

Anti-Debugging and Self-Deletion Routine

By Ouzidane Reda

C Code

```
1 #include <windows.h>
2 #include <winternl.h>
3 #include <stdio.h>
4
5 // Enhanced logging macros
6 #define okay(msg, ...) printf("[+] " msg "\n", ##__VA_ARGS__)
7 #define info(msg, ...) printf("[i] " msg "\n", ##__VA_ARGS__)
8 #define warn(msg, ...) printf("[-] " msg "\n", ##__VA_ARGS__)
9
10 // Inline assembly for anti-debugging
11 __forceinline BOOL IsDebuggerPresentASM()
12 {
13     __asm {
14         mov eax, fs:[30h] // PEB
15         movzx eax, byte ptr [eax+2] // BeingDebugged
16     }
17 }
18
19 // Hardware breakpoint detection
20 BOOL CheckHardwareBreakpoints()
21 {
22     CONTEXT ctx = { 0 };
23     ctx.ContextFlags = CONTEXT_DEBUG_REGISTERS;
24
25     if (!GetThreadContext(GetCurrentThread(), &ctx))
26         return FALSE;
27
28     return (ctx.Dr0 || ctx.Dr1 || ctx.Dr2 || ctx.Dr3);
29 }
30
31 // Enhanced PEB check with obfuscation
32 __forceinline PPEB GetPEBEnhanced()
33 {
34     PPEB pPeb;
35     __asm {
36         xor eax, eax
37         mov eax, fs:[0x30]
38         mov pPeb, eax
39     }
40     return pPeb;
41 }
42
43 // Anti-debugging function with multiple techniques
44 BOOL CheckDebuggerEnhanced()
45 {
46     // 1. Standard PEB check
47     PPEB pPEB = GetPEBEnhanced();
48     if (pPEB->BeingDebugged)
49         return TRUE;
50
51     // 2. NtGlobalFlag check
52     if (pPEB->NtGlobalFlag & (FLG_HEAP_ENABLE_TAIL_CHECK | FLG_HEAP_ENABLE_FREE_CHECK |
53         FLG_HEAP_VALIDATE_PARAMETERS))
```

```

53         return TRUE;
54
55     // 3. Hardware breakpoint check
56     if (CheckHardwareBreakpoints())
57         return TRUE;
58
59     // 4. ASM check
60     if (IsDebuggerPresentASM())
61         return TRUE;
62
63     // 5. QueryPerformanceCounter timing check
64     LARGE_INTEGER t1, t2;
65     QueryPerformanceCounter(&t1);
66     QueryPerformanceCounter(&t2);
67     if ((t2.QuadPart - t1.QuadPart) > 1000) // Threshold may need adjustment
68         return TRUE;
69
70     return FALSE;
71 }
72
73 // Optimized self-deletion function using native API
74 NTSTATUS SelfDeleteOptimized()
75 {
76     NTSTATUS status;
77     HANDLE hFile = NULL;
78     SIZE_T RenameSize;
79     PFILE_RENAME_INFO pRenameInfo = NULL;
80     WCHAR wszFilePath[MAX_PATH * 2] = { 0 };
81     FILE_DISPOSITION_INFO deleteInfo = { 0 };
82     IO_STATUS_BLOCK ioStatus = { 0 };
83
84     const wchar_t* MEMSTREAM = L":CRON";
85     const size_t streamLen = wcslen(MEMSTREAM) * sizeof(WCHAR);
86
87     // Get current executable path
88     if (!GetModuleFileNameW(NULL, wszFilePath, MAX_PATH * 2))
89     {
90         warn("GetModuleFileNameW failed: 0x%08X", GetLastError());
91         return STATUS_UNSUCCESSFUL;
92     }
93
94     // Open file with native API for better performance
95     UNICODE_STRING filePath;
96     RtlInitUnicodeString(&filePath, wszFilePath);
97
98     OBJECT_ATTRIBUTES objAttr = { 0 };
99     InitializeObjectAttributes(&objAttr, &filePath, OBJ_CASE_INSENSITIVE, NULL, NULL);
100
101     status = NtCreateFile(&hFile,
102                         DELETE | SYNCHRONIZE,
103                         &objAttr,
104                         &ioStatus,
105                         NULL,
106                         FILE_ATTRIBUTE_NORMAL,
107                         FILE_SHARE_READ,
108                         FILE_OPEN,
109                         FILE_SYNCHRONOUS_IO_NONALERT,
110                         NULL,
111                         0);
112
113     if (!NT_SUCCESS(status))
114     {
115         warn("NtCreateFile failed: 0x%08X", status);
116         return status;
117     }
118
119     // Allocate rename info structure
120     RenameSize = sizeof(FILE_RENAME_INFO) + streamLen;

```

```

121 pRenameInfo = (PFILE_RENAME_INFO)HeapAlloc(GetProcessHeap(), HEAP_ZERO_MEMORY,
122 RenameSize);
123 if (!pRenameInfo)
124 {
125     warn("HeapAlloc failed: 0x%08X", GetLastError());
126     NtClose(hFile);
127     return STATUS_NO_MEMORY;
128 }
129
130 // Set up rename info
131 pRenameInfo->FileNameLength = streamLen;
132 RtlCopyMemory(pRenameInfo->FileName, MEMSTREAM, streamLen);
133
134 // Rename file to alternate data stream
135 status = NtSetInformationFile(hFile,
136                               &ioStatus,
137                               pRenameInfo,
138                               RenameSize,
139                               FileRenameInformation);
140
141 if (!NT_SUCCESS(status))
142 {
143     warn("NtSetInformationFile (rename) failed: 0x%08X", status);
144     HeapFree(GetProcessHeap(), 0, pRenameInfo);
145     NtClose(hFile);
146     return status;
147 }
148
149 // Close handle to commit changes
150 NtClose(hFile);
151
152 // Reopen file for deletion
153 status = NtCreateFile(&hFile,
154                      DELETE | SYNCHRONIZE,
155                      &objAttr,
156                      &ioStatus,
157                      NULL,
158                      FILE_ATTRIBUTE_NORMAL,
159                      FILE_SHARE_READ,
160                      FILE_OPEN,
161                      FILE_SYNCHRONOUS_IO_NONALERT,
162                      NULL,
163                      0);
164
165 if (!NT_SUCCESS(status))
166 {
167     warn("NtCreateFile (reopen) failed: 0x%08X", status);
168     HeapFree(GetProcessHeap(), 0, pRenameInfo);
169     return status;
170 }
171
172 // Mark file for deletion
173 deleteInfo.DeleteFile = TRUE;
174
175 status = NtSetInformationFile(hFile,
176                               &ioStatus,
177                               &deleteInfo,
178                               sizeof(deleteInfo),
179                               FileDispositionInformation);
180
181 if (!NT_SUCCESS(status))
182 {
183     warn("NtSetInformationFile (delete) failed: 0x%08X", status);
184     HeapFree(GetProcessHeap(), 0, pRenameInfo);
185     NtClose(hFile);
186     return status;
187 }

```

```

188 // Close handle to actually delete the file
189 NtClose(hFile);
190 HeapFree(GetProcessHeap(), 0, pRenameInfo);
191
192 return STATUS_SUCCESS;
193 }
194
195 int main(int argc, char* argv[])
196 {
197     if (!CheckDebuggerEnhanced())
198     {
199         info("Debugger not detected, executing payload");
200         MessageBoxW(NULL, L"KAW KAW KAW", L"NIGHTMARE", MB_ICONEXCLAMATION);
201     }
202     else
203     {
204         warn("Debugger detected! Initiating self-destruct sequence");
205         SelfDeleteOptimized();
206     }
207
208     return 0;
209 }

```