ANEX: AUTOMATED NETWORK EXPLOITATION THROUGH PENETRATION TESTING

A Thesis

presented to

the Faculty of California Polytechnic State University,

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Computer Science

by

Eric Francis Dazet

June 2016

COMMITTEE MEMBERSHIP

TITLE:	ANEX: Automated Network Exploitation through Penetration Testing
AUTHOR:	Eric Francis Dazet
DATE SUBMITTED:	June 2016
COMMITTEE CHAIR:	Bruce Edward DeBruhl II, Ph.D. Assistant Professor of Computer Science
COMMITTEE MEMBER:	Zachary J. Peterson, Ph.D. Associate Professor of Computer Science
COMMITTEE MEMBER:	Phillip Nico, Ph.D. Professor of Computer Science

ABSTRACT

ANEX: Automated Network Exploitation through Penetration Testing

Eric Francis Dazet

Cyber attacks are a growing concern in our modern world, making security evaluation a critical venture. Penetration testing, the process of attempting to compromise a computer network with controlled tests, is a proven method of evaluating a system's security measures. However, penetration tests, and preventive security analysis in general, require considerable investments in money, time, and labor, which can cause them to be overlooked. Alternatively, automated penetration testing programs are used to conduct a security evaluation with less user effort, lower cost, and in a shorter period of time than manual penetration tests. The trade-off is that automated penetration testing tools are not as effective as manual tests. They are not as flexible as manual testing, cannot discover every vulnerability, and can lead to a false sense of security. The development of better automated tools can help organizations quickly and frequently know the state of their security measures and can help improve the manual penetration testing process by accelerating repetitive tasks without sacrificing results.

This thesis presents Automated Network Exploitation through Penetration Testing (ANEX), an automated penetration testing system designed to infiltrate a computer network and map paths from a compromised network machine to a specified target machine. Our goal is to provide an effective security evaluation solution with minimal user involvement that is easily deployable in an existing system. ANEX demonstrates that important security information can be gathered through automated tools based solely on free-to-use programs. ANEX can also enhance the manual penetration testing process by quickly accumulating information about each machine to develop more focused testing procedures.

Our results show that we are able to successfully infiltrate multiple network levels and exploit machines not directly accessible to our testing machine with mixed success. Overall, our design shows the efficacy of utilizing automated and open-source tools for penetration testing.

ACKNOWLEDGMENTS

I'd like to thank my parents for their constant belief and support, Dr. DeBruhl for his guidance, the lights in the grad security lab for always working, and Andrew Gunther for uploading this template.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

The advancements in computational power and efficiency, the flexibility of interconnected networks, and the growth of physical and virtual data storage capabilities provide benefits essential to successful business [36]. However, this increasing reliance on technology has turned more and more companies into targets for malicious activity [36]. According to Symantec's yearly threat report, in 2014, there were several areas of growing concern from both a cyber attack and defense point of view. For example, the twenty-four new zero-day vulnerabilities, security holes in software programs that are unknown to their vendors, was the highest total for a single year in Symantec's history and affected popular services such as Adobe's Flash Player and Internet Explorer [40]. One of these vulnerabilities was the Heartbleed security bug, a widely-publicized vulnerability in the TLS security protocol [10]. A possibly more alarming fact is the amount of time it took for vendors to respond and release patches for these zero-day vulnerabilities. In the case of Heartbleed, the patch took 204 days [40]. Other techniques, including, but not limited to, spear-phishing, targeted communication that tries to trick users into providing private information, and trojanizing, the hiding of infectious code within legitimate programs, were also extremely prevalent and effective. Symantec also concluded that in 2014, 60 percent of all targeted attacks were aimed at small and medium sized organizations, organizations that often have less resources to devote to security measures [40].

More recently, Symantec's monthly report for September 2015 indicated that the finance, insurance and real estate sector was the target for twenty-seven percent of all targeted attacks while the agriculture, forestry, and fishing sector was the most targeted sector for malware, 1 in every 308 emails, and phishing attempts, 1 in every

988 emails [37]. Additionally, forty-six percent of all spear-phishing attempts in September alone were on large businesses [37]. This statistical analysis emphasizes the fact that no company or industry is completely impervious to cyber attacks.

Preventive measures against cyber threats are now a fundamental necessity to help protect private data and maintain the integrity of systems. Directly related to the previous notion is the importance of monitoring and testing the effectiveness of deployed security measures in order to maintain resiliency to the evolving organism that is cyber crime [36].

Penetration testing, or ethical hacking, is one of these testing methods and is one of the most popular software security practices [2]. Penetration testing is the process of identifying ways to exploit vulnerabilities for the purpose of circumventing or defeating the security features of a system [29]. The overall goal of penetration testing is to find ways around security measures in a controlled environment to identify weak spots and fix them before the same weaknesses are exploited by people with malicious intentions [22].

1.1 Problem

Although cyber attacks threaten valuable resources in our modern world, namely intellectual property and the Internet, research has shown that the high value of intellectual property and its susceptibility to cyber attacks often go unappreciated [9]. This fact is highlighted by the U.S. Office of Personnel Management (OPM) security breach that occurred in March of 2014 [23]. After the breach was discovered, an audit conducted by the Office of the Inspector General revealed that OPM, which is responsible for storing the personal information of Federal employees and Federal job applicants, was lacking up-to-date security features in several areas. For instance, the audit found that OPM did not have a complete agency-wide risk management system,

there wasn't a comprehensive inventory of servers, databases, and network devices, known weaknesses weren't being timely addressed and incorporated into plans of action, and there were no standards or compliance checks for any operating platforms [23]. This real-world example emphasizes the need for proper security measures to be put in place and maintaining and updating them appropriately to mitigate the amount of potential damage.

It is important to recognize that security measures are not one-time additions to a system. Proven preventive solutions, like firewalls, can be effective, but can lead to a false sense of security. Therefore, it is critical that security solutions like firewalls are routinely tested and maintained [2]. Similarly, computer applications and services, which have the potential to introduce security flaws into systems, must be kept up to date and patched when necessary. Proper network configuration and permissions also need to be in place to restrict unauthorized data access and unauthorized connections. System security is a continuous endeavor that increases in both importance and resources as systems expand. Fortunately, there are a number of tools and techniques, such as penetration testing, that can help assess this area of concern.

Penetration testing has a clear definition, repeated here as the process of identifying ways to exploit vulnerabilities for the purpose of circumventing or defeating the security features of a system [36]. However, there is also a related process known as vulnerability assessment that is oftentimes confused with penetration testing despite having a different meaning. Vulnerability assessment is the process of identifying, ranking, and reporting vulnerabilities that, if exploited, could compromise the system [36]. A common example of this is conducting a port scan on a machine and then identifying potential vulnerabilities of the ports that are found open. Given the definition above, penetration testing is an extension of vulnerability assessment in that the exploitation of potential security holes can use the results from a vulnerability assessment as a starting point. An example of a penetration test would be using

vulnerability information to launch exploits at a machine in order to gain access and then steal login credentials, install a backdoor into the system, setup a key logger, and other methods that could compromise the security of the machine or its connected network.

1.2 Solution

We present Automated Network Exploitation through Penetration Testing (ANEX), a tool that combines the desirable qualities from both vulnerability assessment and penetration testing to perform what we define as machine targeting, the process of identifying exploitable paths within a network to a target machine. Given a initial network and a particular machine of interest, maybe one that manages a sensitive database or server, ANEX attempts to compromise all of the machines and networks that it can find that are associated with the initial network. Once no additional machines or networks are found, all of the compromised machines and networks are related to the target via parent-child relationships to indicate if and how an attacker can reach a target from a specific location within the network. A high level architecture diagram of ANEX can be found in Figure 1.1.

The specific question that we address is the following: *can we design an automated penetration testing solution that provides useful security information, improves its own performance using data from previous executions, and is able to exploit as many machines and networks as possible?* Our useful function is the machine targeting process. Machine targeting is useful because it can quickly identify if a target is vulnerable to common exploits and identify which machines or networks require security improvements. Improving performance over time is key for both execution runtime and reducing the number of exploits launched at a given machine. Exploiting

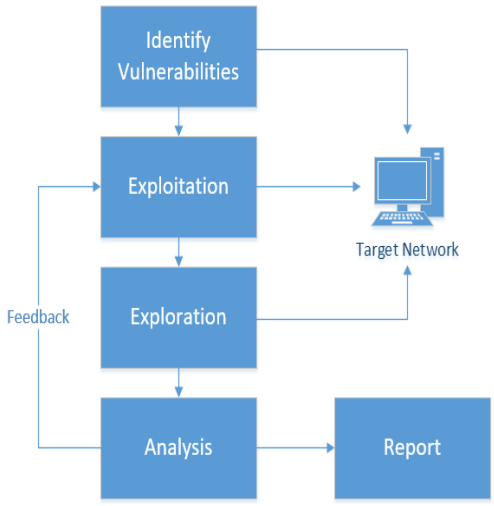

Figure 1.1: High level architecture diagram of ANEX

as many machines and networks as possible provides a better security assessment of the system being tested.

Automated penetration testing solutions are not new. Tools such as FastTrack's Autopwn [14], Core's Impact Pro [1], and Immunity's Canvas [12] all successfully incorporate elements from both vulnerability assessment and penetration testing into their products. Since this was our goal as well, we analyzed these other tools to see why they are successful and then used our analysis to influence our design.

ANEX is designed to run with minimal resources and user interaction while maintaining efficiency, adaptability, and safety in the exploitation process. When complete, ANEX provides a comprehensive report of its testing results and saves information gathered during its execution for future use. Our overall goal is to contribute a solution to the penetration testing domain and promote further development of utilities that combine automated vulnerability assessment tools and manual penetration testing techniques.

Chapter 2

BACKGROUND

In this chapter, we define penetration testing, why penetration testing is important, common types of penetration tests, and the different methodologies that are used. We then narrow our focus and outline the penetration testing process in detail. We then compare penetration testing to vulnerability assessment, a separate process that is often misconstrued as penetration testing in industry. Finally, we describe some popular tools that are used to aid the penetration testing process.

A lingering question around penetration testing is why do it. Why invest money and time to try and exploit a system that is "secure"? Why run potentially dangerous tests that can harm working infrastructure? The simple answer is that it is better for the owners of a system to identify flaws that can compromise their system and fix them so that hackers cannot exploit those same flaws and cause damage [14]. Penetration testing helps shrink the gap between safeguarding a security system and the exposure of its risks by determining if the implemented security measures are adequate and working [8]. In 2015, the Ponemon Research Institute conducted a study to analyze the cost of data breaches across 350 companies in eleven countries. They found the average cost of a data breach to be $3.79 million dollars [30]. Typically, a penetration test will cost significantly less than $3.79 million dollars, but the actual cost of a penetration test is a difficult figure to obtain. Prices vary widely depending on the system in question, the time needed to perform the test, the number of people involved in the testing, and other factors. This leads into the debate over spending money on preventive security measures vs. the cost of potential breaches. However, since the cost of a penetration test is typically a fraction of the data breach figure, it makes good business sense to invest in penetration testing and use the information

it provides to keep security protocols up to date and learn how to better identify potential vulnerabilities in the future [4].

Proper penetration testing takes into account more than just software applications and deployed security measures. For the most comprehensive results, a system's hardware components and the people that interact with the system must be considered as well [4]. There are three common variations of penetration testing: network testing, application testing, and social engineering. Network penetration testing attempts to identify and exploit holes in the design, implementation, and operation of a system's network in order to discover potential software or hardware vulnerabilities that would allow attackers into the test system. Application penetration testing determines the effectiveness of installed security measures, such as firewalls, in order to make sure they are working properly. Social engineering leverages human interaction in order to obtain or compromise sensitive system information. This strategy is used to manage unauthorized access opportunities by assessing the security awareness of employees [4]. In this work, we focus on a combination of network and application penetration testing. Our design tests the effectiveness of local firewalls, application maintenance, and the physical configuration of a network.

Penetration tests are classified into three categories: black box, white box, and grey box [4]. Black box penetration tests are conducted with no prior knowledge about the system being tested in order to determine if accessing the system is possible and if so, how much data can be compromised. This method simulates attackers that have no information about their target. White box penetration tests are conducted with all of the available information about the system being tested. This can involve the testers and the owners of the system working together on what to focus on during the testing process. Grey box penetration tests are in between black box and white box testing where only partial information about the system are provided to the testers [4]. In this work, we focus on black-box testing.

Shifting to network penetration testing, the process can be separated into three phases: preparation, execution, and post-execution [29]. In the preparation phase, it is important that the client and tester communicate to determine the system to test, the scope of the testing, the type of testing, and how to perform the tests [29]. This is also the time for the client to convey the goals of the testing and for the tester to educate the client on what to expect from the testing, such as test duration, what the testing results will look like, etc. [14]. Next, the tester tries to learn as much as possible about the target system. This involves acquiring documentation about the systems components and past penetration testing results, running sample tests, and using third-party resources such as vulnerability scanners [34]. The final step before the execution phase is vulnerability modelling and analysis. The results from the vulnerability analysis are used to formulate attacks, specific exploits and their targets, to use in the execution phase [14].

The execution phase launches the attacks formulated in the preparation phase at the target system or systems. Even though this process sounds simple, it is important to think about the approaches and tools to use in order to provide the most meaningful results. For example, launching every available exploit at once and seeing what happens is not an effective procedure [14]. A large spike in activity can be easily detected and it might not be obvious which exploits were successful or not. It is also important to think ahead and configure the exploits to return useful information that will be used in the post-exploitation phase. This includes determining the types of exploits and exploit payloads to achieve desired results. After determining which exploits were successful, the post-execution phase begins. Post-execution refers to any actions that follow the initial compromise of the system. This includes techniques like pivoting, using a compromised host as a bridge to reach other networks or systems that aren't directly accessible to the attacking machine [11], and privilege escalation, the process of gaining root or administrator privileges if not granted already through

exploitation, that can provide the attacker access to additional systems and data [29]. The post-exploitation phase is critical because this is where the attacker can possess root-level access over the entire system and potentially do the most damage [14].

The post-execution phase also involves compiling all of the test results. Properly reporting the test results is the most important part of the penetration testing process because the results need to accuratly depict the state of the system and make sense to the people responsible for implementing the appropriate changes [14]. The report should include the tests that were run, how they were run, why they were run, and most importantly, the issues that were found and suggestions on how to fix them [14]. If possible, after the appropriate changes are deployed, a re-test should take place in order to validate that the new changes fixed the old problems. It is also important to document and report any changes made to the system during the execution phase and then reverse the changes once testing is complete [29].

Automated penetration testing combines the three phases explained above into readily-deployable, security assessment tools. The use of automated testing procedures has grown in recent years due to the frequent technological changes that are needed to alter and scale networks [8]. Manual penetration tests can take days or even weeks to complete and require lots of planning and labor. Automated solutions can provide results in a couple of hours, be used as frequently as desired, and save labor costs [8]. However, repeated use of these tools can lead to a sense of false security. The tools themselves are only as good as their developers, they cannot guarantee a vulnerability discovery rate of 100%, and are not as flexible as manual tests [8]. Even so, the speed and efficiency of automated penetration testing tools makes them valuable resources on their own and when used to supplement manuel penetration tests.

Table 2.1: Comparison Between Vulnerability Assessment and Penetration Testing, from [29]

	Vulnerability Assessment	Penetration Testing
Purpose	Identify and rank vulnerabilities	Identify ways to exploit vulnerabilities
When	Quarterly or after significant changes	Annually or after significant changes
How	Typically uses automated tools with manual identification of issues, can be done completely manually, static and dynamic analysis (fuzzing)	Manual process that may include automated, tools like vulnerability scanners, typically conducted by experienced third-party engineers
Report	Contains potential risks as a result of the identified vulnerabilities	Detailed description of each verified vulnerability (risks of not fixing it, how and to what extent it can be exploited)
Duration	Relatively short (several seconds to several minutes per host)	Depends on size of system, scope of test, and issues uncovered during testing (days to weeks)
Cost	Some automated tools are free, prices vary between third party services (typically quoted price host)	Same parameters as duration, varies between third-party services

One common point of confusion around penetration testing is the difference between penetration testing and vulnerability assessment. Vulnerability assessment is the analysis and reporting of the severity and impact of vulnerabilities identified within a system [34]. Using the definition of penetration testing from Section 1.1, penetration testing is an extension of vulnerability assessment because the final results of a vulnerability assessment define the attack vectors in the preparation phase of a penetration test. Put simply, vulnerability assessment indicates potential points in a system that can be exploited while penetration testing attempts to exploit those points to figure out if in fact an attacker can breach the system, how much access an attacker can acquire, and how to repair the vulnerable points. A more quantified side-by-side comparison of vulnerability assessment and penetration testing is provided in Table 2.1.

Penetration testing, and security assessment in general, is not without its limitations, particularly from legal and ethical viewpoints. For example, if a penetration test is conducted on a system containing sensitive financial data, preserving the confidentiality of the data by omitting it from a test can detriment the value of the testing because the tester is now contrained where an attacker wouldn't be [8]. Organizations also need to adhere to specific laws and policies, which can make comprehensive, real-world testing difficult. The testers themselves also need to act within legal and ethical guidelines. An example would be testing a system within constraints provided by an organization and not extending beyond those constraints without authorization even if it is safe, easy, or beneficial to do so. Therefore, it is important to define and explicitly communicate the parameters of the test in order to hold the testers liable for their actions [7].

2.1 Popular Penetration Testing Tools

2.1.1 Metasploit Framework

The Metasploit project [14] is a free-to-download, open-source exploitation framework designed to be a comprehensive development environment for penetration testing. Initially released in 2003, Metasploit 1.0 quickly gained popularity within the penetration testing and information security communities. Since being acquired in 2009 by Rapid7, an industry leader in vulnerability scanning, the Metasploit project has grown exponentially; initially released with 11 exploits in 2003, the most recent version (4.11.5) now has over 1500 exploits [33]. In addition to exploits, Metasploit contains other tools to aid the various stages of penetration testing. These tools include scanners, payloads, and session handles. Various network and port scanners are integrated into Metasploit in order to assist host and vulnerability discovery. Metasploit also provides a large number of payloads, code packages that are delivered to target machines via exploits. Payloads, when delivered by a successful exploit, provide post-exploitation capabilities to the tester; one example is a terminal session, such as a reverse shell, which returns a command shell from the target machine to the tester [38]. Session handlers are used to interact with sessions returned from a successful exploit and payload delivery. Interactions include enabling or backgrounding multiple sessions, remote code execution, and the use of features such as routing, hash dumping, and privilege escalation.

2.1.2 Nmap

Nmap or "Network Mapper" [16] is an open source tool used for network exploration and security auditing. Using IP packets, Nmap is able to determine the available hosts on a network, their services, their operating systems, and if some type of firewall or

packet filtering is in place. Nmap is widely acclaimed and one of the most popular tools in the penetration testing community. Its flexibility, power, compatibility, and developer and user support makes it a useful tool for novel developers as well as system and network administrators for monitoring hosts and conducting network inventory.

2.1.3 Nessus

Tenable Network Security's Nessus[17] is a free, multi-platform, vulnerability assessment package that runs automated tests on a target network using various protocol scanners (ICMP, TCP, and UDP) and specific service testing. Nessus also features a wide range of reporting options to convey the resulting information. Nessus is widely used by penetration testing providers to perform a large portion of the network scanning work. It is also worth noting that in 2011, a plugin that integrated Nessus into the Metasploit framework was released, combining two of the most popular penetration testing options into one essential tool [3].

2.1.4 Burp Suite

Burp Suite [32] is a Java-based platform that tests the security of web applications. Burp contains its own internal mapping and exploitation tools that can be combined with manual techniques into an automated, effective testing process. Some of its other features include an application spider crawler, an internal web proxy for traffic inspection, a sequencer that specifically tests session tokens, and support for custom plugins.

2.1.5 Wireshark

Wireshark [15] is a real-time, network packet analyzer and one of the most popular software applications amongst network engineers for its network examination and

troubleshooting capabilities. Wireshark's primary function is to parse and display data from captured network packets. To support this process, Wireshark comes loaded with features such as packet filtering, hex conversion, and statistical analysis.

2.1.6 BeEF

BeEF or "Browser Exploitation Framework" [12] is a free-to-download, open-source penetration testing tool focused on exploiting holes in web browsers. Through the use of a custom API, BeEF provides a penetration tester with the ability to launch client-side attacks directly at web browsers and the ability to launch system-targeted attacks from the system's web browser.

Chapter 3

IMPLEMENTATION

In this chapter, we describe our design's architecture and functionality.

3.1 Components

In this section, we describe the underlying components used to build our design.

3.1.1 Metasploit Framework

As mentioned in Section 2.1.1, Metasploit is a fully-featured, open-source tool designed to help automate the penetration testing process [14], making it a natural choice for our design's foundation. The main reasons for choosing Metasploit over other alternatives include its consistent maintainence and development community. Metasploit is also constantly updated, ensuring that the the most up-to-date exploits and payloads are available for use, and its popularity has led to the development of plugins, interfaces, and programming languages all with goal of extending Metasploit's functionality.

The elements of Metasploit that we leverage are its exploit database, its ability to launch exploits, the Meterpreter payload, and its use of isolated working environments. Metasploit's exploit database is a repository within the framework that stores all of the attacks that can be launched by Metasploit. All exploits are considered either active or passive [26]. Active exploits attempt to exploit a host and either run until completion or are manually terminated. Passive exploits setup listeners that are designed to accept incoming connections from a malicious website, executable, etc. that an attacker has provided or distributed. Each exploit is also given a rank value

indicating its likelihood of crashing its target. More details on our use of this rank value can be found in Section 3.2.3.1. Once an exploit has been selected, Metasploit allows the user to be bring an attack into the foreground, and then customize, launch, and handle the attack's results with few commands in a very straightforward process. Metasploit's Meterpreter shell [18] is a specialy designed injection payload that deliveres a DLL to the target machine and loads an API to facilitate post-exploitation features, such as a spawning command shell and key logging. Meterpreter was the payload of choice for all compatiable exploits because of the features of its API. In cases where Meterpreter wasn't compatible, other shell payloads were used and then transformed into Meterpreter shells; more details can be found in Section 3.2.3.2. By default, the Metasploit framework includes the ability to use isolated working environments, or workspaces [25]. Each workspace is an isolated database that stores host information and scan data. Workspaces allow data from multiple tests and scans to be saved simultaneously without overwritting or contaminating one another.

3.1.2 Armitage

Armitage [19] is a Metasploit-specific collaboration tool. Armitage was developed to provide Metasploit with a graphical visual interface that can be accessed by multiple parties to encourage collaboration within a penetration testing team. Through this interface, host machines and their vulnerabilities are visually presented to the users. Armitage also presents other visual components that deal with session handling, payload selection, and post-exploitation features.

The main feature of Armitage that we leverage is its team server. The team server is initialized on the same machine as the local Metasploit instance and allows multiple users to remotely connect to a single instance of Metasploit. Starting the team server requires the IP address that will be used to accept incoming local and remote

connections and a password for authentication. The team server communicates over SSL and, as a secondary security precaution, the team server generates a SHA-1 hash from its SSL certificate and presents this hash to verify the legitimacy of the server.

Armitage, specifically the Armitage team server, was included in ANEX because it is the utility that allows Cortana scripts, described below, to connect to an instance of Metasploit and utilize Metasploit's resources.

3.1.3 Cortana

Cortana [21] is a scripting language specifically designed for Metasploit that helps automate the penetration testing processs. Cortana's main features include the ability to create persistent bots that resemble penetration testing team members, automating the Metasploit framework, and extending the Armitage GUI. Cortana is built on top of Sleep, a Perl-inspired, Java-based scripting language [20].

Cortana scripts can be run from the Armitage GUI or as standalone scripts from within a console program. We used standalone Cortana scripts to automate the Metasploit framework. In order to use standalone Cortana scripts to automate Metasploit, the Armitage team server needs to be running so that scripts can connect to it, as shown in Figure 3.1. The Armitage team server, as previously explained, allows multiple users to interact with a single instance of Metasploit. Standalone Cortana scripts behave as individual entities, or bots, that when connected to the Armitage team server, resemble fellow penetration testing team members. In order to properly connect to the server, standalone Cortana scripts need to provide proper connection and authentication information through a properties (.prop) file. This file is specified when using the command line to launch a Cortana script. A properties file contains the following information: host IP address, the port number that the team server is listening on, a user name, and the password to connect to the team server.

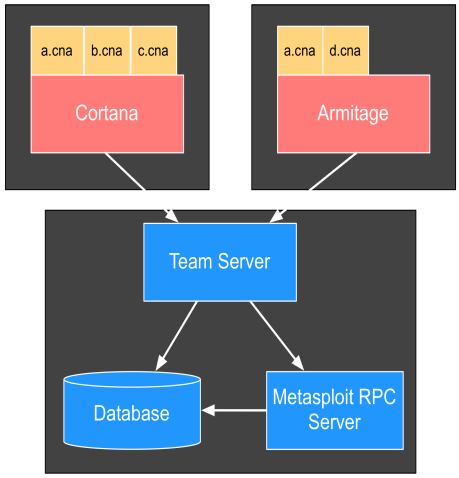

Figure 3.1: Relationship between Metasploit, Armitage, and Cortana, from [19]

All of the interaction between ANEX and Metasploit is done through Cortana scripts. A high level interaction diagram can be found in Figure 3.2. The exact nature of these interactions are explained in more detail in Section 3.2.

3.1.4 Nmap

As mentioned in Section 2.1.2, Nmap is a tool that is used to scan machines and networks to discover characteristics such as operating systems and installed services. We decided to use Metasploit's custom db_nmap command instead of the traditional version. Db_nmap [24], runs the same Nmap executable that can be downloaded online and then automatically stores the results into the current Metasploit workspace. This data is then retrieved and integrated into ANEX's exploit launching process, which is described in Section 3.2.3.1. We chose Nmap as our scanning utility because of its ease of use and its compatability with Metasploit and Armitage, both of which support db_nmap.

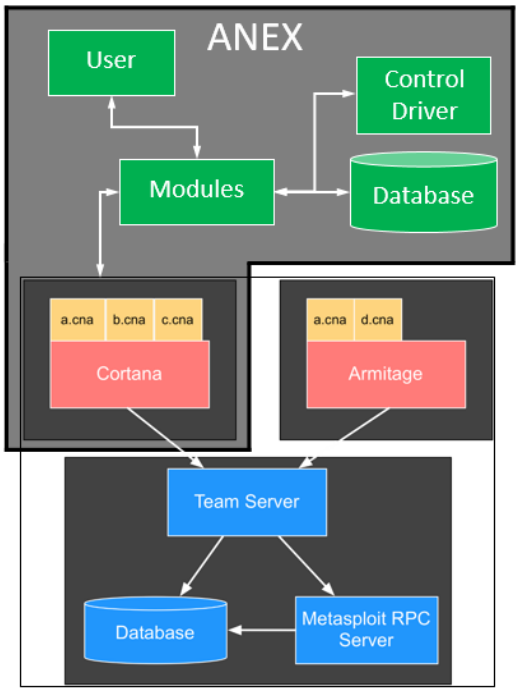

Figure 3.2: Relationship between ANEX, Metasploit, Armitage, and Cortana

3.2 Architecture

Automated Network Exploitation through Penetration Testing (ANEX) is a combination of different modules. We define three types of modules: input, data-generation, and task. Input modules prompt the user for information. Data-generation modules gather data from the local Metasploit database and store it in an easy-to-manage data strucuture. Task modules handle the more complex functions and interact with the target machine or network. We also define four module groups: initialization, recon, exploitation, and reporting. Each of these module groups represent a phase in ANEX's overall execution, the interaction of which can be found in Figure 3.3. The initialization module gathers information from the user and connects to the Armitage team server. The recon module is responsible for gathering information about the system being tested. The exploitation module handles the exploit generation,

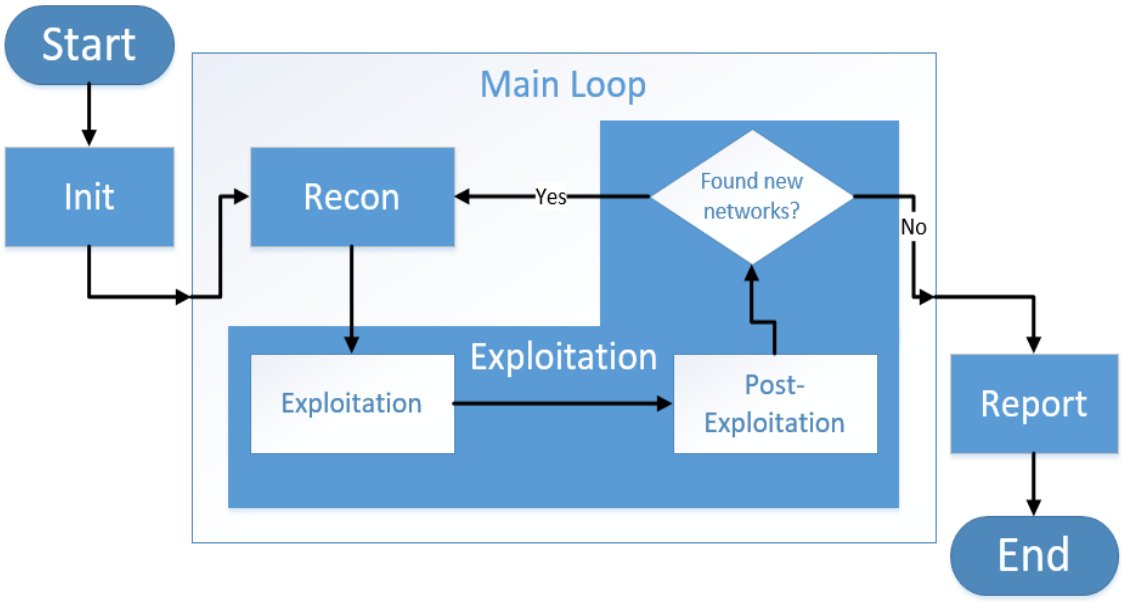

Figure 3.3: ANEX's Flow Diagram

launching, and post-exploitation procedures. The report module compiles all of the information found during testing into a user report.

All of our individual modules are connected together through ANEX's control driver. The control driver's purpose is to facilitate operation by invoking the modules and managing data. This involves collecting data from the input and data-generation modules, distributing it to the task modules, and managing any data returned by the modules.

3.2.1 Initialization

In this section, we describe ANEX's interaction with the user and the retrieval of key information that is used throughout execution. A low level diagram of the initialization group can be found in Figure 3.4.

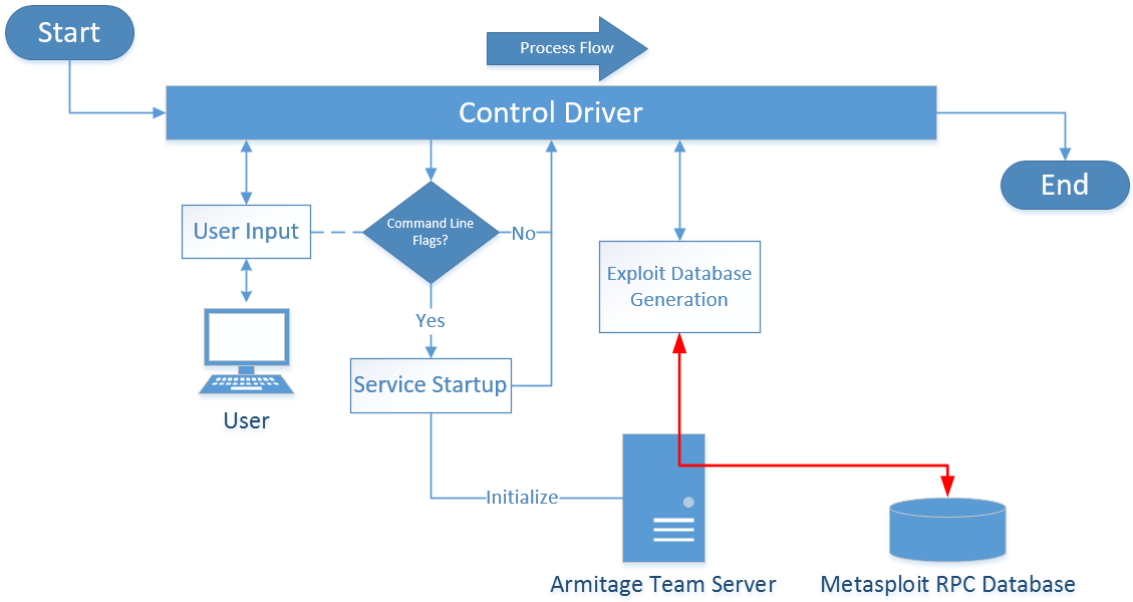

Figure 3.4: Diagram for the Initialization Group

3.2.1.1 User Input

Our user input module prompts the user for information needed at the start of each run. The main job of this module is to collect and pass the user input back to the control driver so that it can be used by other modules. User input includes credentials for initializing or connecting to an Armitage teamserver, IP addresses or ranges specifiying the machines to be tested, an IP address for the target machine, and an exploit rank threshold. As mentioned in Section 3.1.2, the Armitage teamserver requires a host IP address and a password. The user then supplies the IP addresses of the machines to be tested that are reachable from the test machine and specifies the target machine via its IP address. The last piece of information that user provides is an exploit threshold rank value. This value is used to determine which exploits to include in the exploitation process and which exploits to filter out.

3.2.1.2 Service Startup

Our service startup module initializes the programs required by ANEX. These programs include PostgreSQL and the Armitage team server. PostgreSQL is the database management system used by Metasploit. The Armitage teamserver, as described in Section 3.1.2, handles all of the incoming Cortana script connections. PostgreSQL and the Armitage teamserver need to be running in order for Cortana scripts to access the Metasploit database. Therefore, both services need to either be initialized by ANEX or be already running. The service startup module also handles the execution of the Metasploit update command to download and install Metasploit updates, ensuring that our tool is using the most recent information available.

This module is only invoked if special flags are provided as command line arguments. The service initialization flag (-init) is used to indicate that the programs required by ANEX need to be initialized by the service startup module. The Metasploit update flag (-update) is used to run the Metasploit update command and download the most recent version of Metasploit. If no flags are provided, it is assumed that PostgreSQL and an Armitage teamserver are already up and running.

As mentioned in Section 3.1.3, Cortana scripts require a properties file in order to connect to an Armitage team server. When a team server instance is initialized by this module, a properties file with the information used to start the server is created in the working directory. In the case where the team server is not initialized by this module, the property file name is set to the same name as the file that is generated when the teamserver is initialized by our module. In this case, it is the user's responsibility to ensure that information in the property file is accurate before running ANEX.

3.2.1.3 Exploit Database Generation

Our exploit database generation module handles the retrival of exploits from the local Metasploit instance and then formatting them into an easy-to-reference data structure. Each exploit contains the following information:

- A name, represented by a Metasploit directory path (os/service/exploit name)
- The operating system the exploit is designed for
- The port number that the exploit uses
- The date the exploit was added
- An integer value that indicates the exploit's ranking [5]
- A float value equal to the exploit's impact ranking that will be used for rank updating (Section 3.2.4)
- A description of what the exploit does

Our module invokes a Cortana script that runs a Metasploit command that prints the entire Metasploit exploit database to the console and then transforms each exploit into a CSV formatted string to return to the module. The module parses each returned exploit string and stores it in a dictionary for easy lookup and distribution to other modules. Whenever execution completes, the exploit dictionary is written to a file in the working directory. We use this file to generate the exploit dictionary on any subsequent runs of ANEX to avoid the slower process of polling Metasploit's exploit database. This module also updates exploit rankings after the main loop completes according to data found during execution and incorporates these changes prior to writing the file to disk. Generating the exploit database from this file allows ANEX to utilize any alterations made to the exploit rankings by the previous run.

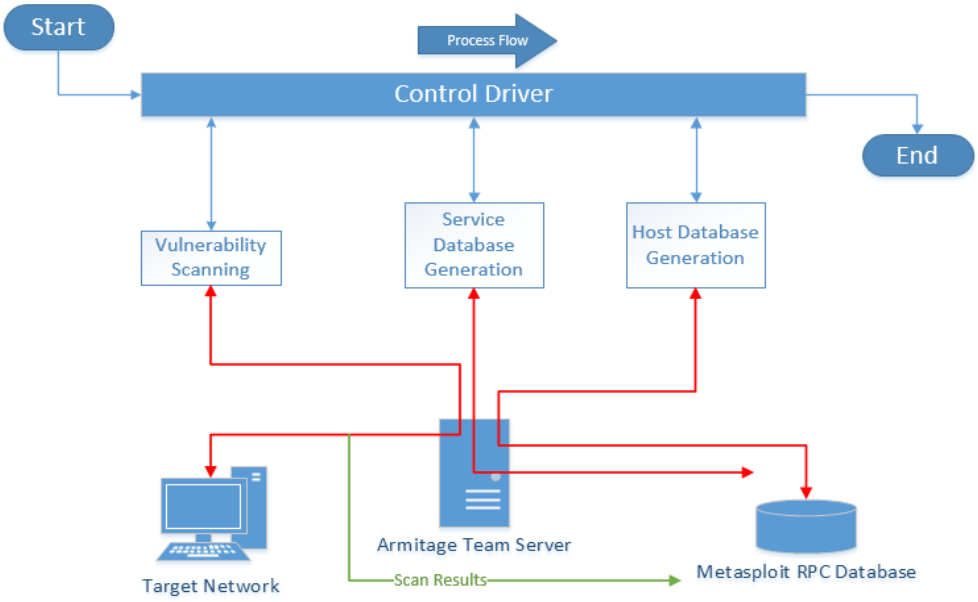

Figure 3.5: Diagram for the Recon Group

3.2.2 Recon

In this section, we describe the network information gathering and consolidation processes. A low level diagram of the recon group can be found in Figure 3.5.

3.2.2.1 Vulnerability Scanning

Our scanning module controls the use of Metasploit workspaces and executes an Nmap scan on the IP addresses initially provided by the user and on the networks found in subsequent loop iterations. Metasploit workspaces are used to retain multiple sets of host and service information within Metasploit without the need to overwrite or re-enter previous data. With this functionality, multiple system configurations can be tested, saved, and re-tested very easily. As previously mentioned in Section 3.1.4, we use Metasploit's db_nmap command to run Nmap and store the scan results into our selected workspace for easy retrieval. The stored information includes details about the machines that were scanned, the open ports that were found, and the services

running on those ports. Our module also uses auxiliary scanners that are integrated into Metasploit [24]. These scanners include a TCP port-scanning module similar to db_nmap and scanners designed to determine the version of a given service. Our design uses the TCP port-scanning module instead of the db_nmap command when attempting to discover information about networks found over the course of ANEX's execution. We then use various version scanners to gather additional information about services found on the discovered machines.

3.2.2.2 Service Database Generation

Our service database generation module retrieves the service information that was stored in the current workspace by our scanners. These results consist of port numbers for each host that was scanned and include details such as whether or not the port is open, what protocol the port is associated with, and a description of the service. This module invokes a Cortana script to run a Metasploit command that prints all of the services in the current workspace to the console. Each service is read from the console and transformed into a CSV formatted string, containing the details listed above, to return to the module. The module parses each returned service string and stores it in a dictionary for easy lookup and distribution to other modules.

3.2.2.3 Host Database Generation

The host database generation module retrieves the host information that was previously stored in the current workspace by our scanners. These results are in the form of IP addresses for each machine that was scanned and includes details such as MAC address, operating system, and operating system version. This module invokes a Cortana script to run a Metasploit command that prints all hosts to the console and then transforms each service into a CSV formatted string to return to the module.

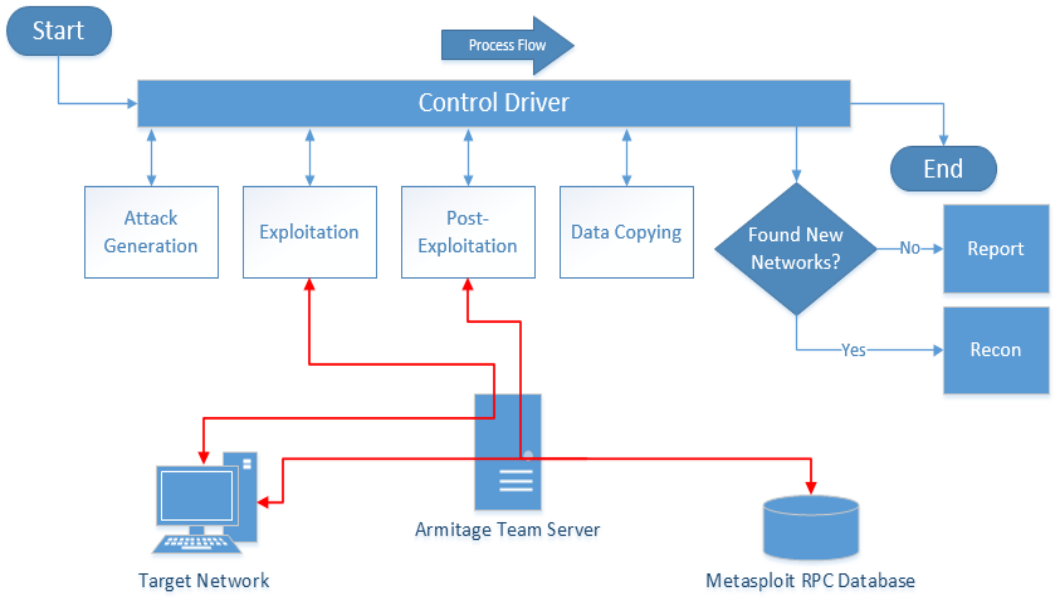

Figure 3.6: Diagram for the Exploitation Group

The module parses each returned host string and stores it in a dictionary for easy lookup and distribution to other modules.

3.2.3 Exploitation

In this section, we describe the attack generation and launching process, the post-exploitation process, and the data preservation module. A low level diagram of the exploitation group can be found in Figure 3.6.

3.2.3.1 Attack Generation and Exploitation

Our attack generation and exploitation module creates a list of exploits for each machine being tested. The exploit, service, and host dictionaries are used to determine the exploits that can be used against a particular machine. Information such as a machine's operating system and its open ports are critical in determining which exploits could potentially work. The user-provided exploit rank threshold is also used

to filter potential exploits by their rank value. Once all of the exploits have been gathered, the lists are passed to a Cortana script along with the user-provided exploit rank threshold.

Before describing the specifics of our Cortana script for this module, it is important to explain the design decisions we made with respect to the order in which we launched exploits. We decided to modify the exploit rankings after each test in order to influence subsequent tests, but we didn't want the exploits for a machine to run in the same order for every test. Therefore, we designed a weighted random sampling system to utilize the exploit rankings and the user-provided exploit ranking threshold to objectively determine the launching order. Using the threshold, we create a number of bins equal to the highest predetermined exploit rank minus the threshold value. Each bin is then populated with exploits corresponding to their exploit rank. For example, with a highest exploit rank of six and an exploit threshold of four, three bins, the difference plus one, are created and then populated with exploits whose rankings are greater than or equal to six, less than six and greater than or equal to five, and less than five and greater than or equal to four respectively. Each bin is then assigned a weight, where the weights are influenced by the number of bins and higher weights are assigned to bins with higher-ranked exploits.

After receiving the lists of exploits and the ranking threshold, our Cortana script creates a dictionary entry for each host with their own set of bins and exploits as previously described. Then, for each host, we use a random number generator to select a bin and then a function to pick and launch a random exploit from the selected bin. Once an exploit is launched, we start a timer that will stop an unsuccessful exploit after a predetermined duration. It is also important to mention that our exploitation module was designed to only use active exploits, mentioned in 3.1.1. Our use of a timeout with no secondary interaction with the host machines reflects this design decision. The exploit selection and launching process continues until

either an exploit is successful or there are no more exploits for a given machine. After successfully exploiting or launching all possible exploits for a given machine, we return the results to our module. Each result contains the following information: if the machine has been exploited, the number of launched exploits, and every launched exploit with a value indicating success of failure. If a machine is successfully exploited, the Metasploit session identification number, the user privileges of the session, the exploit that spawned the session, and the session type, either shell or Meterpreter, are also returned. The module parses these messages and then passes the relevant information to the post-exploitation and exploit database generation modules. The post-exploitation module uses the open session numbers and their associated details for further exploration. The exploit database generation module uses the session information and the lists of the launched exploits to update exploit rankings.

3.2.3.2 Post-Exploitation

Our post-exploitation module uses the spawned session information to interact with the compromised machines. A compromised machine on a network opens the door to pivoting, using a compromised host as a bridge to reach other networks or systems that aren't directly accessible to the attacking machine [11], and privilege escalation, the process of gaining root or administrator privileges if not granted already through exploitation [29].

The module starts by parsing the session information returned by the attack module into a dictionary. Next, the sessions are analyzed to determine if they have root privileges and if they were established using the Meterpreter payload. If a session doesn't have root privileges, it is passed to a Cortana script that attempts to elevate its privileges through Metasploit's post-exploit features. If a session was established with a payload other than Meterpreter, it is passed to a Cortana script that uses a

Metasploit function that transforms shell sessions into Meterpreter sessions. Meterpreter sessions are desired because the built-in API contains features that are key to the network discovery and pivoting processes of our design.

After the sessions have been properly altered, the session dictionary is passed to the Cortana script that discovers new networks. The script interacts with each session and invokes a Meterpreter API call that returns all of the network information on the target machine. This information is parsed for networks that are not associated with localhost or the known IP address of the compromised machine. Just prior to sending the data back to the module, the script invokes a Metasploit command that is used to establish pivots based on the networks that were found through each session. This allows the testing machine to connect to other networks that weren't initially reachable by tunneling through the compromised machine. The returned data is then passed from this module to the scanning module to begin a new iteration of the main loop. If no new networks are found, it is assumed that the traversal of the network has been saturated and the work done by the main loop is complete.

3.2.3.3 Data Copying

Our data copying module collects and maintains gathered data over the course of ANEX's runtime. The program's main loop repeats the process of vulnerability scanning, finding services and hosts, attacking, transforming sessions, and then searching for new networks. At the end of each iteration, the copy module is invoked to take the gathered data and append it to previously collected data. Then, when the main loop is complete, all of the accumulated information is used to update exploit rankings and generate the final report.

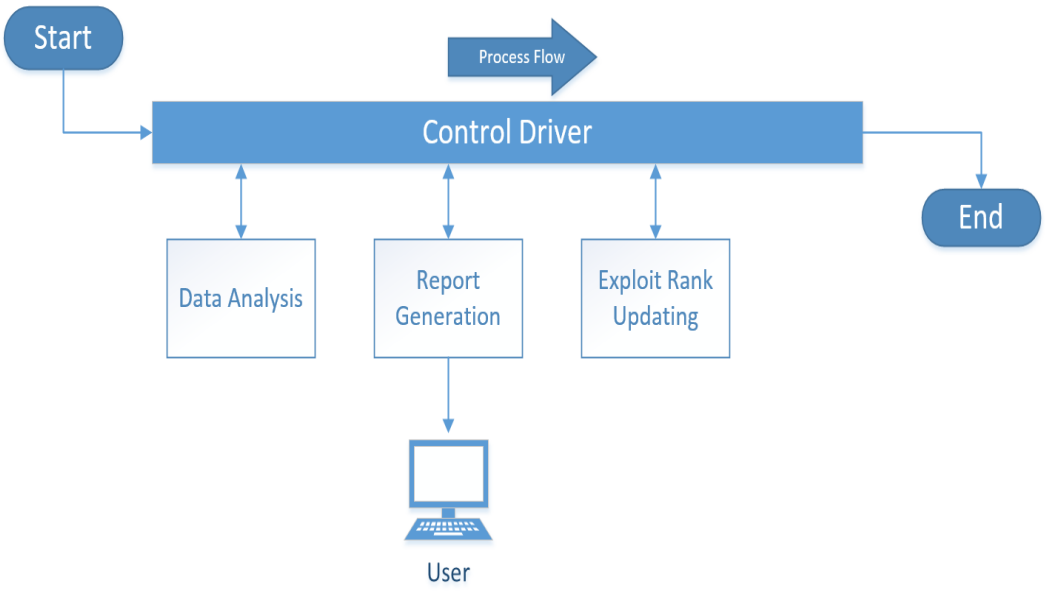

Figure 3.7: Diagram for the Reporting Group

3.2.4 Reporting

In this section, we describe execution analysis, exploit rank updating, and report generation. A low level diagram of the reporting group can be found in Figure 3.7.

3.2.4.1 Targeting

Our targeting module identifies the supplied target machine, if found, within the machines that were exploited and then determines the relationships between the target and the other machines. This is represented by a tree-like structure, with the target as the root, the machines on its same subnet or the machines that share a network with it as its children, and so forth until every found machine is accounted for or if there are no more direct or indirect paths for a machine to the target. The paths in this tree indicate exploitation chains that would need to occur from a machine to the target in order to compromise it.

3.2.4.2 Exploit Ranking Modification

The exploit ranking modification process is used to the update the rankings of the exploits that were used by the attack generation and exploitation module. As previously mentioned, the exploits that are launched at each machine are returned by the attacking module, each with a value indicating if the exploit succeeded or not. This information is then used to modify values in the exploit database that we created. Each exploit that is launched has its rank altered by a weighted value. The weight of the value depends on a few factors: if the exploit succeeded or not, the operating system specified by the exploit, the operating system of the machine being exploited, and the prevalence of the machine's operating system according to market share data. Most Metasploit exploits are designed for a single operating system, but there are some that are classified as multiple operating system exploits and therefore, are launched against a wider range of operating systems. The version of the operating system also matters given that the distribution of different operating systems versions is not equal. For example, as indicated by recent market share data [35], Windows 7 is used more than Windows XP. Therefore, if a single exploit is used to exploit both computers, the weighted increase in rank from the Windows 7 session would be greater then the weighted increase from the Windows XP session.

3.2.4.3 Reporting

Our reporting module compiles all of the provided and gathered information into an overall synopsis of what our tool accomplished. The output report contains the following details:

- The initial data provided by the user (target IP, networks, exploit severity rating)

- The machines that were discovered from the provided data
- All of the machines that were compromised
- The services that were found on each machine
- The number of exploits that were found during the attack generation process
- The number of exploits that were launched during the exploitation process
- The exploits that were used to compromise each machine along with descriptions and their Metasploit vulnerability rankings
- The networks that were discovered during the post-exploitation phase and the machines they were found on
- The machines whose sessions required altering and the sessions that couldn't be altered
- All of the machines that weren't compromised
- If the target was found over the course of ANEX's execution
- The paths to the target arranged in a tree-like hierarchy

The report conveys several things. First, the report identifies if the target machine can be compromised and the relationships of each compromised computer to the target machine. Second, the user is informed of the machines that need to be patched, the specific exploits that need to be addressed, and all of the open services that were found. Finally, the compromised machines and the networks that were found on them are displayed in a network map-like structure, allowing the user to see all of the expected or unknown connections.

Chapter 4

VALIDATION

In this chapter, we explain our validation process. We define our experimental procedure, testing environment, expected and actual results, and the key characteristics that we measured.

4.1 Experimental Procedure

In this section, we describe our testing process for validating the functionality of ANEX, Automated Network Exploitation through Penetration Testing. We built our testing process around ANEXs three main features, exploit launching, the identification and exploration of found networks, and target mapping.

4.1.1 Exploit Launching

ANEXs exploitation process is based entirely on the results from the port scanning module. The results from the port scanning module are in turn based entirely on the services that are running on a specific machine and the configuration of the firewall. Therefore, we purposefully varied the services that were running and the configuration of the firewalls to observe how ANEX responded to the changes and the effect on the overall findings.

4.1.2 Handling Found Networks

ANEX is designed to look for unexplored networks on every compromised system. This feature allows it to find unspecified networks where it can then look for more machines to potentially compromise. Some key behaviors we looked at were the ability

to handle finding multiple new networks on the same machine and the time required to scan these networks.

4.1.3 Target Mapping

We used our different network configurations to determine the robustness and accuracy of the targeting system. We used three main configuration templates, all designed to focus on a specific element of the targeting process. The first configuration was our base case where every machine was connected to a single network. This configuration was used to test if the targeting system was accurate in the simplest case. The second configuration contained two subnets with varying numbers of machines and a target located in one of the subnets. The third configuration contains two subnets similar to the previous configuration, but with the target and a machine in one of the subnets connected via a local area network. This configuration was used to test the discovery and exploration of networks found during the post-exploitation procedure. The three configuration templates can be found below in Figures 4.1 - 4.3.

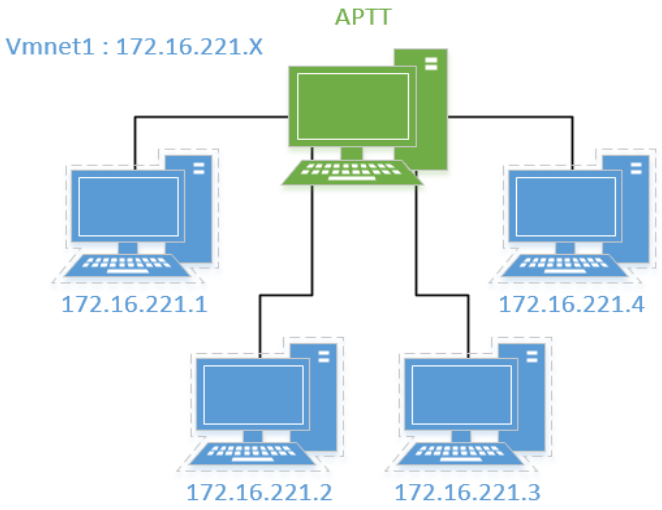

Figure 4.1: Single Network Template

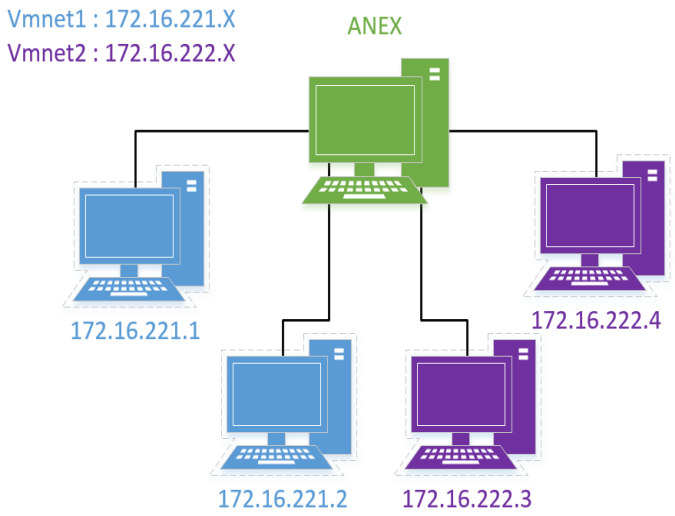

Figure 4.2: Two Subnets Template

4.2 Testing Goals and Parameters

In this section, we describe our testing goals and parameters. Our goals for testing were to show a high exploitation success rate, ANEX's ability to identify and exploit machines that weren't initially accessible to the testing machine, and the ability to take advantage of multiple paths to a single machine in order to exploit it.

We classify our key testing parameters into three categories: static, dynamic, and dependent variables. Our sole static variable, data that remains constant throughout testing, was the Metasploit framework version. We ensure that all of the tests are conducted with the same database of exploits and the rankings determined during testing. Updating the Metasploit framework during testing would potentially alter our results due to the addition or removal of exploits and modifications to the exploit base rankings. Therefore, just prior to the beginning of the experimental trials, we updated the framework and then maintained the same version until testing was complete.

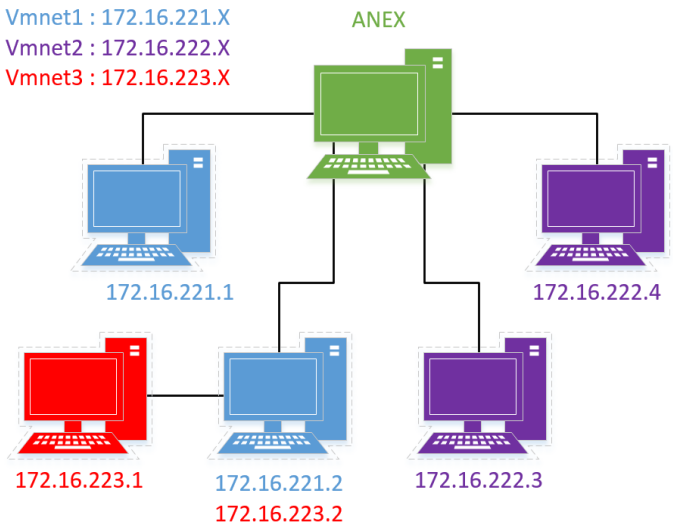

Figure 4.3: Secondary Network Template

Our dynamic variables, data that changes throughout testing, were our test network configurations and the exploit rankings. With ten different virtual machines available, the number of possible test configurations exceeds three million. Therefore, we selected the configurations before any tests are run since testing all possible configurations was not feasible. In addition to varying the network topology, we also varied the number of running services and firewall settings of each machine. Changing these two properties directly affects the results of the port scanning module which directly affects our exploitation process. As mentioned in 3.2.4.2, an exploit's rank is updated depending on if if succeeded or failed. The new rankings affect how the exploits are sorted by our weighted random sampling scheme which affects the likelihood of when or if an exploit is launched during the exploitation phase.

Our dependent variables, data that is a function of the dynamic variables, were exploit success and runtime. We knew initially that our ability to exploit machines was based on the exploit rank threshold and the individual configurations of our machines, but we wanted to test how much impact certain configuration changes would have. Therefore, we measured our exploit success rate as we varied the configurations

of each machine. As for runtime, one of the benefits of automated penetration testing solutions over manual testing is time investment. Automated solutions are typically faster and can be used more frequently. Therefore, it was important that our design ran in a reasonable amount of time, depending on the system being tested. Depending on the configuration, this time varied between a few minutes to a few hours. At the time of testing, we were aware that our scanning and exploitation modules were our main time contributors because of the critical data they provide. Therefore, we watched our runtime as we varied the configurations of the test networks.

4.3 Development and Testing Components

In this section, we describe our testing environment, the virtual machines we used, and the configurations of each virtual machine.

4.3.1 Kali Linux 2016.1

Kali Linux is an open-source, Debian-based Linux distribution, developed by Offensive Security, that is specifically designed to assist penetration testing and security auditing procedures [27]. Designed for penetration testing professionals, Kali contains hundreds of tools that support vulnerability scanning, password cracking, and many other penetration testing and security-related tasks. Kali Linux was specifically chosen as the development and testing platform for our design because the key tools that make up ANEX, Metasploit, Armitage, Cortana, and Nmap, come pre-installed.

4.3.2 VMware Workstation 12.0

VMware Workstation [39] is a popular virtual machine management program that allows multiple machines to be created, managed, and run simultaneously. Virtual

machines with various operating systems were spawned and arranged in various configurations in order to test ANEXs functionality on makeshift corporate network setups. VMware Workstation was chosen to test ANEX because of its easy availability and its ability to easily create virtual networks and LANs.

4.3.3 Virtual Machines

We chose ten different operating systems for our testing machines, listed below:

- Windows XP Bare
- Windows XP SP1
- Windows XP SP2
- Windows XP SP3
- CentOS 6.7 Server
- Ubuntu 12.04 Server
- Ubuntu 14.04 Desktop
- Ubuntu 15.04 Desktop
- Windows 7
- Windows Server 2012 R2

Our choices were based on the types of machines in a corporate environment. Windows 7 was a clear choice for a Windows workstation. The various Windows XP versions were chosen because they are still sparingly used [35]. The three servers, Windows 2012, Ubuntu 12.04, and CentOS 6.7, were chosen for operating system variety and to capture the use of older server versions. Ubuntu 14.04 and 15.04 were

chosen to represent older, stable Linux workstations. The most recent market share data from [35] indicates that our selected virtual machines account for around 60% of the total market.

4.3.4 Installed Services

In order to make the virtual machines resemble systems that would be found in a typical corporate environment, various services were installed on both the workstations and servers. All of the services we installed on our virtual machines can be found in Table 4.1 [1]. All of the services for each machine, including used default, can be found in Figure 4.4.

CentOS 6.7 Server	MySQL, Apache, Samba, Bind9, SSH, Postfix, Dovecot, Proftpd, Squid
Ubuntu 12.04 Server	MySQL, Apache, SSH, Dovecot, LDAP, NFS, LTSP, Vsftpd
Ubuntu 14.04	MySQL, Apache, Samba, SSH, CUPS, Java
Ubuntu 15.04	Apache, Samba, CUPS, Java
Windows 7	Apache, Java, Windows RPC, Netbios-ssn, Microsoft-ds
Windows XP (Default, SP1, SP2, Sp3)	Windows RPC, Netbios-ssn, Microsoft-ds
Windows Server 2012 R2	MySQL, Apache, Windows Services (DHCP, DNS, IIS)

Figure 4.4: All Services

[1] I'd like to personally thank Professor John Bellardo and Raz Chowdhury for their suggestions regarding typical services installed on corporate workstations and servers

Table 4.1: Installed Services

Service	Description	Installed On...
MySQL	Open Source Database Management System	Centos 6.7, Ubuntu 12.04, Ubuntu 14.04, Windows Server 2012 R2
Apache	Open Source HTTP Server	Windows 7, Ubuntu 12.04, Ubuntu 14.04, Ubuntu 15.04, Centos 6.7, Windows Server 2012 R2
Samba	File and Print Service	Centos 6.7, Ubuntu 14.04, Ubuntu 15.04
Bind9	Open Source DNS Utility	Centos 6.7
Squid	Web Proxy Server	Centos 6.7
SSH	Remote Login Utility	Ubuntu 12.04, Ubuntu 14.04, Centos 6.7
Postfix	Open Source Mail Server	Centos 6.7
Dovecot	Open Source Email Server	Centos 6.7, Ubuntu 12.04
CUPS	Print Server	Ubuntu 14.04, Ubuntu 15.04
Java	Standard Desktop Package	Ubuntu 14.04, Ubuntu 15.04, Windows 7
FileZilla	Open Source FTP Server	Windows Server 2012 R2
LDAP	File Server	Ubuntu 12.04
NFS	File Server	Ubuntu 12.04
LTSP	Open Source Terminal Server	Ubuntu 12.04
Proftpd	FTP Server	Centos 6.7
Vsftpd	FTP Server	Ubuntu 12.04
DHCP	Windows DHCP Server	Windows Server 2012 R2
DNS	Windows DNS Server	Windows Server 2012 R2
IIS	Windows Web Server	Windows Server 2012 R2

4.4 Expected Results

For each test, we ran ANEX against a different network configuration built from the templates in Figures 4.1 - 4.3. In each test, we varied the firewall configurations, the exploit rank threshold, and the connections of the secondary networks if appropriate. We expected various firewall configurations and exploit rank thresholds to directly influence the number of exploits run and the overall success of the exploitation process. We anticipated that altering the configuration of the secondary networks wouldn't have a large impact on the overall results. Other than slightly altering the target tree produced in the report, we expected the identification, exploration, and exploitation of machines on the network to remain unchanged.

For our runtime values, we expected to observe a linear-like relationship between the size of the tested network, including secondary networks, and the time required to scan, exploit, and search through them. This time would also be affected by the machines in the networks given that some would be more vulnerable than others and take less time to exploit.

4.5 Results

Our initial hypothesis was: *We can identify and exploit a high percentage of the machines on a given network and show paths from every compromised machine to a target machine.*

Unfortunately, we found that we were only able to exploit the Windows XP machines. The remaining Windows, CentOS, and Ubuntu machines were consistently immune to ANEXs exploitation techniques, regardless of their firewall configuration, number of running services, and the user-provided exploit rank threshold. However,

despite the fact that our exploitation success wasn't as comprehensive as we expected, we also did not want to make our test machines unnecessarily vulnerable.

4.5.1 Analysis

Overall, we observed a couple of things. First, we found no correlation between the number of exploits generated for a machine and successfully exploiting said machine. Although we generated far more exploits for the machines running a large number of services, particularly the servers, than all of the Windows XP machines, we weren't able to retain the same relationship when tallying successful exploitations. Second, we found an obvious relationship between the age and type of operating system and if we were able to successfully exploit it. The machines we were able to exploit were the oldest in terms of release date compared to the other machines and among the more widely-used and popular operating systems that we used for testing. Finally, we found that the configuration of the test network, specifically the configuration of the secondary networks, had a direct effect on the test results. Since we were not able to exploit every machine as we initially thought, if a secondary network was configured on an unexploitable machine, we were not able to find that network. The specific configurations and results for each individual test can be found in Appendix A.

With respect to our initial goals and expectations, we did not achieve the high success rate we were anticipating, but we were able to exploit machines unreachable from our test machine and make use of multiple paths to a single machine. Figure 4.5 shows one of our test networks. This network contains three subnets, two of which contain machines that ANEX cannot directly reach. However, we were able to exploit both machines on the 221 subnet, which allowed us to access the 222 subnet. We then proceeded to exploit the machine on 222 subnet, gain access to the 223 subnet, and then exploit its sole machine as well. This test confirmed ANEX's

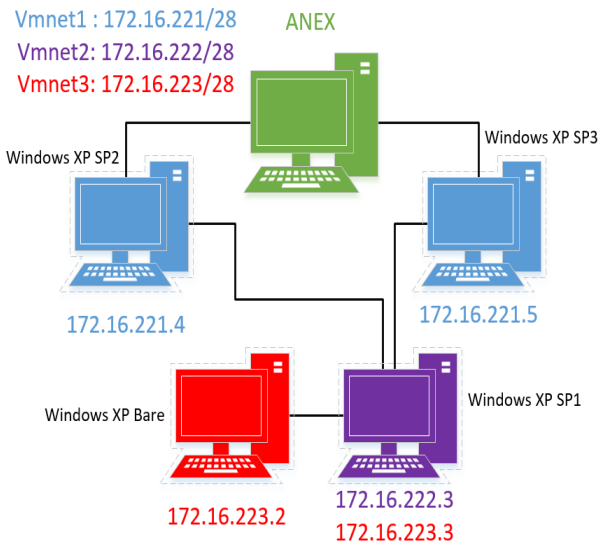

Figure 4.5: Test Network with Unreachable Machines

ability to successfully exploit machines and then use their proximity to other subnets and traverse through initially unreachable networks. Figure 4.6 shows another of our test networks. This network contains a machine whose network interface was completely closed off to incoming traffic on the 221 subnet. Therefore, ANEX was unable to scan it and subsequently exploit it. However, ANEX was able to exploit the Windows XP Bare machine, which contains a connection to the 222 subnet. After scanning this network, we were able to find and subsequently exploit the machine through a different network interface that we initially couldn't find or exploit. This test confirmed ANEX's ability to exploit machines by utilizing mulitple connections on different subnets. The specific details of these two tests can be found in Appendix A.

We also determined several reasons that address our low exploitation rate and the machines we couldn't compromise. First, the Metasploit exploit database is published information. Rapid7, Metasploit's distributor, maintains an online version of Metasploit's exploit database for easy lookup and searching. Therefore, the developers of the programs that are vulnerable to these exploits have an incentive to develop and

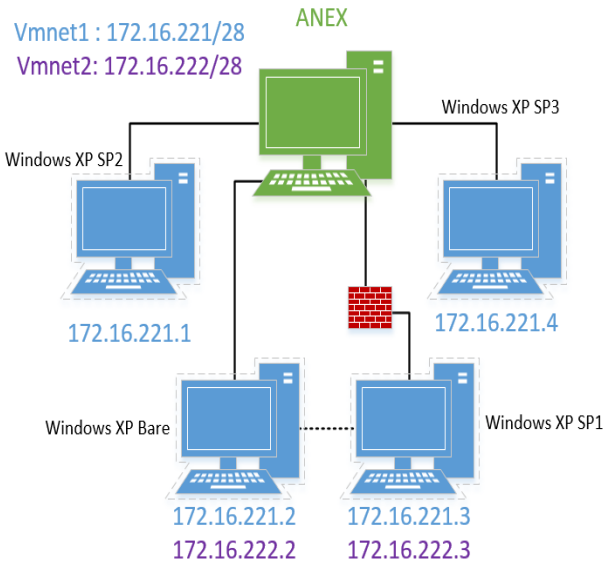

Figure 4.6: Test Network with Multiple Exploitation Paths

release patches for them. Second, ANEX's reliance on an exploit database means we can only use exploits that Metasploit's developers decided to integrate into the framework, and ones that exploit unpatched vulnerabilities. Third, older operating systems tend to be more vulnerable than newer operating systems. Older, widely-used operating systems usually have more known vulnerabilities and lack the newer versions of programs and security updates when compared to their newer counterparts, making them more susceptible to exploitation overall. Finally, the simple premise of active exploits is well known. Some of the Metasploit's active exploits only require operating system information and an IP address in order to take full control of a machine. Now while this is great if it works, the simplicity of this type of exploit makes it unreliable if it is patched. Alternatively, more intricate exploitation methods, such as targeting and tricking a user into visiting a malicious website, are more resistant to updated security measures at the cost of convincing a user to perform a desired action. This targeted method of attack is also performed from the inside-looking-out which is much less regulated, especially in terms of firewalls, than an attack from the outside-looking-in from a network perspective.

4.5.1.1 Exploitation Success

We found that the firewall settings and running services directly affected our ability to exploit machines. Stopping a service that was being exploited or reconfiguring the firewall to block the vulnerable services prevented ANEX from exploiting a machine, which is what we expected. For the machines that couldn't be exploited, any changes to their firewall settings or running services proved ineffective. The exploit rank threshold and exploit updating also affected our exploiting capabilities. We found that if we set the exploit rank threshold too high, we weren't able to exploit a machine that we previously exploited with a lower exploit rank threshold. This was due to the default and updated rankings of certain exploits. We also noticed that throughout testing, the ranks of the exploits that were consistently successful on certain machines were decreasing over time. We attributed this to our exploit rank updating scheme, where we weighted the specific versions of operating systems based on market share research. For example, a Windows exploit that could successfully exploit a Windows XP machine, but not successfully exploit a Windows 7 machine in the same run would have its overall ranking decreased. We eventually needed to lower the exploit rank threshold during testing to ensure exploitation. This was something that we had not initially anticipated and indicates the need to improve our exploitation process to achieve higher exploitation rates and avoid this problem.

4.5.1.2 Runtime

We measured runtime as a function of the test network's parameters, which includes the network topology, firewall settings, running services, and the exploit rank threshold. First, we wanted to observe if runtime scaled with the size of the network. The factors we took into account were the number of initial machines, number of secondary networks, and the number of hosts connected to the secondary networks. We

observed that the number of secondary networks and their varying number of hosts only affected the runtime if the routes to them were through exploitable machines. This is because ANEX cannot reach any networks whose only paths are through unexploitable machines. If ANEX is able to reach a secondary network, the runtime becomes dependent on the network's subnet mask, the number of machines on the network, and the individual settings of each machine. The subnet mask determines the specfic network region to scan, which directly affects the scan's duration. The number of machines on the network and their individual settings affects the total number of generated attacks.

Firewall settings, running services, and the exploit rank threshold also affect the total number of generated attacks. As mentioned previously, the firewall settings and the services running on a given machine affect the results of the scanning module, which affects the number of generated attacks. The exploit rank threshold also affects the number of generated attacks due to exploit rank updating. Overall, we found that the number of generated attacks, based on all of the variables above, is the main contributor to the total runtime because of our decision to have only one active attack at a time. We also observed that our weighted random sampling scheme altered the number of attacks launched and total exploitation time for machines that were exploitable, as expected. However, when the same number of exploits are generated for an unexploitable machine in different program runs, the weighted random sampling provides no benefit because all exploits are launched in both cases.

4.5.1.3 Exploit Rank Updating

As previously mentioned in Section 3.2.3.1, we used weighted random sampling during the exploitation phase to prevent identical exploitation results and the underutilization of potential working exploits. The overall changes to the exploit rankings

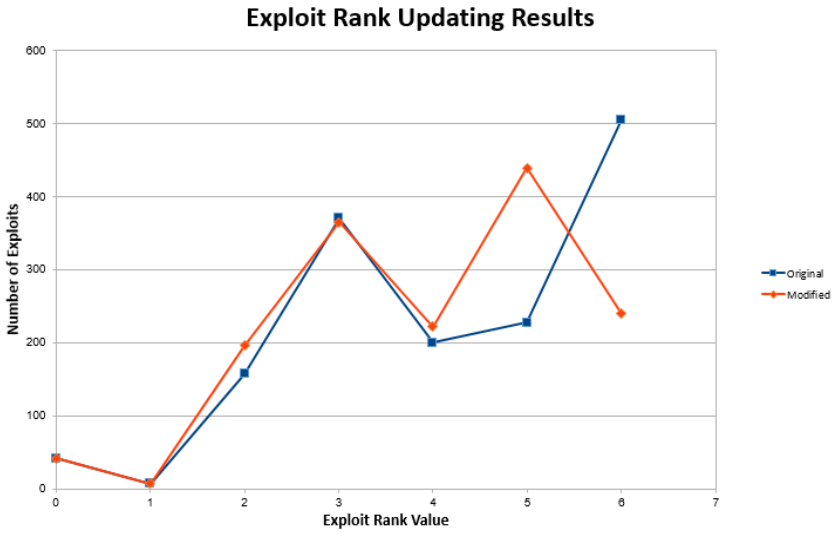

Figure 4.7: Exploit Rank Updating Results

throughout all of our testing are shown in Figure 4.7. From these results, we noticed a trend of decreasing exploit rankings, which was unsurprising given our exploitation success. However, since we were expecting a more equal distribution as a result of higher exploitation success, the overall decreasing trend indicates the need to improve the exploitation process. As of now, the general downward trend could require the use of a lower exploitation threshold in order to try previously unused exploits, which potentially increases the risk of instability due to using exploits with lower initial rankings.

Chapter 5

RELATED WORK

In this chapter, we present several automated penetration testing solutions, contextualize their advantages and disadvantages, and describe how our design was influenced by our analysis of these tools.

5.1 Fast-Track

Fast-Track [14] is a free-to-download, open-source, Python-based tool designed to enhance various penetration testing techniques. Fast-Track is built around the Metasploit framework and utilizes Metasploit's payload database and client-side attack vectors and extends Metasploit's functionality with more exploits, attack vectors, and focused tools such as MSSQL injection and server attacks. One interesting extension is Fast-Track's Autopwn, an improvement upon an older feature of Metasploit also called Autopwn. Metasploit's Autopwn automatically targets and launches all possible exploits at a given port on a specified machine. Fast-Track's Autopwn improved this process by first performing a vulnerability scan and then using its results to launch all possible exploits. However, although Fast-Track does add useful extended functionality to Metasploit, Fast-Track only automates the information gathering and exploit launching phases, leaving all of the post-exploitation techniques for the user to manually perform.

5.2 Tool developed by Haubris and Pauli

Another tool that closely resembles ANEX is the tool described in [13]. This tool is also Python-based and is designed to automate the information gathering and ex-

ploitation process. The authors of [13] utilize several open-source tools including Metasploit [14], Nmap [16], Nessus [17], OWASP's Zed Attack Proxy [28], and the-Harvester [31]. The authors start by running several different vulnerability scans. The results are then parsed and written to a file. The contents of this file are then imported into their automated exploitation process which launches exploits and returns results. The inclusion of multiple methods of gathering information gives this tool a very comprehensive exploitation process, with multiple sources of information and the ability to exploit multiple attack vectors. However, similar to Fast-Track, there is no automated post-exploitation functionality.

5.3 Core Impact Pro

Core Impact Pro [1] [38] is an automated penetration testing application developed by Core Security Technologies. Core Impact Pro is designed to consolidate the penetration testing process and necessary resources into a single framework. This was done to simplify and speed up overall execution and standardize the execution methodology for maximum efficiency and ease of repetition. Core's internal architecture includes the following features: agents, host-controlled software programs that are deployed on compromised systems; modules, an operation or group of operations to be performed by an agent, such as web server attacks or port scanning; the console, a graphical user interface used to launch and manage attacks and display results; entity database, central information repository that contains target information, activity logs, agent deployments, and more.

Core Impact Pro is one of the highest rated automated penetration testing products available. However, Core does struggle with determining if certain exploits will lead to damage or stability issues on the target. Also, its substantial price tag of

$40,000 a year and the lack of a command-line interface are both factors that could prevent its widespread adoption beyond its current market.

5.4 Immunity's Canvas

Immunity's Canvas [12] [38] is a Python-based exploitation framework that helps a user automate their exploitation and testing procedures. Similar to Metasploit, Canvas is not a fully integrated penetration testing tool, but it does provide penetration testers with features to help aid the process. Canvas has a modular architecture similar to that of Core Impact, but it does not offer as much automation and lags behind in other features. Unlike Core, Canvas does have a command-line interface and a much lower price tag. However, the same damage and stability issues when using exploits are present and the learning curve for Canvas is much higher than similar products.

5.5 Beyond Security's AVDS

Beyond Security's Automated Vulnerability Detection System (AVDS) [6] is a network vulnerability assessment application. A self-contained system, AVDS is able to automatically scan networks, identify and test vulnerabilities, and report output, giving it the appearance of an all-in-one penetration testing system similar to Core Impact. However, it is reliant on a constantly updated database of vulnerabilities, meaning that it cannot identify and test vulnerabilities that arent available to it. It also only has a web browser interface, with no command-line interface option available.

5.6 Impact on Design

We analyzed the tools mentioned above to determine the foundation of our design, the features we found desirable, and the drawbacks that we wanted to avoid.

Areas we specifically focused on were functionality, user expertise and involvement, deployability, and cost. We decided to imitate the generation of exploits from vulnerability scan results, similar to FastTrack and the tool from [13], due to their speed and simplicity. We also decided to use an exploit database to determine our potential attacks, similar to AVDS, despite the restrictions on vulnerability identification. A lot of work and research has been accomplished in this area and we determined that using a database was sufficient for our purposes. Finally, we wanted to include automated post-exploitation functionality, something that both Fast-Track and the design from [13], were lacking. However, we chose to narrow the focus of the information gathering and exploitation phases in order to focus on our post-exploitation phase, a tradeoff we deemed reasonable.

The second area we addressed was our target audience and the level of required user involvement. We identified our target audience to be IT system administrators and penetration testing engineers. We chose to target IT system adminstrators because we identified them as our primary use case, people with computer network and security knowledge with the understanding of how to properly deploy our design in a corporate environment. We also wanted to appeal to penetration testing engineers because they also use automated tools, such as Core Impact, Immunity's Canvas, etc., to aid the testing process. With these two cases in mind, we decided on the following features. First, we chose to rely on as little user involvement as possible to promote a low learning curve and the ability to run ANEX in the background. Second, we chose a command-line interface as our sole interaction medium because of its simplicity and our decision to minimize user involvement. The last two, tightly-linked areas

we focused on were deployability and cost. Our primary contribution was to show that important security information could be gathered by an automated penetration testing tool built by combining open-source and free-to-download programs. To accomplish this, we designed ANEX around software that is easily available at no cost to the user.

Chapter 6

FUTURE WORK

This chapter outlines some potential areas for further research.

6.1 Post-Exploitation

Compromising a machine through a shell session, Meterpreter session, etc. gives the tester access to information that can be used to determine the severity of exploiting a certain machine and aid in the exploitation of other machines. We only looked for other connected networks to explore from each compromised machine. However, other post-exploitation tactics such as observing the current processes, interacting with the command shell, or dumping credential hashes can be useful as well. Reporting back the processes currently running on a compromised machine can help determine if any unwanted information is being leaked simply through process names or execution times and the impact of an attacker either stopping a legitimate process or spawning a malicious one. Command line interaction has a wide range of potential uses. In addition to observing and altering processes, the installation of backdoors and altering firewall configurations are just two things that an attacker can do that should be reported by the tester. Using hashed credentials or dumping the hash and uncovering the plaintext can promote the use of exploits that utilize credentials as well as the ability to revisit uncompromised machines with information that could lead to compromising it after the initial attempt.

6.2 Information Gathering

There are several methods to determine if and where a computer is vulnerable, ranging from automated scanners to manual probing. We opted to use Nmap and Metasploit's auxiliary scanning modules for their convenience and fast results. The addition of other scanning modules, specifically Nessus, would seemingly integrate well with our designs similar module for Nmap. Metasploit does support a Nessus plugin and its output along with other information gathering techniques could improve ANEXs exploitation abilities.

6.3 Passive Exploitation and User Evaluation

As mentioned in Section 3.2.3.1, we decided to use only active exploits in our design and not passive ones for a number of reasons. However, passive exploits are still useful and can be used to test the awareness and security practices of users interacting with the system being tested. Adding functionality to spawn exploit handlers and wait for the incoming connections activated by users would be useful for at least two reasons. First, it provides the tester with a different avenue to exploit machines. Second, the tester can gauge user responses to determine how to properly inform and educate users on spotting potentially malicious actions and their impact.

6.4 Exploration of Different Attack Vectors

Tying into the section above about the use of passive exploits, expanding the exploitation capabilities of ANEX beyond services and desktop and server applications is a logical step as well. Web applications and system users are other potentially vulnerable elements that can be exploited to gain control of a system. Automated tools such as OWASP's Zed Attack Proxy (ZAP) and Burp Suite are popular for

web vulnerability analysis. Integrating another specialized tool into ANEX alongside Metasploit could prove to be a very useful utility. User-based tests, such as phishing or masquerading as a high level official, are effective, real-world tactics that can determine if the users of a system are potential weaknesses to security. This ties directly into the use of passive exploits, but also covers things such as accidently or purposefully leaking credentials.

Chapter 7

CONCLUSION

Our idea of designing an easily-deployable, no-cost, and easy-to-use automated penetration testing tool was a worthwhile one, but our tool needs to be improved to handle more diverse system configurations. Despite that, we did have several promising results. We were able to successfully combine several popular and effective free-to-use tools in the manner that we initially desired and build an automated utility that performs all of the phases of a penetration test. We were also able to exploit machines that could not be reached except through other machines as well as utilize multiple paths to exploit a machine. These two scenarios were two of our main goals when we came up with the idea for ANEX and the fact that we were able to complete these goals with ANEX is a fulfilling accomplishment.

Although ANEX did not perform at the level we had initially hoped, it can still be used in its current state to exploit the machines that it can and return vulnerability information about the machines that it cannot exploit. This information is still very useful from a security perspective despite needing to run ANEX on different machines to gather a more complete network and target map. For example, further exploitation and post-exploitation activities can easily build off of the results and network connections created by ANEX, enabling a manuel tester to quickly determine where immediate fixes need to be made and the machines that require more focused attention. In its current state, ANEX is an effective, all-in-one, automated penetration testing tool that can help evaluate the security measures of individual machines and networks, assist the manual penetration testing process, and a foundation for easily-accessible penetration testing utilities based on free tools.

Finally, as a disclaimer, it is important that we state explicitly that our design is not an all-in-one security evaluation tool and should not be used as the sole source of information to indicate the effectiveness or status of the security of a given computer system or network of systems.

BIBLIOGRAPHY

[1] I. Arce. An argument for an automated penetration testing framework. http://www.coresecurity.com/system/files/CORE_IMPACT-WhitePaper.pdf, August 2001.

[2] B. Arkin, S. Stender, and G. McGraw. Software penetration testing. *IEEE Security & Privacy*, 3(1):84–87, 2005.

[3] P. Asadoorian. Using nessus and metasploit together. http://www.tenable.com/blog/using-nessus-and-metasploit-together, August 2011.

[4] A. G. Bacudio and M. Jones. An overview of penetration testing 1. *International Journal of Network Security and Its Applications (IJNSA)*, 3(6), Nov 2011.

[5] T. Beardsley. Exploit ranking. https://github.com/rapid7/metasploit-framework/wiki/Exploit-Ranking, June 2013.

[6] Beyond Security. Avds: Automated vulnerability detection system data sheet. https://beyondsecurity.zendesk.com/hc/en-us/articles/203452449-AVDS-Data-Sheet, January 2015.

[7] M. Bishop. About penetration testing. *Security Privacy (IEEE)*, 5(6):84–87, Nov 2007.

[8] E. Chow. Ethical hacking & penetration testing. http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.475.3877&rank=1, 2011.

[9] J. Dowdy. The cybersecurity threat to u.s. growth and prosperity. In N. Burns and J. Price, editors, *Securing Cyberspace: A New Domain for National Security*. The Aspen Institute, 2012.

[10] Z. Durumeric, J. Kasten, D. Adrian, J. A. Halderman, M. Bailey, F. Li, N. Weaver, J. Amann, J. Beekman, M. Payer, et al. The matter of heartbleed. In *Proceedings of the 2014 Conference on Internet Measurement Conference*, pages 475–488. ACM, 2014.

[11] G. Fraser. White paper: Tunneling, pivoting, and web application penetration testing. `https://www.sans.org/reading-room/whitepapers/testing/tunneling-pivoting-web-application-penetration-testing-36117`, July 2015.

[12] M. Gregg. *The Network Security Lab: A Step-by-Step Guide*. Wiley, Hoboken, NJ, USA, 1st edition, 2015.

[13] K. P. Haubris and J. J. Pauli. Improving the efficiency and effectiveness of penetration test automation. In *Information Technology: New Generations (ITNG), 2013 Tenth International Conference on*, pages 387–391. IEEE, 2013.

[14] D. Kennedy, J. O'Gorman, D. Kearns, and M. Aharoni. *Metasploit: The Penetration Tester's Guide*. No Starch Press, San Francisco, CA, USA, 1st edition, 2011.

[15] U. Lamping, R. Sharpe, and E. Warnicke. Wireshark user's guide: For wireshark 2.1. `https://www.wireshark.org/download/docs/user-guide-us.pdf`, 2014.

[16] G. F. Lyon. *Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning*. Insecure, USA, 2009.

[17] C. McNab. *Network Security Assessment*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2004.

[18] M. Miller. White paper: Metasploit's meterpreter. https://dev.metasploit.com/documents/meterpreter.pdf, 2014.

[19] R. Mudge. Armitage manual. http://www.fastandeasyhacking.com/manual.

[20] R. Mudge. Sleep 2.1 manual. http://sleep.dashnine.org/download/sleep21manual.pdf, June 2008.

[21] R. Mudge. Cortana tutorial. http://www.fastandeasyhacking.com/download/cortana/cortana_tutorial.pdf, May 2013.

[22] S. Northcutt, J. Shenk, D. Shackleford, T. Rosenburg, R. Siles, and S. Mancini. White paper: Penetration testing: Assessing your overall security before attackers do. https://www.sans.org/reading-room/whitepapers/analyst/penetration-testing-assessing-security-attackers-34635, June 2006.

[23] O. of Personnel Management. Opm announces steps to protect federal workers and others from cyber threats. https://www.opm.gov/news/releases/2015/07/opm-announces-steps-to-protect-federal-workers-and-others-from-cyber-threats/, July 2015.

[24] Offensive Security. Information gathering with metasploit: Port scanning. https://www.offensive-security.com/metasploit-unleashed/port-scanning/.

[25] Offensive Security. Metasploit fundamentals: Using the database in metasploit. https://www.offensive-security.com/metasploit-unleashed/using-databases/.

[26] Offensive Security. Metasploit fundamentals: Working with active and passive exploits in metasploit. https://www.offensive-security.com/metasploit-unleashed/exploits/.

[27] Offensive Security. What is kali linux? http://docs.kali.org/introduction/what-is-kali-linux, 2016.

[28] OWASP. Owasp zed attack proxy project. https://www.owasp.org/index.php/ZAP, April 2016.

[29] Penetration Test Guidance Special Interest Group. Penetration testing guidance. Payment Card Industry Security Standards Council, March 2015.

[30] Ponemon Institue LLC. 2015 cost of data breach study: Global analysis. http://www-01.ibm.com/common/ssi/cgi-bin/ssialias?infotype=SA&subtype=WH&htmlfid=SEW03053WWEN, May 2015.

[31] ports. theharvester. http://tools.kali.org/information-gathering/theharvester, February 2014.

[32] Portswigger Web Security. Burp suite. https://portswigger.net/burp/, 2015.

[33] Rapid7 LLC. Metasploit project. https://www.metasploit.com/, November 2015.

[34] S. Shah and B. Mehtre. An overview of vulnerability assessment and penetration testing techniques. *Journal of Computer Virology and Hacking Techniques*, 11(1):27–49, 2015.

[35] Statista Inc. Global market share held by operating systems desktop pcs from january 2012 to december 2015.

http://www.statista.com/statistics/218089/global-market-share-of-windows-7/, January 2016.

[36] Strategic Foresight Initiative. Technological development and dependency. U.S. Department of Homeland Security: FEMA, May 2011.

[37] Symantec. Symantec intelligence report. Symantec Corporation, September 2015.

[38] F. Viggiani. Design and implementation of a non-aggressive automated penetration testing tool. Master's thesis, Norwegian University of Science and Technology, May 2013.

[39] VMware Inc. Vmware workstation pro overview. https://www.vmware.com/products/workstation, 2016.

[40] P. Wood. Internet security threat report. volume 20. Symantec Corporation, April 2015.

APPENDICES

Appendix A

TESTING SPECIFICS

The following figures and tables specify the various testing configurations we used. Each figure represents the test network with the various connections to ANEX, the input provided by the user, the size and number of subnets that were used, and the resulting target tree. The table associated with each figure contains information about each machine in the network. This information includes operating system, the services that were running, whether or not ANEX found the machine, whether or not ANEX exploited the machine, and the number of exploits that were generated and launched per machine. The table also contains the specified exploit rank threshold as well as the overall runtime.

A.1 Test 1

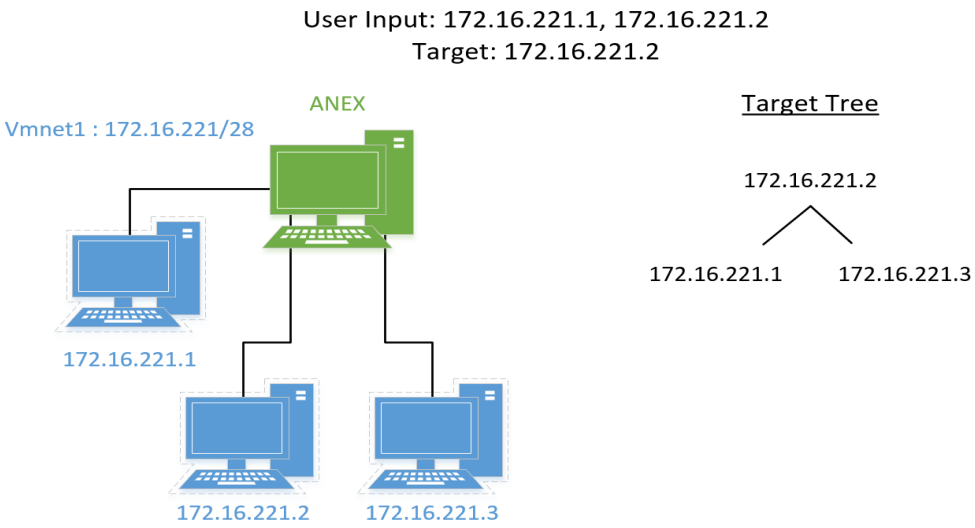

Figure A.1: Configuration for Test 1

Table A.1: Results for Test 1

OS	IP	Services	Found	Exploited	Exploits
Windows XP Bare	172.16.221.1	Netbios-ssn, Widnows RPC, Microsoft-ds	Yes	Yes	Launched: 3, Generated: 11
Windows 7	172.16.221.2	Windows RPC, Microsoft-ds, Microsoft lpd, Netbios-ssn	Yes	No	Launched: 1, Generated: 1
Ubuntu 14.04	172.16.221.3	Samba	Yes	No	Launched: 1, Generated: 1
		Exploit Rank Threshold	**5**	**Runtime**	**2 minutes, 12 seconds**

A.2 Test 2

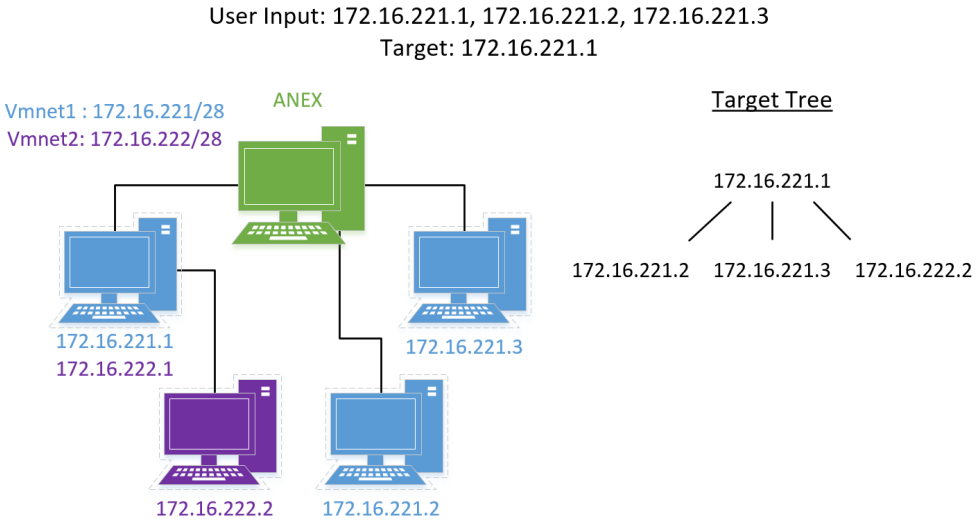

Figure A.2: Configuration for Test 2

Table A.2: Results for Test 2

OS	IP	Services	Found	Exploited	Exploits
Windows XP SP3	172.16.221.1	Netbios-ssn, Windows RPC, Microsoft-ds	Yes	Yes	Launched: 7, Generated: 11
CentOS 6.7	172.16.221.2	Squid, Dovecot, Postfix, SSH	Yes	No	Launched: 211, Generated: 211
Ubuntu 15.04	172.16.221.3	Apache, Samba, CUPS	Yes	No	Launched: 11, Generated: 11
Windows XP SP2	172.16.222.2	Netbios-ssn, Windows RPC, Microsoft-ds	Yes	No	Launched: 11, Generated: 11
		Exploit Rank Threshold	**5**	**Runtime**	**50 minutes, 42 seconds**

A.3 Test 3

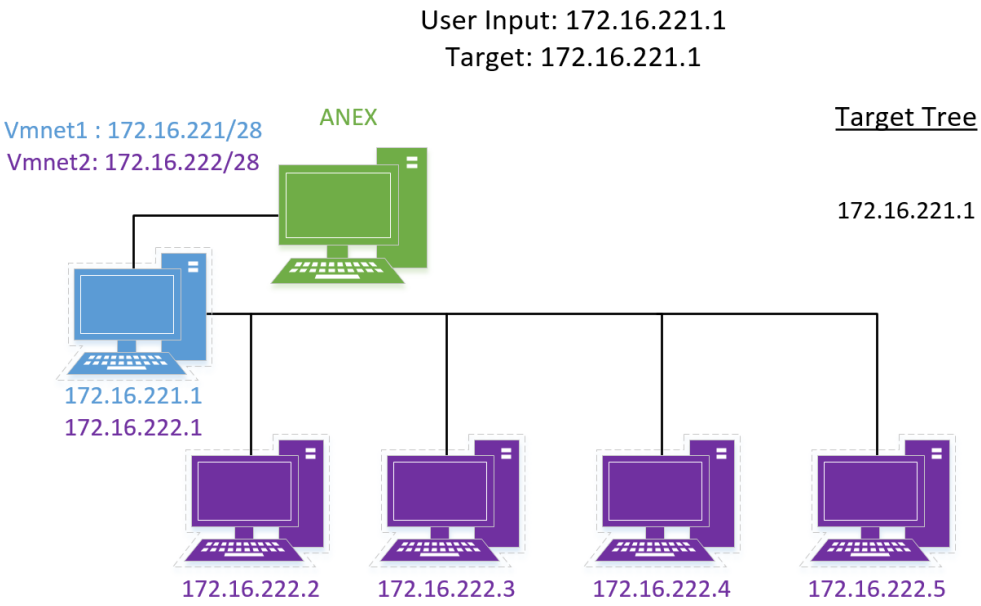

Figure A.3: Configuration for Test 3

Table A.3: Results for Test 3

OS	IP	Services	Found	Exploited	Exploits
Ubuntu 12.04	172.16.221.1	Vstpfd, SSH, Dnsmasq, Apache, Dovecot, Rpcbind, Samba, LDAP, MySQL	Yes	No	Launched: 240, Generated: 240
Windows XP Bare	172.16.222.2	Netbios-ssn, Windows RPC, Microsoft-ds	No	No	Launched: 0, Generated: 0
Windows XP SP1	172.16.222.3	Netbios-ssn, Windows RPC, Microsoft-ds	No	No	Launched: 0, Generated: 0
Windows XP SP2	172.16.222.4	Netbios-ssn, Windows RPC, Microsoft-ds	No	No	Launched: 0, Generated: 0
Windows XP SP3	172.16.222.5	Netbios-ssn, Windows RPC, Microsoft-ds	No	No	Launched: 0, Generated: 0
		Exploit Rank Threshold	**4**	**Runtime**	**45 minutes, 46 seconds**

A.4 Test 4

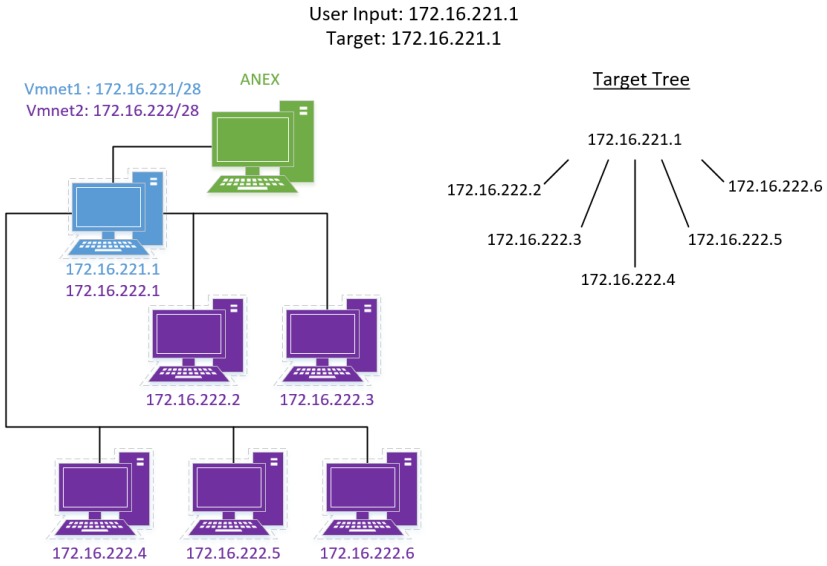

Figure A.4: Configuration for Test 4

Table A.4: Results for Test 4

OS	IP	Services	Found	Exploited	Exploits
Windows XP SP2	172.16.221.1	Netbios-ssn, Windows RPC, Microsoft-ds	Yes	Yes	Launched: 4, Generated: 23
Windows XP Bare	172.16.222.2	Netbios-ssn, Windows RPC, Microsoft-ds	Yes	No	Launched: 23, Generated: 23
Windows XP SP1	172.16.222.3	Netbios-ssn, Windows RPC, Microsoft-ds	Yes	No	Launched: 23, Generated: 23
Windows XP SP3	172.16.222.4	Netbios-ssn, Windows RPC, Microsoft-ds	Yes	No	Launched: 23, Generated: 23
Windows Server 2012 R2	172.16.222.5	Windows DNS, Netbios-ssn, Windows RPC, Microsoft-ds, Apache, MySQL	Yes	No	Launched: 206, Generated: 206
Ubuntu 12.04	172.16.222.6	SSH, Dnsmasq, Apache, Dovecot, Rpcbind, Samba, LDAP, MySQL	Yes	No	Launched: 264, Generated: 264
		Exploit Rank Threshold	**3**	**Runtime**	**105 minutes, 51 seconds**

A.5 Test 5

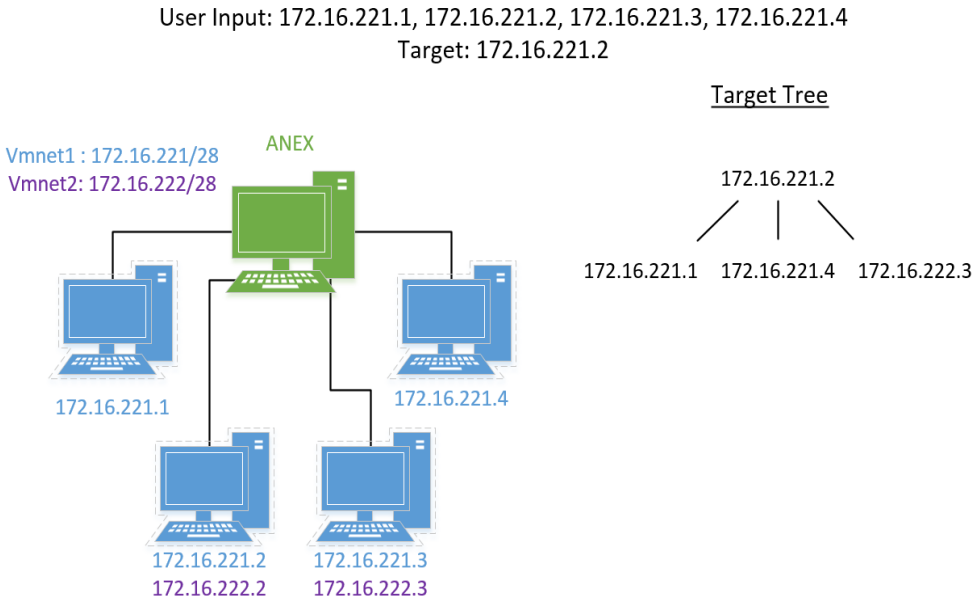

Figure A.5: Configuration for Test 5

Table A.5: Results for Test 5

OS	IP	Services	Found	Exploited	Exploits
Windows XP SP2	172.16.221.1	Microsoft-ds	Yes	Yes	Launched: 10, Generated: 20
Windows XP Bare	172.16.221.2	Windows RPC, Microsoft-ds	Yes	Yes	Launched: 3, Generated: 21
Windows XP SP1	172.16.221.3	None	No	No	Launched: 0, Generated: 0
	172.16.222.3	Netbios-ssn, Windows RPC, Microsoft-ds	Yes	Yes	Launched: 9, Generated: 23
Windows XP SP3	172.16.221.4	Microsoft-ds	Yes	Yes	Launched: 11, Generated: 20
		Exploit Rank Threshold	**3**	**Runtime**	**18 minutes,30 seconds**

A.6 Test 6

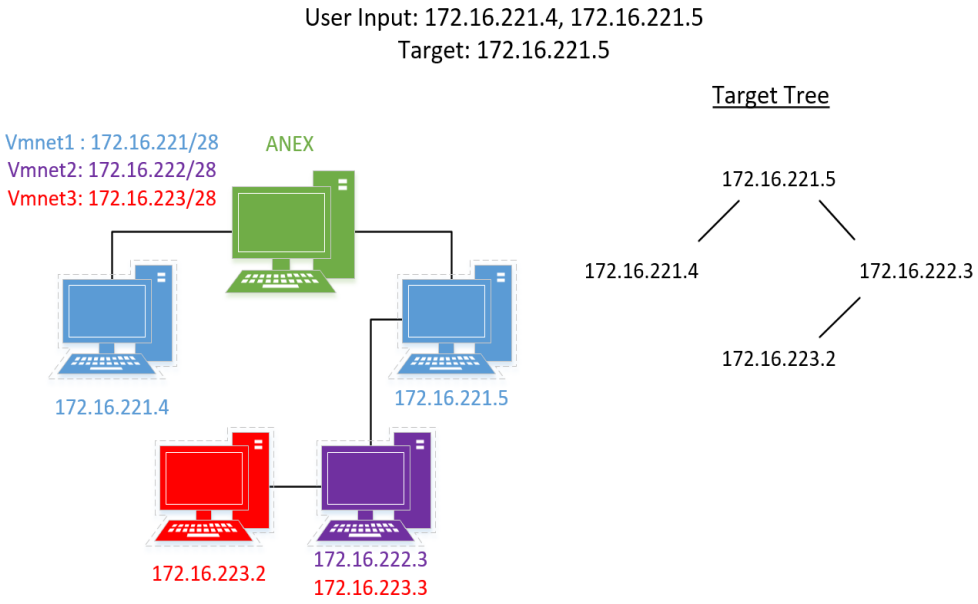

Figure A.6: Configuration for Test 6

Table A.6: Results for Test 6

OS	IP	Services	Found	Exploited	Exploits
Windows XP SP2	172.16.221.4	Microsoft-ds	Yes	Yes	Launched: 6, Generated: 20
Windows XP Bare	172.16.223.2	Netbios-ssn, Windows RPC, Microsoft-ds	Yes	No	Launched: 23, Generated: 23
Windows XP SP1	172.16.222.3	Netbios-ssn, Windows RPC, Microsoft-ds	Yes	Yes	Launched: 3, Generated: 23
Windows XP SP3	172.16.221.5	Microsoft-ds	Yes	Yes	Launched: 6, Generated: 20
		Exploit Rank Threshold	**3**	**Runtime**	**23 minutes, 28 seconds**

A.7 Test 7

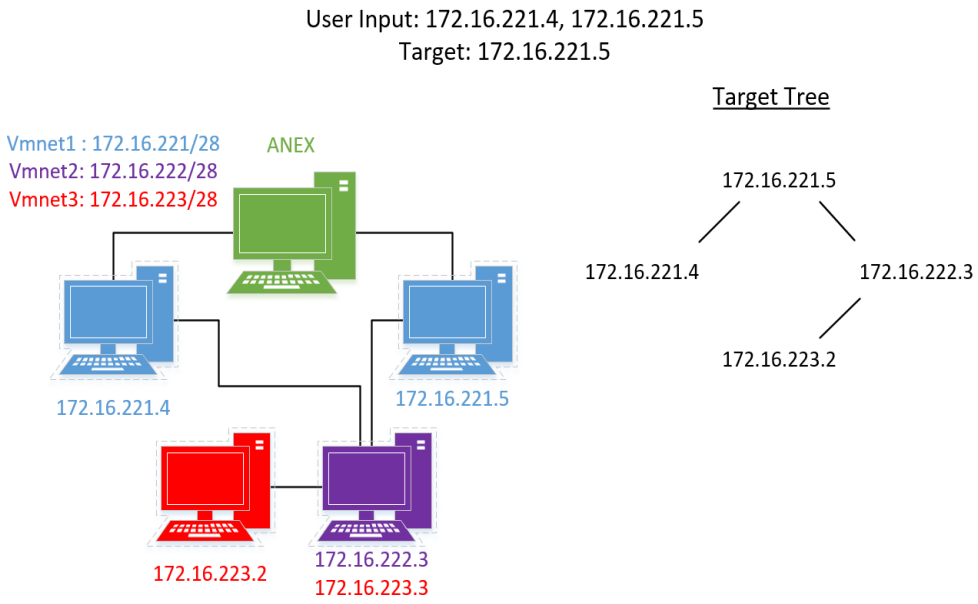

Figure A.7: Configuration for Test 7

Table A.7: Results for Test 7

OS	IP	Services	Found	Exploited	Exploits
Windows XP SP2	172.16.221.4	Microsoft-ds	Yes	Yes	Launched: 14, Generated: 20
Windows XP Bare	172.16.223.2	Netbios-ssn, Windows RPC, Microsoft-ds	Yes	Yes	Launched: 2, Generated: 23
Windows XP SP1	172.16.222.3	Netbios-ssn, Windows RPC, Microsoft-ds	Yes	Yes	Launched: 3, Generated: 23
Windows XP SP3	172.16.221.5	Microsoft-ds	Yes	Yes	Launched: 20, Generated: 20
		Exploit Rank Threshold	**2**	**Runtime**	**25 minutes, 30 seconds**