ACTIVISION
CENTRAL TECH

Remote Administration Payload Malware Analysis Report

May 04, 2023

ACTIVISION | BLIZZARD

SUMMARY

This report covers the malware analysis process of a third stage Remote Administration payload named “InfoTrust.dll” which was delivered as a result of a target of opportunity attack. We sometimes encounter these types of payloads in the course of our work and are tasked with their analysis and reverse engineering. The initial stage of infection was automated, randomly infecting victims across the web, and then sending back initial telemetry such as the victim’s domain name. Once the attacker notices a compromised machine is in a domain of interest, they manually deliver the Remote Administration payload “InfoTrust.dll” with the purpose of gaining complete remote control of the infected machine.

InfoTrust contains 4 payload stages. The initial stage was identified as Blister malware and was simply used to hide additional payloads in a legitimately looking benign software. The final stage is far from being benign – it is a popular commercial adversary simulation tool called Cobalt Strike that is commonly used by Red Teams to hack corporate networks but was stolen and actively used by a wide range of threat actors from ransomware to espionage focused Advanced Persistent Threats (APTs) groups.

This report will cover the journey to the final payload.

Initial Analysis – “InfoTrust.dll”

File Name:	InfoTrust.dll (484 KB)
PDB Path:	C:\UnityCapture\Source\Build\Release-UnityCapturePlugin64bit\UnityCapturePlugin.pdb
Imphash:	f4a69846ab44cc1bedeea23e3b680256
MD5:	06f53d457c530635b34aef0f04c59c7d
SHA-1:	7e30c3aee2e4398ddd860d962e787e1261be38fb
SHA-256:	aeecc65ac8f0f6e10e95a898b60b43bf6ba9e2c0f92161956b1725d68482721d
Compiler:	Microsoft Visual C/C++
Linker:	Microsoft Linker (14.25)[DLL64]
Exports:	CaptureCreateInstance CaptureDeleteInstance CaptureSendTexture UnitySetGraphicsDevice
Imported DLLs:	Kernel32.dll
Entrypoint:	Exported function “CaptureSendTexture”
Command line:	Rundll32.exe C:\ProgramData\InfoTrust\InfoTrust.dll,CaptureSendTexture

https://www.virustotal.com/gui/file/aeecc65ac8f0f6e10e95a898b60b43bf6ba9e2c0f92161956b1725d68482721d

Section	W	X	Raw Offset	Raw Size	Virtual Offset	Virtual Size	Characteristics	Entropy	Heuristics Analysis
.text		Y	0x400	0x11400	0x1000	0x112C0	0x60000020	6.42419	Contains entrypoint
.rdata			0x11800	0xA200	0x13000	0xA0CE	0x40000040	4.96629	
.data	Y		0x1BA00	0xC00	0x1E000	0x1CE0	0xC0000040	2.01607	
.pdata			0x1C600	0x1200	0x20000	0x1074	0x40000040	4.61835	
_RDATA			0x1D800	0x200	0x22000	0x94	0x40000040	1.11788	
.rsrc			0x1DA00	0x5AC00	0x23000	0x5ABA8	0x40000040	7.61368	High entropy (potentially encrypted)
.reloc			0x78600	0x800	0x7E000	0x66C	0x42000040	4.87584	

- **.rsrc** section has high entropy and in fact contains encrypted malware data

Using the PDB path the original source code of the legitimate software used to hide the malware in plain sight was located (https://github.com/schellingb/UnityCapture/blob/master/Source/UnityCapturePlugin.cpp#L74).

Comparing the source code to the disassembled code confirmed it's the same code.

```
72  }
73
74  extern "C" __declspec(dllexport) int CaptureSendTexture(UnityCaptureInstance* c, void* TextureNativePtr, int Timeout, bool UseDoubleBuffering, SharedImageMemory::EResizeMode
    ResizeMode, SharedImageMemory::EMirrorMode MirrorMode, bool IsLinearColorSpace)
75  {
76      if (!c || !TextureNativePtr) return RET_ERROR_PARAMETER;
77      if (g_GraphicsDeviceType != kUnityGfxRendererD3D11) return RET_ERROR_UNSUPPORTEDGRAPHICSDEVICE;
78      if (!c->Sender->SendIsReady()) return RET_WARNING_CAPTUREINACTIVE;
79
80      //Get the active D3D11 context
81      ID3D11DeviceContext* ctx = NULL;
82      g_D3D11GraphicsDevice->GetImmediateContext(&ctx);
83      if (!ctx) return RET_ERROR_UNSUPPORTEDGRAPHICSDEVICE;
84
85      //Read the size and format info from the render texture
86      ID3D11Texture2D* d3dtex = (ID3D11Texture2D*)TextureNativePtr;
87      D3D11_TEXTURE2D_DESC desc = {0};
88      d3dtex->GetDesc(&desc);
```

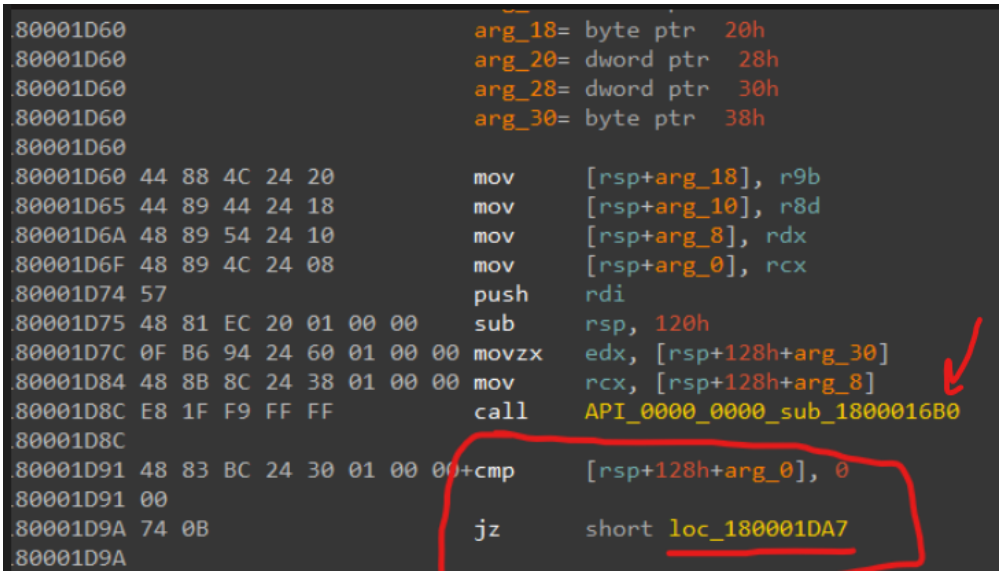
```
30
31  enum
32  {
33      RET_SUCCESS = 0,
34      RET_WARNING_FRAMESKIP = 1,
35      RET_WARNING_CAPTUREINACTIVE = 2,
36      RET_ERROR_UNSUPPORTEDGRAPHICSDEVICE = 100,
37      RET_ERROR_PARAMETER = 101,
38      RET_ERROR_TOOLARGERESOLUTION = 102,
39      RET_ERROR_TEXTUREFORMAT = 103,
40      RET_ERROR_READTEXTURE = 104,
41  };
42
43  #include <d3d11.h>
```

```
.text:0000000180001D60                      arg_18= byte ptr  20h
.text:0000000180001D60                      arg_20= dword ptr  28h
.text:0000000180001D60                      arg_28= dword ptr  30h
.text:0000000180001D60                      arg_30= byte ptr  38h
.text:0000000180001D60
.text:0000000180001D60 44 88 4C 24 20       mov     [rsp+arg_18], r9b
.text:0000000180001D65 44 89 44 24 18       mov     [rsp+arg_10], r8d
.text:0000000180001D6A 48 89 54 24 10       mov     [rsp+arg_8], rdx
.text:0000000180001D6F 48 89 4C 24 08       mov     [rsp+arg_0], rcx
.text:0000000180001D74 57                   push    rdi
.text:0000000180001D75 48 81 EC 20 01 00 00 sub     rsp, 120h
.text:0000000180001D7C 0F B6 94 24 60 01 00 00 movzx   edx, [rsp+128h+arg_30]
.text:0000000180001D84 48 8B 8C 24 38 01 00 00 mov     rcx, [rsp+128h+arg_8]
.text:0000000180001D8C E8 1F F9 FF FF       call    API_0000_0000_sub_1800016B0
.text:0000000180001D8C
.text:0000000180001D91 48 83 BC 24 30 01 00 00+cmp     [rsp+128h+arg_0], 0
.text:0000000180001D91 00
.text:0000000180001D9A 74 0B                jz      short loc_180001DA7
.text:0000000180001D9A
```

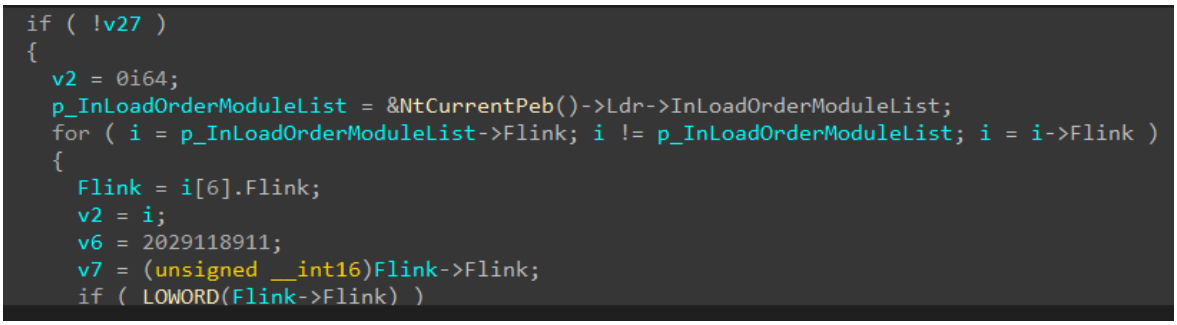
```
.text:0000000180001D9C 48 83 BC 24 38 01 00 00+cmp     [rsp+128h+arg_8], 0
.text:0000000180001D9C 00
.text:0000000180001DA5 75 0A                jnz     short loc_180001DB1
.text:0000000180001DA5
```

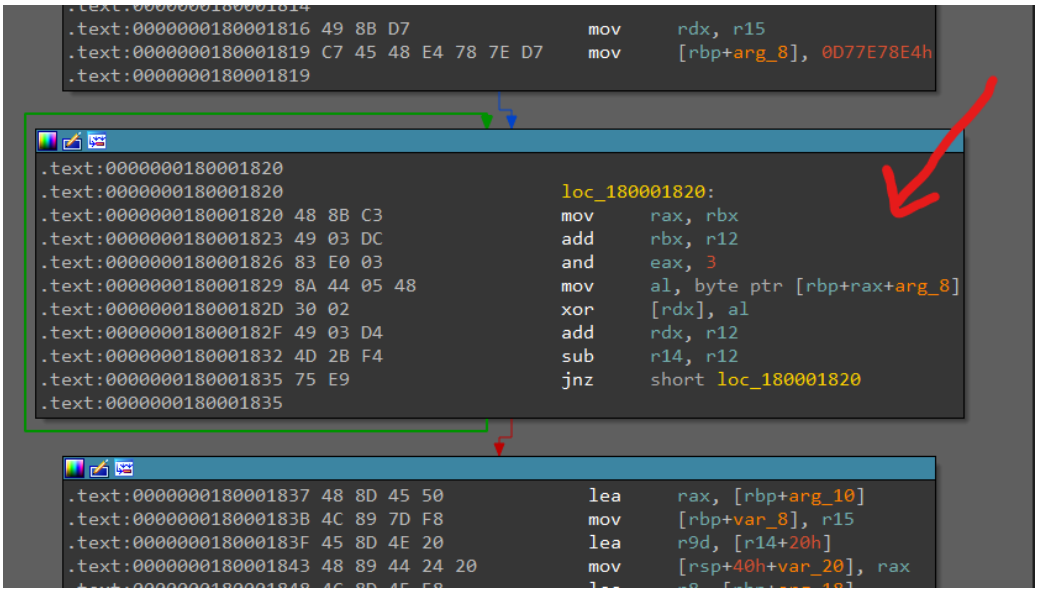
```
.text:0000000180001DA7
.text:0000000180001DA7                      loc_180001DA7:
.text:0000000180001DA7 B8 65 00 00 00       mov     eax, 101
.text:0000000180001DAC E9 BD 05 00 00       jmp     loc_18000236E
.text:0000000180001DAC
```

While comparing the code, a function call was detected in the disassembled code but not in the original source code. This was the malware's entry point hiding itself in plain sight.



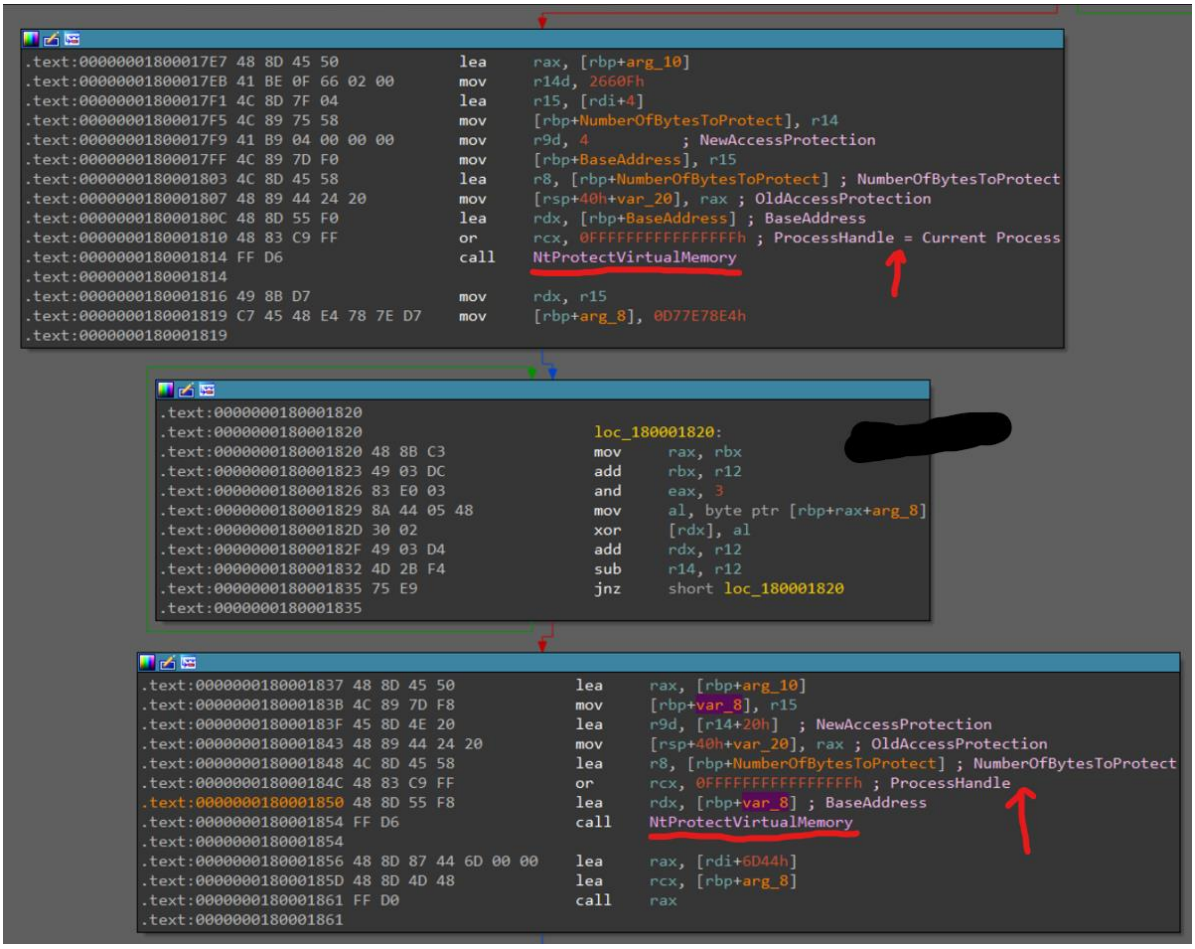
The function showed signs of API usage via "Import by Hash" mechanism, a decryption loop, and a call to an unknown location in the end (assumed to be a decrypted stage 02 payload).





```
    NumberOfBytesToProtecta = 0x2660Fi64;
    BaseAddressa = j + 4;
    NtProtectVirtualMemory(-1i64, &BaseAddressa, (PULONG)&NumberOfBytesToProtecta, PAGE_READWRITE, &v34);
    v26 = j + 4;
    v33 = 0xD77E78E4;
    do
    {
      v27 = v6++;
      *v26++ ^= *((_BYTE *)&v33 + (v27 & 3));
      --v25;
    }
    while ( v25 );
    v30 = j + 4;
    NtProtectVirtualMemory(
      -1i64,                                // current process
      (PVOID *)&v30,
      (PULONG)&NumberOfBytesToProtecta,
      PAGE_EXECUTE_READ,
      &v34);
    ((void (__fastcall *)(int *))(j + 0x6D44))(&v33);
  }
  return 0i64;
}
```

API names hash were broken and allowed annotating the code better, for example:



As well as the decompiler output:

```
NtProtectVirtualMemory(-1i64, &BaseAddressa, (PULONG)&NumberOfBytesToProtecta, PAGE_READWRITE, &v34);
```

To find the address of the encrypted buffer, the malware searches for a signature of bytes: **1A 22 12 71** starting from the return address of the function, and once found they add 4 to skip the signature:

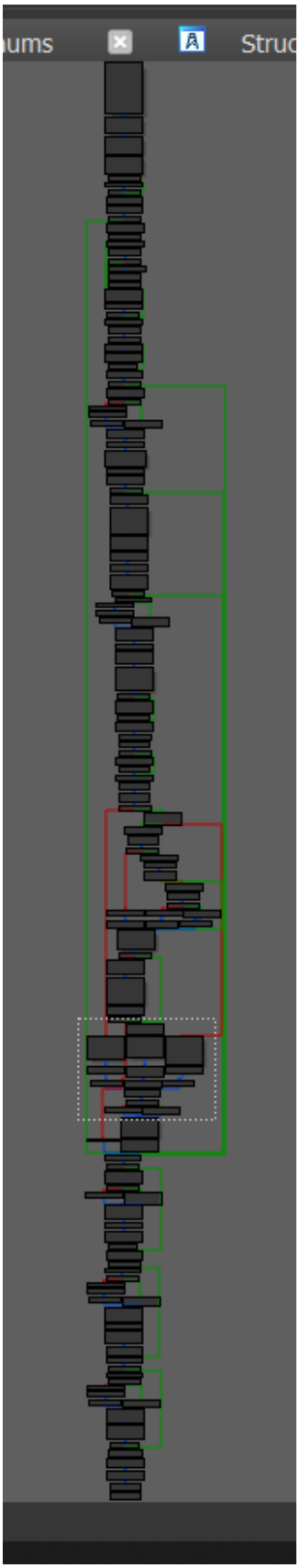
```
for ( j = retaddr; *(_DWORD *)j != 0x711221A; ++j )
  ;
```

```
BaseAddressa = j + 4;
```

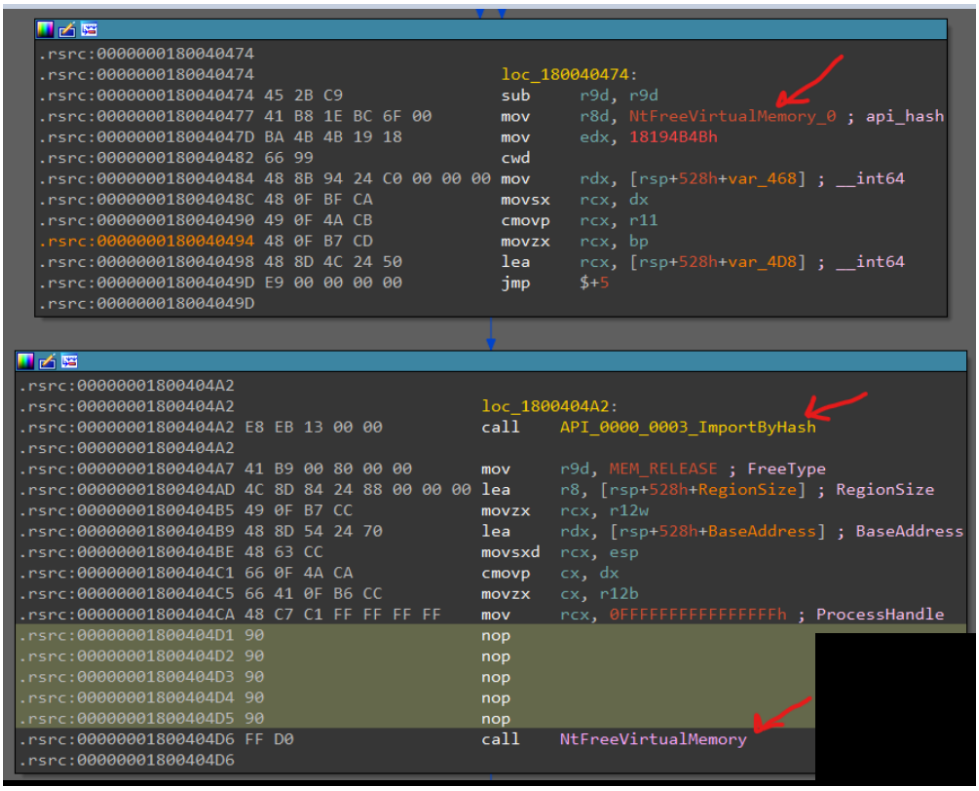
The encrypted buffer was found at 0x180023068 and the size of the buffer is 0x2660F. An IDAPython script was written to decrypt the buffer:

```python
import idaapi

NumberOfBytesToProtect = 0x2660F
BaseAddress = 0x180023068

v26 = BaseAddress
v33 = [0xE4, 0x78, 0x7E, 0xD7]
size_left = NumberOfBytesToProtect
v6 = 0
while size_left > 0:
    v27 = v6
    v6 += 1
    v27 &= 3

    encrypted_byte = idaapi.get_byte(v26)
    decrypted_byte = encrypted_byte ^ v33[v27]
    idaapi.patch_byte(v26, decrypted_byte)

    # Move Next
    v26 += 1
    size_left -= 1
```

Once stage 02 is decrypted, the entry-point is called at 0x018003F90A. The decrypted code's graph-view:



An “Import by Hash” function was detected and a script was written to allow finding all the API calls and annotate them for better code readability:




```
NtCreateDebugObject --> 0x180035f52
NtDebugActiveProcess --> 0x180036010
NtWaitForDebugEvent --> 0x1800361bb
NtGetContextThread --> 0x180036300 , 0x180036901 , 0x180036afe , 0x1800387b3
NtSetContextThread --> 0x1800364d9 , 0x180036634 , 0x1800369df , 0x1800388c8
NtDelayExecution --> 0x180036c30 , 0x180045593
NtRemoveProcessDebug --> 0x180036ccb , 0x1800371af
NtClose --> 0x180036d87 , 0x180037260 , 0x180037d3b , 0x180037ddd , 0x180037e7c , 0x180039006 , 0x1800390c8 , 0x1800391df , 0x18003927c , 0x18003d767 , 0x180040ee3 , 0x180040f84 , 0x18004562d
NtDebugContinue --> 0x180036fe2
RtlFreeHeap --> 0x1800370b4 , 0x180038f1a , 0x18004513d
CreateProcessInternalW --> 0x180037421
NtQueryInformationProcess --> 0x1800375b6
NtReadVirtualMemory --> 0x180037737 , 0x18004667a , 0x1800476bd , 0x1800477b0 , 0x1800478ac , 0x1800479b7
NtAllocateVirtualMemory --> 0x18003784d , 0x180037a19 , 0x180037bd3 , 0x18003809f , 0x18003c907 , 0x18003e47f , 0x18003e93e , 0x18003ec2a , 0x18003eea2 , 0x18003fb9b , 0x1800425d0 , 0x180046514
NtUnmapViewOfSection --> 0x180037b18 , 0x18003d55b , 0x18003e388 , 0x18004032c , 0x180046d3d
NtTerminateProcess --> 0x180037c7a , 0x180039122 , 0x18003fae6 , 0x1800404fe , 0x1800407c1
NtWriteVirtualMemory --> 0x1800383d6 , 0x180038528 , 0x18003ca01 , 0x18003cd5c , 0x180046990
NtProtectVirtualMemory --> 0x180038638 , 0x18003caff , 0x18003cc77 , 0x18003ce73 , 0x18003dc26 , 0x18003ed35 , 0x180042a34 , 0x180045ffb , 0x180046254 , 0x180046868 , 0x180046a8a
NtResumeThread --> 0x180038b1a
NtFreeVirtualMemory --> 0x180038d66 , 0x180038e76 , 0x18003e2ab , 0x18003fd6c , 0x18004022f , 0x1800403e8 , 0x1800404a2 , 0x180046c48 , 0x180046e0f
NtFlushInstructionCache --> 0x18003cf51 , 0x180046b6e
NtCreateSection --> 0x18003d0dd , 0x180040c67
NtMapViewOfSection --> 0x18003d1fe , 0x18003d3b9 , 0x18003d5fa , 0x180040d96
RtlInitAnsiString --> 0x18003e89a , 0x1800416fc , 0x180041f2f , 0x180042302
RtlAnsiStringToUnicodeString --> 0x18003ea41 , 0x180041785 , 0x180041fbd
RtlDecompressBuffer --> 0x18003ef1e
NtSetInformationProcess --> 0x18003f03e
RtlAllocateHeap --> 0x18003f14a , 0x180044996
SHGetSpecialFolderPathW --> 0x18003f371 , 0x180042ca7 , 0x180043445
RtlInitUnicodeString --> 0x18003f555 , 0x180045a23 , 0x180045a6c , 0x180045ac1 , 0x180045b12 , 0x180045b61 , 0x180045bb5 , 0x1800480d4 , 0x1800482c1 , 0x18004850c , 0x1800486d3
LdrLoadDll --> 0x18003f5ba , 0x1800417e9 , 0x18004201d , 0x180045c78 , 0x180048121 , 0x180048552
LockWorkStation --> 0x18003f620
NtQueryVirtualMemory --> 0x18003f6ee
GetComputerNameExW --> 0x1800405b9
RtlDosPathNameToNtPathName_U --> 0x1800409cd
NtCreateFile --> 0x180040b2f
LdrGetProcedureAddress --> 0x180042386
CreateDirectoryW --> 0x180042d99
CoInitializeEx --> 0x180042dff
CoCreateInstance --> 0x180043c69 , 0x1800441d3 , 0x180044675
SHCreateItemFromParsingName --> 0x180043d2d , 0x180043d9e , 0x1800442a0
Sleep --> 0x1800452e0
CreateThread --> 0x180045357
NtQueryInformationThread --> 0x18004549f
LdrUnloadDll --> 0x180048075 , 0x1800484ad
RtlQueryEnvironmentVariable_U --> 0x180048359 , 0x18004875e
```

For hiding strings from a static analysis, the malware used a technique called “Stack Strings” essentially building strings on the stack via code instead of storing the strings statically in the binary. The following example hides the string “shell32.dll” on the stack and uses it later to load that DLL:

```
0001800483B4 C7 44 24 50 73 00 68 00 mov     dword ptr [rsp+78h+SourceString], 680073h
0001800483BC 66 0F CA                bswap   dx
0001800483BF C7 44 24 54 65 00 6C 00 mov     [rsp+78h+var_24], 6C0065h
0001800483C7 49 0F B7 D0             movzx   rdx, r8w
0001800483CB 48 0F C9                bswap   rcx
0001800483CE C7 44 24 58 6C 00 33 00 mov     [rsp+78h+var_20], 33006Ch
0001800483D6 C7 44 24 5C 32 00 2E 00 mov     [rsp+78h+var_1C], 2E0032h
0001800483DE 49 63 C9                movsxd  rcx, r9d
0001800483E1 C7 44 24 60 64 00 6C 00 mov     [rsp+78h+var_18], 6C0064h
0001800483E9 C7 44 24 64 6C 00 00 00 mov     [rsp+78h+var_14], 6Ch ; 'l'
0001800483F1 66 0F 49 CB             cmovns  cx, bx
0001800483F5 0F 9C C6                setl    dh
```

Stage 02 is identified as part of the Blister malware, and online documentation (made by other researchers) shows the following configuration structure for the malware (which isn’t guaranteed to be the same):

```
struct Config {
  uint16_t flag;
  uint32_t payload_export_hash;
  wchar_t w_payload_filename_and_cmdline[783];
  size_t compressed_data_size;
  size_t uncompressed_data_size;
  uint8_t pe_deciphering_key[16];
  uint8_t pe_deciphering_iv[8];
};
```

A couple of struct members suggest encryption and compression are involved when it comes to unprotecting the next payload (stage 3). The original payload was assumed to be compressed before it was encrypted, since lower entropy data can compress better than higher entropy data (e.g. post encryption / ciphertext). An API usage of “RtlDecompressBuffer” was found (0x18003EF7C), and MSDN documents very similar arguments to the config members - compressed and uncompressed data sizes:

C++

```
NT_RTL_COMPRESS_API NTSTATUS RtlDecompressBuffer(
  [in]  USHORT CompressionFormat,
  [out] PUCHAR UncompressedBuffer,
  [in]  ULONG  UncompressedBufferSize,
  [in]  PUCHAR CompressedBuffer,
  [in]  ULONG  CompressedBufferSize,
  [out] PULONG FinalUncompressedSize
);
```

The assumption was that the next payload (stage 3) would be decrypted by the time the decompression was happening and that the next payload will be plain visible after the decompression occurs. The assumption was correct, and the next stage will be dumped (using a debugger) and analyzed in the future.

A huge function (0x180037287) that uses the following APIs seemed to be injecting code into another process:

```
CreateProcessInternalW
NtQueryInformationProcess
NtReadVirtualMemory
NtAllocateVirtualMemory
NtUnmapViewOfSection
NtWriteVirtualMemory
NtProtectVirtualMemory
NtGetContextThread  ---> get thread context
NtSetContextThread ---> set it to new execution location (change RIP)
NtResumeThread --> resume the thread
NtFreeVirtualMemory
RtlFreeHeap
NtTerminateProcess
NtClose [redacted]
```

That process seems to be "\windows\system32\WerFault.exe" as there were multiple references to this process name (using stack strings obfuscation) around (and inside) the call to the code injection function.

```
v12[0] = 'I\0W';                        // "WINDIR"
v12[1] = 'D\0N';
v12[2] = 'R\0I';
v12[3] = 0;
v16[0] = 'S\0\\';                       // "\System32\WerFault.exe"
v16[1] = 's\0y';
v16[2] = 'e\0t';
v16[3] = '3\0m';
v16[4] = '\\\02';
v16[5] = 'e\0W';
v16[6] = 'F\0r';
v16[7] = 'u\0a';
v16[8] = 't\0l';
v16[9] = 'e\0.';
v16[10] = 'e\0x';
v16[11] = 0;
```

Attempting to debug the malware to the point where RtlDecompressBuffer is called (to dump the decompressed payload) failed because of a computer/domain name check. Since “InfoTrust.dll” was manually delivered after the attackers identified the victim was in a domain of interest, an anti-debugging mechanism was inserted automatically as a part of the payload delivery – a custom hash implementation of the targeted victim’s domain name was hardcoded into the payload to be verified upon execution. If the hashes don’t match, the execution ends.

```
if ( (a3 & 0x100000) != 0 && verify_computername_domain_hash((__int64)a1, *(_DWORD *)(a2 + 4))
  || (a3 & 0x1000) != 0
  && ((a3 & 0x2000) == 0 ? (v6 = 600000) : (v6 = *(_DWORD *)(a2 + 0xC)), (unsigned int)sub_180045282(a1, v6) == 1) )
{
  NtTerminateProcess = (void (__fastcall *)(__int64, _QWORD))import_by_hash(a1, a1[1], 0xA2D59EA9i64, 0i64);
  NtTerminateProcess(-1i64, 0i64);
}
```

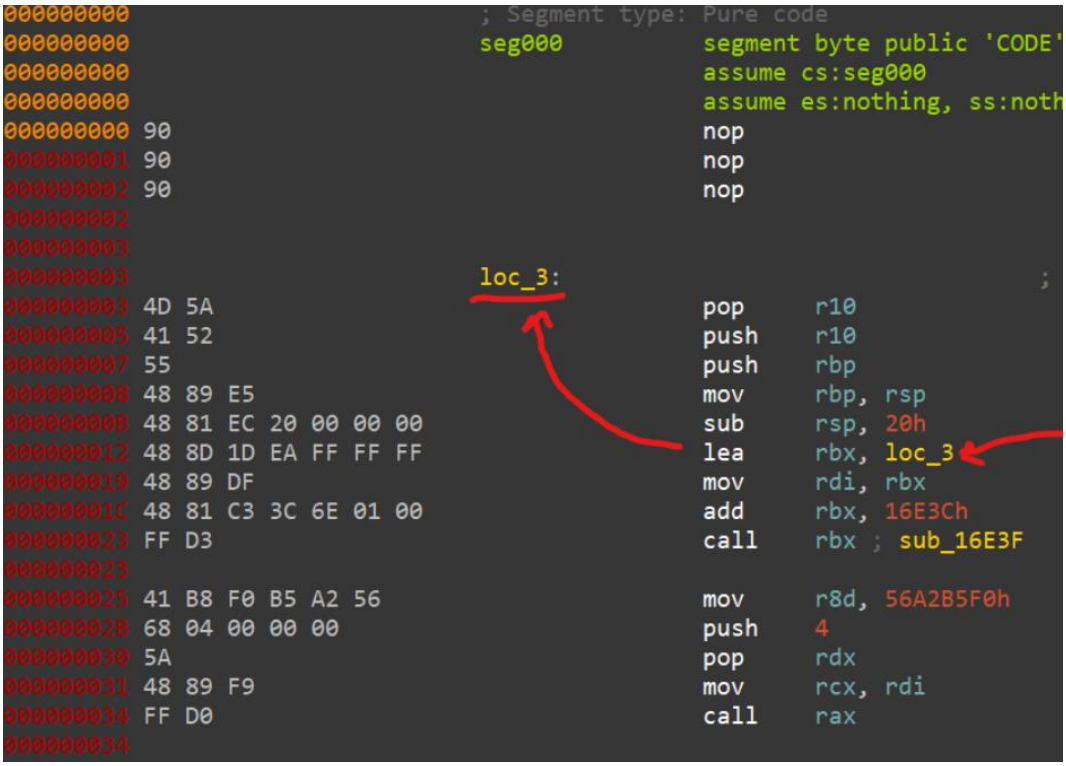
```
C++

BOOL GetComputerNameExW(
  [in]      COMPUTER_NAME_FORMAT NameType,
  [out]     LPWSTR               lpBuffer,
  [in, out] LPDWORD              nSize
);
```

```
1  _BOOL8 __fastcall verify_computername_domain_hash(__int64 a1, int a2)
2  {
3    __int64 v2; // rax
4    __int64 (__fastcall *GetComputerNameExW)(__int64, _WORD *, int *); // rax
5    int v5; // [rsp+0h] [rbp-228h] BYREF
6    int v6; // [rsp+4h] [rbp-224h]
7    _WORD v_computer_name[268]; // [rsp+10h] [rbp-218h] BYREF
8
9    v5 = 522;
10   sub_1800339C4((__int64)v_computer_name, 0, 0x20Ai64);
11   v2 = image_base_by_hash_q(a1, kernel32_dll);
12   GetComputerNameExW = (__int64 (__fastcall *)(__int64, _WORD *, int *))import_by_hash(
13                                                                         a1,
14                                                                         v2,
15                                                                         GetComputerNameExW_0,
16                                                                         0i64);
17   v6 = GetComputerNameExW(ComputerNameDnsHostname, v_computer_name, &v5);
18   return v6 != 1 || (int)sub_18003359E(v_computer_name) <= 0 || (unsigned int)hash(v_computer_name) != a2;
19 }
```

```
1  __int64 __fastcall hash(_WORD *a1)
2  {
3    __int16 _CX; // cx
4    unsigned int v3; // [rsp+0h] [rbp-18h]
5
6    v3 = 0x78F1E5BF;
7    while ( *a1 )
8    {
9      _CX = *a1 ^ v3;
10     __asm { rcl     cl, 56h }
11     v3 = ((0x5BD1E995 * ((unsigned __int16)*a1 ^ v3)) >> 15) ^ (0x5BD1E995 * ((unsigned __int16)*a1 ^ v3));
12     ++a1;
13   }
14   return v3;
15 }
```

This anti-debugging protection was patched to never fail the check.

Breaking at the call to RtlDecompressBuffer allowed dumping of the next payload (stage 3) and a quick initial analysis showed it was a Position Independent Code. This means the payload can be injected into any memory address and it would always work. This can be seen below as RBX is assigned with a RIP-relative address which points to the start of the payload, essentially an offset relative to the current address in the RIP register.



The payload (shellcode) was built into a full Portable Executable (DLL) binary to help the analysis. VirusTotal identified the DLL as a part of Cobalt Strike.

Since Cobalt Strike has been thoroughly analyzed by security researchers, a config-extractor (written by "Sentinel One") was found and used to extract its configuration from the payload (stage 3):

```
python3 parse_beacon_config.py bleh.bin
BeaconType                      - HTTPS
Port                            - 443
SleepTime                       - 25000
MaxGetSize                      - 2097152
Jitter                          - 50
MaxDNS                          - Not Found
PublicKey_MD5                   - cb8f251c467c9411b6b9cb9205e1385d
C2Server                        - albertonne.com,/safebrowsing/d4alBmGBO/HafYg4QZaRhMBwuLAjVmSPc
UserAgent                       - Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36
HttpPostUri                     - /safebrowsing/szKmS/5UeqnouSMfpwYvPeTAAn01
Malleable_C2_Instructions       - Empty
HttpGet_Metadata                - ConstHeaders
                                      Host: domain.com
                                      Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
                                      Accept-Language: en-US,en;q=0.5
                                  Metadata
                                      base64url
                                      prepend "REF=ID=kZdePVW"
                                      header "Cookie"
HttpPost_Metadata               - ConstHeaders
                                      Host: domain.com
                                      Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
                                      Accept-Language: en-US,en;q=0.5
                                  SessionId
                                      base64url
                                      prepend "U=kZdePVW"
                                      prepend "REF=ID="
                                      header "Cookie"
                                  Output
                                      print
PipeName                        - Not Found
DNS_Idle                        - Not Found
DNS_Sleep                       - Not Found
SSH_Host                        - Not Found
SSH_Port                        - Not Found
SSH_Username                    - Not Found
SSH_Password_Plaintext          - Not Found
SSH_Password_Pubkey             - Not Found
SSH_Banner                      - Host: albertonne.com

HttpGet_Verb                    - GET
HttpPost_Verb                   - POST
HttpPostChunk                   - 0
Spawnto_x86                     - %windir%\syswow64\WerFault.exe
Spawnto_x64                     - %windir%\sysnative\WerFault.exe
CryptoScheme                    - 0
Proxy_Config                    - Not Found
Proxy_User                      - Not Found
Proxy_Password                  - Not Found
Proxy_Behavior                  - Use IE settings
Watermark_Hash                  - OQYcMKBK1D9eJxkDC51zwg==
Watermark                       - 1101991775
bStageCleanup                   - True
bCFGCaution                     - False
KillDate                        - 0
bProcInject_StartRWX            - True
bProcInject_UseRWX              - False
bProcInject_MinAllocSize        - 49064
ProcInject_PrependAppend_x86    - b'\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90'
                                  Empty
ProcInject_PrependAppend_x64    - b'\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90'
                                  Empty
ProcInject_Execute              - ntdll.dll:RtlUserThreadStart
                                  NtQueueApcThread-s
                                  SetThreadContext
                                  CreateRemoteThread
                                  kernel32.dll:LoadLibraryA
                                  RtlCreateUserThread
ProcInject_AllocationMethod     - VirtualAllocEx
```

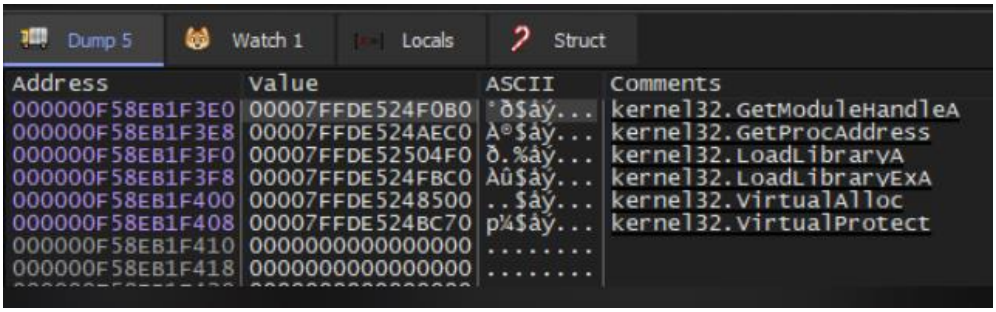
```
ProcInject_AllocationMethod      - VirtualAllocEx
bUsesCookies                     - True
HostHeader                       - Host: albertonne.com

headersToRemove                  - Not Found
DNS_Beaconing                    - Not Found
DNS_get_TypeA                    - Not Found
DNS_get_TypeAAAA                 - Not Found
DNS_get_TypeTXT                  - Not Found
DNS_put_metadata                 - Not Found
DNS_put_output                   - Not Found
DNS_resolver                     - Not Found
DNS_strategy                     - round-robin
DNS_strategy_rotate_seconds      - -1
DNS_strategy_fail_x              - -1
DNS_strategy_fail_seconds        - -1
Retry_Max_Attempts               - 0
Retry_Increase_Attempts          - 0
Retry_Duration                   - 0
```

albertonne[.]com was identified as the Command & Control server for Cobalt Strike.

Analyzing the payload shows an import address table is being manually/dynamically built with the following format:

```
typedef struct
{
QWORD* GetModuleHandleA;
QWORD* GetProcAddress;
QWORD* LoadLibraryA;
QWORD* LoadLibraryExA;
QWORD* VirtualAllocA;
QWORD* VirtualProtect;
}import_table;
```

```
switch ( v6 )
{
  case LoadLibraryA_1:
    import_table_base->LoadLibraryA = (QWORD *)((char *)v5 + *v7);
    break;
  case GetProcAddress_1:
    import_table_base->GetProcAddress = (QWORD *)((char *)v5 + *v7);
    break;
  case VirtualAlloc_1:
    import_table_base->VirtualAllocA = (QWORD *)((char *)v5 + *v7);
    break;
  case VirtualProtect_1:
    import_table_base->VirtualProtect = (QWORD *)((char *)v5 + *v7);
    break;
  case LoadLibraryExA_1:
    import_table_base->LoadLibraryExA = (QWORD *)((char *)v5 + *v7);
    break;
  default:
    import_table_base->GetModuleHandleA = (QWORD *)((char *)v5 + *v7);
    break;
}
--v4;
```



Assembly code is nicely annotated:

```
10  v8 = 0i64;
11  if ( (*(_WORD *)(a2 + 22) & 0x4000) == 0x4000
12    && !((__int64 (__fastcall *)(__int64))a_import_table->GetModuleHandleA)(a3 + 0x44) )
13  {
14    v10 = ((__int64 (__fastcall *)(__int64, _QWORD, __int64))a_import_table->LoadLibraryExA)(a3 + 68, 0i64, 1i64);
15    v9 = a3 + 64;
16    v6 = *(_DWORD *)(a3 + 64) + 15;
17    for ( i = *(_DWORD *)(a3 + 64); v10 != -1 && i < v6 && !v8; ++i )
18      v8 = ((__int64 (__fastcall *)(__int64, _QWORD))a_import_table->GetProcAddress)(v10, i);
19    if ( v8 )
20    {
21      v8 -= v8 % 0x1000;
22      ((void (__fastcall *)(unsigned __int64, _QWORD, _QWORD, char *))a_import_table->VirtualProtect)(
23        v8,
24        *(unsigned int *)(a2 + 80),
25        a4,
26        v7);
27    }
28  }
29  if ( !v8 )
30    return ((__int64 (__fastcall *)(_QWORD, _QWORD, __int64, _QWORD))a_import_table->VirtualAllocA)(
31              0i64,
32              *(unsigned int *)(a2 + 80),
33              12288i64,
34              a4);
35  return v8;
36 }
```

```
mov     rax, [rsp+58h+arg_10]
add     rax, 44h ; 'D'
mov     rcx, rax
mov     rax, [rsp+58h+arg_0]
call    qword ptr [rax]

test    rax, rax
jnz     loc_41993E

mov     rax, [rsp+58h+arg_10]
add     rax, 44h ; 'D'
mov     r8d, 1
xor     edx, edx
mov     rcx, rax
mov     rax, [rsp+58h+arg_0]
call    qword ptr [rax+18h]
```

```
mov     rax, [rsp+58h+arg_10]
add     rax, 44h ; 'D'
mov     rcx, rax        ; lpModuleName
mov     rax, [rsp+58h+arg_0]
call    GetModuleHandleA

test    rax, rax
jnz     loc_41993E

mov     rax, [rsp+58h+arg_10]
add     rax, 44h ; 'D'
mov     r8d, 1          ; dwFlags
xor     edx, edx        ; hFile
mov     rcx, rax        ; lpLibFileName
mov     rax, [rsp+58h+arg_0]
call    LoadLibraryExA
```

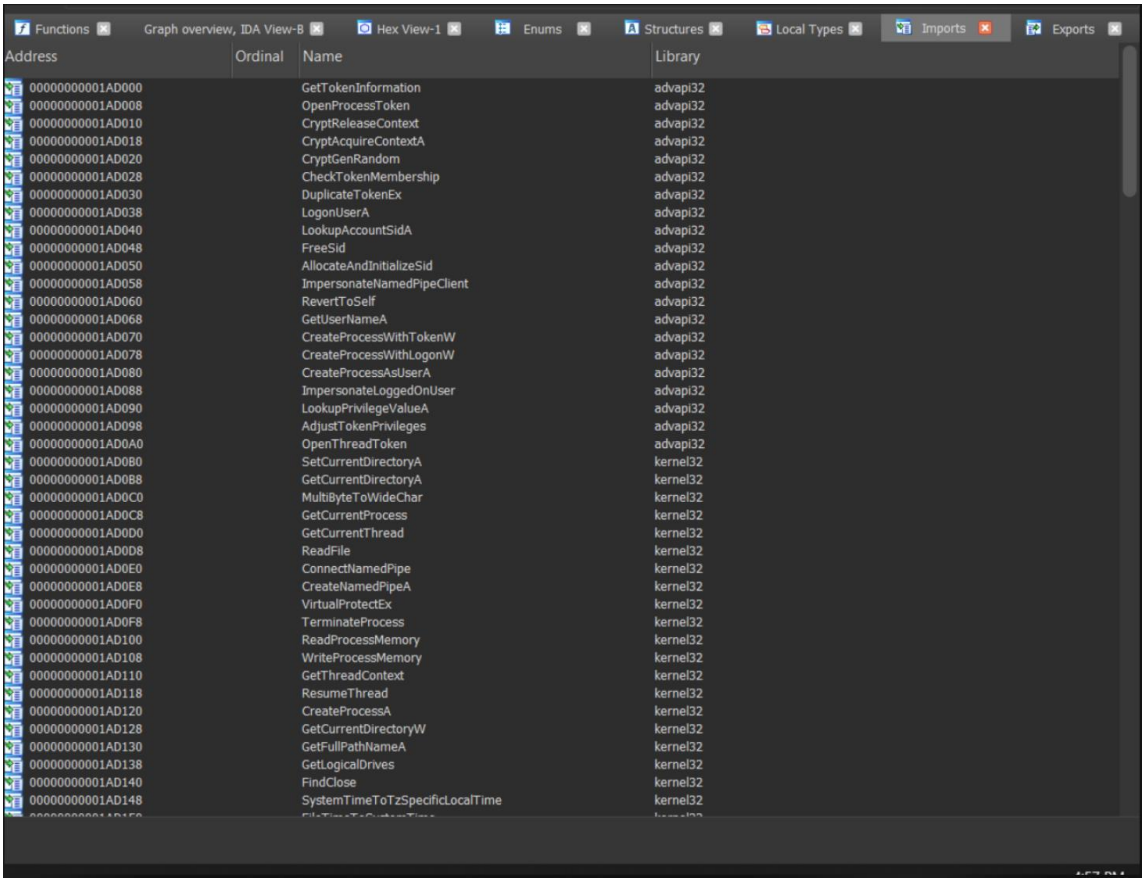
Analyzing the shellcode it was clear it unpacks/prepares the next payload (stage 4) to be called in the end of the shellcode.

```
74  memset(import_table_base, 0, 0x58ui64);
75  if ( (v_process_nt_headers->FileHeader.Characteristics & 0x1000) == 0x1000 )
76    v_AddressOfEntryPoint = v_process_nt_headers->OptionalHeader.LoaderFlags;
77  else
78    v_AddressOfEntryPoint = v_process_nt_headers->OptionalHeader.AddressOfEntryPoint;
79  a4[1] = &v_some_buffer_02[v_AddressOfEntryPoint];
80  ((void (__fastcall *)(BYTE *, __int64, __int64))&v_some_buffer_02[v_AddressOfEntryPoint])(v_some_buffer_02, 1i64, a1);
81 }
```

Using a debugger the final payload was unpacked, merged with the shellcode into one Portable Executable (PE) binary, and import address table was fixed manually to produce a complete clean PE that allowed IDA to identify functions and code patterns to annotate automatically, as well as use Lumina server to pull all the cloud based code annotations made by other researchers.

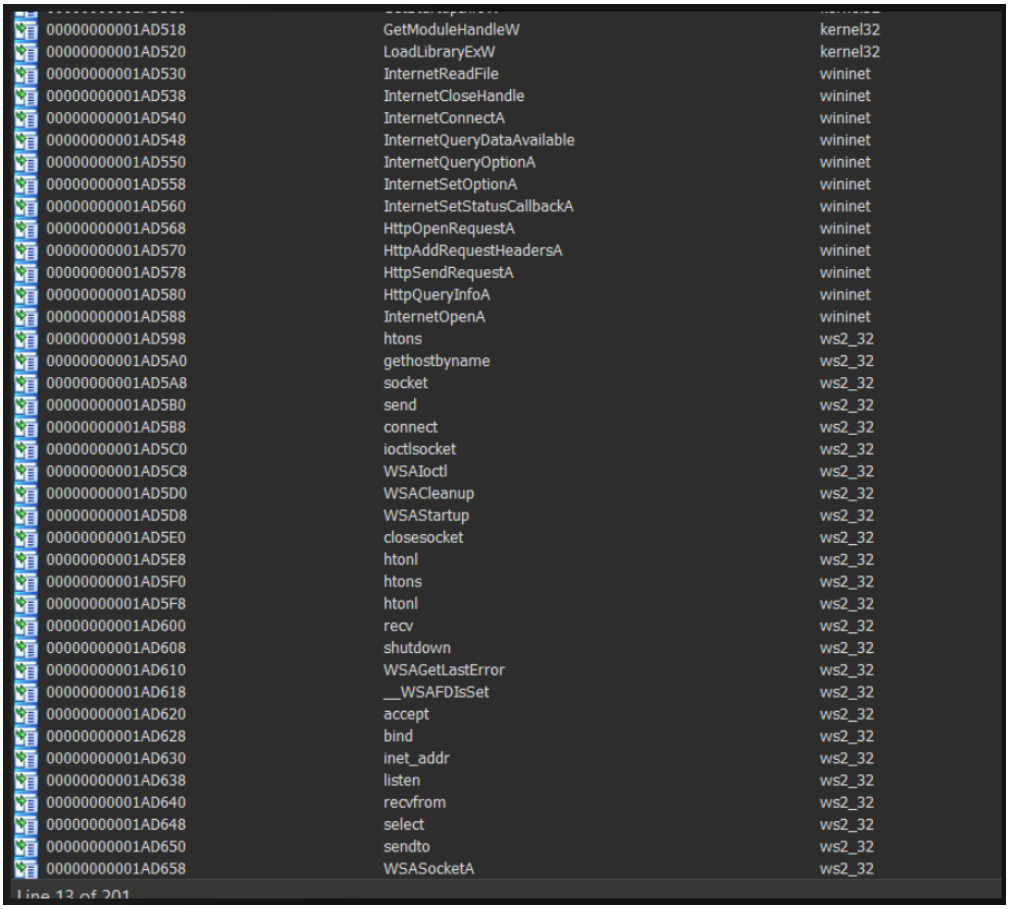
```
stage_04:00000000001A37B4
stage_04:00000000001A37B4
stage_04:00000000001A37B4                       ; Attributes: library function bp-based frame
stage_04:00000000001A37B4
stage_04:00000000001A37B4                       ; void __cdecl _security_init_cookie()
stage_04:00000000001A37B4                       __security_init_cookie proc near
stage_04:00000000001A37B4
stage_04:00000000001A37B4                       arg_0= qword ptr  10h
stage_04:00000000001A37B4                       SystemTimeAsFileTime= _FILETIME ptr  18h
stage_04:00000000001A37B4                       PerformanceCount= LARGE_INTEGER ptr  20h
stage_04:00000000001A37B4                       arg_18= qword ptr  28h
stage_04:00000000001A37B4
stage_04:00000000001A37B4 48 89 5C 24 20        mov     [rsp-8+arg_18], rbx
stage_04:00000000001A37B9 55                    push    rbp
stage_04:00000000001A37BA 48 8B EC              mov     rbp, rsp
stage_04:00000000001A37BD 48 83 EC 20           sub     rsp, 20h
stage_04:00000000001A37C1 48 8B 05 00 D7 01 00  mov     rax, cs:qword_1C0EC8
stage_04:00000000001A37C8 48 83 65 18 00        and     qword ptr [rbp+SystemTimeAsFileTime.dwLowDateTime], 0
stage_04:00000000001A37CD 48 BB 32 A2 DF 2D 99 2B+mov   rbx, 2B992DDFA232h
stage_04:00000000001A37CD 00 00
stage_04:00000000001A37D7 48 3B C3              cmp     rax, rbx
stage_04:00000000001A37DA 75 6F                 jnz     short loc_1A384B
stage_04:00000000001A37DA
```

```
tage_04:00000000001A37DC 48 8D 4D 18            lea     rcx, [rbp+SystemTimeAsFileTime] ; lpSystemTimeAsFileTime
tage_04:00000000001A37E0 FF 15 82 9C 00 00      call    cs:GetSystemTimeAsFileTime
tage_04:00000000001A37E0
tage_04:00000000001A37E6 48 8B 45 18            mov     rax, qword ptr [rbp+SystemTimeAsFileTime.dwLowDateTime]
tage_04:00000000001A37EA 48 89 45 10            mov     [rbp+arg_0], rax
tage_04:00000000001A37EE FF 15 8C 9C 00 00      call    cs:GetCurrentThreadId
tage_04:00000000001A37EE
tage_04:00000000001A37F4 8B C0                  mov     eax, eax
tage_04:00000000001A37F6 48 31 45 10            xor     [rbp+arg_0], rax
tage_04:00000000001A37FA FF 15 90 99 00 00      call    cs:GetCurrentProcessId
tage_04:00000000001A37FA
tage_04:00000000001A3800 48 8D 4D 20            lea     rcx, [rbp+PerformanceCount] ; lpPerformanceCount
tage_04:00000000001A3804 8B C0                  mov     eax, eax
tage_04:00000000001A3806 48 31 45 10            xor     [rbp+arg_0], rax
tage_04:00000000001A380A FF 15 B0 9B 00 00      call    cs:QueryPerformanceCounter
tage_04:00000000001A380A
tage_04:00000000001A3810 8B 45 20               mov     eax, dword ptr [rbp+PerformanceCount]
tage_04:00000000001A3813 48 C1 E0 20            shl     rax, 20h
tage_04:00000000001A3817 48 8D 4D 10            lea     rcx, [rbp+arg_0]
tage_04:00000000001A381B 48 33 45 20            xor     rax, qword ptr [rbp+PerformanceCount]
tage_04:00000000001A381F 48 33 45 10            xor     rax, [rbp+arg_0]
tage_04:00000000001A3823 48 33 C1               xor     rax, rcx
tage_04:00000000001A3826 48 B9 FF FF FF FF FF FF+mov    rcx, 0FFFFFFFFFFFFh
tage_04:00000000001A3826 00 00
tage_04:00000000001A3830 48 23 C1               and     rax, rcx
tage_04:00000000001A3833 48 B9 33 A2 DF 2D 99 2B+mov    rcx, 2B992DDFA233h
tage_04:00000000001A3833 00 00
tage_04:00000000001A383D 48 3B C3               cmp     rax, rbx
tage_04:00000000001A3840 48 0F 44 C1            cmovz   rax, rcx
tage_04:00000000001A3844 48 89 05 7D D6 01 00   mov     cs:qword_1C0EC8, rax
tage_04:00000000001A3844
```



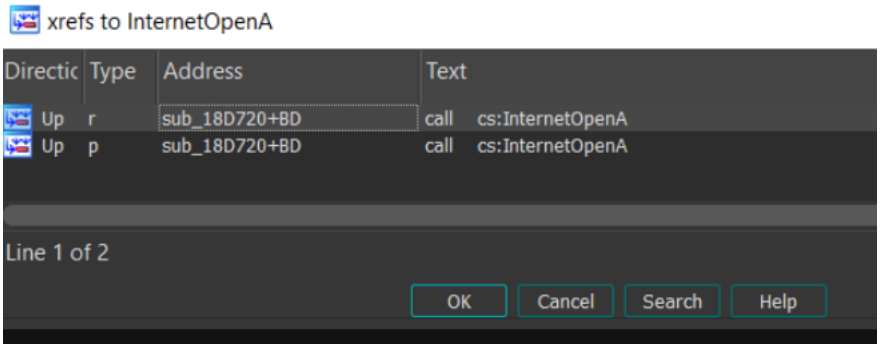
Functions | Graph overview, IDA View-B | Hex View-1 | Enums | Structures | Local Types | Imports | Exports

Address	Ordinal	Name	Library
00000000001AD000		GetTokenInformation	advapi32
00000000001AD008		OpenProcessToken	advapi32
00000000001AD010		CryptReleaseContext	advapi32
00000000001AD018		CryptAcquireContextA	advapi32
00000000001AD020		CryptGenRandom	advapi32
00000000001AD028		CheckTokenMembership	advapi32
00000000001AD030		DuplicateTokenEx	advapi32
00000000001AD038		LogonUserA	advapi32
00000000001AD040		LookupAccountSidA	advapi32
00000000001AD048		FreeSid	advapi32
00000000001AD050		AllocateAndInitializeSid	advapi32
00000000001AD058		ImpersonateNamedPipeClient	advapi32
00000000001AD060		RevertToSelf	advapi32
00000000001AD068		GetUserNameA	advapi32
00000000001AD070		CreateProcessWithTokenW	advapi32
00000000001AD078		CreateProcessWithLogonW	advapi32
00000000001AD080		CreateProcessAsUserA	advapi32
00000000001AD088		ImpersonateLoggedOnUser	advapi32
00000000001AD090		LookupPrivilegeValueA	advapi32
00000000001AD098		AdjustTokenPrivileges	advapi32
00000000001AD0A0		OpenThreadToken	advapi32
00000000001AD0B0		SetCurrentDirectoryA	kernel32
00000000001AD0B8		GetCurrentDirectoryA	kernel32
00000000001AD0C0		MultiByteToWideChar	kernel32
00000000001AD0C8		GetCurrentProcess	kernel32
00000000001AD0D0		GetCurrentThread	kernel32
00000000001AD0D8		ReadFile	kernel32
00000000001AD0E0		ConnectNamedPipe	kernel32
00000000001AD0E8		CreateNamedPipeA	kernel32
00000000001AD0F0		VirtualProtectEx	kernel32
00000000001AD0F8		TerminateProcess	kernel32
00000000001AD100		ReadProcessMemory	kernel32
00000000001AD108		WriteProcessMemory	kernel32
00000000001AD110		GetThreadContext	kernel32
00000000001AD118		ResumeThread	kernel32
00000000001AD120		CreateProcessA	kernel32
00000000001AD128		GetCurrentDirectoryW	kernel32
00000000001AD130		GetFullPathNameA	kernel32
00000000001AD138		GetLogicalDrives	kernel32
00000000001AD140		FindClose	kernel32
00000000001AD148		SystemTimeToTzSpecificLocalTime	kernel32

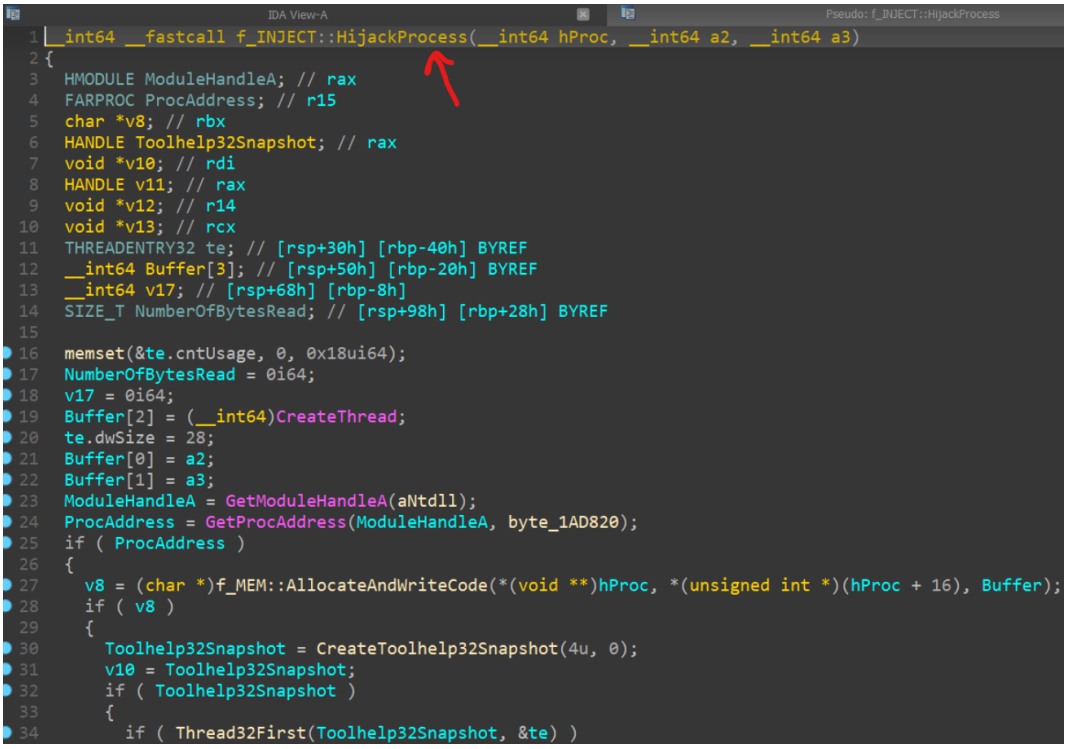


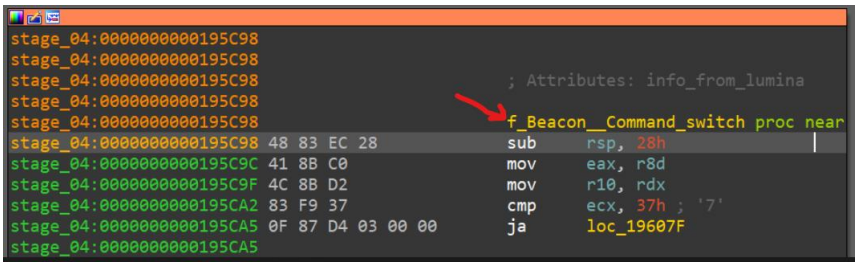
00000000001AD518	GetModuleHandleW	kernel32
00000000001AD520	LoadLibraryExW	kernel32
00000000001AD530	InternetReadFile	wininet
00000000001AD538	InternetCloseHandle	wininet
00000000001AD540	InternetConnectA	wininet
00000000001AD548	InternetQueryDataAvailable	wininet
00000000001AD550	InternetQueryOptionA	wininet
00000000001AD558	InternetSetOptionA	wininet
00000000001AD560	InternetSetStatusCallbackA	wininet
00000000001AD568	HttpOpenRequestA	wininet
00000000001AD570	HttpAddRequestHeadersA	wininet
00000000001AD578	HttpSendRequestA	wininet
00000000001AD580	HttpQueryInfoA	wininet
00000000001AD588	InternetOpenA	wininet
00000000001AD598	htons	ws2_32
00000000001AD5A0	gethostbyname	ws2_32
00000000001AD5A8	socket	ws2_32
00000000001AD5B0	send	ws2_32
00000000001AD5B8	connect	ws2_32
00000000001AD5C0	ioctlsocket	ws2_32
00000000001AD5C8	WSAIoctl	ws2_32
00000000001AD5D0	WSACleanup	ws2_32
00000000001AD5D8	WSAStartup	ws2_32
00000000001AD5E0	closesocket	ws2_32
00000000001AD5E8	htonl	ws2_32
00000000001AD5F0	htons	ws2_32
00000000001AD5F8	htonl	ws2_32
00000000001AD600	recv	ws2_32
00000000001AD608	shutdown	ws2_32
00000000001AD610	WSAGetLastError	ws2_32
00000000001AD618	__WSAFDIsSet	ws2_32
00000000001AD620	accept	ws2_32
00000000001AD628	bind	ws2_32
00000000001AD630	inet_addr	ws2_32
00000000001AD638	listen	ws2_32
00000000001AD640	recvfrom	ws2_32
00000000001AD648	select	ws2_32
00000000001AD650	sendto	ws2_32
00000000001AD658	WSASocketA	ws2_32

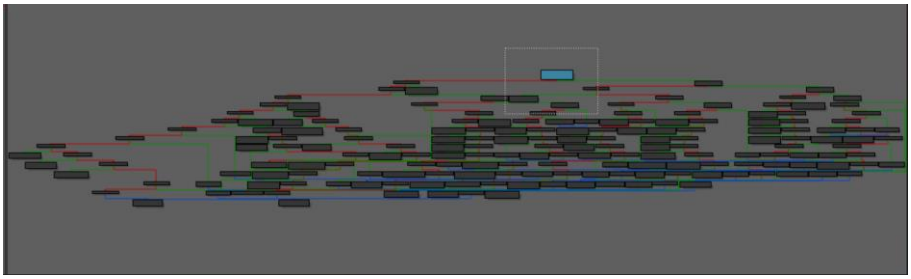
Line 13 of 201



The binary (stage 4) was uploaded to VirusTotal and was identified with Cobalt Strike. This was a huge full blown Remote Administration Tool that was annotated by many researchers in the past.







Additional Resources

- Blister Loader
    - https://www.elastic.co/security-labs/blister-loader
- UnityCapture [Gituhb] – The benign software the malware was embedded into
    - https://github.com/schellingb/UnityCapture/blob/master/Source/UnityCapturePlugin.cpp#L74
- Cobalt Strike Config-Extractor by “Sentinel One”:
    - https://github.com/Sentinel-One/CobaltStrikeParser/blob/master/parse_beacon_config.py

IOCs

InfoTrust.dll	
SHA-256	aeecc65ac8f0f6e10e95a898b60b43bf6ba9e2c0f92161956b1725d68482721d
https://www.virustotal.com/gui/file/aeecc65ac8f0f6e10e95a898b60b43bf6ba9e2c0f92161956b1725d68482721d	

Stage 3	
SHA-256	fc0594d668b01cf523eac836c0eaf57d40ff6ba6792cb3cacfbebedf85044d54
https://www.virustotal.com/gui/file/fc0594d668b01cf523eac836c0eaf57d40ff6ba6792cb3cacfbebedf85044d54	

Stage 4	
SHA-256	f7025f20fb002f665023356b07d1937886c510e4e30e18ddd2bbbc80df2d88f8
https://www.virustotal.com/gui/file/f7025f20fb002f665023356b07d1937886c510e4e30e18ddd2bbbc80df2d88f8	