

ESXi Ransomware – A case study of Royal Ransomware

Prepared by: Vlad Pasca, Senior Malware & Threat Analyst



[SecurityScorecard.com](https://www.securityscorecard.com)
info@securityscorecard.com

Tower 49
12 E 49th Street
Suite 15-001
New York, NY 10017
[1.800.682.1707](tel:18006821707)

Table of contents

Table of contents	1
Executive summary	2
Analysis and findings	2
Indicators of Compromise	19

Executive summary

Royal ransomware joins other ransomware groups targeting ESXi servers. The malware powers off all virtual machines using the `esxcli` tool and doesn't encrypt a list of files that are embedded in the code. As in the case of the [Windows version](#), a parameter called `"-id"` consisting of 32 characters must be specified in the command line.

The files are encrypted using the AES algorithm, with the key and IV being encrypted using the RSA public key that is hard-coded in the executable. The process can partially encrypt a file depending on its size and the value of the `"-ep"` parameter. The extension of the encrypted files is changed to `".royal_u"`.

Analysis and findings

SHA256: 06abc46d5dbd012b170c97d142c6b679183159197e9d3f6a76ba5e5abf999725

The ransomware retrieves the command line arguments and compares them with `"-id"`, `"-ep"`, `"-stopvm"`, `"-vmonly"`, `"-fork"`, and `"-logs"`:

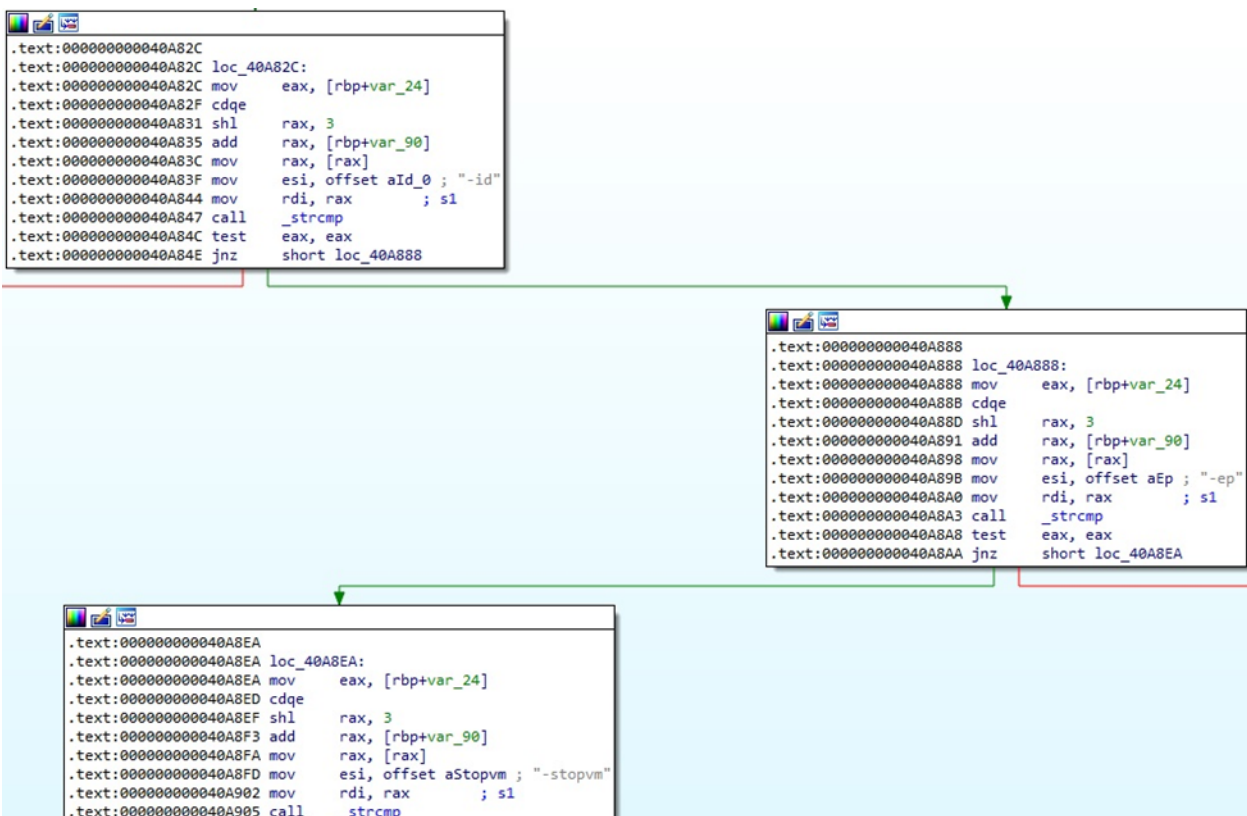


Figure 1

The `"-id"` parameter consisting of 32 characters is mandatory; otherwise, the following message

is displayed:

```
.text:000000000040A9B6 mov     edi, offset s ; "-id: id must be 32 characters"
.text:000000000040A9BB call    _puts
.text:000000000040A9C0 mov     eax, 0
.text:000000000040A9C5 jmp     loc_40AB72
```

Figure 2

The “-ep” parameter represents the encryption percentage of the files. It is converted from string to integer using the atoi function, as shown in figure 3.

```
.text:000000000040A8AC add     [rbp+var_24], 1
.text:000000000040A8B0 mov     eax, [rbp+var_24]
.text:000000000040A8B3 cdq
.text:000000000040A8B5 shl     rax, 3
.text:000000000040A8B9 add     rax, [rbp+var_90]
.text:000000000040A8C0 mov     rax, [rax]
.text:000000000040A8C3 mov     rdi, rax ; nptr
.text:000000000040A8C6 call    _atoi
.text:000000000040A8CB mov     dword ptr [rbp+var_34], eax
.text:000000000040A8CE cmp     dword ptr [rbp+var_34], 0
.text:000000000040A8D2 jle     short loc_40A8DE

.text:000000000040A8D4 cmp     dword ptr [rbp+var_34], 100
.text:000000000040A8D8 jle     loc_40A98C

.text:000000000040A8DE
.text:000000000040A8DE loc_40A8DE:
.text:000000000040A8DE mov     dword ptr [rbp+var_34], 50
.text:000000000040A8E5 jmp     loc_40A98C
```

Figure 3

When running with the “-stopvm” parameter, the process calls a function named stop_vm. It creates a child process via a call to fork (see figure 4).

```

.text:00000000040A199 _ZL7stop_vmv proc near
.text:00000000040A199
.text:00000000040A199 var_5D0= byte ptr -5D0h
.text:00000000040A199 dest= byte ptr -1D0h
.text:00000000040A199 s= byte ptr -0D0h
.text:00000000040A199 size= qword ptr -0A0h
.text:00000000040A199 var_38= dword ptr -38h
.text:00000000040A199 fd= dword ptr -34h
.text:00000000040A199 ptr= qword ptr -30h
.text:00000000040A199 haystack= qword ptr -28h
.text:00000000040A199 var_20= qword ptr -20h
.text:00000000040A199 var_14= dword ptr -14h
.text:00000000040A199
.text:00000000040A199 ; __unwind { // __gxx_personality_v0
.text:00000000040A199 push rbp
.text:00000000040A19A mov rbp, rsp
.text:00000000040A19D push rbx
.text:00000000040A19E sub rsp, 5C8h
.text:00000000040A1A5 call _fork
.text:00000000040A1AA mov [rbp+var_38], eax
.text:00000000040A1AD cmp [rbp+var_38], 0
.text:00000000040A1B1 jnz short loc_40A1E1

```

Figure 4

The child process obtains a list of running virtual machines, which are identified by World ID and Display Name. It saves it in a file called "list":

```

.text:00000000040A1B3 mov r8d, 0
.text:00000000040A1B9 mov ecx, offset aEsxcliVmProces ; "esxcli vm process list > list"
.text:00000000040A1BE mov edx, offset aC ; "-c"
.text:00000000040A1C3 mov esi, offset arg ; "/bin/sh"
.text:00000000040A1C8 mov edi, offset arg ; "/bin/sh"
.text:00000000040A1CD mov eax, 0
.text:00000000040A1D2 call _execlp
.text:00000000040A1D7 mov edi, 0 ; status
.text:00000000040A1DC call _exit

```

Figure 5

The parent process opens the "list" file and gets the file status using the stat method, as displayed below.

```

.text:00000000040A1E1
.text:00000000040A1E1 loc_40A1E1: ; stat_loc
.text:00000000040A1E1 mov edi, 0
.text:00000000040A1E6 call _wait
.text:00000000040A1EB mov esi, 0 ; oflag
.text:00000000040A1F0 mov edi, offset file ; "list"
.text:00000000040A1F5 mov eax, 0
.text:00000000040A1FA call _open
.text:00000000040A1FF mov [rbp+fd], eax
.text:00000000040A202 cmp [rbp+fd], 0FFFFFFFFh
.text:00000000040A206 jz loc_40A404

.text:00000000040A20C lea rax, [rbp+s]
.text:00000000040A213 mov edx, 90h ; n
.text:00000000040A218 mov esi, 0 ; c
.text:00000000040A21D mov rdi, rax ; s
.text:00000000040A220 call _memset
.text:00000000040A225 lea rax, [rbp+s]
.text:00000000040A22C mov rsi, rax ; stat_buf
.text:00000000040A22F mov edi, offset file ; "list"
.text:00000000040A234 call stat

```

Figure 6

The above file's content is read using a function called read_all, which is a wrapper for the read method:

```
.text:0000000040A27D
.text:0000000040A27D loc_40A27D:
.text:0000000040A27D mov     rax, [rbp+size]
.text:0000000040A284 mov     rdx, rax      ; unsigned __int64
.text:0000000040A287 mov     rcx, [rbp+ptr]
.text:0000000040A28B mov     eax, [rbp+fd]
.text:0000000040A28E mov     rsi, rcx      ; unsigned __int8 *
.text:0000000040A291 mov     edi, eax      ; int
.text:0000000040A293 call    _Z8read_alliPhm ; read_all(int,uchar *,ulong)
.text:0000000040A298 xor     eax, 1
.text:0000000040A29B test    al, al
.text:0000000040A29D jz     short loc_40A2BA
```

Figure 7

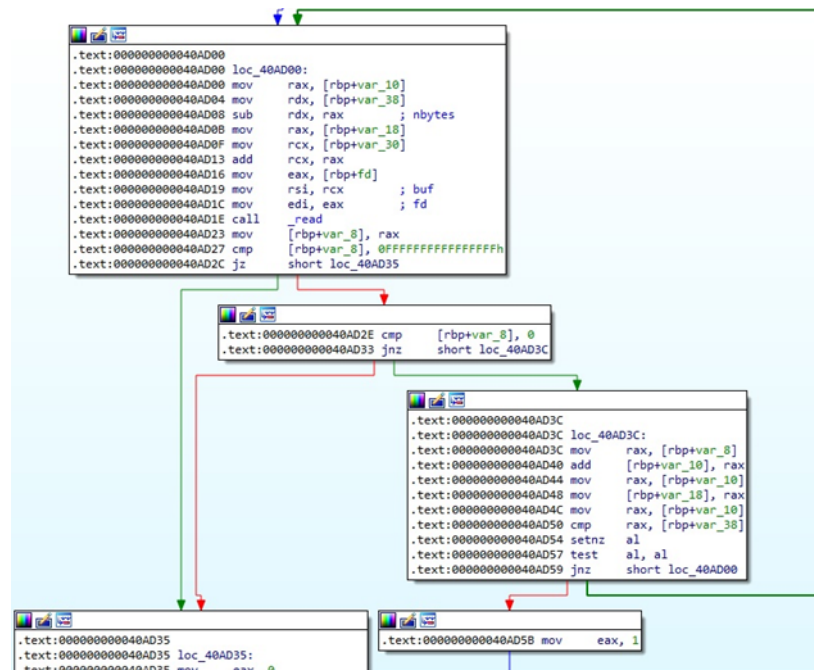


Figure 8

All virtual machines are powered off based on the World ID by spawning a new process:

```
.text:0000000040A3D1
.text:0000000040A3D1 loc_40A3D1:
.text:0000000040A3D1 mov     rax, [rbp+haystack]
.text:0000000040A3D5 mov     esi, offset aWorldId ; "World ID: "
.text:0000000040A3DA mov     rdi, rax      ; haystack
.text:0000000040A3DD call    _strstr
.text:0000000040A3E2 mov     [rbp+haystack], rax
.text:0000000040A3E6 cmp     [rbp+haystack], 0
.text:0000000040A3EB setnz  al, al
.text:0000000040A3EE test    al, al
.text:0000000040A3F0 jnz     loc_40A2EB
```

Figure 9

```

.text:0000000040A366 lea rdx, [rbp+dest]
.text:0000000040A36D lea rax, [rbp+var_5D0]
.text:0000000040A374 mov esi, offset format ; "esxcli vm process kill --type=hard --wo"...
.text:0000000040A379 mov rdi, rax ; s
.text:0000000040A37C mov eax, 0
.text:0000000040A381 call _sprintf
.text:0000000040A386 call _fork
.text:0000000040A38B mov [rbp+var_38], eax
.text:0000000040A38E cmp [rbp+var_38], 0
.text:0000000040A392 jnz short loc_40A3C7

```

```

300000040A3C7
300000040A3C7 loc_40A3C7: ; stat_loc
300000040A3C7 mov edi, 0
300000040A3CC call _wait

```

```

.text:0000000040A394 lea rax, [rbp+var_5D0]
.text:0000000040A39B mov r8d, 0
.text:0000000040A3A1 mov rcx, rax
.text:0000000040A3A4 mov edx, offset aC ; "-c"
.text:0000000040A3A9 mov esi, offset arg ; "/bin/sh"
.text:0000000040A3AE mov edi, offset arg ; "/bin/sh"
.text:0000000040A3B3 mov eax, 0
.text:0000000040A3B8 call _execlp
.text:0000000040A3BD mov edi, 0 ; status
.text:0000000040A3C2 call _exit

```

Figure 10

If the malware is running with the “-vmonly” parameter, then no files are encrypted, and the execution flow isn’t impacted.

Whether the “-logs” parameter is specified, multiple logs are displayed in the standard output:

```

.text:0000000040A963
.text:0000000040A963 loc_40A963:
.text:0000000040A963 mov eax, [rbp+var_24]
.text:0000000040A966 cdqe
.text:0000000040A968 shl rax, 3
.text:0000000040A96C add rax, [rbp+var_90]
.text:0000000040A973 mov rax, [rax]
.text:0000000040A976 mov esi, offset aLogs ; "-logs"
.text:0000000040A97B mov rdi, rax ; this
.text:0000000040A97E call _strcmp
.text:0000000040A983 test eax, eax
.text:0000000040A985 jnz short loc_40A98C

```

```

[rbp+var_25], 1
.text:0000000040A987 call _ZN4logs4initEv ; logs::init(void)

```

Figure 11

```

.text:0000000040C62E ; __int64 __fastcall logs::init(logs * __hidden this)
.text:0000000040C62E public _ZN4logs4initEv
.text:0000000040C62E _ZN4logs4initEv proc near
.text:0000000040C62E ; __unwind { // __gxx_personality_v0
.text:0000000040C62E push rbp
.text:0000000040C62F mov rbp, rsp
.text:0000000040C632 mov rax, cs:stdout@@GLIBC_2_2_5
.text:0000000040C639 mov cs:_ZL3log, rax ; log
.text:0000000040C640 leave
.text:0000000040C641 retn
.text:0000000040C641 ; } // starts at 40C62E
.text:0000000040C641 _ZN4logs4initEv endp
.text:0000000040C641

```

Figure 12

In the case of running with the “-fork” parameter, the executable creates a child process and performs a function call to setsid:

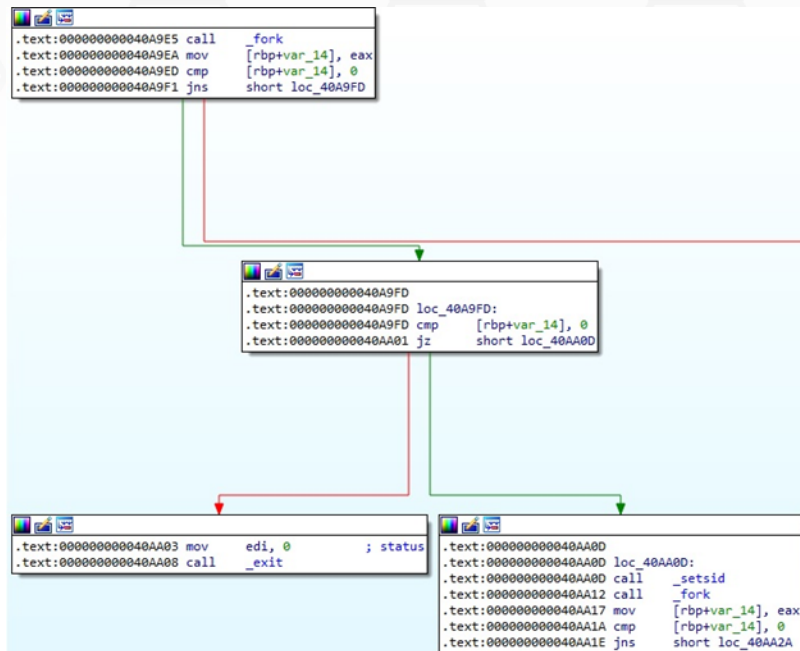


Figure 13

The first parameter should be a directory that will be encrypted. The [Windows version](#) of the ransomware uses the “-path” parameter in order to encrypt a target directory. The malware calls a function called search_files with the targeted path as the first parameter (figure 14).

```
.text:0000000040AA3A
.text:0000000040AA3A loc_40AA3A:
.text:0000000040AA3A mov     eax, dword ptr [rbp+var_34]
.text:0000000040AA3D mov     edi, eax           ; this_
.text:0000000040AA3F call    _ZN10threadpool6createEi ; threadpool::create(int)
.text:0000000040AA44 lea    rax, [rbp+var_41]
.text:0000000040AA48 mov     rdi, rax
.text:0000000040AA4B call    __ZN5a1cE1Ev ; std::allocator<char>::allocator(void)
.text:0000000040AA50 lea    rdx, [rbp+var_41]
.text:0000000040AA54 mov     rcx, [rbp+var_34+4]
.text:0000000040AA58 lea    rax, [rbp+var_50]
.text:0000000040AA5C mov     rsi, rcx
.text:0000000040AA5F mov     rdi, rax
.text:0000000040AA62 ; try {
.text:0000000040AA62 call    __ZN5sC1EPKcRK5a1cE ; std::string::string(char const*,std::allocator<char> const&)
.text:0000000040AA62 ; } // starts at 40AA62
.text:0000000040AA67 lea    rax, [rbp+var_50]
.text:0000000040AA6B mov     esi, 0
.text:0000000040AA70 mov     rdi, rax
.text:0000000040AA73 ; try {
.text:0000000040AA73 call    _Z12search_filesSsb ; search_files(std::string,bool)
.text:0000000040AA73 ; } // starts at 40AA73
.text:0000000040AA78 jmp     short loc_40AA93
```

Figure 14

The process retrieves the number of processors using the sysconf method:

```

.text:0000000040B8D4 ; __int64 __fastcall threadpool::create(threadpool * __hidden this, int)
.text:0000000040B8D4 public_ZN10threadpool6createEi
.text:0000000040B8D4 _ZN10threadpool6createEi proc near
.text:0000000040B8D4
.text:0000000040B8D4 var_14= dword ptr -14h
.text:0000000040B8D4 var_10= qword ptr -10h
.text:0000000040B8D4 var_8= qword ptr -8
.text:0000000040B8D4
.text:0000000040B8D4 ; __unwind { // __gxx_personality_v0
.text:0000000040B8D4 push rbp
.text:0000000040B8D5 mov rbp, rsp
.text:0000000040B8D8 sub rsp, 20h
.text:0000000040B8DC mov [rbp+var_14], edi
.text:0000000040B8DF mov eax, [rbp+var_14]
.text:0000000040B8E2 mov cs:g_ep, eax
.text:0000000040B8E8 mov edi, _SC_NPROCESSORS_ONLN ; name
.text:0000000040B8ED call _sysconf
.text:0000000040B8F2 mov cs:num_threads, rax
.text:0000000040B8F9 mov rax, cs:num_threads
.text:0000000040B900 mov [rbp+var_10], rax
.text:0000000040B904 cmp [rbp+var_10], 0
.text:0000000040B909 jle loc_40B98F

```

Figure 15

Royal ransomware creates 8 * number of processors threads by calling the pthread_create function (see figure 16).

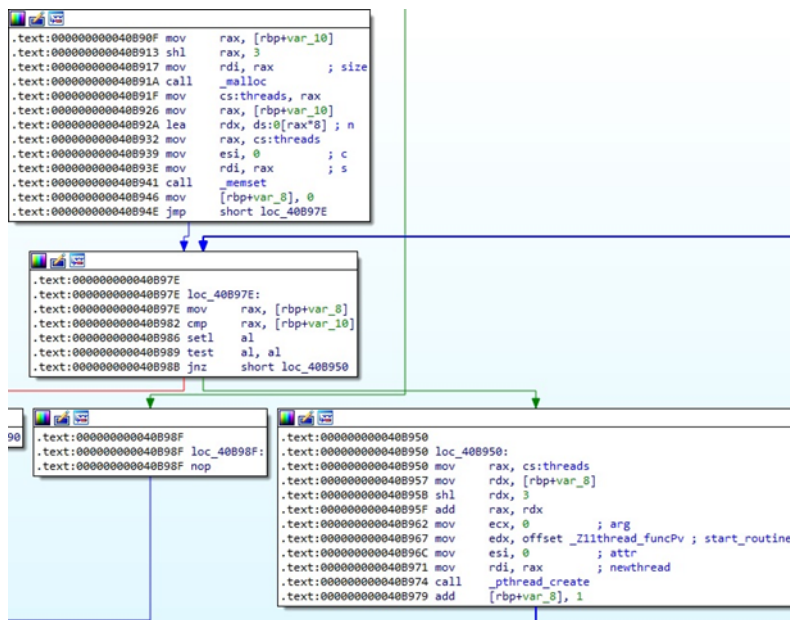


Figure 16

The opendir function is utilized to open the target directory:

```

.text:00000000040A40F ; search_files(std::string, bool)
.text:00000000040A40F public _Z12search_filesSsb
.text:00000000040A40F _Z12search_filesSsb proc near
.text:00000000040A40F
.text:00000000040A40F var_8C= byte ptr -8Ch
.text:00000000040A40F var_88= qword ptr -88h
.text:00000000040A40F var_80= byte ptr -80h
.text:00000000040A40F var_70= byte ptr -70h
.text:00000000040A40F var_60= byte ptr -60h
.text:00000000040A40F var_50= byte ptr -50h
.text:00000000040A40F var_40= byte ptr -40h
.text:00000000040A40F var_30= byte ptr -30h
.text:00000000040A40F dirp= qword ptr -20h
.text:00000000040A40F var_18= qword ptr -18h
.text:00000000040A40F
.text:00000000040A40F ; __unwind { // __gxx_personality_v0
.text:00000000040A40F push rbp
.text:00000000040A410 mov rbp, rsp
.text:00000000040A413 push r12
.text:00000000040A415 push rbx
.text:00000000040A416 add rsp, 0FFFFFFFFFFFFFF80h
.text:00000000040A41A mov [rbp+var_88], rdi
.text:00000000040A421 mov eax, esi
.text:00000000040A423 mov [rbp+var_8C], al
.text:00000000040A429 mov [rbp+dirp], 0
.text:00000000040A431 mov [rbp+var_18], 0
.text:00000000040A439 mov rax, [rbp+var_88]
.text:00000000040A440 mov rdi, rax ; this
.text:00000000040A443 call __ZNKS5c_strEv ; std::string::c_str(void)
.text:00000000040A448 mov rdi, rax ; name
.text:00000000040A44B call _opendir
.text:00000000040A450 mov [rbp+dirp], rax

```

Figure 17

A ransom note called “readme” is created in the traversed directory. The “-id” parameter is also included in the text:

```

.text:00000000040A0E4 ; drop_ransomnote(std::string)
.text:00000000040A0E4 _Z15drop_ransomnoteSs proc near
.text:00000000040A0E4
.text:00000000040A0E4 var_28= qword ptr -28h
.text:00000000040A0E4 var_20= byte ptr -20h
.text:00000000040A0E4 stream= qword ptr -18h
.text:00000000040A0E4
.text:00000000040A0E4 ; __unwind { // __gxx_personality_v0
.text:00000000040A0E4 push rbp
.text:00000000040A0E5 mov rbp, rsp
.text:00000000040A0E8 push r12
.text:00000000040A0EA push rbx
.text:00000000040A0EB sub rsp, 20h
.text:00000000040A0EF mov [rbp+var_28], rdi
.text:00000000040A0F3 lea rax, [rbp+var_20]
.text:00000000040A0F7 mov rcx, [rbp+var_28]
.text:00000000040A0FB mov edx, offset aReadme ; "/readme"
.text:00000000040A100 mov rsi, rcx
.text:00000000040A103 mov rdi, rax ; this
.text:00000000040A106 call _ZStpllcStl1cchar_traitsIcESaIcEESbIT_0_T1_ERKS6_PKS3_ ; std::operator+(char, std::char_traits<char>, std::allocator<char>)(std:
.text:00000000040A108 lea rax, [rbp+var_20]
.text:00000000040A10F mov rdi, rax ; this
.text:00000000040A112 ; try {
.text:00000000040A112 call __ZNKS5c_strEv ; std::string::c_str(void)
.text:00000000040A117 mov esi, offset modes ; "wt"
.text:00000000040A11C mov rdi, rax ; filename
.text:00000000040A11F call _fopen
.text:00000000040A124 mov [rbp+stream], rax
.text:00000000040A128 cmp [rbp+stream], 0
.text:00000000040A12D jz short loc_40A184

```



```

.text:00000000040A12F mov edi, offset g_id ; this
.text:00000000040A134 call __ZNKS5c_strEv ; std::string::c_str(void)
.text:00000000040A139 mov rdx, rax
.text:00000000040A13C mov rcx, cs:g_ransom_note
.text:00000000040A143 mov rax, [rbp+stream]
.text:00000000040A147 mov rsi, rcx ; format
.text:00000000040A14A mov rdi, rax ; stream
.text:00000000040A14D mov eax, 0
.text:00000000040A152 call _fprintf

```

Figure 18

The malware reads the directory by calling the readdir method, as shown below:

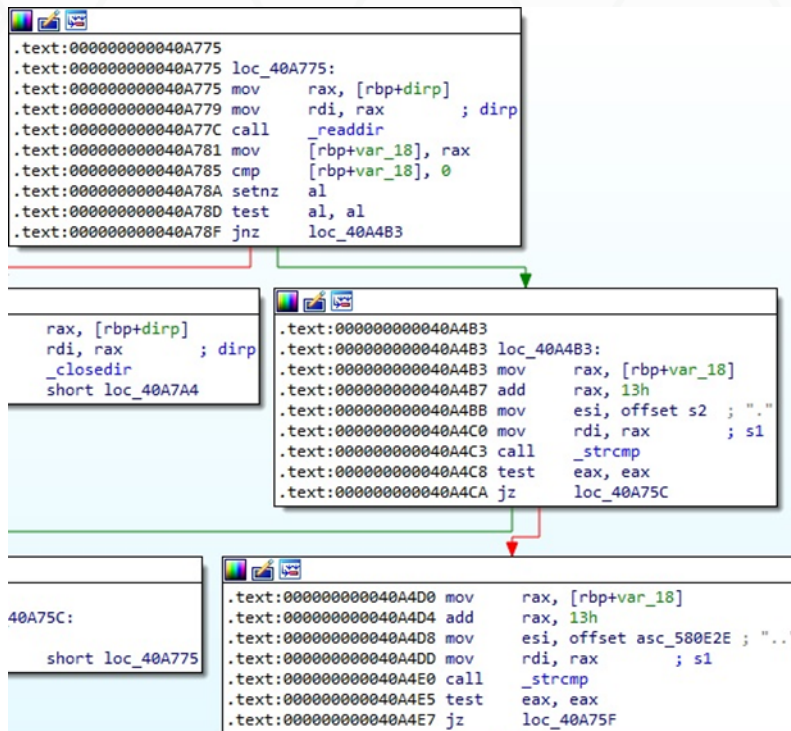


Figure 19

The file type is compared with 0x4 (**DT_DIR**) and 0x8 (**DT_REG**):

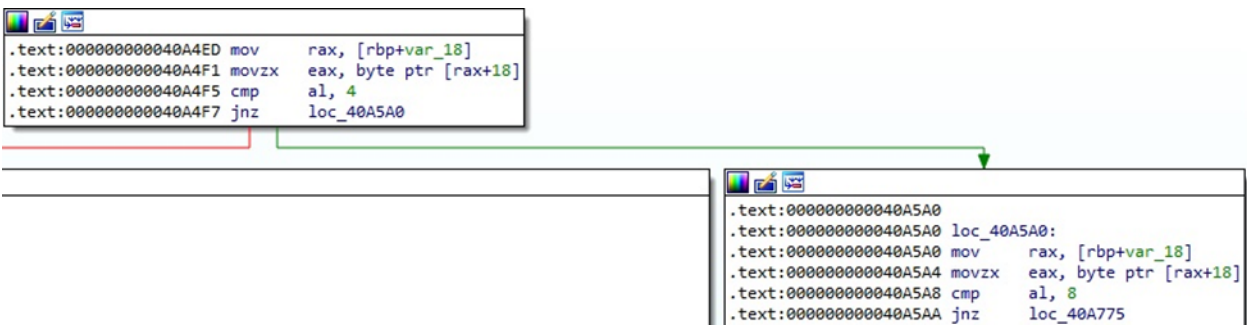


Figure 20

In the case of directories, the search_files function is called recursively. For regular files, the ransomware avoids files containing the following strings: ".royal_u", ".royal_w", ".sf", ".v00", ".b00", "royal_log_", and "readme" (see figure 21).

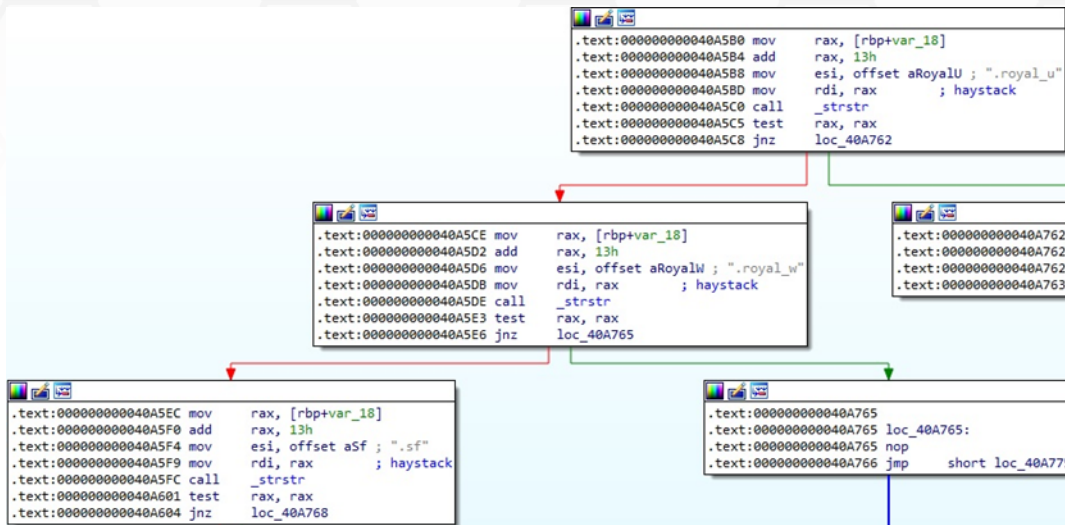


Figure 21

The malicious process imports a hard-coded RSA public key:

```

    .text:00000000040B641 ; void *thread_func(void *)
    .text:00000000040B641 public _Z11thread_funcPv
    .text:00000000040B641 _Z11thread_funcPv proc near
    .text:00000000040B641
    .text:00000000040B641 var_68= qword ptr -68h
    .text:00000000040B641 var_58= qword ptr -58h
    .text:00000000040B641 var_50= byte ptr -50h
    .text:00000000040B641 var_40= byte ptr -40h
    .text:00000000040B641 var_30= byte ptr -30h
    .text:00000000040B641 var_28= qword ptr -28h
    .text:00000000040B641 var_20= qword ptr -20h
    .text:00000000040B641 fd= dword ptr -18h
    .text:00000000040B641 var_11= byte ptr -11h
    .text:00000000040B641
    .text:00000000040B641 ; __unwind { // __gxx_personality_v0
    .text:00000000040B641 push    rbp
    .text:00000000040B642 mov     rbp, rsp
    .text:00000000040B645 push    r12
    .text:00000000040B647 push    rbx
    .text:00000000040B648 sub     rsp, 60h
    .text:00000000040B64C mov     [rbp+var_68], rdi
    .text:00000000040B650 mov     edi, offset _ZL8keybytes ; "-----BEGIN RSA PUBLIC KEY-----\nMIICCAK"...
    .text:00000000040B655 call   _ZL9ImportKeyPKc ; ImportKey(char const*)
  
```

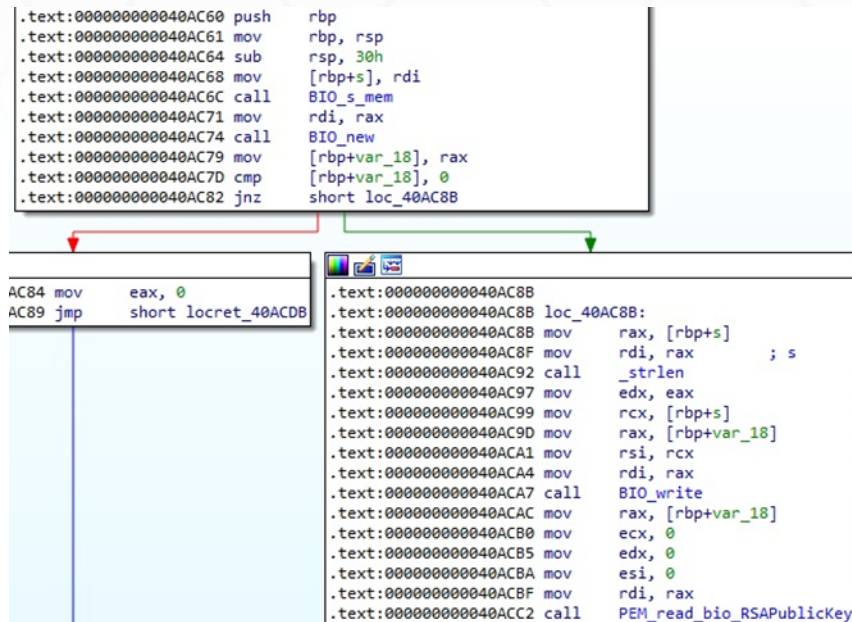
Figure 22

```

    db '-----BEGIN RSA PUBLIC KEY-----',0Ah
    ; DATA XREF: thread_func(void *)+Ff0
    ; threadpool::test_encrypt(void):loc_40BBE7f0
    db 'MIICCAKCAgEAp/24TNvKoZ9rzWMAh9kVgq4x1j+L/tgWH5ncB1TQA6eT5NDtgsQH',0Ah
    db 'jv+6N3IY8P4SPSnG5QU8p9uYm3berObDuLURZ4wGw+HEKY+jNht5JD4aE+SS2Gj1',0Ah
    db '+lht2N+S8lRDAjcyXJZaCePN4pHDWQ65cVHnonyo5FfjKkQpD1zbAZ8/wBY+5gE4',0Ah
    db 'Tex2Fdh7pvs7ek8+cnzkSi19xC0p1j4zoMZBwFQST9iLK7KbRTKnaF1ZAHnDKaTQ',0Ah
    db 'uKcJkcdhpQnaDyuUoJb2k+gD3n+k/on33I19hf04s67gyiIBH03q63CYBJ0XFewJ',0Ah
    db 'cvvahe+nZ3D0ffV/7LN6F0588R8lI2ZH+pMsyUwobI3Tdjkd0HvMgJITrqrCK7BZ',0Ah
    db 'TIKcZ0Rub+RQJ5NowXbC+CbgD138nESpKimPztcd6rZy32Jo7IcvAqPSckRuaghB',0Ah
    db 'rkci/d377b6IT+vOWpNciS87dUQ0lU0mtsI2LLSkwyxauG5Y1W/MDUYZEuhHY1ZM',0Ah
    db 'cKq1SLmu80TitL6bYOEQSy31PtCg2B0t1Su0NzW4pEXvg2hQyuSEbewEGkrJrjTK',0Ah
    db 'v9K7eu+eT5/arOy/onM56ffZSxfVseuK48R9TWktGpPMkszLmwy14rp1ds6S700',0Ah
    db '/HLRayEWjwa0eR0r/GhEHX80C8IU54ksEuf3uHbpbq8jFnN1A+U239q0CAQM=',0Ah
    db '-----END RSA PUBLIC KEY-----',0Ah
  
```

Figure 23

The RSA public key is read by calling the PEM_read_bio_RSAPublicKey function (figure 24).



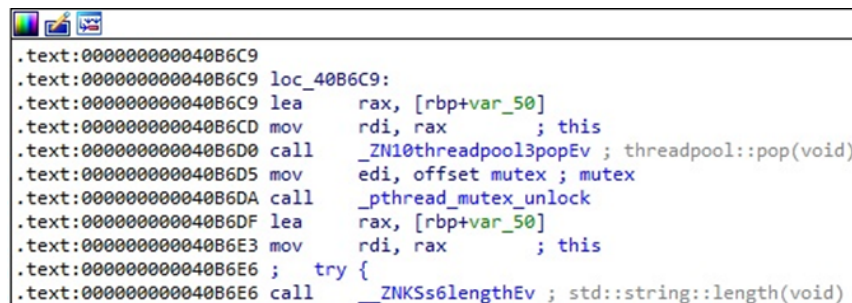
```
.text:00000000040AC60 push rbp
.text:00000000040AC61 mov rbp, rsp
.text:00000000040AC64 sub rsp, 30h
.text:00000000040AC68 mov [rbp+s], rdi
.text:00000000040AC6C call BIO_s_mem
.text:00000000040AC71 mov rdi, rax
.text:00000000040AC74 call BIO_new
.text:00000000040AC79 mov [rbp+var_18], rax
.text:00000000040AC7D cmp [rbp+var_18], 0
.text:00000000040AC82 jnz short loc_40AC8B

AC84 mov eax, 0
AC89 jmp short locret_40ACDB

.text:00000000040AC8B loc_40AC8B:
.text:00000000040AC8B mov rax, [rbp+s]
.text:00000000040AC8F mov rdi, rax ; s
.text:00000000040AC92 call _strlen
.text:00000000040AC97 mov edx, eax
.text:00000000040AC99 mov rcx, [rbp+s]
.text:00000000040AC9D mov rax, [rbp+var_18]
.text:00000000040ACA1 mov rsi, rcx
.text:00000000040ACA4 mov rdi, rax
.text:00000000040ACA7 call BIO_write
.text:00000000040ACAC mov rax, [rbp+var_18]
.text:00000000040ACB0 mov ecx, 0
.text:00000000040ACB5 mov edx, 0
.text:00000000040ACBA mov esi, 0
.text:00000000040ACBF mov rdi, rax
.text:00000000040ACC2 call PEM_read_bio_RSAPublicKey
```

Figure 24

Each of the created threads receives a file to be encrypted as a parameter:



```
.text:00000000040B6C9
.text:00000000040B6C9 loc_40B6C9:
.text:00000000040B6C9 lea rax, [rbp+var_50]
.text:00000000040B6CD mov rdi, rax ; this
.text:00000000040B6D0 call _ZN10threadpool3popEv ; threadpool::pop(void)
.text:00000000040B6D5 mov edi, offset mutex ; mutex
.text:00000000040B6DA call _pthread_mutex_unlock
.text:00000000040B6DF lea rax, [rbp+var_50]
.text:00000000040B6E3 mov rdi, rax ; this
.text:00000000040B6E6 ; try {
.text:00000000040B6E6 call __ZNKSs6lengthEv ; std::string::length(void)
```

Figure 25

The ransomware calls a function named prepare_file for all files to be encrypted, as highlighted in figure 26.

```

.text:00000000040B6FF loc_40B6FF:
.text:00000000040B6FF mov     [rbp+var_58], 0
.text:00000000040B707 lea    rdx, [rbp+var_50]
.text:00000000040B70B lea    rax, [rbp+var_40]
.text:00000000040B70F mov     rsi, rdx ; std::string *
.text:00000000040B712 mov     rdi, rax ; this
.text:00000000040B715 call   __ZN5sC1ERKs ; std::string::string(std::string const&)
.text:00000000040B715 ; } // starts at 40B6E6
.text:00000000040B71A lea    rdx, [rbp+var_58]
.text:00000000040B71E lea    rax, [rbp+var_40]
.text:00000000040B722 mov     rsi, rdx
.text:00000000040B725 mov     rdi, rax
.text:00000000040B728 ; try {
.text:00000000040B728 call   _ZL12prepare_fileSsP1 ; prepare_file(std::string,long *)

```

Figure 26

A file is opened for reading and writing via a function call to open (0x2 = **O_RDWR**):

```

.text:00000000040AE28 lea    rdx, [rbp+s]
.text:00000000040AE2F mov     rsi, rdx ; stat_buf
.text:00000000040AE32 mov     rdi, rax ; filename
.text:00000000040AE35 call   stat
.text:00000000040AE3A mov     rax, [rbp+var_60]
.text:00000000040AE3E test   rax, rax
.text:00000000040AE41 jnz    short loc_40AE4A

:43 mov     eax, 0FFFFFFFh
:48 jmp     short locret_40AE79

.text:00000000040AE4A loc_40AE4A:
.text:00000000040AE4A mov     rdx, [rbp+var_60]
.text:00000000040AE4E mov     rax, [rbp+var_A0]
.text:00000000040AE55 mov     [rax], rdx
.text:00000000040AE58 mov     rax, [rbp+var_98]
.text:00000000040AE5F mov     rdi, rax ; this
.text:00000000040AE62 call   __ZNKs5c_strEv ; std::string::c_str(void)
.text:00000000040AE67 mov     esi, 2 ; oflag
.text:00000000040AE6C mov     rdi, rax ; file
.text:00000000040AE6F mov     eax, 0
.text:00000000040AE74 call   _open

```

Figure 27

If the “-logs” parameter is specified, the process outputs a message containing the file to be encrypted:

```

.text:00000000040B764 lea    rax, [rbp+var_50]
.text:00000000040B768 mov     rdi, rax ; this
.text:00000000040B76B ; try {
.text:00000000040B76B call   __ZNKs5c_strEv ; std::string::c_str(void)
.text:00000000040B770 mov     rsi, rax ; char *
.text:00000000040B773 mov     edi, offset aEncryptingS ; "Encrypting %s"
.text:00000000040B778 mov     eax, 0
.text:00000000040B77D call   _ZN4logs5printEPKcz ; logs::print(char const*,...)
.text:00000000040B782 mov     eax, cs:g_ep
.text:00000000040B788 movsxd rcx, eax
.text:00000000040B78B mov     rdx, [rbp+var_58]
.text:00000000040B78F mov     rsi, [rbp+var_20]
.text:00000000040B793 mov     rbx, [rbp+var_28]
.text:00000000040B797 mov     eax, [rbp+fd]
.text:00000000040B79A mov     r8, rsi
.text:00000000040B79D mov     rsi, rbx
.text:00000000040B7A0 mov     edi, eax
.text:00000000040B7A2 call   _ZL7encryptiP6rsa_stllPh ; encrypt(int,rsa_st *,long,long,uchar *)

```

Figure 28

The logging function implementation is shown in figure 29. It also displays the current date and time obtained using the `current_date_time` method.

```

.text:00000000040C709 lea     rax, [rbp+var_D0]
.text:00000000040C710 mov     rdi, rax
.text:00000000040C713 call    _ZL17current_date_timev ; current_date_time(void)
.text:00000000040C718 lea     rax, [rbp+var_D0]
.text:00000000040C71F mov     rdi, rax ; this
.text:00000000040C722 ; try {
.text:00000000040C722 call    __ZNK5Sc_strEv ; std::string::c_str(void)
.text:00000000040C727 mov     rdx, rax
.text:00000000040C72A mov     rax, cs:_ZL3log ; log
.text:00000000040C731 mov     esi, offset aS_6 ; "[%s] "
.text:00000000040C736 mov     rdi, rax ; stream
.text:00000000040C739 mov     eax, 0
.text:00000000040C73E call    _fprintf
.text:00000000040C73E ; } // starts at 40C722
.text:00000000040C743 jmp     short loc_40C767

```

↓

```

.text:00000000040C767
.text:00000000040C767 loc_40C767:
.text:00000000040C767 lea     rax, [rbp+var_D0]
.text:00000000040C76E mov     rdi, rax ; void *
.text:00000000040C771 call    __ZN5Sd1Ev ; std::string::~string()
.text:00000000040C776 mov     rax, cs:_ZL3log ; log
.text:00000000040C77D lea     rdx, [rbp+arg] ; arg
.text:00000000040C784 mov     rcx, [rbp+format]
.text:00000000040C788 mov     rsi, rcx ; format
.text:00000000040C78E mov     rdi, rax ; s
.text:00000000040C791 call    _vfprintf
.text:00000000040C796 mov     rax, cs:_ZL3log ; log
.text:00000000040C79D mov     rsi, rax ; stream
.text:00000000040C7A0 mov     edi, 0Ah ; c
.text:00000000040C7A5 call    _fputc

```

Figure 29

The malware generates 32 random bytes representing the AES key and 16 random bytes representing the IV:

```

.text:00000000040B1C0 mov     [rbp+fd], edi
.text:00000000040B1C6 mov     [rbp+var_3A0], rsi
.text:00000000040B1CD mov     [rbp+var_3A8], rdx
.text:00000000040B1D4 mov     qword ptr [rbp+var_3B0], rcx
.text:00000000040B1D8 mov     [rbp+var_3B8], r8
.text:00000000040B1E2 lea     rax, [rbp+src]
.text:00000000040B1E9 mov     esi, 32 ; unsigned __int64
.text:00000000040B1EE mov     rdi, rax ; unsigned __int8 *
.text:00000000040B1F1 call    _ZL10gen_randomPhm ; gen_random(uchar *,ulong)
.text:00000000040B1F6 xor     eax, 1
.text:00000000040B1F9 test    al, al
.text:00000000040B1FB jz     short loc_40B207

```

↓

```

.text:00000000040B207
.text:00000000040B207 loc_40B207:
.text:00000000040B207 lea     rax, [rbp+s]
.text:00000000040B208 mov     esi, 16 ; unsigned __int64
.text:00000000040B210 mov     rdi, rax ; unsigned __int8 *
.text:00000000040B213 call    _ZL10gen_randomPhm ; gen_random(uchar *,ulong)
.text:00000000040B218 xor     eax, 1
.text:00000000040B21B test    al, al
.text:00000000040B21D jz     short loc_40B229

```

Figure 30

The randomly generated bytes are encrypted using the RSA public key (see figure 31).

```

.text:0000000040B229
.text:0000000040B229 loc_40B229:
.text:0000000040B229 lea rcx, [rbp+src]
.text:0000000040B230 lea rax, [rbp+dest]
.text:0000000040B237 mov edx, 20h ; ' ' ; n
.text:0000000040B23C mov rsi, rcx ; src
.text:0000000040B23F mov rdi, rax ; dest
.text:0000000040B242 call _memcpy
.text:0000000040B247 lea rax, [rbp+s]
.text:0000000040B24B lea rdx, [rbp+dest]
.text:0000000040B252 lea rcx, [rdx+20h]
.text:0000000040B256 mov edx, 10h ; n
.text:0000000040B25B mov rsi, rax ; src
.text:0000000040B25E mov rdi, rcx ; dest
.text:0000000040B261 call _memcpy
.text:0000000040B266 mov rcx, [rbp+var_3A0]
.text:0000000040B26D lea rdx, [rbp+dest]
.text:0000000040B274 lea rax, [rbp+dest]
.text:0000000040B27B mov r8d, 4
.text:0000000040B281 mov rsi, rax
.text:0000000040B284 mov edi, 48
.text:0000000040B289 call RSA_public_encrypt
.text:0000000040B28E mov [rbp+var_34], eax
.text:0000000040B291 cmp [rbp+var_34], 512
.text:0000000040B298 jz short loc_40B2FB

```

Figure 31

The malicious binary rounds up the file size to a multiple of 16, which is required by the AES algorithm:

```

.text:0000000040B2FB
.text:0000000040B2FB loc_40B2FB:
.text:0000000040B2FB mov rax, [rbp+var_3A8]
.text:0000000040B302 mov [rbp+var_58], rax
.text:0000000040B306 mov rax, [rbp+var_3A8]
.text:0000000040B30D mov esi, 16
.text:0000000040B312 mov rdi, rax
.text:0000000040B315 call _Z9chROUNDUPliET_S0_T0_ ; chROUNDUP<long,int>(long,int)
.text:0000000040B31A mov [rbp+var_3A8], rax
.text:0000000040B321 mov rdx, [rbp+var_58]
.text:0000000040B325 mov eax, [rbp+fd]
.text:0000000040B32B mov rsi, rdx ; _int64
.text:0000000040B32E mov edi, eax ; int
.text:0000000040B330 call _Z11resize_fileil ; resize_file(int,long)
.text:0000000040B335 xor eax, 1
.text:0000000040B338 test al, al
.text:0000000040B33A jz short loc_40B346

```

Figure 32

The entire file content is encrypted if the file length is less than or equal to 5,245,000 bytes or if the “-ep” parameter equals 100. As we’ve already described in our whitepaper about the [Windows version](#), the ransomware can modify the encryption percentage and perform intermittent encryption:

```

.text:0000000040B346
.text:0000000040B346 loc_40B346:
.text:0000000040B346 mov     [rbp+var_30], 0
.text:0000000040B34D mov     [rbp+var_60], 0
.text:0000000040B355 mov     [rbp+offset], 0
.text:0000000040B35D cmp     [rbp+var_34B], 5245000
.text:0000000040B368 jle     short loc_40B377

.text:0000000040B36A mov     rax, qword ptr [rbp+var_380]
.text:0000000040B371 cmp     rax, 100
.text:0000000040B375 jnz     short loc_40B396

.text:0000000040B377 loc_40B377:
.text:0000000040B377 mov     [rbp+var_30], 1
.text:0000000040B37E mov     rax, [rbp+var_34B]
.text:0000000040B385 mov     [rbp+var_60], rax
.text:0000000040B389 mov     qword ptr [rbp+var_380], 100
.text:0000000040B38F jmp     short loc_40B39F

.text:0000000040B396 loc_40B396:
.text:0000000040B396 mov     [rbp+var_30], 0Ah
.text:0000000040B39D mov     rax, qword ptr [rbp+var_380]
.text:0000000040B3A4 mov     ebx, eax
.text:0000000040B3A6 lea     rcx, [rbp+offset] ; __int64 *
.text:0000000040B3AA lea     rdx, [rbp+var_60] ; __int64 *
.text:0000000040B3AE mov     rax, [rbp+var_34B]
.text:0000000040B3B5 mov     esi, ebx ; int
.text:0000000040B3B7 mov     rdi, rax ; __int64
.text:0000000040B3BA call    _ZL9calculatelIP15_ calculate(long,int,long *,long *)

```

Figure 33

The AES key is set for encryption by calling the AES_set_encrypt_key function:

```

.text:0000000040B3BF
.text:0000000040B3BF loc_40B3BF:
.text:0000000040B3BF lea     rdx, [rbp+var_190]
.text:0000000040B3C6 lea     rax, [rbp+src]
.text:0000000040B3CD mov     esi, 256
.text:0000000040B3D2 mov     rdi, rax
.text:0000000040B3D5 call    AES_set_encrypt_key
.text:0000000040B3DA mov     [rbp+var_2C], 0
.text:0000000040B3E1 jmp     loc_40B4FE

```

Figure 34

The file content is read by calling the read_all function (figure 35).

```

.text:0000000040B41E
.text:0000000040B41E loc_40B41E: ; unsigned __int64
.text:0000000040B41E mov     rdx, [rbp+var_20]
.text:0000000040B422 mov     rcx, [rbp+var_388]
.text:0000000040B429 mov     eax, [rbp+fd]
.text:0000000040B42F mov     rsi, rcx ; unsigned __int8 *
.text:0000000040B432 mov     edi, eax ; int
.text:0000000040B434 call    _Z8read_alliPhm ; read_all(int,uchar *,ulong)
.text:0000000040B439 xor     eax, 1
.text:0000000040B43C test    al, al
.text:0000000040B43E jnz     loc_40B4D7

```

Figure 35

The content is encrypted using the AES algorithm in CBC mode:

```

.text:0000000040B444 mov     rax, [rbp+var_28]
.text:0000000040B448 add     rax, [rbp+var_20]
.text:0000000040B44C mov     [rbp+var_28], rax
.text:0000000040B450 lea    rsi, [rbp+s]
.text:0000000040B454 lea    rcx, [rbp+var_190]
.text:0000000040B45B mov     rdx, [rbp+var_20]
.text:0000000040B45F mov     rbx, [rbp+var_3B8]
.text:0000000040B466 mov     rax, [rbp+var_3B8]
.text:0000000040B46D mov     r9d, 1
.text:0000000040B473 mov     r8, rsi
.text:0000000040B476 mov     rsi, rbx
.text:0000000040B479 mov     rdi, rax
.text:0000000040B47C call   AES_cbc_encrypt
.text:0000000040B481 mov     rax, [rbp+var_20]
.text:0000000040B485 mov     rcx, rax
.text:0000000040B488 neg     rcx
.text:0000000040B48B mov     eax, [rbp+fd]
.text:0000000040B491 mov     edx, 1 ; whence
.text:0000000040B496 mov     rsi, rcx ; offset
.text:0000000040B499 mov     edi, eax ; fd
.text:0000000040B49B call   _lseek
.text:0000000040B4A0 mov     rdx, [rbp+var_20] ; unsigned __int64
.text:0000000040B4A4 mov     rcx, [rbp+var_3B8]
.text:0000000040B4AB mov     eax, [rbp+fd]
.text:0000000040B4B1 mov     rsi, rcx ; unsigned __int8 *
.text:0000000040B4B4 mov     edi, eax ; int
.text:0000000040B4B6 call   _ZL9write_alliPhm ; write_all(int,uchar *,ulong)
.text:0000000040B4BB xor     eax, 1
.text:0000000040B4BE test    al, al
.text:0000000040B4C0 jnz    short loc_40B4DA

```

Figure 36

The implementation of the AES_encrypt function from OpenSSL is displayed in the figure below.

```

.text:0000000040DA60 public AES_encrypt
.text:0000000040DA60 AES_encrypt proc near
.text:0000000040DA60
.text:0000000040DA60 var_70= dword ptr -70h
.text:0000000040DA60 var_60= dword ptr -60h
.text:0000000040DA60 var_50= qword ptr -50h
.text:0000000040DA60 var_48= qword ptr -48h
.text:0000000040DA60 var_40= qword ptr -40h
.text:0000000040DA60 var_34= dword ptr -34h
.text:0000000040DA60
.text:0000000040DA60 ; __unwind {
.text:0000000040DA60 push    r15
.text:0000000040DA62 lea    r11, Te1
.text:0000000040DA69 lea    r10, Te2
.text:0000000040DA70 push    r14
.text:0000000040DA72 push    r13
.text:0000000040DA74 push    r12
.text:0000000040DA76 lea    r12, Te0
.text:0000000040DA7D push    rbp
.text:0000000040DA7E lea    rbp, Te3
.text:0000000040DA85 mov     r14, r12
.text:0000000040DA88 push    rbx
.text:0000000040DA89 mov     r13, rbp
.text:0000000040DA8C mov     [rsp+30h+var_40], rdx
.text:0000000040DA91 mov     [rsp+30h+var_50], rsi
.text:0000000040DA96 movzx   eax, byte ptr [rdi]
.text:0000000040DA99 movzx   edx, byte ptr [rdi+3]
.text:0000000040DA9D mov     rcx, [rsp+30h+var_40]
.text:0000000040DAA2 movzx   r9d, byte ptr [rdi+7]
.text:0000000040DAA7 movzx   r8d, byte ptr [rdi+0Bh]
.text:0000000040DAAC movzx   esi, byte ptr [rdi+0Fh]
.text:0000000040DAB0 shl     eax, 10h
.text:0000000040DAB3 xor     r9d, [rcx+4]
.text:0000000040DAB7 xor     r8d, [rcx+8]
.text:0000000040DABB xor     eax, edx
.text:0000000040DABD mov     rdx, [rsp+30h+var_40]
.text:0000000040DAC2 xor     esi, [rcx+0Ch]
.text:0000000040DAC5 mov     ebx, [rcx+0F0h]
.text:0000000040DABC xor     eax, [rdx]
.text:0000000040DACD movzx   edx, byte ptr [rdi+1]
.text:0000000040DAD1 sar     ebx, 1
.text:0000000040DAD3 shl     edx, 10h
.text:0000000040DAD6 xor     eax, edx
.text:0000000040DAD8 movzx   edx, byte ptr [rdi+2]

```

Figure 37

The encrypted AES key and IV (512 bytes), followed by the file length (8 bytes) and the encryption percentage (8 bytes), are written to the encrypted file:

```

.text:00000000040B52B lea rcx, [rbp+dest]
.text:00000000040B532 mov eax, [rbp+fd]
.text:00000000040B538 mov edx, 512 ; unsigned __int64
.text:00000000040B53D mov rsi, rcx ; unsigned __int8 *
.text:00000000040B540 mov edi, eax ; int
.text:00000000040B542 call _ZL9write_allPm ; write_all(int,uchar *,ulong)
.text:00000000040B547 xor eax, 1
.text:00000000040B54A test al, al
.text:00000000040B54C jz short loc_40B558

.text:00000000040B558 loc_40B558:
.text:00000000040B558 lea rcx, [rbp+var_58]
.text:00000000040B55C mov rax, [rbp+var_388]
.text:00000000040B563 mov edx, 8 ; n
.text:00000000040B568 mov rsi, rcx ; src
.text:00000000040B56B mov rdi, rax ; dest
.text:00000000040B56E call _memcpy
.text:00000000040B573 mov rcx, [rbp+var_388]
.text:00000000040B57A mov eax, [rbp+fd]
.text:00000000040B580 mov edx, 8 ; unsigned __int64
.text:00000000040B585 mov rsi, rcx ; unsigned __int8 *
.text:00000000040B588 mov edi, eax ; int
.text:00000000040B58A call _ZL9write_allPm ; write_all(int,uchar *,ulong)
.text:00000000040B58F xor eax, 1
.text:00000000040B592 test al, al
.text:00000000040B594 jz short loc_40B5A0

.text:00000000040B5A0 loc_40B5A0:
.text:00000000040B5A0 lea rcx, [rbp+var_388]
.text:00000000040B5A7 mov rax, [rbp+var_388]
.text:00000000040B5AE mov edx, 8 ; n
.text:00000000040B5B3 mov rsi, rcx ; src
.text:00000000040B5B6 mov rdi, rax ; dest
.text:00000000040B5B9 call _memcpy
.text:00000000040B5BE mov rcx, [rbp+var_388]
.text:00000000040B5C5 mov eax, [rbp+fd]
.text:00000000040B5CB mov edx, 8 ; unsigned __int64
.text:00000000040B5D0 mov rsi, rcx ; unsigned __int8 *
.text:00000000040B5D3 mov edi, eax ; int
.text:00000000040B5D5 call _ZL9write_allPm ; write_all(int,uchar *,ulong)

```

Figure 38

Finally, the extension of all encrypted files is changed to ".royal_u":

```

.text:00000000040B7C4 lea rax, [rbp+var_30]
.text:00000000040B7C8 lea rcx, [rbp+var_50]
.text:00000000040B7CC mov edx, offset aRoyalU_0 ; ".royal_u"
.text:00000000040B7D1 mov rsi, rcx
.text:00000000040B7D4 mov rdi, rax ; this
.text:00000000040B7D7 call _ZStplIcSt11char_traitsIcESaIcEESaIT_T0_T1_ERKS6_PKS3_ ; std::operator+
.text:00000000040B7D7 ; } // starts at 40B76B
.text:00000000040B7DC lea rax, [rbp+var_30]
.text:00000000040B7E0 mov rdi, rax ; this
.text:00000000040B7E3 ; try {
.text:00000000040B7E3 call _ZNKS5c_strEv ; std::string::c_str(void)
.text:00000000040B7E8 mov rbx, rax
.text:00000000040B7EB lea rax, [rbp+var_50]
.text:00000000040B7EF mov rdi, rax ; this
.text:00000000040B7F2 call _ZNKS5c_strEv ; std::string::c_str(void)
.text:00000000040B7F2 ; } // starts at 40B7E3
.text:00000000040B7F7 mov rsi, rbx ; new
.text:00000000040B7FA mov rdi, rax ; old
.text:00000000040B7FD call rename

```

Figure 39

Indicators of Compromise

SHA256

06abc46d5dbd012b170c97d142c6b679183159197e9d3f6a76ba5e5abf999725

Royal Ransom Note

readme

Processes spawned

esxcli vm process list > list

esxcli vm process kill --type=hard --world-id=<World ID>