Reversing Golang Developed Ransomware: SNAKE | 0ffset Training Solutions

By Gabriele Orini

Published: 2022-11-23 · Archived: 2026-04-06 00:37:07 UTC

Introduction

Snake Ransomware (or EKANS Ransomware) is a Golang ransomware which in the past has affected several companies such as Enel and Honda. The MD5 hashing of the analyzed sample is ED3C05BDE9F0EA0F1321355B03AC42D0. This sample in particular is obfuscated with Gobfuscate, an open source obfuscation project available on Github.

Let's start by quickly summarizing the functionality of the malware:

- First, the sample checks the domain to which the infected host belongs to, before continuing execution
- Next, it checks whether the computer is a Backup Domain Controller or Primary Domain Controller, and if so will only drop the ransom note, rather than encrypting the machine
- SNAKE will then isolate the host machine from the network by leveraging the netsh tool
- The shadow copies on the system are deleted using WMI
- SNAKE attempts to terminate any running AV, EDR, and SIEM components
- Finally, local files on the system are encrypted
    - For each file, a unique AES encryption key is generated, which is later encrypted with an RSA-2048 public key and stored within the encrypted file.

Let's start reversing this sample with IDA!

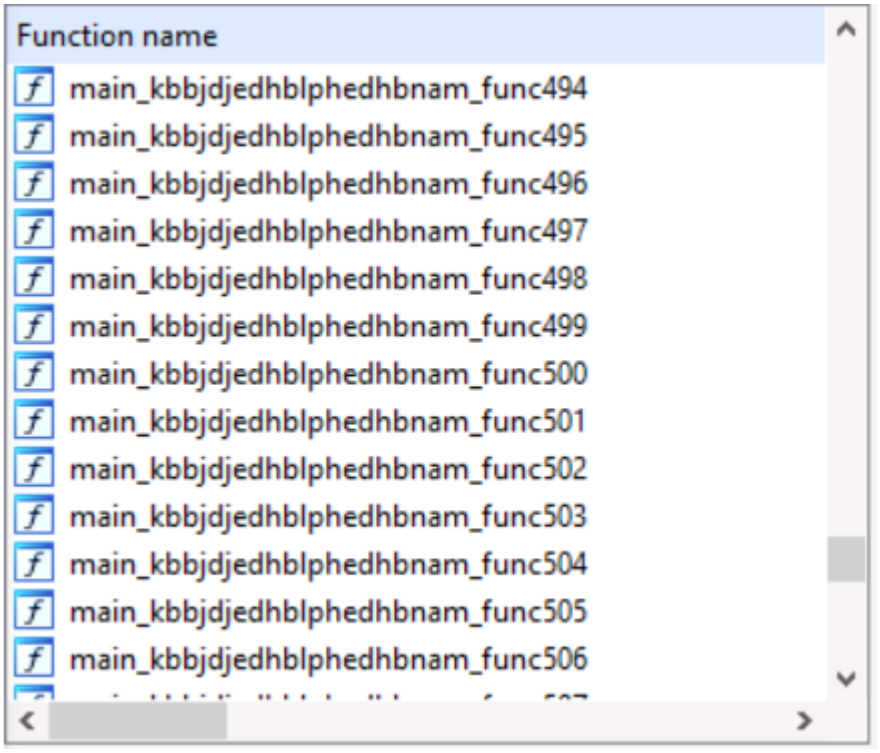
Static Analysis

There are some differences of Go from other languages to keep in mind:

- Functions can return multiple values.
- Function parameters are passed on the stack.
- Strings are typically a sequence of bytes with a fixed length, that are not null terminated; the string is represented by a structure formed by a pointer to a byte array, and the length of the string.
- The constants are stored in one large buffer and sorted by length.
- Many stack manipulations are present within the binaries, that can make the analysis complex.

Obfuscation

Opening the sample with IDA we immediately notice that the function names are very obfuscated:



Gobfuscate obfuscates almost all function names within the binary

Performing a quick search, I found that the malware is obfuscated with **Gobfuscate**. This project performs obfuscation of several components: Global names, Package names, Struct methods, and Strings.

Each string in the binary is replaced by a function call. Each function contains two arrays that are xored to get the original string. When implementing the decryption function, keep in mind that there are different ways in which these arrays are passed to the function **runtime.stringtoslicebyte** – either via a variable, a pointer or a hardcoded value).

Analyzing the sample, you will usually find the call to a main function that contains a large number of other calls within it, where each subroutine performs decryption of only one string. Sometimes only the decryption operations are found within these subroutines, other times additional operations are performed on the decrypted string.

```
5411A 81 EC 98 00 00 00                 sub     esp, 98h
54120 E8 7B A0 02 00                    call    main_igkdjigbmmblodcncggh_func1
54125 8B 44 24 04                       mov     eax, [esp+98h+var_94]
54129 89 44 24 2C                       mov     [esp+98h+var_6C], eax
5412D 8B 0C 24                          mov     ecx, [esp+98h+var_98]
54130 89 4C 24 44                       mov     [esp+98h+var_54], ecx
54134 E8 67 A1 02 00                    call    main_igkdjigbmmblodcncggh_func2
54139 8B 44 24 04                       mov     eax, [esp+98h+var_94]
5413D 89 44 24 28                       mov     [esp+98h+var_70], eax
54141 8B 0C 24                          mov     ecx, [esp+98h+var_98]
54144 89 4C 24 40                       mov     [esp+98h+var_58], ecx
54148 E8 53 A2 02 00                    call    main_igkdjigbmmblodcncggh_func3
5414D 8B 44 24 04                       mov     eax, [esp+98h+var_94]
54151 89 44 24 24                       mov     [esp+98h+var_74], eax
54155 8B 0C 24                          mov     ecx, [esp+98h+var_98]
54158 89 4C 24 3C                       mov     [esp+98h+var_5C], ecx
5415C E8 3F A3 02 00                    call    main_igkdjigbmmblodcncggh_func4
54161 8B 44 24 04                       mov     eax, [esp+98h+var_94]
```

String Decryption Functions

```
54640 89 84 24 3C 23 00 00                mov     [esp+4658h+var_231C], eax
54647 E8 64 AE 02 00                      call    main_kbbjdjedhblphedhbnam_func1
5464C 8B 44 24 04                         mov     eax, [esp+4658h+var_4654]
54650 89 84 24 98 11 00 00                mov     [esp+4658h+var_34C0], eax
54657 8B 0C 24                            mov     ecx, [esp+4658h+var_4658]
5465A 89 8C 24 2C 23 00 00                mov     [esp+4658h+var_232C], ecx
54661 E8 4A AF 02 00                      call    main_kbbjdjedhblphedhbnam_func2
54666 8B 44 24 04                         mov     eax, [esp+4658h+var_4654]
5466A 89 84 24 94 11 00 00                mov     [esp+4658h+var_34C4], eax
54671 8B 0C 24                            mov     ecx, [esp+4658h+var_4658]
54674 89 8C 24 28 23 00 00                mov     [esp+4658h+var_2330], ecx
5467B E8 30 B0 02 00                      call    main_kbbjdjedhblphedhbnam_func3
54680 8B 44 24 04                         mov     eax, [esp+4658h+var_4654]
54684 89 84 24 90 11 00 00                mov     [esp+4658h+var_34C8], eax
5468B 8B 0C 24                            mov     ecx, [esp+4658h+var_4658]
5468E 89 8C 24 24 23 00 00                mov     [esp+4658h+var_2334], ecx
54695 E8 26 B1 02 00                      call    main_kbbjdjedhblphedhbnam_func4
5469A 8B 44 24 04                         mov     eax, [esp+4658h+var_4654]
```

String Decryption Functions

As mentioned, the string encryption is fairly basic, XORing the contents of two arrays together to retrieve the final string.

```
.text:0057BC2A                lea     eax, unk_63113A
.text:0057BC30                mov     [esp+20Ch+var_208], eax
.text:0057BC34                mov     [esp+20Ch+var_204], 1AAh
.text:0057BC3C                call    runtime_stringtoslicebyte
.text:0057BC41                mov     eax, [esp+20Ch+var_200]
.text:0057BC45                mov     [esp+20Ch+var_4], eax
.text:0057BC4C                mov     ecx, [esp+20Ch+var_1FC]
.text:0057BC50                mov     [esp+20Ch+var_1F4], ecx
.text:0057BC54                mov     [esp+20Ch+decrypted_buffer], 0
.text:0057BC5C                lea     edi, [esp+20Ch+decrypted_buffer+2]
.text:0057BC60                xor     eax, eax
.text:0057BC62                call    loc_44AEA6
.text:0057BC67                lea     eax, [esp+20Ch+var_1EE]
.text:0057BC6B                mov     [esp+20Ch+var_20C], eax
.text:0057BC6E                lea     eax, unk_630F90
.text:0057BC74                mov     [esp+20Ch+var_208], eax
.text:0057BC78                mov     [esp+20Ch+var_204], 1AAh
.text:0057BC80                call    runtime_stringtoslicebyte
```

Loading the two arrays for decryption

```
.text:0057BCA3 decryptionLoop:                        ; CODE XREF: decryptAESPublicKey+9A↑j
.text:0057BCA3                cmp     ebp, edx
.text:0057BCA5                jge     short loc_57BCC2
.text:0057BCA7                movzx   esi, byte ptr [ebx+ebp]
.text:0057BCAB                lea     esi, [esi+ebp*2]
.text:0057BCAE                cmp     ebp, eax
.text:0057BCB0                jnb     short loc_57BD03
.text:0057BCB2                movzx   edi, byte ptr [ecx+ebp]
.text:0057BCB6                xor     esi, edi
.text:0057BCB8                cmp     ebp, 1AAh
.text:0057BCBE                jb      short loc_57BC9C
.text:0057BCC0                jmp     short loc_57BD03
```

XORing the arrays together to get the decrypted string

Due to the simplicity of the algorithm, we can develop a simple Python script utilising some regular expressions to locate and decrypt 90% of the encrypted strings within the binary! The script can be found at the end of this post.

```
 startDecryptFunction = b"\x64\x8B\x0D\x14\x00\x00\x00"
sliceStr = b"(" + b"\x8D.....\x89.\$\x04\xC7\x44\$\b...." + b")"
```

```
xorLoop = b"\x0F\xB6<\)1\xFE(\x83|\x81)\xFD"
```

With the majority of the strings decrypted, let's continue the analysis!

Check Environment

One of the first operations I perform when analyzing malware in GO is to jump to the **main.init** function.

The **main.init** is generated for each package by the compiler to initialise all other packages that this sample relies on, as well as the global variables; analysing this function is very important because it allows us to understand a large amount of the malware's functionality and speed up subsequent analysis.

In the **main.init** function we can find, for example, references to encryption: **AES, RSA, SHA1, X509**. In addition, there are several functions for decryption of strings and function names.

```
005CDD3C    call    crypto_aes_init
005CDD41    call    crypto_cipher_init
005CDD46    call    crypto_rand_init
005CDD4B    call    crypto_rsa_init
005CDD50    call    crypto_sha1_init
005CDD55    call    crypto_x509_init
005CDD5A    call    encoding_pem_init
005CDD5F    call    io_init
005CDD64    call    log_init
005CDD69    call    os_init
005CDD6E    call    path_filepath_init
```

Initialisations performed within main.init function

```
8 2C FB FF FF                       call    decryptCreateToolhelp32Snapshot
B 05 F8 92 7C 00                    mov     eax, dword_7C92F8
B 0C 24                             mov     ecx, [esp+10h+var_10]
B 54 24 04                          mov     edx, [esp+10h+var_C]
9 04 24                             mov     [esp+10h+var_10], eax
9 4C 24 04                          mov     [esp+10h+var_C], ecx
9 54 24 08                          mov     [esp+10h+var_8], edx
8 0F 91 F3 FF                       call    syscall___ptr_LazyDLL__NewProc
B 05 50 A9 7D 00                    mov     eax, dword_7DA950
B 4C 24 0C                          mov     ecx, [esp+10h+var_4]
5 C0                                test    eax, eax
F 85 A5 00 00 00                    jnz     loc_523AD8
9 0D 08 93 7C 00                    mov     fCreateToolhelp32Snapshot, ecx

                        loc_523A39:                         ; CODE XREF: FindProcess+18A↓j
8 F2 FB FF FF                       call    decryptProcess32FirstW
B 05 F8 92 7C 00                    mov     eax, dword_7C92F8
B 0C 24                             mov     ecx, [esp+10h+var_10]
B 54 24 04                          mov     edx, [esp+10h+var_C]
9 04 24                             mov     [esp+10h+var_10], eax
9 4C 24 04                          mov     [esp+10h+var_C], ecx
9 54 24 08                          mov     [esp+10h+var_8], edx

00122E2B 00523A2B: FindProcess+CB (Synchronized with Pseudocode-A)
```

API function decryption and loading through NewProc

Now we move on to analyze the **main.main** function. One of the first activities the malware performs is to check the environment before continuing with encryption.

```
00552A30
00552A30                mov     ecx, large fs:14h
00552A37                mov     ecx, [ecx+0]
00552A3D                cmp     esp, [ecx+8]
00552A40                jbe     loc_552BE1
00552A46                sub     esp, 48h
00552A49                call    time_Now
00552A4E                lea     edi, [esp+48h+var_14]
00552A52                mov     esi, esp
00552A54                call    loc_44B3EE
00552A59                mov     eax, STRING_EKANS
00552A5F                mov     ecx, SIZE_EKANS
00552A65                mov     [esp+48h+var_48], ecx
00552A68                mov     [esp+48h+decryptedLenght], eax
00552A6C                call    CheckEnvironment
00552A71                movzx   eax, byte ptr [esp+48h+decryptedString]
00552A76                test    al, al
00552A78                jz      loc_552BBB
```

Call to CheckEnvironment within main.main

The function **CheckEnvironment** starts by attempting to resolve the hostname **mds.honda.com** and compare the returned value with **172[.]108[.]71[.]153**. This check is used to confirm that the infected machine is part of the correct domain. In fact, it is important to remember that this ransomware is deployed at the end of the infection chain by other loaders, and thus it is likely custom-built for the victim.

```
:00553D50                mov     ecx, large fs:14h
:00553D57                mov     ecx, [ecx+0]
:00553D5D                cmp     esp, [ecx+8]
:00553D60                jbe     loc_553EB0
:00553D66                sub     esp, 4Ch
:00553D69                lea     eax, aMdsHondaCom ; "MDS.HONDA.COM"
:00553D6F                mov     [esp+4Ch+var_4C], eax
:00553D72                mov     [esp+4Ch+var_48], 0Dh
:00553D7A                call    net_LookupIP
:00553D7F                mov     eax, [esp+4Ch+var_44]
:00553D83                mov     ecx, [esp+4Ch+var_38]
:00553D87                mov     edx, [esp+4Ch+var_40]
:00553D8B                test    ecx, ecx
:00553D8D                jnz     loc_553EA7
:00553D93                test    edx, edx
:00553D95                jz      loc_553EA7
:00553D9B                mov     [esp+4Ch+var_2C], edx
```

Resolving hostname via DNS

```
:00553DF7 ; ---------------------------------------------------------------------------
:00553DF7
:00553DF7 loc_553DF7:                            ; CODE XREF: checkIPAddress+9E↑j
:00553DF7                mov     [esp+4Ch+var_4C], ecx
:00553DFA                lea     ecx, a17010871153 ; "170.108.71.153"
:00553E00                mov     [esp+4Ch+var_48], ecx
:00553E04                mov     [esp+4Ch+var_44], eax
:00553E08                call    runtime_memequal
:00553E0D                movzx   eax, byte ptr [esp+4Ch+var_40]
:00553E12                test    al, al
:00553E14                jz      short loc_553DF0
:00553E16                mov     eax, 1
:00553E1B                jmp     short loc_553DA5
:00553E1D ; ---------------------------------------------------------------------------
```

Comparing resolved address to hardcoded address

After the first environment check, the malware executes the API calls **CoInitializeEx**, **CoInitializeSecurity** and **CoCreateInstance** to instantiate an object of

the **SWbemLocator** interface. Using the **SWbemLocator** object, SNAKE then invokes the method **SWbemLocator::ConnectServer** and obtains a pointer to an **SWbemServices** object. Finally, with this object, it will execute **ExecQuery** with the following query:

```
select DomainRole from Win32_ComputerSystem
```

In an attempt to determine whether the infected computer is a server or a workstation.

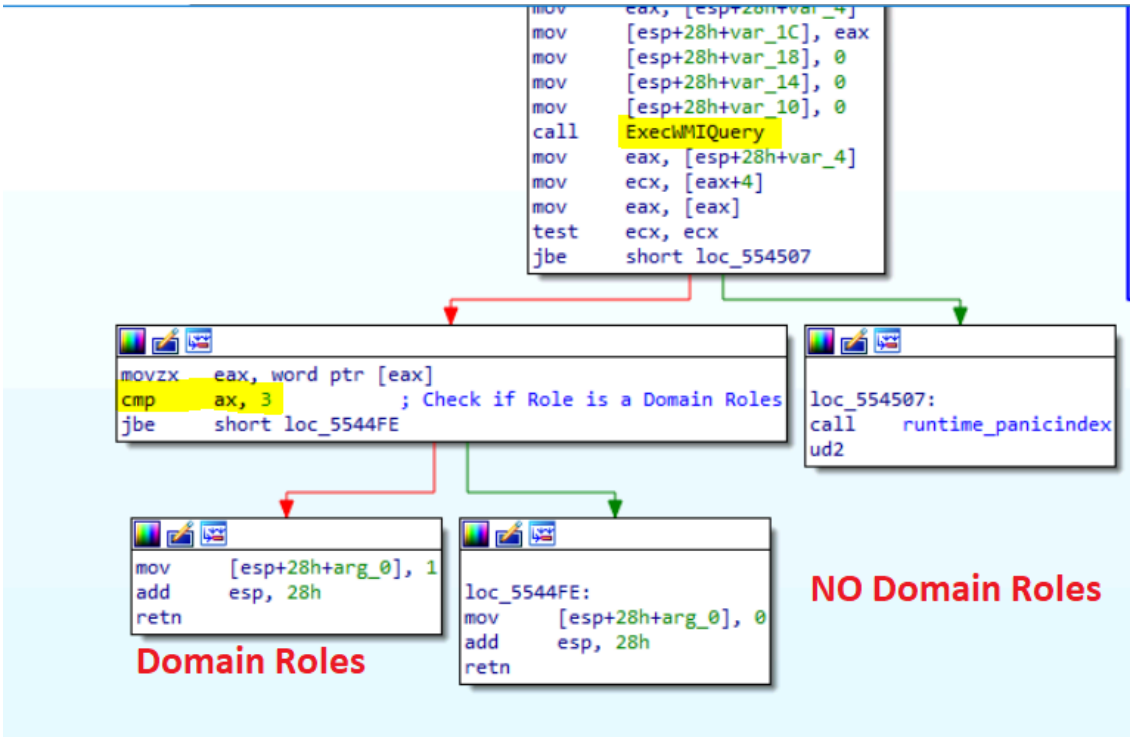
```
:0051CB50
:0051CB50               mov     ecx, large fs:14h
:0051CB57               mov     ecx, [ecx+0]
:0051CB5D               cmp     esp, [ecx+8]
:0051CB60               jbe     loc_51CC44
:0051CB66               sub     esp, 7Ch
:0051CB69               lea     eax, [esp+7Ch+var_24]
:0051CB6D               mov     [esp+7Ch+var_7C], eax
:0051CB70               lea     eax, aWbemscriptingS ; "WbemScripting.SWbemLocator"
:0051CB76               mov     [esp+7Ch+var_78], eax
:0051CB7A               mov     [esp+7Ch+var_74], 1Ah
:0051CB82               call    runtime_stringtoslicebyte
:0051CB87               mov     eax, [esp+7Ch+var_70]
:0051CB8B               mov     [esp+7Ch+var_4], eax
:0051CB8F               mov     ecx, [esp+7Ch+var_6C]
:0051CB93               mov     [esp+7Ch+var_64], ecx
:0051CB97               mov     [esp+7Ch+var_5E], 0
:0051CB9F               lea     edi, [esp+7Ch+var_5E+2]
```

Decryption of object used to perform WMI query

```
:0057F3B4               jbe     loc_57F4A4
:0057F3BA               sub     esp, 8Ch
:0057F3C0               lea     eax, [esp+8Ch+var_4F]
:0057F3C4               mov     [esp+8Ch+var_8C], eax
:0057F3C7               lea     eax, aSelectDomainro ; "select DomainRole FROM Win32_ComputerSy"...
:0057F3CD               mov     [esp+8Ch+var_88], eax
:0057F3D1               mov     [esp+8Ch+var_84], 2Bh ; '+'
:0057F3D9               call    runtime_stringtoslicebyte
:0057F3DE               mov     eax, [esp+8Ch+var_80]
:0057F3E2               mov     [esp+8Ch+var_4], eax
```

Decryption of WMI query to retrieve DomainRole

After making the query, the malware only continues execution if the **DomainRole** value is equal to or less than 3.



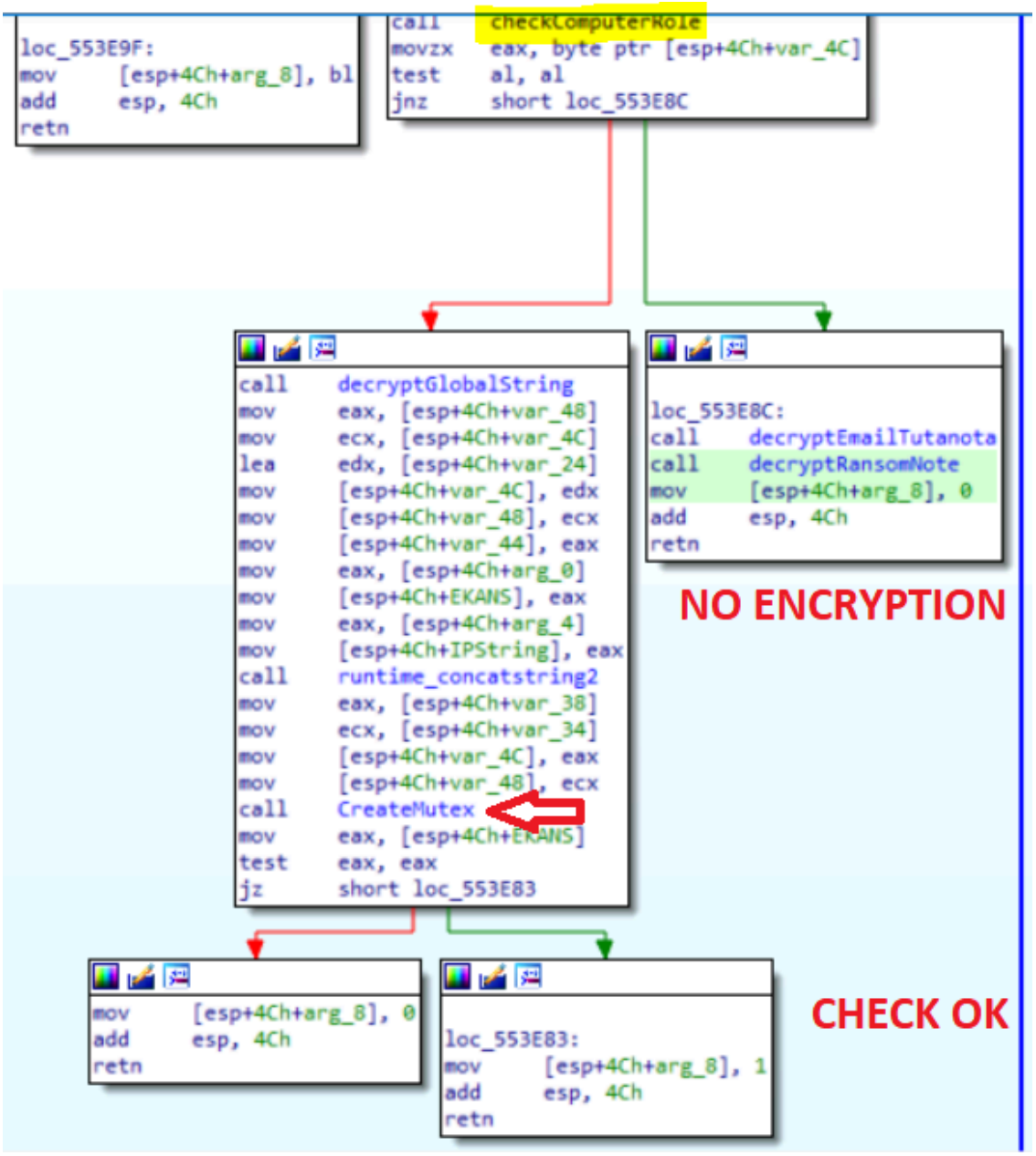
Execution of WMI query and checking of result

According to [Microsoft documentation], the integers returned by the call correspond to different values:

VALUE	MEANING
0	Standalone Workstation
1	Member Workstation
2	Standalone Server
3	Member Server
4	Backup Domain Controller
5	Primary Domain Controller

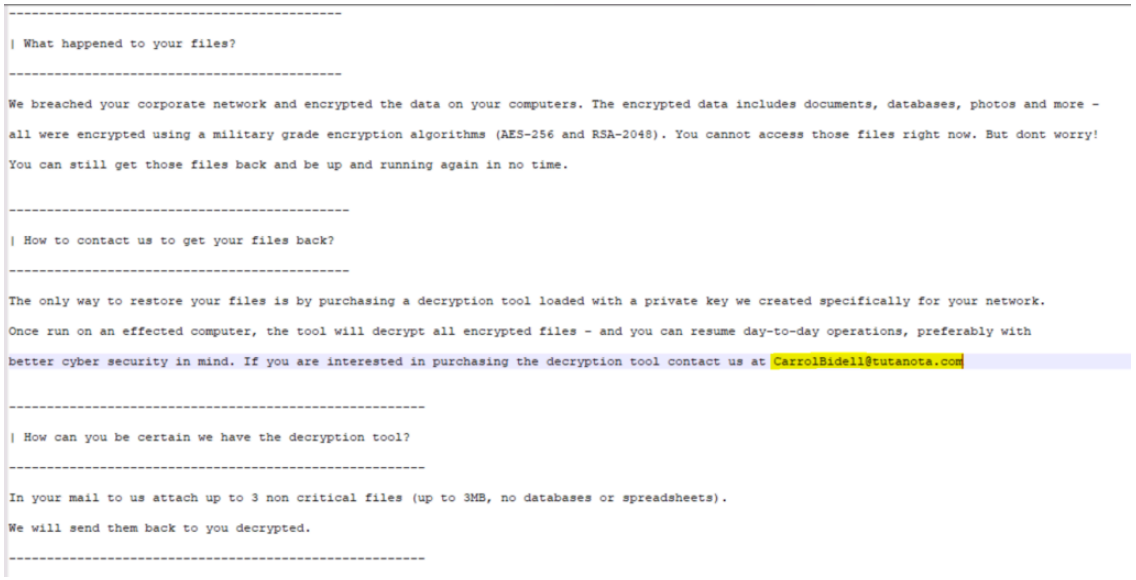
Therefore, the malware performs the infection only if the role obtained of the computer is **Standalone Workstation**, **Member Workstation**, **Standalone Server**, or **Member Server.**

If this check is successful, the mutex **Global\EKANS** is created, and presuming the mutex is created successfully, the sample continues executing.



Execution flow depending on the result of the DomainRole

If, on the other hand, the computer role is either a backup domain controller or primary domain controller, a ransom note is dropped to C:\Users\Public\Desktop, and files are not encrypted. Within the ransom note is an email on how to contact the threat actors, with the email used in this sample being **CarrolBidell@tutanota.com**.

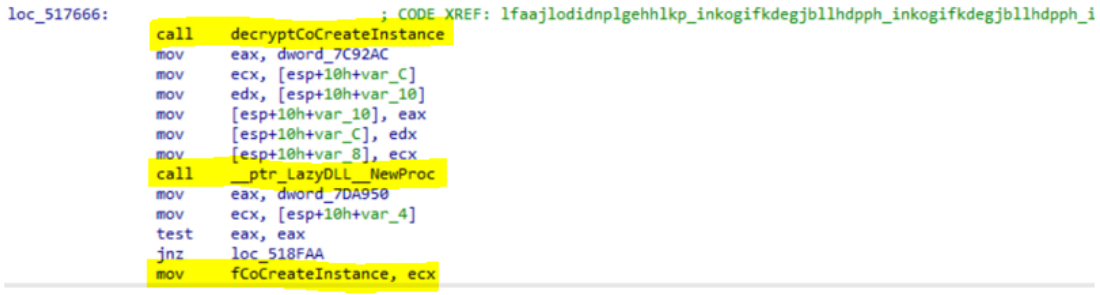


```
-----------------------------------------------

| What happened to your files?

-----------------------------------------------

We breached your corporate network and encrypted the data on your computers. The encrypted data includes documents, databases, photos and more -

all were encrypted using a military grade encryption algorithms (AES-256 and RSA-2048). You cannot access those files right now. But dont worry!

You can still get those files back and be up and running again in no time.

-----------------------------------------------

| How to contact us to get your files back?

-----------------------------------------------

The only way to restore your files is by purchasing a decryption tool loaded with a private key we created specifically for your network.

Once run on an effected computer, the tool will decrypt all encrypted files - and you can resume day-to-day operations, preferably with

better cyber security in mind. If you are interested in purchasing the decryption tool contact us at CarrolBidell@tutanota.com

-----------------------------------------------------------

| How can you be certain we have the decryption tool?

-----------------------------------------------------------

In your mail to us attach up to 3 non critical files (up to 3MB, no databases or spreadsheets).

We will send them back to you decrypted.

-----------------------------------------------------------
```

Ransom note

When analyzing Go developed programs, two functions to pay attention to are **NewLazyDll** (essentially LoadLibrary), and **NewProc** (as you may have guessed, basically GetProcAddress). With the use of Gobfuscate to obfuscate this sample, the names of the libraries and API functions to be passed to the described functions. Pointers to the loaded libraries/functions are stored within DWORDs for later reference.

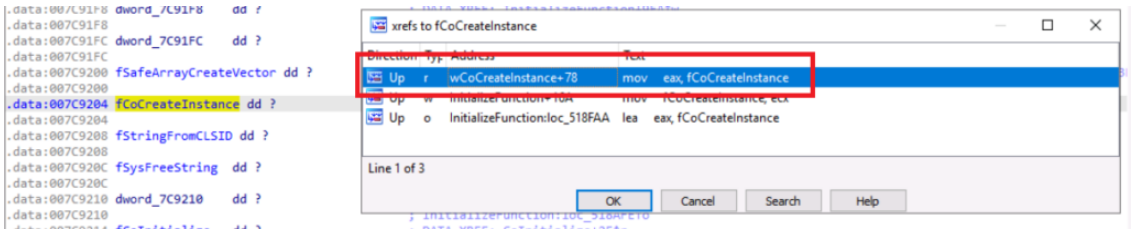
For example, we can see in the sample we have the function that performs the decryption of a function name before calling **LazyDLL.NewProc**:



```
loc_517666:                             ; CODE XREF: lfaajlodidnplgehhlkp_inkogifkdegjbllhdpph_inkogifkdegjbllhdpph_i
call    decryptCoCreateInstance
mov     eax, dword_7C92AC
mov     ecx, [esp+10h+var_C]
mov     edx, [esp+10h+var_10]
mov     [esp+10h+var_10], eax
mov     [esp+10h+var_C], edx
mov     [esp+10h+var_8], ecx
call    __ptr_LazyDLL__NewProc
mov     eax, dword_7DA950
mov     ecx, [esp+10h+var_4]
test    eax, eax
jnz     loc_518FAA
mov     fCoCreateInstance, ecx
```

API function decryption and loading via NewProc

A pointer to the function is saved in a DWORD, so that we can trace to see where the function is called within the binary.



Cross references for API functions

```
.text:0050926E                  mov     [ecx+0Ch], eax
.text:00509271                  mov     eax, [esp+2Ch+var_4]
.text:00509275                  mov     [ecx+10h], eax
.text:00509278                  mov     eax, fCoCreateInstance
.text:0050927E                  mov     [esp+2Ch+var_2C], eax
.text:00509281                  mov     [esp+2Ch+var_28], ecx
.text:00509285                  mov     [esp+2Ch+var_24], 5
.text:0050928D                  mov     [esp+2Ch+var_20], 5
.text:00509295                  call    LazyProc_Call
.text:0050929A                  mov     eax, [esp+2Ch+var_1C]
.text:0050929E                  test    eax, eax
.text:005092A0                  jnz     short loc_5092BC
.text:005092A2                  xor     eax, eax
.text:005092A4                  mov     ecx, eax
.text:005092A6
```

Execution of previously loaded API function

Endpoint Isolation

After the function *CheckEnvironment* has finished, the strings "**netsh advfirewall set allprofiles state on**" and "**netsh advfirewall set allprofiles firewallpolicy blockinbound,blockoutbound**" are decrypted and executed via **cmd.run**. The first command enables Windows Firewall for all network profiles, while the second blocks all incoming and outgoing connections. This is fairly unusual behaviour for ransomware, which typically performs lateral movement across a network to infect additional machines.

```
.text:00554120                  call    decryptNetSh2   ; netsh
.text:00554125                  mov     eax, [esp+98h+var_94]
.text:00554129                  mov     [esp+98h+var_6C], eax
.text:0055412D                  mov     ecx, [esp+98h+var_98]
.text:00554130                  mov     [esp+98h+var_54], ecx
.text:00554134                  call    decryptAdvFirewall2 ; advfirewall
.text:00554139                  mov     eax, [esp+98h+var_94]
.text:0055413D                  mov     [esp+98h+var_70], eax
.text:00554141                  mov     ecx, [esp+98h+var_98]
.text:00554144                  mov     [esp+98h+var_58], ecx
.text:00554148                  call    decryptSet2     ; set
.text:0055414D                  mov     eax, [esp+98h+var_94]
.text:00554151                  mov     [esp+98h+var_74], eax
.text:00554155                  mov     ecx, [esp+98h+var_98]
.text:00554158                  mov     [esp+98h+var_5C], ecx
.text:0055415C                  call    decryptAllProfiles2 ; allprofiles
.text:00554161                  mov     eax, [esp+98h+var_94]
.text:00554165                  mov     [esp+98h+var_78], eax
.text:00554169                  mov     ecx, [esp+98h+var_98]
.text:0055416C                  mov     [esp+98h+var_60], ecx
.text:00554170                  call    decryptFirewallPolicy ; firewallpolicy
.text:00554175                  mov     eax, [esp+98h+var_98]
```

Decryption of netsh commands to alter the firewall

```
:00554445                       lea     eax, [esp+6Ch+var_28]
:00554449                       mov     [esp+8], eax
:0055444D                       mov     dword ptr [esp+0Ch], 5
:00554455                       mov     dword ptr [esp+10h], 5
:0055445D                       call    os_exec_Command
:00554462                       mov     eax, [esp+6Ch+var_58]
:00554466                       mov     [esp+6Ch+var_6C], eax
:00554469                       call    os_exec___ptr_Cmd__Run
:0055446E                       add     esp, 6Ch
:00554471                       retn
```

Execution of above commands through os.exec

Terminate Process And Services

Prior to encryption, the ransomware terminates a number of processes, to reduce the amount of interference with the encryption (for example any open file handles), as well as disable any running EDR/SIEM software on the machine.

```
.text:0055462B                mov     [esp+4658h+var_2328], eax
.text:00554632                mov     [esp+4658h+var_2324], eax
.text:00554639                mov     [esp+4658h+var_2320], eax
.text:00554640                mov     [esp+4658h+var_231C], eax
.text:00554647                call    decryptccflic0_exe ; ccflic0.exe
.text:0055464C                mov     eax, [esp+4658h+var_4654]
.text:00554650                mov     [esp+4658h+var_34C0], eax
.text:00554657                mov     ecx, [esp+4658h+var_4658]
.text:0055465A                mov     [esp+4658h+var_232C], ecx
.text:00554661                call    decryptccflic4_exe ; ccflic4.exe
.text:00554666                mov     eax, [esp+4658h+var_4654]
.text:0055466A                mov     [esp+4658h+var_34C4], eax
.text:00554671                mov     ecx, [esp+4658h+var_4658]
.text:00554674                mov     [esp+4658h+var_2330], ecx
.text:0055467B                call    decrypthealthservice_exe ; healthservice.exe
.text:00554680                mov     eax, [esp+4658h+var_4654]
.text:00554684                mov     [esp+4658h+var_34C8], eax
.text:0055468B                mov     ecx, [esp+4658h+var_4658]
.text:0055468E                mov     [esp+4658h+var_2334], ecx
.text:00554695                call    decryptilicensesvc_exe ; ilicensesvc.exe
.text:0055469A                mov     eax, [esp+4658h+var_4654]
.text:0055469E                mov     [esp+4658h+var_34CC], eax
.text:005546A5                mov     ecx, [esp+4658h+var_4658]
```

Decryption of target processes to be terminated

Processes are terminated using **syscall.OpenProcess** and **syscall.TerminateProcess** calls. In order for SNAKE to retrieve the PIDs of the target processes, the usual calls of CreateToolhelp32Snapshot, Process32FirstW, and Process32NextW are performed.

```
:00554536                sub     esp, 1Ch
:00554539                mov     [esp+1Ch+arg_8], 0
:00554541                mov     [esp+1Ch+arg_C], 0
:00554549                mov     [esp+1Ch+var_1C], 1
:00554550                mov     byte ptr [esp+1Ch+var_18], 0
:00554555                mov     eax, [esp+1Ch+arg_0]
:00554559                mov     [esp+1Ch+var_14], eax
:0055455D                call    syscall_OpenProcess
:00554562                mov     eax, [esp+1Ch+var_10]
:00554566                mov     ecx, [esp+1Ch+var_8]
:0055456A                mov     edx, [esp+1Ch+var_C]
:0055456E                test    edx, edx
:00554570                jnz     short loc_5545CC
:00554572                mov     [esp+1Ch+var_4], eax
:00554576                mov     [esp+1Ch+var_14], eax
:0055457A                mov     [esp+1Ch+var_1C], 0Ch
:00554581                lea     ecx, off_632340
:00554587                mov     [esp+1Ch+var_18], ecx
:0055458B                call    runtime_deferproc
:00554590                test    eax, eax
:00554592                jnz     short loc_5545C2
:00554594                mov     eax, [esp+1Ch+var_4]
:00554598                mov     [esp+1Ch+var_1C], eax
:0055459B                mov     eax, [esp+1Ch+arg_4]
:0055459F                mov     [esp+1Ch+var_18], eax
:005545A3                call    syscall_TerminateProcess
:005545A8                mov     eax, [esp+1Ch+var_10]
:005545AC                mov     ecx, [esp+1Ch+var_14]
:005545B0                mov     [esp+1Ch+arg_8], ecx
:005545B4                mov     [esp+1Ch+arg_C], eax
:005545B8                nop
```

Terminating processes with OpenProcess and TerminateProcess

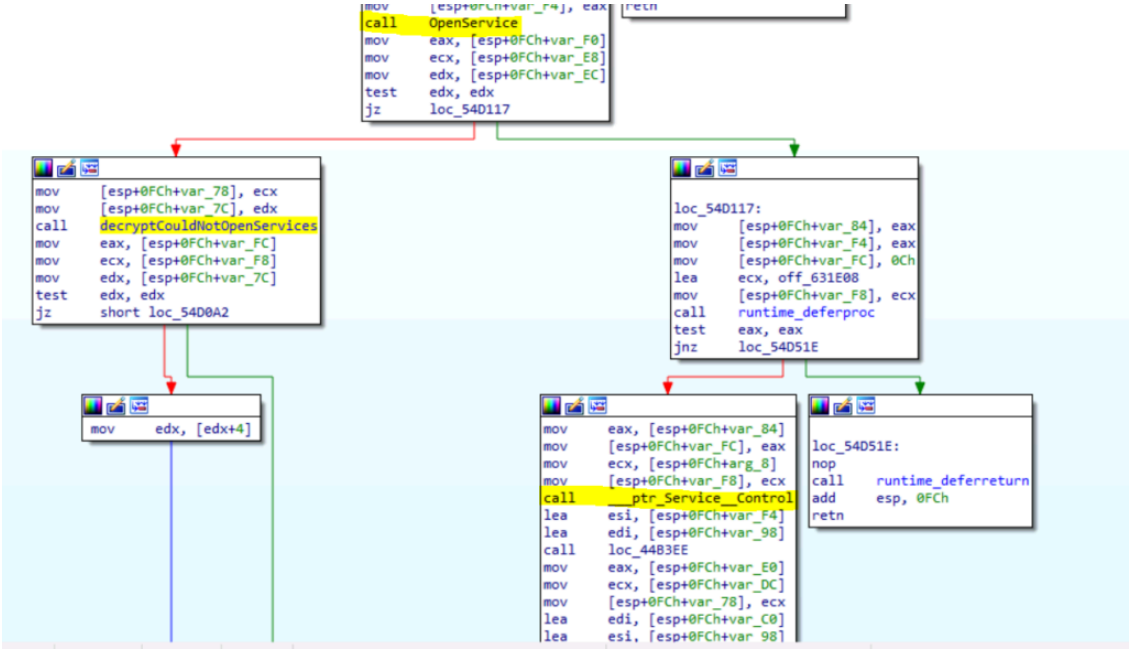
In addition to terminating processes, the ransomware stops more than 200 services related to EDR, SIEM, AV, etc.

```
.text:0054D58A                jbe     loc_551CFE
.text:0054D590                sub     esp, 13ACh
.text:0054D596                mov     eax, 0
.text:0054D59B                lea     edi, [esp+13ACh+var_9F8]
.text:0054D5A2                call    loc_44AF08
.text:0054D5A7                call    decryptAcronis  ; Acronis VSS Provider
.text:0054D5AC                mov     eax, [esp+13ACh+var_13A8]
.text:0054D5B0                mov     [esp+13ACh+var_115C], eax
.text:0054D5B7                mov     ecx, [esp+13ACh+var_13AC]
.text:0054D5BA                mov     [esp+13ACh+var_C98], ecx
.text:0054D5C1                call    decryptEnterpriseClientService ; Enterprise Client Service
.text:0054D5C6                mov     eax, [esp+13ACh+var_13A8]
.text:0054D5CA                mov     [esp+13ACh+var_1160], eax
.text:0054D5D1                mov     ecx, [esp+13ACh+var_13AC]
.text:0054D5D4                mov     [esp+13ACh+var_C9C], ecx
.text:0054D5DB                call    decryptSophosAgent ; Sophos Agent
.text:0054D5E0                mov     eax, [esp+13ACh+var_13A8]
.text:0054D5E4                mov     [esp+13ACh+var_1164], eax
```

Decryption of target service names to be terminated

In order to terminate services, the following API function calls are made:

- **OpenSCManagerA**: gets a service control manager handle for subsequent calls.
- **EnumServicesStatusEx**: enumeration of services.
- **OpenServiceW**: gets a service handle for subsequent calls.
- **QueryServiceStatusEx**: check the status of services.
- **ControlService:** used to stop the service (flag SERVICE_CONTROL_STOP).



Termination of services using OpenService and ControlService

Shadow Copy Deletion

The ransomware executes the WMI query **“SELECT * FROM Win32_ShadowCopy”** to get the IDs of Shadow Copies and will always use WMI for deletion (remember that there are many ways to perform shadow copy deletion).

In addition to the **Wbemscripting.SWbemLocator** object, **WbemScripting.SWbemNamedValueSet** is also created.

For deletion, SNAKE uses the **DeleteInstance** method by passing the ID of previously obtained Shadow Copies.

```
.text:0057C037                mov     ecx, [ecx+0]
.text:0057C03D                cmp     esp, [ecx+8]
.text:0057C040                jbe     loc_57C122
.text:0057C046                sub     esp, 80h
.text:0057C04C                lea     eax, [esp+80h+var_24]
.text:0057C050                mov     [esp+80h+var_80], eax
.text:0057C053                lea     eax, aWbemscriptingS_0 ; "WbemScripting.SWbemNamedValueSet"
.text:0057C059                mov     [esp+80h+var_7C], eax
.text:0057C05D                mov     [esp+80h+var_78], 20h ; ' '
.text:0057C065                call    runtime_stringtoslicebyte
.text:0057C06A                mov     eax, [esp+80h+var_74]
.text:0057C06E                mov     [esp+80h+var_4], eax
.text:0057C072                mov     ecx, [esp+80h+var_70]
```

Decryption of WbemScripting.SWbemNamedValueSet

```
xt:0057D066    sub     esp, 80h
xt:0057D06C    lea     eax, [esp+80h+var_24]
xt:0057D070    mov     [esp+80h+var_80], eax
xt:0057D073    lea     eax, aSelectFromWin3 ; "SELECT * FROM Win32_ShadowCopy"
xt:0057D079    mov     [esp+80h+var_7C], eax
xt:0057D07D    mov     [esp+80h+var_78], 1Eh
xt:0057D085    call    runtime_stringtoslicebyte
xt:0057D08A    mov     eax, [esp+80h+var_74]
xt:0057D08E    mov     [esp+80h+var_4], eax
xt:0057D092    mov     ecx, [esp+80h+var_70]
xt:0057D096    mov     [esp+80h+var_68], ecx
xt:0057D09A    mov     [esp+80h+var_62], 0
xt:0057D0A2    lea     edi, [esp+80h+var_62+2]
xt:0057D0A6    xor     eax, eax
```

Decryption of WMI Query

```
xt:0057CB86    sub     esp, 6Ch
xt:0057CB89    lea     eax, [esp+6Ch+var_24]
xt:0057CB8D    mov     [esp+6Ch+var_6C], eax
xt:0057CB90    lea     eax, aRootCimv2 ; "root\\cimv2"
xt:0057CB96    mov     [esp+6Ch+var_68], eax
xt:0057CB9A    mov     [esp+6Ch+var_64], 0Ah
xt:0057CBA2    call    runtime_stringtoslicebyte
xt:0057CBA7    mov     eax, [esp+6Ch+var_60]
xt:0057CBAB    mov     [esp+6Ch+var_4], eax
xt:0057CBAF    mov     ecx, [esp+6Ch+var_5C]
```

Decryption of WMI namespace

Encryption Process

SNAKE first encrypts all the various files by initializing 8 go-routines (**runtime.newproc**), before beginning to rename the files.

The offset of the function that does the encryption is passed to **runtime.newproc** (**OffsetStartEncryption**).

```
while ( v6 <= 8 )
{
  v11 = v6;
  v9 = v5;
  v7 = a2;
  runtime_newproc(16, (char)&OffsetStartEncryption);
  v6 = v11 + 1;
  v5 = v13;
}
startRenameFile(a3, a4, v12, v5, v7, v9);
runtime_closechan(v12);
sync___ptr_WaitGroup__Wait(v13);
renameFile(v8, v10);
sync___ptr_WaitGroup__Done(a5);
```

Initialisation of go-routines prior to file renaming function

Before beginning encryption of the file, it's checked that it has not already been encrypted by checking for the presence of the string **EKANS** at the end of the file.

```
os___ptr_File__Read(file, v15, sizeFileReaded, v13);
if ( v11 )
{
  if ( dword_7C9848 != v11 )
    return v13;
  runtime_ifaceeq();
  if ( !v3 )
    return v13;
}
if ( sizeFileReaded == SIZE_STRING_EKANS )
{
  runtime_memequal(v15, STRING_EKANS);
  if ( v3 )
    return v8;
}
os___ptr_File__Seek(file, 0, 0, 0, v5, v8);
return v14;
}
```

Checking if file is already encrypted

If the file hasn't yet been encrypted and the files are among those to be encrypted (there is an allowlist and a denylist), encryption is initiated, which takes care of:

- Generating AES key for each file; this key is encrypted with an RSA public key in OAEP Mode.
- Encryption of file via AES in CTR mode, with Random Key (32 bytes) and Random IV (16 bytes).
- A random 5 character is appended to the file extension of encrypted files.
- Adds data to the end of the file: encrypted AES Key, IV and EKANS string.

```
sliceIV = runtime_makeslice64(dword_5F0E80, 16, 0, 16, 0, v17, v21);
pIV = IV;
crypto_rand_Read(IV, IVSize, sliceIV);
helpFunction2(fileb, IV);
sliceKeyAES = runtime_makeslice(dword_5F0E80, 32, 32, cpSliceKey, fileb);
pAESKeySize = AES_KEY_SIZE;
pAESKey = cpSliceKeya;
pSliceKeyAES = sliceKeyAES;
crypto_rand_Read(cpSliceKeya, AES_KEY_SIZE, sliceKeyAES);
helpFunction2(AES_KEY_SIZE, sliceKeyAES);
EncryptFileViaCTR(pFile2, cpSliceKeya, AES_KEY_SIZE, sliceKeyAES, pIV, IVSize, sliceIV);
if ( !sliceIV )
{
  encryptRSAKey(RSAKey, pAESKey, pAESKeySize, pSliceKeyAES);
  ((void (*)(void))loc_44AF07)();
  ((void (*)(void))loc_44B3C6)();
  AddToEndOfFile(cpFile2a, fileClose2, cpAESKeySize, cpSliceKeyb, file, v20, v23, 0, v25, pFile2);
}
```

Key generation, encryption, and metadata being added to file

After instantiating the CTR cipher with **cipher.NewCTR,** encryption is performed with the **XORKeyStream** method of that class.

The function reads 0x19000 bytes at a time and after encryption the file is rewritten using **WriteAt**.

```
.text:00551F29                mov     eax, [esp+70h+arg_18]
.text:00551F30                mov     [esp+70h+var_60], eax
.text:00551F34                call    crypto_cipher_NewCTR
.text:00551F39                mov     eax, [esp+70h+var_58]
.text:00551F3D                mov     [esp+70h+var_18], eax
.text:00551F41                mov     ecx, [esp+70h+var_5C]
.text:00551F45                mov     [esp+70h+var_1C], ecx
.text:00551F49                lea     edx, dword_5E0E80
.text:00551F4F                mov     [esp+70h+var_70], edx
.text:00551F52                mov     [esp+70h+var_6C], 19000h
.text:00551F5A                mov     [esp+70h+var_68], 19000h
.text:00551F62                call    runtime_makeslice
.text:00551F67                mov     eax, [esp+70h+var_64]
.text:00551F6B                mov     [esp+70h+var_14], eax
.text:00551F6F                mov     ecx, [esp+70h+var_5C]
.text:00551F73                mov     [esp+70h+var_34], ecx
.text:00551F77                mov     edx, [esp+70h+var_60]
.text:00551F7B                mov     [esp+70h+var_38], edx
```

Generating the buffer to hold bytes read from the file

```
.text:00552073                mov     ebp, [esp+70h+var_18]
.text:00552077                mov     [esp+70h+var_70], ebp
.text:0055207A                call    esi             ; XORKeyStream
.text:0055207C                mov     eax, [esp+70h+pFile]
.text:00552080                mov     [esp+70h+var_70], eax
.text:00552083                mov     ecx, [esp+70h+var_28]
.text:00552087                mov     [esp+70h+var_6C], ecx
.text:0055208B                mov     ecx, [esp+70h+var_4C]
.text:0055208F                mov     [esp+70h+var_68], ecx
.text:00552093                mov     ecx, [esp+70h+var_48]
.text:00552097                mov     [esp+70h+var_64], ecx
.text:0055209B                mov     ecx, dword ptr [esp+70h+var_40]
.text:0055209F                mov     [esp+70h+var_60], ecx
.text:005520A3                mov     edx, dword ptr [esp+70h+var_40+4]
.text:005520A7                mov     [esp+70h+var_5C], edx
.text:005520AB                call    os___ptr_File__WriteAt
.text:005520B0                mov     eax, [esp+70h+var_44]
.text:005520B4                mov     ecx, eax
```

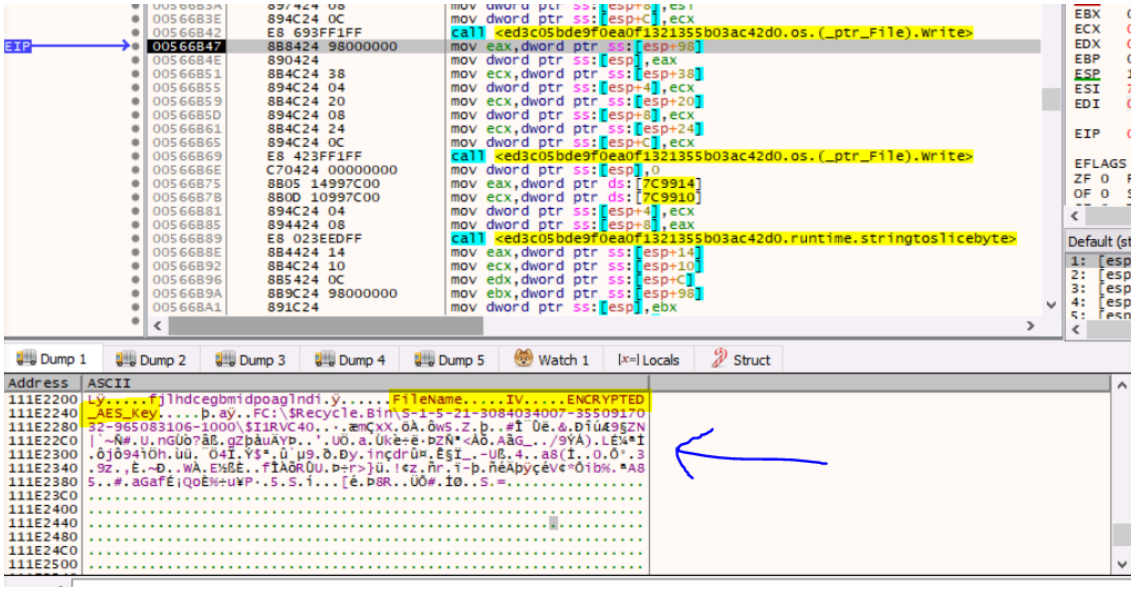
Encrypting buffer data and overwriting file

After finishing the encryption, three more writes are performed on the file:

```
os___ptr_File__Write(a10, *v40 + (v12 & ((int)(v12 - v40[2]) >> 31)), v11 - v12, v40[2] - v12, v24, v28, 0);
os___ptr_File__Write(a10, v41, v35, v36, v25, v29, v31);
v21 = runtime_stringtoslicebyte(0, STRING_EKANS, SIZE_STRING_EKANS);
os___ptr_File__Write(a10, v21, v26, v30, v26, v30, v32);
```

Adding metadata to the file

It’s easy to see that in the last one the string **EKANS** is written (which is used to determine if the file has already been encrypted), while it is much more complex to figure out what is written in the first two. As a result, let’s jump over to a debugger.



Observing metadata being written to end of file using a debugger

The first write adds the following to the file:

- The encrypted AES Key
- The random IV
- The path of encrypted file

The AES Key for each file is encrypted with a public RSA key. After decryption, the public key is parsed with **pem.decode** and **x509.ParsePKCS1PublicKey**.

```
.text:00552A83                call    decryptAESPublicKey ; -----BEGIN RSA PUBLIC KEY-----.MIIBCgKCAQEAt1GCKUHXITsiWc1d8V0vo
.text:00552A83                                            ; 1117C3C0  1Y9Jm18RDZEmMS6OkHI7pZT0RHAThlR.BFITZY9bXrl6RFdUwmIX0WYn5ZqIlhLA
.text:00552A83                                            ; 1117C400  Ee1cqd8RpJ/KK2OeiTn0CJ1CGmOOJvfm.5rFa8whVAU9cnh/iVCcf+aEHJVcHhzB
.text:00552A83                                            ; 1117C440  5tTtiT3l8IdfzaLL6GR5EmytbQ3V3O1Uk.Y4FCKxYOMVoPzPtRG3vo3688uUWpZI
.text:00552A83                                            ; 1117C480  KBV7e6dht/mAhuCEIlRGcdpAEf6f4zUUYf.dtHcDafMVEA4Sy/DDsd76wAyBIM0X
.text:00552A83                                            ; 1117C4C0  KLv1+vH476TN1K1tIR8rR98QFl5mlXkgqz6.h+Wpb/5KYWWvG0ZLZcu6eWOCGmLE
.text:00552A83                                            ; 1117C500  morvWQIDAQAB.-----END RSA PUBLIC KEY-----.....................
.text:00552A88                mov     eax, [esp]
.text:00552A8B                mov     ecx, [esp+4]
.text:00552A8F                mov     [esp+48h+var_48], 0
.text:00552A96                mov     [esp+48h+decryptedLenght], eax
```

Decryption of RSA public key

```
mov     [esp+48h+decryptedLenght], eax
mov     [esp+48h+decryptedString], ecx
call    runtime_stringtoslicebyte
mov     eax, [esp+48h+var_34]
mov     ecx, [esp+48h+var_38]
mov     edx, [esp+48h+var_3C]
mov     [esp], edx
mov     [esp+4], ecx
mov     [esp+8], eax
call    encoding_pem_Decode
mov     eax, [esp+0Ch]
mov     [esp+1Ch], eax
test    eax, eax
jz      loc_552B94
```

```
loc_552B94:
call    printBadpem
mov     [esp+48h+decryptedString], 0
mov     [esp+48h+var_3C], 0
mov     [esp+48h+var_38], 0
call    helpFunction1
jmp     loc_552ACF
```

```
loc_552ACF:
call    decryptEndStrings
mov     eax, [esp+48h+var_48]
mov     ecx, [esp+48h+decryptedLenght]
mov     edx, [esp+48h+var_2C]
mov     ebx, [edx]
mov     ebp, [edx+4]
mov     [esp+48h+var_24], ebx
mov     [esp+48h+var_20], ebp
mov     [esp+48h+var_1C], 0
mov     [esp+48h+var_18], 0
lea     ebx, dword_5F0CC0
mov     [esp+48h+var_1C], ebx
lea     ebx, [esp+48h+var_24]
mov     [esp+48h+var_18], ebx
mov     [esp+48h+var_48], eax
mov     [esp+48h+decryptedLenght], ecx
lea     eax, [esp+48h+var_1C]
mov     [esp+48h+decryptedString], eax
mov     [esp+48h+var_3C], 1
mov     [esp+48h+var_38], 1
call    helpFunction1
mov     eax, [esp+48h+var_2C]
mov     ecx, [eax+0Ch]
mov     edx, [eax+10h]
mov     eax, [eax+14h]
mov     [esp+48h+var_48], ecx
mov     [esp+48h+decryptedLenght], edx
mov     [esp+48h+decryptedString], eax
call    crypto_x509_ParsePKCS1PublicKey
mov     eax, [esp+48h+var_3C]
mov     [esp+48h+publicKey], eax
mov     ecx, [esp+48h+var_38]
```

Parsing of the RSA key

The first parameter of the **EncryptOAEP** function must be the hash function, which in this case is **sha1**:

func EncryptOAEP

```
func EncryptOAEP(hash hash.Hash, random io.Reader, pub *PublicKey, msg []byte, label []byte) ([]byte, error)
```

EncryptOAEP encrypts the given message with RSA-OAEP.

OAEP is parameterised by a hash function that is used as a random oracle. Encryption and decryption of a given message must use the same hash function and sha256.New() is a reasonable choice.

EncryptOAEP Function

```
sha1 = (_DWORD *)runtime_newobject(dword_601BE0);
*sha1 = 0x67452301;
sha1[1] = 0xEFCDAB89;
sha1[2] = 0x98BADCFE;
sha1[3] = 0x10325476;
sha1[4] = 0xC3D2E1F0;
sha1[21] = 0;
sha1[22] = 0;
sha1[23] = 0;
crypto_rsa_EncryptOAEP(&off_6C5E40, sha1, rand1, rand2, RSAKey, AESKey, AESKeySize, sliceKeyAes, 0, 0, 0);
```

Call to EncryptOAEP function

Various extensions, file and folders are excluded for file encryption, also using a regex.

Partially excluded Files:

Iconcache.db	**Ntuser.dat**	**Desktop.ini**
Ntuser.ini	**Usrclass.dat**	**Usrclass.dat.log1**
Usrclass.dat.log2	**Bootmgr**	**Bootnxt**
Ntuser.dat.log1	**Ntuser.dat.log2**	**Boot.ini**
ctfmon.exe	**bootsect.bak**	**ntdlr**

Partially excluded extensions:

Exe	**Dll**	**Sys**
Mui	**Tmp**	**Lnk**
config	**settingcontent-ms**	**Tlb**
Olb	**Bfl**	**ico**
regtrans-ms	**devicemetadata-ms**	**Bat**
Cmd	**Ps1**	

Excluded Paths:

\ProgramData
\Users\All Users
\Temp
\AppData
\Boot
\Local Settings
\Recovery
\Program Files
\System Volume Information
\$Recycle.Bin

> **.+\\Microsoft\\(User Account Pictures|Windows\\(Explorer|Caches)|Device**

And that just about wraps up this post on the SNAKE Ransomware!

Decryption Script

```
 #!/usr/bin/env python3

import re, struct, pefile, sys

pe = None
imageBase = None

def GetRVA(va):
    return pe.get_offset_from_rva(va - imageBase)

def GetVA(raw):
    return imageBase + pe.get_rva_from_offset(raw)

def main():

    global pe, imageBase

    filename = sys.argv[1]

    with open(filename, 'rb') as sample:
        data = bytearray(sample.read())

    pe = pefile.PE(filename)
    imageBase = pe.OPTIONAL_HEADER.ImageBase

    startDecryptFunction = b"\x64\x8B\x0D\x14\x00\x00\x00"
    sliceStr = b"(" + b"\x8D.....\x89.\$\x04\xC7\x44\$\b...." + b")"
    xorLoop = b"\x0F\xB6<\)1\xFE(\x83|\x81)\xFD"

    regex = startDecryptFunction + b".{10,100}" + sliceStr + b".{10,100}" + sliceStr + b".{10,100}" + xorLoop
    pattern = re.compile(regex, re.MULTILINE|re.DOTALL)
    found = pattern.finditer(bytes(data))

    for m in found:

        va = GetVA(m.start())

        funcVA = GetVA(m.start())
        str1VA = struct.unpack("<L", data[m.start(1) + 2 : m.start(1) + 2 + 4])[0]
        str1Len = struct.unpack("<L", data[m.start(1) + 0xE : m.start(1) + 0xE + 4])[0]
```

```
        str2VA = struct.unpack("<L", data[m.start(2) + 2 : m.start(2) + 2 + 4])[0]

        str1RVA = GetRVA(str1VA)
        str2RVA = GetRVA(str2VA)

        decrypted = ""
        for i in range(str1Len):
            decrypted += chr ( ( data[str2RVA+i] ^ (data[str1RVA+i] + i * 2)) & 0xFF)

        print(f"## (hex(funcVA))  - {decrypted}")


if __name__ == "__main__":
         main()
```

Source: https://www.0ffset.net/reverse-engineering/analysing-snake-ransomware/