/Ryuk

Informe malware

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

6 | 03 | 2020

Índice

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.


UN ATAQUE SIMILAR A WANNACRY

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

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

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


≡ EL PAÍS TECNOLOGÍA

MÓVILES REDES SOCIALES BANCO DE PRUEBAS RETINA MERISTATION

Un virus de origen ruso ataca a importantes empresas españolas

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

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.

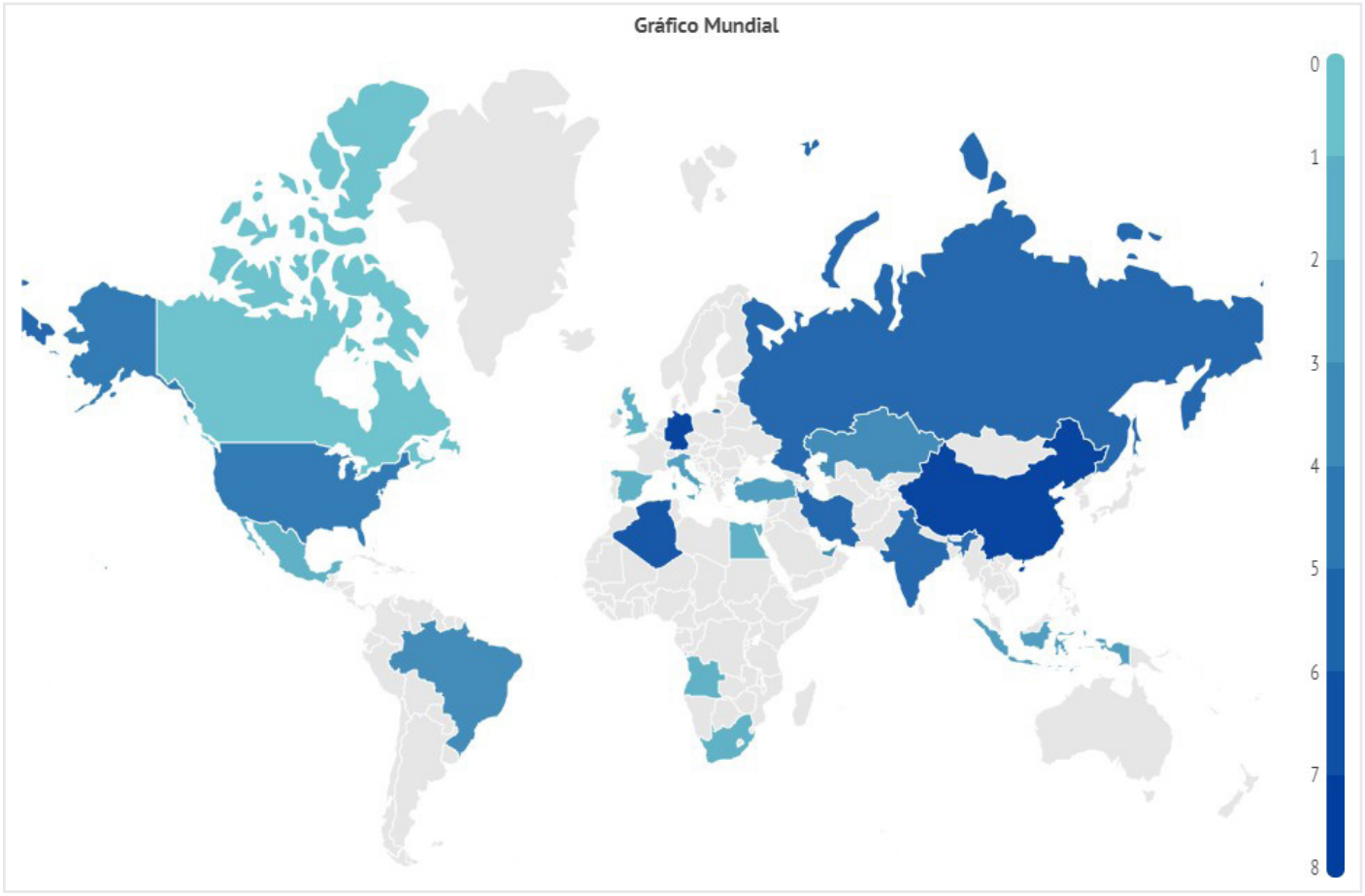


Figura 3: Gráfico mundial de actuación RYUK

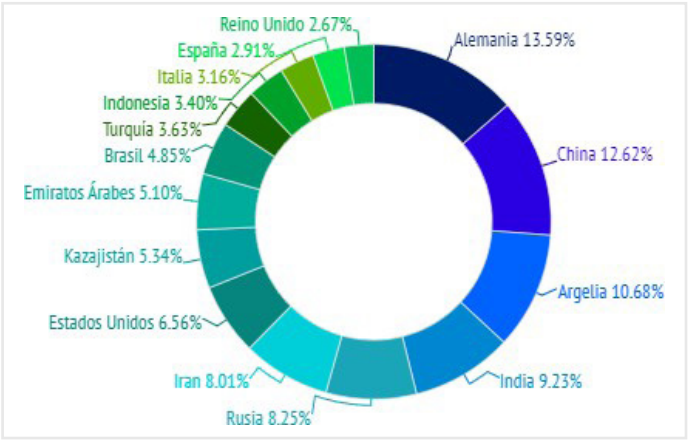


Figura 4: TOP 16 países afectados mundialmente por RYUK

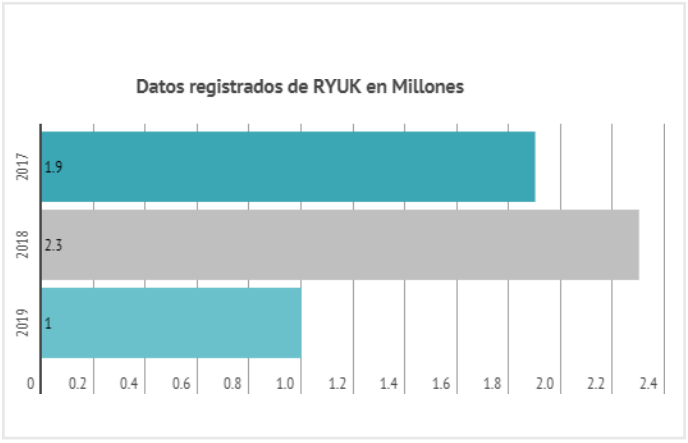


Figura 5: Cantidad de usuarios atacados por RYUK en Millones

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 cibercriminales 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.

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.

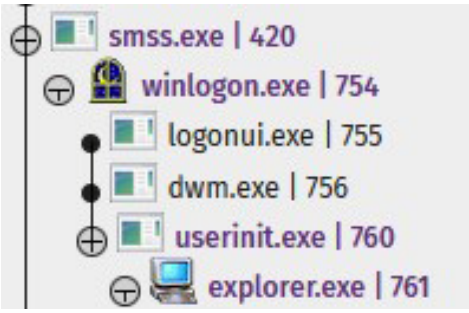


Figura 7: Registro del ataque

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
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

Figura 9: Bloqueo de la muestra

Al ser bloqueado el fichero malicioso el intruso intentó cargar una versión cifrada del ejecutable la cual fue también bloqueada.

#	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...

Figura 10: Conjunto de muestras intentadas ejecutar por el atacante

Por último intentó cargar otro archivo malicioso a través de un powershell cifrado para poder saltarse la protección del antivirus, sin embargo también fue bloqueado.

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

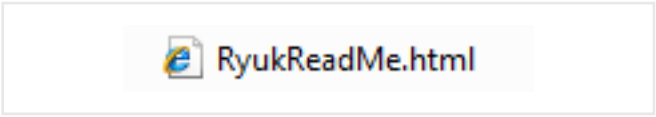
Figura 11: Powershell con contenido malicioso bloqueado

	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...	

Figura 12: Powershell con contenido malicioso bloqueado

4. Loader

En ejecución nos droppea 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**



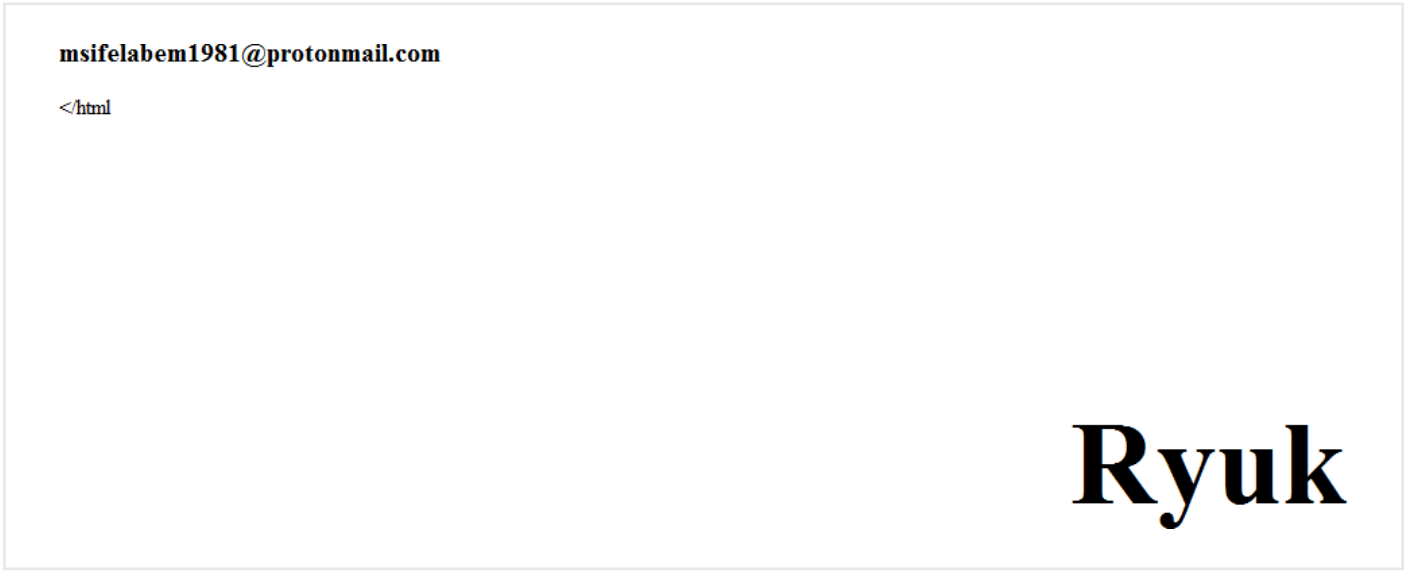
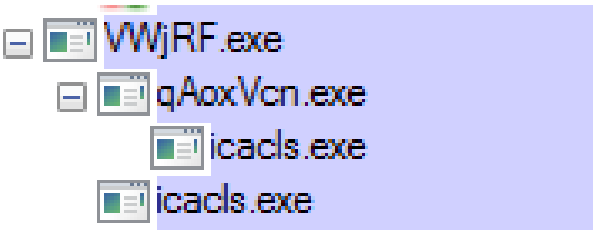


Figura 13: Nota de rescate

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.



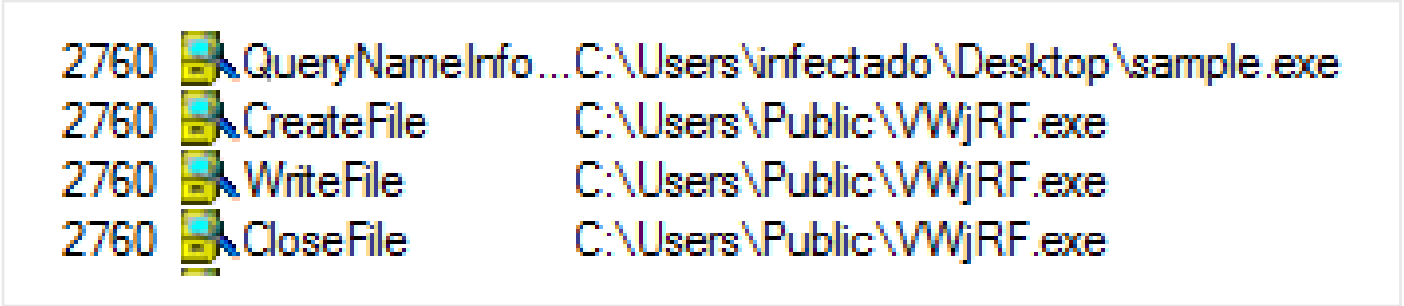


Figura 14: Ejecutables lanzados por las muestra

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 errrores (/C) y sin mostrar ningún mensaje (/Q).

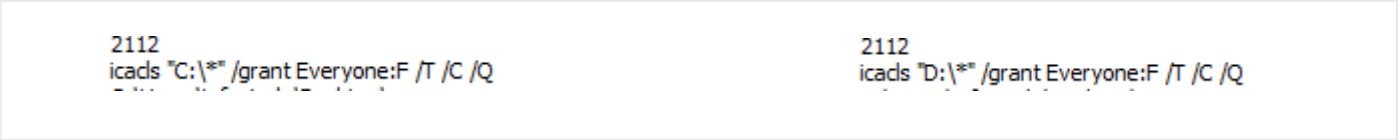


Figura 15: Parámetros de ejecución del icacls.exe lanzado por la muestra

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.

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**.

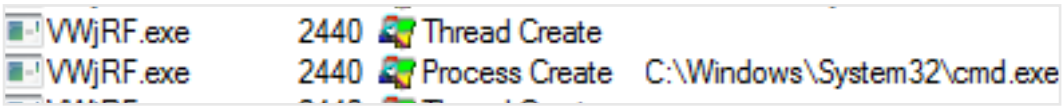


Figura 21: Ejecucióon de VWjRF.exe

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.

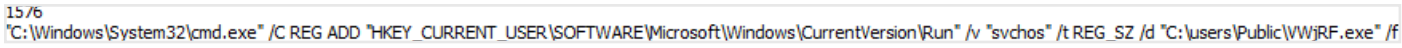


Figura 22: La muestra se asegura su persistencia en la clave de registro

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

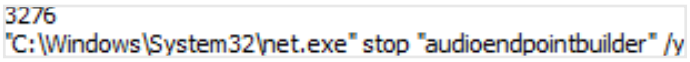


Figura 23: La muestra detiene el servicio de audio en el sistema

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_ACESS** 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 pivilegios 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(0x1FFFFFu, 0, pe.th32ProcessID);
      if ( v7 )
      {
        wcsncpy((wchar_t *)(v1 + 508i64 * v3), pe.szExeFile, 0x103ui64);
        *(v6 - 1) = pe.th32ProcessID;
        if ( OpenProcessToken(v7, 0x20008u, &TokenHandle) )
        {
          GetTokenInformation(TokenHandle, TokenUser, v2, 0, &TokenInformationLength);
          v8 = TokenInformationLength;
          v9 = GetProcessHeap();
          v2 = (PSID *)HeapAlloc(v9, 8u, v8);
          if ( GetTokenInformation(TokenHandle, TokenUser, v2, TokenInformationLength, &TokenInformationLength) )
```

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

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”

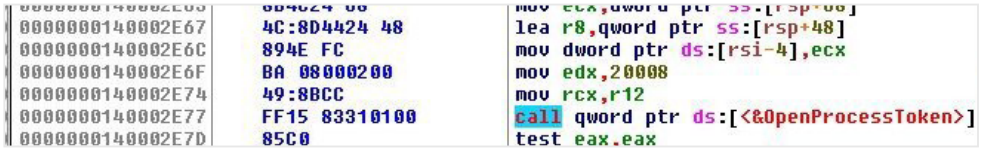


Figura 32: Lectura del token de información del proceso

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

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**.

Figura 34: Comprobación de NT AUTHORITY

Y posteriormente fuera de la rutina comprueba que no sea **csrss.exe, explorer.exe o lsaas.exe**.

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**.

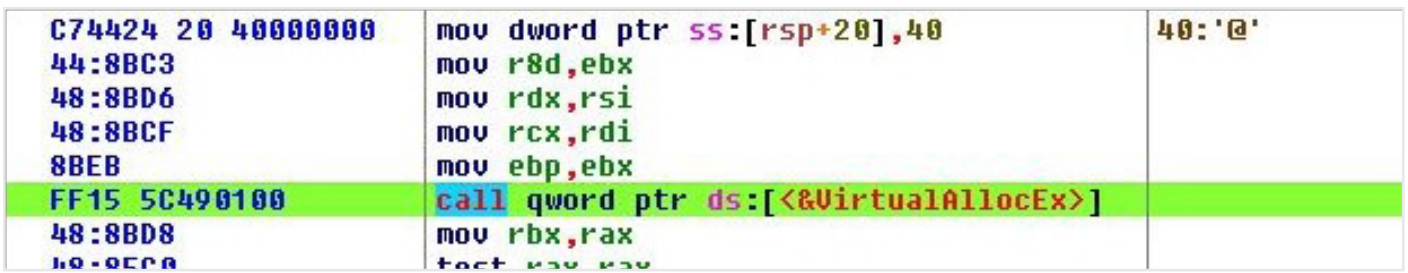
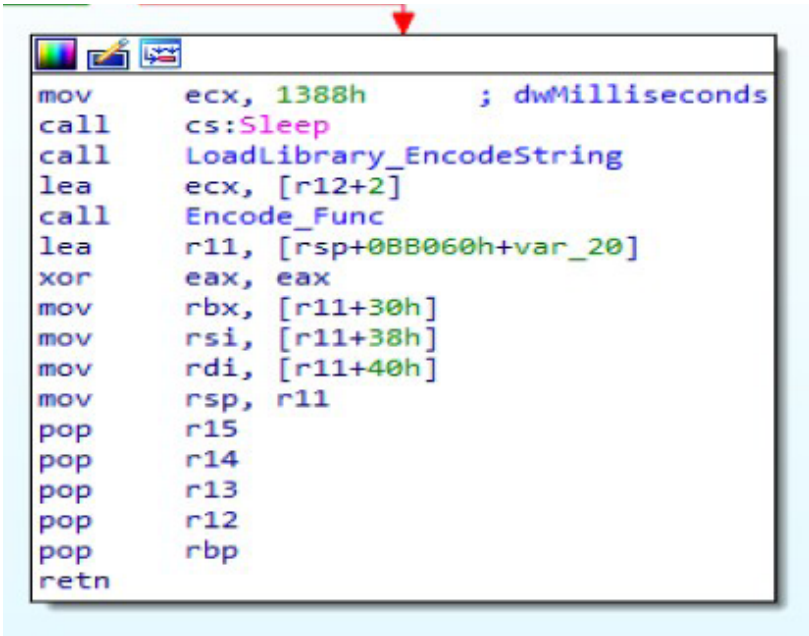


Figura 37: Llamada a VirtualAllocEx

5.4 Cifrado

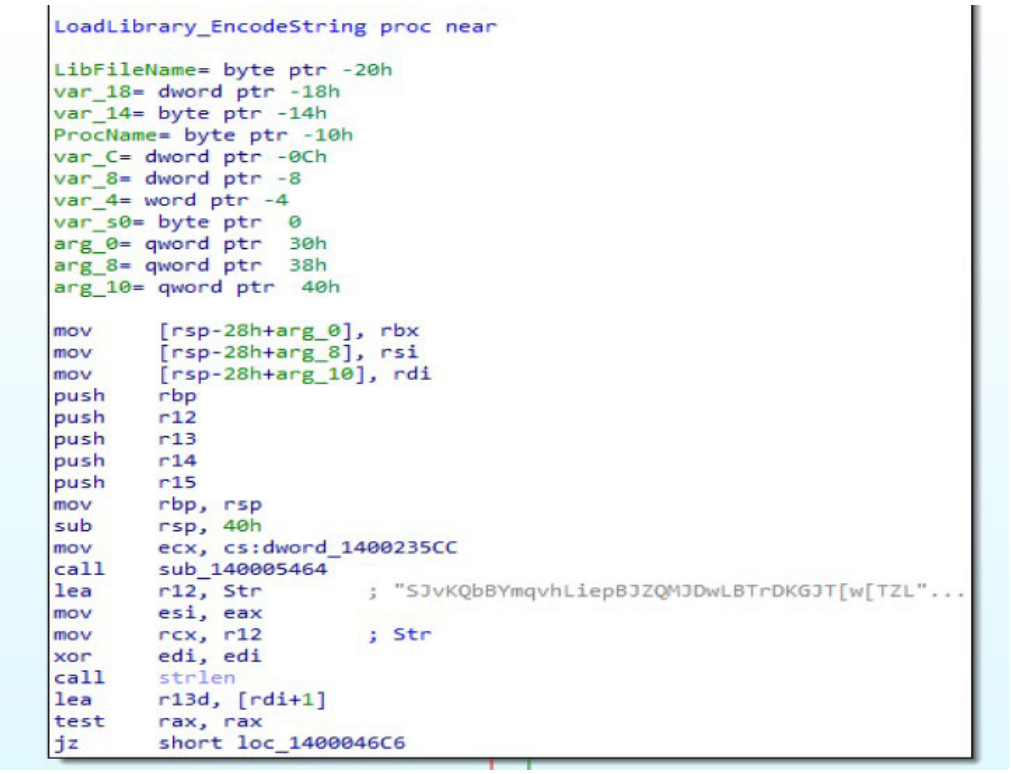
En este apartado vamos a ver la parte de cifrado de nuestra muestra. En la siguiente imagen observamos 2 subrutinas llamadas **“LoadLibrary_EncodeString”** y **“Encode_Func”** que son las encargadas de realizar el procedimiento de cifrado.



```
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.



```
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.

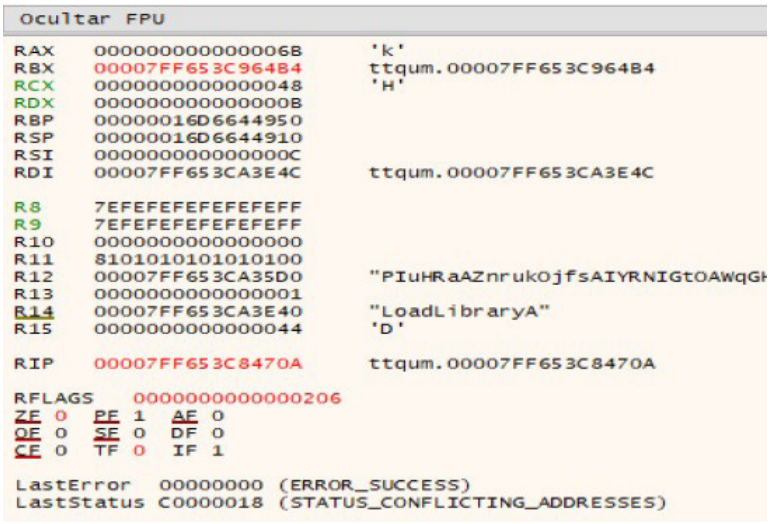
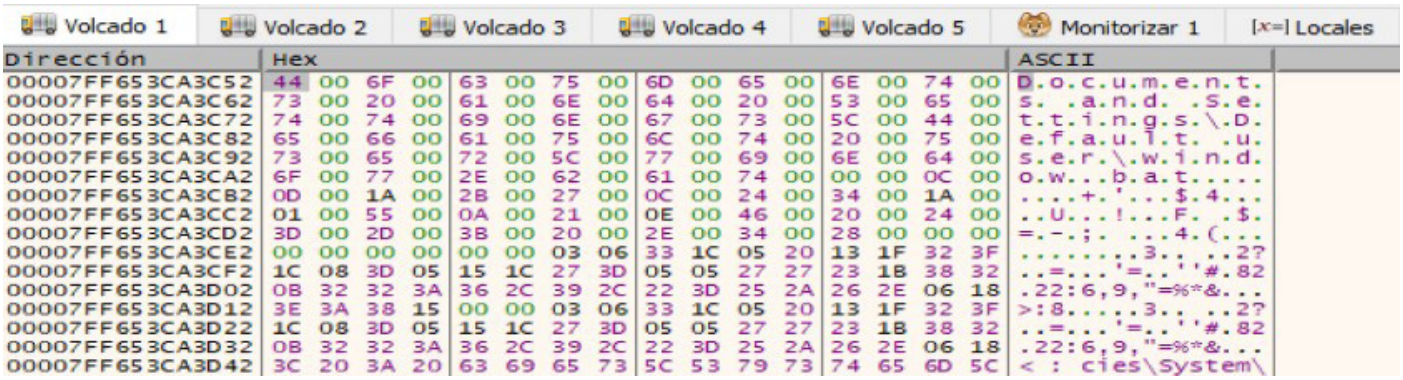


Figura 40: Desofuscación en dinámico

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

Figura 41: Carga de comandos

Posteriormente, carga la ubicación donde soltará 3 ficheros: **Windows.bat, run.sct y start.bat**.



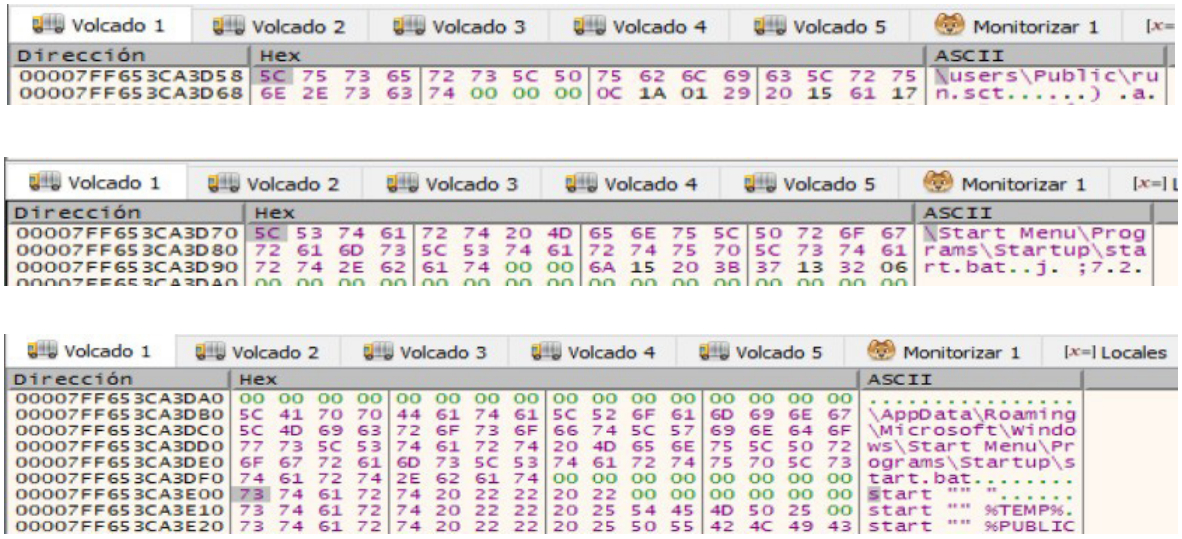


Figura 42: Ubicación de los ficheros

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.

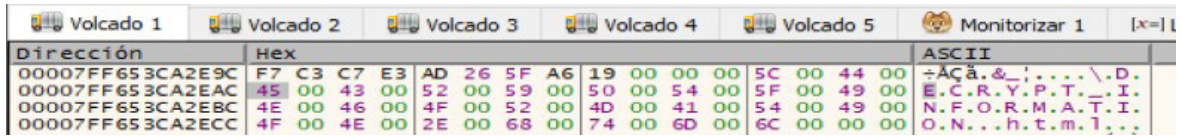


Figura 43: String de DECRYPT INFORMATION.html

El tercero, **UNIQUE_ID_DO_NOT_REMOVE**, contiene la clave codificada que se utilizará en la siguiente subrutina para realizar el cifrado.

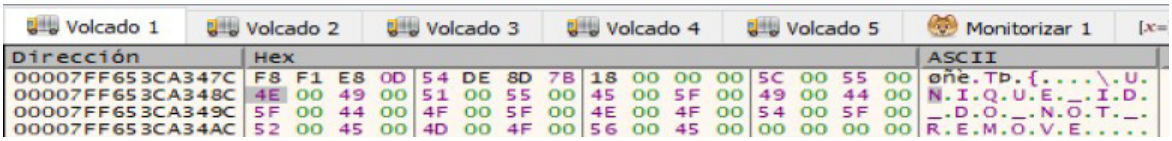


Figura 44: String de UNIQUE ID DO NOT REMOVE

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

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.

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.

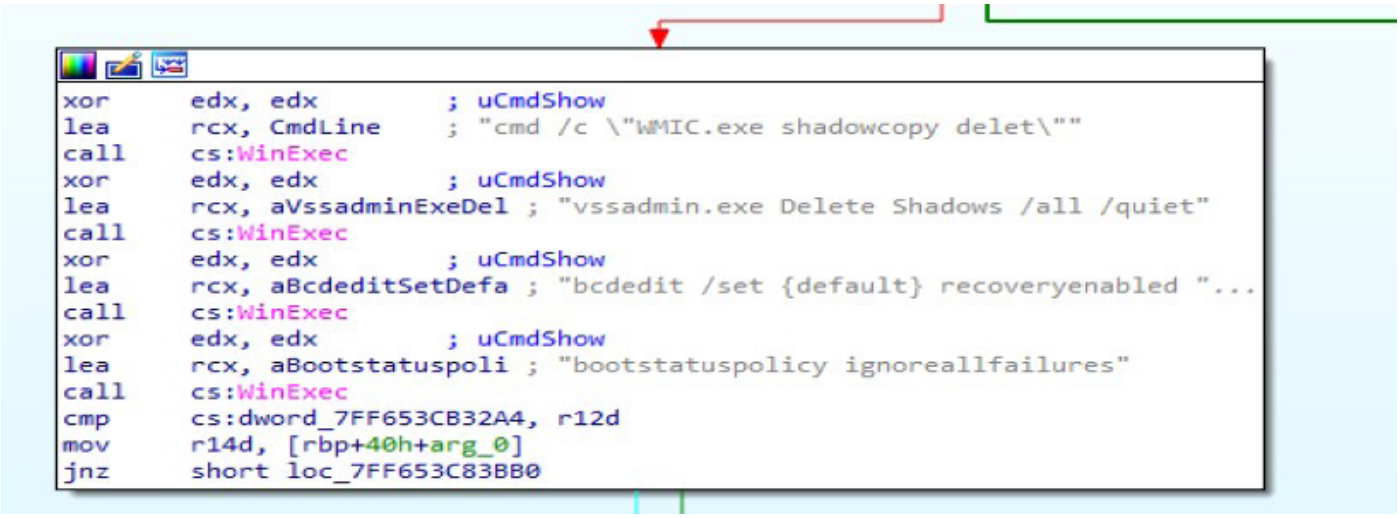


Figura 47: Desactivación de medidas de restauración

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

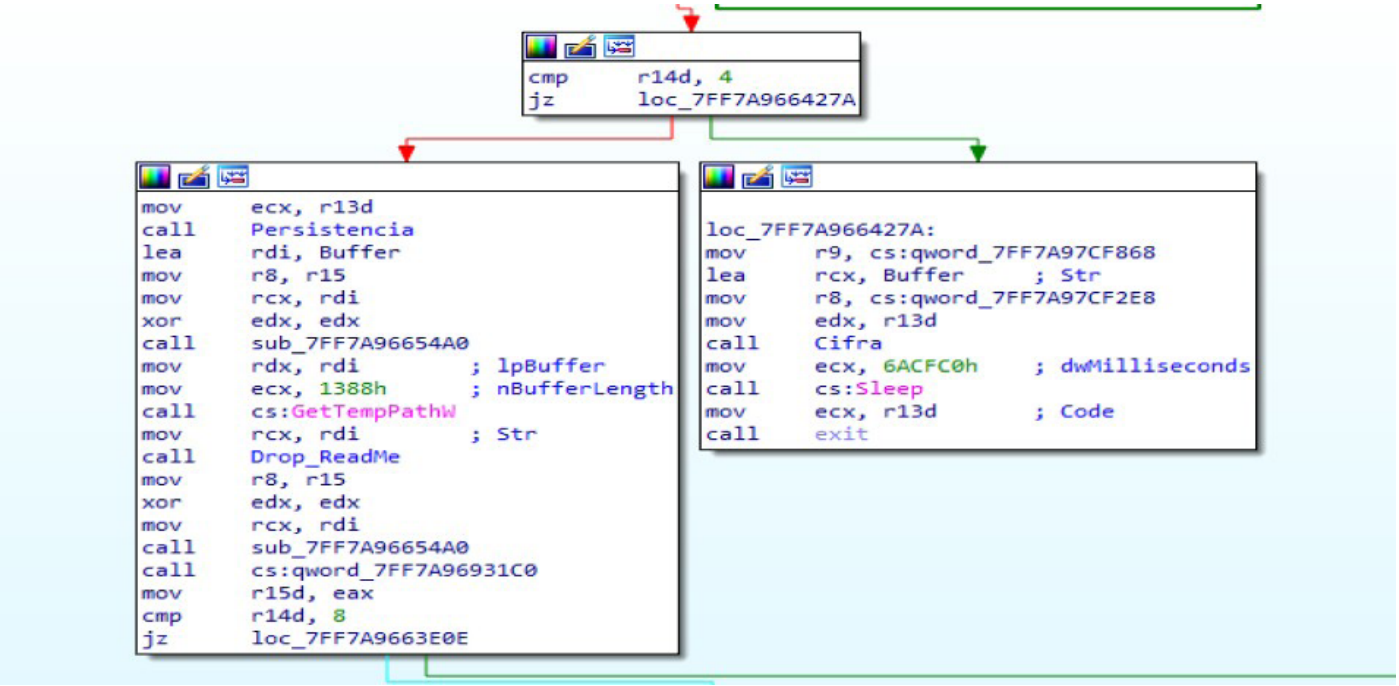


Figura 48: Lanzamiento de nota de rescate

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

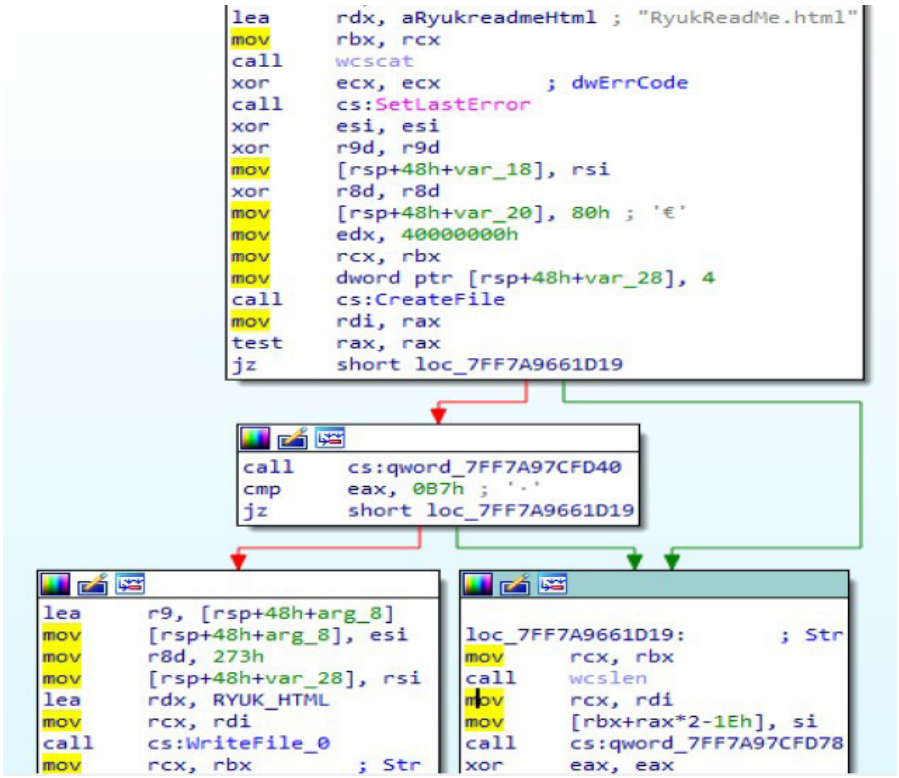


Figura 49: Carga y escritura del contenido del fichero

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

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.

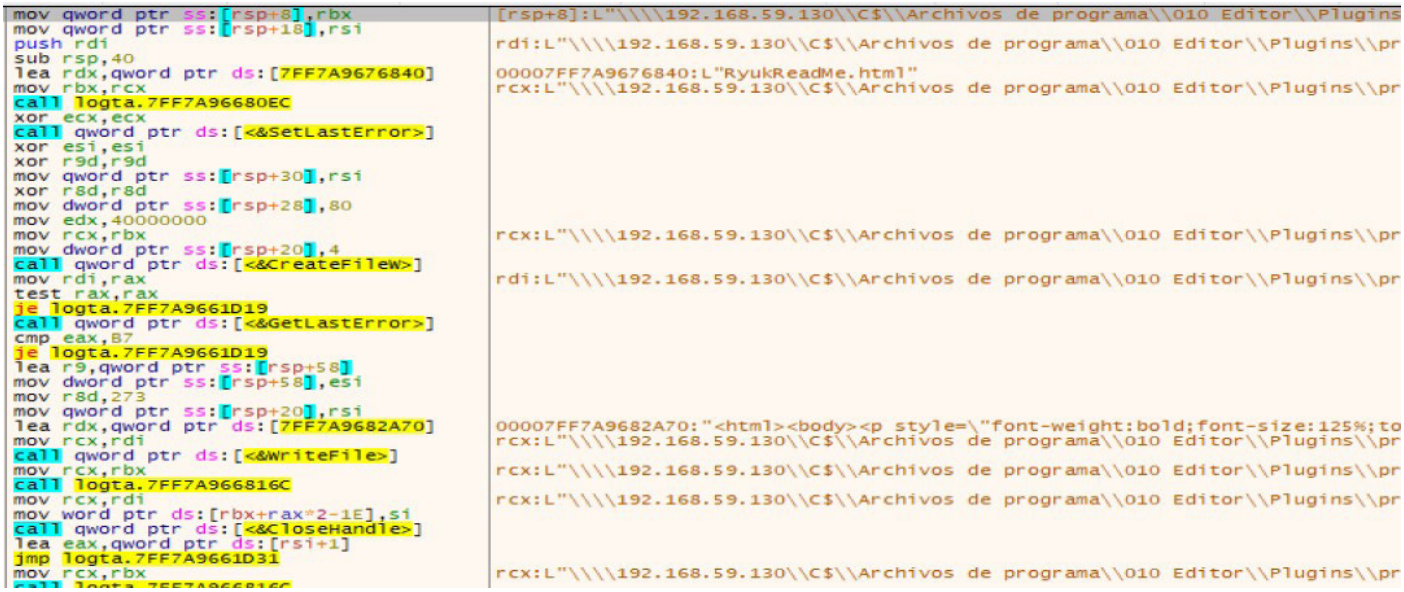


Figura 51: Cifrado de los ficheros del sistema

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	
GetVersionExW	Obtener la Versión SO	**LpVersionInformation**	Determina si es + de Wxp
ImpersonateSelf	Habilita Privilegios a un thread		
GetCUrrentThread	Obtiene el handle de un thread	**DesiredAccess**	Especifica el tipo de acceso al Token
OpenThreadToken	Abre un token perteneciente a un thread	**SeDebugPrivilege**	Sir ve para obtener privilegios avanzados
LookupPrivilegeValueW	Nos da información de la LUID para saber el privilegio ai que escalamos		
AdjustTokenPrivileges	Habilita ciertos permisos para un Token		
CreateToolHelp32Snapsh ot	Realiza una captura de los procesos en estado “Running”		
WriteProcessMemory	Escribe en la memoria de un proceso determinado		
CreateRemoteThread	Crea un Thread en est ado suspendido		
ShellExecuteW	Lanza una shell para ejecutar el payload		
CommandlineToArgW	Parsea una instrucción cmd y devuelve un array de punteros de dicha instrucción		
DeletefileW	Elimina un fichero según los parámetros		
CryptExportKey	Export auna clave crypto		
GetDriveTypeW	Nos da información sobre un disco, para saber si es un USB, una unidad de CD...		
CryptDeriveKey	Crea una key a partir de otra recogiendo datos pasados		
CryptGenKey	Genera una clave de sesión random o un par de claves asimétricas (Pública/Privada)		
GetlogicalDrives	Nos devuelve un bitmask de los discos disponibles en el sist ema		
WNetEnumResourceW	Enumera los disposit ivos de red		
CryptAcquireContextW	Trata de buscar en el CSP el algoritmo de cifrado deseado		
CryptEncrypt	Encripta datos en el algorit mo que se haya designado en el CSP		
CryptDecrypt	Desen cri pta l os da tos encriptados por CryptEncry pt		
CryptDestroyKey	Destru ye el handle de hkey para que no pueda volver a serutil izado	**hkey**	Un handle para abrir la clave de registro de la key
CryptlmportKey	Transfíere una key de un CSP		

7. IOC

MD5

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

Direcciones IP relacionadas

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

Fichero de rescate

- RyukReadMe.html

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/, Publicada el 04/11/2019.

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

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

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

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

Más información:

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