

Malware Report

“Dridex Version 4”

August, 2017

CONTENTS

1. EXECUTIVE SUMMARY	3
2. CHARACTERISTICS OF THE TROJAN	3
3. INFECTION PROCESS	6
3.1. Infection Vectors.....	6
3.2. Interactions with the Affected System	6
4. PERSISTENCE IN THE SYSTEM.....	7
5. INJECTION VIA ATOMBOMBING	9
5.1. Search for the target process	9
5.2. Search for alertable threads	10
5.3. Injection of shellcode in the target process.....	10
5.4. Execution of the shellcode in the target process	12
6. NETWORK CONNECTIONS.....	13
7. IOCS.....	14
8. REFERENCIAS	14

1. Executive Summary

The present document gathers analysis of a new variant of harmful code called "Dridex", specifically the fourth version.

Dridex is a banking Trojan famous for its sophistication and its ability to go undetected on the devices it infects. These devices, once infected, are incorporated onto a modular botnet, at which point malicious characteristics, whether external or their own, can be freely added to them, via modules or libraries.

The first version appeared toward the end of 2014. At the beginning of 2015, a new, important update was launched, giving way to a second version. When looking at the earlier versions of Dridex, the most stable and resistant of them was the third, which was launched in April 2015 and was used in well-known cyberattacks up until the fourth version, the latest known version and subject of this report, which was found in February of 2017.

No new major updates for Dridex had been found since the dismantlement of important components of the botnet, carried out by government agencies in 2015. [1]

This new variant of the banking Trojan incorporates new functionalities. One of these is called AtomBombing, a functionality whose aim is to inject code without calling suspicious APIs to avoid being detected by monitoring systems. It incorporates the DLL hijacking technique to achieve persistence. Finally, various cryptographic methods were optimized and used to obtain the configuration. [2]

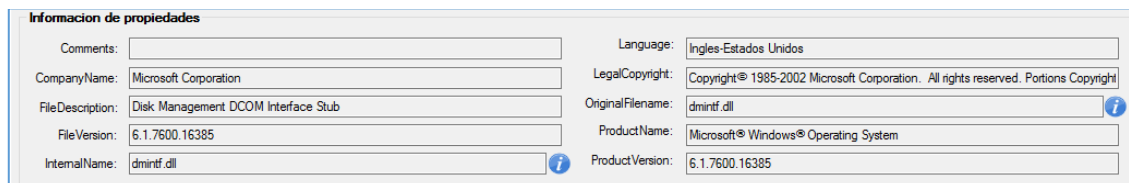
2. CHARACTERISTICS OF THE TROJAN

The following are some static properties of the analysed file.

The hash of the Trojan:

MD5	001fcf14529ac92a458836f7cec03896
SHA256	a6db7759c737cbf6335b6d77d43110044ec049e8d4cbf7fa9bd4087fa7e415c7

The internal date of creation of the analyzed sample is May 16, 2017. The file in question was compiled to be executed in 64 bit environments and, at the same time, simulate the legitimate dll of Microsoft.



The screenshot shows the 'Información de propiedades' (Properties) dialog box for the file 'dmintf.dll'. The 'General' tab is active, displaying the following information:

- Comments: (empty)
- Language: Inglés-Estados Unidos
- CompanyName: Microsoft Corporation
- LegalCopyright: Copyright© 1985-2002 Microsoft Corporation. All rights reserved. Portions Copyright
- FileDescription: Disk Management DCOM Interface Stub
- OriginalFilename: dmintf.dll
- FileVersion: 6.1.7600.16385
- ProductName: Microsoft® Windows® Operating System
- InternalName: dmintf.dll
- ProductVersion: 6.1.7600.16385

Figure 1. File properties

Additionally, it is encrypted with a distinctive algorithm to avoid detection by antiviruses.

It has been observed that the executable has a fairly high number of sections, 11 in total, as we can see in Figure 2:

property	value	value	value	value	value	value	value	value	value	value	value	value
name	.text	.code	.sbss	.rdata	.data	.pdata	DATA	.rsrc	.rsrc	.reloc	.kwgrcd	
virtual-size	0x000005E6 (1510)	0x00001AFC (6906)	0x00000657 (1623)	0x0001DDCB (122315)	0x00002F62 (12130)	0x000009FA (1530)	0x00026B3F (158527)	0x0001C16E (115054)	0x000004C8 (1224)	0x0000056C (1388)	0x00000AE7 (2791)	
virtual-address	0x00001000	0x00002000	0x00004000	0x00005000	0x00023000	0x00026000	0x00027000	0x0004E000	0x0006B000	0x0006C000	0x0006D000	
raw-size	0x00001000 (4096)	0x00002000 (8192)	0x00001000 (4096)	0x0001E000 (122880)	0x00003000 (12288)	0x000026000	0x000027000 (159744)	0x0001D000 (118784)	0x00001000 (4096)	0x0000C000	0x00001000 (4096)	0x00001000 (4096)
raw-data	0x00001000	0x00002000	0x00004000	0x00005000	0x00023000	0x00026000	0x00027000	0x0004E000	0x0006B000	0x0006C000	0x0006D000	
PointerToRelocations	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
PointerToLineNumbers	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
NumberOfRelocations	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
NumberOfLineNumbers	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
md5	A307CFABD21EES...	80A66726FB60EAD...	A388B27F841B4F...	258595205FOC58FBE...	B3AF0999C1DCB41...	3848DA66328342...	D52A19B03FCF1B5...	A6F51B37F58FA376...	3EC94FD83DAC...	CB1AE24217F540...	620F0B67A91F774151B...	
cave	0x00000A1A (2586)	0x00000504 (1284)	0x000009A9 (2473)	0x00000235 (565)	0x0000009E (158)	0x00000406 (2566)	0x000004C1 (1217)	0x00000E92 (3730)	0x00000B38 (2872)	0x00000A94 (2708)	0x00000519 (1305)	
entropy	3.139	5.116	2.033	7.844	2.699	0.543	7.799	7.783	1.307	0.819	0.000	
entry-point	x	-	-	-	-	-	-	-	-	-	-	-
obfuscated	-	-	-	-	-	-	-	-	-	-	-	-
blacklisted	-	-	-	-	-	-	-	-	-	-	-	-
readable	x	x	x	x	x	x	x	x	x	x	x	x
writable	-	-	-	-	x	-	-	x	-	-	-	-
executable	x	x	x	-	-	-	-	-	-	-	-	-
shareable	-	-	-	-	-	-	-	-	-	-	-	-
discardable	-	-	-	-	-	-	-	-	-	x	-	-
cacheable	x	x	x	x	x	x	x	x	x	x	x	x
pageable	x	x	x	x	x	x	x	x	x	x	x	x
initialized-data	-	-	-	x	x	x	x	x	x	x	x	x
uninitialized-data	-	-	-	-	-	-	-	-	-	-	-	-

Figure 2. Static information of the analyzed binary

In the DATA section, we can observe that the entropy is at 7.799, and is a fairly large in size. It is in this section that the highly encrypted and packaged binary (which, once decrypted, becomes the real malicious code) can be found.

In the first decrypted layer, the executable stores memory in the process, then copies the code and, finally, summons it and runs it, as we see in Figure 3:

```

call AddressDOSHeader
mov [rsp+1C8h+var_C8], rax
mov rcx, rax
call AddressPEHeader
mov r8d, [rax+50h]
mov eax, r8d
mov [rsp+1C8h+var_C0], rax
mov rdx, [rsp+1C8h+var_18]
lea rax, [rsp+1C8h+dwSize]
mov rcx, rax
mov [rsp+1C8h+var_1A8], rax
call CopyPEtoMemory
mov rcx, [rsp+1C8h+var_1A8]
call rax ; Execute shellcode
    
```

Figure 3. Jump to shellcode

The first thing the code does is to obtain the addresses of the functions that it will eventually be using. It does this with a dynamic search through the libraries downloaded by the program.

To carry out this task, it runs through the PEB_LDR_DATA structure and the LDR-MODULE structures to locate the base address of the loaded dlls. It proceeds to access the offset of the export table in order to run through all of the functions exported by the dll and find the address of the sought function in the computer's memory.

<pre> mov r9,qword ptr gs:[30] mov rax,qword ptr ds:[r9+60] mov rax,qword ptr ds:[rax+18] mov r9,rax add r9,20 mov rax,qword ptr ds:[rax+20] cmp rax,r9 </pre>	<p>Acceso a] TEB Acceso a] PEB Acceso a] PEB_LDR_DATA Acceso a la lista InMemoryOrderList</p>
--	--

Figure 4. Enumeration of loaded modules

The shellcode, in turn, checks to see whether there is a hook in the undocumented LdrLoadDll function, accessing its address and checking whether the first byte is the same as E9, the equivalent of a jmp assembler.

```

48 81 EC D8 00 00 00 sub rsp,D8
48 8D 05 C8 0B 00 00 lea rax,qword ptr ds:[<LdrLoadDll>]
31 C9 xor ecx,ecx
89 CA mov edx,ecx
4C 8D 84 24 A8 00 00 lea r8,qword ptr ss:[rsp+A8]
80 3D B5 0B 00 00 E9 cmp byte ptr ds:[<LdrLoadDll>],E9
4C 89 84 24 A0 00 00 mov qword ptr ss:[rsp+A0],r8
48 89 84 24 98 00 00 mov qword ptr ss:[rsp+98],rax
48 89 94 24 90 00 00 mov qword ptr ss:[rsp+90],rdx
0F 85 26 02 00 00 jne 122048
                    
```

Figure 5. Hook Verification

If the previous verification was successful, it proceeds to demap the dll memory process with the name "snxhk.dll" which is an Avast and AVG library that creates hooks to monitor processes happening in the sandbox.

4C 8D 0D 54 10 00 00	lea r9,qword ptr ds:[123085]	123085:"snxhk.dll"
45 8A 14 01	mov r10b,byte ptr ds:[r9+rax]	
48 83 C0 01	add rax,1	
44 38 D1	cmp cl,r10b	
48 89 44 24 20	mov qword ptr ss:[rsp+20],rax	[rsp+20]:VirtualAlloc

Figure 6. Library: snxhk.dll

Finally, the shellcode decrypts the executable found in the DATA section in the computer's memory, copies it into the base image's address, and then runs the new resulting executable.

RIP	0000000140028BFC	48 89 5C 24 18	mov qword ptr ss:[rsp+18],rbx
	0000000140028C01	48 89 4C 24 08	mov qword ptr ss:[rsp+8],rcx
	0000000140028C06	55	push rbp
	0000000140028C07	56	push rsi

Figure 7. Decrypted executable

In summary, the full process of the sample being unpacked can be seen in Figure 8, where it is detailed more schematically.

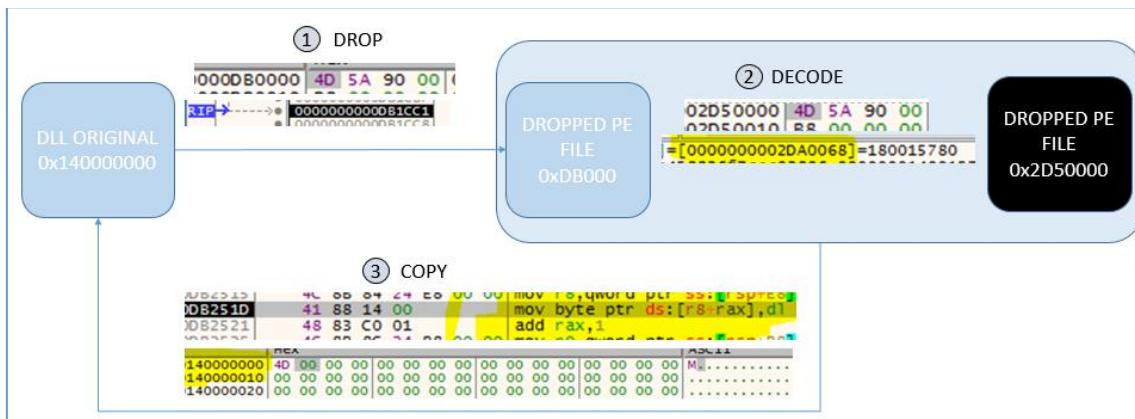


Figure 8. Complete unpacking process

3. INFECTION PROCESS

3.1. Infection Vectors

The infection of the device is not clearly understood. It may come by way of an exploit kit or spam campaign.

3.2. Interactions with the Affected System

Once it is run, the Trojan will proceed to verify if it is the only instance of malware running on the device, as well as to verify if it has already been injected in the explorer.exe process.

All of this is carried out by creating and opening a mutex. In order to achieve this, it first strings together the device name and the username, then calculates its MD5 hash.

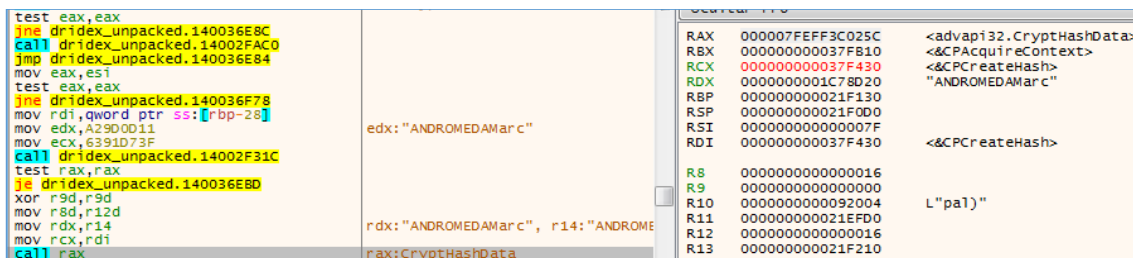


Figure 9. Hash calculation

Next, it adds brackets to the beginning and the end, and separates it with hyphens, similar to a COM object.

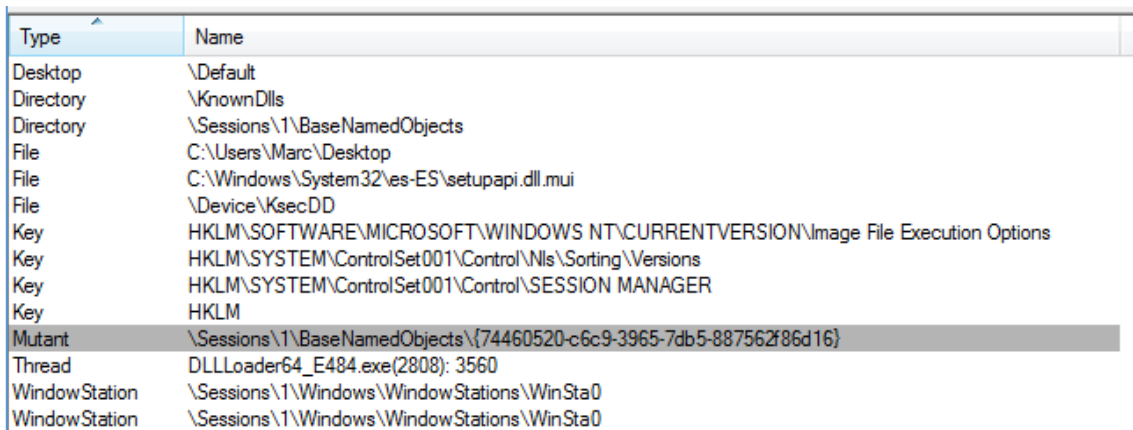


Figure 10. Mutex created in the system

Using this algorithm, it may be possible to develop a vaccine that creates these mutexes in systems to avoid infection by Dridex.

Malware that is not running creates a folder in %WINDOWS%\system32\[0-9]{4}

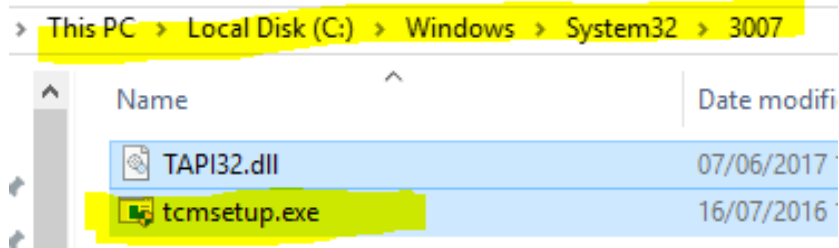


Figure 11. Created folder

The malware copies a legitimate .exe into the folder along with an associated .dll or .cpl. This .dll or .cpl is not legitimate — it's a Trojan. Upon running the .exe from the folder, the malicious .dll or .cpl will load via a technique known as hijacking.

It also programs a task with a randomized name ("Domitxtdoi" in our example in Figure 12), which will run every 60 minutes.

```
schtasks.exe /Create /F /TN "Domitxtdoi" /SC minute /MO 60 /TR "C:\Windows\system32\3007\tcmsetup.exe" /RL highest
```

Figure 12. Creation of task

In this example, we see that the tcmsetup.exe runs so that the malicious .dll, TAPI32.dll, loads, thus beginning the infection process.

After programming the task, it launches a series of commands: it creates a rule in the firewall for explorer.exe, which is where it will be injected:

```
netsh advfirewall firewall add rule name="Core Networking - Multicast Listener Done (ICMPv4-In)" program="C:\Windows\Explorer.EXE" dir=in action=allow protocol=TCP localport=any
```

Creation of the malicious task

```
schtasks.exe /Create /F /TN "Ut dcm" /SC minute /MO 60 /TR "C:\Windows\system32\3007\tcmsetup.exe" /RL highest
```

During this process, the malicious .dll will have been injected into the explorer.exe process using the AtomBombing technique. It will then wait for the user to open a browser like Internet Explorer, Firefox, Chrome, etc.

The moment the user opens a browser, a new shellcode will be injected from explorer.exe to the browser using the same AtomBombing technique.

4. PERSISTENCE IN THE SYSTEM

To ensure its persistence in the system, it carries out the following actions.

It creates a folder with four random numbers on C:\Windows\System32, inside of which it copies a legitimate Windows executable (not always the same one) and a .dll that it knows will be loaded by the executable. This .dll will be modified with the harmful code.

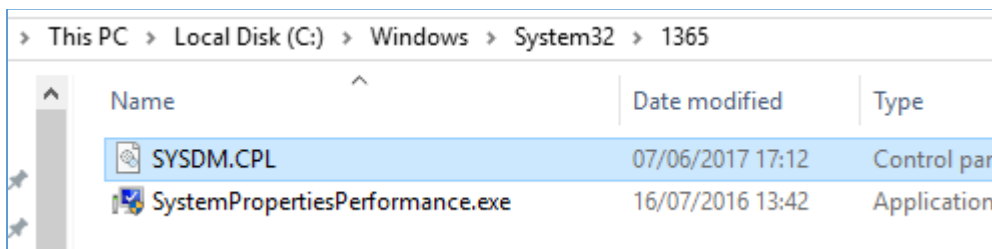


Figure 13. Persistence in the system

This technique is known as DLL hijacking. It takes advantage of the command that allows the system to search libraries/files that it's going to load/use. In the case of the image above, the executable "SystemPropertiesPerformance.exe" will load "SYSDM.CPL" among other libraries. By default, the first place that it will search for the "SYSDM.CPL" file will be in the directory where the application is running, in this case C: \ Windows \ System32 \ 1365. If it does not find it, it will look it up on other routes depending on how the search order of .dlls in the system is set.

When it copies an executable and a modified .dll in the same directory, Dridex's aim is to raise as little suspicion as possible, since its malicious actions are carried out by way of a legitimate program.

To execute the file, it creates a scheduled task to run it in the random number folder (C: \ Windows \ System32 \ 1365) every hour, as indicated in the previous section.

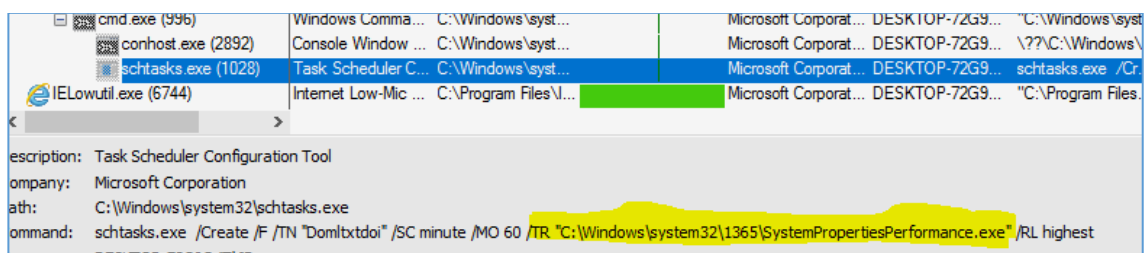


Figure 14. Creation of the programmed task

As already mentioned, the folder is composed of four random numbers, and the executable it creates is not always the same, just like the .dll, so it is aware of which executable loads which library at all times, and is able to modify said library with harmful code.

Going further in our analysis, we see that it acts in the following manner:

1. It will list all executables in the folder "C: \ Windows \ System32 \"
2. It will hash the name of each executable and compare it with a value that has been previously saved. If it matches, it will remain with that executable (in each execution that the hash is different).
3. It will read the IAT of the selected executable and from there choose a .dll for eventual hijacking.
4. It will read the IAT of the .dll selected in point 3.

PANDA ID-0601/2017 Informe de Malware "Trojan:Win64/Dridex"

5. It will make a copy of the malicious code (the .dll itself) and add a section at the end with a random name to copy the IAT obtained in point 4.
6. It will copy both the selected executable (3) and the modified malicious .dll (5) into a random folder.

In this way it obtains persistence in the system and every time that file is executed it will load the malicious .dll.

The malware will also create a copy of itself in executable format along with a registry key in the AppData\Roaming\[random folder name] with the route in "HKCU\Software\Microsoft\Windows\CurrentVersion\Run".

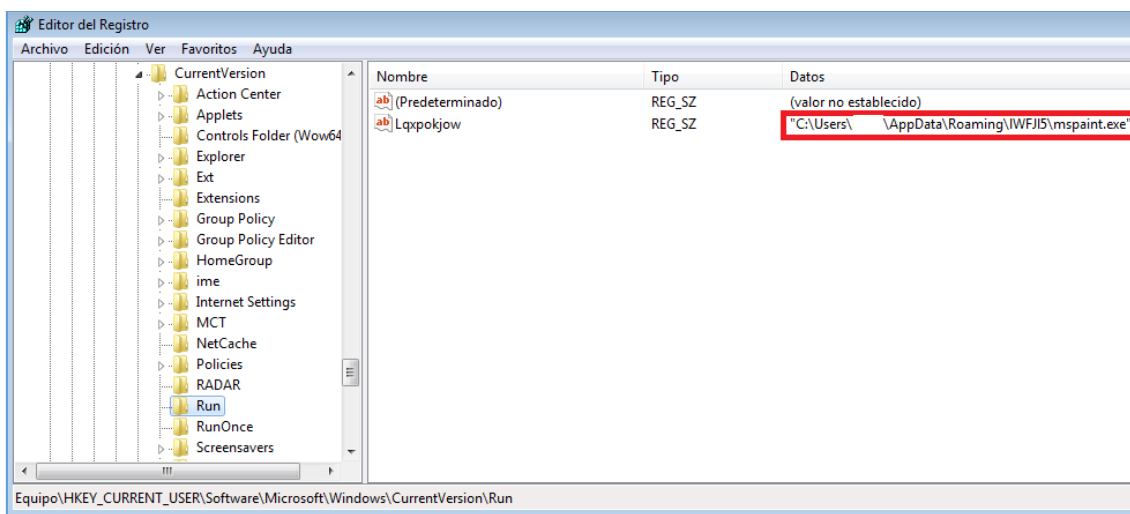


Figure 15. Registry key

5. INJECTION VIA ATOMBOMBING

Dridex uses the AtomBombing technique to write a shellcode in other processes without raising suspicions.

It achieves this through APC calls and one of the most used Windows Executive Objects, called Atoms.

Below are the different phases of injection into another process.

5.1. Search for the target process

The target process in this case is explorer.exe, and to inject into it, it must first be accessed in order to perform an enumeration of the processes involved, making use of functions such as the following:

0000000140038F02	8D 4A 02	lea ecx,dword ptr ds:[rdx+2]	CreateToolhelp32Snapshot	Hide FPU
0000000140038F05	FF D0	call rax		
0000000140038F06	4E 8B 00	mov rdx,rax		
0000000140038F0A	E8 02	jmp a023c6338ed5e50c2c090458eeeca181b.140038F0E		
0000000140038F0C	33 D2	xor edx,edx		
0000000140038F0E	48 80 4C 24 20	lea rcx,qword ptr ss:[rsp+20]	var_2F8	
0000000140038F13	E8 D8 06 FF FF	call a023c6338ed5e50c2c090458eeeca181b.1400295F0		
0000000140038F18	48 80 4C 24 20	lea rcx,qword ptr ss:[rsp+20]	var_2F8	
0000000140038F1D	E8 C2 04 FF FF	call a023c6338ed5e50c2c090458eeeca181b.1400293E4		
0000000140038F22	84 C0	test al,al		
0000000140038F24	75 2E	jne a023c6338ed5e50c2c090458eeeca181b.140038F54		
0000000140038F26	40 38 7C 24 28	cmp byte ptr ss:[rsp+28],d1	var_2F0	
0000000140038F2B	74 0A	je a023c6338ed5e50c2c090458eeeca181b.140038F37		
0000000140038F2D	48 80 4C 24 20	lea rcx,qword ptr ss:[rsp+20]	var_2F8	
0000000140038F32	E8 81 06 FF FF	call a023c6338ed5e50c2c090458eeeca181b.1400295B8		
0000000140038F37	48 80 4D 98	lea rcx,qword ptr ss:[rbp-68]	[rbp-68]: "\\Device\\Harddisk	
0000000140038F38	E8 9C 0D FF FF	call a023c6338ed5e50c2c090458eeeca181b.140029C0C		
0000000140038F40	48 80 4C 24 78	lea rcx,qword ptr ss:[rsp+78]	var_2A0	
0000000140038F45	E8 92 0D FF FF	call a023c6338ed5e50c2c090458eeeca181b.140029C0C		
0000000140038F4A	40 38 7C 24 58	cmp byte ptr ss:[rbp-58],d1	var_2C0	
0000000140038F4F	E9 7D 01 00 00	jmp a023c6338ed5e50c2c090458eeeca181b.1400390D1		
0000000140038F54	BB 38 02 00 00	mov ebx,238		
0000000140038F59	48 80 4D C0	lea rcx,qword ptr ss:[rbp-40]		
0000000140038F5E	33 D2	xor edx,edx		

Once it finds the process explorer.exe, it calls the OpenProcess function to begin enumerating alertable threads.

5.2. Search for alertable threads

000000014003901A	FF D0	call rax	Process32First	Hide FPU
000000014003901C	85 C0	test eax,eax		
0000000140039024	OF 85 50 FF FF FF	jne a023c6338ed5e50c2c090458eeeca181b.140038F74		
0000000140039029	E8 05	call a023c6338ed5e50c2c090458eeeca181b.14002FAC0		
000000014003902B	B8 7F 00 00 00	mov eax,7F		
0000000140039030	85 C0	test eax,eax		
0000000140039032	OF 84 3C FF FF FF	jne a023c6338ed5e50c2c090458eeeca181b.140038F74		
0000000140039038	45 33 E4	xor r12d,r12d		
000000014003903B	44 39 66 04	cmp dword ptr ds:[rsi+4],r12d		
000000014003903F	7E 67	jle a023c6338ed5e50c2c090458eeeca181b.1400390A8		
0000000140039041	41 8B 04	mov edx,r12		
0000000140039044	48 8B C6	mov rcx,rsi		
0000000140039047	E8 80 A7 FF FF	call a023c6338ed5e50c2c090458eeeca181b.1400337FC		
000000014003904C	48 8B F8	mov rdi,rax		
000000014003904F	33 78 34 00	cmp dword ptr ds:[rax+34],0		
0000000140039053	74 4A	je a023c6338ed5e50c2c090458eeeca181b.14003909F		
0000000140039055	33 D8	xor ebx,ebx		
0000000140039057	39 5E 04	cmp dword ptr ds:[rsi+4],ebx		
000000014003905A	7E 38	jle a023c6338ed5e50c2c090458eeeca181b.140039094		
000000014003905C	BB D3	mov edx,ebx		

At this point, the malware will try to find some thread in an alertable state, as this will allow it to make APC calls in order to execute code in the target process.

To find an alertable thread, it first obtains a handle for each thread in explorer.exe. It will then launch a call to NtQueueApcThread as NtSetEvent and wait for any of the threads to respond.

If it works correctly, it will obtain the first thread that answers the call and start with the injection.

5.3. Injection of shellcode in the target process

First, the malicious .dll makes a call to GlobalAddAtomW and creates a new Atom with the content it wishes to inject in the target process, in this case explorer.exe.

Second, the malicious .dll calls the NtQueueApcThread and sends as a parameter the function to be run by explorer.exe.

The first time this is done, it makes a call to memset to make sure that the zone where it will write the shellcode is at 0.

RAX	00000000774DC180	<ntdll.NtQueueApcThread>
RBX	0000000000000000	
RCX	00000000000000F8	'0'
RDX	00000000774DD910	<ntdll.memset>
RBP	00000000000000F8	'0'
RSP	00000000002CEB78	
RSI	00000000774DD910	<ntdll.memset>
RDI	00000000775CAA0	ntdll.00000000775CAA0
R8	00000000775CAA0	ntdll.00000000775CAA0
R9	0000000000000000	
R10	0000000000000080	
R11	00000000002CE678	"q7Uw"
R12	00000000775CAA0	ntdll.00000000775CAA0
R13	000000007758C5F0	ntdll.000000007758C5F0
R14	00000000773D6BF0	<kernel32.GlobalGetAtomNameA>
R15	0000000000000000	
RIP	00000000774DC180	<ntdll.NtQueueApcThread>

Figure 19. Memory wipe

It is important to indicate that the zone that Dridex has chosen for copying the shellcode is in ntdll as we can see in R8. This is because ntdll is always loaded on the same offset in all processes, regardless of the ASLR.

In the following iterations the function passed as parameter of NtQueueApcThread will be GlobalAtomGetAtomNameW, which will cause the target process to get the Atom that just created the malicious .dll and write it in the indicated zone, in such a way that it will write its contents inside the explorer.exe without raising suspicions.

First it will create an IAT for the shellcode.

Address	Address	Comments
00000000775CAA0	00000000774DBFB0	ntdll.NtMapViewOfSection
00000000775CAA8	00000000774DBFD0	ntdll.ZwUnmapViewOfSection
00000000775CAAB0	00000000774DBEB0	ntdll.ZwAllocateVirtualMemory
00000000775CAAB8	0000000077494AF0	ntdll.RtlCreateUserThread
00000000775CAAC0	00000000774DBE10	ntdll.NtSetEvent
00000000775CAAC8	00000000774D9110	ntdll.RtlCopyMemory
00000000775CAAD0	00000000774DC230	ntdll.ZwProtectVirtualMemory
00000000775CAAD8	00000000774DBE20	ntdll.NtClose
00000000775CAAE0	0000000000000CA0	
00000000775CAAE8	0000000000000408	
00000000775CAAF0	0000000000000C28	
00000000775CAAF8	000000000000091A	
00000000775CAB00	000000000000076C	
00000000775CAB08	0000000000000758	
00000000775CAB10	00000000773D6BF0	kernel32.GlobalGetAtomNameA
00000000775CAB18	0044894438EC8348	
00000000775CAB20	0000000000000000	
00000000775CAB28	0000000000000000	
00000000775CAB30	0000000000000000	
00000000775CAB38	0000000000000000	
00000000775CAB40	0000000000000000	

Figure 20. IAT creation in explorer.exe

And after several iterations it will copy the shellcode in explorer.exe completely.

```

000000007758C5F0 40 33      push rbp
000000007758C5F2 53        push rbx
000000007758C5F3 56        push rsi
000000007758C5F4 57        push rdi
000000007758C5F5 41 54      push r12
000000007758C5F7 41 56      push r14
000000007758C5F9 48 80 6C 24 D1 lea rbp,qword ptr ss:[rsp-2F]
000000007758C5FE 48 81 EC 98 00 00 sub rsp,98
000000007758C605 48 88 D9    mov rbx,rcx
000000007758C608 48 88 49 70 mov rcx,qword ptr ds:[rcx+70]
000000007758C60C 41 88 07 00 00 00 mov r8d,7
000000007758C612 48 80 53 78 lea rdx,qword ptr ds:[rbx+78]
000000007758C616 4C 89 45 77 mov qword ptr ss:[rbp+77],r8
000000007758C61A 48 89 40 6F mov qword ptr ss:[rbp+6F],rcx
000000007758C61E FF 53 28   call qword ptr ds:[rbx+28]
000000007758C621 48 8D 45 67 lea rax,qword ptr ss:[rbp+67]
000000007758C625 49 83 CE FF or r14,FFFFFFFFFFFFFFFF
000000007758C629 4C 8D 45 77 lea r8,qword ptr ss:[rbp+77]
000000007758C62D 48 8D 55 6F lea rdx,qword ptr ss:[rbp+6F]
000000007758C631 49 88 CE    mov rcx,r14
000000007758C634 41 89 20 00 00 00 mov r9d,20
000000007758C63A 48 89 44 24 20 mov qword ptr ss:[rsp+20],rax
000000007758C63F FF 53 30   call qword ptr ds:[rbx+30]
000000007758C642 33 F6      xor esi,esi
000000007758C644 48 8D 78 48 lea rdi,qword ptr ds:[rbx+48]
000000007758C648 45 8D 66 03 lea r12,dword ptr ds:[r14+3]
000000007758C64C 8B 07      mov eax,dword ptr ds:[rdi]
000000007758C64E 83 65 E7 00 and dword ptr ss:[rbp-19],0
000000007758C652 83 65 E8 00 and dword ptr ss:[rbp-15],0
000000007758C656 48 83 65 F7 00 and qword ptr ss:[rbp-9],0
000000007758C65B 48 83 65 7F 00 and qword ptr ss:[rbp+7F],0
000000007758C660 48 83 65 EF 00 and qword ptr ss:[rbp-11],0
000000007758C665 C7 44 24 48 04 00 00 mov dword ptr ss:[rsp+48],4
000000007758C66D 83 64 24 40 00 00 and dword ptr ss:[rsp+40],0
000000007758C672 C7 44 24 38 02 00 00 mov dword ptr ss:[rsp+38],2
000000007758C67A 48 8D 4D F7 lea rcx,qword ptr ss:[rbp-9]
000000007758C67E 4C 8D 45 7F lea r8,qword ptr ss:[rbp+7F]
000000007758C682 45 33 C9    xor r9d,r9d
000000007758C685 48 89 4C 24 30 mov qword ptr ss:[rsp+30],rcx
000000007758C68A 48 8D 4D E7 lea rcx,qword ptr ss:[rbp-19]
000000007758C68E 49 88 D6    mov rdx,r14
000000007758C691 48 89 4C 24 28 mov qword ptr ss:[rsp+28],rcx
000000007758C696 48 88 4F F8 mov rcx,qword ptr ds:[rdi-8]
000000007758C69A 48 89 45 77 mov qword ptr ss:[rbp+77],rax
000000007758C69E 48 89 44 24 20 mov qword ptr ss:[rsp+20],rax
000000007758C6A3 FF 13      call qword ptr ds:[rbx]
000000007758C6A5 4C 8D 4D 77 lea r9,qword ptr ss:[rbp+77]
000000007758C6A9 48 8D 55 EF lea rdx,qword ptr ss:[rbp-11]
000000007758C6AD 45 33 C0    xor r9d,r9d
000000007758C6B0 49 88 CE    mov rcx,r14
000000007758C6B3 C7 44 24 28 04 00 00 mov dword ptr ss:[rsp+28],4
000000007758C6B8 C7 44 24 20 00 10 00 mov dword ptr ss:[rsp+20],1000
000000007758C6C3 FF 53 10   call qword ptr ds:[rbx+10]
000000007758C6C6 48 8B 4D EF mov rcx,qword ptr ss:[rbp-11]
000000007758C6CA 44 8B 07    mov r8d,dword ptr ds:[rdi]
000000007758C6CD 48 8B 55 7F mov rdx,qword ptr ss:[rbp+7F]
000000007758C6D1 48 8D 45 FF lea rax,qword ptr ss:[rbp-11]
000000007758C6D5 4C 8D 5D 6F lea r11,qword ptr ss:[rbp+6F]
000000007758C6D9 85 F6      test esi,esi
000000007758C6DB 4C 0F 44 D8 cmovbe r11,rax
000000007758C6DF 49 89 08    mov qword ptr ds:[r11],rcx
000000007758C6E2 FF 53 28   call qword ptr ds:[rbx+28]
000000007758C6E5 48 8B 55 7F mov rdx,qword ptr ss:[rbp+7F]
000000007758C6E9 48 8B C4 14 mov rcx,r14
000000007758C6EC FF 53 08   call qword ptr ds:[rbx+8]
000000007758C6EF FC C6      inc esi
000000007758C6F1 48 83 C7 10 add rdi,10
000000007758C6F5 49 FF CC    dec r12
000000007758C6F8 0F 35 4E FF FF jmp rxd1!.7758C64C
000000007758C6FE 48 8D 45 67 lea rax,qword ptr ss:[rbp+67]
000000007758C702 45 8D 4C 24 20 lea r9d,qword ptr ds:[r12+20]
000000007758C707 4C 8D 45 77 lea r8,qword ptr ss:[rbp+77]
000000007758C70B 48 8D 55 6F lea rdx,qword ptr ss:[rbp+6F]
000000007758C70F 49 88 CE    mov rcx,r14
    
```

Figure 21. Shellcode in explorer.exe

5.4. Execution of the shellcode in the target process

Once the shellcode is copied to the explorer, it must be executed.

To do this, Dridex modifies the GlobalAtomGetAtomNameA function in the same way that it has injected the shellcode, using Atoms.

Original code of the function:

```

00000000773D68F0 48 83 EC 38      sub rsp,38
00000000773D68F4 44 89 44 24 20  mov dword ptr ss:[rsp+20],r8d
00000000773D68F9 4C 8B CA        mov r9,rax
00000000773D68FC 44 0F B7 C1    movzx r8d,cx
00000000773D6C00 33 C9         xor ecx,ecx
00000000773D6C02 33 D2         xor edx,edx
00000000773D6C04 E8 57 FC FF FF call kernel32.773D6860
00000000773D6C09 48 83 C4 38    add rsp,38
00000000773D6C0D C3           ret
00000000773D6C0E 90           nop
    
```

Figure 22. Original function

Here's how the function has been modified:

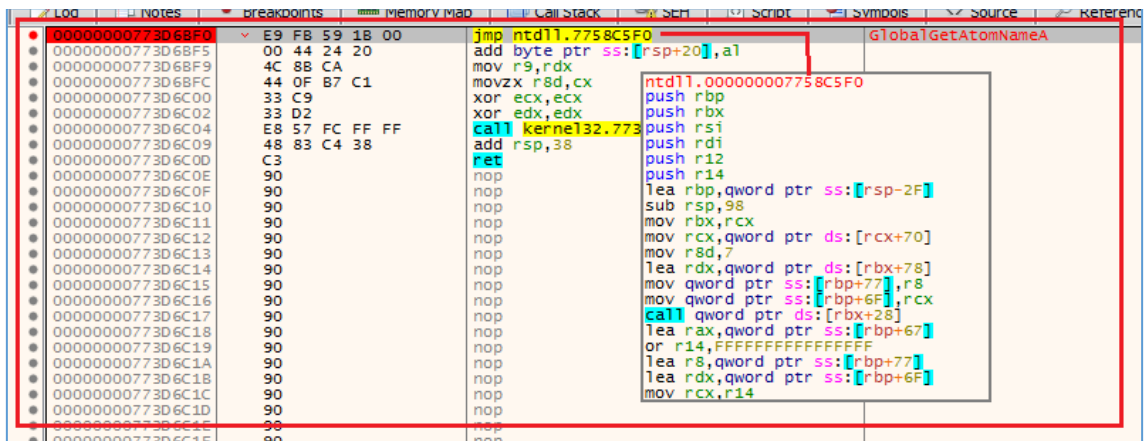


Figure 23. Modified function

As you can see, when you call GlobalAtomGetAtomNameA in explorer.exe the program will execute the shellcode.

After the modification, from the malicious .dll, a call will be made to GlobalAtomGetAtomNameA using NtQueueApcThread.

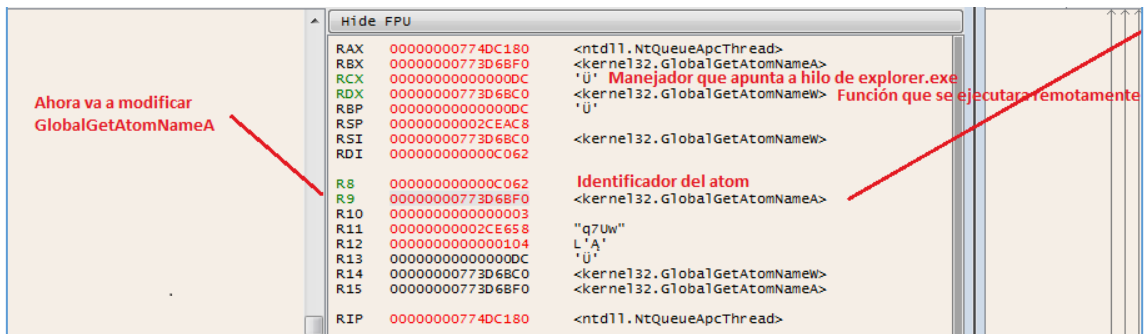


Figure 24. Remote execution of the shellcode

At this point the shellcode will start executing.

After this, GlobalAtomGetAtomNameA is returned to its original state, to avoid suspicion.

6. NETWORK CONNECTIONS

The Trojan, once it has been injected into the explorer.exe process, opens port 443 (usually used for the HTTPS protocol) and waits for some connection.

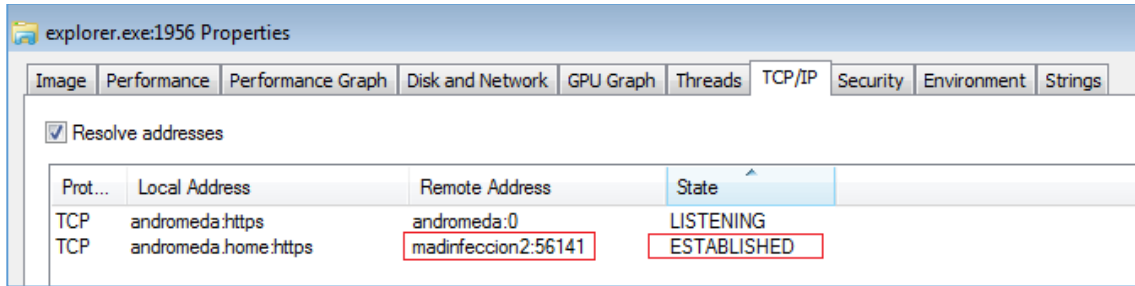


Figure 25. Port 443 opened

7. IOCs

To check if a computer has been compromised by this version of Dridex, the following points should be considered:

- The explorer.exe process has port 443 listening and there is a firewall rule in place allowing network traffic for that process.
- Directories that match the expression %SYSTEM%\[0-9] {4}, and contain a legitimate executable next to a .dll or .cpl file.
- Scheduled tasks that execute a file in path %SYSTEM%\[0-9] {4} in periods of 60 minutes.

8. REFERENCIAS

[1]	<p>Inside the Dridex Malware Takedown</p> <p>Link: http://www.bankinfosecurity.com/dridex-botnet-disruption-lessons-learned-a-8594</p>
[2]	<p>Dridex v4 - AtomBombing and other surprises</p> <p>Link: https://www.virusbulletin.com/conference/vb2017/abstracts/dridex-v4-atombombing-and-other-surprises/</p>
[3]	<p>Dridex Banking Malware Sample Technical Analysis and Solution</p> <p>Link: http://blog.nsfocus.net/dridex-banking-malware-sample-technical-analysis-solution/</p>