Adversary emulation: anatomia de un ciberataque

Tu Nombre

12 de noviembre de 2025

Índice general

Jugez un homme par ses questions plutôt que par ses réponses.

— Voltaire

Capítulo 1

Introducción

¿Porque alguien deberia leer este trabajo?

1.1. Anatomia de un adversario

Que es un adverdario

1.2. Objetivos del trabajo

El objetivo de este trabajo es desarrollar un escenario didáctico que permita a los analistas de ciberseguridad mejorar sus capacidades de deteccion e investigación mediante la comprensión del proceso completo de creación de un ciberataque. El resultado final del proyecto será un conjunto de artefactos forenses de los sistemas infectados, que servirá como base para ejercicios de análisis forense (blue team).

Para generar dicho recurso, el núcleo del trabajo se centrará en la concepción, diseño y desarrollo de un ciberataque realista, su ejecución controlada en un sistema víctima y la documentación íntegra del proceso ofensivo llevado a cabo. De este modo, quien reciba los artefactos forenses podrá analizar la infección desde el punto de vista del analista forense, pero también interpretar las acciones del atacante, entendiendo mejor el cómo y el porqué de cada artefacto encontrado.

En definitiva, este trabajo busca aproximar al analista a la mentalidad del atacante, con el fin de reforzar sus habilidades de detección, interpretación y respuesta ante amenazas reales.

Capítulo 2

Marco Teórico

¿Como la industria consigue tener un lenguaje comun para poder transmitir informacion sobre adversarios de forma efectiva?

2.1. MITRE ATT&CK

el procedimiento debe ser lo suficientemente detallaco como para que el blue team pueda crear una regla de deteccion y el red team pueda recrearlo

2.2. Cyber Kill Chain (CKC)

el framework consiste en dividir un ciberatque en las siguientes fases:

1. Reconocimiento: investigacion del objetivo
2. Armazenamiento: creacion del payload
3. Entrega: envio del payload al objetivo
4. Explotacion: ejecucion del payload
5. Instalacion: sistema de staging
6. Comando y control: establecimiento del canal de comunicacion
7. Acciones sobre objetivos: acciones finales

2.3. Threat Intelligence

2.4. Adversary simulation

Capítulo 3

Creando nuestro adversario

El atacante logra la ejecución de código en la máquina objetivo e inicia el stage 1. Este primer componente se conecta a un servidor controlado por el adversario para descargar el stage 2.

En algunos escenarios, entre ambos stages, se realizan tareas de reconocimiento interno para recopilar información sobre la víctima (por ejemplo, enumeración de usuarios, servicios o topología de red).

El stage 2 consiste en una DLL maliciosa que se hace pasar por una biblioteca legítima del sistema. Esta DLL actúa como proxy de la original, garantizando que las aplicaciones que dependen de ella funcionen correctamente sin generar errores, mientras que, en paralelo, carga en memoria un implante de Sliver.

Una vez inyectado el implante, este ejecuta acciones de escaneo interno, movimiento lateral y establece comunicación periódica (beaconing) con el servidor de comando y control (C2). Cuando el atacante lo considere oportuno, se activará el payload de ransomware, procediendo al cifrado de archivos en el dispositivo comprometido.

3.1. Fase 1. Reconocimiento

Ver el análisis detallado en la Sección 4.1.

3.2. Fase 2. Armamento

Ver el análisis detallado en la Sección 4.2.

3.3. Fase 3: Entrega

Ver el análisis detallado en la Sección 4.3.

Táctica: Initial Access (TA0001) [1].

Técnica: Replication Through Removable Media (T1091) [2].

Procedimiento: Se emplea un dispositivo *BadUSB* que, al ser conectado, emula un dispositivo de interfaz humana (HID) para inyectar y ejecutar un script de PowerShell.

3.4. Fase 4: Explotación

Ver el análisis detallado en la Sección 4.4.

Táctica: Execution (TA0002) [3].

Técnica: Command and Scripting Interpreter: PowerShell (T1059.001) [4].

Procedimiento: El script inicial utiliza técnicas de *marshalling* para evadir la Interfaz de Escaneo Antimalware (AMSI) de PowerShell, permitiendo la ejecución de código malicioso en memoria.

Táctica: Command and Control (TA0011) [5].

Técnica: Ingress Tool Transfer (T1105) [6].

Procedimiento: El *payload* inicial se conecta a un servidor C2 externo para descargar la siguiente fase (*stage 2*).

3.5. Fase 5: Instalación y Persistencia

Ver el análisis detallado en la Sección 4.5.

Tácticas:
- Persistence (TA0003)
- Defense Evasion (TA0005)

Técnica: Hijack Execution Flow: DLL Search Order Hijacking (T1574.001) [7].

Procedimiento: La DLL (*stage 2*) se escribe en disco en una ubicación específica para ser cargada por un proceso legítimo y vulnerable a secuestro de DLL, asegurando la persistencia. Se consultan repositorios como [8] para identificar binarios vulnerables.

Táctica: Defense Evasion (TA0005).

Técnica: Reflective Code Loading (T1620) [9].

Procedimiento: El *stage 2* (la DLL secuestrada) actúa como un cargador (*loader*) que mapea reflectivamente el *payload* final directamente en la memoria del proceso anfitrión, evitando así las llamadas estándar a `LoadLibrary` y la detección basada en módulos cargados.

3.6. Fase 6: Mando y Control (C2)

Ver el análisis detallado en la Sección 4.6.

Táctica: Command and Control (TA0011) [5].

Técnicas:
- Application Layer Protocol: Web Protocols (T1071.001) [10].
- Encrypted Channel (T1573) [11].

Procedimiento: El implante final establece un canal de comunicación (*beaconing*) con el servidor C2. Esta comunicación se tuneliza sobre HTTPS (T1071.001) y utiliza un canal de cifrado (T1573) para ocultar las órdenes y la exfiltración de datos, mimetizando el tráfico web legítimo.

3.7. Fase 7: Acciones sobre el Objetivo

Ver el análisis detallado en la Sección 4.7.

Táctica: Impact (TA0040) [12].

Técnica: Data Encrypted for Impact (T1486) [13].

Procedimiento: El *payload* final ejecuta su objetivo principal: cifrar los archivos del sistema de la víctima y solicitar un rescate para su recuperación.

Capítulo 4

Analizando al adversario

4.1. Análisis del Reconocimiento

4.2. Análisis del Armamento

4.3. Análisis de la Entrega

4.4. Análisis de la Explotación

4.5. Análisis de la Instalación y Persistencia

4.6. Análisis del C2

4.7. Análisis de las acciones sobre el objetivo

Capítulo 5

Anexo A

5.1. Tipos de Malware

RAT ransomware reverse shell

5.2. Listado de EDRs

Sylantstrike puede ser utilizado para hacer pruebas de concepto de EDRs

- CrowdStrike Falcon
- Microsoft Defender for Endpoint

Capítulo 6

Anexo B: Windows Internals

6.1. Memoria Virtual

6.1.1. Problemas Memoria Fisica

- Escasez de memoria
- Fragmentación
- Acceso a memoria (Seguridad)

6.1.2. Acceso indirecto a memoria

Cada proceso tiene su propio espacio de direcciones virtual, Para el proceso es como tener un espacio de momoria fisica contiguo reservado, Pero por detras el SO esta mapeando esas direcciones virtuales a distintas regiones de memoria fisica.

6.1.3. Paginación

Mapear cada byte de la memoria virtual a un byte de la moria fisica seria muy ineficiente, Se crean paginas de moria que son bloques de bytes para simplificar este proceso (normalmente de 4KB | 4096 bytes),

page - memoria virtual frame - memoria fisica

Cuando un proceso se carga sus paginas son cargadas en frames disponibles en moria fisica

6.1.4. Tabla de paginas

Cada proceso tiene una tabla de paginas que mapea las direcciones virtuales a las fisicas, La tabla de paginas es gestionada por la MMU (Memory Management Unit), Para cada pagina se mantiene, su numero de frame, permisos, etc.

6.1.5. MMU

La CPU tiene que accedr a la RAM primero para consultar la tabla de paginasy Despues para acceder a la memoria fisica, para cada lectura tenemos que hacer 2 accesos a memoria, Para optimizar esto, la MMU, hardaware especifico de la CPU, se encarga de traducir las direcciones virtuales a fisicas, la MMU tiene un caché de la tabla de paginas llamado TLB (Translation Lookaside Buffer), La MMU tambien es la encargada de gestion que un proceso no acceda a memoria que no le pertenece,

6.1.6. Acceso a memoria

Cuando un proceso quiere acceder a una dirección de memoria, la CPU consulta la MMU, En la MMU la direccon se divide en dos partes, el numero de pagina y el offset dentro de la pagina,

Pege number - los bits mas significativos de la direccion virtual Page offset - los bits menos significativos de la direccion virtual

Ejemplo: En una pagina de 4KB (4096 bytes), el offset ocupa los 12 bits menos significativos, El resto de bits son los mas significativos y representan el numero de pagina,

6.2. Process

6.2.1. EPROCESS

Estructura que mantiene toda la información de un proceso en el kernel.

6.2.2. Threads

Representan las hebras de ejecución asociadas al proceso.

6.2.3. Handles

Son equivalentes a los *file descriptors* en Linux.

6.2.4. Memory

Gestión del espacio de direcciones y memoria asignada al proceso.

6.2.5. Modules

Lista de módulos y librerías cargadas en el proceso.

6.3. Process Creation (Kernel)

La creación de un proceso en Windows sigue una serie de pasos específicos a nivel del kernel.

(1) Inicialización del espacio de direcciones

- **Modelo en Linux vs. Windows:** En Linux, los nuevos procesos se crean mediante la llamada `fork`, mientras que en Windows los procesos se inicializan completamente desde cero.
- **Mapeo de `KUSER_SHARED_DATA`:** Al crear un nuevo proceso, el kernel debe mapear memoria. Una de las primeras acciones es mapear la página de 4KB denominada `KUSER_SHARED_DATA`, la cual permite intercambiar información entre *user mode* y *kernel mode* sin necesidad de *syscalls*. En esta página se encuentran datos como el reloj del sistema y las rutas a directorios del sistema.

- **Mapeo del ejecutable:** Posteriormente, el kernel carga el ejecutable dentro del proceso. Concretamente, carga el código PE en la sección `.text`.
- **Mapeo de `ntdll.dll`:** El kernel carga el módulo `ntdll.dll`, el cual actúa como intermediario entre el *user-mode* y el kernel. Este módulo contiene las funciones necesarias para la comunicación con el kernel y cumple un rol similar al `ld.so` en Linux.
- **Asignación del PEB (Process Environment Block):** El `PEB` es una estructura de datos en *user mode* (aproximadamente 1–2 páginas de memoria) que contiene:
  - Variables de entorno.
  - Línea de comandos completa con la que se inició el proceso.
  - Directorio de trabajo.
  - Lista de módulos cargados (`Ldr`).
  - Punteros al `stack` y `heap`.
  - Dirección base de la imagen.

(2) Creación del hilo inicial

- **Mapeo del stack:** Se asigna el espacio de pila necesario para el hilo principal.
- **Mapeo del TEB (Thread Environment Block):** El `TEB` es una estructura pequeña (2 páginas aprox.) que almacena información específica de cada hilo. Su función es equivalente al `PEB`, pero a nivel de hilo.
- **Inicialización del puntero de ejecución:** Finalmente, el puntero de ejecución se coloca en la función `ntdll.LdrInitializeThunk`, encargada de completar la carga dinámica del proceso.

6.4. Portable Executable (PE)

El formato *Portable Executable* (PE) es el equivalente en Windows al formato ELF en Linux. Se utiliza para representar ejecutables, bibliotecas dinámicas (DLL) y otros ficheros ejecutables en dicho sistema operativo.

6.4.1. Sections

Son el equivalente a los segmentos en un ELF. Las secciones indican cómo debe ubicarse la memoria dentro del proceso, definiendo distintas regiones con protecciones específicas (lectura, escritura, ejecución).

6.4.2. Imports

Contiene la lista de `DLLs` que son dependencias del ejecutable.

6.4.3. Exports

Si el archivo PE corresponde a una biblioteca compartida y exporta alguna funcionalidad, esta información se registra en la sección de exportaciones. A diferencia de Linux, donde las funciones suelen exportarse por defecto, en Windows deben exportarse explícitamente.

6.4.4. Relocations

Al habilitar `ASLR` (Address Space Layout Randomization), el ejecutable puede necesitar ser reubicado en memoria. Las direcciones a modificar en dicho proceso quedan registradas en esta sección.

6.4.5. AddressOfEntryPoint

Campo que indica la dirección de inicio de la ejecución del programa.

6.4.6. Subsystem

Especifica en qué entorno debe ejecutarse el binario: si será una aplicación de consola, una GUI, etc.

6.4.7. Virtual Address

Dirección relativa que tendrá la sección al cargarse en memoria. Siempre se calcula como:

$$\text{Dirección en memoria} = \text{ImageBase} + \text{Virtual Address} \tag{6.1}$$

6.4.8. Raw Data

Indica la dirección dentro del archivo PE donde se encuentra físicamente la sección en disco.

6.5. Secciones de un proceso

El espacio de direcciones de un proceso en memoria se organiza en diferentes secciones. A continuación, se enumeran las mismas desde las direcciones más altas hasta las más bajas:

Páginas del Kernel
: Ocupan la misma dirección en todos los procesos del sistema y corresponden al kernel. Como todas las páginas de kernel apuntan a los mismos *frames* de memoria física, no se desperdicia espacio cargando el kernel en cada proceso. El acceso se realiza únicamente mediante llamadas al sistema.

Stack (lectura y escritura)
: Estructura de almacenamiento *LIFO*, que crece hacia abajo (desde el final del espacio de direcciones del proceso hacia la sección BSS). Cada función tiene su propio *stack frame*, con variables locales y argumentos. El tamaño suele estar limitado entre 1 y 8 MB, según la arquitectura y el sistema operativo. Su gestión mediante el puntero de pila es muy rápida.

Heap (lectura y escritura)
: Espacio para memoria dinámica gestionada por llamadas como `malloc`. Crece hacia arriba, desde el final de la BSS hasta el final del espacio de direcciones. Su administración mediante el puntero de *heap* es más lenta que la del *stack*.

BSS y Data (lectura y escritura)
: Almacenan las variables globales y estáticas del programa. La sección **BSS** contiene las variables no inicializadas, mientras que la sección **Data** guarda las variables inicializadas.

Text (lectura y ejecución)
: Contiene el código ejecutable del programa, cargado desde el binario en disco. Generalmente está marcada como de solo lectura y ejecución.

6.6. DLL (Dynamic-Link Library)

En Windows, un proceso puede depender de múltiples librerías dinámicas (`DLLs`) para ejecutar ciertas funcionalidades. Estas DLLs se cargan en el espacio de direcciones del proceso y su interacción se realiza a través de estructuras específicas que Windows mantiene en memoria.

6.6.1. Carga de DLLs en un proceso

Cuando un proceso arranca, indica qué DLLs necesita y qué funciones de cada DLL va a utilizar. Windows realiza los siguientes pasos:

1. Verifica si la DLL ya está cargada en memoria; si no lo está, la carga en el espacio de direcciones del proceso.
2. Si ya existe en memoria, se reutiliza la misma copia en memoria física, pero cada proceso mantiene su propio mapeo en su espacio de direcciones virtuales.
3. Localiza dentro de la DLL la función solicitada por el proceso.
4. Proporciona al proceso la dirección en memoria de la función, de manera que cada llamada a esa función realmente salta a la dirección correspondiente dentro de la DLL cargada.

Cada DLL se carga como un módulo completo, incluyendo sus secciones `.text`, `.rdata`, `.data`, etc., mapeadas en memoria contigua a partir de la *base address* del módulo.

6.6.2. PEB y la lista de módulos

El **Process Environment Block (PEB)** es una estructura interna de Windows que contiene información sobre el proceso, incluyendo los módulos (DLLs) cargados.

Dentro del PEB, el campo:

```
PEB->Ldr->InMemoryOrderModuleList
```

es una lista enlazada que mantiene todos los módulos cargados en memoria, incluyendo la DLL principal del proceso y todas las DLLs dependientes. Cada entrada de esta lista contiene:

- La `base address` del módulo en memoria.

- El nombre del archivo de la DLL.
- Información sobre sus secciones.

Recorrer esta lista permite obtener todas las DLLs cargadas y sus direcciones en memoria.

6.6.3. Import Address Table (IAT)

Para que un proceso llame a funciones de una DLL, Windows utiliza la **Import Address Table (IAT)**. La IAT contiene punteros a las funciones importadas por el proceso. Durante la carga:

1. Windows localiza cada función solicitada dentro de la DLL correspondiente.
2. Escribe en la IAT la dirección en memoria de la función.

Cuando el proceso ejecuta una llamada a una función importada, en realidad está saltando a la dirección contenida en la IAT, que apunta a la DLL cargada en memoria.

6.6.4. Resumen

En conjunto, el PEB y la IAT permiten a un proceso interactuar de manera eficiente con sus DLLs:

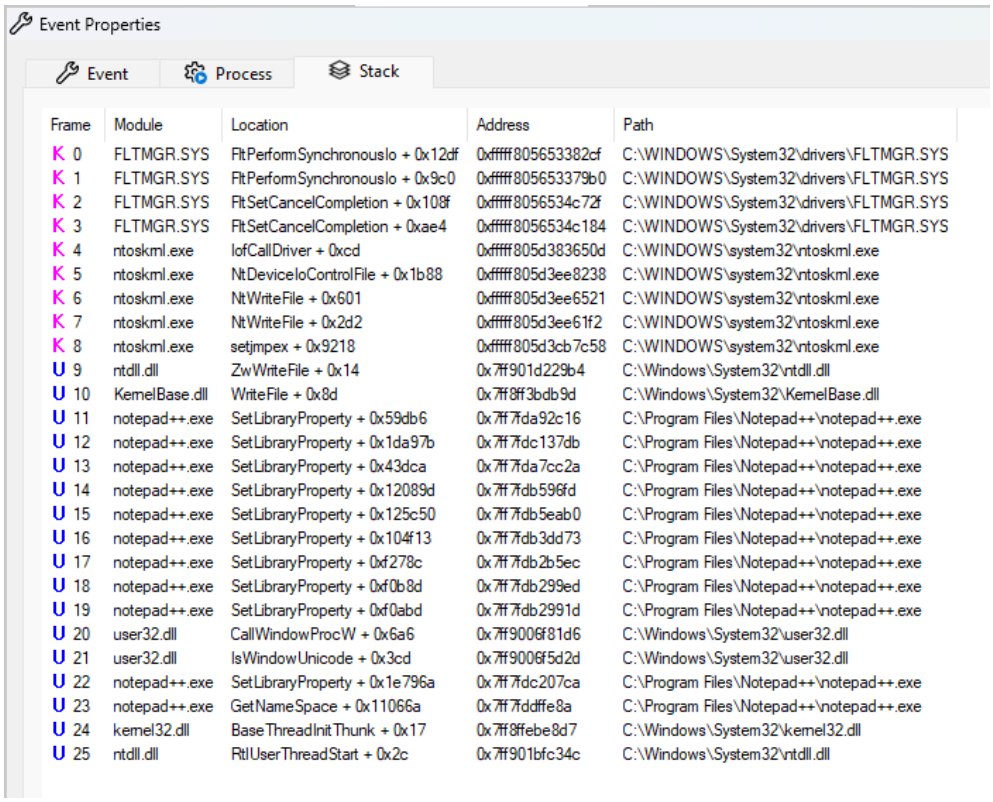
- El PEB mantiene un registro de todos los módulos cargados y sus direcciones.
- La IAT traduce las llamadas a funciones de la DLL a direcciones concretas en memoria.

Esto garantiza que el proceso pueda utilizar código externo (DLLs) sin necesidad de incorporarlo estáticamente en su binario, manteniendo modularidad y eficiencia en el uso de memoria.

Flujo de llamadas: User Mode vs Kernel Mode

1. **User Mode (U) - Aplicación y librerías de alto nivel**
   - notepad++.exe: SetLibraryProperty (operaciones internas de la aplicación)
   - user32.dll: CallWindowProcW, IsWindowUnicode (interacción con interfaz Windows)
   - KernelBase.dll: WriteFile + 0x8d (función de usuario de alto nivel)
   - ntdll.dll: ZwWriteFile (interfaz hacia la syscall)
2. **Kernel Mode (K) - Operaciones de bajo nivel y hardware**
   - FLTMGR.SYS: FltPerformSynchronousIo, FltSetCancelCompletion (filtro de sistema de archivos)
   - ntoskrnl.exe: IoCallDriver, NtDeviceIoControlFile, NtWriteFile (operaciones sobre drivers y disco)

Interpretación: El flujo comienza en User Mode con la aplicación y librerías de alto nivel, y termina en Kernel Mode ejecutando operaciones de escritura en disco con privilegios elevados.



6.7. Win32 API

Nombre de la DLL	Tareas de la DLL
User32.dll	Esta biblioteca contiene funciones para crear ventanas, manejar mensajes y procesar la entrada del usuario.
Kernel32.dll	Esta biblioteca proporciona acceso a una variedad de servicios esenciales del sistema como la gestión de memoria, operaciones de E/S y la creación de procesos e hilos.
Gdi32.dll	Esta biblioteca contiene funciones para dibujar gráficos y mostrar texto.
Comdlg32.dll	Esta biblioteca proporciona diálogos comunes como los diálogos de abrir y guardar.
Advapi32.dll	Esta biblioteca proporciona funciones para trabajar con el registro de Windows y gestionar cuentas de usuario.

Cuadro 6.1: Descripción de las principales DLLs de la Win32 API

6.8. SO Security

6.8.1. ASLR (Address Space Layout Randomization)

6.8.2. DEP (Data Execution Prevention)

6.8.3. Control Flow Guard (CFG)

6.8.4. Code Integrity Guard (CIG)

6.8.5. Protected Process Light (PPL)

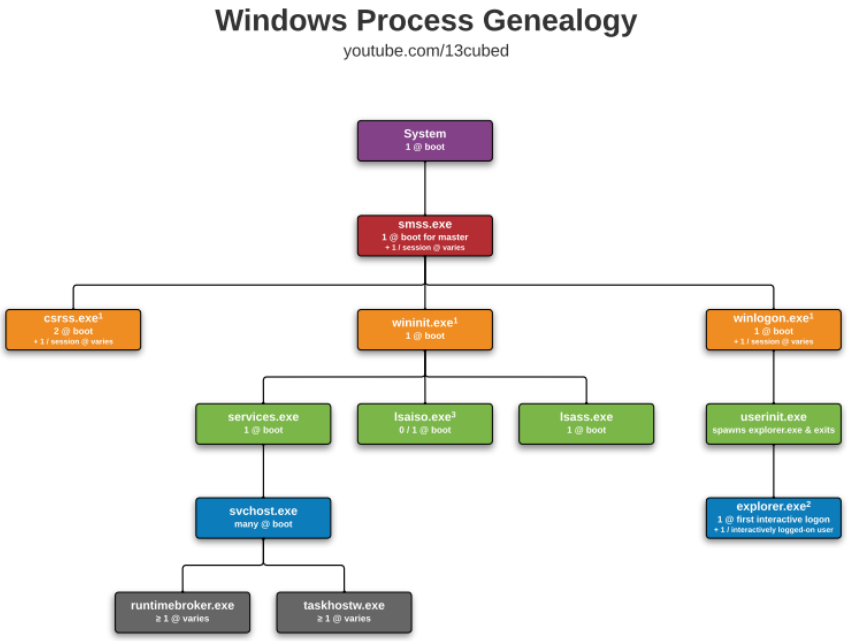
6.8.6. Virtualization-Based Security (VBS)

6.8.7. Kernel-mode code signing (KMCI)

6.9. Genealogía de Procesos en Windows

La genealogía de procesos en Windows representa la jerarquía de creación de procesos desde el arranque del sistema hasta el inicio de sesión del usuario. Comprender esta estructura es esencial para el análisis forense, la respuesta a incidentes y la detección de malware, ya que permite identificar comportamientos anómalos en los procesos del sistema.

A continuación se muestra la genealogía de procesos típica en Windows:



Fuente: `youtube.com/13cubed`

Nivel 0: Proceso raíz

- **System**: Primer proceso de espacio de usuario iniciado por el kernel. Tiene un PID fijo (normalmente 4) y no tiene padre.

Nivel 1: Session Manager

- **smss.exe (Session Manager Subsystem)**: Primer proceso real del espacio de usuario. Se encarga de iniciar las sesiones del sistema, lanzar procesos críticos como `csrss.exe`, `wininit.exe` y `winlogon.exe`.

Nivel 2: Procesos críticos del sistema

- **csrss.exe**: Client/Server Runtime Subsystem. Se encarga de funciones esenciales como la gestión de la consola y la creación de procesos. Existe una instancia por sesión.
- **wininit.exe**: Windows Initialization Process. Lanza procesos esenciales como `services.exe`, `lsaiso.exe` y `lsass.exe`.
- **winlogon.exe**: Gestiona la autenticación del usuario y se mantiene activo durante la sesión interactiva.

Nivel 3: Procesos del sistema

- **services.exe**: Service Control Manager. Se encarga de iniciar y gestionar los servicios del sistema, incluyendo los alojados por `svchost.exe`.
- **lsaiso.exe**: Proceso de seguridad aislado que implementa funciones de cifrado y autenticación en versiones modernas de Windows (opcional).
- **lsass.exe**: Local Security Authority Subsystem Service. Encargado de políticas de seguridad, autenticación y gestión de credenciales.

Nivel 4: Servicios alojados y utilidades del sistema

- **svchost.exe**: Proceso contenedor que aloja múltiples servicios del sistema. Existen muchas instancias según el grupo de servicios que aloje.
- **runtimebroker.exe** / **taskhostw.exe**: Procesos auxiliares para la ejecución de aplicaciones y tareas programadas.

Procesos del usuario interactivo

- **userinit.exe**: Iniciado por `winlogon.exe` tras la autenticación. Lanza el shell principal del usuario y termina.
- **explorer.exe**: Shell gráfico de Windows que representa el escritorio, la barra de tareas y el menú de inicio. Es el proceso raíz del entorno del usuario.

6.10. Debugging flags

6.11. ABI

Calling Convention

- Pasar parámetros Por pila, de derecha a izquierda (el argumento de mas a la izquierda quedara en la direccion mas baja) - Limpieza de pila La función llamada (callee) limpia la pila - Registro de retorno EAX

Bibliografía

[1] The MITRE Corporation, "Initial access - tactic ta0001," MITRE ATT&CK®, 2024, accedido: 11 de noviembre de 2025. [Online]. Available: https://attack.mitre.org/tactics/TA0001/

[2] ——, "Replication through removable media - technique t1091," MITRE ATT&CK®, 2024, accedido: 11 de noviembre de 2025. [Online]. Available: https://attack.mitre.org/techniques/T1091/

[3] ——, "Execution - tactic ta0002," MITRE ATT&CK®, 2024, accedido: 11 de noviembre de 2025. [Online]. Available: https://attack.mitre.org/tactics/TA0002/

[4] ——, "Command and scripting interpreter: Powershell - technique t1059.001," MITRE ATT&CK®, 2024, accedido: 11 de noviembre de 2025. [Online]. Available: https://attack.mitre.org/techniques/T1059/001/

[5] ——, "Command and control - tactic ta0011," MITRE ATT&CK®, 2024, accedido: 11 de noviembre de 2025. [Online]. Available: https://attack.mitre.org/tactics/TA0011/

[6] ——, "Ingress tool transfer - technique t1105," MITRE ATT&CK®, 2024, accedido: 11 de noviembre de 2025. [Online]. Available: https://attack.mitre.org/techniques/T1105/

[7] ——, "Hijack execution flow: Dll search order hijacking - technique t1574.001," MITRE ATT&CK®, 2024, accedido: 11 de noviembre de 2025. [Online]. Available: https://attack.mitre.org/techniques/T1574/001/

[8] HijackLibs, "Hijack libs - project to find hijackable libraries," Página Web, 2024, accedido: 11 de noviembre de 2025. [Online]. Available: https://hijacklibs.net/

[9] The MITRE Corporation, "Reflective code loading - technique t1620," MITRE ATT&CK®, 2024, accedido: 11 de noviembre de 2025. [Online]. Available: https://attack.mitre.org/techniques/T1620/

[10] ——, "Application layer protocol: Web protocols - technique t1071.001," MITRE ATT&CK®, 2024, accedido: 11 de noviembre de 2025. [Online]. Available: https://attack.mitre.org/techniques/T1071/001/

[11] ——, "Encrypted channel - technique t1573," MITRE ATT&CK®, 2024, accedido: 11 de noviembre de 2025. [Online]. Available: https://attack.mitre.org/techniques/T1573/

[12] ——, "Impact - tactic ta0040," MITRE ATT&CK®, 2024, accedido: 11 de noviembre de 2025. [Online]. Available: https://attack.mitre.org/tactics/TA0040/

[13] ——, "Data encrypted for impact - technique t1486," MITRE ATT&CK®, 2024, accedido: 11 de noviembre de 2025. [Online]. Available: https://attack.mitre.org/techniques/T1486/