Windows Processes

What is a Windows Process?

A Windows process is a program or application that is running on a Windows machine. A process can be started by either a user or by the system itself. The process consumes resources such as memory, disk space, and processor time to complete a task.

Process Threads

Windows processes are made up of one or more threads that are all running concurrently. A thread is a set of instructions that can be executed independently within a process. Threads within a process can communicate and share data. Threads are scheduled for execution by the operating system and managed in the context of a process.

Process Memory

Windows processes also use memory to store data and instructions. Memory is allocated to a process when it is created and the amount that is allocated can be set by the process itself. The operating system manages memory using both virtual and physical memory. Virtual memory allows the operating system to use more memory than what is physically available by creating a virtual address space that can be accessed by the applications. These virtual address spaces are divided into "pages" which are then allocated to processes.

Memory Types

Processes can have different types of memory:

- **Private memory** is dedicated to a single process and cannot be shared by other processes. This type of memory is used to store data that is specific to the process.
- **Mapped memory** can be shared between two or more processes. It is used to share data between processes, such as shared libraries, shared memory segments, and shared files. Mapped memory is visible to other processes, but is protected from being modified by other processes.
- **Image memory** contains the code and data of an executable file. It is used to store the code and data that is used by the process, such as the program's code, data, and resources. Image memory is often related to DLL files loaded into a process's address space.

Process Environment Block (PEB)

The Process Environment Block (PEB) is a data structure in Windows that contains information about a process such as its parameters, startup information, allocated heap information, and loaded DLLs, in addition to others. It is used by the operating system to store information about processes as they are

running, and is used by the Windows loader to launch applications. It also stores information about the process such as the process ID (PID) and the path to the executable.

Every process created has its own PEB data structure, that will contain its own set of information about it.

PEB Structure

The PEB struct in C is shown below. The reserved members of this struct can be ignored.

```
typedef struct _PEB {
  BYTE                          Reserved1[2];
  BYTE                          BeingDebugged;
  BYTE                          Reserved2[1];
  PVOID                         Reserved3[2];
  PPEB_LDR_DATA                 Ldr;
  PRTL_USER_PROCESS_PARAMETERS  ProcessParameters;
  PVOID                         Reserved4[3];
  PVOID                         AtlThunkSListPtr;
  PVOID                         Reserved5;
  ULONG                         Reserved6;
  PVOID                         Reserved7;
  ULONG                         Reserved8;
  ULONG                         AtlThunkSListPtr32;
  PVOID                         Reserved9[45];
  BYTE                          Reserved10[96];
  PPS_POST_PROCESS_INIT_ROUTINE PostProcessInitRoutine;
  BYTE                          Reserved11[128];
  PVOID                         Reserved12[1];
  ULONG                         SessionId;
} PEB, *PPEB;
```

The non-reserved members are explained below.

BeingDebugged

BeingDebugged is a flag in the PEB structure that indicates whether the process is being debugged or not. It is set to 1 (TRUE) when the process is being debugged and 0 (FALSE) when it is not. It is used by the Windows loader to determine whether to launch the application with a debugger attached or not.

Ldr

Ldr is a pointer to a `PEB_LDR_DATA` structure in the Process Environment Block (PEB). This structure contains information about the process's loaded dynamic link library (DLL) modules. It includes a list of the DLLs loaded in the process, the base address of each DLL, and the size of each module. It is used by the Windows loader to keep track of DLLs loaded in the process. The `PEB_LDR_DATA` struct is shown below.

```
typedef struct _PEB_LDR_DATA {
  BYTE       Reserved1[8];
  PVOID      Reserved2[3];
  LIST_ENTRY InMemoryOrderModuleList;
} PEB_LDR_DATA, *PPEB_LDR_DATA;
```

`Ldr` can be leveraged to find the base address of a particular DLL, as well as which functions reside within its memory space. This will be used in future modules to build a custom version of GetModuleHandleA/W for added stealth.

ProcessParameters

ProcessParameters is a data structure in the PEB. It contains the command line parameters passed to the process when created. The Windows loader adds these parameters to the process's PEB structure. ProcessParameters is a pointer to the `RTL_USER_PROCESS_PARAMETERS` struct that's shown below.

```
typedef struct _RTL_USER_PROCESS_PARAMETERS {
  BYTE           Reserved1[16];
  PVOID          Reserved2[10];
  UNICODE_STRING ImagePathName;
  UNICODE_STRING CommandLine;
} RTL_USER_PROCESS_PARAMETERS, *PRTL_USER_PROCESS_PARAMETERS;
```

`ProcessParameters` will be leveraged in future modules to perform actions such as command line spoofing.

AtlThunkSListPtr & AtlThunkSListPtr32

`AtlThunkSListPtr` and `AtlThunkSListPtr32` are used by the ATL (Active Template Library) module to store a pointer to a linked list of *thunking functions*. Thunking functions are used to call functions that are implemented in a different address space, these often represent functions exported from a DLL (Dynamic Link Library) file. The linked list of thunking functions is used by the ATL module to manage the thunking process.

PostProcessInitRoutine

The `PostProcessInitRoutine` field in the PEB structure is used to store a pointer to a function that is called by the operating system after TLS (Thread Local Storage) initialization has been completed for all threads in the process. This function can be used to perform any additional initialization tasks that are required for the process.

TLS and TLS callbacks will be discussed in more detail later when required.

SessionId

The SessionID in the PEB is a unique identifier assigned to a single session. It is used to track the activity of the user during the session.

Thread Environment Block (TEB)

Thread Environment Block (TEB) is a data structure in Windows that stores information about a thread. It contains the thread's environment, security context, and other related information. It is stored in the thread's stack and is used by the Windows kernel to manage threads.

TEB Structure

The TEB struct in C is shown below. The reserved members of this struct can be ignored.

```
typedef struct _TEB {
  PVOID Reserved1[12];
  PPEB  ProcessEnvironmentBlock;
  PVOID Reserved2[399];
  BYTE  Reserved3[1952];
  PVOID TlsSlots[64];
  BYTE  Reserved4[8];
  PVOID Reserved5[26];
  PVOID ReservedForOle;
  PVOID Reserved6[4];
  PVOID TlsExpansionSlots;
} TEB, *PTEB;
```

ProcessEnvironmentBlock (PEB)

Is a pointer to the PEB structure explained above, PEB is located inside the Thread Environment Block (TEB) and is used to store information about the currently running process.

TlsSlots

The TLS (Thread Local Storage) Slots are locations in the TEB that are used to store thread-specific data. Each thread in Windows has its own TEB, and each TEB has a set of TLS slots. Applications can use these slots to store data that is specific to that thread, such as thread-specific variables, thread-specific handles, thread-specific states, and so on.

TlsExpansionSlots

The TLS Expansion Slots in the TEB are a set of pointers used to store thread-local storage data for a thread. The TLS Expansion Slots are reserved for use by system DLLs.

Process And Thread Handles

On the Windows operating system, each process has a distinct process identifier or process ID (PID) which the operating system assigns when the process is created. PIDs are used to distinguish one running process from another. The same concept applies to a running thread, where a running thread has a unique ID that is used to differentiate it from the rest of the existing threads (in any process) on the system.

These identifiers can be used to open a handle to a process or a thread using the WinAPIs below.

- OpenProcess - Opens an existing process object handle via its identifier.
- OpenThread - Opens an existing thread object handle via its identifier.

These WinAPIs will be discussed in further detail later on when required. For now, it's enough to know that the opened handle can be used to perform further actions to its relative Windows object, such as suspending a process or thread.

Handles should always be closed once their use is no longer required to avoid handle leaking. This is achieved via the CloseHandle WinAPI call.