

# Informe

Sergio Roselló Morell

## Puesta a punto del ordenador con Ubuntu

- Se han actualizado las dependencias y programas instalados en Ubuntu.
- Se ha visto que Ubuntu viene ya con los programas: Wireshark y Tshark
- Se ha descargado y configurado INetSim
- Se descarga apache2 para poder mantener el ordenador Windows permanentemente desconectado de Internet.

## Puesta a punto del ordenador con Windows (Primer análisis)

- Se ha descargado ProcessMonitor
- Se ha descargado PPEE
- Se ha descargado Python, para poder usar Noriben
- Se ha descargado Noriben, para mantener un registro de ProcessMonitor

## Puesta a punto del ordenador con Windows (Segundo análisis)

Se ha descargado e instalado el ejecutable `FLARE-vm` de fireeye

## Transferencia de archivos desde Ubuntu a Windows

Siguiendo las buenas practicas de un entorno de pruebas de malware, el ordenador victima del virus no debe estar conectado a Internet desde que se acaba de configurar (Esto es, instalar los programas necesarios para el análisis y la monitorización del malware)

Para pasar los virus desde el ordenador Ubuntu al Windows, se ha usado el servidor apache2, en el que le creamos un directorio en `/var/www/html/` llamado `malware` para subir los binarios maliciosos. Desde Windows, y con la interfaz de red de Internet desconectada de Ubuntu, nos conectamos a `192.168.10.10/malware` para descargar el malware.

## Comandos usados para iniciar apache2 e INetSim

Ambos programas se gestionan desde `systemctl`. Este programa es un gestor de servicios. Los servicios son programas, que puede gestionar `systemctl` para iniciar automáticamente al inicio del ordenador. Para iniciar tanto apache2 como INetSim, el comando que se ha usado es:

```
systemctl start <programa>.service
```

En este caso, al ser uno un servidor y otro un simulador del mismo, no pueden ejecutarse a la vez, porque por defecto, usan el mismo puerto.

Esto ha supuesto que se ha tenido que deshabilitar el inicio automático de ambos:

```
systemctl disable <programa>.service
```

Para que no salten errores.

En el momento en el que se quería iniciar un análisis del programa malicioso, se deshabilitaba `apache2` para activar `INetSim` Y viceversa cuando se dejaba de analizar el malware.

## Copia del estado de los dos ordenadores

En este momento, se ha hecho una copia de los ordenadores para poder volver al estado original después del análisis. De esta forma, si nuestro análisis dinámico cambia o rompe nuestro sistema operativo, podemos volver al estado anterior de la ejecución.

Es importante hacer la copia también del ordenador que esta corriendo `INetSim`, ya que, puede que escape el simulador de alguna forma y nos infecte también este equipo.

De esta manera y manteniendo siempre la red al exterior desconectada, nos podemos asegurar un banco de pruebas correcto.

## Descripciones de las herramientas empleadas:

### Radare2

Es la herramienta que se ha usado al analizar el ejecutable. Realiza las mismas funciones que `IDA PRO` o `x64db`. La diferencia mas notoria que tiene es que es basado en línea de comandos. Esto tiene varias ventajas y desventajas. Las ventajas son que es ligero, cuando se domina, el análisis se hace mucho mas sencillo, porque tiene la capacidad de ir al grano. Ademas de hacer análisis estático, también tiene una versión de compilador, esto quiere decir que también se puede usar como herramienta para hacer análisis dinámico.

### INetSim

Es un servicio que permite al analista analizar el malware sin miedo a que el virus se conecte con su servidor `CA`. Hace esto porque tiene una serie de respuestas predeterminadas para cada protocolo que soporta. De esta forma, cuando el malware se intenta conectar con su servidor `CA`, al pasar por `INetSim`, este directamente devuelve una respuesta adecuada para cada protocolo.

## Process Monitor con Noriben

Este es un entorno de desarrollo creado alrededor de ProcessMonitor. Sirve para centrar el estudio del malware en lo realmente importante. Hace esto mediante el uso de Whitelists. Además, puede automatizar el proceso de búsqueda de programas maliciosos que por ejemplo, tienen largos periodos de inactividad. Hace esto porque registra la actividad de los procesos, de forma que informa aunque el analista no este activamente analizando el malware.

## Process Hacker

Es una herramienta diseñada específicamente para analizar detalladamente el comportamiento de Windows. Esto quiere decir, que podemos ver todos los programas y procesos que están siendo usados por el sistema operativo en un mismo instante. La ventaja de esta herramienta es que además de ver los procesos, puedes analizar las peticiones red que hace cada proceso, los hilos que tiene el mismo y, en definitiva, desgranar el proceso a todos los niveles.

## Análisis estático con PPEE:

Durante el análisis estático, hemos encontrado varias pruebas que nos hacen pensar que es malware es una shell reversa. Mas adelante, describimos las pruebas que nos hacen llegar a dicha teoría.

### Cabeceras

Si analizamos las cabeceras con PPEE, vemos que automáticamente detecta que es un ejecutable para la arquitectura de 32 bits. Además, se nos describe el punto de entrada del ejecutable. En este caso, esta en la dirección virtual 000012D0.

- En la cabecera *DOS Header* aparecen:
  - Mimbres Value Comen
  - *Magic*: 5A4D MZ
- En la cabecera *Optional Headers* aparecen:
  - *Magic*: 010B PE32
  - *AddressOfEntryPoint*: 000012D0 .text

Podemos saber que el programa no ha sido comprimido de forma maliciosa porque el `virtualSize` y el `RawSize` son muy similares.

- En la cabecera *Section Headers* aparecen:
  - .text virtual Size = 0009YAC y RawSize = 0009Eco. Esto quiere decir que no ha usado ningún compresor

Aparecen varios DLL listados como importados por el malware. Estos son:

- en `DIRECTORY_ENTRY_IMPORTS` aparecen las siguientes API's
  - ADVAPI32.DLL

- KERNEL32.dll
- msvcrt.dll
- USER32.dll
- WS2\_32.dll

Usando PPEE, las cadenas que nos marca como sospechosas son las siguientes: Mas adelante, veremos que se han perdido muchas cadenas importantes para el análisis estático del archivo.

- Auspicios Seringa in file:
  - pwd

## Análisis estático con radare2:

- Abriendo el binario con radare2:
  - Encontramos una dirección IP en la dirección 0x004A2116.
  - Analizando las cadenas en .data vemos que hay una dirección IP: 10.10.0.121
  - Además, vemos que hay cadenas que hacen referencia a conexiones.
  - Todo esto nos lleva a pensar que este malware es un reverse shell.
  - Encontramos que hay un switch-case de seis posibilidades.
  - Esto posiblemente sean las distintas opciones del reverse shell

Abrimos el programa, luego, analizamos el binario. Esto hace que radare2 reconozca las llamadas a funciones y les asigne alias, generalmente basados en su nombre. Para buscar las cadenas encontradas en la sección .data, escribimos el comando iz. Tras analizar las cadenas, nos aparecen todas las que ha podido encontrar.

```
[0x00410750]> iz
[Strings]
nth  paddr      vaddr      len size section type  string
-----
0  0x000a0c00 0x004a2000 18  19  .rdata  ascii  libgcc_s_dw2-1.dll
1  0x000a0c13 0x004a2013 21  22  .rdata  ascii  _register_frame_info
2  0x000a0c29 0x004a2029 23  24  .rdata  ascii  _deregister_frame_info
3  0x000a0c48 0x004a2048 62  63  .rdata  ascii  =====\n
4  0x000a0c88 0x004a2088 31  32  .rdata  ascii  [*] FOR RESEARCH PURPOSES ONLY\n
5  0x000a0ca8 0x004a20a8 46  47  .rdata  ascii  [*] SEE YOUTUBE CHANNEL FOR MORE INFORMATION:\n
6  0x000a0cd8 0x004a20d8 61  62  .rdata  ascii  [*] https://www.youtube.com/channel/UCo8vV94aQsuvPrkymFc1lYg\n
7  0x000a0d16 0x004a2116 10  11  .rdata  ascii  10.0.0.121
8  0x000a0d21 0x004a2121 29  30  .rdata  ascii  [ ] Attempting to connect to
9  0x000a0d3f 0x004a213f 17  18  .rdata  ascii  [+] Connected to
10 0x000a0d51 0x004a2151 23  24  .rdata  ascii  Waiting for command...
11 0x000a0d69 0x004a2169 18  19  .rdata  ascii  Command received:
12 0x000a0d7c 0x004a217c 7  8  .rdata  ascii  whoami\n
13 0x000a0d84 0x004a2184 4  5  .rdata  ascii  pwd\n
14 0x000a0d89 0x004a2189 9  10  .rdata  ascii  hostname\n
15 0x000a0d93 0x004a2193 11  12  .rdata  ascii  disconnect\n
16 0x000a0d9f 0x004a219f 5  6  .rdata  ascii  exit\n
```

Figure 1: Cadenas en .data

De la captura previa, casi podemos deducir que estamos ante un reverse shell, pero aun así, no podemos asegurarlo, porque no tenemos pruebas. Cabe destacar las cadenas:

- 10.0.0.121

- Attempting to connect to
- Connected to
- Waiting for command
- Command received
- whoami\n
- pwd\n
- hostname\n
- disconnect\n

De estas cadenas, la única que ha encontrado PPEE conjuntamente con Radare2 es `pwd`

Teniendo esta información, es bastante probable, que este usando funciones de gestión de cadenas para comparar las cadenas que envía el actor malintencionado con las posibles opciones que contiene el programa.

Podemos buscar las llamadas a las funciones que usa el malware con **Radare2** usando el comando: `alf`. Si analizamos las llamadas a las funciones, vemos que, efectivamente se llama a las siguientes:

Con el comando `s main` le decimos a radare2 que queremos que nos lleve al `main` del binario. Desde este punto, le decimos que queremos analizarlo: `pdf`.

En la captura anterior vemos que el programa esta autocontenido, ya que todos los saltos son sobre las mismas direcciones. La única que no devuelve al mismo código es la `0x00410794` que es la salida del programa. Entiendo que el código lleva a esta dirección cuando el usuario conectado desde Internet envíe la cadena `disconnect`.

Sabiendo que el programa parece ser una reverse shell, vamos a analizar las funciones a las que se llaman desde el programa. Entre otras, encontramos las que hacen referencia a conexiones con sockets.

Esto verifica que efectivamente, se crea una conexión a la IP `10.0.0.121`.

## Análisis Dinámico

Para hacer el análisis dinámico, se usa **x64dbg**. Antes de correr el programa con el debugger, nos aseguramos de que la maquina virtual de Ubuntu este desconectada de Internet y que tenga Wireshark y INetSim corriendo.

Al correr el malware directamente con permisos de administrador teniendo Wireshark escuchando en Ubuntu, vemos que, efectivamente, el malware se intenta conectar a la dirección IP `10.0.0.121`. Lo podemos ver en la siguiente captura de pantalla:

Esto es prueba definitiva que, efectivamente estamos analizando un reverse shell.

Se ha usado Noriben, de forma que antes de que se inicie el malware, se ejecuta Noriben, que a su vez, ejecuta ProcMon.exe, que es el programa ProcessMonitor. Este programa registra la actividad de los programas ejecutados después de

45	0x004e840c	NONE	FUNC	msvcrt.dll	setlocale
46	0x004e8410	NONE	FUNC	msvcrt.dll	setvbuf
47	0x004e8414	NONE	FUNC	msvcrt.dll	signal
48	0x004e8418	NONE	FUNC	msvcrt.dll	sprintf
49	0x004e841c	NONE	FUNC	msvcrt.dll	strcat
50	0x004e8420	NONE	FUNC	msvcrt.dll	strchr
51	0x004e8424	NONE	FUNC	msvcrt.dll	strcmp
52	0x004e8428	NONE	FUNC	msvcrt.dll	strcoll
53	0x004e842c	NONE	FUNC	msvcrt.dll	strerror
54	0x004e8430	NONE	FUNC	msvcrt.dll	strftime
55	0x004e8434	NONE	FUNC	msvcrt.dll	strlen
56	0x004e8438	NONE	FUNC	msvcrt.dll	strncmp
57	0x004e843c	NONE	FUNC	msvcrt.dll	strtod
58	0x004e8440	NONE	FUNC	msvcrt.dll	strtoul
59	0x004e8444	NONE	FUNC	msvcrt.dll	strxfrm
60	0x004e8448	NONE	FUNC	msvcrt.dll	tolower
61	0x004e844c	NONE	FUNC	msvcrt.dll	towlower
62	0x004e8450	NONE	FUNC	msvcrt.dll	towupper
63	0x004e8454	NONE	FUNC	msvcrt.dll	ungetc
64	0x004e8458	NONE	FUNC	msvcrt.dll	ungetwc
65	0x004e845c	NONE	FUNC	msvcrt.dll	vfprintf
66	0x004e8460	NONE	FUNC	msvcrt.dll	wscoll
67	0x004e8464	NONE	FUNC	msvcrt.dll	wcsftime
68	0x004e8468	NONE	FUNC	msvcrt.dll	wcslen
69	0x004e846c	NONE	FUNC	msvcrt.dll	wcstombs
70	0x004e8470	NONE	FUNC	msvcrt.dll	wcsxfrm

Figure 2: Funciones de análisis de cadenas

```

[0x00410750]> pdf
; CALL XREF from section..text @ +0x14b
; CALL XREF from fcn.004011a0 @ 0x4011e8
;-- 5:
108: int main(int32_t arg_4h);
; var int32_t var_10h @ esp+0xc
; arg int32_t arg_4h @ esp+0x20
0x00410750 83ec1c sub esp, 0x1c
0x00410753 8b442420 mov eax, dword [arg_4h]
0x00410757 c744240c801f. mov dword [var_10h], 0x1f80 ; [0x1f80:4]--1
0x0041075f 83f8fd cmp eax, 0xffffffff
;=< 0x00410762 744c je 0x4107b0
| 0x00410764 83f8fc cmp eax, 0xffffffffc
;=< 0x00410767 742c je 0x410795
|| 0x00410769 85c0 test eax, eax
;==< 0x0041076b 7453 je 0x4107c0
||| ; CODE XREF from main @ 0x4107c5
;----> 0x0041076d 83f8ff cmp eax, 0xffffffff
;====> 0x00410770 7448 je 0x4107ba
|:| 0x00410772 83f8fe cmp eax, 0xffffffffe
;====> 0x00410775 7428 je 0x41079f
|:| 0x00410777 d920 fldenv [eax]
|:| 0x00410779 0fb7401c movzx eax, word [eax + 0x1c]
|:| 0x0041077d 8944240c mov dword [var_10h], eax
|:| ; CODE XREFS from main @ 0x4107a5, 0x4107bc
;----> 0x00410781 f60528704e00. test byte [0x4e7028], 0x10 ; [0x4e7028:1]=0
;====> 0x00410788 7405 je 0x41078f
|:| 0x0041078a 0fae54240c ldmxcsr dword [esp + 0xc]
|:| ; CODE XREF from main @ 0x410788
;----> 0x0041078f 31c0 xor eax, eax
|:| 0x00410791 83c41c add esp, 0x1c
|:| 0x00410794 c3 ret
|:| ; CODE XREF from main @ 0x410767
;----> 0x00410795 c70524004a00. mov dword [0x4a0024], 0xffffffffe ; [0x4a0024:4]--1
|:| ; CODE XREF from main @ 0x410775
;----> 0x0041079f ff1580834e00 call dword [sym.imp.msvcrt.dll__fpreset] ; 0x4e8380 ; ".\x88\x0e"
;====> 0x004107a5 ebda jmp 0x410781
|:| | ; CODE XREF from main @ 0x410762
|:| --> 0x004107b0 c70524004a00. mov dword [0x4a0024], 0xffffffff ; [0x4a0024:4]--1
|:| ; CODE XREF from main @ 0x410770
;----> 0x004107ba dbe3 fninit
;====> 0x004107bc ebc3 jmp 0x410781
|:| ; CODE XREF from main @ 0x41076b
;----> 0x004107c0 a124004a00 mov eax, dword [0x4a0024] ; [0x4a0024:4]--1
;====> 0x004107c5 eba6 jmp 0x410760
[0x00410750]>

```

Figure 3: Main autocontenido

1	0x004e8484	NONE	FUNC	WS2_32.dll	WSACleanup
2	0x004e8488	NONE	FUNC	WS2_32.dll	WSAStartup
3	0x004e848c	NONE	FUNC	WS2_32.dll	closesocket
4	0x004e8490	NONE	FUNC	WS2_32.dll	connect
5	0x004e8494	NONE	FUNC	WS2_32.dll	htons
6	0x004e8498	NONE	FUNC	WS2_32.dll	inet_addr
7	0x004e849c	NONE	FUNC	WS2_32.dll	recv
8	0x004e84a0	NONE	FUNC	WS2_32.dll	send
9	0x004e84a4	NONE	FUNC	WS2_32.dll	socket

Figure 4: Métodos API sockets

2	85.096088023	192.168.10.20	10.0.0.121	TCP	66 49937 → 8080 [SYN] Seq=0
3	86.097316343	192.168.10.20	10.0.0.121	TCP	66 [TCP Retransmission] 4993
4	87.284693059	192.168.10.20	192.168.10.10	DNS	83 Standard query 0x8d3e PTR
5	87.300184991	192.168.10.10	192.168.10.20	DNS	112 Standard query response 0
6	88.113656137	192.168.10.20	10.0.0.121	TCP	66 [TCP Retransmission] 4993
7	88.297003871	192.168.10.20	192.168.10.10	DNS	86 Standard query 0x541e PTR
8	88.313673246	192.168.10.10	192.168.10.20	DNS	115 Standard query response 0
9	90.018814294	PcsCompu_18:e1:b9	PcsCompu_64:02:ff	ARP	60 Who has 192.168.10.10? Te
10	90.018843992	PcsCompu_64:02:ff	PcsCompu_18:e1:b9	ARP	42 192.168.10.10 is at 08:00
11	92.129420858	192.168.10.20	10.0.0.121	TCP	66 [TCP Retransmission] 4993
12	92.499688928	PcsCompu_64:02:ff	PcsCompu_18:e1:b9	ARP	42 Who has 192.168.10.20? Te
13	92.499940297	PcsCompu_18:e1:b9	PcsCompu_64:02:ff	ARP	60 192.168.10.20 is at 08:00
14	100.159645778	192.168.10.20	10.0.0.121	TCP	66 [TCP Retransmission] 4993

Figure 5: Wireshark

haberse iniciado, pero por algún motivo, no se ha conseguido hallar un registro del malware en las diversas pruebas que se han llevado a cabo. Aunque no se ha podido encontrar sacar información valiosa de Noriben, si que hemos podido analizar el malware directamente con ProcessMonitor.

Process Name	PID	Operation	Path
maldev.exe	5684	Process Start	
maldev.exe	5684	Thread Create	
maldev.exe	5684	RegSetValue	HKLM\System\CurrentControlSet\Services\
maldev.exe	5684	Process Create	C:\Windows\System32\Conhost.exe
maldev.exe	5684	Thread Create	
maldev.exe	5684	TCP Reconnect	192.168.10.20:50096 -> 10.0.0.121:8080
maldev.exe	5684	TCP Reconnect	192.168.10.20:50096 -> 10.0.0.121:8080
maldev.exe	5684	TCP Reconnect	192.168.10.20:50096 -> 10.0.0.121:8080
maldev.exe	5684	TCP Reconnect	192.168.10.20:50096 -> 10.0.0.121:8080
maldev.exe	5684	TCP Disconnect	192.168.10.20:50096 -> 10.0.0.121:8080
maldev.exe	5684	Process Exit	
maldev.exe	5684	RegSetValue	HKLM\System\CurrentControlSet\Services\

Figure 6: ProcessMonitor

Podemos ver la serie de acciones que ha seguido `maldev.exe` desde el inicio del proceso, cuando se ejecuta con permisos de administrador, hasta que se cierra automáticamente, al no poder conectarse con el servidor CC.

## Conclusiones

Podemos deducir, basándonos en las pruebas obtenidas durante el análisis del malware que es un reverse shell. En el momento en el que se ejecuta, crea un hilo y se intenta conectar a su servidor CC. Al no conseguirlo, porque esta dentro del entorno de pruebas, se cierra al cabo de unos segundos.



## Análisis numero 2

### Cabeceras

Analizando el archivo, podemos ver que estamos delante de un ejecutable para equipos Windows.

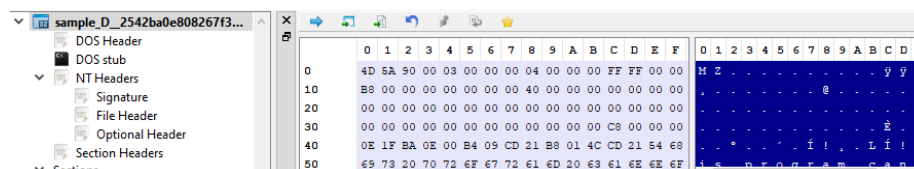


Figure 7: Flag MZ

La inspección de las cabeceras nos muestra que tan solo se importa un archivo y se usan 15 funciones del mismo;

Offset	Name	Func. Count	Bound?	OriginalFirstThun	TimeStamp	Forwarder	NameRVA
C98	ntoskrnl.exe	15	FALSE	20C0	0	0	2256

Call via	Name	Ordinal	Original Thunk	Thunk	Forwarder
2000	IoCompleteRequest	-	2100	2100	-
2004	ObfDereferenceObject	-	2116	2116	-
2008	ZwClose	-	212E	212E	-
200C	ZwAllocateVirtualMemory	-	2138	2138	-
2010	ObOpenObjectByPointer	-	2152	2152	-
2014	PsLookupProcessByProcessId	-	216A	216A	-
2018	MmIsAddressValid	-	2188	2188	-
201C	ZwOpenProcess	-	219C	219C	-
2020	KeServiceDescriptorTable	-	21AC	21AC	-
2024	ExAllocatePoolWithTag	-	21C8	21C8	-
2028	PsSetCreateProcessNotifyRoutine	-	21E0	21E0	-

Figure 8: ntoskrnl.exe imports

Tras una búsqueda de la librería **ntoskrnl** vemos que es el archivo binario correspondiente al núcleo o kernel del sistema operativo en la plataforma Microsoft Windows NT. Es responsable de servicios centrales del sistema, como: la visualización del hardware, el control de procesos, la gestión de memoria, etc. por lo que constituye un módulo fundamental del sistema operativo.

Como vemos en la captura de pantalla anterior, el malware esta llamando directamente a las funciones **Close**, **AllocateVirtualMemory** y **OpenProcess** con el prefijo **Zw** que, si es llamado por un kernel-mode driver, indica al la rutina de servicio del sistema a la que llama que la llamada viene de un driver de kernel fiable, por tanto, no valida los parámetros.

Si se llama a las funciones desde una aplicación en user-mode, la rutina revisa los parámetros para asegurar que están bien tratados. Ahondando mas, la rutina

revisa los buffers proporcionados por el llamador para verificar que están en una sección de memoria válida y correctamente alineados.

- *ZwClose*: Cierra un object handler
- *ZwOpenProcess*: Abre un handler al objeto y establece los permisos de acceso al mismo
- *ZwAllocateVirtualMemory*: Abre un handler al objeto y establece los permisos de acceso al mismo
- *IoCreateDevice*: Crea un device object para que lo use un driver.

## Análisis Estático

Si abrimos el malware con IDA, vemos que detecta varias funciones. Tras un análisis de lo que hace cada una de ellas, podemos llegar a la siguiente nomenclatura para cada función.

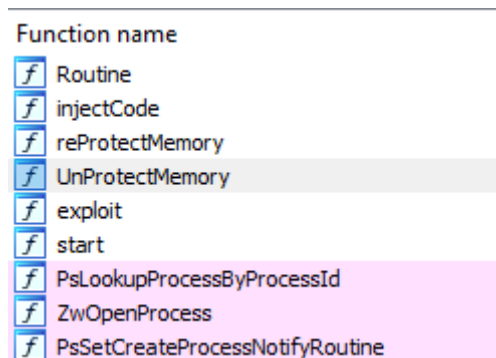


Figure 9: Funciones interpretadas

Analizando estas funciones de forma estática llegamos a la siguiente conclusión:

### unProtectMemory

Una forma de anular el control de acceso a una zona de memoria protegida cambiando el registro `cr0`.

```
sub_100012BC proc near
mov     eax, cr0
and     eax, 0FFFEFFFFh
mov     cr0, eax
retn
sub_100012BC endp
```

Figure 10: Deshabilita control de acceso a zona de memoria controlada

## reProtectMemory

Una forma de habilitar el control de acceso a una zona de memoria protegida cambiando el registro `cr0`.

```
sub_100012B0 proc near
mov     eax, cr0
or      eax, 10000h
mov     cr0, eax
retn
sub_100012B0 endp
```

Figure 11: Habilita control de acceso a zona de memoria controlada

## injectCode

Esta función copia parámetro 3 veces el contenido del puntero pasado en el parámetro 1 a una nueva dirección de memoria (parámetro 2). Puesto en contexto, se puede deducir que esta inyectando código a una zona protegida de memoria, a la que no debería estar accediendo.

## exploit

Esta función es la que se encarga de gestionar la lógica de la inyección de código en zona protegida. Al final, la zona mas interesante de la misma es la parte incluida en la imagen. Se encarga de des proteger la zona de memoria reservada, inyectar el código en esa zona de memoria y volver a proteger la zona de memoria.

## Routine

Es la rutina que se ejecuta en “start”. Busca el proceso creado en “start” y cede el control a si misma, que luego ejecuta “exploit” al haberlo recibido por el registro `esi`.

## start

Es la entrada del programa. Crea un device y genera un enlace simbólico para poder buscarlo desde la rutina. Si esto falla, sale del programa. Se encarga de configurar todo lo necesario para poder llamar a “exploit” desde la rutina y de añadir la rutina al `processNotifier`.

## Análisis Dinámico

Al analizar el driver malicioso con x32dbg, se inicia una consola y se cierra automáticamente el proceso. Esto es debido a que el sample no viene empaquetado en forma de ejecutable, sino un dll.

Para abrirlo, se ha intentado sin éxito utilizar x32dbg.

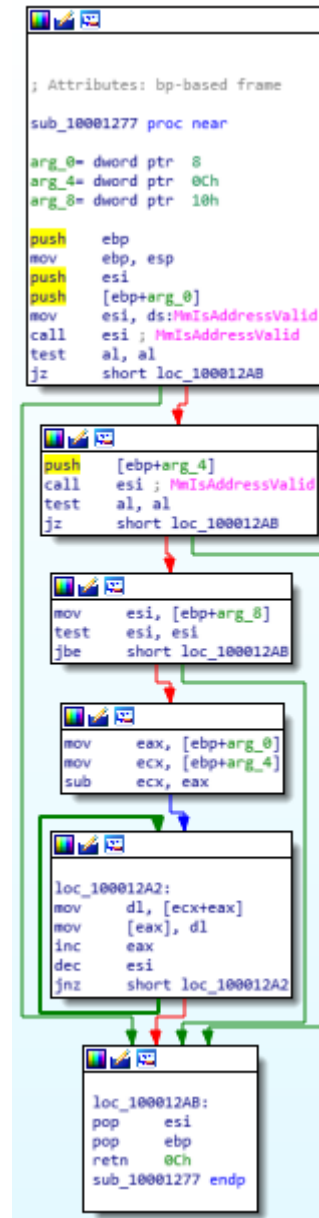
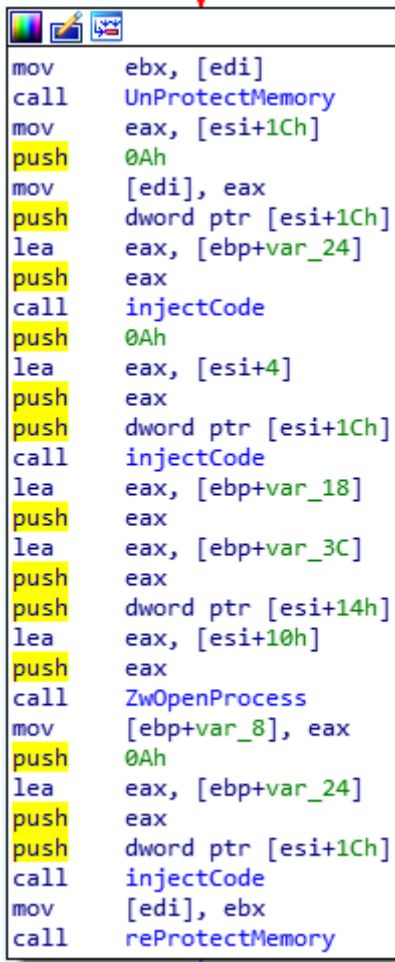


Figure 12: Inyección de código



```

mov     ebx, [edi]
call    UnProtectMemory
mov     eax, [esi+1Ch]
push    0Ah
mov     [edi], eax
push    dword ptr [esi+1Ch]
lea     eax, [ebp+var_24]
push    eax
call    injectCode
push    0Ah
lea     eax, [esi+4]
push    eax
push    dword ptr [esi+1Ch]
call    injectCode
lea     eax, [ebp+var_18]
push    eax
lea     eax, [ebp+var_3C]
push    eax
push    dword ptr [esi+14h]
lea     eax, [esi+10h]
push    eax
call    ZwOpenProcess
mov     [ebp+var_8], eax
push    0Ah
lea     eax, [ebp+var_24]
push    eax
push    dword ptr [esi+1Ch]
call    injectCode
mov     [edi], ebx
call    reProtectMemory

```

Figure 13: Core de la función exploit

```

loc_100015F9:
mov     eax, offset loc_10001000
push    0
push    offset Routine
mov     [esi+40h], eax
mov     [esi+38h], eax
mov     dword ptr [esi+70h], offset exploit
call    PsSetCreateProcessNotifyRoutine
push    eax
mov     eax, ecx
mov     [ebp+var_8], eax
pop     eax
pop     edi
xor     eax, eax
pop     esi
leave
retn    8
start endp

```

Figure 14: Start

El primer procedimiento que se ha seguido es el siguiente:

1. Análisis de las funciones a exportar con PEBear (En este punto, no aparece ninguna)
2. Configurar x32dbg para que añada un punto de ruptura al cargar un DLL y en su punto de entrada.
3. Cargar el `.dll` directamente en x32dbg (Porque genera automáticamente un `.exe` el cual podemos analizar).

El segundo procedimiento que se ha seguido es el siguiente:

1. Análisis de las funciones a exportar con PEBear (En este punto, no aparece ninguna)
2. Configurar x32dbg para que añada un punto de ruptura al cargar un DLL y en su punto de entrada.
3. Cargar `runDLL` en x32dbg
4. Cambiar el comando a ejecutar para que importe el `dll`
  - La sintaxis del comando: `rundll32.exe <full path to dll>, <export function> <optional arguments>`
  - Se ha usado la ruta completa en todos los archivos, incluso en el ejecutable de `rundll`
  - Como no se sabia la función que se tenia que usar, porque PEBear no la mostraba, se ha dejado vacío, respetando la coma después de la ruta al `dll`
5. El resultado de los comandos mencionados previamente siempre ha sido

un error

### **Hipótesis de fallo**

Sospecho que el malware esta cargando los exporta de forma dinámica al ser ejecutado o que tienen una especie de comprobación de entorno de reversing/debugging. Me parece extraño que al analizarlo con PEBear no muestre ningún indicio de ser un .dll porque no muestra ninguna función a exportar.

### **Comentario del Alumno**

Soy consciente que el máximo numero de paginas recomendadas eran menos de 10. En la primera entrada, es a lo que me he ceñido, pero al darse la situación actual y entrar en el foro y leer que se evaluaría un máximo de dos análisis, he decidido analizar otro malware para poder optar a la máxima nota. Espero que se me disculpe la longitud del documento presentado. He intentado mantener el comentario sobre el análisis del segundo malware no mas breve posible. Una vez dicho esto, me gustaría mucho entender por que no he podido realizar un análisis dinámico del malware. Adjunto el enlace al sitio web de donde he descargado el malware por si consideráis oportuno revisarlo. [https://grsecurity.net/malware\\_research/](https://grsecurity.net/malware_research/) Es el archivo llamado: sample\_D\_\_2542ba0e808267f3c35372954ef552fd54859063.