Trabajo de Fin de Máster

Tu Nombre

2 de agosto de 2025

Índice general

1. Introducción **3**

2. EDRs Monitorización **4**
- 2.1. Kernell callbacks . . . 4
- 2.2. Event tracing for Windows (ETW) . . . 4
- 2.3. Antimalware scan interface (AMSI) . . . 4
- 2.4. System call monitoring . . . 5

3. EDRs Bypass **6**
- 3.1. unhooking . . . 6
- 3.2. clonando procesos (Dirty Vanity) . . . 6
- 3.3. syscalls . . . 6
  - 3.3.1. direct system . . . 6
  - 3.3.2. indirect system . . . 7
- 3.4. call stack spoofing . . . 7

4. Anexos **8**
- 4.1. Malware . . . 8
  - 4.1.1. Empaquetadores . . . 8
  - 4.1.2. Cargadores . . . 8
  - 4.1.3. Shellcodes . . . 8
- 4.2. Proceso . . . 8
  - 4.2.1. Secciones de un proceso . . . 8
  - 4.2.2. ASLR - Address Space Layout Randomization . . . 9
- 4.3. Ring level . . . 9
- 4.4. Memoria Virtual . . . 9
  - 4.4.1. Problemas Memoria Fisica . . . 9
  - 4.4.2. Acceso indirecto a memoria . . . 10
  - 4.4.3. Paginación . . . 10
  - 4.4.4. Tabla de paginas . . . 10
  - 4.4.5. MMU . . . 10

4.4.6. Acceso a memoria 10
4.5. Win32 API 11
4.6. Genealogía de Procesos en Windows 11
4.7. Listado de EDRs 14

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 deberia saltar a la syscall salta al edr de modo que lo analiza antes de que se ejecute

Capítulo 3

EDRs Bypass

3.1. unhooking

Todos los procesos de windows el primer dll que cargan es: ntdll.dll

Este dll es un handler para realizar syscalls sobre hilos procesos y memoria etc

Ej: NtCreateThreadEx, NtAllocateVirtualMemory, etc

El edr inyecta en esas funcion un jmp para que antes de saltar a syscalls salte al edr

La primera tecnica consiste en cargar una version limpia de la funcion en la que aun no se ha establecido el hook, y sobreescribir la funcion

hay dos formas de obtener la funcion limpia, cagando una version de ntdll.dll desde fichero (problema los edr vigilan esto)

Otra opcion crear un proceso en suspension, estos solo tiene el codigo de la aplicacion y el ntdll.dll

3.2. 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

3.3. syscalls

3.3.1. direct system

Otra opcion para pasar bypassear 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 solucion de esto fue "hells gate"que cargaba ntdll.dll desde disco y de ahi recuperaba el SSN (muy ruidoso para los edr)

Versiones mas modernas: SysWhispers2

3.3.2. indirect system

Aun no se como funciona

3.4. call stack spoofing

Capítulo 4

Anexos

4.1. Malware

4.1.1. Empaquetadores

4.1.2. Cargadores

4.1.3. Shellcodes

4.2. Proceso

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

4.2.1. 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 momoria 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

4.2.2. ASLR - Address Space Layout Randomization

4.3. 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

4.4. Memoria Virtual

4.4.1. Problemas Memoria Fisica

- Escasez de memoria
- Fragmentación

- Acceso a memoria (Seguridad)

4.4.2. 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.

4.4.3. 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

4.4.4. 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.

4.4.5. 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,

4.4.6. 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,

4.5. Win32 API

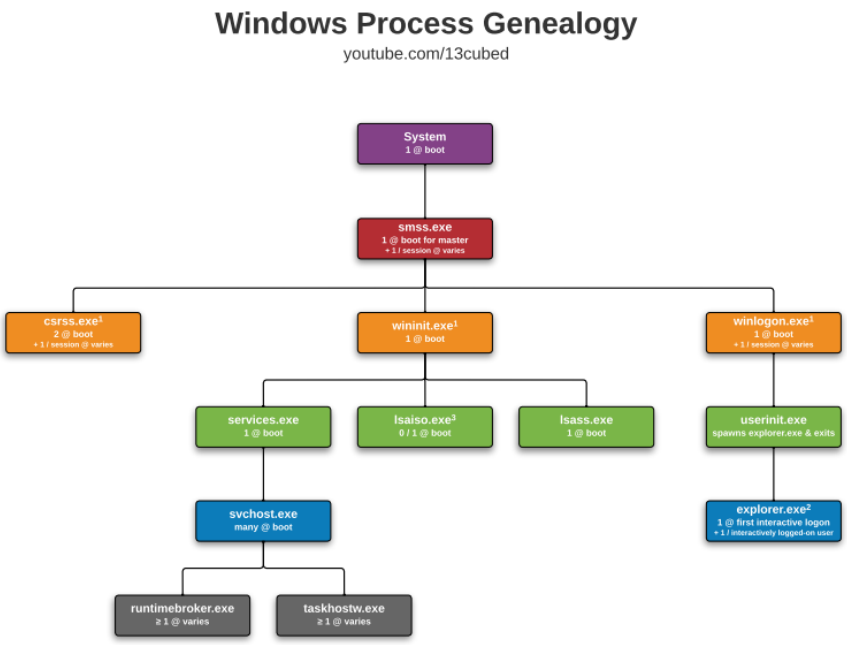
Nombre de la DLL	**Tareas de la DLL**
User32.dll	Esta biblioteca contiene funciones para crear ventanas, manejar mensajes y procesar la entrada del usuario.
Kernel32.dll	Esta biblioteca proporciona acceso a una variedad de servicios esenciales del sistema como la gestión de memoria, operaciones de E/S 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 4.1: Descripción de las principales DLLs de la Win32 API

4.6. 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:



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.

4.7. Listado de EDRs

Sylantstrike puede ser utilizado para hacer pruebas de concepto de EDRs

- CrowdStrike Falcon
- Microsoft Defender for Endpoint