



Reverse Engineering (REV3)

UE 03 – Anti-Debugging – Protokoll

Jakob Mayr

WS 2023/2024

Einleitung

...

Implementierung

C

```
1  #include <windows.h>
2
3  /**
4   * This function is a custom unhandled exception filter.
5   *
6   * @param ExceptionInfo A pointer to an EXCEPTION_POINTERS structure that
7   *                      contains information about the exception.
8   * @return A constant that determines how the exception is handled.
9   *
10  * The function displays a message box with the text "Infected" and
11  * an icon indicating a warning. It's meant to be invoked when an
12  * unhandled exception occurs. The function then returns a constant
13  * indicating that the exception handler (EXCEPTION_EXECUTE_HANDLER)
14  * should be executed.
15  */
16  LONG WINAPI MyUnhandledExceptionFilter(struct _EXCEPTION_POINTERS* ExceptionInfo) {
17      MessageBoxW(NULL, L"Infected", L"Infected", MB_ICONWARNING);
18
19      //ExitProcess(0);
20
21      return EXCEPTION_EXECUTE_HANDLER;
22  }
23
24  /**
25   * The main entry point for the application.
26   *
27   * @return An integer indicating the success or failure of the program.
28   *
29   * The function first checks if a debugger is present using IsDebuggerPresent().
30   * If a debugger is detected, it displays a message box with "Hello" and then
31   * terminates. If no debugger is detected, it sets a custom unhandled
32   * exception filter and then deliberately triggers a breakpoint exception
33   * using DebugBreak(). If the breakpoint is unhandled, it will invoke the
34   * custom unhandled exception filter. After handling the exception,
35   * or if the breakpoint is handled by a debugger, it displays a "Hello" message
36   * box and then exits.
37   *
38   * NOTE: It might happen, that when executing this code on a system where a
39   * debugger is just installed but not running, "DebugBreak()" or "INT 3" might
40   * start the debugger automatically.
41   */
42  int main() {
43
44      if (IsDebuggerPresent()) {
45          MessageBoxW(0, (LPCWSTR)L"Hello", (LPCWSTR)L"Hello", MB_OK);
46          return 0;
47      }
48      SetUnhandledExceptionFilter(MyUnhandledExceptionFilter);
49      DebugBreak();
50      MessageBoxW(NULL, L"Hello", L"Hello", MB_OK);
51
52      return 0;
53  }
54
55
```

Assembler

```
1 .386
2 .model flat,stdcall
3 option casemap :none
4
5 include \masm32\include\windows.inc
6 include \masm32\include\kernel32.inc
7 include \masm32\include\user32.inc
8 includelib \masm32\lib\kernel32.lib
9 includelib \masm32\lib\user32.lib
10
11 .data
12 ; Define strings to be used in the message boxes.
13 HelloWorld dw 'H','e','l','l','o',0
14 Infected dw 'I','n','f','e','c','t','e','d',0
15
16 .code
17 start:
18 ; Entry point of the program. Jumps to the 'begin' label.
19 jmp begin
20
21 NoDebugger:
22 ; This section executes if no debugger is detected.
23 ; Displays a message box with the text "Infected" and a warning icon.
24 push MB_ICONWARNING
25 push offset Infected
26 push offset Infected
27 push 0
28 call MessageBoxW
29 jmp ExitProcessCall
30
31 begin:
32 ; Checks if a debugger is present.
33 ; Calls IsDebuggerPresent and examines the return value.
34 call IsDebuggerPresent
35 test eax, eax
36 jnz DebuggerFound ; If EAX is non-zero, a debugger is detected, jump to DebuggerFound.
37
38 ; If no debugger is detected, set up an exception filter and trigger a breakpoint.
39 invoke SetUnhandledExceptionFilter, NoDebugger
40 int 3 ; Software breakpoint - triggers an exception if no debugger is present.
41 ; Continue to DebuggerFound if execution reaches here.
42 jmp DebuggerFound
43
44 DebuggerFound:
45 ; This section executes if a debugger is detected.
46 ; Displays a message box with the text "Hello".
47 push MB_OK
48 push offset HelloWorld
49 push offset HelloWorld
50 push 0
51 call MessageBoxW
52
53 ExitProcessCall:
54 ; Terminates the process.
55 push 0
56 call ExitProcess
57 end start
58
```

Umgehung - Manuell

C

Um die Anti-debugging-Checks zu umgehen, gibt es mehrere Möglichkeiten.

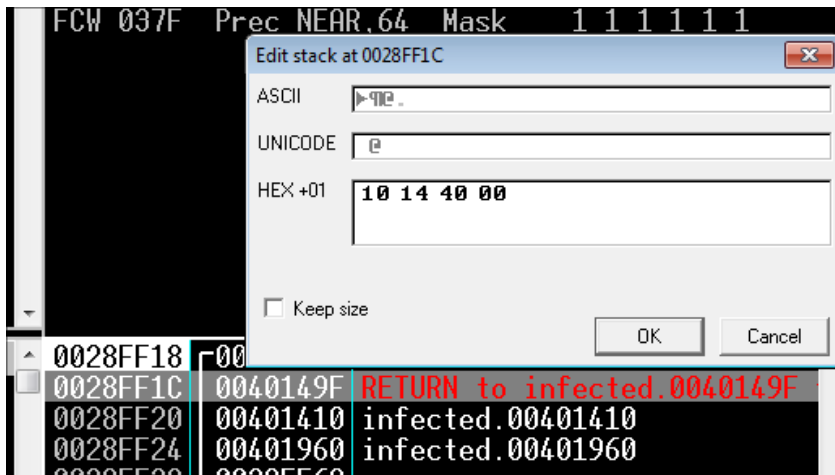
Für den Mechanismus "IsDebuggerPresent()" wurde der "JZ" nach dem Check zu einem "JNZ" geändert (umgekehrt ist dies natürlich auch möglich). Man siehe Adresse "00401463":

00401457	. E8 94050000	CALL infected.004019F0	
0040145C	. E8 7F270000	CALL <JMP.&KERNEL32.IsDebuggerPresent>	[IsDebuggerPresent
00401461	. 85C0	TEST EAX, EAX	
00401463	. JZ 2E	JNZ SHORT infected.00401493	
00401465	. C74424 0C 0000	MOV DWORD PTR SS:[ESP+C], 0	
0040146D	. C74424 08 5654	MOV DWORD PTR SS:[ESP+8], infected.00405056	UNICODE "Hello"
00401475	. C74424 04 5654	MOV DWORD PTR SS:[ESP+4], infected.00405056	UNICODE "Hello"
0040147D	. C70424 00000000	MOV DWORD PTR SS:[ESP], 0	
00401484	. E8 1F270000	CALL <JMP.&USER32.MessageBoxW>	[-MessageBoxW
00401489	. 83EC 10	SUB ESP, 10	
0040148C	. B8 00000000	MOV EAX, 0	
00401491	. EB 40	JMP SHORT infected.004014D3	
00401493	.> C70424 101440	MOV DWORD PTR SS:[ESP], infected.00401410	
0040149A	. E8 29270000	CALL <JMP.&KERNEL32.SetUnhandledExceptionFilter>	[-SetUnhandledExceptionFilter
0040149F	. 83EC 04	SUB ESP, 4	
004014A2	. E8 B1270000	CALL <JMP.&KERNEL32.DebugBreak>	[DebugBreak
004014A7	. C74424 0C 0000	MOV DWORD PTR SS:[ESP+C], 0	
004014AF	. C74424 08 5654	MOV DWORD PTR SS:[ESP+8], infected.00405056	UNICODE "Hello"
004014B7	. C74424 04 5654	MOV DWORD PTR SS:[ESP+4], infected.00405056	UNICODE "Hello"
004014BF	. C70424 00000000	MOV DWORD PTR SS:[ESP], 0	
004014C6	. E8 DD260000	CALL <JMP.&USER32.MessageBoxW>	[-MessageBoxW
004014CB	. 83EC 10	SUB ESP, 10	

Für den Mechanismus "Unhandled Exception" muss ein wenig aufwendiger vorgegangen werden. Die Funktion "SetUnhandledExceptionFilter" setzt den Code welcher bei einer unabgearbeiteten (unhandled) Ausnahme (Exception) ausgeführt werden soll. Im folgenden Screenshot ist zu sehen, dass der Code an Adresse 00401410 ausgeführt werden soll:

00401410	. S5	PUSH EBP	
00401411	. 89E5	MOV EBP, ESP	
00401413	. 83EC 18	SUB ESP, 18	
00401416	. C74424 0C 0000	MOV DWORD PTR SS:[ESP+C], 30	
0040141E	. C74424 08 4454	MOV DWORD PTR SS:[ESP+8], infected.00405044	UNICODE "Infected"
00401426	. C74424 04 4454	MOV DWORD PTR SS:[ESP+4], infected.00405044	UNICODE "Infected"
0040142E	. C70424 00000000	MOV DWORD PTR SS:[ESP], 0	
00401435	. E8 6E270000	CALL <JMP.&USER32.MessageBoxW>	[-MessageBoxW
0040143A	. 83EC 10	SUB ESP, 10	
0040143D	. B8 01000000	MOV EAX, 1	
00401442	. C9	LEAVE	
00401443	. C2 0400	RETN 4	
00401446	.> 8D4C24 04	LEA ECX, DWORD PTR SS:[ESP+4]	
0040144A	. 83E4 F0	AND ESP, FFFFFFF0	
0040144D	. FF71 FC	PUSH DWORD PTR DS:[ECX-4]	
00401450	. S5	PUSH EBP	
00401451	. 89E5	MOV EBP, ESP	
00401453	. 51	PUSH ECX	
00401454	. 83EC 14	SUB ESP, 14	
00401457	. E8 94050000	CALL infected.004019F0	
0040145C	. E8 7F270000	CALL <JMP.&KERNEL32.IsDebuggerPresent>	[IsDebuggerPresent
00401461	. 85C0	TEST EAX, EAX	
00401463	. JZ 2E	JNZ SHORT infected.00401493	
00401465	. C74424 0C 0000	MOV DWORD PTR SS:[ESP+C], 0	
0040146D	. C74424 08 5654	MOV DWORD PTR SS:[ESP+8], infected.00405056	UNICODE "Hello"
00401475	. C74424 04 5654	MOV DWORD PTR SS:[ESP+4], infected.00405056	UNICODE "Hello"
0040147D	. C70424 00000000	MOV DWORD PTR SS:[ESP], 0	
00401484	. E8 1F270000	CALL <JMP.&USER32.MessageBoxW>	[-MessageBoxW
00401489	. 83EC 10	SUB ESP, 10	
0040148C	. B8 00000000	MOV EAX, 0	
00401491	. EB 40	JMP SHORT infected.004014D3	
00401493	.> C70424 101440	MOV DWORD PTR SS:[ESP], infected.00401410	
0040149A	. E8 29270000	CALL <JMP.&KERNEL32.SetUnhandledExceptionFilter>	[-SetUnhandledExceptionFilter
0040149F	. 83EC 04	SUB ESP, 4	
004014A2	. E8 B1270000	CALL <JMP.&KERNEL32.DebugBreak>	[DebugBreak
004014A7	. C74424 0C 0000	MOV DWORD PTR SS:[ESP+C], 0	
004014AF	. C74424 08 5654	MOV DWORD PTR SS:[ESP+8], infected.00405056	UNICODE "Hello"
004014B7	. C74424 04 5654	MOV DWORD PTR SS:[ESP+4], infected.00405056	UNICODE "Hello"
004014BF	. C70424 00000000	MOV DWORD PTR SS:[ESP], 0	

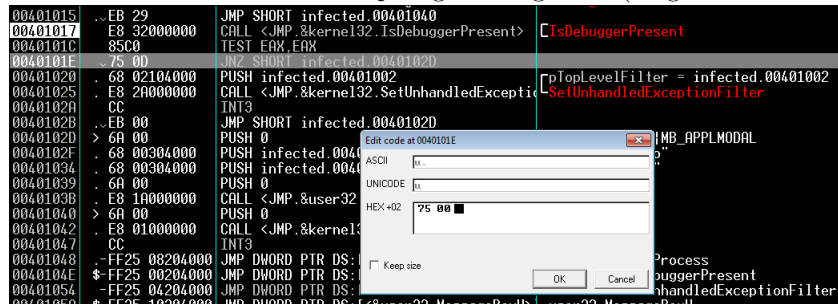
Wird nun allerdings eine Ausnahme (Exception) ausgeführt, so arbeitet der Debugger diese anders ab und im Stack landet somit auch eine andere Rücksprungadresse. Diese kann allerdings während der Laufzeit geändert werden:



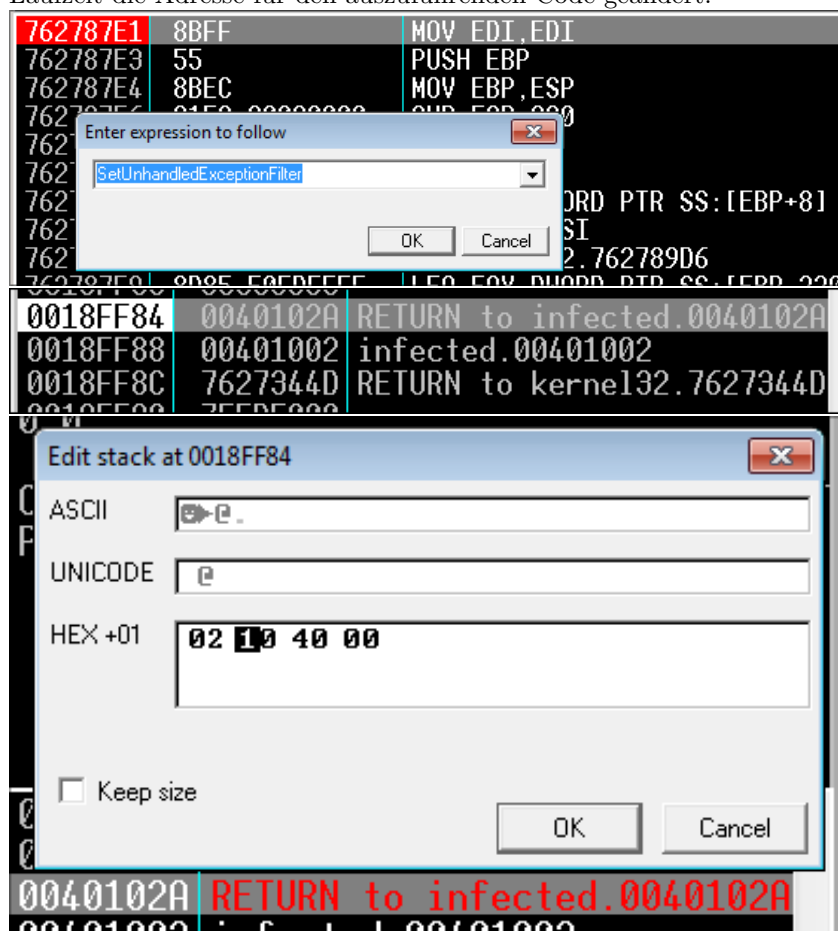
NOTE: Die Verwendung von "Ctrl+G" ist hierbei sehr hilfreich um Breakpoints an den einzelnen Funktionen zu setzen.

Assembler

In Assembler könnte man ident vorgehen. Für den Mechanismus "IsDebuggerPresent()" wurde jedoch der "offset" des konditionalen Sprungs auf 0 gesetzt (Folge: macht nichts).



Für den Mechanismus "Unhandled Exception" wurde wieder ein Breakpoint gesetzt und während der Laufzeit die Adresse für den auszuführenden Code geändert.

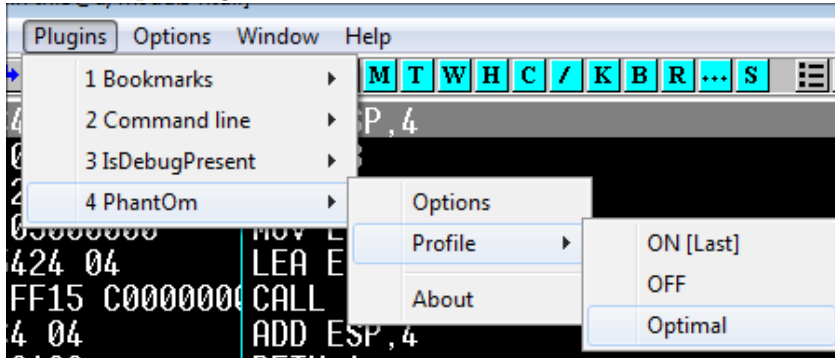


1 Plugins im Debugger

Nach Recherche im Internet wurde als Plugin die PhantOm.dll von "Roman Zaikin" t verwendet:

<https://github.com/romanzaikin/OllyDbg-v1.10-With-Best-Plugins-And-Immunity-Debugger-theme/blob/master/PhantOm.dll>

Aktiviert man in OllyDbg das Plugin mit dem Profil "Optimal", so werden die Anti-Debugging Mechanismen simpel ausgehebelt.



Aktivierte Optionen des Plugins:

