Binary Reverse Engineering And Analysis

Course 3: Static Analysis

Caragea Radu

Last update: 02 March 2021

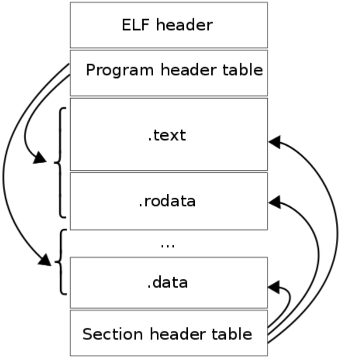
Recap

- Last time: assembly, no ”context”
- Today we put machine code into context:
  - Executable File Formats
  - Rudimentary tools
  - Advanced tools

Executables

- Most executables (ELF/SO, PE/DLL, WASM) have structure
- Based on generic computer science concepts
- Multiple sections/segments:
  - Text section (text == readable by the CPU)
  - Read-only Data section/Read-Write Data Section
  - Relocations/Compiler Stubs

Linux binary format overview

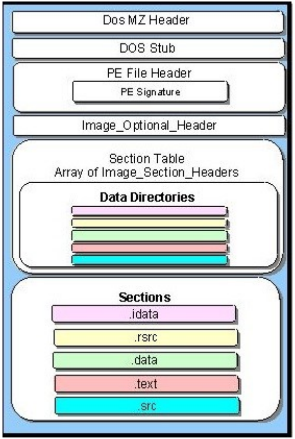


- View 1: by interpreters (using the program header)
- View 2: by linkers (using the section header)
- `https://github.com/corkami/pics/raw/master/binary/ELF101.png`
- `https://ide.kaitai.io`

Linux binary format tools (rudimentary)

- readelf - interpret the file format structures
- objdump - disassemble the code in the text sections
- nm - list symbols
- DEMO time!

Windows binary format overview



- `https://github.com/corkami/pics/raw/master/binary/PE101.png`
- `https://ide.kaitai.io`

Windows binary format tools

- CFF Explorer / PE Studio - full structure interpretation
- PE bear - similar functionality
- DEMO time!

State-of-the-art Analysis Tools

- IDA Pro + Hex-Rays
- Ghidra
- Others: radare, retdec, jeb

Ghidra

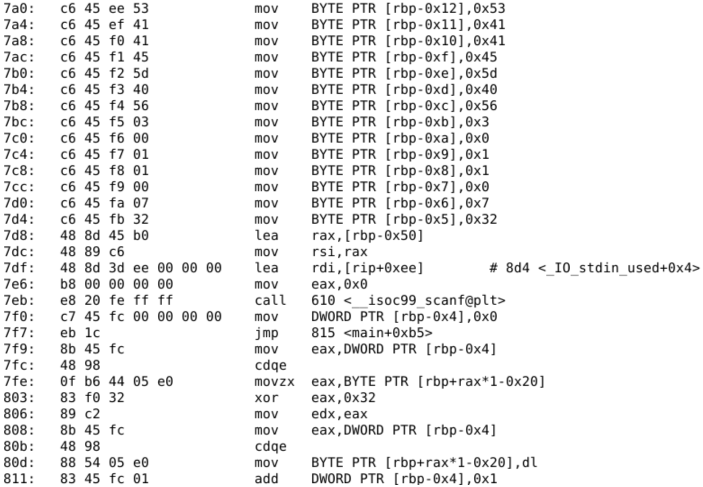
- Open-sourced NSA tool
- Pro: free and hackable
- Pro: decompiles anything it can disassemble
- Con: looks horrible (UI/UX skills zero)
- Con: sometimes the decompilation is impossible to follow
- Prefers gotos (no for loop support)

IDA: Interactive Disassembler

- Swiss army knife of Reverse Engineering
- Pro: Tried and tested
- Pro: Analyze most executable file formats
- Pro: Disassemble most architectures (x86, arm, mips, z80, etc)
- Pro: Decompile some architectures (x86/amd64, arm/arm64, ppc/ppc64, mips32)
- Con: Too expensive
- Con: Piracy is rampant

IDA showcase 1/4

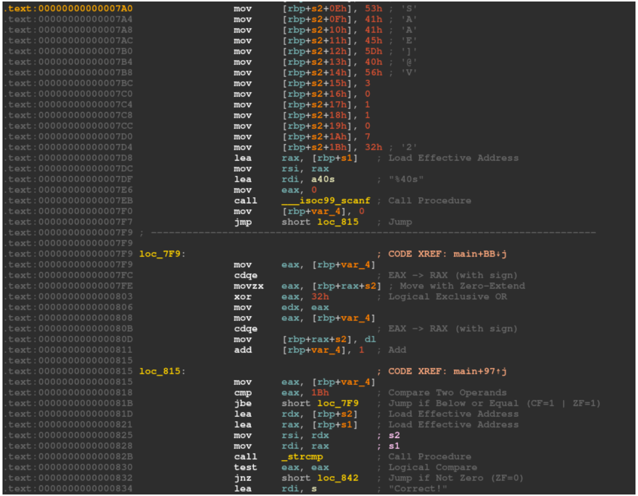
Go from this:



```
7a0:   c6 45 ee 53             mov    BYTE PTR [rbp-0x12],0x53
7a4:   c6 45 ef 41             mov    BYTE PTR [rbp-0x11],0x41
7a8:   c6 45 f0 41             mov    BYTE PTR [rbp-0x10],0x41
7ac:   c6 45 f1 45             mov    BYTE PTR [rbp-0xf],0x45
7b0:   c6 45 f2 5d             mov    BYTE PTR [rbp-0xe],0x5d
7b4:   c6 45 f3 40             mov    BYTE PTR [rbp-0xd],0x40
7b8:   c6 45 f4 56             mov    BYTE PTR [rbp-0xc],0x56
7bc:   c6 45 f5 03             mov    BYTE PTR [rbp-0xb],0x3
7c0:   c6 45 f6 00             mov    BYTE PTR [rbp-0xa],0x0
7c4:   c6 45 f7 01             mov    BYTE PTR [rbp-0x9],0x1
7c8:   c6 45 f8 01             mov    BYTE PTR [rbp-0x8],0x1
7cc:   c6 45 f9 00             mov    BYTE PTR [rbp-0x7],0x0
7d0:   c6 45 fa 07             mov    BYTE PTR [rbp-0x6],0x7
7d4:   c6 45 fb 32             mov    BYTE PTR [rbp-0x5],0x32
7d8:   48 8d 45 b0             lea    rax,[rbp-0x50]
7dc:   48 89 c6                mov    rsi,rax
7df:   48 8d 3d ee 00 00 00    lea    rdi,[rip+0xee]        # 8d4 <_IO_stdin_used+0x4>
7e6:   b8 00 00 00 00          mov    eax,0x0
7eb:   e8 20 fe ff ff          call   610 <__isoc99_scanf@plt>
7f0:   c7 45 fc 00 00 00 00    mov    DWORD PTR [rbp-0x4],0x0
7f7:   eb 1c                   jmp    815 <main+0xb5>
7f9:   8b 45 fc                mov    eax,DWORD PTR [rbp-0x4]
7fc:   48 98                   cdqe
7fe:   0f b6 44 05 e0          movzx  eax,BYTE PTR [rbp+rax*1-0x20]
803:   83 f0 32                xor    eax,0x32
806:   89 c2                   mov    edx,eax
808:   8b 45 fc                mov    eax,DWORD PTR [rbp-0x4]
80b:   48 98                   cdqe
80d:   88 54 05 e0             mov    BYTE PTR [rbp+rax*1-0x20],dl
811:   83 45 fc 01             add    DWORD PTR [rbp-0x4],0x1
```

IDA showcase 2/4

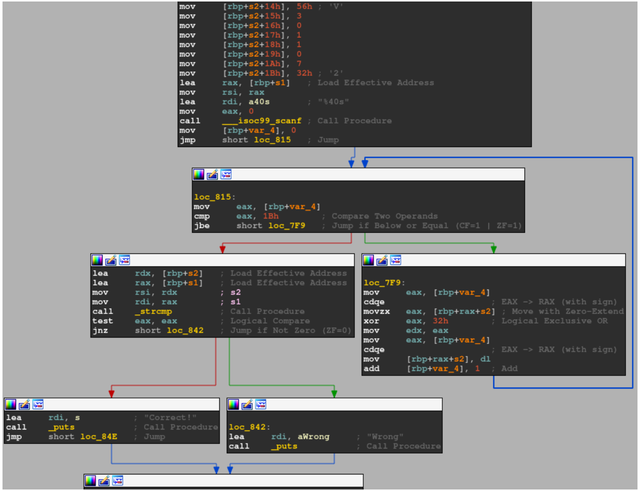
To this:



```
.text:00000000000007A0                 mov     [rbp+s2+0Eh], 53h ; 'S'
.text:00000000000007A4                 mov     [rbp+s2+0Fh], 41h ; 'A'
.text:00000000000007A8                 mov     [rbp+s2+10h], 41h ; 'A'
.text:00000000000007AC                 mov     [rbp+s2+11h], 45h ; 'E'
.text:00000000000007B0                 mov     [rbp+s2+12h], 5Dh ; ']'
.text:00000000000007B4                 mov     [rbp+s2+13h], 40h ; '@'
.text:00000000000007B8                 mov     [rbp+s2+14h], 56h ; 'V'
.text:00000000000007BC                 mov     [rbp+s2+15h], 3
.text:00000000000007C0                 mov     [rbp+s2+16h], 0
.text:00000000000007C4                 mov     [rbp+s2+17h], 1
.text:00000000000007C8                 mov     [rbp+s2+18h], 1
.text:00000000000007CC                 mov     [rbp+s2+19h], 0
.text:00000000000007D0                 mov     [rbp+s2+1Ah], 7
.text:00000000000007D4                 mov     [rbp+s2+1Bh], 32h ; '2'
.text:00000000000007D8                 lea     rax, [rbp+s1]   ; Load Effective Address
.text:00000000000007DC                 mov     rsi, rax
.text:00000000000007DF                 lea     rdi, a40s       ; "%40s"
.text:00000000000007E6                 mov     eax, 0
.text:00000000000007EB                 call    ___isoc99_scanf ; Call Procedure
.text:00000000000007F0                 mov     [rbp+var_4], 0
.text:00000000000007F7                 jmp     short loc_815   ; Jump
.text:00000000000007F9 ; ---------------------------------------------------------------------------
.text:00000000000007F9
.text:00000000000007F9 loc_7F9:                                ; CODE XREF: main+BB↓j
.text:00000000000007F9                 mov     eax, [rbp+var_4]
.text:00000000000007FC                 cdqe                    ; EAX -> RAX (with sign)
.text:00000000000007FE                 movzx   eax, [rbp+rax+s2] ; Move with Zero-Extend
.text:0000000000000803                 xor     eax, 32h        ; Logical Exclusive OR
.text:0000000000000806                 mov     edx, eax
.text:0000000000000808                 mov     eax, [rbp+var_4]
.text:000000000000080B                 cdqe                    ; EAX -> RAX (with sign)
.text:000000000000080D                 mov     [rbp+rax+s2], dl
.text:0000000000000811                 add     [rbp+var_4], 1  ; Add
.text:0000000000000815
.text:0000000000000815 loc_815:                                ; CODE XREF: main+97↑j
.text:0000000000000815                 mov     eax, [rbp+var_4]
.text:0000000000000818                 cmp     eax, 1Bh        ; Compare Two Operands
.text:000000000000081B                 jbe     short loc_7F9   ; Jump if Below or Equal (CF=1 | ZF=1)
.text:000000000000081D                 lea     rdx, [rbp+s2]   ; Load Effective Address
.text:0000000000000821                 lea     rax, [rbp+s1]   ; Load Effective Address
.text:0000000000000825                 mov     rsi, rdx        ; s2
.text:0000000000000828                 mov     rdi, rax        ; s1
.text:000000000000082B                 call    _strcmp         ; Call Procedure
.text:0000000000000830                 test    eax, eax        ; Logical Compare
.text:0000000000000832                 jnz     short loc_842   ; Jump if Not Zero (ZF=0)
.text:0000000000000834                 lea     rdi, s          ; "Correct!"
```

IDA showcase 3/4

To this:



```
mov     [rbp+s2+14h], 56h ; 'V'
mov     [rbp+s2+15h], 3
mov     [rbp+s2+16h], 0
mov     [rbp+s2+17h], 1
mov     [rbp+s2+18h], 1
mov     [rbp+s2+19h], 0
mov     [rbp+s2+1Ah], 7
mov     [rbp+s2+1Bh], 32h ; '2'
lea     rax, [rbp+s1]    ; Load Effective Address
mov     rsi, rax
lea     rdi, a40s        ; "%40s"
mov     eax, 0
call    ___isoc99_scanf ; Call Procedure
mov     [rbp+var_4], 0
jmp     short loc_815    ; Jump
```

```
loc_815:
mov     eax, [rbp+var_4]
cmp     eax, 1Bh        ; Compare Two Operands
jbe     short loc_7F9   ; Jump if Below or Equal (CF=1 | ZF=1)
```

```
lea     rdx, [rbp+s2]   ; Load Effective Address
lea     rax, [rbp+s1]   ; Load Effective Address
mov     rsi, rdx        ; s2
mov     rdi, rax        ; s1
call    _strcmp         ; Call Procedure
test    eax, eax        ; Logical Compare
jnz     short loc_842   ; Jump if Not Zero (ZF=0)
```

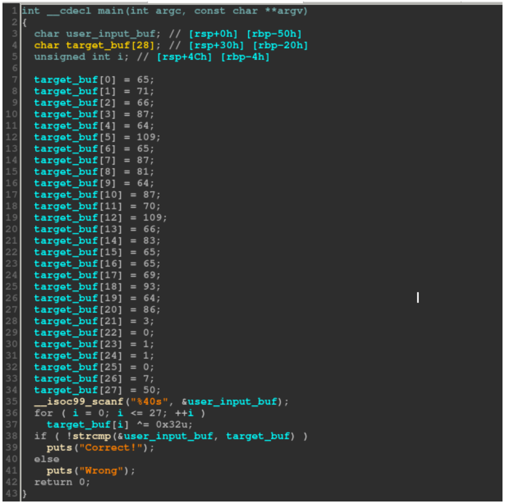
```
loc_7F9:
mov     eax, [rbp+var_4]
cdqe                      ; EAX -> RAX (with sign)
movzx   eax, [rbp+rax+s2] ; Move with Zero-Extend
xor     eax, 32h          ; Logical Exclusive OR
mov     edx, eax
mov     eax, [rbp+var_4]
cdqe                      ; EAX -> RAX (with sign)
mov     [rbp+rax+s2], dl
add     [rbp+var_4], 1  ; Add
```

```
lea     rdi, s          ; "Correct!"
call    _puts           ; Call Procedure
jmp     short loc_84E   ; Jump
```

```
loc_842:
lea     rdi, aWrong     ; "Wrong"
call    _puts           ; Call Procedure
```

IDA showcase 4/4

To this:

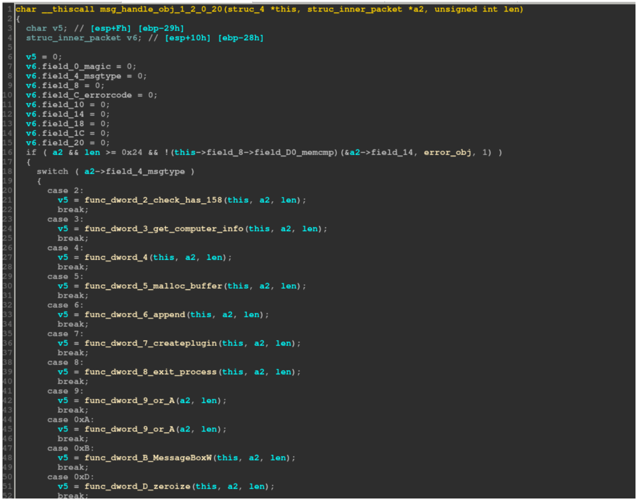


```c
int __cdecl main(int argc, const char **argv)
{
  char user_input_buf; // [rsp+0h] [rbp-50h]
  char target_buf[28]; // [rsp+30h] [rbp-20h]
  unsigned int i; // [rsp+4Ch] [rbp-4h]

  target_buf[0] = 65;
  target_buf[1] = 71;
  target_buf[2] = 66;
  target_buf[3] = 87;
  target_buf[4] = 64;
  target_buf[5] = 109;
  target_buf[6] = 65;
  target_buf[7] = 87;
  target_buf[8] = 81;
  target_buf[9] = 64;
  target_buf[10] = 87;
  target_buf[11] = 70;
  target_buf[12] = 109;
  target_buf[13] = 66;
  target_buf[14] = 83;
  target_buf[15] = 65;
  target_buf[16] = 65;
  target_buf[17] = 69;
  target_buf[18] = 93;
  target_buf[19] = 64;
  target_buf[20] = 86;
  target_buf[21] = 3;
  target_buf[22] = 0;
  target_buf[23] = 1;
  target_buf[24] = 1;
  target_buf[25] = 0;
  target_buf[26] = 7;
  target_buf[27] = 50;
  __isoc99_scanf("%40s", &user_input_buf);
  for ( i = 0; i <= 27; ++i )
    target_buf[i] ^= 0x32u;
  if ( !strcmp(&user_input_buf, target_buf) )
    puts("Correct!");
  else
    puts("Wrong");
  return 0;
}
```

IDA strengths: interactivity

You can rename functions, variables, create structs, etc

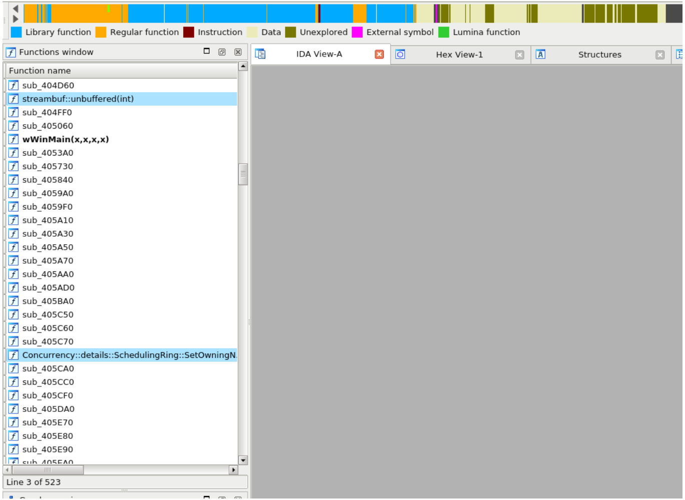


```
char __thiscall msg_handle_obj_1_2_0_20(struc_4 *this, struc_inner_packet *a2, unsigned int len)
{
  char v5; // [esp+Fh] [ebp-29h]
  struc_inner_packet v6; // [esp+10h] [ebp-28h]

  v5 = 0;
  v6.field_0_magic = 0;
  v6.field_4_msgtype = 0;
  v6.field_8 = 0;
  v6.field_C_errorcode = 0;
  v6.field_10 = 0;
  v6.field_14 = 0;
  v6.field_18 = 0;
  v6.field_1C = 0;
  v6.field_20 = 0;
  if ( a2 && len >= 0x24 && !(this->field_8->field_D0_memcmp)(&a2->field_14, error_obj, 1) )
  {
    switch ( a2->field_4_msgtype )
    {
      case 2:
        v5 = func_dword_2_check_has_158(this, a2, len);
        break;
      case 3:
        v5 = func_dword_3_get_computer_info(this, a2, len);
        break;
      case 4:
        v5 = func_dword_4(this, a2, len);
        break;
      case 5:
        v5 = func_dword_5_malloc_buffer(this, a2, len);
        break;
      case 6:
        v5 = func_dword_6_append(this, a2, len);
        break;
      case 7:
        v5 = func_dword_7_createplugin(this, a2, len);
        break;
      case 8:
        v5 = func_dword_8_exit_process(this, a2, len);
        break;
      case 9:
        v5 = func_dword_9_or_A(a2, len);
        break;
      case 0xA:
        v5 = func_dword_9_or_A(a2, len);
        break;
      case 0xB:
        v5 = func_dword_B_MessageBoxW(this, a2, len);
        break;
      case 0xD:
        v5 = func_dword_D_zeroize(this, a2, len);
        break;
```

IDA strengths: reconstruction

Program function reconstruction

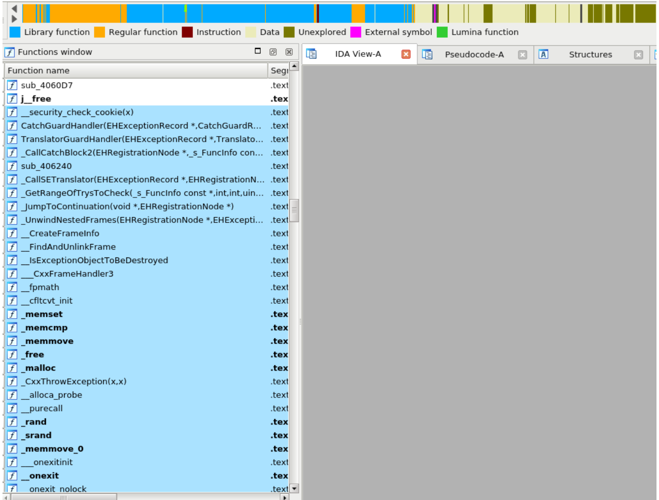


🟦 Library function 🟧 Regular function 🟥 Instruction ▨ Data 🟫 Unexplored 🟪 External symbol 🟩 Lumina function

```
Function name
sub_404D60
streambuf::unbuffered(int)
sub_404FF0
sub_405060
wWinMain(x,x,x,x)
sub_4053A0
sub_405730
sub_405840
sub_4059A0
sub_4059F0
sub_405A10
sub_405A30
sub_405A50
sub_405A70
sub_405AA0
sub_405AD0
sub_405BA0
sub_405C50
sub_405C60
sub_405C70
Concurrency::details::SchedulingRing::SetOwningN
sub_405CA0
sub_405CC0
sub_405CF0
sub_405DA0
sub_405E70
sub_405E80
sub_405E90
sub_405EA0
Line 3 of 523
```

IDA strengths: recognition

Library function recognition by signatures



🟦 Library function 🟧 Regular function 🟥 Instruction 🟨 Data 🟫 Unexplored 🟪 External symbol 🟩 Lumina function

Function name	Segi
sub_4060D7	.text
j__free	**.tex**
__security_check_cookie(x)	.text
CatchGuardHandler(EHExceptionRecord *,CatchGuardR...	.text
TranslatorGuardHandler(EHExceptionRecord *,Translato...	.text
_CallCatchBlock2(EHRegistrationNode *,_s_FuncInfo con...	.text
sub_406240	.text
_CallSETranslator(EHExceptionRecord *,EHRegistrationN...	.text
_GetRangeOfTrysToCheck(_s_FuncInfo const *,int,int,uin...	.text
_JumpToContinuation(void *,EHRegistrationNode *)	.text
_UnwindNestedFrames(EHRegistrationNode *,EHExcepti...	.text
__CreateFrameInfo	.text
__FindAndUnlinkFrame	.text
__IsExceptionObjectToBeDestroyed	.text
___CxxFrameHandler3	.text
__fpmath	.text
__cfltcvt_init	.text
_memset	**.tex**
_memcmp	**.tex**
_memmove	**.tex**
_free	**.tex**
_malloc	**.tex**
_CxxThrowException(x,x)	.text
__alloca_probe	.text
__purecall	.text
_rand	**.tex**
_srand	**.tex**
_memmove_0	**.tex**
___onexitinit	.text
__onexit	**.tex**
__onexit_nolock	.text

Typical workflow in static analysis

- Open the file, wait for the auto-analysis heuristics

Typical workflow in static analysis

- Open the file, wait for the auto-analysis heuristics
- Identify entry point (e.g. main() function), start from there
- DFS or BFS manual parsing of functions (top-down)

Typical workflow in static analysis

- Open the file, wait for the auto-analysis heuristics
- Identify entry point (e.g. main() function), start from there
- DFS or BFS manual parsing of functions (top-down)
- Or start from leafs (bottom-up) and guess based on context

Typical workflow in static analysis

- Open the file, wait for the auto-analysis heuristics
- Identify entry point (e.g. main() function), start from there
- DFS or BFS manual parsing of functions (top-down)
- Or start from leafs (bottom-up) and guess based on context
- Reconstruct: functionality, variable names, function names

Typical workflow in static analysis

- Open the file, wait for the auto-analysis heuristics
- Identify entry point (e.g. main() function), start from there
- DFS or BFS manual parsing of functions (top-down)
- Or start from leafs (bottom-up) and guess based on context
- Reconstruct: functionality, variable names, function names
- Replace C/ASM blocks with descriptive text/comments

Typical workflow in static analysis

- Open the file, wait for the auto-analysis heuristics
- Identify entry point (e.g. main() function), start from there
- DFS or BFS manual parsing of functions (top-down)
- Or start from leafs (bottom-up) and guess based on context
- Reconstruct: functionality, variable names, function names
- Replace C/ASM blocks with descriptive text/comments
- Ultimately, reconstruct comprehensible C code

Demo 1

- Simple "Hello world" in Linux
- Format: ELF with debugging symbols
- Notice:
  - Binary organization: code, data, relocations
  - IDA features: tabs, disassembly, graph view, navbar, xrefs, decompilation, symbols

Demo 2

- Simple ”Hello world” in Windows
- Format: PE without debugging symbols (VS2015/release)
- Notice:
  - Binary organization: code, data, relocations
  - IDA features: xrefs, renaming

Demo 3

- Binary from Lab 01
- Format: ELF without debugging symbols
- Notice:
    - IDA features: data reconstruction

Demo 4

- A deceptive binary
- Format: ELF without debugging symbols
- Notice:
    - IDA decompiler pitfalls

Demo 5

- An adversarial binary
- Format: ELF without debugging symbols
- Notice:
  - IDA decompiler limitations

Other adversarial methods

- Anti-disassembly, anti-decompilation
- Anti-debugging, Anti-VM
- Packers, encrypters, corrupters, obfuscators
- Demo UPX

Practice

- Any Questions?
- `http://pwnthybytes.ro/unibuc_re/03-lab.html`