Análisis de malware

Ejecución encubierta

Gustavo Romero López - gustavo@ugr.es

Updated: 11 de diciembre de 2024

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

Capítulo 10: Depuración del núcleo con WinDbg

Parte 4: Funcionalidad del malware

Capítulo 11: Comportamiento del malware

Capítulo 12: Ejecución encubierta

Capítulo 13: Codificación de datos

Capítulo 14: Firmas de red enfocadas al malware

Parte 5: Anti-ingeniería inversa

Parte 6: Temas especiales

Ejecución encubierta

1. Lanzadores
2. Inyección de procesos
3. Reemplazo de procesos
4. Inyección de ganchos
5. Desvíos (*"Detours"*)
6. Inyección APC

Lanzadores/Cargadores

- Objetivo: ejecución encubierta inmediata o futura.
- Escondido entre los recursos de ejecutable PE o DLL.
- Recursos:
  - Sección más común para esconder ejecutables.
  - Habitualmente comprimidos o cifrados.
  - API: `FindResource`, `LoadResource` y `SizeofResource`
- Suelen requerir privilegios de administrador.
  - Localizar escalada de privilegios.

Inyección de procesos

- Es la técnica más utilizada.
- Inyectar código dentro de otro proceso en ejecución.
- Objetivo: saltarse las medidas de seguridad (cortafuegos).
- API:
  - `VirtualAllocEx`: aloja memoria en un proceso remoto.
  - `WriteProcessMemory`: escribe en dicha memoria.
  - `CreateRemoteThread`: crea una nueva hebra remota.

Inyección DLL

- Fuerza a un proceso a cargar una DLL maliciosa.
- Mecanismo de ocultación:
  - El proceso llama a `LoadLibrary`.
  - El SO llama a la función `DllMain` de la DLL.
- Implementación:
  - buscar proceso con `CreateToolhelp32Snapshot`, `Process32First` y `Process32Next`.
  - obtener manejador con `OpenProcess` y el PID de la víctima.
  - aloja memoria en la víctima con `VirtualAllocEx`.
  - escribe en dicha memoria el nombre de la DLL maliciosa con `WriteProcessMemory`.
  - consigue la dirección de la DLL con `GetProcAddress`.
  - finalmente `CreateRemoteThread` lanza en la victima una hebra que carga la DLL maliciosa.
- Detección: buscar este patrón y estudiar sus cadenas.

Inyección DLL (2)

```
hVictimProcess = OpenProcess(PROCESS_ALL_ACCESS, 0, victimProcessID ❶);

pNameInVictimProcess = VirtualAllocEx(hVictimProcess,...,sizeof(maliciousLibraryName),...,...);
WriteProcessMemory(hVictimProcess,...,maliciousLibraryName, sizeof(maliciousLibraryName),...);
GetModuleHandle("Kernel32.dll");
GetProcAddress(...,"LoadLibraryA");
❷ CreateRemoteThread(hVictimProcess,...,...,LoadLibraryAddress,pNameInVictimProcess,...,...);
```

Listing 12-1: C Pseudocode for DLL injection

Inyección directa

- Inyecta el código malicioso directamente en la víctima.
- Código especial de difícil construcción.
- Patrón del ataque:
  - `VirtualAllocEx` + `WriteProcessMemory` $\longrightarrow$ inyecta datos.
  - `VirtualAllocEx` + `WriteProcessMemory` $\longrightarrow$ inyecta código.
  - `CreateRemoteThread` $\longrightarrow$ hebra remota en la víctima
- Suele inyectarse un shellcode.
- Detección: volcar la memoria con los parámetros de las llamadas a `CreateRemoteThread`

Reemplazo de procesos

- Sobrescribir la memoria de un proceso por otro ejecutable.
- Patrón de ataque:
  - Crear proceso en estado suspendido mediante `CreateProcess(..."svchost.exe"...CREATE_SUSPEND...)`
  - reemplazar sus secciones en memoria con:
    - `ZwUnmapViewOfSection`.
    - `VirtualAllocEx`.
    - `WriteProcessMemory`.
  - establecer punto de entrada con `SetThreadContext`.
  - reiniciar el proceso con `ResumeThread`.
- Forma efectiva de saltarse cortafuegos e IPSs al creerlo un proceso legítimo.

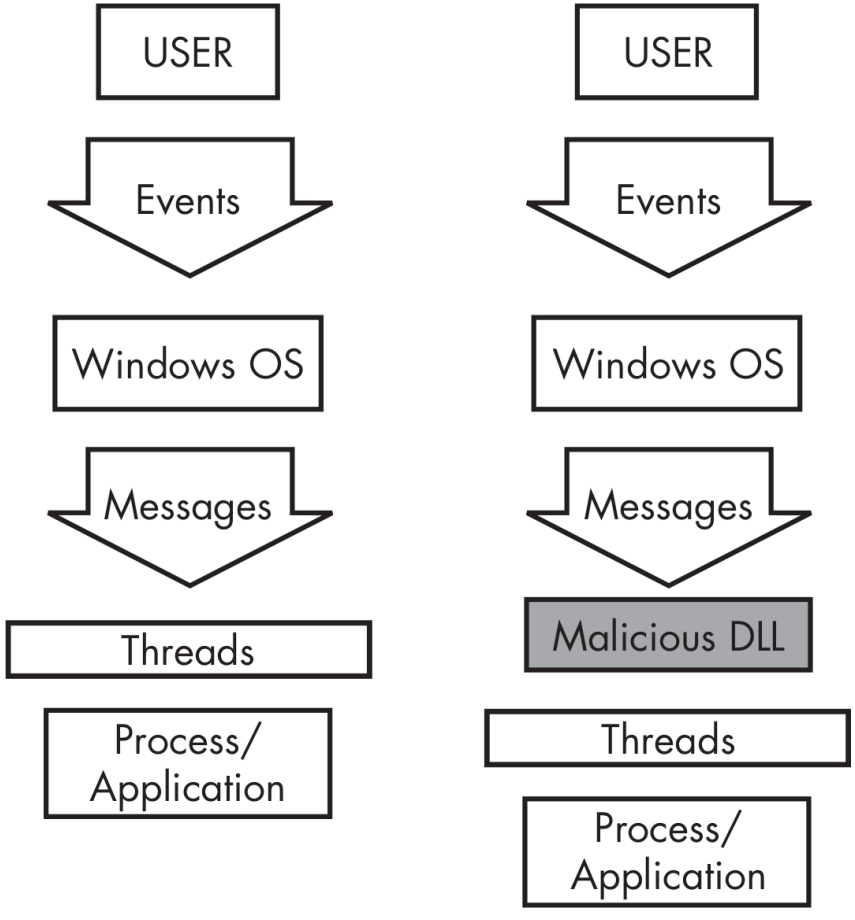
Reemplazo de procesos (2)

```
CreateProcess(...,"svchost.exe",...,CREATE_SUSPEND,...);
ZwUnmapViewOfSection(...);
VirtualAllocEx(...,ImageBase,SizeOfImage,...);
WriteProcessMemory(...,headers,...);
for (i=0; i < NumberOfSections; i++) {
  ❶ WriteProcessMemory(...,section,...);
}
SetThreadContext();
...
ResumeThread();
```

Listing 12-3: C pseudocode for process replacement

Inyección de ganchos ("*Hook Injection*")

- Intercepta mensajes destinados a procesos.
- Objetivos:
  - Ejecutar código cuando se intercepta cierto mensaje.
  - Carga una DLL en el espacio de memoria de la víctima.



Tipos de ganchos locales y remotos

- Tipos:
  - **Locales**: Observan y manipulan mensajes internos.
  - **Remotos**: Observan y manipulan mensajes destinados a otros procesos.
- Tipo de ganchos remotos:
  - Alto nivel: función exportada por una DLL.
  - Bajo nivel: función contenida en el proceso.
- El proceso es notificado antes de que el SO tenga oportunidad de procesar el evento.

Keyloggers que usan ganchos

- Tipos:
  - Alto nivel: `WH_KEYBOARD` puede ejecutarse en un proceso remoto o en el instalador del gancho.
  - Bajo nivel: `WH_KEYBOARD_LL` se ejecuta en el contexto del proceso que instala el gancho.
- Permite interceptar y alterar las pulsaciones antes de almacenarlas y/o pasarlas al proceso.

Uso de `SetWindowsHookEx`

- Parámetros de `SetWindowsHookEx`:
  - `idHook`: tipo de mensaje.
  - `lpfn`: dirección del procedimiento manejador.
  - `hMod`: localización del código
    alto nivel: DLL.
    bajo nivel: módulo local.
  - `dwThreadId`: hebra a la que asociar el mensaje (0 == todas).
- El único requisito del código del gancho es que llame a `CallNextHookEx` para no interrumpir la cadena.

Hebras objetivo de los ganchos

- El malware no suele crear ganchos en todas las hebras ni utilizar ganchos con mensajes de uso muy frecuente...
  - para no degradar el rendimiento del sistema.
  - para no atraer la atención de los IPS.
  - para permanecer oculto.
- Suele utilizarse mensajes de uso poco frecuente (`WH_CBT`).
- Patrón del ataque:
  - `LoadLibray`: localiza una DLL.
  - `GetProcAddress`: localiza un procedimiento de una DLL.
  - Conseguir el identificador de la hebra víctima.
  - `SetWindowsHookEx`: añade el gancho a la cadena.
- Una vez inyectado se ejecuta `DllMain` disfrazado del proceso víctima.
- Es típico desengancharse para evitar interferir el funcionamiento normal con `UnhookWindowsHookEx`.

Hebras objetivo de los ganchos (2)

```
00401100        push    esi
00401101        push    edi
00401102        push    offset LibFileName ; "hook.dll"
00401107        call    LoadLibraryA
0040110D        mov     esi, eax
0040110F        push    offset ProcName ; "MalwareProc"
00401114        push    esi             ; hModule
00401115        call    GetProcAddress
0040111B        mov     edi, eax
0040111D        call    GetNotepadThreadId
00401122        push    eax             ; dwThreadId
00401123        push    esi             ; hmod
00401124        push    edi             ; lpfn
00401125        push    WH_CBT   ; idHook
00401127        call    SetWindowsHookExA
```

Listing 12-4: Hook injection, assembly code

Desvíos (*"Detours"*)

- Biblioteca desarrollada por Microsoft para extender la funcionalidad de SO y aplicaciones.
- Objetivos:
  - Modificar la tabla de importaciones.
  - Añadir DLLs a programas.
  - Añadir ganchos a procesos en ejecución.
- Para conseguirlo se crea una sección nueva: `.detour`
- No siempre se utiliza la biblioteca oficial para conseguirlo.

Inyección APC (*"Asyncronous Procedure Call"*)

- Capacidad de ejecutar una función sobre una hebra.
- Toda hebra tiene una cola de APCs.
- Se procesan...
  - al pasar al estado *"alertable"*.
  - al invocar `WaitForSingleObjectEx`, `WaitForMultipleObjectEx` o `Sleep`.
- Toda hebra ejecuta las APCs de su cola antes de retomar su ejecución normal.
- Tipos:
  - Kernel-mode: APC generada para el sistema o controladores.
  - User-mode: APC generada para una aplicación.

Inyección APC desde espacio de usuario

- Cualquier hebra puede encolar la ejecución de una función en otra hebra.
- El malware busca procesos con hebras en estado de alerta.
- `WaitForSingleObjectEx`: llamada más común de la API.
- `QueueUserAPC`:

```
DWORD QueueUserAPC(
  [in] PAPCFUNC  pfnAPC,  // puntero a la función a ejecutar
  [in] HANDLE    hThread, // manejador de la hebra víctima
  [in] ULONG_PTR dwData); // parámetros para la función
```

- Localizar código que liste hebras y procesos del sistema:
  - `CreateToolhelp32Snapshot`, `{Process,Thread}32{First,Next}`
- Ejemplo de carga de DLL:

```
QueueUserAPC(LoadLibrary, OpenThread(...), DLL)
```

Inyección APC desde espacio de ususario (2)

```
00401DA9        push    [esp+4+dwThreadId]      ; dwThreadId
00401DAD        push    0                       ; bInheritHandle
00401DAF        push    10h                     ; dwDesiredAccess
00401DB1        call    ds:OpenThread ❶
00401DB7        mov     esi, eax
00401DB9        test    esi, esi
00401DBB        jz      short loc_401DCE
00401DBD        push    [esp+4+dwData]          ; dwData = dbnet.dll
00401DC1        push    esi                     ; hThread
00401DC2        push    ds:LoadLibraryA ❷       ; pfnAPC
00401DC8        call    ds:QueueUserAPC
```

Listing 12-5: APC injection from a user-mode application

Inyección APC desde espacio del núcleo

- Tiene el mismo objetivo, ejecutar código.
- El malware en el núcleo necesita ejecutar código en espacio de usuario.
- Víctima habitual: `svchost.exe`
- Código habitual: shellcode.
- Windows API:
  - `KeInitializeApc`
  - `KeInsertQueueApc`

Ejercicios: Lab 12-1

Analice los programas Lab12-01.exe y Lab12-01.dll. Asegúrese de colocarlos en el mismo directorio al analizarlos.

1. ¿Qué ocurre al ejecutar el programa?
2. ¿Qué proceso está siendo inyectado?
3. ¿Cómo detener la aparición de ventanas emergentes?
4. ¿Cómo funciona este malware?

Ejercicios: Lab 12-2

Analice el programa Lab12-02.exe.

1. ¿Cuál es el propósito de este programa?
2. ¿Cómo esconde su lanzador su ejecución?
3. ¿Dónde está almacenada su carga maliciosa?
4. ¿Cómo está protegida la carga maliciosa?
5. ¿Cómo se han protegido las cadenas?