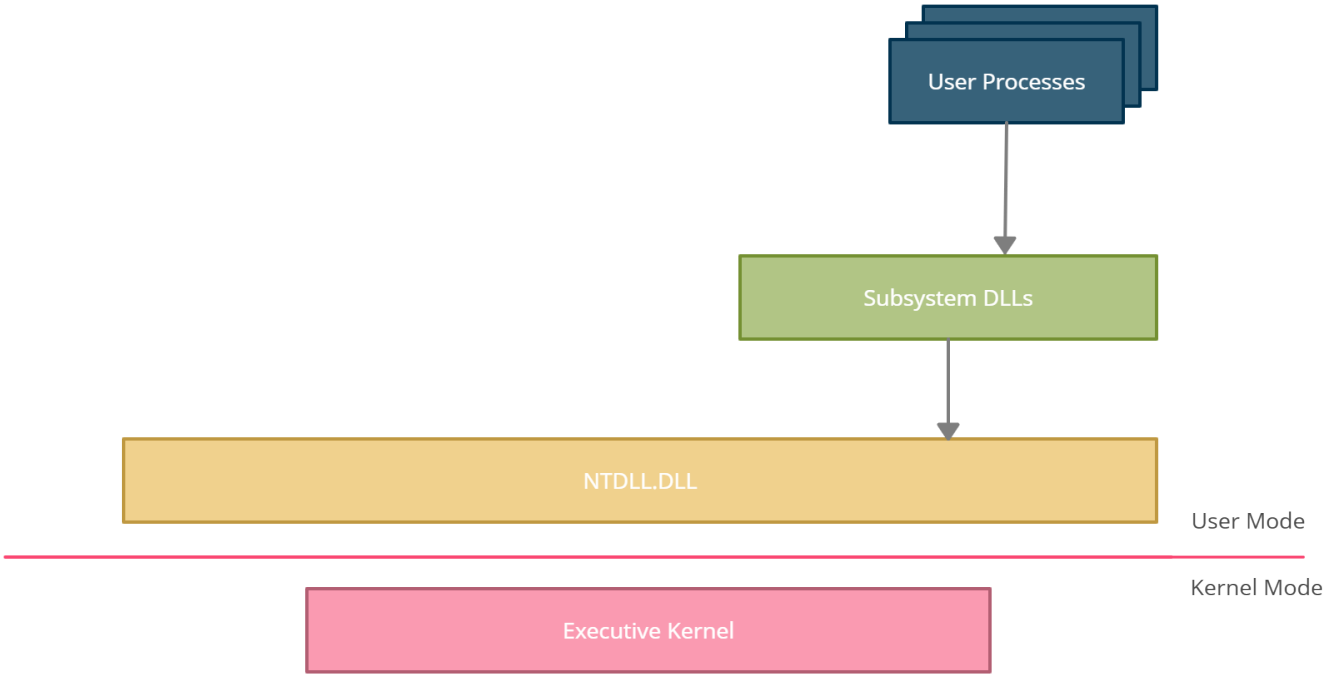
Windows Architecture

Introduction

This module explains the Windows architecture and what happens under the hood of Windows processes and applications.

A processor inside a machine running the Windows operating system can operate under two different modes: User Mode and Kernel Mode. Applications run in user mode, and operating system components run in kernel mode. When an application wants to accomplish a task, such as creating a file, it cannot do so on its own. The only entity that can complete the task is the kernel, so instead applications must follow a specific function call flow. The diagram below shows a high level of this flow.

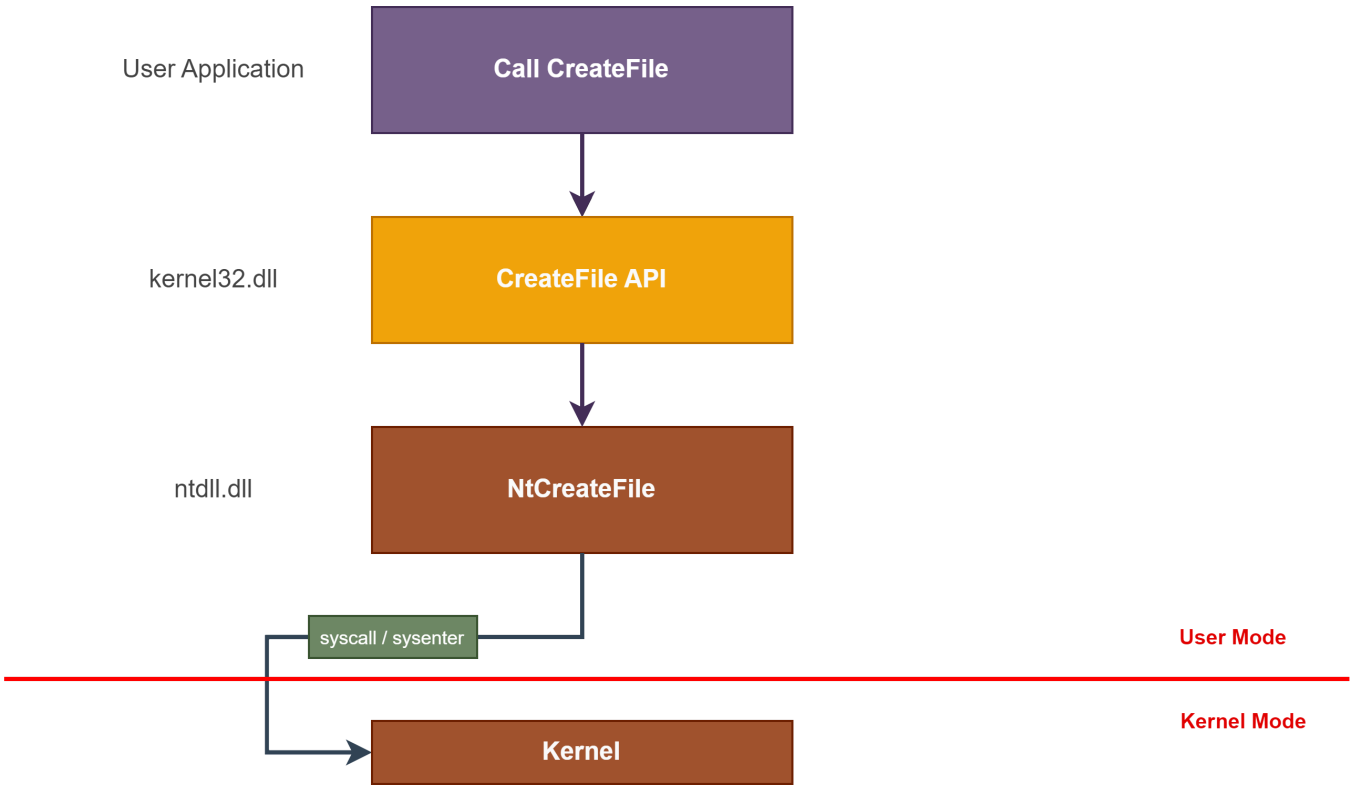


1. **User Processes** - A program/application executed by the user such as Notepad, Google Chrome or Microsoft Word.
2. **Subsystem DLLs** - DLLs that contain API functions that are called by user processes. An example of this would be `kernel32.dll` exporting the CreateFile Windows API (WinAPI) function, other common subsystem DLLs are `ntdll.dll`, `advapi32.dll`, and `user32.dll`.
3. **Ntdll.dll** - A system-wide DLL which is the lowest layer available in user mode. This is a special DLL that creates the transition from user mode to kernel mode. This is often referred to as the Native API or NTAPI.
4. **Executive Kernel** - This is what is known as the Windows Kernel and it calls other drivers and modules available within kernel mode to complete tasks. The Windows kernel is partially stored in a

file called `ntoskrnl.exe` under "C:\Windows\System32".

Function Call Flow

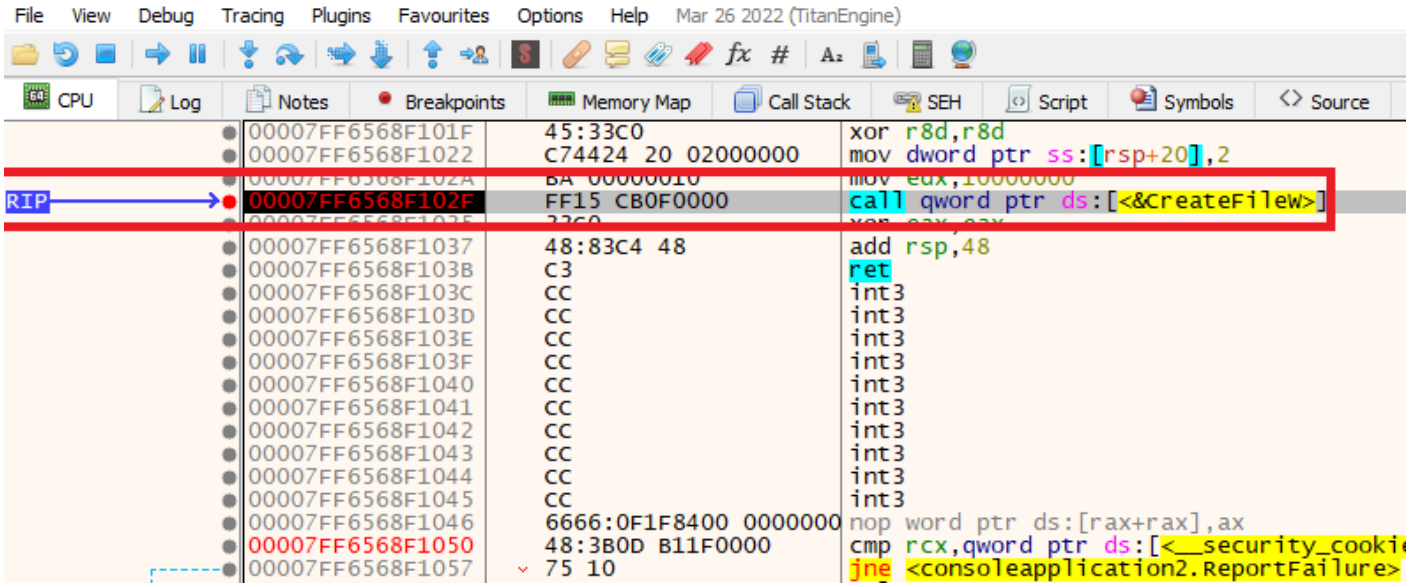
The image below shows an example of an application that creates a file. It begins with the user application calling the `CreateFile` WinAPI function which is available in `kernel32.dll`. `Kernel32.dll` is a critical DLL that exposes applications to the WinAPI and is therefore can be seen loaded by most applications. Next, `CreateFile` calls its equivalent NTAPI function, `NtCreateFile`, which is provided through `ntdll.dll`. `Ntdll.dll` then executes an assembly `sysenter` (x86) or `syscall` (x64) instruction, which transfers execution to kernel mode. The kernel `NtCreateFile` function is then used which calls kernel drivers and modules to perform the requested task.



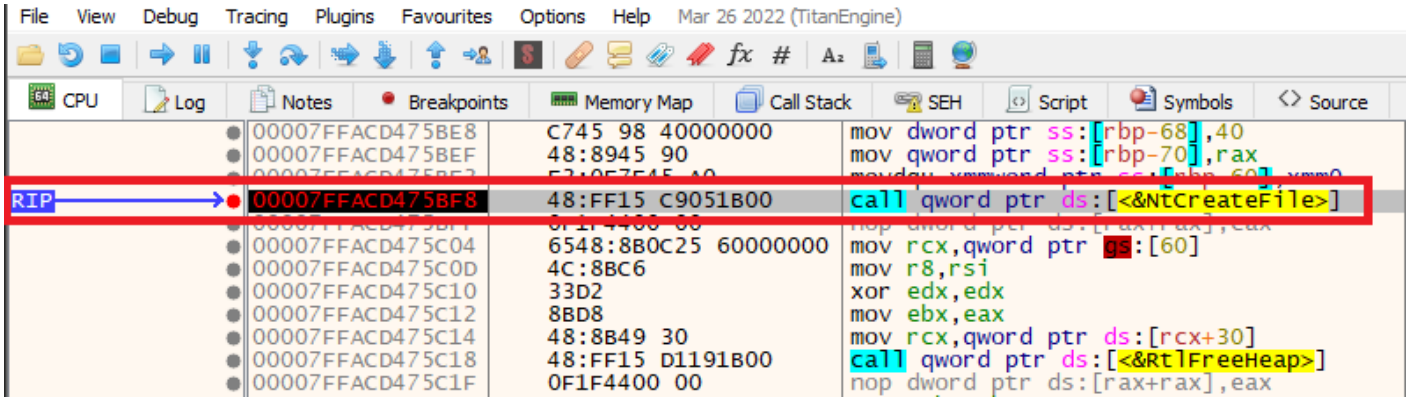
Function Call Flow Example

This example shows the function call flow happening through a debugger. This is done by attaching a debugger to a binary that creates a file via the `CreateFileW` Windows API.

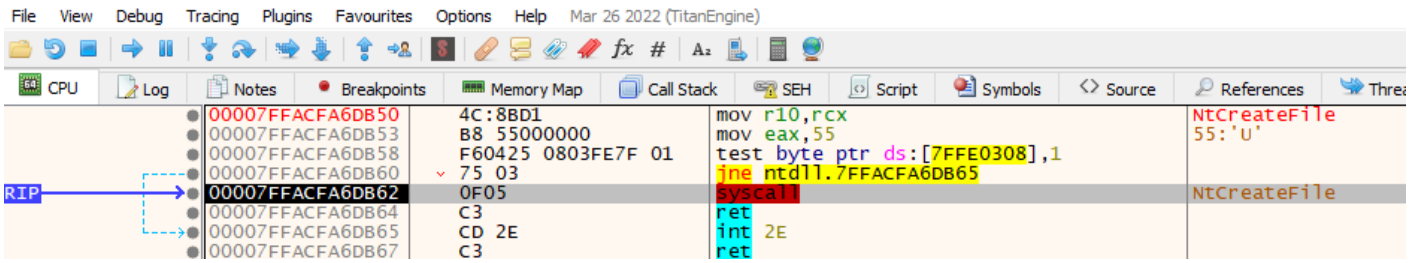
The user application calls the `CreateFileW` WinAPI.



Next, `CreateFileW` calls its equivalent NTAPI function, `NtCreateFile`.



Finally, the `NtCreateFile` function uses a `syscall` assembly instruction to transition from user mode to kernel mode. The kernel will then be the one that creates the file.



Directly Invoking The Native API (NTAPI)

It's important to note that applications can invoke syscalls (i.e. NTDLL functions) directly without having to go through the Windows API. The Windows API simply acts as a wrapper for the Native API. With that being said, the Native API is more difficult to use because it is not officially documented by Microsoft. Furthermore, Microsoft advises against the use of Native API functions because they can be changed at any time without warning.

Future modules will explore the benefits of directly invoking the Native API.