

# Operation DRAGONCLONE: Chinese Telecommunication industry targeted via VELETRIX & VShell malware

By Subhajeet Singha

Published: 2025-06-06 · Archived: 2026-04-06 01:38:43 UTC

## Contents

- Introduction
- Initial Findings
- Infection Chain.
- Technical Analysis
  - Stage 0 – Malicious ZIP File.
  - Stage 1 – Malicious VELETRIX implant.
  - Stage 2 – Malicious V-Shell implant.
- Hunting and Infrastructure.
- Attribution
- Conclusion
- Seqrite Protection.
- IOCs
- MITRE ATT&CK.

**Authors: Subhajeet Singha and Sathwik Ram Prakki**

## Introduction

Seqrite Labs APT-Team has recently found a campaign, which has been targeting the Chinese Telecom Industry. The campaign is aimed at targeting China Mobile Tietong Co., Ltd. which is a well-known subsidiary of China Mobile, one of the major telecom companies in China. The entire malware ecosystem involved in this campaign is based on usage of VELETRIX malware and VShell malware a very well-known adversary simulation tool, which is also known for widely being adopted by threat actors from China to target various western entities in-the-wild.

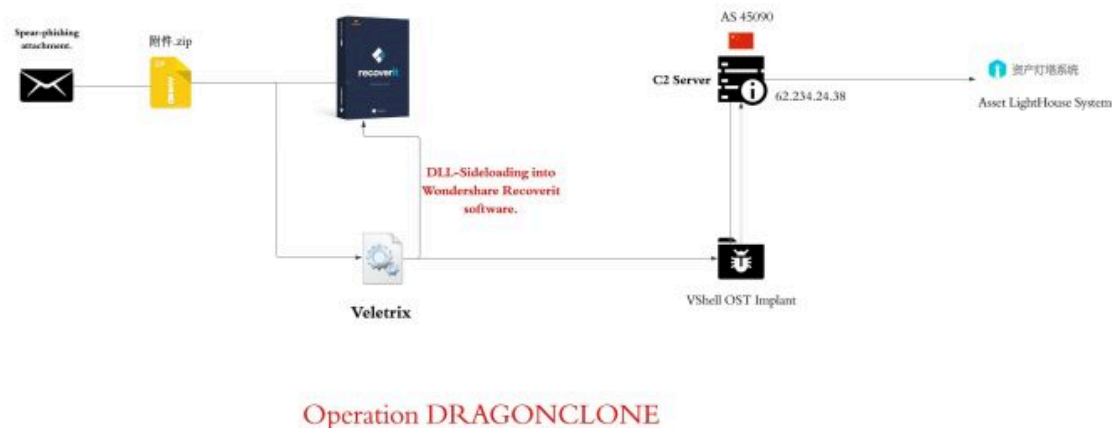
In this blog, we will explore the technical sophistication of the campaign, we encountered during our analysis. We will examine the various stages of this campaign, starting with deep dive into the initial infection stage to implants used in this campaign, ending with a final overview covering the campaign.

## Initial Findings

Recently, on 13th of May, our team found a malicious ZIP file, which surfaced both on various sources like [VirusTotal](#), where ZIP file has been used as preliminary source of infection, containing multiple EXE and DLLs inside the ZIP folder. The same file was also found by other [threat researchers](#) the very same day.

The ZIP contains an interesting executable file known as 2025 China Mobile Tietong Co., Ltd. Internal Training Program is about to launch, please register as soon as possible.exe which loads a bunch of interesting DLLs such as drstat.dll and much more. Then, we decided to look into the workings of these bunch of files.

### Infection Chain



### Operation DRAGONCLONE

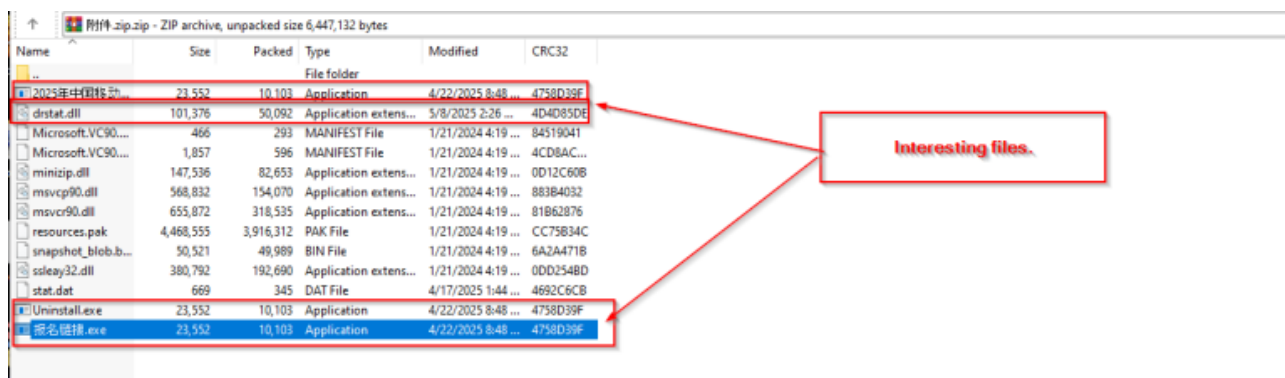
© SEQRTE APT TEAM

### Technical Analysis

We will break down analysis into three different parts, starting with looking into the malicious ZIP attachment, followed by malicious Veletrix implant and then we will look into some brief analysis into the VShell malware.

#### Stage 0 – Malicious ZIP File.

Initially, we found a malicious ZIP file, known as 附件.zip, also known as attachment.zip. Upon, looking into the contents of the ZIP file.

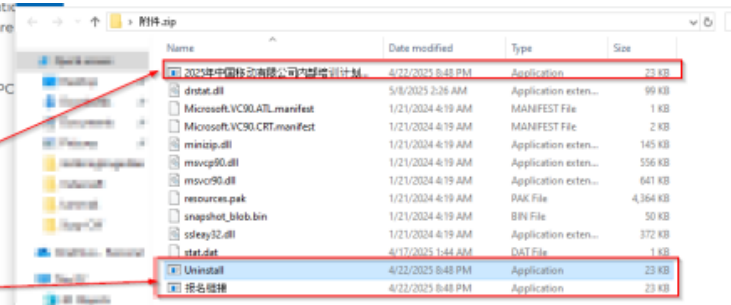


We found a set of interesting EXE and DLL and XML files, amongst them most of them were legitimately Microsoft Signed binaries, whereas some of them had have code-signing certificate by Shenzhen Thunder Networking Technologies Ltd , while an interesting DLL file drstat.dll which is often associated with WonderShare RepairIt software.

location may vary a lot depending on the user's option when installing the application. Repairit\unins000.exe is the full command line if you want to uninstall Wondershare Repairit file has a size of 11.52 MB (12078832 bytes) on disk and is called repairit.exe.

Wondershare Repairit(Build 5.5.3.6) installs the following the executables on your PC

- autoupgrade.exe (37.00 KB)
- bspatch.exe (43.16 KB)
- Bs5ndRpt64.exe (498.88 KB)
- bugreport.exe (550.50 KB)
- BugSplatHD64.exe (324.88 KB)
- cbscustomizedclient\_repairit.exe (17.50 KB)
- closeallprocess\_repairit.exe (16.50 KB)
- CovertRawEx\_preview.exe (16.50 KB)
- drstat.exe (23.00 KB)

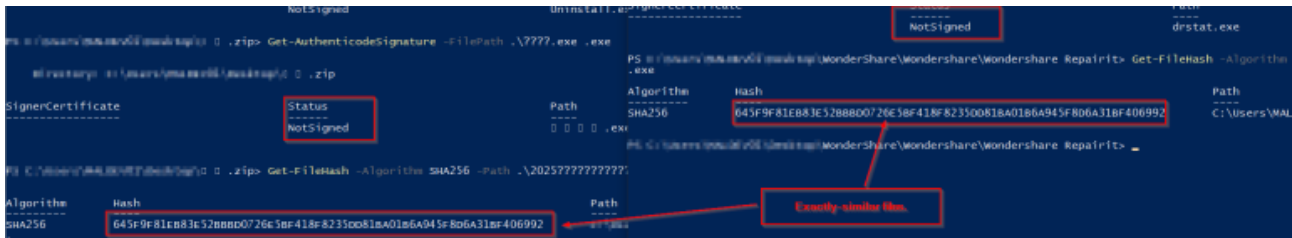


- C:\Program Files\Wondershare\Wondershare Repairit\drstat.dll
- C:\Program Files\Wondershare\Wondershare Repairit\drstat.exe

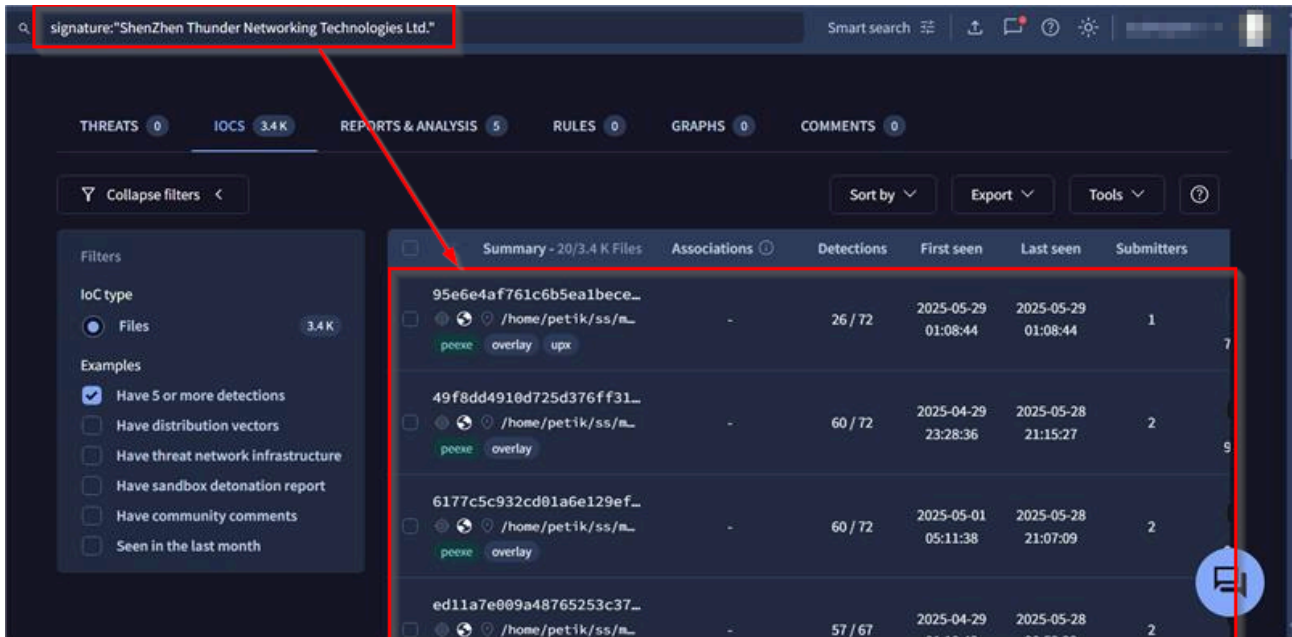
Upon confirming from an official website of Wondershare Repairit , we can confirm that an executable known as drstat.exe which have been renamed and packaged thrice with three different names, which are:

- **China Mobile Limited's 2025 internal training program is about to begin. Please register as soon as possible.**
- Uninstall.
- **Registration-link.**

Next, we decided to confirm further that, either Wondershare does sign the actual binary, which is officially available from their website.

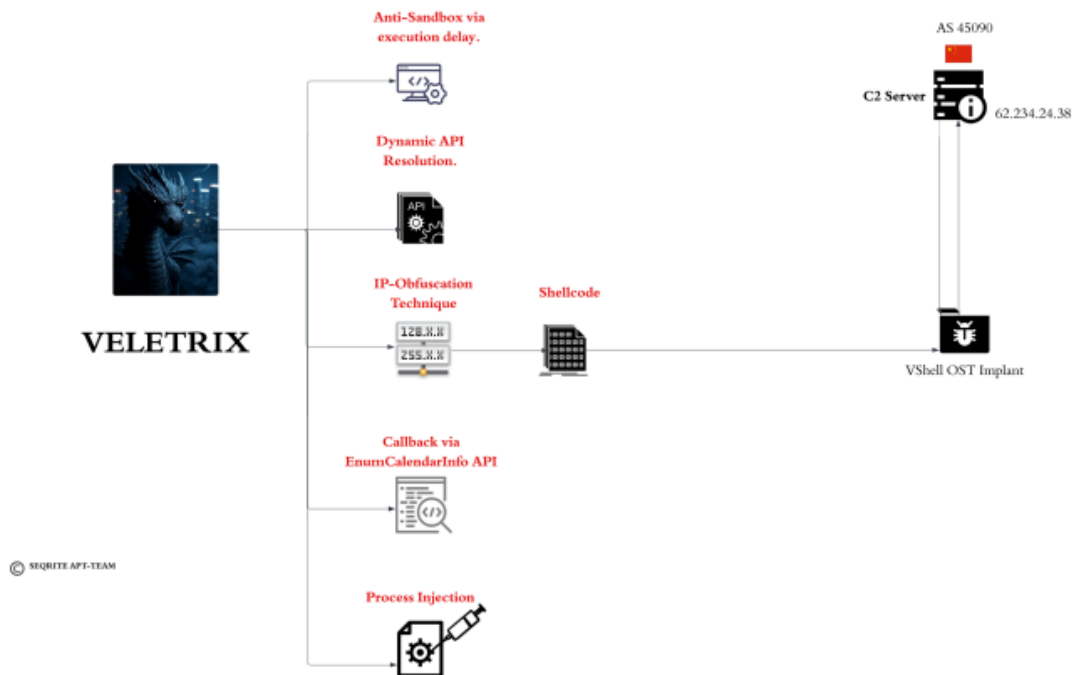


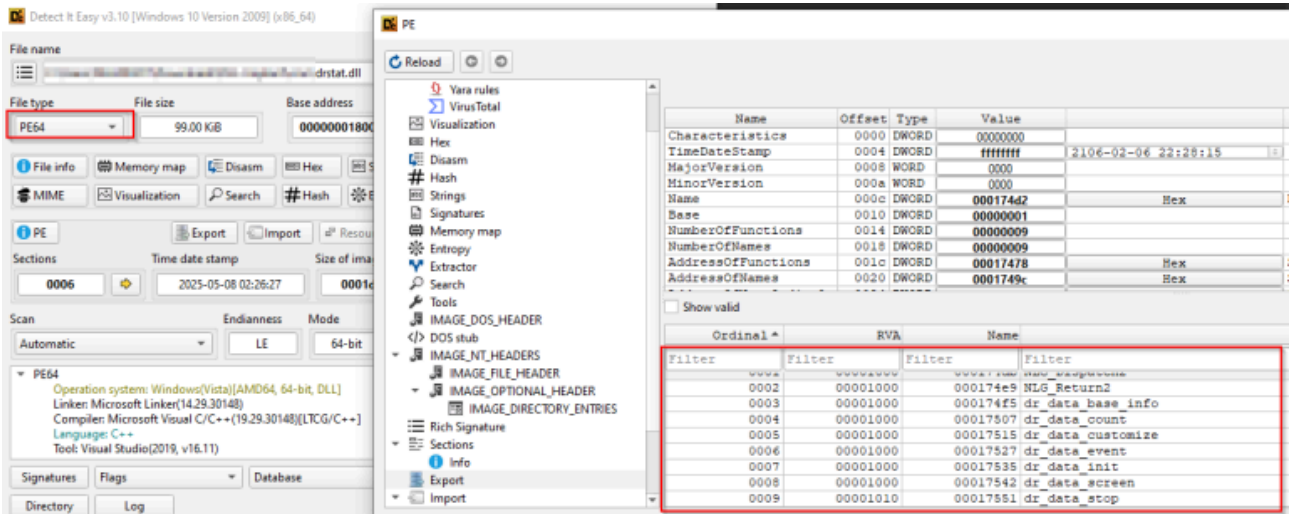
Finally, we could confirm, that the threat entity used the same file, which is available for download from [Wondershare's official website](#). Looking into this code-signing maneuver from Wondershare, and post-analyzing this malicious we can confirm that the threat actor used DLL-Sideloadng against the target to launch the implant, which we have decided to term as VELETRIX .



Before, diving into the next section, we also confirm that the other code signing certificate packed into this compressed executable by ‘Shenzhen Thunder Networking Technologies Ltd’ has frequently been associated with malicious executables in various reports and discussions as abused by Chinese-origin threat entities.

### Stage 1 – Malicious VELETRIX Implant.





Initially, looking into the implant, we figured out a few basic information about the implant, that is it is a 64-bit binary along with which it contains a few interesting export functions. Next, we will focus on the code analysis of this malicious implant.

```

_int64 dr_data_stop()
{
    __int64 v0; // rbx
    HMODULE LibraryA; // rax
    HMODULE v2; // rbx
    FARPROC ProcAddress; // rax
    __int64 (__fastcall *v4)(_QWORD, _QWORD, _QWORD, _QWORD, _DWORD, _DWORD, _QWORD, _QWORD, _QWORD); // r12
    FARPROC v5; // rax
    FARPROC v6; // rax
    HMODULE v7; // rax
    FARPROC v8; // r13
    FARPROC v9; // r15
    HMODULE v10; // rax
    FARPROC v11; // r14
    volatile signed_int8 *v12; // rsi
    char **v13; // rbx
    volatile signed_int8 *v14; // rdi
    unsigned int v15; // eax
    HANDLE ProcessHeap; // rax
    __int64 v17; // rax
    __int64 v18; // rdi
    HANDLE CurrentProcess; // rax
    unsigned int v20; // ebx
    __int64 v21; // rax
    _BYTE v22; // rax
    __int64 v23; // rdx
    __int64 v24; // r8
    FARPROC v26; // [rsp+30h] [rbp-D0h]
    void (__fastcall *v27)(_QWORD, _QWORD, __int64, char *); // [rsp+30h] [rbp-D0h]
    FARPROC v28; // [rsp+38h] [rbp-C8h]
    void (__fastcall *v29)(_QWORD, __int64, __int64, __int64); // [rsp+38h] [rbp-C8h]
    void *v30; // [rsp+40h] [rbp-C6h] BYREF
    char v31[4]; // [rsp+48h] [rbp-B8h] BYREF
    char v32[12]; // [rsp+4Ch] [rbp-B4h] BYREF
}

```

Upon checking into all the exports, out of all the exports, we found dr\_data\_stop to be the one containing interesting malicious code.

```

if ( byte_180019B90 != 1 )
{
    GetTickCount();
    v0 = 10i64;
    do
    {
        Sleep(0x3E8u);
        Beep(1u, 50u);
        --v0;
    }
    while ( v0 );
    GetTickCount();
}

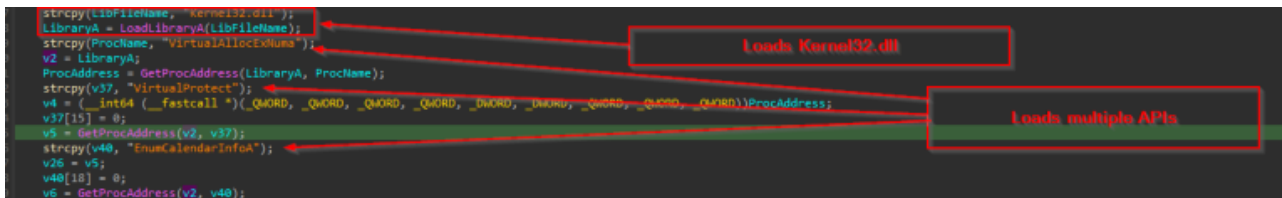
```

Initially, the implant starts with a little anti-analysis trick, which uses a combination of [Sleep](#) & [Beep](#) Windows API, which basically runs inside a do-while loop, which basically runs inside a do-while loop that **delays execution for ~10 seconds and plays a Beep noise to evade automated sandbox analysis**. The loop sleeps for 1

second and beeps 10 times, this entire mechanism is caused to delay the analysis of the analyst or confuse the automated sandbox.

**This technique leverages NtDelayExecution at the system level** – Beep internally call NtDelayExecution, which accepts a “DelayInterval” parameter specifying milliseconds to delay. When executed, NtDelayExecution pauses the calling thread, which causes sandbox timeouts or loss of debugger control making it a **not so harmful, yet effective anti-sandbox technique**. The Beep API is particularly clever because it serves dual purposes: creating execution delays through its internal NtDelayExecution calls while also generating audio artifacts that may trigger different behavior in analysis environments or alert researchers to active code execution.

```
strcpy(LibFileName, "kernel32.dll");
LibraryA = LoadLibraryA(LibFileName);
strcpy(ProcName, "VirtualAllocExNuma");
v2 = LibraryA;
ProcAddress = GetProcAddress(LibraryA, ProcName);
strcpy(v37, "VirtualProtect");
v4 = (__int64 (__fastcall*)(_QWORD, _QWORD, _QWORD, _QWORD, _DWORD, _QWORD, _QWORD))ProcAddress;
v37[15] = 0;
v5 = GetProcAddress(v2, v37);
strcpy(v40, "EnumCalendarInfoA");
v38 = v5;
v40[18] = 8;
v6 = GetProcAddress(v2, v40);
```



Then, it moves ahead with loading kernel32.dll, further once the DLL is being loaded using [LoadLibraryA](#), once the DLL is loaded, further [GetProcAddress](#) is used to resolve some interesting set of APIs, which are VirtualAllocExNuma, VirtualProtect & EnumCalendarInfo.

```
strcpy(v36, "Advapi32.dll");
v28 = v6;
v7 = LoadLibraryA(v36);
strcpy(v38, "SystemFunction036");
strcpy(v34, "HeapAlloc");
v8 = GetProcAddress(v7, v38);
strcpy(v33, "HeapFree");
v9 = GetProcAddress(v2, v34);
GetProcAddress(v2, v33);
```



Similarly, it loads the ADVAPI32.dll and once the DLL is loaded, it resolves using the same technique, which are SystemFunction036, HeapAlloc and HeapFree.

```
strcpy(v32, "ntdll");
v32[6] = 0;
v10 = LoadLibraryA(v32);
strcpy(v41, "RtlIpv4StringToAddressA");
v11 = GetProcAddress(v10, v41);
```

Finally, the ntdll.dll is loaded, and an interesting Windows API is resolved which is known as RtlIpv4StringToAddressA.

```

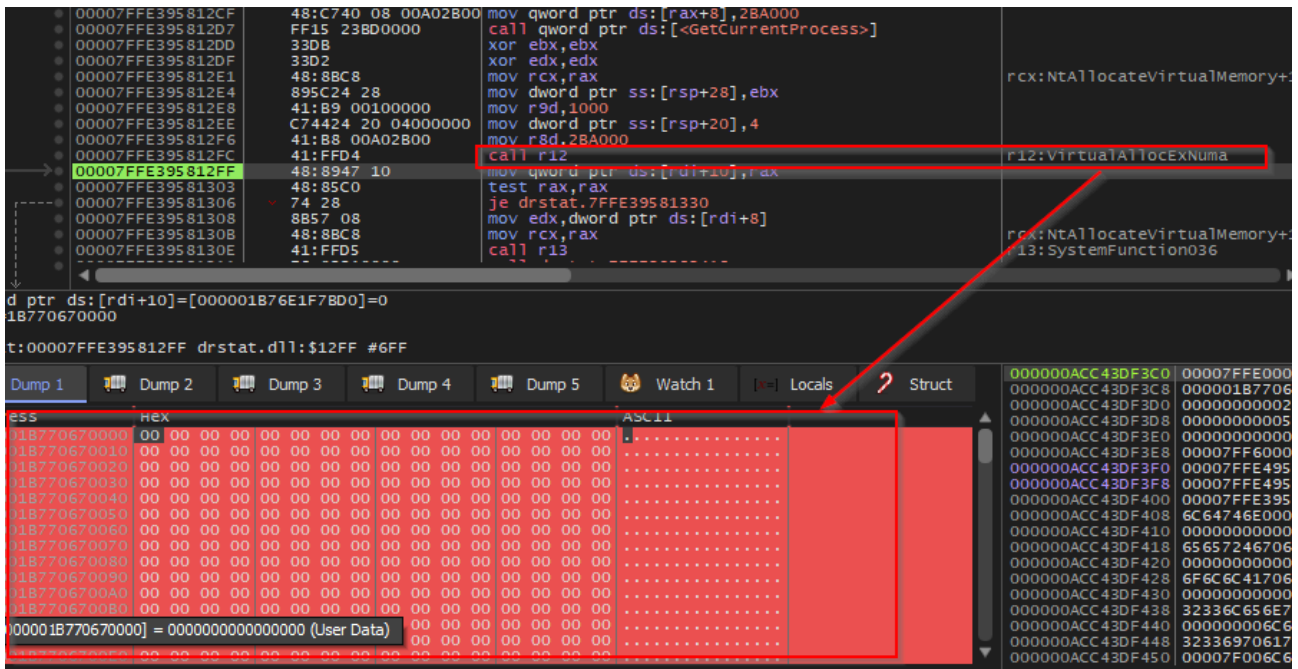
.rdata:0000000180015908 dq offset a6355255255 ; "63.55.255.255"
.rdata:0000000180015908 dq offset a255134108111 ; "255.134.108.111"
.rdata:0000000180015908 dq offset a111111163163 ; "111.111.163.163"
.rdata:0000000180015908 dq offset a163475860 ; "163.47.58.60"
.rdata:0000000180015908 dq offset a57564659 ; "57.56.46.59"
.rdata:0000000180015908 dq offset a46584657 ; "46.58.46.57"
.rdata:0000000180015908 dq offset a465639226 ; "46.56.39.226"
.rdata:0000000180015908 dq offset a195757146 ; "195.75.7.146"
.rdata:0000000180015908 dq offset a14414439238 ; "144.144.39.238"
.rdata:0000000180015908 dq offset a111247108111 ; "111.247.108.111"
.rdata:0000000180015908 dq offset a1112143524 ; "111.214.35.24"
.rdata:0000000180015908 dq offset a7310413565 ; "73.104.135.65"
.rdata:0000000180015A00 dq offset a10711111192 ; "107.111.111.92"
.rdata:0000000180015A08 dq offset a14416842255 ; "144.168.42.255"
.rdata:0000000180015A10 dq offset a26281029 ; "26.28.10.29"
.rdata:0000000180015A18 dq offset a3922818347 ; "39.228.183.47"
.rdata:0000000180015A20 dq offset a2311824539 ; "231.18.245.39"
.rdata:0000000180015A28 dq offset a22634255168 ; "226.34.255.168"
.rdata:0000000180015A30 dq offset a422519293 ; "42.251.92.93"
.rdata:0000000180015A38 dq offset a65119168 ; "65.11.9.168"
.rdata:0000000180015A40 dq offset a4224733 ; "42.247.3.3"
.rdata:0000000180015A48 dq offset a1441839226 ; "144.188.39.226"
.rdata:0000000180015A50 dq offset a3420716842 ; "34.207.168.42"
.rdata:0000000180015A58 dq offset a207242893 ; "207.24.28.93"
.rdata:0000000180015A60 dq offset a4816842203 ; "48.168.42.203"
.rdata:0000000180015A68 dq offset a92936511 ; "92.93.65.11"
.rdata:0000000180015A70 dq offset a916842199 ; "9.168.42.199"
.rdata:0000000180015A78 dq offset a3347231 ; "3.3.47.231"
.rdata:0000000180015A80 dq offset a18197144188 ; "18.197.144.188"
.rdata:0000000180015A88 dq offset a3922634223 ; "39.226.34.223"
.rdata:0000000180015A90 dq offset a16842232 ; "168.42.232.2"
.rdata:0000000180015A98 dq offset a282512168 ; "28.25.12.168"
    
```

Shellcode converted into IP addresses via IPFuscaton technique.

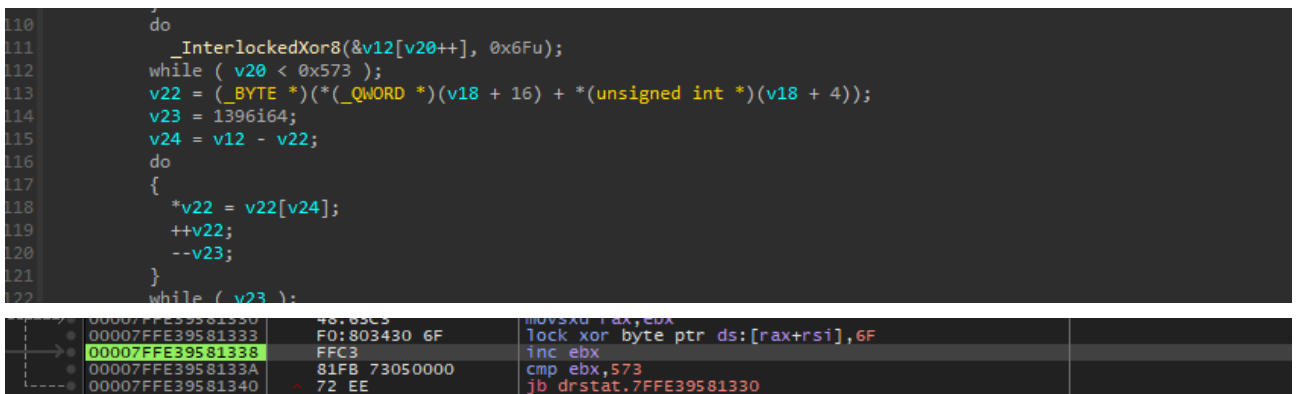
Next, this malicious loader, uses a technique called IPFuscaton, which basically converts the malicious shellcode into a list of IPV4 address.

The screenshot displays a debugger window with assembly code on the left and a memory dump on the right. The assembly code shows instructions for moving registers, calculating addresses, and calling the `RtlIpv4StringToAddressA` function. The memory dump shows a list of memory addresses in hex and their corresponding ASCII values. A red box highlights the instruction `R14=00007FFE4B508D10 (140730161990928d) <ntdll.RtlIpv4StringToAddressA>` in the memory dump, which corresponds to the `call r14, RtlIpv4StringToAddressA` instruction in the assembly code.

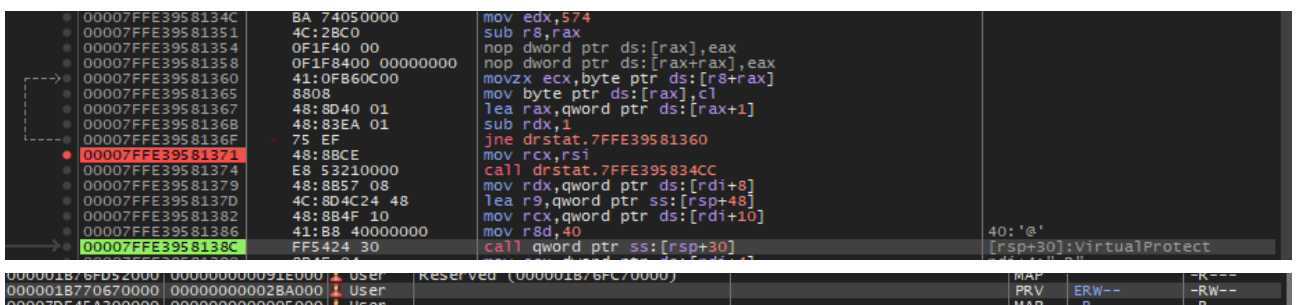
Further, a while-loop along with using the [RtlIpv4StringToAddressA](#) API is used to decode the obfuscated shellcode, which is done by converting the ASCII IP string to binary, where the binary further executes a shellcode.



Once the shellcode is extracted in form of binary, then VirtualAllocExNuma API is used to allocate a fresh memory block with only Read & Write permission into the current process.



Now, once the memory is allocated, further using a simple XOR operation, the encoded blob which was de-obfuscated from the IpFuscation technique via the windows API, is used to further decode via XOR-operation and copied to the allocated memory.



Then, it uses VirtualProtect to change the memory protection of the allocated memory to Execute-Read-Write.

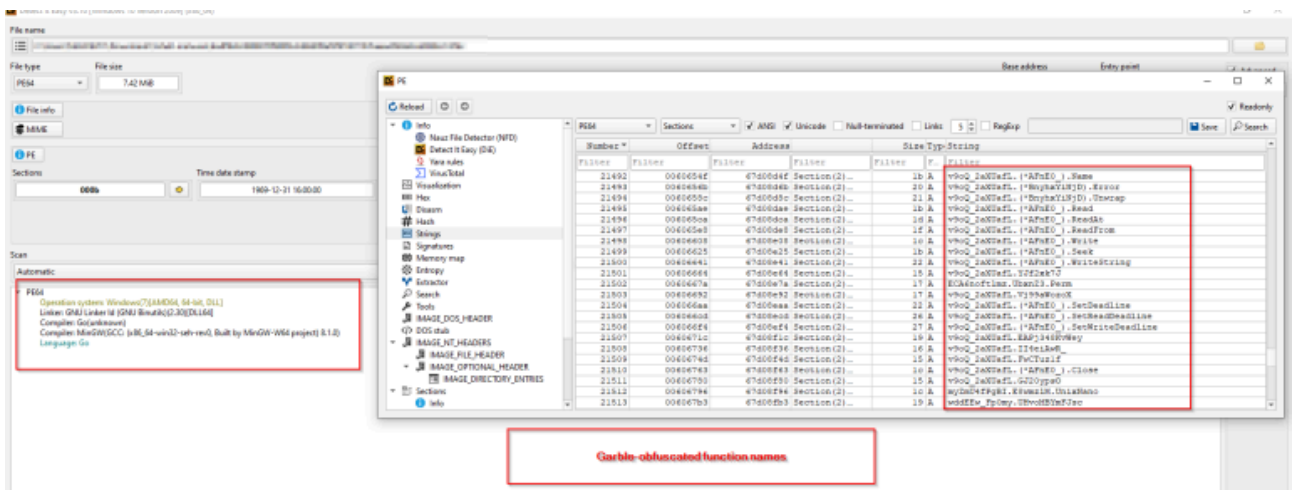
```
C++ Copy  
  
BOOL EnumCalendarInfoA(  
    [in] CALINFO_ENUMPROCA lpCalInfoEnumProc,  
    [in] LCID                Locale,  
    [in] CALID              Calendar,  
    [in] CALTYPE            CalType  
);
```

00007FFE39581379	48:8B57 08	mov rdx,qword ptr ds:[rdi+8]	
00007FFE3958137D	4C:8D4C24 48	lea r9,qword ptr ss:[rsp+48]	
00007FFE39581382	48:8B4F 10	mov rcx,qword ptr ds:[rdi+10]	
00007FFE39581386	41:B8 40000000	mov r8d,40	40: '@'
00007FFE3958138C	FF5424 30	call qword ptr ss:[rsp+30]	[rsp+30]:VirtualProtect
00007FFE39581390	8B4F 04	mov ecx,dword ptr ds:[rdi+4]	rdi+4: "-R"
00007FFE39581393	BA 00080000	mov edx,800	
00007FFE39581398	48:034F 10	add rcx,qword ptr ds:[rdi+10]	
00007FFE3958139C	41:B9 01000000	mov r9d,1	
00007FFE395813A2	41:B8 FFFFFFFF	mov r8d,FFFFFFFF	
00007FFE395813A8	FF5424 38	call qword ptr ss:[rsp+38]	[rsp+38]:EnumCalendarInfoA
00007FFE395813AC	C605 DD870100 01	mov byte ptr ds:[7FFE39599B90],1	
00007FFE395813B3	33C0	xor eax,eax	
00007FFE395813B5	48:8B4D 08	mov rcx,qword ptr ss:[rbp+8]	
00007FFE395813B9	48:33CC	xor rcx,rcx	
00007FFE395813BC	E8 5F000000	call drstat.7FFE39581420	
00007FFE395813C1	4C:8D9C24 10010000	lea r11,qword ptr ss:[rsp+110]	
00007FFE395813C9	49:8B5B 30	mov rbx,qword ptr ds:[r11+30]	
00007FFE395813CD	49:8B73 38	mov rsi,qword ptr ds:[r11+38]	
00007FFE395813D1	49:8B7B 40	mov rdi,qword ptr ds:[r11+40]	
00007FFE395813D5	49:8BE3	mov rsp,r11	
00007FFE395813D8	41:5F	pop r15	r15:RtlAllocateHeap
00007FFE395813DA	41:5E	pop r14	r14:RtlIpv4StringToAddressA
00007FFE395813DC	41:5D	pop r13	r13:SystemFunction036
00007FFE395813DE	41:5C	pop r12	r12:VirtualAllocExNuma
00007FFE395813E0	5D	pop rbp	
00007FFE395813E1	C3	ret	
00007FFE395813E2	CC	int3	
00007FFE395813E3	CC	int3	
00007FFE395813E4	CC	int3	
00007FFE395813E5	CC	int3	
00007FFE395813E6	CC	int3	
00007FFE395813E7	CC	int3	
00007FFE395813E8	CC	int3	
00007FFE395813E9	CC	int3	
00007FFE395813EA	CC	int3	
00007FFE395813EB	CC	int3	
00007FFE395813EC	CC	int3	
00007FFE395813ED	CC	int3	
00007FFE395813EE	CC	int3	

Then, finally, it uses a slightly innovative technique of shellcode execution via callback function, that is by using **EnumCalendarInfoA** API to execute the shellcode. This technique leverages the fact that EnumCalendarInfoA expects a callback function pointer as a parameter – the malware passes its shellcode address as this callback, causing Windows to unknowingly execute the malicious code when the API tries to call what it thinks is a legitimate calendar enumeration function, whereas in our case the shellcode, which is basically an windows implant of the VShell OST framework, is being executed.

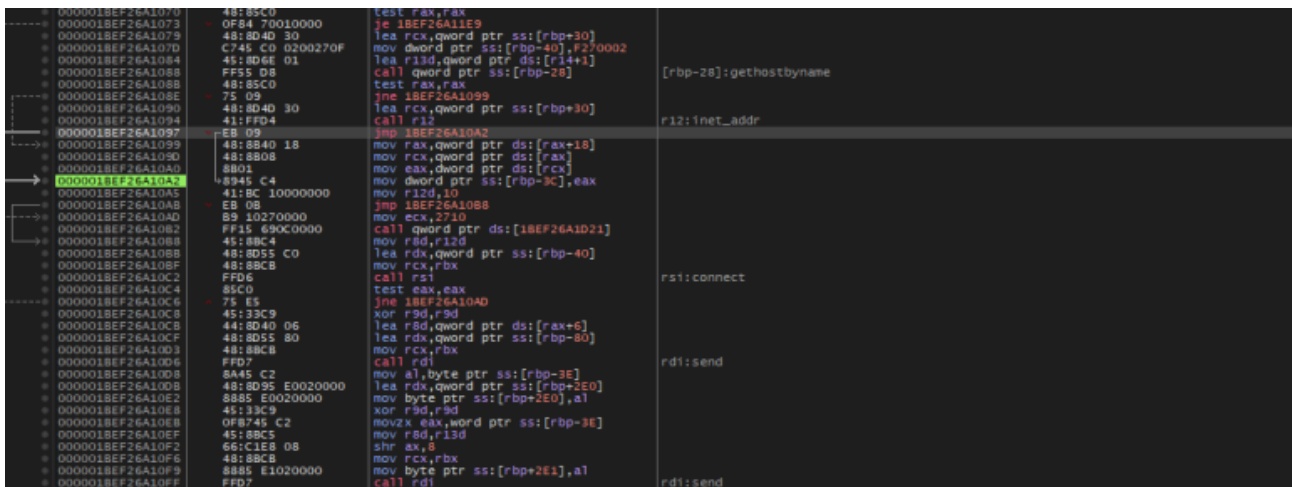
Finally, we can conclude that the Veletrix implant which performs code injection via callback mechanism. In, the next section, we will look into the Vshell implant, which is pretty well known, and look into the workings of it.

### Stage 2 – Malicious Vshell Implant.



Well, [VShell](#), is pretty well-known cross-platform OST framework developed in Golang, initially developed by a researcher, which was later taken-down mysteriously as mentioned in multiple research blogs by various researchers who have tracked various campaigns such as [UNC5174](#) and similar have been used by threat actors originating from Chinese geosphere.

As mentioned, in the previous section VELETRIX loads this windows implant into memory. Looking inside the file, we found that the specific implant, which have been dropped goes by the name `tcp_windows_amd64.dll`. As, this framework is well-researched, we will only look into the key-artefacts and more of a basic overview of the implant.



Upon, looking into the implant, we have multiple functionalities of this implant such as connect, send, receive which is used to interact with the operator. All these functions use underlying code from multiple Windows APIs from WinSock [library](#).

```

000000067A1208E 0F84 7E030000 JE 67A12127
000000067A1208F 48:8850 20 mov rdx,qword ptr ds:[rax+20]
000000067A12090 48:8858 28 mov rdx,qword ptr ds:[rax+28]
000000067A12091 48:8848 10 mov rdx,qword ptr ds:[rax+10]
000000067A12092 48:8878 18 mov rdx,qword ptr ds:[rax+18]
000000067A12093 48:8870 48 mov rdx,qword ptr ds:[rax+48]
000000067A12094 4C:8840 50 mov r8,qword ptr ds:[rax+50]
000000067A12095 4C:8808 08 mov r9,qword ptr ds:[rax]
000000067A12096 4C:8850 08 mov r10,qword ptr ds:[rax+8]
000000067A12097 4C:8860 18 mov r12,qword ptr ds:[rax+18]
000000067A12098 4C:8868 40 mov r13,qword ptr ds:[rax+40]
000000067A12099 4C:8858 60 mov r11,qword ptr ds:[rax+60]
000000067A1209A 4C:893D 9F730E00 lea r15,qword ptr ds:[67A12145]
000000067A1209B 4C:893C24 mov qword ptr ss:[rsp],r15
000000067A1209C 48:8850 08 mov qword ptr ss:[rsp+1],r15
000000067A1209D 4C:896424 10 mov qword ptr ss:[rsp+10],r12
000000067A1209E 4C:896C24 18 mov qword ptr ss:[rsp+18],r13
000000067A1209F 48:89D0 mov rax,rdx
000000067A120A0 EB 785BF5FF call 67890C60
000000067A120A1 48:898424 D8000000 mov qword ptr ss:[rsp+08],rax
000000067A120A2 48:89D8 test rax,rdx
000000067A120A3 0F84 8F000000 JE 67A12185
000000067A120A4 74 04 JE 67A120FC
000000067A120A5 48:8858 08 mov rdx,qword ptr ds:[rax+8]
000000067A120A6 48:89C8 mov rax,rdx
000000067A120A7 31C9 xor ecx,ecx
000000067A120A8 31FF xor edi,edi
000000067A120A9 48:89FE call 6785C0A0
000000067A120AA 48:889424 D8000000 mov rdx,qword ptr ss:[rsp+08]
000000067A120AB 48:89D2 test rdx,rdx
000000067A120AC 74 2A JE 67A12145
000000067A120AD 48:880A mov rdx,qword ptr ds:[rdx]
000000067A120AE 90 nop
000000067A120AF 48:8842 08 mov rax,qword ptr ds:[rdx+8]
000000067A120B0 48:8849 18 mov rax,qword ptr ds:[rdx+18]
000000067A120B1 FF01 call rcx

```

Source	Destination	Protocol	Length	Info
62.234.24.38	10.0.2.15	TCP		60 9999 + 51444 [ACK] Seq=7872 Ack=7997 Win=65535 Len=0
62.234.24.38	10.0.2.15	TCP		60 9999 + 51444 [ACK] Seq=7872 Ack=8053 Win=65535 Len=0
62.234.24.38	10.0.2.15	TCP		97 9999 + 51444 [PSH, ACK] Seq=7872 Ack=8053 Win=65535 Len=43
62.234.24.38	10.0.2.15	TCP		112 9999 + 51444 [PSH, ACK] Seq=7915 Ack=8053 Win=65535 Len=58
10.0.2.15	62.234.24.38	TCP		54 51444 + 9999 [ACK] Seq=8053 Ack=7973 Win=63742 Len=0
10.0.2.15	62.234.24.38	TCP		58 51444 + 9999 [PSH, ACK] Seq=8053 Ack=7973 Win=63742 Len=4
62.234.24.38	10.0.2.15	TCP		60 9999 + 51444 [ACK] Seq=7973 Ack=8057 Win=65535 Len=0
10.0.2.15	62.234.24.38	TCP		89 51444 + 9999 [PSH, ACK] Seq=8057 Ack=7973 Win=63742 Len=35
62.234.24.38	10.0.2.15	TCP		60 9999 + 51444 [ACK] Seq=7973 Ack=8092 Win=65535 Len=0
10.0.2.15	62.234.24.38	TCP		58 51444 + 9999 [PSH, ACK] Seq=8092 Ack=7973 Win=63742 Len=4
62.234.24.38	10.0.2.15	TCP		60 [9999 + 51444 [ACK] Seq=7973 Ack=8096 Win=65535 Len=0
10.0.2.15	62.234.24.38	TCP		112 51444 + 9999 [PSH, ACK] Seq=8096 Ack=7973 Win=63742 Len=58
62.234.24.38	10.0.2.15	TCP		60 9999 + 51444 [ACK] Seq=7973 Ack=8154 Win=65535 Len=0
62.234.24.38	10.0.2.15	TCP		153 9999 + 51444 [PSH, ACK] Seq=7973 Ack=8154 Win=65535 Len=99
10.0.2.15	62.234.24.38	TCP		54 51444 + 9999 [ACK] Seq=8154 Ack=8072 Win=63643 Len=0
10.0.2.15	62.234.24.38	TCP		58 51444 + 9999 [PSH, ACK] Seq=8154 Ack=8072 Win=63643 Len=4
10.0.2.15	62.234.24.38	TCP		89 51444 + 9999 [PSH, ACK] Seq=8158 Ack=8072 Win=63643 Len=35
10.0.2.15	62.234.24.38	TCP		58 51444 + 9999 [PSH, ACK] Seq=8193 Ack=8072 Win=63643 Len=4
10.0.2.15	62.234.24.38	TCP		109 51444 + 9999 [PSH, ACK] Seq=8197 Ack=8072 Win=63643 Len=55
62.234.24.38	10.0.2.15	TCP		60 9999 + 51444 [ACK] Seq=8072 Ack=8158 Win=65535 Len=0
62.234.24.38	10.0.2.15	TCP		60 9999 + 51444 [ACK] Seq=8072 Ack=8193 Win=65535 Len=0
62.234.24.38	10.0.2.15	TCP		60 9999 + 51444 [ACK] Seq=8072 Ack=8197 Win=65535 Len=0

Vshell C2 Config

Further, analyzing we uncovered the command-and-control server along with an import config I.e., the salt which is qwe123qwe . In, the next section, we will look into further, hunting and infrastructural artefacts.

### Hunting and Infrastructure.

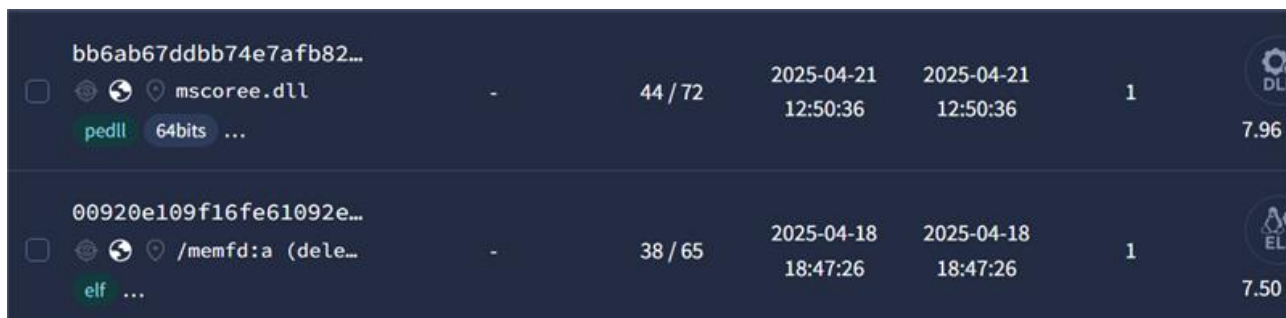
Upon looking into the previous implants, we hunted and found some interesting artefacts.

malware\_config:qwe123qwe

THREATS 0 IOCS 44 REPORTS & ANALYSIS 0 RULES 0 GRAPHS 0 COMMENTS 0

Summary - 40/44 Files	Associations	Detections	First seen	Last seen	Submitters
ba4f9b324809876f906f_	-	25 / 72	2025-05-13 06:39:49	2025-05-27 04:27:21	3
a0f4ee6ea58a8896d291_	-	17 / 66	2025-05-26 08:38:25	2025-05-26 08:38:25	1
59b07d017af08ad29d30_	-	39 / 66	2025-05-26 04:52:49	2025-05-26 04:52:49	1
6566e8788ba9a718786f_	-	36 / 73	2025-05-25	2025-05-25	1

Based on the analysis and extraction of the salt used in the campaign mentioned in this research, we found a total number of 44 implants, using the exact similar salt, that is qwe123qwe. Along, with that as Vshell is a cross-platform tool, we found, multiple EXEs, ELF, DLLs both signed and unsigned.



File Name	Size	Created	Modified	Count	Type
mscoree.dll	7.96 M	2025-04-21 12:50:36	2025-04-21 12:50:36	1	DLL
/memfd:a (dele...)	7.50 M	2025-04-18 18:47:26	2025-04-18 18:47:26	1	ELF

We, also found a few samples whose C2s range from multiple locations such as US, Hong Kong and much more, along with which, we found that a few samples out of 44 implants using same salt, have co-relations with the APT group [Earth Lamia](#) which has targeted Indian entities in few cases. While, upon hunting, we also found, that a lot of similar implants, have multiple overlaps with UNC5174’s campaign abusing [ScreenConnect CVE-2024-1709](#) reported by researchers.

Now, looking into the infrastructural overlaps, the similar indicator has been attributed to the cluster of [China-Nexus-State-Sponsored](#) threat actor which have been abusing CVE-2025-31324 to target SAP NetWeaver Visual Composer.

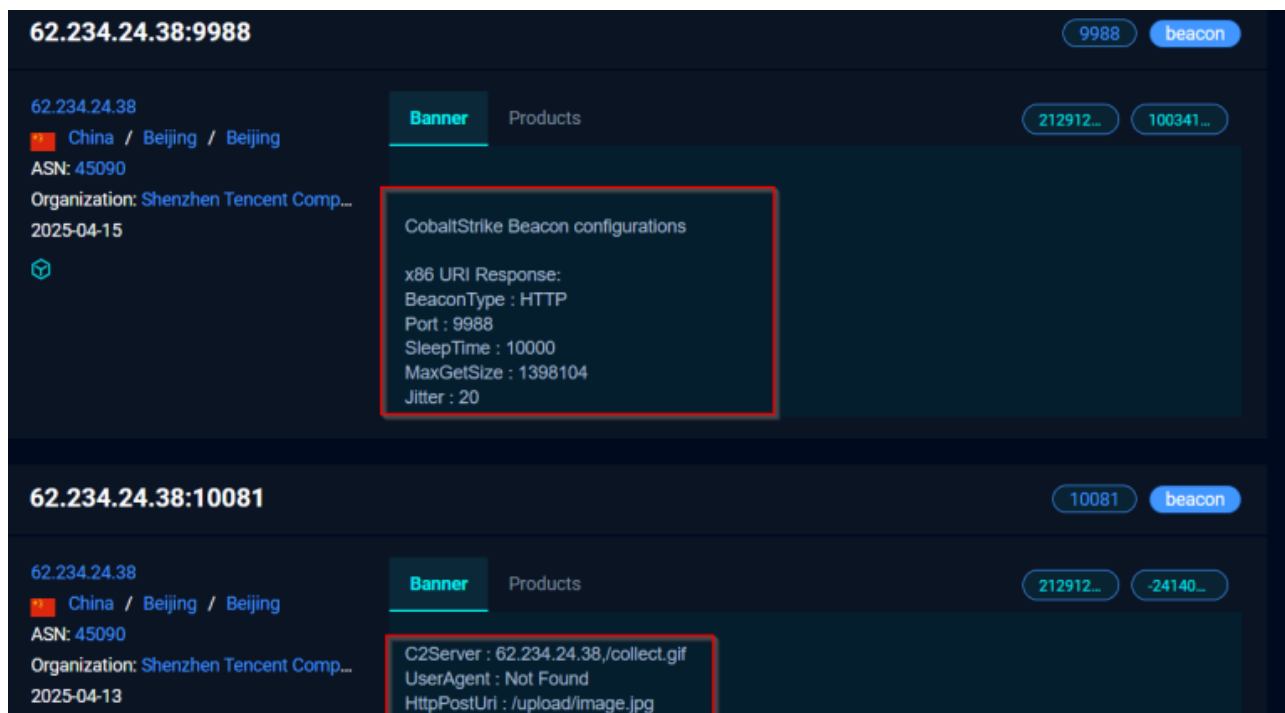


We also found that on the same infrastructure, a login-based webpage has been hosted which is related to the Asset Lighthouse System — an open-source asset discovery and reconnaissance platform developed by [Tophant Competence Center \(TCC\)](#). It is primarily used for mapping external attack surfaces by identifying exposed IPs,

domains, ports, and web services. Therefore, we decided to pivot using these artefacts and found few interesting overlaps.



Post-pivoting, we discovered multiple malicious webservers with similar port-configurations such as running ASL over port 5003, have had hosted Cobalt Strike and SuperShell, which have been known as go-to implants used by *UNC5174 aka Uteus* and along with that we also uncovered multiple webservers with similar port-configurations related to *Earth Lamia*.



Well, the last but not the least, we also saw that the command-and-control server, has also been hosting Cobalt Strike to be used against the targets making it the second post-exploitation framework used by this threat entity.

### **Attribution.**

Through analysis of implant usage and overlapping infrastructure patterns, we identified the threat actor leveraging **VELETRIX**, a relatively new loader designed to execute **VShell** in memory. Although VShell was initially released as an open-source project and later taken down by its original developer, it has since been widely abused by China-aligned threat groups.

Further threat hunting revealed similar behavioral patterns that align with known activity from **UNC5174 (Uteus)** and **Earth Lamia**, as recently documented by researchers. The current infrastructure associated with this actor exhibits consistent use of tools such as **SuperShell**, **Cobalt Strike**, **VShell**, and the **Asset Lighthouse System**—an open-source platform for asset discovery and reconnaissance. These tools have previously been attributed to various China-based APT clusters and observed actively deployed in-the-wild (ITW).

Given the technical and infrastructural overlaps, we assess with high confidence that this threat actor is part of threat entity belong to **China-Nexus cluster**.

### **Conclusion.**

Upon carefully researching the campaign, we found that the China-nexus threat entity which we have termed as Operation DRAGONCLONE has been using DLL-Sideload technique against Wondershare Recoverit software, along with loading VELETRIX DLL implant, which uses interesting techniques such as anti-sandbox, IPFuscation technique along with callback technique to execute Vshell malware, along with having multiple overlaps with UNC5174 and Earth Lamia and the recent campaign have been active since March 2025.

### **Seqrite Protection.**

- AgentCiR

### **IOCs**

SHA-256	Filenames
40450b4212481492d2213d109a0cd0f42de8e813de42d53360da7efac7249df4	附件.zip
ac6e0ee1328cfb1b6ca0541e4dfe7ba6398ea79a300c4019253bd908ab6a3dc0	drstat.dll
645f9f81eb83e52bbbd0726e5bf418f8235dd81ba01b6a945f8d6a31bf406992	drstat.exe
ba4f9b324809876f906f3cb9b90f8af2f97487167beead549a8cddfd9a7c2fdc	tcp_windows_amd64.dll
bb6ab67d8bb74e7afb82bb063744a91f3fecf5fd0f453a179c0776727f6870c7	mscoree.dll
2206cc6bd9d15cf898f175ab845b3deb4b8627102b74e1accefe7a3ff0017112	tcp_windows_amd64.exe
a0f4ee6ea58a8896d2914176d2bfbdb9e16b700f52d2df1f77fe6ce663c1426a	memfd:a(deleted)

### **IP/Domains**

<b>IP</b>
62.234.24.38
47.115.51.44
47.123.7.206

**MITRE ATT&CK**

<b>Tactic</b>	<b>Technique ID</b>	<b>Technique Name</b>	<b>Sub-technique ID</b>	<b>Sub-Technique Name</b>
Reconnaissance	T1595	Active Scanning	T1595.002	Vulnerability Scanning
Reconnaissance	T1588	Obtain Capabilities	T1588.002	Tool
Initial Access	T1566	Phishing	T1566.001	Spear phishing Attachment
Execution	T1204	User Execution	T1204.002	Malicious File.
Persistence				
Defense Evasion	T1140	Deobfuscate/Decode Files or Information		
Defense Evasion	T1574	Hijack Execution Flow	T1574.001	DLL
Defense Evasion	T1027	Obfuscation Files or Information	T1027.007	Dynamic API Resolution
Defense Evasion	T1027	Obfuscation Files or Information	T1027.013	Encrypted/Encoded File
Defense Evasion	T1055	Process Injection		
Defense Evasion	T1497	Virtualization/Sandbox Evasion	T1497.003	Time Based Evasion
Discovery	T1046	Network Service Discovery		

Source: <https://www.seqrte.com/blog/operation-dragonclone-chinese-telecom-veletrix-vshell-malware/>