Análisis de malware

Anti-depuración

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 3: Análisis dinámico básico

Parte 2: Análisis estático avanzado

Capítulo 7: Análisis de programas maliciosos para Windows

Parte 3: Análisis dinámico avanzado

Capítulo 8: Depuración

Capítulo 9: OllyDbg

Parte 4: Funcionalidad del malware

Capítulo 12: Ejecución encubierta

Parte 5: Anti-ingeniería inversa

Capítulo 15: Anti-desensamblado

Capítulo 16: Anti-depuración

Capítulo 17: Anti-máquinas virtuales

Capítulo 18: Compresores y descompresión

Parte 6: Temas especiales

1. Detección del depurador en Windows
2. Identificando el comportamiento del depurador
3. Interfiriendo al depurador
4. Vulnerabilidades en el depurador
5. Laboratorio

¿Qué es el anti-desensamblado?

- ⊙ Técnica para descubrir cuando un código se ejecuta bajo el control de un depurador.
- ⊙ Objetivo: entorpecer el análisis de malware.
- ⊙ El malware pueden alterar sus comportamiento normal en presencia de un depurador.

- ⦿ Método más obvio.
- ⦿ Contramedidas: evitar las llamadas a funciones o alterar su resultado.
- ⦿ API:
 - IsDebuggerPresent: consulta la estructura Process Environment Block (PEB).
 - CheckRemoteDebuggerPresent: consulta PEB en un proceso.
 - NtQueryInformationProcess: consulta ProcessDebugPort.
 - OutputDebugString: envía una cadena al depurador.

Comprobando estructuras manualmente

- ⊙ Método más utilizado.
- ⊙ Las llamadas a la API son fáciles de detectar e interferir.
- ⊙ Prefieren hacer ellos mismos las comprobaciones.
- ⊙ `fs:[30]` → `PEB.BeingDebugged`.
- ⊙ Contramedidas:
 - Cambiar la lógica del código.
 - Cambiar el valor de la bandera por o (plugin).

mov method

```
mov eax, dword ptr fs:[30h]
mov ebx, byte ptr [eax+2]
test ebx, ebx
jz NoDebuggerDetected
```

push/pop method

```
push dword ptr fs:[30h]
pop edx
cmp byte ptr [edx+2], 1
je DebuggerDetected
```

Comprobando banderas ProcessHeap

- ⊙ Cuando conseguimos memoria en el heap desde el depurador queda registrado.
- ⊙ PEB → BYTE Reserved4[104] → ForceFlags y Flags.
- ⊙ Contramedidas: cambiar el valor de la bandera (plugin).

```
mov eax, large fs:30h
mov eax, dword ptr [eax+18h]
cmp dword ptr ds:[eax+10h], 0
jne DebuggerDetected
```

Comprobando NTGlobalFlag

- ⦿ Los procesos se ejecutan de forma ligeramente diferente desde el interior de un depurador.
- ⦿ Ej: PEB → 68 bytes tras el inicio → 70 == depurado.
- ⦿ Contramedidas:
 - Cambiar el valor de la bandera (plugin).
 - Desactivar la depuración del heap en el depurador.

```
FLG_HEAP_ENABLE_TAIL_CHECK|FLG_HEAP_ENABLE_FREE_CHECK|FLG_HEAP_VALIDATE_PARAMETERS
```

```
mov eax, large fs:30h  
cmp dword ptr ds:[eax+68h], 70h  
jz DebuggerDetected
```

Residuos del depurador en el sistema

- ⊙ Claves de registro:

 - HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\AaDebug != DrWatson

- ⊙ Sistema de ficheros: ficheros y directorios pertenecientes a depuradores.

- ⊙ Lista de procesos: FindWindow("OLLYDBG", 0) != NULL.

Identificando el comportamiento del depurador

- ⊙ El depurador afecta al proceso depurado de múltiples formas.
- ⊙ Comprobaciones posibles para detectarlo:
 - Buscar la instrucción **int**.
 - Realizar y compara sumas de verificación.
 - Buscar desviaciones frente al tiempo normal de ejecución.

Buscando `int`

- ⊙ El depurador utiliza “`int3`” como mecanismo para implementar los puntos de ruptura.
 - Utilizar puntos de ruptura hardware.
- ⊙ Cambia la instrucción por esta petición interrupción.
- ⊙ Códigos de operación:
 - `int3` == `0xcc`.
 - `int $3` == `0xcd 0x03`.
- ⊙ Contramedidas:
 - ⊙ Ejemplo de implementación:

```
call $+5
pop edi
sub edi, 5
mov ecx, 400h
mov eax, 0CCh
repne scasb
jz DebuggerDetected
```

Sumas de verificación

- ⊙ Persigue el mismo objetivo que la búsqueda de `int3`, detectar modificaciones del depurador.
- ⊙ Menos frecuente pero igual de efectiva.
- ⊙ Debemos buscar bucles que recorran el código.
- ⊙ Contramedidas:
 - Utilizar puntos de ruptura hardware.
 - Alterar la ejecución para modificar el resultado o evitar la detección.

Comprobaciones de tiempo

- ⊙ Los procesos se ejecutan más despacio al ser depurados.
- ⊙ Mecanismo: buscar desviaciones en tiempos de ejecución.
 - medir el tiempo que tardan unas pocas instrucciones.
 - medir el tiempo que tarda en resolverse una excepción.
- ⊙ Implementaciones:
 - ASM: rdtsc
 - API: `QueryPerformanceCounter/GetTickCount`
- ⊙ Contramedidas:
 - Retrasar puntos de ruptura tras rdtsc.
 - Alterar la ejecución para modificar el resultado o evitar la detección.

- ⊙ Formas de interferir con el comportamiento del depurador:
 - Llamadas TLS (Thread Local Storage)
 - Excepciones
 - Insertando interrupciones: `int3`, `oxcc` ó `oxcd03`
 - Insertando interrupciones: `int $0x2d`
 - Insertando interrupciones: `icebp`
- ⊙ Contramedidas:
 - Pedir al depurador detenerse en “system breakpoint”.
 - Pasar todas las excepciones al programa.
 - Usar depuradores que ignoren “int *” ajenas.

Llamadas TLS (Thread Local Storage)

- ⊙ Los depuradores detienen al proceso en el punto de entrada de la cabecera PE.
- ⊙ Las llamadas TLS se ejecutan antes del punto de entrada.
- ⊙ TLS permite tener datos propios a cada hebra.
- ⊙ Su uso implica la aparición de una sección ".tls".
- ⊙ La mayoría de los programas no las usan pero cada día es más frecuente.
- ⊙ Contramedidas: cambiar el punto de parada por defecto del depurador a "system breakpoint".

Inserción de interrupciones

- ⊙ Forma de molestar al analista o cambiar la forma de ejecución.
- ⊙ Mecanismos:
 - `int3/oxcc`: Engaña al depurador haciéndole creer que ha alcanzado un punto de ruptura.
 - `int $3/oxcd03`: Interfiere WinDbg avanzando `eip` un byte en lugar de dos.
 - `int $0x2d/oxcd2d`: Punto de ruptura para depurar el núcleo.
 - `icebp/oxf1`: Instrucción no documentada para depurar ICE.
- ⊙ Contramedidas:
 - No ejecutar paso a paso.
 - Cambiar por `NOP's`

Vulnerabilidades en el depurador

- ⊙ Vulnerabilidades en cabeceras PE:
 - Alterar la cabecera PE para impedir la carga del ejecutable.
 - Interpretación demasiado estricta del estándar.
 - Muchos cambios posibles pueden evitar la carga correcta del ejecutable.
- ⊙ Vulnerabilidad en OutputDebugString:
 - Usar “%s” como parámetro cuelga al depurador.
- ⊙ Contramedidas:
 - Utilice depuradores sin vulnerabilidades.
 - Modifique las cabeceras PE.

- ⊙ Analice el malware Lab16-02.exe con un depurador. El objetivo es descubrir la contraseña correcta. No deja ninguna carga maliciosa.
- ⊙ Preguntas:
 1. ¿Qué sucede al ejecutar Lab16-02.exe?
 2. ¿Qué sucede al ejecutar Lab16-02.exe con el parámetro adecuado?
 3. ¿Cuál es la contraseña de la línea de órdenes?
 4. Cargue Lab16-02.exe en IDA Pro. ¿Dónde está `strncmp` en `main`?
 5. ¿Qué sucede al cargar Lab16-02.exe en OllyDbg con los parámetros por defecto?
 6. ¿Qué es único acerca de la cabecera PE de Lab16-02.exe?
 7. ¿Dónde se localiza la llamada TLS? Pista: pulse CTRL-E.

8. ¿Qué técnica anti-depurado utiliza para finalizar inmediatamente y como evitarla?
9. ¿Qué contraseña se ve en el depurador tras desactivar la técnica anti-depuración?
10. ¿Funciona la contraseña encontrada en el depurador en la línea de órdenes?
11. ¿Qué técnicas anti-depuración utiliza para las diferentes contraseñas en el depurador y en la línea de órdenes y cómo evitarlas?