Reflective DLL Injection:

- **VirtualAllocEx:** Allocates memory within the target process's address space to hold the DLL image.
- **WriteProcessMemory:** Writes the DLL data from the attacker's source (e.g., memory buffer) into the allocated memory of the target process.
- **CreateRemoteThread:** Creates a thread within the target process to execute the injected DLL's code.
- **Kernel32.dll functions (Reflective Loader DLL):** These functions are often used by the attacker's custom DLL that performs the reflective loading. They might include:
    - **LoadLibraryA/B (Optional):** Used by the reflective DLL to locate itself in memory (can be a giveaway if used excessively).
    - **GetProcAddress:** Retrieves the address of an exported function from a DLL (legitimate use as well).
    - **VirtualProtect:** May be used to change the memory protection of the allocated region to allow for execution (less common with some reflective loaders).

DLL Injection:

- **OpenProcess:** Opens a handle to the target process, allowing interaction.
- **WriteProcessMemory:** Writes the DLL data from the attacker's source (e.g., disk file) into the memory of the target process.
- **VirtualAllocEx (Optional):** May be used in some techniques to allocate memory within the target process for the DLL if the attacker doesn't want to overwrite existing memory.
- **CreateRemoteThread:** Creates a thread within the target process to execute the injected DLL's code.
- **LoadLibraryA/B (Optional - less common):** Forces the target process to load the injected DLL directly (more common in older injection techniques).

PE Injection:

- Similar API calls to DLL injection as it involves writing a Portable Executable (PE) file (executable format) into the target process.

Process Hollowing:

Process Creation and Suspension:

- **CreateProcess (CREATE_SUSPENDED):** Creates a new process in a suspended state, allowing manipulation before its execution.

Memory Manipulation:

- **WriteProcessMemory:** Writes data (malicious payload) into the memory of the suspended target process.

- **VirtualAllocEx:** Allocates memory within the target process's address space for the malicious payload.
- **ReadProcessMemory (Optional):** May be used to read specific sections of the target process's memory (e.g., for retrieving information).
- **NtUnmapViewOfSection/ZwUnmapViewOfSection (Optional - advanced):** Unmaps sections of the target process's memory to "hollow out" space for the payload.

Thread Manipulation:

- **SetThreadContext (Optional):** Used in some techniques to manipulate the target process's thread context before resuming execution.
- **ResumeThread:** Resumes the suspended thread in the target process, effectively starting the execution of the injected payload.

Doppelganging:

Transaction Management (Core Mechanism):

- **CreateTransaction:** Creates a new transaction object.
- **CreateFileTransacted:** Opens a file handle within the context of a transaction (used to overwrite the legitimate executable).
- **RollbackTransaction:** Rolls back the transaction, effectively hiding changes made to the file system (can be a giveaway of doppelganging).

Optional Memory Manipulation:

- **VirtualAllocEx:** Allocates memory within the target process's address space (may be used for staging the malicious payload).
- **WriteProcessMemory:** Writes data (malicious code) into the allocated memory of the target process.

Process Management (Optional):

- **NtCreateProcessEx (undocumented):** Used in some doppelganging techniques to create the doppelganger process with specific flags.
- **NtContinue (undocumented):** May be used to resume execution of a suspended thread (less common).

File APIs (Optional - Depending on Technique):

- **OpenFile:** Might be used to open the legitimate executable for manipulation.
- **WriteFile:** Potentially used to overwrite parts of the legitimate executable with malicious code (can be a giveaway).
- **CreateFile**: Opens or creates a file or device (often used for accessing physical drives).
- **ReadFile**: Reads data from a file or device.
- **DeviceIoControl**: Sends control codes to device drivers (used for disk operations).

- **SetFilePointer**: Moves the file pointer to a specified location in a file

APC Injection:

- **OpenProcess:** Opens a handle to the target process, allowing interaction.
- **QueueUserAPC:** The core function for queuing an APC object (containing malicious code) to a thread in the target process.
- **VirtualAllocEx (Optional):** May be used in some techniques to allocate memory within the target process for the APC object (less common with traditional APC injection).
- **NtWaitForSingleObject/WaitForSingleObject (Optional):** Used in some variants to synchronize thread execution before or after queuing the APC.

EarlyBird Injection:

- **CreateProcess (CREATE_SUSPENDED):** Creates a new process in a suspended state, allowing manipulation before the main thread starts.
- **QueueUserAPC:** Similar to APC injection, queues an APC object containing code to inject the payload or manipulate the target process.
- **ResumeThread:** Resumes the suspended thread in the target process, triggering the APC and potentially executing the malicious code.
- **NtWriteVirtualMemory (Optional - more advanced):** Might be used in some EarlyBird techniques to directly write the payload into the target process's memory space.

API Hooking Mechanisms:

- **SetWindowsHookEx:** Installs hooks on various Windows events, including API calls.
- **UnhookWindowsHookEx:** Removes a previously set hook.

Potential Target APIs (depending on attacker intent):

- **LoadLibraryA/B:** Monitor for hooking attempts related to DLL loading.
- **WriteProcessMemory:** Watch for hooks that might manipulate process memory.
- **VirtualAllocEx:** Could be used for allocating memory for hooking code.
- **NtCreateThread/CreateRemoteThread:** Monitor for suspicious thread creation potentially used for hooking.
- **NtProtectVirtualMemory/ZwProtectVirtualMemory:** Changes memory protection, which might be used to prepare for hooking.

Optional - Memory Manipulation (more advanced):

- **NtMapViewOfSection/ZwMapViewOfSection:** May indicate attempts to manipulate memory sections for hooking purposes.
- **ZwWriteVirtualMemory:** Could be used to inject malicious code for hooking.
- **ZwAllocateVirtualMemory:** Potential allocation of memory for hooking code or data.