What is the Windows API?

The Windows API (WinAPI) is a collection of functions exposed by Microsoft to allow programs to interact with the Windows Operating System. These APIs give user-mode applications access to kernel-level features without breaking security boundaries.

The main goals of the Windows API are:

- Abstract direct hardware/system access
- Provide system services to applications
- Maintain compatibility between Windows versions
- Offer consistent programming interfaces for GUI, file systems, memory, threading, etc.

There are two major categories:

- User-Mode APIs: Functions in `kernel32.dll`, `user32.dll`, `advapi32.dll`, etc.
- Native APIs (NT API): Internal system calls in `ntdll.dll`, exported as `Nt*` and `Zw*` functions

User Mode vs Kernel Mode

User Mode	Kernel Mode
Runs with limited privileges	Runs with highest privileges (Ring 0)
Can’t directly access hardware	Has full access to hardware
Crashes don’t take down the system	Crashes cause BSOD
Calls APIs like CreateFile	Implements APIs like NtCreateFile

API Layers: Win32, NT, and Syscalls

Win32 API (High-Level)

- Most familiar to developers
- Calls like `CreateFileW`, `OpenProcess`, `VirtualAllocEx`
- Found in `kernel32.dll`, `user32.dll`

NT API (Low-Level)

- Found in `ntdll.dll`
- Prefixed by `Nt` or `Zw` (e.g., `NtOpenProcess`)
- Direct wrappers over syscalls
- Used for stealthy operations, especially when bypassing EDRs

Syscalls

- Low-level interface between `ntdll.dll` and the kernel
- `syscall` instruction in x64, `int 0x2e` in older x86
- Each syscall has a System Service Number (SSN), e.g. `NtOpenProcess` = `0x26`

Key DLLs in API Usage

DLL	Role
kernel32.dll	High-level Win32 API (file, memory, threads)
ntdll.dll	NT API / syscall stubs
user32.dll	GUI, input, windows
advapi32.dll	Registry, services, tokens, crypto
gdi32.dll	Graphics Device Interface

Dexter’s Confusion🧠🤔🤯

🧠 What makes `kernel32.dll` “high-level”?

- ✅ Simplified Function Names: Easy to remember (`CreateFileW`) compared to `NtCreateFile`.
- ✅ Error Handling: Returns user-friendly `GetLastError()` codes.
- ✅ Compatibility: Hides version-specific or architecture-specific behavior.
- ✅ Parameter Wrapping: Takes simplified arguments and internally transforms them to what the kernel expects.

🧠 How User Mode Calls Reach the Kernel (Step-by-Step)

When a program in user mode (like `notepad.exe`, `malware.exe`, or your tool) wants to perform a privileged operation (like allocating memory or accessing a file), it cannot talk to the kernel directly. It must go through layers:

🔄 Step-by-Step Flow

1. 🔹 High-Level API Call (Win32 API)
    - You call: `VirtualAllocEx()` (from `kernel32.dll`)
    - It's easy to use, developer-friendly.

2. 🔹 **Wrapper Calls Native API (ntdll.dll)**
    - ○ **Internally, `kernel32.dll` calls a function like `NtAllocateVirtualMemory()` from `ntdll.dll`**
    - ○ **This is the Native API — a low-level API still in user mode.**
    - ○ **`ntdll.dll` knows how to prepare the syscall number and parameters.**
3. 🔻 **Native API Issues a Syscall**
    - ○ **`ntdll.dll` uses the `syscall` instruction (or `int 0x2e` on older systems)**
    - ○ **This switches from user mode (ring 3) to kernel mode (ring 0)**
4. 🔴 **Kernel Handles the Request**
    - ○ **The Windows kernel (`ntoskrnl.exe`) receives the syscall**
    - ○ **It checks permissions and executes the request (e.g., allocates memory)**

💡 Why `ntdll.dll` Is the Key Link

- **`ntdll.dll` is the only user-mode DLL that contains actual syscall stubs**

- **It's the bridge between user mode and the kernel**

[Your App]

|

▼

[Win32 API] → kernel32.dll / user32.dll (high-level)

|

▼

[Native API] → ntdll.dll (contains syscall stubs)

|

▼

[Syscall] → Executes syscall instruction (switches to ring 0)

|

▼

[Windows Kernel] → ntoskrnl.exe handles it

❓ Can any API interact with kernel mode?

🛑 No — not every API call will succeed.

Even if an API reaches the kernel, Windows performs strict permission checks at the kernel level before allowing access to sensitive resources.

🔄 Full Flow: User Mode → Kernel Mode (Permission Checks Happen)

When a user-mode program makes a system call (via `ntdll.dll`), here's what happens:

1. ✅ **Your code (user mode) calls an API (Win32 or Native)**
2. ✅ **The syscall is executed → transitions to kernel mode**

3. **🔐 The kernel checks permissions**
    - **Process token (user SID + group memberships)**
    - **Access rights (e.g., `PROCESS_VM_WRITE`, `FILE_WRITE_DATA`)**
    - **Object security descriptors (ACLs)**
4. **✅/❌ The kernel approves or denies the request**

So, even if the API reaches the kernel, success depends on the access rights and security context of the calling process.