Anti-Debugging and Self-Deletion Routine

By Ouzidane Reda

C Code

```c
#include <windows.h>
#include <winternl.h>
#include <stdio.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__)

// 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);
    QueryPerformanceCounter(&t2);
    if ((t2.QuadPart - t1.QuadPart) > 1000) // Threshold may need adjustment
        return TRUE;

    return FALSE;
}

// Optimized self-deletion function using native API
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) * sizeof(WCHAR);

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

    // Open file with native API for better performance
    UNICODE_STRING filePath;
    RtlInitUnicodeString(&filePath, wszFilePath);

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

    status = 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))
    {
        warn("NtCreateFile failed: 0x%08X", status);
        return status;
    }

    // Allocate rename info structure
    RenameSize = sizeof(FILE_RENAME_INFO) + streamLen;
    pRenameInfo = (PFILE_RENAME_INFO)HeapAlloc(GetProcessHeap(), HEAP_ZERO_MEMORY,
    RenameSize);
    if (!pRenameInfo)
    {
        warn("HeapAlloc failed: 0x%08X", GetLastError());
        NtClose(hFile);
        return STATUS_NO_MEMORY;
    }

    // Set up rename info
    pRenameInfo->FileNameLength = streamLen;
    RtlCopyMemory(pRenameInfo->FileName, MEMSTREAM, streamLen);

    // Rename file to alternate data stream
    status = NtSetInformationFile(hFile,
                                  &ioStatus,
                                  pRenameInfo,
                                  RenameSize,
                                  FileRenameInformation);

    if (!NT_SUCCESS(status))
    {
        warn("NtSetInformationFile (rename) failed: 0x%08X", status);
        HeapFree(GetProcessHeap(), 0, pRenameInfo);
        NtClose(hFile);
        return status;
    }

    // Close handle to commit changes
    NtClose(hFile);

    // Reopen file for deletion
    status = 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))
    {
        warn("NtCreateFile (reopen) failed: 0x%08X", status);
        HeapFree(GetProcessHeap(), 0, pRenameInfo);
        return status;
    }

    // Mark file for deletion
    deleteInfo.DeleteFile = TRUE;

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

    if (!NT_SUCCESS(status))
    {
        warn("NtSetInformationFile (delete) failed: 0x%08X", status);
        HeapFree(GetProcessHeap(), 0, pRenameInfo);
        NtClose(hFile);
        return status;
    }

    // Close handle to actually delete the file
    NtClose(hFile);
    HeapFree(GetProcessHeap(), 0, pRenameInfo);

    return STATUS_SUCCESS;
}

int main(int argc, char* argv[])
{
    if (!CheckDebuggerEnhanced())
    {
        info("Debugger not detected, executing payload");
        MessageBoxW(NULL, L"KAW KAW KAW", L"NIGHTMARE", MB_ICONEXCLAMATION);
    }
    else
    {
        warn("Debugger detected! Initiating self-destruct sequence");
        SelfDeleteOptimized();
    }

    return 0;
}
```