Análisis de malware

Análisis de programas maliciosos para Windows

Gustavo Romero López - gustavo@ugr.es

Updated: 3 de marzo de 2025

Departamento de Ingeniería de Computadores, Automática y Robótica

Por donde vamos...

Parte 1: Análisis básico

- Capítulo 1: Técnicas estáticas básicas
- Capítulo 2: Análisis de malware en máquinas virtuales
- Capítulo 3: Análisis dinámico básico

Parte 2: Análisis estático avanzado

- Capítulo 4: Curso intensivo de ensamblador x86
- Capítulo 5: IDA Pro
- Capítulo 6: Reconocimiento de construcciones de C en ensamblador
- Capítulo 7: Análisis de programas maliciosos para Windows

Parte 3: Análisis dinámico avanzado

Parte 4: Funcionalidad del malware

Parte 5: Anti-ingeniería inversa

Parte 6: Temas especiales

1. La API de Windows
2. El registro de Windows
3. APIs de red
4. Siguiendo la ejecución del malware
5. Modo núcleo y modo usuario
6. La API nativa

Tipos y notación húngara

- La API de Windows utiliza sus propios nombres para los tipos de C, ej: unsigned short int → WORD
- Los nombre de variables añaden prefijos para indicar su tipo, ej: DWORD dwSize

tipo	prefijo	descripción
WORD	w	unsigned short int
DWORD	dw	unsigned int
Handle	H	referencia a un objeto manipular exclusivamente mediante API
Long Pointer	LP	puntero a otro tipo, también Pointer (P) ej: byte* → LPByte
Callback		función llamable desde la API de Windows

Manejadores (“*handles*”)

- Elementos creados por el sistema operativo
- A veces hacen referencia a objetos en memoria
- No permiten aritmética de punteros
- No siempre representan la dirección del objeto
- Solo pueden almacenarse y pasarse a funciones de la API

```
HWND hwnd = CreateWindowExA(...);
...
DestroyWindow(hwnd);
```

Funciones del sistema de ficheros

- CreateFile
- ReadFile
- WriteFile
- CreateFileMapping: carga un fichero en memoria
- MapViewOfFile: puntero al fichero mapeado

```
HANDLE CreateFileA(
  [in]           LPCSTR                lpFileName,
  [in]           DWORD                 dwDesiredAccess,
  [in]           DWORD                 dwShareMode,
  [in, optional] LPSECURITY_ATTRIBUTES lpSecurityAttributes,
  [in]           DWORD                 dwCreationDisposition,
  [in]           DWORD                 dwFlagsAndAttributes,
  [in, optional] HANDLE                hTemplateFile
);
```

Ficheros especiales

- Suelen pasar inadvertidos porque no se muestran.
- Ficheros compartidos (almacenados en red):
  - prefijos `\\serverName\share` y `\\?\serverName\share`
- Ficheros accesibles mediante espacios de nombres:
  - espacio de nombres NT: prefijo `\\.\`
  - permite acceder directamente a dispositivos físicos, ej: `\\.\PhysicalDisk1` y `\Device\PhysicalMemory`
- Flujo de datos alternativo (*"Alternate Data Stream" - ADS*):
  - datos adicionales en ficheros NTFS
  - ejemplo: `normalFile.txt:Stream:$DATA`
  - suelen utilizarse para ocultar datos

El registro de Windows

- Almacena información de configuración
- Versiones antiguas de Windows utilizaban ficheros `.ini`
- Base de datos jerárquica
- Suele utilizarse para lograr la persistencia
- Términos importantes:
  - clave raíz ("root key"): 5 subdivisiones del registro, `HKEY`
  - subclave ("subkey"): especie de subdiretorio
  - clave ("key"): directorio en el registro, puede contener subdirectorios y valores
  - entrada ("value entry"): par ordenado <nombre, valor>
  - valor o dato: dato almacenado en el registro

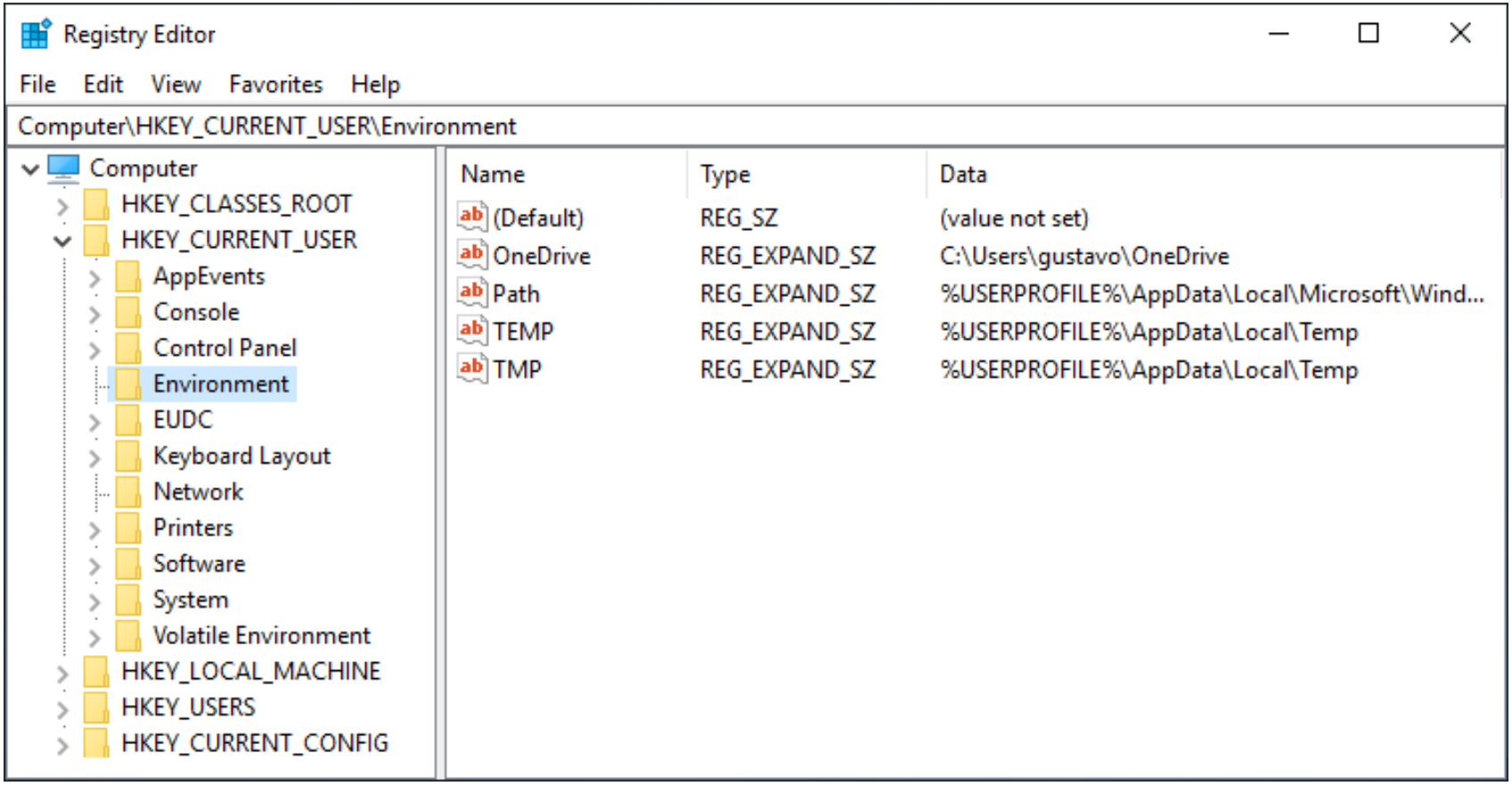
Claves raíz del registro

- `HKEY_LOCAL_MACHINE (HKLM)`: configuración global del sistema
- `HKEY_CURRENT_USER (HKCU)`: configuración de usuario
- `HKEY_CLASSES_ROOT`: definición de tipos
- `HKEY_CURRENT_CONFIG`: configuración hardware
- `HKEY_USERS`: configuración de usuarios

- Algunas claves son virtuales: `HKEY_CURRENT_USER` en realidad se almacena como `HKEY_USERS\\SID`
- `HKEY_LOCAL_MACHINE\\SOFTWARE\\Microsoft\\Windows \\CurrentVersion\\Run`: lista de ejecutables que iniciar automáticamente

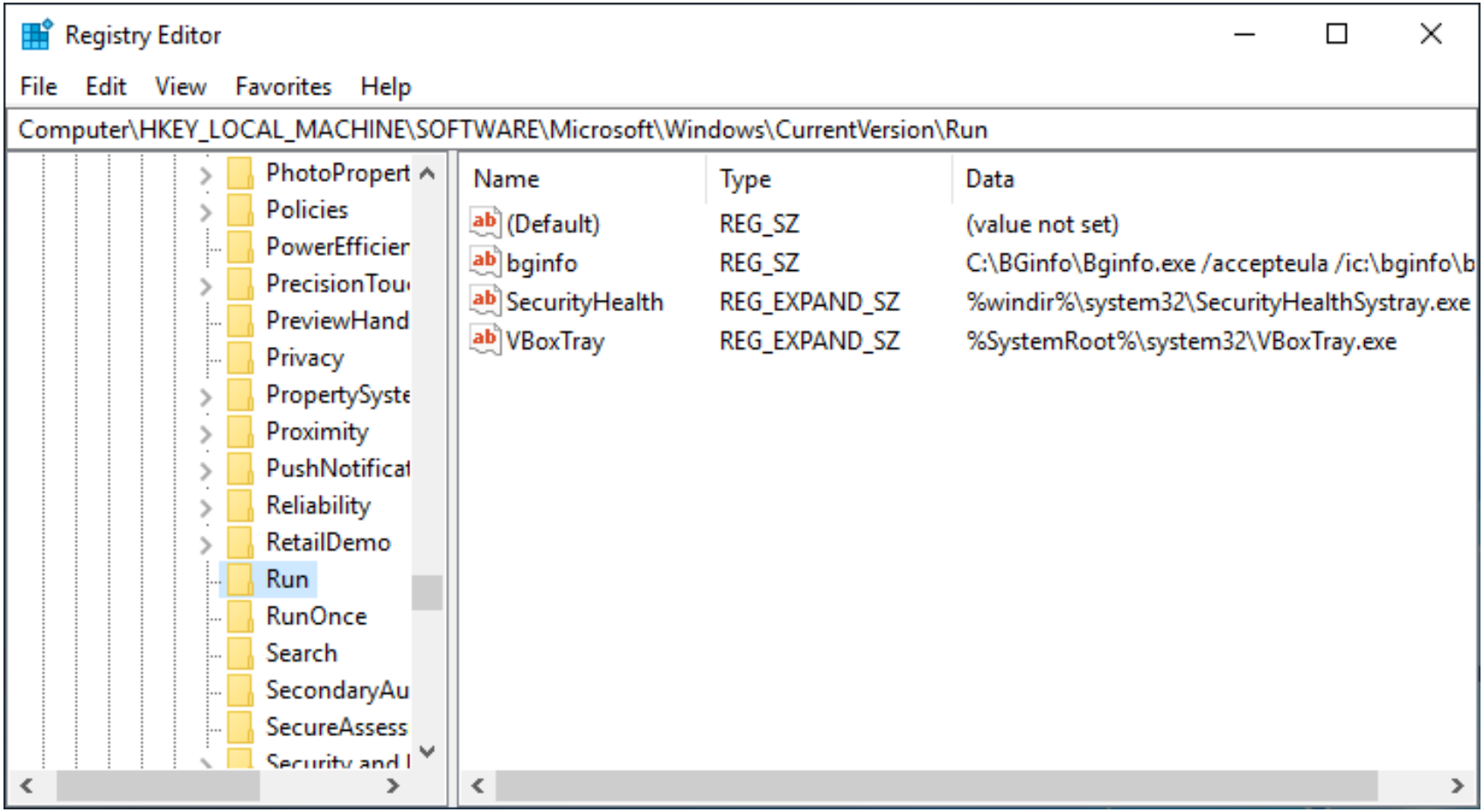
Regedit

- Herramienta de consulta y edición del registro



Programas que se ejecutan automáticamente

- Técnica nada sigilosa pero muy empleada
- `Autoruns`: herramienta para localizar estos programas



Funciones habitules del registro

- Conseguir ejecutarse automáticamente
- Funciones:
  - `RegOpenKeyEx`: abre el registro para consulta y edición
  - `RegSetValueEx`: añade un nuevo valor al registro
  - `RegGetValue`: obtiene el valor de una entrada
- Hay muchas claves que afectan a la seguridad (demasiadas)
- La lista de claves es tan grande que lo más habitual es buscarlas por Internet

```
LSTATUS RegSetValueExW(
        [in]           HKEY       hKey,
        [in, optional] LPCWSTR    lpValueName,
                       DWORD      Reserved,
        [in]           DWORD      dwType,
        [in]           const BYTE *lpData,
        [in]           DWORD      cbData
  );
```

Analizar código de registro en la práctica

- Buscar la función `RegSetValueEx\{A,W\}`
- Buscar comentarios `HKLM` y subclaves "`...\\Run`"
- Buscar cadenas `HKLM` y subclaves "`...\\Run`"
- La más habitual es: "`HKEY_LOCAL_MACHINE\\SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run`"

Scripts para registro con ficheros *.reg

- Ficheros legibles para alterar el registro
- Se ejecutan mediante un doble click
- Método menos empleado que la programación
- Ejemplo:

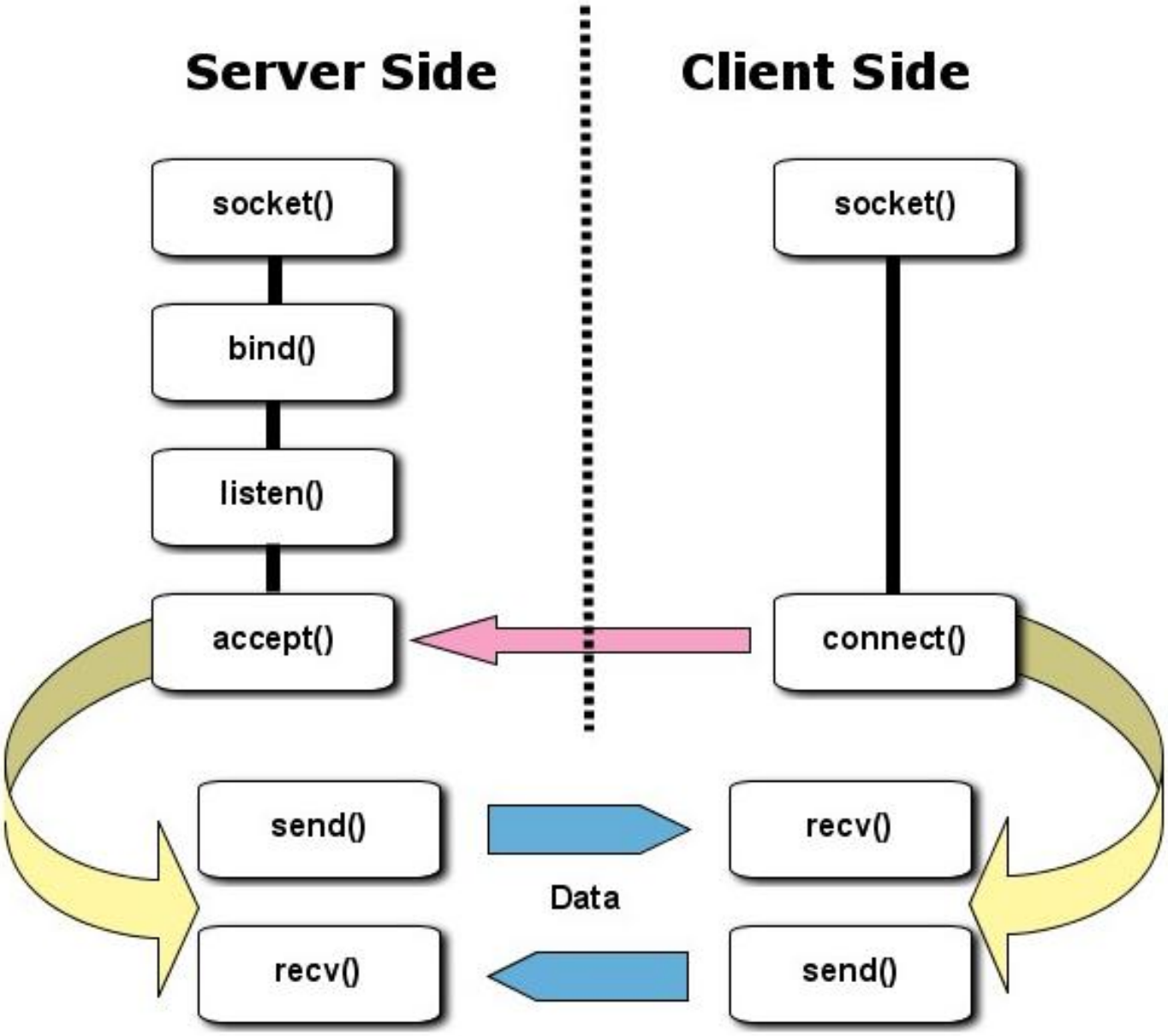
```
Windows Registry Editor Version 5.00

[HKLM\\SOFTWARE\Microsoft\Windows\CurrentVersion\Run]
"MaliciousValue"="C:\Windows\evil.exe"
```

Berkeley Sockets

- Idéntica interfaz en Windows y UNIX
- Biblioteca Winsock en `ws_32.dll`
- Funciones:
  - `socket`: crea un socket
  - `bind`: conecta un socket a un puerto
  - `listen`: pone al socket en espera de conexiones entrantes
  - `accept`: crea una conexión con un socket remoto y acepta la conexión
  - `connect`: crea una conexión con un socket remoto que la esté esperando
  - `recv`: recibe datos de un socket
  - `snd`: envía datos de un socket
- `WSAStartup` inicializa el sistema de sockets de Win32

Cliente y servidor



La API WinINet

- API de mayor nivel que Winsock almacenada en Wininet.dll
- Implementa protocolos de la capa de aplicación como HTTP o FTP
- Funciones:
  - `InternetOpen`: inicializa una conexión
  - `InternetOpenUrl`: conecta con una URL
  - `InternetReadFile`: como ReadFile pero para los datos de un fichero descargado de Internet

Siguiendo la ejecución del malware: DLLs

- Biblioteca de enlace dinámico (*"Dynamic-Link Library"*)
- Forma de compartir código de Windows
- Permiten consumir menos memoria que el enlace estático
- Un mismo código compartido por múltiples procesos
- Muchas de ellas incluidas por defecto en Windows

Como usan los autores de malware las DLLs

- Para almacenar código
  - los procesos solo pueden contener un fichero .exe
  - de esta forma pueden ejecutarse sobre otros procesos
- Utilizando las DLLs de Windows
  - como todo el software utiliza código de Windows para interaccionar con el sistema operativo
  - las funciones importadas nos darán pistas de lo que intenta hacer
- Utilizando DLLs de otros
  - cuando interacción con software de terceros hemos de suponer que lo necesita para lograr su propósito
  - ejemplo: software de criptografía no presente por defecto

Estructura básica de una DLL

- Casi idéntico aun fichero ejecutable
- Solo una bandera en la cabecera PE indica que es una DLL en vez de un ejecutable
- Es común tener más "exports" que "imports"
- La función principal se llama `DllMain`
- `DllMain` es el punto de entrada del ejecutable
- `DllMain` es llamada cada vez que hacemos algo relacionado con la DLL: carga/descarga, creación/destrucción de hebras y procesos,...
- Su código nos da pistas sobre la funcionalidad del malware

Procesos

- Un proceso es...
  - un programa en ejecución
  - un contenedor de recursos
  - un mecanismo de seguridad
- El malware puede ejecutarse fuera de un proceso creando uno nuevo
  - antes: proceso independiente
  - hoy en día: ejecutan su código como otros procesos
- El recurso más importante que comparte un proceso es la memoria
- Cada proceso tiene un espacio de direcciones independiente

Creación de nuevos procesos

- Se utiliza la función `CreateProcess`
  - gran control sobre como se hace
  - permite escapar del ciertos mecanismos de seguridad
  - ej: ejecutar el navegador para acceder a contenido inseguro
- Lanzar un intérprete de órdenes remoto con una sola llamada cambiando std{in,out,err} por sockets
- Otro truco es esconder un ejecutable como un recurso y ejecutarlo en un nuevo proceso

```
BOOL CreateProcessA(LPCSTR                lpApplicationName,
                    LPSTR                 lpCommandLine,
                    LPSECURITY_ATTRIBUTES lpProcessAttributes,
                    LPSECURITY_ATTRIBUTES lpThreadAttributes,
                    BOOL                  bInheritHandles,
                    DWORD                 dwCreationFlags,
                    LPVOID                lpEnvironment,
                    LPCSTR                lpCurrentDirectory,
                    LPSTARTUPINFOA        lpStartupInfo,
                    LPPROCESS_INFORMATION lpProcessInformation);
```

Hebras

- Si los procesos son contenedores de recursos las hebras representan el código que se se ejecuta.
- Todo proceso ejecuta al menos una hebra.
- Todas las hebras de un proceso comparten todos sus recursos salvo los registros de la CPU y la pila.
- Contexto de una hebra:
  - Cada hebra controla un procesador.
  - Al cambiar de hebra es necesario almacenar el estado del procesador para poder restaurarlo cuando queramos volver a ejecutarla.

Creación de una hebra

- Se utiliza la función `CreateThread`
- Hay que especificar una dirección como función de inicio
- Uso habituales en malware:
  - cargar una DLL (`LoadLibrary`)
  - crear dos hebras para manejar la E/S a través de sockets o tuberías y así poderse comunicarse

```
HANDLE CreateThread(
        [in, optional]  LPSECURITY_ATTRIBUTES   lpThreadAttributes,
        [in]            SIZE_T                  dwStackSize,
        [in]            LPTHREAD_START_ROUTINE  lpStartAddress,
        [in, optional]  __drv_aliasesMem LPVOID lpParameter,
        [in]            DWORD                   dwCreationFlags,
        [out, optional] LPDWORD                 lpThreadId
        );
```

Coordinación entre procesos mediante Mutex

- Objetos globales que permiten la coordinación
- Suelen utilizarse para acceder a recursos compartidos
- Solo una hebra puede poseer un mutex
- Los identificadores en el código los hace ideales para ser rastreados
- Funciones:
  - `WaitForSingleObject`: intenta adquirir un mutex
  - `ReleaseMutex`: libera un mutex
  - `CreateMute`: crea un mutex
  - `OpenMutex`: devuelve un manejador al mutex de otro proceso
- El malware los usa para evitar ejecutarse más de una vez

```
DWORD WaitForSingleObject([in] HANDLE hHandle,
                          [in] DWORD  dwMilliseconds);
```

Servicios

- Aplicación que ejecuta el sistema en segundo plano
- Privilegios de la cuenta SYSTEM
- Ejecución manual o automática al inicio
- No aparecen en el gestor de tareas
- Podemos verlas con "`net start`" y `Autoruns`
- Funciones de la API de Windows:
  - `OpenSCManager`: manejador al gestor de servicios
  - `CreateService`: añade un nuevo servicio
  - `StartService`: inicia un servicio
- Uso típico: esconder el código en una DLL y ejecutar con `WIN32_SHARE_PROCESS` compartiendo `svchost.exe`
- También puede ser `WIN32_OWN_PROCESS` y `KERNEL_DRIVER`
- Con SC podemos añadir, borrar, iniciar y consultar servicios

Component Object Model (COM)

- Interfaz para permitir la ejecución de código de terceros
- Permite que cualquier código sea reutilizable
- Funciona con cualquier lenguaje de programación
- El malware lo utiliza para dificultar su análisis
- Modelo cliente/servidor:
  - los cliente hacen uso de objetos COM
  - los servidores proporcionan el código reutilizable
- Es necesario llamar a `OleInitialize` o `CoInitializeEx`
- El analista debe encontrar los identificadores del componente exacto para poder identificarlo

CLSID, IIDs y uso de objetos COM

- Identificadores globales únicos (“*globally unique identifiers - GUIDs*”)
  - identificadores de clase (“*class identifiers - CLSIDs*”)
  - identificadores de interfaz (“*interface identifiers - IIDs*”)
- `CoCreateInstance`: acceso a la funcionalidad COM
- Ejemplo: Lanzamiento de un navegador
  - interfaz: `IWebBrowser2`
  - función: `Navigate`
- IDA Pro puede reconocer algunos servicios
- Localización de CLSIDs:
  - registro: `HK{CU,LM}\SOFTWARE\Classes\CLSID\`
  - cabeceras `*.h` de Windows

Malware en servidores COM

- Se utiliza para ejecutar código en otros procesos
- *Browser Helper Objects - BHOs*
- Ejemplo: plugins para Internet Explorer
- Fácil de detectar por las funciones que debe exportar: `DllCanUnloadNow`, `DllGetClassObjet`, `DllInstall`, `DllRegister` o `DllUnregisterServer`

Excepciones

- Permite lidiar con situaciones fuera de lo normal.
- La mayor parte de las veces se deben a errores, ej: dividir entre 0, acceso inválido a memoria,...
- Se trasfiere el control a código especial.
- Windows almacena en la pila la dirección del manejador y en `fs:0` la dirección de la pila.
- El manejador de excepción retorna a la hebra principal.
- Cada manejador es específico de un tipo de evento.
- Cuando no puede resolverse se pasa al llamador.
- El manejador de nivel superior aborta la aplicación.
- El malware puede cambiar el valor del manejador.

Modo núcleo y modo usuario

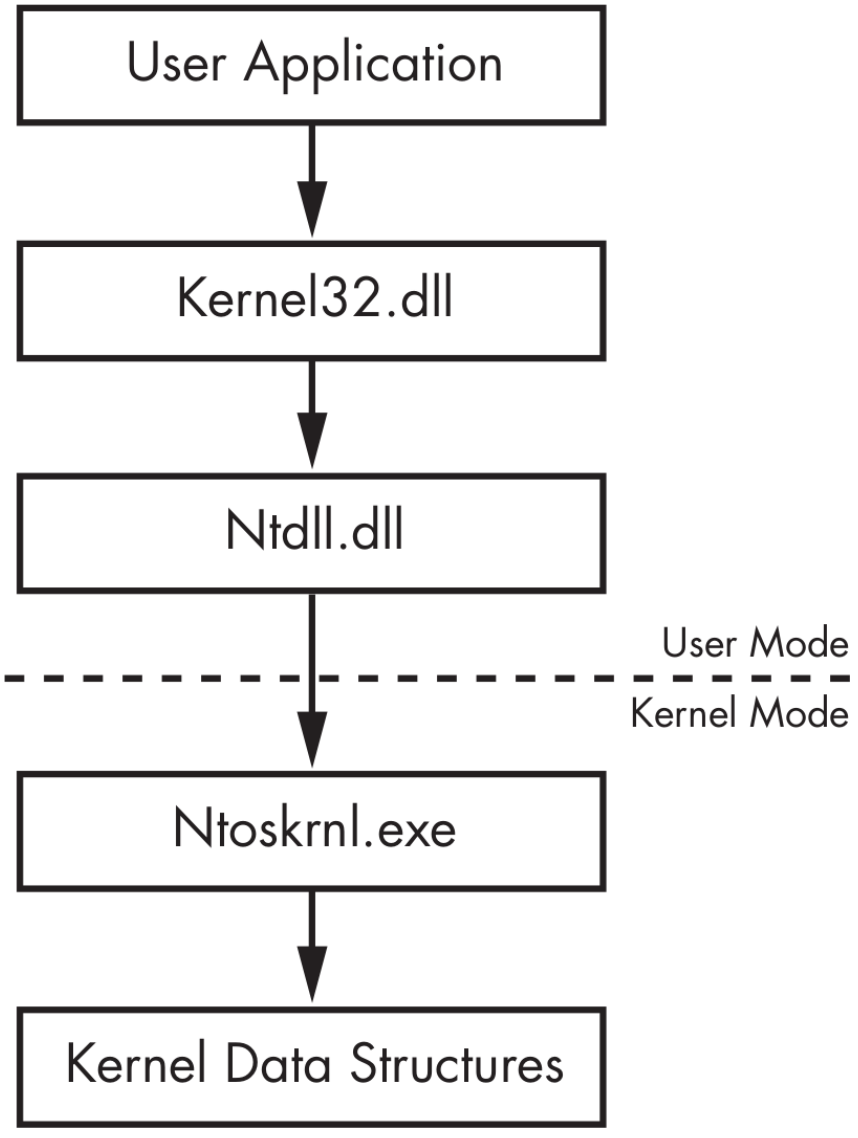
- Todo lo visto hasta ahora hace referencia al modo usuario.
- El modo núcleo tiene funciones equivalentes.
- Acceso a modo núcleo: `SYSENTER`, `SYSCALL`, `INT 0x2E`.
- Menos comprobaciones de seguridad.
- Objetivo primario pero de mayor dificultad.
- No requerido por la mayor parte del malware.

características	usuario	núcleo
memoria	independiente	compartida
hardware	acceso restringido	acceso completo
código	casi todo	SO y drivers
instrucciones	subconjunto	todas
comprobaciones	todas	pocas
manipulaciones	prohibidas	permitidas
auditorías	todas	pocas/ninguna

La API nativa

- Interfaz de más bajo nivel
- Raramente utilizada por programas no maliciosos
- Permite saltarse la API normal de Windows
- Seguridad por ocultación
- API no documentada oficialmente
- Permite hacer cosas imposibles de otro modo
- Antiguamente no monitorizadas, ej:

  `WriteFile` → `NtWriteFile`



Ejercicios: Lab 7-1

Analice el fichero Lab07-01.exe

1. ¿Cómo consigue la persistencia?
2. ¿Para qué utiliza un mutex?
3. ¿Cómo detectaría este malware en el ordenador?
4. ¿Cómo detectaría este malware en la red?
5. ¿Cuál es el propósito de este programa?
6. ¿Cuándo finaliza su ejecución?

Ejercicios: Lab 7-2

Analice el fichero Lab07-02.exe

1. ¿Cómo consigue la persistencia?
2. ¿Cuál es el propósito de este programa?
3. ¿Cuándo finaliza su ejecución?

Ejercicios: Lab 7-3

Analice los ficheros Lab07-03.dll y Lab07-03.exe colocándolos juntos en un mismo directorio.

1. ¿Cómo consigue la persistencia?
2. ¿Cómo detectaría este malware en el ordenador?
3. ¿Cuál es el propósito de este programa?
4. ¿Cuándo lo eliminaría una vez instalado?