Análisis de malware

OllyDbg

Gustavo Romero López - gustavo@ugr.es

Updated: 3 de marzo de 2025

Departamento de Ingeniería de Computadores, Automática y Robótica

Por donde vamos...

Parte 1: Análisis básico
- Capítulo 1: Técnicas estáticas básicas
- Capítulo 2: Análisis de malware en máquinas virtuales
- Capítulo 3: Análisis dinámico básico

Parte 2: Análisis estático avanzado
- Capítulo 4: Curso intensivo de ensamblador x86
- Capítulo 5: IDA Pro
- Capítulo 6: Reconocimiento de construcciones de C en ensamblador
- Capítulo 7: Análisis de programas maliciosos para Windows

Parte 3: Análisis dinámico avanzado
- Capítulo 8: Depuración
- Capítulo 9: OllyDbg
- Capítulo 10: Depuración del núcleo con WinDbg

Parte 4: Funcionalidad del malware

Parte 5: Anti-ingeniería inversa

Parte 6: Temas especiales

OllyDbg

1. Carga de malware
2. Interfaz
3. Mapa de memoria
4. Pilas y hebras
5. Ejecución de código
6. Puntos de ruptura
7. Carga de bibliotecas
8. Trazado
9. Manejo de excepciones
10. Parcheado
11. Análisis de shellcodes
12. Asistencia
13. Plugins
14. Depuración mediante guiones (Scripts)

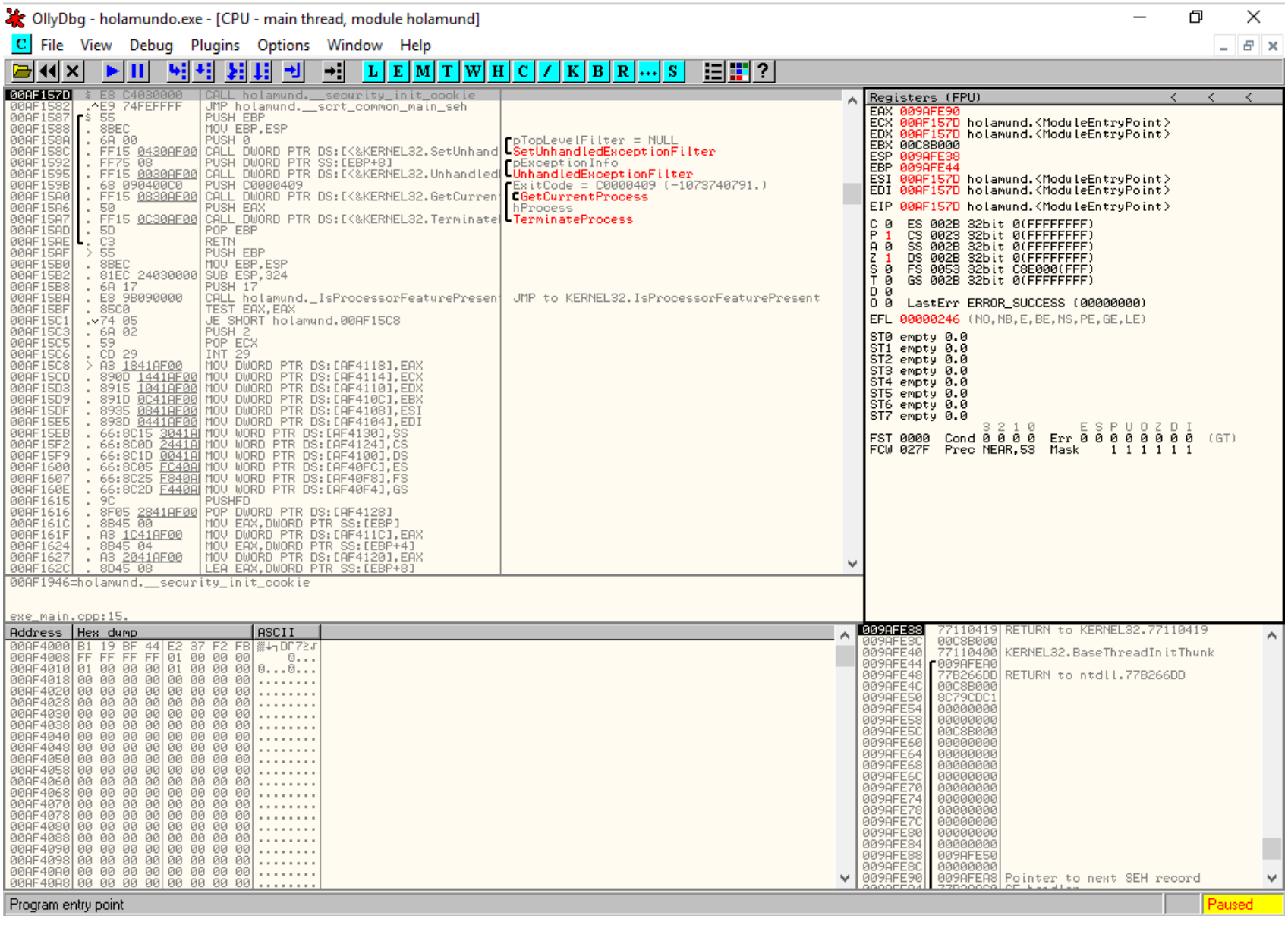
OllyDbg

- Creado por Oleh Yuschuk
- Romper software $\longrightarrow$ análisis de malware
- Comprado por Immunity + GUI + python $\rightarrow$ ImmDbg
- Versiones:
  - 1.1 la más utilizada, sólo 32 bits
  - 2.01 añade 64 bits, considerada beta por algunos

Carga de malware

- Desde un fichero: File → Open
  - paso de parámetros (sólo aquí)
  - pausa en WinMain o punto de entrada en cabecera PE
  - versión 2.0: TLS Callback (anti-debugging)
- Uniéndose a un proceso en ejecución: File → Attach
  - pausa todas las hebras
  - posible ruptura en biblioteca de Windows
  - escapar con un punto de ruptura en sección de código

Interfaz

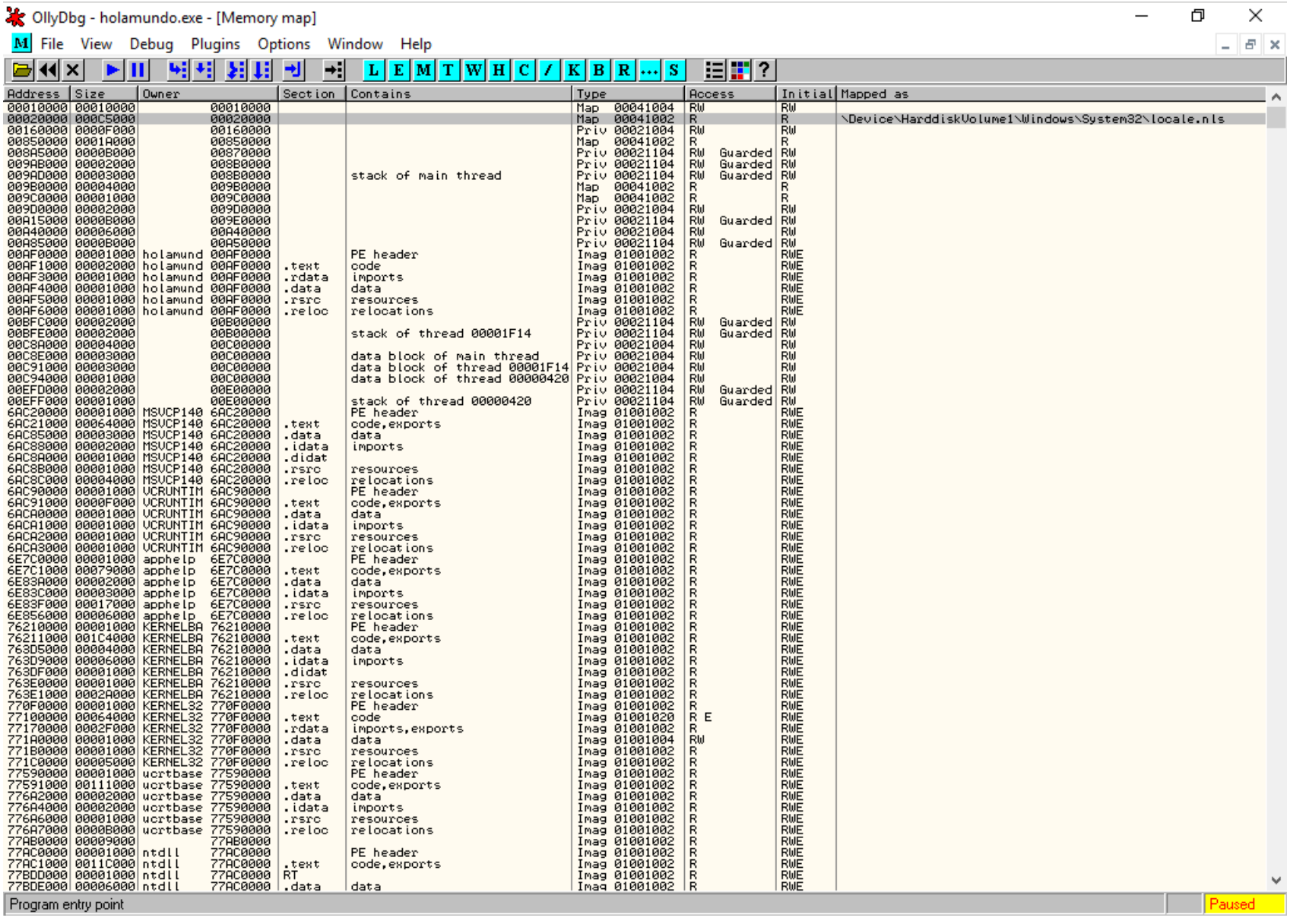

OllyDbg - holamundo.exe - [CPU - main thread, module holamund]
File View Debug Plugins Options Window Help
Registers (FPU)
Program entry point
Paused

Interfaz

Ventanas:

- Desensamblado:
  - pulse espacio para modificar
- Registros
  - cambian de color al ser modificados
  - botón derecho + modificar
- Pila
  - muestra el tope de la pila de hebra actual
  - botón derecho + modificar
- Memoria
  - muestra el volcado de memoria del proceso actual
  - modificar: CTRL+G, click, click derecho + Binary→Edit

Mapa de memoria: doble click/click derecho + View in Disassembler



Address	Size	Owner	Section	Contains	Type	Access	Initial	Mapped as
00010000	00010000		00010000		Map 00041004	RW	RW	
00020000	000C5000		00020000		Map 00041002	R	R	\Device\HarddiskVolume1\Windows\System32\locale.nls
00160000	0000F000		00160000		Priv 00021004	RW	RW	
00850000	0001A000		00850000		Map 00041002	R	R	
008A5000	0000B000		00870000		Priv 00021104	RW Guarded	RW	
009AB000	00002000		008B0000		Priv 00021104	RW Guarded	RW	
009AD000	00003000		008B0000	stack of main thread	Priv 00021104	RW Guarded	RW	
009B0000	00004000		009B0000		Map 00041002	R	R	
009C0000	00001000		009C0000		Map 00041002	R	R	
009D0000	00002000		009D0000		Priv 00021004	RW	RW	
00A15000	0000B000		009E0000		Priv 00021104	RW Guarded	RW	
00A40000	00006000		00A40000		Priv 00021004	RW	RW	
00A85000	0000B000		00A50000		Priv 00021104	RW Guarded	RW	
00AF0000	00001000	holamund	00AF0000	PE header	Imag 01001002	R	RWE	
00AF1000	00002000	holamund	00AF0000	.text code	Imag 01001002	R	RWE	
00AF3000	00001000	holamund	00AF0000	.rdata imports	Imag 01001002	R	RWE	
00AF4000	00001000	holamund	00AF0000	.data data	Imag 01001002	R	RWE	
00AF5000	00001000	holamund	00AF0000	.rsrc resources	Imag 01001002	R	RWE	
00AF6000	00001000	holamund	00AF0000	.reloc relocations	Imag 01001002	R	RWE	
00BFC000	00002000		00B00000		Priv 00021104	RW Guarded	RW	
00BFE000	00002000		00B00000	stack of thread 00001F14	Priv 00021104	RW Guarded	RW	
00C8A000	00004000		00C00000		Priv 00021004	RW	RW	
00C8E000	00003000		00C00000	data block of main thread	Priv 00021004	RW	RW	
00C91000	00003000		00C00000	data block of thread 00001F14	Priv 00021004	RW	RW	
00C94000	00001000		00C00000	data block of thread 00000420	Priv 00021004	RW	RW	
00EFD000	00002000		00E00000		Priv 00021104	RW Guarded	RW	
00EFF000	00001000		00E00000	stack of thread 00000420	Priv 00021104	RW Guarded	RW	
6AC20000	00001000	MSVCP140	6AC20000	PE header	Imag 01001002	R	RWE	
6AC21000	00064000	MSVCP140	6AC20000	.text code,exports	Imag 01001002	R	RWE	
6AC85000	00003000	MSVCP140	6AC20000	.data data	Imag 01001002	R	RWE	
6AC88000	00002000	MSVCP140	6AC20000	.idata imports	Imag 01001002	R	RWE	
6AC8A000	00001000	MSVCP140	6AC20000	.didat	Imag 01001002	R	RWE	
6AC8B000	00001000	MSVCP140	6AC20000	.rsrc resources	Imag 01001002	R	RWE	
6AC8C000	00004000	MSVCP140	6AC20000	.reloc relocations	Imag 01001002	R	RWE	
6AC90000	00001000	VCRUNTIM	6AC90000	PE header	Imag 01001002	R	RWE	
6AC91000	0000F000	VCRUNTIM	6AC90000	.text code,exports	Imag 01001002	R	RWE	
6ACA0000	00001000	VCRUNTIM	6AC90000	.data data	Imag 01001002	R	RWE	
6ACA1000	00001000	VCRUNTIM	6AC90000	.idata imports	Imag 01001002	R	RWE	
6ACA2000	00001000	VCRUNTIM	6AC90000	.rsrc resources	Imag 01001002	R	RWE	
6ACA3000	00001000	VCRUNTIM	6AC90000	.reloc relocations	Imag 01001002	R	RWE	
6E7C0000	00001000	apphelp	6E7C0000	PE header	Imag 01001002	R	RWE	
6E7C1000	00079000	apphelp	6E7C0000	.text code,exports	Imag 01001002	R	RWE	
6E83A000	00002000	apphelp	6E7C0000	.data data	Imag 01001002	R	RWE	
6E83C000	00003000	apphelp	6E7C0000	.idata imports	Imag 01001002	R	RWE	
6E83F000	00017000	apphelp	6E7C0000	.rsrc resources	Imag 01001002	R	RWE	
6E856000	00006000	apphelp	6E7C0000	.reloc relocations	Imag 01001002	R	RWE	
76210000	00001000	KERNELBA	76210000	PE header	Imag 01001002	R	RWE	
76211000	001C4000	KERNELBA	76210000	.text code,exports	Imag 01001002	R	RWE	
763D5000	00004000	KERNELBA	76210000	.data data	Imag 01001002	R	RWE	
763D9000	00006000	KERNELBA	76210000	.idata imports	Imag 01001002	R	RWE	
763DF000	00001000	KERNELBA	76210000	.didat	Imag 01001002	R	RWE	
763E0000	00001000	KERNELBA	76210000	.rsrc resources	Imag 01001002	R	RWE	
763E1000	0002A000	KERNELBA	76210000	.reloc relocations	Imag 01001002	R	RWE	
770F0000	00001000	KERNEL32	770F0000	PE header	Imag 01001002	R	RWE	
77100000	00064000	KERNEL32	770F0000	.text code	Imag 01001020	R E	RWE	
77170000	0002F000	KERNEL32	770F0000	.rdata imports,exports	Imag 01001002	R	RWE	
771A0000	00001000	KERNEL32	770F0000	.data data	Imag 01001004	RW	RWE	
771B0000	00001000	KERNEL32	770F0000	.rsrc resources	Imag 01001002	R	RWE	
771C0000	00005000	KERNEL32	770F0000	.reloc relocations	Imag 01001002	R	RWE	
77590000	00001000	ucrtbase	77590000	PE header	Imag 01001002	R	RWE	
77591000	00111000	ucrtbase	77590000	.text code,exports	Imag 01001002	R	RWE	
776A2000	00002000	ucrtbase	77590000	.data data	Imag 01001002	R	RWE	
776A4000	00002000	ucrtbase	77590000	.idata imports	Imag 01001002	R	RWE	
776A6000	00001000	ucrtbase	77590000	.rsrc resources	Imag 01001002	R	RWE	
776A7000	0000B000	ucrtbase	77590000	.reloc relocations	Imag 01001002	R	RWE	
77AB0000	00009000		77AB0000		Imag 01001002	R	RWE	
77AC0000	00001000	ntdll	77AC0000	PE header	Imag 01001002	R	RWE	
77AC1000	0011C000	ntdll	77AC0000	.text code,exports	Imag 01001002	R	RWE	
77BDD000	00001000	ntdll	77AC0000	RT	Imag 01001002	R	RWE	
77BDE000	00006000	ntdll	77AC0000	.data data	Imag 01001002	R	RWE	

Recodificación de direcciones (Rebasing)

- Cabecera PE → image base: dirección de memoria preferida
- En la mayoría de los ejecutables: 0x400000
- ASLR: medida de seguridad que la cambia aleatoriamente
- Las DLLs de Windows tienen direcciones base diferentes
- "Rebasing": cambiar un programa para que se ejecute correctamente en otra zona de memoria que no es para la que está preparado
- Añade tiempo de carga del binario puesto que requiere transformaciones

Direcciones absolutas y relativas

- Absoluta:
  - `movl 0x80d6024, %eax`
  - `mov eax, dword [0x80d6024]`
- Relativa:
  - `movl 0x2ff6(%ebx),%eax`
  - `mov eax, dword [ebx + 0x2ff6]`
- Cabecera PE → sección .reloc: lista de direcciones de memoria a cambiar al cargar un ejecutable fuera de su base
- Las DLLs se carga en cualquier orden
- Sin .reloc no pueden ser cargadas fuera de su base

Pilas y Hebras

- Se pueden ver con View → Threads.
- Muestra direcciones de memoria y estado.
- Pulsar el botón de pausa las detiene a todas.
- Cada hebra tiene su propia pila.
- Podemo examinar su contenido desde el mapa de memoria.

Ejecucion de código

Function	Menu	Hotkey	Button
Run/Play	Debug ▸ Run	F9	
Pause	Debug ▸ Pause	F12	
Run to selection	Breakpoint ▸ Run to Selection	F4	
Run until return	Debug ▸ Execute till Return	CTRL-F9	
Run until user code	Debug ▸ Execute till User Code	ALT-F9	
Single-step/step-into	Debug ▸ Step Into	F7	
Step-over	Debug ▸ Step Over	F8	

Puntos de ruptura

Function	Right-click menu selection	Hotkey
Software breakpoint	Breakpoint ▸ Toggle	F2
Conditional breakpoint	Breakpoint ▸ Conditional	SHIFT-F2
Hardware breakpoint	Breakpoint ▸ Hardware, on Execution	
Memory breakpoint on access (read, write, or execute)	Breakpoint ▸ Memory, on Access	F2 (select memory)
Memory breakpoint on write	Breakpoint ▸ Memory, on Write	

- Para ver la lista de puntos de ruptura:
  - pulsar B en la barra de herramientas.
  - View → Breakpoints.
- Se recuerdan entre ejecuciones.
- Condicionales: pueden hacerse depender de una condición.
- Hardware: indetectables y de rápida ejecución.
- Memoria: detectan accesos rwx*, lento.

Carga de bibliotecas

- Usa `loaddll.exe` para cargar DLLs.
- Punto de ruptura tras ejecutar punto de entrada (DllMain).
- Podemos llamar a cualquier otra función de la DLL.
- Debug → Call DLL Export.
  - Los parámetros pueden estar en registros o en la pila.
  - Muestra el antes y el después.
  - Podemos hacerlo mediante breakpoints y opciones.

Trazado

- Normal/Registro de movimientos:
  - Permite retroceder (-) o avanzar (+) en la ejecución.
  - Sólo en zonas por las que hayamos pasado.
- Pila de llamadas.
  - View → Call Stack.
  - Sigue el flujo de llamadas entre funciones.
- Traza de ejecución (registro detallado):
  - Almacena todo cambio entre intstucciones.
  - Gran ralentización y consumo de memoria.
  - Activación:
    - marcar + click derecho + Run Trace → Add Selection, luego View → Run Trace
    - Trace Into / Trace Over: como step-into y step-over
    - Evitar trazar programa completo: Debug → Set Condition

Manejo de excepciones

- Cuando ocurre una excepción se detiene el programa.
- OllyDbg puede elegir manejarla o dejar que lo haga el programa.
- Opciones:
  - SHIFT+F7: step-into.
  - SHIFT+F8: step-over.
  - SHIFT+F9: ejecuta el manejador.
- Suele ser buena idea ignorar todas las excepciones puesto que nuestro objetivo no es arreglar el programa.

Parcheado

- Podemos modificar cualquier región de memoria:

  Binary → Edit.
- Podemos modificar rangos: Binary → Fill with NOPs/00's
- Las modificaciones se hacen en memoria pero podemos almacenarlas con:
  - Click derecho + Copy to Executable → All Modifications.
  - Save File.
- Muy útil cuando el análisis requiere salvar varias etapas (trampas).

Análisis de shellcodes

1. Copiar el shellcode desde un editor hexadecimal.
2. Seleccionar una región de memoria privada.
3. Examinar la región para comprobar que solo contiene ceros.
4. Cambiar sus permisos: Click derecho + Set Access → Full Access (rwx).
5. Marcar la zona de memoria y Click derecho + Binary → Binary Paste.
6. Cambiar EIP a la nueva dirección de memoria o Click derecho + New Origin Here sobre la primera intrucción.

Asistencia

Mecanismos de ayuda con el análisis:

- Registro: View → Log.
- Watch: View → Watches.
- Ayuda: Help → Contents.
- Etiquetado: Click derecho + Label.

Plugins

- OllyDbg: gran cantidad disponible.
  - OllyDump: volcado de un proceso a un fichero PE.
  - Hide Debugger: evita ser detectado.
  - Command Line: uso de OllyDbg mediante órdenes.
  - Bookmarks: marcadores para recordar direcciones.
- ImmDbg:
  - Ventaja: añade scripting en python.
  - Inconveniente: no puede utilizar plugins de OllyDbg.

Depuración mediante guiones (*Scripts*)

- ImmDbg permite la escritura de scripts en Python.
- Multitud de scripts en Internet.
- PyCommand: ejecutables desde la línea de órdenes con "!".

Ejercicios: Lab 9-1

Analice el programa Lab09-01.exe con OllyDbg y IDA Pro para responder las siguientes preguntas. Ya debería estar familiarizado con él puesto que lo vimos por primera vez en el Capítulo 3.

1. ¿Como conseguir que el programa se instale?
2. ¿Qué opciones tiene? ¿Qué contraseña necesita?
3. ¿Cómo parchear permanentemente el programa para que no necesite la contraseña?
4. ¿Qué indicios deja en el ordenador?
5. ¿Qué acciones puede ejecutar inducido desde la red?
6. ¿Qué indicios de red existen?

Ejercicios: Lab 9-2

Analice el programa Lab09-02.exe con OllyDbg para responder las siguientes preguntas.

1. ¿Qué cadenas puede apreciar de manera estática?
2. ¿Qué sucede al ejecutar el binario?
3. ¿Cómo conseguir que se ejecute su carga maliciosa?
4. ¿Qué sucede en la dirección 0x00401133?
5. ¿Qué argumentos se le pasan a la subrutina 0x00401089?
6. ¿Qué nombre de dominio utiliza?
7. ¿Qué rutina de codificación utiliza para ofuscar el nombre de dominio?
8. ¿Para qué se utiliza `CreateProcessA` en 0x0040106e?

Ejercicios: Lab 9-3 I

Analice el programa Lab09-02.exe con OllyDbg y IDA Pro. Este malware carga 3 DLLs, DLL1.dll, DLL2.dll y DLL3.dll, que utilizan la misma dirección base. Al cargar las DLLs en OllyDbg y IDA Pro lo harán en direcciones diferentes. El propósito de este ejercicio es familiarizarnos con la localización de código en IDA Pro cuando estamos viéndolo en OllyDbg.

1. ¿Qué DLLs importa Lab09-03.exe?
2. ¿Cuál es la dirección base de DLL1.dll, DLL2.dll y DLL3.dll?
3. ¿Cuáles son las direcciones base de las DLLs al depurar Lab09-03.exe desde OllyDbg?

Ejercicios: Lab 9-3 II

4. ¿Qué hace la función que Lab09-03.exe llama desde DLL1.dll?
5. ¿Cuál es el nombre de fichero que se utiliza al llamar a `WriteFile` desde Lab09-03.exe?
6. ¿De dónde se consiguen los datos del segundo parámetro de la llamada a `NetScheduleJobAdd` que ejecuta La09-03.exe?
7. Al ejecutar Lab09-03.exe se imprimen 3 mensajes. ¿Qué datos proceden de cada una de las 3 DLLs?
8. ¿Cómo se puede cargar DLL2.dll en IDA Pro de manera que lo haga en las mismas direcciones que en OllyDbg?