Mission Name

DumperClearance

History Background

Ethan and Claire hack their clearance status to grant them a global departure with their own recon car.

Technical High-Level Overview

A memory dump from a computer connected to a system that manages authorizations is provided to the player. This memory dump contains a password that allow to modify Claire and Ethan permissions to fly. The password is located on the Google Chrome database, but in this scenario, all artifacts must be extracted from the memory.

Short Description

Your goal is to analyse a memory dump from a computer connected to Skytech Flight Authorization System and get a password related to an email. Keep in mind, password could be in anywhere.

Mission Description

Your goal is to analyse a memory dump from a computer connected to Skytech Flight Authorization System and get a password. This memory dump contains a password to an email that allow to modify Claire and Ethan permissions to fly. You should analyse memory deeply, in order to get the unencrypted password.

Location

- SYLVARCON | PORT 2 | INTERNATIONAL TRANSIT ZONE

Tools

- Volatility 2 -> follow these steps, all, including python3: https://netsidetech.ca/2021/02/07/how-to-install-volatility-in-kali/
- Download PIP from wget https://bootstrap.pypa.io/pip/2.7/get-pip.py
- https://github.com/tijldeneut/dpapilab-ng -> remove **construct==2.5.5-reupload package.**
- Wxhexeditor: https://howtoinstall.co/es/wxhexeditor
- SQLite viewer tool to read the database extracted.
- Pillow Python Module: pip2 install Pillow

Questions

Which is the PID that has encrypted_key key?

- 3464

To provide you the memory evidence, Which program was used?

- Dumpit

Which IP address of the memory provided?

- 192.168.211.163

Hints

1. Use Volatility2 and get all processes running.
2. Use Volatility2, plugin editbox, and extract all the necessary files from the memory.
3. Use any script which allow you decrypt DPAPI blobs.

Write Up

It´s mandatory to use Volatility 2.

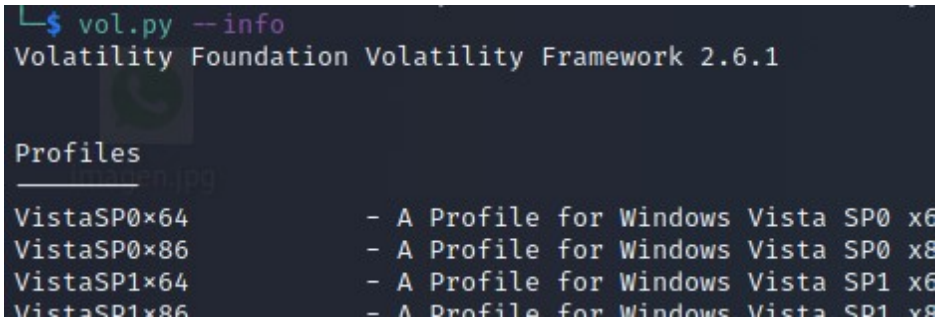


Figure 1

Why volatility2? Editbox plugin it´s mandatory to extract a key from Notepad. Using Volatility3 could increase the level of the challenge. First of all, identify Profile:

```
vol.py  -f /mnt/hgfs/Maquina/C2-M1/evidence/memory imageinfo
```

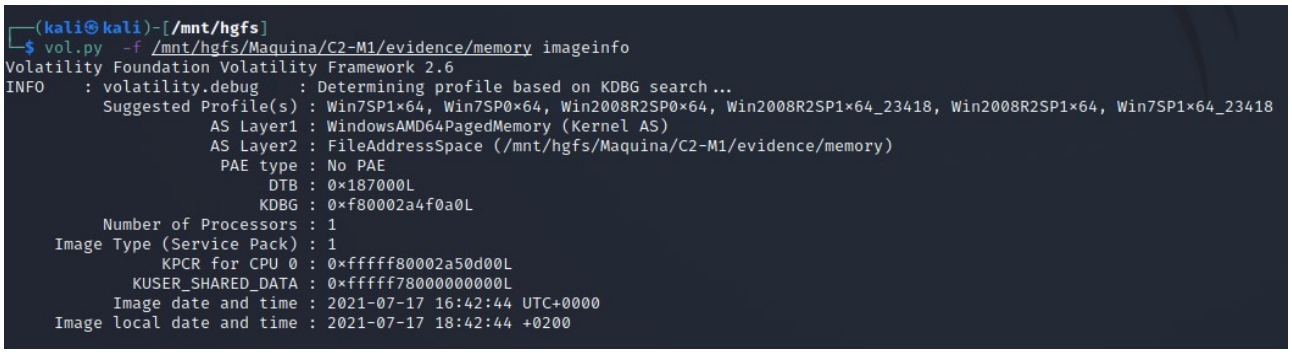


Figure 2

You must install Pillow module for Volatility:

```
pip2 install Pillow

vol.py -f  /mnt/hgfs/Maquina/C2-M1/evidence/memory --profile=Win7SP1x64_23418 screenshot --dump-dir /mnt/hgfs/Maquina/C2-M1/evidence/screenshots
```

Image captured from the memory:

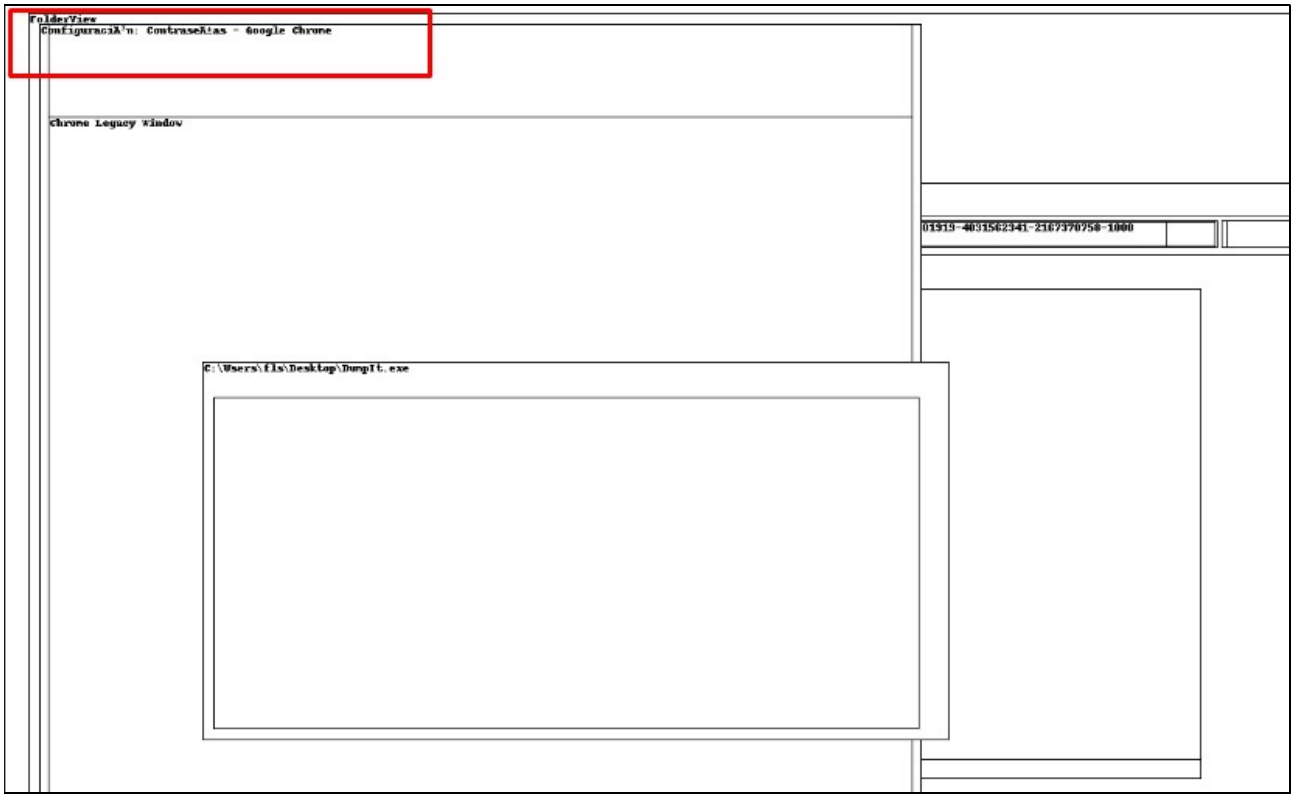


Figure 3

Then, to get windows opened on Windows, player should launch the following command:

```
vol.py -f  /mnt/hgfs/Maquina/C2-M1/evidence/memory --profile=Win7SP1x64_23418 windows > /home/kali/windows.txt

cat /home/kali/windows.txt
```

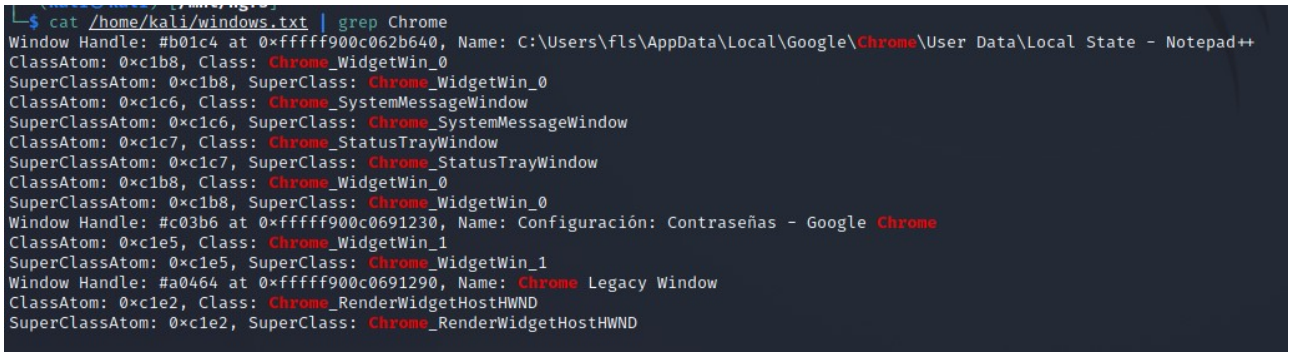


Figure 4

To get windows opened on Windows, player should launch the following command:

vol.py -f /mnt/hgfs/Maquina/C2-M1/evidence/memory --profile=Win7SP1x64_23418 editbox

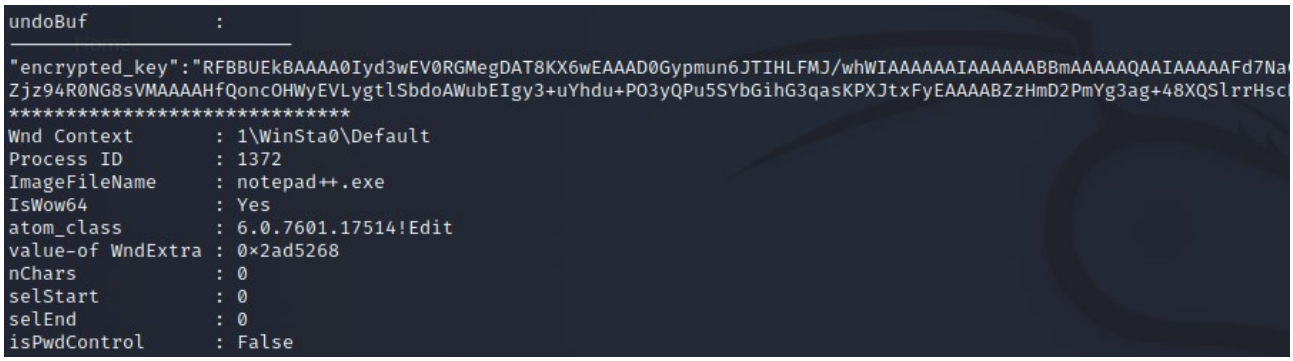


Figure 5

If player performs a search on Google related to encrypted_key, could identify:

- https://chromium.googlesource.com/chromium/src/+/HEAD/docs/user_data_dir.md

RFBBUEkBAAAA0Iyd3wEV0RGMegDAT8KX6wEAAAD0Gypmun6JTIHLFMJ/whWIAAAAAAIAAAAAABBmAAAAAQAAIAAAAAFd7NaChenx91Eq2EdykPy1wnIU/61MQt1C3cQZpetlAAAAAA6AAAAAAgAAIAAAAOhSIha1fZ3cSejUfMWQZcGLNYTPPQXRZjz94R0NG8sVMAAAAHfQoncOHWyEVLygtlSbdoAWubEIgy3+uYhdu+PO3yQPu5SYbGihG3qasKPXJtxFyEAAAABZzHmD2PmYg3ag+48XQSlrrHscBfV/4GAFbSGA4/4IU0PU4CSj4vCbc0fcPFXDUtndA5lZFQqubiW7p/Cxbm8A

Multiple projects to decrypt passwords or cookies based on Chrome, but all relies on CrypunprotectData(), this function only works on Windows live system, not post mortem system.

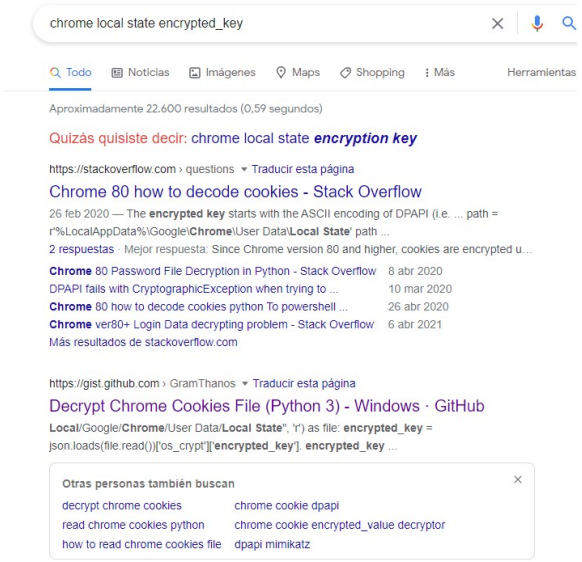
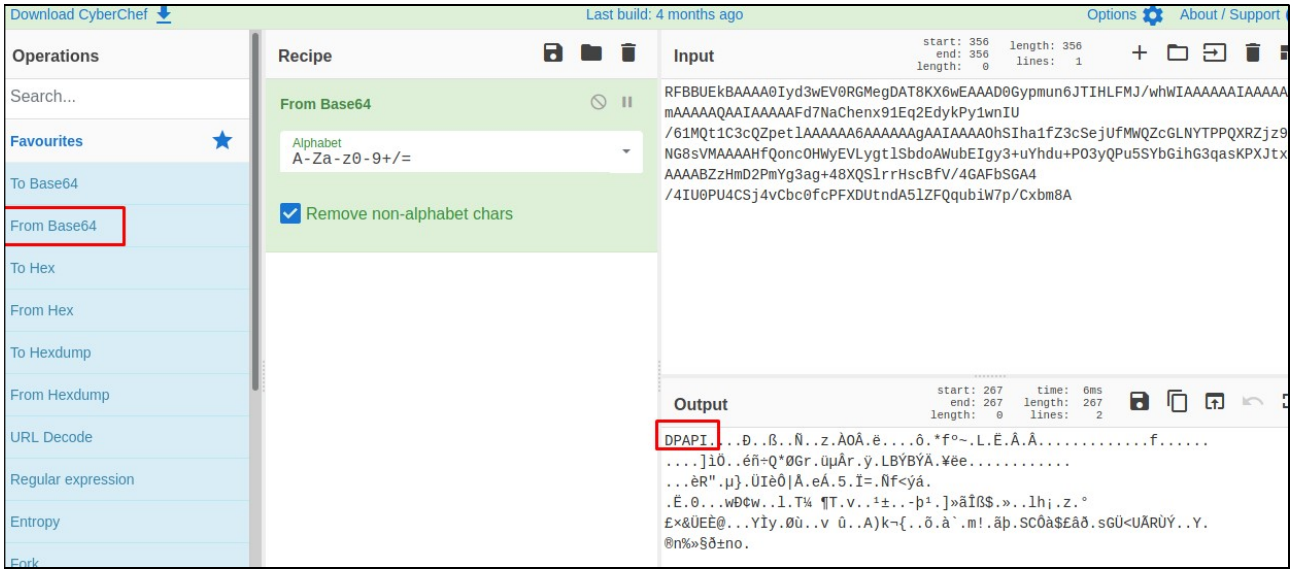


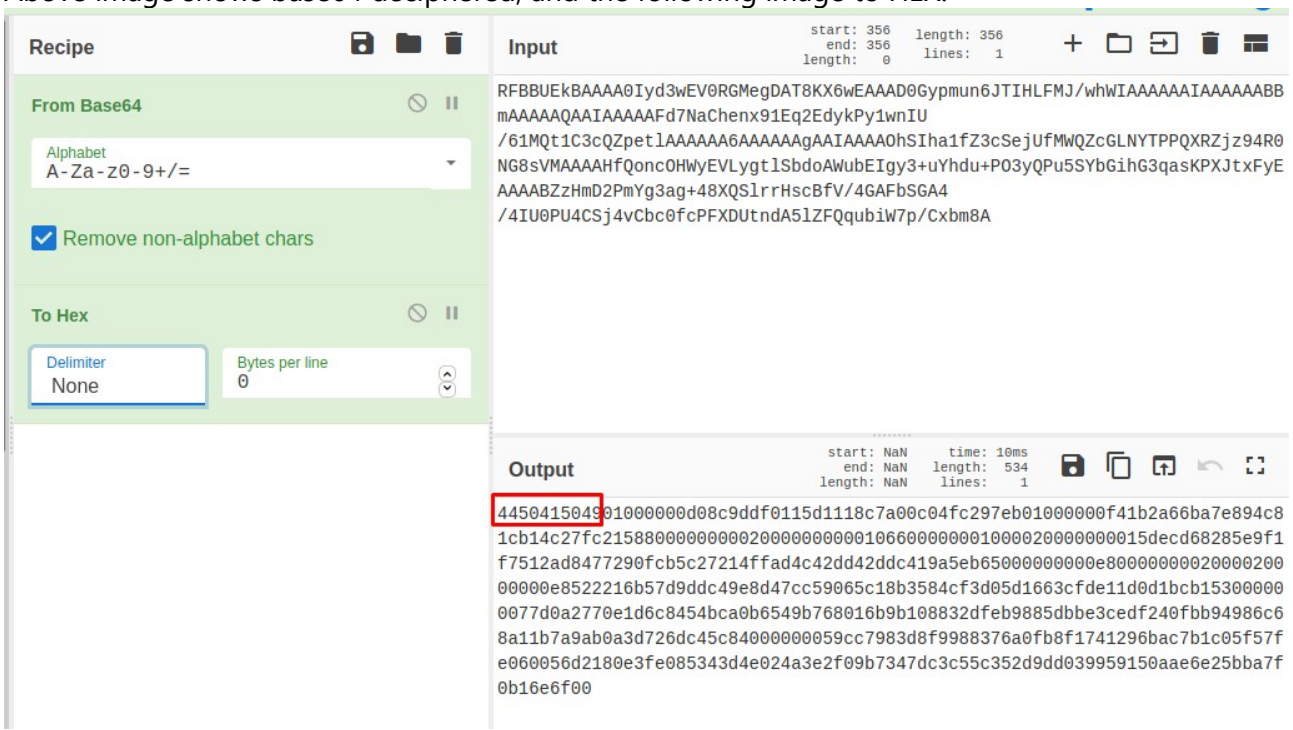
Figure 6

This function only works on the same machine that was encrypted the password. So, to unencrypt password Chrome, we are going to use: https://github.com/tijldeneut/dpapilab-ng

First of all, encrypted_key got it from Volatility , must be deciphered, using CyberChef:



Above image shows base64 deciphered, and the following image to HEX:



We have to remove 5 bytes, related to DPAPI ASCII word:

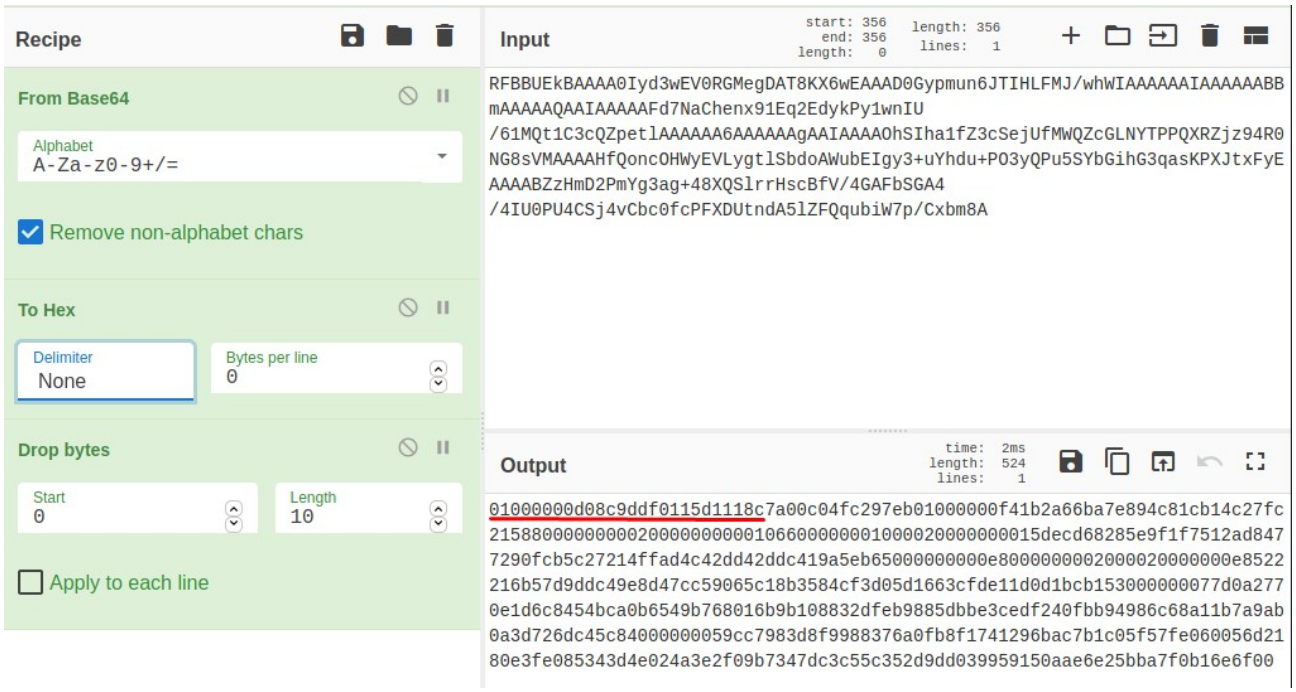


Figure 9

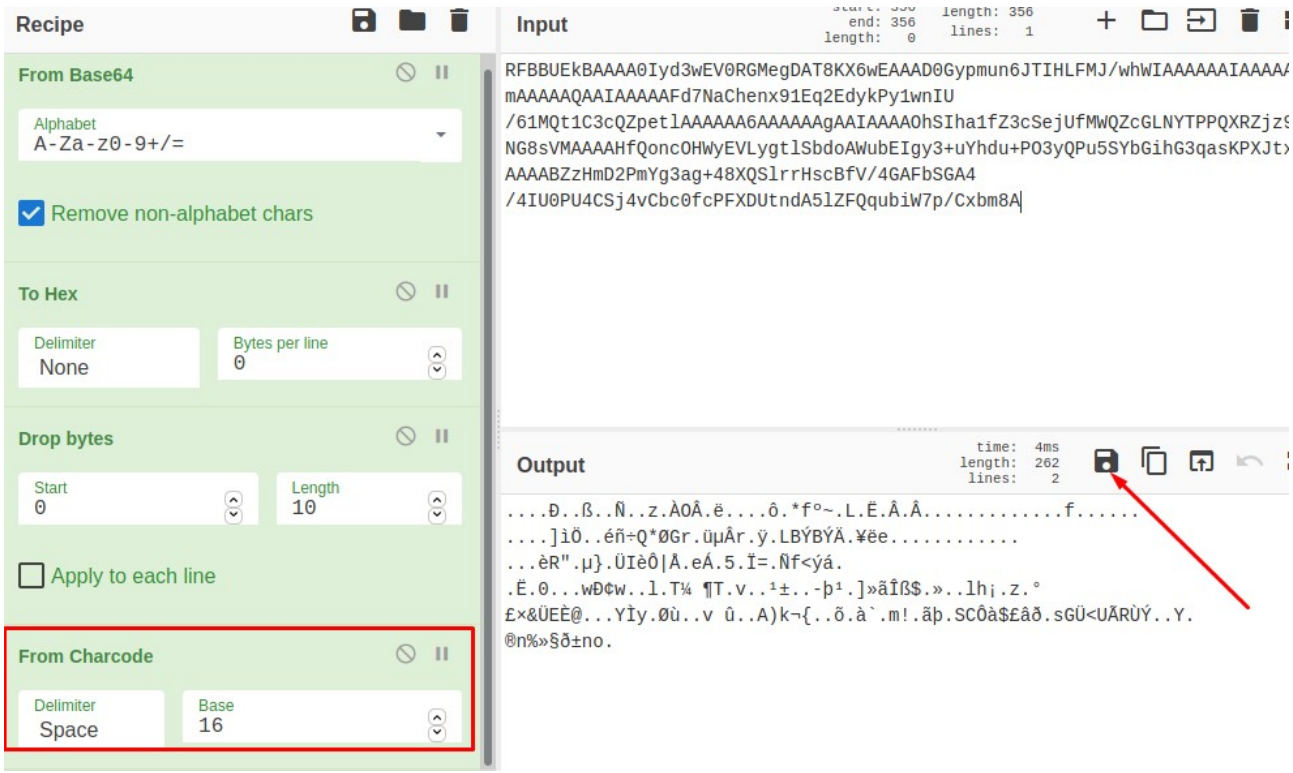


Figure 10

Finally, to charcode and download file generated. Install WxhEditor to check the real DPAPI header:

```
sudo apt-get install wxhexeditor
```

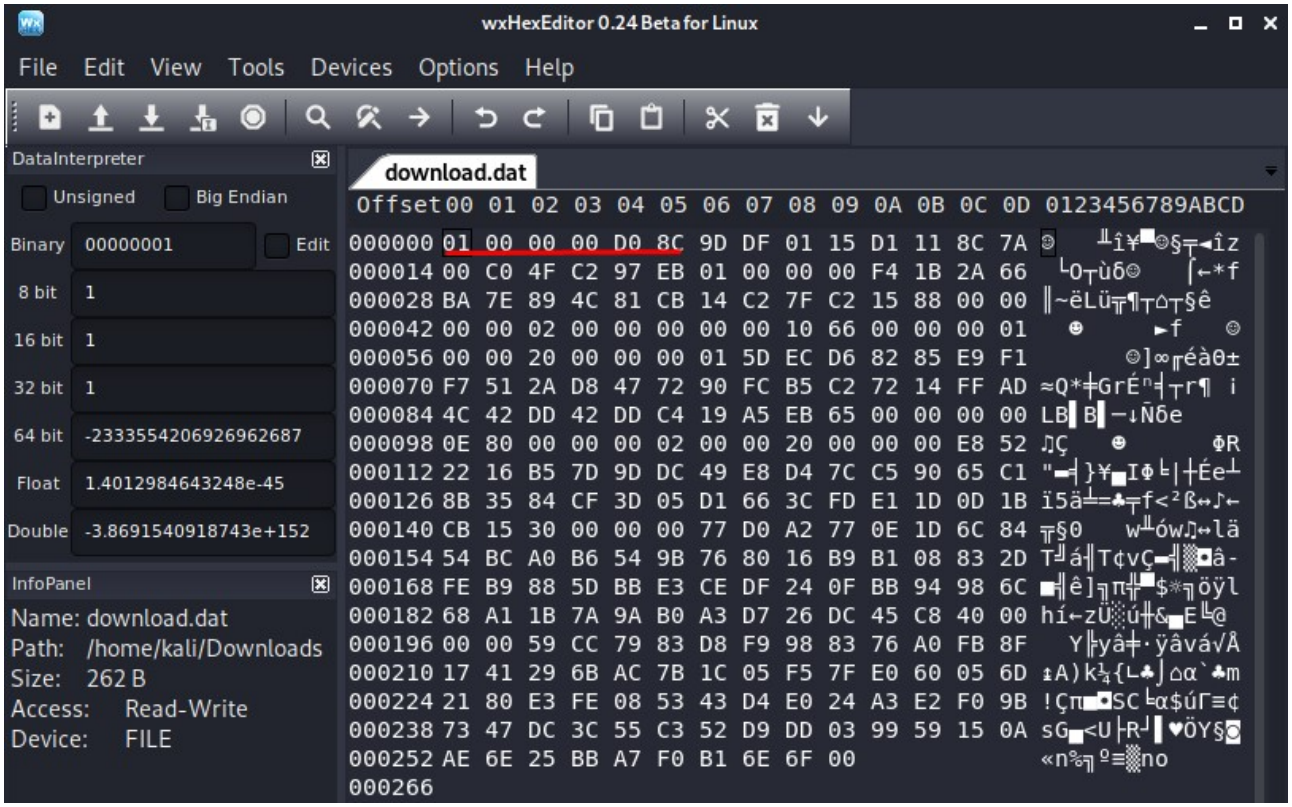


Figure 11

Above image, shows the header of a DPAPI blob. Next step would be to install the following project:

- https://github.com/tijldeneut/dpapilab-ng

```
sudo apt update && sudo apt install -y python3-pip git

python3 -m pip install wheel pytz pycryptodomex python-registry

git clone https://github.com/tijldeneut/dpapilab-ng

cd

#Edit requirements.txt and delete construct==2.5.5-reupload

sudo python3 -m pip install -r requirements.txt

sudo cp -rp *.py /usr/bin/
```

Launch the following command in order to check DPAPI blob data:

```
blobinfo.py /home/kali/Downloads/download.dat
```

Figure 12

At this point we need to extract

- Master key from memory
- Identify the SID of the user
- Locate Password of the user
- SECURITY and SYSTEM hives
- Sqlite Login Data from Chrome

List Master Key offset:

```
vol.py -f  /mnt/hgfs/Maquina/C2-M1/evidence/memory --profile=Win7SP1x64_23418 filescan | grep "Protect"
```

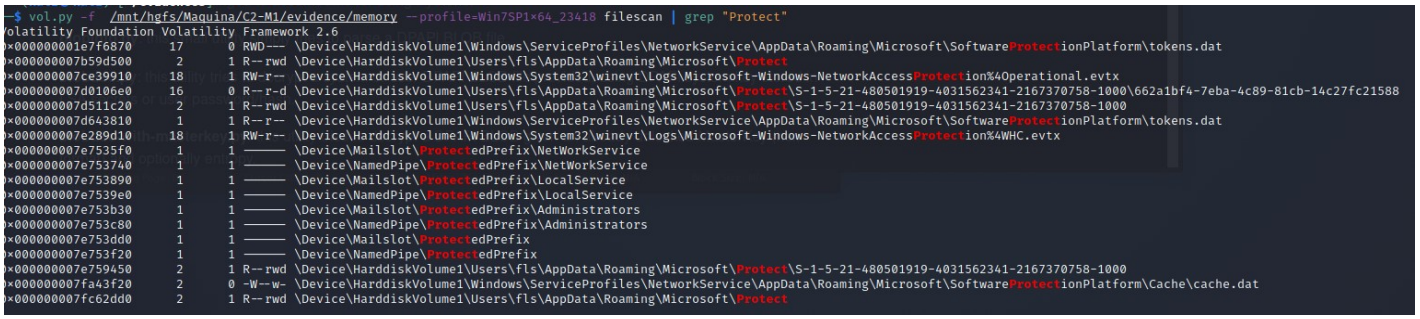


Figure 13

SID could be extracted from path related to the MASTER KEY:

- S-1-5-21-480501919-4031562341-2167370758-1000

Get Master Key using previous offsets:

```
vol.py -f  /mnt/hgfs/Maquina/C2-M1/evidence/memory --profile=Win7SP1x64_23418 dumpfiles -Q 0x000000007d0106e0 -D ./master -u -n
```

```
┌──(kali㉿kali)-[~/evidences]
└─$ vol.py -f  /mnt/hgfs/Maquina/C2-M1/evidence/memory --profile=Win7SP1x64_23418 dumpfiles -Q 0x000000007d0106e0 -D ./master -u -n
Volatility Foundation Volatility Framework 2.6
DataSectionObject 0x7d0106e0   None   \Device\HarddiskVolume1\Users\fls\AppData\Roaming\Microsoft\Protect\S-1-5-21-480501919-4031562341-2167370758-1000\662a1bf4-7eba-4c89-81cb-14c27fc21588
```

List Google Chrome SQLite databases:

```
vol.py -f  /mnt/hgfs/Maquina/C2-M1/evidence/memory --profile=Win7SP1x64_23418 filescan | grep "Login Data"
```

```
└─$ vol.py -f  /mnt/hgfs/Maquina/C2-M1/evidence/memory --profile=Win7SP1x64_23418 filescan | grep "Login Data"
Volatility Foundation Volatility Framework 2.6
0x000000007e2f7070      2      1 RW-rw- \Device\HarddiskVolume1\Users\fls\AppData\Local\Google\Chrome\User Data\Default\Login Data
0x000000007fc679c0      2      1 RW-rw- \Device\HarddiskVolume1\Users\fls\AppData\Local\Google\Chrome\User Data\Default\Login Data For Account
```

Figure 14

Get Sqlite Database:

```
vol.py -f  /mnt/hgfs/Maquina/C2-M1/evidence/memory --profile=Win7SP1x64_23418 dumpfiles -Q 0x000000007e2f7070 -D ./sqlite/ -u -n
```

```
└─$ vol.py -f  /mnt/hgfs/Maquina/C2-M1/evidence/memory --profile=Win7SP1x64_23418 dumpfiles -Q 0x000000007e2f7070 -D ./sqlite/ -u -n
Volatility Foundation Volatility Framework 2.6
DataSectionObject 0x7e2f7070   None   \Device\HarddiskVolume1\Users\fls\AppData\Local\Google\Chrome\User Data\Default\Login Data
SharedCacheMap 0x7e2f7070   None   \Device\HarddiskVolume1\Users\fls\AppData\Local\Google\Chrome\User Data\Default\Login Data
```

Figure 15

Get all windows registry hives:

```
vol.py -f  /mnt/hgfs/Maquina/C2-M1/evidence/memory --profile=Win7SP1x64_23418 dumpregistry -D registry
```

```
┌──(kali㉿kali)-[~/evidences]
└─$ vol.py -f  /mnt/hgfs/Maquina/C2-M1/evidence/memory --profile=Win7SP1x64_23418 dumpregistry -D registry
Volatility Foundation Volatility Framework 2.6
**************************************************
Writing out registry: registry.0xfffff8a0012503e0.SAM.reg

**************************************************
**************************************************
Writing out registry: registry.0xfffff8a001c05010.UsrClassdat.reg

**************************************************
**************************************************
Writing out registry: registry.0xfffff8a000052410.HARDWARE.reg

**************************************************
**************************************************
Writing out registry: registry.0xfffff8a0012eb010.ntuserdat.reg

**************************************************
**************************************************
Writing out registry: registry.0xfffff8a000024010.SYSTEM.reg

**************************************************
**************************************************
Writing out registry: registry.0xfffff8a001b0a010.Syscachehve.reg
```

Figure 16

Dump the current password of the logged user:

```
vol.py -f  /mnt/hgfs/Maquina/C2-M1/evidence/memory --profile=Win7SP1x64_23418 lsadump
```

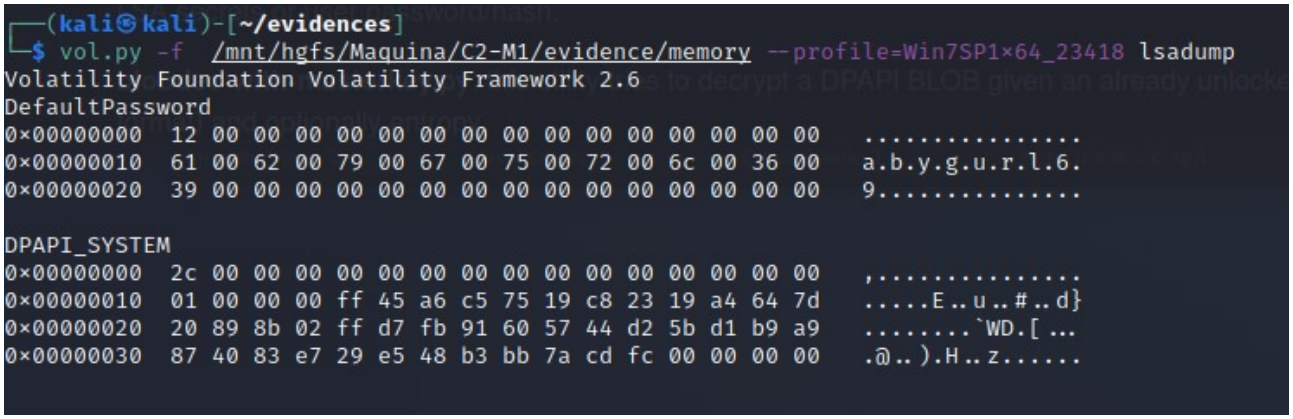


Figure 17

It´s mandatory to rename all hives to their real names, removing the beginning of the files:

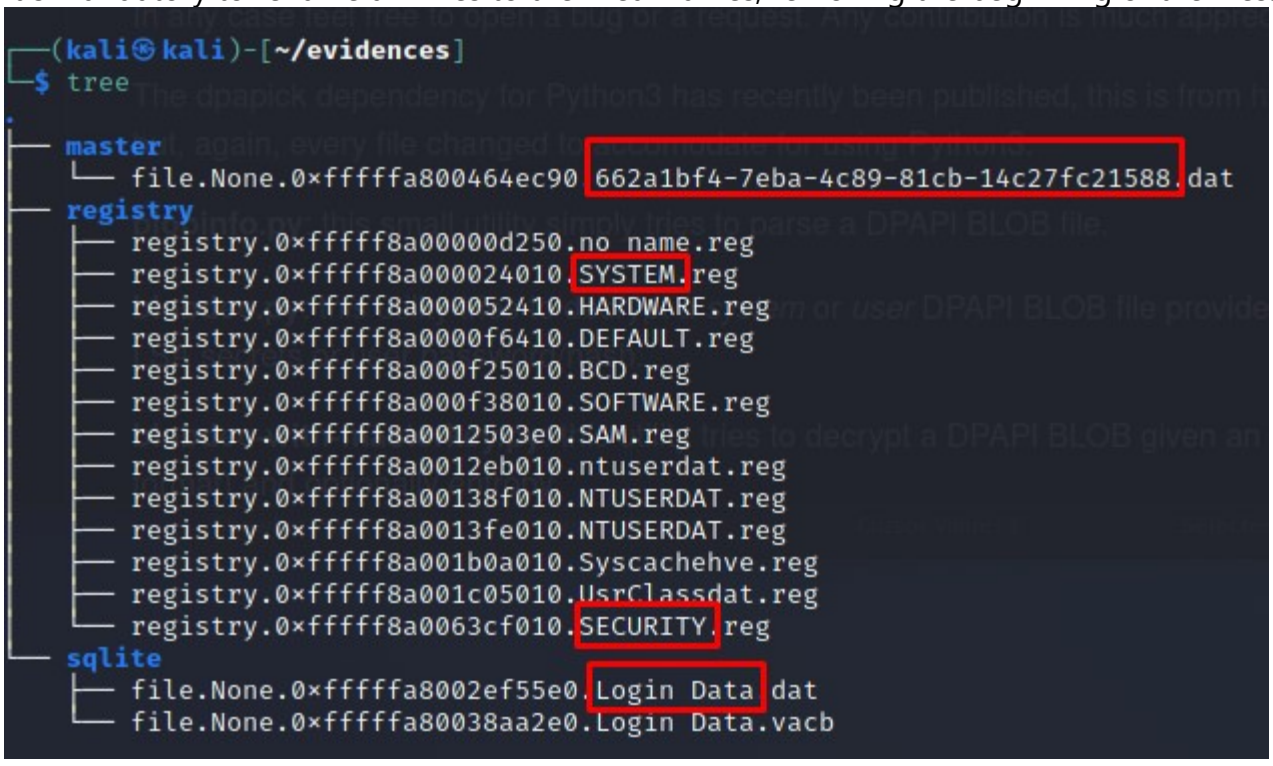


Figure 18

At this point, we hace to use all files extracted from memory and use to decrypt the DPAPI blob:

```
blobdec.py --masterkey master/  --system registry/SYSTEM --security registry/SECURITY --sid S-1-5-21-480501919-4031562341-2167370758-1000 --password abygurl69 /home/kali/Downloads/download.dat
```

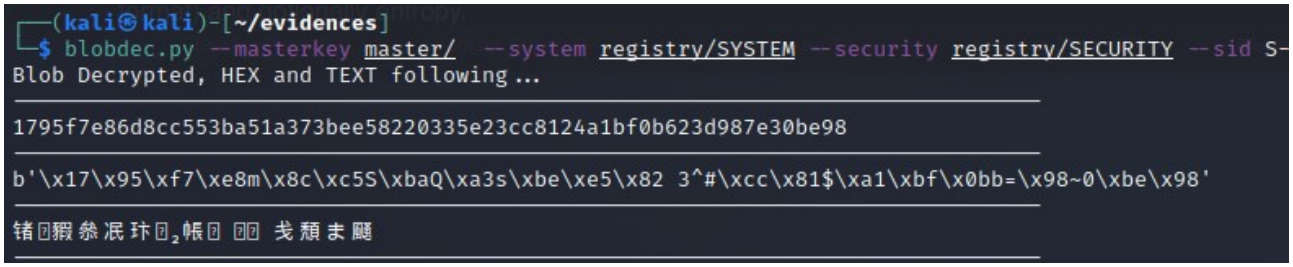


Figure 19

We could use **chrome-edge-opera-dec.py**, but we don´t have the whole content of Local State Chrome file, so we need to modify the following script: **https://github.com/ohyicong/decrypt-chrome-passwords/blob/main/decrypt_chrome_password.py**

```
def get_secret_key():
    try:
        #Remove suffix DPAPI
        secret_key =
        b'\x17\x95\xF7\xE8\x6D\x8C\xC5\x53\xBA\x51\xA3\x73\xBE\xE5\x82\x20\x33\x5E\x23\xCC
        \x81\x24\xA1\xBF\x0B\x62\x3D\x98\x7E\x30\xBE\x98'
        return secret_key
    except Exception as e:
        print("%s"%str(e))
        print("[ERR] Chrome secretkey cannot be found")
        return None

def decrypt_payload(cipher, payload):
    return cipher.decrypt(payload)
```

Figure 20

Now, we have to insert decrypted password on hex format in **decrypt_chrome_password.py**

We need to open Login Data Sqlite database from Google Chrome (previously extracted from memory):

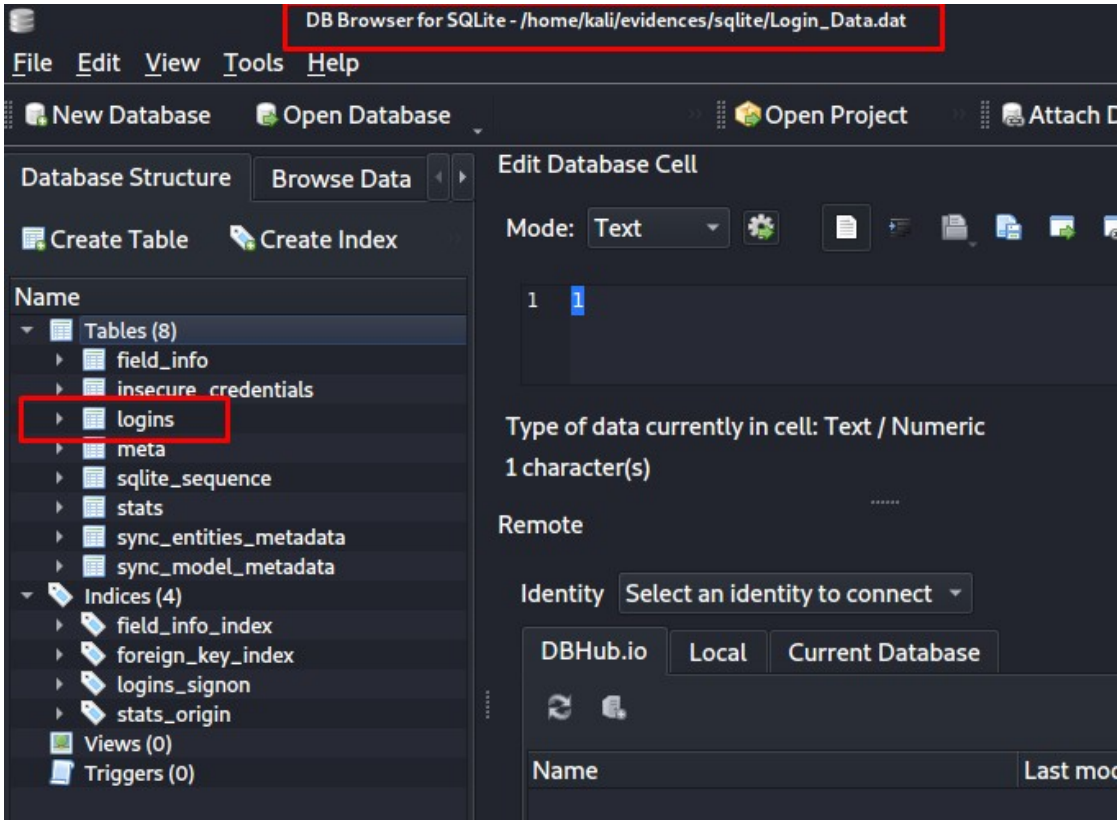


Figure 21

Select logins table, an extract the datablob:

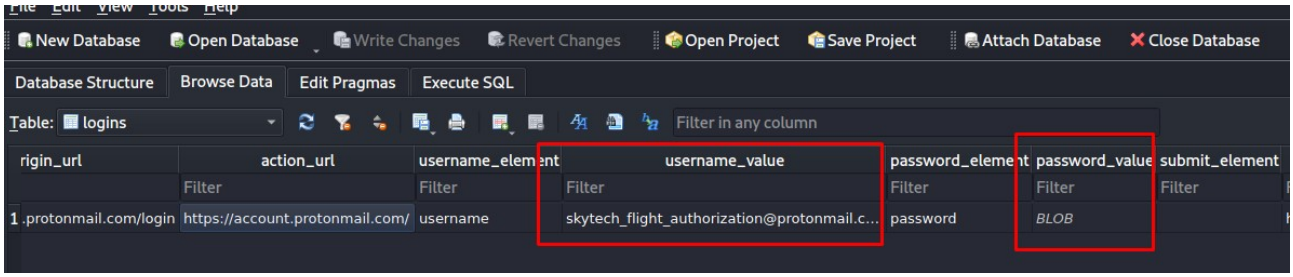


Figure 22

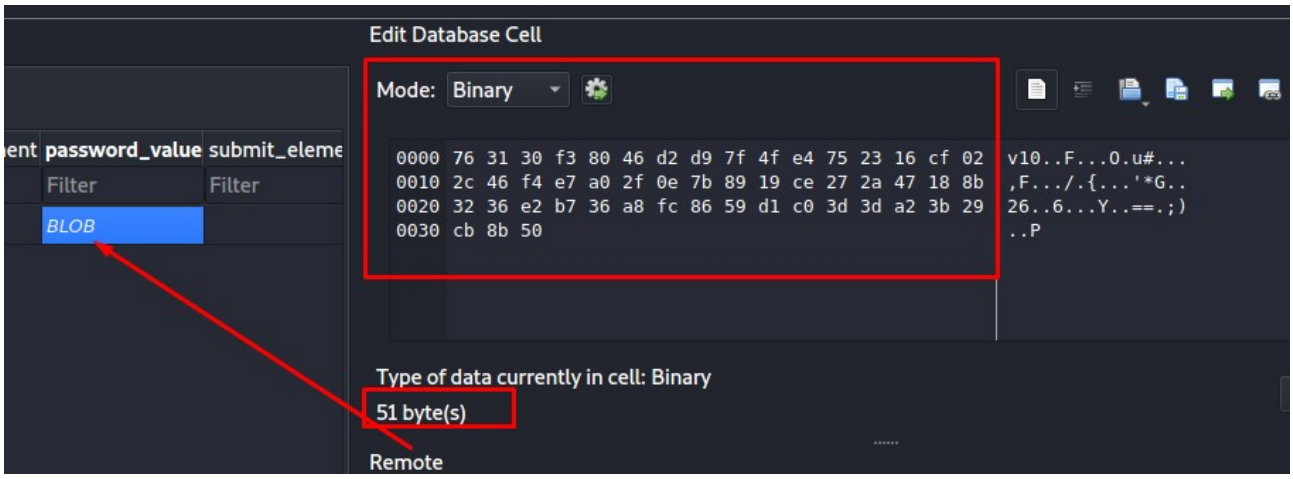


Figure 23

The data blob must be inserted in **decrypt_chrome_password.py**

```
if __name__ == '__main__':
    try:
        secret_key = get_secret_key()
        ciphertext =
        b'\x76\x31\x30\xF3\x80\x46\xD2\xD9\x7F\x4F\xE4\x75\x23\x16\xCF\x02\x2C
        \xA0\x2F\x0E\x7B\x89\x19\xCE\x27\x2A\x47\x18\x8B\x32\x36\xE2\xB7\x36\x
        59\xD1\xC0\x3D\x3D\xA2\x3B\x29\xCB\x8B\x50'
        decrypted_password = decrypt_password(ciphertext, secret_key)
        print("\nPassword: %s\n"%(decrypted_password))
    except Exception as e:
        print("[ERR] "%str(e))
```

Figure 24

```
python3 /mnt/hgfs/Maquina/decrypt.py
```

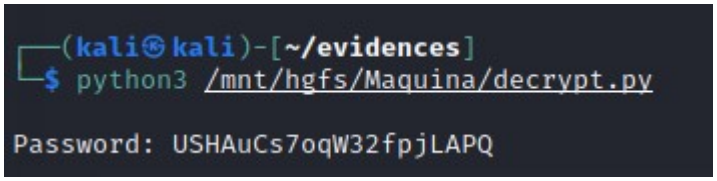


Figure 25

Finally the password is extracted:

- USHAuCs7oqW32fpjLAPQ

Decrypt.py (not to provide players, just for reviewer)

```
import os
import re
import sys
import json
import base64
from Cryptodome.Cipher import AES
import shutil

def get_secret_key():
    try:
        # Remove suffix DPAPI
        secret_key                                                                                                  =
b'\x17\x95\xF7\xE8\x6D\x8C\xC5\x53\xBA\x51\xA3\x73\xBE\xE5\x82\x20\x33\x5E\x23\xCC\x81\x24
\xA1\xBF\x0B\x62\x3D\x98\x7E\x30\xBE\x98'
        return secret_key
    except Exception as e:
        print("[ERR] Chrome secretkey cannot be found: %s" % str(e))
        return None

def decrypt_payload(cipher, payload):
    return cipher.decrypt(payload)

def generate_cipher(aes_key, iv):
    return AES.new(aes_key, AES.MODE_GCM, iv)

def decrypt_password(ciphertext, secret_key):
    try:
        # (3-a) Initialisation vector for AES decryption
        initialisation_vector = ciphertext[3:15]
        # (3-b) Get encrypted password by removing suffix bytes (last 16 bits)
        # Encrypted password is 192 bits
        encrypted_password = ciphertext[15:-16]
        # (4) Build the cipher to decrypt the ciphertext
        cipher = generate_cipher(secret_key, initialisation_vector)
        decrypted_pass = decrypt_payload(cipher, encrypted_password)
        decrypted_pass = decrypted_pass.decode()
        return decrypted_pass  # Return the decrypted password
    except Exception as e:
        print("[ERR] %s" % str(e))
        return None
```

```
if __name__ == '__main__':
    try:
        secret_key = get_secret_key()
        ciphertext                                                                                    =
b'\x76\x31\x30\xf3\x80\x46\xd2\xd9\x7f\x4f\xe4\x75\x23\x16\xcf\x02\x2c\x46\xf4\xe7\xa0\x2f\x0e\
x7b\x89\x19\xce\x27\x2a\x47\x18\x8b\x32\x36\xe2\xb7\x36\xa8\xfc\x86\x59\xd1\xc0\x3d\x3d\xa2
\x3b\x29\xcb\x8b\x50'
        decrypted_password = decrypt_password(ciphertext, secret_key)
        if decrypted_password:
            print("\nPassword: %s\n" % decrypted_password)
    except Exception as e:
        print("[ERR] %s" % str(e))
```

Flag Information

flag{USHAuCs7oqW32fpjLAPQ!}