

# Trabajo de Fin de Máster

Tu Nombre

3 de agosto de 2025

# Índice general

<b>1. Introducción</b>	<b>2</b>
<b>2. EDRs Monitorización</b>	<b>3</b>
2.1. Kernell callbacks . . . . .	3
2.2. Event tracing for Windows (ETW) . . . . .	3
2.3. Antimalware scan interface (AMSI) . . . . .	3
2.4. System call monitoring . . . . .	4
2.5. monitorizacion de memoria . . . . .	4
<b>3. Anexos</b>	<b>5</b>
3.1. Malware . . . . .	5
3.1.1. Empaquetadores . . . . .	5
3.1.2. Cargadores . . . . .	5
3.1.3. Shellcodes . . . . .	5
3.2. Proceso . . . . .	5
3.2.1. Secciones de un proceso . . . . .	5
3.2.2. ASLR - Address Space Layout Randomization . . . . .	6
3.3. Ring level . . . . .	6
3.4. Memoria Virtual . . . . .	6
3.4.1. Problemas Memoria Fisica . . . . .	6
3.4.2. Acceso indirecto a memoria . . . . .	7
3.4.3. Paginación . . . . .	7
3.4.4. Tabla de paginas . . . . .	7
3.4.5. MMU . . . . .	7
3.4.6. Acceso a memoria . . . . .	7
3.5. Win32 API . . . . .	8
3.6. Genealogía de Procesos en Windows . . . . .	8
3.7. Listado de EDRs . . . . .	11

# Capítulo 1

## Introducción

Aquí se presenta la introducción del trabajo, donde se contextualiza el tema y se establecen los objetivos de la investigación.

crearemos los bypass y los intentaremos detectar con herramientas como:

yara volatility

ctf:

ransomware que tiene la clase hardcodeada, tienes o que encontrar el malware y detenerlo antes de que se ejecute o bien una vez ejecutado de un dump de memoria conseguir la clave

# Capítulo 2

## EDRs Monitorización

- kernell callbacks
- event tracing for Windows (ETW)
- antimalware scan interface (AMSI)
- system call monitoring

### 2.1. Kernell callbacks

El edr mantendra un driver en kernel land

El edr recibira callbacks segun eventos de (Procesos, Hilos, DLLs)

### 2.2. Event tracing for Windows (ETW)

Registro de eventos recopila y analizar información sobre el funcionamiento del sistema operativo y las aplicaciones.

### 2.3. Antimalware scan interface (AMSI)

AMSI es una API genérica de escaneo antimalware que permite que las aplicaciones (por ejemplo, PowerShell, WSH, VBScript, Office macros...) envíen contenido potencialmente malicioso a un proveedor antivirus registrado para su análisis antes de ser ejecutado.

## 2.4. System call monitoring

El edr pone hooks en las llamadas de win32api a syscalls pone un jmp y donde debería saltar a la syscall salta al edr de modo que lo analiza antes de que se ejecute

## 2.5. monitorizacion de memoria

yara

# Capítulo 3

## Anexos

### 3.1. Malware

#### 3.1.1. Empaquetadores

#### 3.1.2. Cargadores

#### 3.1.3. Shellcodes

shelcodeless rop

### 3.2. Proceso

Cada proceso tiene su propio espacio de direcciones virtual, Dentro de ese espacio un proceso esta dividido en secciones,

#### 3.2.1. Secciones de un proceso

Listado en orden de mayor direcciones de memoria a menor:

#### Paginas del Kernel

Esta seccion ocupa lo mismo en todos los procesos del sistema, y corresponde a todas las paginas que contienen el kernel del sistema operativo, Como todas las paginas de kernel de todos los procesos hacen referencia a los mismos frames de memoria fisica, no se desperdicia espacio cargando el kernel en cada proceso,

Para acceder a la memoria del kernel, un proceso tiene que hacer una llamada al sistema,

### **Stack - lectura y escritura**

Almacenamiento simple LIFO Crece hacia abajo, desde el final del espacio de direcciones del proceso hasta el final de la BSS,

Cada funcion tiene su propio stack frame, que contiene las variables locales y los argumentos de la funcion,

Limitacion tipica entre 1 y 8MB, dependiendo de la arquitectura y el sistema operativo,

Utiliza el puntero de pila para reservar y liberar espacio en el stack (muy rapido)

### **Heap - lectura y escritura**

Malloc, reserva memoria de forma dinamica Crece hacia arriba, desde el final de la BSS hasta el final del espacio de direcciones del proceso

Utiliza el puntero de heap para reservar y liberar espacio en el heap (más lento que el stack)

### **BSS y Data- lectura y escritura**

Variables estaticas y globales del programa,

### **Text - lectura y ejecución**

Es cargado desde el ejecutable binario desde disco

## **3.2.2. ASLR - Address Space Layout Randomization**

## **3.3. Ring level**

user mode vs kernel mode

idea: pillar un programa como print hello world

desde proc mon ver todas las llamadas que hace e ir comentando cada biblioteca en que nivel esta

## **3.4. Memoria Virtual**

### **3.4.1. Problemas Memoria Fisica**

- Escasez de memoria
- Fragmentación

- Acceso a memoria (Seguridad)

### 3.4.2. Acceso indirecto a memoria

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

### 3.4.3. Paginación

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

page - memoria virtual frame - memoria física

Cuando un proceso se carga sus paginas son cargadas en frames disponibles en memoria física

### 3.4.4. Tabla de paginas

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

### 3.4.5. MMU

La CPU tiene que acceder a la RAM primero para consultar la tabla de paginas y Despues para acceder a la memoria física, para cada lectura tenemos que hacer 2 accesos a memoria, Para optimizar esto, la MMU, hardware especifico de la CPU, se encarga de traducir las direcciones virtuales a físicas, 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,

### 3.4.6. Acceso a memoria

Cuando un proceso quiere acceder a una dirección de memoria, la CPU consulta la MMU, En la MMU la direccion 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,

### 3.5. 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 3.1: Descripción de las principales DLLs de la Win32 API

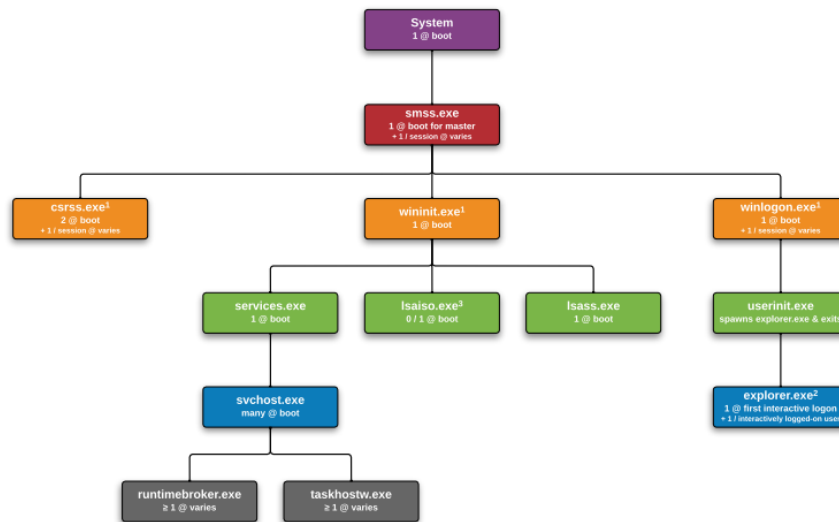
### 3.6. Genealogía de Procesos en Windows

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:

## Windows Process Genealogy

youtube.com/13cubed



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.

## Importancia en Ciberseguridad

Comprender esta estructura es fundamental para:

- Identificar procesos anómalos o fuera de lugar.
- Detectar técnicas de evasión como *Process Injection* o *Parent PID Spoofing*.

- Realizar análisis forense de procesos mediante herramientas como Sysmon, EDR, Process Explorer, o Zeek.

### **3.7. Listado de EDRs**

Sylantstrike puede ser utilizado para hacer pruebas de concepto de EDRs

- CrowdStrike Falcon
- Microsoft Defender for Endpoint