Remote Code Execution Attacks Vulnerability

Shells blooming in Spring

By Mohith Kalyan and Anurag ShandilyaMay 5, 2022

Yet another vulnerability has been reported in the Java platform, this time in the popular Java Spring framework, just a few months after Log4shell, which was in the Java Logging Framework.

Two potential RCE vulnerabilities that are exploited in the wild are :

1. CVE-2022-22965 (a.k.a Springshell), found in the Spring Core framework
2. CVE-2022-22963, found in Spring Cloud

CVE-2022-22965 is an unauthorised Remote Code Execution (RCE) vulnerability found in the Java Spring Framework. The Spring team has released a patch for both vulnerabilities.

The SpringShell vulnerability allows attackers to target applications running with Java Spring Framework. This vulnerability is triggered when parameters that are passed to the Spring application via HTTP GET/POST request contain a class in them which results in those classes getting loaded allowing arbitrary write on the server.

Let’s discuss and dive deep into the analysis of CVE-2022—22965 vulnerability and its impact on organisations.

Note: The mere presence of the Spring framework doesn’t make the application vulnerable. So, it’s not easy to track all the vulnerable products that are affected by this vulnerability. But, here are some that are obvious.

Vulnerable Products

- Apache tomcat <10.0.20
- Apache tomcat <9.0.62
- Apache tomcat <8.5.78
- VMware

To exploit CVE-2022-22965 (Springshell), the victim’s environment must have

- Spring Framework versions 5.3.0 to 5.3.17 or 5.2.0 to 5.2.19, and older versions
- Apache Tomcat as the Servlet container
- JDK 9 or higher
- Packaged as a traditional WAR (Web application archive, in contrast to JAR)
- spring-webmvc or spring-webflux dependency

Note : Embedded Tomcat versions are not vulnerable to this flaw.

And to exploit CVE-2022-22963, the victim's environment must have

- Spring cloud function <=3.1.6 or
- Spring cloud function <=3.2.2

The impact of CVE-2022-22965 would be huge compared to CVE-2022-22963, as SpringShell vulnerability affects the Java core framework (spring) and every organisation/application that uses this vulnerable module is susceptible to this vulnerability. On the other hand, the CVE-2022-22963's scope is limited, as the cloud function module is used primarily in cloud projects and the majority of the spring cloud platforms have patched this vulnerability.

Analysis

This vulnerability lets a local class load in the HTTP parameter, thus leveraging the internal class directly and leading to an RCE. In other words, Injecting malicious values into unsafe properties of Java class via HTTP parameters. The vulnerability leverages DataBinder class and is exploitable by populating an object from request parameters (binding HTTP parameters in this case). You can find data binding rules by spring framework here.

Here is a sample request for an explanation

curl 'http://someip:8030/helloworld/greeting?class.module.classLoader.resources.context.parent.pipeline.first.pattern=shell'

This is a part request of the exploitation. As you can see in the request, the class object is directly being accessed via HTTP requests. The classLoader is being accessed via class.module.classloader parameter.

To achieve Remote Code Execution, we pass as parameters, a location and a template to the unfiltered Java class via a few properties, and that causes the object to create a new file for storing logs in our specified format.

Having seen the above example, here's what happens under the hood to achieve RCE :

A malicious JSP web shell is being created here in our example, and for that, the below-mentioned class properties will be set accordingly as a result of parameter binding.

class.module.classLoader.resources.context.parent.pipeline.first.pattern = <PATTERN_REDACTED>"

This is the actual payload. The pattern property is usually set for a logging pattern. But, we injected our malicious payload into that by keeping the pattern as is.

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

We have set the suffix parameter to .jsp, this is usually .txt or .log as log files are in text format in general.

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

The directory of the web shell is set by this parameter. Usually this will be the location where logs will be stored.

class.module.classLoader.resources.context.parent.pipeline.first.prefix = "shell"

Name of the log file we want to create, here we are creating a web shell with the name "shell"

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

The log file usually starts with a date and time for obvious reasons, and this parameter is used to set that format. This parameter is not used in the exploit.

You can find the exploitation and setting of these exposed class properties on the wire as seen in Figure 1.

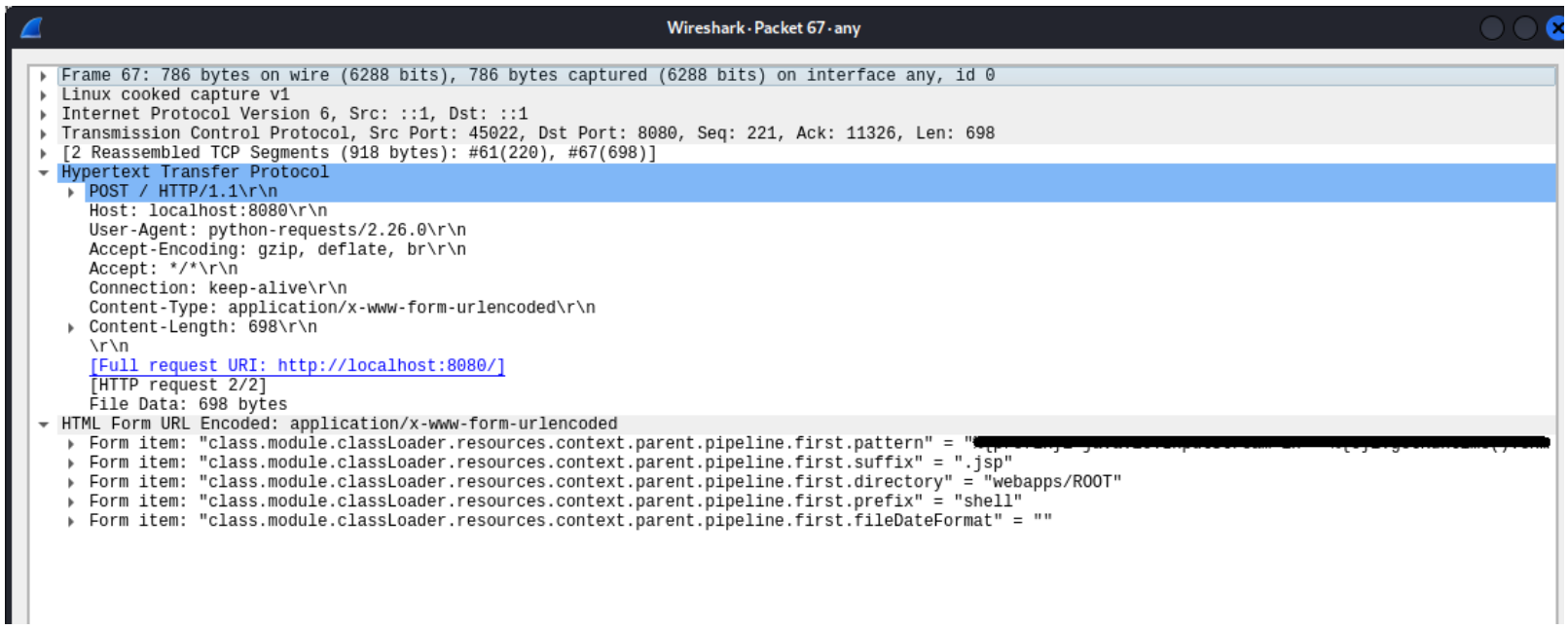

Figure 1: Wireshark capture of the exploitation request sent

The log file is created with the name shell.jsp and the location webapps/root as we mentioned in the parameter. We smuggled in some Java code via pattern property to execute it on the system within the typical log format. If we dissect the payload, we can see the input stream to take in the input command and pass the command to the exec function as shown below.

“Runtime.getRuntime().exec(request.getParameter(“cmd”))”

and throws the output on standard output.

To access the created webshell, we request /shell.jsp and use the ‘cmd’ parameter to pass-along the commands that we want. An example is shown in a subsequent image.

Mirai botnet

Some recent events brought to light the real-world exploitation of the Springshell vulnerability. It was observed to be exploited and Mirai malware was deployed on infected servers as post-exploitation.

Debugging

We deployed the PoC available here to analyse and debug the vulnerability. We deployed the WAR in Apache Tomcat 9.0.58 in Kali Linux and used IntelliJ to debug Tomcat.

AccessLogValve is an extended Class that is used for generating Web Server access logs. As per this information available online, it refers to the currentLogFile, Prefix, Suffix and Pattern fields to write logs.

Figure 2 shows the fields set under normal traffic flow and Figure 3 shows fields under exploitation.

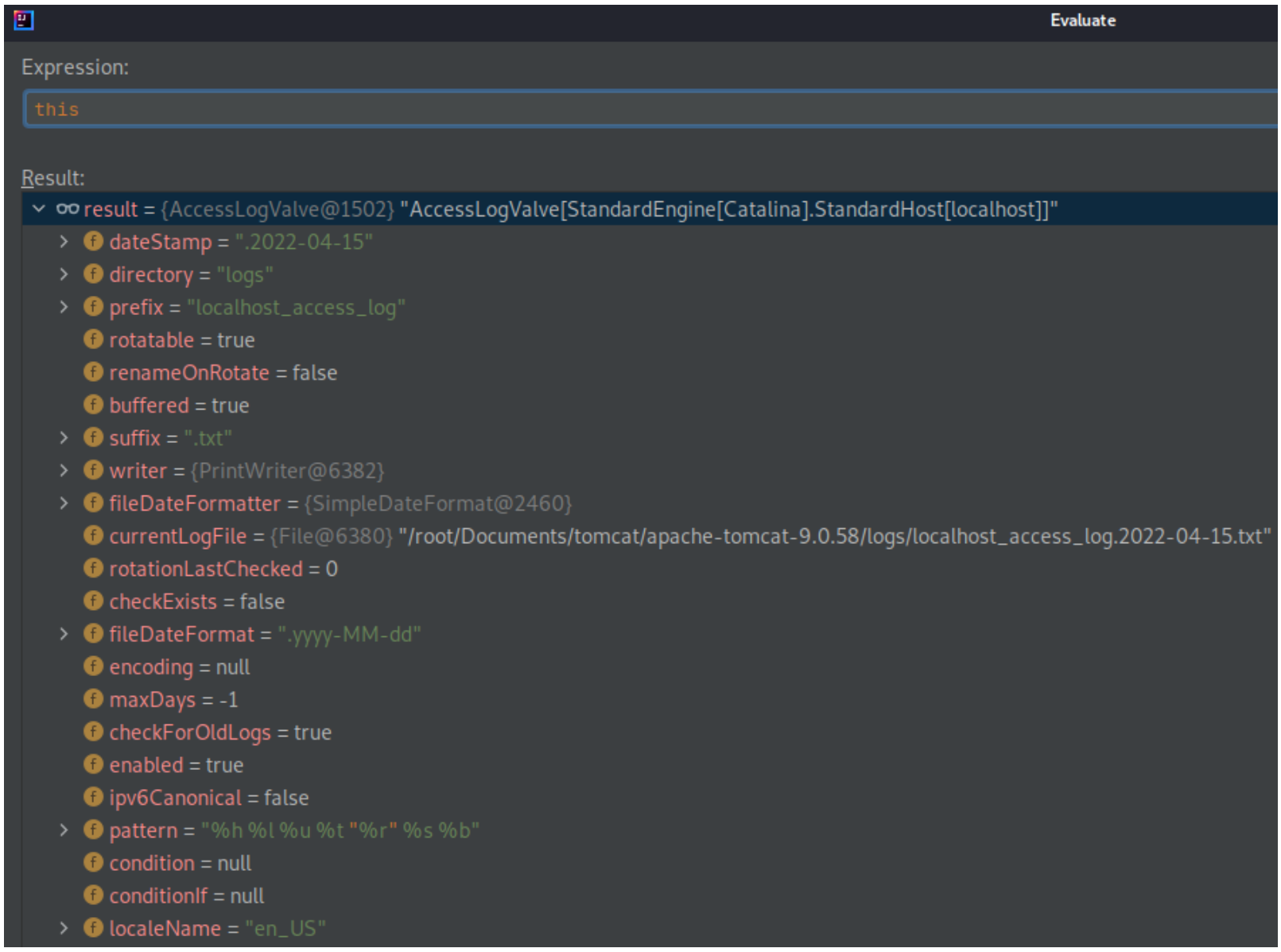

Figure 2: Normal field values

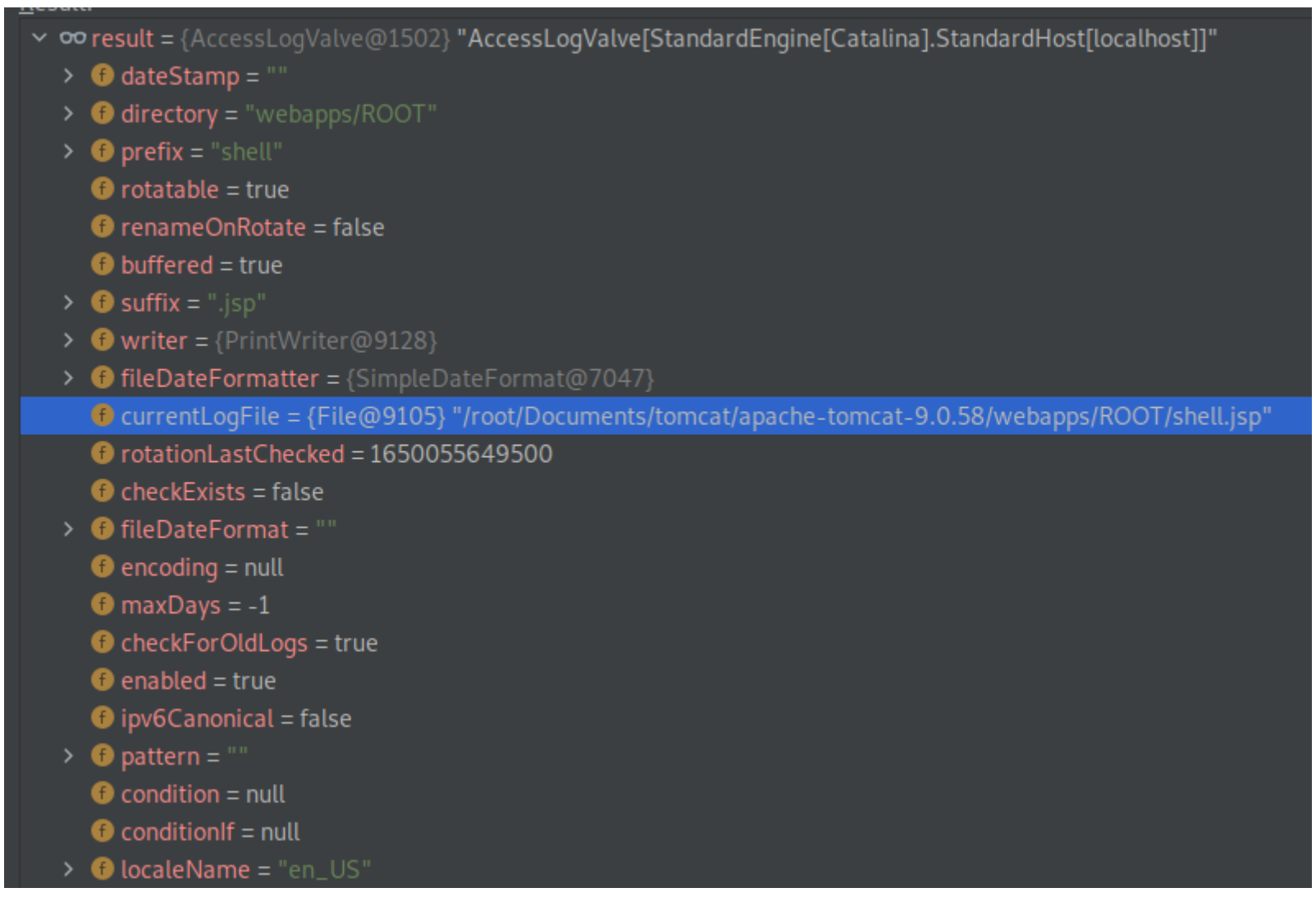

Figure 3: Field values during exploitation

In the figure above, as a result of the CVE-2022-22695, a specific pattern will be written to the shell.JSP log file. Part of this pattern is shown in Figure 4-A which is a web shell, which can be accessed via browser as can be seen in Figure 4-B.

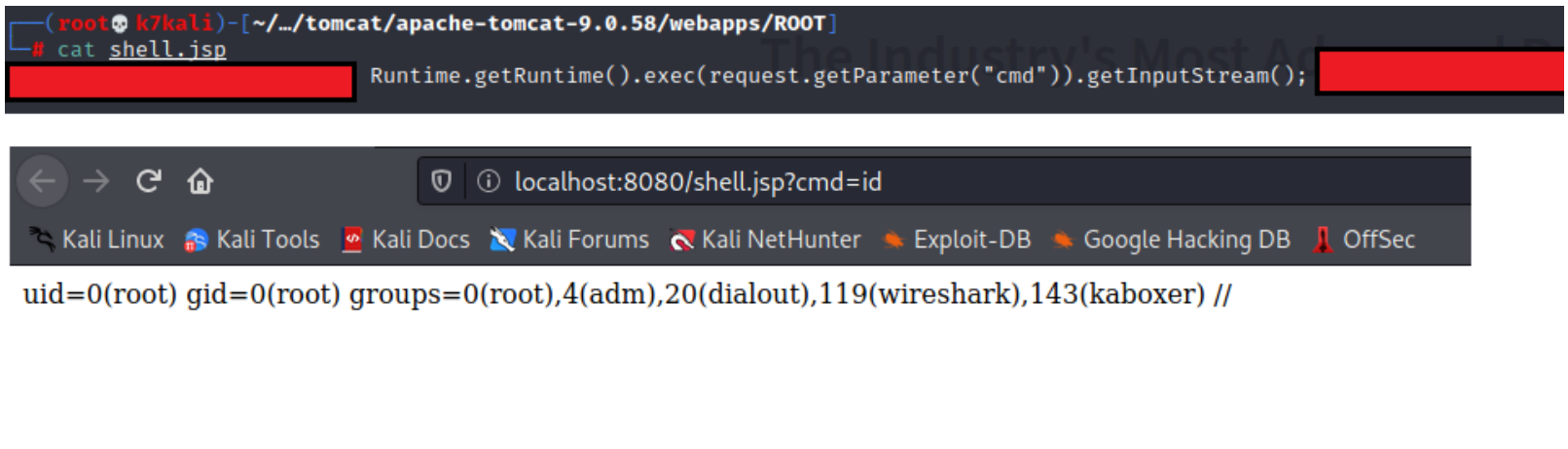

Figure 4: A — Shell uploaded to the Web Server; B — Shell accessed using browser

The root cause(Diff)

The following is a patch diff for springshell vulnerability. The exact patch we are looking at is under CachedIntrospectionResults.java which is restricting property paths under “Class” and properties of types ‘ClassLoader’. By not allowing the binding, and only allowing name variants of class properties, the vulnerability is patched.

“we realized that the disallowedFields configuration setting on WebDataBinder is not intuitive and is not clearly documented.”

“the patterns for disallowedFields in a DataBinder were case sensitive which means a field was not effectively protected” —— Spring team

```
288 289         for (PropertyDescriptor pd : pds) {
289     -         if (Class.class == beanClass &&
290     -             ("classLoader".equals(pd.getName()) ||  "protectionDomain".equals(pd.getName()))) {
291     -           // Ignore Class.getClassLoader() and getProtectionDomain() methods - nobody needs to bind to those
        290 +         if (Class.class == beanClass && (!"name".equals(pd.getName()) && !pd.getName().endsWith("Name"))) {
        291 +           // Only allow all name variants of Class properties
        292 +           continue;
        293 +         }
        294 +         if (pd.getPropertyType() != null && (ClassLoader.class.isAssignableFrom(pd.getPropertyType())
        295 +             || ProtectionDomain.class.isAssignableFrom(pd.getPropertyType()))) {
        296 +           // Ignore ClassLoader and ProtectionDomain types - nobody needs to bind to those
292 297             continue;
293 298           }
```

Figure 5: Patch diff of spring shell vulnerability

Mitigation

1. Update your Spring framework to 5.3.18 and 5.2.20 or later
2. If you are in a restricted environment, it is recommend to implement the following workarounds
    1. Downgrading to Java 8
    2. Upgrading to Tomcat 10.0.20, 9.0.62 or 8.5.78

Further readings

1. https://spring.io/blog/2022/03/31/spring-framework-rce-early-announcement
2. https://tomcat.apache.org/tomcat-7.0-doc/api/org/apache/catalina/valves/AccessLogValve.html
3. https://www.virustotal.com/gui/file/5fb0c8f3daef02b9d2ab285d0bf348cf1cb7c36708b0034ad0dee4998a16b9e9/detection
4. https://github.com/reznok/Spring4Shell-POC
5. https://github.com/dinosn/CVE-2022-22963
6. https://spring.io/blog/2022/04/13/spring-framework-data-binding-rules-vulnerability-cve-2022-22968

Indicators of Compromise (IoCs)

5fb0c8f3daef02b9d2ab285d0bf348cf1cb7c36708b0034ad0dee4998a16b9e9 — Second stage post exploitation file

- Previous Post« Diving into the Emotet Maldoc Boutade
- Next Post

More Posts