, like the NetBIOS and registry plugins. These plugins could enumerate all user accounts and save these usernames in the Knowledge Database so subsequent plugins can try default passwords.

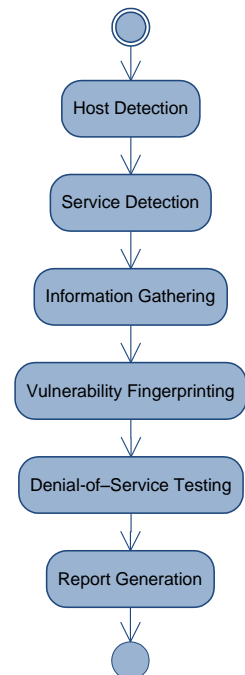


Figure 13 UML state diagram of all stages in Nessus

Besides trying to find vulnerabilities from the network by probing ports, Nessus has the ability to log into the target and access system resources to be able to perform local security checks.

### 6.1.4 Vulnerability Fingerprinting

When the information gathering stage has saved all of its results in the Knowledge Base, the attacking stage starts. The attacking plugins are divided into three categories:

- ACT\_ATTACK,
- ACT\_MIXED\_ATTACK
- ACT\_DESTRUCTIVE\_ATTACK

The reason of dividing them into categories is because of its degree of intrusiveness.

The plugins in the ACT\_ATTACK category query actively the service to find out a particular vulnerability but not to the extent that it could cause it to crash.

“Safe checks” is a setting in Nessus to “guaranty” that the scans will not break any service or system. The second category (ATTACK\_MIXED\_ATTACK) of plugins uses a mixed approach. If “safe checks” are set in Nessus and the corresponding plugins in that category are selected, the plugins will run to an extent that cannot cause any harm to the system. Though, if the “safe checks” are not set, Nessus cannot guaranty that the service will still be running after the test. By having “safe checks” turned on, the result from the vulnerability assessment will not be as thorough and correct as it could be.

When performing a penetration test, it is recommended to have “safe checks” turned off as the vulnerability assessment will be as correct as possible. If the system or its services crashes during the scan, breaking the application is a proof in itself that it is unstable or insecure. Having the option “safe checks” turned on is a safer choice when performing recurrent vulnerability assessment scans on production servers during business hours, as many security checks exercises the application in ways that are not normal user behavior.

The last category (ACT\_DESTRUCTIVE\_ATTACK) conducts hostile security checks that could lead to crashing of services, account lockouts and other destructive behavior. If the “safe checks” are turned on, none of these security checks would run.

### 6.1.5 Denial of Service Testing

Nessus has also the ability to perform Denial of service attacks. It is the last security stage against the target, because in case of a service crash, subsequent plugins will not be able to perform its security checks. The security checks that exist in this stage are plugins that try to kill the service by exploiting programming flaws or by flooding the system until it is impossible to use.

### 6.1.6 Report Generation

All results from the Nessus plugins are stored in the Knowledge Base where subsequent plugins can reuse information based on earlier security checks. This makes the whole concept of plugins more usable and extendable. During the scan, a Nessus result file is created, which will be the report file that the Nessus user can review. This report file is also called a NBE file, which is the suffix of the filename. By default, the Knowledge Base is not saved and removed after each scan. The Knowledge

Base contains a lot of useful information, especially in case of debugging and for audit purposes. The report that is generated is only a subset of all information that exists in the Knowledge Base.

Nessus is a powerful and flexible vulnerability scanner and the combination of these attributes with its free cost made it very popular. Many organizations today use Nessus for their in-house security audits to find in which security state their organization is in. In organizations with hundreds to thousands of different servers that provide services, it is a nightmare for the administrators to be aware of all current vulnerabilities, especially nowadays where new vulnerabilities are being disclosed every day. Nessus provides up-to-date security checks against new vulnerabilities and it is an excellent tool for the administrators to see where they need to improve their security.

## 7. Enterprise Scanning

---

Enterprise scanning is a complicated topic compared to a single conducted scan, as there are many components that could give misleading information or do harm to the infrastructure. The main goal of the enterprise scanner is to have it running without any user interaction and retrieve vulnerability reports. There are some factors that distinguish an enterprise scanner:

- Easy administration
- Periodic scanning
- Report correlation
- Automated updating

Many organizations are divided into different service departments, where each department has a specific responsibility. A common practice is to have a unique department for each operating system that the organization is supporting. It is important to tell each department/business area that there will be frequent vulnerability scans conducted, as the scanner will leave traces in system log files, intrusion detection systems and firewalls. An administrator who is unaware of the scanning process can easily by looking on log files, think that an attempt to break into the systems has occurred and report an incident.

When performing a vulnerability scan, there is always a struggle between the risks involved and the possible benefits of allowing the risks. The two major types of risks involved are producing a denial of service attack or missing valuable information. A vulnerability scan can crash systems, confuse the network and conduct massive user account lockouts if the security checks are too intrusive and hostile.

It is possible to optimize the scanning process by performing its security checks based on banner grabbing, which is the process of extracting version numbers and application names. This will reduce the overall scanning time, but it will not be as thorough as possible, which could lead to more false-positives. Sometimes a patch for fixing a vulnerability will only solve the vulnerability but not provide a new version number for the application. This makes it even more difficult for the banner grabbing process to detect if the system is vulnerable to a specific vulnerability as it cannot distinguish the different version of that application based on the version number.

A good practice to configure the vulnerability scanner and verify if it is correctly set is to have a lab that reflects the organization production system with all of its settings and configurations. It is very difficult to foresee all problems that a vulnerability scanner could cause, due to the total complexity; a system crash in the lab does not necessary mean that the same crash will happen in the production system.

In our own experience, a good way to start the enterprise vulnerability scan is to start with a small pool of computers and successively increase the number of hosts for each new scan and verify after each scan that all hosts are in a normal state, i.e. not broken.



## 7.1 Planning and Deployment

As larger organizations have hundreds to thousands of computers, conducting vulnerability scans on all computers are not feasible as the report will be bloated of all vulnerabilities and it will be difficult to identify the most critical ones. Therefore is it important to identify the most critical assets for the scanning, i.e., the systems that have the greatest impact on the organization if something bad happens. Creating a list of computers ordered with the more critical computers first is a good starting point to create a target list.

## 7.2 Network Topology

The network topology matters when you are doing your security scan. Scanning internal servers from the DMZ will provide different results compared to scanning the same servers from the internal network, because of firewalls and other filtering components. It all depends on the vulnerability state perspective, e.g. what can the attacker see from the DMZ into the internal network. Based on the results, we can prioritize vulnerabilities found on the network. If a security hole is found in a server scanned from the internal network which cannot be seen from the DMZ, the vulnerability could be less prioritized. It is therefore important to plan where the security scan should be conducted from.

Three common distributed network scanning topologies are:

- Star
- Flat
- Islands

The Island topology consists of scanning servers that are completely isolated from each other. It is the highest form of separation and there is no central management control of the servers.

Advantages of the Island topology:

- Information cannot be leaked between islands.
- Any problem from one vulnerability scan will not affect others.
- No additional firewall rules need to be set as the servers could be deployed in the same LAN segment as the targets.

Disadvantages of the Island topology:

- Additional maintenance costs are introduced as there is no central management point.
- No centralized server updates could be performed.
- Data from each scanning server cannot be correlated.

The Flat topology provides scalability compared to the Island topology. Still scanning servers are installed all

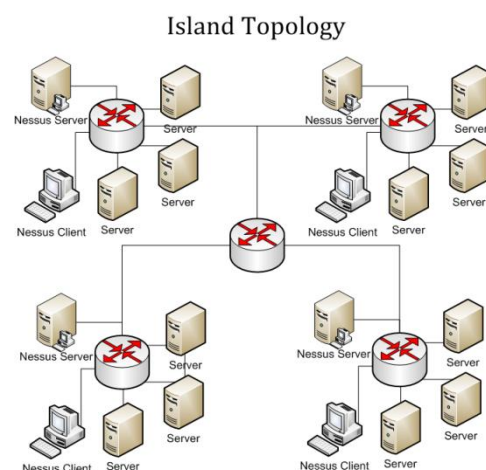


Figure 14 Island Topology

over the network but a vulnerability scan can be conducted from anywhere, which could lead to information disclosure between different network segments. The network is fully open between LAN segments a single scanning server can scan several LAN segments.

Advantages of the Flat topology:

- A single server can scan the whole network.
- Management can be done from anywhere in the network.
- A centralized update server can be used to update all servers in the network.
- Reports could be merged between different servers.

Disadvantages of the Flat topology:

- Information leakage between LAN segments.
- A single server can be used to scan the whole network. If it fails, the whole enterprise scanning process for all LAN segments will fail.
- A single server cannot provide different views, in case of firewalls.
- A single scanning server increases the bandwidth in specific parts of the network.

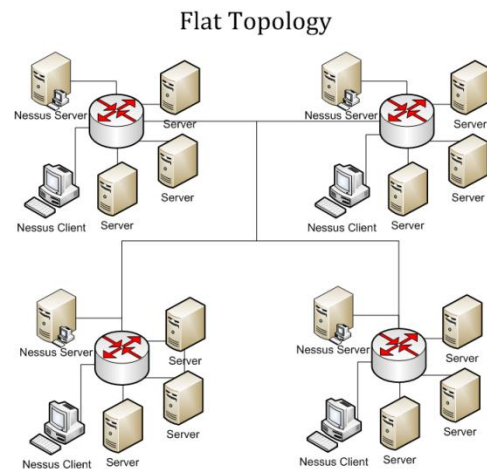


Figure 15 Flat Topology

In the Star topology, scanning servers are spread over the network, but are connected to a management network. All servers are controlled from a central point and data are exclusively transferred to this central point. There is no communication between the different scanning servers in the network.

Advantages of the Star topology:

- Information cannot be leaked between LAN segments.
- Bandwidth consumption is divided between servers.
- A centralized update server can provide updates to all scanning servers.
- Reports can be merged in the central server
- Data from each scanning server can be correlated.

Disadvantages of the Star Topology:

- Management can only be done from a single point
- Additional firewall rules need to be set so the servers can communicate in the management network.
- A single administrator will have the power to control all scanning servers in the organization.

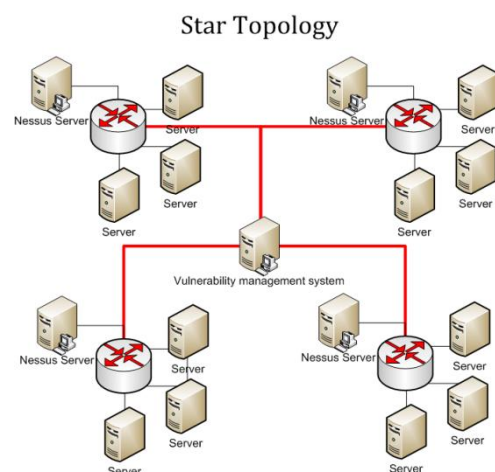


Figure 16 Star Topology

It is a good practice to have a dedicated machine that conducts the vulnerability scanning, because the risk for exploitation increases in a multiuser system, where every user could compromise the server. The server will contain information about vulnerabilities from the whole organization.

# 8. The Implementation of D.E.V.A.S

---

## 8.1 Introduction

The Distributed Enterprise Vulnerability Assessment Scanner (D.E.V.A.S) is a proof of concept application to conduct vulnerability security assessments. The application has been revised several times and initially started to operate in a single scanning mode, i.e., perform all of its security checks from a single host. The final implementation of D.E.V.A.S offers two modes of operation:

- Single scanning mode
- Distributed scanning mode

This section contains all the details from our implementation, the necessary settings and the problems that occurred during our work. The implementation of the single scanning mode application will be first explained and then the extensions that made it possible to operate in a distributed mode. The analysis and correlation tools of the result files will be addressed in the last part of this section.

## 8.2 D.E.V.A.S Configuration

In the end, all security checks are done by the Nessus application, therefore it is very important that Nessus is properly and safely configured. Many of our settings are recommended by Tenable, other settings regarding the performance we had to test and tune, as it is very environment specific.

### 8.2.1 Cron

One of D.E.V.A.S requirements is to run in an automatic manner without any need of human interaction. D.E.V.A.S utilizes Cron, “a time-based scheduling service in Unix-like computer operating systems”, to schedule the security scans against specified targets on a given time/date. The configuration file of Cron is called Crontab, which contains the details for the scheduled jobs. In our case, the Crontab file is setup as Figure 17:

```
# m h dom mon dow    command
0 21 * * * aptitude -y update && aptitude -y upgrade && aptitude -y dist-upgrade
&& aptitude -y autoclean
0 * * * * /usr/sbin/ntpdate ntpServer > /dev/null 2>&1
0 22 * * * /usr/sbin/nessus-update-plugins > /dev/null 2>&1
0 0 * * 5 /root/DEVAS/scripts/singleNessusScan.sh > /dev/null 2>&1
```

Figure 17 Crontab settings

The first entry schedules a daily check to update and upgrade our operating system files at 21:00hrs. The reason of updating the operating system daily is to have the latest security updates in the case a serious vulnerability has been released. This is fully automatic.

- The second entry schedules a time synchronization job that is run daily at midnight to update the local time with the NTP service. To have accurate timestamps is an essential aspect of the scanning exercise, both for auditing and accountability of the actions performed, as well as for troubleshooting purposes. If D.E.V.A.S system clock is skewed, the scans could run during business hours and generate denial of service attacks against the infrastructure.
- The third entry schedules daily updates of Nessus plugins at 22:00hrs.
- The fourth entry schedules the actual D.E.V.A.S security scan, which is currently run on a weekly basis, starting every Friday at 00:00hrs.

## 8.2.2 Nessus Configuration

The Nessus configuration files will determine which actions are to be performed by D.E.V.A.S and therefore it is of utmost importance that these are properly set and maintained. The Nessus server uses one configuration file and the Nessus client uses another one. The main difference regarding the scan features between these configuration files are that the activated plugins are also added in the Nessus client configuration file. But many of the configuration settings for the client can be overridden by the server configuration. It is important to check the server configuration file that a particular setting is correctly set. A hint is to check the client configuration file after a scan and see that the settings in the client configuration did not change. Which settings that can be overridden in the Nessus client configuration file are checked explicitly in both configuration files before the actual Nessus scan.

The following sections are a walkthrough of all the D.E.V.A.S relevant settings that can be found on the various tabs available in the Nessus graphical user interface tool.

## 8.2.3 Plugins

The plugin settings are updated dynamically by the *update-nessusrc.pl* script. Changing the plugin settings in Nessus graphical user interface will not cause a change because those settings will be overridden every time the *update-nessusrc.pl* script is running. This can be explained by the fact that we want to include and exclude plug-ins based on categories and other properties that they possess dynamically. To exclude a specific plug-in, one needs to add that in the *update-nessusrc.pl* script.

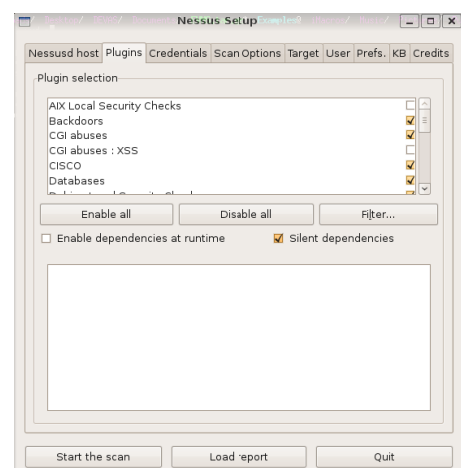


Figure 18 Nessus client Plugin tab

## 8.2.4 Credentials

There is a possibility in Nessus to supply credentials to the target. Nessus will then login and do local security checks on the host. The purpose of D.E.V.A.S is to scan as many targets as possible as thoroughly without breaking anything. Adding credentials will make the scan much slower and be more intrusive. If the purpose was to do a scan as thorough as possible, independently of the risk, then a more suitable choice would be a penetration test.

Any credentials should not be set in this tab. No plugins are selected for Local security checks; therefore there is no point to give any credentials.

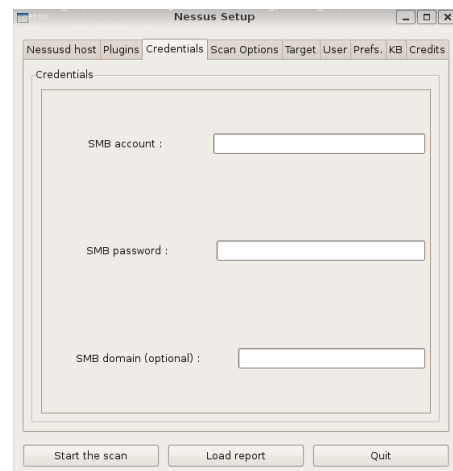


Figure 19 Nessus Credentials tab

## 8.2.5 Scan Options

The port range should be set to “default”, which means that Nessus will do a port scan of the ports declared in Nessus own service.txt file.

“Consider unscanned ports as closed” and “Optimize the test” should be set on. Nessus considers unscanned ports as open if the port has not been scanned. This will trigger many Nessus plugins to run against closed ports which will take a long time, in some occasions this option will make the scan less accurate. Based on the environment where the scanner is operating in, having “Consider unscanned ports as closed” is a worthy optimization.

“Number of hosts to test at the same time” and “Number of checks to perform at the same time” should be set to 40 and 5, respectively, which are the recommended settings.

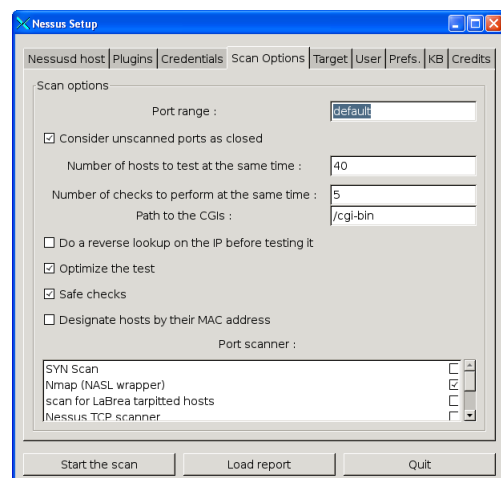


Figure 20 Nessus client Scan Option tab

“Safe checks” should be set on, otherwise Nessus will perform more intrusive checks that could damage some systems.

## 8.2.6 Prefs

In the TCP scanning technique section, “SYN scan”, “Identify the remote OS” and “Use hidden option to identify the remote OS” should be set to enable Nmap OS fingerprinting.

In the Timing policy section; “Aggressive” and “Run dangerous port scans even if safe checks are set” should be enabled for running Nmap with the right parameters. The rest of the options in this tab should be the default ones.

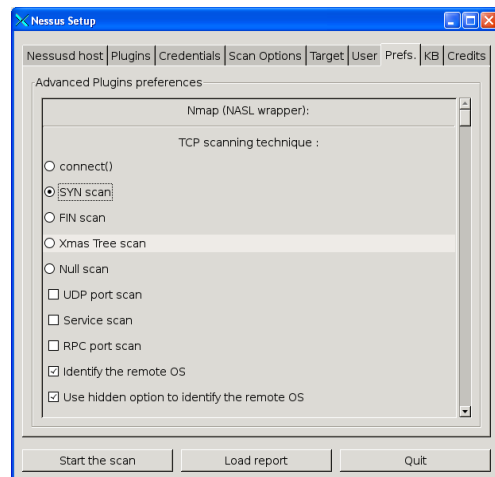


Figure 21 Nessus client Prefs tab

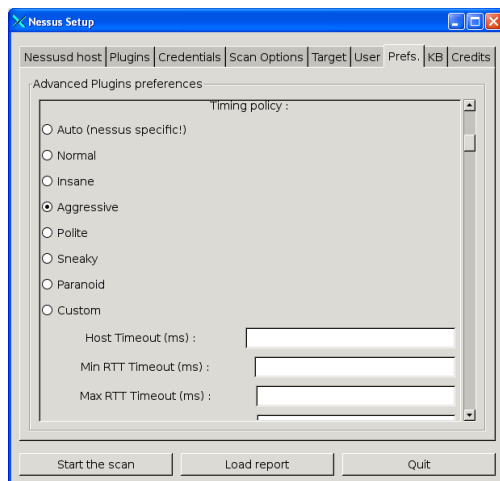


Figure 23 Nessus client Prefs tab 2

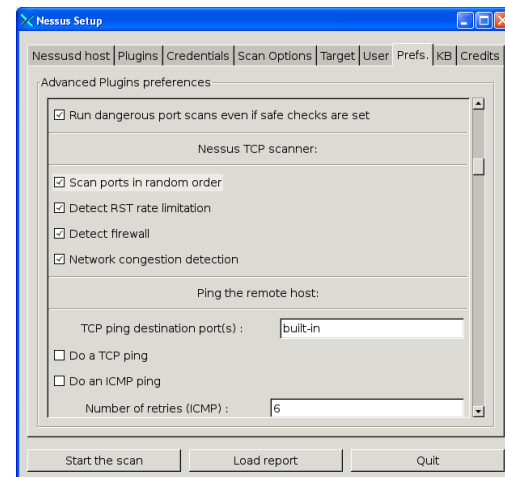


Figure 22 Nessus client Prefs tab 3

## 8.2.7 KB

KB stands for Nessus Knowledge Base, which is a feature for saving scan results for use in the future. Reusing scan results for future scans, like open ports and alive hosts will not provide an accurate up-to-date snapshot over the current fixed and new vulnerabilities. However, enabling “Enable KB saving” for “Test all hosts” is a must for post auditing. The past scans should not interfere with the new scans, therefore “Max age of saved KB” should be set to a value that is less than 7 days and to a value that enables us to review the results, especially in case of an incident. Therefore the value of that option is set to 432000 seconds, i.e. 5 days.

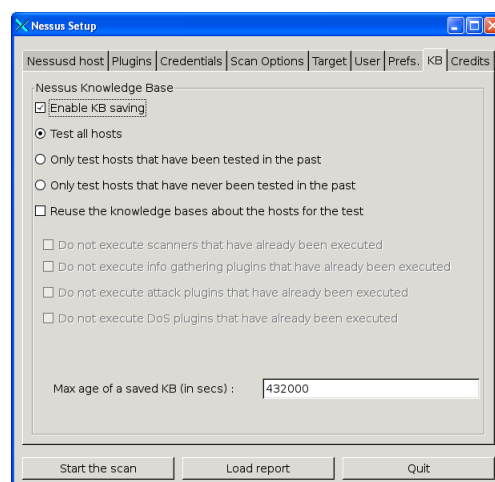


Figure 24 Nessus client KB tab

## 8.3 Application Overview

This section provides a detailed description of the various components that came into play for D.E.V.A.S to function. In the end of this section, all steps of D.E.V.A.S's single scanning mode are illustrated with a UML state diagram.

The main script controlling the whole vulnerability scanning process is called *singleNessusScan.sh*. As already mentioned, D.E.V.A.S is automatically initiated at the pre-determined time/date as specified in the Crontab configuration file. Once active, the application will sequentially perform the processes identified in Figure 32, i.e., it will download and install any newly available Nessus plugins, update the configuration file, identify the hosts to scan, launch a Nessus vulnerability scan on the targeted hosts and send a report by email containing new vulnerabilities. Each process is described in more detail hereunder.

### 8.3.1 Update Plugins

In order to produce reliable security assessment results, the Nessus scanning engine must download the most recent updates and plugins, which is automatically performed during the first step. Initially, *singleNessusScan.sh* calls the *nessus-update-plugins* application to download new Nessus plug-ins from an Internet repository. When the new plugins have been fetched, our embedded perl script updates the configuration file and restarts the *nessusd* server to successfully activate the new plugins. The client configuration file contains a list of all the plugins used by Nessus during scanning, with a corresponding entry indicating whether the plugin is enabled or disabled, like in Figure 25.

```
begin (PLUGIN_SET)
10001 = no
10002 = no
10003 = no
10004 = no
10005 = yes
10006 = no
10007 = no
10008 = yes
10009 = yes
10010 = yes
```

Figure 25 Nessus plugin configuration file settings

Alternatively, one could just utilize the native built-in *nessus-update-plugins* program for this purpose. However, this is not recommended because it will require manual intervention since the downloaded plugins will not be automatically inserted into the configuration file. The end result is that the new plugins will not be used by D.E.V.A.S during the vulnerability scans unless it is manually inserted in the configuration file or all scanning is performed using the graphical user interface (which is not the case in the automated version). The same *update-nessusrc* file is still being utilized by D.E.V.A.S and is embedded in the *scripts/lib/updateSingleScanPlugins.sh* file. Furthermore, using the *update-nessusrc.pl*, we are in a better position to stipulate which plugins should be activated. The script will deny plugin activation for the following categories:



- DOS attacks
- destructive attacks
- floodings
- kills the OS
- scanners (except Nmap)

The following plugin categories have been removed:

- Brute force attacks
- CGI abuses
- CGI abuses : XSS
- Denial of Service
- AIX Local Security Checks
- Debian Local Security Checks
- Fedora Local Security Checks
- FreeBSD Local Security Checks
- Gentoo Local Security Checks
- MacOS X Local Security Checks
- Mandrake Local Security Checks
- Red Hat Local Security Checks
- Solaris Local Security Checks
- SuSE Local Security Checks
- HP-UX Local Security Checks
- Ubuntu Local Security Checks
- CentOS Local Security Checks
- Slackware Local Security Checks
- Default Unix Accounts
- Windows, Windows : User management

Most of the disabled plugins listed above run local security checks that require a local privileged account on each target system. Given that we have user credentials, Nessus logs in remotely and checks local applications for configuration errors or vulnerabilities. In our case, that is not feasible due to more intrusive scans and time spent on scanning. The more plugins enabled during the scan, the longer it will take for the scanning process to complete and the more memory is consumed. The SNMP LANMAN enumeration plugins are also disabled, to avoid enumeration of Windows users and shares since this could result in the locking of user accounts. *nessus-update-plugins* will not function correctly unless the */etc/nessus/nessus-fetch.rc* file contains the correct proxy settings to connect to Internet.

Once new plugins are downloaded and the updating of the configuration file is done, D.E.V.A.S will overwrite *nmap.nasl* and *os\_fingerprint.nasl* in the local Nessus repository with our modified *nmap.nasl* and *os\_fingerprint.nasl* scripts to enable and use the OS detection tag from Nmap which is described in later sections.

### 8.3.2 Blacklisting

Scanning hundreds to thousands of targets can be cumbersome if one needs to add every target's IP to a file. More often, the organization wants to prioritize the vulnerability scanning against targets that possess valuable or confidential information and not scan regular desktop computers. For this reason, the blacklisting process will filter out all unwanted computers for the security assessment that meets a certain criteria. Our proof of concept, is to filter out all standard installation computers that are running Windows XP.

Our first attempt was to use Nmap's capabilities of OS fingerprinting and then based on the result filter out computers that were running Windows XP. The average scan time for determine the operating system in our working environment was 50.2 seconds which is an unfeasible time to filter out machines (the long duration was mainly because of layers of firewalls between the scanner and the target). It would take more than 7 hours to filter 1000 computers. Nmap also generates a lot of traffic to determine the operating system. Instead we tried to use the Xprobe2 application which was faster and generated less traffic. The problem with the xprobe2 application was that its OS fingerprinting method is based on ICMP messages which are often blocked by routers and firewalls. It was also less accurate.

The last attempt was to use the less known SinFP application for OS fingerprinting. SinFP needs to know a single TCP port that is open to determine the operating system. By default, Windows XP has the TCP port 445 open which is the Microsoft-DS service (SMB over TCP), which we used.

By running SinFP against a target that has TCP port 445 open, the total response time was in average 0.8 seconds; this result was a huge improvement compared to Nmap's 50.2 seconds. SinFP only sent three packets to each target, which reduced the amount of traffic sent on the network. Performance problems arose if the targets were down or the TCP port 445 was closed, which delayed the response for three seconds. See Table 1 for the results.

OS application:	detection	Number of packages sent	Average OS detection response time in seconds:
Nmap		20	50
Xprobe2		4	2
SinFP		1-3	0.8

Table 1 OS fingerprint comparisons

Our final solution was to pre-scan the hosts with Nmap's capability of port scanning, to see if the TCP port 445 was open. Nmap sends out a single TCP segment with the SYN flag set to port 445 and if the target responds back with a TCP segment with the SYN+ACK flag set before the timeout, then the target was qualified to be scanned by SinFP. If SinFP found out that the target was running Windows XP then it would be blacklisted. In Figure 26 below is a UML state diagram showing the filtering process.

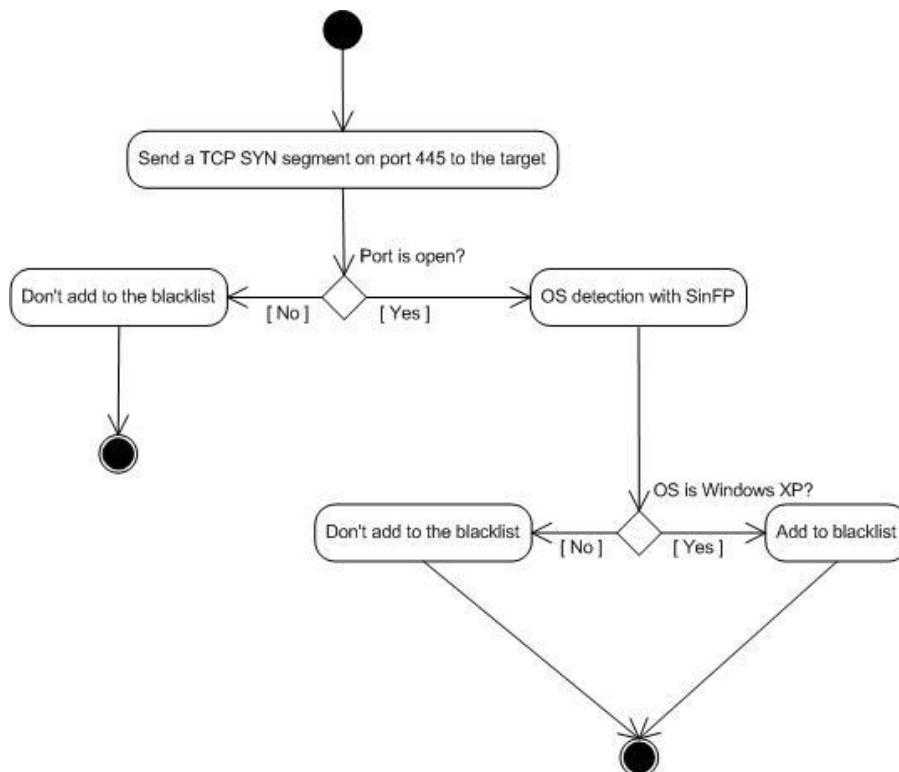


Figure 26 State diagram of blacklisting process

### 8.3.3 Configuration Checking

Because of the complexity of D.E.V.A.S and especially the update-nessusrc.pl script, which modifies the Nessus client configuration file on the fly without any further humans supervision, a configuration checking step has been implemented before launching the Nessus scan.

The *checkNessusConfigFiles.sh* checks that critical options are set correctly. If something critical is misconfigured the vulnerability scan will be cancelled. This is simply a safeguard to avoid scanning with plugins that could enumerate SMB shares and users that could effectively result in a DOS attack on the infrastructure. D.E.V.A.S will automatically compare the plugin settings in the configuration

file against a list of plugins (*blackListPlugins.txt*) that should not be set. It can be further extended to assert that a plugin needs to be activated as well. If any mismatch is identified the scanning process will not be initiated.

Some plugins use more intrusive checks to detect versions and vulnerabilities which can do harm to the system or infrastructure. These intrusive checks are turned off using the “safe\_checks = yes” statement.

Even if intrusive plugins are disabled, they can depend on other plugins which will run them. For this reason, the “auto\_enable\_dependencies = no” statement needs to be set so that other plugins do not run plugins that are in a disabled state. Safe\_checks and auto\_enable\_dependencies settings need to be set in the Nessus server configuration file (*nessusd.conf*) as well because they override the client configuration file, which can cause harm and confusion.

### 8.3.4 Generate Target List

Many of the computers we scanned were not always reachable. They could be switched off, disconnected from the network or the address that we were scanning may belong to a class C network that was not fully populated. The goal of this process step is to generate a dynamic target list that only contains computers that are reachable during the vulnerability scan. This was especially important for the standard implementation of the *nmap.nasl* Nessus script which treated all computers as alive. The scanning problem arose when computers in the target list were down and the Nmap Nessus plugin tried to scan all default ports regardless of whether the computer was up or down. It was very time consuming.

D.E.V.A.S uses the Nmap application to determine if a computer is up or down. The default way to determine if a host is up by Nmap is to send an ICMP echo request message and a TCP segment with the ACK flag set to port number 80. If the D.E.V.A.S system is in the same LAN segment as the target than an additional ARP request packet is sent.

The Nmap command looks like this for determining if the target is up or down:

```
nmap -sP -PE -PP -PM -PS80,8080,139,445,22 -n -iL targetFile.txt --excludefile  
blackListTargets.txt
```

Most of the flags are related to host discovery. “-sP” flag indicates that the scan will do host discovery only, i.e. only to determine if the host is up or down.

“-PE -PP -PM” flags tells Nmap to use different types of ICMP messages. “-PS80,8080,139,445,22” flags tells Nmap to send TCP segments with the SYN flag set to port 80, 8080, 139, 445 and 22 to determine if the host is up. The reason for sending a TCP segment with the SYN flag set is to scan through stateful firewalls that assume such a TCP segment for a TCP connection to be established. “-n” flags removes DNS resolution which is not needed and just adds extra time to resolve each IP address to DNS name.

The result is a dynamic list of computers that contains only those machines that are reachable and can be fed to the vulnerability assessment scanner. This reduces processing overhead for the whole scanning exercise and it will be completed faster.

### 8.3.5 Vulnerability Scanning

The actual vulnerability scanning is performed by the Nessus client and daemon. The Nessus scanning process conducts its scanning in different stages. The first stage is a port scan to determine which ports are open for knowing what the attack surface is. The second stage is the service detection, i.e., understand what services/applications does the target runs, which includes banner grabbing and active querying against the service to determine the application's name and version.

It is important that the scanner does not spend too much time scanning a target, especially if the targets use the default ports set by IANA for its applications. The reason is the considerable longer time needed to scan the whole port range (1-65535) for no extra benefit. It is important that the scan provides valuable and correct results though. Nessus is using its own service file where all the service names and corresponding ports exist. All of Nessus vulnerability checks against a service have an entry in this file. The Nessus service file is a super set of the default UNIX service file.

Some enhancements have also been incorporated in the *nmap.nasl* script to achieve faster port scanning results. Instead of scanning a full range ports even if the host is down, D.E.V.A.S just scans hosts that are up at a faster pace. We also exclude the blacklisted computers as a precaution.

Original piece of code from *nmap.nasl*:

```
argv[i++] = "nmap";
argv[i++] = "-n";
argv[i++] = "-P0"; # Nmap ping is not reliable
argv[i++] = "-oG";
```

Figure 27 Original *nmap.nasl* code snippet

New code:

```
argv[i++] = "nmap";
argv[i++] = "-n";
argv[i++] = "-PE -PP -PM -PS80,8080,139,22 -T4 -excludefile
/root/DEVAS/conf/blackListTargets.txt ";
argv[i++] = "-oX";
```

Figure 28 New *nmap.nasl* code snippet

The Nessus OS identification mechanism uses several methods for finding the correct operating system, either using banner grabbing, actively querying information from services or using OS fingerprinting by looking on TCP/IP stack patterns. Nmap is using the latter method for detecting the remote operating system but that information is not used internally in Nessus to set the operating system for the target.

These are the current Nessus scripts for determining the operating system:

- `os_fingerprint_http.nasl`
- `os_fingerprint_linux_distro.nasl`
- `os_fingerprint_mdns.nasl`
- `os_fingerprint_msrpc.nasl`

- `os_fingerprint_ntp.nasl`
- `os_fingerprint_rdp.nbin`
- `os_fingerprint_sinfo.nasl`
- `os_fingerprint_smb.nasl`
- `os_fingerprint_snmp.nasl`
- `os_fingerprint_ssh.nasl`
- `os_fingerprint_telnet.nasl`
- `os_fingerprint_uname.nasl`
- `os_fingerprint_xprobe.nasl`

We have changed the Nessus OS identification schema to be able to take the Nmap OS fingerprinting results into account if all other methods could not determine the target's operating system. The *nmap.nasl* script is a wrapper around the real Nmap application. It spawns an Nmap process, parses the output and saves all information into Nessus knowledge base system (KBS). In the *nmap.nasl* script, the output format was the deprecated "grepable" format (**-oG**), which does not support the OS identification tag. Instead we created a wrapper around Nmap that is called from our *nmap.nasl*. The output format is the new XML format that supports the OS identification tag. At a later stage, this XML format is converted to a modified grepable format that supports the OS identification tag that we later parse in the *nmap.nasl* script.

The *os\_fingerprint.nasl* script is then modified in pseudo syntax as follows:

```

if all the os_fingerprint scripts do not find a matching OS;
then
    OS = the operating system that Nmap found
end if

```

Figure 29 New *os\_fingerprint.nasl* pseudo code

It is important that the most recent version of Nmap is used in order to have the latest OS fingerprint database to be able to identify the correct operating system. Figure 30 is a state diagram that shows how the wrappers work between each layer.

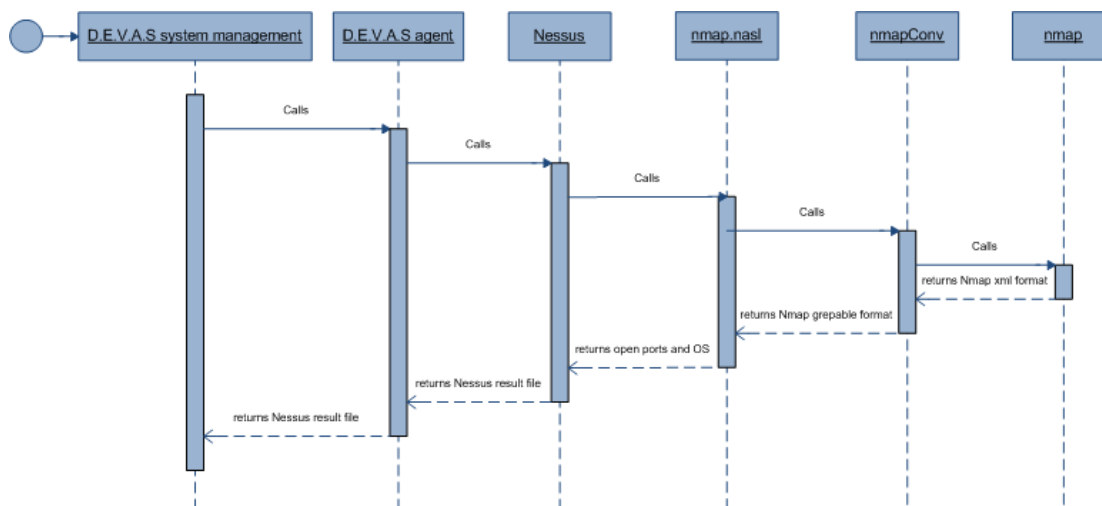


Figure 30 State diagram of the Nessus Nmap change

### 8.3.6 Logging

Each step in the scanning process is logged for backtracking and auditing purposes.

We have implemented various tools for monitoring each part of the D.E.V.A.S scanning process. To monitor D.E.V.A.S single scanning mode and an agent, the debugAgent.sh script will display a screen split into three monitoring panes containing information from:

- Scanlog.txt, which is the D.E.V.A.S log file. It records information about each stage of the process and a corresponding timestamp. It also lists the number of live hosts scanned.
- The remaining two files deliver messages from the Nessus daemon and its plugins, these are */var/log/nessus/nessusd.messages* and */var/lib/nessus/nessusd.dump*.

The D.E.V.A.S management systems log file is called distributedLog.txt which shows all of its process states, including alive targets and alive agents. The monitorAgents.sh script logs into all running agents and provides real-time monitoring of the agents Scanlog.txt file. In the end of the vulnerability scan the internal Nessus database and the Nessus daemon logs files are archived and saved.

Figure 31 and Figure 32 show examples of the two log files; one from D.E.V.A.S management system and one from an agent.

```

DEVAS scan launched, Wed Aug 20 11:08:29 2008
Check that the configuration is fine before active host list generation Wed Aug 20
11:08:29 2008
Generation of the alive host list launched, Wed Aug 20 11:08:30 2008
Generation of the alive host list finished, Wed Aug 20 11:08:32 2008
Alive host list:
10.145.26.43
10.145.26.44
10.145.26.89
10.145.26.96
10.145.26.107
10.145.26.112
10.148.201.14
10.107.11.27
10.107.21.138
10.107.21.139
Number of alive hosts: 10
nessusd was already running, Wed Aug 20 11:08:32 2008
Nessus starting to scan, Wed Aug 20 11:08:32 2008
Nessus finished the scan, Wed Aug 20 11:14:57 2008
DEVAS scan is finished, Wed Aug 20 11:15:11 2008
  
```

Figure 31 D.E.V.A.S agent log

```
Distributed EVMS scan launched, Wed Aug 09 17:46:07 2008
Generation of the alive host list launched, Wed Aug 09 17:46:07 2008
Generation of the alive host list finished, Wed Aug 09 17:46:07 2008
Alive target list:
10.145.26.82
10.107.21.138
10.107.21.139
Number of alive targets: 3
Checking how many agents that is alive, Wed Aug 09 17:46:07 2008
Checking that agent: 10.140.2.34 is up and that ssh publickey is properly set up, Wed
Aug 09 17:46:07 2008
Agent: 10.140.2.34 is properly set up, Wed Aug 09 17:46:07 2008
Checking that agent: 10.140.2.35 is up and that ssh publickey is properly set up, Wed
Aug 09 17:46:07 2008
Agent: 10.140.2.35 is properly set up, Wed Aug 09 17:46:07 2008
Alive agent list:
10.140.2.34
10.140.2.35
Number of alive agents: 2
Sending the target list to 10.140.2.34, Wed Aug 09 17:46:07 2008
Sending the target list to 10.140.2.35, Wed Aug 09 17:46:08 2008
Start the agents:
Agent: 10.140.2.34 starts to scan, Wed Aug 09 17:46:08 2008
Agent: 10.140.2.35 starts to scan, Wed Aug 09 17:46:08 2008
Agent: 10.140.2.35 scan done, Wed Aug 09 17:50:41 2008
Agent: 10.140.2.34 scan done, Wed Aug 09 17:50:42 2008
Merge the scan reports from the agents, Wed Aug 09 17:50:42 2008
Merge done, Wed Aug 09 17:50:42 2008
Distributed EVMS scan is finished, Wed Aug 09 17:50:57 2008
```

Figure 32 D.E.V.A.S management system log

### 8.3.7 Health Report

D.E.V.A.S makes a best effort to find vulnerabilities without side effects that could do harm to the infrastructure.

When a vulnerability scan is performed, a summary health report is sent to the administrators. The purpose of the health report is to provide a quick overview of the scan progress and to determine if the scan finished successfully without side effects.

The report includes:

- The log file from the last D.E.V.A.S scan
- Nessus clients that are running after the scan
- Number of Nessus plugins
- Disk space usage
- Newly discovered vulnerabilities
- Information whether any Windows accounts have been enumerated.

Even though Nessus is configured with safe checks, plugins has been removed from the system and an extra check is done before the actual scan, we cannot guaranty that D.E.V.A.S will not do any harm to the infrastructure. In addition to set the `safe_checks` option to prevent damages to the systems being scanned, it is important to have a post check to notify the stakeholders if something goes wrong.



## D.E.V.A.S single scanning mode

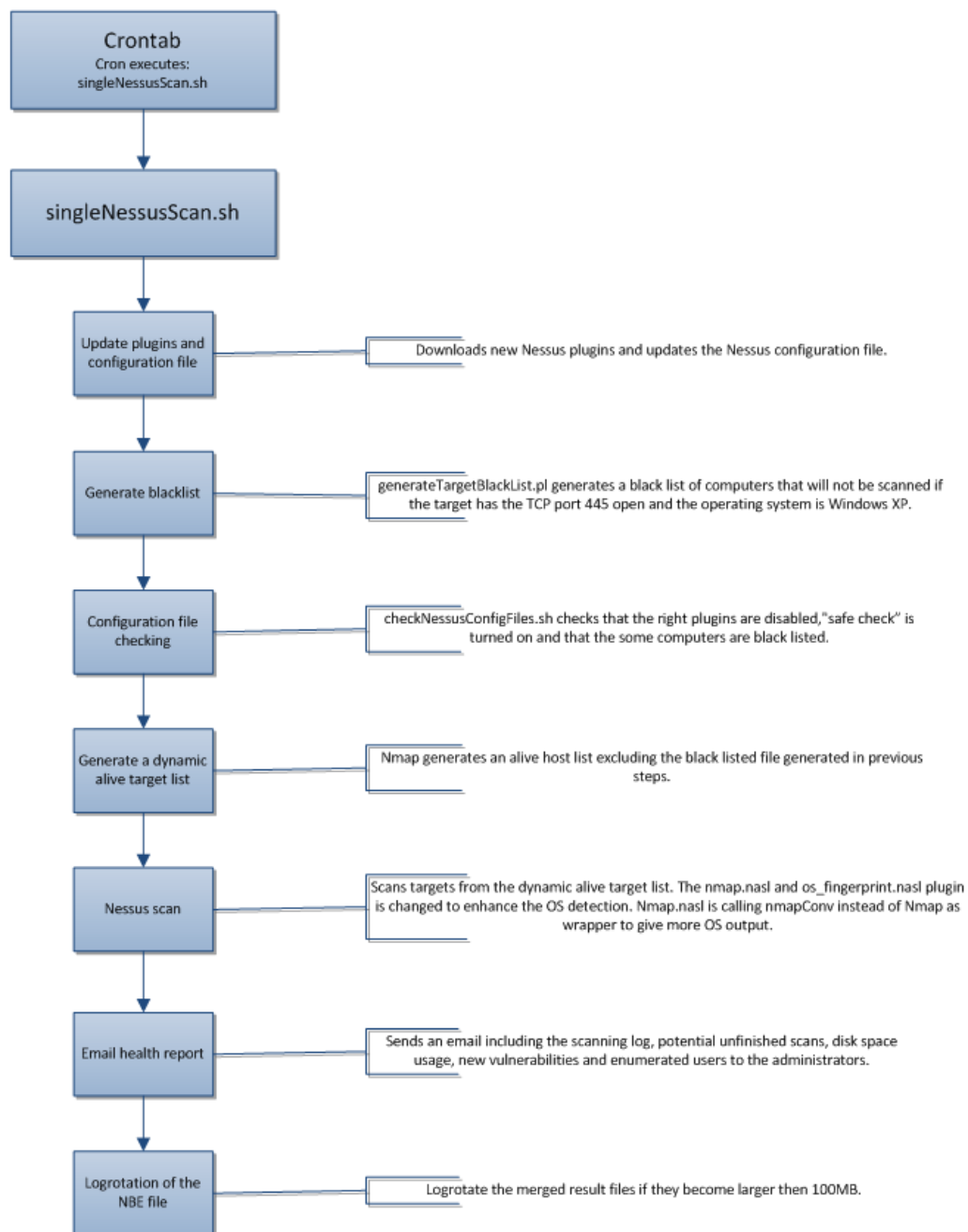


Figure 33 D.E.V.A.S single mode state diagram

## 8.4 Distributed Scanning

The single mode scan would be sufficient if the number of targets is low and all the targets are in the same LAN segment. But to fit most network architectures and the need to scan many targets during a limited time, an alternative approach to share the targets between many scanning servers is needed.

Vulnerability assessment scanners are great tools for enhancing the overall security in an organization, but there exists a risk of causing damage because of the intrusive behavior of the application. Because of these risks, the vulnerability assessment scanner can interfere with normal business traffic and should be run after business hours, which imposes a deadline for the scanning. Being able to share the scanning load between many computers as a way to reduce the total scanning time becomes a necessity. Distributed scanning also gives the possibility of distributing targets to computers where other computers cannot operate, e.g. many computers are protected behind firewalls, which makes it difficult for a computer outside to be able to detect all services or even hosts. By deploying a computer behind the firewall and change the rule set to open a port for SSH traffic to pass through, we can scan targets behind the firewall and send the results to the server that initiated the scan.

D.E.V.A.S works by having a D.E.V.A.S management system, which is the server that initiates the scanning procedure. It distributes its targets to other computers called “D.E.V.A.S agents”, which perform the scan and send the result back to the D.E.V.A.S management system which merges the results together.

The distributed scanning mode has two major purposes:

1. Sharing targets between D.E.V.A.S agents
2. Scan targets protected by packet filter and stateful firewalls

Each D.E.V.A.S agent works in a similar way as the single scanning mode. The only major difference is that the Nessus result file will be transmitted back to the D.E.V.A.S management system where the result files will be merged.

The D.E.V.A.S management system main tasks are:

- Distribute new plugins to agents
- Balance targets between agents
- Start agents
- Collect and merge the individual agent scan results

All communication between the agents and the server is performed using SSH. To take advantage of full automation between agents and the D.E.V.A.S management system, pre-shared keys are setup in SSH to skip the manual authentication step. Figure 33 shows how the D.E.V.A.S system can be deployed in an organization.

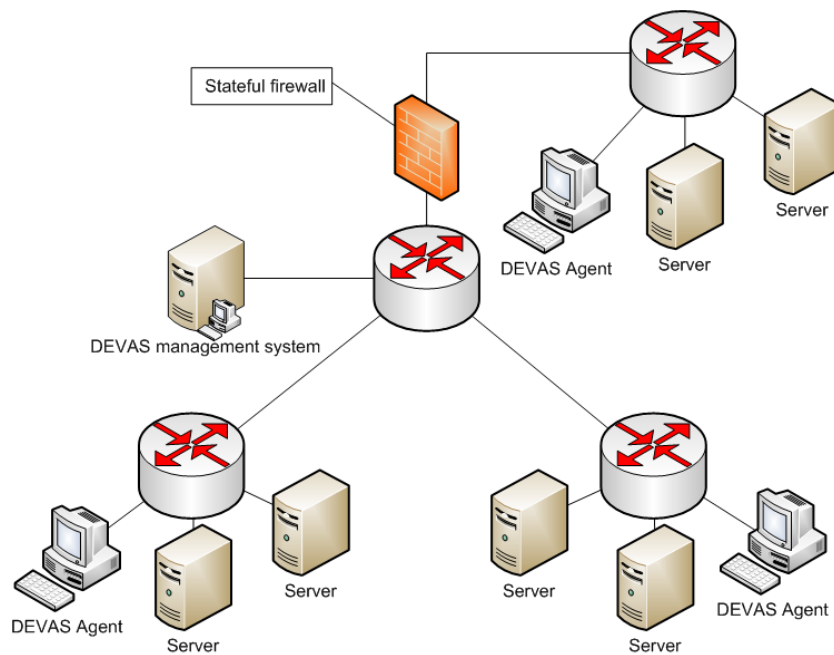


Figure 34 D.E.V.A.S network setup behind firewall

Below will each step of D.E.V.A.S distributed mode's workflow be discussed in detail. In the end of this chapter, a UML state diagram will present a clear view of how everything is tied together.

### 8.4.1 Update and Distribute New Plugins to Agents

As D.E.V.A.S agents cannot always reach the Internet directly to fetch new Nessus plugins, it is necessary to have a centralized plugin-updating functionality.

Getting new plugins for the D.E.V.A.S agents is done by the D.E.V.A.S management system and are later distributed to its agents. There are currently (2008-11-10) over 20.000 Nessus plugins with a total size is over 300 Mbyte. Therefore, instead of replacing all plugins for every agent when a new plugin is available, the *rsync* application will only transfer new and modified plugins to all agents – which is much more efficient. When new plugins are available for the D.E.V.A.S agent, the agent updates its own configuration file to encounter the new plugins.

### 8.4.2 Generation Of Alive Target List

The generation of “alive” targets works differently in the distributed mode compared to the single scanning mode. When targets are behind firewalls, the D.E.V.A.S management system needs to dedicate agents to perform vulnerability assessment just on those targets. Therefore, targets that will be sent to dedicated agents will not be pre-scanned from the D.E.V.A.S management system as the targets may not been seen from the it. All other agents that are not dedicated to targets will share the remaining targets.

### 8.4.3 Generation Of Alive Agent List

As targets can be down during the scan, as well can agents be. We ensure in this step that the whole scanning exercise will still function as intended, even if an agent is not alive during the scan. Therefore, the system creates a list of alive agents in order to distribute the targets accordingly.

#### 8.4.4 Sharing Targets Between Agents

One very important aspect about enterprise scanning is not to overload the network with traffic, especially during business hours. It is important that the scans are finished before the deadline, which is normally when business hours start. It is less probable that D.E.V.A.S needs to halt the scan if the targets are properly distributed amongst the agent, since there will be an overall dependency on the agent with the highest number of targets, whereas others with less load will have completed their task. The solution is to appropriately balance the load between the agents in an attempt to have all agents complete their individual tasks at approximately the same time. If the scanning process is not finished before the deadline, all agents will instantly be stopped by the D.E.V.A.S management system. The deadline is set by a time/date in D.E.V.A.S management system.

There are two different types of targets:

- **dedicated targets** are those that have to be assigned to a specific agent (e.g. can only be accessed by an agent that is placed on a particular subnet or placed behind a firewall)
- **pooled targets** are those targets that can be accessed by multiple agents and can therefore be distributed to a pool of agents without restrictions.

All targets and agents in D.E.V.A.S are listed into text files, where each line is either a target or an agent. All configuration files can be found in the conf/ directory.

Dedicated targets (e.g. targets behind firewalls) need to be linked to specific agents and therefore require a particular syntax: targetIP[:agentIP] in the conf/targets.txt file. Below is an example target file.

```
10.0.0.1:10.0.1.1
10.0.0.2:10.0.1.1
10.0.0.3:10.0.1.1
10.0.0.4
10.0.0.5
```

Figure 35 D.E.V.A.S example target list

In figure 34, the first three entries indicate that agent 10.0.1.1 will process dedicated targets, i.e. 10.0.0.1 to 10.0.0.3. The remaining two entries are a pool of targets that will be shared between the remaining agents. It is also possible to use the Nmap syntax as target entries, for example a dedicated target could look like this:

```
10.0.0.0/24:10.0.1.1.
```

The target entry in above will scan the 10.0.0.0 Class C network from the agent with an IP address of 10.0.1.1. All communication between the D.E.V.A.S management system and the agents are sent through secure channels. All signals are sent through SSH and files are sent through SCP.

### 8.4.5 Start, Collect and Merge Agent Result Files

After the agents receive their targets, the D.E.V.A.S management system sends a start signal to the agents to initiate the scanning process. There are two communication modes between the server and the agent: “half-duplex mode” and “full-duplex mode”. When the communication is running in full-duplex mode, the agent initiates an SSH communication back to the D.E.V.A.S management system when it returns the result. If the D.E.V.A.S agent is deployed in the DMZ, it is likely that the agent cannot initiate an SSH connection back to Intranet where the D.E.V.A.S management system is located. It is a good practice to have firewall polices that limits the amount of connection between the DMZ and Intranet, especially communication channels initiated from the DMZ. Figure 35 shows a simplified picture of how it could be setup.

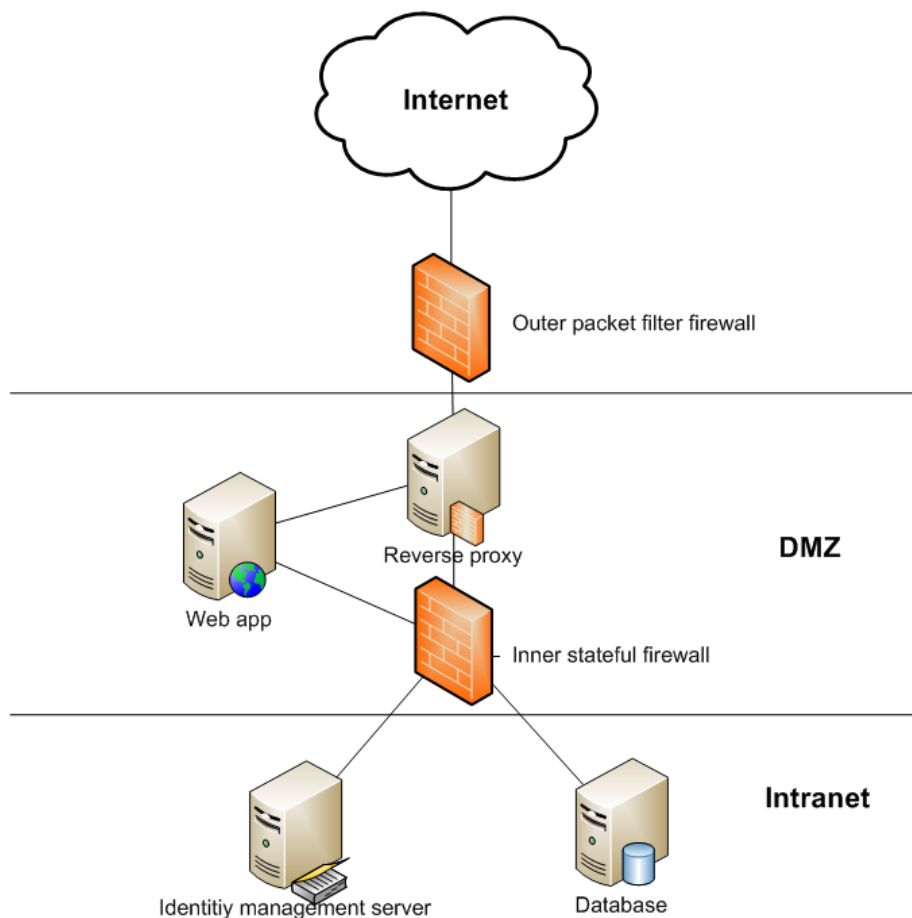


Figure 36 Simplified network perimeter architecture

It is standard practice to have an inner stateful firewall that blocks communication channels initiated from inside the DMZ, which makes the full duplex mode unusable. To overcome this problem, an alternative solution has been implemented; half-duplex mode. In half-duplex mode, all SSH communication are initiated from the D.E.V.A.S management system. The initial signal from the server is kept alive during the whole scan and eventually pulls the result from the agent when it is finished. Figure 36 is a state diagram of the final D.E.V.A.S implementation.

## DEVAS distributed scanning process

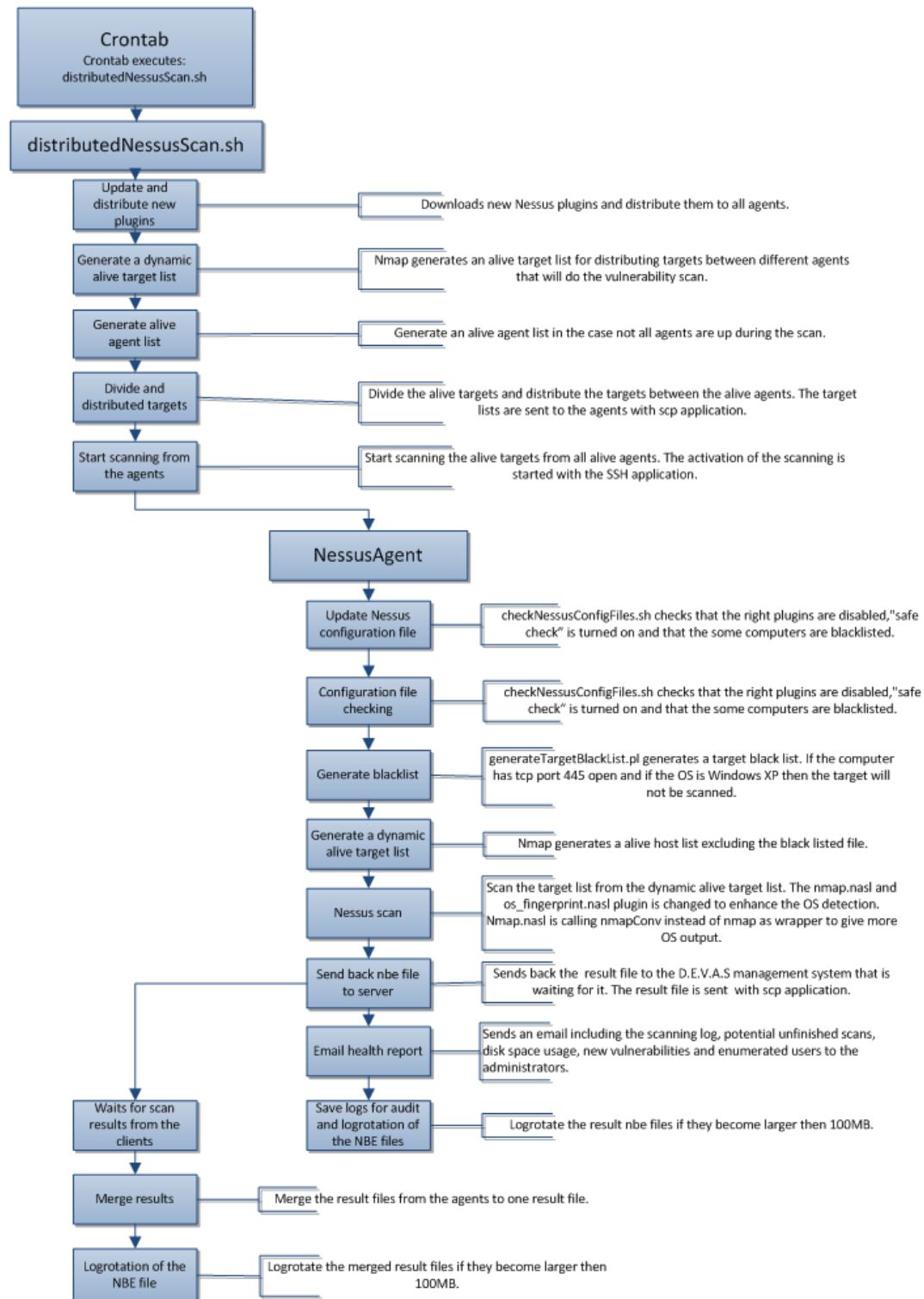


Figure 37 D.E.V.A.S distributed mode state diagram

## 8.5 Data Analysis and Correlation

In order to make an analysis and correlation on results from Nessus, we need to be able to parse and understand the file format. Fortunately, the Nessus result file is easy to understand and simple to parse with standard regular expressions. Our data analysis and correlation tools are separated into two scripts: *nbeQuery.sh* for making analysis from a single result file and *nbeDiff.sh* for making correlations between two result files.

The Nessus package includes a graphical user interface normally used to review the results. The graphical user interface is a thick client application which makes it difficult to present the results to different stake holders effectively. As it does not include any features for recurrent scanning and correlation, it makes it difficult to keep track of the vulnerabilities during many scans; that is why we implemented these two tools.

### 8.5.1 Analysis from a Single Result File

NbeQuery.sh extracts and analyzes data from the Nessus result file in a presentable format. The idea behind the tool is to present the current state of all vulnerabilities that are found. The tool can also search for patterns within the current scan.

The following is a description of each option that nbeQuery gives:

<b>-x</b>	<ul style="list-style-type: none"><li>▪ <code>nbeQuery.sh -x [-f output.txt] file.nbe</code></li></ul> <p>This generates a summary with general vulnerability statistics and statistics sorted by operating system. It includes number of hosts scanned, number of identified vulnerabilities (holes) and average vulnerabilities per host.</p> <p>A ranking of the Top Ten most vulnerable hosts and Top Ten existing vulnerabilities is also provided.</p>
<b>-t</b>	<ul style="list-style-type: none"><li>▪ <code>nbeQuery.sh -t hole warning note all [-b ipNumber] [-f output.txt] file.nbe</code></li></ul> <p>Lists all identified vulnerabilities, warnings and notes from all targets or from one particular category (e.g. holes).</p>
<b>-s</b>	<ul style="list-style-type: none"><li>▪ <code>nbeQuery.sh -s -b ipNumber [-f output.txt] file.nbe</code></li></ul> <p>List all services running on a particular host.</p>
<b>-w</b>	<ul style="list-style-type: none"><li>▪ <code>nbeQuery.sh -w "vulnerability reference" [-f output.txt] file.nbe</code></li></ul> <p>List all targets that have a specific vulnerability (e.g. CVE-1999-0103).</p>
<b>-p</b>	<ul style="list-style-type: none"><li>▪ <code>nbeQuery.sh -p nessusPluginID [-f output.txt] file.nbe</code></li></ul> <p>List all target that ran a specific Nessus plug-in.</p>

The reason for presenting these statistics is to give a snapshot of the current vulnerability state for decision makers: what system or vulnerability should one prioritize? By sorting some of the statistics

by operating system/department, we show also what systems are best maintained and patched. Figure 37 presents an example.

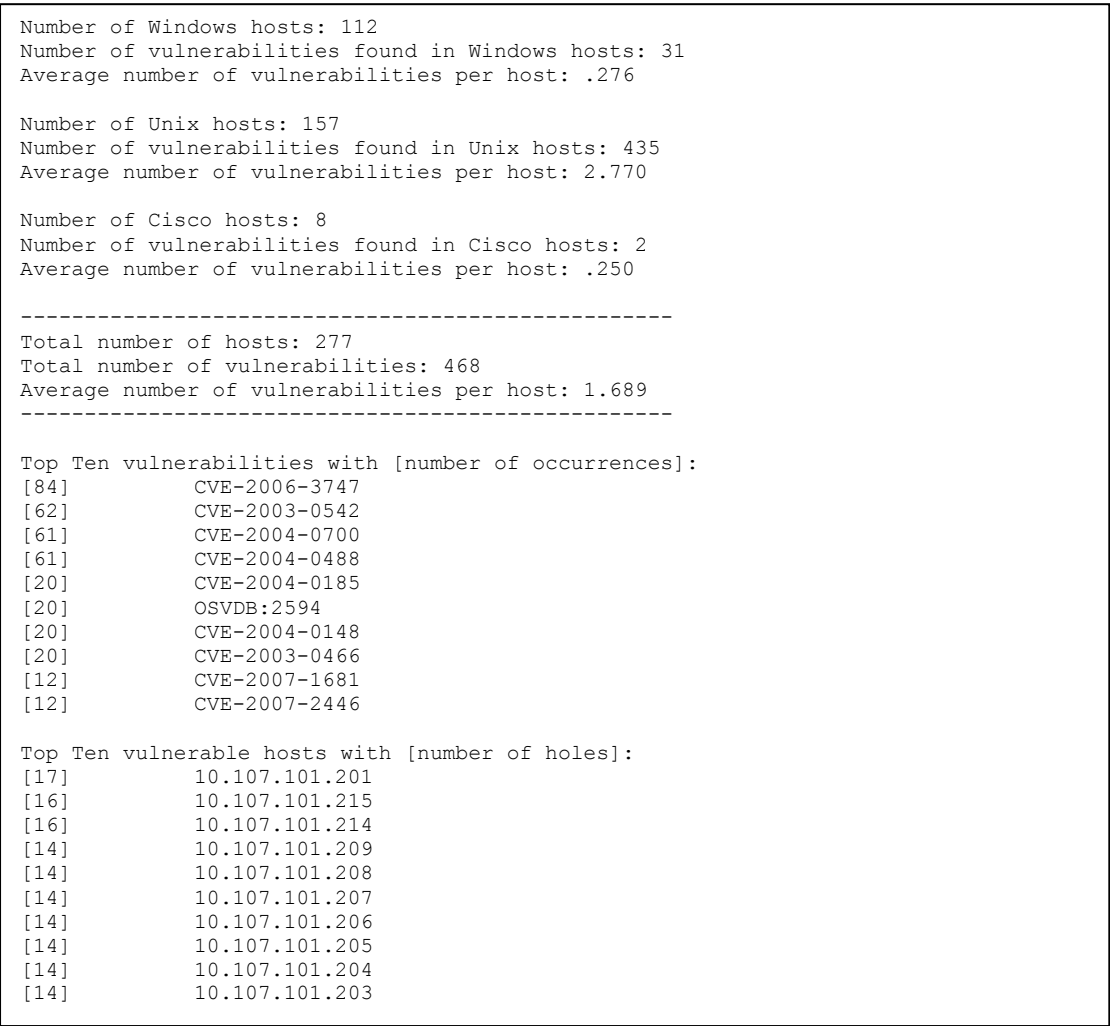


Figure 38 NbeQuery.sh result statistics

### 8.6 Correlation Between Two Result Files

NbeDiff.sh extracts and makes a correlation between two Nessus result files. The purpose is to see the differences between two scans.

The following is a description of each option that nbeDiff.sh gives:

<b>-h</b>	<ul style="list-style-type: none"><li>▪ <code>nbeDiff.sh -h [-v] [-f output.txt] oldFile.nbe newFile.nbe</code></li></ul> <p>Lists all new hosts and hosts that are down from the latest scan.</p>
<b>-s</b>	<code>nbeDiff.sh -s [-v] [-f output.txt] oldFile.nbe newFile.nbe</code> <p>Lists all new services and services that are down from the last scan.</p>
<b>-w</b>	<ul style="list-style-type: none"><li>▪ <code>nbeDiff.sh -w [-v] [-f output.txt] oldFile.nbe newFile.nbe</code></li></ul>



	Lists all new vulnerabilities from the latest scan.
<b>-p</b>	<ul style="list-style-type: none"> <li>▪ <code>nbeDiff.sh -p [-v] [-f output.txt] oldFile.nbe newFile.nbe</code></li> </ul> <p>Lists all “fixed” vulnerabilities from the latest scan.</p>
<b>-u</b>	<ul style="list-style-type: none"> <li>▪ <code>nbeDiff.sh -u [-v] [-f output.txt] oldFile.nbe newFile.nbe</code></li> </ul> <p>Lists all “unfixed” vulnerabilities from the latest scan.</p>

All these scans will just provide a snapshot of the current vulnerability state. Hosts and services could be turned off on purpose, accident or even turned off to hide itself from the actual scan. Therefore is it important to show these changes as it changes our attack surface and gives a false picture. NbeDiff.sh shows this information by comparing this two result files. Figure 38 shows an example.

```

===== Hosts up and down =====
New hosts:
10.107.11.27
10.107.21.138
10.107.21.139
10.145.26.109
10.145.26.112
10.145.26.40
10.145.26.82

Number of hosts that are new: 7

Hosts down:
10.145.26.116
10.145.26.43

Number of hosts that are down: 2

===== Services up and down =====
==== 10.145.26.107 ====
New services:
    ddt (1052/tcp)
    optima-vnet (1051/tcp)

Stopped services:
    ms-sql-s (1433/tcp)
    neod1 (1047/tcp)
    neod2 (1048/tcp)

```

Figure 39 NbeDiff.sh differential result report on hosts and services

The result files will contain all current vulnerabilities found. The patching process of the organization is normally not as fast as it should be. This is especially true for vulnerabilities that are not top prioritized. It can take weeks or even months before the vulnerability has been fixed. We are most likely to receive an enormous amount of recurrent vulnerabilities after each scan, especially when there are hundreds to thousands of targets being scanned. To filter out new vulnerabilities by hand using the Nessus graphical client is not feasible. A new vulnerability is a vulnerability from:

- a new host
- a new service

- a new vulnerability found from an old service but found with a new Nessus plugin

Therefore searching through all these possible scenarios by hand and comparing with the old result file is not feasible. It is important to find out only the new vulnerabilities because there could be a new vulnerability that is severe for the organization and it needs to be top prioritized. Figure 39 shows an example.

```
===== New vulnerabilities =====
Counting number of vulnerabilities: 2
Windows hosts:

10.145.26.40
    general/tcp,Synopsis :The remote host is running a version of Microsoft
    Windows which is not supported by Microsoft any more. A lack of support implies that
    no new security patches will be released for this operating system.Solution :
    Upgrade to Windows XP or newer See also :
    http://support.microsoft.com/gp/lifean18Risk factor : High

Unix hosts:
10.107.11.27
    ldap (389/tcp),Synopsis :The remote LDAP server is prone to multiple
    vulnerabilities. Description :The remote host is running the Sun Java System
    Directory Server; an LDAP server from Sun Microsystems. The remote version of this
    service is Solution :Upgrade to Sun Java System Directory Server 5.2 Patch 5 or
    6.1Risk factor :High / CVSS Base Score : 7.8(CVSS2#AV:N/AC:L/Au:N/C:N/I:N/A:C)CVE :
    CVE-2006-4175; CVE-2007-2466; CVE-2007-3224; CVE-2007-3225BID : 23117; 23743; 24467;
    24468Other references : OSVDB:33524; OSVDB:35743; OSVDB:37246; OSVDB:37247

Number of new vulnerabilities: 2
```

Figure 40 NbeDiff.sh result report on new vulnerabilities

To be able to verify that a vulnerability has been fixed is dependent on how one defines “fixed”. Our definition of a fixed vulnerability is a vulnerability that does not show up anymore in the new Nessus result file with two exceptions:

- The host is still reported as running in the new Nessus result file
- The service runs on the same service port in both Nessus result files

By shutting down the service/host or remapping the port number for the service is not defined as a “fixed” vulnerability. The only way of getting a “fixed” vulnerability is either to patch the service or make a reconfiguration. The reason for such strict definition is that a service or a host can be down during scans and should not be presented as in a “fixed” state. Figure 40 shows an example.

```

===== Checking for patched services =====

Windows hosts:

10.145.26.107
    Patched [nessus_script_id:14259 nessus_script_id:14259]: general/tcp)

10.145.26.109
    Patched [nessus_script_id:14259 nessus_script_id:14259]: general/tcp)

10.145.26.112
    Patched [nessus_script_id:14259 nessus_script_id:14259]: general/tcp)

10.145.26.116
    Patched [nessus_script_id:14259 nessus_script_id:14259]: general/tcp)

10.145.26.40
    Patched [nessus_script_id:14259 nessus_script_id:14259]: general/tcp)
    Patched [nessus_script_id:19699]: general/tcp)
    Patched [nessus_script_id:19699]: general/tcp)
    Patched [nessus_script_id:19699]: general/tcp)

-----

Number of common Windows vulnerabilities: 2
Windows services that are patched: 8
Windows Patch Ratio: 4.000

Number of common Unix vulnerabilities: 0
Unix services that are patched: 0
Unix Patch Ratio: 0

Number of common Cisco vulnerabilities: 0
Cisco services that are patched: 0
Cisco Patch Ratio: 0

Total patched services: 8
Total number of common vulnerabilities: 2
Total Patch Ratio: 4.000

```

**Figure 41 NbeDiff.sh result report on patched services**

We have also provided the option of showing the “unfixed” services between security assessment scans. This option gives a picture how well the patching process works in organization with statistics to differentiate between operating systems. Figure 41 shows an example.

```

===== Checking for unpatched services =====

Windows hosts:

10.145.26.40
    Unpatched [CVE-2006-1314, CVE-2006-1315]: netbios-ssn (139/tcp)
    Unpatched [nessus_script_id:21626]: general/tcp)

Unix hosts:

10.107.11.27
    Unpatched [CVE-2006-4175, CVE-2007-2466, CVE-2007-3224, CVE-2007-3225]: ldap (389/tcp)

-----

Number of Windows services that are unpatched: 2
Number of Unix services that are unpatched: 1
Number of Cisco services that are unpatched: 0

Total unpatched services: 3

```

**Figure 42 NbeDiff.sh result report on unpatched services**

Sometimes it is very difficult to count the number of vulnerabilities that is found. Hosts can have multiple NICs or using virtual IPs that uses the same NIC. In order to find unique hosts and not duplicates, one has to find a unique signature for each host. Our first guess was to look at what services the hosts were running. This has been proven to be very difficult as many hosts have the same ports opened, especially in a big organization where many hosts are installed with the same default image. Our tools do not take this into account.

## 9. Future improvements

---

### 9.1 Upgrade to Nessus 3.x

The Nessus version running in D.E.V.A.S is an old open source version (2.x). The newer versions of Nessus (3.X) provide greater scanning speed and functionality. By using the newest version of Nessus, the overall scanning time and memory consumption will be reduced.

### 9.2 Rewrite D.E.V.A.S

D.E.V.A.S is a proof of concept application; it does not support all the paradigms in software engineering, like maintainability, reusability clearness. The code is written in the Bash shell scripting language, SED/AWK, Perl and NASL with many programming tricks, also known as hacks. Rewriting the main code in a more reusable and easily understandable language like Python or Ruby would make it easier to extend and reuse code.

The application should be implemented in an interpreter language for fast and easy changes to adapt easily into new working environments. Therefore, implementing the application in the language C/C++ is not a good choice; the main code would run faster but it will take much more time to write the code and cause more difficulties when making changes. Speed improvements would be minimal because Nessus will run its legacy code most of the time, which we will not change.

The object orientated programming languages Python or Ruby are more suitable because there exist many libraries for these languages and their syntax is clean and more easily understandable. These features make development more reusable and faster.

### 9.3 Save Scanning Results into a Database

The current implementation of D.E.V.A.S saves all the result files into a directory and performs analysis on individual files. A more scalable and long term solution would be to save all the result files into a database. Making analysis and correlation could be done with SQL statements. A database would be a better solution for a Web frontend that would query directly from a database instead of calling intermediate scripts or handling the analysis within the Web application itself.

The idea would be to make a column for every attribute from the Nessus result file and use the same boolean logic to present and make correlations, like the analyze scripts in current implementation of D.E.V.A.S. Additional columns for date and vulnerability tracking should also be added so that the database handles all relevant information.

## 9.4 Web Frontend

The analysis scripts present the results by writing it out to a terminal (by default). There are several problems with this design. The main security concern is that all users need to have a shell account for accessing the Nessus result file. Having normal user shell accounts on the D.E.V.A.S host increases the risk for information disclosure of all the vulnerabilities found, e.g., a normal user with a shell account can use a local exploit to escalate its privileges. A compromised D.E.V.A.S host could also start other hostile scans against hosts which are critical for the organization.

To reduce manual work and distribute all the vulnerabilities to the concerning stake holders, there is a need to delegate work in an automatic way. Therefore to have a Web frontend would satisfy our requirements. The benefits with a Web frontend are:

- There is no need for a thick client. Users can access the data with a normal web browser.
- The Web frontend can easily be integrated in to the existent identity management infrastructure, like LDAP or Active Directory.
- Fine grained ACLs can be defined per role/group/user, which only shows vulnerabilities relevant to a specific user. Some vulnerabilities can be devastating for the organization if they leak out internally and/or become public. The military “need to know” principle could be applied.
- User logins are saved in the LDAP/Active Directory server and further logging can be implemented in the Web application for auditing purposes.
- The integration with a database is easy.

The Web frontend should have these requirements:

- Adding targets directly in the Web frontend instead of adding it manually in one of the text files.
- Add vulnerability tracking information like: department, contact person, initial and last contact date with the contact person and other relevant information in the Web frontend.
- Set a vulnerability as a false-positive.
- Show new vulnerabilities.
- Show recurrent vulnerabilities.
- Be able to show all services per host.
- Be able to show all vulnerabilities per host.
- Be able to sort all hosts dependent on what type of operating system the host is running. The reason for this requirement is that normally in a mid- to large enterprise differ their departments depending of what operating system they are managing.
- Be able to see when a host started to be scanned and when it was last scanned.
- Be able to see when a vulnerability was introduced and patched.
- Be able to make a summary on:
  - Most common vulnerabilities.
  - Top vulnerable hosts.
  - Number of vulnerabilities per hosts in average sorted by the operating system.
  - Trend charts of:

- Number of vulnerabilities and warnings totally and per operating system.
- Number of patched services totally and per operating system.

All this information that should be presented in the web frontend would either be in the database or be computed based on the data that lies in the database. As D.E.V.A.S is a proof of concept, we made instead an Excel spread sheet and relied on the manual work of taking the output from the analyzing scripts and populate the spread sheet. Another feature that could be implemented would be to automatically notifying the departments when a new vulnerability has been found. This could easily be done using the analysis script in the end of the scan by sending out an email to the department that manages the host.

## 9.5 Integration with Intrusion Detection Systems

Intrusion detection systems are becoming more common these days. In contrast to the proactive role vulnerability assessment systems have, intrusion detection systems work in a passive manner, i.e., they wait until an attack is conducted. False positives are a major problem with current intrusion detection systems, where the intrusion detection system alerts many intrusion attempts that never occurred. Snort is an open source intrusion detection system that works by having signatures/rules for possible attack attempts. Deploying Snort in the organizations backbone and turning on all rules requires massive processing power, either through a cluster or a super computer. Having this type of intrusion detection system deployment will generate a lot of false positives that is infeasible to review by hand.

An alternative solution, where the intrusion detecting system only cares about the vulnerabilities per system that exists in the organization, will reduce the number of false positives greatly and give quality alerts. Almost every public vulnerability is given a unique ID by either Common Vulnerability and Exposures (CVE), Open Source Vulnerability Database (OSVDB), bugtraq or the Microsoft Security Bulletin. Each vulnerability check done by Nessus corresponds to one of the unique IDs above. Rules in Snort have also this type of tagging for each vulnerability it tries to find. It is possible to make a mapping between the vulnerabilities found by Nessus with the rules to turn on in Snort. Snort will then be adaptive to the vulnerabilities found by Nessus. We have found solutions that configure Snort based on vulnerabilities found by Nessus as stated above. The more distributed the correlation system are (Nessus and Snort), the faster will the total vulnerability assessment scan take and the better intrusion detection alerts will the organization receive.

To conclude, below we suggest how this could be implemented:

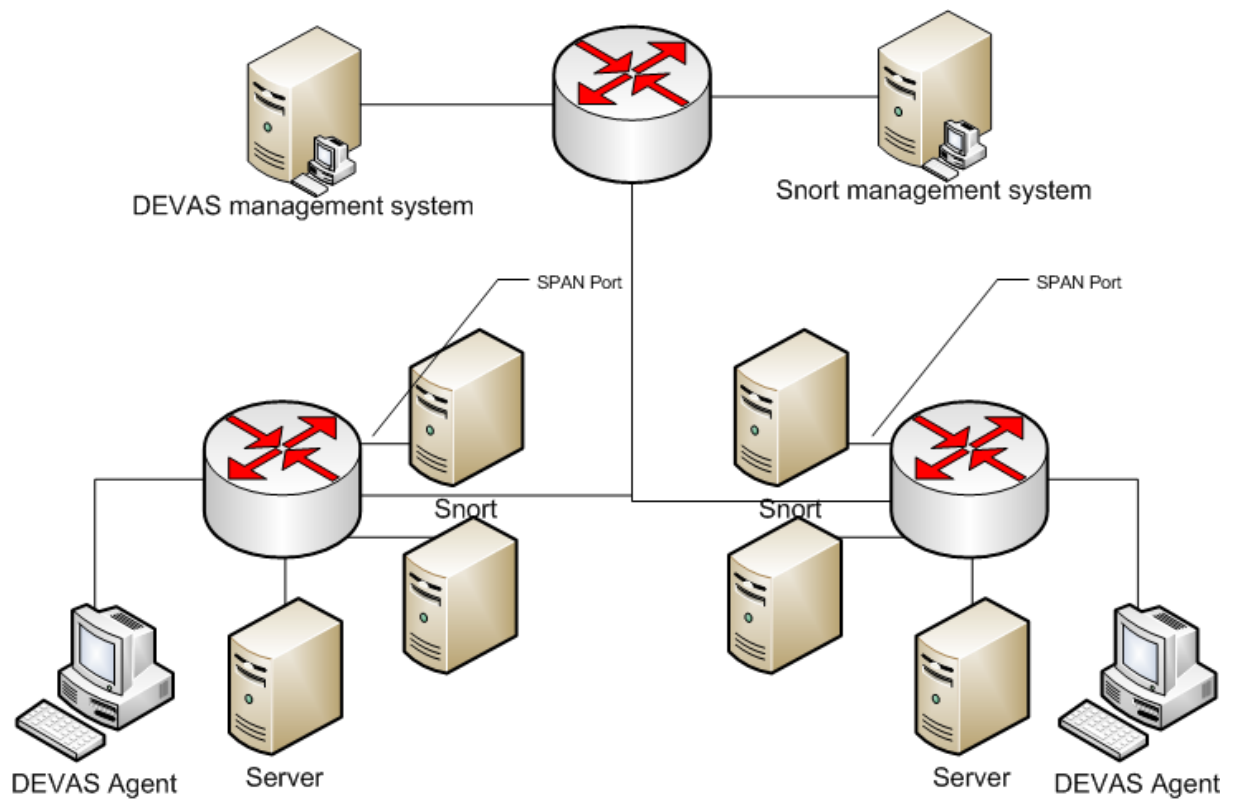


Figure 43 Correlation between Nessus and Snort IDS



# 10. Conclusions

---

Even if D.E.V.A.S is a proof of concept application with potential enhancements, it is currently used in production in a major European institution. Overall, the project has been a success. All the requirements that we setup in the beginning have been implemented and even more features have been added during development.

The application has been refined and improved many times as we have learned more about the topic and the environment which the security scanner was operating in. The initial goal of the project - to provide a better view about the organizational security condition and improve it - is an ongoing process. Our project was just a kick-off in this process.

We can conclude that enterprise scanning is not a precise science as the whole infrastructural system is very complex and always changing. It is important to know in advance how the network infrastructure is built to be able to implement an efficient and accurate scanning procedure. Being able to have a test environment that clones the real production environment as much as possible is the only way of being closer to a safe scanning procedure.

Even with a good knowledge of the organizational infrastructure it is very difficult to foresee any problem that the security scanner can cause. If the security scanner is targeting a host, it might happen that the targeted service relays all attacks to a third party host. The probability that a relay attack will happen is even higher nowadays, when organizations strive towards a more decoupled architecture especially now when service oriented architectures are popular.

As the infrastructure in various organizations differ, it is necessary to have a decoupled security scanner architecture to be able to integrate the security scanner into the business. Very often these decoupled objects are linked together with a fast prototyped language like Bash or Perl. Different organizations have different needs and different security practices. If Nessus did not implement the client-server architecture and the different result output formats for easy manipulation, it would be very difficult to implement an efficient distributed security scanner.

The current vulnerability assessment tools on the market including Nessus produce many false positives and the concept of having a fully automated tool that provides accurate results without any further human intervention is not possible today. There is still a need for a skilled analyst to review potential vulnerabilities. Due to the number of false positives and to the lack of trend and correlation analysis features in current vulnerability assessment tools, it becomes more difficult to reach the initial goal.

In the end, vulnerability assessment is a highly effective proactive method to get a glance of how the current vulnerability state is in the organization. Conducting vulnerability assessment scans in larger organizations, from a central management point, introduces many problems which a distributed vulnerability assessment solution can solve. It is an invaluable tool for organizations to use and be able to withstand today's hostile digital environment.

# 11. Appendix

---

## Installation guides

### Fast installation for single host scan

#### Install packages

1. `apt-get install -y sun-java5-jdk sun-java5-jre`
2. `apt-get install -y mysql-server`
3. `apt-get install -y nessus nessusd`
4. `apt-get install -y tomcat5.5`
5. `apt-get install -y subversion libc6-dev flex bison g++`
6. `apt-get install -y libnmap-parser-perl nmap xprobe2`
7. `wget http://heanet.dl.sourceforge.net/sourceforge/sinfp/SinFP-2.06-1.tar.gz`
8. `tar xfvz SinFP-2.06-1.tar.gz`
9. `cd SinFP-2.06-1`
10. `make`
11. `make install`
12. `wget http://kent.dl.sourceforge.net/sourceforge/heirloom/mailx-12.4.tar.bz2`
13. `tar xfvj mailx-12.4.tar.bz2`
14. `cd mailx-12.4`
15. `make`
16. `make install`

#### Checkout from the SVN repository

1. Login in as root to DEVAS
2. `cd /root`
3. `svn co svn://XXX.XXX.XXX.XXX/DEVAS`

#### Nessus

1. `cp /root/DEVAS/BACKUP/nessusServerConf/nessus-fetch.rc /etc/nessus/nessus-fetch.rc`
2. `cp /root/DEVAS/BACKUP/nessusServerConf/nessusd.conf /etc/nessus/nessusd.conf`
3. `nessus-adduser # user: koersel`
4. `/root/DEVAS/scripts/lib/updateSingleScanPlugins.sh`
5. `cp /root/DEVAS/BACKUP/nmapConv /usr/local/bin/`
6. `/etc/init.d/nessusd restart`

Interesting fields that matters in the `nessus-fetch.rc` file:

- `proxy=XXX.XXX.XXX.XXX`
- `proxy_port=8080`
- `proxy_username=*****`
- `proxy_password=*****`

Interesting fields that matters in the `nessusd.conf` file

- `nasl_no_signature_check = yes`

We have modified the `nmap.nasl`, `os_fingerprint.nasl` file so the signature that is on that file is not validated anymore.

### **Crontab**

Set up cron jobs for launching the system

1. **`cp /root/DEVAS/BACKUP/crontab /var/spool/cron/crontabs/root`**

### **Access to files**

Only root should be able to read,execute and write the files.

1. **`cd /root`**
2. **`chmod -R go-rwx DEVAS`**

## Server installation for distributed scanning

### Install packages

17. `apt-get install -y sun-java5-jdk sun-java5-jre`
18. `apt-get install -y mysql-server`
19. `apt-get install -y nessus nessusd`
20. `apt-get install -y tomcat5.5`
21. `apt-get install -y subversion libc6-dev flex bison g++`
22. `apt-get install -y libnmap-parser-perl nmap xprobe2 multital`
23. `wget http://heanet.dl.sourceforge.net/sourceforge/sinfp/SinFP-2.06-1.tar.gz`
24. `tar xfvz SinFP-2.06-1.tar.gz`
25. `cd SinFP-2.06-1`
26. `make`
27. `make install`
28. `wget http://kent.dl.sourceforge.net/sourceforge/heirloom/mailx-12.4.tar.bz2`
29. `tar xfvj mailx-12.4.tar.bz2`
30. `cd mailx-12.4`
31. `make`
32. `make install`

### Checkout from the SVN repository

4. Login in as root to DEVAS
5. `cd /root`
6. `svn co svn:/XXX.XXX.XXX.XXX/DEVAS`

### SSH

1. `sudo -s` # become root
2. `passwd` # set the root password
3. `ssh-keygen -t dsa`
4. Use the default settings and when they ask about the password just press enter (no password)
5. `cat /root/.ssh/id_dsa.pub >> /root/.ssh/authorized_keys2`
6. `scp id_dsa.pub root@host:/root/`
7. `ssh root@host` # login to the user account on the remote box
8. `cat /root/id_dsa.pub >> /root/.ssh/authorized_keys2`
9. `rm /root/id_dsa.pub`

### Nessus

7. `cp /root/DEVAS/BACKUP/nessusServerConf/nessus-fetch.rc /etc/nessus/nessus-fetch.rc`
8. `cp /root/DEVAS/BACKUP/nessusServerConf/nessusd.conf /etc/nessus/nessusd.conf`
9. `nessus-adduser # user: koersel`
10. `/root/DEVAS/scripts/lib/updateSingleScanPlugins.sh`
11. `cp /root/DEVAS/scripts/nmapConv /usr/local/bin/`
12. `/etc/init.d/nessusd restart`

Interesting fields that matters in the `nessus-fetch.rc` file:

- `proxy=XXX.XXX.XXX.XXX`
- `proxy_port=8080`
- `proxy_username=*****`
- `proxy_password=*****`

Interesting fields that matters in the `nessusd.conf` file

- `nasl_no_signature_check = yes`

We have modified the `nmap.nasl` file so the signature that is on that file is not validated anymore. The signature on that file is included in the `nmap.nasl` file.

## Crontab

Set up cron jobs for launching the system

2. `cp /root/DEVAS/BACKUP/crontab /var/spool/cron/crontabs/root`

## Access to files

Only root should be able to read,execute and write the files.

3. `cd /root`
4. `chmod -R go-rwx DEVAS`

## Client installation for distributed scanning

### Install packages

1. `apt-get install -y sun-java5-jdk sun-java5-jre`
2. `apt-get install -y mysql-server`
3. `apt-get install -y nessus nessusd`
4. `apt-get install -y tomcat5.5`
5. `apt-get install -y subversion libc6-dev flex bison g++`
6. `apt-get install -y libnmap-parser-perl nmap xprobe2 multital`
7. `wget http://heanet.dl.sourceforge.net/sourceforge/sinfp/SinFP-2.06-1.tar.gz`
8. `tar xfvz SinFP-2.06-1.tar.gz`
9. `cd SinFP-2.06-1`
10. `make`
11. `make install`

### Checkout from the SVN repository

1. Login in as root to DEVAS
2. `cd /root`
3. `svn co svn://XXX.XXX.XXX.XXX/DEVAS`

### SSH

1. `sudo -s` # become root
2. `passwd` # set the root password
3. `ssh-keygen -t dsa`
4. Use the default settings and when they ask about the password just press enter (no password)
5. `cat /root/.ssh/id_dsa.pub >> /root/.ssh/authorized_keys2`
6. `scp id_dsa.pub root@host:/root/`
7. `ssh root@host` # login to the user account on the remote box
8. `cat /root/id_dsa.pub >> /root/.ssh/authorized_keys2`
9. `rm /root/id_dsa.pub`

### Nessus

13. `cp /root/DEVAS/BACKUP/nessusServerConf/nessus-fetch.rc /etc/nessus/nessus-fetch.rc`
14. `cp /root/DEVAS/BACKUP/nessusServerConf/nessusd.conf /etc/nessus/nessusd.conf`
15. `nessus-adduser # user: koersel`
16. `/root/DEVAS/scripts/lib/updateSingleScanPlugins.sh`
17. `cp /root/DEVAS/scripts/nmapConv /usr/local/bin/`
18. `/etc/init.d/nessusd restart`

Interesting fields that matters in the `nessus-fetch.rc` file:

- `proxy=XXX.XXX.XXX.XXX`
- `proxy_port=8080`
- `proxy_username=*****`
- `proxy_password=*****`

Interesting fields that matters in the `nessusd.conf` file

- `nasl_no_signature_check = yes`

We have modified the `nmap.nasl` file so the signature that is on that file is not validated anymore. The signature on that file is included in the `nmap.nasl` file.

## Crontab

Set up cron jobs for launching the system

3. `cp /root/DEVAS/BACKUP/crontabClient /var/spool/cron/crontabs/root`

## Access to files

Only root should be able to read,execute and write the files.

5. `cd /root`
6. `chmod -R go-rwx DEVAS`

## Directory layout

### The directory layout

```
drwxr-xr-x 6 root root 4096 Oct 16 15:47 BACKUP/
drwxr-xr-x 3 root root 4096 Oct 16 17:57 conf/
drwxr-xr-x 3 root root 4096 Sep 17 16:51 docs/
drwxr-xr-x 3 root root 4096 Sep 17 11:44 logs/
drwxr-xr-x 5 root root 4096 Oct 1 17:00 results/
drwxr-xr-x 7 root root 4096 Oct 16 17:05 scripts/
drwxr-xr-x 3 root root 4096 Sep 5 11:18 test/
```

### BACKUP

```
-rw-r--r-- 1 root root 504 Sep 2 15:47 crontab
-rw-r--r-- 1 root root 636 Sep 2 15:47 crontabClient
-rw-r--r-- 1 root root 99238 Sep 2 15:47 nasl.vim
drwxr-xr-x 3 root root 4096 Sep 16 17:53 nessusServerConf/
-rw-r--r-- 1 root root 13387 Oct 16 15:44 nmap.nasl
-rwxr-xr-x 1 root root 2554 Sep 2 15:47 nmapConv*
-rw-r--r-- 1 root root 16379 Sep 2 15:47 nmapOriginal.nasl
drwxr-xr-x 3 root root 4096 Sep 17 11:40 oldScripts/
-rw-r--r-- 1 root root 4187 Sep 2 15:47 os_fingerprint.nasl
```

- crontab file is the crontab for the single scanning mode and for the server in the distributed mode.
- Nasl.vim file is syntax highlighting for vim.
- nessusServerConf/ contains nessusd.conf and nessus-fetch.rc. Nessusd.conf is the nessusd configuration file. Nessus-fetch.rc is the configuration file for downloading new plugins which contains user credentials and proxy settings.
- Nmap.nasl file is our modified Nessus plugins script.
- nmapConv is a wrapper around nmap for the nmap.nasl Nessus script.
- nmapOriginal.nasl file is the unmodified Nessus plugin script.
- oldScripts/ contains old DEVAS scripts that are not used anymore.
- os\_fingerprint.nasl file is our modified Nessus script file.

### conf

```
-rw-r--r-- 1 root root 12 Oct 16 12:38 agents.txt
-rw-r--r-- 1 root root 1056 Oct 16 15:42 blackListPlugins.txt
-rw-r--r-- 1 root root 0 Oct 16 15:42 blackListTargets.txt
-rw----- 1 root root 270008 Oct 16 15:19 nessusConfig.txt
-rw-r--r-- 1 root root 53 Oct 10 17:52 targets.txt
```

- agents.txt file contains a list of IPs which are agents for the distributed scanning.
- blackListPlugins.txt file contains a list of Nessus plugins ID that are black listened.
- blackListTargets.txt file contains a list of IPs that are black listened.
- nessusConfig.txt file is the configuration file for all Nessus clients.
- targets.txt file contains a list of targets.



## logs

```
-rw-r--r-- 1 root root 21444 Oct  1 17:00 distributedLog.txt
-rw-r--r-- 1 root root 14831 Oct 16 15:02 scanLog.txt
```

- distributedLog.txt file contains the log for DEVAS distributed server.
- scanLog.txt file contains the log for DEVAS single scanning mode and the log file for the agent.

## results

```
drwxr-xr-x 3 root root 4096 Oct 17 11:21 kbs/
-rw----- 1 root root 12090 Sep 17 11:13 nessus-report-2008-09-01.nbe
drwxr-xr-x 3 root root 4096 Oct  1 17:00 resultsPerAgent/
```

- The kbs directory contains archived Nessus internal databases after every scan for audit and tracking purposes.
- nessus-report-2008-09-01.nbe file is a Nessus report.
- The resultsPerAgent directory contains sub-reports from agents.

## scripts

```
drwxr-xr-x 3 root root 4096 Oct 16 15:39 debugScripts/
-rwxr-xr-x 1 root root 18350 Oct 16 15:39 distributedNessusScan.sh*
drwxr-xr-x 3 root root 4096 Oct 16 17:05 lib/
drwxr-xr-x 3 root root 4096 Oct  8 16:05 nbeScripts/
-rwxr-xr-x 1 root root 13414 Oct 17 11:15 singleNessusScan.sh*
drwxr-xr-x 3 root root 4096 Sep 16 18:36 sqlScripts/
```

- The debugScripts directory contains debug scripts.
- distributedNessusScan.sh file is the DEVAS server for the distributed scanning mode.
- The lib directory contains scripts that are run or included from other scripts. They are not normally run standalone.
- The nbeScripts directory contains scripts for analyzing Nessus nbe result files.
- singleNessusScan.sh file is the DEVAS script for the single scanning mode.

## Scripts/debugScripts

```
-rwxr-xr-x 1 root root 757 Oct 16 15:39 checkAgentsAreUp.sh*
-rwxr-xr-x 1 root root 148 Sep 17 11:40 debugAgent.sh*
-rwxr-xr-x 1 root root 270 Sep 17 11:41 monitorAgents.sh*
```

- checkAgentsAreUp.sh file checks that all agents added in the conf/agents.txt file have setup SSH correctly.

- debugAgent.sh opens all log files for DEVAS and Nessus for showing information in realtime on one screen.
- monitorAgents.sh opens all DEVAS logs for all agents that are alive on one display and displays data in realtime.

## Scripts/nbeScripts

```
-rwxr-xr-x 1 root root 26485 Sep  2 15:47 nbeDiff.sh*
-rwxr-xr-x 1 root root 17152 Sep  2 15:47 nbeQuery.sh*
```

- nbeDiff.sh is a script for analyzing and making diffs between Nessus result files.
- nbeQuery.sh is a script for analyzing and make statistics from a single Nessus result file.

## Scripts/lib

```
-rwxr-xr-x 1 root root 13574 Oct 28 14:49 agent.sh*
-rwxr-xr-x 1 root root  2090 Oct 16 15:45 checkNessusConfigFiles.sh*
-rwxr-xr-x 1 root root  2099 Oct 16 15:45 generateTargetBlackList.pl*
-rwxr-xr-x 1 root root 34672 Oct 28 14:49 updateDistributedScanPlugins.sh*
-rwxr-xr-x 1 root root 34336 Oct 28 14:51 updateSingleScanPlugins.sh*
```

- agent.sh file is the DEVAS agent for the distributed scanning mode.
- checkNessusConfigFiles.sh file checks the nessusd.conf and the nessusConf.txt file that the configuration is set correctly before the Nessus scan.
- generateTargetBlackList.pl file generates black listed computers based on a certain criteria.
- updateDistributedScanPlugins.sh file downloads new Nessus plugins and updates the configuration file for the DEVAS distributed scanning mode
- updateSingleScanPlugins.sh file downloads new Nessus plugins and updates the configuration file for the DEVAS single scanning mode.

# 12. Glossary

---

ACK – Acknowledge

ACL – Access Control List

AD – Active Directory

ARPANET – Advanced Research Projects Agency Network

CIDR – Classless Inter Domain Routing Scheme

CRC – Checksum Redundancy Check

DARPA – Defense Advanced Research Projects Agency

DHCP – Dynamic Host Configuration Protocol

DNS – Domain Name System

DOS – Denial of Service

Extranet – External Network

FTP – File Transfer Protocol

GPL – Gnu Public License

HTTP – Hyper Text Transfer Protocol

IANA – Internet Assigned Numbers Authority

ICANN – Internet Corporation for Assigned Names and Numbers

ICMP – Internet Control Message Protocol

Intranet – Internal Network

IPID – Internet Protocol Identification

IPSec – Internet Protocol Security

IRC – Internet Relay

KB - Nessus Knowledge Database

LAN – Local Area Network

LAN – Local Area Network

LDAP – Lightweight Directory Protocol

NASL – Nessus Attack Scripting Language

NAT – Network Address Translation

NCP – Network Control Protocol

NFS – Network File Server

NTP – Network Time Protocol

OS – Operating System

OSI – Open Systems Interconnection

PAT – Port Address Translation

POP3 – Post Office Protocol 3

RST – Reset

SCP – Secure Copy

SMTP – Simple Mail Transfer Protocol

SNMP – Simple Network Management Protocol

SQL –Structured Query Language

SSH – Secure Shell

SSL – Secure Socket Layer

SYN – Synchronize

TCP/IP – Transmission Control Protocol / Internet Protocol

TTL – Time To Live

UDP – User data Protocol

VOIP – Voice over Internet Protocol

XML – Extensible Markup Language

# 13. Bibliography

---

- A comprehensive guide to nmap with screenshots.* (n.d.). Retrieved 09 10, 2008, from <http://www.linuxhaxor.net/2007/07/10/a-comprehensive-guide-to-nmap-with-screenshots/4/>
- Active Directory.* (n.d.). Retrieved 8 4, 2008, from [http://en.wikipedia.org/wiki/Active\\_Directory](http://en.wikipedia.org/wiki/Active_Directory)
- Archibald, N., Ramirez, G., & Rathaus, N. (2005). *Nessus, Snort, & Ethereal Power Tools: Customizing Open Source Security Applications*. Rockland: Syngress.
- ARPANET.* (n.d.). Retrieved 11 12, 2008, from <http://en.wikipedia.org/wiki/ARPANET>
- Awk.* (n.d.). Retrieved 8 20, 2008, from <http://en.wikipedia.org/wiki/Awk>
- Bash.* (n.d.). Retrieved 8 20, 2008, from <http://www.gnu.org/software/bash/>
- Beale, J. (2008). *Nessus Network Auditing, Second Edition*. Rockland: Syngress.
- Bugtraq.* (n.d.). Retrieved 8 13, 2008, from <http://www.securityfocus.com/archive/1>
- Comer, D. E. (2005). *Internetworking with TCP/IP, Vol 1 (5th Edition)*. Upper Saddle River: Prentice Hall.
- Common Vulnerability and Exposures.* (n.d.). Retrieved 8 13, 2008, from <http://cve.mitre.org/>
- Correlation between Snort and Nessus and easy supervision with Prelude.* (n.d.). Retrieved 12 22, 2008, from <http://linuxgateway.free.fr/>
- Cron.* (n.d.). Retrieved 8 4, 2008, from <http://en.wikipedia.org/wiki/Cron>
- DARPA.* (n.d.). Retrieved 12 22, 2008, from [http://en.wikipedia.org/wiki/Defense\\_Advanced\\_Research\\_Projects\\_Agency](http://en.wikipedia.org/wiki/Defense_Advanced_Research_Projects_Agency)
- DMZ.* (n.d.). Retrieved 10 16, 2008, from <http://en.wikipedia.org/wiki/DMZ>
- Finding Sensitive Data as a Consultant with Nessus.* (n.d.). Retrieved 12 6, 2008, from <http://blog.tenablesecurity.com/2007/08/finding-sensitive.html>
- GPL.* (n.d.). Retrieved 8 4, 2008, from <http://www.gnu.org/licenses/gpl.html>
- IANA.* (n.d.). Retrieved 12 22, 2008, from <http://www.iana.org/assignments/port-numbers>
- Information security.* (n.d.). Retrieved 12 4, 2008, from [http://en.wikipedia.org/wiki/Information\\_security](http://en.wikipedia.org/wiki/Information_security)
- Introduction to Nessus.* (n.d.). Retrieved 8 4, 2008, from <http://www.securityfocus.com/infocus/1741>
- IPv4.* (n.d.). Retrieved 11 5, 2008, from <http://en.wikipedia.org/wiki/IPv4>

*LDAP*. (n.d.). Retrieved 8 4, 2008, from [http://en.wikipedia.org/wiki/Lightweight\\_Directory\\_Access\\_Protocol](http://en.wikipedia.org/wiki/Lightweight_Directory_Access_Protocol)

*List of TCP and UDP port numbers*. (n.d.). Retrieved 11 5, 2008, from [http://en.wikipedia.org/wiki/List\\_of\\_TCP\\_and\\_UDP\\_port\\_numbers](http://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers)

*Microsoft Security Bulletin*. (n.d.). Retrieved 8 13, 2008, from <http://www.microsoft.com/technet/security/Current.aspx>

*Multitail*. (n.d.). Retrieved 9 14, 2008, from <http://www.vanheusden.com/multitail/>

Naugle, M. (1998). *Illustrated TCP/IP: a graphic guide to the protocol suite*. New York: John Wiley & Sons.

*Nessus 3.x vs. Nessus 2.x*. (n.d.). Retrieved 10 10, 2008, from <http://www.nessus.org/documentation/index.php?doc=nessus3>

*Nessus credential checks*. (n.d.). Retrieved 11 3, 2008, from [http://www.nessus.org/documentation/nessus\\_credential\\_checks.pdf](http://www.nessus.org/documentation/nessus_credential_checks.pdf)

*Nessus, Part 2: Scanning*. (n.d.). Retrieved 8 4, 2008, from <http://www.securityfocus.com/infocus/1753>

*Nessus, Part 3: Analysing Reports*. (n.d.). Retrieved 8 4, 2008, from <http://www.securityfocus.com/infocus/1759>

*Network interface card*. (n.d.). Retrieved 10 18, 2008, from [http://en.wikipedia.org/wiki/Network\\_interface\\_card](http://en.wikipedia.org/wiki/Network_interface_card)

*Network Time Protocol*. (n.d.). Retrieved 11 2, 2008, from [http://en.wikipedia.org/wiki/Network\\_Time\\_Protocol](http://en.wikipedia.org/wiki/Network_Time_Protocol)

*Network uptime*. (n.d.). Retrieved 12 12, 2008, from <http://www.networkuptime.com/nmap/page3-3.shtml>

*Nmap*. (n.d.). Retrieved 12 3, 2008, from <http://en.wikipedia.org/wiki/Nmap>

*Nmap Matrix reloaded*. (n.d.). Retrieved 12 22, 2008, from <http://nmap.org/images/matrix/trinity-nmapscreen-hd-crop-1200x728.jpg>

*Nmap OS Detection*. (n.d.). Retrieved 12 15, 2008, from <http://nmap.org/book/osdetect.html>

*Nmap Port Scanning Techniques*. (n.d.). Retrieved 12 1, 2008, from <http://nmap.org/book/man-port-scanning-techniques.html>

*Nmap Timing and Performance*. (n.d.). Retrieved 9 20, 2008, from <http://nmap.org/book/man-performance.html>

*Open Source Vulnerability Database*. (n.d.). Retrieved 8 13, 2008, from <http://osvdb.org/>

*OpenSSH*. (n.d.). Retrieved 11 9, 2008, from <http://www.openssh.com/>

Orebaugh, A., & Pinkard, B. (2008). *Nmap in the Enterprise: Your Guide to Network Scanning*. Rockland: Syngress.

*OSSIM*. (n.d.). Retrieved 12 17, 2008, from <http://www.ossim.net/>

*PCI DSS*. (n.d.). Retrieved 1 7, 2009, from [http://en.wikipedia.org/wiki/PCI\\_DSS](http://en.wikipedia.org/wiki/PCI_DSS)

*Perl*. (n.d.). Retrieved 9 10, 2008, from <http://www.perl.org/>

*Protocols in multi-service networks*. (n.d.). Retrieved 1 5, 2009, from <http://openlearn.open.ac.uk/mod/resource/view.php?id=175853>

*Python*. (n.d.). Retrieved 9 5, 2008, from <http://www.python.org/>

*Reuters*. (n.d.). Retrieved 1 16, 2009, from <http://en.wikipedia.org/wiki/Reuters>

*rsync*. (n.d.). Retrieved 12 3, 2008, from <http://samba.anu.edu.au/rsync/>

*Ruby*. (n.d.). Retrieved 8 30, 2008, from <http://www.ruby-lang.org/en/>

*SATAN security scanner*. (n.d.). Retrieved 12 5, 2008, from [http://en.wikipedia.org/wiki/Security\\_Administrator\\_Tool\\_for\\_Analyzing\\_Networks](http://en.wikipedia.org/wiki/Security_Administrator_Tool_for_Analyzing_Networks)

*scp*. (n.d.). Retrieved 8 3, 2008, from <http://www.openssh.com/>

*Security Policy Compliance Testing*. (n.d.). Retrieved 12 20, 2008, from <http://www.networkuptime.com/nmap/page12-04.shtml>

*Sed*. (n.d.). Retrieved 8 22, 2008, from <http://en.wikipedia.org/wiki/Sed>

*SinFP*. (n.d.). Retrieved 10 16, 2008, from <http://www.gomor.org/bin/view/Sinfp/WebHome>

*Snort*. (n.d.). Retrieved 12 19, 2008, from <http://www.snort.org/>

Stevens, W. R. (1994). *TCP/IP Illustrated, Volume 1: The Protocols*. Reading, Mass: Addison-Wesley.

Stevens, W. R., & Rago, S. A. (2005). *Advanced Programming in the UNIX(R) Environment (2nd Edition)*. Reading, Mass: Addison-Wesley.

*TCP/IP*. (n.d.). Retrieved 11 30, 2008, from [http://en.wikipedia.org/wiki/TCP\\_IP](http://en.wikipedia.org/wiki/TCP_IP)

*TCP/IP stack fingerprinting*. (n.d.). Retrieved 12 4, 2008, from [http://en.wikipedia.org/wiki/TCP/IP\\_stack\\_fingerprinting](http://en.wikipedia.org/wiki/TCP/IP_stack_fingerprinting)

*TCP/IP versus OSI*. (n.d.). Retrieved 12 16, 2008, from <http://www.cellsoft.de/telecom/tcpiposi.htm>

*Tenable homepage*. (n.d.). Retrieved from [www.tenable.com](http://www.tenable.com)

*Tenable security*. (n.d.). (Nessus) Retrieved 8 3, 2008, from <http://www.tenablesecurity.com>

*The Art of Port Scanning*. (n.d.). Retrieved 11 17, 2008, from [www.phrack.org/issues.html?issue=51&id=11](http://www.phrack.org/issues.html?issue=51&id=11)

*The TCP/IP Guide.* (n.d.). Retrieved 10 17, 2008, from <http://www.tcpipguide.com/free/index.htm>

*Understanding the Nessus "Safe Checks" Option.* (n.d.). Retrieved 8 15, 2008, from [http://blog.tenablesecurity.com/2006/09/understanding\\_t.html](http://blog.tenablesecurity.com/2006/09/understanding_t.html)

*Update-nessusrc.* (n.d.). Retrieved 8 16, 2008, from <http://www.tifaware.com/perl/update-nessusrc/>

*Using Nessus to scan hosts behind a firewall.* (n.d.). Retrieved 9 5, 2008, from [http://blog.tenablesecurity.com/2006/08/using\\_nessus\\_to.html](http://blog.tenablesecurity.com/2006/08/using_nessus_to.html)

*wikipedia.* (n.d.). Retrieved from [www.wikipedia.com/service\\_file](http://www.wikipedia.com/service_file)

*Wikipedia.* (n.d.). Retrieved from [en.wikipedia.com/information\\_security](http://en.wikipedia.com/information_security)

*Virtual IP.* (n.d.). Retrieved 10 30, 2008, from [http://en.wikipedia.org/wiki/Virtual\\_IP](http://en.wikipedia.org/wiki/Virtual_IP)

*Xprobe.* (n.d.). Retrieved 9 3, 2008, from <http://xprobe.sourceforge.net/>



# 14. Figures Index

---

Figure 1 Ongoing security process .....	8
Figure 2 Intranet, Extranet and Internet.....	10
Figure 3 OSI Model.....	12
Figure 4 Complete TCP session .....	16
Figure 5 TCP state diagram (RFC 793).....	17
Figure 6 Common ports and descriptions.....	20
Figure 7 Image from movie Matrix Reloaded. ....	23
Figure 8 Zenmap .....	24
Figure 9 Nmap -sP scan .....	26
Figure 10 Nmap's TCP SYN scan.....	28
Figure 11 Ongoing security process .....	30
Figure 12 Nessus plugins interaction with the KB.....	31
Figure 13 UML state diagram of all stages in Nessus.....	32
Figure 14 Island Topology .....	36
Figure 15 Flat Topology.....	37
Figure 16 Star Topology .....	37
Figure 17 Crontab settings .....	39
Figure 18 Nessus client Plugin tab .....	40
Figure 19 Nessus Credentials tab.....	41
Figure 20 Nessus client Scan Option tab.....	41
Figure 21 Nessus client Prefs tab.....	42
Figure 22 Nessus client Prefs tab 3 .....	42
Figure 23 Nessus client Prefs tab 2 .....	42
Figure 24 Nessus client KB tab .....	42
Figure 25 Nessus plugin configuration file settings .....	43
Figure 26 State diagram of blacklisting process.....	46
Figure 27 Original nmap.nasl code snippet .....	48
Figure 28 New nmap.nasl code snippet.....	48
Figure 29 New os_fingerprint.nasl pseudo code .....	49
Figure 30 State diagram of the Nessus Nmap change .....	50
Figure 31 D.E.V.A.S agent log.....	50
Figure 32 D.E.V.A.S management system log .....	51
Figure 33 D.E.V.A.S single mode state diagram .....	52
Figure 34 D.E.V.A.S network setup behind firewall .....	54
Figure 35 D.E.V.A.S example target list.....	55
Figure 36 Simplified network perimeter architecture .....	56
Figure 37 D.E.V.A.S distributed mode state diagram.....	57
Figure 38 NbeQuery.sh result statistics .....	59
Figure 39 NbeDiff.sh differential result report on hosts and services.....	60
Figure 40 NbeDiff.sh result report on new vulnerabilities .....	61
Figure 41 NbeDiff.sh result report on patched services .....	62
Figure 42 NbeDiff.sh result report on unpatched services.....	62

Figure 43 Correlation between Nessus and Snort IDS.....	67
---	----