RATs and stealers rush through “Heaven’s Gate” with new loader

By Holger Unterbrink

Published: 2019-07-01 · Archived: 2026-04-05 15:04:28 UTC

Monday, July 1, 2019 11:20

By [Holger Unterbrink] and [Edmund Brumaghin].

Executive summary

Malware is constantly finding new ways to avoid detection. This doesn't mean that some will never be detected, but it does allow adversaries to increase the period of time between initial release and detection. Flying under the radar for just a few days is enough to infect sufficient machines to earn a decent amount of revenue for an attack. Cisco Talos recently discovered a new campaign delivering the HawkEye Reborn keylogger and other malware that proves attackers are constantly creating new ways to avoid antivirus detection. In this campaign, the attackers built a complex loader to ensure antivirus systems to not detect the payload malware. Among these features is the infamous "Heaven's Gate" technique — a trick that allows 32-bit malware running on 64-bit systems to hide API calls by switching to a 64-bit environment. In this blog, we will show how to analyze this loader quickly, and provide an overview of how these attackers deliver the well-known HawkEye Reborn malware. During our analysis, we also discovered several notable malware families, including Remcos and various cryptocurrency mining trojans, leveraging the same loader in an attempt to evade detection and impede analysis.

Technical overview

First, let's go through a high-level overview of the loader that's used to hide and execute HawkEye Reborn. The "technical details" section will describe these stages in detail. Even if the final malware is packed and coming with

its own obfuscation, it is never written to the disk. It's always hidden inside the loader, so it's difficult for antivirus systems to detect it.

1. Find and resolve some basic API calls by CRC32.
2. Decode encoded code from the .data section.
3. Jump to this code.
4. Perform some anti-debug/anti-analysis checks.
5. Load two resources (in this case, UDXCUSCK and SCCJZ) from the loader's PE file.
6. Decode the configuration stored in the UDXCUSCK resource.
7. Copy loader to %APPDATA% folder and make it persistent via StartUp link.
8. Decode the malware payload (in this case HawkEye) stored in SCCJZ resource.
9. Start the legitimate RegAsm.exe process.
10. Inject and execute malware payload (HawkEye) into this process via process-hollowing.
11. Protect injected malware code.
12. Exit loader process.

The majority of API calls are executed by a function we called "Exec_Function":

- This function takes a custom hash value for the wanted API call as one of its arguments.
- It finds the kernel.dll address via its CRC32 checksum.
- Then, it resolves the addresses of basic API calls by name by iterating over the InMemoryOrderModuleList in the PEB_LDR_DATA structure.
- Next, it resolves the address of the wanted API call by using a custom hash function.
- Finally, it uses CallWindowProcW to execute the resolved API call.

Besides the aforementioned obfuscation techniques, some API calls are additionally obfuscated by using direct syscalls via the sysenter instruction on 32-bit systems and the Heaven's Gate technique on x64 systems. The latter means the code switches between 32- and 64-bit systems. Some antivirus applications and debuggers are missing these calls as far as they are not expecting a 32-bit application running under the Microsoft WOW64 technology on a 64-bit system to use 64-bit calls directly.

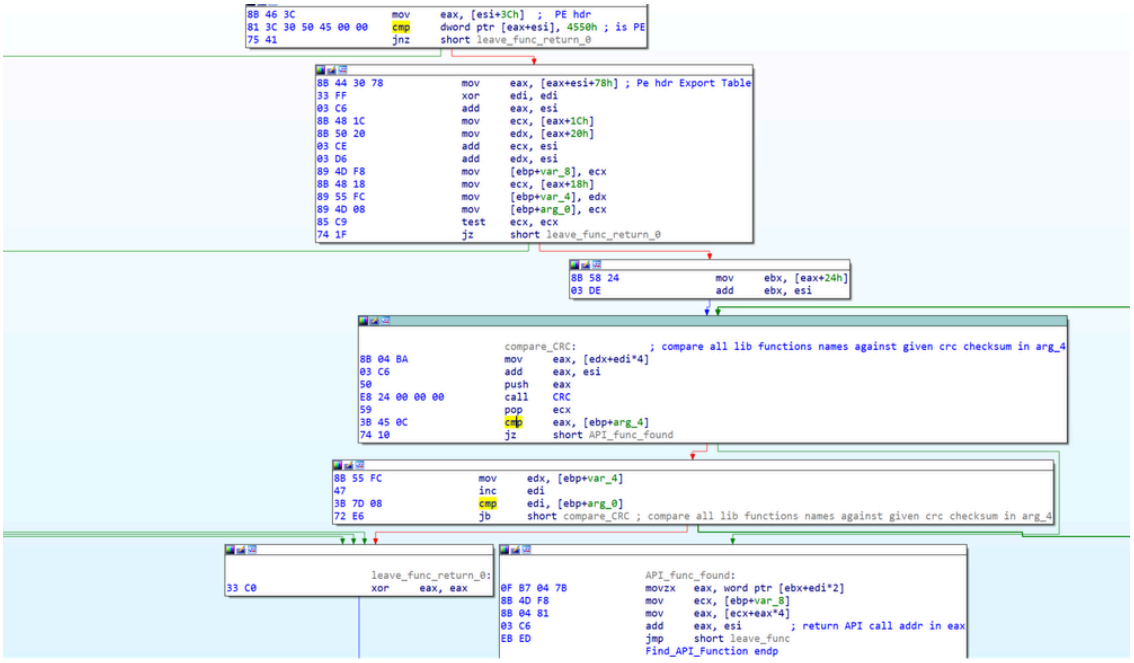
Technical details

The sample starts with some interesting calls to sub_401000.

```
.text:00401114 argv            = dword ptr  0Ch
.text:00401114 envp            = dword ptr  10h
.text:00401114
.text:00401114                 push    ebp
.text:00401115                 mov     ebp, esp
.text:00401117                 sub     esp, 78h
.text:0040111A                 push    ebx
.text:0040111B                 push    esi
.text:0040111C                 mov     esi, ds:LoadLibraryW
.text:00401122                 push    edi
.text:00401123                 push    1881CADEh       ; GetConsoleWindow
.text:00401128                 push    offset LibFileName ; "Kernel32.dll"
.text:0040112D                 mov     [ebp+var_C], 1
.text:00401134                 call    esi ; LoadLibraryW
.text:00401136                 push    eax
.text:00401137                 call    sub_401000
.text:0040113C                 pop     ecx
.text:0040113D                 pop     ecx
.text:0040113E                 push    0D535BD37h      ; SetWindowPos
.text:00401143                 push    offset aUser32Dll ; "User32.dll"
.text:00401148                 mov     edi, eax
.text:0040114A                 call    esi ; LoadLibraryW
.text:0040114C                 push    eax
.text:0040114D                 call    sub_401000
.text:00401152                 pop     ecx
.text:00401153                 pop     ecx
```

_main - Function

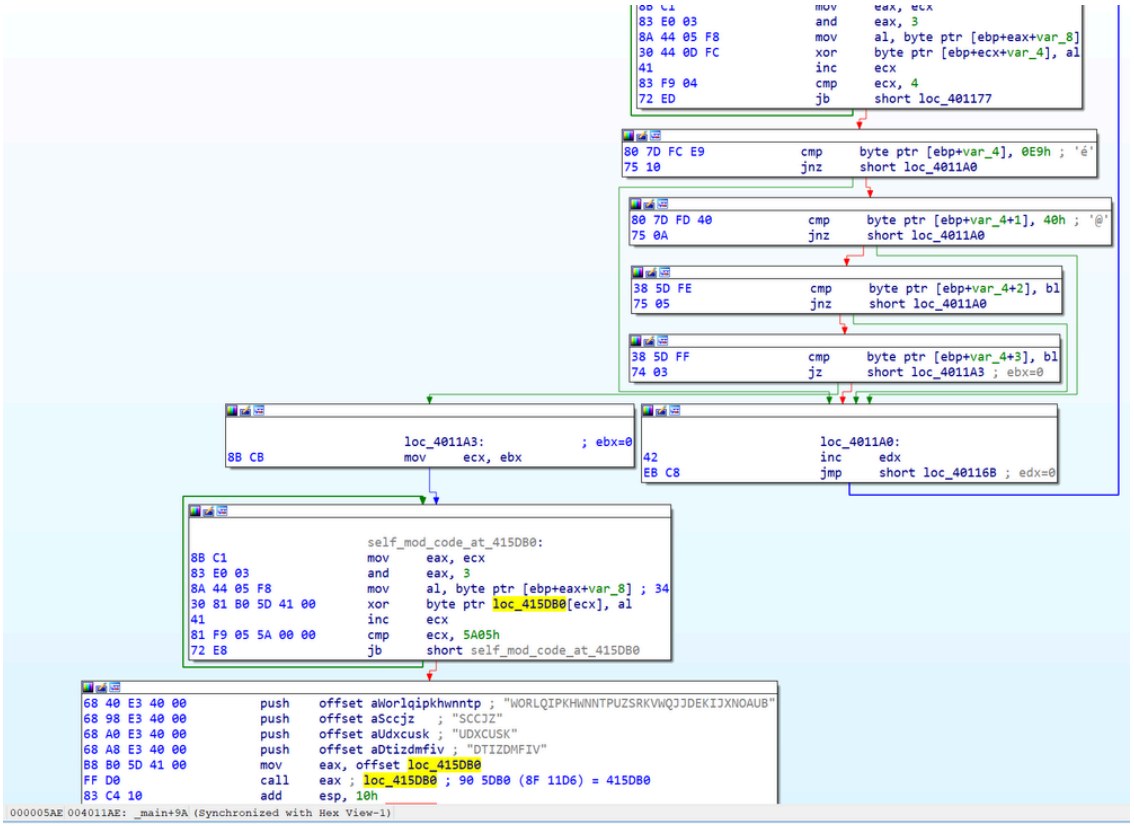
This function is iterating through the function list names in the export table of the PE header in memory. Then, it generates the CRC32 checksum for the function name string and compares it with the given argument arg_4 (see 1881CADEh above). Finally, it returns the address for the exported API function. We renamed this function "Find_API_Function_by_CRC32."



Sub_401000 (Find_API_Function_by_CRC32) function.

The returned API function address is then stored in a register or local variable, which is called when the sample needs the API function. The sample is using this and similar API call obfuscations for most of its API calls. This makes it more difficult to understand what the sample is actually doing during static analysis. The bad news is, this is the simplest one of the obfuscation techniques the sample is using.

After some initialization and decoding of the data at 415DB0 (upper right part of the picture below), the next notable call is 'call eax,' which calls the formerly decoded code at 415DB0 (see below).



Beginning of the second stage.

The function arguments aUdtizdmfiv and aSccjz are pointing to the names of two resource sections in the PE header of the sample file and DTIZDMFIV is their resource type. This makes it even more interesting. So let's look into this function. Unfortunately, we can't in our static analysis, because the data is encoded and then decoded at runtime.

```
.data:00415DB0 byte_415DB0     db 0DDh                 ; CODE XREF: _main+C2↑p
.data:00415DB0                                         ; DATA XREF: _main+9A↑w ...
.data:00415DB1                 db  79h ; y
.data:00415DB2                 db  38h ; 8
.data:00415DB3                 db  37h ; 7
.data:00415DB4                 db  34h ; 4
.data:00415DB5                 db  6Ch ; l
.data:00415DB6                 db 0B3h ; ³
.data:00415DB7                 db 0DBh ; Û
.data:00415DB8                 db  65h ; e
.data:00415DB9                 db 0B2h ; ²
.data:00415DBA                 db  7Dh ; }
.data:00415DBB                 db  3Fh ; ?
.data:00415DBC                 db 0BDh ; ½
.data:00415DBD                 db  7Ch ; |
```

Encoded next stage of the dropper.

We wrote an IDA Python script which decodes this data for us.

```
addr = 0x415db0
end = 0x5A05

magicval = [ 0x34, 0x39, 0x38, 0x37 ]

for j in range(0, end):
  a = idc.GetManyBytes(addr+j, 1)
  b = int(a.encode("hex"),16)
  b ^= magicval[j % 4]
  patch_byte(addr+j, b)
```

The script decodes the bytes and allows us to convert it into code.

```
.data:00415DB0                jmp     loc_415DF5
.data:00415DB5
.data:00415DB5 ; =============== S U B R O U T I N E =======================================
.data:00415DB5
.data:00415DB5 ; Attributes: bp-based frame
.data:00415DB5
.data:00415DB5 sub_415DB5     proc near               ; CODE XREF: .data:00416CA0↓p
.data:00415DB5
.data:00415DB5 var_4          = dword ptr -4
.data:00415DB5 arg_0          = dword ptr  8
.data:00415DB5 arg_4          = dword ptr  0Ch
.data:00415DB5
.data:00415DB5                push    ebp
.data:00415DB6                mov     ebp, esp
.data:00415DB8                push    ecx
.data:00415DB9                mov     eax, [ebp+arg_0]
```

Jump to main malware function.

The code at 415DB0 is actually a jump to the start of the main malware function at 415DF5.

```
.data:00415DF5                push    ebp
.data:00415DF6                mov     ebp, esp
.data:00415DF8                sub     esp, 8B0h
.data:00415DFE                push    ebx
.data:00415DFF                push    esi
.data:00415E00                push    edi
.data:00415E01                mov     eax, 6Bh ; 'k'
.data:00415E06                mov     [ebp-24h], ax
.data:00415E0A                mov     ecx, 65h ; 'e'
.data:00415E0F                mov     [ebp-22h], cx
.data:00415E13                mov     edx, 72h ; 'r'
.data:00415E18                mov     [ebp-20h], dx
.data:00415E1C                mov     eax, 6Eh ; 'n'
.data:00415E21                mov     [ebp-1Eh], ax
.data:00415E25                mov     ecx, 65h ; 'e'
.data:00415E2A                mov     [ebp-1Ch], cx
.data:00415E2E                mov     edx, 6Ch ; 'l'
.data:00415E33                mov     [ebp-1Ah], dx
.data:00415E37                mov     eax, 33h ; '3'
.data:00415E3C                mov     [ebp-18h], ax
.data:00415E40                mov     ecx, 32h ; '2'
.data:00415E45                mov     [ebp-16h], cx
.data:00415E49                mov     edx, 2Eh ; '.'
.data:00415E4E                mov     [ebp-14h], dx
.data:00415E52                mov     eax, 64h ; 'd'
.data:00415E57                mov     [ebp-12h], ax
.data:00415E5B                mov     ecx, 6Ch ; 'l'
.data:00415E60                mov     [ebp-10h], cx
.data:00415E64                mov     edx, 6Ch ; 'l'
.data:00415E69                mov     [ebp-0Eh], dx
.data:00415E6D                xor     eax, eax
.data:00415E6F                mov     [ebp-0Ch], ax
.data:00415E73                mov     dword ptr [ebp-30h], 0
.data:00415E7A                mov     dword ptr [ebp-2A4h], 0
.data:00415E84                mov     dword ptr [ebp-38h], 0
.data:00415E8B                mov     ecx, 43h ; 'C'
.data:00415E90                mov     [ebp-144h], cx
.data:00415E97                mov     edx, 3Ah ; ':'
.data:00415E9C                mov     [ebp-142h], dx
.data:00415EA3                mov     eax, 5Ch ; '\'
.data:00415EA8                mov     [ebp-140h], ax
.data:00415EAF                mov     ecx, 57h ; 'W'
.data:00415EB4                mov     [ebp-13Eh], cx
.data:00415EBB                mov     edx, 69h ; 'i'
.data:00415EC0                mov     [ebp-13Ch], dx
.data:00415EC7                mov     eax, 6Eh ; 'n'
```

String obfuscation.

The sample stores all characters of its strings, such as "kernel32.dll" in local variables (see above). It does this in many other locations, too. The next call at 416C74 is also worth breaking down.

```
.data:00416C46                  mov     [ebp+var_40], ax
.data:00416C4A                  mov     ecx, 6Ch ; 'l'
.data:00416C4F                  mov     [ebp+var_3E], cx
.data:00416C53                  xor     edx, edx
.data:00416C55                  mov     [ebp+var_3C], dx
.data:00416C59                  push    4
.data:00416C5B                  push    3000h
.data:00416C60                  push    17D78400h
.data:00416C65                  push    0
.data:00416C67                  push    4
.data:00416C69                  push    7554284Ch
.data:00416C6E                  lea     eax, [ebp+var_24]
.data:00416C71                  push    eax
.data:00416C72                  push    1
.data:00416C74                  call    sub_41B285
.data:00416C79                  add     esp, 20h
.data:00416C7C                  mov     [ebp+var_28], eax
.data:00416C7F                  push    eax
.data:00416C80                  cmp     eax, 11h
.data:00416C83                  jz      short loc_416C8A
.data:00416C85                  mov     eax, eax
.data:00416C87                  cld
.data:00416C88                  test    ecx, ecx
```

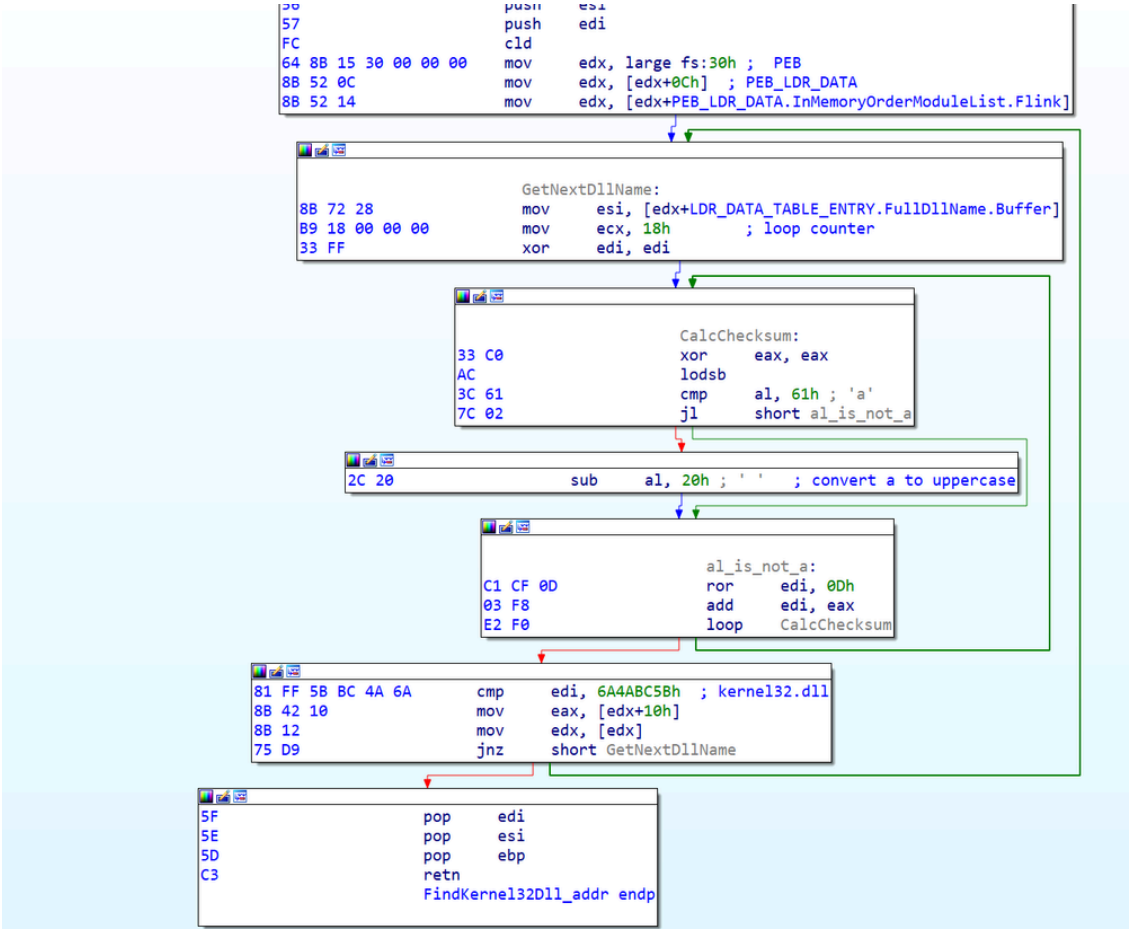
Call to sub_41B285 (Exec_Function).

Following the call into the function, we see another call at the beginning (41B28B).

```
.data:0041B286                  mov     ebp, esp
.data:0041B288                  sub     esp, 78h
.data:0041B28B                  call    sub_41AF15
.data:0041B290                  mov     [ebp+var_10], eax
.data:0041B293                  mov     [ebp+var_40], 4Ch ; 'L'
.data:0041B297                  mov     [ebp+var_3F], 6Fh ; 'o'
.data:0041B29B                  mov     [ebp+var_3E], 61h ; 'a'
.data:0041B29F                  mov     [ebp+var_3D], 64h ; 'd'
.data:0041B2A3                  mov     [ebp+var_3C], 4Ch ; 'L'
```

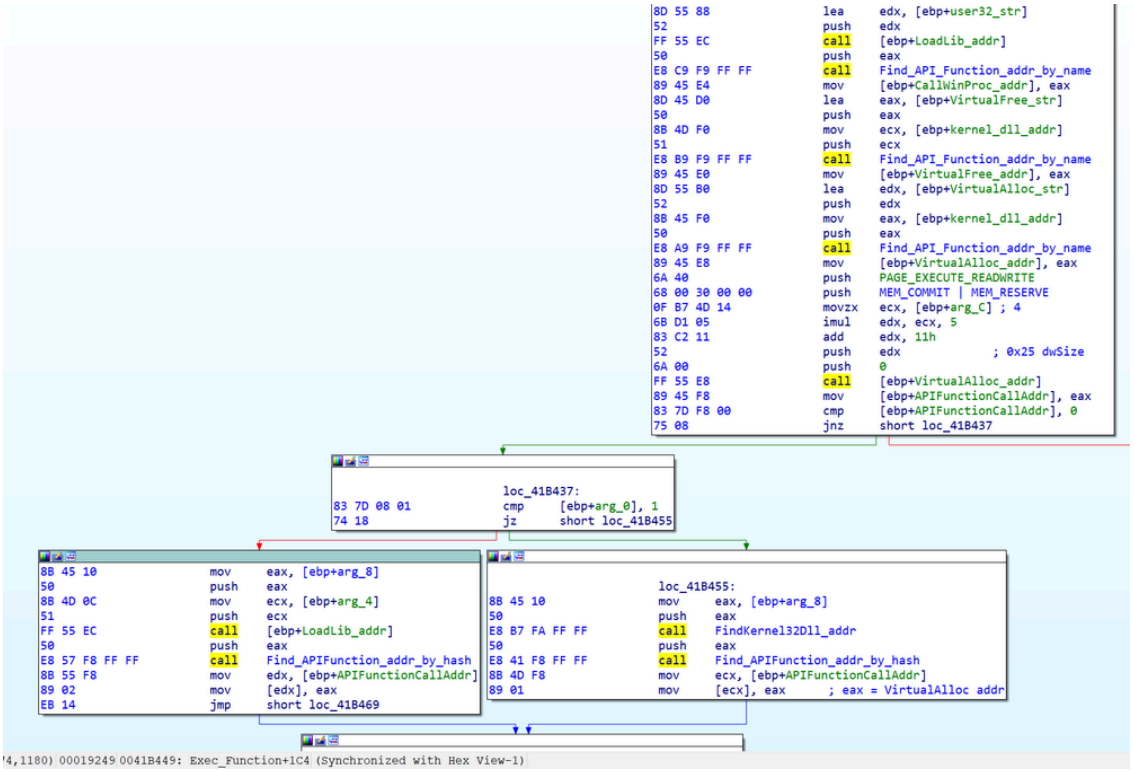
Call to sub_41AF15 (FindKernel32DLL).

After analysing it, we see it resolves the address of the loaded kernel32.dll. It uses a typical shellcode technique by parsing the PEB and some underlying structures. By finding the InMemoryOrderModuleList in PEB_LDR_DATA, it can iterate through all loaded module names and find kernel32.dll by comparing the generated checksum (6A4ABC5B).



FindKernel32DLL function.

Now, let's go back to the upper function, the one which has called "FindKernel32DLL_addr." After storing the kernel32.dll address and initializing more local variables with some strings (not shown in the picture), the code resolves a bunch of API function addresses (ex. LoadLibrary, CallWindowsProcedure, etc.) by using the function Find_API_function_addr_by_name (see below).

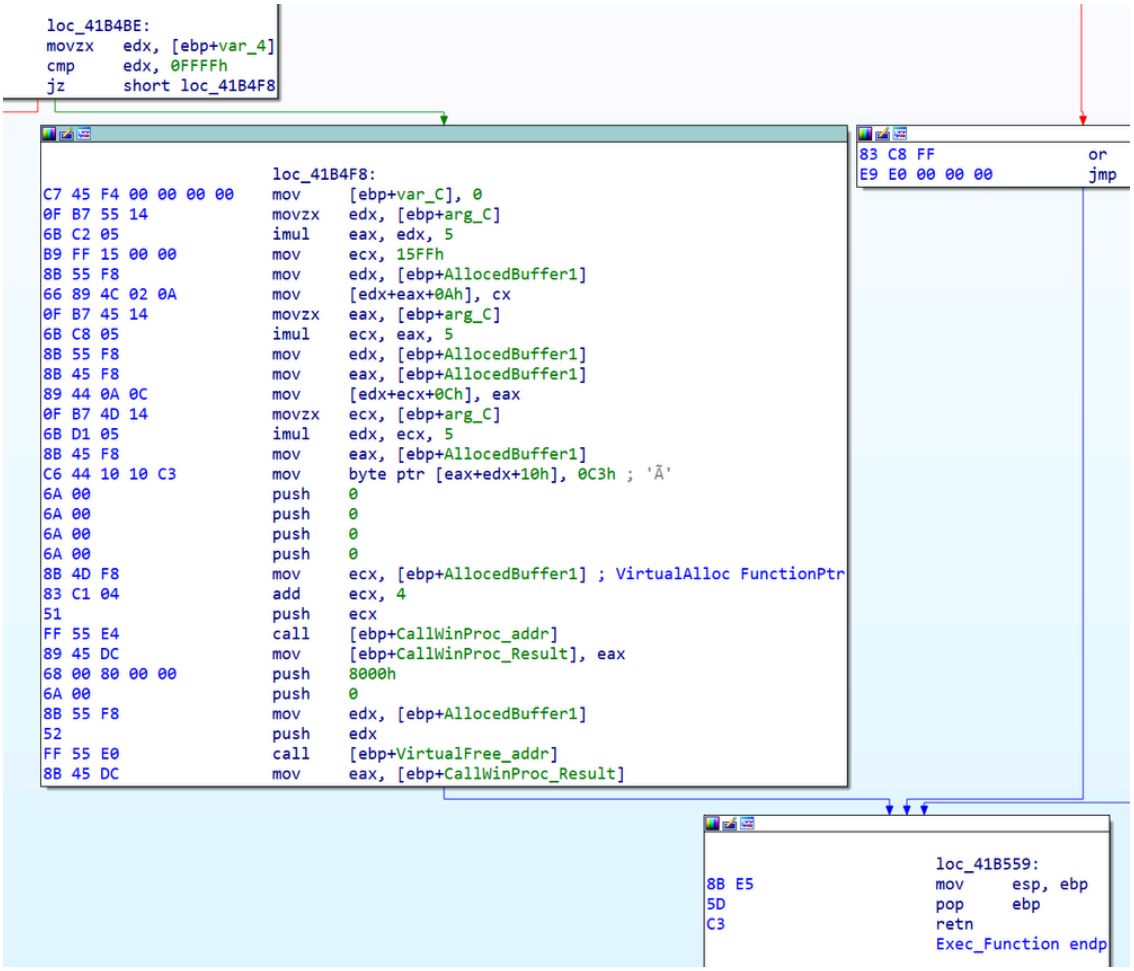


Exec_Function find API call address.

Then, it uses the given third argument (arg_8 - 7554284Ch, the custom hash of the API function) to find the corresponding API function address. The used custom hashing function is based on the following pseudocode algorithm:

```
i = 112186;
while ( *a1 )
  i = (char)*a1++ + 33 * i;
return i;
```

Finally, it uses CallWindowProcW to execute the resolved API function (see below). The latter is also an old shellcode trick used by many exploits to execute position independent code stored in some buffer. It misuses the CallWindowProcW function and leverages the fact that CallWindowProcW is simply executing the function pointer in the first argument, assuming it is either the address of a window or dialog box procedure. From an obfuscation point, this makes the static analysis more difficult and might also confuse weak antivirus products.



Exec_Function (sub_41B285).

We can rename the sub_41B285 function "Exec_Function." The picture below shows how it works. It can be used to execute most of the important Windows API calls. It is no surprise that the sample is leveraging it for most of its API calls.

```
.data:00416C46 66 89 45 C0          mov     [ebp+var_40], ax
.data:00416C4A B9 6C 00 00 00       mov     ecx, 6Ch ; 'l'
.data:00416C4F 66 89 4D C2          mov     [ebp+var_3E], cx
.data:00416C53 33 D2                xor     edx, edx
.data:00416C55 66 89 55 C4          mov     [ebp+var_3C], dx
.data:00416C59 6A 04                push    4               ; arg4 - e.g. flProtect
.data:00416C5B 68 00 30 00 00       push    3000h           ; arg3 - e.g. flAllocationType
.data:00416C60 68 00 84 D7 17       push    17D78400h       ; arg2 - e.g. dwSize
.data:00416C65 6A 00                push    0               ; arg1 - e.g. lpAddress
.data:00416C67 6A 04                push    4               ; Num arguments
.data:00416C69 68 4C 28 54 75       push    7554284Ch       ; API Call hash e.g. VirtualAlloc
.data:00416C6E 8D 45 DC             lea     eax, [ebp+var_24]
.data:00416C71 50                   push    eax
.data:00416C72 6A 01                push    1
.data:00416C74 E8 0C 46 00 00       call    Exec_Function   ; VirtualAlloc
.data:00416C79 83 C4 20             add     esp, 20h
.data:00416C7C 89 45 D8             mov     [ebp+var_28], eax
.data:00416C7F 50                   push    eax
.data:00416C80 83 F8 11             cmp     eax, 11h
.data:00416C83 74 05                jz      short loc_416C8A
.data:00416C85 8B C0                mov     eax, eax
.data:00416C87 FC                   cld
.data:00416C88 85 C9                test    ecx, ecx
```

Exec_Function parameters.

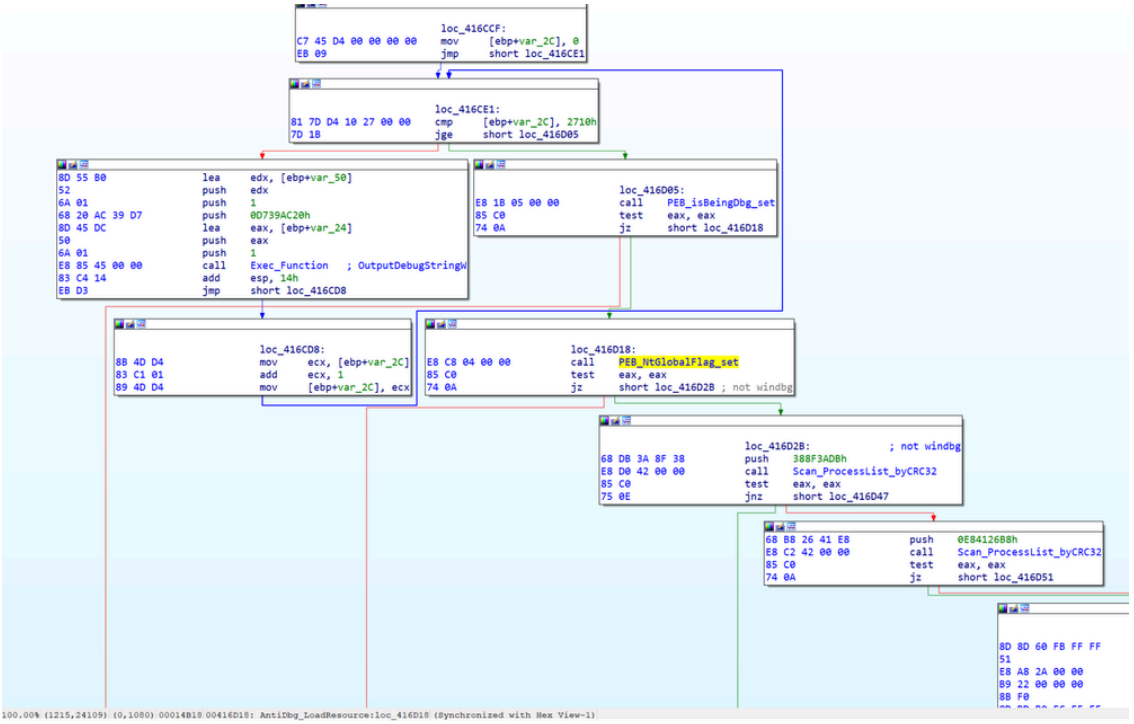
As far as "call Exec_Function" doesn't tell us which API function is called, we wrote another small IDA Python script, which parses all XREFs to this function, checks arg_8 (e.g. 7554284Ch) and resolves the given hash to an

API function call name (e.g. VirtualAlloc). Then it writes a comment to the call Exec_Function, which names the API function name that is going to be executed.

Next, the sample executes some anti-analysis checks. This includes a function, which is checking for certain processes by parsing the processlist and comparing the names against a CRC32 checksum. We called it "Scan_ProcessList_byCRC32." These checks are not only done at this location, they are distributed all over the sample and looking for the following names:

- 0x388f3adb = mple.exe
- 0xe84126b8 = sample.exe
- 0x6b68c4c6 = avastui.exe
- 0x923d5594 = avgui.exe
- 0x6b68c4c6 = avastui.exe
- 0x923d5594 = avgui.exe
- 0x6b68c4c6 = avastui.exe
- 0x923d5594 = avgui.exe
- 0x958e9b43 = extsszf.exe

We haven't checked every location, but where we did, the sample kills itself if those processes are found.



Debug checks.

After the debug checks, the sample is extracting the two resources UDXCUSK and SCCJZ from the PE resource section and stores them in two buffers for later use (see below). Then, it decodes the configuration stored in the UDXCUSK buffer. The function DecodeConfigFromResourceUDXCUSK stores pointers to the decoded data blobs on the stack. It finds them later via dword ptr ss:[ebp+eax-2C8], where eax is the offset to the different data

blobs/config parameters. Later on, these parameters help to decrypt the actual final malware embedded in the SCCJZ resource.

```
E8 A8 2A 00 00          call    Build_IAT
B9 22 00 00 00          mov     ecx, 22h ; '"'
8B F0                   mov     esi, eax
8D BD B0 FC FF FF       lea     edi, [ebp+IAT_buffer]
F3 A5                   rep movsd
C7 45 CC 00 00 00 00    mov     [ebp+res_UDXCUSK_buffer], 0
8D 95 5C FD FF FF       lea     edx, [ebp+var_size_res_UDXCUSK]
52                      push    edx
8B 45 0C                mov     eax, [ebp+arg_4_UDXCUSK] ; sub resource 1
50                      push    eax
8B 4D 08                mov     ecx, [ebp+arg_0_resource_type] ; DTIZDMFIV
51                      push    ecx
E8 DE 04 00 00          call    LoadResourceToBuffer
89 45 CC                mov     [ebp+res_UDXCUSK_buffer], eax
8D 55 C8                lea     edx, [ebp+var_size_res_SCCJZ]
52                      push    edx
8B 45 10                mov     eax, [ebp+arg_8_SCCJZ] ;  sub resource 2
50                      push    eax
8B 4D 08                mov     ecx, [ebp+arg_0_resource_type] ; DTIZDMFIV
51                      push    ecx
E8 CA 04 00 00          call    LoadResourceToBuffer
89 45 D0                mov     [ebp+res_SCCJZ_buffer], eax
8B 55 14                mov     edx, [ebp+arg_C_WORLQIPK] ; "WORLQIPKHWNNTPUZSRKVWQJJDEKIJXNOAUB"
52                      push    edx
8D 85 38 FD FF FF       lea     eax, [ebp+var_2C8_config_base] ; "0123456789:;<=>?"
50                      push    eax
8B 4D CC                mov     ecx, [ebp+res_UDXCUSK_buffer] ; "WORLQIPKHWNNTPUZS...KVWQJJDEKIJXNOAUB1"
51                      push    ecx
E8 B3 47 00 00          call    DecodeConfigFromResourceUDXCUSK
6A 32                   push    32h ; '2'
8D 95 E8 FB FF FF       lea     edx, [ebp+var_418_kgehorzlnr]
```

Load resources and decode configuration.

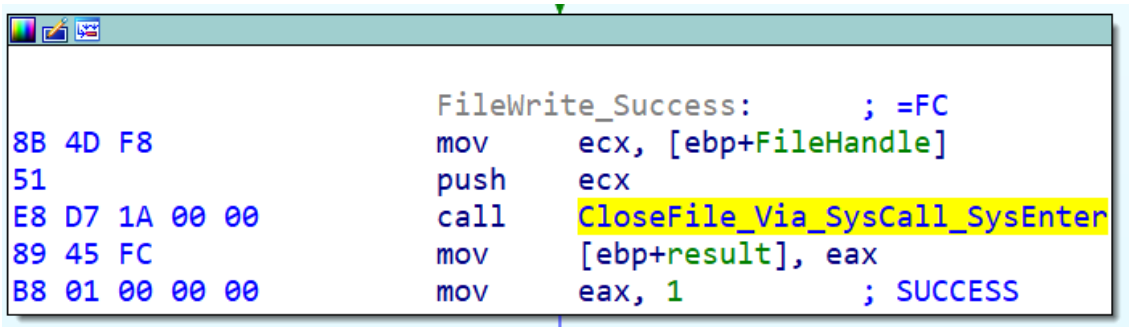
Then, makes itself persistent by copying over to <%APPDATA%>/kgehorzlnr/zqwlnpeijybtmkv.exe and placing a link to the file into the Windows startup folder.

```
6A 00                   push    0               ; if not, try again
6A 00                   push    0
8D 8D 4C FC FF FF       lea     ecx, [ebp+var_3B4] ; L"zqwlnpeijybtmkv"
51                      push    ecx
8D 95 E8 FB FF FF       lea     edx, [ebp+var_418] ; L"kgehorzlnr"
52                      push    edx
E8 20 0B 00 00          call    CopyFilesAndCreateStartupLink
89 85 58 FD FF FF       mov     [ebp+result], eax ; SUCCESS = 1
83 BD 58 FD FF FF 00    cmp     [ebp+result], 0
75 36                   jnz     short GetModFilename ; jmp if file created successfully

68 C6 C4 68 6B          push    6B68C4C6h       ; avastui.exe
E8 37 41 00 00          call    Scan_ProcessList_byCRC32
```

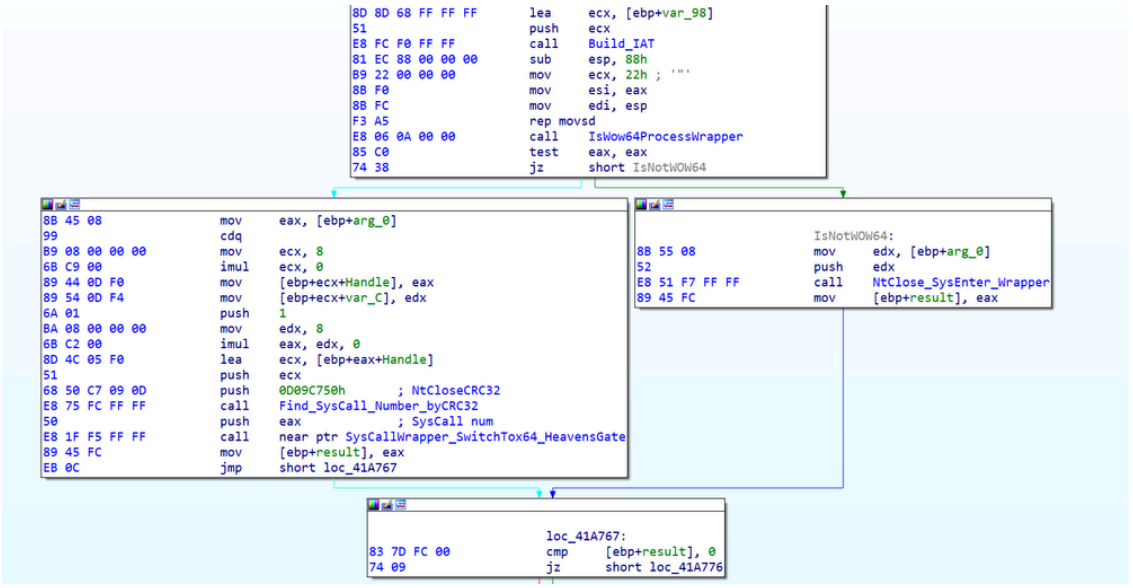
Copy loader and make it persistent.

CopyFilesAndCreateStartupLink is a complex function with a few sub functions. It is mostly using the obfuscation techniques that we've already seen, but it is also uses Heaven's Gate for some of the API calls, such as CloseFile.



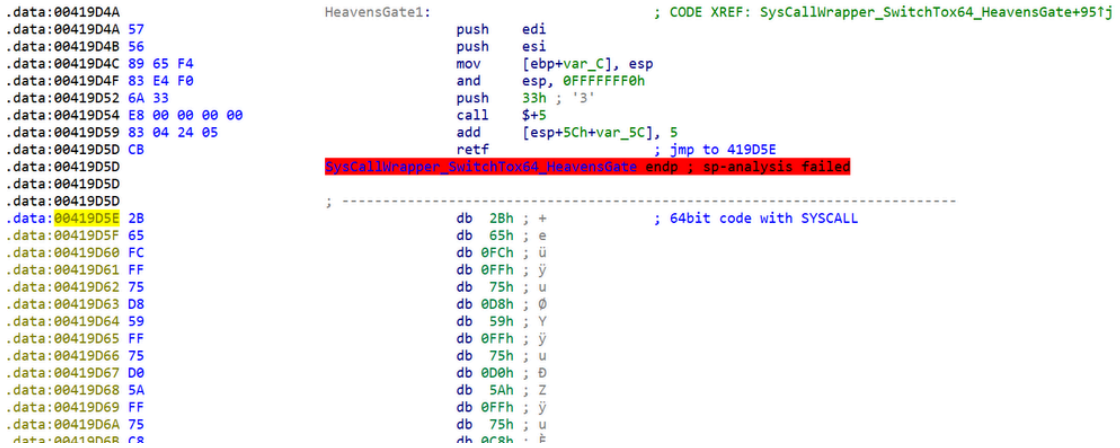
Leveraging syscalls for obfuscation.

If we dig into the CloseFile_Via_syscall_SysEnter function, we see that it is checking if it is running as a 32-bit process on a 64-bit system under the SysWOW64 technology. Depending on this check, it either uses the 32-bit sysenter instruction or the Heaven's Gate trick to execute the API call directly via the 64-bit syscall instruction. If it is using the 64-bit world, it is getting the syscall number in a similar way to what we've seen before with the API calls. It is parsing ntdll.dll for the hash of the function — such as NtCloseFile = 0D09C750h — and then it finds the corresponding syscall number.



WOW64 check and syscall resolution.

We can see the switch from 32-bit to 64-bit code inside of the SysCallwrapper_SwitchTox64_HeavensGate function. First, it pushes 33h onto the stack. Then, it performs the call $+5 trick, which means it just calls the next instruction at 419D59, but the call instruction is also pushing the instruction pointer address to the stack (419D54). The 'add' instruction adds five to this value. In other words, we have the values 419D5E and 33h on the stack. If the CPU executes 'retf,' it is jumping to 419D5E and changing the CS register to 33h (far jump). The latter means switching to 64-bit mode. You can read the details here.



Heaven's Gate

Unfortunately, this means we need to switch to the 64-bit version of IDA for the code starting at 41D55E. In 64-bit, we can see that it is simply preparing the function arguments and then calling the syscall instruction. The sample uses this for calls listed in the disassembler comments in the picture below.

```
HeavensGate proc far

var_8= dword ptr -8
var_4= dword ptr -4
arg_10= qword ptr  20h

sub     esp, [rbp-4]
push    qword ptr [rbp-28h]
pop     rcx
push    qword ptr [rbp-30h]
pop     rdx
push    qword ptr [rbp-38h]
pop     r8
push    qword ptr [rbp-40h]
pop     r9
push    qword ptr [rbp-20h]
pop     rdi
push    qword ptr [rbp-18h]
pop     rsi
test    esi, esi
jz      short loc_31
```

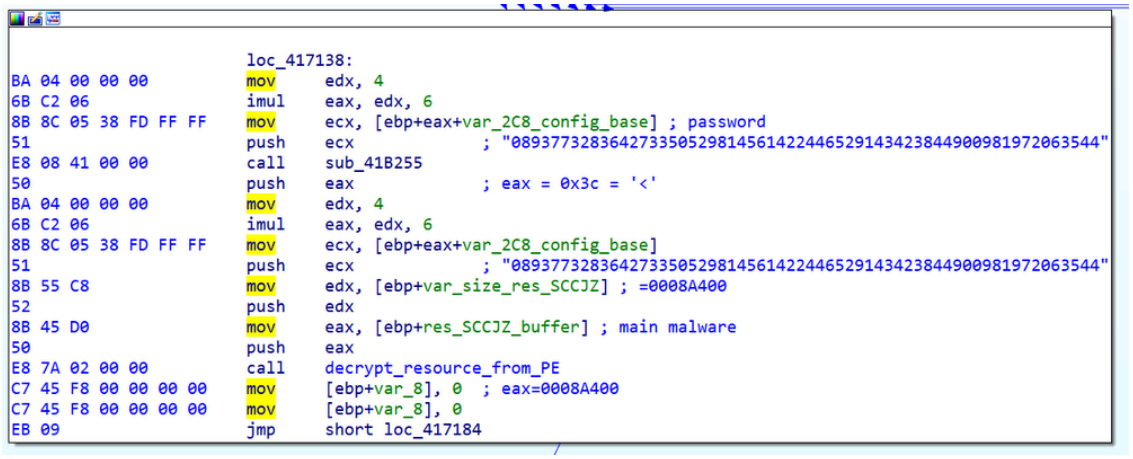
```
loc_21:
mov     rcx, [edi+esi*8]
mov     [esp+esi*8+arg_10], rcx
sub     esi, 1
jnz     short loc_21
```

```
loc_31:
push    qword ptr [rbp-28h]
pop     r10
mov     eax, [rbp+8]    ; rax = SysCallNr:
                        ; 0x47=NtCreateSection
                        ; 0x25=NtMapViewOfSection
                        ; 0x37=NtWriteVirtualMemory
                        ; 0x4f=NtResumeThread
                        ; 0x27=NtUnmapViewOfSection
                        ; 0x29=NtTerminateProcess
syscall                 ; Low latency system call
mov     [rbp-10h], eax
add     esp, [rbp-4]
call    $+5
mov     [rsp+8+var_4], 23h ; '#'
add     [rsp+8+var_8], 0Dh
retf
```

64-bit code — syscall execution.

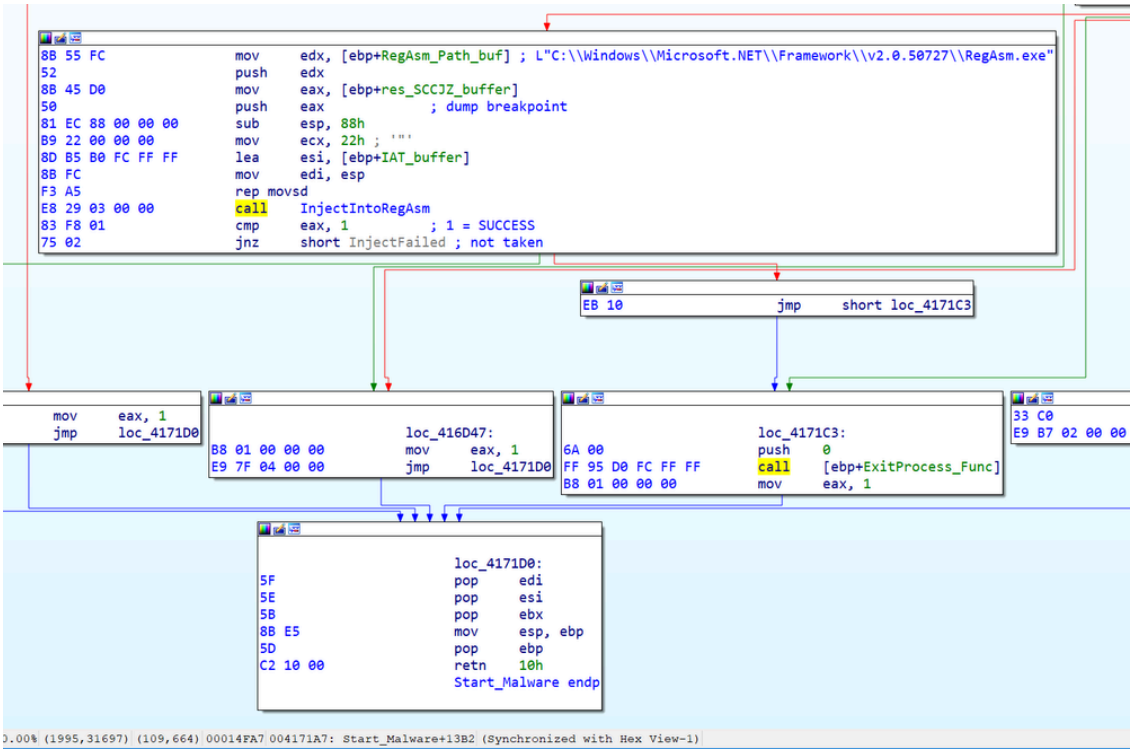
Executing 64-bit calls in a 32-bit application can also cause certain antivirus products to miss these calls, thus missing the real behavior of the application.

Now we are going back to the main malware routine. Remember that the malware has already extracted the *SCCJZ* resource into the *res_SCCJZ_buffer*. It has also already decoded the configuration that includes the "*089377328364273...981972063544*" string to decode the *SCCJZ* resource. It is stored in *ebp+eax+var_2c8*_config_base, where eax is *0x18* (-> "*089377328364273...981972063544*").



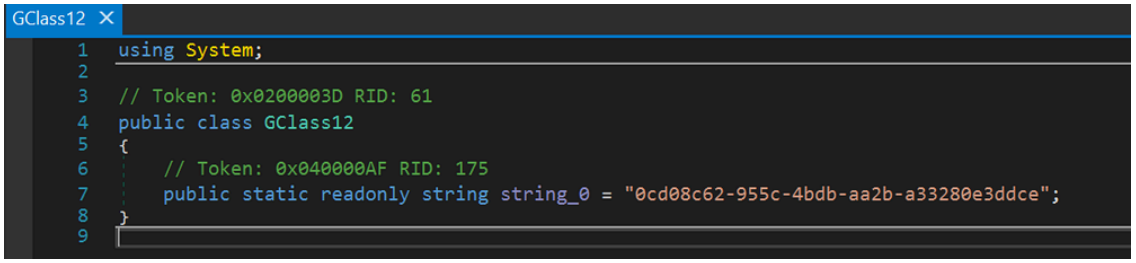
Decoding the dropped malware.

The next step is starting the legitimate RegAsm.exe process and injecting the decoded data from the resource section via the typical process-hollowing technique. Using the same obfuscation tricks previously described, we called this function "InjectIntoRegAsm" below.



InjectIntoRegAsm

In this case, the final malware injected into RegAsm.exe is our old information-stealer friend HawkEye Reborn v9, Version=9.0.1.6. As usual, it is obfuscated with ConfuserEx described in our previous research. The stolen data is exfiltrated via the email account *sartaj@jaguarline.com* to the mail server *mail.jaguarline.com*. The HawkEye Reborn configuration decryption password is: *0cd08c62-955c-4bdb-aa2b-a33280e3ddce*.



Hawkeye password

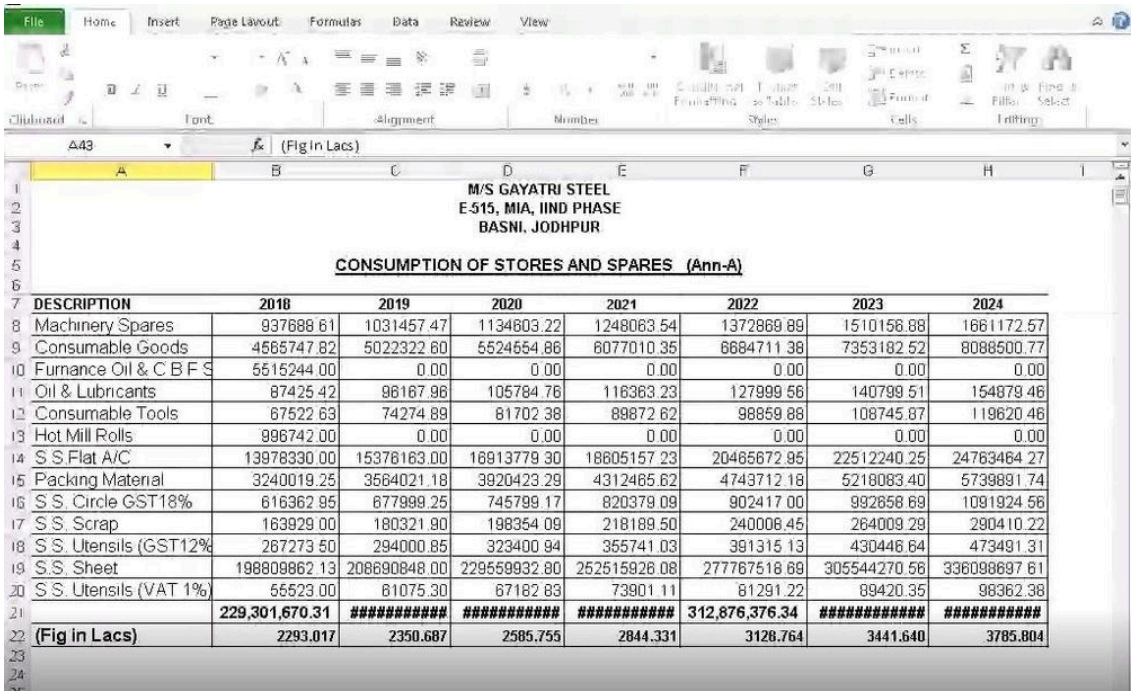
Distribution activity

After analyzing the previously described loader, we began to analyze what malware families may be leveraging it to infect victims. The most widely observed malware family at this time is HawkEye Reborn, version 9.0.1.6. Talos already broke down this malware in a previous post. We also observed several other commodity malware distribution campaigns leveraging the same loader to infect victims with Remcos, as well as cryptocurrency mining malware. This activity demonstrates how advanced techniques such as Heaven's Gate can be quickly integrated across large portions of the threat landscape. In many cases, the cybercriminals leveraging these kits lack the expertise to implement this type of functionality natively, but can instead leverage available loaders to achieve the same goal.

Email distribution

In all of the malware distribution campaigns we observed, the infection process starts very consistent with what we previously observed from threats like HawkEye Reborn, Remcos, Agent Tesla, and other commodity malware. The attackers send emails to victims disguised as invoices, banking statements and other financial-related topics.

These emails typically contain Microsoft Excel spreadsheets or Microsoft Word documents that leverage CVE-2017-11882, a vulnerability affecting Microsoft Equation Editor. When opened by victims, these malicious documents function as malware downloaders, reaching out to web servers on which the attacker is hosting their malware payload. The contents of the documents varies, but one example is below:



M/S GAYATRI STEEL
E-515, MIA, IIND PHASE
BASNI, JODHPUR

CONSUMPTION OF STORES AND SPARES (Ann-A)

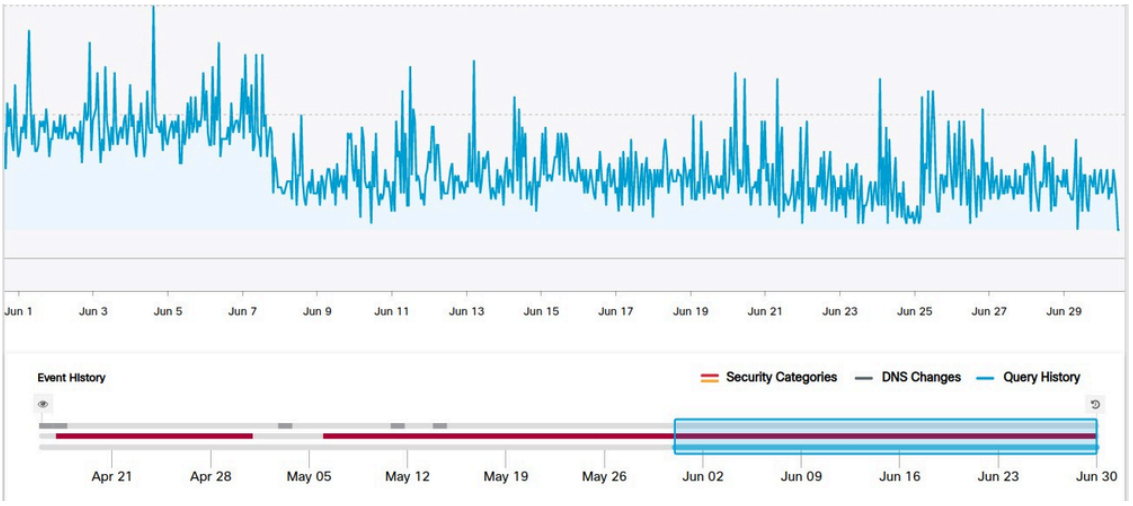
DESCRIPTION	2018	2019	2020	2021	2022	2023	2024
Machinery Spares	937688.61	1031457.47	1134603.22	1248063.54	1372869.89	1510156.88	1661172.57
Consumable Goods	4565747.82	5022322.60	5524554.86	6077010.35	6684711.38	7353182.52	8088500.77
Furnance Oil & C B F S	5515244.00	0.00	0.00	0.00	0.00	0.00	0.00
Oil & Lubricants	87425.42	96167.96	105784.76	116363.23	127999.56	140799.51	154879.46
Consumable Tools	67522.63	74274.89	81702.38	89872.62	98859.88	108745.87	119620.46
Hot Mill Rolls	996742.00	0.00	0.00	0.00	0.00	0.00	0.00
S.S.Flat A/C	13978330.00	15376163.00	16913779.30	18605157.23	20465672.95	22512240.25	24763464.27
Packing Material	3240019.25	3564021.18	3920423.29	4312465.62	4743712.18	5218083.40	5739891.74
S.S. Circle GST18%	616362.95	677999.25	745799.17	820379.09	902417.00	992658.69	1091924.56
S.S. Scrap	163929.00	180321.90	198354.09	218189.50	240008.45	264009.29	290410.22
S.S. Utensils (GST12%	267273.50	294000.85	323400.94	355741.03	391315.13	430446.64	473491.31
S.S. Sheet	198809862.13	208690848.00	229559932.80	252515926.08	277767518.69	305544270.56	336098697.61
S.S. Utensils (VAT 1%)	55523.00	61075.30	67182.83	73901.11	81291.22	89420.35	98362.38
	229,301,670.31	**##########**	**##########**	**##########**	**312,876,376.34**	**##########**	**##########**
(Fig in Lacs)	**2293.017**	**2350.687**	**2585.755**	**2844.331**	**3128.764**	**3441.640**	**3785.804**

These campaigns are ongoing, with new binaries being hosted and new emails being sent on a regular basis.

URLs ⓘ

Scanned	Detections	URL
2019-06-30	10 / 70	http://terryhill.top/proforma/
2019-06-30	10 / 70	http://terryhill.top/invoice/
2019-06-29	12 / 70	http://terryhill.top/proforma/ME.exe
2019-06-28	12 / 71	http://terryhill.top/proforma/tkraw_Protected.exe
2019-06-28	10 / 70	http://terryhill.top/proforma
2019-06-28	12 / 71	http://terryhill.top/invoice/benucrypt.exe
2019-06-27	13 / 70	http://terryhill.top/proforma/bobnewcr.exe
2019-06-27	15 / 71	http://terryhill.top/proforma/crpholi.exe
2019-06-27	13 / 71	http://terryhill.top/proforma/Joko.bat.exe
2019-06-25	15 / 71	http://terryhill.top/proforma/ttkoooo.exe
2019-06-25	10 / 70	http://terryhill.top/proforma/benuc.exe
2019-06-25	10 / 70	http://terryhill.top/proforma/playerc4.exe
2019-06-25	14 / 71	http://terryhill.top/proforma/50knewc.exe
2019-06-25	12 / 70	http://terryhill.top/proforma/benu7.exe
2019-06-25	13 / 72	http://terryhill.top/invoice/tkcrypt.exe

Below is a graph showing DNS resolution activity associated with one of the domains that is being used to host the malicious PE32 executables, and is reflective of the consistent, ongoing nature of these campaigns.



Conclusion

This campaign is the latest example of what modern malware uses to fly under the radar. With the described process, the actors are able to hide the original malware inside the loader. The Malware is only decrypted at runtime and injected into memory — it's never unencrypted on the hard drive. This means, if any antivirus tools scans the malware, it has no chance to identify the malware on the disk.

The adversaries in this case used sophisticated loaders that leverage several different low-level operating system techniques to make it as hard as possible for antivirus programs to detect the malware. By using these loaders, they can quickly and easily change the final malware or in other words the payload of the loader.

Coverage

Additional ways our customers can detect and block this threat are listed below.

Product	Protection
AMP	✓
Cloudlock	N/A
CWS	✓
Email Security	✓
Network Security	✓
Stealthwatch	N/A
Stealthwatch Cloud	N/A
Threat Grid	✓
Umbrella	✓
WSA	✓

Advanced Malware Protection (AMP) is ideally suited to prevent the execution of the malware detailed in this post. Below is a screenshot showing how AMP can protect customers from this threat. Try AMP for free here.

Cisco Cloud Web Security (CWS) or Web Security Appliance (WSA) web scanning prevents access to malicious websites and detects malware used in these attacks.

Network Security appliances such as Next-Generation Firewall (NGFW), Next-Generation Intrusion Prevention System (NGIPS), and Meraki MX can detect malicious activity associated with this threat.

AMP Threat Grid helps identify malicious binaries and build protection into all Cisco Security products.

Umbrella, our secure internet gateway (SIG), blocks users from connecting to malicious domains, IPs, and URLs, whether users are on or off the corporate network.

Additional protections with context to your specific environment and threat data are available from the Firepower Management Center.

Open Source Snort Subscriber Rule Set customers can stay up to date by downloading the latest rule pack available for purchase on Snort.org.

Indicators of Compromise (IOCs)

The following indicators of compromise can be used to identify malicious activity associated with these malware distribution campaigns.

Domains:
www[.]kemostarlogistics[.]co[.]ke
www[.]terryhill[.]top
mail[.]jaguarline[.]com

IP Addresses:
173.254.126[.]115
164.160.128[.]110

Email:
Email: sartaj@jaguarline.com
Mailserver: mail.jaguarline.com

Link in Windows Startup folder:
L"[InternetShortcut]\nURL=file:///C:\\Users\\Dex Dexter\\AppData\\Roaming\\kgehorzlnr\\zqwlnpeijybtmkv.exe"

Malicious Document Hashes (SHA256):

```
cf0a3dadba03f32d90e84401451c9acc1a1d2378d5bdae8e87fc2ab9c6ff0f12
e23d16a5b770a04664dd42f8d2153ad62ce5fbf65af2a6dfd791ad70deef61b0
```

PE32 Hashes (SHA256) analyzed:

```
 01349f0b7a30d36f21711455486024516 43d670870f8863f8baeec4f76cf83a0
10149bf87feb3276a7d6bfb864864c655b4e11aa2ed6d677c177353dbffdfc25
c2e98978063f02f9769d8372d10abc3fe734cd7e686c6ab5dedb08dd57076b17
fc31b4107bec4352fac3e1a13d91031b6b49969e21abff2301609219c43cd472
```

SHA256 of related samples using very likely the same loader:

```
 b97d550a3d4e5bc0f5f01fb3989f30e0047a8cac56b9e6be4e46ad527c9835df
53b2eead3c1b8e4652c3ec079dbf0f9cb2e1bbf51a9883b7c5c2c5414e43b54b
3c1f585dd6df5cc0e0391f2924768da9fb2c9ac2f46af9a1f50325cb362728a6
5839edf29a8841b66e6da0a821ea1e2be60a4c9c0765c1ab27df03c2d8b3d22a
c6544c438662421fda4ebb8212526f64588081bf54e233da78a8720b9e0f5532
d8977770d90fcda7b502db771ca6398ae90601ac8f2eddf1484285c2a7b4a098
f067364aa4d565aba90d38afca9c21d67253b16fec8a5c36c6cfb84d6295c108
2b139cdc6423fed45dfb5adfb18e3a141eeb7df9248bf2ac9be1696778851484
0cfa75e3eebaff86209b51e8647ecd091308a6b0083f59e011c8a8fa21af27b6
8025eaa9dad0eeecb73f95d4336dbab72d711189846c5196dba37d3846c276d8
9e25b13c1f12e1e61935b763692c204ccb8427192d83b1e4c248782fc8c6af15
4c03059b3a796b093a754a767b18f7945357bb410779d8cd3d447ff02e1ea88b
3fffc31eb23a2838069284a3de74399601cb1b2846b5615c1032232f3c0e5c41
b51afa3b4cca3020daff7a93ca38060794d5f02e7e1925db17d01e5aa8031492
1a76083a711946b9a6cf9c8b14985e7ae4872784ec8e16ca3e129aa385243e57
94392d43e99f605f189219c25d61051b92d3e0089d261226be49a69768fd170f
e0f293cbdb97cfec3e0307783e0a1065d38745fd80035ba4c04999c2dce0ebe6
d47f46adfdcc0925ebcd3d29c7ddeb8528a90bf7aa43067c9247713ee3199c45
fc04f3f5f993e0743cdebfe26820f1a2ae9ab101318577ddcfaf2b5864eb7808
780b1e40fc5b8a2f3d0cbd5c02455064606281fb2a24ee88633340178e021bae
254af6d5f33bd179b07dff10836e86574567c5f2bdee0e8e26a90af601d16d0a
d746ee2369a12c0c37acc3f3812f926e3345cf7edfd736774304e4d3c27c42ad
e61a9672421d30c721ff58deb54961736bb62bbbc34dd3890edf9062370c48d4
efa28604a547613b68480f7e8ac59f8d02931f5b8d4be6971ea96aff253d5d1a
08c4f972f9e712adc66e7b51b4843ddc6399fdfcc64e2b8245b1eca42ff5359d
e8cc0d6364caf0231c6b48d7eb44a10645b739bb5659d81e31dd5924eed29110
a3b39af055b7432c7098a01736877b036fd88ffadb54502ef88d517dd5715f33
d0ab8fa84459da9cfcbe85fc5bfe2ac3e1613aba4272698847d49efaad110587
ba63feae30f438dac75134670d2f250b4a4154b7d71bbaf793a12b7a7b1227a5
b3538b49f8d864f30c5f38d1358e7e44e1290193586a0ae4541e06bf8696b70a
f48d377a6dd312fd8572be77793984db7c0e381243dad5f5a66f5d1444e52af7
765b183479a088f6dfd2242ddd88e52c44f39f92f0d03edd44b27c22fec0ffac
813ba89c3dea3342d34b35f56fb27a53c0487d9de9444090448e2904581bac9d
b08521e9d4c442477c2f67ddc64faddb0ab13f95838b140f80dbe4dbe47ac7f3
a39ebe0acc9dfc0b22f642fd953d3f729ff976e7076fc788e8cd2be4d0b196e6
79a91c943baca8578abaddc8e4d5c96d91deb60281e71118e4b1931ff58f8107
87b0e89cb6cf9d14c36004996558ab6da6e0f8ca4e357674417a4296fd247bc2
8090fbedfcdfc8e646ff20d1f1405b9e5b9e1223c5a6260cbccf298fdd1e2ff9
cbfb44b652ca595d4125f81910e29095c679fd9ea031b31068ab85b9b5e268b1
```

```
5eb0c3a8b143b7c96fa61435d8753b6ae6650054d35ad1d0ebc357424e27e472
c2233de1fba765c99ca87f7d77af842344003291469a9e6333347ce73651939f
22076a5556127a3f4bddfcc0ac67abf85bcf76d4fdd4df9245a8f90ce41421c7
8f72b6b45672692941c78f4abc473ef4c7c93c905ccaa13090b4fa8c9ae8a94b
097e633b34a4a827a2f900bad46831716b8eb1efdc3a48e122f6676786ea3b58
373ef46e431d3ef483549bed093830ec544c28202bd552d54992a3d9bce6ebc5
0fa49302f135ddbc06d290dcc4801f87e9249ce9f313b3ebed2e42337171a9c3
358eecbf97a739a88051c3131a9957c874c86d3bc920daf1b903c7945fd7948d
00d4d2851bb0b7570b20722a82b2f2d844bcda76d44042382cfe3a7be94804b7
01349f0b7a30d36f2171145548602451643d670870f8863f8baeec4f76cf83a0
2228980ef182ccaed184556e2f3347ea5d7877da8eeddd18969df7f0e98474db
5a5ddd3a9fd0af5628754cfdb2fbcbec80d0e1c09e2e7238eac99ab7c26b850e
5d70909670129df76288c83041e686b334170b1944e1393354a8a40f8386915d
4a42df7490d910a91d3ad0d2b22a7386ac89e799ca91755e719d39eb9ce3105f
bd65e1f2474187ec69980e3bcc237d5eb5953c413e858b57f75bbc1efab13087
550778509464f5cd5e7be0b21a01fb80cff8857133cff9d65290c1dbb4b13f4b
c9117b4e20a2f439c2a4a3e9fb8554f1aee832d615666021d5cf5f6a62a15109
e33066539456b2b4b16ca938cea71ceea097320d8445d7a48d461b65accc4d7c
8a4d4491deaea94a51586c5098055c335831b37c17f3d8449fba197dfe73a83d
1191a2a9d100e235358f4b737a1bda17d27731721403d53cb4cba09979987bad
0f8572fbce7270f8462acf1b3a54d249630f138dea29203f95a35a647a150f92
78e7eb147b419da410064536a83e8856c50ddd42fefe2f84616d90bc5aa96c9b
4dc9503ba10fcaa9d2db3e031dc7854a6f7e6815efa24baba670f1f3e3c192f0
9b811b1b8d88cbb2f6ad92a7ff042c1112520bdcff7e9ce42b801de4bb241979
057eba3c5e001c212b565f5136119234ebd54f2e652d9092dfb3ba32439e9770
c217bd6a78e26c783b13d6ef935271758ac81115a7fd86133f8905329168ec7e
26ecac734753116e6afae428ede31eda76499caa52ed76e54f094b0ab04a24af
3f512ccc3610441d90a33d7aaaa72303541b57d28e7e98b46d446b99534659fc
9a4680547af935cc1f5369f60e77d148de3e318ff6b2cbe63e75b8fad051d8b6
20293acec1ea7c8b3c26b34e4715f8a9796315f3d5a19aeeff86f660bb62a379
ed01ab2b61aa9a68e7a3a0961ff9e60c3bbba7c944f1a1a7a9bf674c44c97ad2
c2e98978063f02f9769d8372d10abc3fe734cd7e686c6ab5dedb08dd57076b17
319d22b549bcbabce103c5d1359ac65f8e8ae49bff6287de21f3f9ef3138646d
f62e27717a0a9d7c5271697a65c9052362aa7a273ab78aa981b23ffa7f37f569
e3f6293b7214b9c5d51a12ca7de2ea0b82d909971abaac73143d40de6fb46168
a278d236c958536e57adf595d742021bf9249f18ec08d89f292a38e4adeecfa3
923a10cd5a45810a4f50fd88d630835d7476183ff53e2056e7e2b71cbc488684
f1fc25ceb569ee834f38061b3a0f176f58802e09b2e2987596a1ae843bf50b04
2bf7904fb0720d95ddf88e68d256f63adbe228a99e65a650faaca64101afce27
64aebb452f4208ae53a8dda999c4af052ad46030b536b161ceb4ab09a1d1320b
4687b4ccaee1db427df6649218aec766083cf28345edfb48fb2e5efc0366f0ca
37034b8e9d3889a7959aa13d99dc7ac3027d03effeb1ccdc62c39f937b303026
3111c73f9faf4127711feaf94c68cb1e08918c0b3c00b7066e2336ca7a28ae81
4897c42869fe16f3658f5f75cf0e8632c63ae424524afdce93947a5f03d0f804
ab8505cfd29a70606abe22e655741412bae9db3f18eee63f5bfa13114655d651
2b4f00ad30839c2a2c37801bb86dab1ab5df209d58788f840e4dbdd00ad06d4a
1e73aba842ebacff5998c303e91a7de845d74020fdf951506cd60b658dbaef2e
```

```
96232a85116e9c9fd3a864586055ddd7bd762a321dbc64b3e6daa2cc650eca2d
ebb8a01f413b16233df797a0d5db5c4d93e91ec5dcd09f42da9a1e367584d0dd
84d57f57ddb11d8d46db94d3cdfc06593902797833a98a623f267c5d2640823c
6376344a3e5c38687912533c5d1535173981c10e16d448b76a9deb2e8695e62c
deb037f991f1ffa4e6e9b48d6f40834fb192c04dec3afe740a4768206255ca65
9bd4ac435b07d0f7a76dfb17608b1b1027a459b10a5aa9ca5233416d2fce0448
3644203a80018fd90877d658d18452bc693b046a2a76461851294f44f4fd163f
291eed0d38fe2a0e08bf8bf207b4e1a521bce442e675fe3621c545ac1223c069
b19e62bf6617aec9cdc3d7f96fcfc23a63c1587099e632fff025ef51278d7dd4
1fe03f53fca1151eae46b3026d3b4f54f33dd3f235abb8b613adebe6ff31e385
3e5685381ca9ecc0fe592a39410b1ce079d069c2642537614852bfb4bd925003
02c10ef570d587d798c1e996fcec2d60b23be098e77694e4bfcc35818a8c9b20
7a29eb9a38d43b86698fa5a9e245eb8b27a0a27407e7d1ac8025db93b3926449
bd7510baa936badee2c828d9e563994e489396baf5538eda224482e54766554c
433ed1c9ca574cc7d179252d94d1a0b6afe5cfcb6724491c86ae230f0a5c692e
eecf481a6429117419fba24552a0a2ad690e40fdebfc51b4d83c5560188b32a7
737b0f10471e7d73ec2227dba9250c5130f16b083bc34773e112d72ded4f9e8b
8553959c6311aa9d127ada87a60ff1a9b50fe6a90388978ed1edea76277d2191
750995e29c178cb75e2602a1640276c08f73055a70657dbc274a0a32b81b3f9c
f0b067ded59292e46d0591dc9e609f0f9ee44d2a20f94613b3055abc525fc5ec
10149bf87feb3276a7d6bfb864864c655b4e11aa2ed6d677c177353dbffdfc25
f17863323f4b772be857efe9727d619a678c633b1bb2f97f0d885f584685cb60
76ddb6dbbbc196bd72f6c243f96cb564e43711d95fe8bef0e0c384d7a2629790
38e1ff632d9ac836a9f66d5b011ba7bd4ccbe097518135d9c6223b1287eb4711
e7254c9407d43e354acc21c0758c2cbb55a7cfb505abfff189d81cea5743f3b6
04971450515772684479fb5287933f0b2c7ad93433e3c69b7f235b57a1e2f128
f1076ddd97862d36741a916fd6d08ef1397d1c7cb540240a0a11821e65339e27
a310e24a341bcdee48a8809624b83ea3936109e85781f678faa1e689e79f6987
ec2c6d2747c6c64d66c84ec2fbfc826d564dbf8d682069ca928a13df4aea19e5
e9bad189d2ddd095081b8f3021ce208a18b4acab7c91b1b03dac152c8f5d9479
be072317ffe55b115fe4487d40dbe5540be7ac70f438441c9e04a8cadb15fea4
4101c74a521ffd0b9b3612fb68463219546bcee5de025e550a107d9b166fb014
66921675bfb81e0d421558f3040788294c1187facdb6d995c3ff01e07dd90b87
9000027ab595bd189e817e81d04ac0da6eed31435905186c063fd97d4faf050a
34409c20c3fa37509b3005ae1ca46b79927668636a596edaa2aba17168620c80
c27897d027f65dfcf24736f29c833122d336b2f81ce81a7e2ec1f4790ad14a81
4ed52fa4dcfa57186321211eec23375b7aee513cb3768e7f73d3fdaa56e04711
15934abb6f87f6524821c1b705e930284294a9d1bf5c82f18f3c85acb97a514a
295ec383bca0df3673988083545301e787970d770de470b4c95e9e1e071adf5e
e667a01dcadcc35b8c9c2c995e5e421f308de18f1b7c025dbb119375dc70ff0c
66c12d55e157dc4a0eb24f766e650daf516553455d789598fbe84e69eeecc7c4
7fbceb75a87b853c2a654ad50022a7a1d3e6aad60316719a2cc6b1ac20cb4e43
0836b272ffce3f1b521062ca1c38f748a26d06cec15ea0356638633cbdb2ba53
b8747a987023ed3f0c2037004cfa79b179e3cb4f45ef867529ee7c803d030e14
6bb8477fb9fcfb71db0826c700f47ef2d80606b7c9077dbce4b7254ae3db31eb
68cb2951432abdd01fcc8deab8b532c86a885a28dbb153f0ac3e344f94b0d700
00648ec07eb8960c8575f4895c7144a60bcc0ae377a7f2cf494461460e8c6e7e
```

```
5c760518415d0bb341c74333c70edd3a2881439a033fac7d4f97aa9d7d67c870
060ca8f51ea06930057376c297e0ec63228f1f3d7c76b526729d8ddd7abb0882
1be9bbc42dd8893e2eada6fbc2f6213f76d1500a467ac3ab5f3cc95745509f0e
2bd52bf833bfdda313e96a734f9860893d4e2f93974b3531faf5af88f10db84b
1e9fd0235b73788a8501fff71650129a445f4fe2b0ad97a15a63d9f0c19f349d
b7263ca0af63c8ec330a4ff62d3c2c77aca4092b9c855cfa4b0a934108e43924
277e472088dead8947e72e03387b670b85b14d73bed01d6de03bb7c42a5cb9ad
316522e4f97f2d4f6d568093a043624cbb02d46eb5a7e0f6accfdb188cf1528f
ce49da92395aac3471bce77e49d84c527aaacfadf7b517a192ff9edb02655a06
b0d0e414f0cfbf44ded2ce467bd483d566aa5aaeab6d1fba75338a9fc04f6fb3
e1a882a28f8053e96c4b6ab5648956b108cfe79170cb608d396ba455da099ec6
491a055581260b3b5c639bbec3c6c04bbc0db7d17e0b8b9cc5109c86b6aa3ad1
277d27828068293d9b9887482f5e77a787f83212289901c485480bfb0fce26bb
13d17a83959f398a303329eb63317791ed03ae30b15696de9c136cc3d97f953d
03655588004affd0d5761d961f20648086458a97bf973a9011cae56968c8e96d
8ace05215d51668fa14ca0ec731e59823d0cb392cf1313868da5750562771651
e2aed629ae9a752d1ba02149b2225a7cc1ed246131cddcba0f0b0540bf0dc045
3824d9ec7507fd567da81996592ff8003c8d7041a26f4faae98a901127644bdf
3c22205d4adc025ea1325fe8879e5d8f2c12270a5fe22da325c297c744826644
59fc7c8e2c8bed1f381630c6016ecacc4ecd9f1bcbf4b5a6139be9d47083a8ae
7149d25fe6e1a335322087f7be5494a24e246d43267286b945f4db1307764883
5e767ab245543fc4edb71cf5a3b54bdbc96ec7e3f84e171ee65ecf59f8884090
b342bc2b980b2ede862c104224dbb4784df5019b6f338bbd0c6784f27696715f
f11e2320f00e1737dcb44c433cbda5a7c84a00f547e108a185c21bd350a913a6
82bb3154f394418dc7664935de651ded32e12e85d289a27c75211d4037b1957f
6d0606c4fcb65639c8d1ce5e7995268a2a89f21bac9f91096fe00da157517d66
dae8d642c9911371c9c435e980239d262c179d0797d4dea959002f14e7613a17
09dd021835938eb6c6ec930950b41815cd4d7702f35c06faccf3b133e4004861
f0c30f73412470e8411770b12b2d4f6ab917cabce55517ae348a7697e23a2c3f
fe6e0f82a1f503d5d595691f6bff426fd8381477d21b9d23c4ee103de5e6a6a6
52bb518a4e7906fa68a4a0bb1813dc4c4ae575a891ae9730573738e032c24c5f
bf6aeb53a7beb8460e18f5b546fa3ea215837fe6819021e3306dbdfa2dde0789
62068961c64d65544dcabb5640fcbd50a05694732823b53cadb82cddc0d72c64
98ece7de8b60e356d6a965c8fecc089b86e67e2c29faa941f7cae0a64537abb9
39b14c7b01c68dbd67963156b813ff89c3755b4f12643e6bc92f6ff4b14f40ee
7d7245b0e82b6f570c65da62720f419cbfc11ae4abeb7d062bef1fccb7dd667c
cc84071ebf2a49ff92f12efec31736d0dfcb8a028d62ed65668c9b6641012d97
ecd96450a7f835bbd469dfec0e439cb6aa83b45830a6732e622b092e6173514d
959e75dbe87d962c5e65ad9059f2f73e3f1ce6e4e13cd349adc1bb209cbade5f
cf83ef20629f4ef2cd3ffe3dd7861e321bc3d2dfe6f6aeaf8447bb0a1ca8b042
a42fdf3685f82a662136a36099ba05ef38d2282fac999ce29d4dc183f8bfb01a
b576fa9d8cdc7ff7e788f2cc4b460dfe284c8abf0cbdc16f26704c0e82771bd1
3357103a936771b6a07ad55ee10d2fba8c3bc287d9a7761bb773f242ef979357
ccbe1d4c7ad7296788dfca1e0a2f45a3c065defcbd2d639ac98809d5d469c124
aa024428adc6a1eb38fb1c26a73224f5a9d221b1b0c92de2fedd6fba7322d415
84919891e03f6de5bce0855627644a13ad5c616bcb654d1509e60308640de678
4dd4374787c01ce4de4c389b03442e6b0ebed570bfad642d4cf42ce6b30a38e2
```

```
6ec195c71e7f5b2561381e6457ff3c90255cde408d6c2451bc7342c945767161
ba0c598fb174eb943d66e908065566e7698f0946a368f05135b004761239761f
2c1d4304c0ac9b2481669e6aefef515aa69f8f8cea4f2cac41dafae8d76ac2a8
0cfb09396d1891b95710302b0bc2e8031279805510b7b47ee151daaecb79f7fa
0b84d8793c5362b0d10f238fd2bcf8fc4a1d063ea1c4e767f2d7081872b1a050
a35863cf5014a3c7c3aeee5f80e59635885c0893b3fb9325869c9a4de8df412f
7a7be7f6e662a8f7c2039de122ca08da4193d681b8a8a4ac5a25de6223ebcdf4
be2870dc539c827d85401d865166db981ba2a136629fe25ea466ec0ecbbf6e48
78e536063c280fadb0a78c24c976cd70c0bc0139908b8ec1805c8faa676a8027
9e030cd415efb6e040d03a3f69dada5d4f08c0c430cf8400be706c2d254c42ec
f8e27ad560a6b5a2fe9f156cf8bcf2dd2fc57cb6c620cd1a3159f124e41af8c0
50d35c2baa8045c6455d26370142942bcc6d7c648250cdeb76f1a771c2ad13e2
f4c44d4c9f98f0a738837b3bc5965558915522be3f44a9c31fa5560546ff96fc
f504e794ee3f7f537177a22afdd0a830e4a96e1531734171919c86f96909f334
f70997b46388f104340a35e901dd3172a4a98b2bc08cf75ea5a9f79831b0f84d
ea123c9b6299186b1319ec6572bd16fb6a28185f2e9ddb9aa1bf3e52f1911b5d
76baee103f6d1e0f63a22deefff38e815e53c1f9627f41b9220cc08cd32e5434
de88e75b519fd4c118e3756b2ca48ebfbbeae1c084603e642022c074676de362
7ccb34bd9651f6f27d531128d839d8d0c1853f2b6f29fed69b7e19448bfd3024
faad82acc0652954a8dced1b5b119189be888e897fe7b4378587627d9a1ab0ed
53bbc252c80c752c62dab6a1c8f9d200c20e1e35789bf87716b866c649401e5f
2dc6944f17feefdcb4ed6f2faef0abe0af4f3e883c3d1dbed136a7458438398f
32422502bcdeeea7371d8a17307da010af8d30ae20eb3d6d6c52685448646931
76229d35d18642da6e365259353937c22b4e2dfb933e6dd9f6e5825c33f39681
9918d0b4a9d3a2775d8fa813db7d8110b47903c23056a60ce70e200dc33ed019
66ad5506a0b31b6ad852f2609354d507085b311f6bd67d68ec3f6c7f73b36bf8
3a8d43e2d37874ed3dfcb68d3f77ffed07e7ba5a0fc91707c9a920427410ed34
b720c635fcade5e16367ab730b6b7ef09a31d0cf579ffe5bb115b8d3264047bc
24de434e2c57449c820f49c0c6e49b529964904b2e534edf6d2e92515d9546ee
44320a15b6bead25ac56454e8ecc65071136d25e55074356bfa9ceabbeb62012
a8714f23b9805dd8421f2817381fbe1a8d4141ee612a452b954394481ccd19ab
9f4c4e2f52ef3c30949d2b12e69062a434f3413aff92f6460a9b3151a4850514
0df480f2bec98324d9a870e5019eb64f4858bcb8daf66163700e0cb6a2937767
3d3c726175d7b31be7471eb3198861a1642aa1226d9634bf6aa593a4fb9d3bcc
f71e494c7c028b165f7956fea8144ffc0edaf538c0147e59754f61788c47938a
35fd7e301a0e78a759cd50414b9d0cfe1b4d6c5c9316bf3f8dc5afb388f0ad7e
526e70806ea478014016fb49cd9f297d15e3c9d322aae366f9b0a521c8ea0efa
4f62a36d85afcaf7b6417b60a45f17d5a921aaa5614c83135b29c0419c02e199
85eb95c03b47f3f5c0f4003a444ee976ebbc1d36a427d9ba576e5b7fe51f23f3
f3f85da23f46c34b270c1d82c01521586e2130ded1c3b55ae8f5a5a0f59f1001
813ebe94ed822fc8c2b42904602e62363b1a24ac95d8d0d56990e1d92afdd8c1
33f54e062132e8846af89f6f6e37ee191a177c6c9eff3cfe1b784e2ad14e8c43
385827beab7a004fd55ac137b9a19f88829e3f4d4bfbca6b6ffce6d49d1dcd37
29db21426f77a935c7b82a973ec62f97a3627b5967b0a2a8bb4c4a078c6f02de
ec8da89ced1ba93ff9e9ad0674cf487b40bc8aa66ef37d52bdb6972d20059886
bda55e17c599b80c688e93249375fb027754aef373ecf8a05f205f1ff4bbf21d
07c38e258efe82eae706fa4b53826082e1246188383910e13e7e524d02b8c814
```

```
d6cee6f5c99675818a8a523df6864c6b8cec92729b1e4e5aa5aa1e11b87be528
81e5f237a9432a1068290a0bf59f5e3e889009ab38c4113ffe587445f452e396
8992be7dcc77bbe8d2010b49c71b99badf860d4ca064e71005076801ff9fb9fd
67a823dd0acd8fc0d86e402b90564670b6c88d6872d263f31169acba325a18ba
1aad661acc01d544c084fa5f1957c1f61480feebc28e0e63c42b9f0324b4b651
36ba85a2d278fb599de9dd36adbe289c39264055996b764d8979f45bcf123535
8f241077b523d9db54c5075303fb88e1416804a33fa2575d64650b714c0dc94b
c082a8a175ffa2d35a93851a8c963224fe29938eac91fbadfe299c7a1e72139c
5eb5e0022df44e595f2b362f6eb5278a2211d317fed684e584901bf507d5a033
99cd4d6be9a58331cb148cf0696d60c40ac72c8a46ba592c984bf9bfffb07ede
c05402a8708719dde5ad315d9e4236c5128ad8cd7ca1fc28228b4836061d7503
bc0e103c8eae02eba219cfa761eef4924385c7328dd59e6b72f49babf8f3189a
159d09cdbd90e5ce221f9ca7fd30646268cb2521d4279d707b346602b0eda59d
d6c67ba1b4a3cfdd48e8f835e12d898b79e5741bab699c3956f615fe6bf28572
636137367dd6f556f0740e9361edfc2418d3f4cac4fc166f174da18c3e2c7131
655d514709f31f22ab07b16fbd33c2cffbf8abc1331027a3f144558e09970d4b
b54b0c73ba7f1b37b7af59ad459aefae4dbc3cc0c7a324ac94ca39bc23580c0c
2171f379b875969f797ebe9c23bc46d827ff221c5da4ed8cdd65bc4b2855ce7d
ef80c882447928a079e7d786fd680d6b6c2a1a3751841165a49b00226aebc2ff
37af5ff460fc62d754829b7ade4ae1e0b6c4881fece1e3f089032d12301c0bb0
1fa15197c1e9a1fa5dfb1916c6a514778a609889193f9b65f24383a4b7693ec5
e432256ff7867c4abeb51885011d5f65b09566e11782147a9e2b47d55b8adf8d
42409f1eb65931970e4063c2ed63a8ee44d2f5c6c5853bf2ec35686d2766e268
18622af49909851dfdfc7926fd23285784b5efd3adfd4293663003825025d88f
4dc61ea009d78dfe1399c5c7fc9c60ed6a144378cc51753cf495819c22c247ee
d21f035e7429a0f801155a1ccb22d8d782867aa46b960b6909b74212749bd821
64f5687a6b2e99953e18c218fd184883cea7009a10fc680da520ecefda5338c1
a06e1dc7e077b073bf3b8e616efcf67b8ff9428293fcb89ae6f937a1a0a5b3d8
ba11b9b4c9e0084e5ae5d0de45761b6bd6ebbb62d41c93c7a23ceeda8461d4b1
d608b33a3c5695e828b7afec1ff9276b42cf750f7fb42a14079b4b29a7b59c3e
57baab61c3e3e5af27c22ef7a6c9ebdc6c280cc9ac3840572c76827954656a54
6f93663fdea32014c733f460385ed3a55cabaafbc798283edf1ec1b1c12203a2
f091381bb04045479f969e5f5dbcb8325590bf278b29d760bbc9bc08adb97e63
988316a8842a2be9636a222a8b6f1d22f6587e4f648473d2b9166946c2b2b90f
1602287d45ea448e91c4605cc8c5301d264cc2785c8ecb0aa3526bcf0e6b74a8
bfefbd8050f0dfbe1047ddcc07e951967a5b8395190127d97d0c3a4441c919bf
517dc095b262c6eeb7d80c3b025728278cc251f4e244af1c7eafff135a41ba4a
f13de5281ef8a3bdec7e6f309870ffe9613da4796baa4ba8a37b748002bc06ab
9c748ece1010fa49b15faf8e589709381b08e5451c00774b538ad3020e2291bf
757bdd96c6547a00d13e88c58995fd3d9372bc7ff0fa717402337d9ea7cff513
dbbc786378fb59ade9c6c99baec054206ac4504328e4be7044a624170a6c1863
8cd8d977dc8764927c9d7dd24b79c984fa94ef9aa1b309363d61a759ed9ad8e5
909ffaca9d78ba1e391fa62382770774f37204fdc5de24ec4234d5bc636d38b0
2c78fce765f51d101fb68e4cc3e98ee2cf949ca8a8d987f4b5d01fa4158239b2
62f655f3e9766e9923c6241859d567339c643ee5bad7f6ebe43b2ffd94e93d0e
1f9d3dc21f1f816e5b816d3633ad9ec2696138519d2da972f179a8ccb4689f83
fc31b4107bec4352fac3e1a13d91031b6b49969e21abff2301609219c43cd472
```

```
7f2413783e24f161a06e13fa1e3ba14d8455e9e6de1bb71eb5082f80251c1e30
97878c51723426766561d3c3c319cf649d049514b1330c90389c2b5d45cbc759
74e9408af14a09e5c8af61780dc62112054876ae357da3f6ebf8319799b14d14
a10486e5696f8585ed7c88e4bcfb440eea7b67ece05e6a871572282916f72d0b
885bc554e5fc03b250b359afd5a360431bd4768f60772a60a66c65a241909e49
46fda32fdf5d29190b8315e30e1fd6df3a4aaf43308c8a354a6be203a892e6ab
c3d4e6bb0c9f22101a14c8455677acf98fdae3e2cb8063c76c5119b92c73e0d7
243460e5b641862bed80bd004ff280f3ae97fe18415616401ce33988402a14dd
3eb2472ea1d712d6285ca3a442debdad94575a4d1fb9b9caf67c41dd255e98c0
4e987f55eabd6f9efd1947dbcdf85af455a1f7087962019f23f66a308a1073c8
451057e11ab32bd3f6e7d71b8709c287c1d82e4e2036df712d688f7e0afc73b6
9679abcdadaf4dfb9f3ae2a79b7bbaec08921483ae45e4486397dbd4afcbd462
9a43f3338818991e28edd5e4be7531bfe2c5dbd4ffb3f55a05969e99141c8001
6e6b891c5b86dc44cf37398291e01e623408e2bf7c47b88fc557e5fd222e140e
490ae3ed23b6e40d1e7b6fd8b20ba69d4c20d9cbc5e7a53d9bc3a1824d4c45e9
1977b89113e160518a917cd3a0da2e1e61c466251a0690464d425c191e3efbe5
1efb2130e792e899d3fee5b0582e61b54f9bdafd00ae43e727d618d462a64a42
8772387a55e177ff01fa20b6941dddde054c594eee8098cdf96a57e2ccb78b7d
abed1f8ff3882e4737125b583f703683c3ac5786c8732832953e4e528d3af5f9
0484de9ff7c50e22ba4cbc6d9e20fe5d680ee352af1d3a650df0c770848d9b90
44821f79cf871b34e62aa86ee3e4577a18dca74a39f743081e2a8dec994025e8
cf830a8c2868b346090401ff03c4ab44b13f50e33534add2c101ee8ff362776e
28c495032494011c1b70b68ce584a929841ba9ba0d22a83e4084e886f6db2721
88543e7295c60ecbd0063c71bb6d4a9d2a3397c79cc19c6f3213d88527a4d3bc
b5ceafde9b29c186afd1a17ab534ccb1f578ff9a3de9d8b20ad55f437fc030cc
e560a9c1961eff0a0cc1dc309dd03511b43d7bba36571caa6dc2e36132126e65
f74fec6680f946ad601522ba1f87d58c0fd8c4747ec181083134fc4d353fc0f0
19782d2daa25113c0d57c2a3e980bc224150055159acfe852e520803a905e908
5e43116e3b18615353968d8d7acd8f535aa8cb95568e78bb6ad926db6688d37e
aa54bce71f6e34a0f20f87085150a5060764fa4a594a7f6d91d01779f03023a9
d55319e0315279969ce264b10da904aaf38fc16ea29b4ea30624a0c2fcf7fc5d
939aeff288528182c81cbadce24a687fc962fa8c50b1e696169be8f4e7c4fcc9
195e76b41f7256e146e5b3aedd8dcdb6e15fd40deb8159b46c8e8b2af157aae5
e287aea0eb69b9b98dda453b22c3defbd00f344b2258ec2fb57677e77571d089
2238e42e6cd3db765c046bdecb28bab4142f9b9c8df59a1f55bfb93e499e5825
b9f7c411d951fa455180aa9e293ce1bcc9939b8040b726cb57f9afdf718603b5
cf8d41d14f7d2ed82b6b0199a0335b15c2956c403607fc92af11378e4c02f577
37f5b2c50274063c5948e5a425e50528205bb0adfe4d7084b4d756b0de594289
5aaedf8c7d06cab042b23025ec4c0ab1215009178a6dbd44d1e6edad527356eb
```

Source: https://blog.talosintelligence.com/rats-and-stealers-rush-through-heavens/