**Challenge:** Silent Breach Lab

**Platform:** CyberDefenders

**Category:** Endpoint Forensics

**Difficulty:** Medium

**Tools Used:** FTK Imager, Browsing History View, DB Browser for SQLite, Strings, Grep, ChatGPT

**Summary:** This challenge involves analysing a provided disk image using a suite of forensic tools and techniques. I found it extremely enjoyable, and it sure did test my forensics abilities. One component of this challenge involved analysing Windows Live Mail messages, which is something I have never seen before. For those that enjoy digital forensics, I highly recommend this challenge.

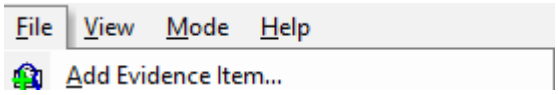
**Scenario:** The IMF is hit by a cyber attack compromising sensitive data. Luther sends Ethan to retrieve crucial information from a compromised server. Despite warnings, Ethan downloads the intel, which later becomes unreadable. To recover it, he creates a forensic image and asks Benji for help in decoding the files.

**Resources:**

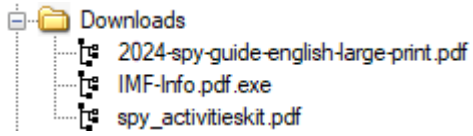
- https://boncaldoforensics.wordpress.com/2018/12/09/microsoft-hxstore-hxd-email-research/

**What is the MD5 hash of the potentially malicious EXE file the user downloaded?**

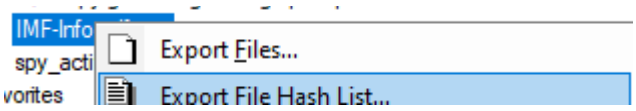
Before we find the MD5 hash of the potentially malicious binary, we need to import the disk image into FTK imager. Once you have extracted the Lab file, launch FTK Imager and navigate to File > Add Evident Item:




Select image file and browse to the .ad1 file (AD1 is a disk image format created by FTK Imager). The question asks about an EXE the user downloaded, so a good start is to check out this users downloads directory:




Here we can find one executable file, to export the file hashes simply right click the file and select Export File Hash List:

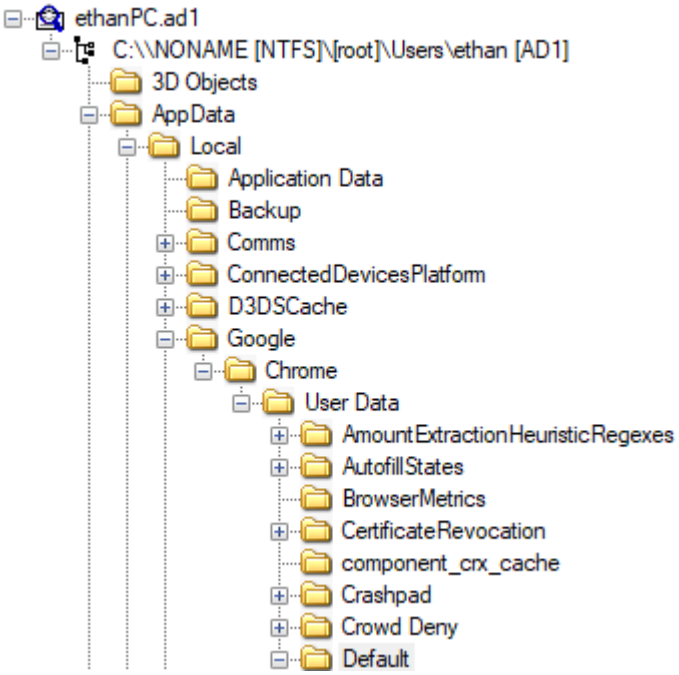


| MD5 | SHA1 | File Names |
| --- | --- | --- |
| 336a7cf476ebc7548c93507339196abb | f0847df8a58527cc34b96876efaae941865555fd | ethanPC.ad1\C:\\NONAME [NTFS]\[root]\Users\ethan [AD1]\Downloads\IMF-Info.pdf.exe |

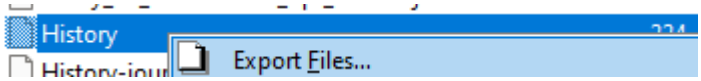
Answer: 336a7cf476ebc7548c93507339196abb

**What is the URL from which the file was downloaded?**

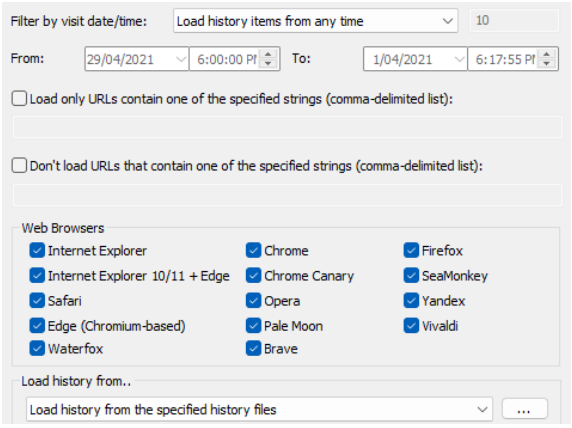
Based on the wording of the question, it is pretty clear that we need to take a look at the users browsing history. You can do so by using DB Browser for SQLite, or a tool like Browsing History View. In my case, I chose the latter. The user's chrome history can be found in C:\Users\<username>\AppData\Local\Google\Chrome\User Data\Default:

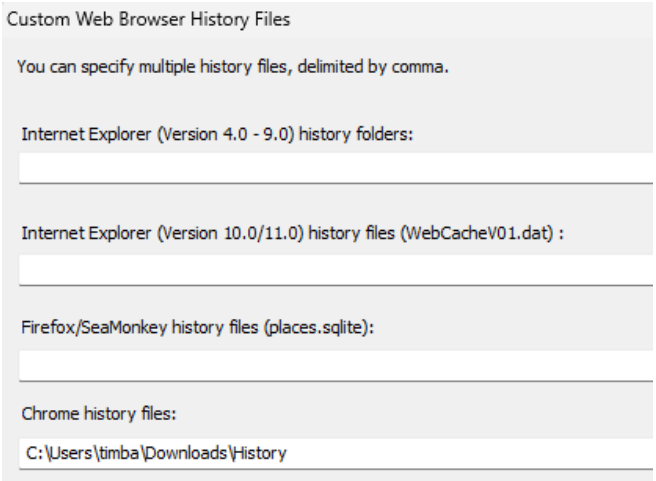



You can export the History file by Right-Clicking the file and selecting Export Files:




To see the browsing history from this file, launch Browser History View and make sure to supply the path to the History File:




After clicking okay, you will be presented with the users browsing history. After looking through their Chrome history, I found nothing. So, I then checked their Edge history by navigating to C:\Users\<username>\AppData\Local\Microsoft\Edge\User Data\Default and downloaded the History file. If you open up this history file in DB Browser for SQLite and choose the downloads table to see the users download history. Here we can find the path to the malicious binary found in the first question:

| current_path ▲ |
| --- |
| Filter |
| C:\Users\ethan\Downloads\2024-spy-guide-english-large-print.pdf |
| C:\Users\ethan\Downloads\ChromeSetup.exe |
| C:\Users\ethan\Downloads\IMF-Info.pdf.exe |

Along with the referrer URL (i.e., where it was downloaded from):

| referrer |
| --- |
| Filter |
| https://spy-museum.s3.amazonaws.com... |
| https://www.google.com/ |
| http://192.168.16.128:8000/ |

Answer: http://192.168.16.128:8000/IMF-Info.pdf.exe

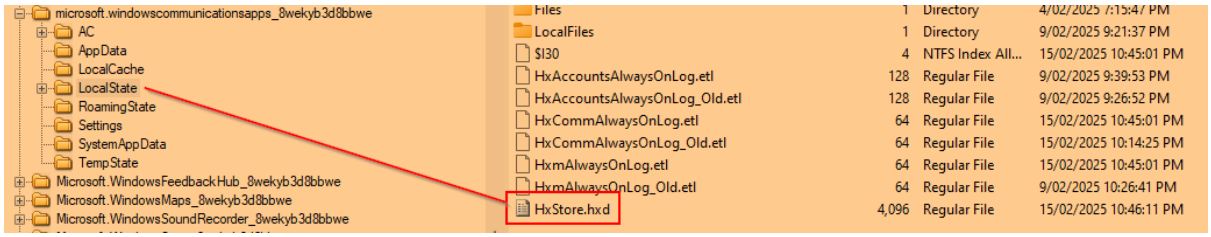
**What application did the user use to download this file?**

We found the download history within the users Microsoft Edge files.

Answer: Microsoft Edge

**By examining Windows Mail artifacts, we found an email address mentioning three IP addresses of servers that are at risk or compromised. What are the IP addresses?**

A Windows Live Mail message is an email message created or saved using Windows Live Mail. This is something I had never heard of up until this point. In order to answer this question, you should take a look at the provided resource. We can find stored emails in C:\Users\<username>\AppData\Local\Packages\microsoft.windowscommunicationsapps_82ekyb3d8bbwe\LocalState\ in a file called HxStore.hxd:




I recommend exporting this file, and running the strings command against it. After looking at the results, you start to see emails:

```
b>From: </b><a href="mailto:l
.stickell@outlook
p</a><brQ
Monday, February 10, 2025 12:58 AM3
z.hunt08
Subjec
        TOP SECRET: Cyber Attack
- Mission Details
```

```
IP: 145.67.29.88
```

```
212.33.10.112
&  V-&
192.168.16.128
```

Answer: 145.67.29.88, 212.33.10.112, 192.168.16.128

**By examining the malicious executable, we found that it uses an obfuscated PowerShell script to decrypt specific files. What predefined password does the script use for encryption?**

The first thing I did was export the malicious binary and run the strings command on it. Due to its size, it wouldn't be feasible to look through each result (something like floss would do a better job), Therefore, I grepped for "powershell" to try and see the script:

```
timba@TimsPC:/mnt/c/Users/timba/Downloads$ strings IMF-Info.pdf.exe | grep powershell -A 1 -B 15
path
scriptPath
main
process
TEMP
C:\Users\ethan\AppData\Local\Temp
Gz3m6mG3j2TyAqF2Zx4v.ps1
$wy7qIGPnm36HpvjrL2TMUaRbz = "K0QfK0QZjJ3bG1CIlxWaGRXdw5WakASblRXStUmdv1WZSBCIgAiCNoQDpgSZz9GbD5SbhVm
SZz9GbD5SbhVmc0N1b0BXeyNGJgACIgoQDK0QKos2YvxmQsFmbpZEazVHbG5SbhVmc0N1b0BXeyNGJgACIgoQDpgGdn5WZM5yclRX
Gd5JkbpFGbwRCKlRXaydlLtFWZyR3UvRHc5J3YkACIgAiCNoQDpUGdpJ3V6oTXlR2bN1WYlJHdT9GdwlncD5SeoBXYyd2b0BXeyNk
3UbBCLy9Gdwlncj5WZkACLtFWZyR3U0V3bkgSbhVmc0N1b0BXeyNkL5hGchJ3ZvRHc5J3QukHdpJXdjV2Uu0WZ0NXeTBCdjVmai9U
Hc5J3YkACIgAiCNkSZ0FWZyNkO60VZk9WTlxWaG5yTJ5SblR3c5N1WgwSZslmR0VHc0V3bkgSbhVmc0NVZslmRu8USu0WZ0NXeTBC
WZyR3U0V3bkACIgAiCNoQDpUGbpZEd1BnbpRCKzVGd5JEbsFEZhVmU6oTXlxWaG5yTJ5SblR3c5N1Wg0DIzVGd5JkbpFGbwRCIgAC
WRlRXYlJ3QuMXZhRCI9AicvRHc5J3YuVGJgACIgoQDK0wNTN0SQpjOdVGZv10ZulGZkFGUukHawFmcn9GdwlncD5Se0lmc1NWZT5S
WYQ5yclFGJgACIgoQDDJ0Q6oTXlR2bNJXZoBXaD5SeoBXYyd2b0BXeyNkL5RXayV3YlNlLtVGdzl3UbBSPgUGZv1kLzVWYkACIgAi
CIgACIK0QeltGJg0DI5V2SuMXZhRCIgACIK0QKoUGdhVmcDpjOdNXZB5SeoBXYyd2b0BXeyNkL5RXayV3YlNlLtVGdzl3UbBSPgMX
mLnACLnQiZkBnLcdCIlNWYsBXZy1CIlxWaGRXdw5WakASPgUGbpZEd1BHd19GJgACIgoQD7BSKzVGbpZEd1BnbpRCIulGIlxWaGRX
QDK0QKK0gImRGcu42bpN3cp1ULG1UScxFcvR3azVGRcxlbhhGdlxFXzJXZzVFXcpzQiACIgAiCNwiImRGcuQXZyNWZT1iRNlEXcB3
```

All I did was chuck this obfuscated script into ChatGPT, and asked it to decode it for me. After doing so, it found the password:

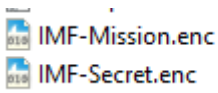
```
$password = "Imf!nfo#2025Sec$"
```

Answer: Imf!nfo#2025Sec$

**After identifying how the script works, decrypt the files and submit the secret string.**

In the obfuscated script from the previous question, it took in two input files:

```
$inputFiles = @(
    "C:\\Users\\ethan\\Desktop\\IMF-Secret.pdf",
    "C:\\Users\\ethan\\Desktop\\IMF-Mission.pdf"
)
```




After exporting the files, I asked ChatGPT to create me a python script that can decrypt these files:

“””

```
from pathlib import Path
from Crypto.Cipher import AES
from Crypto.Protocol.KDF import PBKDF2

# === Configuration ===
password = b"Imf!nfo#2025Sec$"
salt = bytes([0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08])
iterations = 10000
key_size = 32
iv_size = 16

# === Derive Key and IV ===
derived = PBKDF2(password, salt, dkLen=key_size + iv_size, count=iterations)
key = derived[:key_size]
iv = derived[key_size:]

# === Decrypt Function ===
def decrypt_file(input_file: Path):
    try:
        with input_file.open("rb") as f:
            ciphertext = f.read()

        cipher = AES.new(key, AES.MODE_CBC, iv)
        plaintext = cipher.decrypt(ciphertext)

        # Strip PKCS7 padding
        pad_len = plaintext[-1]
        if pad_len > 16:
            raise ValueError("Padding length invalid — possibly wrong key/IV.")
        plaintext = plaintext[:-pad_len]
```

```
        output_file = input_file.with_name(input_file.stem + ".decrypted.pdf")
        with output_file.open("wb") as f:
            f.write(plaintext)

        print(f"[+] Decrypted: {input_file.name} -> {output_file.name}")
    except Exception as e:
        print(f"[!] Failed to decrypt {input_file.name}: {e}")

# === Main Script ===
if __name__ == "__main__":
    current_dir = Path(__file__).resolve().parent
    encrypted_files = list(current_dir.glob("*.enc"))

    if not encrypted_files:
        print("[!] No .enc files found in this directory.")
    else:
        for enc_file in encrypted_files:
            decrypt_file(enc_file)
"""
```

**Flag - CyberDefenders{N3v3r_eX3cuTe_F!l3$_dOwnL0ded_fr0m_M@lic10u5_$erV3r}**

Answer: CyberDefenders{N3v3r_eX3cuTe_F!l3$_dOwnL0ded_fr0m_M@lic10u5_$erV3r}