Revix Linux Ransomware

angle.ankura.com/post/102hcny/revix-linux-ransomware

Vishal Thakur

In the first half of 2021, we started to see the REvil ransomware operators pivot to targeting Linux-based systems with a new Linux version of their ransomware, similar to the malware they commonly used on Windows systems. Since then, there have been a few versions of this Linux-based malware.

In this post, we look at the latest version of their Linux-based ransomware "1.2a".

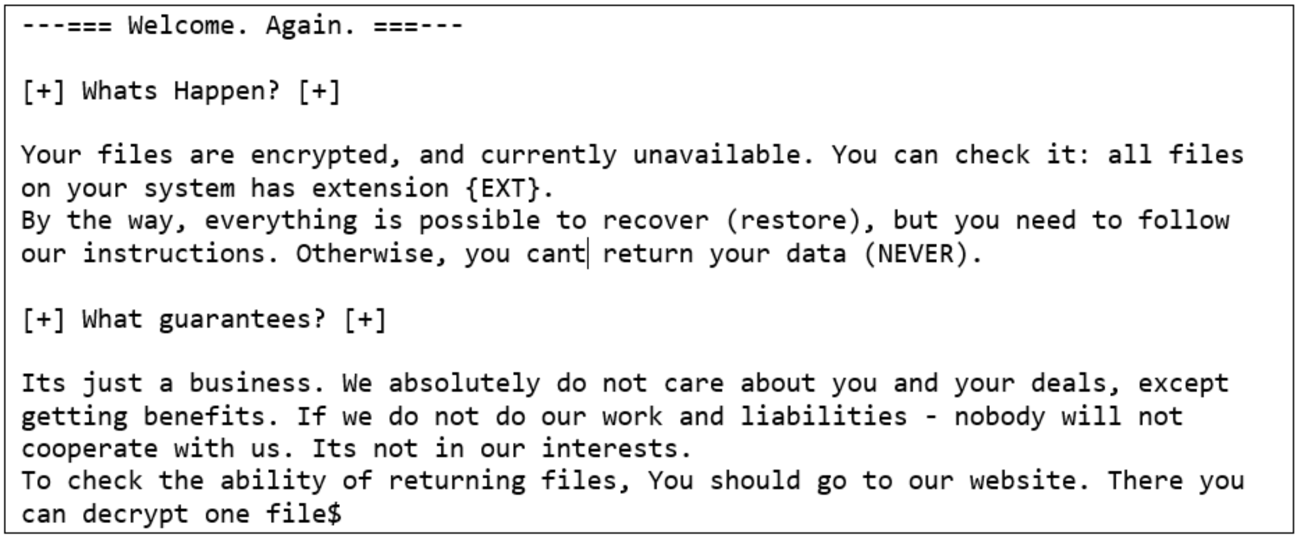
Quick Snapshot:

The malicious file is a Linux executable
Class: ELF64
Type: Dynamically Linked
Machine: X86-64
Number of section headers: 28
Entry Point: 0x401650
callq: __libc_start_main@plt
MD5: c83df66c46bcbc05cd987661882ff061
Yara Rules:

https://github.com/YaraExchange/yarasigs/blob/master/ransomware/crime_lin_revil.yar
https://github.com/YaraExchange/yarasigs/blob/master/ransomware/crime_lin_revix.yar

Introduction

The execution of this malware is straightforward. It traverses through the directories specified as targets and encrypts the files present in those directories. Once encryption is complete, it drops a ransom note in the directory with the usual ransom message and instructions on paying the threat actor to get the decryption key.



```
---=== Welcome. Again. ===---

[+] Whats Happen? [+]

Your files are encrypted, and currently unavailable. You can check it: all files
on your system has extension {EXT}.
By the way, everything is possible to recover (restore), but you need to follow
our instructions. Otherwise, you cant return your data (NEVER).

[+] What guarantees? [+]

Its just a business. We absolutely do not care about you and your deals, except
getting benefits. If we do not do our work and liabilities - nobody will not
cooperate with us. Its not in our interests.
To check the ability of returning files, You should go to our website. There you
can decrypt one file$
```

This variant of Revix requires a couple of parameters to be passed to execute successfully. It also requires escalated privileges to run and encrypt files on the disk successfully. Additionally, the malware checks the files in the target directories to see if they are already encrypted.

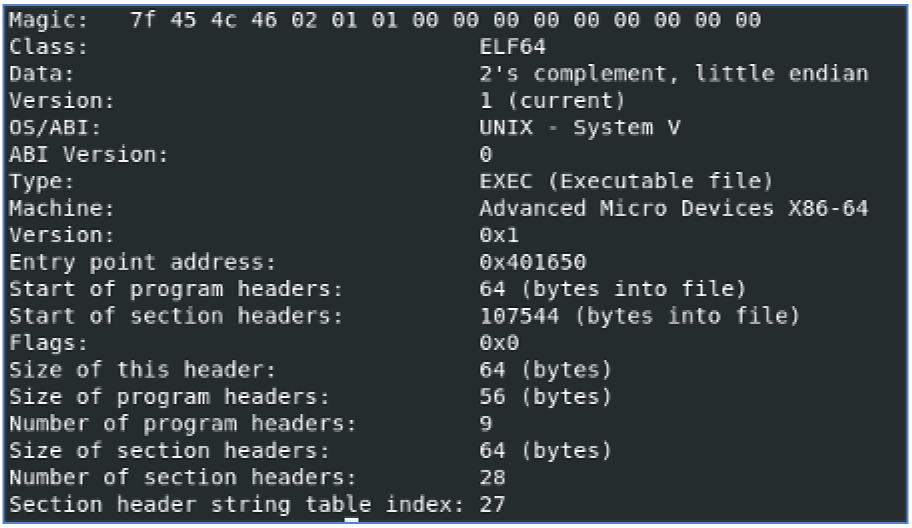
One of the main targets for this malware is VMware ESX platform's, which we've seen before in a different Linux ransomware from DarkSide.

Analysis

For this post, we analyzed Revix both statically and dynamically. Both methodologies have been used together throughout the analysis process presented below.

Let’s take a quick look at a couple of sections of this executable so that we have the offsets to some of the initial calls that can be used for further analysis.

Header:

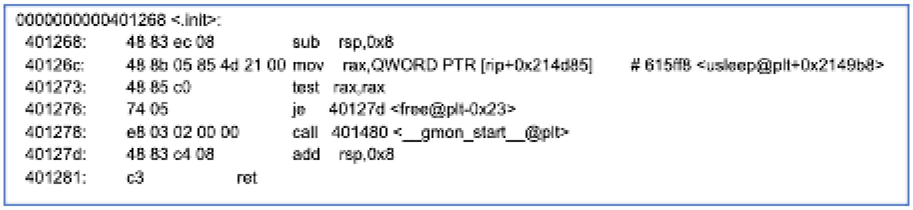


```
Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
Class:                             ELF64
Data:                              2's complement, little endian
Version:                           1 (current)
OS/ABI:                            UNIX - System V
ABI Version:                       0
Type:                              EXEC (Executable file)
Machine:                           Advanced Micro Devices X86-64
Version:                           0x1
Entry point address:               0x401650
Start of program headers:          64 (bytes into file)
Start of section headers:          107544 (bytes into file)
Flags:                             0x0
Size of this header:               64 (bytes)
Size of program headers:           56 (bytes)
Number of program headers:         9
Size of section headers:           64 (bytes)
Number of section headers:         28
Section header string table index: 27
```

Figure 1: Header Information

Section .init:

This section holds executable instructions that need to be executed before the main program entry point.

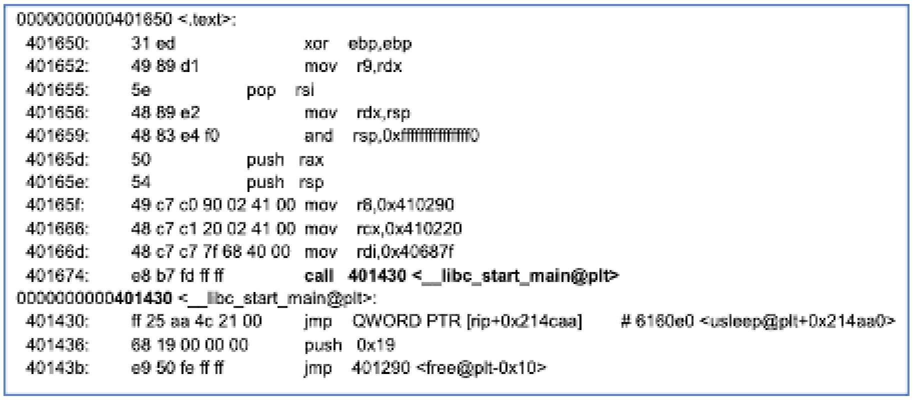


```
0000000000401268 <.init>:
  401268:       48 83 ec 08             sub    rsp,0x8
  40126c:       48 8b 05 85 4d 21 00 mov    rax,QWORD PTR [rip+0x214d85]        # 615ff8 <usleep@plt+0x2149b8>
  401273:       48 85 c0                test   rax,rax
  401276:       74 05                   je     40127d <free@plt-0x23>
  401278:       e8 03 02 00 00          call   401480 <__gmon_start__@plt>
  40127d:       48 83 c4 08             add    rsp,0x8
  401281:       c3              ret
```

Figure 2: section

Section .text:

This section contains executable code.



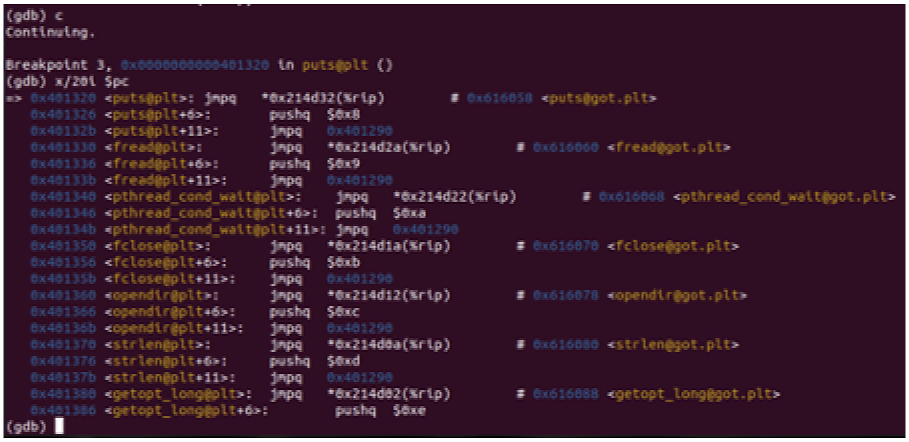
```
0000000000401650 <.text>:
  401650:       31 ed                   xor    ebp,ebp
  401652:       49 89 d1                mov    r9,rdx
  401655:       5e              pop    rsi
  401656:       48 89 e2                mov    rdx,rsp
  401659:       48 83 e4 f0             and    rsp,0xfffffffffffffff0
  40165d:       50              push   rax
  40165e:       54              push   rsp
  40165f:       49 c7 c0 90 02 41 00  mov    r8,0x410290
  401666:       48 c7 c1 20 02 41 00  mov    rcx,0x410220
  40166d:       48 c7 c7 7f 68 40 00  mov    rdi,0x40687f
  401674:       e8 b7 fd ff ff          call   401430 <__libc_start_main@plt>
0000000000401430 <__libc_start_main@plt>:
  401430:       ff 25 aa 4c 21 00       jmp    QWORD PTR [rip+0x214caa]        # 6160e0 <usleep@plt+0x214aa0>
  401436:       68 19 00 00 00          push   0x19
  40143b:       e9 50 fe ff ff          jmp    401290 <free@plt-0x10>
```

Figure 3: section

Functions

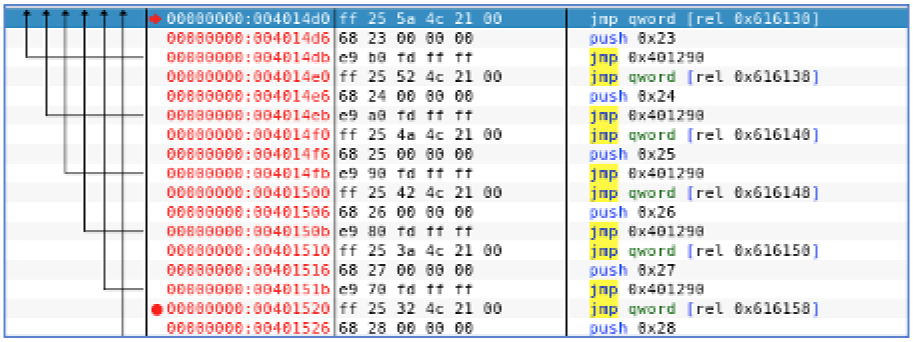
Revix loads several functions upon initialization. Following are some of the more interesting functions we can extract useful information from, to understand the flow of execution, along with developing threat detections that we've provided at the end of this post.

We execute the malware while attached to a debugger and break at the main function to view these functions presented below. Once we hit the main function, we follow the jump to 'puts' function to look at the CPU at that location. We can see all the loaded functions at this point.



```
(gdb) c
Continuing.

Breakpoint 3, 0x0000000000401320 in puts@plt ()
(gdb) x/20i $pc
=> 0x401320 <puts@plt>: jmpq   *0x214d32(%rip)        # 0x616058 <puts@got.plt>
   0x401326 <puts@plt+6>:       pushq  $0x8
   0x40132b <puts@plt+11>:      jmpq   0x401290
   0x401330 <fread@plt>:        jmpq   *0x214d2a(%rip)        # 0x616060 <fread@got.plt>
   0x401336 <fread@plt+6>:      pushq  $0x9
   0x40133b <fread@plt+11>:     jmpq   0x401290
   0x401340 <pthread_cond_wait@plt>:    jmpq   *0x214d22(%rip)        # 0x616068 <pthread_cond_wait@got.plt>
   0x401346 <pthread_cond_wait@plt+6>:  pushq  $0xa
   0x40134b <pthread_cond_wait@plt+11>: jmpq   0x401290
   0x401350 <fclose@plt>:       jmpq   *0x214d1a(%rip)        # 0x616070 <fclose@got.plt>
   0x401356 <fclose@plt+6>:     pushq  $0xb
   0x40135b <fclose@plt+11>:    jmpq   0x401290
   0x401360 <opendir@plt>:      jmpq   *0x214d12(%rip)        # 0x616078 <opendir@got.plt>
   0x401366 <opendir@plt+6>:    pushq  $0xc
   0x40136b <opendir@plt+11>:   jmpq   0x401290
   0x401370 <strlen@plt>:       jmpq   *0x214d0a(%rip)        # 0x616080 <strlen@got.plt>
   0x401376 <strlen@plt+6>:     pushq  $0xd
   0x40137b <strlen@plt+11>:    jmpq   0x401290
   0x401380 <getopt_long@plt>:  jmpq   *0x214d02(%rip)        # 0x616088 <getopt_long@got.plt>
   0x401386 <getopt_long@plt+6>:        pushq  $0xe
(gdb) 
```

Figure 4: Malware functions loaded upon initialization



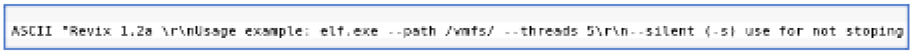
```
00000000:004014d0 ff 25 5a 4c 21 00    jmp qword [rel 0x616130]
00000000:004014d6 68 23 00 00 00       push 0x23
00000000:004014db e9 b0 fd ff ff       jmp 0x401290
00000000:004014e0 ff 25 52 4c 21 00    jmp qword [rel 0x616138]
00000000:004014e6 68 24 00 00 00       push 0x24
00000000:004014eb e9 a0 fd ff ff       jmp 0x401290
00000000:004014f0 ff 25 4a 4c 21 00    jmp qword [rel 0x616140]
00000000:004014f6 68 25 00 00 00       push 0x25
00000000:004014fb e9 90 fd ff ff       jmp 0x401290
00000000:00401500 ff 25 42 4c 21 00    jmp qword [rel 0x616148]
00000000:00401506 68 26 00 00 00       push 0x26
00000000:0040150b e9 80 fd ff ff       jmp 0x401290
00000000:00401510 ff 25 3a 4c 21 00    jmp qword [rel 0x616150]
00000000:00401516 68 27 00 00 00       push 0x27
00000000:0040151b e9 70 fd ff ff       jmp 0x401290
00000000:00401520 ff 25 32 4c 21 00    jmp qword [rel 0x616158]
00000000:00401526 68 28 00 00 00       push 0x28
```

Figure 5: Function sequence during execution

Initialization

Let's take a quick look at the program initialization:

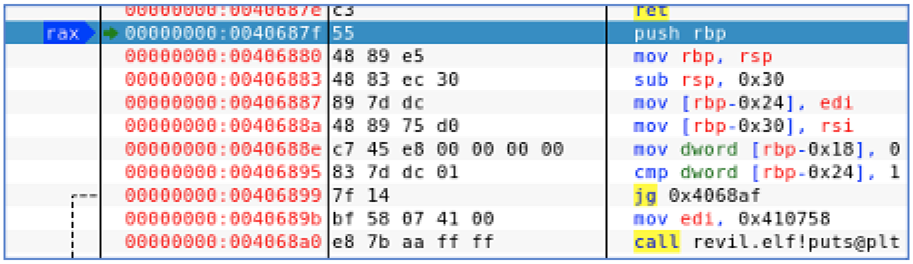
The malware requires to be run with a couple of command-line arguments. We can see these being passed through the stack in the image below/



```
ASCII "Revix 1.2a \r\nUsage example: elf.exe --path /vmfs/ --threads 5\r\n--silent (-s) use for not stoping
```

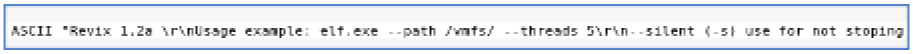
Figure 6: Parameters for the command-line arguments

The image below shows another view from the CPU that shows the program execution in flight.



```
                00000000:0040687e c3                             ret
      rax    →  00000000:0040687f 55                             push rbp
                00000000:00406880 48 89 e5                       mov rbp, rsp
                00000000:00406883 48 83 ec 30                    sub rsp, 0x30
                00000000:00406887 89 7d dc                       mov [rbp-0x24], edi
                00000000:0040688a 48 89 75 d0                    mov [rbp-0x30], rsi
                00000000:0040688e c7 45 e8 00 00 00 00           mov dword [rbp-0x18], 0
                00000000:00406895 83 7d dc 01                    cmp dword [rbp-0x24], 1
                00000000:00406899 7f 14                          jg 0x4068af
                00000000:0040689b bf 58 07 41 00                 mov edi, 0x410758
                00000000:004068a0 e8 7b aa ff ff                 call revil.elf!puts@plt
```

Figure 7: Program execution in flight



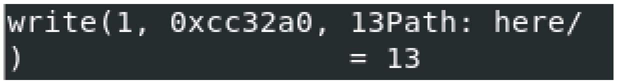
```
ASCII "Revix 1.2a \r\nUsage example: elf.exe --path /vmfs/ --threads 5\r\n--silent (-s) use for not stoping
```

Figure 8: Stack view

Execution

When executed as a non-privileged user, the malware is not able to achieve full execution.

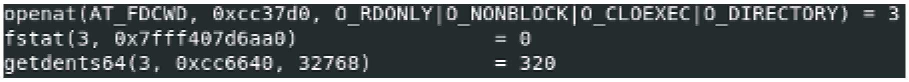
As shown in the image below, the malware has been provided with the directory 'here/' for this analysis.



```
write(1, 0xcc32a0, 13Path: here/
)                      = 13
```

Figure 9: Write execution on dir

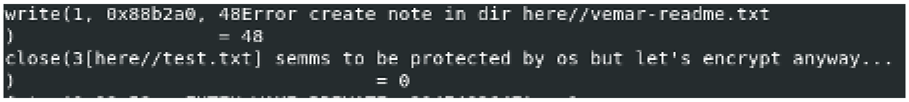
The malware tries to access the data in this directory for read/write and is unsuccessful, as shown below.



```
openat(AT_FDCWD, 0xcc37d0, O_RDONLY|O_NONBLOCK|O_CLOEXEC|O_DIRECTORY) = 3
fstat(3, 0x7fff407d6aa0)                = 0
getdents64(3, 0xcc6640, 32768)          = 320
```

Figure 10: getdent64 unsuccessful

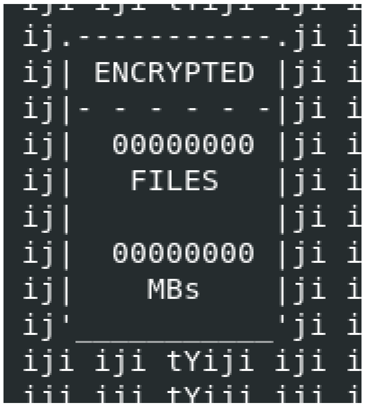
The malware also tries to encrypt a test file that we used in our analysis, but the encryption process fails as that action requires higher privileges.



```
write(1, 0x88b2a0, 48Error create note in dir here//vemar-readme.txt
)                    = 48
close(3[here//test.txt] semms to be protected by os but let's encrypt anyway...
)                                 = 0
```

Figure 11: Encryption unsuccessful

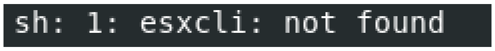
As a result, the execution fails to achieve the desired outcome for the malware, as shown below.



```
iji iji tYiji iji i
ij.-----------.ji i
ij| ENCRYPTED |ji i
ij|- - - - - -|ji i
ij|  00000000 |ji i
ij|   FILES   |ji i
ij|           |ji i
ij|  00000000 |ji i
ij|    MBs    |ji i
ij'___________'ji i
iji iji tYiji iji i
iji iji tYiji iji i
```

Figure 12: Encryption failed

Another point of interest from this failed execution is that the malware attempted to execute a esxcli command but this action fails as there is no esxcli on our test machine.

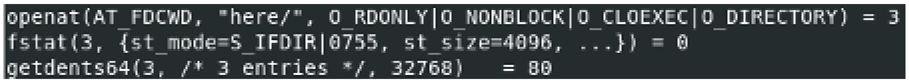


```
sh: 1: esxcli: not found
```

Figure 13: esxcli not found

When we execute Revix with elevated privileges, we start to see more successful activity from the malware.

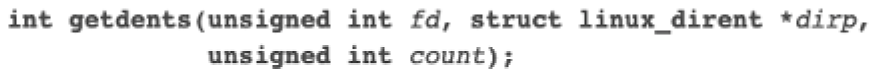
Firstly, Revix can access the data in the target directory.



```
openat(AT_FDCWD, "here/", O_RDONLY|O_NONBLOCK|O_CLOEXEC|O_DIRECTORY) = 3
fstat(3, {st_mode=S_IFDIR|0755, st_size=4096, ...}) = 0
getdents64(3, /* 3 entries */, 32768)   = 80
```

Figure 14: getdents64 successful

We can see in the image above, the system call ‘getdents’. This system call returns directory entries for the directory it’s run against.

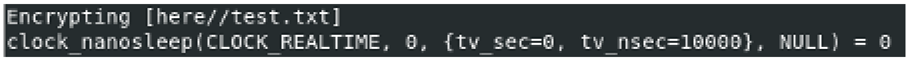


```
int getdents(unsigned int fd, struct linux_dirent *dirp,
             unsigned int count);
```

Figure 15: getdents64(2) Synopsis

In this case, there are three entries as we can see from the result shown in the image above.

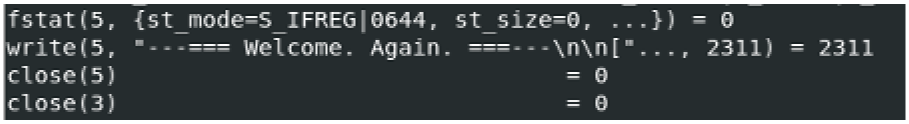
Next, we can see that Revix is able to perform read/write functions on the data in the target directories, resulting in successful encryption of files.



```
Encrypting [here//test.txt]
clock_nanosleep(CLOCK_REALTIME, 0, {tv_sec=0, tv_nsec=10000}, NULL) = 0
```

Figure 16: Encryption successful

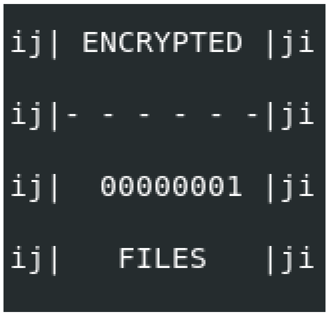
The Revix output below shows that it can write the ransom note text file to the victim's disk.



```
fstat(5, {st_mode=S_IFREG|0644, st_size=0, ...}) = 0
write(5, "---=== Welcome. Again. ===---\n\n["..., 2311) = 2311
close(5)                                = 0
close(3)                                = 0
```

Figure 17: Ransom note write successful

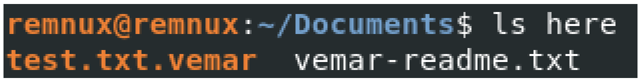
Finally, we can see that the execution is completed successfully, resulting in the data present in the target directory being encrypted:



```
ij| ENCRYPTED |ji

ij|- - - - - -|ji

ij|  00000001 |ji

ij|   FILES   |ji
```

Figure 18: Execution complete

The file we provided in the target directory is now encrypted, and a ransom note is created in the same directory:

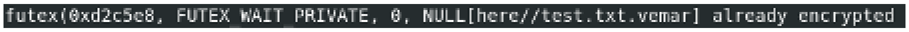


```
remnux@remnux:~/Documents$ ls here
test.txt.vemar  vemar-readme.txt
```

Figure 19: Execution complete, file encrypted

The malware also checks if the data in the target directory is already encrypted. To demonstrate this, we ran Revix against the same target directory one more time.

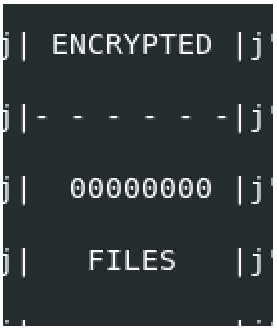
Upon execution, Revix runs a check on the data present in the target directory and identifies it to be already encrypted:



```
futex(0xd2c5e8, FUTEX_WAIT_PRIVATE, 0, NULL[here//test.txt.vemar] already encrypted
```

Figure 20: Encryption check performed

As a result, the execution ends up with no data being encrypted.



```
j| ENCRYPTED |j

j|- - - - - -|j

j|  00000000 |j

j|   FILES   |j
```

Figure 21: Execution complete

VMware ESX Targeting

Revix also tries to use esxcli, the command line interface for VMware’s ESX platform.

Let's take a quick look at the parameters passed to esxcli by Revix when it executes:

esxcli --formatter=csv --format-param=fields=="WorldID,DisplayName" vm process list | awk -F "\",\"*" '{system("esxcli vm process kill --type=force --world-id=" $1)}'*

vm process list

List the virtual machines on this system. This command currently will only list running VMs on the system.

vm process kill

Used to forcibly kill Virtual Machines that are stuck and not responding to normal stop operations.

--type

There are three types of VM kills that can be attempted: [soft, hard, force].

--world-id | -w

The World ID of the Virtual Machine to kill. This can be obtained from the 'vm process list' command (required)

Essentially, these ESX command-line arguments are shutting down all virtual machines running on the ESX platform.

Revix attempts to target the '/vmfs' directory and encrypt all the data present in that directory, so all the virtual machines are rendered inoperable until the data is decrypted. This targeting is similar to that seen in DarkSide's Linux variant.

Command-line Arguments

The malware requires the following parameters to be passed for its execution to begin:

elf.exe --path /vmfs/ --threads 5

It also allows the '--silent' option that executes the malware without stopping any VMs

--silent (-s) use for not stoping VMs mode *

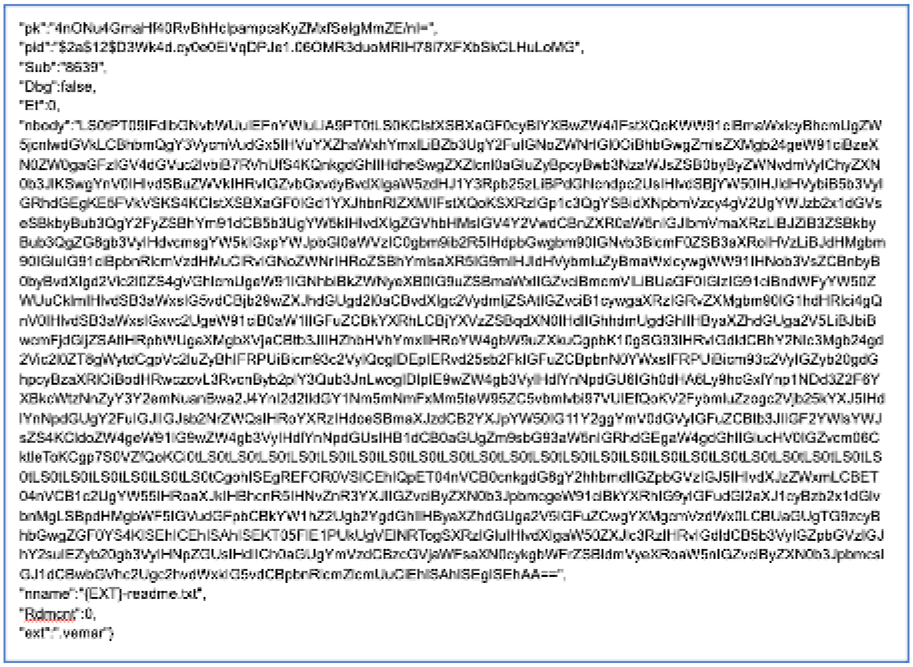
Parameter	Purpose
--path	Specifies the path of the data that needs to be encrypted
--threads	Specifies the number of threads, by default the malware uses 50 threads
--silent	Executes the malware without stopping the VMs running on ESX

Configuration

The configuration of Revix is similar to that of its Windows variant, only with fewer fields.

Field	Description
pk	Public Key
pid	ID
Sub	Tag
Dbg	Debug mode
nbody	Base64-encoded body of the ransom-note
nname	Filename of the ransom-note
rdmcnt	Readme Count
ext	File extension of the encrypted files

Here's an image showing the configuration we were able to extract from the sample we analysed:



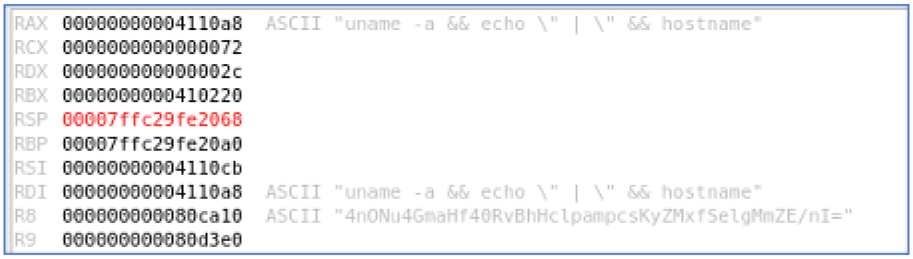
```
"pk":"4nONu4GmaHf40RvBhHclpampcsKyZMxfSelgMmZE/nI=",
"pid":"$2a$12$D3Wk4d.cy0e0EIVqDPJe1.06OMR3duoMRIH78I7XFXbSkCLHuLoMG",
"Sub":"8639",
"Dbg":false,
"Et":0,
"nbody":"LS0tPT09IFdlbGNvbWUuIEFnYWluLiA9PT0tLS0KClstXSBXaGF0cyBIYXBwZW4/IFstXQoKWW91ciBmaWxlcyBhcmUgZW
5jcnlwdGVkLCBhbmQgY3VycmVudGx5IHVuYXZhaWxhYmxlLiBZb3UgY2FuIGNoZWNrIGl0OiBhbGwgZmlsZXMgb24geW91ciBzeX
N0ZW0gaGFzIGV4dGVuc2lvbiB7RVhUfS4KQnkgdGhlIHdheSwgZXZlcnl0aGluZyBpcyBwb3NzaWJsZSB0byByZWNvdmVyIChyZXN
0b3JlKSwgYnV0IHlvdSBuZWVkIHRvIGZvbGxvdyBvdXIgaW5zdHJ1Y3Rpb25zLiBPdGhlcndpc2UsIHlvdSBjYW50IHJldHVybiB5b3Vy
IGRhdGEgKE5FVkVSKS4KClstXSBXaGF0IGd1YXJhbnRlZXMIFstXQoKSXRzIGp1c3QgYSBidXNpbmVzcy4gV2UgYWJzb2x1dGVs
eSBkbyBub3QgY2FyZSBhYm91dCB5b3UgYW5kIHlvdXIgZGVhbHMsIGV4Y2VwdCBnZXR0aW5nIGJlbmVmaXRzLiBJZiB3ZSBkby
Bub3QgZG8gb3VyIHdvcmsgYW5kIGxpYWJpbGl0aWVzIC0gbm9ib2R5IHdpbGwgbm90IGNvb3BlcmF0ZSB3aXRoIHVzLiBJdHMgbm
90IGluIG91ciBpbnRlcmVzdHMuCiRvIGNoZWNrIHRoZSBhYmlsaXR5IG9mIHJldHVybmluZyBmaWxlcywgWW91IHNob3VsZCBnbyB
0byBvdXIgd2Vic2l0ZS4gVGhlcmUgeW91IGNhbiBkZWNyeXB0IG9uZSBmaWxlIGZvciBmcmVlLiBUaGF0IGlzIG91ciBndWFyYW50Z
WUuCkImIHlvdSB3aWxsIG5vdCBjb29wZXJhdGUgd2l0aCBvdXIgc2VydmljZSAtIGZvciB1cywgaXRzIGRvZXMgbm90IG1hdHRlci4gQ
nV0IHlvdSB3aWxsIGxvc2UgeW91ciB0aW1lIGFuZCBkYXRhLCBjYXVzZSBqdXN0IHdlIGhhdmUgdGhlIHByaXZhdGUga2V5LiBJbiB
wcmFjdGljZSAtIHRpbWUgaXMgbXVjaCBtb3JlIHZhbHVhYmxlIHRoYW4gbW9uZXkuCgphK10gSG93IHRvIGdldCBhY2Nlc3Mgb24gd
2Vic2l0ZT8gWytdCgpVc2luZyBhIFRPUiBicm93c2VyIQoglDEpIERvd25sb2FkIGFuZCBpbnN0YWxsIFRPUiBicm93c2VyIGZyb20gdG
hpcyBzaXRlOiBodHRwczovL3RvcnByb2plY3Qub3JnLwoglDIpIE9wZW4gb3VyIHdlYnNpdGU6IGh0dHA6Ly9hcGxlYnp1NDd3Z2F6Y
XBkcWtzNnZyY3Y2emNuanBwa2J4YnI2d2lkGY1Nm5mNmFxMm5teW95ZC5vbmlvbi87VUlEfQoKV2FybmluZzogc2Vjb25kYXJ5IHd
lYnNpdGUgY2FuIGJlIGJsb2NrZWQsIHRoYXRzIHdoeSBmaXJzdCB2YXJpYW50IG11Y2ggYmV0dGVyIGFuZCBtb3JlIGF2YWlsYWJ
sZS4KCldoZW4geW91IG9wZW4gb3VyIHdlYnNpdGUsIHB1dCB0aGUgZm9sbG93aW5nIGRhdGEgaW4gdGhlIGlucHV0IGZvcm06C
kIleToKCgp7S0VZfQoKCi0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS
0tLS0tLS0tLS0tLS0tLS0tLS0tCgohISEgREFOR0VSICEhIQpET04nVCB0cnkgdG8gY2hhbmdlIGZpbGVzIGJ5IHlvdXJzZWxmLCBET
04nVCB1c2UgYW55IHRoaXJkIHBhcnR5IHNvZnR3YXJlIGZvciByZXN0b3JpbmcgeW91ciBkYXRhIG9yIGFudGl2aXJ1cyBzb2x1dGlv
bnMgLSBpdHMgbWF5IGVudGFpbCBkYW1hZ2Ugb2YgdGhlIHByaXZhdGUga2V5IGFuZCwgYXMgcmVzdWx0LCBUaGUgTG9zcyB
hbGwgZGF0YS4KISEhICEhISAhISEKT05FIE1PUkUgVElNRTogSXRzIGluIHlvdXIgaW50ZXJlc3RzIHRvIGdldCB5b3VyIGZpbGVzIGJ
hY2suIEZyb20gb3VyIHNpZGUsIHdlIChoaGUgYmVzdCBzcGVjaWFsaXN0cykgbWFrZSBldmVyeXRoaW5nIGZvciByZXN0b3JpbmcsI
GJ1dCBwbGVhc2Ugc2hvdWxkIG5vdCBpbnRlcmZlcmUuCiEhISAhISEgISEhAA==",
"nname":"{EXT}-readme.txt",
"Rdmcnt":0,
"ext":".vemar"}
```

Figure 22: Configuration

Profiling

Revix also gathers information about the victim machine by running the "uname" command:

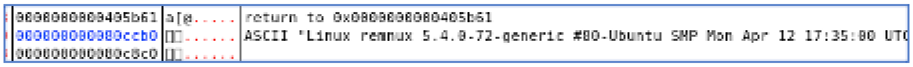
uname -a && echo " | " && hostname



```
RAX 00000000004110a8  ASCII "uname -a && echo \" | \" && hostname"
RCX 0000000000000072
RDX 000000000000002c
RBX 0000000000410220
RSP 00007ffc29fe2068
RBP 00007ffc29fe20a0
RSI 00000000004110cb
RDI 00000000004110a8  ASCII "uname -a && echo \" | \" && hostname"
R8  000000000080ca10  ASCII "4nONu4GmaHf40RvBhHclpampcsKyZMxfSelgMmZE/nI="
R9  000000000080d3e0
```

Figure 23: System profiling

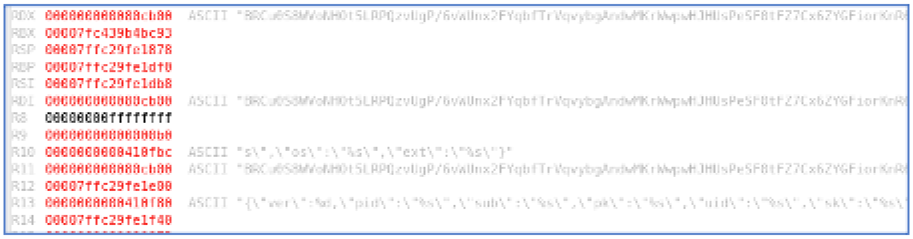
The results of the above command appear in the stack:



```
0000000000405b61 a[@.....  return to 0x0000000000405b61
000000000080ccb0 ........  ASCII "Linux remnux 5.4.0-72-generic #80-Ubuntu SMP Mon Apr 12 17:35:00 UTC
000000000080c8c0 ........
```

Figure 24: Stack view of system profiling in action

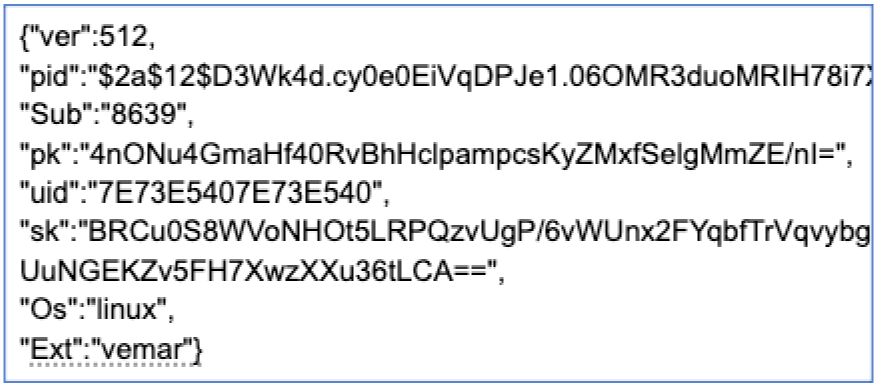
The info is then passed through the registers:



```
RDX 000000000000cb00  ASCII "BRCu0S8WVoNHOt5LRPQzvUgP/6vWUnx2FYqbfTrVqvybgAndwMKrWwpwHJHUsPeSF8tF27Cx6ZYGFiorKnR
RBX 00007fc439b4bc93
RSP 00007ffc29fe1878
RBP 00007ffc29fe1df0
RSI 00007ffc29fe1db8
RDI 000000000000cb00  ASCII "BRCu0S8WVoNHOt5LRPQzvUgP/6vWUnx2FYqbfTrVqvybgAndwMKrWwpwHJHUsPeSF8tF27Cx6ZYGFiorKnR
R8  00000000ffffffff
R9  00000000000000b0
R10 0000000000418fbc  ASCII "s\",\"os\":\"%s\",\"ext\":\"%s\"}"
R11 000000000000cb00  ASCII "BRCu0S8WVoNHOt5LRPQzvUgP/6vWUnx2FYqbfTrVqvybgAndwMKrWwpwHJHUsPeSF8tF27Cx6ZYGFiorKnR
R12 00007ffc29fe1e00
R13 0000000000418f88  ASCII "{\"ver\":%d,\"pid\":\"%s\",\"sub\":\"%s\",\"pk\":\"%s\",\"uid\":\"%s\",\"sk\":\"%s\
R14 00007ffc29fe1f40
```

Figure 25: Register view od system profiling in action

And the end-result is created in the form of this configuration with the victim information:

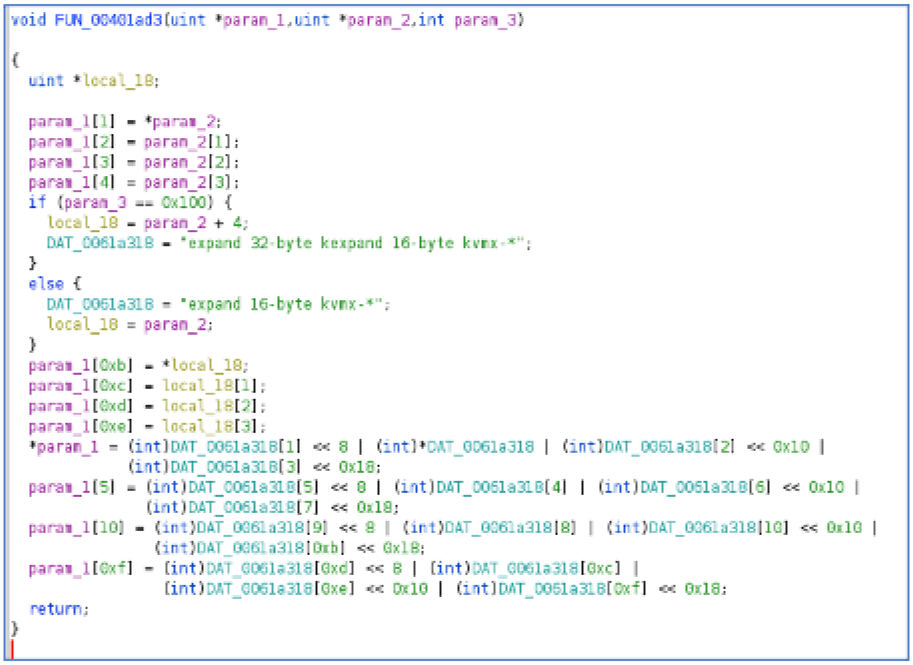


```
{"ver":512,
"pid":"$2a$12$D3Wk4d.cy0e0EiVqDPJe1.06OMR3duoMRIH78i7
"Sub":"8639",
"pk":"4nONu4GmaHf40RvBhHclpampcsKyZMxfSelgMmZE/nl=",
"uid":"7E73E5407E73E540",
"sk":"BRCu0S8WVoNHOt5LRPQzvUgP/6vWUnx2FYqbfTrVqvybg
UuNGEKZv5FH7XwzXXu36tLCA==",
"Os":"linux",
"Ext":"vemar"}
```

Figure 26: System profiling complete

Encryption

The malware uses Salsa20 encryption algorithm, just like its Windows variant, to encrypt the data. Here is the pseudocode for the function that implements this encryption:



```
void FUN_00401ad3(uint *param_1,uint *param_2,int param_3)

{
  uint *local_18;

  param_1[1] = *param_2;
  param_1[2] = param_2[1];
  param_1[3] = param_2[2];
  param_1[4] = param_2[3];
  if (param_3 == 0x100) {
    local_18 = param_2 + 4;
    DAT_0061a318 = "expand 32-byte kexpand 16-byte kvmx-*";
  }
  else {
    DAT_0061a318 = "expand 16-byte kvmx-*";
    local_18 = param_2;
  }
  param_1[0xb] = *local_18;
  param_1[0xc] = local_18[1];
  param_1[0xd] = local_18[2];
  param_1[0xe] = local_18[3];
  *param_1 = (int)DAT_0061a318[1] << 8 | (int)*DAT_0061a318 | (int)DAT_0061a318[2] << 0x10 |
             (int)DAT_0061a318[3] << 0x18;
  param_1[5] = (int)DAT_0061a318[5] << 8 | (int)DAT_0061a318[4] | (int)DAT_0061a318[6] << 0x10 |
               (int)DAT_0061a318[7] << 0x18;
  param_1[10] = (int)DAT_0061a318[9] << 8 | (int)DAT_0061a318[8] | (int)DAT_0061a318[10] << 0x10 |
                (int)DAT_0061a318[0xb] << 0x18;
  param_1[0xf] = (int)DAT_0061a318[0xd] << 8 | (int)DAT_0061a318[0xc] |
                 (int)DAT_0061a318[0xe] << 0x10 | (int)DAT_0061a318[0xf] << 0x18;
  return;
}
```

Figure 27: Pseudo-code for the encryption algorithm

Mitigation

Detections

Commands

Revix runs this command to determine machine info:

```
uname -a && echo " | " && hostname
```

Revix tries to query this directory:

```
/dev/urandom
```

Revix runs the below command to stop VMs running on the ESX platform in order to encrypt the data on those VMs:

```
esxcli --formatter=csv --format-param=fields=="WorldID,DisplayName" vm process list | awk -F "\"*,\"*" '{system("esxcli vm process kill --type=force --world-id=" $1)}'
```

Typos:

In some instances, typos that malware authors commit to the code are useful in detecting specific malware or similar code used in other malware families. Below are some of the typos we found in this variant of Revix:

--silent (-s) use for not ***stoping*** *VMs mode*

semms *to be protected by os but let's encrypt anyway…*

YARA Ruleset 1

```
rule Revix {

  meta:

description = "Detects REvil Linux - Revix 1.1 and 1.2"
    author = "Josh Lemon"
    reference = "https://angle.ankura.com/post/102hcny/revix-linux-ransomware"
    date = "2021-11-04"
    version = "1.0"
    hash1 =
"f864922f947a6bb7d894245b53795b54b9378c0f7633c521240488e86f60c2c5"
    hash2 = "559e9c0a2ef6898fabaf0a5fb10ac4a0f8d721edde4758351910200fe16b5fa7"
    hash3 =
"ea1872b2835128e3cb49a0bc27e4727ca33c4e6eba1e80422db19b505f965bc4"
  strings:
    $s1 = "Usage example: elf.exe --path /vmfs/ --threads 5" fullword ascii
    $s2 = "uname -a && echo \" | \" && hostname" fullword ascii
    $s3 = "esxcli --formatter=csv --format-param=fields==\"WorldID,DisplayName\" vm
process list" ascii
    $s4 = "awk -F \"\\\"*,\\\"*\" '{system(\"esxcli" ascii
    $s5 = "--silent (-s) use for not stoping VMs mode" fullword ascii
    $s6 = "!!!BY DEFAULT THIS SOFTWARE USES 50 THREADS!!!" fullword ascii
    $s7 = "%d:%d: Comment not allowed here" fullword ascii
    $s8 = "Error decoding user_id %d " fullword ascii
    $s9 = "Error read urandm line %d!" fullword ascii
    $s10 = "%d:%d: Unexpected `%c` in comment opening sequence" fullword ascii
    $s11 = "%d:%d: Unexpected EOF in block comment" fullword ascii
    $s12 = "Using silent mode, if you on esxi - stop VMs manualy" fullword ascii
    $s13 = "rand: try to read %hu but get %lu bytes" fullword ascii
    $s14 = "Revix" fullword ascii
    $s15 = "without --path encrypts current dir" fullword ascii      $e1 = "[%s] already
encrypted" fullword ascii
    $e2 = "File [%s] was encrypted" fullword ascii
    $e3 = "File [%s] was NOT encrypted" fullword ascii
    $e4 = "Encrypting [%s]" fullword ascii
  condition:
    uint16(0) == 0x457f and filesize
}
```

YARA Ruleset 2

```
/*
author = "Vishal Thakur - malienist.medium.com"
date = "2021-11-15"
version = "1"
```

```
description = "Detects Revix-1.2a and earlier versions of Revix"
info = "Generated from information extracted from the malware sample by manual
analysis."
*/

import "pe"

rule revixStatic {

  strings:

    $header = { 7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00 02 00 3e 00 01 00 00 00
50 16 40 00 00 00 00 00 }

    $config = { 7B 22 76 65 72 22 3A ?? ?? 2C 22 70 69 64 22 3A 22 ?? ?? 22 2C 22 73 75
62 22 3A 22 ?? ?? 22 2C 22 70 6B 22 3A 22 ?? ?? 22 2C 22 75 69 64 22 3A 22 ?? ?? 22
2C 22 73 6B 22 3A 22 ?? ?? 22 2C 22 6F 73 22 3A 22 ?? ?? 22 2C 22 65 78 74 22 3A 22
?? ?? 22 7D }

    $uname = { 75 6E 61 6D 65 20 2D 61 20 26 26 20 65 63 68 6F }

  condition:

    all of them and

    filesize

}

rule revixCode {

    strings:

    $err1 = { 45 72 72 6F 72 20 6F 70 65 6E 20 75 72 61 6E 64 6D }

    $err2 = { 45 72 72 6F 72 20 64 65 63 6F 64 69 6E 67 20 6D 61 73 74 65 72 5F 70 6B }

    $err3 = { 66 61 74 61 6C 20 65 72 72 6F 72 2C 6D 61 73 74 65 72 5F 70 6B 20 73 69
7A 65 20 69 73 20 62 61 64 }

    $err4 = { 45 72 72 6F 72 20 64 65 63 6F 64 69 6E 67 20 75 73 65 72 5F 69 64 }

    $err5 = { 45 72 72 6F 72 20 64 65 63 6F 64 69 6E 67 20 6E 6F 74 65 5F 62 6F 64 79 }

    $form1 = { 65 78 70 61 6E 64 20 33 32 2D 62 79 74 65 ?? ?? }

    $form2 = { 65 78 70 61 6E 64 20 31 36 2D 62 79 74 65 ?? ?? }

    $config = { 7B 22 76 65 72 22 3A ?? ?? 2C 22 70 69 64 22 3A 22 ?? ?? 22 2C 22 73 75
62 22 3A 22 ?? ?? 22 2C 22 70 6B 22 3A 22 ?? ?? 22 2C 22 75 69 64 22 3A 22 ?? ?? 22
2C 22 73 6B 22 3A 22 ?? ?? 22 2C 22 6F 73 22 3A 22 ?? ?? 22 2C 22 65 78 74 22 3A 22
?? ?? 22 7D }

  condition:
    all of them and

    filesize

}

rule revixESX {

  strings:

    $cmd1 = { 65 73 78 63 6C 69 }

    $cmd2 = { 2D 66 6F 72 6D 61 74 74 65 72 3D ?? ?? ?? }

    $cmd3 = { 2D 2D 66 6F 72 6D 61 74 2D 70 61 72 61 6D }

    $cmd4 = { 76 6D 20 70 72 6F 63 65 73 73 20 6C 69 73 74 }

    $cmd5 = { 65 73 78 63 6C 69 20 76 6D 20 70 72 6F 63 65 73 73 20 6B 69 6C 6C }

    $cmd6 = { 2D 2D 77 6F 72 6C 64 2D 69 64 3D 22 ?? ?? ?? }

    $config = { 7B 22 76 65 72 22 3A ?? ?? 2C 22 70 69 64 22 3A 22 ?? ?? 22 2C 22 73 75 62 22 3A 22 ?? ?? 22 2C 22 70 6B 22 3A 22 ?? ?? 22 2C 22 75 69 64 22 3A 22 ?? ?? 22 2C 22 73 6B 22 3A 22 ?? ?? 22 2C 22 6F 73 22 3A 22 ?? ?? 22 2C 22 65 78 74 22 3A 22 ?? ?? 22 7D }

  condition:

    all of them and

    filesize

}

rule revixPE {

  condition:

    pe.entry_point == 0x401650

}
```

Conclusion

As we can see in the analysis shown above, the execution of Revix is a bit clunky in this variant. It requires multiple conditions to be met before the ransomware is successful in encrypting data.

Revix needs to be executed as a command-line argument with elevated privileges, specified target directories, and the number of threads. Basically, it's not a standalone application at this time and is quite noisy as well.

If Revix is not run with silent mode enabled, it will try to stop any VMWare ESX virtual machines, triggering incident response processes from the victim. Revix could quite possibly fail to encrypt the virtual machines due to reduced/restricted access of where they reside on a Linux system.

As new variants for the Revix ransomware are released, we expect the execution to be more efficient, requiring fewer manual processes from the threat actor.

References

ESXi 7.0 U3 ESXCLI Command Reference

DarkSide on Linux: Virtual Machines Targeted - Naiim, M.,2021

getdents64(2) - Linux man page

Code Analysis details by Intezer Analyse

© Copyright 2021. The views expressed herein are those of the author(s) and not necessarily the views of Ankura Consulting Group, LLC., its management, its subsidiaries, its affiliates, or its other professionals. Ankura is not a law firm and cannot provide legal advice.