CTF Report

Chrome Room - TryHackMe

darleep@yahoo.com

August 24, 2024

Table of Contents

1 High-Level Summary

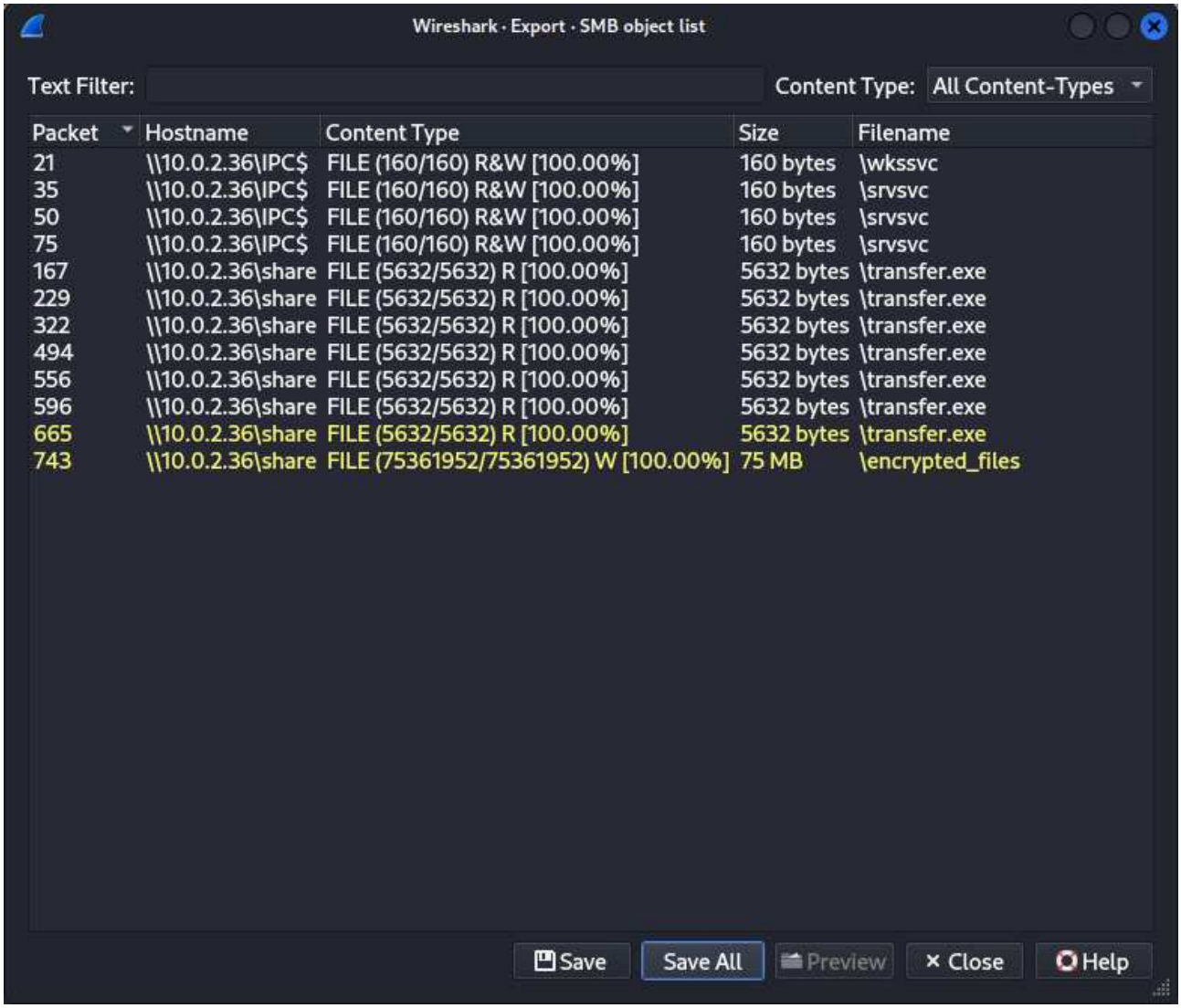
This report details the process of extracting and decrypting sensitive information from a compromised system without relying on a Windows virtual machine. The investigation focuses on analyzing network traffic, decompiling a .NET binary, and retrieving encrypted credentials stored in Google Chrome's password manager.

2 Detailed Findings

2.1 Analyzing The Contents of PCAP-NG

At the outset of the exercise, a zip archive was provided containing a single packet capture file for analysis.

The analysis of the PCAP-NG file in Wireshark revealed significant data exfiltration via SMB, particularly the frequent transfer of transfer.exe. Using Wireshark's "Export Objects" feature under SMB, all successful transfers were identified, with the encrypted_files (75MB) standing out as potentially crucial. These files were exported for further analysis.



When executing the file using Wine, it was observed that it requires a specific zip file located at `C:\Users\hadri\Downloads\files.zip`

```
└─$ ./transfer.exe
Error: Could not find file "/home/0×b0b/Documents/tryhackme/chrome/C:\Users\hadri\Downloads\files.zip"
```

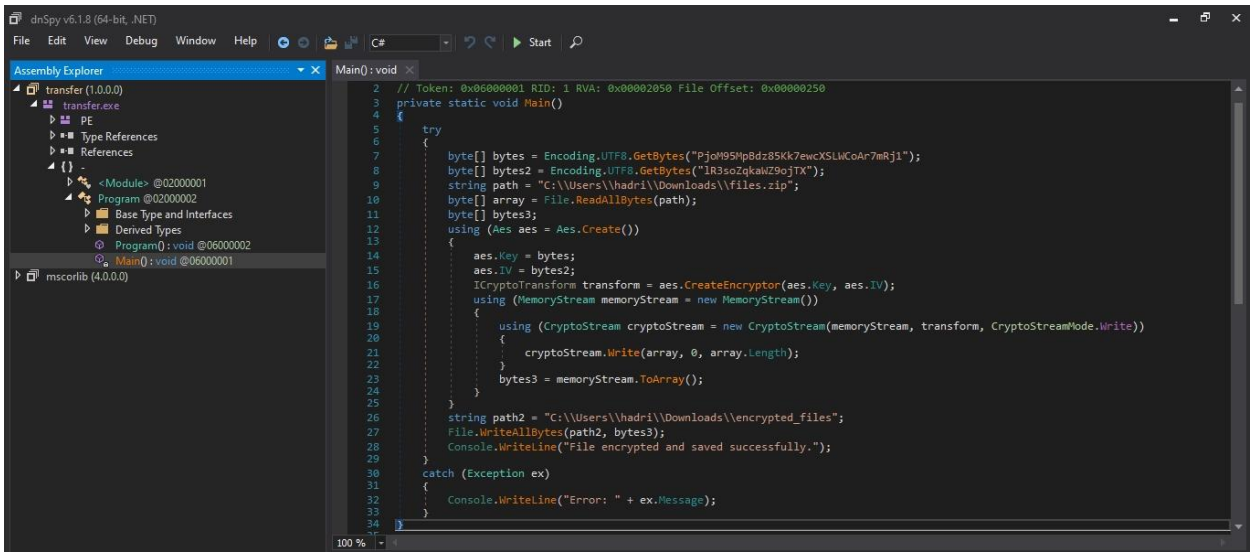
`transfer.exe` was identified as a .NET assembly, indicating it was built using the .NET framework. In contrast, the `encrypted_files` appeared as random data, suggesting it may have been encrypted or obfuscated and necessitating further decryption to determine its contents.

```
❯ lsa
total 72M
drwxr-xr-x 2 kali kali 4.0K Aug 26 19:19 .
drwxr-xr-x 3 kali kali 4.0K Aug 26 19:16 ..
-rw-r--r-- 1 kali kali  72M Aug 26 19:17 encrypted_files
-rw-r--r-- 1 kali kali 5.5K Aug 26 19:17 transfer.exe
❯ file transfer.exe
transfer.exe: PE32 executable (console) Intel 80386 Mono/.Net assembly, for MS Windows, 3 sections
```

2.2 Decompilation and Analysis

Analyzing `transfer.exe`:

- Given that `transfer.exe` is a .NET assembly, it was analyzed using dnSpy. By examining the main function in dnSpy, we were able to gain insights into the file's functionality and operations.



- The analysis of transfer.exe revealed that it declares two byte arrays, bytes and bytes2, which are used as the encryption key and initialization vector (IV) for AES encryption.
- The application reads the contents of the file located at `C:\Users\hadri\Downloads\files.zip`, performs encryption on this data, and then writes the encrypted output to `C:\Users\hadri\Downloads\encrypted_files`

Reversing the Encryption Process

- To reverse the encryption process, we can create a Python script specifying the key, IV, and encryption mode. Running this script will decrypt the data, restoring the original zip file, which can then be unpacked and inspected.

```
from Crypto.Cipher import AES
from Crypto.Util.Padding import unpad
import os
import sys

def decrypt_file(input_path, output_path, key, iv):
    try:

        key = key.encode('utf-8')
        iv = iv.encode('utf-8')

        # Read the encrypted file
        with open(input_path, 'rb') as file:
            encrypted_data = file.read()

        # Create AES cipher object for decryption
        cipher = AES.new(key, AES.MODE_CBC, iv)
```

```
        # Decrypt the data
        decrypted_data = unpad(cipher.decrypt(encrypted_data), AES.block_size)

        # Write the decrypted data to the output file
        with open(output_path, 'wb') as file:
            file.write(decrypted_data)

        print("File decrypted and saved successfully.")
    except FileNotFoundError:
        print("Error: The file was not found.")
    except ValueError as ve:
        print(f"Decryption error: {ve}")
    except Exception as ex:
        print(f"An error occurred: {ex}")

def main():
    if len(sys.argv) !=5:
        print("Usage: python decryptor.py <key> <iv> <input_file> <output_file>")
        sys.exit(1)

    # Define key and IV (Initialization Vector)
    key = sys.argv[1]
    iv = sys.argv[2]

    # Paths to the encrypted and decrypted files
    encrypted_file_path = sys.argv[3]
    decrypted_file_path = sys.argv[4]

    # Decrypt the file
    decrypt_file(encrypted_file_path, decrypted_file_path, key, iv)

if __name__ == "__main__":
    main()
```

Commands:

- `python decryptor.py PjoM95MpBdz85Kk7ewcXSLWCoAr7mRj1 lR3soZqkaWZ9ojTX encrypted_files decrypted_file`

```
❯ python decryptor.py PjoM95MpBdz85Kk7ewcXSLWCoAr7mRj1 lR3soZqkaWZ9ojTX encrypted_files decrypted_file
File decrypted and saved successfully.
```

Keys
PjoM95MpBdz85Kk7ewcXSLWCoAr7mRj1
lR3soZqkaWZ9ojTX

Result:

- By executing the script, we successfully decrypt the data, resulting in the restoration of the `files.zip` as `decrypted_file` archive.

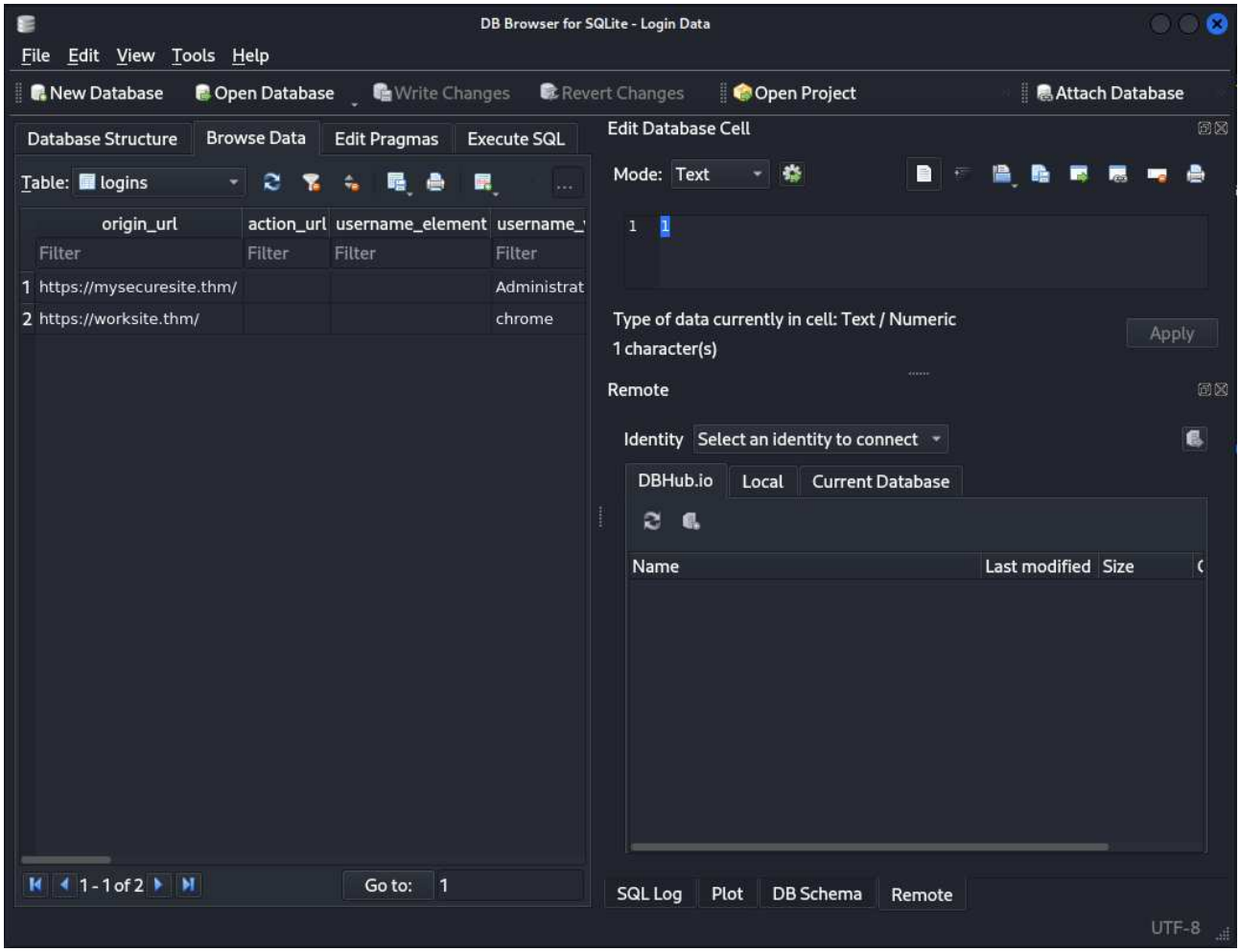
```
❯ file decrypted_files
decrypted_files: Zip archive data, at least v1.0 to extract, compression method=store
```

2.3 Inspect Decrypted ZIP File

- The files.zip archive contains the AppData folder of a Windows user, including Google Chrome user data located under AppData/Local/Google/Chrome/. This includes the Login Data SQLite database, which stores usernames, passwords, and other authentication information saved by Chrome.

```
kali@kali:~/Documents/Chrome_CTF_Forensics/SMB/AppData
❯ ls -la
total 20
drwxr-xr-x 5 kali kali 4096 Dec 21  2022 .
drwxr-xr-x 3 kali kali 4096 Aug 26 19:57 ..
drwxr-xr-x 3 kali kali 4096 Dec 21  2022 Local
drwxr-xr-x 3 kali kali 4096 Dec 21  2022 LocalLow
drwxr-xr-x 4 kali kali 4096 Dec 21  2022 Roaming
```

- The logins table in the database revealed two entries with encrypted passwords for two websites. While URLs can be processed with tools like CyberChef for certain tasks, the encrypted passwords require additional steps for decryption.



- Upon examining the extracted files, two key items were identified:

```
kali@kali:~/Documents/Chrome_CTF_Forensics/SMB/AppData
❯ ls -la
total 20
drwxr-xr-x 5 kali kali 4096 Dec 21  2022 .
drwxr-xr-x 3 kali kali 4096 Aug 26 19:57 ..
drwxr-xr-x 3 kali kali 4096 Dec 21  2022 Local
drwxr-xr-x 3 kali kali 4096 Dec 21  2022 LocalLow
drwxr-xr-x 4 kali kali 4096 Dec 21  2022 Roaming
```

- Google Chrome data located at `/AppData/Local/Google/Chrome/`
- A DPAPI masterkey at `/AppData/Roaming/Microsoft/Protect/S-1-5-21-3854677062-280096443-3674533662-1001/8c6b6187-8eaa-48bd-be16-98212a441580`

```
❯ pwd
/home/kali/Documents/Chrome_CTF_Forensics/SMB/AppData/Roaming/Microsoft/Protect
❯ ls -la
total 16
drwxr-xr-x 3 kali kali 4096 Dec 21  2022 .
drwxr-xr-x 3 kali kali 4096 Dec 21  2022 ..
-rwxr-xr-x 1 kali kali   24 Dec 21  2022 CREDHIST
drwxr-xr-x 2 kali kali 4096 Dec 21  2022 S-1-5-21-3854677062-280096443-3674533662-1001
```

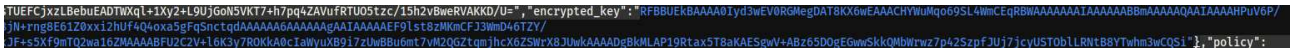
- To proceed with decrypting the passwords saved by Google Chrome, it's essential to understand that:
  - Chrome stores saved passwords in an encrypted state within the Login Data SQLite database file located at `/AppData/Local/Google/Chrome/User Data/Default/Login Data`.

```
kali@kali:~/Documents/Chrome_CTF_Forensics/SMB/AppData/Local/Google/Chrome/User Data/Default
❯ lsa
total 128K
drwxr-xr-x 20 kali kali 4.0K Dec 21  2022  .
drwxr-xr-x 26 kali kali 4.0K Dec 21  2022  ..
drwxr-xr-x  3 kali kali 4.0K Dec 21  2022 'Feature Engagement Tracker'
drwxr-xr-x  3 kali kali 4.0K Dec 21  2022 'GCM Store'
drwxr-xr-x  2 kali kali 4.0K Dec 21  2022  GPUCache
drwxr-xr-x  3 kali kali 4.0K Dec 21  2022 'Local Extension Settings'
drwxr-xr-x  3 kali kali 4.0K Dec 21  2022 'Local Storage'
-rwxr-xr-x  1 kali kali  46K Dec 21  2022 'Login Data'
drwxr-xr-x  2 kali kali 4.0K Dec 21  2022  Network
drwxr-xr-x  2 kali kali 4.0K Dec 21  2022 'Safe Browsing Network'
drwxr-xr-x  5 kali kali 4.0K Dec 21  2022 'Segmentation Platform'
drwxr-xr-x  2 kali kali 4.0K Dec 21  2022 'Session Storage'
drwxr-xr-x  2 kali kali 4.0K Dec 21  2022  Sessions
drwxr-xr-x  2 kali kali 4.0K Dec 21  2022 'Site Characteristics Database'
drwxr-xr-x  3 kali kali 4.0K Dec 21  2022  Storage
drwxr-xr-x  3 kali kali 4.0K Dec 21  2022 'Sync Data'
drwxr-xr-x  4 kali kali 4.0K Dec 21  2022 'Web Applications'
drwxr-xr-x  2 kali kali 4.0K Dec 21  2022  optimization_guide_hint_cache_store
drwxr-xr-x  2 kali kali 4.0K Dec 21  2022  optimization_guide_model_metadata_store
drwxr-xr-x  8 kali kali 4.0K Dec 21  2022  optimization_guide_prediction_model_downloads
drwxr-xr-x  3 kali kali 4.0K Dec 21  2022  shared_proto_db
```

- The encryption key for these passwords is stored in the Local State JSON file at `/AppData/Local/Google/Chrome/User Data/Local State`, under the `os_crypt.encrypted_key` field. This key is itself encrypted using DPAPI.

```
┌──(kali㉿kali)-[~/Documents/Chrome_CTF_Forensics/SMB/AppData/Local/Google/Chrome/User Data]
└─$ lsa
total 172K
drwxr-xr-x 26 kali kali 4.0K Dec 21  2022  .
drwxr-xr-x  3 kali kali 4.0K Dec 21  2022  ..
drwxr-xr-x 20 kali kali 4.0K Dec 21  2022  Default
drwxr-xr-x  2 kali kali 4.0K Dec 21  2022  DesktopSharingHub
drwxr-xr-x  2 kali kali 4.0K Dec 21  2022  FileTypePolicies
drwxr-xr-x  2 kali kali 4.0K Dec 21  2022  FirstPartySetsPreloaded
drwxr-xr-x  2 kali kali 4.0K Dec 21  2022  GrShaderCache
-rwxr-xr-x  1 kali kali  67K Dec 21  2022 'Local State'
drwxr-xr-x  2 kali kali 4.0K Dec 21  2022  MEIPreload
drwxr-xr-x  3 kali kali 4.0K Dec 21  2022  MediaFoundationWidevineCdm
drwxr-xr-x  3 kali kali 4.0K Dec 21  2022  OnDeviceHeadSuggestModel
drwxr-xr-x  3 kali kali 4.0K Dec 21  2022  OptimizationHints
drwxr-xr-x  2 kali kali 4.0K Dec 21  2022  OriginTrials
drwxr-xr-x  2 kali kali 4.0K Dec 21  2022  PKIMetadata
drwxr-xr-x  3 kali kali 4.0K Dec 21  2022  RecoveryImproved
drwxr-xr-x  2 kali kali 4.0K Dec 21  2022  SSLErrorAssistant
drwxr-xr-x  2 kali kali 4.0K Dec 21  2022 'Safe Browsing'
drwxr-xr-x  2 kali kali 4.0K Dec 21  2022  SafetyTips
drwxr-xr-x  2 kali kali 4.0K Dec 21  2022  ShaderCache
drwxr-xr-x  4 kali kali 4.0K Dec 21  2022 'Subresource Filter'
drwxr-xr-x  2 kali kali 4.0K Dec 21  2022  SwReporter
drwxr-xr-x  2 kali kali 4.0K Dec 21  2022  ThirdPartyModuleList64
drwxr-xr-x  2 kali kali 4.0K Dec 21  2022  UrlParamClassifications
drwxr-xr-x  2 kali kali 4.0K Dec 21  2022  WidevineCdm
drwxr-xr-x  2 kali kali 4.0K Dec 21  2022  ZxcvbnData
drwxr-xr-x  2 kali kali 4.0K Dec 21  2022  hyphen-data
drwxr-xr-x  3 kali kali 4.0K Dec 21  2022  pnacl
```

- DPAPI encryption relies on masterkeys, which are also encrypted with a key derived from the user's password.



2.4 Retrieving Credentials in Chrome Password Manager

Cracking of Masterkey

- To decrypt the masterkey, the user's password is required. In the absence of the password, a brute-force approach is necessary.
- The process begins by using DPAPImk2john to convert the masterkey into a format that is compatible with John the Ripper, facilitating the password cracking procedure.
- By utilizing John the Ripper to crack the hash, we successfully obtained the user's password, enabling us to address the first question of the challenge.

Tools:

- DPAPImk2john
- John the Ripper

Commands:

- `DPAPImk2john -mk AppData/Roaming/Microsoft/Protect/S-1-5-21-3854677062-280096443-3674533662-1001/8c6b6187-8eaa-48bd-be16-98212a441580 -c local -S S-1-5-21-3854677062-280096443-3674533662-1001 > hash`
- `john hash --wordlist=/usr/share/wordlists/rockyou.txt`


```
Using default input encoding: UTF-8
Loaded 1 password hash (DPAPImk, DPAPI masterkey file v1 and v2 [SHA1/MD4 PBKDF2-(SHA1/SHA512)-DPAPI-variant 3DES/AES256 128/128 SSE2 4x])
Cost 1 (iteration count) is 8000 for all loaded hashes
Will run 2 OpenMP threads
Proceeding with wordlist:/usr/share/john/password.lst
Press 'q' or Ctrl-C to abort, almost any other key for status
bubbles          (?)
1g 0:00:00:01 DONE (2024-08-26 20:30) 0.5050g/s 250.5p/s 250.5c/s 250.5C/s brenda..chance
Use the "--show" option to display all of the cracked passwords reliably
Session completed.
```

Result:

- The outcome of the cracking process revealed that the user's password is `bubbles`.

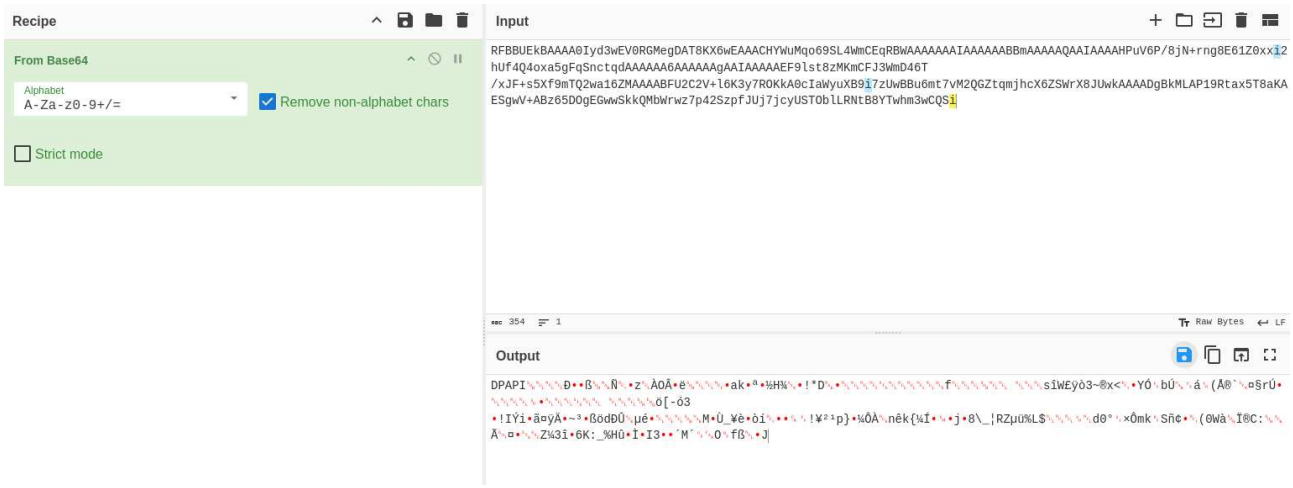
Retrieve The Master Key

- To extract the masterkey, we utilize the `impacket-dpapi` tool. By specifying the masterkey location, the SID, and the decrypted user password, we successfully obtain the decrypted masterkey.

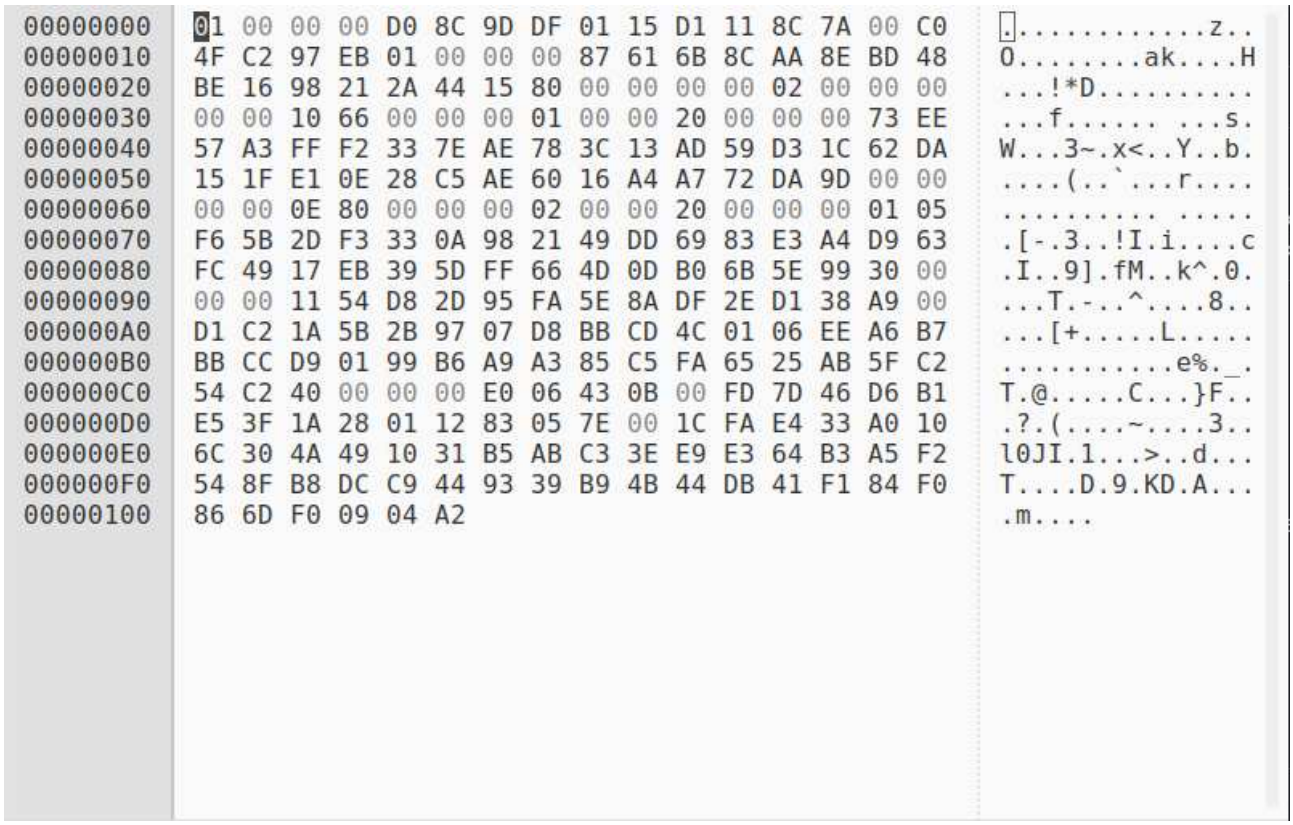
```
❯ impacket-dpapi masterkey -file 8c6b6187-8eaa-48bd-be16-98212a441580  -password bubbles -sid S-1-5-21-3854677062-280096443-3674533662-1001
Impacket v0.11.0 - Copyright 2023 Fortra

[MASTERKEYFILE]
Version     :        2 (2)
Guid        : 8c6b6187-8eaa-48bd-be16-98212a441580
Flags       :        5 (5)
Policy      :        0 (0)
MasterKeyLen: 000000b0 (176)
BackupKeyLen: 00000090 (144)
CredHistLen : 00000014 (20)
DomainKeyLen: 00000000 (0)

Decrypted key with User Key (SHA1)
Decrypted key: 0xca4387eb0a71fc0eea23e27f54b9ae240379c9e82a05d6fca73ecee13ca2e0e4d98390844697d8ed10715415c56152653edf460a47b70ddb868a03ee6a3f9840
```

Decode the Encrypted Key of Local State

- With the decrypted masterkey in hand, we can proceed to decrypt the key stored in `AppData/Local/Google/Chrome/User Data/Local State`. This requires converting the encrypted key back to its original binary form. As an intermediate step, we can use `CyberChef` to perform this conversion.



- To prepare for decryption, we first remove the DPAPI-specific data from the encrypted key using a hex editor. This step ensures that we can accurately process the key for further decryption.



Decrypt the Encrypted Key of Local State

- Using the `blobdec-with-masterkey.py` script, we decrypt the decoded key with the decrypted masterkey. This step provides us with the final decryption key required to access the encrypted data.

```
❯ blobdec-with-masterkey.py --masterkey ca4387eb0a71fc0eea23e27f54b9ae240379c9e82a05d6fca73ecee13ca2e0e4d98390844697d8ed10715415c56152653edf460a47b70ddb868a03ee6a3f9840 encrypted_dpapi
[+] Blob Decrypted, HEX and TEXT following...
----------------------------------------------------------------------------------
9a3094f1bfe3e19d5d039fb569d35d49ad083ac34dbcd5d9e42a506b8d4a192c
----------------------------------------------------------------------------------
b'\x9a0\x94\xf1\xbf\xe3\xe1\x9d]\x03\x9f\xb5i\xd3]I\xad\x08:\xc3M\xbc\xd5\xd9\xe4*Pk\x8dJ\x19,'
----------------------------------------------------------------------------------
```

Commands:

- `blobdec-with-masterkey.py --masterkey ca4387eb0a71fc0eea23e27f54b9ae240379c9e82a05d6fca73ecee13ca2e0e4d98390844697d8ed10715415c56152653edf460a47b70ddb868a03ee6a3f9840 enc_key.dat`

Retrieve the Passwords in Login Data

- With the decrypted Local State key, we are now able to retrieve login entries from the database.

Tools:

- dcp.py

Commands:

- `python dcp.py -S 9a3094f1bfe3e19d5d039fb569d35d49ad083ac34dbcd5d9e42a506b8d4a192c`

3 Conclusion

The assessment of the Chrome room on TryHackMe uncovered critical security issues, including the ability to access and decrypt sensitive user credentials. By resolving these vulnerabilities, the overall security of the system can be significantly improved.

Findings	**Value**
First password	bubbles
URL found in the first index	https://mysecuresite.thm/
password found in the first index	Sup3rPaS$w0rd1
URL found in the second index	https://worksite.thm/
Password found in the second index	Sup3rSecuR3!

4 References

- **TryHackMe Chrome Room** - *Link*
- **Forensic Recovery of Chrome Based Browser Passwords** - *Medium Post*
- DCP Tool for Retrieving Passwords - *Link*
- Windows DPAPI-NG lab in Python 3 - *Link*
- Dumping DPAPI secrets - *Link*

End of Report