NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

DEVELOPING SIMULATED CYBER ATTACK SCENARIOS AGAINST VIRTUALIZED ADVERSARY NETWORKS

by

Luis E. Aybar

March 2017

Thesis Advisor:	Alan Shaffer
Co-Advisor:	Gurminder Singh

Approved for public release. Distribution is unlimited.

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704–0188

1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** March 2017	**3. REPORT TYPE AND DATES COVERED** Master's thesis

4. TITLE AND SUBTITLE
DEVELOPING SIMULATED CYBER ATTACK SCENARIOS AGAINST VIRTUALIZED ADVERSARY NETWORKS

5. FUNDING NUMBERS

6. AUTHOR(S) Luis E. Aybar

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Marine Forces Cyber Command
Fort Meade, MD 20755

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol number ____N/A____.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release. Distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (maximum 200 words).

Cyberspace is now recognized as a critical center of gravity for modern military forces. The ability to maintain operational networks, while degrading the enemy's network capability, is a key consideration for military commanders. Conducting effective cyber-attacks against sophisticated adversaries requires the ability to develop, test, and refine cyber-attack scenarios before they are used operationally, a requirement that is not as well defined in the cyber domain as it is in the physical domain. This research introduces several concepts to address this need, and creates a prototype for cyber-attack scenario development and testing in a virtual test environment. Commercial and custom software tools that provide the ability to conduct network vulnerability testing are reviewed for their suitability as candidates for the framework of this project. Leveraging the extensible architecture of the Malicious Activity Simulation Tool (MAST) custom framework allowed for the implementation of new interaction parameters, and provided temporal specificity and target discrimination of cyber-attack scenario tests. The prototype successfully integrated a virtualized test environment used to simulate an adversary network and the enhanced MAST capability to demonstrate the viability of a cyber-attack scenario development platform to address the needs of modern offensive cyber operations. Based on these results, we recommend continued development of MAST with the intent to ultimately deploy to Department of Defense cyber operations teams.

14. SUBJECT TERMS
offensive, malware, cyber, virtualization, attack, simulated, modeling, MAST, MAVNATT

15. NUMBER OF PAGES
103

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

DEVELOPING SIMULATED CYBER ATTACK SCENARIOS AGAINST VIRTUALIZED ADVERSARY NETWORKS

Luis E. Aybar
Lieutenant, United States Navy
B.S., University of Houston, 1993

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN CYBER SYSTEMS AND OPERATIONS

from the

NAVAL POSTGRADUATE SCHOOL
March 2017

Approved by: Alan Shaffer
Thesis Advisor

Gurminder Singh
Co-Advisor

Cynthia Irvine
Chair, Cyber Academic Group

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Cyberspace is now recognized as a critical center of gravity for modern military forces. The ability to maintain operational networks, while degrading the enemy's network capability, is a key consideration for military commanders. Conducting effective cyber-attacks against sophisticated adversaries requires the ability to develop, test, and refine cyber-attack scenarios before they are used operationally, a requirement that is not as well defined in the cyber domain as it is in the physical domain. This research introduces several concepts to address this need, and creates a prototype for cyber-attack scenario development and testing in a virtual test environment. Commercial and custom software tools that provide the ability to conduct network vulnerability testing are reviewed for their suitability as candidates for the framework of this project. Leveraging the extensible architecture of the Malicious Activity Simulation Tool (MAST) custom framework allowed for the implementation of new interaction parameters, and provided temporal specificity and target discrimination of cyber-attack scenario tests. The prototype successfully integrated a virtualized test environment used to simulate an adversary network and the enhanced MAST capability to demonstrate the viability of a cyber-attack scenario development platform to address the needs of modern offensive cyber operations. Based on these results, we recommend continued development of MAST with the intent to ultimately deploy to Department of Defense cyber operations teams.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

API	Advanced Programming Interface
CD	compact disk
CEH	Certified Ethical Hacker
CERT	Computer Emergency Response Team
CG	Change Group
CLI	Command Line Interface
COMPOSE	Common PC Operating System Environment
CYBL	Cyber Battle Lab
DARPA	Defense Advanced Research Projects Agency
DOD	Department of Defense
DoS	Denial of Service
DDoS	Distributed Denial of Service
GUI	Graphical User Interface
HBSS	Host Based Security System
IBM	International Business Machines
IDE	Integrated Development Environment
IDS	Intrusion Detection System
IP	Internet Protocol
ISO	International Organization for Standards
ISR	Intelligence, Surveillance, Reconnaissance
JDK	Java Development Kit
LAN	Local Area Network
MAC	Media Access Control
MAST	Malicious Activity Simulation Tool
MAVNATT	Mapping, Awareness, and Virtualization Network Administrator Training Tool
MCP	MAST Communication Protocol
MSF	Metasploit Framework
NCOR	Navy Cyber Operating Range
NPS	Naval Postgraduate School

OCO	Offensive Cyber Operations
PTES	Penetration Testing Execution Standard
RC	Return Code
ROC	Rehearsal of Concept
SCADA	Supervisory Control and Data Acquisition
SCC	Send Client Command
SEI	Software Engineering Institute
SES	Scenario Execution Server
SGC	Send Group Command
SGS	Scenario Generation Server
SQL	Standard Query Language
USD	United States dollar
VM	virtual machine
VMDK	virtual machine disk

ACKNOWLEDGMENTS

First and foremost, I would like to thank my wife, Adrienne, for her support in this endeavor. No other person has had to make more of a sacrifice for to me achieve this goal. Over the past two years, you persevered through all the long days, weekends, and holidays I spent working nearly non-stop, yet you remained steadfast with your support and encouragement. You kept the house running smoothly and took care of our new son so I could concentrate on my studies. The sacrifices you make as a Navy spouse largely go unrecognized, but I want to publicly thank you for your support and love during this difficult but rewarding journey.

I would also like to thank the other faculty with whom I've had the pleasure of sharing a class. The depth of your knowledge and passion for your subjects has been especially refreshing for me. I'm glad I had the opportunity to get to know you all and hope that I can reach back for guidance in the future.

I would like to recognize my cohort and NPS friends who helped keep me sane as the deadlines and assignments would pile ever higher. Your can-do attitude and kind words helped me keep everything in perspective. It's been a pleasure serving with you all during this time.

I would like to express my greatest appreciation and gratitude to my advisors, Dr. Alan Shaffer and Dr. Gurminder Singh. I appreciate your quick response to any request, steady hand on the rudder, and general belief in the project. It has been a pleasure to work with you both.

Lastly, I would like to thank Big Navy for providing me the opportunity to pursue a master's degree in residence. Having the ability to direct all my time to the pursuit of this academic goal, and the accompanying academic enrichment, has been especially rewarding. This degree will forever be part of my portfolio, and I look forward to leveraging the knowledge gained during the last two years here for the rest of my career.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

In today’s military, a computer network is an essential element of combat power with uses extending from command and control, communications, logistics, computer network attack, to computer network exploitation. Given the reliance of armed forces on digital networks to carry out an array of warfare functions, these strategic centers of gravity have become high priority targets in modern military conflicts. From the uppermost echelons of command to the most remote deployed units, the exploitation of enemy networks is key to military engagements now and in the future.

In the traditional warfare domains of air, land, and sea, the ability to ascertain the capabilities and orders-of-battle (OOB) of adversary forces is well understood. This knowledge of enemy forces allows for the rigorous testing and refinement of warfare capabilities, which facilitates effective operational planning. Different parameters and scenarios can be set against an adversary’s capabilities, allowing for a variety of courses of action to be wargamed effectively. This ability to test warfare capabilities, from the operational to the tactical level, is a key element that ensures the military readiness of today’s combat forces.

In the cyber domain, the ability to know an adversary’s capabilities, vulnerabilities, and OOB is less well defined. Furthermore, the need to be able to develop and test offensive cyber capabilities and scenarios against an accurate instantiation of enemy networks to ensure operational success is not as mature or robust as in the physical domain(s). To address that need, this research examined various tools, techniques, and existing software frameworks that could be used or modified to develop simulated attack scenarios for effective testing against virtualized adversary networks.

A. PROBLEM STATEMENT

Offensive cyber operations (OCO) are often executed without the ability to first simulate and test their effectiveness against a realistic model of an adversary’s network. Having such an environment could help cyber operators to better understand the effects of specific cyber-attacks, their impact on an adversary’s systems and operations, and the collateral and secondary effects that could result.

1. Primary Question

How can offensive cyber tools and exploits be developed and tested effectively in a controlled environment against virtualized models of adversary networks?

2. Secondary Questions

What control mechanisms or methods would allow malware to focus on a user-defined target set or grouping (e.g., a single host or single subnet)?

What methods can be used to perform the temporal sequencing of malware?

What methods would allow for the simulation of malware within a virtualized network?

B. SCOPE

The primary scope of this thesis was to design a methodology and lay the foundation for a process that can be used to design, develop, and test simulated cyber-attack scenarios in a virtualized environment. The resulting methodology facilitates follow-on research and the creation of enhanced software capabilities for use in offensive cyber scenarios. The systems developed in this thesis are not intended to supplant existing DOD and service cyber test ranges, but are intended to serve a specialized need for the DOD cyber warrior in the early stages of scenario development

Cyber mission planning, which precedes the development of cyber-attack scenarios, defines the mission objectives and desired end-state of any offensive cyber operation. Achieving those goals via cyber means requires distinct knowledge of the target network in order to create effects on different aspects of an adversary’s computer systems, which was a focus of this thesis. The mission planning aspects of offensive cyber-attacks, although they are a necessary first step to focus the simulation development, is beyond the scope of this thesis.

C. OBJECTIVES

The main objective of this work was to provide a proof-of-concept via an established custom software framework that enables the development of software modules for use in simulated offensive cyber scenarios. An additional objective was to develop the capability for both temporal and target specificity in cyber-attacks. Finally, a virtualized test environment was needed to simulate an adversary network on which to demonstrate new offensive cyber capabilities developed and implemented in this research.

D. APPROACH

A survey of several existing technologies and customized solutions that attempt to develop and test cyber-attack schemas was performed to evaluate each of their suitability as a candidate for the foundational platform. Then, after identifying the advanced interaction parameters needed to enhance the simulated malware's usability, software modules were designed to support the integration of new capabilities into the customized framework selected as the foundation. Finally, a prototype was implemented to demonstrate scenarios to test the interaction behaviors, and set the conditions for follow-on research.

E. BENEFITS OF STUDY

This work has devised, designed, and implemented advanced simulated malware interaction methods to provide the U.S. cyber warrior with more accurate methods for the design and testing of advanced cyber-attack simulations. These new capabilities will also allow DOD and U.S. cyber forces to model adversary capabilities and responses with a higher degree of precision in order to more effectively conduct wargames and test various strategies and courses of action prior to an actual conflict. As a result, cyber-attack developers will have greater confidence that their cyber-attack scenario will perform as planned. This will enable the cyber simulation designers to provide a high level of assurance to their senior leaders that the virtually-derived cyber-attack scenario will perform as planned in the physical world.

F. ORGANIZATION

The rest of the thesis is organized in the following manner:

Chapter II: Background. This chapter examines basic concepts and definitions commonly used within the cyber lexicon. Additionally, this chapter surveys existing software tools and frameworks to determine their viability as the foundational platform for the development of simulated cyber-attack scenarios.

Chapter III: Design and Methodology. This chapter puts forth key design requirements for the creation of a virtual environment in which to conduct simulated cyber-attacks against a virtualized adversary network. It also reviews different approaches to realize the virtualized test environment, and different interaction methodologies for the simulated malware to provide results that translate into real-world solutions. Finally, it defines factors essential to the optimization of the virtualized environment for testing offensive cyber operations.

Chapter IV: System Implementation. This chapter details the test environment, the creation of the new software modules, and the testing process. It also reviews the design, use, and capabilities of the test environment utilized to model an adversary network. Additionally, the chapter examines the functions of the new software modules and demonstrates the derived interaction behaviors and their applicability in offensive cyber scenarios.

Chapter V: Conclusions and Future Work. This chapter examines the results of the implemented software modules and their effectiveness within the test environment. It also reviews the software framework for its efficacy as an execution platform. Additionally, areas that can be enhanced further to provide more robust interactions with even greater capabilities for the offensive oriented cyber scenario author are identified. Finally, suggestions are provided for follow-on research.

II. BACKGROUND

A. OVERVIEW

The extensive use of cyberspace to conduct worldwide communications, commerce, and banking, as well as to provide the control of critical national infrastructure, has made the connectivity that cyberspace provides indispensable for operations in modern societies. In a military context, the use of cyberspace has become intricately woven into most military operations—creating greater efficiencies, but also providing a new attack surface an adversary might exploit to wage war. The ability to conduct successful attacks in the physical domain(s) requires detailed planning, robust intelligence, surveillance, and reconnaissance (ISR), as well as methods to rehearse and simulate the attack. Although the cyber domain has many differences from the traditional physical domains, such as the speed of execution, they share common elements for the execution of a successful attack—detailed planning, good intelligence, and repeated rehearsals to ensure every detail of the plan is correct. In the cyber domain, the ability to be able "*rehearse*" attacks via offensive cyber operations continues to be a pressing need within the U.S. military. However, the requirement to find software or network vulnerabilities and then develop exploits as either an attacker or a defender of computer networks has existed for many years. It is through this nexus of common needs that we examine the following software frameworks.

In this chapter, we survey existing software products and methods that could provide a framework within which these exploits and simulated attack scenarios could be developed. Our review examines both custom-designed and commercial-off-the-shelf (COTS) products to determine the range of existing capability available, and how well each could meet the DOD's needs as a lightweight, tactical based system that could be used in offensive cyber operations.

B. DEFINITIONS

This section defines common terms used in the cyber operations lexicon that are necessary for an understanding of the development of simulated attack scenarios.

1. Exploit

An *exploit* is the means by which a pen-tester or malicious actor uses a flaw or vulnerability in some software, system or service to penetrate a system or network of interest. Common examples of cyber-attack exploits are buffer overflows and manipulation of security misconfigurations (i.e., default passwords) [1].

2. Payload

A *payload* is the code that will create the desired effect that is delivered by the exploit. An exploit is, metaphorically speaking, a delivery vehicle for the cyber payload. Software such as a Trojan horse or a reverse shell are common examples of a payload [1], [2].

3. Penetration Testing

Penetration testing is the process of testing a network or system by "simulating real attacks to assess the risk associated with potential security breaches" [1]. Using various software tools, the pen-tester will attempt to uncover vulnerabilities in a system, and exploit them in an attempt to simulate how a real threat might attack the system [1].

4. Red Teams

Red teams are groups whose task is to "to emulate the adaptive character of an enemy" [3]. In effect, they can challenge the network defenders utilizing similar tactics and procedures that a particular adversary might utilize. Their true strength lies in their ability to portray properly the thinking and methods of a real-world adversary [3].

5. Network Sandbox

A *network sandbox* is a metaphorical reference in computer security to a separate and isolated network space that can be used for testing or developing malware, or any software that could potentially damage the system outside the prescribed environment. Common techniques to create this network isolation include physically air-gapping the network, utilizing hardware/software virtualization, or leveraging other features of the operating system that keep a process confined to a subset of the system [4].

6. Black Hat

Black hat is a term used for individuals with extraordinary computing knowledge who use their skills and abilities for malicious activities, such as illegally breaking into computer networks or creating viruses. The term is a reference to a commonly held cultural association of the fictional evil character wearing a black hat [5], [6].

7. White Hat

White hat is a term used for individuals with extraordinary computing knowledge who use their skills for defensive purposes; they may also serve as computer or network security analysts. Again, the term is a cultural or stereotypical reference to fictional white-hatted characters who enforce the law, or who are represented as virtuous [2], [6].

C. MAVNATT

The Mapping, Awareness, and Virtualization Network Administrator Training Tool (MAVNATT) is a software framework developed at NPS in 2015 to enable tactical network administrators the ability to train on and evaluate live networks [7]. MAVNATT provides three core capabilities: 1) the ability to map a network, 2) the ability to maintain awareness of the changing status of the mapped network, and 3) a virtualized instantiation of the mapped network that can be used for training network administrators and situational awareness of user activity [7]. This description of the MAVNATT framework will be brief, as the full details of the MAVNATT framework and project are described in Daniel McBride's thesis, "Mapping, awareness, and virtualization network administrator training tool (MAVNATT) architecture and framework" [7].

The MAVNATT framework was designed to demonstrate that a "lightweight system can be utilized in tactical environments to provide training and evaluation capabilities to tactical network administrators without jeopardizing the unit mission and without jeopardizing the network on which users are operating" [7]. The MAVNATT architecture consists of three separate components or modules—Mapping, Awareness, and Virtualization—all connected through an underlying framework. Its design, depicted in Figure 1, portrays the interrelationship between the modules and the scope of responsibility of each component. The Mapping Module interfaces with and acquires the network topology data from the live network [8], while the Virtualization Module interacts with the virtual representation of the live network through virtualization management software. The Awareness Module maintains the symmetry of the architecture as it interacts with both the live network and virtualized network and is the nexus for the *monitor* and *train* functions [9].

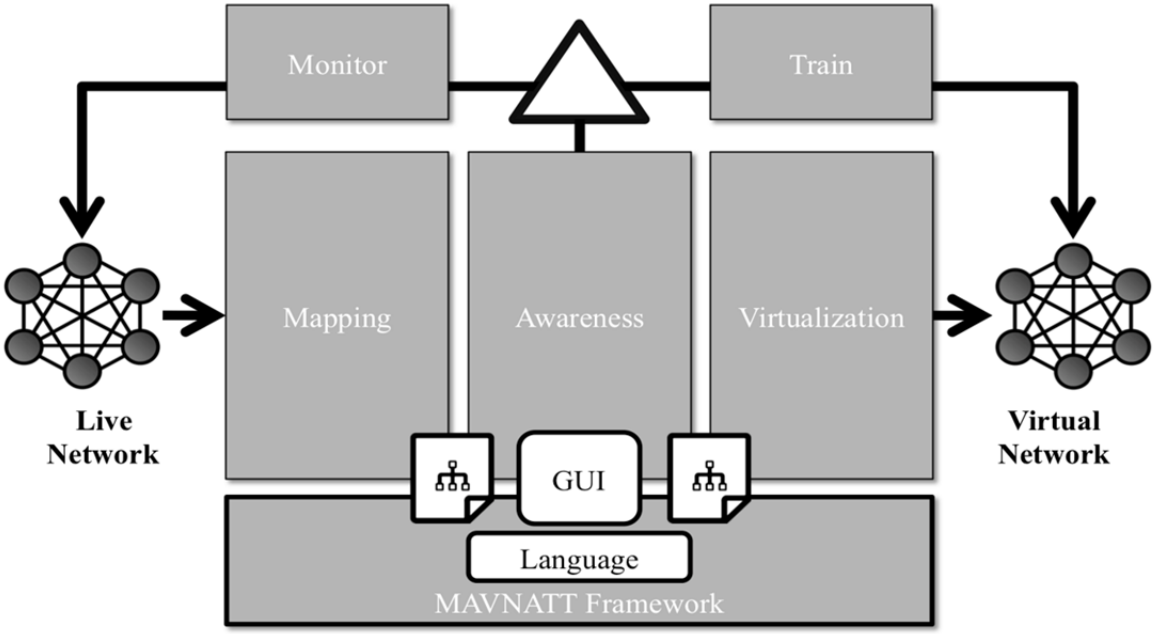


Figure 1. MAVNATT Conceptual Model. Source: [7].

1. Mapping Module

The Mapping Module is responsible for mapping and enumerating the components of a network, and their connections, to determine the physical topology of that network [8]. The resultant information is parsed into an open-format graph file that contains information on the attributes of each host, and their interconnections [8], [9].

Once the open-graph format file has been created, it may be passed to the Visualization Module to be instantiated as a virtual network by the hypervisor [9]. The Mapping Module will continue to update the open-graph format file with updates from the Awareness Module to reflect changes in the live network configuration.

2. Awareness Module

The Awareness Module's function is to provide a visual representation of the network devices in an estimated topological layout based on their attributes and connections, and the status of the various components in the network [7]. This visual representation permits a user the ability to quickly gain and maintain situational awareness of the network under examination. This module also provides the additional capability of allowing virtual devices to be inserted into the network representation to facilitate training scenarios [7].

3. Virtualization Module

Once the Mapping Module has created the open-graph file of a live network, the Virtualization Module uses the file data to create a virtualized instantiation of the network in a logically separate partition. This module also creates the interface between MAVNATT and a hypervisor to support the creation of the virtual network [7], [9].

D. MAST

The Malicious Activity Simulation Tool (MAST) system utilizes a three-tiered client-server model, that was designed to be able to test an organization's users, administrators, and security tools [10], [11]. An overview of the MAST architecture is shown in Figure 2. It consists of a top-level Scenario Generation Server (SGS), a second-tier Scenario Execution Server (SES), and a third tier comprised of the MAST client(s). This architecture allows the SGS to be geographically disparate from the SES and the MAST client(s) [11].

MAST uses two types of files to facilitate its operations, client module files and scenario files. Client module files are what contain the core malware simulation code known as Simulated Malware (SimWare), while the scenario files contain instructions for particular clients to run specific client modules [11] .

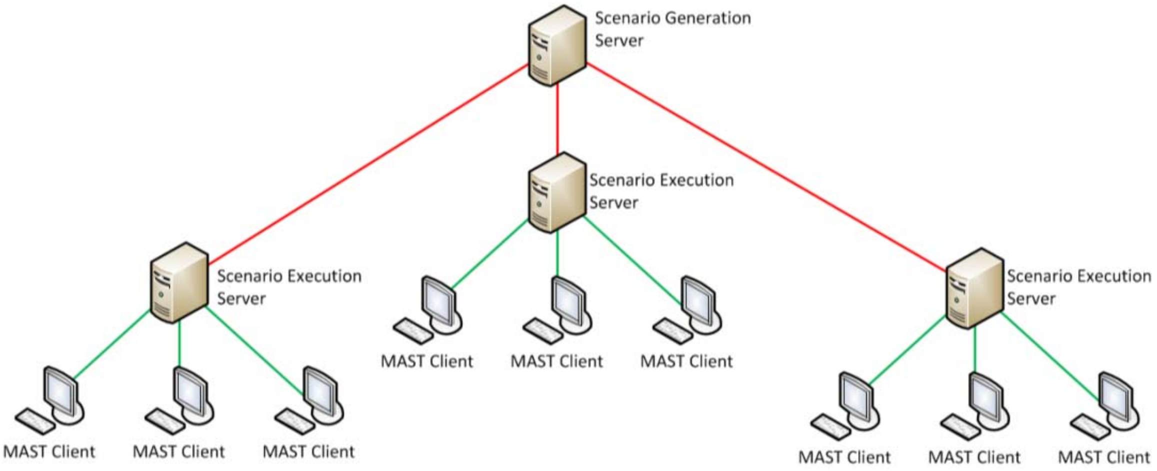

Scenario Generation Server
Scenario Execution Server
Scenario Execution Server
Scenario Execution Server
MAST Client
MAST Client
MAST Client
MAST Client
MAST Client
MAST Client
MAST Client
MAST Client
MAST Client

Figure 2. MAST Three-Tier Client Server Architecture. Source: [11].

1. Scenario Generation Server

The SGS sits atop the tree structure of MAST and is responsible for many of the command level functions. It installs as a Java desktop application program and can be run from a local or remote location to control the execution server(s) [11]. The SGS is responsible for:

- Generating and disseminating training scenarios
- Providing a central repository for all available scenarios
- Controlling connected Scenario Execution Servers
- Running independent scenarios on multiple networks simultaneously.

2. Scenario Execution Server

The SES is the middle-tier of the MAST architecture, receives commands from the SGS, and communicates directly with the MAST clients. The SES also handles the execution of the scenarios on the clients and requires a local install on the operational or target network for training [11]. As the middle manager of MAST, SES is responsible for a number of intermediate tasks that include [12]:

- Distributing SimWare and scenario files to MAST clients
- Maintaining MAST client and SimWare status
- Overseeing all scenario activity and MAST clients
- Handling “kill switch” functionality in case of emergency shutdown
- Maintaining a copy of deployed SimWare and scenario files.

3. Scenarios

MAST Scenario files are a compilation of SimWare modules along with various options and commands needed to run them. These scenario files or scripts allow the tester to combine various amalgamations of the SimWare modules as needed to exercise various features on the system under test. The SGS and SES run these scenario files and collect the results, and in some instances respond with the appropriate “reply” in accordance with the scenario parameters [11].

4. Clients

The MAST client software is a Java program that runs on each of the client machines that is a part of the MAST test network [10], [11]. The client application is controlled by the SES and does the following:

- Reports the status of the running scenario to the MAST SES
- Lists all the modules that are installed on the host client to the MAST SES
- Executes the scenario as directed from the MAST SES.

5. Simulation Malware modules

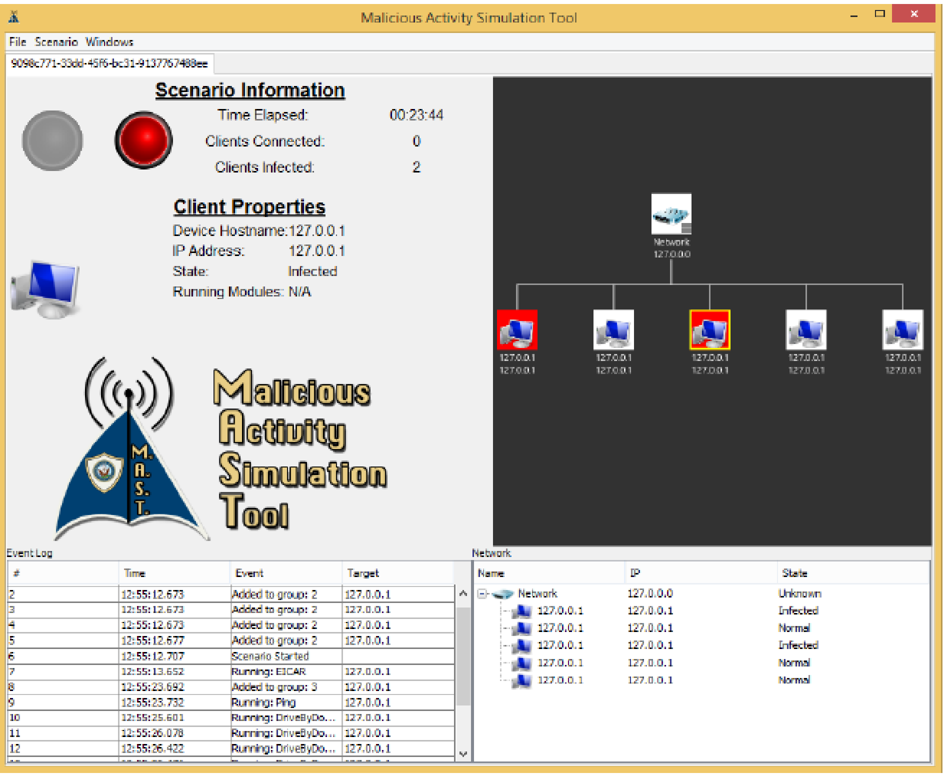
MAST SimWare modules contain the code that provides the malware mimicking behavior and are at the heart of the MAST framework. These mimics can simulate a range of malevolent behaviors such as scanning, pinging, and hijacking of browsers.

Additionally, these malware modules could be identified as a particular type of attack by their binary signature matching a known type of malware [11].

Due to the modular nature of these software objects and the extensibility of the MAST framework, new SimWare modules can be developed to mimic more advanced malware and then be inserted for use into the MAST framework.

6. Graphical User Interface

The MAST framework also has a Graphical User Interface (GUI) that allows for the starting, stopping, and selection of various scenarios that can be executed. The GUI also provides a way to visualize the feedback from the clients controlled by the SES. Additionally, the MAST GUI and the server instantiation can be on different computers and in different locations due the GUI's client-server based architecture. An example of the MAST GUI, as shown in Figure 3, displays the scenario information, client properties, and the network of computers being tested with MAST.



Scenario Information

Time Elapsed: 00:23:44
Clients Connected: 0
Clients Infected: 2

Client Properties

Device Hostname: 127.0.0.1
IP Address: 127.0.0.1
State: Infected
Running Modules: N/A

#	Time	Event	Target
2	12:55:12.673	Added to group: 2	127.0.0.1
3	12:55:12.673	Added to group: 2	127.0.0.1
4	12:55:12.673	Added to group: 2	127.0.0.1
5	12:55:12.677	Added to group: 2	127.0.0.1
6	12:55:12.707	Scenario Started	
7	12:55:13.652	Running: EICAR	127.0.0.1
8	12:55:23.692	Added to group: 3	127.0.0.1
9	12:55:23.732	Running: Ping	127.0.0.1
10	12:55:25.601	Running: DriveByDo…	127.0.0.1
11	12:55:26.078	Running: DriveByDo…	127.0.0.1
12	12:55:26.422	Running: DriveByDo…	127.0.0.1

Name	IP	State
Network	127.0.0.0	Unknown
127.0.0.1	127.0.0.1	Infected
127.0.0.1	127.0.0.1	Normal
127.0.0.1	127.0.0.1	Infected
127.0.0.1	127.0.0.1	Normal
127.0.0.1	127.0.0.1	Normal

Figure 3. MAST Graphical User Interface. Source: [10].

7. Evolutionary Progress

Malware Mimics, now known as Malicious Activity Simulation Tool (MAST), is a software framework based on the NPS thesis work of William Taff and Paul Salevski in 2011 [13]. Their software framework, Malware Mimics, was designed to meet the following objectives: 1) provide a system for training system administrators that could mimic malware on live networks, 2) allow the mimicking system to be geographically separate from the system or network being tested, and, 3) cause no harm to the live system being used as the training environment. Their original system provided a useful network-testing tool that could complement Red Team pen-testers and provide additional monitoring and testing capabilities to network administrators.

In recent years, several NPS students have made enhancements to the original framework. In 2012, Justin Neff compared the Malware Mimics framework, renamed MAST, to other commercial software platforms to determine the effectiveness of MAST as a training tool. He concluded that MAST was a useful and effective tool that provided significant complementary capability to Red Team activities [14]. Also in 2012, Ray Longoria Jr. proved that the MAST framework was capable of scaling to support a large number of additional clients without significant detrimental impact to the remaining system and network resources [15].

In 2013, Aaron M. Littlejohn and Ehab Makhlouf proved that the MAST framework was able to perform its training and malware mimicking functions on a shipboard LAN, using the Common PC Operating System Environment (COMPOSE) via the Navy Cyber Operating Range (NCOR) [16].

Following this, Brian Diana, in 2015, identified security vulnerabilities within the MAST framework, and recommended encrypting its communication channels, and developing an authentication scheme for its modules [12]. Subsequent work by Adam Farber and Robert Rawls in 2015 addressed the security concerns raised by Diana by implementing secure communication between the endpoints, and a digitally signed verification protocol between the Generation and Execution servers of MAST. Also in 2015, Erik Lowney published the MAST Communication Protocol (MCP), which allowed MAST to synchronize scenarios across the network, and implemented the emergency “kill switch” behavior which allows the MAST server(s) to immediately terminate the simulation scenario [10], [11].

Finally, in 2016, Gregory Belli published a scenario scripting language for MAST to allow the incorporation of multiple malware mimics into a scenario file. These scenario files can be customized, saved, and run on any instance of a MAST client. Additionally, Belli described the ability for MAST to be responsive to signals received from MAST clients and to reply with programmed responses to simulate complex interactions between MAST and the client [10].

E. SURVEY OF EXISTING TOOLS AND METHODS

The desire to be able to test one’s own network systems via a collection of software exploits that a black hat attacker or white hat tester might utilize has spurned the creation of several commercial and open-source products that provide that penetration testing capability. Their continued improvement over the last fifteen years has led to robust products that contain an array of features [1], [17], [18]. As a result, several of these open-source and COTS penetration-testing products are used by a variety of government, academic, and commercial organizations for routine testing of their own networks.

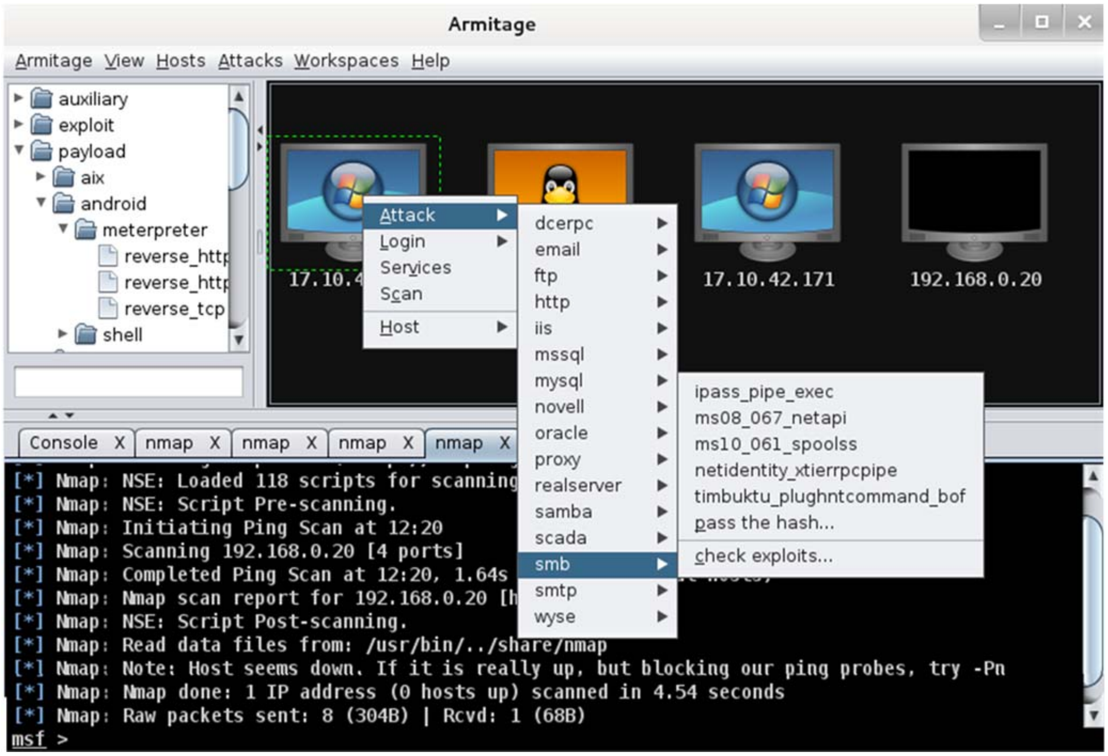
In sections B and C of this chapter, we examined a few custom software frameworks that that were architected explicitly for the design and development of simulated malware (SimWare) attack scenarios and the mapping, awareness, and virtualization of networks (MAST and MAVNATT, respectively). In the following survey, we review some of the more common penetration testing tools for their ability to be used to develop and test simulated cyber-attack scenarios. We assess if they could provide a software framework or methods that could be useful in the development and testing of simulated cyber-attack scenarios and serve as a basis for this research effort. The evaluation criteria, as mentioned earlier, will be: 1) a software framework in which to create the malware mimics and provide a mechanism to start, aim, and control said exploits; 2) a testing environment in which these created exploits can be exercised and tested; and 3) methods to observe the resulting behavior in the testing environment.

1. Metasploit Framework

The Metasploit Framework (MSF) is a software framework for penetration testing. It has been available to the public since 2004, has undergone many revisions over the years, and was acquired by Rapid 7 in 2009 [1]. The MSF is an open-source product that is free to download; however, the makers of the MSF also offer two commercial versions (Metasploit Express and Nexpose Ultimate with Metasploit Pro) that provide a web interface, increased automation, and more robust reporting features that are available for $5,000 to $13,000 USD, respectively [1], [17], [19].

The MSF has a rich set of features that enable it to be a premier penetration testing tool but MSF also has additional capabilities that lend themselves to the development of simulated cyber-attack scenarios. One of the foremost capabilities of MSF is the ability for users to develop and deploy their own custom designed modules. Within the context of MSF, a module is any collection of code that can be executed within MSF (i.e., an exploit or a payload) [1]. MSF also comes preloaded with hundreds of common exploits and payloads written in the Ruby programming language. Any of these can be edited or modified to form a new payload or exploit that may be needed for a particular scenario. Additionally, MSF contains a class of auxiliary modules that are used to perform non-exploit tasks such as scanning, fuzzing, or brute force password cracking [1]. These auxiliary modules can be modified and customized, providing an additional range of capabilities when developing simulated cyber-attack scenarios. Moreover, if the needed exploits exist in another programming language, C for example, MSF facilitates the porting over to the exploit to the MSF. This creation, customization, and porting capability of new and existing modules allows for a wide variety of exploits and auxiliary functions to be added to the framework, thereby allowing for the development of wholly custom attack scenarios [1]. These MSF modules are analogues to the code segments that are run by the MAST clients; however, the MSF modules were not designed to mimic malware and simulate the behavior of an exploit or payload, but to be the actual exploit/payload.

MSF also provides three methods to aim, launch, and interact with the various stock or custom-developed MSF modules. These methods are the terminal-like MSF console, the Armitage GUI, and the Meterpreter shell. The MSF console allows a user to launch exploits, create listeners, and access all the settings within the MSF using instructions typed at a command prompt. The Meterpreter is a shell-type interface that allows the user to interact with an exploited machine and utilize the full suite of MSF features. The Meterpreter can utilize a full range of organic MSF commands, but also supports creation of new scripts to provide any needed functionality that a user may require [1]. The Armitage GUI is a full-featured interface allowing a user to click and select the various services of the framework with standard point and click methodology. From the GUI, all the commands and preloaded exploits that are available in the framework can be selected and executed. Figure 4 is an example of the Armitage GUI, which shows the range of attacks the framework has identified for a particular host based on its vulnerability scan.



```
[*] Nmap: NSE: Loaded 118 scripts for scanning
[*] Nmap: NSE: Script Pre-scanning.
[*] Nmap: Initiating Ping Scan at 12:20
[*] Nmap: Scanning 192.168.0.20 [4 ports]
[*] Nmap: Completed Ping Scan at 12:20, 1.64s
[*] Nmap: Nmap scan report for 192.168.0.20 [h
[*] Nmap: NSE: Script Post-scanning.
[*] Nmap: Read data files from: /usr/bin/../share/nmap
[*] Nmap: Note: Host seems down. If it is really up, but blocking our ping probes, try -Pn
[*] Nmap: Nmap done: 1 IP address (0 hosts up) scanned in 4.54 seconds
[*] Nmap: Raw packets sent: 8 (304B) | Rcvd: 1 (68B)
msf >
```

Figure 4. Example of Metasploit Armitage GUI Interface.

The MSF also provides a testing platform called Metasploitable, a Linux-based virtual machine. This virtual machine is designed to be a test platform that a user could use to test stock or custom developed MSF modules against [1]. Although it is a functional test environment, Metasploitable lacks the ability to be rapidly customizable to add any components that would be needed to test against specific hardware, operating systems, network configurations, or embedded intrusion detections systems.

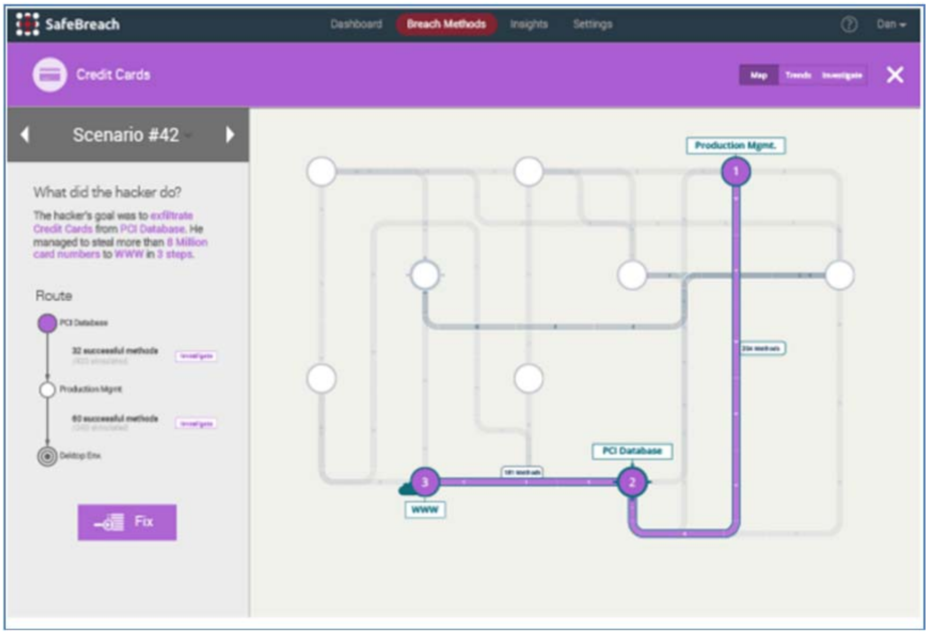
Overall, MSF provides a powerful architecture for penetration testing; however, the testing environment that comes with the framework is not highly customizable nor does it provide a method to simulate an intrusion or host detection systems within a network construct. Additionally, the MSF modules are not designed to simulate exploits and payloads but to be an actual instantiation of said payload/exploit, which differs from a MAST type instantiation of modules that utilize SimWare for their attack scenarios while the MAVNATT system can utilize its generic virtualization capability to produce a customized simulated testing environment.

2. SafeBreach

SafeBreach is a software penetration testing tool that utilizes SafeBreach's proprietary collection of known offensive network penetration techniques, their "Hacker's Playbook™," via an automated penetration-testing algorithm it continuously attempts to penetrate a target network throughout a customer's network [20]. Their algorithms are then able to identify the vulnerabilities in the target system and suggest remedial actions to a network administrator or the penetration tester. The tool can also simulate infiltration methods across the entire cyber-attack kill chain (i.e., Reconnaissance, Weaponization, Delivery, Exploitation, Installation, Command and Control, Actions on Objectives), and provide continuous validation of risks while staying up-to-date with the most current black hat methods and techniques [21].

SafeBreach uses a two-tiered architecture to administer its penetration-testing services. The first tier consists of the software simulators that are deployed on a customer's network which attempt to establish connections to network devices [20]. These simulators are analogous to the MAST clients described in Section C of this chapter. SafeBreach utilizes a cloud-based service, the Orchestrator, which contains the proprietary algorithms that control and direct the simulators based on their feedback from connection attempts and other actions. The algorithms then determine what penetration tactics to try next. Similar to the MAST architecture, the SafeBreach Orchestrator performs the combined functions of the SGS and SES servers in MAST as detailed in Section C of this chapter. Additionally, SafeBreach system developers frequently update their algorithms with the newest black hat methods as they are discovered to provide a more robust and up-to-date product [20], [21].

A graphic example of how the SafeBreach system can identify a vulnerability and show the network route of penetration to the point of data compromise [20] is shown in Figure 5. In this figure, a particular breach scenario has been executed from their Hacker's Playbook by the controlling algorithms of the Orchestrator. Between each numbered pivot point, SafeBreach indicates how many exploits were tried and how many exploits successfully overcame the security at that point and an attempt to graphically depict the steps and exploits an attacker could use the penetrate this system. SafeBreach also provides suggestions about how these vulnerabilities it found can be remedied [21].



What did the hacker do?

The hacker's goal was to exfiltrate Credit Cards from PCI Database. He managed to steal more than 8 Million card numbers to WWW in 3 steps.

Figure 5. SafeBreach GUI Identifies Route of Attack. Source: [20].

SafeBreach differs from some other penetration-testing tools, such as MSF, in that it offers continuous testing of the networks over time utilizing intelligent algorithms to direct the attack, as compared to a snapshot view provided by other static or batch processing oriented software penetration testing tools. This continuous validation and retesting through various network and software upgrades, patch installs, and other network changes that could introduce new vulnerabilities is one of the main features of this product. Pricing of the system is based on the number of simulators deployed. The base level contract of ten simulators with a service agreement for one year is $50,000 USD [20].

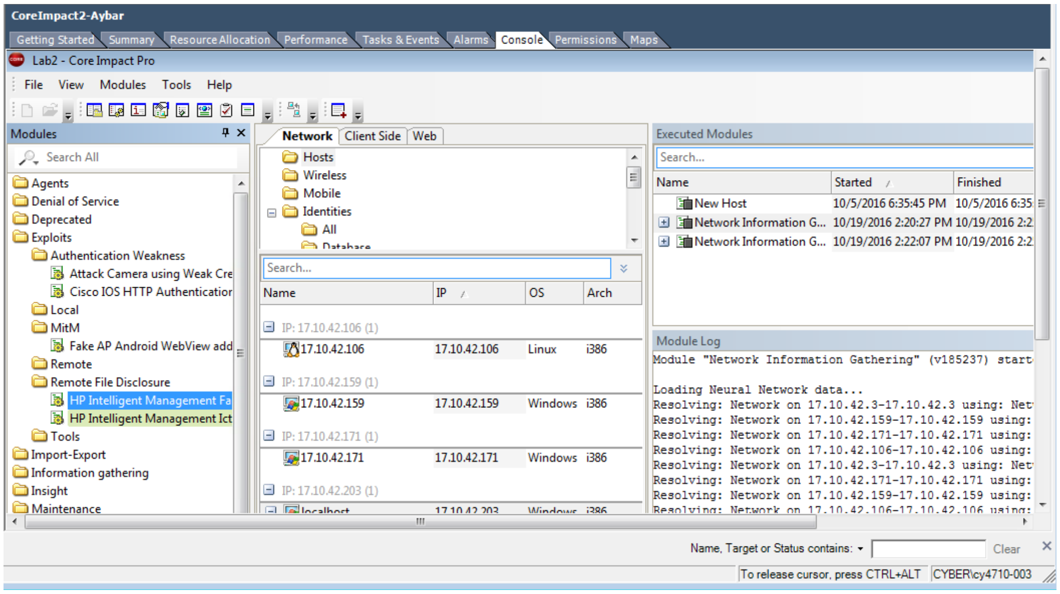
The SafeBreach system also uses models to represent the behavior of malware to increase the accuracy and realism of their testing process. By modeling the malware in much the same way as a vaccine uses an inert copy of a virus, SafeBreach stimulates the defenses of the host system to verify a proper reaction. This malware modeling is another similar concept that is utilized by the MAST framework [21]. However, the SafeBreach platform does not lend itself to the development of wholly new attack scenarios or exploits, and neither does it offer a way for these exploits to be tested in a rapidly configurable and benign test environment.

3. Core Impact

Core Impact from Core Security is a software framework that provides vulnerability assessment and penetration testing of target networks. Core Security has been providing an ever-increasing, sophisticated suite of tools that provides network detection, vulnerability mitigation, and access management penetration testing and vulnerability assessment since the late-1990s [18].

Their flagship product, Core Impact, is able to execute a suite of automated penetration tests across all avenues of access to a user’s systems—network, web, and mobile—and identify areas in need of security remediation. It is also able to enhance the effectiveness of perimeter defense and anti-virus systems, while ensuring detected vulnerabilities are remediated through continuous retesting. It provides a GUI to facilitate the system testing with their provided exploits. An example of the Core Impact Pro GUI is shown in Figure 6, which depicts the modules or attacks available on the left-hand side. In the center panels of the GUI, the network and hosts options are selected and show the discovered hosts in the test network in the panel below. The executed modules and their associated log information are displayed in the left-hand panels.

Core Impact also allows for dynamic integration with third-party products, such as the Nessus vulnerability scanner, to increase its range of capabilities. Core Impact is used by global enterprises such as Credit Suisse, MasterCard, and Dell EMC. A standard deployment of the product can be purchased for approximately $40,000 USD [22].



Name	IP	OS	Arch
IP: 17.10.42.106 (1)			
17.10.42.106	17.10.42.106	Linux	i386
IP: 17.10.42.159 (1)			
17.10.42.159	17.10.42.159	Windows	i386
IP: 17.10.42.171 (1)			
17.10.42.171	17.10.42.171	Windows	i386
IP: 17.10.42.203 (1)			
localhost	17.10.42.203	Windows	i386

Name	Started	Finished
New Host	10/5/2016 6:35:45 PM	10/5/2016 6:35
Network Information G...	10/19/2016 2:20:27 PM	10/19/2016 2:2
Network Information G...	10/19/2016 2:22:07 PM	10/19/2016 2:2

```
Module "Network Information Gathering" (v185237) start

Loading Neural Network data...
Resolving: Network on 17.10.42.3-17.10.42.3 using: Net
Resolving: Network on 17.10.42.159-17.10.42.159 using:
Resolving: Network on 17.10.42.171-17.10.42.171 using:
Resolving: Network on 17.10.42.106-17.10.42.106 using:
Resolving: Network on 17.10.42.3-17.10.42.3 using: Net
Resolving: Network on 17.10.42.171-17.10.42.171 using:
Resolving: Network on 17.10.42.159-17.10.42.159 using:
Resolving: Network on 17.10.42.106-17.10.42.106 using:
```

Figure 6. Core Impact Pro User Interface

Like the MAVNATT network mapping tool, Core Security provides another product called Core Insight, which has a virtual machine that is able to rapidly map a network environment. It compares the mapped information against an extensive list of vulnerabilities to find the ones that absolutely need remediation. This product costs $66,000 USD for the virtual machine, which also comes with the support of two remote auditors [23].

Although, the suite of Core Security products provides an array of useful features, the ability to wholly customize or create unique attack scenarios is not one of them. It also lacks an organic test environment (virtualized or actual) as it is designed to test, scan, and execute against a user-specified network.

4. Simulation Modeling

Another approach used by Kistner, Kuhl, Costantini, and Sudit, researchers at Rochester University, was to model computer networks, intrusion detection systems (IDS), and the behavior of the network as an alternative way to simulate cyber-attack scenarios [24]. Their approach requires knowledge of the system or network being tested to construct an accurate simulation model, and uses the Arena® discrete event simulation software to demonstrate the concepts. IDS sensors are placed throughout the simulated network to produce real alerts based on the type of network traffic they observe in keeping with the realism of the simulation. Their model allows the user to construct a simulated test network and then devise, build, and execute various cyber-attack scenarios against this network via a GUI. The attack scenarios can be generated manually or automatically, and allowing for the specification of the path through the network as well as timing of the attack. Parameters for stealth, efficiency, and skill of the attack may be set to adjust the behavior of the cyber-attack scenario. Once the cyber-attack scenario is created, it can be saved and run on different computer network simulations [24].

This simulation model was designed primarily for testing cyber-attack scenarios, but it also can be used by system administrators to conduct testing of various cyber-attack methods against a simulated model of their own networks. An example of a simulation model in the Arena software GUI is shown in Figure 7. The ability to create custom devices in the Arena software to model machines, connectors, and subnets allows for the quick setup of customized computer networks used for testing of the cyber-attack scenarios.

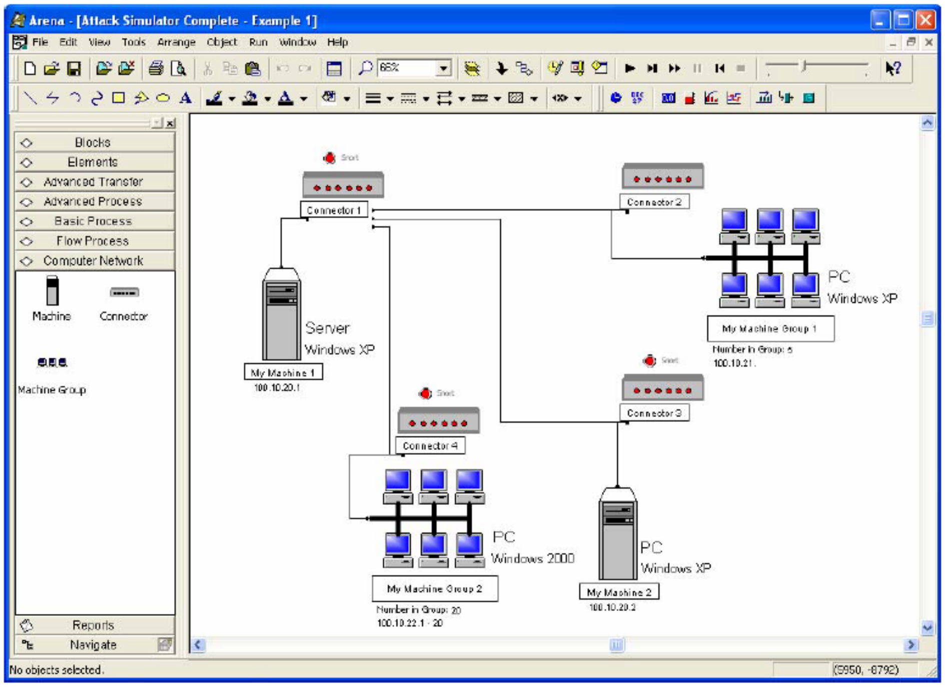


Figure 7. Sample Network Model with Arena Interface. Source: [24].

The cyber-attack simulation model produced by these researchers addresses the full range of criteria we were reviewing in the various commercial tools. The simulation tool also allows a user to customize additional behaviors to mimic various skill levels of an attacker such as stealth, skill, and efficiency to various levels while setting up the attack scenario [24]. However, this was only a research project performed in 2007 and was detailed in the paper “*Cyber Attack Modeling and Simulation for Network Analysis*.” Although the tool is not available for purchase, it does reveal an ingenious method to create cyber-attack simulations using an unrelated software tool as a foundational base.

5. STEPfwd

The STEPfwd (Simulation, Training and, Exercise Platform) platform is an online cyber/information assurance workforce development tool developed by the Computer Readiness Emergency Team (CERT) at the Software Engineering Institute at Carnegie Mellon University, a federally funded research and development center .

As a training tool for cybersecurity personnel, STEPfwd has a robust capability to provide virtualized training simulations that closely mimic real-world infrastructures and attacks [25]. Two proprietary technologies used by STEPfwd to enable this robust simulation environment are Carnegie Mellon's Software Engineering Institute's Virtual Training Environment (VTE) and CERT's Exercise Network (XNET). The VTE facilitates the knowledge and skill building phase of the training encounter by providing online access to cybersecurity material in the form of technical demonstrations, lecture slides, and written materials [25]. XNET is a centrally managed infrastructure platform that allows remote instructors to create customized, full-scale cyber exercise scenarios to simulate real-world environments. Users are also able to customize scenarios by creating their own network environment, events, and timeline with the added ability to inject attacks/anomalies, create robust traffic generation and modify the timeline or event library. The XNET console also supports the participation of multiple cyber operators from different locations on the same scenario via its cloud-based architecture to facilitate real-world interactions between participants on a complex interactive scenario while creating an experience-building event. [25].

The designers of the CERT ecosystem also had to address the scalability issue regarding the storage requirements (memory & disk) that several hundreds or thousands of Virtual Machines (VM) all running simultaneously would consume using the standard paradigm of taking snap shots. They developed a customized virtualized solution that modified the snapshot process of the hypervisor to use a single base disk to produce other needed instances of guest VMs [26]. For example, this method can use a single base disk image of a Windows 2012 server and create a snapshot to instantiate a domain controller, an Exchange server, and a web server and so on while staying within their hypervisor's storage limits. An illustration of this virtualized solution and the savings that can be realized in disk storage and server memory by using one base disk image versus the traditional method of snapshots creating multiple restore points for a single image is shown in Figure 8.

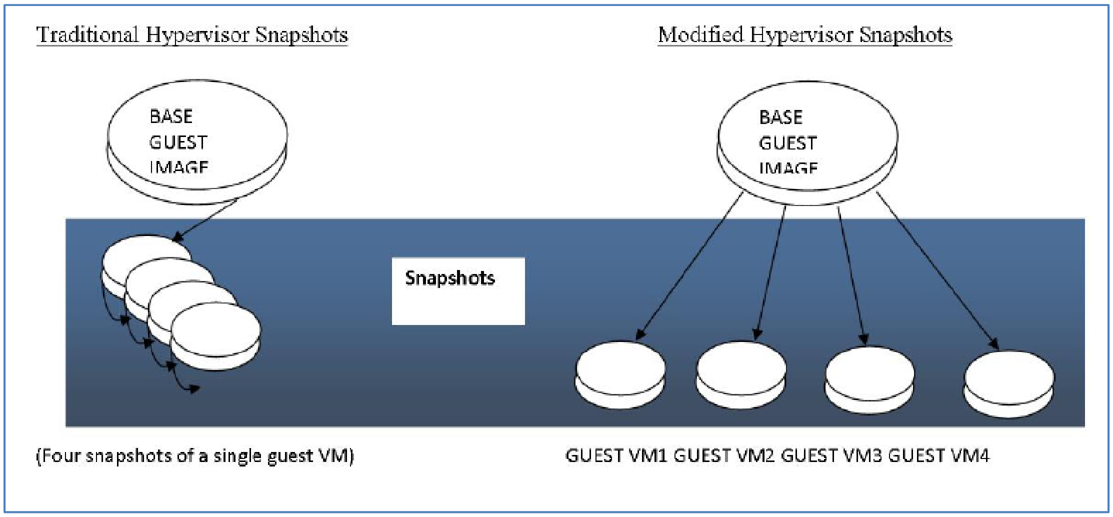

Traditional Hypervisor Snapshots
Modified Hypervisor Snapshots
BASE GUEST IMAGE
BASE GUEST IMAGE
Snapshots
(Four snapshots of a single guest VM)
GUEST VM1 GUEST VM2 GUEST VM3 GUEST VM4

Figure 8. Design implementation for STEPfwd and XNET. Source: [26].

This allows for the creation and startup of an entire exercise environment in just a few minutes with a few base disks, installation scripts and a startup sequence on one or more servers as needed. This architecture forms the core of the VTE and the SEI's CERT STEPfwd system [26].

In a 2009 U.S. Air Force Cyber Operations defensive network training event, CERT was able to provide a scenario using the XNET platform that "encompassed the anatomy of a real attack—reconnaissance, botnet and malware staging, data exfiltration, and massive communications disruption—and involved more than 100 virtualized computers and infrastructure devices" [25]. This example highlights the range of capabilities of the STEPfwd system with regard to the development of cyber-attack scenarios. If the emphasis were put on developing virtualized cyber-attack scenarios and using real defensive-minded operators in the testing process, this tool could be a very powerful instrument in the development of offensive cyber weapons. As an online or cloud-based solution, it provides tremendous scalability and allows for collaboration between geographically disparate teams and team members.

The STEPfwd and XNET systems provide a range of capabilities that could be used for the development of simulated attack scenarios against virtualized adversary networks. Specifically, they provide the means to develop complex attack scenarios, a virtual environment to execute these scenarios within, and an ability to monitor the scenario as it progresses. However, as a virtualized online solution, these systems are not able to map new networks, nor can they be used in an operational environment, as MAVNATT can.

6. Summary of Existing Tools and Methods

The purpose of this survey was to examine some software tools that are available, both commercially and custom developed, to ascertain what exists currently and if an existing product could meet the requirements to serve as the foundational platform for the development of simulated cyber-attack scenarios. The evaluation criteria, as described in the beginning of the background section, were used to determine which of the surveyed products would be best for the foundational base of the thesis research.

The commercial products examined demonstrated a strong portfolio of features; however, the Safe-Breach and the Core Security products, although very robust, lack the ability to design unique attack scenarios. Conversely, the MSF does allow for the design and development of user designed exploits, modules, and scripts to be created, implemented, and utilized against a user-specified target network. MSF also provides an exploitable Linux ISO (a CD-ROM image saved in ISO-9660 format) for use within a virtualized environment to experiment with newly created or existing exploits.

Although not as mature or robust as the surveyed commercial products, MAVNATT does more closely align to the original evaluation criteria of providing a software framework that provides the ability to create custom cyber-attack scenarios, an organically generated virtual testing environment wholly separated from the live network, and a method to observe the resulting behavior in the testing environment. MAVNATT, as part of its “*train*” mission, was designed to be able to interact with the MAST framework via Virtual Machine Disk (VMDK) files allowing the MAVNATT Virtualization Module to render an interactive session with a virtualized MAST instantiation [9]. This capability of MAVNATT makes it an ideal foundational platform for the stated goal of this thesis to develop simulated cyber-attack scenarios for use against virtualized adversary networks. A comparison of various features of the tools and frameworks examined in this chapter is shown in Table 1.

Table 1. Commercial Tool Summary

	Metasploit	Core Impact	Safe Breach	Arena	STEPfwd
Host OS Platform	All Primary OSes	Win OSes	All Primary OSes*	Win OSes	Web based
Custom Scenarios	Yes	No	No	Yes	Yes
Organic Test Env	Metasploitable	No	No	Yes	Yes
Cost	Free	~ $40,000	~ $50,000	$90/seat/month	Variable
API	Yes	Yes, 3rd Party	No*	No	No

F. SUMMARY

This chapter provided a high-level overview of MAVNATT and its core elements, and examined the architecture of MAST. Additionally, it provided a survey of some of the more widely used commercial products for penetration testing to ascertain the suitability of these products as a base element for the work of creating simulated cyber-attack scenarios. The next chapter provides a thorough examination of the necessary elements for developing robust and realistic simulated cyber-attack scenarios.

THIS PAGE INTENTIONALLY LEFT BLANK

III. DESIGN AND METHODOLOGY

A. OVERVIEW

Effective design, implementation, and testing of simulated cyber-attacks require a number of components, to include the following: knowledge about the target network, requirements of the virtualization platform, and a method to handle imperfect knowledge about the target system or network. We examine each of these components in detail, as well as the different interaction parameters for simulated cyber-attack software, and the implementation strategies for modeling such methods to mimic real-world network interactions. Finally, we outline the key requirements of a simulation environment necessary for planning, developing, and rehearsing offensive cyber operations.

B. VIRTUALIZED ATTACK SCENARIO NEEDS

The existing software products and methods reviewed in the previous chapter demonstrate an array of capabilities that could be used for developing and simulating offensive cyber-attack scenarios. Modeling cyber-attacks in a simulated or virtual environment, however, requires additional components and information, such as detailed knowledge about the network that will be attacked, and a virtualization or hardware-testing platform that can replicate the target network to a very high degree of fidelity. The size of the network to be modeled is also an important data point since it will affect the hardware solution used for virtualization. Furthermore, if the cyber-attack scenario is to simulate effects in the physical world (e.g., by acting on supervisory control and data acquisition (SCADA) systems), those systems and their associated hardware must be modeled accurately to achieve relevant results. Finally, knowledge of the adversary network will likely be imperfect, which can introduce uncertainty into the derived cyber solution

1. Network Knowledge

Creating a simulation that would have a high probability of success against a real-world system first demands knowledge of the target network. At its core, offensive cyber operations uses the network knowledge to determine a vulnerability that can be subsequently exploited. The *knowledge of the network* consists of some, if not all, of the following information: network specific (IP addresses, network topology, domain names), host specific (user names, architecture type(s) (e.g., x86), OS variant and version, services running, ports open/closed), and security specific (firewalls, IDS/HBSS running with associated detection methods, password complexity requirements and change frequency), and physical security of the network (e.g., locks, keycards, hardened facility) [27]. This knowledge allows the offensive-minded operator to bypass target defenses and enter the system to achieve the mission objective(s). Knowledge of the network vulnerabilities can be achieved through active or passive computer scanning, and intelligence gathering through traditional sources of human, signals, and open source methods.

To acquire knowledge about an adversary network via computer means would require following the process that white- or black-hat hackers use when gathering intelligence on a network. This process is referred to as the hacking phases [28], and is used to glean information about the target system before a tactical cyber-attack scenario is devised. Two well-known models, which outline the phases, are described by the Certified Ethical Hacker (CEH) association and by Stuart McClure's "Anatomy of a Hack" in the book *Hacking Exposed* [29]. The two models share the same objective but differ slightly on terminology and number of steps in their respective approach. The general steps of these hacking phases are as follows:

- Reconnaissance
- Scanning
- Gaining Access
- Maintaining Access
- Covering Tracks.

The first two steps are common to most models, although the term *footprinting* may be used instead of reconnaissance. The need to find out about the target system, its defenses, its software variants and versions, and its outward facing configurations are the objectives of these initial steps. Reconnaissance is a preparatory phase in which information is gathered about the target organization, its network, employees, and operations. This step can be performed using passive methods such as searching through publicly available records and data, or with more active methods that involve interacting with the target directly. Additionally, traditional methods of espionage (human and signals intelligence) can also be employed to gather information or to recruit and use human sources with placement and access to gain greater knowledge of an enemy's systems.

A more aggressive form of network discovery involves scanning the adversary network to reveal an array of logical and physical information about the target systems. The information gathered through scanning could include operating system software that is being run by the target machines, a list of active hosts, the status of computer ports on the hosts, services that are running, and the network topology. A number of popular commercial tools can carry out these scans, such as Nessus [30] and Zenmap [31].

The knowledge gained about an adversary network, its services, applications, and vulnerabilities during the reconnaissance and scanning phases enables the subsequent steps in the hacking process. Moreover, this information is critical for creating an accurate network model on which to simulate and test cyber-attacks. The network information that could not be ascertained, be it operating system software on some hosts or proprietary software on a router, would require assumptions to fill the gaps. These assumptions could significantly weaken the efficacy of the network model, which reaffirms the importance of obtaining knowledge on the adversary network.

2. Virtualization Platform

A key requirement of the virtualization platform would be the capability to model a large selection of computer and network devices accurately for the test environment. This modeling of computer devices could be done via a physical instantiation of the target network or be completely virtualized. For the former alternative, cyber test ranges have been built by the DOD at a number of different sites, such as the National Cyber Range (NCR) and the Navy Cyberspace Operations Range (NCOR). A detailed discussion of the capabilities of these cyber test ranges can be found in the thesis by Nathaniel Hayes, “A Definitive Interoperability Test Methodology for the Malicious Activity Simulation Tool (MAST)” [32]. For this research, we focused on virtualized test environments, which could provide a portable and customizable solution to simulating an adversary network for offensive cyber operations.

A hypervisor is commonly used for developing a network of virtualized hosts. Hypervisors normally work in one of two ways. One method is to run the hypervisor directly on the host machine’s firmware (Type I / bare metal*)*; the other is to layer the virtualization application on top of the host machines OS (Type II / hosted) [33], [34]. Both implementations are used to allocate the physical resources of the host machine for the virtual machine instantiations. MAVNATT utilizes a Type II hypervisor for its virtualization module, while SEI’s XNET uses a Type I hypervisor implementation [9], [25]. The hypervisor implementation shown in Figure 9 illustrates a Type I bare metal implementation on the left, and a hosted (Type II) model on the right. In either case, the virtual machines always interface with the host machine through the hypervisor, never directly to the host machine’s hardware or OS.

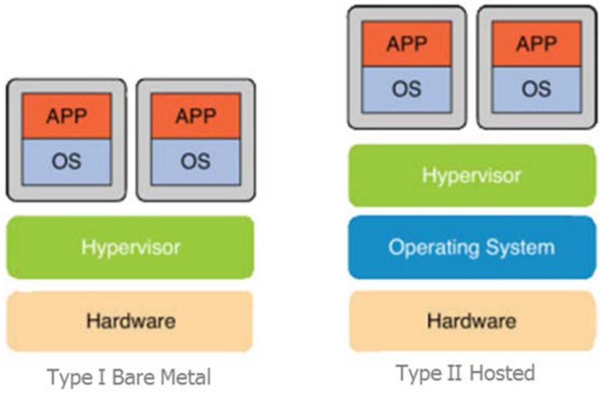


Figure 9. Illustration of Different Hypervisor Implementations. Source [34].

The incorporation of device-specific information gathered about the adversary network into the virtualized environment is vital. This requires a virtualization capability that could instantiate hosts, connectors, subnets, and routers in a way that allows for the input of the device’s defining characteristics, as captured during the intelligence gathering phase (active services, open/closed ports, operating system software, and software patch levels). Moreover, the virtualized network environment should also be able to simulate regular network traffic and noise in the test environment in order to simulate the realism of a physical network. The noise would provide the additional network traffic that could cause bottlenecks, reduced availability of network services and other congestion issues that the simulated malware should be able to overcome.

Another important requirement of the virtualized network is that it should properly simulate the network defense mechanisms (i.e., IDS, HBSS, firewalls and others). These network defenses should be configurable, just like they are on a physical network, and should respond to proper provocations from SimWare. These simulated network defenses should also employ a range of common detection algorithms, which should include the following: signature-based, host-based, and hybrid variants of the two.

Finally, the ability to support an omnipotent administrator role with multiple simulation participants is a critical need to support a large-scale simulation environment testing complex cyber scenarios. Multiple defenders or offensive cyber warriors should be able to collaborate within the virtualized environment while working on the same problem to mimic real-world interactions. Additionally, the scenario administrator would have the capability to dynamically add or remove events from the scenario, such as a Denial of Service (DoS) event, and be able to review each participant’s response(s) to the scenario and the injected events.

3. Scale Factor

The size of an operational network to be virtualized is a key consideration when simulating cyber-attacks. Many modern networks are comprised of hundreds or even thousands of devices, all interacting and communicating according to the activity of individual users and automated processes. The ability to replicate a large network in a virtualized environment would be required in order to conduct accurate simulations of real-world network systems.

For the simulation to have a high degree of accuracy, each individual virtualized object would need to incorporate all of the information acquired about its real-life counterpart. To have this degree of fidelity in the network model would require significant amounts of computing resources. However, to have the capability to conduct network simulations and offensive cyber scenario development in a tactical environment would require some significant tradeoffs in either the fidelity of the model or the size of the network modeled. Each alternative would carry significant risk to the derived cyber solution.

4. Accounting for the Unknown

When developing a virtualized attack scenario, imperfect intelligence of the adversary network must be recognized and accounted for. This awareness of one’s own lack of knowledge, known as “Socratic ignorance” [35], highlights the importance of the unknown elements of the enemy network. Similarly, when trying to accurately model the computer system of an adversary, many factors cannot be known with a high degree of certainty, which could significantly affect the precision of the network simulation. Such factors could include the fluidity of real-world networks, the time lag of patch releases to application, the frequency of penetration tests, advanced proprietary IDSs, active defensive measures, and the variable actions of network operators and defenders.

Real-world networks are frequently upgraded, patched, reconfigured, and penetration tested, creating a rapidly changing target for simulation. This results in incomplete knowledge, and thus imperfect modeling of an adversary network. This shortfall can cause cyber-attack scenarios to be developed and tested in an imprecise simulation environment, leading potentially to failed cyber operations against the enemy network. Furthermore, the perishability or obsolescence of cyber-attack methods is a very real issue in cyber mission planning as most cyber weapons depend upon the exploitation of a vulnerability in the target system to achieve their mission effects [36]. These vulnerabilities could be remedied in the course of an adversary’s normal cyber hygiene yet remain unknown to offensive cyber operator. This could render the cyber weapon based on that vulnerability inert.

Another unknown about the adversary network that generates problems for the creation of an accurate simulation is the type of IDS being used and the type of detection schema employed. The IDS could either be host-based or network-based and use either a signature based or behavior based algorithm or combination thereof. If the intrusion detection scheme cannot be determined with certainty, the simulated cyber-attack solution may not perform as expected in the physical network.

A target's employment of active defensives is also challenging to replicate in a simulated environment. Active defenses refer to proactive cyber security measures that may be taken by an organization to defend its operational networks. These measures could include denial and deception techniques such as tarpits or honeypots [37]. Tarpits are a defensive computer technique used to delay incoming connections with a slow response from the server, while honeypots provide an inviting albeit false target for attackers [37], [38]. Their active defense measures could also consist of hunting teams, who are highly skilled cyber defenders who actively seek out malicious software on their network that has evaded passive security measures such as firewalls and anti-virus scanners [37], [39]. These active defensive methods may be approximated to some degree in simulated environments, but the behavior is difficult to replicate perfectly.

Finally, the skill level and resilience of the enemy cyber defender cannot be known completely. The resilience factor would be difficult to replicate in simulations, as it varies by individual cyber operator, depending on their dedication and work ethic. The skill level of individual operators can result in a large variance across an organization that represents expertise peaks and valleys that can occur over the course of a day, week, or month. Furthermore, there is the changing dynamic of improving skills through advanced training or declining skills through inactivity of the individual cyber defender.

In summary, numerous aspects of operational networks are either highly dynamic or hard to quantify, and therefore difficult to model precisely. There are ways to quantify the skill level or expertise of cyber defenders by defining and implementing interaction parameters into the simulation, but imperfect knowledge of dynamically changing computer systems, and unknown active defenses could introduce a degree of uncertainty into any simulation.

C. INTERACTION PARAMETERS

Creating accurate simulations of offensive cyber operations requires interaction parameters that can define the behaviors of SimWare. These parameters can modify the manner in which the SimWare traverses through the network to accurately mimic real-world behaviors. The interaction parameters can also allow for complimentary behaviors, such as skill and stealth, to be combined to form a new hybrid behavior that affects SimWare differently than either would independently. In the following section, various interaction parameters will be defined and described as to how they could enhance the network simulation.

1. Propagation

Propagation refers to how SimWare will move through the simulated network, either by self-propagation or by some other means. A self-propagating selection would not require another program to act as a conveyor and could leverage network resources and its associated connectivity to spread. Furthermore, the way in which the propagation parameter affects SimWare must emulate standard network routing protocols, and must address its interaction with security measures such as firewalls, network/port address translations, and active defenses.

2. Specificity

Specificity refers to the ability of a SimWare module to discriminate a particular target with a high degree of precision. The Stuxnet virus provided an example of malware that was highly focused on a very particular target, the specific make and model of the Iranian centrifuges used at Natanz, but did not damage any other systems [40]. Specificity would ensure that the SimWare would target a particular media access code (MAC) address, IP address, or SCADA device, while not affecting any other systems on the network. If the SimWare module were not able to find its specific target, it would do nothing or try to move onto another system to keep searching for its target.

3. Timing

The timing parameter provides temporal control of when a SimWare module executes its intended behavior. A simulation might include, for example, a timeline of various SimWare cyber actions, each with timing dependencies such that one action must be completed before another. The ability for the SimWare to act only after a certain time has passed or a particular event has occurred is crucial to realistic modeling. The Y2K bug, although not malware, is a good example of a software problem that was triggered by a specific time event.

4. Stealth

Stealth refers to how well a cyber-attack avoids detection [24]. Conducting a cyber-attack in stealth mode will require extra precautions, and may not necessarily result in the most expedient traversal of a network. For example, maintaining stealth may require traversing through authorized ports that expect encrypted web traffic, such as port 443, or having malware be resident on an existing process or daemon to avoid leaving any trace in memory. These methods, although stealthy, are not the most expedient ways to achieve the desired cyber effect.

5. Efficiency

Efficiency relates to how directly an attack will proceed [24]. A SimWare module with a high efficiency selection would not take steps to avoid detection, traps, or detour around obstacles. High efficiency would be desirable when the shortest time for the malware to achieve its effects is the most important factor. A high efficiency value applied to SimWare would be analogous to walking through the front door to burglarize a house because it is the quickest way to get to the intended target. Although, doing so would result in being caught on the security camera instead of crawling through an open upstairs window which would take more time (lower efficiency), but might avoid active surveillance systems (greater stealth).

6. Skill

This interaction parameter is used to determine the probability that the cyber-attack will overcome the simulated network's defensives and obstacles [24]. In the physical world, a skillful thief can efficiently pick locks, evade security systems, and avoid leaving behind any physical traces of his activities. Similarly, in a virtualized environment, SimWare with a high skill level may have a higher likelihood of evading or bypassing computer defenses.

7. Summary

These interaction parameters form the basis of interaction methods within a simulated attack scenario. These methods could be controlled by allowing the user to select a numerical value from a range with a low value of one to a high value of ten. The algorithms of the simulation would then resolve the parameter governing the interactions. For example, if a SimWare module had a high stealth value (nine) and a high skill value (seven), then it should be able to overcome most defenses while remaining hidden. The simulation software would also need to modify the propagation rate, and interaction results based on the input selections of the modifiers.

D. SIMULATED ATTACK REHEARSAL ESSENTIALS

As cyber operations become more complex and more highly integrated with kinetic operations, the need for better collaboration and rehearsal between cyber and physical domain operators becomes increasingly critical. A visual and sequenced run through the plan with all the participants used to enhance understanding is a common technique in the physical domain known as a Rehearsal of Concept (ROC). For a virtualized network to be used effectively in ROC drills, a number of capabilities are necessary. These include the ability of the test environment to do the following: to reset and reconfigure the environment rapidly, provide a large menu of configurable SimWare options to achieve various effects, set timelines for activities and their execution, incorporate intelligence on vulnerabilities of the adversary's network or system, and an ability for autonomous testing of the simulation.

1. Reset and Reconfigure

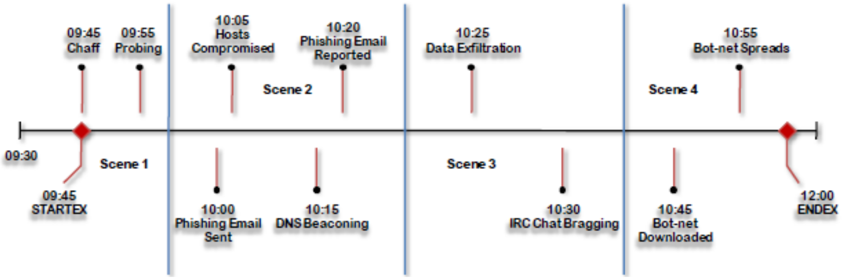
For a virtualized network to be used effectively for testing offensive cyber operations, its simulated environment should be resettable and reconfigurable quickly and efficiently. This would include the capability to do the following: add or remove components to the simulation, modify components with new parameters that take effect immediately, save simulation run results with used network parameters run, and capture the behaviors of each simulation participant.

2. Menu of SimWare

Another important factor for OCO planning and rehearsal is the ability to quickly select and utilize SimWare modules from a menu. The ability to match capabilities to intended effects is referred to as *weaponeering* in the military lexicon [41]. Once the desired cyber effects have been determined from earlier planning efforts, achieving those effects will depend upon the ability to rapidly test various cyber-attack capabilities on a simulated adversary network. Moreover, new capabilities may need to be developed if repeated testing proves the existing tools are not able to achieve mission objectives. A detailed SimWare menu allows a cyber operator to quickly scan through the available list of tools to find a particular tool, or combination of tools, that could exploit the identified vulnerability and create the desired effect in an expeditious manner.

3. Timelines

Being able to see a timeline of various events in the cyber-attack simulation is a key capability for planning and rehearsing offensive cyber-attacks. As a scenario becomes increasingly complex, with multiple participants and interdependent effects, the need to precisely coordinate the different elements and effects will become more pronounced. The ability to add SimWare behaviors and events onto a timeline would be indispensable to structuring advanced time-interdependent simulation scenarios. The SimWare menu could be configured to add or remove various events and attacks on a timeline to aid in the planning of cyber-attack simulations. An illustration of how such a timeline might be presented to the user is shown in Figure 10.

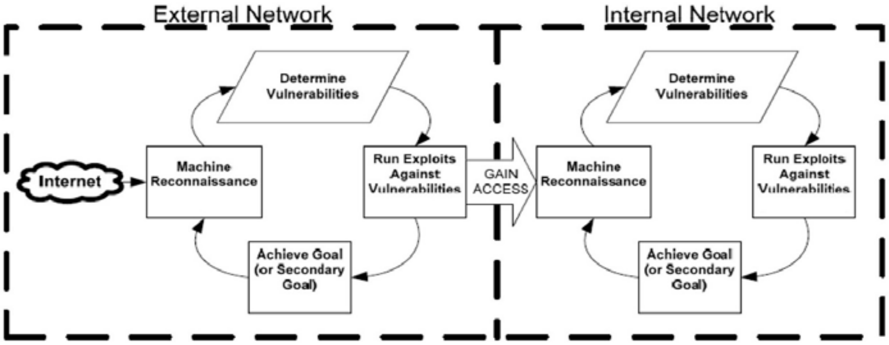


A visual script that depicts cyber events from STARTEX to ENDEX.

Figure 10. Visually Scripting the Timing of Various Cyber Events. Source [42].

4. Vulnerability Intelligence

Cyber-attack scenarios are commonly focused on target network vulnerabilities determined during the information-gathering phase. To conduct a successful cyber-attack against an adversary network will require knowledge of identified or high-probability vulnerabilities. By utilizing the hacking steps described earlier in Chapter III and exploiting the determined vulnerabilities, we can start from the public-facing Internet to compromise the external network, and then repeating the process to penetrate the internal network, which is depicted in Figure 11. For both the external and the internal network, the vulnerabilities facilitate the system breach. Having an external public-facing presence and a private internal enclave, as shown in Figure 11, is a common design for many secure networks. Different variations of a cyber-attack scenario can be rehearsed based on the desired mission effects, the vulnerabilities identified, and the cyber tools available.


External Network
Internal Network
Internet
Determine Vulnerabilities
Machine Reconnaissance
Run Exploits Against Vulnerabilities
Achieve Goal (or Secondary Goal)
GAIN ACCESS
Determine Vulnerabilities
Machine Reconnaissance
Run Exploits Against Vulnerabilities
Achieve Goal (or Secondary Goal)

Cyber-attack scenario traversing from Internet to External network to internal network using hacking phases.

Figure 11. An Overview of a Cyber-Attack Scenario. Source: [24].

5. Automated Testing

When developing cyber-attack scenarios with imperfect intelligence of the target network, the capability to perform automated testing would be especially valuable. Defensive measures that prevent scanning of the target network can obfuscate network host parameters such as operating systems, active services, and open ports. Iterating through all the possible variants for these unknown factors could result in thousands, if not millions, of different network permutations.

An automated testing capability in the virtualized environment could provide the ability to execute the prescribed cyber-attack scenario against a virtualized adversary network while each time varying a parameter of the adversary network. The automated testing could relieve the cyber scenario developer from the precarious task of making assumptions for the unknown attributes of the adversary network and extrapolating the results of a limited number of test runs into a confidence factor for the derived cyber solution. Some of the commercial systems examined in Chapter II exhibit some automated testing features. The MSF allows a *Hail Mary* style attack where every exploit in the framework repository can be launched at a particular host or network without the need to configure each individual attack. SafeBreach also supports a similar penetration test mode that will continuously attempt to penetrate a customer's network in attempt to provide real-time awareness of any new vulnerabilities.

As both the networks to model and the number of unknowns of the enemy network grow, the ability for automated testing would become an essential element of a robust cyber scenario development platform. In the recent Defense Advanced Research Projects Agency (DARPA) Cyber Grand Challenge (2016), seven autonomous machines competed in a capture-the-flag computer hacking competition in Las Vegas, NV [43]. In capture-the-flag, teams are responsible for protecting their server data (the flag) while trying to find, diagnose, and fix their own vulnerabilities. At the same time, they attempt to exploit another team's vulnerabilities to capture their flag. For the first time ever, these algorithms competed at this human-centric game in 96 rounds of 270 seconds each. The algorithms developed 421 replacement binaries and detected 650 vulnerabilities, some of which were unknown to the challenge creators [44]. This challenge demonstrated that an automated, machine-speed, vulnerability detecting and patching system is a real capability [45]. These algorithms could form the basis of an automated testing capability for a DOD offensive cyber scenario development platform to aid with the testing of derived cyber-attack solutions against many permutations of an enemy network.

With the ability to run many thousands of tests of the proposed cyber scenario solution against many permutations of the enemy network, a large array of test results would become available. These results could provide the basis for an analysis, which could determine which combinations of variables from the enemy network and the derived cyber solution resulted in success. Using the resulting analysis, a cyber scenario author could associate a high confidence factor to those cyber solutions that achieved a greater degree of success against multiple variations of an enemy network.

E. ATTACK TYPES

A primary goal of this thesis is to define and develop the capability to simulate cyber-attacks. A key component of this capability is to identify the various attack categories available. Cyber-attacks can be classified in different ways based on their methods, capabilities, and various attack vectors, but in general they can be grouped into four main categories: reconnaissance, access, denial-of-service, and data manipulation [46]. These categories enable attacks on all components of computer system security, where confidentiality represents the prevention of unauthorized viewing, integrity refers to the prevention of unauthorized alteration, and availability is a measure of the operational readiness of a computer system and its data when needed.

1. Reconnaissance

Attacks in this category focus on information gathering. The information acquired does not directly compromise any part of a computer system’s security, but is a necessary step to enable the other categories of attacks. Reconnaissance uses open source intelligence (OSINT) gathering methods to leverage publicly available information about a target network and its systems through overt collection. This may include such methods as accessing public records on people associated with the target, or researching

Internet Protocol (IP) registration information through the American Registry of Internet Numbers [46].

Another aspect of reconnaissance is electronic intelligence gathering on the target network, using network and port scanning techniques such as nmap or traceroute. The data gathered might consist of connectivity of the various hosts on the network, the software environment running on each host, status of server and security settings (e.g., which ports are open or blocked), network protection software (e.g., HBSS), and computer architecture employed (e.g., scalable processor architecture). The types of methods used, and the data acquired, are similar to information gathering techniques mentioned earlier in Chapter III (Section B.1 Network Knowledge), however the topic is readdressed here as it is a relevant part of the attack taxonomy when developing cyber scenarios.

2. Access

Access-based attacks include methods that focus on gaining or elevating privilege to unauthorized computer resources [46]. A user attempting to gain privileged access could be an individual or part of a group external to the target organization, or it could be a trusted insider attempting to gain higher authorization than his privileges allow. Most computer systems today use a password-based identification and authentication system as a first line of access protection. Using password-based attack methods to determine user passwords via brute force, trying default passwords, or cracking a password file are among the techniques used to circumvent this first level of defense [47].

Social engineering and phishing techniques are used by attackers to gain privileged access to a target system. Social engineering techniques attack authorized users by presenting a plausible ruse or questions that persuade the user to divulge their password or other sensitive network data [48]. Phishing is a particular form of email-based social engineering that attempts to entice the authorized user to innocently access a malicious web link that will install malicious software on the user system, providing system access for the attacker. This class of attacks compromises a computer system’s confidentiality by overcoming access restrictions and providing key benefits that would be advantageous in the development of most cyber-attack scenarios.

3. Denial of Service

Denial of Service (DoS) attacks compromise the availability of computer systems to their intended users. DoS attacks can be accomplished in a number of different ways. A common method is to flood a targeted network or system with requests so that the target cannot provide service to any valid user. A more aggressive form of denial uses many computers to overwhelm the targeted network with traffic. This distributed denial of service (DDoS) can be effected using hundreds or thousands of compromised hosts, known as a botnet, that are controlled by the attacker's system [46].

Another DoS attack surface uses methods that cause the target computer or its software not to function correctly. A buffer overflow attack exploits software flaws that allow an input field or parameter in a program to accept too large data, which then overwrites legitimate computer data in the software's memory space [48]. By overwriting a portion of memory called the *execution stack* with invalid inputs, or more seriously, illegal instructions or malicious code, the attacker can cause the system to crash or allow an attacker to gain control of the host. This technique can lead to a DoS to the targeted computer system.

4. Data Manipulation

These types of attacks can compromise the integrity of computer systems' data. These attacks often exploit vulnerabilities in networking communication protocols [48], using techniques such as Internet Protocol (IP) spoofing, man-in-the-middle, and session hijacking and replay attacks. The ability to alter data to/from a target computer system provides a cyber-attack scenario an array of capabilities. For example, commands to a SCADA device could be altered to elicit a range of desired effects (e.g., shutdown, reset, or ignore safety limits), as could modified orders/messages used for command and control purposes.

Internet protocol spoofing allows an attacker to deceive defensive packet filtering schemas by impersonating another host [46]. Most packet filtering and computer identity recognition protocols use IP addresses to determine a packet’s source and destination. By manipulating the IP addresses in the packets, attackers can conceal their identity, bypass defenses, and introduce malicious data or commands into a system thereby compromising its integrity.

Man-in-the-middle attacks occur when an attacker is placed between the source and destination of a network communication [48]. From this vantage, attackers can see and modify network traffic between the endpoints, unbeknownst to the participants. Similarly, session replay is related to the man-in-the-middle attack, and is used by attackers to manipulate captured packet data and then replay it (e.g., in a banking transaction scenario) for a number of different effects [46].

F. SUMMARY

This chapter detailed capabilities needed to create a virtualized environment to conduct meaningful OCO testing. It examined the key factors required to create a virtualized scenario, as well as the interaction parameters needed to provide greater adherence to real-world malware behaviors. Finally, a list of attributes that are essential for the virtualized environment to be used for OCO planning, and rehearsal were derived. The next chapter describes the implementation of the derived interaction parameters into software modules, the virtualized test environment, and the methods used to test the new modules.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. IMPLEMENTATION

A. OVERVIEW

This chapter describes the approach used to implement advanced attack scenarios according to the requirements outlined in Chapter III. The interaction parameters described were the basis for new SimWare modules designed to test interesting types of cyber-attack scenarios. These scenarios were tested using MAST, which was the console used to launch the offensive cyber-attack modules, while a virtualized network provided the training area. A description of the test environment, the setup of the scenario files, and the attack modules developed are detailed in this chapter.

B. TEST ENVIRONMENT

The test environment is comprised of three important segments: the virtualized network infrastructure, the software tools used for execution of the cyber-attack scenarios, and the virtualized target hosts. All were important to designing and testing the cyber-attack scenarios, which were based on the derived interaction parameters and coded into the test attack modules. A detailed description of each of the segments follows.

1. Virtual Network Infrastructure

The test environment consisted of a virtual network infrastructure, using a Type I hypervisor implementation. The NPS Computer Science Department’s Cyber Battle Lab (CYBL) was used for the testing environment. This lab provided numerous benefits that facilitated the scenario testing, such as an infrastructure that was professionally managed, sufficient computer resources that allowed for the simulation of a sizeable network, and an isolated testing area to prevent collateral damage to any operational computer resources. This allowed the use of virtual hosts that had a high degree of configurability, a key requirement for the accurate modeling of an adversary network. vSphere, a server virtualization platform by VMware [49], served as the interface tool for the virtual environment.

A private network was also created and configured within the CYBL virtual space to allow communication between hosts on the subnet 10.1.99.1 – 10.1.99.255. Each host had a unique IP address, and required its own instance of the MAST client to be run locally. The network architecture was intentionally kept flat, with no additional subnets, to allow better examination of the behaviors of the new modules, and to avoid additional complexity that could have obscured the testing results.

2. Software Tools

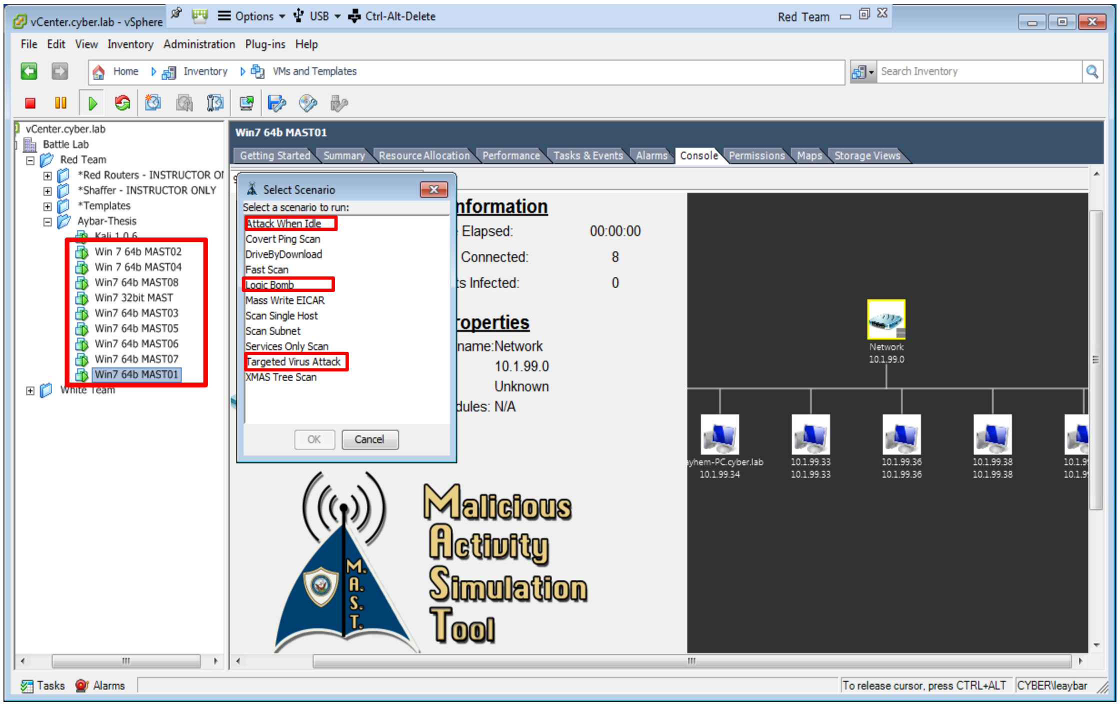
MAST, a malware simulation tool described in Chapter II, served as the foundational platform from which the offensive cyber-attack methods were demonstrated. MAST is a custom software framework originally designed to facilitate the training of network administrators on live networks using SimWare. The MAST framework supports a capability to easily plug-in new SimWare modules for simulating malware effects, and allows the scenario author to combine SimWare modules to meet scenario objectives. New modules can support additional malware behaviors, attack types, or some combination thereof.

IBM’s Java Integrated Development Environment (IDE) tool, Eclipse, was used to develop and compile the Java-based modules used for testing. Since MAST is also developed in Java, Eclipse was used to compile, run, and monitor its various components. The Java Development Kit (JDK), version 1.8, was used for compiling the prototype modules to maintain compatibility with MAST’s current Java libraries.

3. Virtual Hosts

The testing environment utilized eight virtualized Microsoft Windows-based clients running within vSphere. Each host was a Windows 7 Service Pack 1 VM, loaded with MAST and other tools needed to execute the tests. The host VMs included a full instantiation of the Windows OS and GUI, which allowed the workstations to be configured in any way that was needed to support the attack module and scenario. Every VM client required 2 gigabytes of host memory and 34 gigabytes of host storage for a total of 16 gigabytes of host memory and 272 gigabytes of host storage. Normally, intelligence would be gained via reconnaissance and scanning of the target network to configure host services, ports, and firewalls to represent them with a high degree of accuracy. These tests, however, used default settings for the services and ports while the firewall was set to a permissive setting. These settings were chosen since they would not obscure the test results. For example, if a virtual host's communication to another host had been blocked due to a firewall setting, the test of the module would have failed and thus the result would have been a false negative.

The test setup used within the virtually hosted environment is shown in Figure 12. The VM hosts are listed on the far left-side, within the red box, while the network configuration within MAST and running clients are shown on the right-side of the figure. MAST does not resize the contents within its viewing area, so all eight hosts are not visible in this image. The center window allows selection of a scenario from a predefined list. The three new scenarios for this research are indicated in red, and include: "Attack When Idle," "Logic Bomb," and "Targeted Virus Attack."



Elapsed: 00:00:00
Connected: 8
s Infected: 0

name:Network
10.1.99.0
Unknown
dules: N/A

Figure 12. Virtualized Test Network

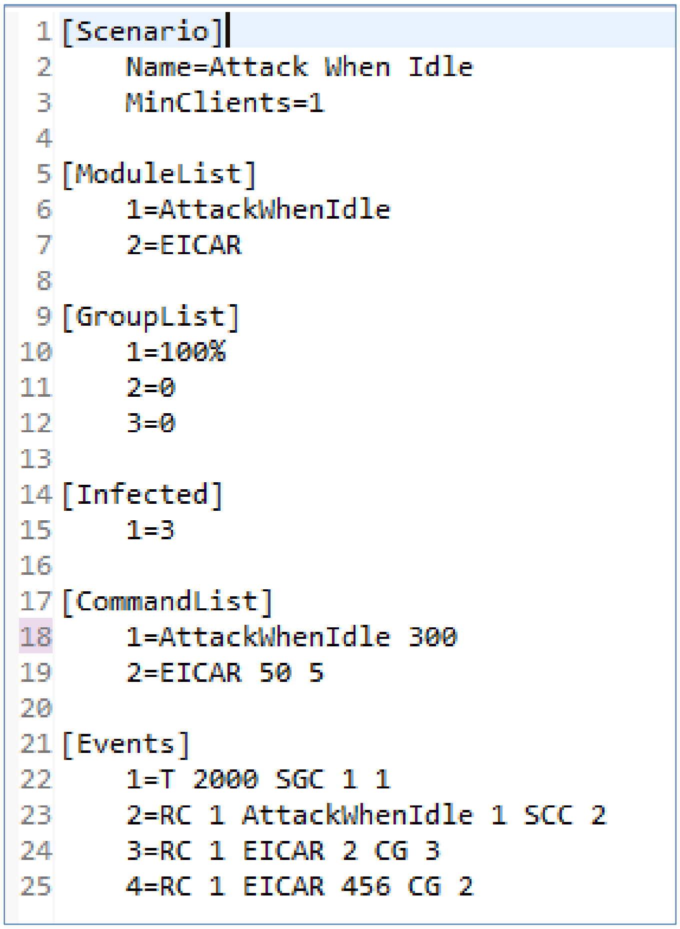
C. SCENARIO FILE

The scenario file is a set of instructions made up of key-value pairs and read by MAST to execute the desired scenario. A detailed description of the design choices and how MAST interacts with the scenario file are described in Belli's thesis, "Extensible SimWare architecture for flexible training scenarios" [10]. Here, we only provide a brief overview of how the scenario file is used for this research. MAST can parse the commands in the scenario file and then direct the execution of those commands to the appropriate MAST clients. The MAST client runs as an application process on the host VMs.

The scenario file consists of six sections that provide the MAST framework with instructions for how to execute the SimWare module(s), and how to perform subsequent actions based on return codes. Not all the sections need to be used in every scenario; for example, a scenario may not result in an infected host, in which case the *infected* section would be omitted. The scenario in Figure 13 was designed to launch an attack after five minutes (300 seconds) of user inactivity, illustrating an attack against a host whose user had walked away, thus leaving the machine idle. The attack was launched by the MAST framework to test a variant of the *timing specific* interaction parameter. We step through this scenario to explain each of the sections in the scenario file.

1. Scenario

This section is mandatory and provides two critical items of information to MAST. The first item is the name of the scenario. The name placed here will be displayed in the MAST GUI (reference Figure 12) scenario list. The user is allowed to select the desired scenario from this list. The second item, MinClients, is the minimum number of clients needed for the scenario to run properly, as determined by the scenario author. For this scenario, we selected the name "Attack When Idle" to provide a clear description of the scenario's objective, with a requirement that at least one client be running.



```
[Scenario]
    Name=Attack When Idle
    MinClients=1

[ModuleList]
    1=AttackWhenIdle
    2=EICAR

[GroupList]
    1=100%
    2=0
    3=0

[Infected]
    1=3

[CommandList]
    1=AttackWhenIdle 300
    2=EICAR 50 5

[Events]
    1=T 2000 SGC 1 1
    2=RC 1 AttackWhenIdle 1 SCC 2
    3=RC 1 EICAR 2 CG 3
    4=RC 1 EICAR 456 CG 2
```

Figure 13. The “Attack When Idle” Scenario File

2. Module List

This section provides a listing of the SimWare modules that will be used for the scenario. It is referenced in the scenario file as [ModuleList], as shown in Figure 13. The listing order is not important because the individual modules are referenced by their number. Modules are the executable code, and do not have spaces in their name. The scenario in Figure 13 used “AttackWhenIdle”, a new module developed in Java to determine when a host machine is idle (i.e., when no user input is detected for a specified amount of time). It also used a pre-existing module, “EICAR,” which simulates a virus by writing a known virus signature pattern file to disk.

3. Group List

In this section, the scenario author is able to specify the number of different groups that the clients will be assigned. The listing of the different groups is collected under the heading [GroupList]. This scenario used three groups, at least one group is always required. Groups can be assigned either a fixed number or a percentage of the total number of clients. In this instance, all of the clients are in Group #1.

Having multiple groups allows the scenario writer to attack different groups with dissimilar attack events within the same scenario or move clients to new groups after some event, such as an infection. In this example, there is an infected segment that correlates to Group #3. Clients moved into Group #3 were marked as infected. This example denotes that all hosts will initially be in Group #1.

4. Infected List

This is the only optional section. The groups beneath the [Infected] section header tell MAST which group or groups contain infected hosts. Hosts that are infected via the scenario are colored red in the GUI viewing window (reference Figure 3) and a count of the number of infected hosts is also shown on the GUI console. Clients that have been infected but have not had their group changed to one of the infected groups will not show as infected on the GUI console.

5. Command List

This section lists the commands that will be subsequently executed via events. Commands are comprised of modules and their command line arguments, and are referenced by their index number. In the example shown in Figure 13, the module “AttackWhenIdle” accepts an input (300) for the number of seconds to wait before concluding that the client is idle. The “EICAR” module uses two command line parameter inputs to create a randomized wait time before writing its output file. The “EICAR” module was included to test whether the

“AttackWhenIdle” module could be stacked with other modules and perform the combined function correctly. The “EICAR” module was executed after the “AttackWhenIdle” module returned from a host that had been without user input for 300 seconds.

6. Events

This section combines the scenario commands and modules with the MAST event functionality to create functional scenario events. Commands are referenced by their index while modules are referenced by their name. The commands listed in the Command List can be ordered as the scenario designer desires, and can be directed at the different groups in the Group List. Event #1 in our example (see Figure 13) is a timer (T) event:

1=T 2000 SGC 1 1

The [Events] list is required to have a timer event as the first element. The timer event’s first parameter is the wait interval in milliseconds, 2000 in the example above. The rest of the event (SGC 1 1) instructs MAST to send a specific command from the Command List to a specific group from the Group List. The Send Group Command (SGC) [10] has the general form:

(SGC) to <GROUPNUM> <COMMANDNUM>

In the example Event #1, Command #1 (“AttackWhenIdle 300”) will be sent to all the clients in Group #1. Event #2 is a conditional event, based on the return code (RC) from the previous command in Event #1 (“AttackWhenIdle”):

2=RC 1 AttackWhenIdle 1 SCC 2

The return code event [10] has the general form:

RC <GROUPNUM> <MODULENAME> <RETURNCODE>

Event #2 evaluates the return code from the “AttackWhenIdle” module that was executed against the clients in Group #1. If the return code is a 1 from any client, then that client would be sent the second command from the Command List for execution (“EICAR 50 5”) [10]. This command has the general form:

Send Client Command (SCC) <COMMANDNUM>

In our scenario, Event #1 invoked the Command #1 on Group #1 clients. Event #2 checked to see whether the module returned any success codes (return code = 1). If so, then Command #2 (EICAR 50 5) will be issued to Group #1 clients. Event #3 is another return code (RC) conditional event:

3=RC 1 EICAR 2 CG 3

As before, this event will evaluate the return code from Group #1 clients from the execution of the “EICAR” command. If the return code from any client is 2, then the Change Group (CG) command will be executed on that particular client. Any client moved into Group #3 will have been displayed by the MAST GUI as infected because Group #3 was our infected group. Finally, Event #4 is another return code (RC) conditional event:

4=RC 1 EICAR 456 CG 2

This event will move any client that had a return code of 456 from the execution of the “EICAR” module from Group #1 to Group #2. These scenario files can be very complex when working with multiple groupings of clients and several command modules. This example was designed to demonstrate the scenario file concepts and was also used to test the “AttackWhenIdle” module.

D. MODULES

These programs and scenario files are at the heart of the MAST framework. They are the SimWare that creates the offensive cyber effects in the scenario. MAST allows the scenario designer a large degree of latitude to construct SimWare modules to meet mission objectives. The modules can be written in a compiled, scripted, or batch programming language. For this thesis, three new modules were developed, using the Java programming language, to implement several of the interaction parameters derived in Chapter III.

1. Detect Idle Host

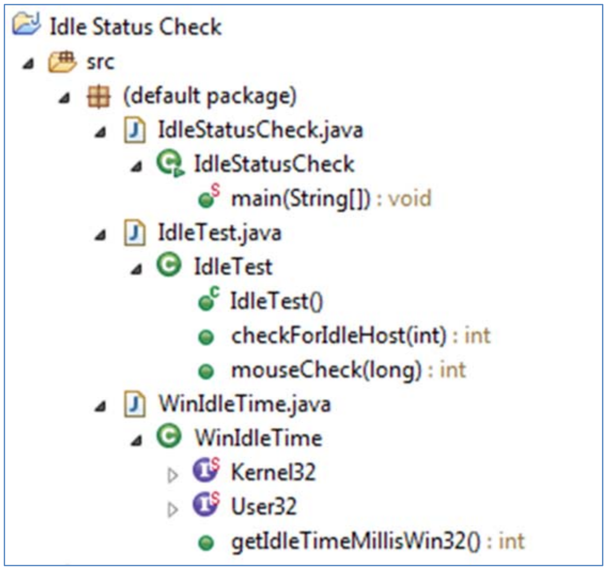
For the offensive cyber-attack scenario developer, the timing of an attack can be just as important as how it occurs. The capability to manipulate the temporal specificity is paramount for the development of realistic scenarios. This module provides a scenario author the ability to have his attack run at time when the user has not interacted with the target machine for some specified amount of time, to further obfuscate the attacker's activity.

In general, to determine user inactivity, this module can detect user inputs (mouse clicks or keyboard entries) and measures the amount of time a client system has been idle. If the idle time is greater than the timeout value passed as an argument, the module exits with a successful return code. If a user event has been detected, the timer is reset and the waiting begins anew. The motivation for this module is to find a time when the user is not at their computer so that subsequent exploit modules can be executed unnoticed.

The code for this component was written in Java using JDK 1.8. For Windows systems, we invoked native shared libraries through the code using the capabilities within Java native access libraries [50]. This provided access to native Windows library functions that returns the time (in milliseconds) since the last user input action (i.e., keypress or mouse action). For non-Windows systems, a similar looping construct checks for any changes in the mouse coordinates over time. If no change occurred during the defined period, the system is determined to be idle, and a successful exit code is returned.

The input parameter for the "AttackWhenIdle" module is the minimum number of seconds during which no user activity is detected. For example, if the scenario developer required two hours of user inactivity before executing subsequent SimWare events, he would pass 7,200 seconds as the input parameter. The return code for this module must correspond to what is expected in the scenario file. This module returns a 1 to denote success, however, if the client continues to receive user inputs, the timer would be reset each time and the module would keep waiting and checking an idle period beyond the timeout parameter. The logic for the module starts with an initial query of the idle time of the host. If the idle time is less than the input time, the delta between the two is calculated. This differential time is used for the sleep interval. If the desired idle time has passed after the sleep interval has expired, the module will return the success code. Otherwise, the time differential is calculated again and the sleep process is repeated.

The three Java classes shown in Figure 14 were used to implement the module. The IdleStatusCheck.java class parses the input arguments and returns the final status code to the calling program, MAST in this case. The IdleTest.java class performs the looping and differential time logic described in the preceding paragraph, while the WinIdleTime.java class returns the idle time in milliseconds via native Windows systems calls. The full code listing for the "AttackWhenIdle" executable module in shown in Appendix A.



```
Idle Status Check
  src
    (default package)
      IdleStatusCheck.java
        IdleStatusCheck
          main(String[]) : void
      IdleTest.java
        IdleTest
          IdleTest()
          checkForIdleHost(int) : int
          mouseCheck(long) : int
      WinIdleTime.java
        WinIdleTime
          Kernel32
          User32
          getIdleTimeMillisWin32() : int
```

Figure 14. The Idle Test Classes Used for "AttackWhenIdle"

2. Logic Bomb

This module name draws its inspiration from the common term given to malware that will activate when a specific set of conditions occur, often at a pre-designated time, and thus a "logical time bomb." This module provides the scenario author the ability to manipulate the temporal targeting aspect of the scenario, albeit in a materially different way than with the “AttackWhenIdle” module. Its modular design also allows it to work in concert with other modules, such as the “EICAR” module, similar to the way “AttackWhenIdle” was shown in Figure 13.

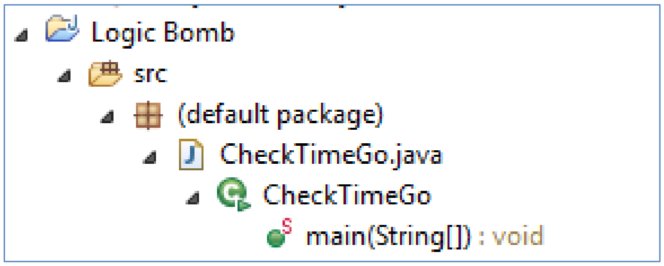
The logic for this module was designed to be efficient and stealthy. It uses a loop construct to check the current date against the date to execute supplied by the scenario author. If the execute date is after the current date, the differential time is calculated and is used as the parameter for the sleep function. Since the module cannot proceed until the current time is after the execute time, there is nothing gained by checking the time more frequently. When the execute date is less than the current date, the module will return a successful exit code. If the date argument given is less than the current date, the module will return a successful exit code immediately.

Like its sibling, “AttackWhenIdle,” this module was written in Java using the 1.8 version of the JDK. The only input parameter is the date when the SimWare is programmed to activate (i.e., return a success code). The format for the input parameters is a four-digit year, two-digit month, and two-digit day, followed by a two-digit hour (24hr format), with two-digit minutes and two-digit seconds. The general form is as follows: yyyy-MM-dd HH:mm:ss. It would appear similar to the following example:

```
[CommandList]
1=LogicBomb 2017-03-05 13:30:10
```

This module accepts its input parameter as a passed argument in the scenario file, similar to the demonstrated scenario in Figure 13, or it can receive the threshold time argument from an input window. The output, upon completion of a successful waiting period, is a successful exit code of 1. If the module receives no input time or bad input time data, either malformed or nonsensical, it will fail and return with an exit code of 456.

The Logic Bomb Java project consists of one Java class, CheckTimeGo.java, shown in Figure 15. This class has one method, main, which contains all the logic described for this module. The full code listing of the Logic Bomb module is shown in Appendix B.



```
Logic Bomb
  src
    (default package)
      CheckTimeGo.java
        CheckTimeGo
          main(String[]) : void
```

Figure 15. Logic Bomb Project Hierarchy

3. Target Specific Host

The last two modules focused on some aspect of temporal specificity. This module enabled the target specificity on the logical level. The ability to target an adversary network or host by IP address or MAC address is vital for a scenario author to develop complex offensive scenarios.

This module accepts one or more IP addresses, MAC addresses, or computer host names as the input parameter(s). This provides maximum flexibility for the scenario designer depending of the piece identifying data they may possess. For example, a scenario might have several hosts, with the goal to have the SimWare run on the two hosts that match the given IP addresses. Since a separate instance of the "TargetSpecificHost" module will run on each host, there can only be one matching IP address even if two IP addresses were given to the module. If the module determines a successful match of an IP address, MAC address, or host name it will exit with a return code of 2, 3, or 4, respectively. If no matches are found, the module exits with a returned code of 456. Similar to Figure 13, the input in the command list for the scenario file would appear similar to the following example:

[CommandList]

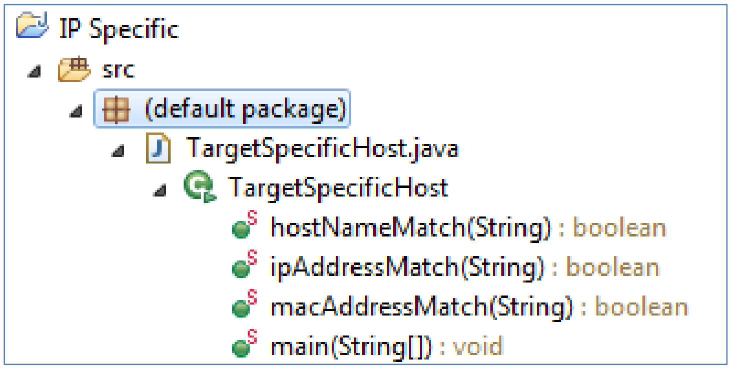
1=TargetSpecificHost 10.1.99.11 10.1.99.34

Or

1=TargetSpecificHost 00-50-56-9C-4A-EF

This logic for this module proceeds in a linear fashion. It loops through the input parameters and compares each one the host's IP address. Java library functions are able to return the IP address for each host and for all its network interfaces (Ethernet, Wi-Fi, virtual). The hosts used in the testing had two network adapters each. If no matches are found, the same looping process is repeated again with the input parameters comparing them to each host's computer name. The last looping iteration traverses the input list and compares each item in the list to each host's MAC address checking for a match.

The IP Specific Java project also consists of one Java class, TargetSpecificHost.java, shown in Figure 16. This class has four methods, which contains all the logic described for this module. The full code listing of the Target Specific Host module is shown in Appendix C.



```
IP Specific
  src
    (default package)
      TargetSpecificHost.java
        TargetSpecificHost
          hostNameMatch(String) : boolean
          ipAddressMatch(String) : boolean
          macAddressMatch(String) : boolean
          main(String[]) : void
```

Figure 16. The Classes and Methods Used for the "TargetSpecificHost"

E. SUMMARY

In this chapter, we detailed the setup of the test environment and the processes used to conduct cyber scenario tests. Additionally, we examined the contents of a scenario file within the context of a new module. This should provide a better understanding of the workings of the scenario file, the capabilities of the new modules, and how they can be combined to create scenarios with increasingly complex interactions and behaviors. In the next chapter, the results of these scenario tests are analyzed, and a way ahead is plotted for future work.

THIS PAGE INTENTIONALLY LEFT BLANK

V. CONCLUSIONS AND FUTURE RESEARCH

A. SUMMARY

The objective of this thesis was to identify requirements for and build a system to support the development and testing of simulated offensive cyber operations scenarios. Although the DOD has cyber ranges where cyber scenario testing can take place, these ranges are in high demand, are expensive to operate and manage, and are not mobile. Moreover, the majority of commercial scenario development frameworks are for testing of network vulnerabilities and do not support the customization necessary to adapt the tool to DOD's needs. This reality highlighted a capability gap that exists at the mid-to-lower end of the offensive cyber scenario development and testing spectrum.

At NPS, an active research initiative called MAST was designed to help train system administrators through the use of simulated malware (SimWare). This research was able to repurpose parts of MAST for the creation and testing of offensive cyber scenarios, focused on determining the elements useful for the formation of a robust offensive scenario development capability. From this work, we extended MAST with those elements to include new interaction parameters, a virtualized test environment, and a working prototype within an integrated, lightweight system.

B. CONCLUSIONS

The research objectives for this thesis were successfully accomplished. The derived requirements, modules, and research of existing frameworks yielded a capability that can be used for the development of offensive cyber-attack scenarios. The following examines the body of work produced by this thesis in light of the specific research questions set out in the objectives of the research.

1. Research Questions

Primary Question: How can offensive cyber tools and exploits be developed and tested effectively in a controlled environment against virtualized models of adversary networks?

Conclusion: This thesis derived several new concepts, and a working prototype that was able to support the development and testing of cyber-attack scenarios. We successfully repurposed and extended MAST to execute scenarios and modules specifically developed for this project. Additionally, a virtualized test environment was constructed within the NPS CYBL, which allowed for the modeling of a scalable notional adversary network. The network consisted of hosts that were complete instances of Windows 7 virtual machines. This allowed for effective testing of the new SimWare modules developed during this research.

Secondary Question 2: What control mechanisms or methods would allow simulated malware to focus on a user-defined target set grouping (e.g., a single host or single subnet)?

Conclusion: One test module was developed and successfully tested to target a specific host by IP address or MAC address addresses this question. The "TargetSpecificHost" module targeted one specific IP and one specific MAC address from among the eight virtual clients in the test environment. The system was able to successfully identify and infect that particular host, and only that host. The module is currently not able to target a single subnet (such as all hosts on 10.1.99.x), but can be extended later to support this capability.

Secondary Question 3: What methods can be used to perform temporal sequencing of malware mimics?

Conclusion: The development and implementation of two temporal specific modules, "LogicBomb" and "AttackWhenIdle," provided a new capability that addresses this requirement. The "LogicBomb" module provided the ability to delay execution of subsequent modules in the scenario file to some pre-determined time in the future. The "AttackWhenIdle" module provided the ability to detect user inputs to a host machine, and delay attack until some author-specified idle time period had passed.

Several test iterations were run based on various date/time inputs for the "LogicBomb" module, and various wait intervals for the "AttackWhenIdle" module. The "LogicBomb" module performed flawlessly across a series of clients. During one series of early tests, it seemed that it may have been performing incorrectly on some clients, but further analysis revealed that the system time on these VM hosts was not correct, and the module was performing as designed. Similarly, when the “AttackWhenIdle” module was tested across multiple hosts, the results showed some clients becoming infected much quicker than expected. Further analysis revealed that there had been no user input on those clients, across multiple test iterations, for more than five minutes (the threshold value used or the AttackWhenIdle testing) and thus the module had performed as intended.

C. RECOMMENDATIONS FOR FUTURE RESEARCH

This thesis was able to achieve a number of important milestones toward the creation of an offensive-oriented cyber scenario development platform. However, this thesis is only the first step in this process, having laid the groundwork upon which additional research can add even greater capabilities to the framework. Future research could focus on three core areas that would provide substantial improvement to the process of developing simulated cyber-attack scenarios: further development of advanced interaction parameters, integration of the MAST framework with the mapping and virtualization capabilities of MAVNATT (described in Chapter II), and enhancement of the MAST framework to support more robust scenario development.

1. Advanced Parameters

The temporal and target specificity modules provided the MAST framework with a significant increase in behavioral complexity that could be applied to other SimWare modules. However, several additional parameters were considered in this research, but were not implemented. Parameters such as skill, efficiency and stealth would add greater realism and nuanced behavior to the developed scenarios.

These additional parameters could be developed and integrated within the MAST framework and selected via menus on the GUI. In this way, the parameters could be applied to any of the scenarios developed and used in the framework. For example, the skill parameter could be implemented as a weighted average, based on a selected value from the GUI menu, and could be applied to the module behavior that would be evaluated at decision points during its execution. A physical analog of this idea would be a surface-to-surface missile, where the guidance control systems provide accurate target discrimination and the operator has control over when the missile is launched. However, if the missile flies with a high arching trajectory, it is still vulnerable to adversary countermeasures. If the missile were equipped with terrain hugging flight capabilities (stealth) to avoid enemy radar and evasive maneuvers to thwart enemy countermeasures (skill), these same features (stealth and skill) could be added to the MAST framework in a programmatic fashion to provide greater realism to the designed scenarios.

The menu of SimWare modules available to MAST is still quite small. Before this research, there were only a small number of scans, virus emulator, and drive-by-download type modules. The three new modules added to the framework nearly doubled the types of behaviors that could be used to create scenarios. Nonetheless, additional SimWare functionality still can be implemented that will provide other types of cyber-attack such as DOS attacks, identity spoofing, man-in-the-middle, and a delivery vehicle such as a Trojan horse. These additional types of SimWare would greatly add to the range of scenarios that could be developed.

2. Integration with MAVNATT

For this thesis, a virtual network environment was constructed for the testing of the interaction behaviors. However, the success of the scenario ultimately depends on the accuracy of the model of the adversary network. MAVNATT has the capability to automatically map a network and then produce a virtualized instance of that network. Since MAST has been demonstrated to run successfully within a virtual environment, the marriage of MAST and MAVNATT could produce a synergy that leverages the best of both research initiatives. Integrating MAST within the virtual instance of a network mapped by MAVNATT would allow the full capability of MAST to generate and test scenarios against a virtualized replica of an adversary network.

3. MAST Framework

There are also improvements that can be made to the MAST framework itself to make it a more robust cyber-attack development platform. The first enhancement would be a menu of available SimWare modules that can be selected from a GUI-style menu and loaded automatically into a scenario file shell. The second major improvement would be to add support for common programming constructs to the scenario file. If these improvements could be implemented, they would greatly increase the effectiveness of MAST as an offensive cyber-attack scenario development platform.

Currently, the MAST scenario files have to be constructed manually by the scenario author. Knowing what modules are available for use in the scenario, and having quick access to them, would greatly facilitate scenario development. Moreover, a scenario that can be quickly constructed by selecting the desired modules from the SimWare menu and have MAST assemble them into a scenario file would speed the development of the various scenarios.

MAST only allows linear processing of the scenario file at this time. There are no looping constructs, complex conditionals, or variable assignments within the current framework. These advanced programming features are very common in modern scripting languages such as Java Script, Bash shell, and others. If these advanced scripting constructs were implemented, this would allow the creation of scenarios many times more complex than what is currently possible.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. ATTACK WHEN IDLE SOURCE CODE

```
public class IdleStatusCheck {

      /**
       * Check and wait for the host to observe required idle time
       * return success when this has occurred. Can be waiting for
       * a long time.
       * @param args idle time in seconds
       * @return Success 1, error or fail 456
       */

      public static void main(String[] args) {

             int timeToWait = 0;
             int returnCode = 456;

             //Read the input argument
             if(args.length != 0 ){
                timeToWait = Integer.parseInt(args[0]);
                 System.out.println("Time to wait: " + timeToWait);
                }
              else {
                     System.err.println("No INPUT wait time given!!");
                     System.err.println("exiting with failure return code!");
                     System.exit(returnCode);
               }

              IdleTest idleCk = new IdleTest();

              returnCode = idleCk.checkForIdleHost(timeToWait);

         System.exit(returnCode);
      }


}
```

IdleTest.java

```
import java.awt.MouseInfo;

public class IdleTest {


   public IdleTest() {

   }
   public int checkForIdleHost(int timeToWait) {


         if (System.getProperty("os.name").contains("Windows")) {

         System.out.println("Our system: " + System.getProperty("os.name"));
         WinIdleTime idleTime = new WinIdleTime();

         for (;;) {
             int idleSec = idleTime.getIdleTimeMillisWin32() / 1000;

             if (idleSec >= timeToWait) {
                     System.out.println("Waited " + timeToWait + " seconds.
The user must be away. :)");
                     System.out.println("Returning success code!");
              return(1);
             }
             try {
               // Calculate remaining time to sleep,
               // want to minimize frequency of checks
               // timeToWait and idleSec is in seconds
               // sleepTime is differential between idleSec and timeToWait
               long sleepTime = ((timeToWait - idleSec) + 1) * 1000;
               Thread.sleep(sleepTime);
             } catch (Exception ex) {
               ex.printStackTrace();
               }
           }
         }
         else {
                    // if not windows OS, check for mouse movements
                 // send timeToWait in milliseconds
                 System.out.println("Our system: " +
                                      System.getProperty("os.name"));
                 long mouseInterval = timeToWait*1000;
                 return(mouseCheck(mouseInterval));
         }

 }

         /**
          * threshold is in milliseconds
          *  keeps checking for mouse position
          *  If time has elapsed and mouse hasn't moved
          *  assumption is system is unattended by user
          *  @threshold - amount of time to determine if system idle.
          */

        public int mouseCheck(long threshold)
         {

             long startTime;
             long currentTime;
             long diff;
```

```
            int startXPos;
            int currentXPos;

            startTime = System.currentTimeMillis();
            startXPos = MouseInfo.getPointerInfo().getLocation().x;

            do {
              currentTime = System.currentTimeMillis();
              diff = currentTime - startTime;
              currentXPos = MouseInfo.getPointerInfo().getLocation().x;

              // If the mouse moved, we have to reset
              if ( currentXPos != startXPos) {
               startXPos = currentXPos;
               diff = 0;
               startTime = System.currentTimeMillis();
               System.out.println("Waited " + diff + " seconds but mouse
moved");
               System.out.println("Staring the wait all over");
              }

              // sleep for differential time between
              // IdleTime and the elapsed time
              try {
               long sleepTime = threshold - diff;
                  Thread.sleep(sleepTime);
              } catch (Exception ex) {}

            } while (diff > threshold && currentXPos == startXPos);

            return(1);
        }

}
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B. LOGIC BOMB SOURCE CODE

```
import java.util.*;
import java.text.*;
import javax.swing.JOptionPane;

/**  Format for input date time string  four digit year yyyy,
 *   two digit month MM, two digit day dd, followed by a
 *   two digit hr (24hr format), two digit minute,
 *   and a two digit second yyyy-MM-dd HH:mm:ss
 *   @author L. Aybar
 */

public class CheckTimeGo {

       public static void main(String[] args) {

             String date1 = null;
             String time1 = null;
             String exeDateTime = null;
             Date executeTime = new Date(10000);

        // get current time
             Date currentTime = new Date();
        System.out.println("Current Time: " + currentTime.toString());

        // format of date to read from command args
             SimpleDateFormat ft =
                                   new SimpleDateFormat ("yyyy-MM-dd HH:mm:ss");

             //Check to see if date,time was passed via command line
             if(args.length == 2){
               date1 = args[0];
               time1 = args[1];
              //make the input date;
              exeDateTime = date1 + " " + time1;
             }

             // make sure we get some input
             if ( exeDateTime == null){
             exeDateTime = (String)JOptionPane.showInputDialog(
                              null,
                               "Enter the date and time to start:\n",
                              "Temporal attack",
                              JOptionPane.INFORMATION_MESSAGE,
                              null,
                              null,
                             "yyyy-mm-dd hh:mm:ss");

             }

             // if still no input values, just exit with an error code
             if ((exeDateTime == null) || (exeDateTime.charAt(1) == 'y')){
                   System.err.println("Nothing received for date-time
                                                           value, exiting!");
                   System.exit(456);
             }


        // try to parse input date from inputs
        try
        {
             executeTime = ft.parse(exeDateTime);
             System.out.println("Execute time: " + executeTime);
        }
         catch (ParseException e)
        {
             System.out.println("Unparseable using " + ft);
             System.exit(456);
        }

       // if executeTime is in the future, then wait for it
        while (executeTime.after(currentTime))
        {
             // go to sleep, wake when it's time
             // sleep for differential time between the future time and
             // the current time, then check again.
             long sleepTime;
             long tmpTime = (executeTime.getTime() - currentTime.getTime());
             if ( tmpTime < 0 ){
               // we have passed our time, time to exit
             }
             else{
                   sleepTime = tmpTime +1;
                   System.out.println("Sleep time: " + sleepTime);
                ctry {
                         Thread.sleep(sleepTime);
                         currentTime = new Date();
                      } catch (InterruptedException e) {
                         // TODO Auto-generated catch block
                         e.printStackTrace();
                         System.exit(456);
                      }
           }
        }

        System.out.println("Current time: " + currentTime);
        System.out.println("Time to go break things!");
        System.exit(1);
       }

}
```

APPENDIX C. TARGET SPECIFIC SOURCE CODE

```java
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.net.InetAddress;
import java.net.NetworkInterface;
import java.net.SocketException;
import java.net.UnknownHostException;
import java.util.Enumeration;

public class TargetSpecificHost {

        /**
         * Get host IP address, compare it passed IP address
         * @param myIPAddress
         * @return true for a match, false if no match
         * @author L. Aybar
         */
        public static boolean ipAddressMatch(String myIPAddress){

                Enumeration<?> e = null;
                try {
                        e = NetworkInterface.getNetworkInterfaces();
                } catch (SocketException e1) {
                        e1.printStackTrace();
                }
                while(e.hasMoreElements())
                {
                    NetworkInterface n = (NetworkInterface) e.nextElement();
                    Enumeration<?> ee = n.getInetAddresses();
                    while (ee.hasMoreElements())
                    {
                        InetAddress i = (InetAddress) ee.nextElement();
                        String tmpIPAddress = i.getHostAddress();
                        if( tmpIPAddress.equalsIgnoreCase(myIPAddress)){
                           System.out.println("Found match! " + myIPAddress + "
                                      ClientIPAddress: " + i.getHostAddress());;
                           return true;
                        }
                        System.out.println(i.getHostAddress());
                    }
                }
                return false;

        }

        /**
         * Get host machine name, compare it passed host name
         * @param hostName
         * @return true if matches, false if does not match.
         */
        public static boolean hostNameMatch(String hostName) {
        // see if client host name matches

        InetAddress IP = null;
        String clientHost = null;

        try {
                IP = InetAddress.getLocalHost();
        } catch (UnknownHostException e) {
                        e.printStackTrace();
        }
        System.out.println("Host name of my system is := "+IP.getHostName());

        clientHost = IP.getHostName();
        if ( clientHost.equalsIgnoreCase(hostName) ) {
                System.out.println("Host name match := "+ IP.getHostName() +"
Passed hostname: " + hostName);
          return true;
        }
        else
          return false;

}

        /**
         * Get host MAC address, compare to MAC address passed.
         * @param myMacAddress
         * @return true for match, false for no match
         */
        public static boolean macAddressMatch(String myMacAddress){

        InetAddress ip;
        String macAdd = null;
        try {

                ip = InetAddress.getLocalHost();
                NetworkInterface network = NetworkInterface.getByInetAddress(ip);

                byte[] mac = network.getHardwareAddress();

                System.out.print("Host MAC address : ");

                StringBuilder sb = new StringBuilder();
                for (int i = 0; i < mac.length; i++) {
                        sb.append(String.format("%02X%s", mac[i], (i < mac.length
- 1) ? "-" : ""));
                }

                System.out.println(sb.toString());
                macAdd = sb.toString();

        } catch (UnknownHostException e) {

                e.printStackTrace();
```

```
       } catch (SocketException e){

             e.printStackTrace();

       }

       if( macAdd.equalsIgnoreCase(myMacAddress) ){
             System.out.println("MAC address match!");
             System.out.println("Passed MAC address: " + myMacAddress);
             return true;
       }
             else {
             return false;
       }

   }

/**
 * Could pass in one or multiple IP addresses, host names, or MAC addresses
 * Would allow targeted attacks against more than one IP
 * Check to see if any of the clients are a match, an IP match returns 2
 * a host name match returns 3, a MAC address returns 4
 * If no matches found, returns 456
 * @param args IPaddress(es), Host name(s), MAC address(es)
 * @throws IOException
 */
       public static void main(String[] args) throws IOException {

             // defaults
           String[] inputs = new String[]{"1"};
           int returnCode = 456;

           File f1 = new File("targetSpecific.log");
             FileWriter fw = null;
             try {
                   fw = new FileWriter(f1);
             } catch (IOException e) {
                   e.printStackTrace();
             }

             //Read the input argument(s)
             if(args.length != 0 ){
                  for( int i=0; i < args.length; i++){
                   inputs[i] = args[i];
                   System.out.println("Input arguments: " + inputs[i]);
                   try {
                          fw.write("Inputs: " + inputs[i] + " ");
                    } catch (IOException e) {
                          e.printStackTrace();
                    }
                }
              }
               else {
                     System.err.println("No INPUT arguments given!!");
                     System.err.println("exiting with failure return code!");
                     fw.write("No INPUTS arguments given!");
                     System.exit(returnCode);
               }

       // check for matching IP addresses
      for(int i=0; i < inputs.length; i++){
           if( ipAddressMatch(inputs[i]) ){
          fw.write(inputs[i] + " ");
         returnCode = 2;
              }
       }

      // check for matching host name(s)
       for(int i=0; i < inputs.length; i++){
         if ( hostNameMatch(inputs[i]) ) {
          fw.write(inputs[i] + " ");
         returnCode = 3;
         }
       }

       // check for matching MAC addresses
       for(int i=0; i < inputs.length; i++){
        if ( macAddressMatch(inputs[i]) ) {
           fw.write(inputs[i] + " ");
            returnCode = 4;
        }
       }

        System.out.println("returnCode: " + returnCode);
        fw.write("returnCode" + returnCode);
        fw.flush();
        fw.close();

    // return
      System.exit(returnCode);

      }  // end main

}   // end TargetSpecificHost
```

LIST OF REFERENCES

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

[2] C. Hopkins. (2015, Dec. 11). The definitive glossary of hacking terminology. [Online]. Available: http://www.dailydot.com/debug/hacking-security-glossary-adware-bot-doxing/

[3] J. Sandoz, “Red teaming: Shaping the transformation process," Inst. for Defense Analysis, Alexandria, 2001.

[4] R. Love. (2013, Dec. 08). In computer security, what is a sandbox? [Online]. Available: https://www.quora.com/In-computer-security-what-is-a-sandbox

[5] P. Engebretson, *The Basics of Hacking and Penetration Testing*, 2nd Ed. Waltham, MA: Syngress, 2013.

[6] S. Oriyano, *Hacker Techniques, Tools, and Incident Handling*, 2nd Ed. Burlington, MA: Jones & Bartlett Learning, 2013.

[7] D. McBride, "Mapping, awareness, and virtualization network administrator training tool (MAVNATT) architecture and framework," M.S. thesis, Dept. of Comp. Sci., Naval Postgraduate School, Monterey, CA, 2015.

[8] A. Collier, "Automated network mapping and topology verification," M.S. thesis, Dept. of Comp. Sci., Naval Postgraduate School, Monterey, CA, 2106.

[9] E. Berndt, "Mapping, awareness, and virtualization network administrator training tool virtualization module," M.S. thesis, Dept. of Info. Sci., Naval Postgraduate School, Monterey, CA, 2015.

[10] G. Belli, "Extensible simware architecture for flexible training scenarios," M.S. thesis, Dept. of Comp. Sci., Naval Postgraduate School, Monterey, CA, 2016.

[11] E. Lowney, "Network communications protocol for the malicious activity simulation tool (MAST)," M.S. thesis, Dept. of Comp. Sci., Naval Postgraduate School, Monterey, CA, 2015.

[12] B. Diana, "Malicious activity simulation tool (MAST) and trust," M.S. thesis, Dept. of Comp. Sci., Naval Postgraduate School, Monterey, CA, 2015.

[13] W. Taff and P. Salevski, "Malware mimics for network security assessment," M.S. thesis, Dept. of Comp. Sci., Naval Postgraduate School, Monterey, CA, 2011.

[14] J. Neff, "Verification and validation of the malicious activity simulation tool (MAST) for network administrator training and evaluation," M.S. thesis, Dept. of Comp. Sci., Naval Postgraduate School, Monterey, CA, 2012.

[15] R. Longoria, "Scalability assessments for the malicious activity simulation tool (MAST)," M.S. thesis, Dept. of Comp. Sci., Naval Postgraduate School, Monterey, CA, 2012.

[16] A. Littlejohn and E. Makhlouf, "Test and evaluation of the malicious activity simulation tool (MAST) in a local area network (LAN) running the common PC operating system environment (COMPOSE)," M.S. thesis, Dept. of Comp. Sci., Naval Postgraduate School, Monterey, CA, 2013.

[17] P. Stephenson. (2015, Mar. 02). Rapid7 nexpose ultimate appliance. [Online]. Available: http://www.scmagazine.com/rapid7-nexpose-ultimate-appliance/review/4340/

[18] R. Penko, *Corporate Computer and Network Security*. New York, NY: Cram101 Inc., 2015.

[19] Metasploit. (2011, Apr. 09). Security Tools. [Online]. Available: http://sectools.org/tool/metasploit/

[20] T. Greene. (2016, Jan. 26). Security startup wages continuous war games against networks. [Online]. Available: http://www.networkworld.com/article/3026525/security/security-startup-wages-continuous-war-games-against-networks.html

[21] P. Stephenson. (2016, Mar. 01). SafeBreach continuous security validation platform. [Online]. Available: http://www.scmagazine.com/safebreach-continuous-security-validation-platform/article/474352/

[22] P. Stephenson. (2013, Feb. 01). Core impact professional. [Online]. Available: http://www.scmagazine.com/core-impact-professional/review/3791/

[23] P. Stephenson. (2015, Mar. 02). Core security core insight. [Online]. Available: http://www.scmagazine.com/core-security-core-insight/review/4337/

[24] M. Sudit, M. Kistner, J. Kistner and K. Costantini, "Cyber attack modeling and simulation for network security analysis," presented at *IEEE Winter Simulation Conference*, Washington, DC, 2007.

[25] J. Hammerstein and C. May, "The CERT approach to cybersecurity workforce development," Carnegie Mellon Univ. Software Eng. Inst., Pittsburgh, PA, Tech. Rep. CMU/SEI-2010-TR-45, Dec. 2010.

[26] J. Mayes, "Modeling large-scale networks using virtual machines and physical appliances," Carnegie Mellon Univ. Software Eng. Inst., Pittsburgh, PA, Tech. Rep. DM-0000921, Jan. 2014.

[27] S. McClure, J. Scambray, and G. Kurtz, *Hacking Exposed 7, Network Security Secrets and Solutions.* New York: McGraw Hill, 2012.

[28] EC Council Press, *Ethical Hacking and Countermeasures: Attack Phases*. Boston: Cengage Learning, 2016.

[29] S. McClure, J. Scambray, and G. Kurtz, *Hacking Exposed: Network Security: Secrets and Solutions*. New York, NY: McGraw Hill, 1999.

[30] S. Jajodia, S. Noel, and B. O'Berry, "Topological Analysis of Network Attack Vulnerability," in *Mananging Cyber Threats*, V. Kumar, Ed., New York, NY: Springer, 2005, pp. 247-266.

[31] A. Orebaugh and B. Pinkard, *Nmap in the Enterprise, Your Guide to Network Scanning*. Burlington, MA: Syngress Publishing Inc., 2008.

[32] N. Hayes, "A definitive interoperability test methodology for the malicious activity simulation tool (MAST)," M.S. thesis, Dept. of Comp. Sci., Naval Postgraduate School, Monterey, CA, 2013.

[33] R. Goldberg, “Architectural principles for virtual computer systems,” Ph.D. dissertation, Dept. of Comp. Sci., Harvard Univ., Cambridge, MA, 1972.

[34] Free Virtualization Software & Hypervisors. (2015, Dec. 13). Web Tech Magazine. [Online]. Available: http://webtechmag.com/free-virtualization-software-hypervisors/

[35] J. Ambury. (2008, Jun. 11). Internet Encyclopedia of Philosophy. [Online]. Available: http://www.iep.utm.edu/socrates/

[36] R. Axlerod and R. Lliev, "Timing of cyber conflict," *Proceedings of the Nat. Academy of Sciences*, vol. 111, no. 4, pp. 1298-1303, Jan. 2014.

[37] D. Blair, M. Chertoff, F. Cillufo, and N. O'Connor, "Into the gray zone, The private sector and active defenses," Center for Cyber & Homeland Security, Washington, DC, Oct. 2016.

[38] N. Provos and T. Holz, *Virtual Honeypots, from Botnet Tracking to Intrusion Detection, Boston*. Boston, MA: Addison Wesley, 2007.

[39] *Cyberspace Operations*, Joint Publication 3-12, Department of Defense, Washington, DC, 2013, pp. II-2 – II-3.

[40] R. Langer, "To kill a centrifuge, A technical analysis of what Stuxnet's creators tried to achieve," Langer Grp., Arlington, VA, Rep. TR-001, Nov. 2013.

[41] *Joint Targeting*, Joint Publication 3-60, Department of Defense, Washington, DC, 2007, pp. II-15 – II-18.

[42] C. May, "Operationalizing DOD cyber workforce development," unpublished.

[43] D. Coldewey. (2016, Aug. 06). Carnegie Mellon's Mayhem AI takes home $2 million from DARPA's cyber grand challenge. [Online]. Available: https://techcrunch.com/2016/08/05/carnegie-mellons-mayhem-ai-takes-home-2-million-from-darpas-cyber-grand-challenge/

[44] C. Pellerin. (2016, Aug. 06). Three teams earn prizes in DARPA cyber grand challenge. [Online]. Available: https://www.defense.gov/News/Article/Article/906931/three-teams-earn-prizes-in-darpa-cyber-grand-challenge

[45] Mayhem declared preliminary winner of historic cyber grand challenge. (2016, Aug. 04). DARPA. [Online]. Available: http://www.darpa.mil/news-events/2016-08-04

[46] The four primary types of network attacks. (2012, Feb. 04). eTutorials. [Online]. Available: http://etutorials.org/Networking/Cisco+Certified+Security+Professional+Certification/Part+I+Introduction+to+Network+Security/Chapter+1+Understanding+Network+Security+Threats/The+Four+Primary+Types+of+Network+Attack/

[47] Common types of network attack. (2016, Aug. 01) Microsoft. [Online]. Available: https://technet.microsoft.com/en-us/library/cc959354.aspx

[48] Network security threats and solutions. (2015, Sep. 19). ComputerNetworkingNotes.com. [Online]. Available: http://www.computernetworkingnotes.com/ccna-study-guide/network-security-threat-and-solutions.html

[49] F. Stroud. (2016, Feb. 14). VMware vSphere. [Online]. Available: http://www.webopedia.com/TERM/V/vmware-vsphere.html

[50] Java Native Access. (2009, Jul. 20). MVN Repository. [Online]. Available: https://mvnrepository.com/artifact/com.sun.jna/jna

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California