

Informe Malware

“Dridex versión 4”



SIN CLASIFICAR

Agosto de 2017

ÍNDICE

1. RESUMEN EJECUTIVO	3
2. CARACTERÍSTICAS DEL TROYANO	4
3. PROCEDIMIENTO DE INFECCIÓN.....	7
3.1. Vectores de Infección.....	7
3.2. Interacciones con el sistema afectado.....	7
4. PERSISTENCIA EN EL SISTEMA	8
5. INYECCIÓN POR ATOMBOMBING	11
5.1. Búsqueda del proceso objetivo	11
5.2. Búsqueda de hilos en estado alertable	11
5.3. Inyección de la shellcode en el proceso objetivo	12
5.4. Ejecución de la shellcode en el proceso objetivo	14
6. CONEXIONES DE RED	16
7. IOCS.....	17
8. REFERENCIAS	18

1. RESUMEN EJECUTIVO

El presente documento recoge el análisis de una nueva variante del código dañino "Dridex", en concreto la versión número cuatro.

Dridex es un famoso troyano bancario conocido por su sofisticación y capacidad de pasar desapercibido en los equipos infectados. Dichos equipos son incorporados a una *botnet* modular que les permite añadir nuevas características maliciosas, propias o externas, por medio de módulos o librerías (vendidas por separado).

La primera versión apareció a finales 2014; a principios del 2015 se lanzó una nueva gran actualización y se pasó a la segunda versión del troyano. Cuando se trata de las versiones principales de Dridex, la versión más estable y resistente hasta la fecha ha sido la tercera, que se lanzó en abril de 2015 y se ha utilizado en todas las campañas de ataque conocidas hasta la cuarta versión. La última versión conocida es la versión 4, objeto de este informe, y que fue encontrada por primera vez en febrero de 2017.

No se habían observado nuevas grandes actualizaciones de Dridex desde que se realizó el desmantelamiento de componentes clave de la botnet por agencias gubernamentales en 2015. [\[1\]](#)

Esta nueva variante del troyano bancario incorpora nuevas funcionalidades. Una de esas nuevas funcionalidades es el AtomBombing que tiene como objetivo realizar inyección de código sin llamar a APIs sospechosas para evitar ser detectado por los sistemas de monitorización. Por otra parte, incorpora también la técnica de DLL hijacking para obtener persistencia y se realizan varias optimizaciones en los métodos criptográficos usados para obtener la configuración. [\[2\]](#)

2. CARACTERISTICAS DEL TROYANO

A continuación se muestran algunas propiedades estáticas del fichero analizado.

El hash del troyano es el siguiente:

MD5	001fcf14529ac92a458836f7cec03896
SHA256	a6db7759c737cbf6335b6d77d43110044ec049e8d4cbf7fa9bd4087fa7e415c7

La fecha interna de creación de la muestra analizada es el 16 de Mayo de 2017. El fichero en cuestión se ha compilado para ser ejecutado en entornos de 64 bits y a la vez simula ser una dll legítima de Microsoft.

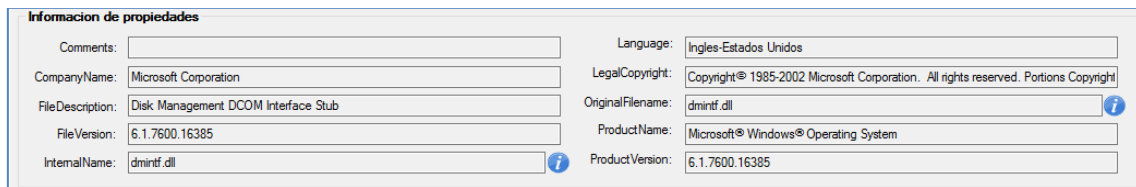


Ilustración 1. Propiedades del fichero

Además, se encuentra cifrado con un algoritmo propio de cara a evadir las detecciones de los antivirus.

Por otra parte, se ha observado que el ejecutable tiene una cantidad bastante elevada de secciones, 11 en total, tal y como se observa a continuación:

property	value	value	value	value	value	value	value	value	value	value	value
name	.text	.code	.sbss	.rdata	.data	.pdata	DATA	.rsrc	.reloc	.kwgrcd	
virtual-size	0x00000566 (1510)	0x00001AFC (6908)	0x00000657 (1623)	0x0001D0CB (122315)	0x00002F62 (12130)	0x000005FA (1530)	0x0002683F (158527)	0x0001C16E (115054)	0x000004C8 (1224)	0x0000056C (1388)	0x00000AE7 (2791)
virtual-address	0x00001000	0x00002000	0x00004000	0x00005000	0x00002000	0x00002000	0x00027000	0x0004E000	0x00068000	0x0006C000	0x0006D000
raw-size	0x00001000 (4096)	0x00002000 (8192)	0x00001000 (4096)	0x0001E000 (122880)	0x00003000 (12288)	0x00001000 (4096)	0x00027000 (159744)	0x0001D000 (118784)	0x00001000 (4096)	0x00001000 (4096)	0x00001000 (4096)
raw-data	0x00001000	0x00002000	0x00004000	0x00005000	0x00002000	0x00002000	0x00027000	0x0004E000	0x00068000	0x0006C000	0x0006D000
PointerToRelocations	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
PointerToLinenumbers	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
NumberOfRelocations	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
NumberOfLinenumbers	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
md5	A307CFBD21EE5...	80A66726FB860EAD...	A388B27F841B4F...	258595205F0C58FB...	B3AF0999C1DCB41...	3848DA66328342...	D52A19803FCF1B5...	A6F51B37F58FA376...	3ECC4FD93DACC...	CB1AE24217F540...	620F0867A91F7F74151B...
cape	0x00000A1A (2586)	0x00000504 (1284)	0x000009A9 (2473)	0x00000235 (565)	0x0000009E (158)	0x00000A06 (2566)	0x000004C1 (1217)	0x00000E92 (3730)	0x00000838 (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
writable	-	-	-	-	-	-	-	-	-	-	-
executable	x	x	x	-	-	-	-	-	-	-	-
shareable	-	-	-	-	-	-	-	-	-	-	-
discardable	-	-	-	-	-	-	-	-	-	-	-
cacheable	x	x	x	x	x	x	x	x	x	x	x
pageable	x	x	x	x	x	x	x	x	x	x	x
initialized-data	-	-	-	x	x	x	x	x	x	x	x
uninitialized-data	-	-	-	-	-	-	-	-	-	-	-

Ilustración 2. Información estática del binario analizado

En la sección DATA podemos observar que la entropía es de 7.799 y tiene un tamaño bastante grande. En dicha sección es donde se encuentra el binario altamente cifrado y empaquetado el cual una vez descifrado genera el código dañino real.

En la primera capa de descifrado, el ejecutable aloja memoria en el proceso, a continuación copia el código a ejecutar y por último lo llama y lo ejecuta tal y cómo vemos a continuación:

```

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
    
```

Ilustración 3. Salto a la shellcode

Lo primero que hace dicho código es obtener las direcciones de las funciones que va a utilizar más adelante, esto lo realizará de forma dinámica buscando en las librerías cargadas por el programa.

Para llevar a cabo esta tarea recorre la estructura PEB_LDR_DATA y las estructuras LDR_MODULE para encontrar la dirección base de las dlls cargadas. A continuación, accede al offset de la export table, para recorrer todas las funciones exportadas por la dll y encontrar la dirección en memoria de la función buscada.

mov r9,qword ptr gs:[30]	Acceso al TEB
mov rax,qword ptr ds:[r9+60]	Acceso al PEB
mov rax,qword ptr ds:[rax+18]	Acceso al PEB_LDR_DATA
mov r9,rax	
add r9,20	
mov rax,qword ptr ds:[rax+20]	Acceso a la lista InMemoryOrderList
cmp rax,r9	

Ilustración 4. Enumeración de módulos cargados

Por otra parte, la shellcode comprueba si se ha realizado un hook en la función no documentada LdrLoadDll, accediendo a su dirección y comprobando si el primer bytes es igual a E9 que equivale a un jmp en ensamblador.

48 81 EC D8 00 00 00	sub rsp,D8
48 80 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 08 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 12204B

Ilustración 5. Comprobación Hooks

En caso de que la comprobación anterior sea exitosa, va a proceder a desmapear de la memoria del proceso la dll con nombre "snxhk.dll" que se trata de una librería de Avast y AVG que realiza hooks para monitorizar los procesos en su 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

Ilustración 6. Librería snxhk.dll

Por último, la shellcode descifra el ejecutable que se encuentra en la sección DATA en memoria, lo copia en la dirección de la imagen base y pasa la ejecución a ese nuevo ejecutable.

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

Ilustración 7. Ejecutable descifrado

A modo de resumen, en la imagen que se adjunta a continuación se puede ver todo el proceso completo de desempaquetado de la muestra de un modo más gráfico y esquemático.

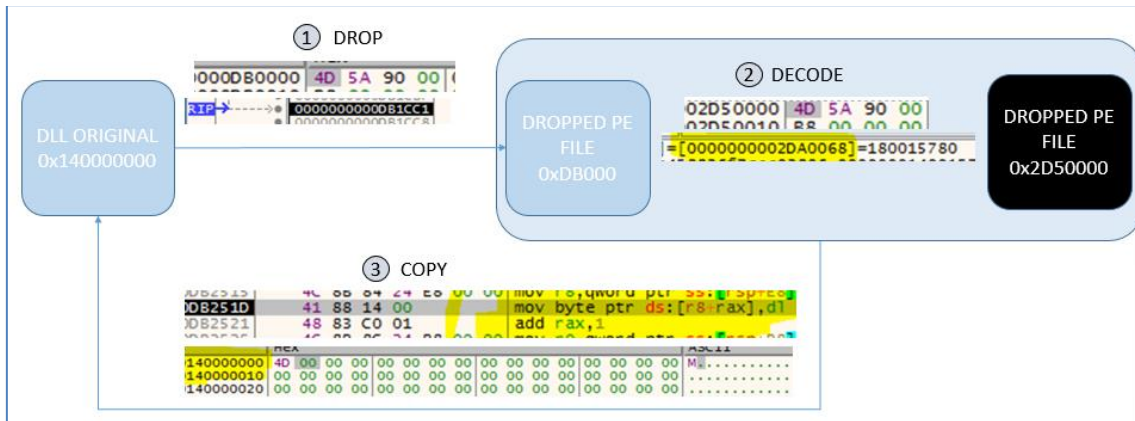


Ilustración 8. Proceso completo de desempaquetado.

3. PROCEDIMIENTO DE INFECCIÓN

3.1. Vectores de Infección

La infección en el equipo no está clara, podría proceder de alguna campaña vía exploit kit o spam.

3.2. Interacciones con el sistema afectado

Una vez ejecutado, el troyano procederá a comprobar si es la única instancia del malware ejecutandose en el equipo, además comprueba si ya se encuentra inyectado en el proceso explorer.exe.

Todo eso lo realiza mediante la creación y apertura de un mutex creado por él mismo. Para ello, primero concatena el nombre del equipo y el del usuario y calcula su hash MD5.

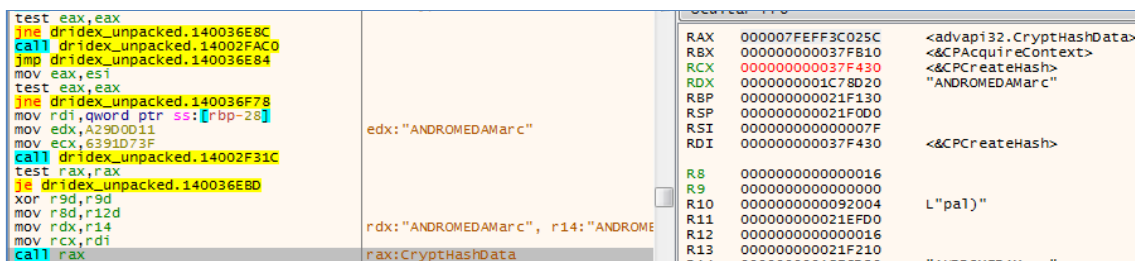


Ilustración 9. Cálculo del hash

A continuación, le añade corchetes al principio y al final y lo separa por guiones similar a un Objeto COM.

Type	Name
Desktop	\Default
Directory	\KnownDlls
Directory	\Sessions\1\BaseNamedObjects
File	C:\Users\Marc\Desktop
File	C:\Windows\System32\es-ES\setupapi.dll.mui
File	\Device\KsecDD
Key	HKLM\SOFTWARE\MICROSOFT\WINDOWS NT\CURRENTVERSION\Image File Execution Options
Key	HKLM\SYSTEM\ControlSet001\Control\Nls\Sorting\Versions
Key	HKLM\SYSTEM\ControlSet001\Control\SESSION MANAGER
Key	HKLM
Mutant	\Sessions\1\BaseNamedObjects\[74460520-c6c9-3965-7db5-887562f86d16]
Thread	DLLLoader64_E484.exe(2808): 3560
WindowStation	\Sessions\1\Windows\WindowStations\WinSta0
WindowStation	\Sessions\1\Windows\WindowStations\WinSta0

Ilustración 10. Mutex creado en el sistema

A partir de dicho algoritmo sería posible desarrollar una vacuna que cree dichos Mutex en los sistemas para evitar la infección por Dridex.

En el caso de que el malware no esté en ejecución, este creará una carpeta en %WINDOWS%\system32\[0-9]{4}

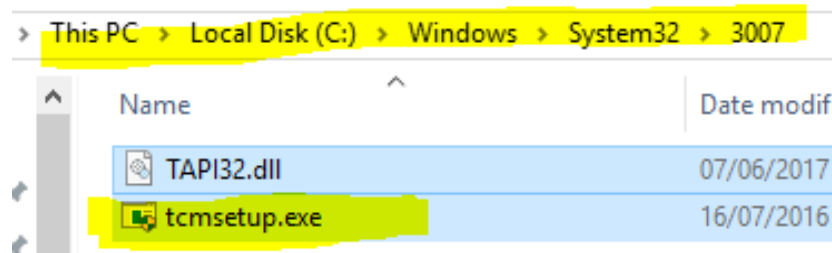


Ilustración 11. Carpeta creada

En el interior de esta carpeta copiará un .exe legítimo junto con una .dll o .cpl asociada al mismo, pero en este caso no es legítima, se trata del troyano, de tal forma que al ejecutar el .exe de la carpeta cargará la dll o cpl maliciosa mediante la técnica conocida como dll hijacking.

Además creará una tarea programada con un nombre aleatorio, en el ejemplo "Domitxtdoi", que se ejecutará cada 60 minutos.

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

Ilustración 12. Tarea creada

En este ejemplo, vemos como ejecutará tcmsetup.exe para que así se cargue la dll maliciosa TAPI32.dll que comienza de nuevo todo el proceso de infección.

Después de crear la tarea programada lanza una serie de comandos:

Creación de una regla en el firewall para explorer.exe, que es donde se va a inyectar.

```
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
```

Creación de la tarea que ejecutará cada 60 minutos la dll maliciosa.

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

Durante este proceso, la dll maliciosa se habrá inyectado en el proceso explorer.exe mediante la técnica de AtomBombing y quedará a la espera de que el usuario abra un navegador como Internet Explorer, Firefox, Chrome, etc.

En el momento que el usuario abra un navegador, se inyectará una nueva shellcode desde explorer.exe al navegador usando la misma técnica de AtomBombing.

4. PERSISTENCIA EN EL SISTEMA

El código dañino, para asegurar su persistencia en el sistema, realiza las siguientes acciones:

Crea una carpeta con 4 numeros aleatorios en C:\Windows\System32 y dentro de esa carpeta copia un ejecutable legitimo (no es siempre el mismo) de Windows y una DLL que sabe que sera cargada por el ejecutable legitimo, esta DLL está modificada con codigo dañino.

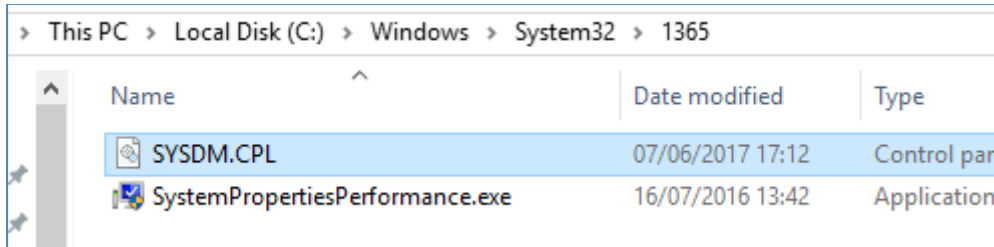


Ilustración 13. Persistencia en el sistema

Esta tecnica es conocida como DLL hijacking la cual se aprovecha del orden en el que el sistema busca las librerías/ficheros que va a cargar/utilizar. En el caso de la imagen de arriba, el ejecutable "SystemPropertiesPerformance.exe" cargará "SYSDM.CPL" entre otras librerías. Por definición en el primer sitio que buscará el fichero "SYSDM.CPL" para cargarlo es en directorio en el que se está ejecutando la aplicación, en este caso C:\Windows\System32\1365, de no encontrarlo, lo buscará en otras rutas dependiendo de cómo esté establecido el orden de búsqueda de dlls en el sistema.

Por lo tanto, el objetivo de Dridex al copiar en el mismo directorio un ejecutable y una dll modificada es el de levantar menos sospechas ya que las acciones maliciosas se realizan desde un programa legítimo.

Para ejecutar el fichero que copia crea una tarea programada para que se ejecute cada hora el fichero que ha copiado en la carpeta de números aleatorios (C:\Windows\System32\1365), como hemos indicado en el apartado anterior.

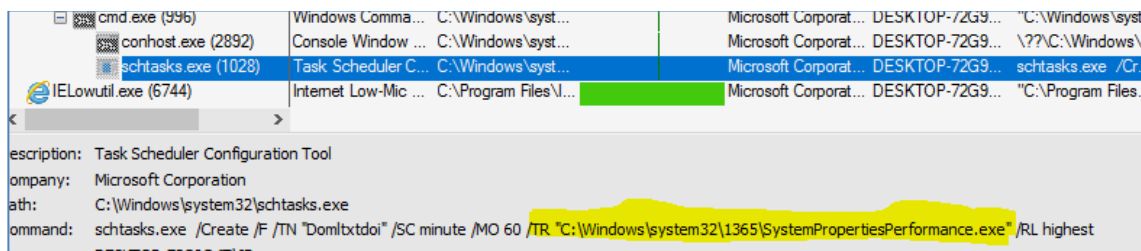


Ilustración 14. Creación de la tarea programada

Como ya se ha dicho, la carpeta que crea se compone de 4 números aleatorios, y el ejecutable que crea no siempre es el mismo, al igual que la DLL, por lo que en todo momento es conocedor de qué ejecutable carga qué librería y es capaz de modificar dicha librería con código dañino.

Si analizamos un poco más esta parte vemos que actúa de la siguiente forma:

1. Enumera todos los ejecutables de la carpeta "C:\Windows\System32\"

2. "Hashea" el nombre de cada ejecutable y lo compara con un valor que tiene guardado, si coincide se queda con ese ejecutable (en cada ejecución ese hash es distinto).
3. Lee las IAT del ejecutable seleccionado y de ahí elige una DLL para posteriormente hacer el DLL hijacking.
4. Lee la IAT de la DLL seleccionada en el **punto 3**.
5. Hace una copia del código malicioso (la propia DLL) y añade una sección al final con nombre aleatorio para copiar la IAT obtenida en el **punto 4**.
6. Copia tanto el ejecutable seleccionado (**3**) como la dll maliciosa modificada (**5**) en una carpeta aleatoria.

De esta forma consigue persistencia en el sistema y cada vez que se ejecute ese fichero cargará la DLL maliciosa.

Además el malware creará en la carpeta AppData\Roaming\[nombre carpeta aleatorio] una copia del malware en formato ejecutable junto con una clave de registro en "HKCU\Software\Microsoft\Windows\CurrentVersion\Run".

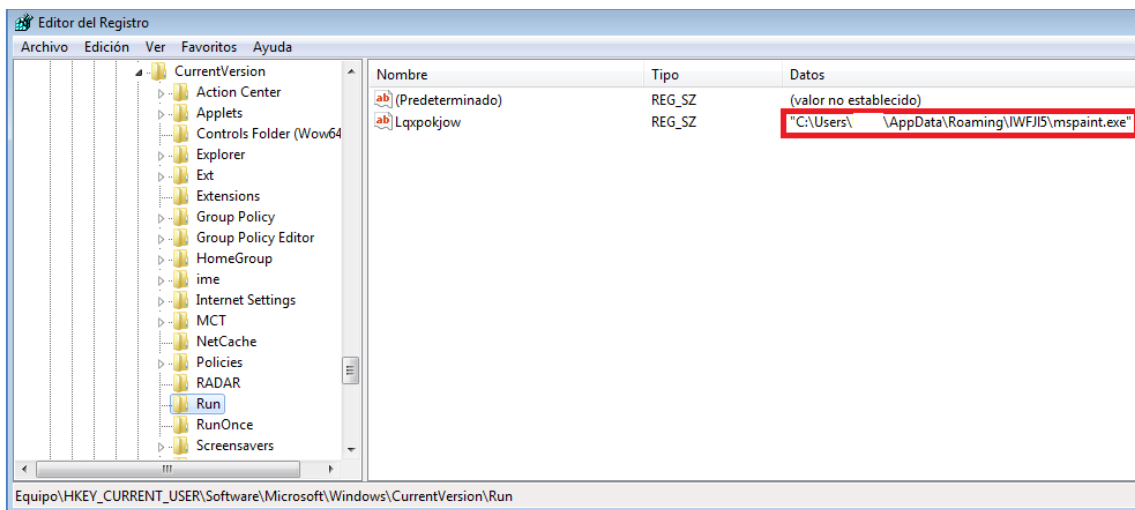


Ilustración 15. Clave de registro

5. INYECCIÓN POR ATOMBOMBING

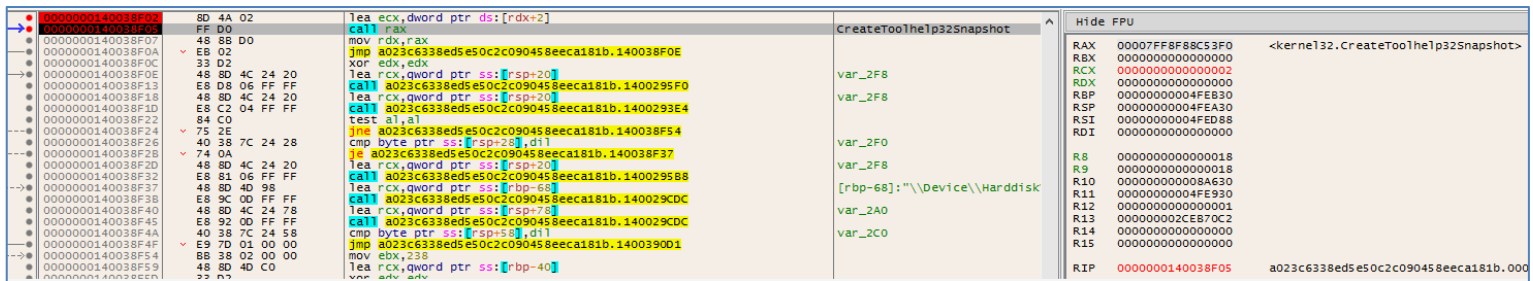
La técnica de AtomBombing es la utilizada por Dridex para escribir una shellcode en otros procesos sin levantar sospechas.

Esto lo hará mediante llamadas APC y uno de los Windows Executive Object más usado por Windows que son los Atoms.

A continuación se detallan las distintas fases para llegar a inyectarse en otro proceso.

5.1. Búsqueda del proceso objetivo

El proceso objetivo en este caso es explorer.exe y para inyectarse en él, primero debe de acceder a él, por lo que realiza una enumeración de procesos, haciendo uso de funciones como:



Una vez encuentra el proceso explorer.exe hace una llamada a la función OpenProcess para empezar a enumerar los hilos en estado alertable.

5.2. Búsqueda de hilos en estado alertable

En este punto, el malware intentará encontrar algún hilo en estado alertable, ya que gracias a ello podrá hacer llamadas APC con el objetivo de ejecutar código en el

Ilustración 16. Creación del snapshot

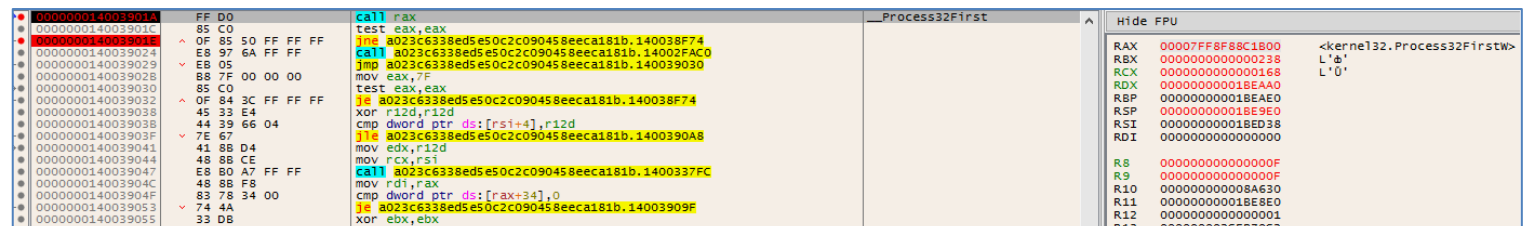


Ilustración 17. Enumeración del primer proceso

proceso objetivo.

Para encontrar un hilo en estado alertable, primero obtendrá un manejador por cada hilo de explorer.exe. Después lanzará una llamada a NtQueueApcThread pasándole como función NtSetEvent y esperará a que alguno de los hilos responda.

En caso de que funcione correctamente, obtendrá el primer hilo que responda a la llamada y comenzará con la inyección.

5.3. Inyección de la shellcode en el proceso objetivo

1º La dll maliciosa hace una llamada a GlobalAddAtomW y crea un nuevo Atom con el contenido que le interesa inyectar en el proceso objetivo, en este caso explorer.exe.

2º La dll maliciosa hace una llamada a NtQueueApcThread pasándole como parámetros la función que quiere que ejecute el explorer.exe.

La primera vez que esto se realiza, hace una llamada a memset para asegurarse de que la zona donde va a escribir la shellcode está a 0.

RAX	00000000774DC180	<ntdll.NtQueueApcThread>
RBX	0000000000000000	
RCX	00000000000000F8	'0'
RDX	00000000774DD910	<ntdll.memset>
RBP	00000000000000F8	'0'
RSP	00000000002CEB78	
RSI	00000000774DD910	<ntdll.memset>
RDI	00000000775CAAA0	ntdll.00000000775CAAA0
R8	00000000775CAAA0	ntdll.00000000775CAAA0
R9	0000000000000000	
R10	0000000000000080	
R11	00000000002CE678	"q7Uw"
R12	00000000775CAAA0	ntdll.00000000775CAAA0
R13	000000007758C5F0	ntdll.000000007758C5F0
R14	00000000773D6BF0	<kernel32.GlobalGetAtomNameA>
R15	0000000000000000	
RIP	00000000774DC180	<ntdll.NtQueueApcThread>

Ilustración 19. Borrado de memoria

Es importante indicar que la zona que ha elegido Dridex para realizar la copia de la shellcode es en ntdll como podemos ver en R8. Esto es porque ntdll siempre se carga en el mismo offset en todos los procesos, independientemente del ASLR.

En las siguientes iteraciones la función pasada como parámetro de NtQueueApcThread será GlobalAtomGetAtomNameW, que hará que el proceso objetivo obtenga el Atom que acaba de crear la dll maliciosa y lo escriba en la zona indicada, de tal forma que escribirá su contenido dentro del explorer.exe sin levantar sospechas.

Primero creará una IAT para la shellcode.

Address	Address	Comments
00000000775CAA0	00000000774DBF80	ntd11.NtMapViewOfSection
00000000775CAA8	00000000774DBFD0	ntd11.ZwUnmapViewOfSection
00000000775CAAB0	00000000774DBEB0	ntd11.ZwAllocateVirtualMemory
00000000775CAAB8	0000000077494AF0	ntd11.RtlCreateUserThread
00000000775CAAC0	00000000774DBE10	ntd11.NtSetEvent
00000000775CAAC8	00000000774D9110	ntd11.RtlCopyMemory
00000000775CAAD0	00000000774DC230	ntd11.ZwProtectVirtualMemory
00000000775CAAD8	00000000774DBE20	ntd11.NtClose
00000000775CAAE0	00000000000000CA0	
00000000775CAAE8	00000000000000408	
00000000775CAAF0	00000000000000C28	
00000000775CAAF8	0000000000000091A	
00000000775CAB00	0000000000000076C	
00000000775CAB08	00000000000000758	
00000000775CAB10	00000000773D6BF0	kernel32.GlobalGetAtomNameA
00000000775CAB18	0044894438EC8348	
00000000775CAB20	00000000000000000	
00000000775CAB28	00000000000000000	
00000000775CAB30	00000000000000000	
00000000775CAB38	00000000000000000	

Ilustración 20. Creación IAT en el explorer.exe

Y después de varias iteraciones copiará la shellcode en explorer.exe por completo.

000000007758CF0	50 59	push rbp	
000000007758CF2	53	push rbx	
000000007758CF3	56	push rsi	rsi:"Top of worker loop\n"
000000007758CF4	57	push rdi	
000000007758CF5	41 54	push r12	
000000007758CF7	41 56	push r14	r14:"wait completed with STATUS_USER_APC\n"
000000007758CF9	48 8D 6C 24 01	lea rbp,qword ptr ss:[rsp-2F]	
000000007758CFE	48 81 EC 98 00 00 00	sub rsp,98	
000000007758C05	48 88 D9	mov rbx,rcx	
000000007758C08	48 88 49 70	mov rcx,qword ptr ds:[rcx+70]	
000000007758C0C	41 88 07 00 00 00	mov r8d,0	
000000007758C12	48 8D 53 78	lea rdx,qword ptr ds:[rbx+78]	
000000007758C16	4C 89 45 77	mov qword ptr ss:[rbp+77],r8	
000000007758C1A	48 89 40 6F	mov qword ptr ss:[rbp+6F],rcx	
000000007758C1E	FF 53 28	call qword ptr ds:[rbx+28]	
000000007758C21	48 8D 45 67	lea rax,qword ptr ss:[rbp+67]	
000000007758C25	49 83 CE FF	or r14,FFFFFFFFFFFFFFF	r14:"wait completed with STATUS_USER_APC\n"
000000007758C29	4C 8D 45 77	lea r8,qword ptr ss:[rbp+77]	
000000007758C2D	48 8D 55 6F	lea rdx,qword ptr ss:[rbp+6F]	
000000007758C31	49 88 CE	mov rcx,r14	r14:"wait completed with STATUS_USER_APC\n"
000000007758C34	41 89 20 00 00 00	mov r9d,20	
000000007758C3A	48 89 44 24 20	mov qword ptr ss:[rsp+20],rax	
000000007758C3F	FF 53 30	call qword ptr ds:[rbx+30]	
000000007758C42	33 F6	xor esi,esi	esi:"Top of worker loop\n"
000000007758C44	48 8D 78 48	lea rdi,qword ptr ds:[rbx+48]	
000000007758C48	45 8D 66 03	lea r12,dword ptr ds:[r14+3]	r14+3:"t completed with STATUS_USER_APC\n"
000000007758C4C	8B 07	mov eax,dword ptr ds:[rdi]	
000000007758C4E	83 65 E7 00	and dword ptr ss:[rbp-19],0	
000000007758C52	83 65 E8 00	and dword ptr ss:[rbp-15],0	
000000007758C56	48 83 65 F7 00	and qword ptr ss:[rbp-9],0	
000000007758C5B	48 83 65 F7 00	and qword ptr ss:[rbp-9],0	
000000007758C60	48 83 65 EF 00	and qword ptr ss:[rbp-11],0	
000000007758C65	C7 44 24 48 04 00 00	mov dword ptr ss:[rsp+48],4	
000000007758C6D	83 64 24 40 00 00	and dword ptr ss:[rsp+40],0	
000000007758C72	C7 44 24 38 02 00 00	mov dword ptr ss:[rsp+38],2	
000000007758C7A	48 8D 40 F7	lea rcx,qword ptr ss:[rbp-9]	
000000007758C7E	4C 8D 45 7F	lea r8,qword ptr ss:[rbp+7F]	
000000007758C82	45 33 C9	xor r9d,r9d	
000000007758C85	48 89 4C 24 30	mov qword ptr ss:[rsp+30],rcx	
000000007758C8A	48 8D 4D E7	lea rcx,qword ptr ss:[rbp-19]	
000000007758C8E	49 8B D6	mov rdx,r14	r14:"wait completed with STATUS_USER_APC\n"
000000007758C91	48 89 4C 24 28	mov qword ptr ss:[rsp+28],rcx	
000000007758C96	48 88 4F F8	mov rcx,qword ptr ds:[rdi-8]	
000000007758C9A	48 89 45 77	mov qword ptr ss:[rbp+77],rax	
000000007758C9E	48 89 44 24 20	mov qword ptr ss:[rsp+20],rax	
000000007758CA3	FF 13	call qword ptr ds:[rbx]	
000000007758CA5	4C 8D 40 77	lea r9,qword ptr ss:[rbp+77]	
000000007758CA9	48 8D 55 EF	lea rdx,qword ptr ss:[rbp-11]	
000000007758CAD	45 33 C0	xor r8d,r8d	
000000007758CB0	49 88 CE	mov rcx,r14	r14:"wait completed with STATUS_USER_APC\n"
000000007758CB3	C7 44 24 28 04 00 00	mov dword ptr ss:[rsp+28],4	
000000007758CB8	C7 44 24 20 00 10 00	mov dword ptr ss:[rsp+20],1000	
000000007758CC3	FF 53 10	call qword ptr ds:[rbx+10]	
000000007758CC6	48 8B 4D EF	mov rcx,qword ptr ss:[rbp-11]	
000000007758CCA	44 8B 07	mov r8d,dword ptr ds:[rdi]	
000000007758CCD	48 8B 55 7F	mov rdx,qword ptr ss:[rbp+7F]	
000000007758CD1	48 8D 45 FF	lea rax,qword ptr ss:[rbp-1]	
000000007758CD5	4C 8D 5D 6F	lea r11,qword ptr ss:[rbp+6F]	
000000007758CD9	85 F6	test esi,esi	esi:"Top of worker loop\n"
000000007758CDB	4C 0F 44 D8	cmovbe r11,rax	
000000007758CDE	49 89 08	mov qword ptr ds:[r11],rcx	
000000007758CE2	FF 53 28	call qword ptr ds:[rbx+28]	
000000007758CE5	48 8B 55 7F	mov rdx,qword ptr ss:[rbp+7F]	
000000007758CE9	49 88 CE	mov rcx,r14	r14:"wait completed with STATUS_USER_APC\n"
000000007758CEC	FF 53 08	call qword ptr ds:[rbx+8]	
000000007758CEF	FF C6	inc esi	esi:"Top of worker loop\n"
000000007758CF1	48 83 C7 10	add rdi,10	
000000007758CF5	49 FF CC	dec r12	
000000007758CF8	OF 85 4E FF FF FF	jne ntd11.7758C64C	
000000007758CFE	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 8B CE	mov rcx,r14	r14:"wait completed with STATUS_USER_APC\n"

Ilustración 21. Shellcode en explorer.exe

5.4. Ejecución de la shellcode en el proceso objetivo

Una vez la shellcode está copiada en el explorer, esta se debe de ejecutar.

Para ello Drindex modifica la función GlobalAtomGetAtomNameA del mismo modo que ha inyectado la shellcode, usando los Atoms.

Código original de la función:

00000000773D68F0	48 83 EC 38	sub rsp,38	GlobalAtomGetAtomNameA
00000000773D68F4	44 89 44 24 20	mov dword ptr ss:[rsp+20],r8d	
00000000773D68F9	4C 8B CA	mov r9,rdx	
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	

Ilustración 22. Función original

A continuación, se puede ver cómo la función ha sido modificada:

00000000773D68F0	E9 FB 59 18 00	jmp ntd11.7758C5F0	GlobalAtomGetAtomNameA
00000000773D68F5	00 44 24 20	add byte ptr ss:[rsp+20],al	
00000000773D68F9	4C 8B CA	mov r9,rdx	
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	
00000000773D6C0F	90	nop	
00000000773D6C10	90	nop	
00000000773D6C11	90	nop	
00000000773D6C12	90	nop	
00000000773D6C13	90	nop	
00000000773D6C14	90	nop	
00000000773D6C15	90	nop	
00000000773D6C16	90	nop	
00000000773D6C17	90	nop	
00000000773D6C18	90	nop	
00000000773D6C19	90	nop	
00000000773D6C1A	90	nop	
00000000773D6C1B	90	nop	
00000000773D6C1C	90	nop	
00000000773D6C1D	90	nop	
00000000773D6C1E	90	nop	
00000000773D6C1F	90	nop	

Ilustración 23. Función modificada

Como se puede observar, cuando se llame a GlobalAtomGetAtomNameA en explorer.exe el programa ejecutará la shellcode.

Después de realizar la modificación, desde la dll maliciosa se hará una llamada a GlobalAtomGetAtomNameA usando NtQueueApcThread.

Hide FPU	
RAX	00000000774DC180 <ntd11.NtQueueApcThread>
RBX	00000000773D68F0 <kernel32.GlobalAtomNameA>
RCX	00000000000000DC 'ü' Manejador que apunta a hilo de explorer.exe
RDX	00000000773D68C0 <kernel32.GlobalAtomNameA> Función que se ejecutará remotamente
RBP	00000000000000DC 'ü'
RSP	0000000002CEAC8
RSI	00000000773D68C0 <kernel32.GlobalAtomNameA>
RDI	000000000000C062
R8	000000000000C062 Identificador del atom
R9	00000000773D68F0 <kernel32.GlobalAtomNameA>
R10	0000000000000003
R11	00000000002CE658 "q7Uw"
R12	0000000000000104 L'A'
R13	00000000000000DC 'ü'
R14	00000000773D68C0 <kernel32.GlobalAtomNameA>
R15	00000000773D68F0 <kernel32.GlobalAtomNameA>
RIP	00000000774DC180 <ntd11.NtQueueApcThread>

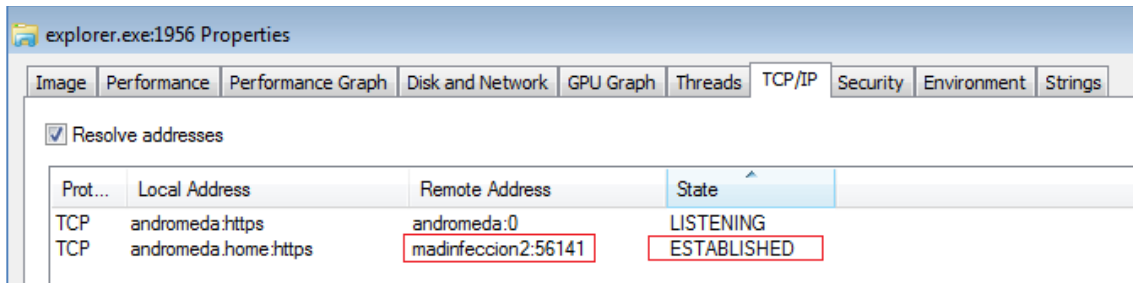
Ilustración 24. Ejecución remota de la shellcode

En este momento la shellcode comenzará su ejecución.

Después de esto, se devuelve GlobalAtomGetAtomNameA a su estado original, para evitar sospechas.

6. CONEXIONES DE RED

El troyano, una vez que se ha inyectado en el proceso explorer.exe, abre el puerto 443 (normalmente utilizado para el protocolo HTTPS) y se queda a la espera de alguna conexión.



Prot...	Local Address	Remote Address	State
TCP	andromeda.https	andromeda:0	LISTENING
TCP	andromeda.home.https	madinfeccion2:56141	ESTABLISHED

Ilustración 25. Puerto 443 abierto

7. IOCs

Para comprobar si un equipo puede estar comprometido por esta versión de Dridex se deberán revisar los siguientes puntos:

- El proceso explorer.exe tiene el puerto 443 a la escucha y existe una regla de firewall que permite tráfico de red para ese proceso.
- Directorios que coincidan con la expresión %SYSTEM%\[0-9]{4}, y que contengan un ejecutable legítimo junto a una .dll o .cpl.
- Tareas programadas que cada 60 minutos ejecuten un fichero en la ruta %SYSTEM%\[0-9]{4}.

8. REFERENCIAS

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

