

# Trabajo de Fin de Máster

Tu Nombre

17 de agosto de 2025

# Índice general

<b>1. Alcance</b>	<b>3</b>
<b>2. Introducción</b>	<b>4</b>
2.1. Reconocimiento . . . . .	4
2.1.1. Objetivo . . . . .	4
2.1.2. Metodología . . . . .	4
2.1.3. Hallazgos . . . . .	5
2.2. Weaponization . . . . .	5
2.2.1. Objetivo . . . . .	5
2.2.2. Metodología . . . . .	5
2.3. Entrega . . . . .	5
2.4. Explotación . . . . .	5
2.5. Instalación . . . . .	6
2.6. Command and Control (C2) . . . . .	6
2.7. Acciones sobre el objetivo . . . . .	6
<b>3. Malware</b>	<b>7</b>
3.1. Loader . . . . .	7
3.1.1. DLL Manipulation . . . . .	8
3.1.2. Process Injection . . . . .	8
3.2. Shellcode . . . . .	8
3.3. Packer . . . . .	8
<b>4. Como funcionan los EDRs</b>	<b>9</b>
4.1. Kernell callbacks . . . . .	9
4.2. Event tracing for Windows (ETW) . . . . .	9
4.3. Antimalware scan interface (AMSI) . . . . .	9
4.4. System call monitoring . . . . .	9
4.5. Monitorizacion de Memoria . . . . .	10

<b>5. EDR Bypass</b>	<b>11</b>
5.1. BYOVD . . . . .	11
5.2. Syscalls . . . . .	11
5.2.1. Direct Syscall . . . . .	11
5.2.2. Indirect Syscall . . . . .	11
5.3. Call Stack Spoofing . . . . .	11
<b>6. Anexo</b>	<b>12</b>
6.1. Tipos de Malware . . . . .	12
6.2. Memoria Virtual . . . . .	13
6.2.1. Problemas Memoria Fisica . . . . .	13
6.2.2. Acceso indirecto a memoria . . . . .	13
6.2.3. Paginación . . . . .	13
6.2.4. Tabla de paginas . . . . .	13
6.2.5. MMU . . . . .	13
6.2.6. Acceso a memoria . . . . .	14
6.3. Proceso . . . . .	15
6.3.1. Secciones de un proceso . . . . .	15
6.3.2. Procesos hijo e hilos . . . . .	16
6.4. Ejecutables Windows (PE) . . . . .	17
6.5. DLL (Dynamic-Link Library) . . . . .	18
6.6. Carga de DLLs en un proceso . . . . .	18
6.7. PEB y la lista de módulos . . . . .	18
6.8. Import Address Table (IAT) . . . . .	19
6.9. Resumen . . . . .	19
6.10. Ring level . . . . .	20
6.11. Win32 API . . . . .	21
6.12. SO Security . . . . .	22
6.12.1. ASLR (Address Space Layout Randomization) . . . . .	22
6.12.2. DEP (Data Execution Prevention) . . . . .	22
6.12.3. Control Flow Guard (CFG) . . . . .	22
6.12.4. Code Integrity Guard (CIG) . . . . .	22
6.12.5. Protected Process Light (PPL) . . . . .	22
6.12.6. Virtualization-Based Security (VBS) . . . . .	22
6.12.7. Kernel-mode code signing (KMCI) . . . . .	22
6.13. Genealogía de Procesos en Windows . . . . .	23
6.14. Listado de EDRs . . . . .	26

# Capítulo 1

## Alcance

*Contenido: Aquí se presenta la idea del trabajo y qué se pretende conseguir.*

El objetivo de este trabajo es desarrollar un escenario didáctico que permita a los investigadores de malware mejorar sus capacidades de análisis mediante la comprensión del proceso completo de creación de una malware. El resultado final del proyecto será un volcado de memoria de un sistema infectado, el cual 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 malware realista, su ejecución controlada en un sistema víctima, y la documentación íntegra del proceso ofensivo llevado a cabo por el red team. De este modo, el participante que reciba el volcado 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 por qué de cada artefacto encontrado en memoria.

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

# Capítulo 2

## Introducción

*Contenido: Esta sección dará un vistazo general al ataque que se va llevar a cabo, apoyándose en el Cyber Kill Chain de Lockheed Martin y el modelo MITRE ATTACK.*

### 2.1. Reconocimiento

#### 2.1.1. Objetivo

Identificar cualquier elemento dentro de la organización víctima que pueda conducir a una ejecución remota de código (RCE).

- Vulnerabilidades en servicios expuestos
  - Buffer overflow
  - ROP
- Credenciales expuestas
  - VPN
  - RDP

#### 2.1.2. Metodología

1. Enumeración de servicios expuestos asociados a la víctima.
2. Búsqueda de vulnerabilidades conocidas (CVEs) asociadas a dichos servicios.

3. Identificación del mayor número posible de usuarios relacionados con la víctima.
4. Rastreo de posibles credenciales expuestas en fuentes abiertas.

### 2.1.3. Hallazgos

En nuestro caso, se ha identificado un servidor web expuesto **vulnerable a técnicas de explotación basadas en ROP chain**, lo que podría permitir la ejecución de código arbitrario.

## 2.2. Weaponization

### 2.2.1. Objetivo

Crear el payload que se utilizará para explotar la vulnerabilidad identificada en el paso anterior.

### 2.2.2. Metodología

Normalmente se utilizan herramientas en este caso los payloads se crean a mano por fines didácticos.

## 2.3. Entrega

Como se entrega el payload a la víctima

Vectores típicos: phishing, vulnerabilidades sin parchear, credenciales expuestas.

El objetivo aquí es hacer llegar el malware o el loader al sistema de la víctima.

## 2.4. Explotación

Como conseguir ejecución del payload en la víctima

Una vez entregado, se consigue ejecutar código en el sistema.

Puede ser directo (el ransomware se ejecuta) o mediante un loader que se encarga de preparar el entorno y luego lanzar el malware.

## **2.5. Instalación**

El malware se instala en el sistema de la víctima. El loader instala el ransomware, a menudo asegurando persistencia, o simplemente prepara el entorno para que el malware se ejecute correctamente.

## **2.6. Command and Control (C2)**

El malware establece comunicación con el atacante para recibir instrucciones.

## **2.7. Acciones sobre el objetivo**

El atacante realiza su objetivo final - exfiltracion - encriptacion - criptominingo

# Capítulo 3

## Malware

*Contenido: En esta sección se explicará todo lo relacionado con la creación de un malware*

Un malware es un software diseñado para realizar acciones maliciosas en un sistema.

En este trabajo, el malware que desarrollaremos tendrá como objetivo encriptar todo el disco de la víctima.

El primer desafío consiste en conseguir ejecutar código en el sistema de la víctima. Las formas más habituales de lograrlo incluyen: phishing, explotación de vulnerabilidades sin parchear o uso de credenciales expuestas. Por ejemplo, si se dispone de credenciales que permiten acceder a una VPN, podríamos ejecutar código de manera remota.

El segundo desafío es garantizar que el malware no sea bloqueado por un EDR. Para ello, desarrollaremos un loader, un componente encargado de preparar el entorno y lanzar el malware de manera segura y silenciosa.

### 3.1. Loader

1. Carga en memoria (In-memory execution) 2. Inyección en procesos legítimos (Process Injection) 3. Evasión de defensas 4. Persistencia 5. Descarga o desempaquetado de payloads



### **3.1.1. DLL Manipulation**

**Phantom DLL Injection**

**DLL reflection**

### **3.1.2. Process Injection**

inyeccion de proceso verdadera seria desde un proceso de espacio de usuario a otro proceso de espacio de usuario objetivo benigno  
por otro lado esta el process spawning y hollowing

**clonando procesos (Dirty Vanity)**

Para realizar process hollowing, se tiene que llamar a 5 syscalls (muy ruidoso) Pero si se llaman a 3 desde un proceso y dos desde su hijo, los edrs no detectaban esa actividad

## **3.2. Shellcode**

## **3.3. Packer**

# Capítulo 4

## Como funcionan los EDRs

### 4.1. Kernell callbacks

El edr mantendra un driver en kernel land

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

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

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

### 4.3. 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.

### 4.4. System call monitoring

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

## 4.5. Monitorizacion de Memoria

yara

# Capítulo 5

## EDR Bypass

*Contenido: En esta sección se como los EDRs monitorizan el sistema y cómo se pueden evadir.*

### 5.1. BYOVD

### 5.2. Syscalls

#### 5.2.1. Direct Syscall

Otra opción para pasar bypasear los hooks del edr es no utilizar ntdll.dll y llamar directamente a la syscall

el problema es que hace falta el SSN y es un valor arbitrario que cambia muy frecuentemente entre builds de windows

la primera solución de esto fue "hells gate" que cargaba ntdll.dll desde disco y de ahí recuperaba el SSN (muy ruidoso para los edr)

Versiones más modernas: SysWhispers2

#### 5.2.2. Indirect Syscall

Aun no se como funciona

### 5.3. Call Stack Spoofing

# Capítulo 6

## Anexo

### 6.1. Tipos de Malware

RAT ransomware reverse shell

## 6.2. Memoria Virtual

### 6.2.1. Problemas Memoria Fisica

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

### 6.2.2. Acceso indirecto a memoria

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

### 6.2.3. Paginación

Mapear cada byte de la memoria virtual a un byte de la memoria fisica 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 fisica

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

### 6.2.4. 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.

### 6.2.5. MMU

La CPU tiene que acceder a la RAM primero para consultar la tabla de paginas y Despues para acceder a la memoria fisica, 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 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,

### **6.2.6. Acceso a memoria**

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

Page number - los bits más significativos de la dirección virtual  
Page offset - los bits menos significativos de la dirección virtual

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

## 6.3. Proceso

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

### 6.3.1. 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

### **6.3.2. Procesos hijo e hilos**

## 6.4. Ejecutables Windows (PE)

## 6.5. 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. 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.

## 6.7. 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.

## 6.8. 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.

## 6.9. 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.

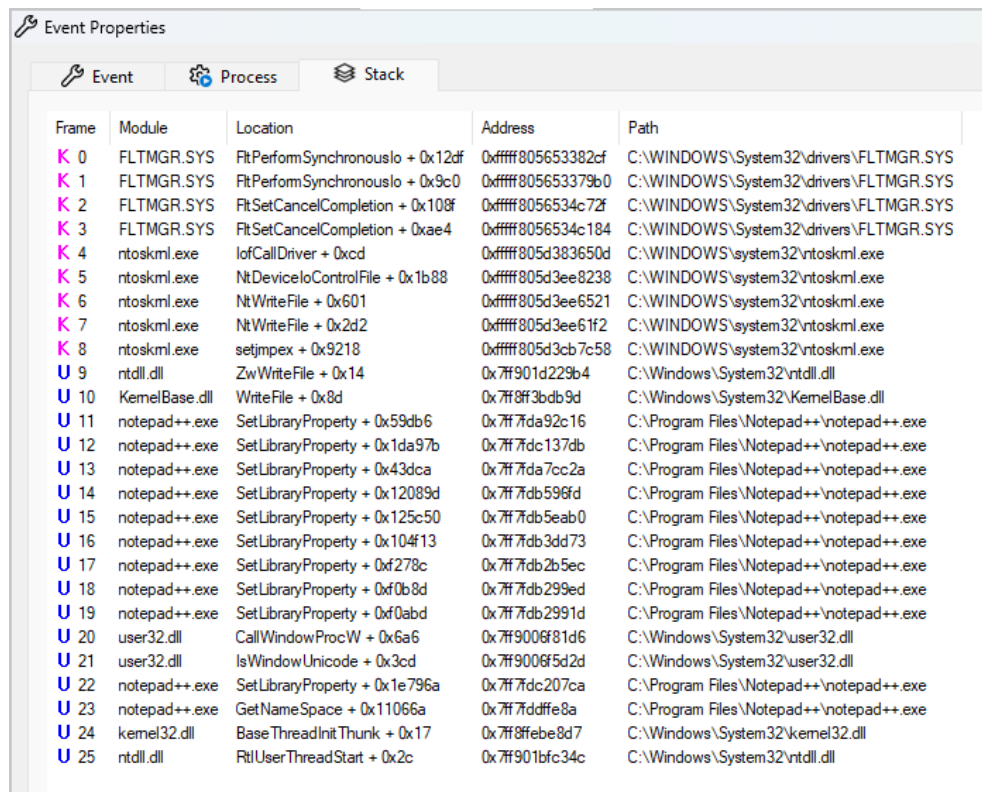
Esto garantiza que el proceso pueda utilizar código externo (DLLs) sin necesidad de incorporarlo estáticamente en su binario, manteniendo modularidad y eficiencia en el uso de memoria.

## 6.10. 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



Frame	Module	Location	Address	Path
K 0	FLTMGR.SYS	RtPerformSynchronousIo + 0x12df	0xffff805653382cf	C:\WINDOWS\System32\drivers\FLTMGR.SYS
K 1	FLTMGR.SYS	RtPerformSynchronousIo + 0x9c0	0xffff805653379b0	C:\WINDOWS\System32\drivers\FLTMGR.SYS
K 2	FLTMGR.SYS	RtSetCancelCompletion + 0x108f	0xffff8056534c72f	C:\WINDOWS\System32\drivers\FLTMGR.SYS
K 3	FLTMGR.SYS	RtSetCancelCompletion + 0xae4	0xffff8056534c184	C:\WINDOWS\System32\drivers\FLTMGR.SYS
K 4	ntoskml.exe	IoCallDriver + 0xcd	0xffff805d383650d	C:\WINDOWS\system32\ntoskml.exe
K 5	ntoskml.exe	NtDeviceIoControlFile + 0x1b88	0xffff805d3ee8238	C:\WINDOWS\system32\ntoskml.exe
K 6	ntoskml.exe	NtWriteFile + 0x601	0xffff805d3ee6521	C:\WINDOWS\system32\ntoskml.exe
K 7	ntoskml.exe	NtWriteFile + 0x2d2	0xffff805d3ee61f2	C:\WINDOWS\system32\ntoskml.exe
K 8	ntoskml.exe	setjmpex + 0x9218	0xffff805d3cb7c58	C:\WINDOWS\system32\ntoskml.exe
U 9	ntdll.dll	ZwWriteFile + 0x14	0x7ff901d229b4	C:\Windows\System32\ntdll.dll
U 10	KernelBase.dll	WriteFile + 0x8d	0x7ff8ff3bdb9d	C:\Windows\System32\KernelBase.dll
U 11	notepad++.exe	SetLibraryProperty + 0x59db6	0x7ff7da92c16	C:\Program Files\Notepad++\notepad++.exe
U 12	notepad++.exe	SetLibraryProperty + 0x1da97b	0x7ff7dc137db	C:\Program Files\Notepad++\notepad++.exe
U 13	notepad++.exe	SetLibraryProperty + 0x43dca	0x7ff7da7cc2a	C:\Program Files\Notepad++\notepad++.exe
U 14	notepad++.exe	SetLibraryProperty + 0x12089d	0x7ff7db596fd	C:\Program Files\Notepad++\notepad++.exe
U 15	notepad++.exe	SetLibraryProperty + 0x125c50	0x7ff7db5eab0	C:\Program Files\Notepad++\notepad++.exe
U 16	notepad++.exe	SetLibraryProperty + 0x104f13	0x7ff7db3dd73	C:\Program Files\Notepad++\notepad++.exe
U 17	notepad++.exe	SetLibraryProperty + 0xf278c	0x7ff7db2b5ec	C:\Program Files\Notepad++\notepad++.exe
U 18	notepad++.exe	SetLibraryProperty + 0xf0b8d	0x7ff7db299ed	C:\Program Files\Notepad++\notepad++.exe
U 19	notepad++.exe	SetLibraryProperty + 0xf0abd	0x7ff7db2991d	C:\Program Files\Notepad++\notepad++.exe
U 20	user32.dll	CallWindowProcW + 0x6a6	0x7ff9006f81d6	C:\Windows\System32\user32.dll
U 21	user32.dll	IsWindowUnicode + 0x3cd	0x7ff9006f5d2d	C:\Windows\System32\user32.dll
U 22	notepad++.exe	SetLibraryProperty + 0x1e796a	0x7ff7dc207ca	C:\Program Files\Notepad++\notepad++.exe
U 23	notepad++.exe	GetNameSpace + 0x11066a	0x7ff7ddffe8a	C:\Program Files\Notepad++\notepad++.exe
U 24	kernel32.dll	BaseThreadInitThunk + 0x17	0x7ff8febe8d7	C:\Windows\System32\kernel32.dll
U 25	ntdll.dll	RtlUserThreadStart + 0x2c	0x7ff901bfc34c	C:\Windows\System32\ntdll.dll

## 6.11. 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

## 6.12. SO Security

6.12.1. ASLR (Address Space Layout Randomization)

6.12.2. DEP (Data Execution Prevention)

6.12.3. Control Flow Guard (CFG)

6.12.4. Code Integrity Guard (CIG)

6.12.5. Protected Process Light (PPL)

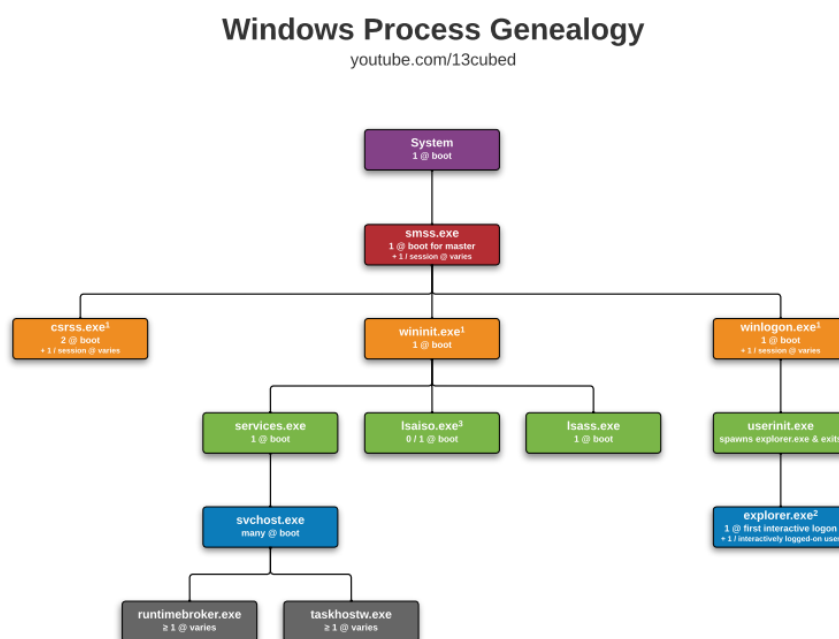
6.12.6. Virtualization-Based Security (VBS)

6.12.7. Kernel-mode code signing (KMCI)

## 6.13. 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.

## 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.

## 6.14. Listado de EDRs

Sylantstrike puede ser utilizado para hacer pruebas de concepto de EDRs

- CrowdStrike Falcon
- Microsoft Defender for Endpoint