Log4shell: Redefining the Web Attack Surface

Douglas Everson, Long Cheng, Zhenkai Zhang
School of Computing, Clemson University, USA

***Abstract*—The log4shell vulnerability has been called one of the most significant cybersecurity vulnerabilities in recent history. For weeks after initial disclosure, companies around the globe scrambled to respond by patching their systems or by applying mitigating security measures to protect systems that could not be readily patched. There are many possible ways to detect if and where an organization is vulnerable to log4shell, each with advantages and disadvantages. Penetration testing in particular is one possible solution, though its results can be misleading if not interpreted in the proper context. Mitigation measures have varying degrees of success: Web Application Firewalls (WAFs) could be bypassed, whereas our analysis revealed that outbound network restrictions would have provided an effective protection given the rapidly evolving patch cycle. Ultimately, log4shell should change the way we look at web attack surfaces; doing so will ensure we can be better prepared for the next critical zero-day Remote Code Execution (RCE) vulnerability.**

***Index Terms*—Log4j; Web Attack Surface; Penetration Testing**

I. Introduction

Log4shell has been referred to as one of the most significant cybersecurity vulnerabilities in the modern age for several reasons. It received a Common Vulnerability Scoring System (CVSS) rating of 10 [1], the highest and most dangerous rating possible on that scale; it allowed Remote Code Execution (RCE) on a target; and it did not require user interaction for successful exploitation. The only requirement is that a specially-crafted string of characters is received by a system with outbound access to the Internet and then logged by that same system using an open-source Java library called log4j. Most importantly, millions of systems of all types use Java, and many of those use log4j for their logging—this is one reason why experts claim it may be years before the vulnerability is completely resolved [2].

Log4shell (and vulnerabilities like it) force us to redefine how we see the web attack surface. While traditional web RCE vulnerabilities allow us to directly run code on the system we are impacting, the log4shell vulnerability has a potentially much wider scope. Any application that logs the specially-crafted string can be exploited, and it doesn't matter how that string is delivered. It can be delivered directly to a web server that keeps logs of user-controlled input. It can be intercepted by a Web Application Firewall (WAF) and then logged. It can even be received in data imported from an unrelated third-party site. Systems thought to be well-protected or untouched are now potentially vulnerable.

In this paper we will make the following contributions:

- we introduce the log4j Java logging library and detail the log4shell vulnerability.
- we analyze the methods used to detect and mitigate the log4shell vulnerability in various systems.
- we discuss how log4shell should change the security professional's view of the web attack surface.

Section II provides a background of log4j and log4shell, Section III discusses detection methods and the role of penetration testing, Section IV provides the results of our preliminary analysis of log4shell mitigations, and Section V provides conclusions.

II. Log4shell Vulnerability

A. Log4j

Log4j is a free and open-source library implementing a logging framework [3]. There are a number of reasons, security and otherwise, that a developer should implement logging. Cheng *et al.* [4] provided an overview of logging platforms, including log4j, as an alternative to Java's built-in logging system. The authors explained the benefits of using a single logging solution and reviewed the different capabilities a developer can use to track significant events generated by their application.

B. Log4shell

In 2013, a log4j user requested that a feature be added to log4j that allowed the use of Java Naming and Directory Interface (JNDI) lookups [5]. 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 [6]. Respectively, these vulnerabilities are generally classified as RCE, Information Disclosure, or Denial of Service (DoS). The RCE vulnerability is the source of the

Workshop on Measurements, Attacks, and Defenses for the Web (MADWeb) 2022
28 April 2022, San Diego, CA, USA
ISBN 1-891562-78-9
https://dx.doi.org/10.14722/madweb.2022.23010
www.ndss-symposium.org

"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 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.

```java
private static final Logger logger = (Logger)
    LogManager.getLogger(Vulnerable.class);
...
BufferedReader reader = new BufferedReader(new
     InputStreamReader(System.in));
...
String data = reader.readLine();
System.out.println("Your data is: " + data);
logger.error(data);
...
```

Listing 1: Example Vulnerable Application

1) Remote Code Execution: RCE vulnerabilities allow an attacker to run their code on a victim machine. Biswas *et al.* [7] conducted a case study of web-based RCE vulnerabilities. The authors discussed different types of RCE vulnerabilities in web applications and provided demonstrations of how they might be exploited, to include tools commonly used by attackers. Bier *et al.* [8] studied RCE vulnerabilities in Android applications and Apache web server software to analyze the prevalence of RCE vulnerabilities as compared to others.

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 (not just the logged application) 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 resulted in RCE with no user interaction.

A recent Apache Struts vulnerability has many similarities to the log4shell RCE. In 2017, threat actors discovered and exploited CVE-2017-5638, a vulnerability in Apache Struts software [9]. This RCE vulnerability allowed an attacker to run malicious code on the application server. Struts is a web framework for Java applications, and by placing a specially-crafted string in the `Content-Type` header, an attacker could run arbitrary commands on the Struts application server. In one publicized use of this vulnerability, Personally Identifiable Information (PII) of about 143 million Equifax customers was exposed [10].

2) Information Disclosure: An Information Disclosure vulnerability allows an attacker to compel a server to reveal data it was not designed to reveal. A prominent and recent example of this class of vulnerability is Heartbleed, a flaw in OpenSSL through which a threat actor could send a crafted request to a server and convince it to send blocks of its heap memory just under 64KB in size. This memory could include private keys, passwords, or other sensitive data, depending on what was adjacent to the string in question on the heap [11].

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 we verified in a test network on Cloudlab [12]. 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.

3) Denial of Service: Distributed Denial of Service (DDoS) is a well-known attack that involves harnessing a large number of network nodes to fling packets at a target in the hopes of overwhelming it [13]. 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.

Researchers identified a DoS bug not fixed by the 2.16.0 patch [14]. In certain conditions, threat actors can create a recursive Lookup condition which crashes the application using log4j. The example the author provided was when log4j was directed to log an entry based on a Context Lookup of an attacker-controlled value. The attacker could simply make that value equal to the Context Lookup for that value, thus creating a recursive condition that would quickly crash the application.

C. 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 I: Log4shell Timeline

A timeline for the log4shell event can be found in Figure 1. The patch history for the log4shell vulnerability is shown in Table I [15] and can be summarized as follows:

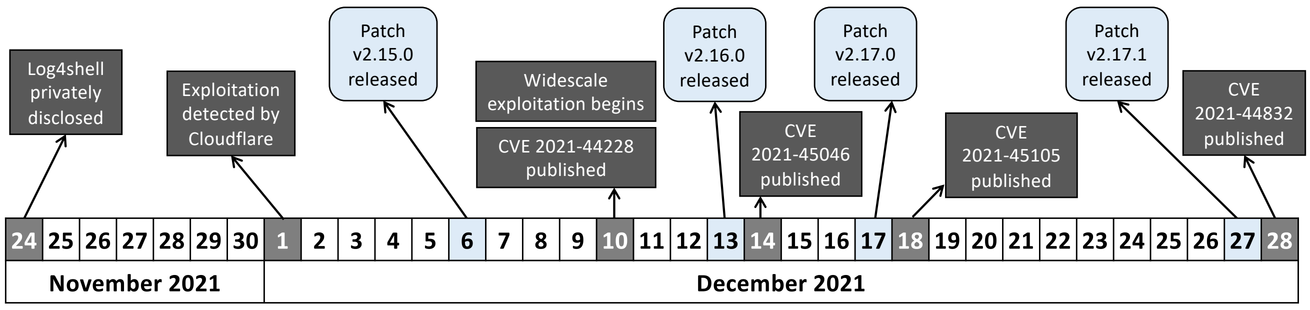

Fig. 1: Significant Log4shell Events

1) 2.15.0: This was the original patch introduced to fix log4shell. Released on 12/6/2021, it disabled lookups within message text by default, which was the only known attack vector at the time. For those unable to install the patch, a mitigating action was recommended to use a command-line switch which disabled the same lookup feature.

2) 2.16.0, 2.12.2: Once the log4shell vulnerability gained notoriety, security researchers were quick to find another attack vector, non-default Pattern Layouts using Context Lookups. This patch, released on 12/13/2021, disabled JNDI by default and removed support for message lookups. For those unable to install the patch, removing the JNDILookup class from the log4j-core JAR file was a sufficient mitigation. Note this patch cycle introduced a fix for Java 7 (2.12.2) as well as Java 8 and later (2.16.0).

3) 2.17.0, 2.12.3, 2.3.1: On 12/17/2021, Apache released another patch to address a DoS vulnerability caused by infinite recursion. The patch changed the code to limit recursion and limited JNDI to the Java protocol. This patch also introduced 2.3.1, a fix for Java 6.

4) 2.17.1, 2.12.4, 2.3.2: The patch released on 12/27/2021 caused a stir because it fixed another possible RCE vulnerability. However exploiting this RCE would require access to the server configuration—and anyone with that access already controlled the server anyway. As a result, this vulnerability was not deemed critical.

D. Log4shell Targets

1) 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 are an excellent example. When an HTTP request is made, headers are sent with the request to fulfill any number of functions. The Cookie header helps sites maintain state. The Host header tells the IP address receiving the request what host was entered in the browser URL bar. The User Agent header tells the server what type of browser, computer, and even operating system are being used to request the page so it can be rendered for optimal user experience. However, since these headers are in the user-controlled request, they can be manipulated by users to unexpected values that can impact servers [16]. It may be valuable for a developer to log the user agent or a cookie, or any number of other significant values communicated by headers. If these values are logged by a vulnerable version of log4j, they can be used to exploit the log4shell vulnerability and cause code to be executed on the server.

Post parameters are another potential attack vector, and with login pages being accessible to all, the username makes a great choice. If an application uses a vulnerable version of log4j to record login attempts, it is trivial for a threat actor to submit a "username" with an attack string that triggers the RCE vulnerability. While it may not be readily apparent, post parameters and HTTP headers can be manipulated with developer tools integrated into most browsers or with proxy/repeater tools like Burp Proxy Suite [17].

2) 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. For example, imagine a network Intrusion Detection System (IDS) written in Java used a vulnerable version of log4j and was configured to log suspicious events, with details from the network packet. An attacker only needs to create a suspicious event on the network and include the log4shell attack somewhere in the packet. The IDS would log the packet and immediately execute the embedded JNDI request, potentially downloading the vulnerable code and executing it. The damage done by compromising a device that sees all network traffic could be substantial.

3) Backend Servers: Much as a Structured Query Language (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.

E. Web Application Firewalls

When faced with the task of patching so many systems in a short time, organizations frequently turn to WAFs [18]. 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.

There is also the possibility that an encoded value could be decoded and then logged. One example of this is Base64, an encoding mechanism that turns any data into a string containing any of 64 possible characters. Base64 can be used to encode binary data into text that can be safely sent via a text-only protocol. However, it can also be used to hide a log4shell attack. If a log4shell attack can be encoded in a Base64 string, it can be extremely difficult for a WAF to detect it. This is compounded because the log4shell attack works even if padded on either or both sides with any other data. Since the attack can also be padded internally with nesting, the Base64-encoded values likely yield too many possible strings to match with a WAF. Pen testers and threat actors alike can examine valid requests containing Base64-encoded values, and then embed an encoded attack in the parameters or headers containing those values to see if the victim server reaches out to the IP address they specified in the attack. If it does, they know the server is vulnerable.

III. Susceptibility Analysis

Susceptibility can be defined as how easily one can be harmed by something, or the inability to resist something [19]. 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. Another variable to be considered is the threat model.

A. Threat Model Impact on Susceptibility

The threat model is the collection of all malicious activities that an attacker can perform against an entity [20]. It follows from this definition that 1) the threat model varies from organization to organization, and 2) the public disclosure of a new vulnerability could add potential activities to a threat model. In the case of a vulnerability with an extremely broad impact like log4shell, the range of impacted threat models is extensive.

The threat model of an organization being targeted by an Advanced Persistent Threat (APT) is far different than the threat model of an arbitrary organization with data of financial value, and that organization’s threat model is different again than that of an organization with no data of significant value. Security researchers reported spraying of attacks shortly after log4shell went public [21], and these attacks are in all three threat models. However, organizations with something of value must be ready for more targeted attacks, since criminals will spend more time trying to identify the log4shell attack surface and developing tailored attacks to bypass WAFs or fit custom code that will be overlooked by automated scanners.

B. 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 [22]. However, it is problematic to check 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.

Penetration testing is a vulnerability hunt from a threat’s perspective [23]. A major advantage of pen testing is that a tester can create a proof of concept that proves a vulnerability exists. A disadvantage is that the tester only finds what they have the knowledge, skill, ability, and access to find.

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

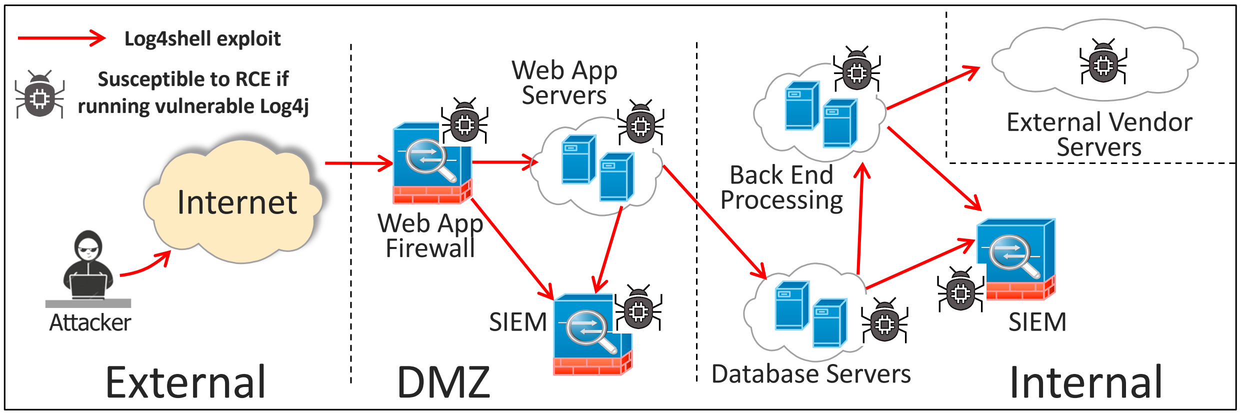

Fig. 2: Example of Log4shell-susceptible Web Application

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.

C. Limitations of Penetration Testing

1) 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.

2) Automated Log4shell Scanners: The Cybersecurity and Infrastructure Security Agency (CISA) published an automated scanner designed to hunt for exposed services containing the log4shell vulnerability [24]. 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 [25].

3) Watering Hole Attacks: Pen testers and other security researchers must be alert for watering hole attacks. A watering hole attack occurs when a threat actor finds a site likely to be visited by their target and “poisons” the site with malicious code in the hopes that it will be used by the target organization [26]. In the hours and days immediately following disclosure of a critical zero-day like log4shell, security testers and other security personnel at companies around the world are under pressure to answer questions from their executives and managers. Tools to reproduce or test for the vulnerability are in high-demand, and it would be very easy for someone to release malware under the guise of a helpful tool. Once downloaded and run on an internal network, the “tool” could compromise the very systems the tester meant to protect. One example of this was a seemingly humorous attempt to cause a “fork bomb” DoS on a system by tricking someone into running a bash command containing the code for the fork bomb, which looks similar to a log4shell attack [27].

Using code from a reputable, verifiable source is a good practice to follow. Determining the line between good code and bad code can be difficult, especially under tight timelines. A malicious actor could purposely publish a malicious tool, or a well-meaning programmer could inadvertently create buggy code that damages systems [28]. Using a list of verified developers can reduce this risk. Even when using a verified developer, carefully reviewing the source code and then compiling the tool on a trusted system is advised. Several tools were released following the log4shell vulnerability’s disclosure. One tool was published by Veracode [29], a reputable company specializing in security services like static and dynamic code analysis. The Veracode tool, published well before the log4shell disclosure, provided a rogue LDAP server to help find JNDI injection vulnerabilities in general. The code was published on their Github account, so this could be reasonably trusted, and Veracode made the source code available as well. Other tools were published by less-well-known researchers, so reviewing the source code and compiling the application oneself becomes even more important. In the end, this is a risk decision—and a well-defined policy regarding the use of open source test tools, even when responding to a critical security incident, will help pen testers and others avoid trouble.

D. Internal Systems and Susceptibility

Risk rating frameworks like CVSS rate vulnerabilities as much more severe if they can be exploited from the Internet [30]. 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, we have seen in the past how command injection attacks allow an attacker to run code on web servers, even when command interpreter ports 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 Internet of Things (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 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.

An example of a vulnerability that allowed an attack on a transparent system was CVE-2016-2208 [31], a flaw affecting all Symantec Antivirus products that allowed root-level RCE on an affected system [32]. The vulnerability was triggered when the antivirus product scanned a file. Normally there's no way for an attacker to interact with an antivirus scanning engine directly, but by interacting with any other software running on the server that allows file uploads or creation, an attacker could indirectly exploit the attack and subsequently gain direct root access to any and all systems that scanned the file for viruses. Similarly, a web application could pass a log4shell attack string to a vulnerable system for processing which was being monitored by a SIEM using a vulnerable Splunk add-on [33], subjecting the SIEM to RCE.

IV. Mitigation Effectiveness

A. 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) [19]. 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 [34]. 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.

B. 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 [35].

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`

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 II: Experimental Results

`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.

C. *Proof of Concept*

We used Cloudlab [12] to create a test environment in which to run the vulnerable application in Listing 1. Our 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 [36]), 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.

For each attack, we tested a baseline case 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. Our results can be found in Table II. 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, we were able 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 our vulnerable application used the most basic use case of log4j. None of the mitigations had a negative impact on the application's functionality.

V. Conclusion

Traditionally, a web attack surface has been the listening ports exposed to a threat [23]. 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, we must acknowledge 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. But perhaps the real lesson here is that widely-used open source libraries like log4j can serve as a conduit, connecting our most protected, sensitive systems to our most exposed—and our risk assessment processes must be prepared for the inevitable public disclosure of the next critical open-source library vulnerability.

Acknowledgment

We thank the anonymous reviewers for their valuable comments. This work is supported in part by National Science Foundation (NSF) under the Grant No. 2114920 and 2031002.

References

[1] National Institute of Standards and Technology. NVD - CVE-2021-44228, 2021.

[2] Phil Muncaster. Experts: Log4j Bug Could Be Exploited for "Years", December 2021.

[3] Apache Software Foundation. Log4j – Changes, December 2021.

[4] Fu Cheng. The platform logging API and service. In *Exploring java 9*, pages 81–86. Springer, 2018.

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

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

[7] Saikat Biswas, MMHK Sajal, T Afrin, T Bhuiyan, and MM Hassan. A study on remote code execution vulnerability in web applications. In *International conference on cyber security and computer science (ICONCS 2018)*, 2018.

[8] Stephen Bier, Brian Fajardo, Obinna Ezeadum, German Guzman, Kazi Zakia Sultana, and Vaibhav Anu. Mitigating remote code execution vulnerabilities: A study on tomcat and android security updates. In *2021 IEEE international IOT, electronics and mechatronics conference (IEMTRONICS)*, pages 1–6, 2021. tex.organization: IEEE.

[9] Jeff Luszcz. Apache struts 2: how technical and development gaps caused the equifax breach. *Network Security*, 2018(1):5–8, 2018. Publisher: Elsevier.

[10] Vex Woo. Apache Struts 2.3.5 < 2.3.31 / 2.5 < 2.5.10 - Remote Code Execution, March 2017.

[11] JK Harris. HEARTBLEED: A CASE STUDY. *Issues in Information Systems*, 19(2), 2018.

[12] Dmitry Duplyakin, Robert Ricci, Aleksander Maricq, Gary Wong, Jonathon Duerig, Eric Eide, Leigh Stoller, Mike Hibler, David Johnson, Kirk Webb, Aditya Akella, Kuangching Wang, Glenn Ricart, Larry Landweber, Chip Elliott, Michael Zink, Emmanuel Cecchet, Snigdhaswin Kar, and Prabodh Mishra. The Design and Operation of CloudLab. In *Proceedings of the USENIX Annual Technical Conference (ATC)*, pages 1–14, July 2019.

[13] 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.

[14] Guy Lederfein. Zero Day Initiative — CVE-2021-45105: Denial of Service via Uncontrolled Recursion in Log4j StrSubstitutor, 2021.

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

[16] Sanjib Sinha. Header injection and URL redirection. In *Bug bounty hunting for web security*, pages 79–96. Springer, 2019.

[17] Portswigger. Burp Suite Professional, 2021.

[18] 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.

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

[20] Norah Ahmed Almubairik and Gary Wills. Automated penetration testing based on a threat model. In *2016 11th international conference for internet technology and secured transactions (ICITST)*, pages 413–414, 2016. tex.organization: IEEE.

[21] Dan Goodin. The Log4Shell 0-day, four days on: What is it, and how bad is it really? | Ars Technica. Type: misc.

[22] 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.

[23] 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.

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

[25] Portswigger. Using burp intruder - PortSwigger, March 2022.

[26] Braeden Bowen, Jeremy Eraybar, Iyanuoluwa Odebode, Douglas D Hodson, and Michael R Grimaila. The new office threat: A simulation of watering hole cyberattacks. In *Advances in parallel & distributed processing, and applications*, pages 35–42. Springer, 2021.

[27] egyp7. This is why it's important to understand how your shell works, folks., December 2021.

[28] Paul B De Laat. How can contributors to open-source communities be trusted? On the assumption, inference, and substitution of trust. *Ethics and information technology*, 12(4):327–341, 2010. Publisher: Springer.

[29] Veracode. veracode-research/rogue-jndi, January 2022. original-date: 2019-11-13T18:11:16Z.

[30] 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.

[31] National Institute of Standards and Technology. NVD - CVE-2016-2208, 2016.

[32] Tavis Ormandy. Project Zero: How to Compromise the Enterprise Endpoint, June 2016.

[33] Splunk Security Advisory for Apache Log4j (CVE-2021-44228 and CVE-2021-45046), 2021.

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

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

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