



# «Санкт-Петербургский политехнический университет Петра Великого»

## Институт компьютерных наук и технологий **Высшая школа киберфизических систем и управления**

### **Разработка и применение методов защиты программы от отладчика**

Направление 27.03.04 – Управление в технических системах  
Профиль 27.03.04 \_05 – Интеллектуальные системы  
обработки информации и управления

Выполнил

студент гр. 3532704/90501  
Шкалин Кирилл Павлович

Научный руководитель

доцент ВШ КФСУ  
Сальников Вячеслав Юрьевич



# Цель и задачи

## Цель работы:

разработка метода обеспечения защиты программы от взлома отладчиком. Метод должен как обеспечивать обнаружение факта изменения кода программы отладчиком, так и препятствовать самому процессу отладки

## Задачи:

- 1) Изучить принцип действия отладчика;
- 2) Рассмотреть известные методы защиты от отладчика;
- 3) Разработать алгоритм защиты от отладчика;
- 4) Реализовать полученный алгоритм, обеспечив при этом достаточный уровень скрытности;
- 5) Провести тестирование полученной системы защиты.

## Рассмотренные методы:

1. Замер времени выполнения;
2. Возможности предоставляемые ОС;
3. Поиск отладчика среди запущенных в системе процессов;
4. Подсчет контрольной суммы критической секции.

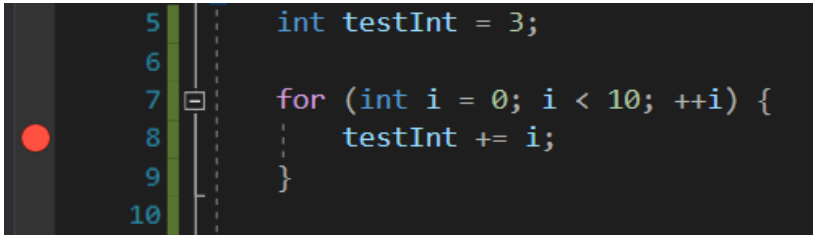


# Алгоритм работы отладчика

```
CreateProcess("FileName.exe", ..., DEBUG_PROCESS, ...);  
for (;;) {  
    WaitForDebugEvent(&dbgEv, INFINITE);  
    switch(dbgEv.dwDebugEventCode)  
    {  
        case EXCEPTION_DEBUG_EVENT:  
            ...  
    }  
    ContinueDebugEvent( dbgEv.dwProcessId,  
                        dbgEv.dwThreadId,  
                        dwContinueStatus );  
}
```

Общая схема работы отладчика

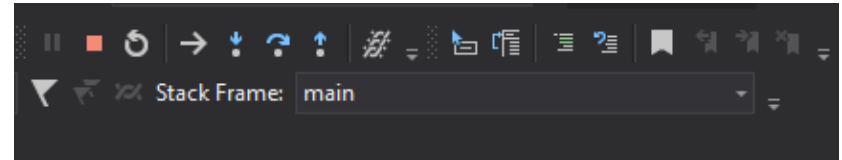
- Точка останова



Точка останова — место в коде программы, дойдя до которого, процессор должен прервать выполнение программы и передать управление отладчику.

Чтобы поставить точку останова отладчик должен заменить один байт в коде программы на инструкцию с кодом `0xCC`.

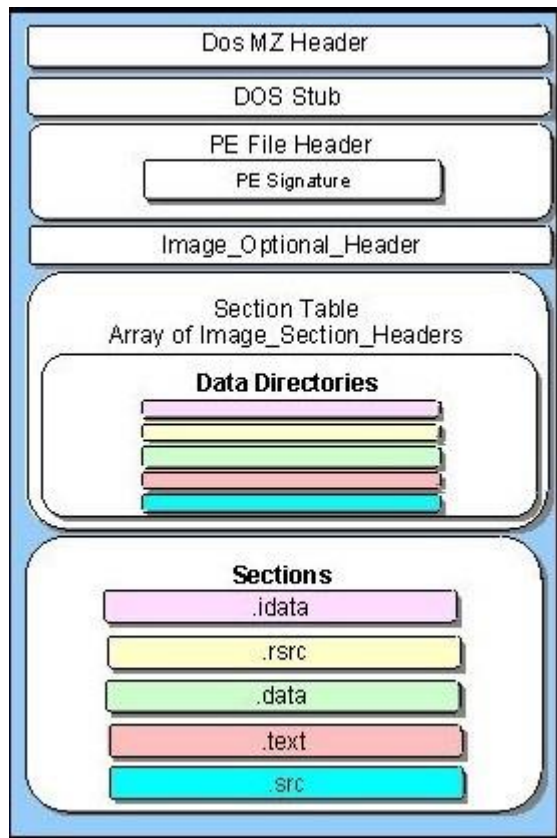
- Трассирование



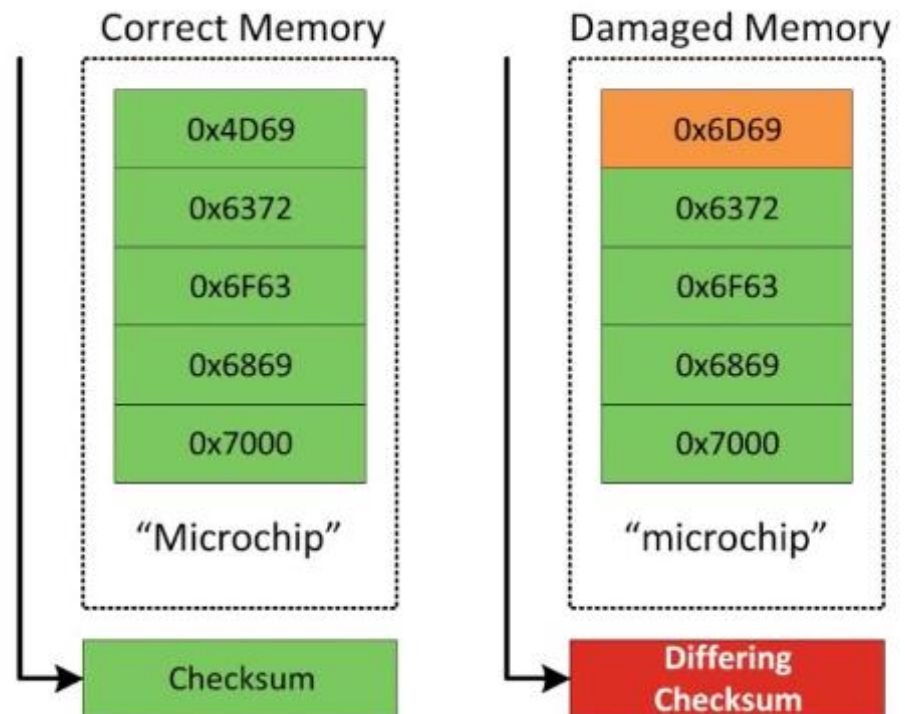
Трассировка — последовательное выполнение программы, при котором после каждой инструкции управление передается отладчику. В этом режиме программист может детально отследить изменения значений всех параметров процесса. Обеспечение режима пошагового выполнения программы предусмотрено на аппаратном уровне.

# Данные необходимые для организации защиты

**PE-формат** – формат исполняемых файлов, используемый в 32- и 64-разрядных версиях ОС Windows

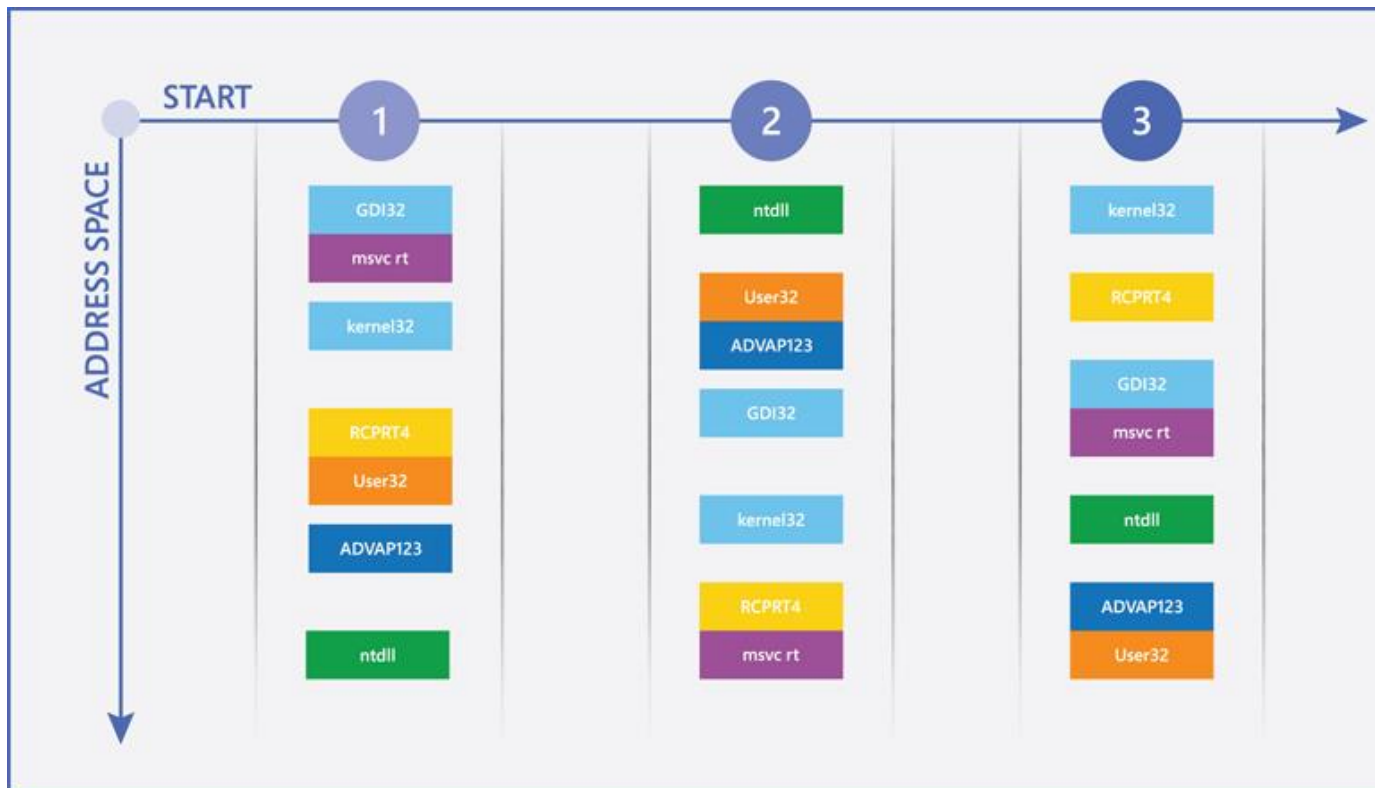


**Циклический избыточный код** (англ. Cyclic redundancy check, CRC) — алгоритм нахождения контрольной суммы, предназначенный для проверки целостности данных.



# Проблема вызванная ASLR

**ASLR** (address space load randomization) – это метод компьютерной безопасности, предназначенный для предотвращения использования уязвимостей, связанных с повреждением памяти. ASLR случайным образом упорядочивает позиции адресного пространства ключевых областей данных процесса, включая базовый адрес загрузки, позиции стека, кучи и загружаемых библиотек.



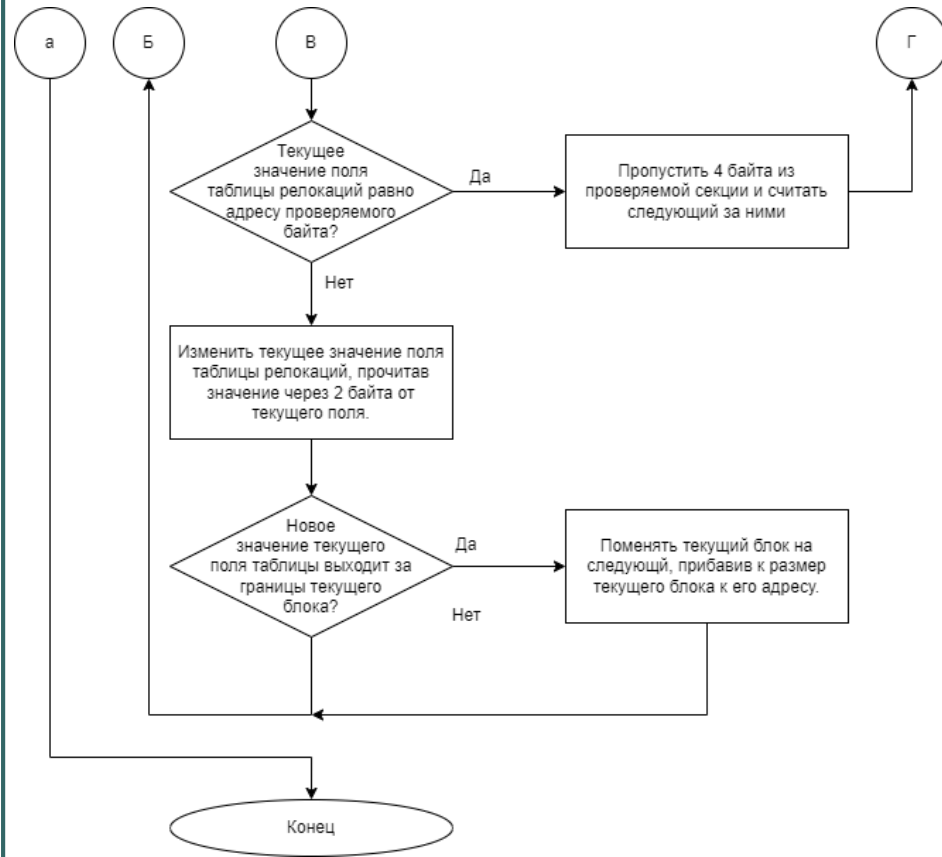
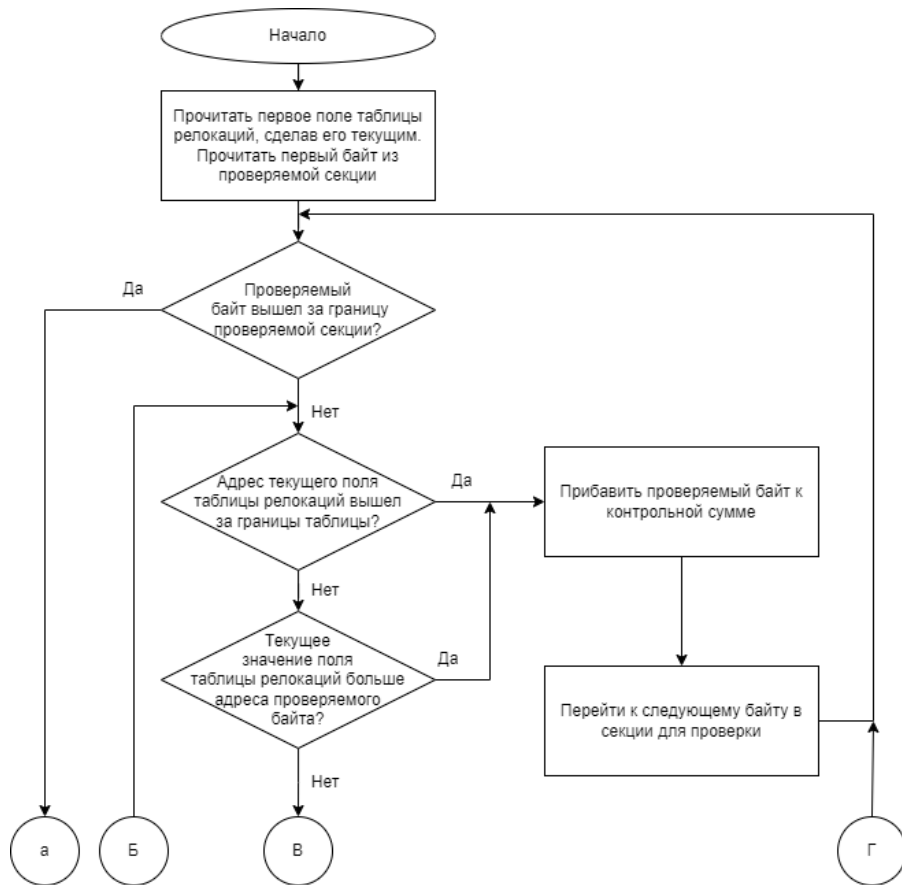
# Таблица базовых релокаций

Структура блока исправлений:

| Смещение | Размер | Поле                                     | Значение  |
|----------|--------|--|---|
| 0        | 4      | Относительный виртуальный адрес страницы | Базовый адрес загрузки и относительный виртуальный адрес страницы прибавляется к каждому смещению в таблице, чтобы получить виртуальный адрес по которому необходимо провести исправление |
| 4        | 4      | Размер блока исправлений                 | Общее количество байтов, занимаемых блоком исправлений, включая эту структуру.  |

Структура поля таблицы базовых релокаций:

| Смещение | Размер  | Поле     | Значение  |
|----------|---------|----------|---|
| 0        | 4 бита  | Тип      | Значение, размещенное в старших четырех битах слова, указывает на тип исправления. Всего типов исправлений может быть 16. Каждый тип указывает, как провести коррекцию значения в памяти.               |
| 0        | 12 бита | Смещение | Значение, размещенное в младших двенадцати битах слова, указывает смещение относительно относительного виртуального адреса страницы. Это смещение указывает место, где необходимо произвести коррекцию. |







# Реализация алгоритма

## Листинг макроса для нахождения контрольной суммы

```
#define GET_CRC(reg_A, reg_B, reg_C, reg_D, reg_SI, reg_DI, out_var) __asm \
{ \
    __asm mov e##reg_A##x, hInst /* base load address in eax ## */ \
    __asm mov e##reg_B##x, e##reg_A##x \
    __asm add e##reg_B##x, 60 /* sixty is _lfanew offset */ \
    __asm mov e##reg_B##x, [e##reg_B##x] /* _lfanew value is in ebx */ \
    __asm add e##reg_A##x, e##reg_B##x /* pe header is in eax */ \
    __asm xor e##reg_C##x, e##reg_C##x \
    __asm xor e##reg_SI, e##reg_SI \
    __asm mov reg_C##x, [e##reg_A##x + 6] /* number of sections is in ecx */ \
    __asm mov reg_SI, [e##reg_A##x + 20] /* Size of optional header is in esi */ \
    __asm add e##reg_A##x, 24 /* Optional pe header is in eax */ \
    __asm mov e##reg_B##x, e##reg_A##x \
    __asm add e##reg_B##x, 96 /* DataDirectory is in ebx */ \
    __asm add e##reg_B##x, 5 * 8 /* Base relocation table field is in ebx */ \
    __asm push[e##reg_B##x] /* Add reloc rva in stack */ \
    __asm push[e##reg_B##x + 4] /* Add reloc size in stack */ \
    __asm add e##reg_A##x, e##reg_SI /* Start of section table in eax */ \
    __asm mov e##reg_SI, 1 \
    __asm mov e##reg_D##x, e##reg_A##x \
} \
sections_loop: \
__asm { \
    __asm mov e##reg_B##x, [e##reg_D##x + 36] /* Characteristics of section is in ebx */ \
    __asm and e##reg_B##x, 0x20 /* if ebx is not zero, then the section is being executed */ \
    __asm jnz section_found \
    __asm add e##reg_D##x, 40 /* in edx next section table */ \
    __asm inc e##reg_SI /* in esi number off current section */ \
    __asm cmp e##reg_SI, e##reg_C##x /* if esi and ecx are equal, then there isn't code section */ \
    __asm je no_section \
    __asm jmp sections_loop \
} \
section_found: \
__asm { \
    /* In edx code section table */ \
    __asm push [e##reg_D##x + 12] /* Add code section rva in stack */ \
    __asm push [e##reg_D##x + 8] /* Add code section rva in stack */ \
    /*===== Main algorithm =====*/ \
    __asm mov e##reg_DI, [esp + 12] /* reloc rva is in edi */ \
    __asm add e##reg_DI, hInst /* reloc address is in edi. So edi contains current reloc block */ \
    __asm mov e##reg_SI, e##reg_DI /* esi contains current block */ \
    __asm add e##reg_SI, 8 /* esi contains current rf */ \
    __asm xor e##reg_C##x, e##reg_C##x /* In ecx will be iterator for main loop */ \
    __asm xor e##reg_D##x, e##reg_D##x /* In edx will be checksum */ \
} \
main_loop: \
    __asm { \
        __asm cmp e##reg_C##x, [esp] /* Check is iterator less than code section size */ \
        __asm jge main_loop_end \
    } \
    find_reloc_loop: \
    __asm { \
        /* Check is current rf in relocation table */ \
        __asm mov e##reg_B##x, [esp + 12] /* Reloc section rva in edx */ \
        __asm add e##reg_B##x, [esp + 8] /* rva of end address reloc table is in edx */ \
        \
        __asm add e##reg_B##x, hInst /* end address reloc table is in edx */ \
        __asm cmp e##reg_SI, e##reg_B##x \
        __asm jge end_reloc_loop \
        /* Check equals current rf checked byte */ \
        __asm mov reg_A##x, [e##reg_SI] /* Place 2 bytes reloc in eax */ \
        __asm and e##reg_A##x, 0xFF /* Remove type of relocation */ \
        __asm add e##reg_A##x, [e##reg_DI] /* Add section rva to value from reloc field */ \
        Reloc address is in eax \
        __asm mov e##reg_B##x, [esp + 4] /* code section rva in ebx */ \
        __asm add e##reg_B##x, e##reg_C##x /* Current checked byte rva in ebx */ \
        __asm cmp e##reg_A##x, e##reg_B##x /* Compare checked byte and current relocation */ \
        __asm jg checked_is_not_in_reloc \
        __asm je checked_in_reloc \
        /* Check is current rf in current block */ \
        __asm mov e##reg_B##x, e##reg_DI /* reloc block is in edx */ \
        __asm add e##reg_B##x, [e##reg_B##x + 4] /* end address of relocation block is in edx */ \
        __asm cmp e##reg_SI, e##reg_B##x /* compare current rf with end address of relocation block */ \
        __asm jl in_old_block \
        __asm add e##reg_DI, [e##reg_DI + 4] /* Switch to next reloc block */ \
        __asm lea e##reg_SI, [e##reg_DI + 6] /* Current rf in new block */ \
    } \
    in_old_block: \
    __asm { \
        __asm add e##reg_SI, 2 /* Next rf */ \
        __asm jmp find_reloc_loop \
    } \
    checked_is_not_in_reloc: \
    __asm { \
        __asm xor e##reg_A##x, e##reg_A##x /* if checked byte isn't in reloc, then eax contains 0 */ \
        __asm jmp end_reloc_loop \
    } \
    checked_in_reloc: \
    __asm { \
        __asm mov e##reg_A##x, 1 /* if checked byte is in reloc, then eax contains 1 */ \
        __asm jmp end_reloc_loop /* TODO delete this instruction */ \
    } \
} \
end_reloc_loop: \
__asm { \
    __asm test e##reg_A##x, e##reg_A##x \
    __asm jnz skip_byte \
    __asm mov e##reg_B##x, hInst /* Place hInst in */ \
    __asm add e##reg_B##x, [esp + 4] /* Add code section rva */ \
    __asm add e##reg_B##x, e##reg_C##x /* Add iterator of checked byte */ \
    __asm xor e##reg_A##x, e##reg_A##x \
    __asm mov reg_A##1, byte ptr [e##reg_B##x] /* I */ \
    code section for adding in crc \
    __asm add e##reg_D##x, e##reg_A##x /* Add value */ \
    __asm inc e##reg_C##x /* Increment iterator */ \
    __asm jmp main_loop \
} \
skip_byte: \
__asm { \
    __asm add e##reg_C##x, 4 \
    __asm jmp main_loop \
} \
main_loop_end: \
__asm { \
    __asm mov out_var, e##reg_D##x \
    __asm add esp, 2*4 \
} \
no_section: \
__asm { \
    __asm add esp, 2*4 /* clear stack */ \
} \
}
```



# Модули для работы с секциями PE-файла

В ходе работы были реализованы модули, обеспечивающие нахождение контрольной суммы различных секций программ, загруженных в память и расположенных на диске.

## Пример использования реализованных классов

```
1 #include <iostream>
2 #include "CRC_exe.h"
3 #include "CRC_image.h"
4
5 static volatile uint32_t CRC = 0x4C52454B;
6
7 int main(int argc, char *argv[])
8 {
9     std::string path(argv[0]);
10    CRC_exe finder;
11    CRC_image image_finder;
12    finder.read_file(path);
13    uint32_t crc = finder.get_sections_CRC(IMAGE_SCN_CNT_CODE);
14    uint32_t crc_in_loaded = image_finder.get_sections_CRC(IMAGE_SCN_CNT_CODE);
15    std::cout << "This exe contains key: " << std::boolalpha << finder.any_key(CRC) << std::endl;
16    finder.replace_all_key(CRC, crc);
17    finder.write_file(std::string("Cracked") + argv[0]);
18    printf("CRC in file: %08X\n", crc);
19    printf("CRC in memory: %08X\n", crc_in_loaded);
20    // ...
44 }
```

### Microsoft Visual Studio Debug Console

This exe contains key: true

CRC in file: 001D2EAF

CRC in memory: 001D2EAF

H:\work\C++\CRC\_class\Release\CRC\_class.exe (process 13952) exited with code 0.

To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.

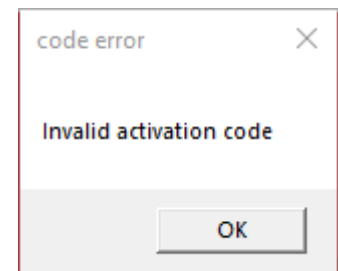
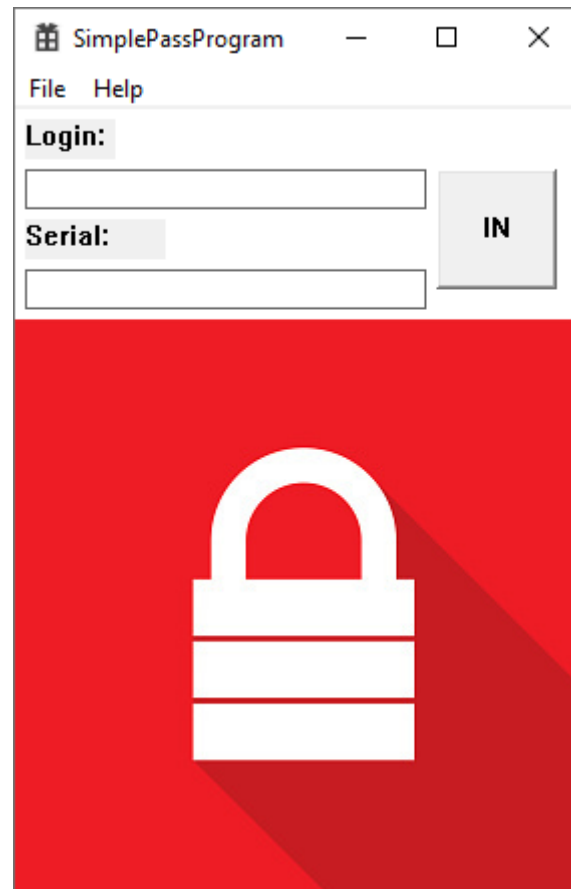
Press any key to close this window . . .



# Демонстрационная программа

POLYTECH

Для проведения тестов была написана демонстрационная программа на языке C с использованием только WinAPI. В случае, если введенный пользователем ключ не совпадает с тем, который сгенерировала программа, выводится окно сообщения, уведомляющее пользователя о неудаче.



# Взлом демонстрационной программы

OlyDbg - With\_Protection.exe - [CPU - main thread, module With\_Protection]

File View Debug Trace Plugins Options Windows Help

008B1409 - 8DCE E8 LEA EDI, [EBP-18]  
008B140C - 8995 30FFFFFF MOV DWORD PTR SS:[EBP-0D0], EDI  
008B1412 - 8D45 D4 LEA EAX, [EBP-2C]  
008B1415 - 8985 34FFFFFF MOV DWORD PTR SS:[EBP-0CC], EAX  
008B1418 - 88D0 34FFFFFF MOV ECK, DWORD PTR SS:[EBP-0CC]  
008B1421 - 8A11 MOV DL, BYTE PTR DS:[ECX]  
008B1423 - 8895 47FFFFFF MOV BYTE PTR SS:[EBP-0B9], DL  
008B1429 - 8885 30FFFFFF MOV EAX, DWORD PTR SS:[EBP-0D0]  
008B142F - 3B10 CMP DL, BYTE PTR DS:[EAX]  
008B1431 - 75 46 JNE SHORT 008B1479  
008B1433 - 80BD 47FFFFFF 00 CMP BYTE PTR SS:[EBP-0B9], 0  
008B1438 - 74 34 JE SHORT 008B146D  
008B143C - 88BD 34FFFFFF MOV ECK, DWORD PTR SS:[EBP-0CC]  
008B1442 - 8A51 01 MOV DL, BYTE PTR DS:[ECX+1]  
008B1445 - 8895 46FFFFFF MOV BYTE PTR SS:[EBP-0BA], DL  
008B1448 - 8885 30FFFFFF MOV EAX, DWORD PTR SS:[EBP-0D0]  
008B1451 - 3A50 01 CMP DL, BYTE PTR DS:[EAX+1]  
008B1454 - 75 23 JNE SHORT 008B1479  
008B1456 - 8885 34FFFFFF 02 ADD DWORD PTR SS:[EBP-0CC], 2  
008B145B - 8385 30FFFFFF 02 ADD DWORD PTR SS:[EBP-0D0], 2  
008B1464 - 80BD 46FFFFFF 00 CMP BYTE PTR SS:[EBP-0BA], 0  
008B146B - 75 2E JNE SHORT 008B141B  
008B146D - 8B50 MOV DWORD PTR SS:[EBP-0D4], 0  
008B1472 - EB 0B JMP SHORT 008B1484  
008B1479 - 1BC9 SBB ECK, ECK  
008B147B - 83C9 01 OR ECK, 00000001  
008B147E - 898D 2CFFFFFF MOV DWORD PTR SS:[EBP-0D4], ECK  
008B1484 - 8B95 2CFFFFFF MOV EDI, DWORD PTR SS:[EBP-0D4]  
008B148A - 8995 20FFFFFF MOV DWORD PTR SS:[EBP-0E0], EDI  
008B1490 - 83B5 20FFFFFF 00 CMP DWORD PTR SS:[EBP-0E0], 0  
008B1497 - 75 1F JNE SHORT 008B14B8  
008B1499 - 6A 05 PUSH 5  
008B149B - A1 90548B00 MOV EAX, DWORD PTR DS:[hStUnLock]  
008B14A0 - 50 PUSH EAX  
008B14A1 - FF15 80308B00 CALL DWORD PTR DS:[<USER32.ShowWindow>]  
008B14A7 - 6A 00 PUSH 0  
008B14A9 - 8B0D 98548B00 MOV ECK, DWORD PTR DS:[hStLock]  
008B14AF - 51 PUSH ECK  
008B14B0 - FF15 80308B00 CALL DWORD PTR DS:[<USER32.ShowWindow>]  
008B14B6 - EB 33 JMP SHORT 008B14EB  
008B14B8 - 6A 05 PUSH 5  
008B14C0 - 8B15 98548B00 MOV EDI, DWORD PTR DS:[hStLock]  
008B14C7 - 52 PUSH EDI  
008B14C9 - FF15 80308B00 CALL DWORD PTR DS:[<USER32.ShowWindow>]  
008B14D7 - 6A 00 PUSH 0  
008B14D9 - A1 90548B00 MOV EAX, DWORD PTR DS:[hStUnLock]  
008B14DE - 50 PUSH EAX  
008B14E0 - FF15 80308B00 CALL DWORD PTR DS:[<USER32.ShowWindow>]  
008B14E6 - 6A 00 PUSH 0  
008B14E8 - 68 38328B00 PUSH OFFSEI 008B3238  
008B14ED - 68 74328B00 PUSH OFFSEI 008B3274  
008B14F1 - 8B4D 08 MOV ECK, DWORD PTR SS:[EBP+8]  
008B14F4 - 51 PUSH ECK  
008B14F6 - FF15 90308B00 CALL DWORD PTR DS:[<USER32.MessageBox>]  
008B14FB - EB 1B JMP SHORT 008B1508  
008B14FD - 8B55 14 MOV EDI, DWORD PTR SS:[EBP+14]  
008B14F0 - 52 PUSH EDI  
008B14F2 - 8B45 10 MOV EAX, DWORD PTR SS:[EBP+10]  
008B14F4 - 50 PUSH EAX

inline strcmp

Out from strcmp, if strings are equal

Entering an error flag in ecx

ecx contains 0 if strings are equal and 1 else

Checking for correct password

hWnd => [8B5490] = NULL

Show = SW\_HIDE

hWnd => [8B5498] = NULL

Show = SW\_SHOW

hWnd => [8B5498] = NULL

Show = SW\_HIDE

hWnd => [8B5490] = NULL

USER32.ShowWindow

Type = MB\_OK|MB\_DEFBUTTON1|MB\_APPLMODAL

Caption = "code error"

Text = "Invalid activation code"

hOwner

USER32.MessageBox

Registers (FPU)

EAX 003FFC4C  
ECX 008B1E08 With\_Protection.<ModuleEntryPoint>  
EDX 008B1E08 With\_Protection.<ModuleEntryPoint>  
EBX 0045F000  
ESP 003FF0F4  
EBP 003FF000  
ESI 008B1E08 With\_Protection.<ModuleEntryPoint>  
EDI 008B1E08 With\_Protection.<ModuleEntryPoint>  
EIP 008B1E08 With\_Protection.<ModuleEntryPoint>  
C 0 ES 002B 32bit 0 (FFFFFFFF)  
P 1 CS 0023 32bit 0 (FFFFFFFF)  
D 0 SS 002B 32bit 0 (FFFFFFFF)  
Z 1 DS 002B 32bit 0 (FFFFFFFF)  
S 0 FS 0053 32bit 462000 (FFFF)  
T 0 GS 002B 32bit 0 (FFFFFFFF)  
0 0  
0 0 LastErr 00003687 ERROR\_SXS\_KEY\_NOT\_FOUND  
EFL 00000246 (NO, MB, E, BE, WS, PE, GE, IE)  
ST0 empty 0.0  
ST1 empty 0.0  
ST2 empty 0.0  
ST3 empty 0.0  
ST4 empty 0.0  
ST5 empty 0.0  
ST6 empty 0.0  
ST7 empty 0.0  
FST 0000 Cond 0 0 0 0 Err 0 0 0 0 0 0 0 (ST)  
FCW 027F Prec NEAR, S3 Hask 1 1 1 1 1  
Last cmd 0000:00000000  
XMM0 00000000 00000000 00000000 00000000  
XMM1 00000000 00000000 00000000 00000000  
XMM2 00000000 00000000 00000000 00000000  
XMM3 00000000 00000000 00000000 00000000  
XMM4 00000000 00000000 00000000 00000000  
XMM5 00000000 00000000 00000000 00000000  
XMM6 00000000 00000000 00000000 00000000  
XMM7 00000000 00000000 00000000 00000000  
P 0 0 0 0 0 0  
XCSR 00000000 FZ 0 0 0 0 Err 0 0 0 0 0  
0  
Rnd NEAR Hask 1 1 1 1 1

Address Hex dump ASCII (ANSI) - w

008B5010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
008B5018 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
008B5020 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
008B5030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
008B5040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
008B5050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
008B5060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
008B5070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
008B5080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
008B5090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
008B50A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
008B50B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
008B50C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
008B50D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
008B50E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
008B50F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
008B5100 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

003FF0F4 750000C9 JZ RETURN to KERNEL32.BaseThreadInitThunk+19  
003FF0FC 75000080 JZ KERNEL32.BaseThreadInitThunk  
003FF0FD 003FF0C0 JZ RETURN to ntdll.7724784E  
003FF0FE 7724784E RETN  
003FF0FF 26626FCB RETN  
003FF100 00000000  
003FF104 00000000  
003FF108 0045F000 EE  
003FF10C 00000000  
003FF110 00000000  
003FF114 00000000  
003FF118 00000000  
003FF11C 00000000  
003FF120 00000000  
003FF124 00000000  
003FF128 00000000  
003FF12C 00000000  
003FF130 00000000  
003FF134 00000000  
003FF138 00000000  
003FF13C 00000000  
003FF140 00000000

Entry point of main module

Paused

Окно отладчика OllyDBG с загруженной демонстрационной программой



POLYTECH

# Код защиты

OllyDbg - With\_Protection.exe - [CPU - main thread, module With\_Protection]

File View Debug Trace Plugins Options Windows Help

Registers (FPU)

EAX 00DAFC78  
ECX 00561E08 With\_Protection.<ModuleEntryPoint>  
EDX 00561E08 With\_Protection.<ModuleEntryPoint>  
EBX 00F1A000  
ESP 00DAFC20  
EBP 00DAFC2C  
ESI 00561E08 With\_Protection.<ModuleEntryPoint>  
EDI 00561E08 With\_Protection.<ModuleEntryPoint>  
EIP 00561E08 With\_Protection.<ModuleEntryPoint>

C 0 ES 002B 32bit 0(FFFFFFFF)  
P 1 CS 0023 32bit 0(FFFFFFFF)  
A 0 SS 002B 32bit 0(FFFFFFFF)  
Z 1 DS 002B 32bit 0(FFFFFFFF)  
S 0 FS 0053 32bit FID000(FFF)  
T 0 GS 002B 32bit 0(FFFFFFFF)  
D 0  
O 0 LastErr 000036B7 ERROR\_SXS\_KEY\_NOT\_FOUND  
EFL 00000246 (NO.WB.E.BE.NS.PE.GE.LE)

ST0 empty 0.0  
ST1 empty 0.0  
ST2 empty 0.0  
ST3 empty 0.0  
ST4 empty 0.0  
ST5 empty 0.0  
ST6 empty 0.0  
ST7 empty 0.0

3 2 1 0 ESP U O Z D I  
FST 0000 Cond 0 0 0 0 Err 0 0 0 0 0 0 0 (GT)  
FCW 027F Prec NEAR.53 Mask 1 1 1 1 1 1  
Last cwnd 0000:00000000

MMX0 00000000 00000000 00000000 00000000  
MMX1 00000000 00000000 00000000 00000000  
MMX2 00000000 00000000 00000000 00000000  
MMX3 00000000 00000000 00000000 00000000  
MMX4 00000000 00000000 00000000 00000000  
MMX5 00000000 00000000 00000000 00000000  
MMX6 00000000 00000000 00000000 00000000  
MMX7 00000000 00000000 00000000 00000000

P U O Z D I  
MXCSR 00001F80 FZ 0 DZ 0 Err 0 0 0 0 0  
Rnd NEAR Mask 1 1 1 1 1 1

Address Hex dump ASCII (ANSI - кириллица)

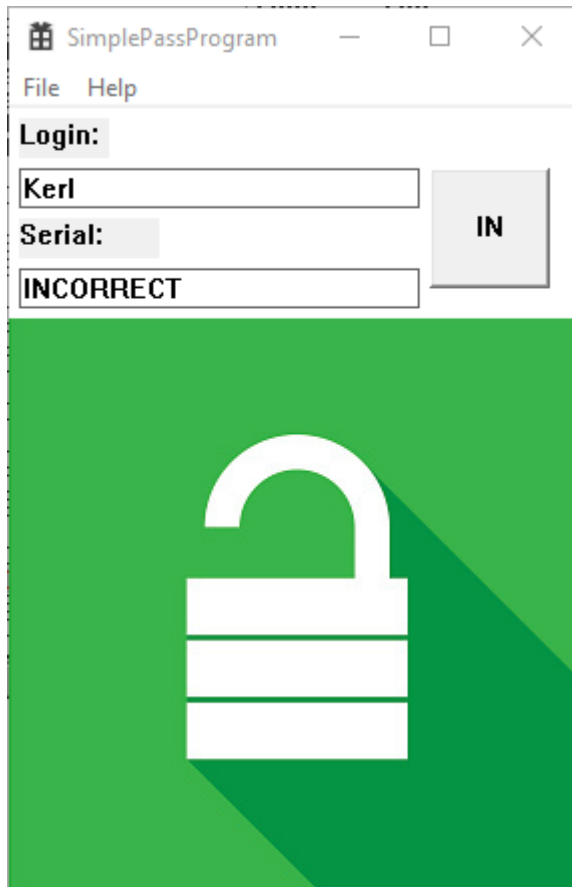
00DAFC20 777900C9 yw RETURN to KERNEL32.BaseThreadInitThunk+19  
00DAFC24 00F1A000 c KERNEL32.BaseThreadInitThunk  
00DAFC28 777900B0 yw  
00DAFC2C 00DAFCB8 tsb  
00DAFC30 77797B4E H(w  
00DAFC34 00F1A000 c  
00DAFC38 F24089D8 WZ@m  
00DAFC3C 00000000  
00DAFC40 00000000  
00DAFC44 00F1A000 c  
00DAFC48 00000000  
00DAFC4C 00000000  
00DAFC50 00000000

Entry point of main module

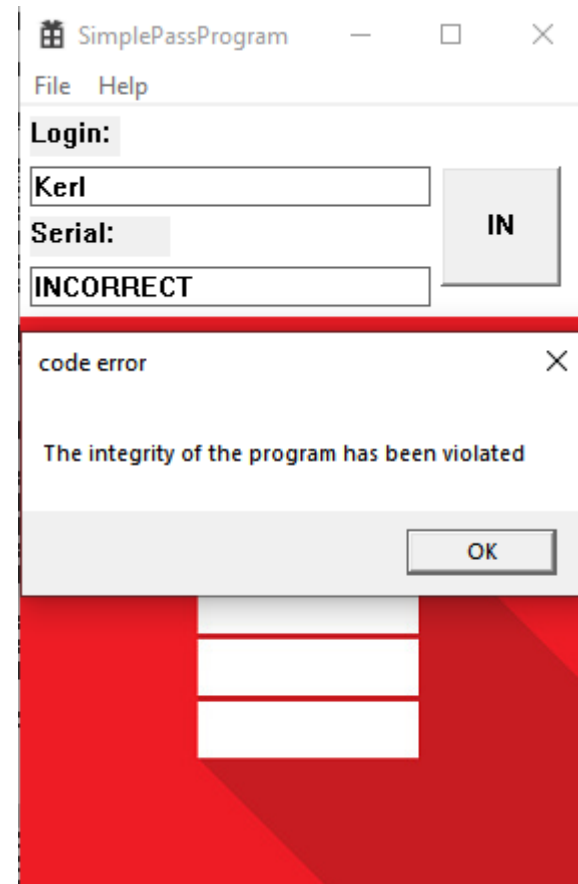
Окно отладчика OllyDBG, в котором открыт участок с кодом защиты

# Сравнение работы программы с защитой и без

## Программа без защиты



## Программа с защитой



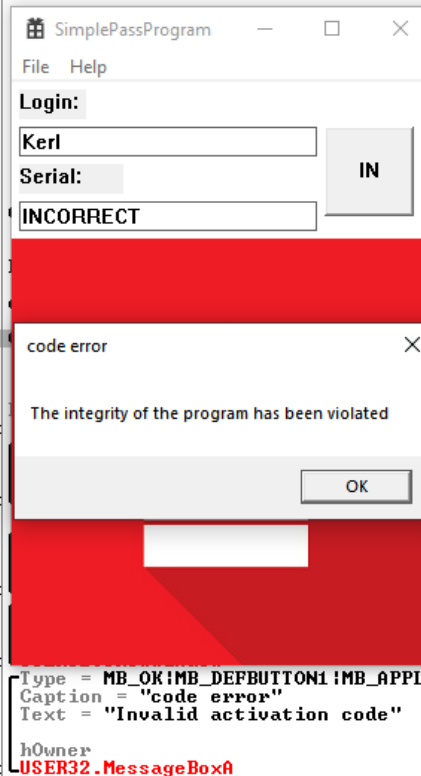


POLYTECH

# Детектирование точки останова

```
008B1409 . 8D55 E8 LEA EDX,[EBP-18]
008B140C . 8995 30FFFFFF MOV DWORD PTR SS:[EBP-0D0],EDX
008B1412 . 8D45 D4 LEA EAX,[EBP-2C]
008B1415 . 8985 34FFFFFF MOV DWORD PTR SS:[EBP-0CC],EAX
008B141B > 8B8D 34FFFFFF MOV ECX,DWORD PTR SS:[EBP-0CC]
008B1421 . 8A11 MOV DL, BYTE PTR DS:[ECX]
008B1423 . 8895 47FFFFFF MOV BYTE PTR SS:[EBP-0B9],DL
008B1429 . 8B85 30FFFFFF MOV EAX,DWORD PTR SS:[EBP-0D0]
008B142F . 3A10 CMP DL, BYTE PTR DS:[EAX]
008B1431 . 75 46 JNE SHORT 008B1479
008B1433 . 80BD 47FFFFFF CMP BYTE PTR SS:[EBP-0B9],0
008B143A . 74 31 JE SHORT 008B146D
008B143C . 8B8D 34FFFFFF MOV ECX,DWORD PTR SS:[EBP-0CC]
008B1442 . 8A51 01 MOV DL, BYTE PTR DS:[ECX+1]
008B1445 . 8895 46FFFFFF MOV BYTE PTR SS:[EBP-0BA],DL
008B144B . 8B85 30FFFFFF MOV EAX,DWORD PTR SS:[EBP-0D0]
008B1451 . 3A50 01 CMP DL, BYTE PTR DS:[EAX+1]
008B1454 . 75 23 JNE SHORT 008B1479
008B1456 . 8385 34FFFFFF ADD DWORD PTR SS:[EBP-0CC],2
008B145D . 8385 30FFFFFF ADD DWORD PTR SS:[EBP-0D0],2
008B1464 . 80BD 46FFFFFF CMP BYTE PTR SS:[EBP-0BA],0
008B146B . 75 AE JNE SHORT 008B141B
008B146D . C785 2CFFFFFF MOV DWORD PTR SS:[EBP-0D4],0
008B1477 . EB 0B JMP SHORT 008B1484
008B1479 > 1BC9 SBB ECX,ECX
008B147B . 83C9 01 OR ECX,00000001
008B147E . 898D 2CFFFFFF MOV DWORD PTR SS:[EBP-0D4],ECX
008B1484 > 8B95 2CFFFFFF MOV EDX,DWORD PTR SS:[EBP-0D4]
008B148A . 8995 20FFFFFF MOV DWORD PTR SS:[EBP-0E0],EDX
008B1490 > 83BD 20FFFFFF CMP DWORD PTR SS:[EBP-0E0],0
008B1497 . 75 1F JNE SHORT 008B14B8
008B1499 . 6A 05 PUSH 5
008B149B . A1 90548B00 MOV EAX,DWORD PTR DS:[hStUnlock]
008B14A0 . 50 MOV EAX
008B14A1 . FF15 80308B00 CALL DWORD PTR DS:[&USER32.ShowWindowEx]
008B14A7 . 6A 00 PUSH 0
008B14A9 . 8B0D 98548B00 MOV ECX,DWORD PTR DS:[hStLock]
008B14AF . 51 MOV ECX
008B14B0 . FF15 80308B00 CALL DWORD PTR DS:[&USER32.ShowWindowEx]
008B14B6 . EB 33 JMP SHORT 008B14EB
008B14B8 > 6A 05 PUSH 5
008B14BA . 8B15 98548B00 MOV EDX,DWORD PTR DS:[hStLock]
008B14C0 . 52 MOV EDX
008B14C1 . FF15 80308B00 CALL DWORD PTR DS:[&USER32.ShowWindowEx]
008B14C7 . 6A 00 PUSH 0
008B14C9 . A1 90548B00 MOV EAX,DWORD PTR DS:[hStUnlock]
008B14CE . 50 MOV EAX
008B14CF . FF15 80308B00 CALL DWORD PTR DS:[&USER32.ShowWindowEx]
008B14D5 . 6A 00 PUSH 0
008B14D7 . 68 38328B00 PUSH OFFSETOFFSET 008B3238
008B14DC . 68 74328B00 PUSH OFFSETOFFSET 008B3274
008B14E1 . 8B4D 08 MOV ECX,DWORD PTR SS:[EBP+8]
008B14E4 . 51 MOV ECX
008B14E5 . FF15 90308B00 CALL DWORD PTR DS:[&USER32.MessageBoxA]
008B14EB > EB 1B JMP SHORT 008B1508
008B14ED > 8B55 14 MOV EDX,DWORD PTR SS:[EBP+14]
008B14F0 . 52 MOV EDX
008B14F1 . 8B45 10 MOV EAX,DWORD PTR SS:[EBP+10]
008B14F4 . 50 MOV EAX
```

inline strcmp



Отлаживаемый процесс обнаружил установленную точки останова

В рамках ВКР был разработан метод защиты программного обеспечения от отладчика.

В ходе выполнения были выполнены следующие задачи:

- 1) Изучен принцип работы отладчика;
- 2) Разработан алгоритм защиты от отладчика, основанный на проверке контрольной суммы;
- 3) Полученный алгоритм реализован на языке ассемблера с достаточным уровнем скрытности;
- 4) Написаны программные модули на языке C++ для работы с секциями PE-файла;
- 5) Полученная система защиты была протестирована.

Результаты тестирования показали, что разработанный метод обеспечивает защиту на достойном уровне