July, 2020

Snatch Ransomware

Malware Analysis Case Study

LIFARS
your digital world, secured

Contents

Overview 3

Malware Analysis 3

Static Analysis and Metadata 3

Behavioral Analysis 6

Reverse Engineering 7

Strings Obfuscation 9

Attribution..13

Conclusion 14

References...14

IOCs..14

Appendix 1 – Example of deobfuscated strings 15

Appendix 2 – IDA Python script for strings deobfuscation 18

OVERVIEW

During last month LIFARS DFIR Team encountered various variants of Snatch Ransomware. This ransomware is known for its capability to reboot affected devices into Safe Mode, where most of the services and security tools are disabled. Then, in this weakened state, it encrypts user data.

We found multiple variants, either 32-bit and 64-bit binaries written in Go and packed with UPX packer. These Go binaries also contain obfuscated strings, so for accelerating the analysis we developed the IDA Python script for IDA Pro Disassembler, which can be used to automate strings extraction and deobfuscation.

The Snatch ransomware is operated by Snatch Team, which prepares unique samples tailored to their victims – the attackers can recognize these samples and appropriate decryption keys either by Victim name or by extension of encrypted files.

MALWARE ANALYSIS

Disclaimer

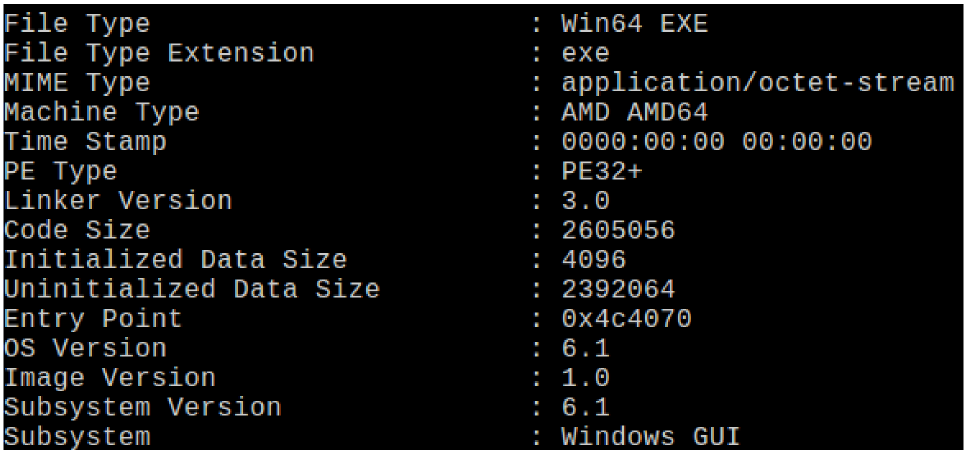
To maintain the privacy of our clients, we do not include the private samples in this case study. Instead, we use some publicly available samples for the analysis and demonstration of our results.

STATIC ANALYSIS AND METADATA

This case study is based on analysis of two publicly available samples with these SHA256 hashes:

- 794d549579812a90e14ef36b70c660900086e25a989e987bb642dff5239ee133
- 3160b4308dd9434ebb99e5747ec90d63722a640d329384b1ed536b59352dace6

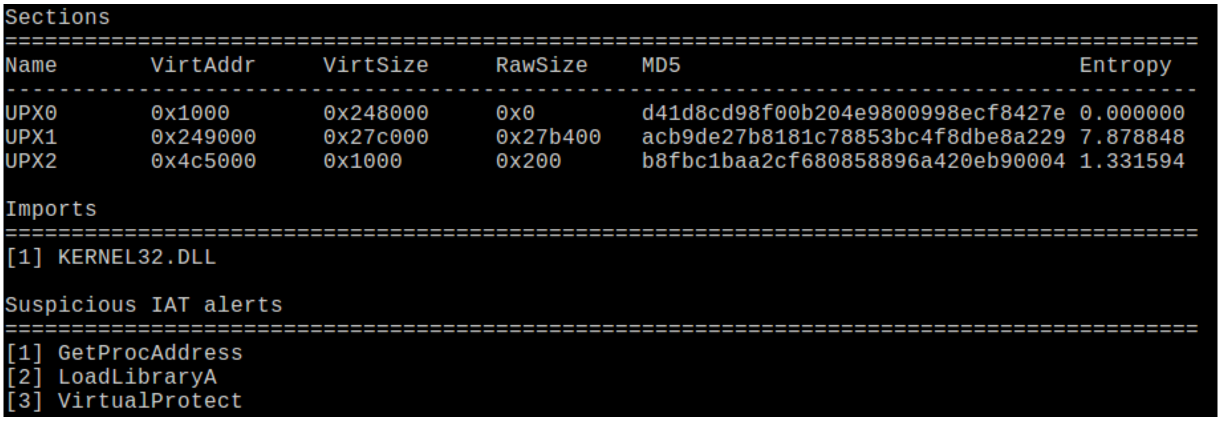
The samples are Portable Executable (PE, .exe) files with file size of approximately 2.5 MB. They are 64-bit applications designed for Windows 7 and later (OS Version 6.1), as we can see from `exiftool` and `file` commands.



```
File Type                       : Win64 EXE
File Type Extension             : exe
MIME Type                       : application/octet-stream
Machine Type                    : AMD AMD64
Time Stamp                      : 0000:00:00 00:00:00
PE Type                         : PE32+
Linker Version                  : 3.0
Code Size                       : 2605056
Initialized Data Size           : 4096
Uninitialized Data Size         : 2392064
Entry Point                     : 0x4c4070
OS Version                      : 6.1
Image Version                   : 1.0
Subsystem Version               : 6.1
Subsystem                       : Windows GUI
```

Figure 1. Meta information of samples extracted by Exiftool

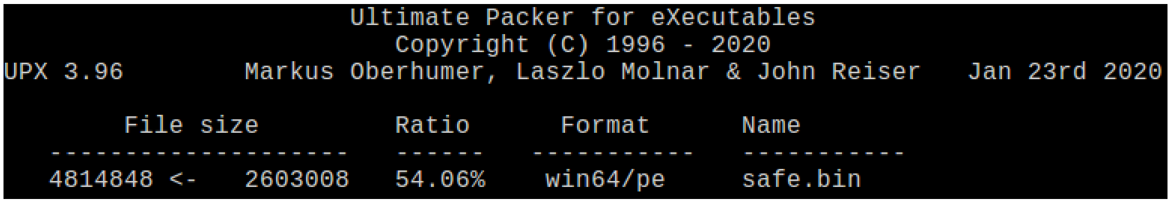
All the samples we captured as well as those downloaded from public malware repositories have been packed with UPX packer. This packer can be easily detected by suspicious names of the PE sections (UPX0, UPX1 and UPX2), their entropy and their virtual as well as real size. Also, the imported Win32 API functions such as GetProcAddress(), LoadLibraryA() and VirtualProtect() should give us another clue.



```
Sections
===========================================================================================
Name       VirtAddr     VirtSize     RawSize    MD5                              Entropy
-------------------------------------------------------------------------------------------
UPX0       0x1000       0x248000     0x0        d41d8cd98f00b204e9800998ecf8427e 0.000000
UPX1       0x249000     0x27c000     0x27b400   acb9de27b8181c78853bc4f8dbe8a229 7.878848
UPX2       0x4c5000     0x1000       0x200      b8fbc1baa2cf680858896a420eb90004 1.331594

Imports
===========================================================================================
[1] KERNEL32.DLL

Suspicious IAT alerts
===========================================================================================
[1] GetProcAddress
[2] LoadLibraryA
[3] VirtualProtect
```

Figure 2. Example of PE Sections of the analyzed sample suggesting the usage of UPX packer

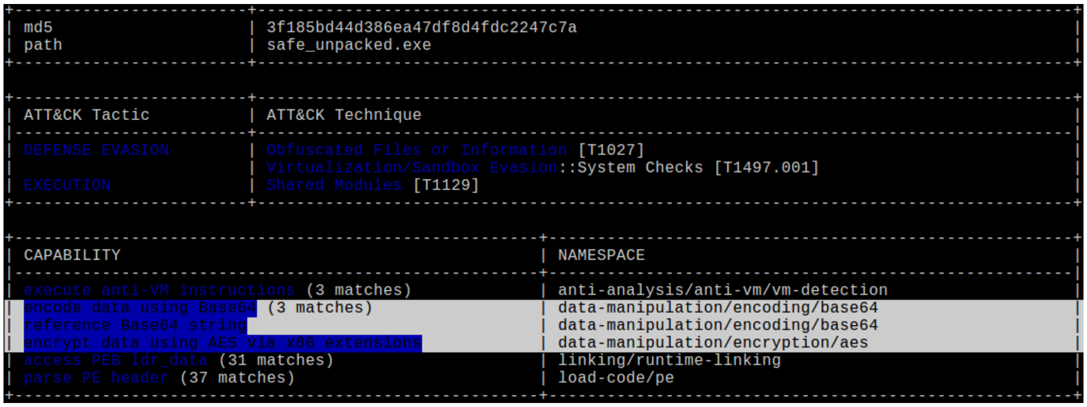
Unpacking of these samples is easy with the decompress option of the standard UPX tool. As a result, we get unpacked samples with size of approximately 4.6 MB.



```
                       Ultimate Packer for eXecutables
                          Copyright (C) 1996 - 2020
UPX 3.96        Markus Oberhumer, Laszlo Molnar & John Reiser   Jan 23rd 2020

        File size         Ratio      Format      Name
   --------------------   ------   -----------   -----------
   4814848 <-   2603008   54.06%    win64/pe     safe.bin
```

Figure 3. Unpacking the samples

When we examine capabilities of these unpacked samples with recently published tool called `capa`, we can see that it contains Base64-encoded strings (method of obfuscation) and that it should also be able to encrypt data. There are couple of other capabilities too, but for now, only these two are relevant for us.

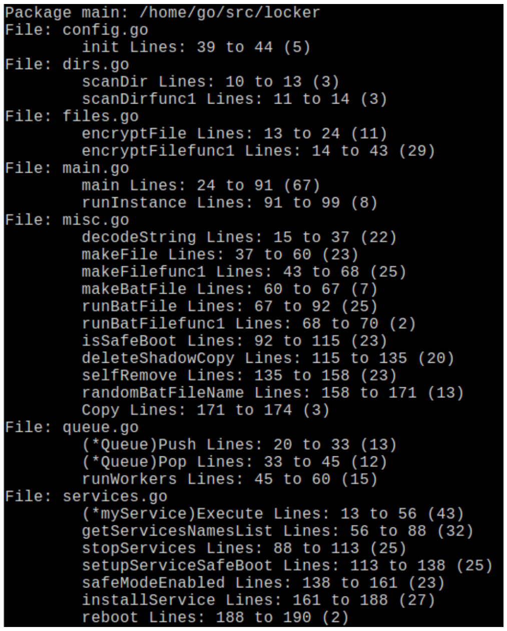


```
+------------------------+------------------------------------------------------------------------------------+
| md5                    | 3f185bd44d386ea47df8d4fdc2247c7a                                                   |
| path                   | safe_unpacked.exe                                                                  |
+------------------------+------------------------------------------------------------------------------------+

+------------------------+------------------------------------------------------------------------------------+
| ATT&CK Tactic          | ATT&CK Technique                                                                   |
|------------------------+------------------------------------------------------------------------------------|
| DEFENSE EVASION        | Obfuscated Files or Information [T1027]                                            |
|                        | Virtualization/Sandbox Evasion::System Checks [T1497.001]                          |
| EXECUTION              | Shared Modules [T1129]                                                             |
+------------------------+------------------------------------------------------------------------------------+

+-------------------------------------------------------+------------------------------------------------------+
| CAPABILITY                                            | NAMESPACE                                            |
|-------------------------------------------------------+------------------------------------------------------|
| execute anti-VM instructions (3 matches)              | anti-analysis/anti-vm/vm-detection                   |
| encode data using Base64 (3 matches)                  | data-manipulation/encoding/base64                    |
| reference Base64 string                               | data-manipulation/encoding/base64                    |
| encrypt data using AES via x86 extensions             | data-manipulation/encryption/aes                     |
| access PEB ldr_data (31 matches)                      | linking/runtime-linking                              |
| parse PE header (37 matches)                          | load-code/pe                                         |
+-------------------------------------------------------+------------------------------------------------------+
```

Figure 4. Capabilities of unpacked sample identified by the capa tool

These unpacked samples have been written in Go and compiled with compiler `go1.13.6 (2020-01-09T19:00:05Z).`

Further examination also reveals the code layout of original Go source code as well as file path on the attacker's dev machine: "/home/go/src/locker". We can see the original names of the functions and files as well as number of lines in the source code files for each function, see Figure 5.



```
Package main: /home/go/src/locker
File: config.go
        init Lines: 39 to 44 (5)
File: dirs.go
        scanDir Lines: 10 to 13 (3)
        scanDirfunc1 Lines: 11 to 14 (3)
File: files.go
        encryptFile Lines: 13 to 24 (11)
        encryptFilefunc1 Lines: 14 to 43 (29)
File: main.go
        main Lines: 24 to 91 (67)
        runInstance Lines: 91 to 99 (8)
File: misc.go
        decodeString Lines: 15 to 37 (22)
        makeFile Lines: 37 to 60 (23)
        makeFilefunc1 Lines: 43 to 68 (25)
        makeBatFile Lines: 60 to 67 (7)
        runBatFile Lines: 67 to 92 (25)
        runBatFilefunc1 Lines: 68 to 70 (2)
        isSafeBoot Lines: 92 to 115 (23)
        deleteShadowCopy Lines: 115 to 135 (20)
        selfRemove Lines: 135 to 158 (23)
        randomBatFileName Lines: 158 to 171 (13)
        Copy Lines: 171 to 174 (3)
File: queue.go
        (*Queue)Push Lines: 20 to 33 (13)
        (*Queue)Pop Lines: 33 to 45 (12)
        runWorkers Lines: 45 to 60 (15)
File: services.go
        (*myService)Execute Lines: 13 to 56 (43)
        getServicesNamesList Lines: 56 to 88 (32)
        stopServices Lines: 88 to 113 (25)
        setupServiceSafeBoot Lines: 113 to 138 (25)
        safeModeEnabled Lines: 138 to 161 (23)
        installService Lines: 161 to 188 (27)
        reboot Lines: 188 to 190 (2)
```

Figure 5. Source code layout of Snatch ransomware written in Go

While in other cases all these names can be fake to confuse the analyst and hide the main purpose of the analyzed sample (just imagine names such as InnocentFunction1, InnocentFunction2, etc.), in this case they seem to be relevant.

The source code layout and function names suggest that these samples will be able to scan directories and encrypt files in these directories – the main purpose of ransomware. Also, function `deleteShadowCopy` is probably used for removal of volume shadow snapshots – another characteristic action of many ransomwares.

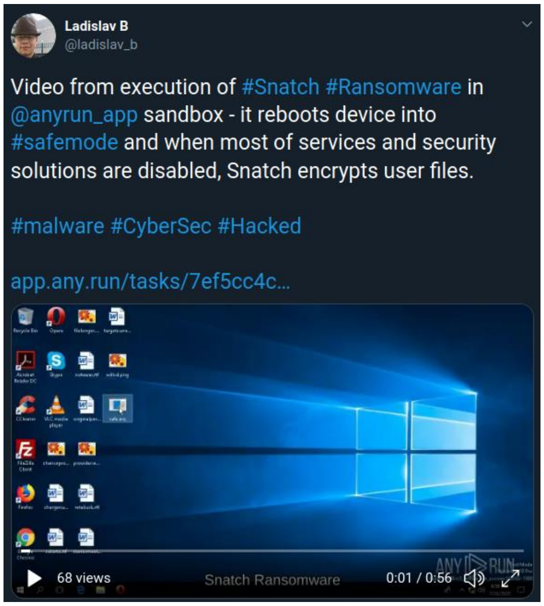
On the other hand, there are also functions related to batch files, services, safe mode and reboot – these can be used for setup of persistence and rebooting compromised device in safe mode. In safe mode, most of the services and security tools are not enabled, thus the encryption of user data should not be stopped or prevented by standard antivirus solution.

But all these ideas are currently only our hypotheses and we need to verify them.

BEHAVIORAL ANALYSIS

The easiest verification of our hypothesis can be achieved by execution of these samples in sandboxes. And because samples covered in this case study are public, we can use some of the public services. Or alternatively, perform search of previous uploads of these samples to some sandboxes and then review their behavior under normal circumstances.

In the sandbox analysis of one of the sample published by Ladislav Baco on Twitter we can see that our assumptions are correct.



Ladislav B
@ladislav_b

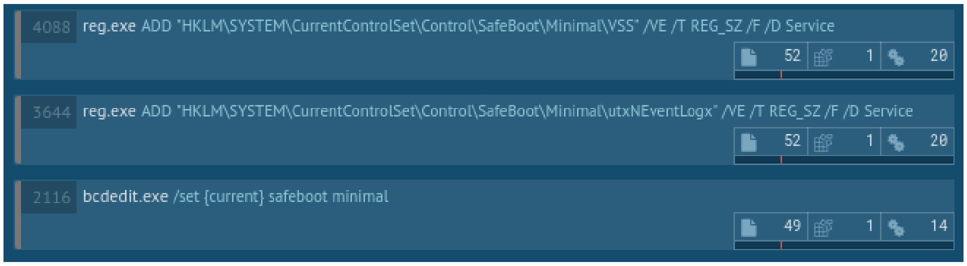
Video from execution of #Snatch #Ransomware in @anyrun_app sandbox - it reboots device into #safemode and when most of services and security solutions are disabled, Snatch encrypts user files.

#malware #CyberSec #Hacked

app.any.run/tasks/7ef5cc4c...

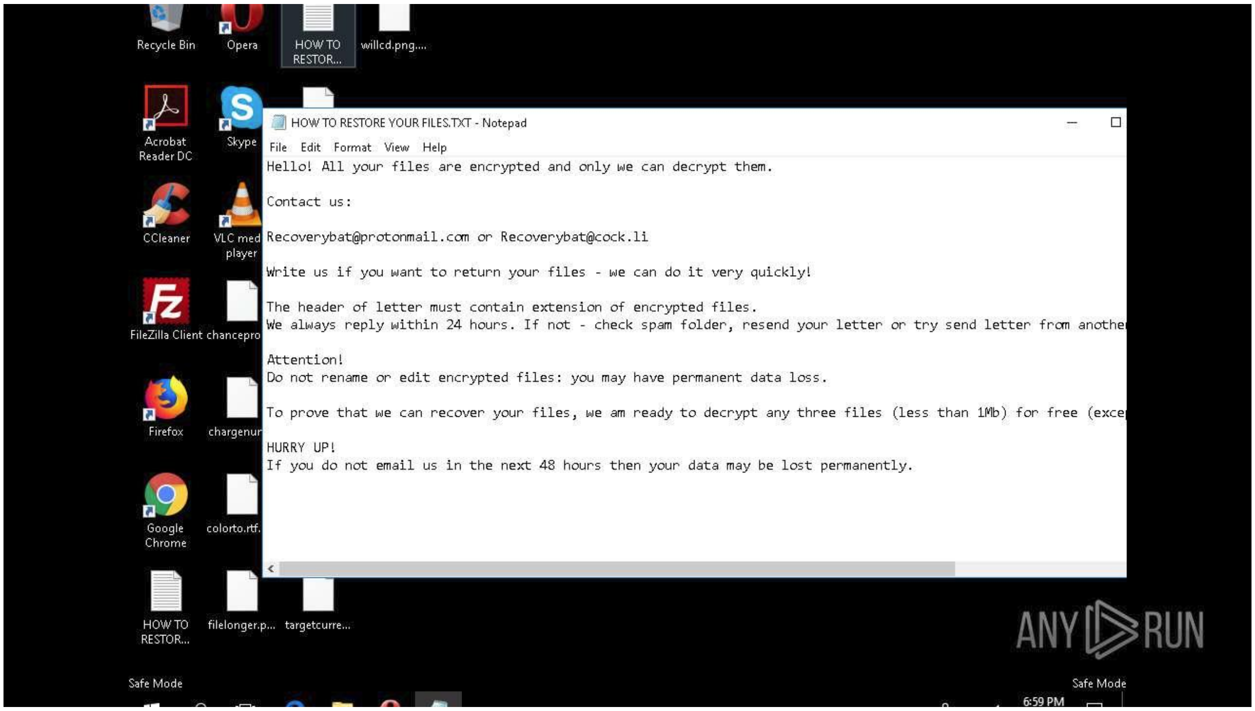
Figure 6. Sandbox Analysis and video published on Twitter

The sample installs itself as a service which runs also in minimal Safe Mode (and enables the VSS service) and then reboots the device into Safe Mode in which it continues with the encryption of user data.



```
4088 reg.exe ADD "HKLM\SYSTEM\CurrentControlSet\Control\SafeBoot\Minimal\VSS" /VE /T REG_SZ /F /D Service
3644 reg.exe ADD "HKLM\SYSTEM\CurrentControlSet\Control\SafeBoot\Minimal\utxNEventLogx" /VE /T REG_SZ /F /D Service
2116 bcdedit.exe /set {current} safeboot minimal
```

Figure 7. Installing services which will run also in minimal Safe Mode



```
Hello! All your files are encrypted and only we can decrypt them.

Contact us:

Recoverybat@protonmail.com or Recoverybat@cock.li

Write us if you want to return your files - we can do it very quickly!

The header of letter must contain extension of encrypted files.
We always reply within 24 hours. If not - check spam folder, resend your letter or try send letter from anothe

Attention!
Do not rename or edit encrypted files: you may have permanent data loss.

To prove that we can recover your files, we am ready to decrypt any three files (less than 1Mb) for free (exce

HURRY UP!
If you do not email us in the next 48 hours then your data may be lost permanently.
```

Figure 8. Files encrypted in Safe Mode and dropped Ransom Note by Snatch Ransomware

REVERSE ENGINEERING

Now, after the static analysis, we are aware of the capabilities of analyzed Snatch ransomware and we have verified them during the behavioral analysis in sandbox.

However, there are still couple of unanswered questions which can be valuable for our clients. Especially the following:

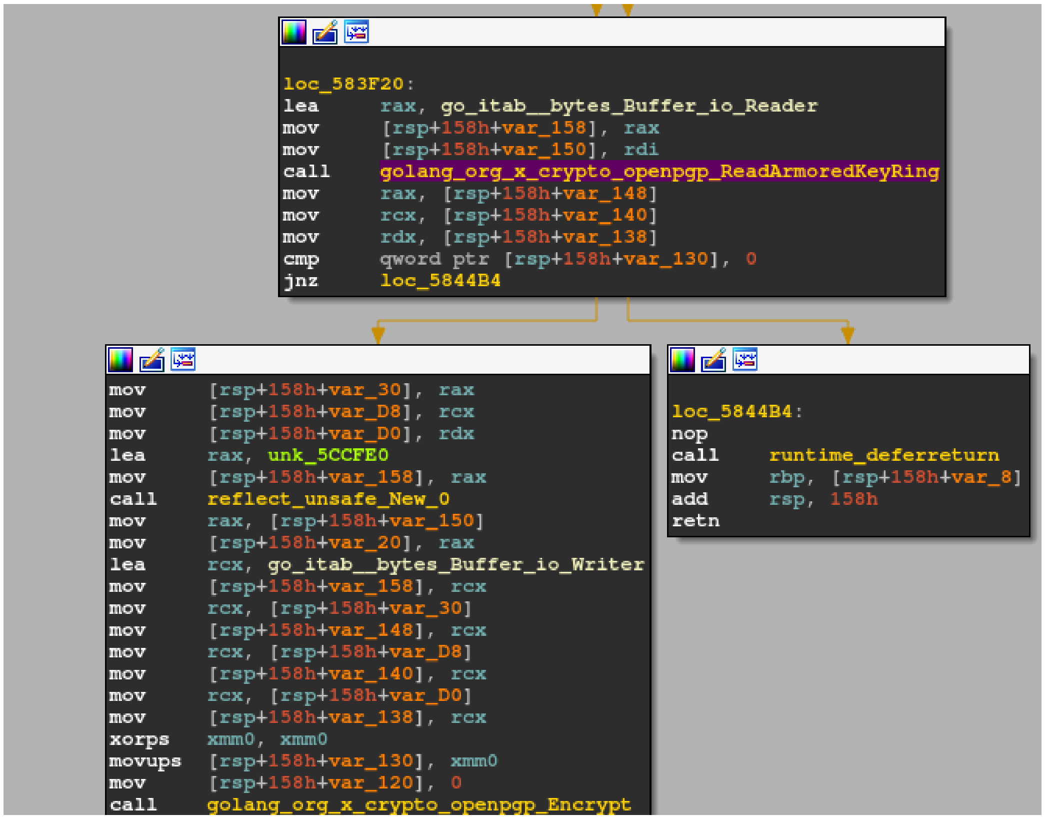
- Is there some data exfiltration performed by this ransomware?
- How is it spreading? Can it spread by its own, or is it delivered by some other malware?
- What kind of cipher/encryption does it use?

- What types of files does it encrypt?
- Who is behind this ransomware?

To answer the first four questions and partially the fifth question, we proceeded with reverse engineering of the ransomware. During this process, we disassembled and analyzed its code for any evidence of network activity, which could indicate data exfiltration or spreading of the ransomware. While some Go packages for network communication are present, they are not used. We also didn't observe any network activity during behavioral analysis.

We also didn't see any capabilities of self-spreading. Instead, based on our research and analysis of our cases, the attackers seem to deliver this ransomware by another way after they obtain initial access to the victim's network. For example, it can be deployed via RDP or by using similar remote access methods.

When we look on the `encryptFile` function from `main` package, we notice calls to `ReadArmoredKeyRing` and `Encrypt` from the package `golang.org/x/crypto/openpgp`. By analyzing the function for file encryption, we can see that these files are encrypted using PGP, public-key cryptography. The public key is hardcoded and stored as obfuscated string (see below for more info about this obfuscation).



```
loc_583F20:
lea     rax, go_itab__bytes_Buffer_io_Reader
mov     [rsp+158h+var_158], rax
mov     [rsp+158h+var_150], rdi
call    golang_org_x_crypto_openpgp_ReadArmoredKeyRing
mov     rax, [rsp+158h+var_148]
mov     rcx, [rsp+158h+var_140]
mov     rdx, [rsp+158h+var_138]
cmp     qword ptr [rsp+158h+var_130], 0
jnz     loc_5844B4
```

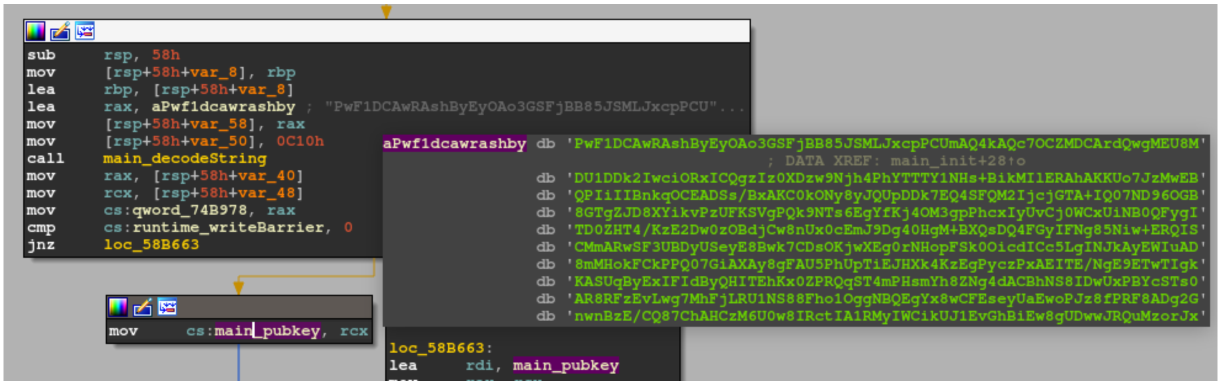
```
mov     [rsp+158h+var_30], rax
mov     [rsp+158h+var_D8], rcx
mov     [rsp+158h+var_D0], rdx
lea     rax, unk_5CCFE0
mov     [rsp+158h+var_158], rax
call    reflect_unsafe_New_0
mov     rax, [rsp+158h+var_150]
mov     [rsp+158h+var_20], rax
lea     rcx, go_itab__bytes_Buffer_io_Writer
mov     [rsp+158h+var_158], rcx
mov     rcx, [rsp+158h+var_30]
mov     [rsp+158h+var_148], rcx
mov     rcx, [rsp+158h+var_D8]
mov     [rsp+158h+var_140], rcx
mov     rcx, [rsp+158h+var_D0]
mov     [rsp+158h+var_138], rcx
xorps   xmm0, xmm0
movups  [rsp+158h+var_130], xmm0
mov     [rsp+158h+var_120], 0
call    golang_org_x_crypto_openpgp_Encrypt
```

```
loc_5844B4:
nop
call    runtime_deferreturn
mov     rbp, [rsp+158h+var_8]
add     rsp, 158h
retn
```

Figure 9. Importing public key and preparing PGP for file encryption

STRINGS OBFUSCATION

The pubkey and other strings used by Snatch ransomware are stored in obfuscated form, and they are decoded in the main.init function. They are stored as Base64-encoded strings, which are xored with hardcoded key and again encoded with Base64.



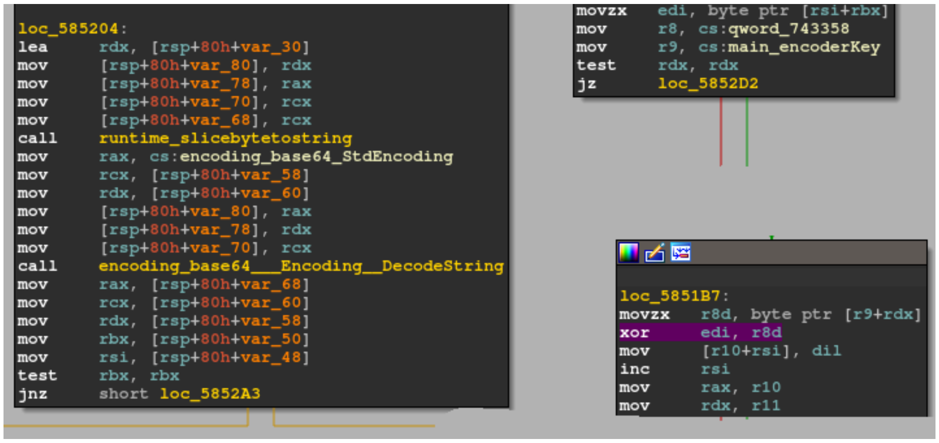
```
sub     rsp, 58h
mov     [rsp+58h+var_8], rbp
lea     rbp, [rsp+58h+var_8]
lea     rax, aPwf1dcawrashby ; "PwF1DCAwRAshByEyOAo3GSFjBB85JSMLJxcpPCU"...
mov     [rsp+58h+var_58], rax
mov     [rsp+58h+var_50], 0C10h
call    main_decodeString
mov     rax, [rsp+58h+var_40]
mov     rcx, [rsp+58h+var_48]
mov     cs:qword_74B978, rax
cmp     cs:runtime_writeBarrier, 0
jnz     loc_58B663
```

```
aPwf1dcawrashby db 'PwF1DCAwRAshByEyOAo3GSFjBB85JSMLJxcpPCUmAQ4kAQc7OCZMDCArdQwgMEU8M'
                                        ; DATA XREF: main_init+28↑o
                db 'DU1DDk2IwciORxICQgzIz0XDzw9Njh4PhYTTTY1NHs+BikMIlERAhAKKUo7JzMwEB'
                db 'QPIiIIBnkqOCEADSs/BxAKC0kONy8yJQUpDDk7EQ4SFQM2IjcjGTA+IQ07ND96OGB'
                db '8GTgZJD8XYikvPzUFKSVgPQk9NTs6EgYfKj4OM3gpPhcxIyUvCj0WCxUiNB0QFygI'
                db 'TD0ZHT4/KzE2Dw0zOBdjCw8nUx0cEmJ9Dg40HgM+BXQsDQ4FGyIFNg85Niw+ERQIS'
                db 'CMmARwSF3UBDyUSeyE8Bwk7CDsOKjwXEg0rNHopFSk0OicdICc5LgINJkAyEWIuAD'
                db '8mMHokFCkPPQ07GiAXAy8gFAU5PhUpTiEJHXk4KzEgPyczPxAEITE/NgE9ETwTIgk'
                db 'KASUqByExIFIdByQHITEhKx0ZPRQqST4mPHsmYh8ZNg4dACBhNS8IDwUxPBYcSTs0'
                db 'AR8RFzEvLwg7MhFjLRU1NS88Fho1OggNBQEgYx8wCFEseyUaEwoPJz8fPRF8ADg2G'
                db 'nwnBzE/CQ87ChAHCzM6U0w8IRctIA1RMyIWCikUJ1EvGhBiEw8gUDwwJRQuMzorJx'
```

```
mov     cs:main_pubkey, rcx
```

```
loc_58B663:
lea     rdi, main_pubkey
```

Figure 10. Decoding of obfuscated public key

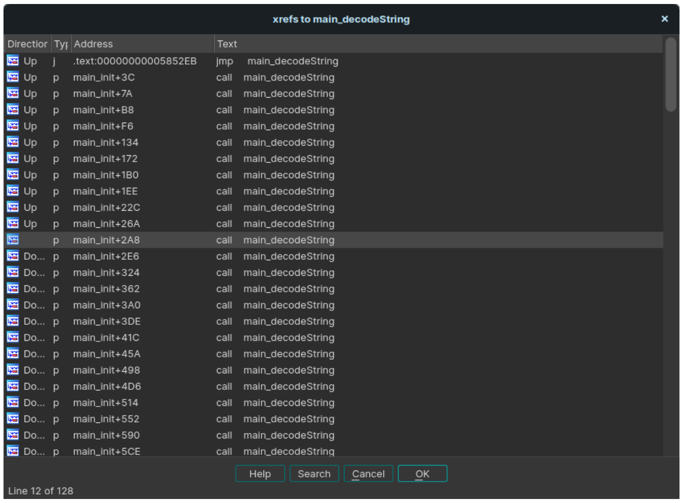


```
loc_585204:
lea     rdx, [rsp+80h+var_30]
mov     [rsp+80h+var_80], rdx
mov     [rsp+80h+var_78], rax
mov     [rsp+80h+var_70], rcx
mov     [rsp+80h+var_68], rcx
call    runtime_slicebytetostring
mov     rax, cs:encoding_base64_StdEncoding
mov     rcx, [rsp+80h+var_58]
mov     rdx, [rsp+80h+var_60]
mov     [rsp+80h+var_80], rax
mov     [rsp+80h+var_78], rdx
mov     [rsp+80h+var_70], rcx
call    encoding_base64___Encoding__DecodeString
mov     rax, [rsp+80h+var_68]
mov     rcx, [rsp+80h+var_60]
mov     rdx, [rsp+80h+var_58]
mov     rbx, [rsp+80h+var_50]
mov     rsi, [rsp+80h+var_48]
test    rbx, rbx
jnz     short loc_5852A3
```

```
movzx   edi, byte ptr [rsi+rbx]
mov     r8, cs:qword_743358
mov     r9, cs:main_encoderKey
test    rdx, rdx
jz      loc_5852D2
```

```
loc_5851B7:
movzx   r8d, byte ptr [r9+rdx]
xor     edi, r8d
mov     [r10+rsi], dil
inc     rsi
mov     rax, r10
mov     rdx, r11
```

Figure 11. Excerpts from decodeString function – Base64 encoding and encoderKey used for XOR cipher



xrefs to main_decodeString

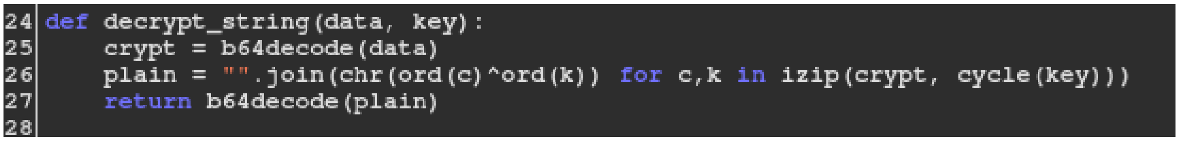
Directior	Typ	Address	Text
Up	j	.text:00000000005852EB	jmp main_decodeString
Up	p	main_init+3C	call main_decodeString
Up	p	main_init+7A	call main_decodeString
Up	p	main_init+B8	call main_decodeString
Up	p	main_init+F6	call main_decodeString
Up	p	main_init+134	call main_decodeString
Up	p	main_init+172	call main_decodeString
Up	p	main_init+1B0	call main_decodeString
Up	p	main_init+1EE	call main_decodeString
Up	p	main_init+22C	call main_decodeString
Up	p	main_init+26A	call main_decodeString
	p	main_init+2A8	call main_decodeString
Do...	p	main_init+2E6	call main_decodeString
Do...	p	main_init+324	call main_decodeString
Do...	p	main_init+362	call main_decodeString
Do...	p	main_init+3A0	call main_decodeString
Do...	p	main_init+3DE	call main_decodeString
Do...	p	main_init+41C	call main_decodeString
Do...	p	main_init+45A	call main_decodeString
Do...	p	main_init+498	call main_decodeString
Do...	p	main_init+4D6	call main_decodeString
Do...	p	main_init+514	call main_decodeString
Do...	p	main_init+552	call main_decodeString
Do...	p	main_init+590	call main_decodeString
Do...	p	main_init+5CE	call main_decodeString

Line 12 of 128

Figure 12. Calls to decodeString function

There are lot of calls to `main.decodeString` function in `main.init`, and while it is possible to decode these obfuscated strings manually, it is time consuming. So, we decided to automate the process of deobfuscation and we created our own IDA Python script for this.

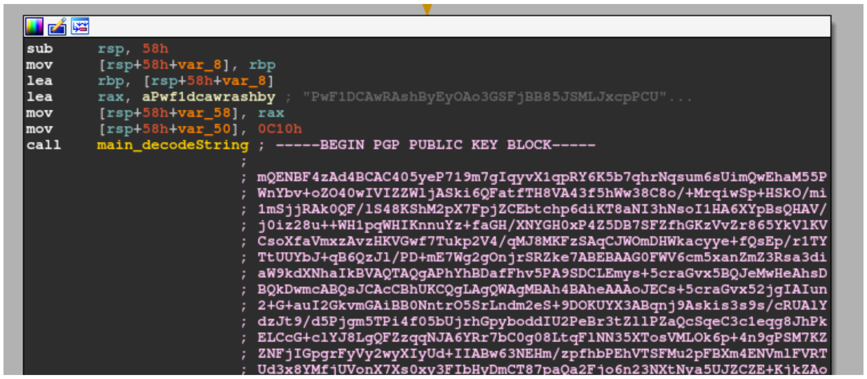
The main idea is to reimplement the base64 – xor – base64 decoding in Python, identify all calls to `decodeString` function, extract its parameters (the address of obfuscated string and its length) and extract the hardcoded `encoderKey` used for xor-cipher.



```
def decrypt_string(data, key):
    crypt = b64decode(data)
    plain = "".join(chr(ord(c)^ord(k)) for c,k in izip(crypt, cycle(key)))
    return b64decode(plain)
```

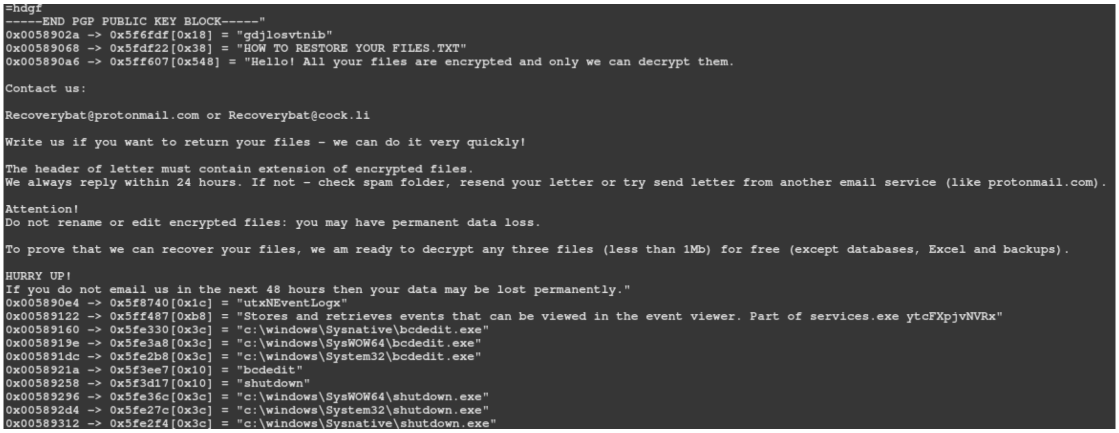
Figure 13. IDA Python reimplementation of base64-xor-base64 decoding

Because we noticed both 32-bit and 64-bit versions of Snatch ransomware, our script supports both 32-bit and 64-bit platforms for the string deobfuscation. And as a result, our script puts the deobfuscated string as a comment to each call to main.decodeString function as well as it produces list of all deobfuscated strings in output windows of IDA.



```
sub      rsp, 58h
mov      [rsp+58h+var_8], rbp
lea      rbp, [rsp+58h+var_8]
lea      rax, aPwf1dcawrashby ; "PwF1DCAwRAshByEyOAo3GSFjBB85JSMLJxcpPCU"...
mov      [rsp+58h+var_58], rax
mov      [rsp+58h+var_50], 0C10h
call     main_decodeString ; -----BEGIN PGP PUBLIC KEY BLOCK-----
                          ;
                          ; mQENBF4zAd4BCAC405yeP719m7gIqyvX1qpRY6K5b7qhrNqsum6sUimQwEhaM55P
                          ; WnYbv+oZO40wIVIZZWljASki6QFatfTH8VA43f5hWw38C8o/+MrqiwSp+HSkO/mi
                          ; 1mSjjRAk0QF/lS48KShM2pX7FpjZCEbtchp6diKT8aNI3hNsoI1HA6XYpBsQHAV/
                          ; j0iz28u++WH1pqWHIKnnuYz+faGH/XNYGH0xP4Z5DB7SFZfhGKzVvZr865YkVlKV
                          ; CsoXfaVmxzAvzHKVGwf7Tukp2V4/qMJ8MKFzSAqCJWOmDHWkacyye+fQsEp/r1TY
                          ; TtUUYbJ+qB6QzJl/PD+mE7Wg2gOnjrSRZke7ABEBAAG0FWV6cm5xanZmZ3Rsa3di
                          ; aW9kdXNhaIkBVAQTAQgAPhYhBDafFhv5PA9SDCLEmys+5craGvx5BQJeMwHeAhsD
                          ; BQkDwmcABQsJCAcCBhUKCQgLAgQWAgMBAh4BAheAAAoJECs+5craGvx52jgIAIun
                          ; 2+G+auI2GkvmGAiBB0NntrO5SrLndm2eS+9DOKUYX3ABqnj9Askis3s9s/cRUAlY
                          ; dzJt9/d5Pjgm5TPi4f05bUjrhGpyboddIU2PeBr3tZllPZaQcSqeC3c1eqg8JhPk
                          ; ELCcG+clYJ8LgQFZzqqNJA6YRr7bC0g08LtqFlNN35XTosVMLOk6p+4n9gPSM7KZ
                          ; ZNFjIGpgrFyVy2wyXIyUd+IIABw63NEHm/zpfhbPEhVTSFMu2pFBXm4ENVmlFVRT
                          ; Ud3x8YMfjUVonX7Xs0xy3FIbHyDmCT87paQa2Fjo6n23NXtNya5UJZCZE+KjkZAo
```

Figure 14. Deobfuscated public key



```
=hdgf
-----END PGP PUBLIC KEY BLOCK-----"
0x0058902a -> 0x5f6fdf[0x18] = "gdjlosvtnib"
0x00589068 -> 0x5fdf22[0x38] = "HOW TO RESTORE YOUR FILES.TXT"
0x005890a6 -> 0x5ff607[0x548] = "Hello! All your files are encrypted and only we can decrypt them.

Contact us:

Recoverybat@protonmail.com or Recoverybat@cock.li

Write us if you want to return your files - we can do it very quickly!

The header of letter must contain extension of encrypted files.
We always reply within 24 hours. If not - check spam folder, resend your letter or try send letter from another email service (like protonmail.com).

Attention!
Do not rename or edit encrypted files: you may have permanent data loss.

To prove that we can recover your files, we am ready to decrypt any three files (less than 1Mb) for free (except databases, Excel and backups).

HURRY UP!
If you do not email us in the next 48 hours then your data may be lost permanently."
0x005890e4 -> 0x5f8740[0x1c] = "utxNEventLogx"
0x00589122 -> 0x5ff487[0xb8] = "Stores and retrieves events that can be viewed in the event viewer. Part of services.exe ytcFXpjvNVRx"
0x00589160 -> 0x5fe330[0x3c] = "c:\windows\Sysnative\bcdedit.exe"
0x0058919e -> 0x5fe3a8[0x3c] = "c:\windows\SysWOW64\bcdedit.exe"
0x005891dc -> 0x5fe2b8[0x3c] = "c:\windows\System32\bcdedit.exe"
0x0058921a -> 0x5f3ee7[0x10] = "bcdedit"
0x00589258 -> 0x5f3d17[0x10] = "shutdown"
0x00589296 -> 0x5fe36c[0x3c] = "c:\windows\SysWOW64\shutdown.exe"
0x005892d4 -> 0x5fe27c[0x3c] = "c:\windows\System32\shutdown.exe"
0x00589312 -> 0x5fe2f4[0x3c] = "c:\windows\Sysnative\shutdown.exe"
```

Figure 15. Deobfuscated strings – file extension, ransom note, service name, commands, ...

By reviewing the deobfuscated strings, the variables, their location and usage, we can see what kind of files are encrypted by Snatch ransomware.

There is a variable called `ignoreFileExtensions`. It contains list of extensions which are ignored in `main.scanDir` function. The following file extensions are ignored:

- .exe,
- .dll,
- .sys,
- .ini,
- .bat,
- .lnk,
- Already encrypted files (with already deobfuscated extension from config).

In addition, there are couple of files and locations which are ignored during scanDir and encryption process.

Ignored files:

- Files with ransomnote (deobfuscated filename from config)

Ignored Program Data folders:

- microsoft,
- start menu,
- templates,
- favorites.

Ignored Program Files Folders:

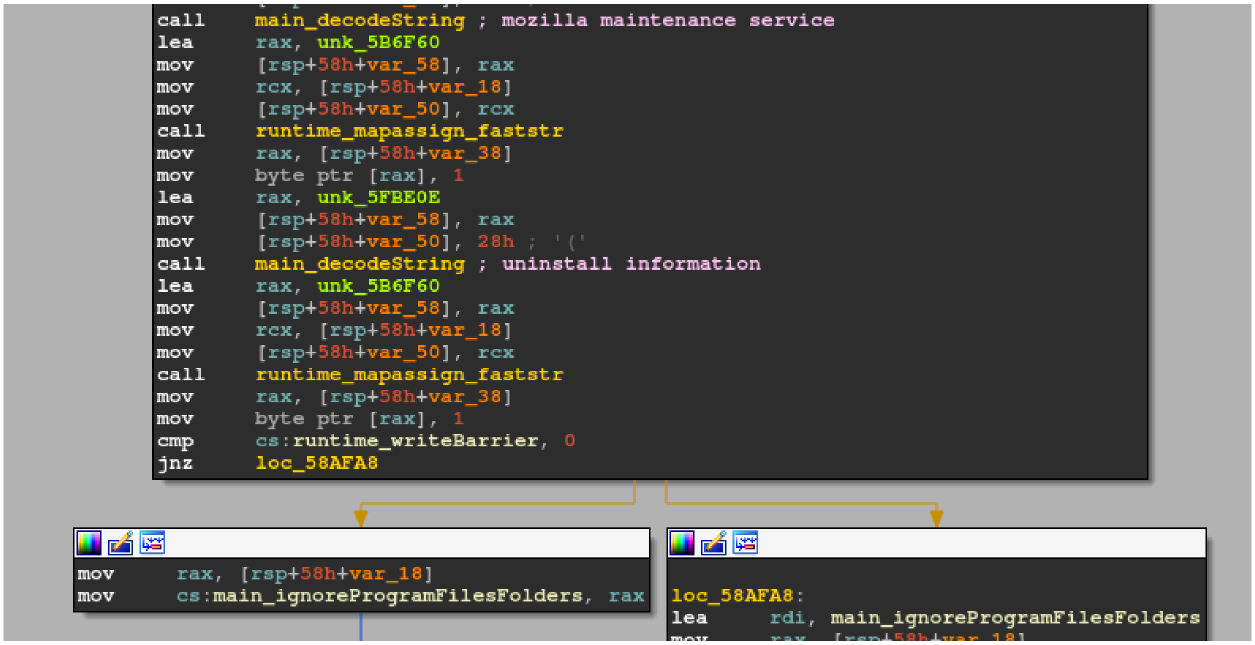
- windows,
- perflogs,
- $recycle.bin,
- system volume information,
- common files,
- dvdmaker,
- internet explorer,
- msbuild,
- microsoft games,
- mozilla firefox,
- reference assemblies,
- tap-windows,
- windows journal,
- windows mail,

- windows media player,
- windows nt,
- windows photo viewer,
- windows sidebar,
- windows portable devices,
- microsoft.net,
- mozilla maintenance service,
- uninstall information.

Ignored Root Folders:

- windows,
- perflogs,
- recovery,
- $recycle.bin,
- system volume information.

Therefore, all files are encrypted except those on the above ignore lists.



```
call    main_decodeString ; mozilla maintenance service
lea     rax, unk_5B6F60
mov     [rsp+58h+var_58], rax
mov     rcx, [rsp+58h+var_18]
mov     [rsp+58h+var_50], rcx
call    runtime_mapassign_faststr
mov     rax, [rsp+58h+var_38]
mov     byte ptr [rax], 1
lea     rax, unk_5FBE0E
mov     [rsp+58h+var_58], rax
mov     [rsp+58h+var_50], 28h ; '('
call    main_decodeString ; uninstall information
lea     rax, unk_5B6F60
mov     [rsp+58h+var_58], rax
mov     rcx, [rsp+58h+var_18]
mov     [rsp+58h+var_50], rcx
call    runtime_mapassign_faststr
mov     rax, [rsp+58h+var_38]
mov     byte ptr [rax], 1
cmp     cs:runtime_writeBarrier, 0
jnz     loc_58AFA8
```

```
mov     rax, [rsp+58h+var_18]
mov     cs:main_ignoreProgramFilesFolders, rax
```

```
loc_58AFA8:
lea     rdi, main_ignoreProgramFilesFolders
mov     rax, [rsp+58h+var_18]
```

Figure 16. Ignored folders example

ATTRIBUTION

One unanswered question still remains. Who is behind it?

, In a report by Sophos from December 2019 , they notice that threat actors behind Snatch ransomware refer to themselves as “Snatch Team”. It seems that the members of Snatch Team are probably Russian-speaking. They were looking for new members and partners and they write their posts on forums in Russian language and they seem to be working with Russians-speaking people only.

It looks like their TTPs also include automation in brute-force attacks against the vulnerable services, and they also cooperate with partners who already established such access to their victim's networks.

Translated from one of their message:

“Looking for affiliate partners with access to RDP\VNC\TeamViewer\WebShell\SQL inj [SQL injection] in corporate networks, stores and other companies.”

(source: https://news.sophos.com/en-us/2019/12/09/snatch-ransomware-reboots-pcs-into-safe-mode-to-bypass-protection/)

Also, in recently documented cases from June, the Snatch team was able to bruteforce domain admin credentials via RDP, then use meterpreter for lateral movement, compromise the domain controller and encrypt the whole network in less than 5 hours.

(source: https://thedfirreport.com/2020/06/21/snatch-ransomware/)

This threat actor requests the ransom in Bitcoin. Prices for decryption are from few thousand up to hundred thousand dollars, but it seems that it is possible to negotiate the ransom. During our investigation, we found several transactions to their Bitcoin wallets, with raising trend.

CONCLUSION

LIFARS DFIR Team investigated incidents during which we found multiple versions of the Snatch Ransomware. We analyzed these samples and developed helper tool for malware analysts for the decoding of the obfuscated strings embedded in this ransomware.

We also observed similar behavior and TTPs of the Threat actor as were documented previously.

REFERENCES

- https://github.com/Lifars/IDA-scripts/blob/master/snatch_decrypt_strings.py
- https://twitter.com/ladislav_b/status/1286766312149200896
- https://app.any.run/tasks/7ef5cc4c-e450-4385-a134-2fbe56d35ef2/
- https://app.any.run/tasks/4bf66978-116d-49e6-b935-9ec7c2fe4c7f/
- https://news.sophos.com/en-us/2019/12/09/snatch-ransomware-reboots-pcs-into-safe-mode-to-bypass-protection/
- https://thedfirreport.com/2020/06/21/snatch-ransomware/

IOCS

- MD5:
    - 6660b6386c3a860f05da7199d78d2b2f
    - 2bbff2111232d73a93cd435300d0a07e
- SHA1:
    - d09300e3919cd44128e595864ccafd837843b9e0
    - b93d633d379052f0a15b0f9c7094829461a86dbb
- SHA256:
    - 794d549579812a90e14ef36b70c660900086e25a989e987bb642dff5239ee133
    - 3160b4308dd9434ebb99e5747ec90d63722a640d329384b1ed536b59352dace6
- FileName:
    - safe.exe

APPENDIX 1 – EXAMPLE OF DEOBFUSCATED STRINGS

```
0x00588fec -> 0x5ffb4f[0xc10] = "-----BEGIN PGP PUBLIC KEY BLOCK-----
mQENBF4zAd4BCAC405yeP719m7gIqyvX1qpRY6K5b7qhrNqsum6sUimQwEhaM55P
WnYbv+oZO40wIVIZZWljASki6QFatfTH8VA43f5hWw38C8o/+MrqiwSp+HSkO/mi
1mSjjRAk0QF/lS48KShM2pX7FpjZCEbtchp6diKT8aNI3hNsoI1HA6XYpBsQHAV/
j0iz28u++WH1pqWHIKnnuYz+faGH/XNYGH0xP4Z5DB7SFZfhGKzVvZr865YkVlKV
CsoXfaVmxzAvzHKVGwf7Tukp2V4/qMJ8MKFzSAqCJWOmDHWkacyye+fQsEp/r1TY
TtUUYbJ+qB6QzJl/PD+mE7Wg2gOnjrSRZke7ABEBAAG0FWV6cm5xanZmZ3Rsa3di
aW9kdXNhaIkBVAQTAQgAPhYhBDafFhv5PA9SDCLEmys+5craGvx5BQJeMwHeAhsD
BQkDwmcABQsJCAcCBhUKCQgLAgQWAgMBAh4BAheAAAoJECs+5craGvx52jgIAIun
2+G+auI2GkvmGAiBB0NntrO5SrLndm2eS+9DOKUYX3ABqnj9Askis3s9s/cRUAlY
dzJt9/d5Pjgm5TPi4f05bUjrhGpyboddIU2PeBr3tZllPZaQcSqeC3c1eqg8JhPk
ELCcG+clYJ8LgQFZzqqNJA6YRr7bC0g08LtqFlNN35XTosVMLOk6p+4n9gPSM7KZ
ZNFjIGpgrFyVy2wyXIyUd+IIABw63NEHm/zpfhbPEhVTSFMu2pFBXm4ENVmlFVRT
Ud3x8YMfjUVonX7Xs0xy3FIbHyDmCT87paQa2Fjo6n23NXtNya5UJZCZE+KjkZAo
VYoExCYR08rNJeq+k765AQ0EXjMB3gEIAO1BuQYOuZzFXTgbdNfr5iTHR8b8LJpg
KyQZoaJcV1x+o5qxKk2xt3PDCYXxBLJjYKWzB9yF/EVHf43FokCSdwUXAcsbDbWl
wL/QsPtl7PAOtOGXU17dWCp9/YvNVqw+KqBr3FilJFD7eoJ4mjcj9ARJairleg7i
sEn2DwRaVXZ4Co454DWEAFgrdqZWNupzWDsxkH49iCbf0KE9m5GvhQSpUASmEw45
g+IE4S/jctUYJMeEye/zoCTBWG1z2cSp0Q7CcK97/wDXKMhTHnTAkQI6PEFdmaLr
jJg0ApfHT2I/qyldbA/+fyTHuSfRz09Yu6R9tp+gOs88blsIUtv84SEAEQEAAYkB
NgQYAQgAIBYhBDafFhv5PA9SDCLEmys+5craGvx5BQJeMwHeAhsMAAoJECs+5cra
Gvx5HzoH/3+O1cCiGUksiiHJ+niw5NPFIwUv4Doqzpuqn4K4GxcEzU+W/iWOtJXC
cm4AmSLcDJ2ODb+KcMf+DjxKLsiHiKYPoN+CdPD2HtC+HxQrgGCLtXs/6Ru/MWP4
emsKhzmcRBUJVG4Kp5HwClcNLp8EQI2mTAGfTXC/gwvjPPebfDcPgHbQcJ3hvqXC
BgVNLue1GROzMSpHXenbG5ubyi4eVTeIsI34Q/22+nyr5U4dGM22La4RkDqlvlkN
poEyoHbxRnC2+lI551Do5t1jUoddy3vTQvg6TJxyEfg5Wg7RhP5EWhc2QfSpySNR
88nJHhx5gEgL6BHQTbioZsVg+EPqODk=
=hdgf
-----END PGP PUBLIC KEY BLOCK-----"
0x0058902a -> 0x5f6fdf[0x18] = "gdjlosvtnib"
0x00589068 -> 0x5fdf22[0x38] = "HOW TO RESTORE YOUR FILES.TXT"
0x005890a6 -> 0x5ff607[0x548] = "Hello! All your files are encrypted and only we can decrypt
them.

Contact us:

Recoverybat@protonmail.com or Recoverybat@cock.li

Write us if you want to return your files - we can do it very quickly!

The header of letter must contain extension of encrypted files.
We always reply within 24 hours. If not - check spam folder, resend your letter or try send
letter from another email service (like protonmail.com).

Attention!
Do not rename or edit encrypted files: you may have permanent data loss.

To prove that we can recover your files, we am ready to decrypt any three files (less than
1Mb) for free (except databases, Excel and backups).

HURRY UP!
If you do not email us in the next 48 hours then your data may be lost permanently."
0x005890e4 -> 0x5f8740[0x1c] = "utxNEventLogx"
0x00589122 -> 0x5ff487[0xb8] = "Stores and retrieves events that can be viewed in the event
viewer. Part of services.exe ytcFXpjvNVRx"
0x00589160 -> 0x5fe330[0x3c] = "c:\windows\Sysnative\bcdedit.exe"
0x0058919e -> 0x5fe3a8[0x3c] = "c:\windows\SysWOW64\bcdedit.exe"
0x005891dc -> 0x5fe2b8[0x3c] = "c:\windows\System32\bcdedit.exe"
0x0058921a -> 0x5f3ee7[0x10] = "bcdedit"
0x00589258 -> 0x5f3d17[0x10] = "shutdown"
0x00589296 -> 0x5fe36c[0x3c] = "c:\windows\SysWOW64\shutdown.exe"
0x005892d4 -> 0x5fe27c[0x3c] = "c:\windows\System32\shutdown.exe"
0x00589312 -> 0x5fe2f4[0x3c] = "c:\windows\Sysnative\shutdown.exe"
0x00589350 -> 0x5f1c1a[0x8] = "A:"
0x0058938e -> 0x5f1c2a[0x8] = "B:"
```

```
0x005893cc -> 0x5f1c22[0x8] = "C:"
0x0058940a -> 0x5f1c02[0x8] = "D:"
0x00589448 -> 0x5f1bfa[0x8] = "E:"
0x00589486 -> 0x5f1c12[0x8] = "F:"
0x005894c4 -> 0x5f1c0a[0x8] = "G:"
0x00589502 -> 0x5f1be2[0x8] = "H:"
0x00589540 -> 0x5f1bda[0x8] = "I:"
0x0058957e -> 0x5f1bf2[0x8] = "J:"
0x005895bc -> 0x5f1bea[0x8] = "K:"
0x005895fa -> 0x5f1caa[0x8] = "L:"
0x00589638 -> 0x5f1ca2[0x8] = "M:"
0x00589676 -> 0x5f1cba[0x8] = "N:"
0x005896b4 -> 0x5f1cb2[0x8] = "O:"
0x005896f2 -> 0x5f1c8a[0x8] = "P:"
0x00589730 -> 0x5f1c82[0x8] = "Q:"
0x0058976e -> 0x5f1c9a[0x8] = "R:"
0x005897ac -> 0x5f1c92[0x8] = "S:"
0x005897ea -> 0x5f1c62[0x8] = "T:"
0x00589828 -> 0x5f1c5a[0x8] = "U:"
0x00589866 -> 0x5f1c72[0x8] = "V:"
0x005898a4 -> 0x5f1c6a[0x8] = "W:"
0x005898e2 -> 0x5f1c4a[0x8] = "X:"
0x00589920 -> 0x5f1c42[0x8] = "Y:"
0x0058995e -> 0x5f1c52[0x8] = "Z:"
0x0058999c -> 0x5f1d5a[0x8] = "/r"
0x005899da -> 0x5f1d52[0x8] = "/f"
0x00589a18 -> 0x5f1d62[0x8] = "/t"
0x00589a56 -> 0x5f1d82[0x8] = "00"
0x00589a94 -> 0x5f3dc7[0x10] = "vssadmin"
0x00589ad2 -> 0x5f2c0c[0xc] = "delete"
0x00589b10 -> 0x5f3d07[0x10] = "shadows"
0x00589b4e -> 0x5f2c48[0xc] = "/all"
0x00589b8c -> 0x5f2c54[0xc] = "/quiet"
0x00589bca -> 0x5f1cf2[0x8] = "cmd"
0x00589c08 -> 0x5f1d4a[0x8] = "/c"
0x00589c46 -> 0x5f2b70[0xc] = "ping"
0x00589c84 -> 0x5f3f47[0x10] = "127.0.0.1"
0x00589cc2 -> 0x5f1d32[0x8] = "&"
0x00589d00 -> 0x5f1cd2[0x8] = "del"
0x00589d3e -> 0x5f1d52[0x8] = "/f"
0x00589d7c -> 0x5f8804[0x1c] = "Program Files"
0x00589dba -> 0x5fbe36[0x28] = "Program Files (x86)"
0x00589df8 -> 0x5f6fc7[0x18] = "ProgramData"
0x00589e36 -> 0x5f1cfa[0x8] = "\"
0x00589e74 -> 0x5f1d2a[0x8] = "%v"
0x00589eb2 -> 0x5f1d22[0x8] = "
"
0x00589ef0 -> 0x5f3e87[0x10] = "@echo off"
0x00589f2e -> 0x5fef56[0x80] = "REG QUERY "HKLM\SYSTEM\CurrentControlSet\Control" /v
SystemStartOptions"
0x00589f6c -> 0x5ff32f[0xac] = "REG ADD
"HKLM\SYSTEM\CurrentControlSet\Control\SafeBoot\Minimal\VSS" /VE /T REG_SZ /F /D Service"
0x00589faa -> 0x5ff3db[0xac] = "REG ADD
"HKLM\SYSTEM\CurrentControlSet\Control\SafeBoot\Minimal\%s" /VE /T REG_SZ /F /D Service"
0x00589fe8 -> 0x5f3eb7[0x10] = "SAFEBOOT"
0x0058a026 -> 0x5f2c60[0xc] = "/set"
0x0058a064 -> 0x5f3da7[0x10] = "{current}"
0x0058a0a2 -> 0x5f3cf7[0x10] = "safeboot"
0x0058a0e0 -> 0x5f3d27[0x10] = "minimal"
0x0058a11e -> 0x5f2b7c[0xc] = "safe"
0x0058a15c -> 0x5f1d3a[0x8] = " "
0x0058a19a -> 0x5fe240[0x3c] = "SC QUERY | FINDSTR SERVICE_NAME"
0x0058a1d8 -> 0x5fee02[0x6c] = "([a-zA-Z0-9])*((?i)TEAM|SQL|EXCHANGE|BACKUP)([a-zA-Z0-9]*)"
0x0058a216 -> 0x5f3d57[0x10] = "net stop"
0x0058a254 -> 0x5f2c78[0xc] = ".bat"
0x0058a292 -> 0x5fed30[0x68] = "sc create %s binPath= "%s" DisplayName= "%s" start= auto"
0x0058a2d0 -> 0x5f1d72[0x8] = ":"
0x0058a30e -> 0x5f3f37[0x10] = "1073|1078"
0x0058a34c -> 0x5f2c30[0xc] = "%v %v"
0x0058a38a -> 0x5f87cc[0x1c] = "C:\Windows\%s"
0x0058a3c8 -> 0x5f3d67[0x10] = "override"
```

```
0x0058a406 -> 0x5f2b64[0xc] = "path"
0x0058a444 -> 0x5f1d6a[0x8] = ";"
0x0058a482 -> 0x5f2bf4[0xc] = "Users"
0x0058a4c0 -> 0x5f3e67[0x10] = "Default"
0x0058a4fe -> 0x5fcadb[0x2c] = "Documents and Settings"
0x0058a53c -> 0x5f6faf[0x18] = "Default User"
0x0058a588 -> 0x5f3d77[0x10] = "windows"
0x0058a5c3 -> 0x5f3cd7[0x10] = "perflogs"
0x0058a5fe -> 0x5f3ce7[0x10] = "recovery"
0x0058a639 -> 0x5f7057[0x18] = "$recycle.bin"
0x0058a674 -> 0x5fd225[0x30] = "system volume information"
0x0058a6d6 -> 0x5f3d77[0x10] = "windows"
0x0058a711 -> 0x5f3cd7[0x10] = "perflogs"
0x0058a74c -> 0x5f7057[0x18] = "$recycle.bin"
0x0058a787 -> 0x5fd225[0x30] = "system volume information"
0x0058a7c2 -> 0x5f6ff7[0x18] = "common files"
0x0058a7fd -> 0x5f3ec7[0x10] = "dvd maker"
0x0058a838 -> 0x5f9d40[0x20] = "internet explorer"
0x0058a873 -> 0x5f3d47[0x10] = "msbuild"
0x0058a8ae -> 0x5f8698[0x1c] = "microsoft games"
0x0058a8e9 -> 0x5f86b4[0x1c] = "mozilla firefox"
0x0058a924 -> 0x5fbd96[0x28] = "reference assemblies"
0x0058a95f -> 0x5f6f4f[0x18] = "tap-windows"
0x0058a99a -> 0x5f9d60[0x20] = "windows defender"
0x0058a9d5 -> 0x5f8708[0x1c] = "windows journal"
0x0058aa10 -> 0x5f6f1f[0x18] = "windows mail"
0x0058aa4b -> 0x5fbdbe[0x28] = "windows media player"
0x0058aa86 -> 0x5f6f37[0x18] = "windows nt"
0x0058aac1 -> 0x5fbde6[0x28] = "windows photo viewer"
0x0058aafc -> 0x5f86ec[0x1c] = "windows sidebar"
0x0058ab37 -> 0x5fcaaf[0x2c] = "windows portable devices"
0x0058ab72 -> 0x5f867c[0x1c] = "microsoft.net"
0x0058abad -> 0x5fd255[0x30] = "mozilla maintenance service"
0x0058abe8 -> 0x5fbe0e[0x28] = "uninstall information"
0x0058ac4a -> 0x5f3d37[0x10] = "microsoft"
0x0058ac85 -> 0x5f6f07[0x18] = "start menu"
0x0058acc0 -> 0x5f3db7[0x10] = "templates"
0x0058acfb -> 0x5f3ed7[0x10] = "favorites"
0x0058adc2 -> 0x5f1cc2[0x8] = "exe"
0x0058adfd -> 0x5f1cca[0x8] = "dll"
0x0058ae38 -> 0x5f1b72[0x8] = "sys"
0x0058ae73 -> 0x5f1b82[0x8] = "ini"
0x0058aeae -> 0x5f1cea[0x8] = "bat"
0x0058aee9 -> 0x5f1b7a[0x8] = "lnk"
```

APPENDIX 2 – IDA PYTHON SCRIPT FOR STRINGS DEOBFUSCATION

```
# Author: Ladislav Baco, LIFARS
# Date: July 22, 2020
#
# (c) 2020 LIFARS
# This code is licensed under MIT license (see LICENSE for details)

import idaapi
import idautils
import ida_name
import ida_bytes

from base64 import b64decode
from itertools import cycle, izip

def get_params(ea):
        data_addr = None
        length = None
        inst = idautils.DecodePreviousInstruction(ea)
        if (inst != None) and (inst.get_canon_mnem() == "mov"):
                length = inst.Op2.value
                inst2 = idautils.DecodePreviousInstruction(inst.ea)
                if (inst2 != None) and (inst2.get_canon_mnem() == "mov"):
                        inst3 = idautils.DecodePreviousInstruction(inst2.ea)
                        if (inst3 != None) and (inst3.get_canon_mnem() == "lea"):
                                length = inst.Op2.value
                                data_addr = inst3.Op2.addr

        return data_addr, length


def decrypt_string(data, key):
        crypt = b64decode(data)
        plain = "".join(chr(ord(c)^ord(k)) for c,k in izip(crypt, cycle(key)))
        return b64decode(plain)

arch = 64 if idaapi.get_inf_structure().is_64bit() else 32
ea = ida_name.get_name_ea(idaapi.BADADDR, "main.decodeString")
key_addr = ida_name.get_name_ea(idaapi.BADADDR, "main.encoderKey")
key = ida_bytes.get_bytes(ida_bytes.get_dword(key_addr), ida_bytes.get_dword(key_addr +
arch/8))


for xref in idautils.CodeRefsTo(ea, True):
      inst = idautils.DecodeInstruction(xref)
      if (inst != None) and (inst.get_canon_mnem() == "call"):
              data_addr, length = get_params(xref)
              data = ida_bytes.get_bytes(data_addr, length)
              decrypted_str = decrypt_string(data, key)
              print "0x{:08x} -> 0x{:x}[0x{:x}] = \"{}\"".format(xref, data_addr, length,
      decrypted_str)
              ida_bytes.set_cmt(xref, decrypted_str, False)
```