



NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

**RED TEAM IN A BOX (RTIB):
DEVELOPING AUTOMATED TOOLS TO IDENTIFY, ASSESS,
AND EXPOSE CYBERSECURITY VULNERABILITIES IN
DEPARTMENT OF THE NAVY SYSTEMS**

by

Joseph A. Plot

June 2019

Thesis Advisor:
Co-Advisor:

Alan B. Shaffer
Gurminder Singh

Approved for public release. Distribution is unlimited.

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			<i>Form Approved OMB No. 0704-0188</i>	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington, DC 20503.				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE June 2019	3. REPORT TYPE AND DATES COVERED Master's thesis	
4. TITLE AND SUBTITLE RED TEAM IN A BOX (RTIB): DEVELOPING AUTOMATED TOOLS TO IDENTIFY, ASSESS, AND EXPOSE CYBERSECURITY VULNERABILITIES IN DEPARTMENT OF THE NAVY SYSTEMS			5. FUNDING NUMBERS R6MG9	
6. AUTHOR(S) Joseph A. Plot				
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) OSD			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.				
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.			12b. DISTRIBUTION CODE A	
13. ABSTRACT (maximum 200 words) The U.S. Navy and Marine Corps manage a vast number of computer systems, both afloat and ashore, many of which are neither directly connected to an external Internet Protocol (IP) network nor updated regularly, but do occasionally interact with other IP-connected devices. As malicious actors advance their capabilities to exploit and penetrate computer networks, the Department of the Navy (DoN) must be able to verify whether or not its computer systems are susceptible to cyber-attacks. A current mitigation technique is to use a cyber red team to assess a friendly network in a controlled environment; however, this method of conducting assessments can be costly and time-consuming, and may not target specific critical systems. This thesis developed a proof-of-concept tool called Red Team in a Box (RTIB) that addresses the current resource limitations of cyber red teams by leveraging open source software and other methods to discover, identify, and conduct a vulnerability scan on a computer system's software via a graphical user interface. The results of the vulnerability scan offer the RTIB user possible mitigation strategies to lower the risk from potential cyber-attacks without the need for a dedicated cyber red team operating on the target host or network. This research fundamentally provides the foundation to further develop an automated tool that Sailors and Marines with limited expertise can use to conduct a thorough cybersecurity vulnerability assessment on DoN systems.				
14. SUBJECT TERMS red team, cyber, offensive cyber operations, vulnerability assessment, automation			15. NUMBER OF PAGES 89	
			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.

**RED TEAM IN A BOX (RTIB): DEVELOPING AUTOMATED TOOLS TO
IDENTIFY, ASSESS, AND EXPOSE CYBERSECURITY VULNERABILITIES IN
DEPARTMENT OF THE NAVY SYSTEMS**

Joseph A. Plot
Major, United States Marine Corps
BS, U.S. Naval Academy, 2005

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

**NAVAL POSTGRADUATE SCHOOL
June 2019**

Approved by: Alan B. Shaffer
Advisor

Gurminder Singh
Co-Advisor

Peter J. Denning
Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The U.S. Navy and Marine Corps manage a vast number of computer systems, both afloat and ashore, many of which are neither directly connected to an external Internet Protocol (IP) network nor updated regularly, but do occasionally interact with other IP-connected devices. As malicious actors advance their capabilities to exploit and penetrate computer networks, the Department of the Navy (DoN) must be able to verify whether or not its computer systems are susceptible to cyber-attacks. A current mitigation technique is to use a cyber red team to assess a friendly network in a controlled environment; however, this method of conducting assessments can be costly and time-consuming, and may not target specific critical systems. This thesis developed a proof-of-concept tool called Red Team in a Box (RTIB) that addresses the current resource limitations of cyber red teams by leveraging open source software and other methods to discover, identify, and conduct a vulnerability scan on a computer system's software via a graphical user interface. The results of the vulnerability scan offer the RTIB user possible mitigation strategies to lower the risk from potential cyber-attacks without the need for a dedicated cyber red team operating on the target host or network. This research fundamentally provides the foundation to further develop an automated tool that Sailors and Marines with limited expertise can use to conduct a thorough cybersecurity vulnerability assessment on DoN systems.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I.	INTRODUCTION.....	1
A.	PROBLEM STATEMENT	2
B.	SCOPE	2
C.	BENEFITS OF STUDY.....	3
D.	THESIS OUTLINE.....	3
	1. Chapter II: Background.....	3
	2. Chapter III: Design and Methodology	3
	3. Chapter IV: System Implementation	4
	4. Chapter V: Conclusion and Future Work.....	4
II.	BACKGROUND	5
A.	CYBERSPACE OPERATIONS AND THE CYBER KILL CHAIN	5
B.	CYBERSPACE THREATS TO THE DEPARTMENT OF DEFENSE	5
C.	DEPARTMENT OF DEFENSE RED TEAMS	6
D.	ISSUES FACING DEPARTMENT OF DEFENSE CYBER RED TEAMS	7
	1. Funding Limitations and Competing Industry	7
	2. Time to Train.....	8
	3. Limitations imposed on DoD Cyber Red Teams.....	8
E.	RED TEAM TOOLS	9
F.	RED TEAM IN A BOX BACKGROUND RESEARCH.....	10
	1. Vulnerability Assessment Framework.....	10
	2. Dynamic Taint Analysis and Forward Symbolic Execution	11
	3. Net-Nirikshak 1.0	11
	4. Pentest Box	12
	5. Automated Intrusion Detection	13
	6. Firmalice	14
	7. Fuzzing	15
	8. Mayhem Cyber Reasoning System.....	16
	9. RTIB on Industrial Control Systems	17
	10. Driller	18
	11. Angr.....	18
	12. PovFuzzer, Rex, Colorguard, and Patcherex	19
	13. Angr on Industrial Internet of Things	19
G.	CHAPTER SUMMARY.....	22

III.	DESIGN AND METHODOLOGY	23
A.	OVERVIEW	23
B.	PENETRATION TESTING DISTRIBUTIONS.....	24
C.	HOST DISCOVERY AND OPERATING SYSTEM FINGERPRINTING	25
D.	VULNERABILITY SCANNING	27
E.	FIRMWARE EXTRACTION AND EMULATION.....	29
F.	SYSTEM AUTOMATION AND FEEDBACK.....	31
G.	SUMMARY	33
IV.	SYSTEM IMPLEMENTATION.....	35
A.	OVERVIEW	35
B.	ENVIRONMENT.....	35
1.	Background	35
2.	RTIB GUI	37
C.	SCENARIO AND RTIB FUNCTIONALITY	39
D.	TESTING AND RESULTS.....	45
1.	Testing a /30 Network.....	45
2.	Testing a /24 Network.....	49
3.	Firmware Extraction	52
E.	SUMMARY	52
V.	CONCLUSIONS AND FUTURE WORK.....	53
A.	SUMMARY	53
B.	CONCLUSIONS	53
1.	Primary Research Question	54
2.	Secondary Research Question	54
C.	RECOMMENDATIONS FOR FUTURE WORK.....	55
1.	NSA Open Source Software	55
2.	Automating User Feedback.....	55
3.	Firmware Extraction	56
4.	Human Subject Testing	56
	APPENDIX A. RTIB SOFTWARE BLOCK DIAGRAM.....	57
	APPENDIX B. RTIB SOURCE CODE	59
	LIST OF REFERENCES.....	65
	INITIAL DISTRIBUTION LIST	71

LIST OF FIGURES

Figure 1.	Sample RTIB Layout Diagram	24
Figure 2.	CVEs discovered by OpenVAS and Nessus. Source [46].	29
Figure 3.	Firmware Analysis Toolkit Screenshot. Source: [48].	31
Figure 4.	Typical Type I Hypervisor Architecture. Source: [49].	36
Figure 5.	Operating System Market Share by Version. Source: [53].	37
Figure 6.	RTIB GUI Screenshot.....	39
Figure 7.	RTIB Flow Diagram	40
Figure 8.	Host Discovery Flow Diagram	42
Figure 9.	RTIB Host Discovery Screenshot.....	42
Figure 10.	OS Discovery Phase Flow Diagram	43
Figure 11.	MSF & OpenVAS Flow Diagram.....	44
Figure 12.	Host Discovery Results.....	46
Figure 13.	NMAP OS Discovery Results.....	46
Figure 14.	Snapshot of p0f Capture.....	46
Figure 15.	GSA Target Creation Window. Source: [57].	47
Figure 16.	GSA Hosts Topology for the 10.2.99.84/30 Network. Source: [57].	48
Figure 17.	GSA NVTs by Severity Class on the 10.2.99.84/30 Network. Source: [57].	48
Figure 18.	GSA Scan Result Overview for the 10.2.99.84/30 Network. Source: [57].	49
Figure 19.	GSA Hosts Topology for the 10.2.99.0/24 Network. Source: [57].	50
Figure 20.	GSA Scan Result Snapshot for the 10.2.99.0/24 Network. Source: [57].	51

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

Table 1.	Summary of Background Research	21
Table 2.	Common Penetration Testing Distributions.....	25
Table 3.	Common OS Fingerprint Values	26

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

AFL	American Fuzzy Lop
APT	Advanced Persistent Threat
CERT	Cyber Emergency Response Team
CIDR	Classless Inter-Domain Routing
CLI	Command-Line Interface
CSV	Comma-Separated Values
CVE	Common Vulnerabilities and Exposures
CYBL	Cyber Battle Lab
DARPA	Defense Advanced Research Projects Agency
DCS	Distributed Control Systems
DoD	Department of Defense
DoN	Department of the Navy
DOT&E	Director, Operational Test & Evaluation
F2T2EA	Find, Fix, Track, Target, Engage and Assess
FAT	Firmware Analysis Toolkit
FY	Fiscal Year
GAO	Government Accountability Office
GSA	Greenbone Security Assistant
GUI	Graphical User Interface
ICMP	Internet Control Message Protocol
ICS	Industrial Control Systems
IIoT	Industrial Internet of Things
IoT	Internet of Things
IP	Internet Protocol
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
IT	Information Systems Technician
JP	Joint Publication
LAN	Local Area Network
MAC	Media Access Control

MOS	Military Occupational Specialty
MSF	Metasploit Framework
MSS	Maximum Segment Size
NPS	Naval Postgraduate School
NCS	National Cyber Strategy
NMAP	Network Mapper
NSA	National Security Agency
NVD	National Vulnerability Database
OMP	OpenVAS Management Protocol
OS	Operating System
OpenVAS	Open Vulnerability Assessment System
PDF	Portable Document Format
PLC	Programmable Logic Controllers
QEMU	Quick Emulator
RAND	Research and Development
ROP	Return-Oriented Programming
RTIB	Red Team in a Box
SCADA	Supervisory Control and Data Acquisition
SPAWAR	Space and Naval Warfare Systems Command
SQL	Structured Query Language
SSH	Secure Shell
TCP	Transmission Control Protocol
TTL	Time To Live
UDP	User Datagram Protocol
UEFI	Unified Extensible Firmware Interface
USB	Universal Serial Bus
VM	Virtual Machine
XML	Extensible Markup Language

ACKNOWLEDGMENTS

I want to thank my thesis advisors, Dr. Alan Shaffer, and Dr. Gurminder Singh, for their patience, guidance, and words of encouragement as I tackled this thesis. I would also like to thank all of the other faculty members who provided me with the knowledge and insight to conduct this research, including Chris Eagle, Paul Clark, J. D. Fulp, Loren Peitso, Dr. Dennis Volpano, Dr. Geoffrey Xie, and Dr. Robert Beverly.

Most importantly, I would like to thank my family and friends for their support over the past two years. I truly appreciate the time and effort everyone has spent being patient with me as I spent my time coding and writing.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

Technologically advanced weapon systems have given the United States military a competitive edge over their adversaries across all warfare domains, but their current dependency on a vast number of embedded and networked computer systems presents a critical vulnerability in cyberspace due to the increased number of attack surfaces that malicious actors can exploit. Many of the vulnerabilities discovered in old and unpatched software on a secure computer network can be exploited through the introduction of malware via an infected device. Regardless, cyber risks have threatened weapon systems for decades, and the United States is still grappling with how to address all of its cyber vulnerabilities. The cybersecurity posture of a military organization's computer devices, especially those without Internet connectivity, can be easily overlooked unless they are regularly scanned, patched, or updated. In fact, according to operational testing conducted by the Government Accountability Office (GAO), the "[Department of Defense] routinely found mission-critical cyber vulnerabilities in systems that were under development, yet program officials GAO met with believed their systems were secure and discounted some test results as unrealistic," [1]. More importantly, the GAO noted that they discovered vulnerabilities that likely only represent a small fraction of the total number of cybersecurity threats.

The GAO further stated that operational test and evaluation organizations within each service branch conduct their cybersecurity assessments on new weapon systems and occasionally receive support from the National Security Agency (NSA) and U.S. Cyber Command; however, these two organizations are not primarily responsible for identifying vulnerabilities on new weapon systems. Furthermore, the 2019 Secretary of the Navy Cybersecurity Readiness Review states that "phishing attacks, poor cyber hygiene, and failure to update and patch software are the root cause of the vast majority of cyber incidents," [2]. The military's current policy is ostensibly allowing the "commander to 'make the call' on the risk mitigation for his/her installation, facility, or vessel," [3]. Commanders often rely on red teams to conduct cybersecurity assessments of their networks and systems. Regrettably, an in-depth and independent assessment of a computer

network conducted by a cyber red team may be unfeasible due to time, financial, and personnel expertise constraints.

The purpose of this research was to develop a portable cyber red teaming tool called Red Team in a Box (RTIB) that can be used to identify and assess the cybersecurity vulnerabilities on Department of the Navy (DoN) computer systems not directly connected to the Internet, and to provide users with recommendations to mitigate the cyber threats associated with these vulnerabilities. This tool is designed to overcome the resource limitations of current red teams conducting remote cybersecurity assessments on cyber-physical systems. Ideally, RTIB would widely deploy as a cheap, convenient, and effective cybersecurity tool that would help enhance the security posture on DoN computer systems by complementing other defense in depth measures.

A. PROBLEM STATEMENT

This thesis addresses the following research questions.

1. **Primary Question:** How can a portable set of software tools be developed to automatically detect and expose security vulnerabilities on DoN computer systems?
2. **Secondary Question:** How would such a tool report discovered vulnerabilities without interrupting the standard operating protocols of the host system?

B. SCOPE

This thesis analyzed previous research conducted on automated cybersecurity assessments to provide a proof-of-concept tool that can scan for vulnerabilities on computer systems not directly connected to the Internet. The methodologies explored in this research will facilitate future work in the creation of advanced capabilities for an automated red teaming device.

Attempting to assess a target system over a commercial Internet Protocol (IP) network was out of scope for this thesis. We also assumed that all physical connections

between RTIB and the target device were successfully achieved since this thesis focuses only on software tools. Ultimately, RTIB will enable non-experts to conduct automated red team assessments on DoN networks using a non-destructive offensive cyber operations tool that reveals shortcomings in the target system without interrupting its normal operations.

C. BENEFITS OF STUDY

Sailors and Marines located in environments with limited Internet connectivity, such as onboard ships or at forward operating bases, have limited capabilities to scan their computer systems and other embedded devices for vulnerabilities. The benefit of this research is to provide a portable, intuitive, and practical framework for Sailors and Marines with little cyber expertise to assess their internal systems against an array of known cyber vulnerabilities. Two research scenarios tested a simulated Navy and Marine Corps system with the intent of identifying security vulnerabilities and providing the user with possible recommendations for mitigating the cyber-related threats.

D. THESIS OUTLINE

1. Chapter II: Background

Chapter II examines current Department of Defense (DoD) policies to describe cyberspace operations. This chapter also delineates DoD cyber red teams capabilities and their struggles to acquire and maintain resources to function effectively. Finally, a summary of independent academic, industrial, and government projects are reviewed to provide an understanding of the methodologies they employed to automate cybersecurity processes.

2. Chapter III: Design and Methodology

Chapter III defines the RTIB framework by proposing methods for host discovery, operating system fingerprinting, vulnerability scanning, firmware extraction, emulation, and user feedback. The chapter also describes a portable and automated tool that can seamlessly combine traditional red team actions into a single user-friendly tool.

3. Chapter IV: System Implementation

Chapter IV describes the virtual environment, testing scenarios, and subsequent results from the RTIB prototype. The chapter also lays out a step-by-step approach to implement the RTIB properly, track results, and view recommendations provided by an open source vulnerability assessment tool.

4. Chapter V: Conclusion and Future Work

Chapter V examines the results of the prototype system and the conclusions that can be gleaned from its implementation. Additionally, the chapter provides recommendations for future work to expand the capabilities of RTIB.

II. BACKGROUND

A. CYBERSPACE OPERATIONS AND THE CYBER KILL CHAIN

The United States DoD needs to effectively maneuver through the cyber domain and integrate its offensive and defensive cyber capabilities to meet the emerging demands of this operational environment. The DoD's actions in cyberspace should focus on maintaining a military advantage over other nation states and actors that can threaten U.S. national security and economic prosperity [4]. Joint Publication (JP) 3-12 defines cyberspace operations as “the employment of cyberspace capabilities where the primary purpose is to achieve objectives in or through cyberspace” [5]. Furthermore, JP 3-12 dissects cyberspace into a three-layer model: the physical network, the logical network, and cyber-personas. This thesis focuses on the logical network and the actions taken by cyber-operators.

To identify and actively defend against cyber threats a military commander may face, the DoD uses a formal methodology developed by the Lockheed Martin Corporation called the Cyber Kill Chain [6]. The Cyber Kill Chain stems from the dynamic targeting steps, commonly referred to as the Find, Fix, Track, Target, Engage and Assess (F2T2EA) process, found in the DoD's Joint Targeting Publication (JP 3-60) [7]. Lockheed Martin Corporation's kill chain steps are reconnaissance, weaponization, delivery, exploitation, installation, command & control, and actions on objectives [6]. Its purpose is to detail the sequence of events that must occur in order for an intruder to successfully conduct an attack on a specified computer system. The F2T2EA process is primarily used to offensively prosecute targets discovered during deliberate or dynamic targeting [7]. However, the Cyber Kill Chain was designed for use in a defensive role, where the DoD's ultimate goal is to prevent an adversary from attacking its cyber capability.

B. CYBERSPACE THREATS TO THE DEPARTMENT OF DEFENSE

The DoD is continuously facing attacks from Advanced Persistent Threats (APTs) that are sophisticated, well-resourced, highly motivated, and whose goal is to extract or compromise sensitive data [6]. An APT can conduct an attack over several years and target

“highly sensitive economic, proprietary, or national security information” [6]. In 2018, the GAO released a report to the U.S. Senate Committee on Armed Services detailing the increasing number of threats the DoD is facing due to a large number of complex computerized weapon systems developed for use against the U.S. arsenal [1]. The report outlined several steps the U.S. government can take to create robust weapon systems and provide a defense-in-depth approach against advanced cyberspace threats. The DoD Office of the Director, Operational Test & Evaluation (DOT&E) also provided a similar analysis of the risk of adversarial cyberspace operations in their FY17 annual report. The report stated that although “DoD cyber defenses are improving, ... [they] are not enough to stop adversarial teams from penetrating defenses, operating undetected, and degrading missions” [8]. The DOT&E report also noted that troops had a false sense of security during large-scale exercises because the cyber environment was not hostile enough to accurately depict the actual threat faced by most DoD systems against a persistent adversary [3], [8]. The concern is that DoD forces are not appropriately training against the cyberspace capabilities of peer or near-peer adversaries.

The recently released National Cyber Strategy (NCS) goes one step further and names explicitly Russia, China, Iran, and North Korea as APTs that have used cyberspace to steal intellectual property, participate in economic espionage, and “sow discord in our democratic processes” [9]. The DoD’s cyber strategy agrees with the NCS assessment and takes the extra step of defining its role in cyberspace as securing any sensitive data contained within DoD systems, deterring cyber-attacks against the United States, and conducting offensive cyberspace operations, if deemed necessary [4]. These strategic documents demonstrate the importance of identifying the threats facing the DoD, reducing vulnerabilities, and ultimately protecting the national interests of the United States.

C. DEPARTMENT OF DEFENSE RED TEAMS

A DoD cyber red team is composed of trained military, civilian, and contractor personnel who have the authority to mimic an adversary’s behavior by conducting exploitation techniques or cyber-attacks against a specific target or government capability [10]. They are officially defined as “an independent and focused threat-based effort by a

multi-disciplinary, opposing force using active and passive capabilities; based on formal, time-bounded tasking to expose and exploit information operations vulnerabilities of friendly forces as a means to improve the readiness of U.S. units, organizations, and facilities” [11]. DoD cyber red teams can be tasked to expose a target’s vulnerabilities; degrade, disrupt, or deny a user’s ability to access a particular cyber environment; test the techniques and skills of a defensive cyber force, and support operational security surveys. In the DoD, the NSA is the designated certification authority that manages the formal certification process for cyber red teams while U.S. Strategic Command maintains its accreditation [10]. Since FY16, the demand for cybersecurity assessments in the DoD has doubled as more weapon systems require an in-depth evaluation per the annual National Defense Authorization Act [12]. However, the DoD has recently faced a shortfall in providing enough certified cyber red teams that can realistically depict adversarial threats because of limited resources to thoroughly conduct proper evaluations [8]. To counter these shortfalls, a portable set of red-teaming tools could provide the DoD with a distributed solution that does not require the human resources or training of a traditional cyber red team which would relieve some of the burdens on DoD cyber red teams in conducting cybersecurity assessments.

D. ISSUES FACING DEPARTMENT OF DEFENSE CYBER RED TEAMS

1. Funding Limitations and Competing Industry

The DoD has invested heavily in cyberspace operations, as evidenced by their 2017–2021 spending plan, which brings the total cyber budget to \$34.6 billion. In FY16, \$500 million of the DoD budget was set aside to fund personnel in cybersecurity roles, but subsequent reports showed that cyber red teams did not receive any additional funds in terms of salary increases, hardware upgrades, or software purchases [13]. Although there are too many variables to determine the precise cost of training an individual in a DoD cyber red team, a recent Research and Development (RAND) Corporation study suggested that cyber training can cost the government well over \$200,000 per individual [14]. As the private sector continues to offer hefty salaries and compensation packages to

cybersecurity professionals, the DoD continues to find itself struggling to retain cyber red team personnel [13].

2. Time to Train

It can take as long as five to seven years for military members to be adequately screened and receive the extensive training required to become proficient in cyber red team operations [14]. However, the FY17 DOT&E observed that military personnel assigned to cyber billets are on a regular duty station rotation cycle, typically leaving after three years, which prevents them from gaining the required cyber experience to transition from journeyman to master [8]. Further exacerbating the problem, many journeymen leave the DoD shortly after fulfilling their initial military service obligation and are quickly hired by the civilian sector to serve as contractors for the DoD [8], [12]. The FY17 DOT&E assessment also recognized the importance of retaining skilled civilian and contractor personnel through selective hiring practices and job continuity [8]. They further recommended keeping personnel who can understand a government computer system fully and quickly recognize abnormalities on a network.

3. Limitations imposed on DoD Cyber Red Teams

DoD cyber red teams typically cannot fully exploit a target system due to restrictions imposed on them by a local military commander. For example, the commander will set forth Rules of Engagement that specifically prevent a red team from “[doing] any harm to the system” [13]. This apprehension from DoD leaders stems from a lack of understanding of the benefits of using a red team to expose their network deficiencies. Combatant commanders trained in traditional military tactics are reluctant to build realistic cyber threat scenarios and incorporate them into their regular training regimen because of the fear that the cyberspace operations may interrupt the command’s training objectives. Additionally, most DoD personnel forgo any emphasis on cybersecurity defense by treating it as an administrative function rather than a warfighting capability [12].

E. RED TEAM TOOLS

Several private cybersecurity firms are providing red team assessments to determine how well an organization adheres to its information technology or cybersecurity policies. These assessments are designed to covertly find vulnerabilities that may be caused by poorly trained users, incorrect security settings, or unpatched software. Although red teams and penetration testers use similar tools and techniques, red teams attempt to exploit an organization's system holistically by iterating through all phases of a cyber-attack. They typically try to gain access onto an organization's networks by conducting thorough cyber reconnaissance, establishing a foothold, elevating privileges, pivoting within the network, and then compromising any discovered sensitive data. Ultimately, a red team's goal is to observe how the target organization identifies and reacts to attacks. Various tools and frameworks are available that can help a red team accomplish this, including Cobalt Strike and Metasploit [13]. Additionally, some red teams create custom tools that automate specific repetitive or tedious processes when attempting to gain access to a network.

Cobalt Strike is a tool used during red team assessments and adversary simulations that focuses on benefiting incident response teams and improving security operations within an organization [15]. It offers social engineering attacks, covert command and control capabilities, lateral movements on the network after repurposing credentials, and even large-scale red team operations, but a one-year license can cost \$3,500 per user [15]. According to recent observations, Cobalt Strike is the preferred tool used by DoD cyber red teams [13], [15]. However, the DoD's dependency on using a commercial red team tool was criticized by the FY16 DOT&E assessment due to the possibility that an advanced adversary would be able to detect Cobalt Strike's well-known methods [13].

The Metasploit Framework (MSF) is another tool that offers a modular and flexible architecture that helps red teams and penetration testers quickly filter and create exploits for vulnerabilities discovered on a target system [16]. The Metasploit Project developed MSF in 2003, and it subsequently acquired by the Rapid7 computer security company in 2009 [17], [18]. It offers a Command-Line Interface (CLI) that allows the attacker to customize and launch exploits, load various payload modules, conduct an enumeration of users on the system, and perform multiple tasks using auxiliary modules. Although MSF

is an open-source project, its professional version starts at \$15,000 per year and offers access to all of its features including automating everyday tasks, creating custom payloads, and conducting brute force attacks [19], [20]. A free, open-source version is alternately available through the download of Kali Linux; however, Rapid7 provides no customer support for this.

F. RED TEAM IN A BOX BACKGROUND RESEARCH

Numerous academic researchers have spent their time and resources in developing tools and techniques to identify vulnerabilities found in software applications and cyber-physical systems. The DoD maintains a cadre of cyber red teams, but they lack a universally accepted tool that can identify, assess, and mitigate threats on software systems. The following section lists several recent endeavors and research projects that have attempted to improve a computer system's effectiveness by attacking its vulnerabilities from an adversarial point of view.

1. Vulnerability Assessment Framework

It is reasonable to conduct a thorough vulnerability assessment of a small network manually, but it becomes prohibitively cumbersome to assess a large and complex network due to the time, effort, and skill requirement. An automated approach, if possible, would be preferred. Today, there are several open-source vulnerability assessment tools available for download, such as Open Vulnerability Assessment System (OpenVAS), Nexpose Community, and Nikto. All of the available commercial and open-source tools have their strengths and weaknesses ranging from the user interface to the number of platforms supported and their ability to succinctly provide a detailed report of the discovered vulnerabilities. However, few products can integrate multiple tools and then objectively analyze the results when used simultaneously [21].

One solution is to use a vulnerability assessment framework that can integrate the devices and applications that communicate with each other by sorting scan results through a familiar management interface and then setting management policies [21]. The MSF is an example of a popular open-source tool that accomplishes this by discovering exploits and releasing payloads onto a target system. Furthermore, it is designed to use third-party

vulnerability assessment tools such as Nessus and Core Impact to scan for vulnerabilities on an individual system or a network of targets [22].

2. Dynamic Taint Analysis and Forward Symbolic Execution

An approach for vulnerability analysis that monitors code as it executes uses two powerful analysis techniques called dynamic taint analysis and dynamic forward symbolic execution. In [23], the authors explain that dynamic taint analysis monitors a computer program and determines what processes are affected by a specific source, such as a user's input. The authors also describe dynamic forward symbolic execution as the ability to analyze a program and determine which inputs cause a program to take a particular execution path. These two methods have been used to augment security applications such as malware analysis, input filter generation, test case generation, and vulnerability discovery [23]. A simple benefit of using dynamic taint analysis includes the prevention of code injection attacks by observing whether or not a user's input is ever processed.

Furthermore, automatically creating and applying filters that analyze a user's inputs and remove any known exploits is one method of using dynamic forward symbolic execution. Both of the techniques are used in conjunction to determine how data "flows through a malware binary, [and to] explore trigger-based behavior, and detect emulators," [23]. They work by generating a series of inputs that will cause different behaviors to occur within the same program.

3. Net-Nirikshak 1.0

Net-Nirikshak 1.0 is a vulnerability assessment and penetration testing tool developed for Indian banks that can automatically detect Structured Query Language (SQL) injection vulnerabilities along with any confidential information from the targeted system [24]. After its scan, it sends an email to a designated user that contains detailed information about the scan along with a vulnerability report in a Portable Document Format (PDF). Afterward, the tool removes all traces of the scan and the report summary to ensure the confidential data is not left behind and exposed to subsequent attacks.

The tool has eight modules that work in the following five phases: information gathering, scanning, vulnerability detection, exploitation, and report generation. It was designed to be a passive vulnerability detection system to prevent any interference with applications and services running on the target's operating systems. To keep its repository of vulnerabilities up-to-date, Net-Nirikshak 1.0 connects and interacts with the National Vulnerability Database maintained by the U.S. government [24]. Lastly, to reduce the user's workload, the tool provides an interactive window developed using Python that is fully automated and requires minimal knowledge to implement.

4. Pentest Box

In 2015, researchers from Northern Kentucky University developed a semi-automated system, called Pentest box, that scans and reports network vulnerabilities by using a miniaturized computer to host all of the necessary equipment needed for the penetration tester, cybersecurity professional, or system administrator [25]. In this case, the researchers were attempting to reduce the cost of conducting white hat hacking, or ethical penetration testing of an organization's information systems. They used Raspberry Pi computers as a cost-effective alternative to a commercial penetration testing device or software, with the intent of discovering vulnerabilities and protecting a company's information technology assets.

Their proposed device would connect directly to a corporate network's switch, behind any Internet router or firewall, but it would collect vulnerability data by establishing a Secure Shell (SSH) tunnel with a virtual private server on the Internet. The overall goal of the device is to allow a penetration tester to connect to the virtual private server and then conduct an assessment of the vulnerabilities through a web-based application. The Pentest box runs on Kali Linux and primarily uses Network Mapper (NMAP), MSF, and OpenVAS as its penetration testing tools. To automate the penetration testing process, it runs a script that conducts a reconnaissance scan of the Local Area Network (LAN), and then sends any hosts, open ports, and known vulnerabilities it discovers to the MSF database. These researchers, however, did not experiment any further than the reconnaissance phase of the

Cyber Kill Chain and only built a simple web interface with minimal assessment functionality.

5. Automated Intrusion Detection

Another study conducted by researchers at the Technological Educational Institute of Crete demonstrated their ability to combine several tools and technologies to create an automated intrusion detection process. In an attempt to raise cybersecurity awareness, they referenced recent international cyber-attacks as examples, including how the Islamic State Hacking Division, also known as the Cyber Caliphate, were able to “deface 600 Russian sites in an apparently automated fashion,” [26]. They proposed designing an automated tool that could help mitigate cyber attacks by combining traditional penetration testing techniques with NMAP, MSF, and Python scripts to expose vulnerabilities on an unpatched system. The penetration testing process they used was comprised of footprinting, scanning, enumerating, gaining access, and reporting any results back to the attacker. During the footprinting and scanning phases, they used NMAP to identify hosts and conduct port scanning and used banner grabbing techniques to determine the type of services provided by each host. Afterward, they referenced the Common Vulnerabilities and Exposures (CVE) database maintained by the MITRE Corporation to discover any possible and unpatched vulnerabilities on the hosts. After this, the red team could use MSF to gain access to the target system by choosing and configuring an exploit, checking for susceptibility, configuring a payload, choosing an encoding system, and finally executing the exploit.

The researchers tested their framework by using Python to create a script to run an automated process that combined all of their tools and techniques. For example, they developed a Python script that exploited a Tomcat servlet vulnerability (CVE 2009-4188) by discovering the server’s listening port via NMAP. It then ran an MSF resource file that found a default username and password with manager privileges, extracted the credentials, and finally deployed a module that gave the attacker a meterpreter shell. Here, the Python script was designed to attack one target, but it could easily be scaled to infiltrate a more extensive computer network. Due to the ease with which they gained remote access to an

unpatched system, they recommended enacting security procedures to defend computer systems and networks.

6. Firmalice

The DoN depends on a large number of cyber-physical systems that often operate in harsh and austere environments. Most commercially available cybersecurity assessment tools can scan network devices connected via an IP-based interface, but very few tools exist that inspect embedded devices such as those found on ships, aircraft, or motor vehicles [27]. For operational security reasons, many DoN embedded devices are typically not directly connected to the Internet, but this will not prevent viruses from infecting the devices. The propagation of the Stuxnet worm in Iran, which spread through the use of infected Universal Serial Bus (USB) devices, proved that a zero-day cyber-attack could be executed even onto an air-gapped network [28].

Embedded devices are specifically designed to accomplish a particular purpose and “often interact with the physical world through a number of peripherals connected to sensors and actuators,” [29]. Such a device has low-level software called firmware that acts as a device’s operating system (OS) and controls most of its functions. However, firmware can be vulnerable to a large number of software errors such as command injection vulnerabilities, memory corruption flaws, and application logic flaws (e.g., authentication bypass, which is colloquially known as a backdoor) [30]. Backdoors are sometimes intentionally created by software developers in order to gain access to the device for maintenance and upgrade purposes. Further complicating their security, many embedded devices contain encrypted proprietary software that runs directly on the device’s hardware.

To address these shortcomings, researchers at the University of California, Santa Barbara developed Firmalice, a “binary analysis framework to support the analysis of firmware running on embedded devices” that automates the process of examining software to expose any logic flaws [30]. Their model works by loading an image of the firmware from the target device, analyzing the security policy, and then driving a symbolic execution engine by using static analysis which is checked against the security policy to discover violations. However, the researchers noted that Firmalice is a large, sophisticated, and

sluggish tool that contains over 14,000 lines of Python code and 3,000 lines of C code, and executes via a single thread against a single device [30]. Another constraint that impacts Firmalice's effectiveness is its inability to identify math-based backdoors that accept multiple valid solutions or a malware program that dynamically evades detection.

7. Fuzzing

Certain types of vulnerabilities may cause embedded devices to behave differently than typical desktop applications. Most embedded devices often lack the layers of defense, such as memory isolation, protection mechanisms, and memory structures integrity checks, that a hardened desktop computer may employ [29]. Because of these shortcomings, embedded devices are often more susceptible to memory corruption. Fuzzing is a standard method of monitoring software behavior for this vulnerability, but most fuzzing tools were developed to run on desktop computers instead of embedded devices. In recent work, researchers at EURECOM were able to demonstrate the decreased effectiveness of fuzzing tools on embedded devices due to their limited I/O and computing resources [29]. The main obstacles they faced while conducting fuzzing experiments on embedded devices were fault detection, performance and scalability, and instrumentation. To address these problems, they developed six different solutions with varying levels of capabilities and limitations. These include static instrumentation, binary rewriting, physical re-hosting, full emulation, partial emulation, and hardware-supported instrumentation.

The first three mitigation solutions require some level of firmware modification, which may not be an option when conducting a security analysis on proprietary or third-party software. The last three techniques do not modify the original firmware but do require additional tools such as a Quick Emulator (QEMU) or special hardware features found on many ARM and Intel microprocessors that can trace the branching of instructions and their memory access. The researchers concluded that full emulation was the best strategy, but it is often difficult to achieve due to a lack of available firmware documentation and a high level of manual effort to ensure the emulator functions appropriately. They also determined that relying on fuzzing techniques alone would miss most vulnerabilities, but partial

emulation is a good compromise between fuzzing and emulation that can accurately detect vulnerabilities despite the lack of a physical device.

8. Mayhem Cyber Reasoning System

The Mayhem Cyber Reasoning System is yet another vulnerability analysis tool that was recently developed by researchers at Carnegie Mellon University. Mayhem is slightly different from previous vulnerability analysis tools in that it autonomously searches and fixes vulnerabilities in executable programs without the need for human intervention [31]. Mayhem works by “actively managing execution paths without exhausting memory, and reasoning about symbolic memory indices,” which means it searches for bugs at the binary level by using hybrid symbolic execution and index-based memory modeling [32]. Specifically, Mayhem searches for exploits by determining whether a computer bug can redirect an instruction pointer, whether or not malicious code can be implanted in memory, and if that code can then be executed. Hybrid symbolic execution alternates between online and offline modes to produce a more efficient process for managing memory without the need to perform previously executed instructions [32]. Mayhem also forks every time it encounters a conditional branch and saves this new path into its memory [31].

With index-based memory modeling, Mayhem changes the way it treats symbolic memory based on the value of the index [32]. Interestingly, Mayhem does not attempt to bypass address space layout randomization or data execution prevention because the researchers believed that a broken exploit defeated by these defenses could quickly and automatically adapt to circumvent them during future attacks. Mayhem also manipulated open-sourced fuzzing tools to search for bugs at the binary level during the 2016 Defense Advanced Research Projects Agency (DARPA) Cyber Grand Challenge, but the overall process was slow with only 65 out of 131 bugs found in 24 hours [31]. Unfortunately, Mayhem resides in a large server rack, which makes it infeasible to use as a portable vulnerability analysis or red teaming tool.

9. RTIB on Industrial Control Systems

The rapid proliferation of cheap and general-purpose networked industrial devices has unintentionally introduced a host of cybersecurity vulnerabilities. In 2015, there were 295 cyber-attacks reported to the Industrial Control Systems (ICS)-Cyber Emergency Response Team (CERT) [33]. There are two types of ICS: Distributed Control System (DCS) and Supervisory Control and Data Acquisition (SCADA), each controlled in a decentralized or centralized manner, respectively. To provide a defense against process-aware attacks on ICS, researchers developed a method for detecting specific anomalies used as an early indicator of malicious cyber activity [33]. They noted that traditional network security approaches would fail to address the threats faced by an ICS due to the coupling of cyber and physical components. They also assumed the attack designer was a technically proficient APT with sufficient resources, but the attack launcher was a “technically incapable adversary,” such as a janitor, who was coerced or financially motivated to conduct an attack on an ICS facility by connecting a “Red-Team-In-a-Box (RTIB)” that would automatically launch the cyber-attack [33]. Of note, this RTIB definition differs from that of this thesis, since the researchers are describing a possible delivery method to launch an attack on a network instead of developing a portable cyber red teaming tool.

The researchers used a mobile phone with a Linux-based OS that would automatically launch malicious payloads targeting Programmable Logic Controllers (PLCs) connected to the victim network. Although the rest of their research focused on using machine learning to create robust and fault-tolerant control systems, they recommended a mitigation strategy of using secondary or backup systems that were sufficiently different from the primary controller to increase the architecture’s complexity, thereby increasing redundancy and decreasing the system’s susceptibility from an attack. Red team tool designers must take into account the possibility that software designers may have purposefully introduced both hardware and software complexity into their device to thwart unauthorized access.

10. Driller

Most of the researchers who designed FIRMALICE also created Driller, a hybrid excavation tool that uses fuzzing techniques and concolic execution to discover deep bugs in complex computer systems [34]. Fuzzing requires a broad set of manually created input test cases to drive program execution that will identify security flaws in a system or program. Some of the techniques the researchers leveraged included guided fuzzing, which attempts to identify code that is susceptible to buffer overflows, and Whitebox fuzzing, which collects input constraints, negates them and attempts to generate inputs to force the program to take a new path. Concolic execution, on the other hand, can easily be impeded by “path explosion,” which limits its ability to explore all paths as the number of conditional branches increases based on the provided input [34].

Driller works by combining the speed at which fuzzing can exercise the binary compartments of an application with the capabilities of “concolic execution ... to generate inputs which satisfy the complex checks separating compartments” until an input causes a crash [34]. In doing so, Driller was able to find bugs that fuzzing and concolic execution by themselves were unable to locate. Driller was used during DARPA’s 2016 Cyber Grand Challenge and found nine bugs that traditional fuzzing techniques would have overlooked.

11. Angr

The Angr researchers focused on binary code since many interpreted languages, core OS constructs, and Internet of Things (IoT) devices all compile down to binary instructions [35]. Angr allows users “to reproduce results in the fields of vulnerability discovery, exploit replaying, automatic exploit generation, compilation of ROP shellcode, and exploit hardening” [35]. To address some of the shortcomings of static and dynamic vulnerability discovery, Angr analyzes a program and develops a control-flow graph that uses “forced execution, backwards slicing, and symbolic execution” to track all of the jumps a program may take [35].

Unfortunately, Angr generates a large amount of data, which ultimately slows down its performance. Furthermore, Angr’s algorithm cannot handle code that can only be reached through unrecoverable indirect jumps or sections that are executed but whose

result is not used in any other portion of the program, which is also known as dead code. Despite these issues, Angr can provide several techniques that can automatically identify vulnerabilities in binary code.

12. PovFuzzer, Rex, Colorguard, and Patcherex

The same group of researchers that developed Firmalice, Driller, and Angr also designed two tools that execute a set of binary code repeatedly until a crash is discovered and then subsequently create a set of exploits. These techniques work by modifying the memory registers of computer bugs after finding them using static analysis, fuzzing, and symbolic execution. The automatic exploitation systems, called PovFuzzer and Rex, were able to find most simple bugs using a brute force repetitive process and Return-Oriented Programming (ROP) shellcode but failed to consider a buffer over-read that would not cause a system crash [36].

The researchers from Team Shellphish (one of the teams from the DARPA Cyber Grand Challenge) also created a third system called Colorguard that “traces the execution of a binary with a particular input and checks for flag data being leaked out,” [36]. Interestingly, Team Shellphish developed a methodology for patching the vulnerabilities of the target binary code through several techniques. This approach, called Patcherex, was built on top of Angr and uses generic binary hardening, optimization, and anti-analysis techniques to prevent adversarial intrusions. The result is a single, typically one byte, modification to the binary code that prevented the system from crashing.

13. Angr on Industrial Internet of Things

A separate DoD team at Space and Naval Warfare Systems Command (SPAWAR) leveraged the University of California, Santa Barbara’s Python-based Angr framework and cyber reasoning system, along with open-source virtualization tools, to perform limited automated analysis on embedded systems and IIoT devices [37]. They intended to conduct an automated Industrial Internet of Things (IIoT) firmware analysis in search of malicious content.

To accomplish this, they first extracted the firmware from an IIoT device and then emulated the software in a separate operational environment not directly connected to the original hardware. Afterward, the team used Angr and Driller to perform static, dynamic, and symbolic analysis. Additionally, the SPAWAR team used American Fuzzy Lop (AFL), OpenPLC, Firmadyne, and QEMU to expose firmware vulnerabilities, mitigate them, and ultimately improve the overall security posture for IIoT devices. This approach led to previously undiscovered authentication bypass and non-existent stack protection vulnerabilities in numerous IIoT devices.

Table 1 provides a summary of all of the background research related to automated vulnerability analysis.

Table 1. Summary of Background Research

Background Research Topic	Description	Year Published	Additional Tools Used	Programming Language	Automation Level	Network Capability	Operating System	Is the System Portable?	Cyber Physical System Used	User Feedback
Vulnerability Assessment Framework	Implement an automated network vulnerability assessment framework that integrates various tools	2007	MSF with Nessus & Core Impact	Various	Partial	IP-based	Multiple	Yes	N/A	System Report
Dynamic Taint Analysis and Forward Symbolic Execution	Describe the algorithms for dynamic taint analysis and forward symbolic execution	2010	Custom Tool (only tested in lab)	Custom (SimplL)	Partial	Not Tested	Not Applicable	Yes	N/A	Not Developed
Net-Nirikshak 1.0	Vulnerability assessment and penetration testing tool to help banks assess their services and analyze their security posture	2014	Gmail Services, National Vulnerability Database	SQL, Python	Partial	IP-based	Not Stated	Yes	N/A	Email
Pentest Box	A semi-automated system that can scan and report on the vulnerabilities of a network	2015	NMAP, MSF, OpenVAS	Ruby	Partial	IP-based	Kali Linux	Yes	Raspberry Pi	Webpage
Automated Intrusion Detection	A tool capable of automating a cyber-attack using open source tools	2015	NMAP, MSF	Python	Partial	IP-based	Windows, Linux	Yes	N/A	Python Print Statement
Firmalice	A binary analysis framework for firmware running on embedded devices	2015	None	Python, C	Partial	IP-based & Standalone Devices	Binary Blob Firmware, Linux	Yes	Smart Meter, Camera, Printer	System Report
Fuzzing	Demonstrate that memory corruptions on embedded devices behave differently than on desktop systems	2018	boofuzz, QEMU	Python, C	Partial	IP-based & Standalone Devices	Linux	Varies	Various	Avatar and PANDA reports
Mayhem Cyber Reasoning System	A system that automatically finds exploitable bugs in binary programs	2016	Custom Tools	C/C++, Ocaml, BAP	Full	Intranet	Windows, Linux, DECREE OS	No	Custom Server Blade	System Report
RTIB on ICS	Develop a process-aware defense and mitigation strategy delivered through small payloads	2016	MATLAB, Wireshark, CODESYS	Not Stated	Full	Intranet	Firmware, Linux	Yes	PLCs	System Report
Driller	A hybrid vulnerability excavation tool that uses fuzzing and concolic execution to find bugs	2016	AFL, QEMU, Angr	Python, C/C++	Full	Intranet	DECREE OS is implied	No	Experiments conducted on AMD64 processors	System Report
Angr	An open source system that implements a number of techniques for the automated identification and exploitation of vulnerabilities in binaries	2016	AFL, libVEX	Python	Partial	IP-based & Standalone Devices	Windows, Linux, DECREE OS	Yes	IoT Devices	IPython Interactive Shell
PovFuzzer, Rex, Colorguard, Patherex	Automatic exploitation systems used to find simple bugs, trace the execution of an input, and patch vulnerabilities	2016	Angr, Driller	Python	Full (when used with Shellphish Cyber Reasoning System)	Intranet	DECREE OS	No	Custom Server Blade	System Report
Angr on IIoT	Perform semi-automated firmware analysis on embedded systems in search of vulnerabilities	2017	AFL, OpenPLC, Firmadyne, QEMU, Binwalk, Sasquatch, Jefferson	Python	Partial	IP-based	Linux	Yes	PLCs, Raspberry Pi	IPython Interactive Shell

G. CHAPTER SUMMARY

This chapter provided a broad overview of the DoD's definition of cyberspace operations, the Cyber Kill Chain methodology, and some of the cyber threats the DoD is facing today. Furthermore, it described DoD cyber red teams and some of the issues they face, including funding, training requirements, competition from the civilian sector, and limitations imposed during evaluations. It also briefly described some of the tools used by DoD cyber red teams, and it detailed numerous research papers and projects that have conducted automated software analysis for security vulnerabilities (summarized in Table 1). The next chapter describes the overall framework of RTIB.

III. DESIGN AND METHODOLOGY

A. OVERVIEW

The previous chapter described several tools and technologies that were developed, refined, and employed to find vulnerabilities on networks and embedded devices. This chapter focuses on the RTIB framework by proposing methods for host discovery, OS fingerprinting, vulnerability scanning, firmware extraction, emulation, and user feedback. We have designed a portable and automated tool that can seamlessly combine the steps above into a single user-friendly device. This device detects and exposes cybersecurity vulnerabilities by systematically analyzing devices that are not commonly monitored by network administrators but are still vulnerable to cyber-attacks.

RTIB is designed to test DoD networked and embedded devices not directly connected to the Internet. These devices include various DoD mission and non-mission critical computer systems onboard aircraft, ground vehicles, ships, and submersibles. Furthermore, to focus the scope of this research, it is assumed that the targeted devices may receive occasional software or firmware updates via a standalone intermediary device such as a laptop or a USB flash drive.

The RTIB uses open-source frameworks and tools to reap the cost, security, and flexibility benefits of crowd-sourced and peer-reviewed software. The goal is to leverage the open-source community's ability to continually check for flaws in software rather than providing cybersecurity through obfuscation or behind a private company's intellectual property copyright. Future iterations of RTIB will also be capable of using some of the open-source solutions supplied by the NSA to assess "the security state of an ARM-based device" or "quantitatively measure the effectiveness of [a device's] security posture" by using the Maplesyrup and Unfetter programs, respectively; however, those capabilities will be outside the scope of this thesis [38]. Lastly, this thesis does not focus on the physical interoperability of RTIB with cyber-physical devices. Those connections are abstracted away, and this thesis assumes a physical connection between RTIB and the target devices. Section B of Chapter IV discusses how virtual machines (VMs) are used to simulate the

physical connections between devices. Figure 1 shows a simple diagram of an RTIB device connected to a host system on a local network.

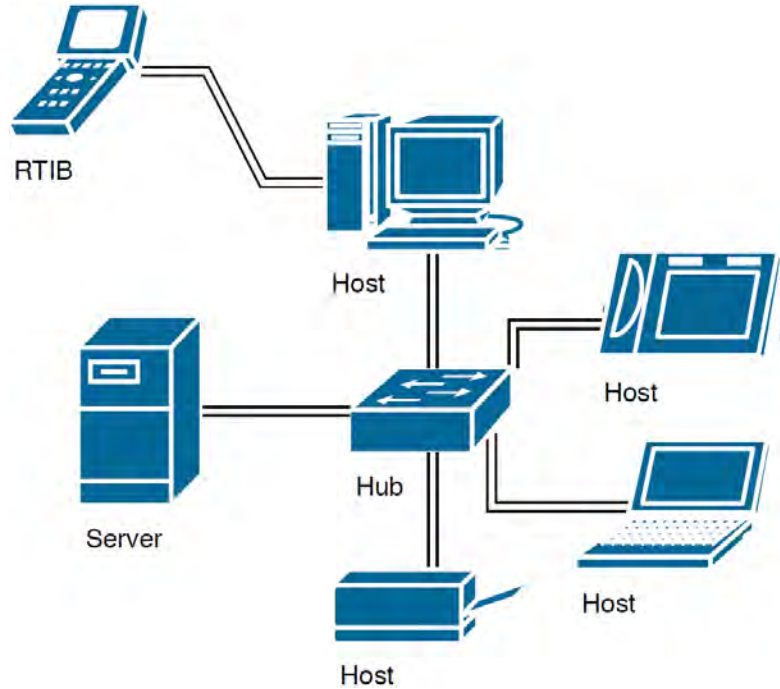


Figure 1. Sample RTIB Layout Diagram

B. PENETRATION TESTING DISTRIBUTIONS

Penetration testing distributions are frequently used to simulate a cyber-attack on a friendly system designated as a target. Currently, there are several open-source distributions used by ethical computer hackers and security experts wishing to conduct security evaluations on vulnerable computer systems. For instance, Offensive Security developed one of the most popular and advanced penetration testing platforms in 2013, called Kali Linux [39]. The Kali Linux project originated from the BackTrack Linux distribution and has over six hundred pre-installed tools and packages that range from password cracking to port scanning. Another small and versatile distribution is Raspberry Pwn developed by Pwnie Express. Raspberry Pwn gives a penetration tester the ability to use many of the same tools offered by Kali Linux, but on a small circuit board called a Raspberry Pi [40]. The main benefit of these distributions from an RTIB point-of-view is

that they are small enough to be used on a laptop, personal digital assistant, or even a single-board computer. Table 2 contains several examples of different penetration testing distributions and their system requirements.

Table 2. Common Penetration Testing Distributions

Distribution Name	Year Released	CPU Architecture Supported	Operating System	Disk Space Required	RAM Required	Boot Method
BackBox	2010	i386 or x86-64	Ubuntu LTS	4.4 GB	512 MB	DVD-ROM or USB
BlackArch	2013	i686 or x86-64	Arch Linux	At least 1 GB. 20 GB recommended	512 MB. 2 GB recommended	USB
Kali Linux	2013	i386 or x86-64	Debian	20 GB	1 GB min, 2 GB+ recommended	CD-DVD or USB
Parrot Security OS	2013	i386 or x86-64	Debian	16 GB	At least 256MB for i386 and 320MB for AMD64. 512MB recommended	Legacy BIOS preferred
Pentoo Linux	2005	i686 or x86-64	Gentoo Linux	20 GB. 30 GB+ recommended	2 GB	DVD
Raspberry Pwn	2012	i386 or x86-64	Debian	4 GB	1 GB	SD card

Adapted from [40], [41].

Unfortunately, all of these distributions require an intimate knowledge of the pre-installed tools, which can be daunting for a novice user or someone unfamiliar with penetration testing. Specifically, the distributions require the user to be comfortable navigating through the CLI of a Unix system as opposed to using a user-friendly Graphical User Interface (GUI). To reduce the user's learning curve and to make the system more comfortable to use, our RTIB uses scripts through a GUI to execute everyday tasks while conducting its red team assessment on a target device. This shields novice users from becoming overwhelmed by the Unix CLI and automates portions of the red team process. Furthermore, this thesis focuses on using Kali Linux as the primary RTIB distribution due to the high number of pre-installed tools, available support documents, and its robust online community.

C. HOST DISCOVERY AND OPERATING SYSTEM FINGERPRINTING

Scanning and enumerating a networked environment is an essential step in determining which services, ports, and applications are accessible and available. Techniques that allow a red team member to discover active hosts and services on a network include ping sweeps, port scanning, banner grabbing, and OS fingerprinting. One of RTIB's first functions is to determine the type of host it is scanning through a simple set

of user commands. Ideally, a preliminary scan will allow RTIB to accurately identify the OS on each host by analyzing numerous markers historically aligned with an OS's default settings. Many conventional operating systems can be passively identified by examining captured Transmission Control Protocol (TCP) packets. For example, p0f is a fingerprinting tool that compares a packet's Time To Live (TTL) value, IP header flags, Maximum Segment Size (MSS), and window size to ascertain what type of OS is actively communicating with other devices on a network [41].

Alternatively, there are more active approaches used by other network scanning tools for OS fingerprinting. For instance, NMAP compares the responses it receives from TCP and User Datagram Protocol (UDP) packet requests against a database of over 2,600 known OS fingerprints [42]. Nonetheless, quickly discovering hosts on a network and accurately identifying their OS will enable RTIB to tailor its vulnerability scan, decrease the number of unnecessary follow-up scans, and ultimately reduce the number of false positives. Table 3 provides a summary of some of the features commonly used by network scanners to distinguish different operating systems.

Table 3. Common OS Fingerprint Values

Operating System	TTL (milliseconds)	Window Size (bytes)	Packet Size (bytes)	No Operation Flag (NOP)	Don't Fragment Flag (DF)	Time Stamp Option
Cisco IOS	255	4,128	44	0	0	0
Linux	64	5,840	60	1	1	1
MAC	64	65,535	60	1	1	1
OpenBSD	64	16,384	64	1	1	1
Solaris 7	255	8,760	44	0	1	0
Windows 2000	128	16,384	48	1	1	0
Windows 7	128	8,192	52	1	1	0
Windows XP	128	65,535	48	1	1	0

Adapted from [43].

However, RTIB faces the likely prospect of not being able to correctly identify a target OS due to fingerprint value ambiguities or encountering a proprietary OS whose values are not on a network scanning database. This possibility arises from the number of independent government contractors that have developed unique software solutions for

various military projects. In this situation, RTIB will alert the user that the host OS is either unknown or too ambiguous to be accurately identified. The RTIB user will then decide whether to conduct another scan or search through the device's firmware. Although current firmware analysis programs provide a more in-depth analysis of the target system, each framework produces a unique user feedback report that RTIB would compile to a standardized format within RTIB. That capability is discussed further in Chapter V.

D. VULNERABILITY SCANNING

RTIB has the advantage of being able to connect directly onto a target host or network, thereby increasing the speed and accuracy of its vulnerability scan which allows it to forgo the potentially cumbersome process of attempting to gain an initial foothold on a target system. However, this does not guarantee complete and unfettered access to the host. An adequately defended host or network will deny a potential attacker, whether acting with malicious intent or not, from accessing any valuable data. As expected, there are a variety of available tools that allow a user to automatically scan for vulnerabilities, including OpenVAS, Nessus, Core Impact, and Nikto. To keep RTIB as a practical and inexpensive tool, we use the open-source vulnerability scanner OpenVAS and eschew the pricey licensing fees of Nessus and Core Impact.

Fortunately, OpenVAS is designed to work as a module within the MSF which allows an RTIB user the opportunity to create targets and run vulnerability scans while using a single CLI. Of note, launching OpenVAS using a traditional command-line argument within Kali Linux automatically starts a web-based GUI called the Greenbone Security Assistant (GSA). The GSA contains several tabs to facilitate vulnerability scans, including configuring targets, filtering results, and identifying the OS of each host.

Other open-source frameworks provide a high level of automation for red teams wishing to conduct vulnerability scans. For example, AutoSploit introduced in 2018 is a tool that collects vulnerable targets via the Shodan, Censys, and Zoomeye online search engines and then attempts to run MSF modules to exploit them by creating "reverse TCP shells and/or Meterpreter sessions" [44]. However, this framework would fail to be a useful RTIB vulnerability scanning tool due to its requirement to use databases found on the

Internet at runtime. Alternatively, a Windows OS specific tool called PowerSploit released in 2014 was PowerShell's first offensive security framework [45]. Although PowerSploit contains a repository of capabilities that leverages the functionality of PowerShell on a Windows machine, RTIB will use a Unix based framework to reduce the complexity of swapping between PowerShell and Unix commands and increase its compatibility with other available RT tools.

OpenVAS, through the Greenbone network, uses the National Vulnerability Database (NVD) which is maintained by the National Institute of Standards and Technology as a repository to aid in the automation of vulnerability management. Specifically, the NVD provides OpenVAS with an updated collection of CVEs, misconfigurations, and security flaws to help red team members quickly analyze hosts. Unfortunately, using OpenVAS or any other vulnerability scanner does not necessarily offer a panacea in identifying all potential vulnerabilities found on a target system. For example, an information security specialist was able to demonstrate that OpenVAS and Nessus failed to detect 51.6% of known vulnerabilities on the CVE database, as seen in Figure 2; however, he admitted that this discrepancy could be that the vulnerability assessment vendors are ignoring old software vulnerabilities that only exist in deprecated OS distributions such as Mandriva Linux [46]. Regardless, this does not mean that the information provided by the vulnerability assessment tools should be rejected, but rather that they only offer a glimpse of the possible vulnerabilities maintained inside a threat database. After conducting a vulnerability scan, any abnormalities discovered by RTIB are saved locally on the device which allows the user the ability to conduct further analysis of the data after the completion of the initial RTIB assessment.

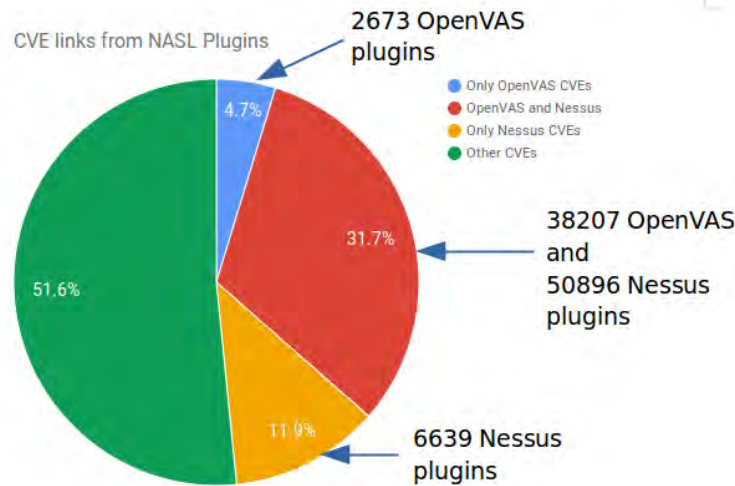


Figure 2. CVEs discovered by OpenVAS and Nessus. Source [46].

Interestingly, RTIB will also need to receive periodic CVE updates via the Internet to provide the most current and relevant protection against cyber threats. A potential “Catch-22” situation arises for RTIB since there exists the possibility that RTIB could inadvertently infect an isolated and malware-free system during a routine vulnerability scan (if the RTIB device were itself infected with malware). However, the possibility of this threat is low and should not hinder a user from conducting a red team analysis on a target system. The purpose of RTIB is to expose vulnerabilities and harden DoD computer systems. Thus, the benefits of taking an active cybersecurity approach outweigh the risks associated with possibly infecting the target host or network. RTIB merely is one layer in the defense-in-depth model, which employs physical, technical, and administrative security controls. RTIB is a tool that aids in identifying possible attack vectors and should not be treated as a comprehensive security solution. It merely provides a snapshot for users to measure the effectiveness of their cybersecurity operations.

E. FIRMWARE EXTRACTION AND EMULATION

Merely discovering the host or network and running a vulnerability assessment scan may not achieve all of the desired security goals for proper red team analysis. Although being able to connect to the host directly does have its advantages, some systems may prove to be more complicated due to their unique software architecture. Additionally, RTIB may

encounter target devices with firmware that has neither been seen by open-source developers nor commercial tool vendors. To address this shortfall, RTIB uses a range of other tools and techniques that are still capable of conducting a vulnerability assessment without the benefit of a readily accessible vulnerability database.

In this case, RTIB must extract and emulate the firmware on the target device. Several tools discussed in Section F of Chapter II can accomplish this type of analysis, including the open-sourced QEMU and Firmadyne. Another tool available on the Kali Linux distribution called Binwalk searches “a given binary image for embedded files and code” located inside of a firmware image, including a Unified Extensible Firmware Interface (UEFI) [47]. A short Python script can be used with Binwalk to automatically perform a system scan and return the results to the user.

The Firmware Analysis Toolkit, or FAT, is another tool built on top of Firmadyne, Binwalk, and several other open-source programs [48]. FAT was constructed to identify and analyze vulnerabilities in IoT and embedded devices. It automates Firmadyne by running a simple Python script and only requires the firmware filename or path as an argument on the CLI. Integrating well developed Python scripting tools such as Binwalk and FAT with RTIB will significantly simplify the user’s ability to conduct vulnerability assessments on firmware images. Figure 3 shows a screenshot of a FAT run on a binary file named “DIR850LB1_FW210WWb03.bin” which is the command line argument for the Python script. The script then asks for a brand name for database storage purposes. Finally, it runs the emulated firmware and gives the user the ability to analyze the image further.

```
$ python fat.py DIR850LB1_FW210WWb03.bin

      _ _ _ _ _
     / _ \   / _ \
    | | | | | | | |
    | | | | | | | |
    | | | | | | | |
    | | | | | | | |

Welcome to the Firmware Analysis Toolkit - v0.2
Offensive IoT Exploitation Training - http://offensiveiotexploitation.com
By Attify - https://attify.com | @attifyme

[?] Enter the name or absolute path of the firmware you want to analyse : DIR850LB1_FW210WWb03.bin
[?] Enter the brand of the firmware : dlink
[+] Now going to extract the firmware. Hold on..
[+] Firmware : DIR850LB1_FW210WWb03.bin
[+] Brand : dlink
[+] Database image ID : 1
[+] Identifying architecture
[+] Architecture : mipseb
[+] Storing filesystem in database
[!] Filesystem already exists
[+] Building QEMU disk image
[+] Setting up the network connection, please standby
[+] Network interfaces : [('br0', '192.168.0.1'), ('br1', '192.168.7.1')]
[+] Running the firmware finally
[+] command line : sudo /home/ec/firmadyne/scratch/1/run.sh
[*] Press ENTER to run the firmware...
```

Figure 3. Firmware Analysis Toolkit Screenshot. Source: [48].

Nevertheless, there are some limitations when using emulators and other firmware analysis tools. First, many cyber-physical devices are designed to control motors or servos, thus making it very difficult to replicate any feedback from those systems in an emulated environment. Second, the target system may be too large or complex and may exceed the memory storage capacity of the RTIB. Lastly, the emulator within RTIB may enter an endless loop or experience a system crash during its testing. In any of these cases, RTIB will attempt to provide exception handling for the runtime errors through the use of signals to interrupt and kill processes.

F. SYSTEM AUTOMATION AND FEEDBACK

An essential aspect of conducting a red team analysis on a target host or network is the ability to quickly enumerate through files, directories, or binary code in search of vulnerabilities. Using automated tools and techniques dramatically improves a user's ability to capture significant weaknesses within the system and reduces the risks caused by

human error. In this case, RTIB is meant to be used by individuals without any profound knowledge of red teaming techniques or penetration testing, but they do have a basic understanding of cybersecurity fundamentals, based on required annual cyber awareness training. As mentioned before in this chapter, the intent is to use a GUI to provide a level of abstraction for the user just above the CLI. The RTIB front-end will give the user a set of options to conduct various portions of the vulnerability scan and assessment, while the back-end will run Python scripts with the previously mentioned tools on a Kali Linux distribution. The front-end GUI design is intended to be simple and provide limited functionality with the overall goal of accomplishing RTIB's main red teaming tasks. Given the limitation of time and resources for this work, it is essential to note that this thesis has focused on developing a proof-of-concept tool without conducting any usability testing on real users.

RTIB provides the user with feedback through a summary of the results after it conducts its assessment. As mentioned before, RTIB was designed to combine outputs from all of the individual tools it uses, and to provide a comprehensive summary to the user in a human-readable format. The biggest obstacle in achieving this goal is the disparity between the output data types generated by each tool. For instance, as the "User Feedback" column in Table 1 demonstrates, the output from vulnerability assessment tools can range from a simple system report on a text file to an interactive Python shell. Consolidating all of these different data types onto a single form or display is a difficult undertaking and will be further discussed in Chapter V as future work. Regardless, the RTIB summary will not only describe the vulnerabilities it discovers but also provide the user with recommended courses of action. For example, suppose an RTIB scan returns several vulnerabilities that were identified as malware by the CVE database. Those results would then be presented to the user along with instructions to proceed to the NVD website for a listing of advisories, solutions, and downloadable tools. The updates or patches could then be downloaded onto the target system, ensuring the identified vulnerabilities have been fixed.

G. SUMMARY

This chapter provided an RTIB framework by proposing methods for host discovery, vulnerability scanning, firmware extraction, emulation, and user feedback. Numerous open-source distributions are well suited for conducting red team assessments on a target system due to their pre-installed tools and applications, but RTIB uses Kali Linux as a testbed. Regarding host discovery, RTIB uses both active and passive means to identify and classify the hosts it detects on a target network. The goal of host discovery is to allow all follow-on steps, such as vulnerability scans, to be more focused and productive. If the host contains proprietary software or an unrecognized version of firmware, the RTIB user can leverage one of the firmware analysis tools to carefully sift through the binary code. Afterward, RTIB provides the user with feedback and a possible recommendation to address and fix the discovered vulnerabilities. The next chapter will describe how VMs are used to test and record RTIB's performance using scripted scenarios while scanning for vulnerabilities on a rudimentary network.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. SYSTEM IMPLEMENTATION

A. OVERVIEW

Chapter III described the RTIB framework by proposing methods for host discovery, OS fingerprinting, vulnerability scanning, firmware extraction, emulation, and user feedback. This chapter describes a prototype implementation of RTIB, including the virtual environment, testing scenario, and initial test results. The goal is to outline the step-by-step approach to implementing RTIB properly.

B. ENVIRONMENT

1. Background

As mentioned in Section B of Chapter III, we used the Kali Linux distribution as the foundation for the RTIB due to its pre-installed tools as well as the availability of a LAN of VMs at the Naval Postgraduate School (NPS) Cyber Battle Lab (CYBL). The CYBL is a Type I hypervisor physically located on the NPS campus that contains several VM templates for servers and other desktop operating systems. Figure 4 shows an example of the architecture of a typical Type I hypervisor. Accessing the CYBL is accomplished through either an online browser user interface or via a VMware Horizon Client desktop application. The VMs for this experiment ran various Windows and Linux distributions including Windows 7 Professional (Service Pack 1), Windows XP Professional (Service Pack 3), and Ubuntu 8.10 (Intrepid Ibex) running a Linux 2.6.27-7 kernel.

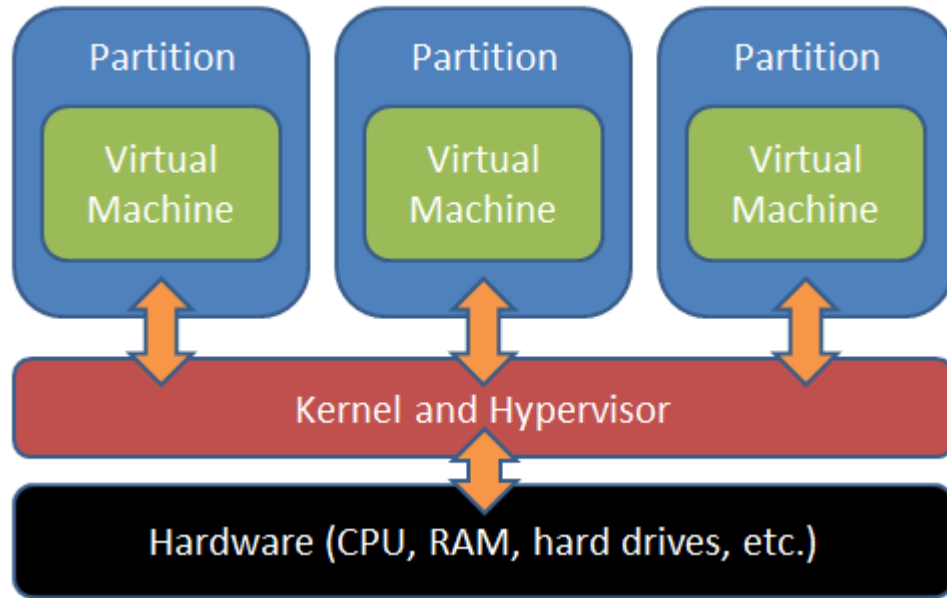


Figure 4. Typical Type I Hypervisor Architecture. Source: [49].

Microsoft ceased providing software support for Windows XP in 2018 and has publicly stated that Windows 7 will no longer be receiving any support or security updates after January 2020 [50], [51]. Similarly, Ubuntu 8.10 reached its end-of-life support in 2010 [52]. Despite these circumstances, using each of these OSs has valuable research potential for several reasons. First, Windows 7 still commands over a third of the market share for global desktop OS usage, according to NetMarketShare, which “tracks [the real-time] usage share of web technologies” by filtering out web robots to discern real users on the Internet [53]. Figure 5 shows a screenshot of the data collected by the NetMarketShare team comparing the top four OSs as a percentage of the global market share from April 2018 to March 2019. Second, according to Secretary of the Navy’s Cybersecurity Readiness Review released in March 2019, the USS *Gerald R. Ford* (CVN-78) aircraft carrier, commissioned in July 2017, was installed with Windows XP [2]. The concern here is that the U.S. Navy’s newest aircraft carrier is operating with software that Microsoft has explicitly stated “will still work but [the computer] might become more vulnerable to security risks and viruses,” due to the overall lack of cybersecurity support, especially when using Internet Explorer [50]. Smaller embedded devices typically employ Linux kernels because they are free and lightweight (in terms of memory usage and total lines of code),

so we also tested an older Ubuntu OS distribution. Finally, many of the cybersecurity vulnerabilities on these older OS versions have been well documented and cataloged by the NVD which feeds into several common vulnerability management systems, including OpenVAS.

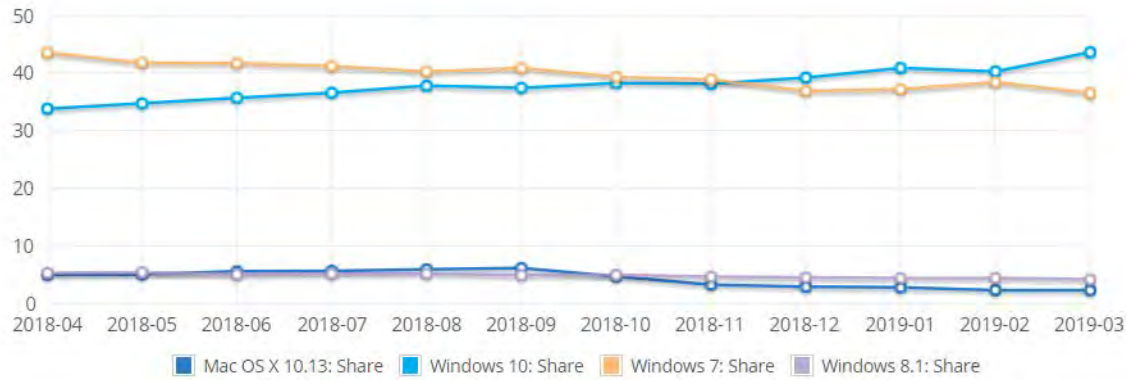


Figure 5. Operating System Market Share by Version. Source: [53].

To illustrate the dangers of using older OSs that have not been updated and patched, we will briefly discuss CVE-2008-4250 which can enable attackers to arbitrarily execute code on a Windows XP machine by exploiting a vulnerability within the Remote Procedure Call service that causes an “overflow during path canonicalization,” [54]. In this particular example, MSF has an exploit labeled `ms08_67_netapi` that takes advantage of a parsing flaw in the `NetAPI32.dll` of Microsoft’s 67th Windows patch released in 2008. This exploit allows the attacker to quickly gain system or administrator level privileges on the target system, which allows the attacker to gain unauthorized access to all information on the infected device. For the purposes of this thesis research, the target VMs on the virtual LAN do not contain all of the latest available software patches which allows RTIB to locate, identify, and report vulnerabilities to the user within a controlled environment.

2. RTIB GUI

RTIB uses a Python GUI library called Tkinter, short for Toolkit Interface, that creates a simple interface between the user and the CLI in an attempt to abstract away some

of the complexities of directly interacting with a command-line prompt. There are various interactive software toolkits available for Python, but Tkinter is free, relatively simple to use, and has achieved acceptance as the de facto Python GUI platform. Tkinter works by combining two independent software packages called Tcl and Tk. Tcl is a cross-platform and straightforward object-oriented programming language that provides applications with the ability to communicate with each other. Tk is an open-source widget toolkit that offers a set of basic building blocks for developing GUIs.

Tkinter creates a rectangular window called a frame on the RTIB user's display screen with several widgets that function as buttons. Each widget is configured and packed into position according to its assigned attributes. The widgets are also bounded explicitly to a specific function. In RTIB's case, each button calls a function that initiates a set of predetermined commands to be executed on the CLI. A lexical analysis library in Python called "Shlex" then takes a Unix command line string and parses it into a Python list for follow-on processing. RTIB also uses the "Subprocess" library for subprocess management, which allows it to spawn new processes and pipe commands together so that the output of one process is passed as the input to another process. Figure 6 is a screenshot of the RTIB GUI built using Tkinter.

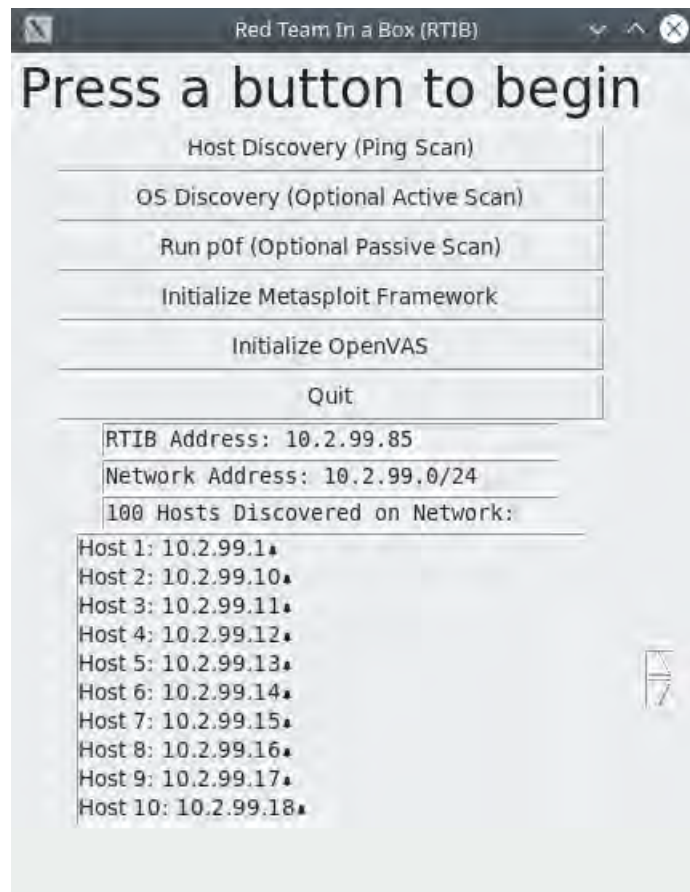


Figure 6. RTIB GUI Screenshot

Although Tkinter creates a simple interface and is visually appealing, it negatively impacts the speed, precision, and customization provided by a CLI. For example, if an RTIB user wanted to change the standard input or output stream while conducting a vulnerability analysis, a change in the RTIB's source code as seen in Appendix B would be required as opposed to a one-line command string on the CLI. On the other hand, a proposed benefit of using the RTIB is reducing the steep learning curve required for CLI usage. Furthermore, the controlled nature of the GUI can limit the user's ability to cause unintended, harmful, or destructive effects on the target system.

C. SCENARIO AND RTIB FUNCTIONALITY

In our test scenario, we assume that either a Sailor with an Information Systems Technician (IT) rating or a Marine with a Cyber Security Technician Military Occupational

Specialty (MOS) is tasked with conducting a regularly scheduled cybersecurity vulnerability assessment on their command's automated weapon system. Most service members within the command have user-level privileges on the weapon system and can access applications on a variety of individual computer systems, but none of the users has direct access to the Internet. The lack of Internet connectivity prevents a DoD cyber red team from conducting remote analysis on the weapon system. To perform the vulnerability assessment, an RTIB can be directly connected by a technician on the closed network during regular working hours while multiple users conduct routine operations. For this scenario, we also assume the network topology is similar to the LAN in Figure 1. The specific operational details of the weapon system are irrelevant to this scenario as long as the RTIB user can physically connect to the network. Figure 7 is a flow diagram that summarizes the overall RTIB process.

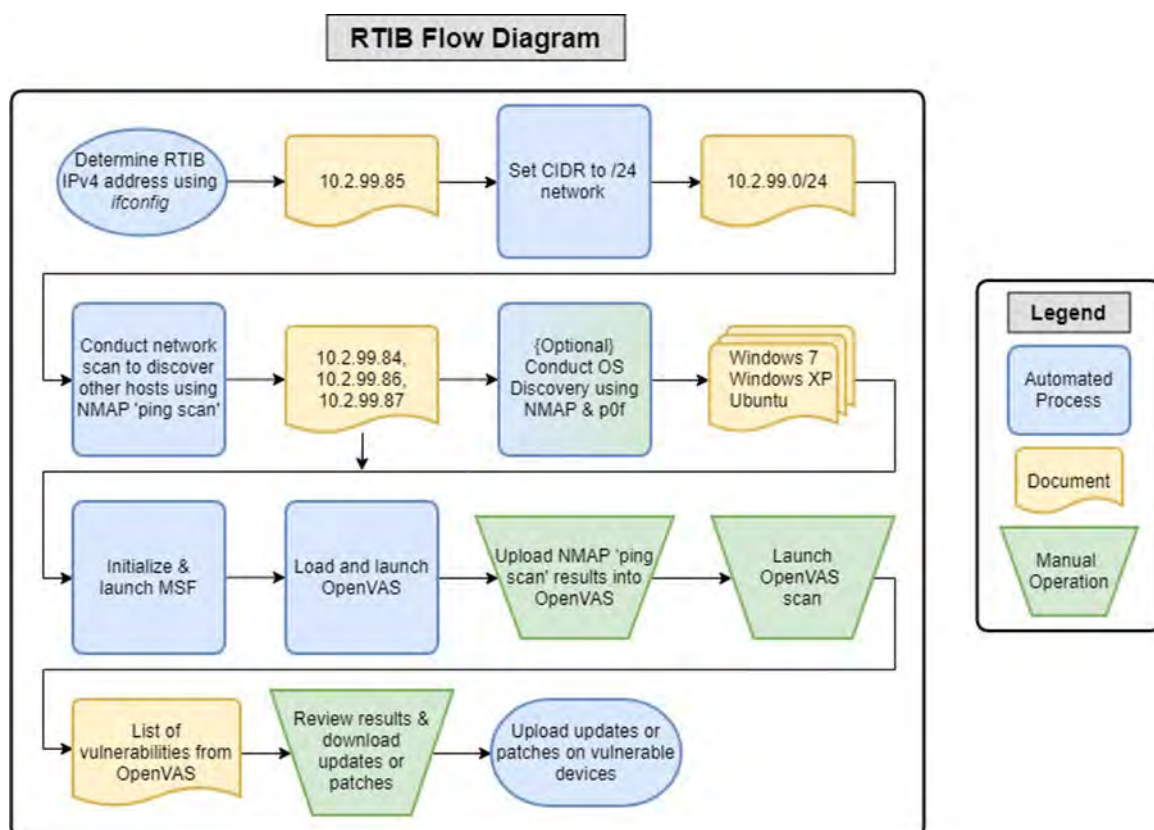


Figure 7. RTIB Flow Diagram

During the cyber reconnaissance phase, the technician begins the first RTIB task by conducting a quick reconnaissance of the target system to determine the overall network topography. After the user initiates the RTIB utility, Kali Linux leverages the *ifconfig* system utility on the CLI to retrieve a listing of the host device's network interface configuration, including the host device's active interfaces, IP addresses, network mask, and hardware Media Access Control (MAC) address. The RTIB parses this output data in search of its newly assigned IP version 4 (IPv4) address in dot-decimal notation. It then re-parses the IPv4 address and converts it into /24 Classless Inter-Domain Routing (CIDR) notation. We have chosen a /24 CIDR prefix since it gives RTIB the ability to scan through 256 IP addresses; however, the size of the network can be manually adjusted to be larger or smaller in the source code. Although RTIB can scan through all 256 IP addresses in a /24 network, only 254 of the addresses are usable for hosts since the ".0" address represents the overall network address, and ".255" is the broadcast address. After the reconnaissance phase, RTIB creates a text file that stores the new network address in CIDR notation and displays both the RTIB assigned address and the network address to the user on the RTIB window frame.

Next, RTIB uses the host's network address to discover all of the other live hosts on the network through an active NMAP scan. Since the goal is to enumerate the hosts on the network quickly, the *-sn* option is used. This option tells NMAP to forgo a port scan and output the hosts that responded to the discovery probe queries. The *-sn* or "ping scan" option sends an Internet Control Message Protocol (ICMP) echo request by sending a "TCP SYN to port 443, TCP ACK to port 80, and an ICMP timestamp request" to each host on the /24 network [55]. This option is preferred and is more reliable than sending pings over the broadcast address because some devices are configured not to respond to broadcast queries. Additionally, the NMAP reference guides mention that the *-sn* option does not have any detrimental performance effects on the host during the scan. The results of this ping sweep are then recorded on another text file created by the RTIB which is subsequently parsed and displayed to the user on the RTIB GUI. Figure 8 is a flow diagram of the host discovery process, and Figure 9 shows an example of the output from the host discovery phase.

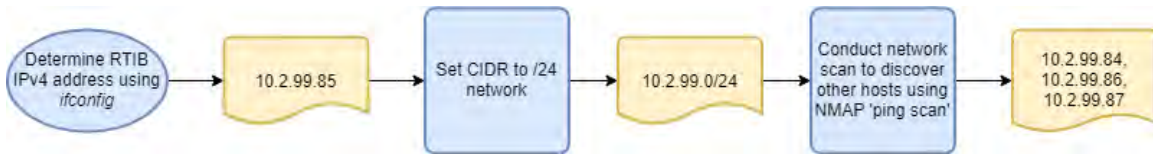


Figure 8. Host Discovery Flow Diagram

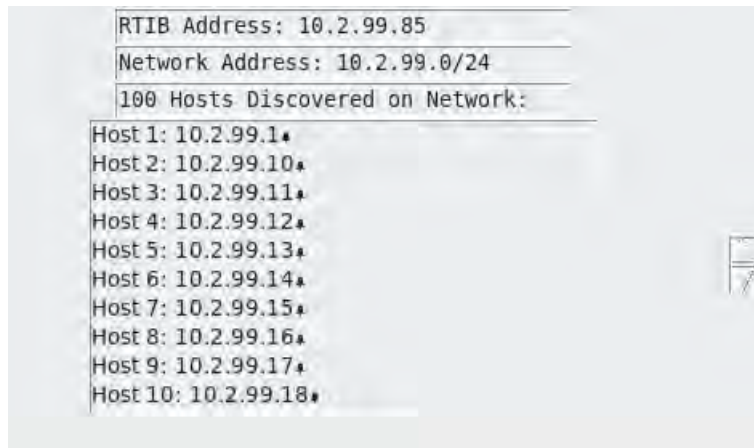


Figure 9. RTIB Host Discovery Screenshot

At this point, the RTIB has enumerated through the /24 network and displayed all of the active hosts it discovered through the ping sweep in a scrollable section within the GUI. Next, the RTIB user has the option to enter the OS discovery phase. Here, the previous text file created during the ping sweep is used to detect which type of OS the host may be using. RTIB uses NMAP's -O option, which allows NMAP to send several TCP and UDP packets to the target systems for TCP/IP stack fingerprinting [42]. The output from this scan provides a description of the OS, vendor name, version number, and device type. This information may be useful for an RTIB user to identify and understand the types of OSs on their network during a large-scale audit.

Additionally, the RTIB user has the option to augment the OS detection phase by conducting a passive scan leveraging the TCP/IP stack fingerprinting capabilities of p0f. Unlike NMAP's noisy and active scanning methods, p0f's approach passively collects and analyzes traffic generated by a target host as it communicates with other devices. It is important to note that selecting this option would not be an effective method to determine

the OS on a standalone host since p0f assumes the target host shares a telecommunication medium with another device. Regardless, the results of the p0f scan are then stored in a separate log file for future analysis. Figure 10 is a flow diagram that shows the results of the OS discovery phase.

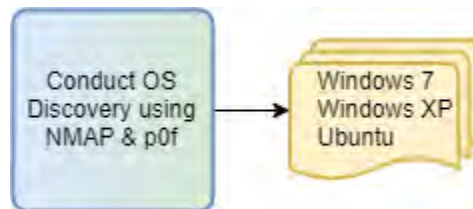


Figure 10. OS Discovery Phase Flow Diagram

After the OS detection phase is complete, RTIB enters the vulnerability scanning phase by configuring and initializing the MSF database through a short series of commands to the CLI. Since Kali Linux is a widely used penetration testing platform, its software developers created strict network service policies that attempt to minimize the exposure in potentially hostile or hazardous network environments. They do so by disallowing any network services to remain persistently on or open by default on the Kali Linux device, especially after a reboot. To open the services required by MSF, RTIB starts an open-source relational database management system called PostgreSQL and then initializes the MSF database [56]. Afterward, *msfconsole* is launched to provide a centralized interface between RTIB and MSF's capabilities to incorporate executing external commands.

Once *msfconsole* is running, RTIB sends a series of commands to configure and initialize OpenVAS. Although the OpenVAS command line utility allows users to configure targets, run vulnerability scans, and retrieve reports, it lacks some functionality provided through the more-capable GSA web-based GUI. Specifically, whenever a user sets a device as a target, the user is required to input the local and remote host IP addresses as well as other amplifying information. The newer version of MSF, unfortunately, creates a long alphanumeric string as a unique identifier for each targeted host. The manual entry of each identifier via the CLI can be time consuming and error prone and does not scale well on more extensive networks. However, the GSA GUI does provide an option to import

a list of IP addresses which alleviates the user's burden of having to type each alphanumeric string manually. Since RTIB is configured to save a list of the live or active hosts that it encountered during its host discovery phase, the RTIB user can directly import the information into the GSA target list and initiate a sequential scan of all of the hosts. Although this process causes the RTIB user to switch GUIs after configuring OpenVAS, it provides the user with a scalable solution for scanning large networks.

A cumulative summary of the vulnerability findings becomes available after performing the GSA scan and is available for export in various formats including text, PDF, extensible markup language (XML), and Comma-Separated Values (CSV). The report provides a listing of each discovered vulnerability with a brief explanation of the vulnerability's impact, affected software, available solutions, mitigation actions to reduce the overall threat and web links to source documents about the vulnerability. Of note, most of the recommendations provided by the summary typically instruct the user to install updated software on the target system. The RTIB user would then be responsible for conducting any further research of the vulnerability, downloading patches from the Internet, and uploading the updated software on the vulnerable machines. This level of automation falls outside RTIB's current capabilities, and Chapter V discusses possible solutions in future work. Figure 11 provides a flow diagram of the MSF and OpenVAS process.

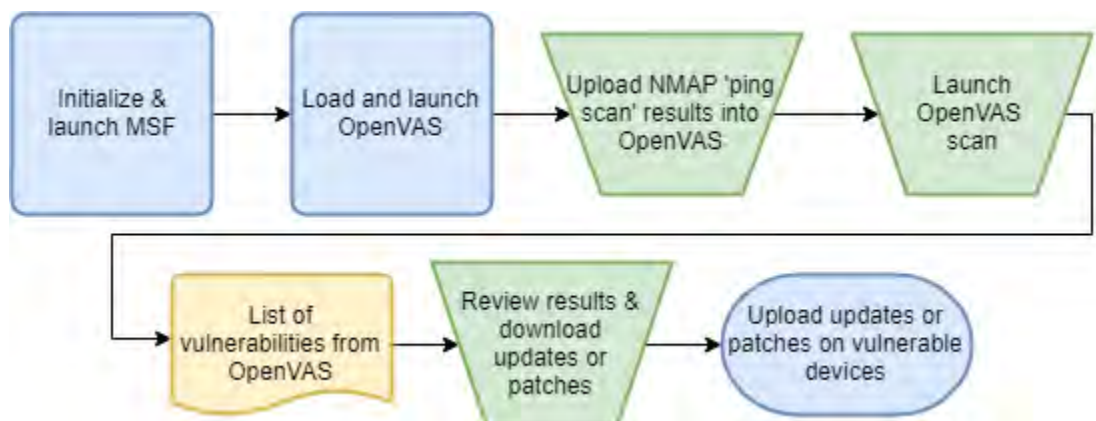


Figure 11. MSF & OpenVAS Flow Diagram

D. TESTING AND RESULTS

We conducted RTIB testing on two identical closed networks within the NPS CYBL. The results discussed in this thesis only reference the 10.2.99.0/24 LAN since the overall network space did not affect the outcomes of the vulnerability scan. Each target VM on the network was automatically assigned an IPv4 address using Dynamic Host Configuration Protocol (DHCP); however, a static IPv4 address of 10.2.99.85 was assigned to the RTIB since the current Kali Linux distribution assigns an IP version 6 (IPv6) address by default. Within each LAN, two separate tests were conducted to determine RTIB's scalability and effectiveness. The first test was run as a proof-of-concept with only three hosts, while the second was intended to be a larger scale test with one hundred hosts. We repeated each experiment at least twice, but no noticeable deviations between repeated tests were observed during OpenVAS's final vulnerability report.

1. Testing a /30 Network

For the first test, the CIDR prefix was set to 10.2.99.84/30 to ensure that the RTIB examined a maximum of three outdated, vulnerable hosts. The goal was to scope the RTIB scan to a small address space before proceeding to a full /24 network. After running the scan, RTIB accurately found all of the devices on the network and made an initial determination of each host's OS. Figure 12 and Figure 13 show the host and the NMAP OS discovery results, respectively. Figure 14 shows a small snapshot of the results from a passive p0f run on a Windows 7 target. Although the RTIB had some difficulty uniquely identifying the Windows 7 machine using its OS discovery tools, this did not have any detrimental effects on OpenVAS's ability to identify critical vulnerabilities on the target device during its subsequent vulnerability scan.

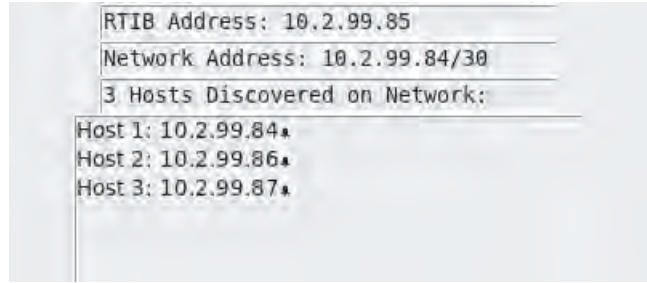


Figure 12. Host Discovery Results

IPv4 Address	Actual OS	RTIB OS Discovery Results
10.2.99.84	Microsoft Windows 7 SP 1	Microsoft Windows Server 2008 R2 or Windows 8.1, Microsoft Windows 7 Professional or Windows 8, ... Windows 7 SP1
10.2.99.86	Microsoft Windows XP SP3	Microsoft Windows XP SP2 or SP3, or Windows Server 2003
10.2.99.87	Linux 2.6.27-7	Linux 2.6.18 - 2.6.31

Figure 13. NMAP OS Discovery Results

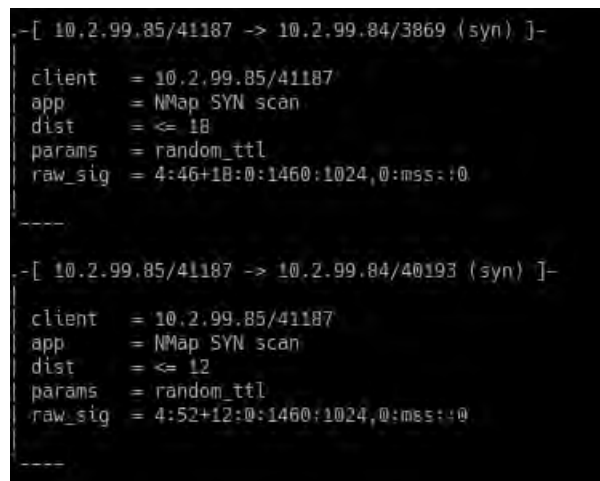


Figure 14. Snapshot of p0f Capture

The output of the host discovery scan was sent to a separate text file labeled “hosts_to_scan.txt” for use by the GSA when it runs a vulnerability scan within the OpenVAS framework. To do so, the RTIB user must manually upload the text file as a new target under the GSA “Configuration” tab, as seen in Figure 15.

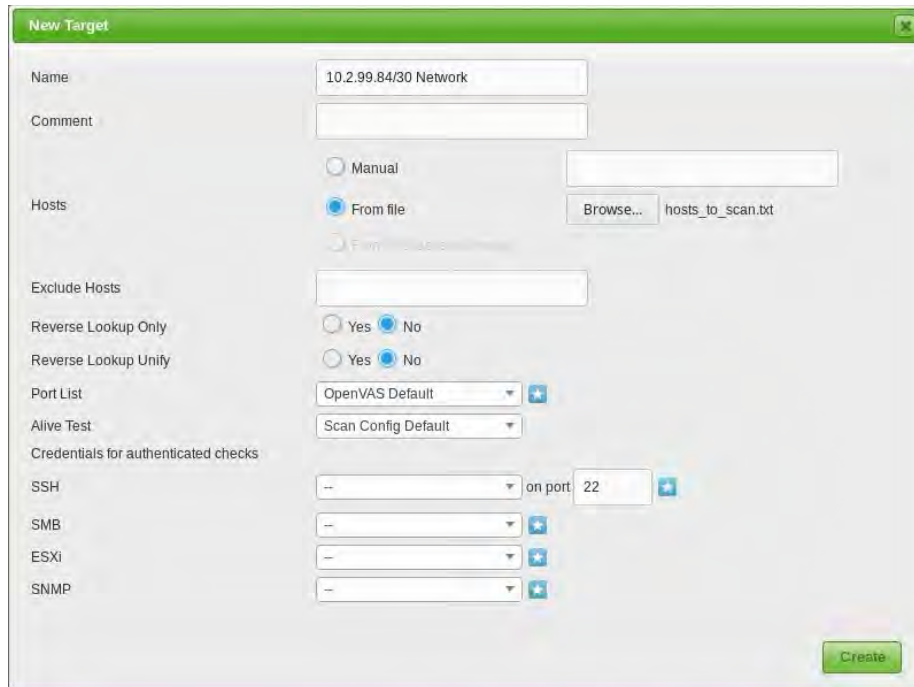


Figure 15. GSA Target Creation Window. Source: [57].

The GSA then automatically parses the file and lists all of the hosts to be scanned as comma-separated values. After a vulnerability scan was created and launched on the GSA, it iterated through each host and stored the discovered vulnerabilities in an exportable report. Each scan on the /30 network took about thirty minutes which could be a product of the computing resources available on the CYBL hypervisor or due to the delays that occur as the GSA algorithm iterates through its repository of vulnerabilities on each host. The GSA source documents do reveal that if a scan is run on more than one system at a time, “the scan might have a negative impact on either the performance of the scanned systems, the network or the [GSA] appliance itself” [58].

Interestingly, the GSA is also able to provide a simple vertex-labeled graph that depicts the network’s topology as a two-dimensional model which provides a rudimentary visual representation of a network and could be utilized for simple graph analysis by an RTIB user. An example of the GSA-generated graph is shown in Figure 16, where the green center node is the RTIB, and the red nodes are the vulnerable hosts labeled by their IPv4 addresses.

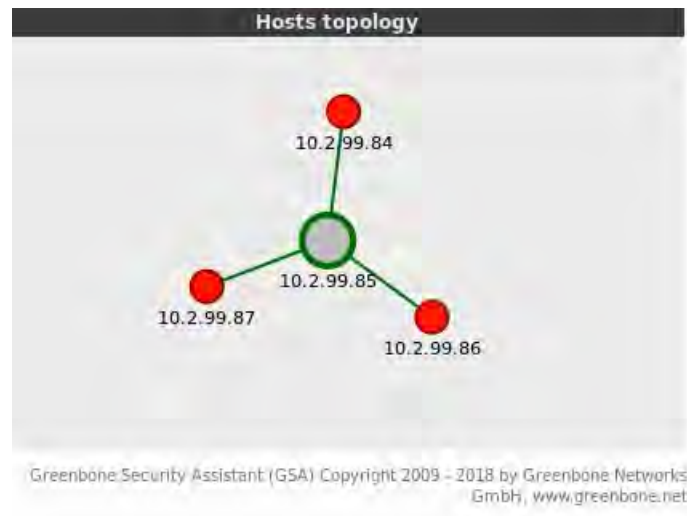


Figure 16. GSA Hosts Topology for the 10.2.99.84/30 Network.
Source: [57].

Furthermore, the GSA also provides an aggregate display of the number of vulnerabilities on the network by severity, shown in Figure 17. Unfortunately, this graph does not break up the vulnerabilities by individual hosts. A better representation of the consolidated cybersecurity vulnerabilities by host is shown in the report summary in Figure 18.

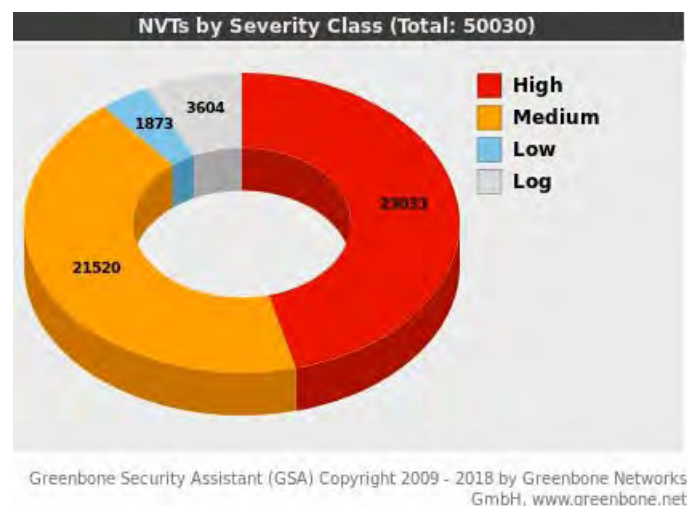


Figure 17. GSA NVTs by Severity Class on the 10.2.99.84/30 Network. Source: [57].

Host	High	Medium	Low	Log	False Positive
10.2.99.84 GeorgiaWeidman.cyber.lab	1	3	1	23	0
10.2.99.85	0	0	0	3	0
10.2.99.86	120	190	15	64	0
10.2.99.87	77	89	14	35	0
Total: 4	198	282	30	125	0

Figure 18. GSA Scan Result Overview for the 10.2.99.84/30 Network.
Source: [57]

2. Testing a /24 Network

The second round of tests was conducted using a /24 network with one hundred VMs to determine how RTIB would perform against the larger subnet shown in Figure 19. In this case, all of the environmental variables remained the same except the total number of hosts connected to the network due to the modification of the CIDR variable in the source code. During these tests, the RTIB did not have any significant delays in determining how many hosts were on a network or identifying the host's OS. As expected, a considerable delay occurred after launching the GSA scan. Scanning through one hundred VMs in search of known vulnerabilities took, on average, about 5.25 hours.

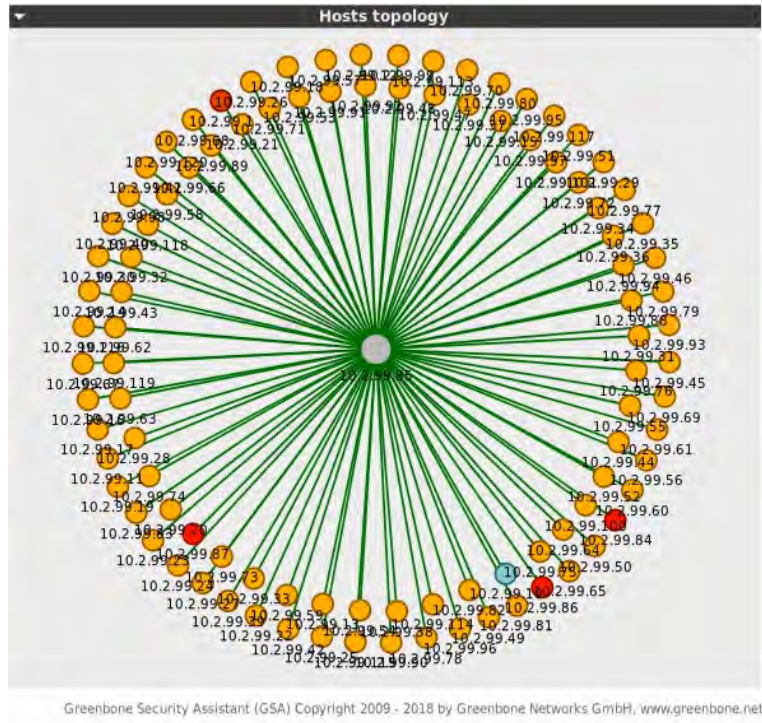


Figure 19. GSA Hosts Topology for the 10.2.99.0/24 Network.
Source: [57].

Although analyzing the amount of time it takes the RTIB to conduct a scan successfully was not an objective of this research, it is nonetheless an essential consideration for the RTIB user. Specifically, RTIB users would need to understand that scanning an extensive network of devices may have secondary effects on RTIB's resources such as losing battery power or entering a sleep state due to inactivity which could ultimately delay or abort the vulnerability scan. Regardless, the GSA worked as anticipated and produced similar results during both rounds of tests. Figure 20 shows a snapshot of the RTIB scan results for one of the Windows 7 machines on the /24 network. The "References" section at the bottom of Figure 20 is particularly useful for the RTIB user because it provides several web links from the software vendor that contains information on how to update or patch the vulnerability.

High (CVSS: 10.0) NVT: MS15-034 HTTP.sys Remote Code Execution Vulnerability (remote check)
Summary This host is missing an important security update according to Microsoft Bulletin MS15-034.
Vulnerability Detection Result Vulnerability was detected according to the Vulnerability Detection Method.
Impact Successful exploitation will allow remote attackers to run arbitrary code in the context of the current user and to perform actions in the security context of the current user.
Solution Solution type: VendorFix Run Windows Update and update the listed hotfixes or download and install the hotfixes from the referenced advisory.
Affected Software/OS Microsoft Windows 8 x32/x64 Microsoft Windows 8.1 x32/x64 Microsoft Windows Server 2012 Microsoft Windows Server 2012 R2 Microsoft Windows Server 2008 x32/x64 Service Pack 2 and prior Microsoft Windows 7 x32/x64 Service Pack 1 and prior
Vulnerability Insight Flaw exists due to the HTTP protocol stack 'HTTP.sys' that is triggered when parsing HTTP requests.
Vulnerability Detection Method Send a special crafted HTTP GET request and check the response Details: MS15-034 HTTP.sys Remote Code Execution Vulnerability (remote check) OID:1.3.6.1.4.1.25623.1.0.105257 Version used: \$Revision: 13679 \$
References CVE: CVE-2015-1635 Other: URL: https://support.microsoft.com/kb/3042553 URL: https://technet.microsoft.com/library/security/MS15-034 URL: http://pastebin.com/ypURDPc4

Figure 20. GSA Scan Result Snapshot for the 10.2.99.0/24 Network.
Source: [57].

3. Firmware Extraction

Unfortunately, we are unable to successfully extract and test firmware binary images in the current version of RTIB. The critical issue was that most firmware images reside inside a flash memory integrated circuit, and extracting firmware from flash memory typically involves some form of reverse engineering such as physically removing soldered circuits from a printed circuit board or tapping into a memory bus and then dumping the contents onto another device. Although it is possible to gain access to certain types of firmware through a command shell, the knowledge required to extract the binary image successfully through this method goes beyond the assumed expertise level of a typical RTIB user. Chapter V discusses RTIB's ability to conduct this type of extraction as a future capability.

E. SUMMARY

This chapter described the virtual environment, testing scenario, functionality, and difficulties experienced while designing and implementing an RTIB. Appendix A also contains a more detailed block diagram of the inputs, outputs, and functions within RTIB.

Our testing demonstrated that RTIB could accurately iterate through an unknown network and identify the active hosts within an assigned subnet, regardless of the total number of active devices. Although RTIB's ability to scan for active hosts was nearly instantaneous, there was a substantial delay once the GSA began its scan for vulnerabilities. However, the reports generated by the OpenVAS GSA were sufficient in providing the RTIB user with an in-depth analysis of the vulnerabilities on each host. The next chapter discusses the overall conclusions and potential areas of focus for future work.

V. CONCLUSIONS AND FUTURE WORK

A. SUMMARY

The main goal of this research was to develop a portable cyber red teaming tool capable of identifying and assessing the cybersecurity vulnerabilities on DoN computer systems not directly connected to the Internet, and providing users with recommendations to mitigate the cyber threat against those vulnerabilities. We developed a prototype tool called Red Team in a Box (RTIB) that can automatically search for other devices on its IPv4 subnet once it is directly connected on an individual host or LAN. It provides the user with a simple GUI that lists the other devices on the network, and can also conduct and log active and passive OS fingerprinting scans. Finally, it creates a text file used by an open source vulnerability tool called OpenVAS that performs a vulnerability scan and provides the user with recommendations to mitigate discovered threats.

RTIB was designed to address the current resource limitations of DoD cyber red teams by providing a convenient and useful cybersecurity tool that capitalizes on the open source architectures of the Metasploit Framework (MSF) and OpenVAS. We tested two scenarios on the prototype tool to demonstrate that RTIB could successfully scan and discover all active hosts within a subnet and subsequently provide a list of IPv4 addresses to OpenVAS for further detailed analysis. This thesis presented a proof-of-concept tool that helps enhance the security posture on DoN computer systems to complement other defense in depth measures.

B. CONCLUSIONS

RTIB leverages existing open source software and can be used as a basic framework to develop more advanced capabilities in automated vulnerability assessments on devices not connected to the Internet. Most of the objectives and goals of this thesis were accomplished through careful research and testing; however, the current version of RTIB is unable to extract firmware images from embedded devices. Other open source applications, including binary analysis tools such as Binwalk and FAT, could be integrated

into a future version of RTIB to provide the user with a complete assortment of capabilities to analyze any electronic device, regardless of its OS or firmware image.

1. Primary Research Question

How can a portable set of software tools be developed to automatically detect and expose security vulnerabilities on DoN computer systems?

We successfully developed a simple prototype that automates a series of common CLI commands used by DoD cyber red teams and penetration testers to conduct vulnerability assessments on networked systems. After understanding the tools and techniques other researchers use to analyze software applications and cyber-physical systems, we determined that creating a simple Python GUI running on a Kali Linux distribution provides the user with the convenience to quickly scan and identify a set of vulnerable hosts connected on a LAN. We also concluded that using the MSF to launch the OpenVAS GSA offers a relatively seamless transition from RTIB's core functions of host discovery and OS fingerprinting to a vulnerability scan and assessment. We demonstrated this capability by conducting a series of tests with minimal user input on various exploitable machines within the NPS CYBL.

2. Secondary Research Question

How would such a tool report discover vulnerabilities without interrupting the standard operating protocols of the host system?

The tools used during the host discovery phase did not have any significant effects on each host's performance since the ping sweeps are quick, use little bandwidth, and do not launch an exhaustive port scan. During the OS fingerprinting phase, the active scans can be very noisy due to the number of TCP and UDP packets being sent over the wire; however, RTIB offers a passive solution through p0f that does not impact any of the communication protocols between hosts. OpenVAS's vulnerability scans proved to be time-consuming and could have an impact on the performance of the host system according to the GSA source documents. Future RTIB testing could determine whether OpenVAS's

impact would be sufficiently noticeable to a user since our tests were conducted without any human subjects on a simulated network.

C. RECOMMENDATIONS FOR FUTURE WORK

The RTIB was designed to provide a user with limited cybersecurity knowledge the ability to launch a vulnerability scan on a device and subsequently receive a list of recommendations to mitigate the cybersecurity threat. However, other tools and techniques were considered in this thesis but were not integrated into the prototype due to various limitations. Future research could target some of RTIB's deficiencies and provide considerable advances in automating vulnerability assessments on embedded devices without Internet connectivity.

1. NSA Open Source Software

The NSA maintains a publicly available repository of open source software that could be incorporated into RTIB's current functions [38][38]. For example, AtomicWatch was designed to be used by network administrators to recursively parse through a directory of log files and return any "results if a positive match is found" [38]. In RTIB's case, AtomicWatch could be used to scan for keywords on the log file that p0f creates, as shown in Appendix A. Another NSA tool called Maplesyrup shows "the low-level operating configuration of the system, and can be used to help determine the security state of a device" [38]. This would be useful in determining security settings on Linux devices by displaying the read, write, and execute permissions enforced by the kernel and whether or not RTIB would have access to certain regions of memory on the target device.

2. Automating User Feedback

As shown in the "User Feedback" column in Table 1 of Chapter II, there are many different outputs provided to the user after a vulnerability assessment tool is used. In RTIB's case, the PDF produced by OpenVAS can be rather extensive depending on the number of hosts and their discovered vulnerabilities. Although the overall summary is very comprehensive, it would be beneficial for the user to see a condensed version that details the exact steps the user must take to mitigate the threat. For instance, RTIB could prompt

the user to proceed to a hyperlinked CVE web address that discusses the vulnerabilities or to a website with patches and upgrades provided by the vendor. Ideally, after all of the software is downloaded onto RTIB, it would then automatically update the vulnerable systems once it was physically reconnected to the target host or the network.

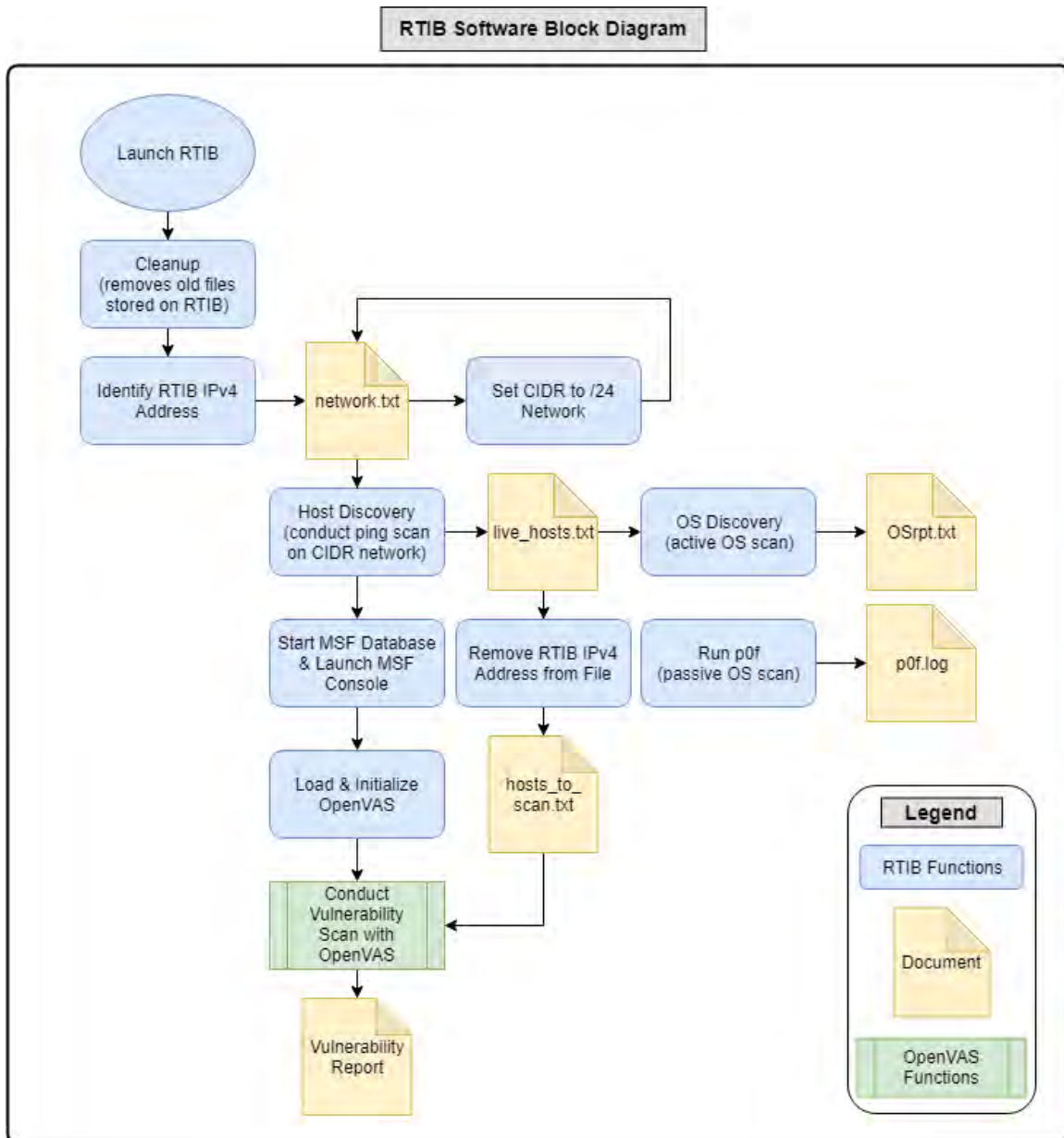
3. Firmware Extraction

As mentioned in Section D of Chapter IV, we were unable to extract and test firmware binary images due to the requirement to reverse engineer the flash memory integrated circuit of a target device. Since most firmware extraction techniques involve physical interactions with the circuit boards and at least a basic knowledge in reverse engineering, this is currently beyond the scope of RTIB's capabilities. However, if the firmware image could be pulled from a device and uploaded to RTIB, then it would be feasible for RTIB to perform a vulnerability assessment using some of the tools mentioned in Section E of Chapter III.

4. Human Subject Testing

Since the intent of RTIB is to provide users with limited cybersecurity expertise the ability to scan devices in search of vulnerabilities, it would be beneficial to observe human subjects completing simple tasks in a controlled environment. For example, a laptop loaded with Kali Linux and running the RTIB application could be interfaced with a small LAN of desktops running various services. A series of tests could then be performed to observe the human-computer interaction and determine whether the human subjects can complete any assigned tasks using the RTIB GUI.

APPENDIX A. RTIB SOFTWARE BLOCK DIAGRAM



THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B. RTIB SOURCE CODE

```
from tkinter import *
import subprocess as sub
import shlex
import time

##### [GLOBAL VARIABLES] #####

FONT = "Arial Bold"
FONT_SIZE = 25
WIDTH = 35
HEIGHT = 1
ROW = 0
COLUMN = 0
WINDOW_SIZE = "425x525" #width x height
SECS = 1
CIDR = "0/24" #Classless Inter-Domain Routing prefix
LAST_OCTET = 3 #used to replace dot-decimal notation
MSF_PROC = None
RTIB_ADDRESS = ''

##### [HELPER FUNCTIONS] #####

#Remove extraneous or old files from the desktop
def cleanup():
    cmd1 = "rm network.txt live_hosts.txt hosts_to_scan.txt OSrpt.txt\nmap_subnet.gnmap nmap_subnet.nmap nmap_subnet.xml"
    args1 = shlex.split(cmd1)
    proc1 = sub.run(args1)
    return

#Identify RTIB's assigned IPv4 address
def id_Address():
    with open("network.txt", "w+") as f:
        cmd1 = "ifconfig"
        cmd2 = "grep -oE 'inet ([0-9]{1,3}\.){3}[0-9]{1,3}'"
        cmd3 = "grep -oE '([0-9]{1,3}\.){3}[0-9]{1,3}'"
        cmd4 = "grep -v '127'"
        args1 = shlex.split(cmd1)
        args2 = shlex.split(cmd2)
        args3 = shlex.split(cmd3)
        args4 = shlex.split(cmd4)
        proc1 = sub.Popen(args1, stdout = sub.PIPE)
        proc2 = sub.Popen(args2, stdin = proc1.stdout, stdout = sub.PIPE)
        proc3 = sub.Popen(args3, stdin = proc2.stdout, stdout = sub.PIPE)
        proc4 = sub.Popen(args4, stdin = proc3.stdout, stdout = f)
    return

#Set CIDR notation
def set_CIDR():
    global RTIB_ADDRESS
    with open("network.txt", "r") as f:
        network = f.readlines()
        RTIB_ADDRESS = network[0]
    with open("network.txt", "w") as f:
        for line in network:
            octet = line.split('.')
            octet.pop(LAST_OCTET)
```

```

        octet.insert(LAST_OCTET, CIDR)
        CIDR_network = '.'.join(octet)
        f.write(CIDR_network)
    return

#Conduct ping scan on network
def host_Discovery():
    with open("network.txt", "r") as f:
        CIDR_network = f.read()
        cmd1 = "nmap -sn -oA nmap_subnet " + CIDR_network
        args1 = shlex.split(cmd1)
        proc1 = sub.Popen(args1)
        time.sleep(SECS)

    with open("live_hosts.txt", "w+") as f:
        cmd2 = "grep Up nmap_subnet.gnmap"
        cmd3 = "cut -d ' ' -f 2"
        args2 = shlex.split(cmd2)
        args3 = shlex.split(cmd3)
        proc2 = sub.Popen(args2, stdout = sub.PIPE)
        proc3 = sub.Popen(args3, stdin = proc2.stdout, stdout = f)
    return

#Conduct active OS scan
def os_Discovery():
    with open("live_hosts.txt", "r") as f:
        live_host = f.readlines()
        live_host_len = len(live_host)
    with open("OSrpt.txt", "w+") as f:
        i = 0
        for i in range(live_host_len):
            cmd1 = "nmap -O " + live_host[i]
            cmd2 = "grep 'OS details'"
            args1 = shlex.split(cmd1)
            args2 = shlex.split(cmd2)
            proc1 = sub.Popen(args1, stdout = sub.PIPE)
            proc2 = sub.Popen(args2, stdin = proc1.stdout, stdout = f)
            i += 1
    return

#Conduct passive OS scan
def run_p0f():
    cmd1 = "p0f -i eth0 -p -o /root/Desktop/p0f.log"
    args1 = shlex.split(cmd1)
    proc1 = sub.run(args1)
    time.sleep(SECS)
    proc1.terminate()
    return

#Start MSF database and launch MSF console
def setup_MSf():
    global MSF_PROC
    cmd1 = "service postgresql start"
    cmd2 = "msfdb init"
    cmd3 = "msfconsole -q"
    args1 = shlex.split(cmd1)
    args2 = shlex.split(cmd2)

```

```

args3 = shlex.split(cmd3)
proc1 = sub.run(args1)
proc2 = sub.run(args2)
MSF_PROC = sub.Popen(args3, stdin = sub.PIPE, stderr = sub.PIPE)
return

#Load and initilaize OpenVAS
def run_OpenVAS():
    global MSF_PROC
    if MSF_PROC == None:
        print("MSF is not running")
        return
    MSF_PROC.stdin.write(b'load openvas\n')
    time.sleep(SECS)
    MSF_PROC.stdin.write(b'openvas_connect admin toor localhost 9390\n')
    time.sleep(SECS)
    MSF_PROC.stdin.write(b'openvas-start\n')
    time.sleep(SECS)
    MSF_PROC.stdin.flush()
    return

##### [GUI SETUP] #####

gui = Tk()
gui.title("Red Team In a Box (RTIB)")

header = Label(gui, text = "Press a button to begin", font = (FONT,
FONT_SIZE))
gui.geometry(WINDOW_SIZE)
header.grid(column = COLUMN, row = ROW)
ROW += 1

host_Discovery_button = Button(gui, text = "Host Discovery (Ping Scan)",
width = WIDTH, command = host_Discovery)
host_Discovery_button.grid(column = COLUMN, row = ROW)
ROW += 1

os_Discovery_button = Button(gui, text = "OS Discovery (Optional Active
Scan)", width = WIDTH, command = os_Discovery)
os_Discovery_button.grid(column = COLUMN, row = ROW)
ROW += 1

run_p0f_button = Button(gui, text = "Run p0f (Optional Passive Scan)", width
= WIDTH, command = run_p0f)
run_p0f_button.grid(column = COLUMN, row = ROW)
ROW += 1

setup_MSf_button = Button(gui, text = "Initialize Metasploit Framework",
width = WIDTH, command = setup_MSf)
setup_MSf_button.grid(column = COLUMN, row = ROW)
ROW += 1

run_OpenVAS_button = Button(gui, text = "Initialize OpenVAS", width = WIDTH,
command = run_OpenVAS)
run_OpenVAS_button.grid(column = COLUMN, row = ROW)
ROW += 1

```

```

quit_button = Button(gui, text = "Quit", width = WIDTH, command =
gui.destroy)
quit_button.grid(column = COLUMN, row = ROW)
ROW += 1

##### [MAIN FUNCTION CALLS] #####

if __name__ == "__main__":
    cleanup()
    time.sleep(SECS)
    id_Address()
    time.sleep(SECS)

    RTIB_addr_text = Text(gui, height = HEIGHT, width = WIDTH)
    with open("network.txt", "r") as f:
        for network in f:
            network = network.replace("\n", "")
            RTIB_addr_text.insert(END, "RTIB Address: %s" % network)
    RTIB_addr_text.grid(column = COLUMN, row = ROW)
    ROW += 1

    set_CIDR()
    time.sleep(SECS)

    Network_addr_text = Text(gui, height = HEIGHT, width = WIDTH)
    with open("network.txt", "r") as f:
        for network in f:
            Network_addr_text.insert(END, "Network Address: %s" % network)
    Network_addr_text.grid(column = COLUMN, row = ROW)
    ROW += 1

    host_Discovery()
    time.sleep(SECS)

    with open("live_hosts.txt", "r") as f:
        live_host = f.readlines()
        live_host_len = len(live_host)
        total = live_host_len - 1
        i = 0
        for i in range(live_host_len):
            if live_host[i] == RTIB_ADDRESS:
                live_host.pop(i)

    Host_disc_text = Text(gui, height = HEIGHT, width = WIDTH)
    Host_disc_text.insert(END, "%d Hosts Discovered on Network:" % total)
    Host_disc_text.grid(column = COLUMN, row = ROW)
    COLUMN += 1
    ROW += 1

    with open("hosts_to_scan.txt", "w+") as f:
        i = 0
        for i in range(total):
            f.write(live_host[i])

```

```

host_scroll = Scrollbar(gui)
host_scroll.grid(column = COLUMN, row = ROW)
host_list = Listbox(gui, yscrollcommand = host_scroll.set, width = WIDTH)
COLUMN -= 1
host_list.grid(column = COLUMN, row = ROW)
host_scroll.config(command = host_list.yview)

with open("hosts_to_scan.txt", "r") as f:
    host_to_scan = f.readlines()
    host_to_scan_len = len(host_to_scan)
    i = 1
    j = 0
    for host in range(host_to_scan_len):
        host = "Host %d: " % i + "%s" % host_to_scan[j]
        host_list.insert(END, host)
        i += 1
        j += 1

gui.mainloop()

```

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

- [1] C. Chaplain, “Weapon Systems Cybersecurity: DoD just beginning to grapple with scale of vulnerabilities,” Washington, DC, USA, GAO Report No. GAO-19-128, 2018.
- [2] M. J. Bayer, J. M. B. O’Connor, R. S. Moultrie, and W. H. Swanson, “Secretary of the Navy: Cybersecurity readiness review,” Washington, DC, USA, 2019.
- [3] S. Buchanan, “Cyber space security: dispelling the myth of computer network defense by true red teaming the Marine Corps and Navy,” Quantico, VA, USA, 2010. [Online]. Available: <https://apps.dtic.mil/dtic/tr/fulltext/u2/a536674.pdf>
- [4] Department of Defense, “Department of Defense cyber strategy summary 2018,” Washington, DC, USA, 2018.
- [5] *Doctrine for the Armed Forces of the United States*, JP 3-12, Joint Chiefs of Staff, Washington, DC, USA, 2018. [Online]. Available: https://www.jcs.mil/Portals/36/Documents/Doctrine/pubs/jp3_12.pdf
- [6] E. Hutchins, M. Cloppert, and R. Amin, “Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains,” in *6th Intl. Conf. on Warfare and Sec.*, 2011, pp. 80–106.
- [7] *Doctrine for the Armed Forces of the United States*, JP 3-60, Joint Chiefs of Staff, Washington, DC, USA, 2018. [Online]. Available: <https://www.jcs.mil/Doctrine/Joint-Doctrine-Pubs/3-0-Operations-Series/>
- [8] R.F. Behler, “The office of the director, operational test & evaluation FY2017 annual report,” Washington, DC, USA, 2018.
- [9] D. Trump, “National cyber strategy of the United States of America,” Washington, DC, USA, 2018.
- [10] *Department of Defense Cyber Red Team Certification and Accreditation*, CJCSM 6510.03, Department of Defense, Washington, DC, USA, 2013. [Online]. Available: <https://www.jcs.mil/Portals/36/Documents/Library/Manuals/m651003.pdf?ver=2016-02-05-175711-083>
- [11] *Information Assurance Workforce Improvement Program*, DoD 8570.01-M, Department of Defense, Washington, DC, USA, 2015. [Online]. Available: <https://www.esd.whs.mil/Portals/54/Documents/DD/issuances/dodm/857001m.pdf>

- [12] J. M. Gilmore, “The office of the director, operational test & evaluation FY2016 annual report,” Washington, DC, USA, 2016.
- [13] J. Schab, “Tackling DoD cyber red team deficiencies through systems engineering,” SANS Institute, Philadelphia, PA, USA, 2017. [Online]. Available: <https://www.sans.org/reading-room/whitepapers/testing/tackling-dod-cyber-red-team-deficiencies-systems-engineering-38020>
- [14] J. J. Li and L. Dougherty, “Training cyber warriors: What can be learned from defense language training?,” RAND Corp., Santa Monica, CA, USA, RR-476-OSD, 2015. [Online]. Available: https://www.rand.org/content/dam/rand/pubs/research_reports/RR400/RR476/RAND_RR476.pdf
- [15] R. Mudge, “Cobalt Strike: advanced threat tactics for penetration testers,” Cobalt Strike, 2015. [Online]. Available: <https://www.cobaltstrike.com/downloads/cs2015slick.pdf>
- [16] G. Weidman, *Penetration Testing: A Hands-On Introduction to Hacking*, 1st ed. San Francisco, CA, USA: No Starch Press, 2014.
- [17] M. Rouse, “Metasploit project—Metasploit framework,” WhatIs, August 2011. [Online]. Available: <https://whatis.techtarget.com/definition/Metasploit-Project-Metasploit-Framework>
- [18] D. Kennedy, J. O’Gorman, D. Kearns, and M. Aharoni, *Metasploit: The Penetration Tester’s Guide*, 1st ed. San Francisco, CA, USA: No Starch Press, 2011.
- [19] UpGuard, “Core Security vs. Rapid7,” UpGuard, September 3, 2018. [Online]. Available: <https://www.upguard.com/articles/core-security-vs-rapid7>
- [20] MSF Development Staff, “Getting started: Metasploit framework,” Metasploit, November 24, 2018. [Online]. Available: <https://metasploit.help.rapid7.com/docs/getting-started>
- [21] J. Yoon and W. Sim, “Implementation of the automated network vulnerability assessment framework,” in *Innov. in Info. Tech.* 2007. [Online]. doi: 10.1109/IIT.2007.4430423
- [22] MSF Development Staff, “Importing data,” Metasploit, October 17, 2018. [Online]. Available: <https://metasploit.help.rapid7.com/docs/importing-data>
- [23] E. J. Schwartz, T. Avgerinos, and D. Brumley, “All you ever wanted to know about dynamic taint analysis and forward symbolic execution (but might have been afraid to ask),” presented at the IEEE Symp. on Sec. and Priv., Oakland, CA, USA, May 16-19, 2010.

- [24] S. Shah and B. M. Mehtre, "An automated approach to vulnerability assessment and penetration testing using Net-Nirikshak 1.0," in *IEEE Intl. Conf. on Adv. Comms., Ctrl. and Comp. Tech.*, Ramanathapuram, India, May 8-10, 2014, pp. 707–712.
- [25] L. Epling, B. Hinkel, and Y. Hu, "Penetration testing in a box," presented at the InfoSec '15, Kennesaw, GA, USA, October 10, 2015.
- [26] V. Tilemachos and C. Manifavas, "An automated network intrusion process and countermeasures," in *19th Pan. Conf. on Info.*, Athens, Greece, 2015, pp. 156–160.
- [27] L.-A. Ponirakis, "Red team in a box for embedded and non-IP devices," Navy SBIR, May 22, 2018. [Online]. Available: https://www.navysbir.com/n18_2/N182-131.htm
- [28] R. Langner, "Stuxnet: dissecting a cyberwarfare weapon," *IEEE Security & Privacy*, vol. 9, no. 3, pp. 49–51, Jun. 2011. [Online]. doi: 10.1109/MSP.2011.67
- [29] M. Muench, J. Stijohann, F. Kargl, A. Francillon, and D. Balzarotti, "What you corrupt is not what you crash: challenges in fuzzing embedded devices," presented at the NDSS Symp. 2018, San Diego, CA, USA, February 18-21, 2018.
- [30] Y. Shoshitaishvili, R. Wang, C. Houser, C. Kruegel, and G. Vigna, "Firmallice - automatic detection of authentication bypass vulnerabilities in binary firmware," in *NDSS'15*, 2015 [Online]. Available: <http://dx.doi.org/10.14722/ndss.2015.23294>
- [31] T. Avgerinos *et al.*, "The Mayhem cyber reasoning system," *IEEE Security & Privacy*, vol. 16, no. 2, pp. 52–60, Mar. 2018. [Online]. doi: 10.1109/MSP.2018.1870873
- [32] S. K. Cha, T. Avgerinos, A. Rebert, and D. Brumley, "Unleashing Mayhem on binary code," in *Proc. of the 2012 IEEE Symp. on Sec. and Priv.*, 2012, pp. 380–394.
- [33] A. Keliris, H. Salehghaffari, B. Cairl, P. Krishnamurthy, M. Maniatakos, and F. Khorrami, "Machine learning-based defense against process-aware attacks on industrial control systems," in *Intl. Test Conf.*, 2016, pp. 1–10.
- [34] N. Stephens *et al.*, "Driller: augmenting fuzzing through selective symbolic execution," in *NDSS '16*, 2016. [Online]. Available: <http://dx.doi.org/10.14722/ndss.2016.23368>

- [35] Y. Shoshitaishvili *et al.*, “SOK: (state of) the art of war: offensive techniques in binary analysis,” in *IEEE Symp. on Sec. and Priv.*, 2016. [Online]. doi: 10.1109/SP.2016.17
- [36] Team Shellphish, “Cyber grand Shellphish,” Phrack, January 25, 2017. [Online]. Available: http://www.phrack.org/papers/cyber_grand_shellphish.html
- [37] G. Palavicini Jr, J. Bryan, E. Sheets, M. Kline, and J. San Miguel, “Towards firmware analysis of Industrial Internet of Things (IIoT) - applying symbolic analysis to IIoT firmware vetting,” in *2nd Intl. Conf. on IoT, Big Data and Sec.*, 2017, pp. 470–477.
- [38] NSA, “Open Source @ NSA,” NSA, 2019. [Online]. Available: code.nsa.gov
- [39] Offensive Security, “About Kali Linux,” Kali Training, 2019. [Online]. Available: <https://kali.training/topic/a-bit-of-history/>
- [40] Pwnie Express, “Raspberry Pwn: a pentesting release for the Raspberry Pi,” Pwnie Express, May 28, 2013. [Online]. Available: <https://www.pwnieexpress.com/blog/raspberry-pwn-pentesting-release-raspberry-pi>
- [41] M. Zalewski, “p0f v3 (version 3.09b),” Icamtuf, 2014. [Online]. Available: <http://lcamtuf.coredump.cx/p0f3/>
- [42] G. Lyon, “NMAP network scanning: OS detection,” NMAP, September 1997. [Online]. Available: <https://nmap.org/book/man-os-detection.html>
- [43] T. Paul, B. Thanudas, and B. S. Manoj, “Packet inspection for unauthorized OS detection in enterprises,” InfoQ, October 10, 2015. [Online]. Available: <https://www.infoq.com/articles/tcp-syn-security-unauthorized-os>
- [44] V. NullArray, “AutoSploit,” GitHub, January 30, 2018. [Online]. Available: <https://github.com/NullArray/AutoSploit>
- [45] M. Graeber, “PowerShell magazine: PowerSploit,” PowerShell Magazine, July 8, 2014. [Online]. Available: <https://www.powershellmagazine.com/2014/07/08/powersploit/>
- [46] A. V. Leonov, “Fast comparison of Nessus and OpenVAS knowledge bases,” AV Leonov, November 27, 2017. [Online]. Available: <https://avleonov.com/2016/11/27/fast-comparison-of-nessus-and-openvas-knowledge-bases/>
- [47] C. Heffner, “Binwalk,” Kali, February 15, 2014. [Online]. Available: <https://tools.kali.org/forensics/binwalk>

- [48] Attify, “Firmware Analysis Toolkit,” GitHub, July 8, 2018. [Online]. Available: <https://github.com/attify/firmware-analysis-toolkit>
- [49] Altaro Software Staff, “Hyper-V terminology - host operating system or parent partition?,” Altaro, February 6, 2012. [Online]. Available: <https://www.altaro.com/hyper-v/hyper-v-terminology-host-operating-system-or-parent-partition/>
- [50] Microsoft Support, “Windows XP support has ended,” Microsoft, 2019. [Online]. Available: <https://support.microsoft.com/en-us/help/14223/windows-xp-end-of-support>
- [51] Microsoft Support, “Support for Windows 7 is ending,” Microsoft, 2019. [Online]. Available: <https://www.microsoft.com/en-us/windowsforbusiness/end-of-windows-7-support>
- [52] S. Langasek, “Ubuntu 8.10 reaches end-of-life on April 30, 2010,” Ubuntu, December 15, 2012. [Online]. Available: <https://web.archive.org/web/20121215114257/https://lists.ubuntu.com/archives/ubuntu-security-announce/2010-March/001067.html>
- [53] NetMarketShare Support, “Operating system share by version,” Net Marketshare, April 14, 2019. [Online]. Available: <https://tinyurl.com/y4fb78co>
- [54] MITRE, “CVE-2008-4250 detail,” NIST, October, 23, 2008. [Online]. Available: <https://nvd.nist.gov/vuln/detail/cve-2008-4250#VulnChangeHistorySection>
- [55] G. Lyon, “NMAP network scanning: host discovery,” NMAP, September 1997. [Online]. Available: <https://nmap.org/book/man-host-discovery.html>
- [56] PostgreSQL Global Development Group, “What is PostgreSQL,” PostgreSQL, 2019. [Online]. Available: <https://www.postgresql.org/about/>
- [57] Greenbone Networks GmBH, *Greenbone Security Assistant*. Greenbone Networks, 2018. [Online]. Available: www.greenbone.net
- [58] Greenbone Networks GmBH, “Vulnerability management,” Greenbone Networks, 2019. [Online]. Available: <https://docs.greenbone.net/GSM-Manual/gos-4/en/vulnerabilitymanagement.html>

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