Windows Malware

1.1 Windows API

Malicious software designed to run on the Windows operating on the Windows operating system frequently leverages the Windows Application Programming Interface (API). This approach is highly favored because the Windows API provides comprehensive access to system-related operations, often eliminating the need for complex, third-party libraries.

The Windows API utilizes several special data types integral to its operations, most notably the `HANDLE` . The API also adheres to Hungarian Notation for naming convention. In Hungarian Notation, variable names are prefixed with a few characters indicating the variable's special data type. For example, a variable named `dwSize` indicates that `Size` is of the special type `DWORD` , which is a 32-bit unsigned integer. Common special types include:

- `WORD` : 16-bit unsigned integer.
- `DWORD` : 32-bit unsigned integer.
- `HANDLE` : A reference to an object by the operating system (e.g., a file, a process, or a mutex). While similar to a pointer, a `HANDLE` cannot be used for arithmetic operations.
- *Long Pointer*: The term for a true memory address or pointer within the API, often represented by the prefix `LP` . The type is frequently further specified; for instance, `LPCSTR` denotes a Long Pointer to a Constant String.
- *Callbacks*: Functions that are passed as arguments to another function, which then executes the passed function at a later time.

Core System Functions. Malware analysis often revolves around identifying the usage of key API calls that perform critical system tasks:

- `CreateFile` : This versatile function is used to create a new file, open an existing file, or open a special I/O device (like a pipe). The `dwCreationDisposition` parameter is crucial as it dictates whether the call should create a new file, open an existing one, or a combination of both.
- `ReadFile` and `WriteFile` : These functions are used to read or write a specified number of bytes to a file or I/O device. The operating system maintains an internal file pointer for the open file handle. After reading or writing X bytes in one call, the next subsequent call of the same type will begin its operation from the position immediately following the X bytes written/read in the previous call.
- `CreateFileMapping` and `MapViewOfFile` : These functions are central to the technique of Memory-Mapped Files (MMF), which is often used by malware for process injection or code execution. `CreateFileMapping` establishes a named or unnamed kernel object that represents a section of a file (or the system paging file). `MapViewOfFile` maps a view of that file mapping object into the process's address space, effectively loading the file into memory where it can be parsed, modified, and executed. This call is highly effective

for loading and executing additional code or modifying the memory of existing processes, conceptually mimiking some functionality of the Windows program loader.

Special Files and Paths. Certain system resources and files cannot be accessed using standard file system paths (e.g., `C:\ProgramFiles`). These resources require special path syntax and are a key target for malware seeking to hide or access privileged resources. The Win32 device namespace allows direct access to physical hardware and internal kernel objects. This namespace is represented by the path prefix `\\.\` . From this path, it is possible to access a physical disk directly, bypassing the file system, for example, with the path `\\.\PhysicalDisk1` .

Shared Files are files located in shared network folders which are typically accessed using Universal Naming Convention (UNC) paths, which generally follow the format `\\serverName\share` . To avoid issues with string parsing, especially for UNC paths or paths exceeding the standard 260-character limit, malware often uses the extended path prefix `\\?\` . When applied to a UNC path, this becomes `\\?\UNC\serverName\share` , which ensures the path is passed directly to the file system I/O manager without being interpreted as a network share path initially.

Windows Registry. The Windows Registry is a hierarchical database that centrally stores configuration information for the operating system, hardware devices, and installed programs. Malware frequently interacts with the Registry for purposes such as persistence and storing dynamic configuration data. The Registry is organized into containers called keys (analogous to folders), which hold values (analogous to files). Malware often modifies keys located under `HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run` or `HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run` . By adding a val ue entry here that points to the malware's executable file, the program will run every time the corresponding user or the system starts.

The common Windows API functions used to manipulate the Registry are:

- `RegOpenKeyEx` : Opens a specified key, granting a handle that can be used for subsequent read or write operations.
- `RegSetValueEx` : Creates a new value or sets the data for an existing value entry within an opened key.
- `RegGetValue` : Retrieves the data stored in a specific value entry within a key.
- *Registry Files*: It is possible to merge Registry data stored in a plain text file, known as a `.reg` file, with the existing system Registry by executing it.

Networking API. Windows implements networking functionality using the Berkeley Sockets model, which defines how processes interact over a network. The functions are primarily exposed through Winsock (Windows Sockets). All network routines must be initialized before any other socket function can be called. `WSAStartup` is the essential first function call, used to load the appropriate Windows Sockets DDL and initialize its data structures. The typical sequence of calls for establishing a server connection is:

(1) `socket` : Creates a socket - and endpoint for communication - and returns a descriptor.

(2) `bind` : Associates the created socket with a specific local IP address and port number.

(3) `listen` : Puts the socket into a state where it is ready to listen for incoming connection requests.

(4) `accept` : Accepts an incoming connection request from a remote socket and creates a new socket for established connection.

(5) `recv` : Used to receive incoming data from a connected remote socket.

(6) `send` : Used to transmit data to a connected remote socket.

Malware often uses a higher-level API, particularly for HTTP and HTTPS communication, which abstracts away the complexities of raw sockets and protocol handling. These functions are typically found in the `WinINet.dll` library.

- `InternetOpen` : Establishes the initial connection session to the Internet and prepares the application for subsequent requests.
- `InternetOpenUrl` : Connects to and opens a resource at a specific URL.
- `InternetReadFile` : Reads data from the file or resource downloaded from the Internet using the handle returned by a function like `InternetOpenUrl` .

Processes and Threads. In Windows, execution occurs within processes and threads. A process is an instance of a running program that provides an isolated environment, including its own memory space, resources, and security context. A thread is an independent sequence of instructions executed within a process. Multiple threads can exist within a single process, sharing the same memory space but maintaining their own register values and stack.

`CreateProcess` is the key API function used to create a new process based on various parameters. Malware often employs `CreateProcess` to launch a new process (e.g., `cmd.exe`) and then redirects its standard input (`stdin`), standard output (`stdout`), and standard error (`stderr`) to a network socket. This redirection is managed through the `STARTUPINFO` data structure. An external process, acting as a listener, can then send commands to the newly created shell process and receive its output, effectively establishing a remote shell.

Each thread saves its operational state, including the values of all registers, in a thread context structure. This mechanism allows the operating system to manage concurrent execution by switching between threads while preserving their individual states. `CreateThread` is the function used to create a new thread of execution within the calling process. The function takes the starting address of the routine to be executed (often referred to as the `start` function or thread procedure) and a parameter to be passed to that function. Attackers can use `CreateThread` for malicious operations, such as injecting a malicious DLL. By passing the starting address of the DLL's loading mechanism (via `LoadLibrary` or directly to `DLLMain`), the thread will execute the malicious library's initialization code. It can also be used to split operations (e.g., one thread for reading and another for writing) for modularity or to evade detection.

Mutexes (Mutants). Mutexes (short for Mutual Exclusions, often called Mutants in the Windows kernel) are global synchronization objects used to control access to a shared resource among multiple threads or processes. They ensure atomicity - that only one thread or process can access a critical section of code or data at any given time, thereby preventing race conditions.

- *Acquisition and Release*: A thread or process must acquire the mutex before accessing the shared resource. Once acquired, the thread holds the mutex until it explicitly releases it.
- `WaitForSingleObject` : A thread or process calls this function to request ownership of a mutex. The thread will block (wait) until the mutex is free and is assigned to the requesting thread.
- `ReleaseMutex` : The owning thread calls this function to relinquish control of the mutex, allowing another waiting thread or process to acquire it.
- *Malware Analysis Relevance*: Mutexes are crucial for malware analysis because they often use hardcoded, unique names (e.g., a specific hash or GUID). A running instance of the malware will attempt to create or open a mutex with this unique name. If the mutex already exists, the malware knows another instance is already running, which is a common technique to prevent multiple copies from running simultaneously on the same system.

Services. Services are executable code units (binaries or DLLs) that run in the background, independent of a specific user login, and are managed by the Service Control Manager (SCM). Services typically run under the `SYSTEM` account, granting them higher privileges than a normal or even administrator user. They are a highly effective method for malware persistence because they start automatically upon system boot and are less likely to be detected by users checking the standard Task Manager's application tab.

- `OpenSCManager` : Returns a handle to the Service Control Manager database, which is required for all subsequent service-related operations.
- `CreateService` : Adds a new service entry to the SCM database, allowing the attacker to specify parameters like the service executable path and its start-up type (e.g., whether it runs automatically at boot time),
- `StartService` : Manually initiates the execution of a registered service.
- `WIN32_OWN_PROCESS` : The service's code is stored in a standalone executable file that runs in its own dedicated process.
- `WIN32_SHARE_PROCESS` : The service's code is often stored in a DLL, and the SCM allows multiple services to share a single host process. This type is famously used by `svchost.exe` to conserve system resources.
- `KERNEL_DRIVER` : Loads code directly into the Windows kernel space.

Component Object Model (COM). The Component Object Model (COM) is a language-agnostic, standardized interface for creating reusable software components that allow inter-process and intra-process communication. It works for both object-oriented (OOP) and non-OOP languages. COM allows a client application to request a specific functionality (e.g., "open a browser") without needing explicit knowledge of the underlying server process or its exact implementation details. It is frequently used in client-server architectures where a client demands services from COM objects provided by a server. COM objects are uniquely identified using Globally Unique Identifiers (GUIDs). These are often referred to as Class Identifiers (CLSID) for the object itself and Interface Identifiers (IID) for a specific set of methods the object exposes. These GUIDs are stored in the Registry, which maps the CLSID/IID to the physical location and process responsible for the component's execution. Any thread that intends to use COM objects must first initialize the COM library by calling either `CoInitialize` (for standard apartment threading model) or `OleInitialize` (which also initializes the OLE library and is used for specific COM objects).