Threat Intelligence Report

Targeted Snake Ransomware

James Haughom,
VMware Threat Analysis Unit

Table of Contents

Executive Summary

In the last few weeks our telemetry revealed the submission of a Windows executable Ransomware sample, written in Go, which is related to the Snake Ransomware family. This ransomware specifically targeted the Honda network, and was found to be quite sophisticated. The ransomware appears primarily to be targeting servers, as it has logic to check for the type of host it is infecting, and it attempts to stop many server-specific services/processes. Hard-coded strings are encrypted, source code is obfuscated, and the ransomware attempts to stop anti-virus, endpoint security, and server log monitoring and correlation components. This ransomware family has ties to Iran and has historically been observed targeting critical infrastructure such as SCADA and ICS systems. More recently, the malware has been observed targeting healthcare organizations. Most interestingly, and unlike other variants, the malware analyzed in this threat report does not drop any ransom note to desktop machines.

Key Points

- Sophisticated/targeted Go ransomware of the Snake family
- Requires execution in Honda's network
- Writes "EKANS" ("SNAKE" backwards) as well as a custom stub to the tail of encrypted files
- Leverages AES-256 algorithm for encrypting files on attacked hosts
- Unique encryption key is generated for each encrypted file
- Encryption key is encrypted with an RSA-2048 public key
- Hard-coded strings are each encrypted with a simple algorithm
- Contains source code obfuscation
  - Compile-time debugging symbols/names are randomized
  - Strings are encrypted to hinder static analysis tools
- Appears to target servers, although it will also infect desktops
- Kills AV, EDR, and SIEM components
- Leverages COM/WMI execution to avoid detection

Malware Triage

The ransomware is a 32-bit Windows Portable Executable written in Go. Go programs are cross-platform (but are compiled for a target platform such as 64bit Windows) and are completely standalone, meaning they will execute properly even without having a Go interpreter installed on a system, as if written in C/C++. This is achieved through Go statically compiling necessary libraries (packages), which then invoke the standard Windows APIs. Due to this layer of abstraction, analyzing static properties, such as file import tables, is not helpful, as these are used by non-malicious Go library "middleware" code. Other avenues of basic static analysis, such as strings, were also not interesting, as we later found that the malware decrypted these values at runtime.

MD5	ed3c05bde9f0ea0f1321355b03ac42d0
SHA1	e2e14949d0cbc14cd3893da035cc13b509e70a18
SHA256	d4da69e424241c291c173c8b3756639c654432706e7def5025a649730868c4a1
Imphash	96c44fa1eee2c4e9b9e77d7bf42d59e6
SSDEEP	49152:nlpnltflwvk8sd4zs22ahkjzf/3odd8l9akyyxp02+:ntrwkmkff
File Type	Win32 Portable Executable (PE EXE)
File Size	3965952 bytes

Table 1: Static Properties.

Technical Deep Dive

As previously mentioned, the Snake ransomware is written in Go, which makes it challenging to reverse engineer, as well as to detect maliciousness via static file analysis. Interesting functionality in Go binaries begins with an init() function that initializes packages necessary for the binary to run properly (see Figure 1).

```
.text:005CDD12    mov     byte_7DA710, 1
.text:005CDD19    call    sub_4A16C0
.text:005CDD1E    call    sub_4A4530
.text:005CDD23    call    sub_462810
.text:005CDD28    call    sub_472E20
.text:005CDD2D    call    sub_4C94C0
.text:005CDD32    call    sub_4D0AF0
.text:005CDD37    call    sub_4D1190
.text:005CDD3C    call    sub_4D2F70
.text:005CDD41    call    sub_4D1A20
.text:005CDD46    call    sub_4E4AD0
.text:005CDD4B    call    sub_4E5650
.text:005CDD50    call    sub_4E6D80
.text:005CDD55    call    sub_4F9740
.text:005CDD5A    call    sub_4EA290
.text:005CDD5F    call    sub_45ACC0
.text:005CDD64    call    sub_4FA920
.text:005CDD69    call    sub_47EC20
.text:005CDD6E    call    sub_4FC040
.text:005CDD73    call    sub_45A4D0
.text:005CDD78    call    sub_520970
.text:005CDD7D    call    sub_522960
.text:005CDD82    call    sub_517530
.text:005CDD87    call    sub_519500
.text:005CDD8C    call    sub_505A10
.text:005CDD91    call    sub_51DEC0
.text:005CDD96    call    sub_4A6C90
.text:005CDD9B    call    sub_4AE000
.text:005CDDA0    call    sub_508A20
.text:005CDDA5    call    sub_523960
.text:005CDDAA    call    sub_539A30
.text:005CDDAF    call    sub_4D5AD0
.text:005CDDB4    call    sub_4D3080
.text:005CDDB9    call    sub_549D60
.text:005CDDBE    call    sub_566C00
.text:005CDDC3    call    sub_45CBD0
.text:005CDDC8    mov     eax, dword_7DA950
.text:005CDDCE    mov     ecx, [esp+18h+var_10]
.text:005CDDD2    test    eax, eax
.text:005CDDD4    jnz     loc_5CDF20
```

Figure 1. Primary *init()* function.

These package and function names can be resolved with the help of tools, such as IDAGolangHelper. The malware author obfuscated many names at the source code level (see Figure 2).

```
mov     byte_7DA710, 1
call    fmt_init
call    strings_init
call    syscall_init
call    time_init
call    agfkpbpbpmhpmjgifgmf_ocldgdobgbccgabahbki_pdllhaickabpmhmjmcda_apbnkncjkhnoefmmldne_init
call    agfkpbpbpmhpmjgifgmf_ocldgdobgbccgabahbki_pdllhaickabpmhmjmcda_apbnkncjkhnoefmmldne_cnfadk
call    agfkpbpbpmhpmjgifgmf_ocldgdobgbccgabahbki_pdllhaickabpmhmjmcda_apbnkncjkhnoefmmldne_cnfadk
call    crypto_aes_init
call    crypto_cipher_init
call    crypto_rand_init
call    crypto_rsa_init
call    crypto_sha1_init
call    crypto_x509_init
call    encoding_pem_init
call    io_init
call    log_init
call    os_init
call    path_filepath_init
call    sync_init
call    lfaajlodidnplgehhlkp_khcljihlalcdghmhocib_knbhbdocakfimdpjcmcg_aajaajpnelkfdggdkoka_init
call    lfaajlodidnplgehhlkp_lplcknbdahegdnfhcbog_miijbndkghgmdllibelj_init
call    lfaajlodidnplgehhlkp_inkogifkdegjbllhdpph_inkogifkdegjbllhdpph_init
call    lfaajlodidnplgehhlkp_inkogifkdegjbllhdpph_inkogifkdegjbllhdpph_nfmpmppgaeockelhgdcc_init
call    io_ioutil_init
call    lfaajlodidnplgehhlkp_Jjkdodnkjclmncibjjen_pbopnijecnfbnimbiham_init
call    math_rand_init
call    net_init
call    os_exec_init
call    lfaajlodidnplgehhlkp_oeckogbjfbefcnaofdan_eojlodflbpbkkklndmjd_init
call    regexp_init
call    bytes_init
call    encoding_binary_init
call    encoding_gob_init
call    main_glob__func1
call    syscall_NewLazyDLL
mov     eax, dword_7DA950
mov     ecx, [esp+18h+var_10]
test    eax, eax
jnz     loc_5CDF20
```

Figure 2: Primary *init()* function with resolved names.

Following the init() function, the control flow of the application is passed to the main() function. The true functionality of the ransomware can be seen being invoked in the following code block, which contains function names that we manually identified and labeled as shown in Listing 1.

```
mov     [esp+48h+var_48], ecx
mov     [esp+48h+var_44], edx
call    call_runtime_gopanic
call    main_kill_services
call    main_kill_processes
call    main_COM_routine
call    main_decrypt_whitelist_and_blacklist
mov     eax, [esp+48h+var_28]
mov     [esp+48h+var_48], eax
call    main_encryption_routine
call    main_disable_firewall
add     esp, 48h
retn
```

Listing 1: Important code block in main() with renamed functions.

Targeting Honda

Prior to showing any behavior, the malware first performs a check to confirm that it is running in the target network. The sample is clearly targeting Honda, as it will only execute if it is able to properly resolve an internal hostname.

A Windows Management Instrumentation (WMI) query is executed via the Windows Component Object Model (COM), which identifies the DomainRole of the victim machine. The function returns 1 if the type is a Domain Controller, or else it will return 0. It performs this check to see if the DomainRole return value is 4 or 5, as seen in Figure 4 (see Table 1 for the meaning of each DomainRole type).

0	{"Standalone Workstation"}
1	{"Member Workstation"}
2	{"Standalone Server"}
3	{"Member Server"}
4	{"Backup Domain Controller"}
5	{"Primary Domain Controller"}

Table 2: DomainRole types.

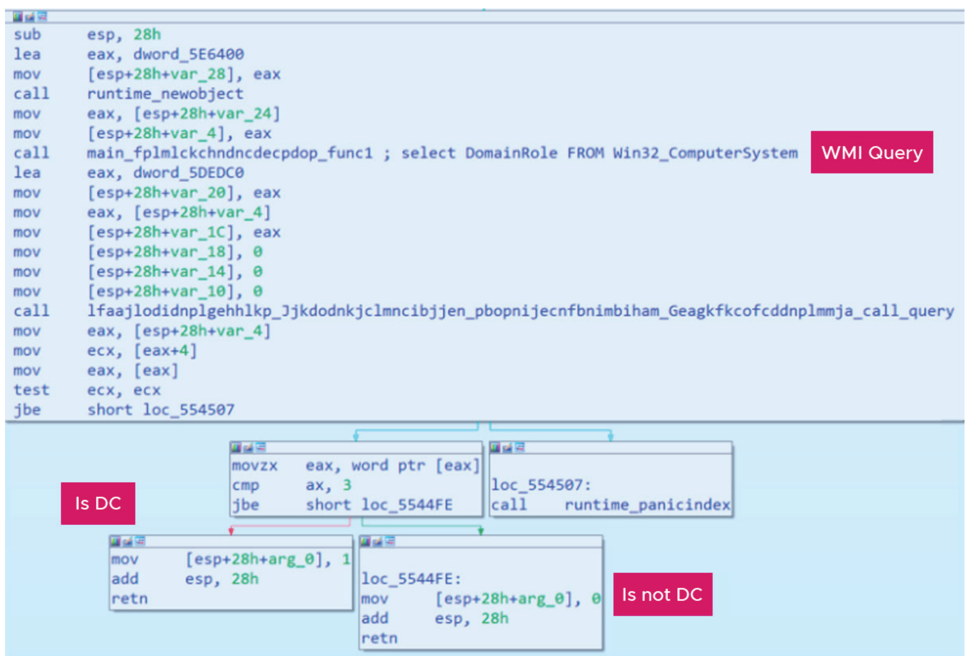

Figure 3: Domain Controller check.

If the victim machine is a Domain Controller, the malware will drop a ransom note and exit.

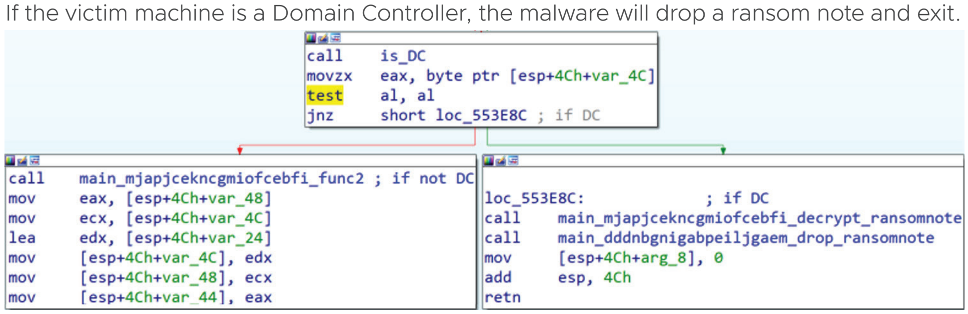

Figure 4: Drop ransom note if a Domain Controller.

Listing 2 shows the Go net.LookupIP procedure used to get the IP address of the internal hostname.

```
.text:00553D69        lea     eax, aMdsHondaCommap ; MDS.HONDA.COM
.text:00553D6F        mov     [esp+4Ch+var_4C], eax
.text:00553D72        mov     [esp+4Ch+var_48], 0Dh
.text:00553D7A        call    net_LookupIP
```

Listing 2: Lookup procedure.

If the hostname is not resolved, the error message displayed in Figure 4 is decrypted, and the malware will exit. If the hostname is resolved, the ransomware performs a secondary check to see if the hostname resolves to the expected IP address (170.108.71[.]153) through the runtime.memequal function displayed in Listing 3.

```
.text:00553DDE              call    net_IP_String
.text:00553DE3              mov     eax, [esp+4Ch+var_3C]
.text:00553DE7              mov     ecx, [esp+4Ch+var_40]
.text:00553DEB              cmp     eax, 0Dh
.text:00553DEE              jz      short loc_553DF7
.text:00553DF0
.text:00553DF0 loc_553DF0: ; CODE XREF: main_domain_check+C4↓j
.text:00553DF0              movzx   eax, [esp+4Ch+var_2D]
.text:00553DF5              jmp     short loc_553DA5
.text:00553DF7 ; -------------------------------------------------------------
-----------------
.text:00553DF7
.text:00553DF7 loc_553DF7: ; CODE XREF: main_domain_check+9E↑j
.text:00553DF7              mov     [esp+4Ch+var_4C], ecx
.text:00553DFA              lea     ecx, a17010871153814;170.108.71.153
.text:00553E00              mov     [esp+4Ch+var_48], ecx
.text:00553E04              mov     [esp+4Ch+var_44], eax
```

Listing 3: IP address check.

The expected IP address is decrypted in memory (Figure 5) for the check mentioned above.

Address	Hex	ASCII
0061F13A	31 37 30 2E 31 30 38 2E 37 31 2E 31 35 33 38 31	170.108.71.15381
0061F14A	34 36 39 37 32 36 35 36 32 35 33 43 6F E0 35 C3	46972656253Coà5Ã
0061F15A	B7 92 DC A5 60 D8 C7 33 9C 9C 0B 28 F0 00 D9 A1	·.Ü¥`ØÇ3...(ð.Ù¡

Figure 5: The IP address to be checked is loaded into memory.

If the domain properly resolves to the IP address 170.108.71[.]153, the malware will continue execution. If not, it will exit as shown in Figure 6 on the following page.

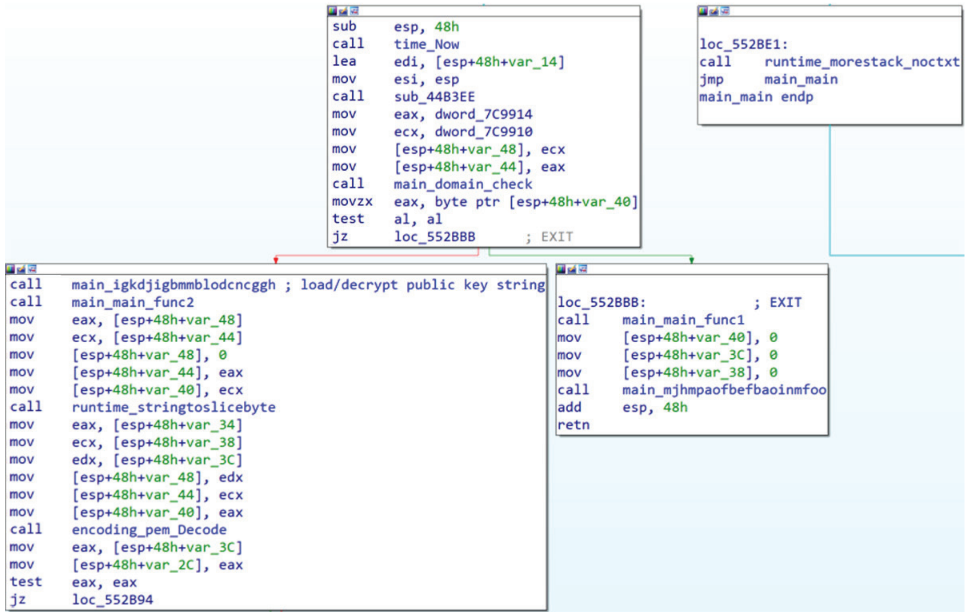

Figure 6: Branch exiting the program.

The true functionality of the ransomware then begins with the RSA-2048 public key (Listing 4) being loaded, decrypted, and decoded. It will later be used to encrypt each AES-256 key used for file encryption.

```
-----BEGIN RSA PUBLIC KEY-----
MIIBCgKCAQEAt1GCKUHXITsiWc1d8V0vo1Y9Jm18RDZEmMS6OkHI7pZT0RHAThlR
\nBFITZY9bXrl6RFdUwmIX0WYn5ZqIlhLAEe1cqd8RpJ/KK2OeiTn0CJ1CGmOOJv
fm\n5rFa8whVAU9cnh/iVCcf+aEHJVcHhzB5tTtiT3lBIdfzaLL6GR5EmytbQ3V3
O1Uk\nY4FCKxYOMVoPzPtRG3vo3688uUWpZIKBV7e6dht mAhuCEIlRGcdpAEf6f
4zUUYf\ndtHcDafMVEA4Sy/DDsd76wAyBIM0XKLv1+vH476TN1K1tIRBrR98QFl5m
lXkgqz6\nh+Wpb/5KYWWvG0ZLZcu6eWOCGmLEmorvWQIDAQAB
-----END RSA PUBLIC KEY-----
```

Listing 4: RSA public key used for encryption.

Stopping Windows Processes/Services

The next routine the malware enters is to terminate target processes and services. This is primarily to ensure that any pre-existing handles to critical files (databases, documents, etc.) are released, so that the malware can successfully encrypt important data (Figure 7 shows some of the decryption routines). If these handles are not released from existing processes/services, then the malware will be unable to obtain handles to encrypt these files. The malware also targets processes/services associated with administration software likely to protect itself and "lock out" admins. Like previous versions of Snake, the malware kills processes associated with ICS/SCADA systems, such as General Electric Proficy Software. This is likely leftover/included from previous versions of Snake, as this particular version is obviously targeting the automotive industry and Microsoft Windows environments. Other processes/services the malware kills are intended to evade detection, as it kills AV/EDR, such as Sophos and Cylance, but also services associated with logging, such as Splunk Windows Eventlog Forwarders.

All interesting strings within the malware sample are encrypted, and each string is assigned a unique XOR key of the same length as the string itself. Each string's decryption routine uses the same algorithm.

```
(xor_key[i] + (2 * i)) ^ encrypted_string[i]
```

Listing 5: Decryption algorithm.

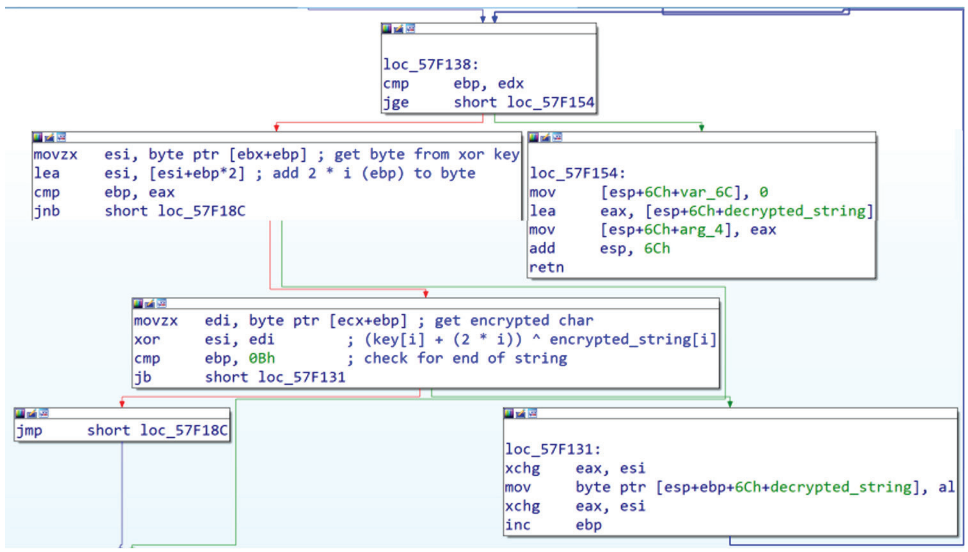

Figure 7: Decryption routine.

The hexdump in Figure 8 shows an example of the buffer containing the output, input, and key of the string decryption routine:

Line 1: Encrypted string(input)

Line 2: 16 null bytes

Line 3: XOR Key (unique to each string)

Line 4: Decrypted String(output)

```
Address   Hex                                               ASCII
12F22EC4  AA CA FA 7A 2A D0 CE E0 F8 EF E4 00 00 00 00 00   ªÊúz*ÐÎàøïä.....
12F22ED4  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
12F22EE4  CB A4 92 04 50 B5 9C 7B 84 78 83 00 00 00 00 00   Ë¤..Pµ.{.x......
Address   Hex                                               ASCII
12F22EB9  61 6C 6C 70 72 6F 66 69 6C 65 73 AA CA FA 7A 2A   allprofilesªÊúz*
```

Figure 8: Example of buffer used by the decryption routine.

Example of first char from above being decrypted via Python:

```
>>> for i in range(len(key)):
...     dec_string += chr((key[i] + i * 2) ^ enc_string[i])
...
>>> dec_string
'allprofiles'
```

Listing 6: Example of decryption of first char.

Many of these string decryption routines are called from this large function. We cannot confirm if it is purposefully obfuscated or simply the result of the combination of Go + compiler optimization.

Next, a list (see Appendix) of process names is decrypted using the string decryption routine mentioned earlier.

Address	Hex	ASCII
130427C0	73 74 77 61 74 63 68 64 6F 67 2E 65 78 65 00 00	stwatchdog.exe..
130427D0	75 73 62 67 75 61 72 64 2E 65 78 65 00 00 00 00	usbguard.exe....
130427E0	75 70 6C 6F 61 64 72 65 63 6F 72 64 2E 65 78 65	uploadrecord.exe
130427F0	73 62 61 6D 73 76 63 2E 65 78 65 00 00 00 00 00	sbamsvc.exe.....
13042800	76 72 76 6D 61 69 6C 2E 65 78 65 00 00 00 00 00	vrvmail.exe.....
13042810	76 72 76 6D 6F 6E 2E 65 78 65 00 00 00 00 00 00	vrvmon.exe......
13042820	76 72 76 6E 65 74 2E 65 78 65 00 00 00 00 00 00	vrvnet.exe......
13042830	77 72 73 61 2E 65 78 65 78 61 67 74 2E 65 78 65	wrsa.exexagt.exe
13042840	6E 65 74 77 6F 72 6B 61 67 65 6E 74 2E 65 78 65	networkagent.exe
13042850	6D 70 63 6D 64 72 75 6E 2E 65 78 65 00 00 00 00	mpcmdrun.exe....
13042860	6D 73 61 73 63 75 69 2E 65 78 65 00 00 00 00 00	msascui.exe.....
13042870	6D 73 6D 70 65 6E 67 2E 65 78 65 00 00 00 00 00	msmpeng.exe.....
13042880	6D 73 70 6D 73 70 73 76 2E 65 78 65 00 00 00 00	mspmspsv.exe....
13042890	6B 62 38 39 31 37 31 31 2E 65 78 65 00 00 00 00	kb891711.exe....
130428A0	7A 61 76 61 75 78 2E 65 78 65 00 00 00 00 00 00	zavaux.exe......
130428B0	7A 61 76 63 6F 72 65 2E 65 78 65 00 00 00 00 00	zavcore.exe.....
130428C0	7A 69 6C 6C 79 61 2E 65 78 65 00 00 00 00 00 00	zillya.exe......
130428D0	7A 6C 63 6C 69 65 6E 74 2E 65 78 65 00 00 00 00	zlclient.exe....
130428E0	76 73 6D 6F 6E 2E 65 78 65 00 00 00 00 00 00 00	vsmon.exe.......
130428F0	66 6F 72 63 65 66 69 65 6C 64 2E 65 78 65 00 00	forcefield.exe..

Figure 10: Process list in memory.

After target process names are decrypted, CreateToolhelp32Snapshot is called to get a list of running processes.

```
0044B770    8B 5C 24 04       mov ebx,dword ptr ss:[esp+4]
0044B774    64 C7 05 34 00    mov dword ptr fs:[34],0
0044B77F    89 E5             mov ebp,esp
0044B781    8B 4B 04          mov ecx,dword ptr ds:[ebx+4]
0044B784    89 C8             mov eax,ecx
0044B786    C1 E0 02          shl eax,2
0044B789    29 C4             sub esp,eax
0044B78B    89 E7             mov edi,esp
0044B78D    8B 73 08          mov esi,dword ptr ds:[ebx+8]
0044B790    FC                cld
0044B791    F3 A5             repe movsd
0044B793    FF 13             call dword ptr ds:[ebx]       [ebx]:CreateToolhelp32Snapshot
```

Figure 11: Call to WinAPI function to get process list.

The malware then begins iterating through this list via Process32(First|Next)W.

```
0044B770    8B 5C 24 04       mov ebx,dword ptr ss:[esp+4]
0044B774    64 C7 05 34 00    mov dword ptr fs:[34],0
0044B77F    89 E5             mov ebp,esp
0044B781    8B 4B 04          mov ecx,dword ptr ds:[ebx+4]
0044B784    89 C8             mov eax,ecx
0044B786    C1 E0 02          shl eax,2
0044B789    29 C4             sub esp,eax
0044B78B    89 E7             mov edi,esp
0044B78D    8B 73 08          mov esi,dword ptr ds:[ebx+8]
0044B790    FC                cld
0044B791    F3 A5             repe movsd
0044B793    FF 13             call dword ptr ds:[ebx]       [ebx]:Process32FirstW
```

Figure 12: Iterating through process list.

```
Address  | Hex                                             | ASCII
13042990 | 53 79 73 74 65 6D 00 00 00 00 00 00 00 00 00 00 | System..........
130429A0 | 88 01 00 00 00 40 04 13 88 01 00 00 00 40 04 13 | .....@.......@..
130429B0 | 73 6D 73 73 2E 65 78 65 00 00 00 00 00 00 00 00 | smss.exe........
130429C0 | 63 73 72 73 73 2E 65 78 65 00 00 00 00 00 00 00 | csrss.exe.......
130429D0 | 88 01 00 00 00 40 04 13 88 01 00 00 00 40 04 13 | .....@.......@..
130429E0 | 77 69 6E 69 6E 69 74 2E 65 78 65 00 00 00 00 00 | wininit.exe.....
130429F0 | 73 65 72 76 69 63 65 73 2E 65 78 65 00 00 00 00 | services.exe....
13042A00 | 88 01 00 00 00 40 04 13 88 01 00 00 00 40 04 13 | .....@.......@..
13042A10 | 6C 73 61 73 73 2E 65 78 65 00 00 00 00 00 00 00 | lsass.exe.......
13042A20 | 88 01 00 00 00 40 04 13 88 01 00 00 00 40 04 13 | .....@.......@..
13042A30 | 73 76 63 68 6F 73 74 2E 65 78 65 00 00 00 00 00 | svchost.exe.....
13042A40 | 73 76 63 68 6F 73 74 2E 65 78 65 00 00 00 00 00 | svchost.exe.....
13042A50 | 88 01 00 00 00 40 04 13 88 01 00 00 00 40 04 13 | .....@.......@..
13042A60 | 73 76 63 68 6F 73 74 2E 65 78 65 00 00 00 00 00 | svchost.exe.....
13042A70 | 73 76 63 68 6F 73 74 2E 65 78 65 00 00 00 00 00 | svchost.exe.....
13042A80 | 88 01 00 00 00 40 04 13 88 01 00 00 00 40 04 13 | .....@.......@..
13042A90 | 73 76 63 68 6F 73 74 2E 65 78 65 00 00 00 00 00 | svchost.exe.....
13042AA0 | 73 76 63 68 6F 73 74 2E 65 78 65 00 00 00 00 00 | svchost.exe.....
13042AB0 | 88 01 00 00 00 40 04 13 88 01 00 00 00 40 04 13 | .....@.......@..
13042AC0 | 73 76 63 68 6F 73 74 2E 65 78 65 00 00 00 00 00 | svchost.exe.....
```

Figure 13. Running processes in memory.

The sample then calls strings.EqualFold to compare each running process name against the list of decrypted process names.

```
0055696D   89 04 24        mov dword ptr ss:[esp],eax      [esp]:"[System Process]"
00556970   89 4C 24 04     mov dword ptr ss:[esp+4],ecx
00556974   89 6C 24 08     mov dword ptr ss:[esp+8],ebp    [esp+8]:"ccflic0.exe"
00556978   89 5C 24 0C     mov dword ptr ss:[esp+C],ebx
0055697C   E8 DF C7 F4 FF  call <snake.strings_EqualFold>
```

Figure 14. Checking running process against list of targeted processes.

If the process name matches, the malware will obtain a handle to said process and kill it.

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

Listing 7. Process termination routine.

If unable to kill the process, the malware decrypts a hard-coded error message.

```
130460C0  63 61 6E 74 20 6B 69 6C 6C 20 70 72 6F 63 65 73  cant kill proces
130460D0  73 20 25 76 20 3A 20 25 76 0A 00 00 00 00 00 00  s %v : %v......
```

Listing 8. Error message for killing processes.

The malware then repeats this process, but this time for services.

```
.text:0054CB2B          mov     [esp+0E8h+var_98], ecx
.text:0054CB2F          mov     edx, [esp+0E8h+arg_0]
.text:0054CB36          mov     ebx, [edx]
.text:0054CB38          mov     [esp+0E8h+var_E8], ebx
.text:0054CB3B          mov     [esp+0E8h+var_E4], 0
.text:0054CB43          mov     [esp+0E8h+var_E0], SERVICE_WIN32
.text:0054CB4B          mov     [esp+0E8h+var_DC], SERVICE_STATE_ALL
.text:0054CB53          mov     [esp+0E8h+var_D8], eax
.text:0054CB57          mov     [esp+0E8h+var_D4], ecx
.text:0054CB5B          lea     eax, [esp+0E8h+var_9C]
.text:0054CB5F          mov     [esp+0E8h+var_D0], eax
.text:0054CB63          lea     ebx, [esp+0E8h+var_B8]
.text:0054CB67          mov     [esp+0E8h+var_CC], ebx
.text:0054CB6B          mov     [esp+0E8h+var_C8], 0
.text:0054CB73          mov     [esp+0E8h+var_C4], 0
.text:0054CB7B          call    agfkpb_EnumServicesStatusEx
```

Listing 9. Obtaining list of services.

Services are terminated via OpenService + Service Control (calls ControlService).

```
  OpenService(v40, a1, a2, v27, v28, v29); // get handle to service
  if ( v28 )
  {
    v42 = v29;
    v41 = v28;
    main_ifdjiignopgdooedfgie_func1(v11, v25);
    v4 = v41;
    if ( v41 )
      v4 = *(_DWORD *)(v41 + 4);
    v51 = v4;
    v52 = v42;
    fmt_Errorf(v12, v25, &v51, 1, 1, v29, v30);
    return runtime_deferreturn(v13);
  }
  v39 = v27;
  v26 = v27;
  if ( runtime_deferproc(12, &off_6275CC) )
    return runtime_deferreturn(v10);
ptr_Service_Control(v39, a3, v26, v27, v28, v29, v30, v31, v32); /* kill
service */
```

Listing 10. Service termination routine.

WMI/COM Capabilities and Interactions

The string decryption routine is applied to a string that decrypts to a reference to the WMI scripting library.

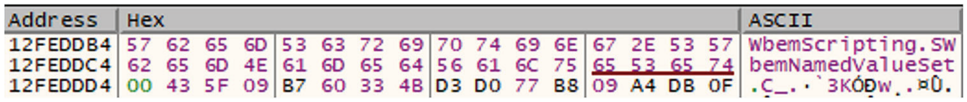

Figure 15. Reference to WbemScripting library in memory.

This string is decrypted shortly after COM library initialization (screenshot below), and an instance of this object is then created via CoCreateInstance.

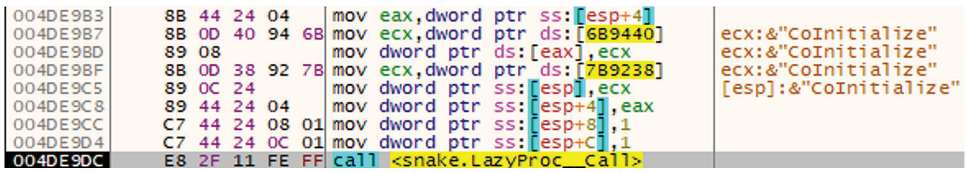

Figure 16. Initialization of COM library.

An instance of WbemScripting.SWbemLocator is also created

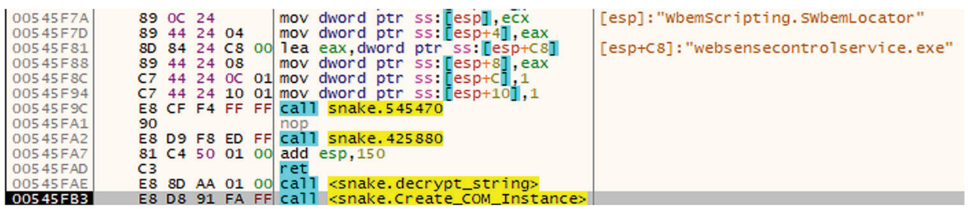

Figure 17. Instance of WBemLocator.

The malware then decrypts a handful of strings, the two most interesting being root\\cimv2 and ConnectServer.

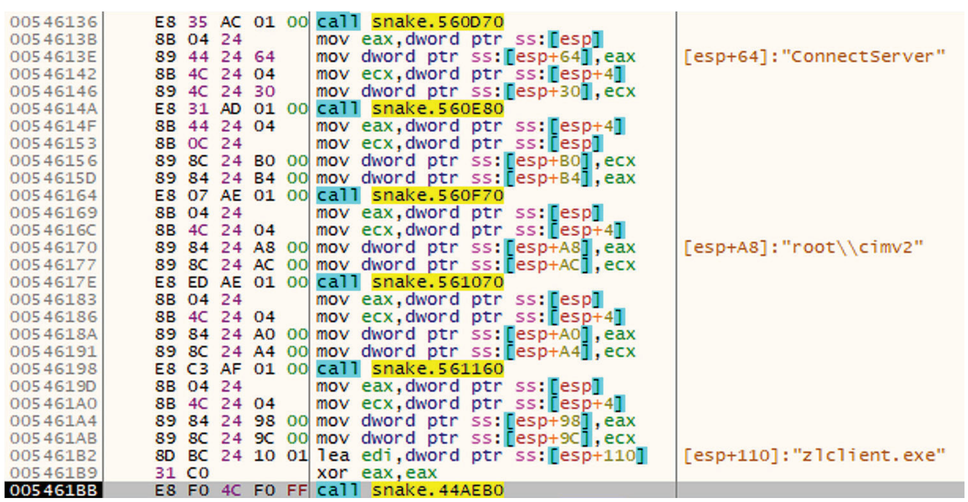

Figure 18. WMI-related strings decrypted.

Two more strings are decrypted, regarding the execution of a WMI query (WQL).

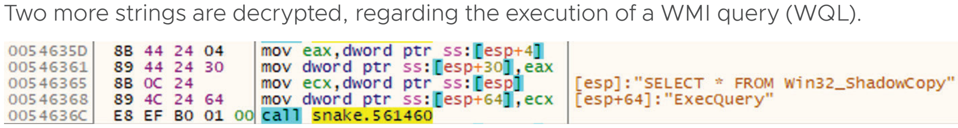

Figure 19. WMI query decrypted.

COM/WMI Capabilities

Classes:

- WbemScripting
- WbemLocator

Methods:

- ConnectServer
- ExecQuery
- Add

Blacklist/Whitelist Decryption

The malware then enters a routine to decrypt a few important lists of files and directories to both avoid and target. The first list of strings decrypted consists of file extensions that the malware targets.

```
13043210 E4 2D A0 02 2E 64 6F 63 78 00 2E 61 63 63 64 62 ä- ..docx..accdb
13043220 53 79 73 53 74 72 69 6E 67 4C 65 6E 00 00 00 00 SysStringLen....
13043230 2E 61 63 63 64 65 2E 61 63 63 64 72 00 00 00 00 .accde.accdr....
13043240 2E 61 63 63 64 74 00 00 2E 61 73 70 00 00 00 00 .accdt...asp....
13043250 2E 61 73 70 78 2E 62 61 63 6B 00 00 00 00 00 00 .aspx.back......
13043260 2E 62 61 63 6B 75 70 2E 62 61 63 6B 75 70 64 62 .backup.backupdb
13043270 2E 62 61 6B 2E 6D 64 62 2E 6D 64 63 2E 6D 64 66 .bak.mdb.mdc.mdf
13043280 2E 77 61 72 2E 78 6C 73 2E 78 6C 73 78 00 00 00 .war.xls.xlsx...
13043290 2E 78 6C 73 6D 00 00 00 2E 78 6C 72 2E 7A 69 70 .xlsm....xlr.zip
130432A0 2E 72 61 72 2E 73 71 6C 69 74 65 64 62 00 00 00 .rar.sqlitedb...
130432B0 2E 73 71 6C 2E 70 79 2E 70 70 61 6D 2E 70 70 73 .sql.py.ppam.pps
130432C0 2E 70 70 73 6D 2E 70 70 73 78 00 00 2E 70 70 74 .ppsm.ppsx...ppt
130432D0 70 70 74 6D 2E 70 70 74 78 00 00 00 2E 68 70 70 pptm.pptx....hpp
130432E0 2E 6A 61 76 61 00 00 00 2E 6A 73 70 2E 70 68 70 .java....jsp.php
130432F0 2E 64 6F 63 2E 64 6F 63 6D 00 00 00 2E 70 73 74 .doc.docm....pst
13043300 2E 70 73 64 2E 64 6F 74 64 6F 74 6D 2E 63 70 70 .psd.dotdotm.cpp
13043310 2E 63 73 00 2E 63 73 76 2E 62 6B 70 2E 64 62 00 .cs..csv.bkp.db.
13043320 2E 64 62 2D 6A 6F 75 72 6E 61 6C 00 00 00 00 00 .db-journal.....
13043330 2E 63 73 70 72 6F 6A 00 2E 73 6C 6E 2E 6D 64 00 .csproj..sln.md.
13043340 2E 70 6C 2E 6A 73 2E 68 74 6D 6C 00 2E 68 74 6D .pl.js.html..htm
13043350 2E 64 62 66 2E 72 64 6F 2E 61 72 63 2E 76 68 64 .dbf.rdo.arc.vhd
13043360 2E 76 6D 64 6B 00 00 00 2E 76 64 69 00 00 00 00 .vmdk....vdi....
13043370 2E 76 68 64 78 00 00 00 2E 65 64 62 2E 63 2E 68 .vhdx....edb.c.h
```

Figure 20. Target file extensions.

Another batch of filenames and extensions are then decrypted; these are whitelisted names for select System/Program directories.

```
13043380 2E 64 6C 6C 2E 65 78 65 2E 73 79 73 2E 6D 75 69 .dll.exe.sys.mui
13043390 2E 74 6D 70 2E 6C 6E 6B 2E 63 6F 6E 66 69 67 00 .tmp.lnk.config.
130433A0 2E 6D 61 6E 69 66 65 73 74 00 00 00 2E 74 6C 62 .manifest....tlb
130433B0 2E 6F 6C 62 2E 62 6C 66 2E 69 63 6F 2E 62 61 74 .olb.blf.ico.bat
130433C0 2E 72 65 67 74 72 61 6E 73 2D 6D 73 00 00 00 00 .regtrans-ms....
130433D0 2E 63 6D 64 2E 70 73 31 62 6F 6F 74 6D 67 72 00 .cmd.ps1bootmgr.
130433E0 64 65 73 6B 74 6F 70 2E 69 6E 69 00 00 00 00 00 desktop.ini.....
130433F0 69 63 6F 6E 63 61 63 68 65 2E 64 62 00 00 00 00 iconcache.db....
13043400 6E 74 75 73 65 72 2E 64 61 74 00 00 00 00 00 00 ntuser.dat......
13043410 6E 74 75 73 65 72 2E 69 6E 69 00 00 00 00 00 00 ntuser.ini......
13043420 6E 74 75 73 65 72 2E 64 61 74 2E 6C 6F 67 31 00 ntuser.dat.log1.
13043430 6E 74 75 73 65 72 2E 64 61 74 2E 6C 6F 67 32 00 ntuser.dat.log2.
13043440 75 73 72 63 6C 61 73 73 2E 64 61 74 00 00 00 00 usrclass.dat....
```

Figure 21. File extensions select whitelist.

More strings are decrypted, this time for whitelisted directories (selective) and filenames for the encryption routine.

Address	Hex	ASCII
130434A0	3A 5C 24 52 65 63 79 63 6C 65 2E 42 69 6E 00 00	:\$Recycle.Bin..
130434B0	3A 5C 50 72 6F 67 72 61 6D 44 61 74 61 00 00 00	:\ProgramData...
130434C0	3A 5C 50 72 6F 67 72 61 6D 20 46 69 6C 65 73 00	:\Program Files.
130434D0	3A 5C 4C 6F 63 61 6C 20 53 65 74 74 69 6E 67 73	:\Local Settings
130434E0	3A 5C 42 6F 6F 74 3A 5C 52 65 63 6F 76 65 72 79	:\Boot:\Recovery
130434F0	5C 41 70 70 44 61 74 61 5C 00 00 00 00 00 00 00	\AppData\.......

Figure 22. Select folders.

Address	Hex	ASCII
130434F9	6E 74 6C 64 72 00 00 4E 54 44 45 54 45 43 54 2E	ntldr..NTDETECT.
13043509	43 4F 4D 00 00 00 00 62 6F 6F 74 2E 69 6E 69 00	COM....boot.ini.
13043519	00 00 00 00 00 00 00 62 6F 6F 74 66 6F 6E 74 2E	.......bootfont.
13043529	62 69 6E 00 00 00 00 62 6F 6F 74 73 65 63 74 2E	bin....bootsect.
13043539	62 61 6B 00 00 00 00 64 65 73 6B 74 6F 70 2E 69	bak....desktop.i
13043549	6E 69 00 00 00 00 00 63 74 66 6D 6F 6E 2E 65 78	ni.....ctfmon.ex
13043559	65 00 00 00 00 00 00 69 63 6F 6E 63 61 63 68 65	e......iconcache
13043569	2E 64 62 00 00 00 00 6E 74 75 73 65 72 2E 64 61	.db....ntuser.da
13043579	74 00 00 00 00 00 00 6E 74 75 73 65 72 2E 64 61	t......ntuser.da
13043589	74 2E 6C 6F 67 00 00 6E 74 75 73 65 72 2E 69 6E	t.log..ntuser.in
13043599	69 00 00 00 00 00 00 74 68 75 6D 62 73 2E 64 62	i......thumbs.db

Figure 23. Select files.

The malware then calls GetLogicDriveStringsW, which will be modified and passed to GetDriveTypeW, whose results are checked to determine if the drive is fixed or removable.

```
005170E8 | 8B 0D 04 91 7B  | mov ecx,dword ptr ds:[7B9104]     |
005170EE | 89 0C 24        | mov dword ptr ss:[esp],ecx        | [esp]:&"GetDriveTypeW"
005170F1 | 89 44 24 04     | mov dword ptr ss:[esp+4],eax      | [esp+4]:&L"C:"
005170F5 | C7 44 24 08 01  | mov dword ptr ss:[esp+8],1        |
005170FD | C7 44 24 0C 01  | mov dword ptr ss:[esp+C],1        |
00517105 | E8 06 8A FA FF  | call <snake.LazyProc__Call>       |
```

Figure 24. Call to GetDriveType.

```
.text:005170EE                 mov     [esp+138h+var_138], ecx
.text:005170F1                 mov     [esp+138h+var_134], eax
.text:005170F5                 mov     [esp+138h+var_130], 1
.text:005170FD                 mov     [esp+138h+var_12C], 1
.text:00517105                 call    __ptr_LazyProc_Call
.text:0051710A                 mov     eax, [esp+138h+var_128]
.text:0051710E                 test    eax, eax
.text:00517110                 jz      loc_517659
.text:00517116                 mov     [esp+138h+var_100], eax
.text:0051711A                 cmp     eax, 2 ; DRIVE_REMOVABLE
.text:0051711D                 jz      short loc_51714B
.text:0051711F                 cmp     eax, 3 ; DRIVE_FIXED
.text:00517122                 jz      short loc_51714B
```

Listing 11. Checking drive type.

Encryption Routine

The file encryption routine follows common ransomware techniques: crawl each directory, obtain handles to each targeted file type, and perform the encryption.

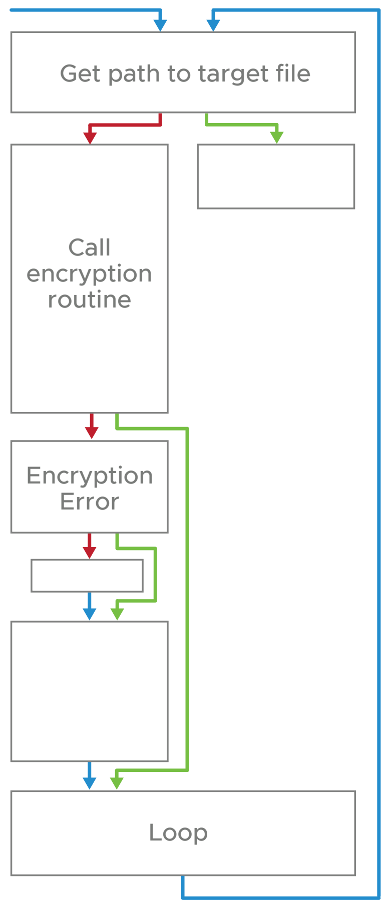

Figure 25. Control-flow graph of the file encryption routine.

The Go file package is used heavily throughout the encryption routine. Iterating through the file system, the malware repeatedly calls file.ReadDir, file.Open, file.Seek, and file.Read. An AES-256 key is generated via the Go rand.Read function for each individual file. The 32-byte key is then passed to the file encryption routine.

```
crypto_rand_Read(key_buffer, key_len);
call_runtime_gopanic(key_len, v20);
main_encrypt_file(v35, key_buffer, key_len, v20, v34, v24, v27, v27);
```

Listing 12. Encryption routine.

Example of output of rand.Read (AES-256 key).

Address	Hex																ASCII
1329EEC0	C4	9E	8B	DF	E1	22	E6	E9	57	9A	E4	4C	52	33	BB	60	Ä..ßá"æéW.äLR3»`
1329EED0	7B	F6	3A	FA	5E	7F	A3	2B	AC	5C	9D	C4	AF	47	36	5F	{ö:ú^.£+¬\.Ä¯G6_
1329EEE0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
1329EEF0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....

Figure 26. Random 32 byte sequence to be used as key.

Crypto Routine for encrypting contents of each target file:

1. Generate key – rand.Read (32 bytes)
2. Create Cipher Block – aes.NewCipher AES-256
3. Create crypto stream – cipher.NewCTR
4. Read target file into buffer – file.Read
5. Encrypt contents of file buffer – XORKeyStream
6. Overwrite file on disk with ciphertext – file.WriteAt
7. Loop/Find next file:
    a. Control flow is transferred via deferred functions
    b. WaitGroup is used to ensure that important crypto tasks complete prior to moving to the next task
    c. runtime.chanrecv2 is used to help the loop to get next file path

The figure below shows how the malware author sets a deferred function to close the file it is encrypting once the crypto routine finishes.

```
.text:0055218C         mov     [esp+80h+arg_C], 0
.text:00552197         mov     [esp+80h+arg_10], 0
.text:005521A2         mov     eax, [esp+80h+arg_0]
.text:005521A9         mov     [esp+80h+var_80], eax
.text:005521AC         mov     ecx, [esp+80h+arg_4]
.text:005521B3         mov     [esp+80h+var_7C], ecx
.text:005521B7         mov     [esp+80h+var_78], 2
.text:005521BF         mov     [esp+80h+var_74], 1EDh
.text:005521C7         call    os_OpenFile
.text:005521CC         mov     eax, [esp+80h+var_70]
.text:005521D0         mov     ecx, [esp+80h+var_6C]
.text:005521D4         mov     edx, [esp+80h+var_68]
.text:005521D8         mov     [esp+80h+arg_C], ecx
.text:005521DF         mov     [esp+80h+arg_10], edx
.text:005521E6         test    ecx, ecx
.text:005521E8         jnz     loc_552509
.text:005521EE         mov     [esp+80h+var_2C], eax
.text:005521F2         mov     [esp+80h+var_78], eax
.text:005521F6         mov     [esp+80h+var_80], 0Ch
.text:005521FD         lea     ecx, off_6320F4 ; os__ptr_File_Close
.text:00552203         mov     [esp+80h+var_7C], ecx
.text:00552207         call    runtime_deferproc ; set deferred func
```

Listing 13. Setting deferred function.

This is the end of the crypto routine, indicating that the deferred function will run and the file handle will be closed.

```
.text:00552284 loc_552284:                              ; CODE XREF:
main_main_crypto+14D↓j
.text:00552284                  lea     ecx, off_6C5410
.text:0055228A                  mov     [esp+80h+arg_C], ecx
.text:00552291                  mov     [esp+80h+arg_10], eax
.text:00552298                  nop
.text:00552299                  call    runtime_deferreturn ; exec
deferred func
.text:0055229E                  add     esp, 80h
.text:005522A4                  retn
```

Listing 14. Invocation of deferred function.

Before any of this encryption routine is executed, the malware first checks to see if it has already encrypted the file. It does this by checking for the known Snake Ransomware “EKANS” string at the end of the file.

```
.text:0055221B        call    check_EKANS_string
.text:00552220        movzx   eax, byte ptr [esp+80h+var_7C]
.text:00552225        mov     ecx, [esp+80h+var_6C]
.text:00552229        mov     edx, [esp+80h+var_70]
.text:0055222D        mov     [esp+80h+arg_C], edx
.text:00552234        mov     [esp+80h+arg_10], ecx
.text:0055223B        test    edx, edx
.text:0055223D        jnz     loc_5524EF
.text:00552243        test    al, al
.text:00552245        jz      short goto_encrypt_file
```

Listing 15. Encrypt file only if EKANS tail is not found.

This is a hexdump of the tail of the target file for encryption.

```
00035010  44 3A 5C 5F 63 6F 64 65 5C 69 44 65 66 5C 53 79  D:\_code\iDef\Sy
00035020  73 41 6E 61 6C 79 7A 65 72 5C 70 72 6F 63 5F 77  sAnalyzer\proc_w
00035030  61 74 63 68 2E 70 64 62 00                       atch.pdb.
```

Figure 27. Tail of target file.

Checking whether the EKANS tag is in the tail of file (last 5 bytes) is performed through the runtime.memequal function.

```
00556D81  8B 44 24 30     mov eax,dword ptr ss:[esp+30]    [esp+30]:".pdb"
00556D85  89 04 24        mov dword ptr ss:[esp],eax       [esp]:".pdb"
00556D88  8B 05 70 98 7B  mov eax,dword ptr ds:[7B9870]    eax:"EKANS", 007B9870:&"EKANS"
00556D8E  89 44 24 04     mov dword ptr ss:[esp+4],eax     [esp+4]:"EKANS"
00556D92  89 4C 24 08     mov dword ptr ss:[esp+8],ecx
00556D96  E8 65 3D EF FF  call <snake.runtime_mem_equal>
```

Figure 28. Checking for the EKANS tail.

If the EKANS string is found, a message is decrypted “already encrypted”. This file will then be skipped, and the next file in the directory is passed to the encryption routine.

```
005471C7    85 D2           test edx,edx
005471C9  v 0F 85 54 03 00  jne snake.547523
005471CF    84 C0           test al,al
005471D1  v 74 78           je snake.54724B
005471D3    E8 D8 AB 01 00  call snake.561DB0
005471D8    8B 04 24        mov eax,dword ptr ss:[esp]   [esp]:"already encrypted"
```

Figure 29. Message if file is already encrypted.

If not yet encrypted, the target file will be passed to the file encryption routine. The randomly generated 32-byte string will be passed to aes.NewCipher as a key to create a new cipher block. This cipher block is passed to crypto.cipher.NewCTR to create the stream used for encryption, and the file.Read function is then called to get the contents of the target file. (See Listing 16 on the following page.)

```
.text:00551ED3                  call    crypto_aes_NewCipher
.text:00551ED8                  mov     eax, [esp+70h+var_58]
.text:00551EDC                  mov     [esp+70h+var_4], eax
---SNIP---
.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_5F0E80
---SNIP---
.text:00551F9F                  mov     esi, [esp+70h+arg_0]
.text:00551FA3                  mov     [esp+70h+var_70], esi
.text:00551FA6                  mov     [esp+70h+var_6C], eax
.text:00551FAA                  mov     [esp+70h+var_68], edx
.text:00551FAE                  mov     [esp+70h+var_64], ecx
.text:00551FB2                  call    os__ptr_File_Read ; read target file
```

Listing 16: Crypto housekeeping and reading of target file.

```
Address  | Hex                                             | ASCII
132F4000 | 4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00 | MZ..........ÿÿ..
132F4010 | B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 | ¸.......@.......
132F4020 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
132F4030 | 00 00 00 00 00 00 00 00 00 00 00 00 E0 00 00 00 | ............à...
132F4040 | 0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68 | ..º..´.Í!¸.LÍ!Th
132F4050 | 69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F | is program canno
132F4060 | 74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20 | t be run in DOS 
132F4070 | 6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00 | mode....$.......
132F4080 | C2 75 49 99 86 14 27 CA 86 14 27 CA 86 14 27 CA | ÂuI...'Ê..'Ê..'Ê
132F4090 | FD 08 2B CA 87 14 27 CA 6E 0B 2C CA 87 14 27 CA | ý.+Ê..'Ên.,Ê..'Ê
132F40A0 | 05 08 29 CA 8D 14 27 CA 6E 0B 2D CA BF 14 27 CA | ..)Ê..'Ên.-Ê¿.'Ê
132F40B0 | E4 0B 34 CA 85 14 27 CA 86 14 26 CA B1 14 27 CA | ä.4Ê..'Ê..&Ê±.'Ê
132F40C0 | 6E 0B 31 CA 87 14 27 CA 52 69 63 68 86 14 27 CA | n.1Ê..'ÊRich..'Ê
132F40D0 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
```

Figure 30: Output buffer for read.File for an executable.

The buffer is passed to the XORKeyStream function to be encrypted. The contents of the original file are then overwritten with this ciphertext via file.WriteAt.

```
mov     [esp+70h+var_60], edi
mov     [esp+70h+var_5C], ebx
mov     [esp+70h+var_58], ebp
mov     ebp, [esp+70h+var_18]
mov     [esp+70h+var_70], ebp
call    esi                ; XORKeyStream - encrypt buffer
mov     eax, [esp+70h+arg_0]
mov     [esp+70h+var_70], eax
mov     ecx, [esp+70h+var_28]
mov     [esp+70h+var_6C], ecx
mov     ecx, [esp+70h+var_4C]
mov     [esp+70h+var_68], ecx
mov     ecx, [esp+70h+var_48]
mov     [esp+70h+var_64], ecx
mov     ecx, [esp+70h+var_40]
mov     [esp+70h+var_60], ecx
mov     edx, [esp+70h+var_3C]
mov     [esp+70h+var_5C], edx
call    os__ptr_File_WriteAt ; overwrite target file with ciphertext
```

Listing 17: Encrypting buffer and overwriting target file.

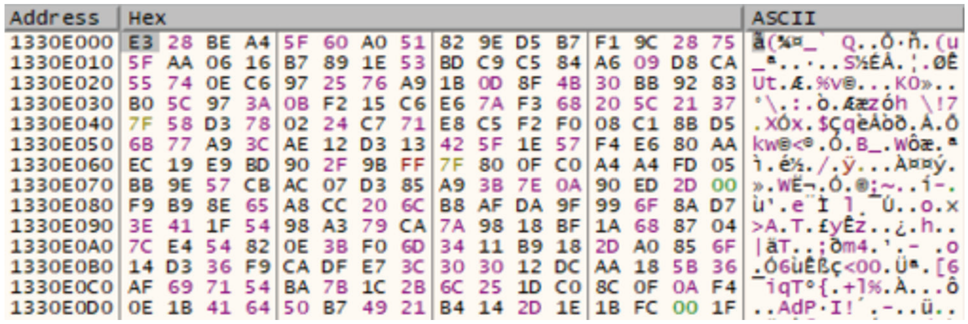

Figure 31: Encrypted buffer after XORKeyStream function call.

Once the file is encrypted, a custom footer/stub is written to the tail of the file.

Stub contents:

- Header
- Name from source code
- RSA-2048 encrypted AES-256 key
- Full path to encrypted file
- 4 byte string
- EKANS string

Original tail:

`00011FF0     60 6B 86 34 9F 8E1F E9 F4 AA F0 35 E5 65 3A   ` `k†d4ŸŽ.éôªô5åe:`

First write operation:

- Header
- Path to file
- Encrypted AES key

Second write operation:

- 4 byte string

Third write operation:

- EKANS string

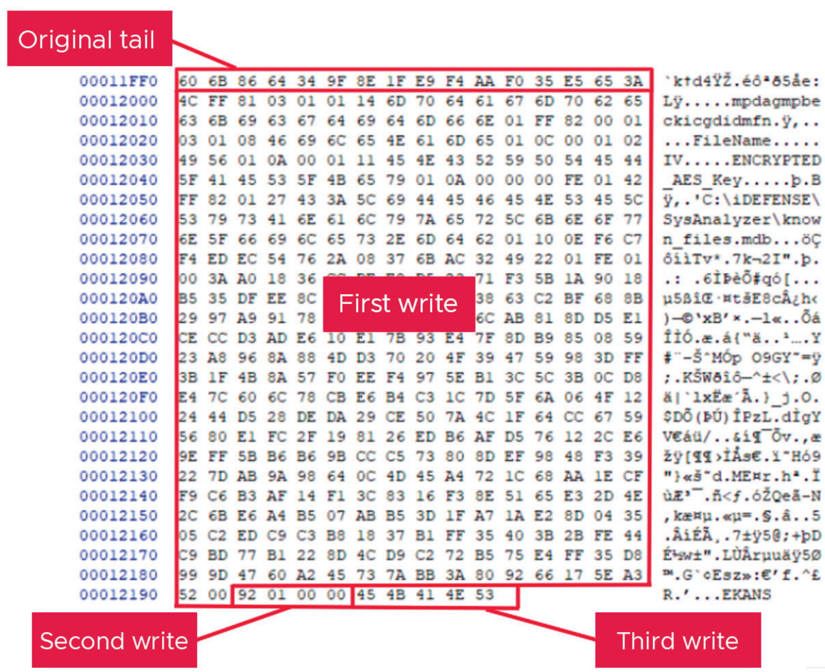

Figure 32. Encrypted buffer after XORKeyStream function call.

Once this function completes, the malware then repeats this process on the next file in the directory.

A random 5 character string is appended to the file extension of encrypted files.

```
    Directory: C:\Users\rem\Desktop


Mode                LastWriteTime         Length Name
----                -------------         ------ ----
d-----        5/13/2020   4:18 PM                Extra Tools
d-----        9/23/2019   8:23 PM                fakenet_logs
d-----        5/13/2020   4:18 PM                flare-vm-master
d-----        5/13/2020   4:18 PM                mal
da----         5/4/2018   1:02 PM                Malware
d-----        5/13/2020   4:22 PM                PS_Transcripts
d-----        5/13/2020   4:20 PM                xia
-a----        5/13/2020   4:14 PM         563518 5120577605926912.zipVfowM
-a----        5/13/2020   4:14 PM         130298 5972913821777920.zipULavF
-a----        5/13/2020   4:14 PM           2091 Administrator Command Prompt.lnkZMbiE
-a----        5/13/2020   4:14 PM           2146 API Monitor x32.lnkOwnFO
-a----        5/13/2020   4:14 PM           2146 API Monitor x64.lnkNAIYt
-a----        5/13/2020   4:14 PM           1821 BinText.lnknlCes
-a----        5/13/2020   4:14 PM           1825 Bytehist.lnkUdYAc
-a----        5/13/2020   4:14 PM           1368 CFF Explorer.lnkpWHdX
-a----        5/13/2020   4:14 PM           1502 Detect It Easy.lnkLlnrs
-a----        5/13/2020   4:14 PM           1934 Exeinfo PE.lnkgCKOR
-a----        5/13/2020   4:14 PM           2385 Fiddler.lnkRdxuN
-a----        5/13/2020   4:14 PM           1734 FLARE.lnkngVVo
-a----        5/13/2020   4:14 PM        1659285 flounder.execlkgA
-a----        5/13/2020   4:14 PM        1659293 flounder_no_aslr.exeYQTeU
-a----        5/13/2020   4:14 PM           1301 IDA Freeware.lnkeGkWV
-a----        5/13/2020   4:14 PM           1267 ida.keyCLiAh
-a----        5/13/2020   4:14 PM           1524 ida_0A1C2D38F332_48-B511-7074-EC.licdyvTv
-a----        5/13/2020   4:14 PM           1737 Internet Explorer.lnkaVGux
```

Figure 33: Directory listing showing the extensions appended to encrypted files.

Executables (among other file types) are whitelisted/not to be encrypted in System/Program directories.

```
PS C:\Program Files (x86)\xorstrings> dir


    Directory: C:\Program Files (x86)\xorstrings


Mode                LastWriteTime         Length Name
----                -------------         ------ ----
d-----        5/13/2020  12:47 PM                OSX
-a----        5/13/2020  12:39 PM          16280 XORStrings.cmuPjs
-a----         3/8/2013   4:37 PM          73176 xorstrings.exe
```

Figure 34: Directory listing showing how executables are whitelisted in System/Program Directories.

Ransom Note

Very interestingly, this sample does not display the ransom note on desktop machines at this point in the program, which we have observed with previous samples. Instead, it spawns the native windows utility netsh to disable the local firewall (see Figure 34) and then exits.

```
0055440A | 8B 44 24 34    | mov eax,dword ptr ss:[esp+34]    | [esp+34]:"allprofilesstateoff.\\netsh"
0055440E | 89 44 24 54    | mov dword ptr ss:[esp+54],eax    | [esp+54]:"allprofilesstateoff.\\netsh"
00554412 | 8B 44 24 20    | mov eax,dword ptr ss:[esp+20]    |
00554416 | 89 44 24 58    | mov dword ptr ss:[esp+58],eax    |
0055441A | 8B 44 24 30    | mov eax,dword ptr ss:[esp+30]    | [esp+30]:&"oint"
0055441E | 89 44 24 5C    | mov dword ptr ss:[esp+5C],eax    | [esp+5C]:&"oint"
00554422 | 8B 44 24 1C    | mov eax,dword ptr ss:[esp+1C]    |
00554426 | 89 44 24 60    | mov dword ptr ss:[esp+60],eax    |
0055442A | 89 54 24 64    | mov dword ptr ss:[esp+64],edx    | [esp+64]:"off.\\netsh"
0055442E | 8B 44 24 18    | mov eax,dword ptr ss:[esp+18]    |
00554432 | 89 44 24 68    | mov dword ptr ss:[esp+68],eax    |
00554436 | 8B 44 24 40    | mov eax,dword ptr ss:[esp+40]    | [esp+40]:"netsh"
0055443A | 89 04 24       | mov dword ptr ss:[esp],eax       | [esp]:"netsh"
0055443D | 8B 44 24 2C    | mov eax,dword ptr ss:[esp+2C]    |
00554441 | 89 44 24 04    | mov dword ptr ss:[esp+4],eax     |
00554445 | 8D 44 24 44    | lea eax,dword ptr ss:[esp+44]    | [esp+44]:"advfirewallset"
00554449 | 89 44 24 08    | mov dword ptr ss:[esp+8],eax     | [esp+8]:&"advfirewallset"
0055444D | C7 44 24 0C 05 | mov dword ptr ss:[esp+C],5       |
00554455 | C7 44 24 10 05 | mov dword ptr ss:[esp+10],5      |
0055445D | E8 DE 16 FB FF | call <honda_snake.os_exec_Command> |
00554462 | 8B 44 24 14    | mov eax,dword ptr ss:[esp+14]    | [esp+14]:&"C:\\WINDOWS\\system32\\netsh.exe"
00554466 | 89 04 24       | mov dword ptr ss:[esp],eax       | [esp]:"netsh"
00554469 | E8 B2 25 FB FF | call <honda_snake.os_exec_Cmd_Run> |
```

Figure 35: Disabling firewall.

As mentioned earlier, if the victim machine is a Domain Controller, the malware will drop the ransom note, and exit without encrypting files.

```
Decrypt-Your-Files.txt

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

| 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.

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

Figure 36: Ransom note to display if run on a Domain Controller.

We properly detect this sample, along with other variants of SNAKE, through anomalies that are present as a result of source-code obfuscation (notable mention is a sample targeting Enel Global).

ANALYSIS OVERVIEW

SEVERITY	TYPE	DESCRIPTION	ATT&CK TACTIC(S)	ATT&CK TECHNIQUE(S)
100	Signature	Identified ransomware code		
70	Anomaly	Obfuscated application written in Golang		
5	Network	Failing to communicate with server (DNS failure)	Command and Control	Standard Application Layer Protocol
5	Memory	Presence of cryptographic constants (AES)		
5	Evasion	Detecting the presence of WINE	Defense Evasion, Discovery	Virtualization/Sandbox Evasion

Figure 37: VMware Advanced Threat Analzyer.

Conclusions

This was clearly a targeted attack, as the malware was tailored to execute in the Honda network (and largely aimed at servers). The sample is self-defending, as it leverages source code obfuscation, encrypted strings, and kills AV, EDR, and SIEM components. Strong encryption is used (RSA with AES-256), and the encryption routine will cause many applications to cease functioning properly. Each encrypted file has a unique randomly generated encryption key, which itself is encrypted, and then written to a stub at the end of each file, along with the "EKANS" string.

Appendix

Indicators of Compromise (IoCs)

DNS Query	MDS.HONDA.COM
Email from Ransom Note	CarrolBidell@tutanota.com

Ransomware:

MD5	ed3c05bde9f0ea0f1321355b03ac42d0
SHA1	e2e14949d0cbc14cd3893da035cc13b509e70a18
SHA256	d4da69e424241c291c173c8b3756639c654432706e7def5025a649730868c4a1
Imphash	96c44fa1eee2c4e9b9e77d7bf42d59e6
SSDEEP	49152:nlpnltflwvk8sd4zs22ahkjzf/3odd8l9akyyxp02+:ntrwkmkff