computers

MDPI

Article

Distributed Attack Deployment Capability for Modern Automated Penetration Testing

Jack Hance, Jordan Milbrath, Noah Ross and Jeremy Straub *

Department of Computer Science, North Dakota State University, Fargo, ND 58102, USA; jack.hance@ndsu.edu (J.H.); jordan.milbrath@ndsu.edu (J.M.); noah.ross@ndsu.edu (N.R.)
* Correspondence: jeremy.straub@ndsu.edu; Tel.: +1-701-231-8196

Abstract: Cybersecurity is an ever-changing landscape. The threats of the future are hard to predict and even harder to prepare for. This paper presents work designed to prepare for the cybersecurity landscape of tomorrow by creating a key support capability for an autonomous cybersecurity testing system. This system is designed to test and prepare critical infrastructure for what the future of cyberattacks looks like. It proposes a new type of attack framework that provides precise and granular attack control and higher perception within a set of infected infrastructure. The proposed attack framework is intelligent, supports the fetching and execution of arbitrary attacks, and has a small memory and network footprint. This framework facilitates autonomous rapid penetration testing as well as the evaluation of where detection systems and procedures are underdeveloped and require further improvement in preparation for rapid autonomous cyber-attacks.

Keywords: cybersecurity; automate penetration testing; attack automation; artificial intelligence; Blackboard Architecture

Citation: Hance, J.; Milbrath, J.; Ross, N.; Straub, J. Distributed Attack Deployment Capability for Modern Automated Penetration Testing. *Computers* **2022**, *11*, 33. https://doi.org/10.3390/computers11030033

Academic Editor: Paolo Bellavista

Received: 3 February 2022
Accepted: 21 February 2022
Published: 23 February 2022

1. Introduction

As the sophistication of hackers and cyberattacks increases, the need for advanced defense techniques has grown as well. One facet of this is being able to prepare for attacks by testing systems through simulated intrusions. Penetration testing has become a standard part of many organizations' cybersecurity programs. It allows the entity to detect vulnerabilities and misconfigurations before attackers can exploit them. Due to the skills required and scope of testing that may be needed for large enterprises, penetration testing can be expensive or may be incomplete.

The goal of this project is to facilitate the testing of the security of large, interconnected systems, reduce cost, and expand testing coverage. This is achieved through using a distributed attack platform that employs an intelligent control scheme to automate vulnerability detection. This paper presents a distributed command mechanism for a system, which, when all of the distinct aspects are assembled, will act like a combination of a command and control system and a botnet.

Complex systems can be difficult to fully assess. The larger a network gets, the more potential routes there may be into that system. Intruders may target the furthest edges of the system that are vulnerable to very specific types of attacks. These types of vulnerabilities could be easily missed during testing. Automated testing allows large systems to be examined more quickly and thoroughly. The proposed system has a node-based architecture, which contributes to the ability to scan networks efficiently. It also provides a realistic simulation of potential complex attacks.

If one attack attempt is thwarted, the attacker—in many cases—will not simply stop attacking. A testing system with independent nodes, such as the one proposed herein allows a realistic persistent threat to be simulated. This will be beneficial for testing infrastructure and similar networks that cannot be fully evaluated without testing multi-homed combined effort and other complex attacks.

The Autonomous Networked Testing System (ANTS) distributed command system presented herein integrates with a Blackboard Architecture-based command system (conceptually based on [1]) to provide the key capabilities of remote attack triggering and node acquisition. The Blackboard Architecture-based command system provides a centralized, multi-homed decision-making capability that can coordinate multiple different attacks. Similar to a determined real-world attacker, seemingly unrelated vulnerabilities in a system may be exploited by the command system to gain access to a system and attempt to achieve attack objectives. This provides a level of complex testing that helps simulate actual risks to critical infrastructure and other key networked systems that require high security.

This paper describes and evaluates the attack node command system developed for the Blackboard Architecture-based command system. These attack nodes will be installed on target systems and discretely receive instructions to carry out commands. They are intentionally designed to be small, modular, and disposable. For testing, this minimizes their interference with normal network operations. It also increases the simulation's fidelity as they are similarly difficult to detect, like many types of malware that might be created for nefarious use. The nodes are easily replicable and replaceable, allowing the system to continue to function even if some nodes are detected.

This paper continues with a review of prior work in Section 2, which provides the foundation for the work presented herein. Next, an overview of the system is presented in Section 3. Section 4 presents the experimental methodology. This is followed by data presentation and analysis in Section 5, before the paper presents its conclusions and discusses needed areas of future work in Section 6.

2. Background

The knowledge used to develop the distributed command functionality for the automated network testing solution presented herein draws from several different areas of previous research. Relevant prior work is now discussed.

First, a general overview of the state of cybersecurity that demonstrates the need for this system is presented. Next, command and control and the MITRE ATT&CK framework are discussed. In Section 2.3, prior work on security testing automation is reviewed. Finally, an overview of relevant past work related to the Blackboard Architecture and artificial intelligence is provided.

2.1. General Cybersecurity

The world is becoming progressively more connected with growing links between firms' internal systems and external entities to support numerous organizational objectives. This increased connectedness brings with it potential security risks—each new pathway conceivably provides a new vector for an attacker to penetrate a system. The benefit of the connections' increased speed and efficiency, thus, brings with it the risk of security vulnerabilities and increased attack surface area [2].

To combat these risks, security staff perform network and software configuration, testing, auditing, and other tasks. Increased risk factors necessitate larger security teams. Larger teams increase cybersecurity costs; however, the potential return on investment (ROI) for cybersecurity is difficult to predict as clear answers about the financial risks of falling behind in cybersecurity are elusive [3]. Firms may see minimal ROI while things are going well; however, should an attack occur without the needed security in place, the loss may be quite notable—if not catastrophic for the firm. This is difficult to juxtapose with other aspects of organizations that may provide easily calculable revenue, cost reductions, or other benefits.

Because of these difficulties, a number of frameworks have been proposed for risk assessment. Mateski et al. [4] present a "generic threat matrix" which can be informed by other techniques, such as MITRE Corporation's "Cyber Prep methodology". Mavroeidis et al. [5] discuss how threat actors can be assessed and described using an ontology, which they present. King et al. [6] focus on the "human factors that contribute" to the problem, focusing

on characterizing threat actors' maliciousness. Zhao et al. [7] show how graph learning can be used as part of the threat prediction process. Multiple studies [8–11] have demonstrated different approaches and benefits to cyber threat intelligence activities.

Informed by these and other risk analysis techniques, firms must consider the cost (including replacement and organizational impairment) of what they are protecting. Organizational reputation is a key asset, and firms that invest in cybersecurity and are viewed as more trustworthy may attract more customers. Firms that suffer a data breach, alternately, suffer from a damaged reputation [12] and may both lose existing customers and fail to attract new ones. Due to the prevalence of attacks, companies are gaining an awareness of the potential threat [13]. Security incidents have been shown [14] to have over an 80% chance of costing a company $10,000, while data-oriented breaches have over a 20% chance of costing the company $1 million.

Numerous approaches to cyber-attack risk reduction have been proposed. Some focus on identifying and securing the attack vectors that an adversary may use. Techniques such as the MITRE ATT&CK [15], Lockheed Martin Cyber Kill Chain [16], and Microsoft STRIDE [17] frameworks can be used for this purpose. However, this may be a task that is beyond the financial means of some firms.

An alternate approach is finding the weakest points through network analysis to direct funding to the most acute areas of need [18]. Tree-based structures [19], such as attack graphs, are tools that can be used for this purpose. Attack graphs [20] consider the network as a collection of individual nodes, which can be attack states, security conditions, vulnerabilities, or exploits. Arrows are drawn between the nodes to indicate their interconnections. This type of diagram can be used to plan an attack via finding the shortest connection to a given target (or the node with the highest value). However, they are also useful for defensive purposes as they allow security staff to determine what areas an attacker might target to enhance these areas' security [20]. Cybersecurity automation using tree-based structures has been previously proposed [21].

2.2. Command and Control

With the increasing focus placed on network security, cybersecurity command and control (C2) systems have been studied extensively. Well-designed C2 systems can enhance attack evasiveness. For example, they can avoid direct (and potentially traceable) connections to the server by having the control unit write commands to a common website where the compromised system's local control software can retrieve them from [22]. Similarly, malware has had to adapt to anti-virus software capabilities to maintain its efficacy. As systems use signatures, or specific characteristics, to identify malware (for example, using regular expressions to look for byte patterns within a file) [23], polymorphism has been used to evade detection. Under this model, the attacker runs the file through a mutator function (that changes its byte patterns, but not its function) before deploying it [24]. This, by itself, will prevent some anti-virus software from detecting the file. Attackers may also use specialized routines to determine if anti-virus software is present that could detect relevant types of malicious activity. If it is detected, the program terminates and does not execute malicious code (preventing detection). The malware, thus, avoids detection by delaying activity until the anti-virus software is not searching for it [23].

Another area of advancement is the use of distributed, node-based attacks, as opposed to one attacker/one target ones. Decentralization and the use of nodes provides redundancy and other capabilities that may aid in attacking protected systems [25]. This model allows nodes that are detected or fail to self-terminate without stopping the overall attack. The approach also supports multi-vector attacks and has significant resilience because, in order to stop the attack, all attack nodes must be located and terminated.

Problematically, this approach creates large amounts of network traffic for malware nodes' intercommunications. Network administrators can use traffic analysis techniques to locate and shut down malicious code based on tracking communications. Cisco's NetFlow, for example, collects metadata, such as IP addresses, network ports, and protocols from

traffic [26], and analyzes it to look for unusual patterns or protocols that may indicate the presence of malware. However, these systems' capabilities are limited by storage, as it would take 7.2 terabytes of space to store one day's data from a single 1 Gbps router [23]. This means that the actual data cannot be stored and analyzed, necessitating a focus on scrutinizing the metadata. To evade these systems, some attackers have begun to hide their traffic behind common protocols using common ports and network protocols. For example, the SkypeMorph malware hid Tor traffic by disguising it as a Skype call [23]. Techniques such as this are effective because they allow malware nodes to communicate while not triggering automatic detection systems utilizing traffic metadata analysis.

C2 systems also need to support lateral movement (moving between multiple machines with the same privilege level on a network to gain access to a target) [27]. This involves moving between compromised machines and often requires complex security capabilities to detect. An attacker can enter through a low-privilege computer (with less security) and move around the internal network to target higher privilege computers (with more security that are connected to fewer devices). Lateral movement can be advantageous, as it can provide a direct connection to the target and reduce the risk of detection [28].

The MITRE ATT&CK framework was created to aid in the understanding of malicious parties' attacks, including those that are C2-based [27]. It allows security teams to consider their system from an adversary's perspective, and, to this end, it provides guidelines as to the stages of an attack [18]. It also facilitates behavioral analysis. Hacks et al. [29], for example, demonstrated how an exercise in the analysis of adversarial behaviors could also be used to analyze security team behaviors during attacks.

2.3. Security Testing Automation

As computer systems and networks become more complex, demonstrating their security has grown in difficulty. While testing known scenarios can be performed easily, this does not prove system security. Systems are still potentially vulnerable to unforeseen attacks [30]. Many current penetration testing tools go through a library of attacks, and, if they do not find success, that may be where testing ends. These tests may be relatively basic and outdated. Increasingly complex systems dictate a need for complex exploits to find their vulnerabilities and secure them [31].

Existing vulnerability assessment tools may fall short of companies' needs. Moreover, despite not fully identifying system vulnerabilities, they may damage the company's network or even a facility's power systems [32]. The tolls cost and potential for damage can deter companies from using them. For critical networks, they may risk causing the very outage that they seek to protect against and prevent, deterring their use [33]. Testing limitations are not the only issue with network security, though. Monitoring systems that overwhelm operators and a variety of other issues are also pervasive [34].

A significant amount of ongoing research focuses on the automation of cybersecurity processes and penetration testing, in particular. While some automated tools are currently publicly available [35], numerous research efforts are poised to prospectively enhance future generations of testing automation tools. Automation techniques for testing Blockchain contracts [36], cloud applications [37,38], Internet of Things devices [39], WiFi networks [40], and web services [41–44] have been proposed, among others. Researchers have also developed vulnerability-specific testers, such as for SQL injection [45].

In addition to this, prior work has introduced a variety of techniques and technologies for automated testing. Approaches based on threat models [46], ontologies and big data techniques [47], static and dynamic analysis [48], Petri nets [49], fuzzy classifiers [50], expert systems [51], planning automation [52], automated pathfinding [53] (including map/graph-based techniques [54]), and agent-based modeling [55] have been proposed. Prior research has also developed a variety of machine learning techniques, such as those using reinforcement learning [56–60] and deep reinforcement learning [61,62], as well as supporting technologies, such as a description language [63].

Despite all of this, some suggest that human penetration testing is still needed due to the capability of human testers to think creatively. Stefinko, Piskozub, and Banakh [64], for example, note that "manual penetration tests are still more popular and useful"; however, they note that manual testers can still make use of automation in their work.

2.4. Blackboard Architecture and AI

The system which the capabilities proposed herein support is based on the Blackboard Architecture. The Blackboard Architecture was introduced by Hayes-Roth [65] based on prior work on the Hearsay II system [66]. It is similar, conceptually, to rule-fact expert systems, giving it lineage back to early AI systems, such as Dendral and Mycin [67–69].

The Blackboard Architecture has been used for developing complex computing structures for systems where a simple algorithm cannot be readily implemented [70]. A basic Blackboard Architecture system can be based on a central node (called a Blackboard) that contains all of the data that is known throughout the system [71]. Any collected data is sent back to the central node [71]. Controller nodes direct the system towards its goals and gather data, while other agent nodes collect information and report to the controllers [71]. Some Blackboard systems are comprised of only a few nodes; however, other larger systems utilize independent nodes to survive the failure of a controller.

The Blackboard Architecture has been used for a variety of applications, including tutoring [72], robotics [73,74], modeling proteins [75], and vehicular control [76]. The system that the capabilities described herein are designed to work with is based on a simplified modern Blackboard Architecture implementation [77], which utilizes rule, fact, and action nodes for decision making and actualization.

2.5. System Need

The proposed system is part of a trend towards assessing the potential for different forms of artificial intelligence's potential use to improve cybersecurity. Juniper research has predicted that the cost of cybersecurity incidents will grow by eleven percent each year in the future [78]. These breaches can be caused by criminal organizations, hobbyist hackers, and industrial spies, among others [79]. Enforcement may be key to stopping them. Since cybercriminals are typically caught using the tracks that they leave behind on systems that they have attacked, this can be difficult as it requires organizations to have enough time to find and process this data. The variance in the data also makes it difficult for security professionals to determine which information is relevant [80]. Intrusion detection and prevention systems (IDPS) are one type of currently available AI countermeasure. Despite not being a complete solution, they can prevent many problems. They are utilized by security teams to detect potentially malicious network activities and abnormal behavior and to immediately provide notification or take protective actions [80].

3. System Overview

This section presents an overview of the overarching system, shown in Figure 1, that the C2 technology described herein is designed to support. This system is based on an extended form of the Blackboard Architecture and utilizes three different node types that work together to perform automatic decision making and attack execution. The first node type is the command node, which is a Blackboard Architecture network and processing system. The system also includes attack nodes, which launch attacks, and verifier nodes, which determine whether attacks have been successful or not. The command node also incorporates a scanning system, which identifies new nodes on the network and classifies them. While the attack node is the focus of this paper, knowledge of the other parts of the system is key to understanding the attack node's integration and operations.

The command node is the controller of the system. It uses a Blackboard Architecture, which stores data as facts and uses a collection of rules to make decisions and launch actions based on the operating environment. The command node communicates with the attack nodes and verifier nodes to send commands and retrieve status information. The system

is designed to allow operators to identify a target in the command system and then let it run autonomously to survey the network environment and launch attacks. For single-node configurations, due to the large amount of data potentially stored on the Blackboard, it should be hosted in a safe environment. Distributed versions of the system are planned that will remove a single command node as a single point of system failure.

Verifier nodes are used to ascertain whether or not targets have been compromised by attacks. They facilitate the assessment of the attack nodes' functionality and are key to updating the command node after an attack in regards to its success (or not) status. Verifier nodes can assess the machine they are installed on or other machines remotely. This type of node is lightweight and has little functionality other than communications and monitoring capabilities.

Attack nodes' purpose, as is expected, is to run attacks on a given system (either against that computer itself, if it is the target or another target) as ordered by the command node. Attack nodes are placed by the command node or its supporting systems. The process starts by gaining access to a foreign system and installing the attack node software. Once the attack node software is present, it connects to the internet and then remains dormant while it waits for the command node to contact it with instructions. When instructions are received by the attack node, they are processed to carry out the attack. Communication from the command system is largely one-way, potentially via intermediaries, so the attack nodes do not report back on their successes or failures. Instead, verifiers are used to collect this data.

It is important to note that while they may be working in unison, the verifier nodes and attack nodes do not communicate directly with each other. Both types of nodes have limited functionality and depend on the command node for decision making, thus they perform little decision making on their own and instead respond to commands and events.

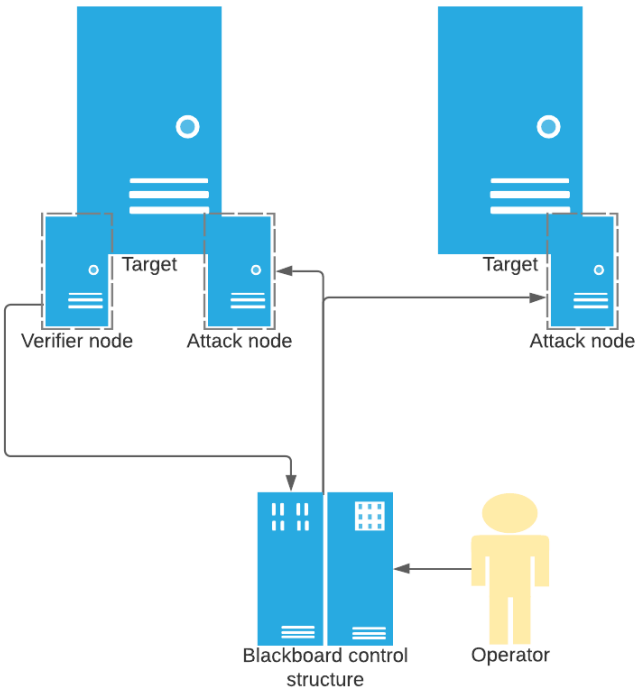

Figure 1. Diagram of a potential installation of ANTS on two target servers.

Commands for the attack nodes can be provided in two formats: attack scripts and attack binaries. Though the two formats store different amounts of meta information, the functional information in both is the same. Attack scripts are instructions for the nodes and are created in a human and machine-readable format. They contain the name of the attack, its description, attack options, and the uncompressed Lua code to be executed. Binaries are created by running the scripts through a processing application that extracts the

information needed for attack node operations, making them largely not human-readable because their information is compressed and stored as binary data. The Lua code is also minified to reduce its size, and information that is not needed by the nodes, such as the attack name and description, is removed.

The attack scripts use options to specify parameters values for the attack. Each option has a name to identify it. They also have associated data types that inform the parser how to interpret the value in the attack binary. The system understands several types, including number, string, data, bool, and list (which is comprised of other data types, including other lists).

4. Attack Node System Design

The attack nodes play a key role in the overall penetration testing system. They are designed to be discretely installed and to then sit dormant while waiting for instructions from either the automated command system or a manual operator. In typical use, the nodes have no ability to communicate back to the system or individual commanding them.

A key design goal was to minimize the size and footprint of the attack nodes while operating on a target system and to reduce network traffic. All of these design choices help reduce the likelihood of node discovery. Thus, a set of standards for their creation and operations has been developed. The attack nodes receive instructions through the network but do not send information back to the controller directly. This helps keep operations hidden; however, it impairs the currency of the controller's information about the current state of the running attacks and the system they are affecting. The developed standards are designed to help attacks execute consistently, reducing the need for feedback data.

Attack nodes are designed to support numerous system types and are, thus, designed to be easily modified and recompiled for new targeted system types. The node code was written in the commonly used language C++, which has compilers created for virtually all potentially targeted systems. Calls to lower-level system functions (which may be different between systems) have been wrapped to allow for easier modification and compilation. The constraints also assist in developing new attacks by providing a standard for how they are presented, reducing variability and making them easier to understand.

Attack scripts are written in a standard format that includes two parts: the attack header and body. The header is human-readable, has minimal syntax requirements and contains information about the attack, including its name, options, and option meta-information indicating whether they are required and, if not, what their default value is. Headers help human system operators understand what is being selected and actioned and aid the attack nodes in comparing the information supplied to the requirements (included in the header). The attack body is written in standard Lua, allowing for portability and flexibility, as Lua is an interpreted language.

After an attack is developed, it is processed to create an attack binary. This is a smaller file that is sent to the nodes, reducing the network footprint. Extraneous information is removed, such as the attack name and whitespace. The Lua is also minified while maintaining code functionality. Binaries are stored in memory by the attack node, providing a library of attacks to execute. The space is used efficiently, with most being devoted to the data itself while still being quickly interpretable. The steps used for attack file processing and validity checking are shown in Figure 2.

When an attack binary is received by an attack node, its processing begins, as shown in Figure 3, with the binary parser checking if the file is valid. It reads the first four bytes of the file, which should be a specific arbitrary number indicating that the file is an attack binary. If the number is not the expected value, the parsing process is aborted, and the binary is discarded. Otherwise, the parsing process continues. Once the attack binary has been verified, the parser reads the number of options that the attack contains, which can be up to 65,535 distinct options. The parser then enters a loop where it reads the name, type, and default value (if provided) for each option. The remaining data in the file is a compressed version of the Lua code.

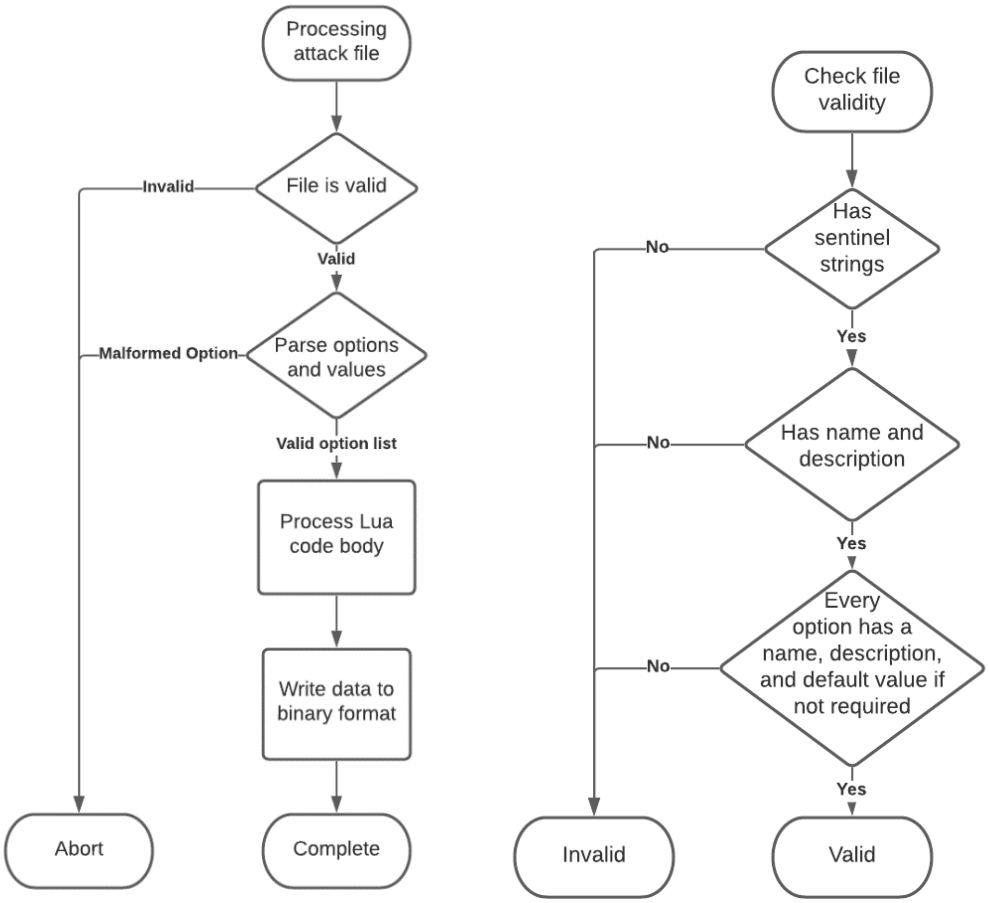

Figure 2. (**Left**): Flowchart for attack file processing. (**Right**): Flowchart for checking attack file validity.

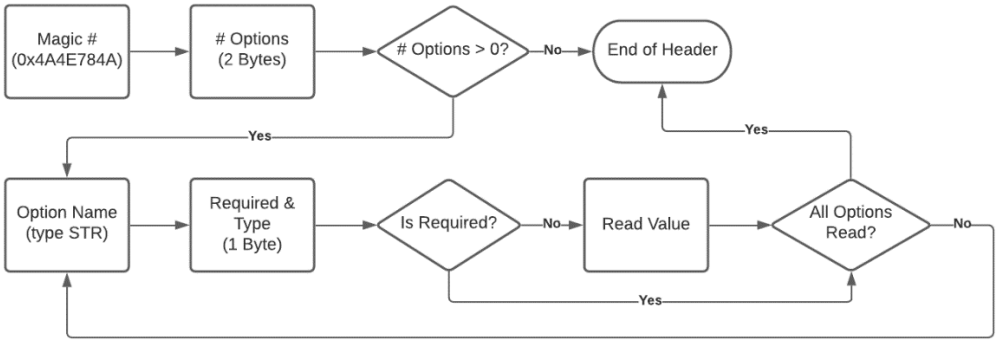

Figure 3. Flowchart for an attack node parsing an attack file binary.

There are five data types used for attack options: NUM, STR, DATA, LIST, and BOOL. NUM is a number type, equivalent to a double-precision floating-point (float64) and is eight bytes in size. STR is a string type, which is a sequence of ASCII characters terminated by a binary zero, with a max length of 128 characters. The methodology for processing the string type is shown in Figure 4. DATA is a data arbitrary type; it is prefixed by an unsigned eight-byte integer that indicates the number of following bytes. LIST is a list format, which consists of any number of any of the other types (including other lists). List entries do not need to be of homogeneous type. The steps used for processing the LIST type are shown in Figure 5. Finally, BOOL is a single-byte integer for which any value besides zero represents true, and exactly zero represents false.

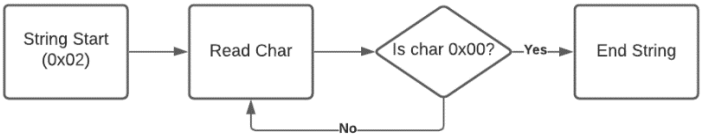

Figure 4. Flowchart for how the string type is processed.

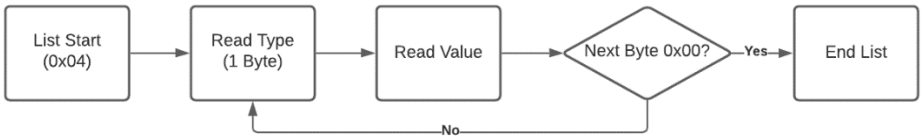

Figure 5. Flowchart for how the list type is processed.

Each attack has an attack identifier, which is an MD5 hash of the original attack script. This allows attacks to be differentiated by both their body contents and headers, as different default header values can completely change how an attack operates.

Attacks are executed by sending a command to the node with information for an existing loaded binary. Attack commands use a similar format to the attack header and are similarly processed for size reduction. The command contains the attack identifier and the options' values.

Once an attack node receives a command, it parses it (as shown in Figure 6) to verify that it has the binary requested (as shown in Figure 7). If so, the node executes the Lua using the supplied options as parameters. The Lua is interpreted and executed through a proprietary library that has been integrated directly into the node software itself. If the binary is not present, the node is unable to process the command further.

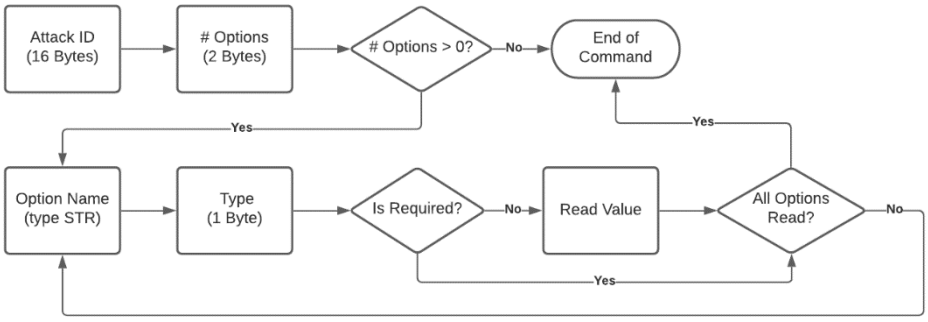

Figure 6. Flowchart for attack command parsing.

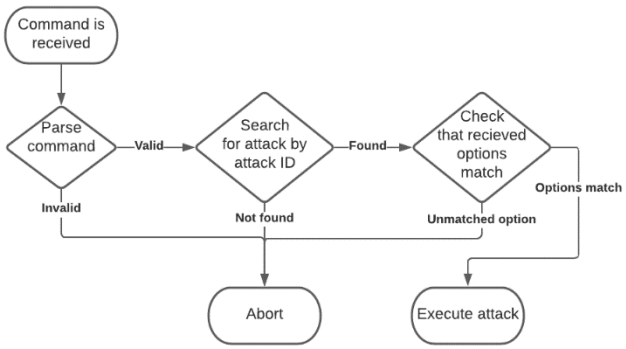

Figure 7. Flowchart for how an attack node decides to run a received command.

Lua was selected for its small size and the ease in which it can be statically compiled into and then executed from a single binary file. Scripts written in the language can also be compressed down quite significantly due to the immense malleability of it. The largest disadvantage of Lua is the fact that it lacks most of the functionality of typical programming languages and custom-built cyber-attack languages. This is most prominently apparent

when executing commands on a host system and with networking functionality. A library has been implemented using multiple tables of functions that are loaded before script execution. It aids in executing commands on host systems and retrieving their output. A sockets library is also included providing limited networking functionality.

The system intentionally includes no mechanism to assess whether an attack has been executed successfully as there is no current or planned functionality to respond back to the command node. If confirmation is needed, this can be obtained through external verification of the results of the attack. This could be verifying that an existing post-attack state exists. For example, an exploit that planted a reverse shell on a system could be verified by determining whether a response is received to the specified endpoint. Similarly, a denial-of-service attack's success could be verified via attempting to connect to the targeted server to see if it responds or not. In some cases, verification may not be required, such as in a scenario where many machines are targeted concurrently, making the success of any one of limited importance.

5. System Evaluation

This section evaluates the proposed autonomous penetration testing system. First, in Section 5.1, a real-world attack scenario is used to evaluate the system's performance. Second, the limitations of the system are discussed. Third, the system is compared to other conceptually similar systems. Fourth, metrics that can be used to evaluate the system are discussed and used for evaluation purposes.

5.1. Evaluation of Performance under Attack Scenarios

The attack node software has been designed to carry out attacks on both the system hosting the node and on a remote system targeted by the node. Figure 8 depicts this capability. To validate this capability and demonstrate the efficacy of the system concept and design, several attacks were developed and deployed. One was designed to target the host computer, and two were designed to target a remote computer.

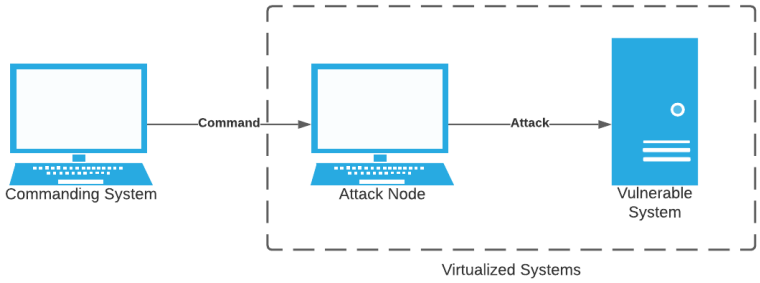

Figure 8. A visualization of the testing environment.

The tests were conducted in a virtualized and sandboxed testing environment. Both the attack node and the vulnerable system that was being targeted were virtual machines. The operating system used for the attack node was Lubuntu 20.04.3 LTS. Aside from the executable attack node being placed onto the system, it was in a completely default install state with no additional packages installed. The attack node was running in 64-bit mode, though a 32-bit system would perform identically. The vulnerable system that served as a target was MetasploiTable 2.0.0. A default install of this operating system was used, with no modifications made to it.

Each scenario began with the attack node software already running on the attacking system. This configuration could be the result of an attack against this system that successfully compromised it and used this access to install the attack node software and configure it to automatically start. Alternately, other mechanisms used to distribute malware could be used to reach this state. For some penetration testing scenarios, all systems might start with the system pre-installed in preparation for testing. Each scenario involved a command being sent to the attack system from the command system. The results were then observed.

5.1.1. Attack against Attacking System

The first scenario, the attack that targets the attack node itself, was a simple denial of service attack. With the attack node software operating on the computer, a command that shuts down the computer running it was issued by the attack node software. This command is operating system-type specific, and the attack was, thus, designed for a Linux-based system. It ran successfully on the Lubuntu system, resulting in a denial of service to other prospective system users. In a real-world environment, the command module would need to detect the operating system type and trigger the operating system-appropriate command. This would already be known, in most cases, from the process of compromising the system and placing the attack node; if it was not, it would need to be remotely detected. This attack was designed with no options; thus, the attack command simply identifies the attack to start it, with no additional details being required.

5.1.2. VSFTPD Attack

The second scenario, which was the first attack that targeted a remote machine, used the well-known VSFTPD v2.3.4 backdoor [81]. This is an exploit that MetasploiTable 2.0.0 is intentionally vulnerable to.

The attack is initiated by supplying a command, a remote host IP address, and a remote port, which the FTP server is running on. The attack attempts a login with a username ending in ":)" and an intentionally invalid password. It then closes the socket. If the attack is successful, the backdoor opens a console that can be telnetted to on port 6200. The attack then attempts to connect to port 6200 and sends a command (the Linux command "id" was used) to verify the success of this first step. If this is successful, the supplied command option is executed on the remote system. For the purposes of this scenario, a shutdown command was used.

This scenario was executed successfully against a MetasploiTable 2.0.0 system. Success was verified by confirming that the system shut down.

5.1.3. UnrealIRCD Attack

The third scenario, which was the second attack that targeted a remote machine, used the UnrealIRCD 3.2.8.1 backdoor [82]. This is another exploit that MetasploiTable 2.0.0 is intentionally vulnerable to.

The attack is, similarly, initiated by supplying a command, a remote host IP address, and a remote port that the IRC server is running on. The attack is carried out by connecting to the IRC server and sending a string that begins with "AB;" followed by the command to execute. The supplied command was prepended with "AB;" and the attack node then connected to the target IRC server and sent the string. Like with the previous scenario, a shutdown command was used for this scenario as well.

This command was also tested against a MetasploiTable 2.0.0 system. The success of the attack was verified by confirming that the system shut down.

5.1.4. Results and Analysis

All three attacks were able to execute successfully in the testing environment. Each achieved arbitrary command execution on either the system running the attack node or the vulnerable remote target. The two remote attacks are of note, as using the arbitrary command execution capability, they would allow the attack node to upload and execute a copy of itself on the remote system, facilitating the growth of the attack node collection.

While the tests were successful, the system has several limitations that could lead to detection in an actual deployment, which are discussed in the subsequent section.

5.2. Limitations

The system has a number of limitations. These are now analyzed.

Perhaps the most obvious limitation is how networking is being handled. The attack node is bound to an opened port so that it can listen for instructions from the control node.

Binding to a port could be noticed by scans that are regularly performed by information technology professionals to prevent exactly this type of activity. The network traffic sent to the node is also not encrypted or hidden in any way, meaning that anyone inspecting relevant packets can see the instructions transmitted. Not only could this be used to detect system operations it could also reveal the location of the control node.

Overall, the framework provides significant functionality and performs similarly to other attack frameworks. A large number of attacks can be performed with basic networking, host command execution, and a Turing complete language. Many attacks require low-level networking control, and having system sockets available to attack scripts allows attacks written for other frameworks (such as Metasploit) to be duplicated with ease.

A key drawback of only having low-level sockets functionality is the impairment of being able to quickly implement attacks that use common, high-level protocols. For example, the lack of TLS support makes working with HTTPS connections cumbersome.

Similarly, while Lua works well for quick scripts, the language itself is missing several key features. A variety of simple capabilities, such as splitting strings at delimiters, are not included in Lua. This could be overcome with the creating of a larger "standard library" for use for this system, which is a key area for future work. Existing methods, which have been added to Lua from the host language of C++, have benefited from faster performance than they would with the interpreted language. The aforementioned string splitting operation, implemented in C++ instead of Lua, for example, performs faster with less overhead.

In general, the attack nodes are principally limited by Lua itself. While most scripting tasks can be readily performed with the current system, computationally expensive and complex tasks can be difficult to implement. Lua, for example, lacks multithreading and parallel computation capabilities. This impairs its ability to implement an exploit that involves a race condition (i.e., two actions run with minimal delay in between them). While there are existing libraries that add this functionality [83], the goal of minimizing attack node size would likely require that limited versions of these capabilities be implemented from scratch. Lua also lacks a way to suspend a program's execution for an amount of time. While a sleep command can be executed on the host computer, this requires effort to determine what operating system the host is running. There is also no native Lua functionality to determine which operating system it is running on.

In addition to limited networking support, there are a variety of other limitations that could be overcome with attack node enhancements. Another example is the lack of included functionality to support deploying new attack nodes, leaving all required functionality in this area to attack script development. A set of methods to automatically supply bitstreams for use in attack scripts that can be used to send data to the target system could aid in this area. There is also no current ability for attack scripts to trigger or interact with other attack scripts on a system, and the lack of a feedback mechanism prevents the command node from simply triggering scripts in succession after each completes. A mechanism could be added to support one script calling others. There is similarly no way for a node to access a binary that it is instructed to use but does not have. Adding a mechanism for obtaining binaries—either from other nearby attack nodes or centralized locations—would enable operations without the command node having to track what is available on the attack nodes.

Another issue is that adding bespoke functionality via C++ requires modification to the attack node code and recompiling. This could be remedied with enhanced pre-created functionality and the development of a "plug-ins folder" capability where C++ files can be dropped into, being automatically compiled into the project, immediately exposing their new functionality for use in attack scripts.

Other key potential improvements include the creation of a standard attack library that would install with the base node software, ensuring that even fresh nodes have the ability to perform basic attacks. Node types or base configurations could be developed that have purpose-driven collections of base attacks. Nodes could also be augmented with the ability to communicate back to—or be queried by—the command node, which could

be used to gain an understanding of what is going on when attacks do not appear to be succeeding (potentially reducing overall network traffic by reducing guesswork).

It will also be important for production systems to ensure that commands have come from a trusted party. Preventing a penetration testing system from being taken over by a malicious party is a key concern and integral to eventual system adoption for production environments.

5.3. Performance Comparison

It is difficult to compare the design of the proposed system to other projects with similar methodologies or goals due to the secrecy of the implementation surrounding this type of software. Much of the discussion of the topic is theoretical, redacted, or vague. While the goals of the systems are somewhat different, penetration testing versus malicious activities, perhaps the most similar type of software in wide use is botnets. These are collections of compromised machines that host software placing them under the control of a single person or group (typically a hacker with malicious intent or state actor). Vulnerable and widely available devices (such as Internet of Things devices) will be targeted and compromised for use as bots allowing the controller to have computing power spread over a large area.

The approach provides resiliency and attack capability uptime. A botnet that has control of over 200,000 devices, such as the Mirai botnet [84], can readily lose 1000 or more with minimal impact to functionality and capability. Unlike botnets, which may seek to perform large, distributed denial of service (DDoS) attacks, ANTS is intended to operate within a single network (or group of closely interconnected networks) being tested. Thus, it has somewhat different design goals, though some methods of achieving these goals are shared.

Resiliency through redundancy is also a design goal of ANTS; however, given the different goals of the two systems, it has been approached somewhat differently. Instead of having hundreds or thousands of nodes running at the same time, ANTS uses a smaller and more focused approach. The design, which was described in Section 4, allows the control node to send commands to alternate nodes if one fails. Nodes are also designed to be able to be quickly created, allowing rapid deployment, if needed. Thus, while not having the same scope or redundancy as bot nets such as Mirai—nor needing it for penetration testing—the system design facilitates nodes' availability to be commanded by the control node for testing.

ANTS and botnets are closely related in their need to command remote nodes. Vogt, Aycock, and Jacobson [85] note that botnets will often communicate through IRC channels, spoofing browser traffic, or otherwise disguising their commands to make them less detectable. While the initial implementation of ANTS is designed to use open ports for communications, it could easily be converted to utilize communications via an intermediary (such as social media posts) or via a disguised channel. Thus, ANTS could readily employ similar techniques to botnets to make its traffic less detectable.

Given the foregoing, it is apparent that ANTS draws on ideas from botnets but focuses on the needs of its penetration testing application by using smaller groups of targets where each node plays a more direct role in the system's functionality. This more-targeted approach could be aptly termed to be a "spearnet", based on its greater precision and resource localization.

5.4. Metrics

One key metric by which the proposed system can be evaluated is the amount of data that is sent between the control and attack nodes. This is an important consideration, as the time at which this data is being sent is one of the most dangerous parts of the lifecycle of a node. Reducing this metric is very important to reduce the detectability of the system.

When operating in TCP mode, ANTS requires a minimum of four packets to send a command to an attack node. Three of these packets are used for the TCP handshake

process, and the fourth is the data packet that contains the command and its parameters. More data packets may be used when sending large commands or large data parameters.

This can be compared to the Mirai botnet, which was analyzed in [84]. Mirai also connects to a remote node through a TCP handshake. Mirai sends its commands in PSH-ACK packets, which the node replies to with an ACK packet. The Mirai network also sends upkeep packets, once a minute, to maintain connections.

This means that, in the scenario where both systems use the smallest number of packets possible, Mirai will send five packets and ANTS uses only four. ANTS can further reduce this number if switched to using the UDP protocol instead, where communications would require only a single packet.

The size of the packets being sent is important as well. For both systems, the TCP handshake packet contents will be largely the same. For Mirai to execute a DDoS attack, it would send this example command in a packet:

.UDP 40.81.59.133 30142 20 32 0 10

This would start a DDoS towards the targeted IP at the given port. In order to perform a similar attack with ANTS, this command could be sent (hexadecimal values are represented by a "\x" with the next two characters being the hex encoded value):

1234567890123456\x02rhost\x0240.81.59.133rport\x0130142dur\x0120

Which is a command that is 54 bytes long. While the ANTS system requires slightly bigger packets, it has the ability to execute larger and more complex attacks than a system such as Mirai, and thus, needs the ability to identify a selection between multiple attack types.

Installation size is another metric that this type of software can be measured by. A design goal of ANTS was to keep its install binary as small as possible while retaining broad functionality. The current ANTS installation is 543 KB, which is quite small and easily portable. However, plans exist to expand this with additional functionality (such as more advanced command parsing features), so this file size is expected to grow. For purposes of comparison, the botnet known as Waldac once had almost 100,000 installed nodes with an install file size of 860 KB [86]. Although ANTS' binary size will continue to increase as functionality is added in the future, it is demonstrably more capable in terms of the attacks that it can run with a file size that is approximately two-thirds of the Waldac example. The ANTS file size is unlikely to grow large enough that it will create installation issues.

6. Conclusions and Future Work

The proposed system presents an opportunity for enhanced modern penetration testing through the use of a tool that is able to readily imitate complex human or automated attacks. It is poised to enhance the speed and thoroughness of penetration tests, aiding system administrators in enhancing network security. The system also presents the capability to rapidly re-check networks to ensure that vulnerabilities are resolved after corrections are made. Issues can also be tested for on a recurring basis, to catch issues that may arise as users and administrators settle back into older habits after the primary test is completed.

The system was designed to be able to evolve along with the attacks it imitates. It has been designed with a modular architecture to allow easy ongoing development, add-on creation, and enhancement. This is designed to allow the system to remain relevant and useful far into the future.

It also provides a capability to allow inexperienced security testers to quickly set up attacks and start finding vulnerabilities that may not have been even thought to be potential risks. The system is able to execute complex attacks that imitate real-world vulnerabilities that have been previously exploited, allowing testers to make sure that well-known vulnerabilities are not present while being able to focus on developing new

test-specific attacks, as needed. A potential community repository of attacks could be developed and shared, reducing individual tester's work needed to respond to new threats.

The overall structure of the system allows nodes to be created and destroyed without impacting the safety of the control node. This allows attack nodes to be deployed on an ongoing basis, allowing testers to scan a larger amount of potentially vulnerable systems and to work around and test network defensive mechanisms. It also limits the impact of losing nodes if they are detected or systems are taken offline.

This paper has demonstrated the efficacy of the distributed attack deployment capability for the proposed system. In Section 5.1, tests were presented demonstrating its capability to successfully launch three different attack types, which all had their intended effect. Section 5.4 compared the data utilization of the system's proposed communications format to the Mirai botnet, showing that it can operate with only 80% of the packets in TCP mode and only 20% of the required packets in UDP mode. The command packet for the proposed system was shown to be larger than the Mirai botnet (at 54 bytes); however, this is still a very small level of data on modern networks. Additionally, the file size of the proposed system was shown to be less than 65% of the Waldoc botnet, which successfully had nearly 100,000 nodes. Limitations of the system were also discussed in Section 5.2.

In addition to the potential future enhancements described in Section 5.2, two larger areas of focus also exist for future work. First, additional tools need to be developed to make the process of preparing attacks easier. A file parser is planned to take an attack file and identify the information that is necessary to run an attack. A Lua minifier will also be used to reduce attack binary size by removing whitespace, renaming variables, and other optimizations. This will make the network footprint of commands smaller and faster and less detectible. These two tools will be designed to work together.

A second potential future enhancement is to expand the scope of influence of attack nodes beyond the single computer that they are housed on. The ability for an attack node to implement command-node behavior on its target system and search for more vulnerabilities within a network than are initially apparent from the central command node could help identify potential weaknesses that would otherwise be missed due to complexity. Deploying command node functionality to a system using an attack node will allow that node to develop and execute secondary and supporting attacks, as needed for testing performance. The combination of allowing attack nodes to install command-node features on a target system and the ability to communicate back to the central command system, as discussed previously, will allow the system to provide more robust and potentially more efficient tests.

Finally, the demonstration of the distributed attack deployment capability, described herein, in conjunction with the broader system (which was depicted in Figure 1), will be a key area of future work.

Author Contributions: Conceptualization, J.H., J.M., N.R. and J.S.; methodology, J.H., J.M., N.R. and J.S.; software, J.H., J.M. and N.R.; validation, J.H.; data curation, N.R.; writing—original draft preparation, J.H., J.M. and N.R.; writing—review and editing, J.H., J.M., N.R. and J.S.; visualization, J.H. and N.R.; supervision, J.S.; project administration, J.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: No dataset was analyzed in this article. The experimentation performed and its results are described herein.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Straub, J. Blackboard-based electronic warfare system. In Proceedings of the ACM Conference on Computer and Communications Security, Denver, CO, USA, 12–16 October 2015.
2. Hasan, S.; Ghafouri, A.; Dubey, A.; Karsai, G.; Koutsoukos, X. Vulnerability analysis of power systems based on cyber-attack and defense models. In Proceedings of the 2018 IEEE Power Energy Society Innovative Smart Grid Technologies Conference, ISGT 2018, Washington, DC, USA, 19–22 February 2018; pp. 1–5. [CrossRef]
3. Eling, M.; Wirfs, J. What are the actual costs of cyber risk events? *Eur. J. Oper. Res.* **2019**, *272*, 1109–1119. [CrossRef]
4. Mateski, M.; Trevino, C.M.; Veitch, C.K.; Michalski, J.; Harris, J.M.; Maruoka, S.; Frye, J. *Cyber Threat Metrics*; Sandia National Laboratories: Albuquerque, NM, USA, 2012.
5. Mavroeidis, V.; Hohimer, R.; Casey, T.; Jesang, A. Threat Actor Type Inference and Characterization within Cyber Threat Intelligence. In Proceedings of the International Conference on Cyber Conflict, CYCON 2021, Tallinn, Estonia, 25–28 May 2021; pp. 327–352. [CrossRef]
6. King, Z.M.; Henshel, D.S.; Flora, L.; Cains, M.G.; Hoffman, B.; Sample, C. Characterizing and measuring maliciousness for cybersecurity risk assessment. *Front. Psychol.* **2018**, *9*, 1–19. [CrossRef] [PubMed]
7. Zhao, J.; Shao, M.; Wang, H.; Yu, X.; Li, B.; Liu, X. Cyber threat prediction using dynamic heterogeneous graph learning. *Knowl.-Based Syst.* **2022**, *240*, 108086. [CrossRef]
8. Gao, Y.; Li, X.; Peng, H.; Fang, B.; Yu, P.S. HinCTI: A Cyber Threat Intelligence Modeling and Identification System Based on Heterogeneous Information Network. *IEEE Trans. Knowl. Data Eng.* **2020**, *34*, 708–722. [CrossRef]
9. Sipper, J.A. Cyber Threat Intelligence and the Cyber Meta-Reality and Cyber Microbiome. In Proceedings of the International Conference on Cyber Security and Protection of Digital Services, Cyber Security 2020, Dublin, Ireland, 15–19 June 2020. [CrossRef]
10. Parmar, M.; Domingo, A. On the Use of Cyber Threat Intelligence (CTI) in Support of Developing the Commander's Understanding of the Adversary. In Proceedings of the IEEE Military Communications Conference MILCOM 2019, Norfolk, VA, USA, 12–14 November 2019. [CrossRef]
11. Ullah, S.; Shetty, S.; Nayak, A.; Hassanzadeh, A.; Hasan, K. Cyber Threat Analysis Based on Characterizing Adversarial Behavior for Energy Delivery System. In Proceedings of the Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering LNICST 2019, 305 LNICST, Orlando, VA, USA, 23–25 October 2019; pp. 146–160. [CrossRef]
12. Kesswani, N.; Kumar, S. Maintaining Cyber Security: Implications, Cost and Returns. In Proceedings of the SIGMIS-CPR'15, Newport Beach, CA, USA, 4–6 June 2015; pp. 161–164.
13. Gordon, L.A.; Loeb, M.P. The Economics of Information Security Investment. *ACM Trans. Inf. Syst. Secur.* **2002**, *5*, 438–457. [CrossRef]
14. Dreyer, P.; Jones, T.; Klima, K.; Oberholtzer, J.; Strong, A.; Welburn, J.W.; Winkelman, Z. *Estimating the Global Cost of Cyber Risk: Methodology and Examples*; RAND: Santa Monica, CA, USA, 2018.
15. Strom, B.E.; Battaglia, J.A.; Kemmerer, M.S.; Kupersanin, W.; Miller, D.P.; Wampler, C.; Whitley, S.M.; Wolf, R.D. *Finding Cyber Threats with ATT&CKTM-Based Analytics*; The MITRE Corporation: Bedford, MA, USA, 2017.
16. Yadav, T.; Rao, A.M. Technical Aspects of Cyber Kill Chain. In *Proceedings of the International Symposium on Security in Computing and Communication, Kochi, India, 10–13 August 2015*; Springer: Berlin/Heidelberg, Germany, 2015; pp. 438–452.
17. Khan, R.; McLaughlin, K.; Laverty, D.; Sezer, S. STRIDE-based threat modeling for cyber-physical systems. In *Proceedings of the 2017 IEEE PES Innovative Smart Grid Technologies Conference Europe, ISGT-Europe 2017, Torino, Italy, 26–29 September 2017*; Institute of Electrical and Electronics Engineers Inc.: Piscataway, NJ, USA, 2017; pp. 1–6.
18. Bhuiyan, T.H.; Nandi, A.K.; Medal, H.; Halappanavar, M. Minimizing expected maximum risk from cyber-Attacks with probabilistic attack success. In Proceedings of the 2016 IEEE Symp. Technol. Homel. Secur. HST 2016, Waltham, MA, USA, 10–11 May 2016. [CrossRef]
19. Lallie, H.S.; Debattista, K.; Bal, J. A review of attack graph and attack tree visual syntax in cyber security. *Comput. Sci. Rev.* **2020**, *35*, 100219. [CrossRef]
20. Nandi, A.K.; Medal, H.R.; Vadlamani, S. Interdicting attack graphs to protect organizations from cyber attacks: A bi-level defender–attacker model. *Comput. Oper. Res.* **2016**, *75*, 118–131. [CrossRef]
21. Straub, J. Modeling Attack, Defense and Threat Trees and the Cyber Kill Chain, ATTCK and STRIDE Frameworks as Blackboard Architecture Networks. In Proceedings of the 2020 IEEE International Conference on Smart Cloud, Washington, DC, USA, 6–8 November 2020; Institute of Electrical and Electronics Engineers Inc.: Washington, DC, USA, 2020; pp. 148–153.
22. Gu, G.; Zhang, J.; Lee, W. BotSniffer: Detecting Botnet Command and Control Channels in Network Traffic. In Proceedings of the 15th Annual Network and Distributed System Security Symposium, San Diego, CA, USA, 10–13 February 2008.
23. Gardiner, J.; Cova, M.; Nagaraja, S. Command & Control: Understanding, Denying and Detecting—A review of malware C2 techniques, detection and defences. *arXiv* **2014**, arXiv:1408.11362014.
24. Fogla, P.; Sharif, M.; Perdisci, R.; Kolesnikov, O.; Lee, W. Polymorphic Blending Attacks. In Proceedings of the Security '06: 15th USENIX Security Symposium, Vancouver, BC, Canada, 31 July–4 August 2006; USENIX Association: Berkeley, CA, USA, 2006; pp. 241–256.
25. Dittrich, D.; Dietrich, S. Command and Control Structures in Malware. *Login* **2007**, *32*, 8–17.
26. Cisco Systems, I. Cisco IOS NetFlow. Available online: https://www.cisco.com/c/en/us/products/ios-nx-os-software/ios-netflow/index.html (accessed on 26 January 2022).

27. CrowdStrike What Is Lateral Movement. Available online: https://www.crowdstrike.com/cybersecurity-101/lateral-movement/ (accessed on 28 January 2022).
28. Fawaz, A.; Bohara, A.; Cheh, C.; Sanders, W.H. Lateral Movement Detection Using Distributed Data Fusion. In Proceedings of the IEEE Symposium, Reliable Distributed Systems, Budapest, Hungary, 26–29 September 2016; pp. 21–30. [CrossRef]
29. Hacks, S.; Butun, I.; Lagerström, R.; Buhaiu, A.; Georgiadou, A.; Michalitsi -Psarrou, A. Integrating Security Behavior into Attack Simulations. In Proceedings of the ARES 2021 Conference, Vienna, Austria, 17–20 August 2021; p. 13.
30. Wotawa, F. On the automation of security testing. In Proceedings of the 2016 International Conference on Software Security and Assurance, ICSSA 2016, Sankt Pölten, Austria, 24–25 August 2016; pp. 11–16. [CrossRef]
31. Thompson, H.H. Why security testing is hard. *IEEE Secur. Priv.* **2003**, *1*, 83–86. [CrossRef]
32. Guo, F.; Yu, Y.; Chiueh, T.C. Automated and safe vulnerability assessment. In Proceedings of the Annual Computer Security Applications Conference ACSAC, Los Angeles, CA, USA, 7–11 December 2005; Volume 2005, pp. 150–159. [CrossRef]
33. Mohammad, S.M.; Surya, L. Security Automation in Information Technology. *Int. J. Creat. Res. Thoughts* **2018**, *6*, 901–905.
34. Metheny, M. Continuous monitoring through security automation. *Fed. Cloud Comput.* **2017**, 453–472. [CrossRef]
35. Shah, M.P. Comparative Analysis of the Automated Penetration Testing Tools. Ph.D. Theis, National College of Ireland, Dublin, Ireland, 2020.
36. Bhardwaj, A.; Shah, S.B.H.; Shankar, A.; Alazab, M.; Kumar, M.; Gadekallu, T.R. Penetration testing framework for smart contract Blockchain. *Peer-to-Peer Netw. Appl.* **2021**, *14*, 2635–2650. [CrossRef]
37. Casola, V.; de Benedictis, A.; Rak, M.; Villano, U. A methodology for automated penetration testing of cloud applications. *Int. J. Grid Util. Comput.* **2020**, *11*, 267–277. [CrossRef]
38. Casola, V.; de Benedictis, A.; Rak, M.; Villano, U. Towards automated penetration testing for cloud applications. In Proceedings of the 2018 IEEE 27th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises WETICE 2018, Paris, France, 27–29 June 2018; pp. 30–35. [CrossRef]
39. Yadav, G.; Allakany, A.; Kumar, V.; Paul, K.; Okamura, K. Penetration Testing Framework for IoT. In Proceedings of the 2019 8th International Congress on Advanced Applied Informatics, IIAI-AAI 2019, Toyama, Japan, 7–11 July 2019; pp. 477–482. [CrossRef]
40. Kadam, S.P.; Mahajan, B.; Patanwala, M.; Sanas, P.; Vidyarthi, S. Automated Wi-Fi penetration testing. In Proceedings of the 2016 International Conference on Electrical, Electronics, and Optimization Techniques, ICEEOT 2016, Chennai, India, 3–5 March 2016; pp. 1092–1096. [CrossRef]
41. Falkenberg, A.; Mainka, C.; Somorovsky, J.; Schwenk, J. A new approach towards DoS penetration testing on web services. In Proceedings of the IEEE 20th International Conference on Web Services ICWS 2013, Santa Clara, CA, USA, 28 June–3 July 2013; pp. 491–498. [CrossRef]
42. Antunes, N.; Vieira, M. Penetration testing for web services. *Computer* **2014**, *47*, 30–36. [CrossRef]
43. Mainka, C.; Somorovsky, J.; Schwenk, J. Penetration testing tool for web services security. In Proceedings of the 2012 IEEE 8th World Congress on Services, Honolulu, HI, USA, 24–29 June 2012; pp. 163–170. [CrossRef]
44. Singh, N.; Meherhomji, V.; Chandavarkar, B.R. Automated versus Manual Approach of Web Application Penetration Testing. In Proceedings of the 2020 11th International Conference on Computing Communication and Networking Technologies ICCCNT 2020, Kharagpur, India, 1–3 July 2020. [CrossRef]
45. Shah, S.; Mehtre, B.M. An automated approach to vulnerability assessment and penetration testing using net-nirikshak 1.0. In Proceedings of the 2014 IEEE International Conference on Advanced Communications, Control and Computing Technologies ICACCCT, Ramanathapuram, India, 8–10 May 2014; Volume 2015, pp. 707–712. [CrossRef]
46. Almubairik, N.A.; Wills, G. Automated penetration testing based on a threat model. In Proceedings of the 2016 11th International Conference for Internet Technology and Secured Transactions ICITST, Barcelona, Spain, 5–7 December 2016; pp. 413–414. [CrossRef]
47. Stepanova, T.; Pechenkin, A.; Lavrova, D. Ontology-based big data approach to automated penetration testing of large-scale heterogeneous systems. In Proceedings of the 8th International Conference on Security of Information and Networks, Sochi, Russia, 8–10 September 2015. [CrossRef]
48. Halfond, W.G.J.; Choudhary, S.R.; Orso, A. Improving penetration testing through static and dynamic analysis. *Softw. Test. Verif. Reliab.* **2011**, *21*, 195–214. [CrossRef]
49. Luan, J.; Wang, J.; Xue, M. Automated Vulnerability Modeling and Verification for Penetration Testing Using Petri Nets. *Lect. Notes Comput. Sci.* **2016**, *10040*, 71–82. [CrossRef]
50. Alhassan, J.K.; Misra, S.; Umar, A.; Maskeliūnas, R.; Damaševičius, R.; Adewumi, A. A Fuzzy Classifier-Based Penetration Testing for Web Applications. *Adv. Intell. Syst. Comput.* **2018**, *721*, 95–104. [CrossRef]
51. Rak, M.; Salzillo, G.; Granata, D. ESSecA: An automated expert system for threat modelling and penetration testing for IoT ecosystems. *Comput. Electr. Eng.* **2022**, *99*, 107721. [CrossRef]
52. Greenwald, L.; Shanley, R. Automated planning for remote penetration testing. In Proceedings of the IEEE Military Communications Conference, Boston, MA, USA, 18–21 October 2009. [CrossRef]
53. Zhou, T.Y.; Zang, Y.C.; Zhu, J.H.; Wang, Q.X. NIG-AP: A new method for automated penetration testing. *Front. Inf. Technol. Electron. Eng.* **2019**, *20*, 1277–1288. [CrossRef]

54. Chowdhary, A.; Huang, D.; Mahendran, J.S.; Romo, D.; Deng, Y.; Sabur, A. Autonomous security analysis and penetration testing. In Proceedings of the 2020 16th International Conference on Mobility, Sensing and Networking, MSN 2020, Tokyo, Japan, 17–19 December 2020; pp. 508–515. [CrossRef]
55. Chu, G.; Lisitsa, A. Poster: Agent-based (BDI) modeling for automation of penetration testing. In Proceedings of the 2018 16th Annual Conference on Privacy, Security and Trust, PST 2018, Belfast, Ireland, 28–30 August 2018. [CrossRef]
56. Ghanem, M.C.; Chen, T.M. Reinforcement Learning for Intelligent Penetration Testing. In Proceedings of the 2nd World Conference on Smart Trends in Systems, Security and Sustainability, WorldS4, London, UK, 30–31 October 2018; pp. 90–95. [CrossRef]
57. Schwartz, J.; Kurniawati, H. Autonomous Penetration Testing using Reinforcement Learning. *arXiv* **2019**, arXiv:1905.05965.
58. Gangupantulu, R.; Cody, T.; Park, P.; Rahman, A.; Eisenbeiser, L.; Radke, D.; Clark, R. Using Cyber Terrain in Reinforcement Learning for Penetration Testing. *arXiv* **2021**, arXiv:2108.07124.
59. Ghanem, M.C.; Chen, T.M. Reinforcement Learning for Efficient Network Penetration Testing. *Information* **2020**, *11*, 6. [CrossRef]
60. Chaudhary, S.; O'Brien, A.; Xu, S. Automated Post-Breach Penetration Testing through Reinforcement Learning. In Proceedings of the 2020 IEEE Communications and Network Security CNS 2020, Avignon, France, 29 June–1 July 2020. [CrossRef]
61. Hu, Z.; Beuran, R.; Tan, Y. Automated Penetration Testing Using Deep Reinforcement Learning. In Proceedings of the 5th IEEE European Symposium on Security and Privacy Workshops (EuroS&PW 2020), Genoa, Italy, 7–11 September 2020; pp. 2–10. [CrossRef]
62. Tran, K.; Akella, A.; Standen, M.; Kim, J.; Bowman, D.; Richer, T.; Lin, C.-T. Deep hierarchical reinforcement agents for automated penetration testing. *arXiv* **2021**, arXiv:2109.06449.
63. Dai, Z.; Lv, L.; Liang, X.; Bo, Y. Network penetration testing scheme description language. In Proceedings of the 2011 International Conference on Computational and Information Sciences ICCIS 2011, Chengdu, China, 21–23 October 2011; pp. 804–808. [CrossRef]
64. Stefinko, Y.; Piskozub, A.; Banakh, R. Manual and automated penetration testing. Benefits and drawbacks. Modern tendency. In Proceedings of the 13th International Conference on Modern Problems of Radio Engineering, Telecommunications and Computer Science, TCSET 2016, Lviv, Ukraine, 23–26 April 2016; pp. 488–491. [CrossRef]
65. Hayes-Roth, B. A blackboard architecture for control. *Artif. Intell.* **1985**, *26*, 251–321. [CrossRef]
66. Erman, L.D.; Hayes-Roth, F.; Lesser, V.R.; Reddy, D.R. The Hearsay-II speech-understanding system: Integrating knowledge to resolve uncertainty. *ACM Comput. Surv.* **1980**, *12*, 213–253. [CrossRef]
67. Feigenbaum, E.A.; Buchanan, B.G.; Lederberg, J. *On Generality and Problem Solving: A Case Study Using the DENDRAL Program*; Stanford University Rep: Stanford, CA, USA, 1970.
68. Zwass, V. Expert System. Available online: https://www.britannica.com/technology/expert-system (accessed on 24 February 2021).
69. Lindsay, R.K.; Buchanan, B.G.; Feigenbaum, E.A.; Lederberg, J. DENDRAL: A case study of the first expert system for scientific hypothesis formation. *Artif. Intell.* **1993**, *61*, 209–261. [CrossRef]
70. Corkill, D.D. *Blackboard Systems*; AI Expert: Modena, Italy, 1991; Volume 6.
71. Dong, J.; Chen, S.; Jeng, J.-J. Event-based blackboard architecture for multi-agent systems. In Proceedings of the International Conference on Information Technology: Coding and Computing, ITCC 2005, Las Vegas, NV, USA, 4–6 April 2015; Volume 2, pp. 379–384.
72. Huang, M.-J.; Chiang, H.-K.; Wu, P.-F.; Hsieh, Y.-J. A multi-strategy machine learning student modeling for intelligent tutoring systems: Based on Blackboard approach. *Libr. Hi Tech* **2013**, *31*, 6. [CrossRef]
73. Brzykcy, G.; Martinek, J.; Meissner, A.; Skrzypczynski, P. Multi-agent blackboard architecture for a mobile robot. In Proceedings of the 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems, Maui, HI, USA, 29 October–3 November 2001; Volume 4, pp. 2369–2374.
74. Yang, Y.; Tian, Y.; Mei, H. Cooperative Q learning based on blackboard architecture. In Proceedings of the International Conference on Computational Intelligence and Security Workshops, Harbin, China, 15–19 December 2007; pp. 224–227.
75. Johnson, M.V., Jr.; Hayes-Roth, B. Integrating Diverse Reasoning Methods in the BBP Blackboard Control Architecture. In Proceedings of the AAAI-87 Conference, AAAI, Seattle, WA, USA, 13–17 July 1987; pp. 30–35.
76. De Campos, A.M.; Monteiro de Macedo, M.J. A blackboard architecture for perception planning in autonomous vehicles. In Proceedings of the 1992 International Conference on Industrial Electronics, Control, Instrumentation, and Automation, San Diego, CA, USA, 13 November 1992; pp. 826–831.
77. Straub, J. A modern Blackboard Architecture implementation with external command execution capability. *Softw. Impacts* **2022**, *11*, 100183. [CrossRef]
78. Juniper Research Business Losses to Cybercrime Data Breaches to Exceed $5 Trillion. Available online: https://www.juniperresearch.com/press/business-losses-cybercrime-data-breaches (accessed on 26 January 2022).
79. Zeadally, S.; Adi, E.; Baig, Z.; Khan, I.A. Harnessing artificial intelligence capabilities to improve cybersecurity. *IEEE Access* **2020**, *8*, 23817–23837. [CrossRef]
80. Wirkuttis, N.; Klein, H. Artificial Intelligence in Cybersecurity. *Cyber Intell. Secur.* **2017**, *1*, 103.
81. Rapid7 VSFTPD v2.3.4 Backdoor Command Execution. Available online: https://www.rapid7.com/db/modules/exploit/unix/ftp/vsftpd_234_backdoor/ (accessed on 20 February 2022).

82. Rapid7 UnrealIRCD 3.2.8.1 Backdoor Command Execution. Available online: https://www.rapid7.com/db/modules/exploit/unix/irc/unreal_ircd_3281_backdoor/ (accessed on 20 February 2022).
83. Kauppi, A.; Germain, B. Lua Lanes—Multithreading in Lua. Available online: https://lualanes.github.io/lanes/ (accessed on 28 January 2022).
84. Jovanovic, E.D.; Vuletic, P.V. Analysis and Characterization of IoT Malware Command and Control Communication. In Proceedings of the 27th Telecommunications Forum, TELFOR 2019, Belgrade, Serbia, 26–27 November 2019. [CrossRef]
85. Vogt, R.; Aycock, J.; Jacobson, M.J.J. Army of Botnets. In Proceedings of the Network and Distributed System Security Symposium, San Diego, CA, USA, 28 February–2 March 2007.
86. Calvet, J.; Davis, C.R.; Bureau, P.M. Malware authors don't learn, and that's good! In Proceedings of the 2009 4th International Conference Malicious Unwanted Software, MALWARE 2009, Montreal, QC, Canada, 13–14 October 2009; pp. 88–97. [CrossRef]