PV204 Laboratory of security and applied cryptography II

Lecture: Reverse code engineering

Powerfull knowledge, lot of fun and legal for several purposes!
Basic information available in Wikipedia article on reverse engineering (IS copy REWiki.pdf)

- RE in general, example with B-29, synthetic chemistry/biology
- Legality
  - Own binary without documentation
  - Interoperability
  - Anti-virus research
  - Fair use, education
  - Problem with recent copyright laws (attempt to circumvent is illegal, not only selling circumvented content)
  - Forensics
- Disassembler vs. debugger
  - Static vs. dynamic code analysis
  - Debugger vs. Debugger with advanced modification tools (Visual Studio vs. OllyDbg)
- Assembler vs. bytecode
  - Instruction set (downloadable)
  - Register-based vs. stack-based execution
- Structured code vs. sequence of executed instructions
  - Structured code contains code for all branches (runable binary)
  - Sequence of executed instructions only from branches taken (power analysis of smart card)
- Example: Java Card bytecode
  - sspush, sspush, add, ifeq
- Example: Win32 binary
  - Lena tutorials 1 and 2
  - Name of the registers (EAX 32bit, AX 16bit, AH/AL 8bit)
  - Flags (Zero/Sign/Carry)

Java(Card) bytecode

Intermediate code interpreted by virtual machine (see JavaCard222_ops.pdf).

- Usually easier to understand then assembler code.
- Stack-based oriented execution, no registers are used (all operands at the top of the stack).
- Operation takes its operands from stack and return result there.
- JavaCard example selected because of lower number of opcodes.
- Same principle works for Java, .NET CLI ...

```
 // ENCRYPT INCOMING BUFFER
  void Encrypt(APDU apdu) {
      byte[]    apdubuf = apdu.getBuffer();
      short     dataLen = apdu.setIncomingAndReceive();
      short     i;

      // CHECK EXPECTED LENGTH (MULTIPLY OF 64 bites)
      if ((dataLen % 8) != 0)
         ISOException.throwIt(SW_CIPHER_DATA_LENGTH_BAD);

      // ENCRYPT INCOMING BUFFER
      m_encryptCipher.doFinal(apdubuf, ISO7816.OFFSET_CDATA, dataLen,
                              m_ramArray, (short) 0);

      // COPY ENCRYPTED DATA INTO OUTGOING BUFFER
      Util.arrayCopyNonAtomic(m_ramArray, (short) 0, apdubuf,
                              ISO7816.OFFSET_CDATA, dataLen);

      // SEND OUTGOING BUFFER
      apdu.setOutgoingAndSend(ISO7816.OFFSET_CDATA, dataLen);

  }
```

Original JavaCard source code

```
.method  Encrypt(Ljavacard/framework/APDU;)V 129 {
    .stack 6;
    .locals 3;
    .descriptor     Ljavacard/framework/APDU;      0.10;
         L0:    aload_1;
                invokevirtual 30;
                astore_2;
                aload_1;
                invokevirtual 42;
                sstore_3;
                sload_3;
                bspush 8;
                srem;
                ifeq L2;
         L1:    sspush 26384;
                invokestatic 41;
                goto L2;
         L2:    getfield_a_this 1;
                aload_2;
                sconst_5;
                sload_3;
                getfield_a_this 10;
                sconst_0;
                invokevirtual 43;
```

```
            pop;
            getfield_a_this 10;
            sconst_0;
            aload_2;
            sconst_5;
            sload_3;
            invokestatic 44;
            pop;
            aload_1;
            sconst_5;
            sload_3;
            invokevirtual 45;
            return;
}
```

Resulting JavaCard bytecode

Native binary code (assembler)

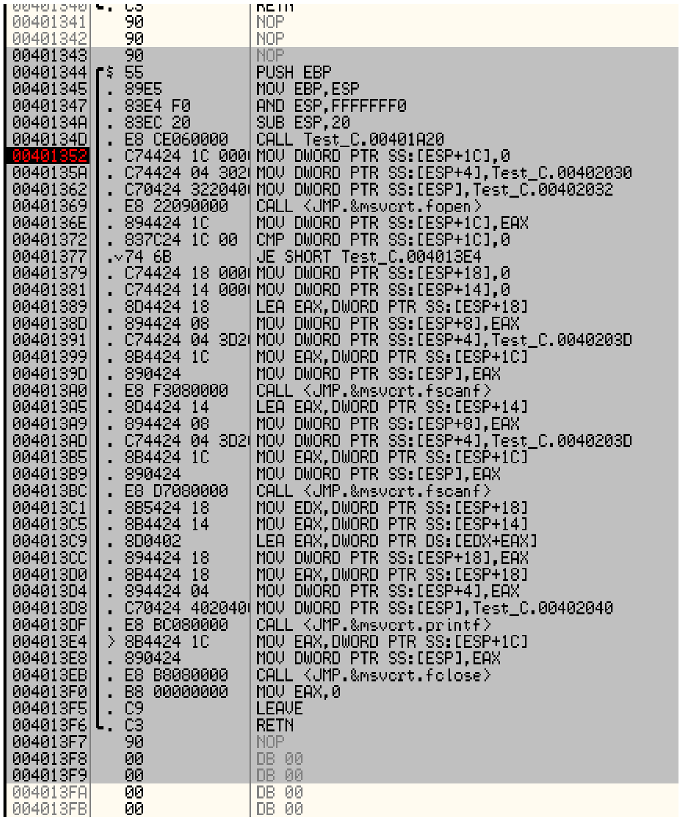
How to start quickly with assembler (mixed mode)

Most current IDE supports mixed source code/assembler instructions mode (Visual Studio, QT Creator...). Mode is usually available during a debugging.

1. Write simple code (e.g., if then else condition), insert breakpoint and start debugging
2. Switch to mixed mode
    a. Visual Studio→RClick →Go to disassembly
    b. QTCreator→Debug→Operate by Instruction
3. Learn how particular source code is translated into assembler code

```
#include <stdio.h>
int main() {
    FILE* file = NULL;
    file = fopen("values.txt", "r");

    if (file) {
        int value1 = 0;
        int value2 = 0;
        fscanf(file, "%d", &value1);
        fscanf(file, "%d", &value2);

        value1 = value1 + value2;

        printf("Result: %d", value1);
    }
    fclose(file);
}
```

Original C source code

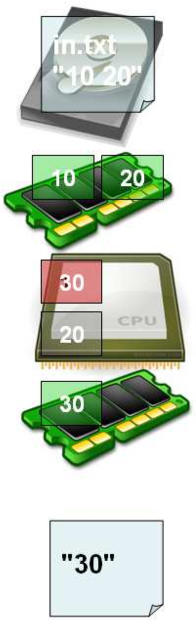


```
00401340 . C3              RETN
00401341   90              NOP
00401342   90              NOP
00401343   90              NOP
00401344 $ 55              PUSH EBP
00401345 . 89E5            MOV EBP,ESP
00401347 . 83E4 F0         AND ESP,FFFFFFF0
0040134A . 83EC 20         SUB ESP,20
0040134D . E8 CE060000     CALL Test_C.00401A20
00401352 . C74424 1C 000(  MOV DWORD PTR SS:[ESP+1C],0
0040135A . C74424 04 302(  MOV DWORD PTR SS:[ESP+4],Test_C.00402030
00401362 . C70424 322040(  MOV DWORD PTR SS:[ESP],Test_C.00402032
00401369 . E8 22090000     CALL <JMP.&msvcrt.fopen>
0040136E . 894424 1C       MOV DWORD PTR SS:[ESP+1C],EAX
00401372 . 837C24 1C 00    CMP DWORD PTR SS:[ESP+1C],0
00401377 .v74 6B           JE SHORT Test_C.004013E4
00401379 . C74424 18 000(  MOV DWORD PTR SS:[ESP+18],0
00401381 . C74424 14 000(  MOV DWORD PTR SS:[ESP+14],0
00401389 . 8D4424 18       LEA EAX,DWORD PTR SS:[ESP+18]
0040138D . 894424 08       MOV DWORD PTR SS:[ESP+8],EAX
00401391 . C74424 04 3D2(  MOV DWORD PTR SS:[ESP+4],Test_C.0040203D
00401399 . 8B4424 1C       MOV EAX,DWORD PTR SS:[ESP+1C]
0040139D . 890424          MOV DWORD PTR SS:[ESP],EAX
004013A0 . E8 F3080000     CALL <JMP.&msvcrt.fscanf>
004013A5 . 8D4424 14       LEA EAX,DWORD PTR SS:[ESP+14]
004013A9 . 894424 08       MOV DWORD PTR SS:[ESP+8],EAX
004013AD . C74424 04 3D2(  MOV DWORD PTR SS:[ESP+4],Test_C.0040203D
004013B5 . 8B4424 1C       MOV EAX,DWORD PTR SS:[ESP+1C]
004013B9 . 890424          MOV DWORD PTR SS:[ESP],EAX
004013BC . E8 D7080000     CALL <JMP.&msvcrt.fscanf>
004013C1 . 8B5424 18       MOV EDX,DWORD PTR SS:[ESP+18]
004013C5 . 8B4424 14       MOV EAX,DWORD PTR SS:[ESP+14]
004013C9 . 8D0402          LEA EAX,DWORD PTR DS:[EDX+EAX]
004013CC . 894424 18       MOV DWORD PTR SS:[ESP+18],EAX
004013D0 . 8B4424 18       MOV EAX,DWORD PTR SS:[ESP+18]
004013D4 . 894424 04       MOV DWORD PTR SS:[ESP+4],EAX
004013D8 . C70424 402040(  MOV DWORD PTR SS:[ESP],Test_C.00402040
004013DF . E8 BC080000     CALL <JMP.&msvcrt.printf>
004013E4 > 8B4424 1C       MOV EAX,DWORD PTR SS:[ESP+1C]
004013E8 . 890424          MOV DWORD PTR SS:[ESP],EAX
004013EB . E8 B8080000     CALL <JMP.&msvcrt.fclose>
004013F0 . B8 00000000     MOV EAX,0
004013F5 . C9              LEAVE
004013F6 . C3              RETN
004013F7   90              NOP
004013F8   00              DB 00
004013F9   00              DB 00
004013FA   00              DB 00
004013FB   00              DB 00
```

Relevant snapshot from executable binary

1. Načtení hodnot z HDD do RAM paměti
   - `fscanf("%d", &value);`
2. Přesun hodnot z RAM paměti do registru CPU
   - `MOV    0x48(%esp),%eax`
3. Provedení instrukce procesoru (např. ADD)
   - `ADD     %edx,%eax`
4. Uložení výsledku registru CPU do RAM
   - `MOV    %eax, 0x48(%esp)`
5. Vypsání na standardní výstup
   - `printf("%d", value);`

`value = value + value2;`



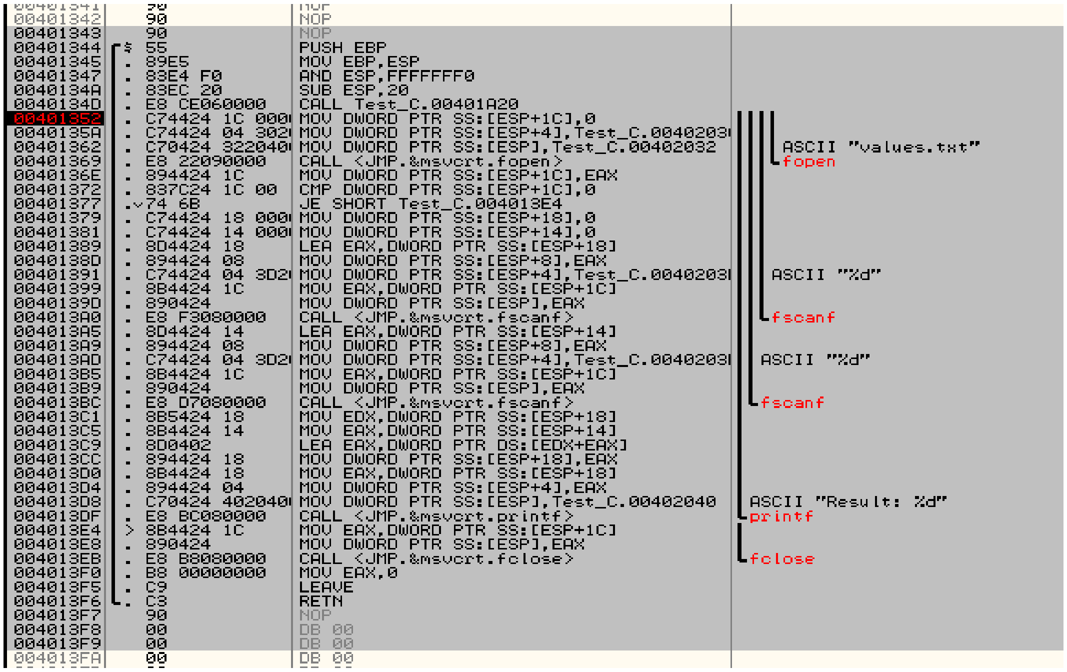
```
      Dump of assembler code for function main:
        2      int main() {
0x00401344  <+0>:             push   %ebp
0x00401345  <+1>:             mov    %esp,%ebp
0x00401347  <+3>:             and    $0xfffffff0,%esp
0x0040134a  <+6>:             sub    $0x20,%esp
0x0040134d  <+9>:             call   0x401a20 <__main>

        3          FILE* file = NULL;
0x00401352  <+14>:            movl   $0x0,0x1c(%esp)

        4          file = fopen("values.txt", "r");
0x0040135a  <+22>:            movl   $0x402030,0x4(%esp)
0x00401362  <+30>:            movl   $0x402032,(%esp)
0x00401369  <+37>:            call   0x401c90 <fopen>
0x0040136e  <+42>:            mov    %eax,0x1c(%esp)

        5
        6          if (file) {
0x00401372  <+46>:            cmpl   $0x0,0x1c(%esp)
0x00401377  <+51>:            je     0x4013e4 <main+160>

        7              int value1 = 0;
0x00401379  <+53>:            movl   $0x0,0x18(%esp)

        8              int value2 = 0;
0x00401381  <+61>:            movl   $0x0,0x14(%esp)

        9              fscanf(file, "%d", &value1);
0x00401389  <+69>:            lea    0x18(%esp),%eax
0x0040138d  <+73>:            mov    %eax,0x8(%esp)
0x00401391  <+77>:            movl   $0x40203d,0x4(%esp)
0x00401399  <+85>:            mov    0x1c(%esp),%eax
0x0040139d  <+89>:            mov    %eax,(%esp)
0x004013a0  <+92>:            call   0x401c98 <fscanf>

        10             fscanf(file, "%d", &value2);
0x004013a5  <+97>:            lea    0x14(%esp),%eax
0x004013a9  <+101>:           mov    %eax,0x8(%esp)
0x004013ad  <+105>:           movl   $0x40203d,0x4(%esp)
0x004013b5  <+113>:           mov    0x1c(%esp),%eax
0x004013b9  <+117>:           mov    %eax,(%esp)
0x004013bc  <+120>:           call   0x401c98 <fscanf>

        11
        12             value1 = value1 + value2;
0x004013c1  <+125>:           mov    0x18(%esp),%edx
0x004013c5  <+129>:           mov    0x14(%esp),%eax
0x004013c9  <+133>:           lea    (%edx,%eax,1),%eax
0x004013cc  <+136>:           mov    %eax,0x18(%esp)

        13
        14             printf("Result: %d", value1);
0x004013d0  <+140>:           mov    0x18(%esp),%eax
0x004013d4  <+144>:           mov    %eax,0x4(%esp)
0x004013d8  <+148>:           movl   $0x402040,(%esp)
0x004013df  <+155>:           call   0x401ca0 <printf>

        15         }
        16         fclose(file);
0x004013e4  <+160>:           mov    0x1c(%esp),%eax
0x004013e8  <+164>:           mov    %eax,(%esp)
0x004013eb  <+167>:           call   0x401ca8 <fclose>
0x004013f0  <+172>:           mov    $0x0,%eax

        17     }
0x004013f5  <+177>:           leave
0x004013f6  <+178>:           ret

        End of assembler dump.
```

Display of mixed mode of source code and resulting assembler instructions

Disassembling binary code (OllyDbg)

In case when only binary code is available (no source code), other approach is required. We will work with OllyDbg (www.ollydbg.de) program that is easy-to-use disassembler and debugger.

- Download OllyDbg 1.10 (freeware) either from http://www.ollydbg.de/ or (better) from IS (OllyDbg.zip).
- Download tutorials I and II. by Lena from IS (tut1.rar and tut2.rar). Additional tutorials can be obtained from http://www.tuts4you.com.
- Download Assembler basics from IS (BasicsOfAssembler.pdf)



```
00401341     90             NOP
00401342     90             NOP
00401343     90             NOP
00401344  /$ 55             PUSH EBP
00401345  |. 89E5           MOV EBP,ESP
00401347  |. 83E4 F0        AND ESP,FFFFFFF0
0040134A  |. 83EC 20        SUB ESP,20
0040134D  |. E8 CE060000    CALL Test_C.00401A20
00401352  |. C74424 1C 000  MOV DWORD PTR SS:[ESP+1C],0
0040135A  |. C74424 04 302  MOV DWORD PTR SS:[ESP+4],Test_C.0040203
00401362  |. C70424 322040  MOV DWORD PTR SS:[ESP],Test_C.00402032    ASCII "values.txt"
00401369  |. E8 22090000    CALL <JMP.&msvcrt.fopen>                  fopen
0040136E  |. 894424 1C      MOV DWORD PTR SS:[ESP+1C],EAX
00401372  |. 837C24 1C 00   CMP DWORD PTR SS:[ESP+1C],0
00401377  |.v74 6B          JE SHORT Test_C.004013E4
00401379  |. C74424 18 000  MOV DWORD PTR SS:[ESP+18],0
00401381  |. C74424 14 000  MOV DWORD PTR SS:[ESP+14],0
00401389  |. 8D4424 18      LEA EAX,DWORD PTR SS:[ESP+18]
0040138D  |. 894424 08      MOV DWORD PTR SS:[ESP+8],EAX
00401391  |. C74424 04 3D2  MOV DWORD PTR SS:[ESP+4],Test_C.0040203   ASCII "%d"
00401399  |. 8B4424 1C      MOV EAX,DWORD PTR SS:[ESP+1C]
0040139D  |. 890424         MOV DWORD PTR SS:[ESP],EAX
004013A0  |. E8 F3080000    CALL <JMP.&msvcrt.fscanf>                 fscanf
004013A5  |. 8D4424 14      LEA EAX,DWORD PTR SS:[ESP+14]
004013A9  |. 894424 08      MOV DWORD PTR SS:[ESP+8],EAX
004013AD  |. C74424 04 3D2  MOV DWORD PTR SS:[ESP+4],Test_C.0040203   ASCII "%d"
004013B5  |. 8B4424 1C      MOV EAX,DWORD PTR SS:[ESP+1C]
004013B9  |. 890424         MOV DWORD PTR SS:[ESP],EAX
004013BC  |. E8 D7080000    CALL <JMP.&msvcrt.fscanf>                 fscanf
004013C1  |. 8B5424 18      MOV EDX,DWORD PTR SS:[ESP+18]
004013C5  |. 8B4424 14      MOV EAX,DWORD PTR SS:[ESP+14]
004013C9  |. 8D0402         LEA EAX,DWORD PTR DS:[EDX+EAX]
004013CC  |. 894424 18      MOV DWORD PTR SS:[ESP+18],EAX
004013D0  |. 8B4424 18      MOV EAX,DWORD PTR SS:[ESP+18]
004013D4  |. 894424 04      MOV DWORD PTR SS:[ESP+4],EAX
004013D8  |. C70424 402040  MOV DWORD PTR SS:[ESP],Test_C.00402040    ASCII "Result: %d"
004013DF  |. E8 BC080000    CALL <JMP.&msvcrt.printf>                 printf
004013E4  |> 8B4424 1C      MOV EAX,DWORD PTR SS:[ESP+1C]
004013E8  |. 890424         MOV DWORD PTR SS:[ESP],EAX
004013EB  |. E8 B8080000    CALL <JMP.&msvcrt.fclose>                 fclose
004013F0  |. B8 00000000    MOV EAX,0
004013F5  |. C9             LEAVE
004013F6  \. C3             RETN
004013F7     90             NOP
004013F8     00             DB 00
004013F9     00             DB 00
004013FA     00             DB 00
```

Disassembled information provided by OllyDbg

OllyDbg shortcuts & most important commands

F3 ... Open binary file
F2 ... Toggle breakpoint (on opcodes, or double click)
F9 ... Run debugged program
Ctrl+F2 ... Restart program, all temporary changes are lost!
F8 ... Step over
F7 ... Step into
Spacebar or double click ... allows to set new opcode. Use when you like to change program behaviour, e.g., replacing conditional jump (JGE) by unconditional jump (JMP) or to discard instruction (NOP).
Alt+BkSp ... Undo change
***Rightclick*->Search for->All referenced text strings** ... Constant text strings referenced in code. Use to find strings like hardcoded passwords, important messages (“Wrong license”). Double click on string will takes you to referencing instruction. Helps you to build mind model quickly.

***Rightclick*->Find references to->Address constant** ... will find references to particular memory elsewhere in the code – use when you like to know where in code the memory is set, changed or otherwise used.
Ctrl+F1 ... Help on Win32 API (WIN32 API help file already prepared in OllyDbg directory (WIN32.HLP)). Use to get meaning of the parameters pushed to stack just before the API function is called.
; ... add or edit your comment for specific code line. Use to write down things you already understand. Use classic paper as well (program mind model)
***Rightclick*->Copy to executable->All modifications (**or **Selection**) … make changes permanent. New window with modified code is opened. ***Rightclick*->Save file** to write patched binary to disk.

Registers (FPU):
Z – zero flag, C – carry flag, S – sign flag. Invert bit flag by double click.
EIP ... next address to execute (instruction pointer)
EBX ... usually loop counter

Some hints

- Assembler is not as difficult as it may seem at first sight. You are not required to write your own program in assembler – you are usually only required to understand existing code, where only very limited set of assembler operations is used.
- Using mixed mode in IDE debugger will quickly provide you an insight, how common programming constructions (assignments, conditional branching, cycles...) are transformed from source code into executable. Usually, you will get only 5-15 instructions per line of the source code, with MOV instruction being the most common.
- Conditional branching is usually realized by two consecutive operations:
    - Comparison operation setting Flags register
    - Conditional jumping operation to address based on Flags (Branch 1)
    - If not jumped then Branch 2 code is directly present on the next instruction, or unconditional jump JMP to Branch 2 is present.
- Comparison operation
    - CMP EAX, -1 - will set flag(s) in Registers, Zero and Sign flags are usually of interest. If two values are same (EAX == -1), Zero flag is set to 1.
    - TEST A, B (usually TEST EAX, EAX) – logical AND operation, results not saved, Flags are set. TEST EAX, EAX will test if value in EAX is equal to 0. If EAX == 0 then Zero flag == 1, 0 otherwise.
- Jump operation
    - Unconditional JMP – jump every time
    - Conditional - based on the current value of flag(s)

JA*	Jump if (unsigned) above	- CF=0 and ZF=0
JB*	Jump if (unsigned) below	- CF=1
JE**	Jump if equal	- ZF=1
JG*	Jump if (signed) greater	- ZF=0 and SF=OF (SF = Sign Flag)
JGE*	Jump if (signed) greater or equal	- SF=OF
JL*	Jump if (signed) less	- SF != OF (!= is not)
JLE*	Jump if (signed) less or equal	- ZF=1 and OF != OF
JMP**	Jump	- Jumps always
JNE**	Jump if not equal	- ZF=0

Disassembling binary code (IDAPro freeware)

Interactive Disassembler is legendary fullfledged disassembler with ability to disassemble many different platforms.

- Free version available for non-comercial uses
- http://www.hex-rays.com/idapro/idadownfreeware.htm
- Free version disassemble only Windows binaries
- Very nice visualization and debuger feature (similar as OllyDbg)
- Try it!

Decompiling binary code

Decompiler is able to produce source code from binary code. Decompiler needs to do disassembling first and then try to create code that will in turn produce binary code you have at the beginning.

- Resulting code will NOT contain information removed during compilation (comments, function names, formatting...)
- Read http://www.debugmode.com/dcompile/ for more info
- Still can be of great help!
- Problem to find well working free disassembler :(
- http://en.wikibooks.org/wiki/X86_Disassembly/Disassemblers_and_Decompilers

Other resources

RE on Wikipedia: http://en.wikipedia.org/wiki/Reverse_engineering
The Reverse Code Engineering Community: http://www.reverse-engineering.net/
Tutorials for You: http://www.tuts4you.com
Disassembling tutorial http://www.codeproject.com/KB/cpp/reversedisasm.aspx

Homework

The goal of this assignment is to reverse engineer supplied crack me file (LabakCrackMe.exe), obtain information about its behavior and make program to continue successfully without error message by a) patching, b) creating valid license info. More principally different solutions for the same problem will be awarded by extra points.

Hints:
- You may use OllyDbg or any other disassembler.
- Function *fread* fail by null exception if invalid file handle is supplied.

Submit:
- Short description of program behavior in text form or as **annotated** C source code (not only output of some disassembler) (source code version will be awarded by 1 extra point).
- Patched crack me binary that let the program run every time successfully with no error **without** valid license info.
- Valid license info that let program run successfully **without binary modification**.