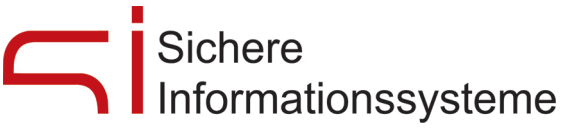


Reverse Engineering (REV3)

UE 03 – Anti-Debugging – Protokoll

Jakob Mayr

WS 2023/2024

Einleitung

...

Implementierung

C

```c
#include <windows.h>

/**
* This function is a custom unhandled exception filter.
*
* @param ExceptionInfo A pointer to an EXCEPTION_POINTERS structure that
*                      contains information about the exception.
* @return A constant that determines how the exception is handled.
*
* The function displays a message box with the text "Infected" and
* an icon indicating a warning. It's meant to be invoked when an
* unhandled exception occurs. The function then returns a constant
* indicating that the exception handler (EXCEPTION_EXECUTE_HANDLER)
* should be executed.
*/
LONG WINAPI MyUnhandledExceptionFilter(struct _EXCEPTION_POINTERS* ExceptionInfo) {
  MessageBoxW(NULL, L"Infected", L"Infected", MB_ICONWARNING);

  //ExitProcess(0);

  return EXCEPTION_EXECUTE_HANDLER;
}

/**
* The main entry point for the application.
*
* @return An integer indicating the success or failure of the program.
*
* The function first checks if a debugger is present using IsDebuggerPresent().
* If a debugger is detected, it displays a message box with "Hello" and then
* terminates. If no debugger is detected, it sets a custom unhandled
* exception filter and then deliberately triggers a breakpoint exception
* using DebugBreak(). If the breakpoint is unhandled, it will invoke the
* custom unhandled exception filter. After handling the exception,
* or if the breakpoint is handled by a debugger, it displays a "Hello" message
* box and then exits.
*
* NOTE: It might happen, that when executing this code on a system where a
* debugger is just installed but not running, "DebugBreak()" or "INT 3" might
* start the debugger automatically.
*/
int main() {

  if (IsDebuggerPresent()) {
    MessageBoxW(0, (LPCWSTR)L"Hello", (LPCWSTR)L"Hello", MB_OK);
    return 0;
  }
  SetUnhandledExceptionFilter(MyUnhandledExceptionFilter);
  DebugBreak();
  MessageBoxW(NULL, L"Hello", L"Hello", MB_OK);

  return 0;
}


```

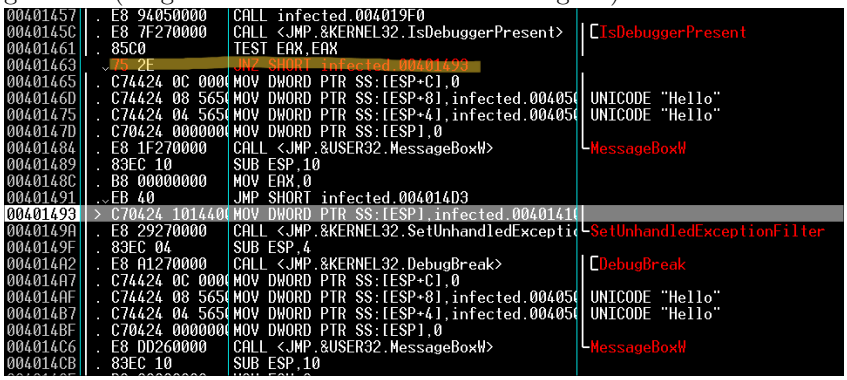
Assembler

```
.386
.model flat,stdcall
option casemap :none

include \masm32\include\windows.inc
include \masm32\include\kernel32.inc
include \masm32\include\user32.inc
includelib \masm32\lib\kernel32.lib
includelib \masm32\lib\user32.lib

.data
; Define strings to be used in the message boxes.
HelloWorld dw 'H','e','l','l','o',0
Infected dw 'I','n','f','e','c','t','e','d',0

.code
start:
  ; Entry point of the program. Jumps to the 'begin' label.
  jmp begin

NoDebugger:
  ; This section executes if no debugger is detected.
  ; Displays a message box with the text "Infected" and a warning icon.
  push MB_ICONWARNING
  push offset Infected
  push offset Infected
  push 0
  call MessageBoxW
  jmp ExitProcessCall

begin:
  ; Checks if a debugger is present.
  ; Calls IsDebuggerPresent and examines the return value.
  call IsDebuggerPresent
  test eax, eax
  jnz DebuggerFound ; If EAX is non-zero, a debugger is detected, jump to DebuggerFound.

  ; If no debugger is detected, set up an exception filter and trigger a breakpoint.
  invoke SetUnhandledExceptionFilter, NoDebugger
  int 3 ; Software breakpoint - triggers an exception if no debugger is present.
  ; Continue to DebuggerFound if execution reaches here.
  jmp DebuggerFound

DebuggerFound:
  ; This section executes if a debugger is detected.
  ; Displays a message box with the text "Hello".
  push MB_OK
  push offset HelloWorld
  push offset HelloWorld
  push 0
  call MessageBoxW

ExitProcessCall:
  ; Terminates the process.
  push 0
  call ExitProcess
  end start

```

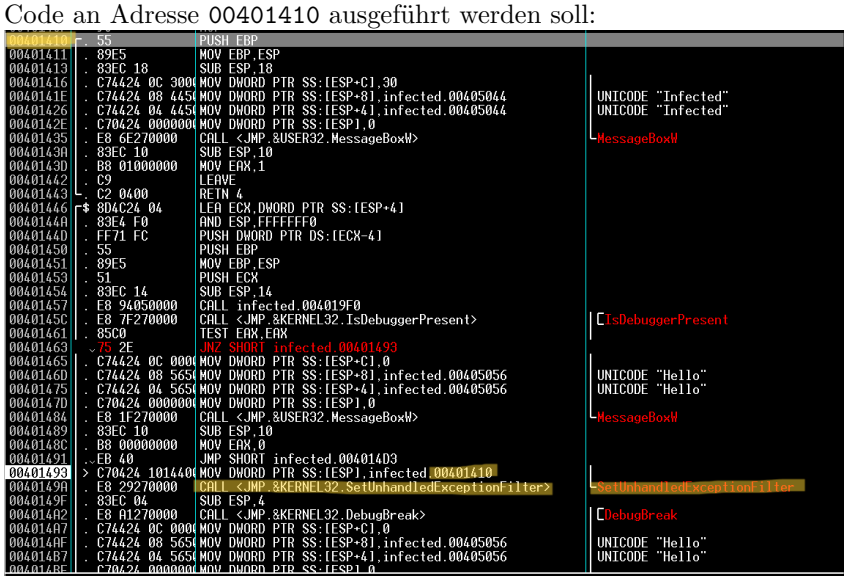
Umgehung - Manuell

C

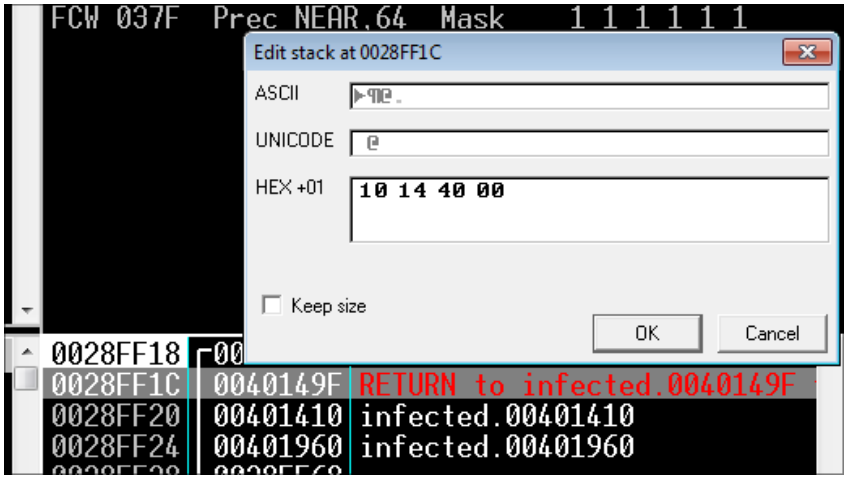
Um die Anti-debugging-Checks zu umgehen, gibt es mehrere Möglichkeiten.
Für den Mechanismus "`IsDebuggerPresent()`" wurde der "JZ" nach dem Check zu einem "JNZ" geändert (umgekehrt ist dies natürlich auch möglich). Man siehe Addresse "`00401463`":



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:



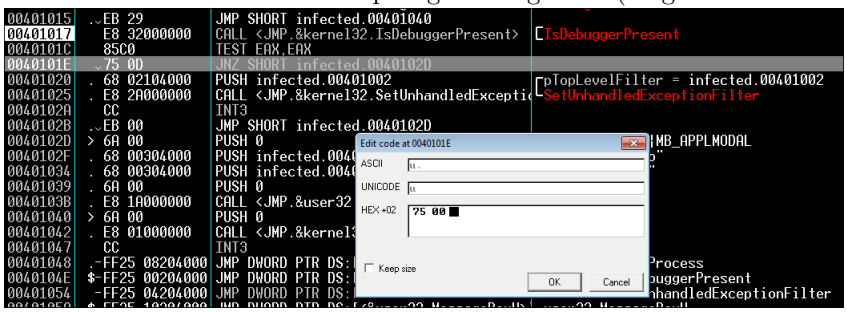
Wird nun allerdings eine Ausnahme (Exception) ausgeführt, so arbeitet der Debugger diese anders ab und im Stack landet somit auch eine andere Rücksprungadresse. Diese kann allerdings während der Laufzeit geändert werden:



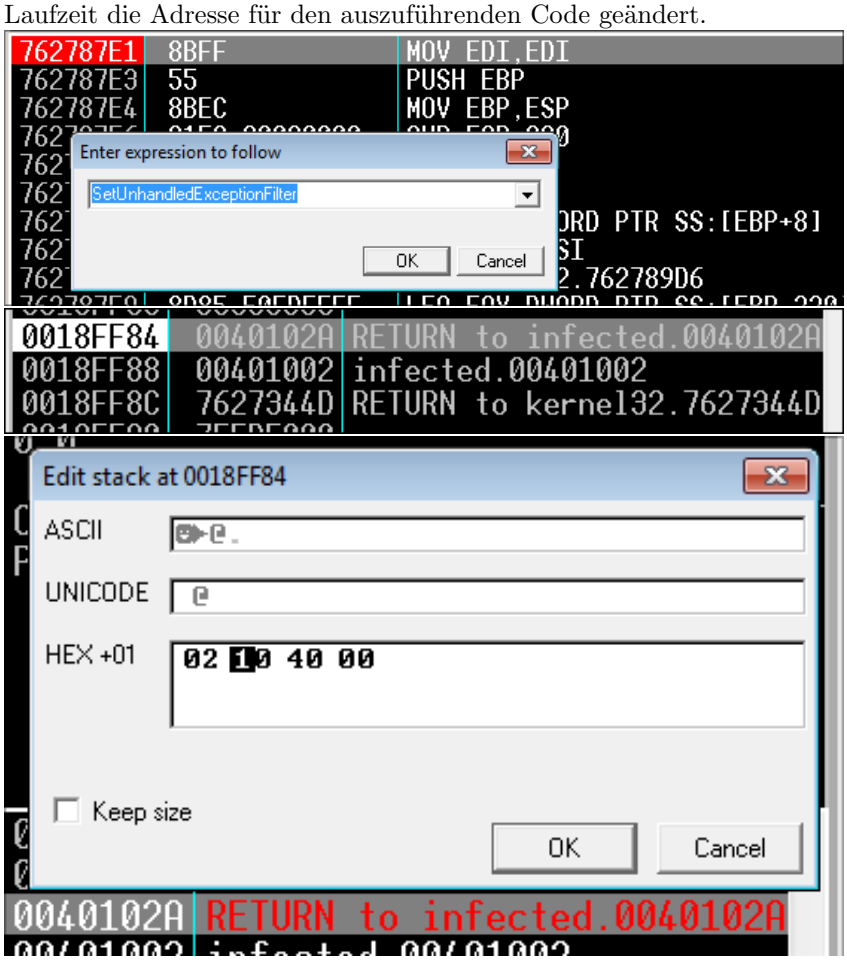
NOTE: Die Verwendung von "Ctrl+G" ist hierbei sehr hilfreich um Breakpoints an den einzelnen Funktionen zu setzen.

Assembler

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

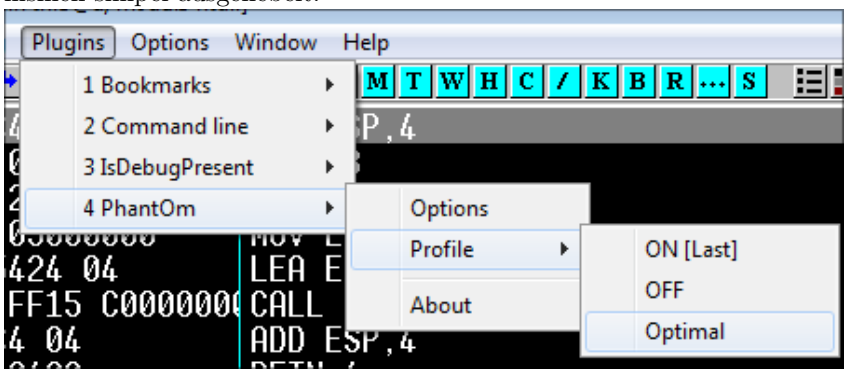


Für den Mechanismus "`Unhandled Exception`" wurde wieder ein Breakpoint gesetzt und während der Laufzeit die Adresse für den auszuführenden Code geändert.



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: