CARTT: Cyber Automated Red Team Tool

Joseph Plot
Naval Postgraduate School
japlot@nps.edu

Alan Shaffer
Naval Postgraduate School
alan.shaffer@nps.edu

Gurminder Singh
Naval Postgraduate School
gsingh@nps.edu

Abstract

Military weapon systems are often built using embedded, non-IP (Internet Protocol) based computer systems that are not regularly updated and patched due to their isolation. As adversaries expand their capability to exploit and penetrate these systems, we must be able to verify they are not susceptible to cyber-attack. Currently, cyber red teams are employed to assess the security of systems and networks in isolated environments, however, this method can be costly and time-consuming, and the availability of red teams is limited. To address this need and resource shortfall, we have developed the Cyber Automated Red Team Tool (CARTT) that leverages open source software and methods to discover, identify, and conduct a vulnerability scan on a computer system's software. The results of the vulnerability scan offer 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.

Keywords: Red team, cyber, network, security, software vulnerability

1. Introduction

The cybersecurity posture of a military organization's computer devices, specifically those without Internet connectivity, is often overlooked. 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 the service branches conduct cybersecurity assessments on new weapon systems with support from the National Security Agency (NSA) and U.S. Cyber Command, although these two organizations are not primarily responsible for identifying vulnerabilities in 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 effectively allows 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 key contribution of this work is a portable cyber red teaming tool called Cyber Automated Red Team Tool (CARTT) that is designed to identify and assess cybersecurity vulnerabilities on computer systems not directly connected to the Internet, and to provide users with recommendations to mitigate the cyber threats associated with these vulnerabilities. CARTT is designed to overcome the resource limitations of current red teams conducting remote cybersecurity assessments on cyber-physical systems. Ideally, CARTT could be widely deployed as a cheap, convenient, and effective cybersecurity tool that would enhance computer systems security by complementing other defense-in-depth measures.

This paper represents work in progress on automating the actions of red teams by automatically executing a series of commands to search and identify hosts on a target network, and then automatically scanning those hosts for vulnerabilities. Following this, CARTT will provide recommendations to mitigate the cyber threats based on the identified vulnerabilities, and automatically launch cyber exploits against those threats in order to fully "red team" the target network.

The rest of this paper is organized into four sections: background, CARTT system design, CARTT system implementation, and conclusions.

URI: https://hdl.handle.net/10125/64562
978-0-9981331-3-3
(CC BY-NC-ND 4.0)

2. Background

The DoD continuously faces attacks from Advanced Persistent Threats (APTs) that are sophisticated, well-resourced, and highly motivated, and whose goal is to extract or compromise sensitive data. An APT can conduct an attack over several years and target "highly sensitive economic, proprietary, or national security information" [4]. 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" [5]. The report also noted that troops have a false sense of security during large-scale exercises because the cyber environment is not hostile enough to accurately depict the actual threat faced by most DoD systems against an APT [3], [5]. 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 further by naming 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" [6]. 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 [7]. Both of 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.

2.1. DoD Cyber Red Teams

A DoD cyber red team is authorized to mimic an adversary's behavior by conducting exploitation techniques or cyber-attacks against a specific target or government capability [8]. 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 [8]. 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 [9]. 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 [5]. The reasons for this shortfall are manifold.

It can take as long as seven years for military members to be adequately screened and to receive the extensive training required to become proficient in cyber red team operations [10]. However, in 2017 DOT&E observed that military personnel assigned to cyber billets are kept to a regular duty station rotation cycle, typically leaving after three years. This prevents them from gaining the required cyber experience to transition from journeyman to master during a cyber tour [5]. 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 [5], [9]. The assessment also recognized the importance of retaining skilled civilian and contractor personnel through selective hiring practices and job continuity [5]. It recommended keeping personnel who can fully understand a government computer system and quickly recognize abnormalities on a network.

Regrettably, even though DoD cyber red teams are trained to mimic an adversary's behavior, in practice, they 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" [11]. This apprehension from DoD leaders stems from a lack of understanding of the benefits of using a red team to expose their network's vulnerabilities. Combatant commanders trained in traditional military tactics are often reluctant to build realistic cyber threat scenarios and incorporate them into their regular training regimen due to the fear that the cyberspace operations may interrupt the command's primary training objectives. Additionally, most DoD personnel forgo any emphasis on cybersecurity defense by treating it as an administrative function rather than a warfighting capability [9].

2.2. Related Work

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 required, therefore an automated approach 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 their results simultaneously [12].

One solution is to use a vulnerability assessment framework that can integrate the devices and applications that communicate by sorting scan results through a management interface and then setting management policies [12]. The Metasploit Framework (MSF) is an example of such an 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 to scan for vulnerabilities on an individual system or a network of targets [13].

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 [14]. 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, with the intent of discovering vulnerabilities and protecting a company's information technology assets.

The Pentest box runs 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.

The Mayhem Cyber Reasoning System is yet another vulnerability analysis tool, 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 [15]. 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 [16]. 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. Mayhem was shown to manipulate 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 [15]. Unfortunately, Mayhem resides in a large server rack which makes it infeasible to use as a portable vulnerability analysis or red teaming tool.

A DoD team at the Naval Information Warfare Center (NIWC), formerly 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 Industrial Internet of Things (IIoT) devices [17]. They intended to conduct an automated IIoT firmware analysis in search of malicious content.

To accomplish this, NIWC researchers 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 NIWC 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.

Although this is not an exhaustive list, there are a large number of academic papers and research projects that are actively attempting to automate the process of identifying, assessing, and mitigating the risks associated with automated vulnerability assessments. A summary of related work is provided in Table 1.

Table 1. Summary of related work

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 exuction**	**2010**	**Custom Tool (only tested in lab)**	**Custom (SimpIL)**	**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

3. CARTT System Design

CARTT was designed to be used by individuals without expert knowledge of red teaming techniques, or penetration testing. To this end, it automates the various phases of a red teaming event, so that they may be performed by network administration users who are not qualified cyber red team members.

CARTT functions through a GUI that is at a level of abstraction above the CLI, which is the normal sphere of operation for red team operations. The CARTT frontend gives the user a set of options to conduct various portions of the vulnerability scan and assessment, while the backend runs Python scripts with the previously mentioned tools on a Kali Linux distribution. The front-end GUI design is intended to be simple and to support the overall goal of CARTT's red teaming tasks.

We have developed CARTT as a framework that leverages open source tools used for host discovery, OS fingerprinting, vulnerability scanning, and user feedback, and can seamlessly combine these tasks into a single user-friendly device. CARTT was designed to test DoD networked and embedded devices not directly connected to the Internet, to include mission and non-mission critical computer systems onboard aircraft, ground vehicles, ships, and submersibles. 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.

CARTT 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 attempting to provide cybersecurity through obfuscation or behind a private company's intellectual property copyright.

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.

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. Also, the distributions often require the user to be comfortable navigating through the Command-Line Interface (CLI) of a Unix system, as opposed to a user-friendly Graphical User Interface (GUI). To reduce the user's learning curve and to make the system more comfortable to use, CARTT uses GUI-based scripts to execute tasks for 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. This research focused on using Kali Linux as the primary CARTT distribution due to its high number of pre-installed tools, available support documents, and robust online community.

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 CARTT'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 CARTT 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 [18].

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 [19]. Nonetheless, quickly discovering hosts on a network and accurately identifying their OS enables CARTT to tailor its vulnerability scan, decrease the number of unnecessary follow-up scans, and ultimately reduce the number of false positives.

CARTT 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 bypass the potentially cumbersome process of attempting to gain initial access onto 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. From here, a variety of available tools can be used to automatically scan for vulnerabilities, including OpenVAS, Nessus, Core Impact, and Nikto. To keep CARTT as a practical and inexpensive tool, we used the open-source vulnerability scanner OpenVAS, eschewing the pricey licensing fees of Nessus and Core Impact.

Fortunately, OpenVAS is designed to work as a module within the MSF which allows a CARTT user the opportunity to create targets and run vulnerability scans from 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 attempts to run MSF modules to exploit them by creating "reverse TCP shells and/or Meterpreter sessions" [20]. However, this framework would fail to be a useful CARTT 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 [21]. Although PowerSploit contains a repository of capabilities that leverages the functionality of Windows PowerShell, CARTT will use a Unix based framework to reduce the complexity of swapping between PowerShell and Unix commands, and increase its compatibility with other tools.

OpenVAS, through the Greenbone network, uses the National Vulnerability Database (NVD) which is maintained by the National Institute of Standards and Technology (NIST) as a repository to aid in the

automation of vulnerability management. Specifically, the NVD provides OpenVAS with an updated collection of Common Vulnerabilities and Exposures (CVEs), misconfigurations, and security flaws to help red team members quickly analyze hosts.

Unfortunately, no single vulnerability scanner can identify all potential vulnerabilities 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 in the CVE database; however, he admitted that this discrepancy could be that the vulnerability assessment vendors are ignoring old software vulnerabilities that only exist in outdated or deprecated OS distributions [22]. This does not mean the information provided by a vulnerability assessment tool should be rejected, but rather its results should be seen as a subset of the possible vulnerabilities maintained in a threat database.

Interestingly, CARTT also needs 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 CARTT since there exists the possibility that it could inadvertently infect an isolated and malware-free system during a routine vulnerability scan (if the CARTT 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 CARTT 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. CARTT is only one layer in a comprehensive defense-in-depth strategy that employs physical, technical, and administrative security controls.

4. CARTT System Implementation

CARTT was implemented using a Kali Linux distribution due to the latter's wealth of pre-installed penetration testing tools. The initial test system was built using a LAN of VMs within the Cyber Battle Lab (CYBL), a Type I hypervisor physically located at the Naval Postgraduate School (NPS) campus. The VMs used in this experimental system 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.

Microsoft ceased providing software support for Windows XP in 2018 and has publicly stated that Windows 7 will no longer be receiving support or security updates after January 2020 [23], [24]. Similarly, Ubuntu 8.10 reached its end-of-life support in 2010 [25]. Despite this, using these OSs for our testing provided 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 [26]. Second, according to the 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 [23]. Further, small embedded devices typically employ Linux kernels because they are free and lightweight (in terms of memory usage and total lines of code), so we 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.

4.1. The CARTT GUI

CARTT uses a Python GUI library called Tkinter 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 system prompt (see Figure 1). 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.

For the CARTT GUI, each button created by Tkinter is used to call a function that initiates a set of predetermined commands to be executed on the CLI. 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 a CARTT user wants to change the standard input or output stream while conducting a vulnerability analysis, a change in the CARTT source code is required, as opposed to simply updating the command string on the CLI. However, we feel that the benefit of using CARTT greatly reduces the steep learning curve required for CLI usage. Further, the controlled nature of the GUI can limit the user's ability to cause unintended, harmful, or destructive effects on the target system.

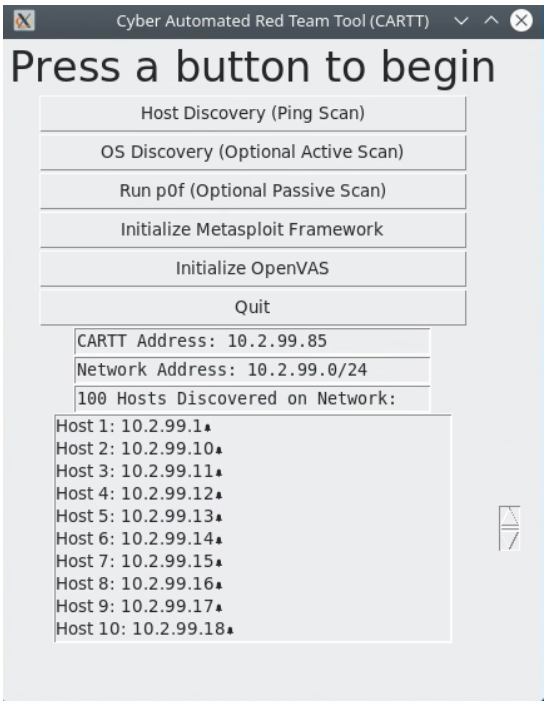

Figure 1. CARTT GUI screenshot

4.2. Scenario and CARTT Functionality

In our test scenario, we assumed that a system operator is tasked with conducting a regularly scheduled cybersecurity vulnerability assessment on their command's automated weapon system. Most service members within a DoD 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. To perform the vulnerability assessment, CARTT can be directly connected by a technician to the closed network during regular working hours, while system users conduct routine operations. The specific operational details of the weapon system are irrelevant to this scenario as long as the CARTT user can physically connect to the network. Figure 2 shows the overall CARTT process flow.

During the cyber reconnaissance phase, the technician begins the first CARTT task by conducting a reconnaissance of the target system to determine the overall network topography. After the user initiates the CARTT 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. CARTT parses this output data in search of its newly assigned IP version 4 (IPv4) address in dot-

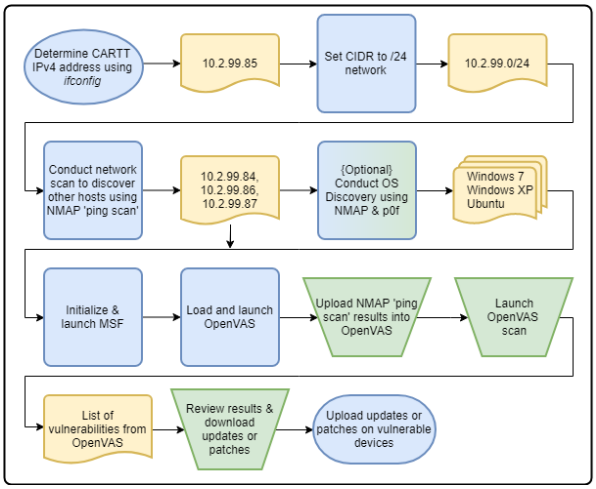

Figure 2. CARTT flow diagram

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 CARTT 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, based on the CIDR used. After the reconnaissance phase, CARTT creates a text file that stores the new network address in CIDR notation and displays both the CARTT assigned address and the network address to the user on the CARTT window frame.

Next, CARTT uses the host's network address to discover all 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, which 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 [27]. 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 *-sn* option does not have any detrimental performance effects on the host during the scan. The results of this ping sweep are then recorded in another text file created by CARTT, which is subsequently parsed and displayed to the user on the CARTT GUI.

At this point, CARTT 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 CARTT 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. CARTT uses NMAP's *-O* option, which allows NMAP to send several TCP and UDP packets to the target systems for TCP/IP stack fingerprinting [19]. The output from this scan provides a description of the OS, vendor name, version number, and device type. This information may be useful for a CARTT user to identify and understand the types of OSs on their network during a large-scale audit.

Additionally, the CARTT 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.

After the OS detection phase is complete, CARTT 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, CARTT starts an open-source relational database management system called PostgreSQL and then initializes the MSF database [28]. Afterward, *msfconsole* is launched to provide a centralized interface between CARTT and MSF's capabilities to incorporate executing external commands.

Once *msfconsole* is running, CARTT 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. 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 CARTT is configured to save a list of the live or active hosts that it encountered during its host discovery phase, the CARTT 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 CARTT 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, XML, and 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 CARTT 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.

4.3. CARTT Testing

We conducted CARTT testing on two identical closed networks within the NPS CYBL. The results discussed in this research 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 CARTT. Within each LAN, two separate tests were conducted to determine CARTT'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.

For the first test, the CIDR prefix was set to 10.2.99.84/30 to ensure that CARTT examined a maximum of three outdated, vulnerable hosts. The goal was to scope the CARTT scan to a small address space before proceeding to a /24 network. After running the scan, CARTT accurately found all of the devices on the network and made an initial determination of each host's OS. Although CARTT 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.

The output of the host discovery scan was sent to a separate text file for use by the GSA when it runs a vulnerability scan within the OpenVAS framework. To do so, the CARTT user must manually upload the text file as a new target under the GSA "Configuration" tab. 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 " [29].

The second round of tests was conducted using a /24 network with one hundred VMs to determine how CARTT would perform against a larger subnet. 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, CARTT 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.

Although analyzing the amount of time it takes CARTT to conduct a scan successfully was not an objective of this research, it is nonetheless an essential consideration for the CARTT user. Specifically, CARTT users would need to understand that scanning an extensive network of devices may have secondary effects on CARTT'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.

5. Conclusions

The main goal of this research was to develop a portable cyber red teaming tool, CARTT, capable of identifying and assessing the cybersecurity vulnerabilities on DoN computer systems not directly connected to the Internet, and of providing users with recommendations to mitigate cyber threats against those vulnerabilities. CARTT automates a series of common tasks used by DoD cyber red teams to conduct vulnerability assessments on networked systems. From our testing we concluded that CARTT can be effectively employed by users with limited cybersecurity knowledge to perform vulnerability scans on networks, and subsequently receive recommendations to mitigate associated cybersecurity threats.

While our approach has been successful, several other tools and techniques were considered in this research, but not integrated into the prototype due to time and resource limitations. Future research could target CARTT's deficiencies and extend its capability for automating vulnerability assessments on embedded devices without Internet connectivity.

For instance, NSA maintains a publicly available repository of open source software that could be incorporated into CARTT's current functions [30]. Their 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" [30]. In CARTT's case, AtomicWatch could be used to scan for keywords or phrases on the log file created by p0f. 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" [30]. This tool would be used in CARTT to determine security settings on Linux devices by displaying the read, write, and execute permissions enforced by the kernel, and help ascertain whether or not CARTT has access to certain regions of memory on the target device.

To fully assess a target device's vulnerabilities, we acknowledge that it would be relevant to test its firmware binary. However, since most firmware extraction techniques involve physical interactions with the circuit boards and at least a basic knowledge of reverse engineering, this was beyond the scope of CARTT's current capabilities. If the firmware image could be pulled from a device and uploaded to CARTT, then it would be feasible for CARTT to perform a vulnerability assessment using a tool such as Binwalk on the Kali Linux distribution.

6. 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] R.F. Behler, “The office of the director, operational test & evaluation FY2017 annual report,” Washington, DC, USA, 2018.

[6] D. Trump, “National cyber strategy of the United States of America,” Washington, DC, USA, 2018.

[7] 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.

[8] *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.

[9] J. M. Gilmore, “The office of the director, operational test & evaluation FY2016 annual report,” Washington, DC, USA, 2016.

[10] 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.

[11] 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.

[12] 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.

[13] MSF Development Staff, “Importing data,” Metasploit, October 17, 2018. [Online]. Available: https://metasploit.help.rapid7.com/docs/importing-data.

[14] L. Epling, B. Hinkel, and Y. Hu, “Penetration testing in a box,” presented at the InfoSec ’15, Kennesaw, GA, USA, October 10, 2015.

[15] T. Avgerinos *et al.*, “The Mayhem cyber reasoning system,” *IEEE Security & Privacy*, vol. 16, no. 2, pp. 52–60, Mar. 2018.

[16] 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.

[17] 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.

[18] M. Zalewski, “p0f v3 (version 3.09b),” Icamtuf, 2014. [Online]. Available: http://lcamtuf.coredump.cx/p0f3/.

[19] G. Lyon, “NMAP network scanning: OS detection,” NMAP, September 1997. [Online]. Available: https://nmap.org/book/man-os-detection.html.

[20] V. NullArray, “AutoSploit,” GitHub, January 30, 2018. [Online].Available: https://github.com/NullArray/AutoSploit.

[21] M. Graeber, “PowerShell magazine: PowerSploit,” PowerShell Magazine, July 8, 2014. [Online]. Available: https://www.powershellmagazine.com/2014/07/08/powersploit/.

[22] 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/.

[23] Microsoft Support, “Windows XP support has ended,” Microsoft, 2019. [Online]. Available: https://support.microsoft.com/en-us/help/14223/windows-xp-end-of-support.

[24] Microsoft Support, “Support for Windows 7 is ending,” Microsoft, 2019. [Online]. Available: https://www.microsoft.com/en-us/windowsforbusiness/end-of-windows-7-support.

[25] 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.

[26] NetMarketShare Support, “Operating system share by version,” Net Marketshare, April 14, 2019. [Online]. Available: https://tinyurl.com/y4fb78co.

[27] G. Lyon, “NMAP network scanning: host discovery,” NMAP, September 1997. [Online]. Available: https://nmap.org/book/man-host-discovery.html.

[28] PostgreSQL Global Development Group, “What is PostgreSQL,” PostgreSQL, 2019. [Online]. Available: https://www.postgresql.org/about/.

[29] Greenbone Networks GmBH, “Vulnerability management,” Greenbone Networks, 2019. [Online]. Available: https://docs.greenbone.net/GSM-Manual/gos-4/en/vulnerabilitymanagement.html.

[30] NSA, “Open Source @ NSA,” NSA, 2019. [Online]. Available: code.nsa.gov.