Reverse Engineering

;0x400a14 : "By Redouane"

```
mov rdi, rax
call sym.imp.strncmp;[gm]
test eax, eax
je 0x400977;[gn]
```

```
 0x40086f ;[gr]
lea rax, qword [local_b0h]
   ; const char * s2
   ; " "
lea rsi, qword 0x00400b1b
   ; char *s1
mov rdi, rax
call sym.imp.strtok;[gp]
mov qword [local_10h], rax
cmp qword [local_10h], 0
jne 0x4008a6;[gq]
```

```
 0x400977 ;[gn]
nop
```

```
 0x400890 ;[gs]
     ; const char * s
```

```
 0x4008a6 ;[gq]
mov rax, qword [local_10h]
```

```
 0x400978 ;[gx]
     ; JMP XREF from 0x00400975 (sym.main)
```

SHELLMATES

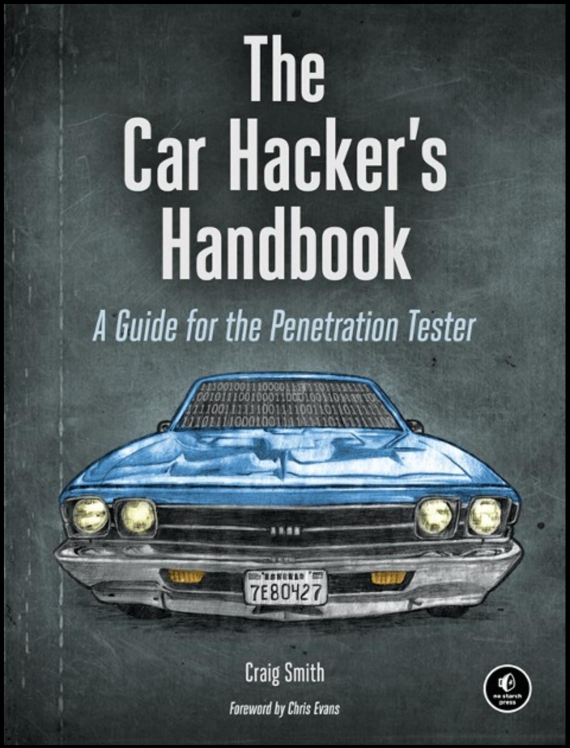
Program

- Introduction : What is Reverse Engineering ?
- A quick look into executable files
- How is Reverse Engineering achieved
- What can make Reverse Engineering more difficult

Introduction

- Reverse Engineering is the process of analysing a product to understand how it works

- In computer science, it’s the analysis of software or hardware to understand how it was built and how it works



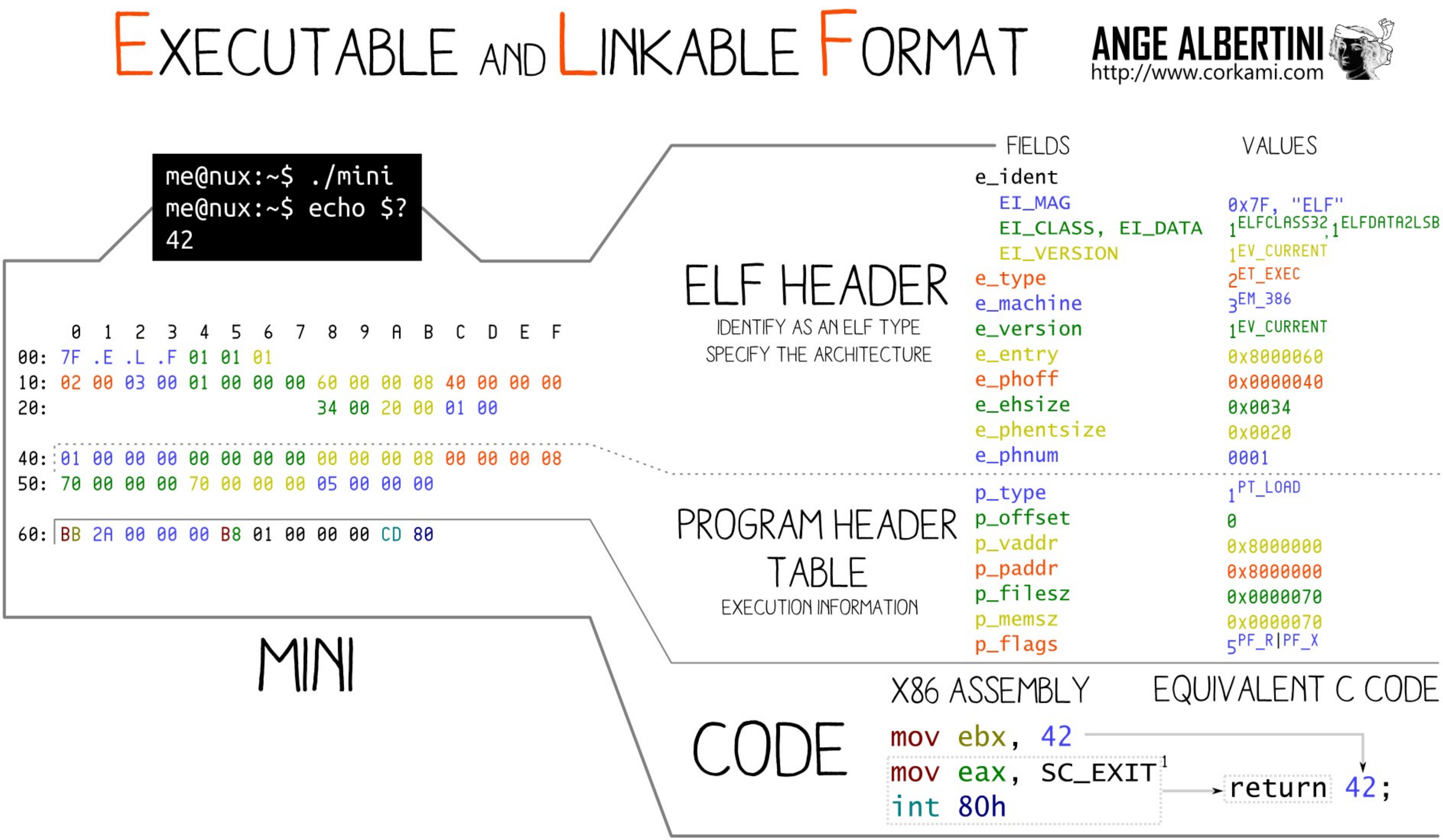
A book on car reverse engineering

A quick look into executable files

- Each operating system has its own «native» executable format

- Examples for some operating systems :
  - PE on Windows
  - ELF on Unix-Like
  - PEF on Mac OS

A quick look into executable files



Header of an ELF file

```
┌─[redouane@Red-Dell]─[~/infosec]
└──╼ $readelf -h prog1
En-tête ELF:
  Magique:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Classe:                            ELF32
  Données:                          complément à 2, système à octets de poids faible d'abord (little endian)
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  Version ABI:                       0
  Type:                              EXEC (fichier exécutable)
  Machine:                           Intel 80386
  Version:                           0x1
  Adresse du point d'entrée:         0x8048510
  Début des en-têtes de programme :  52 (octets dans le fichier)
  Début des en-têtes de section :    4352 (octets dans le fichier)
  Fanions:                           0x0
  Taille de cet en-tête:             52 (octets)
  Taille de l'en-tête du programme:  32 (octets)
  Nombre d'en-tête du programme:     9
  Taille des en-têtes de section:    40 (octets)
  Nombre d'en-têtes de section:      26
  Table d'indexes des chaînes d'en-tête de section: 25
```

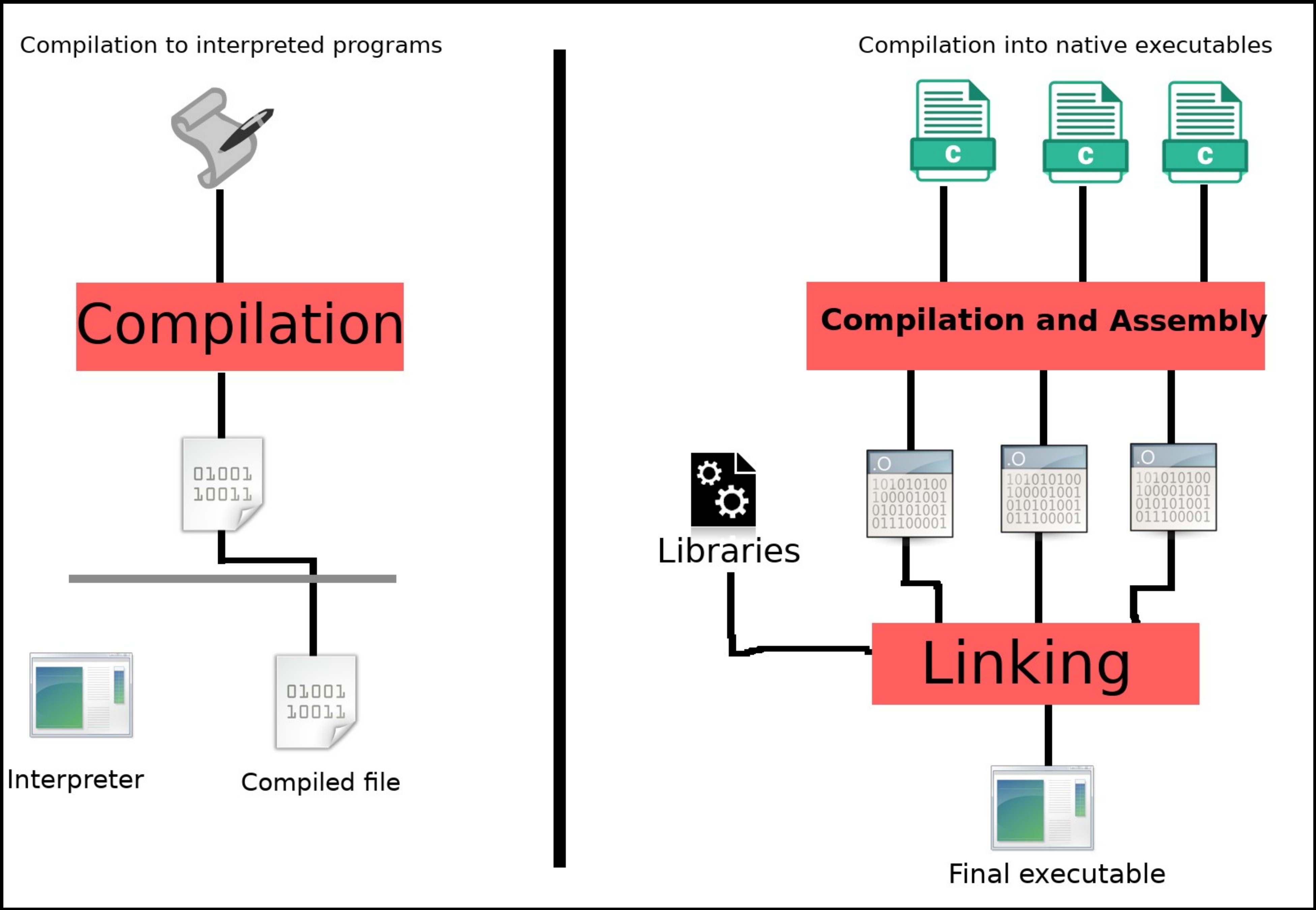
Readelf parsing the header of an ELF file

Sections of an ELF file

```
En-têtes de section :
  [Nr] Nom               Type            Adr      Décala.Taille ES Fan LN Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .interp           PROGBITS        08048154 000154 000013 00   A  0   0  1
  [ 2] .note.ABI-tag     NOTE            08048168 000168 000020 00   A  0   0  4
  [ 3] .hash             HASH            08048188 000188 000050 04   A  5   0  4
  [ 4] .gnu.hash         GNU_HASH        080481d8 0001d8 00002c 04   A  5   0  4
  [ 5] .dynsym           DYNSYM          08048204 000204 0000f0 10   A  6   1  4
  [ 6] .dynstr           STRTAB          080482f4 0002f4 00008f 00   A  0   0  1
  [ 7] .gnu.version      VERSYM          08048384 000384 00001e 02   A  5   0  2
  [ 8] .gnu.version_r    VERNEED         080483a4 0003a4 000020 00   A  6   1  4
  [ 9] .rel.dyn          REL             080483c4 0003c4 000018 08   A  5   0  4
  [10] .rel.plt          REL             080483dc 0003dc 000058 08  AI  5  21  4
  [11] .init             PROGBITS        08048434 000434 000017 00  AX  0   0  4
  [12] .plt              PROGBITS        0804844c 00044c 0000c0 04  AX  0   0  4
  [13] .text             PROGBITS        08048510 000510 00057c 00  AX  0   0 16
  [14] .fini             PROGBITS        08048a8c 000a8c 00001c 00  AX  0   0  4
  [15] .rodata           PROGBITS        08048aa8 000aa8 00012e 00   A  0   0  4
  [16] .eh_frame         PROGBITS        08048bd8 000bd8 000004 00   A  0   0  4
  [17] .ctors            PROGBITS        08049ed0 000ed0 000008 00  WA  0   0  4
  [18] .dtors            PROGBITS        08049ed8 000ed8 000008 00  WA  0   0  4
  [19] .jcr              PROGBITS        08049ee0 000ee0 000004 00  WA  0   0  4
  [20] .dynamic          DYNAMIC         08049ee4 000ee4 0000e0 08  WA  6   0  4
  [21] .got              PROGBITS        08049fc4 000fc4 00003c 04  WA  0   0  4
  [22] .data             PROGBITS        0804a000 001000 000008 00  WA  0   0  4
  [23] .bss              NOBITS          0804a020 001008 00002c 00  WA  0   0 32
  [24] .comment          PROGBITS        00000000 001008 000033 01  MS  0   0  1
  [25] .shstrtab         STRTAB          00000000 00103b 0000c2 00      0   0  1
Clé des fanions :
  W (écriture), A (allocation), X (exécution), M (fusion), S (chaînes), I (info),
  L (ordre des liens), O (traitement supplémentaire par l'OS requis), G (groupe),
  T (TLS), C (compressé), x (inconnu), o (spécifique à l'OS), E (exclu),
  p (processor specific)
:
```

Sections of an ELF file (output of readelf --sections prog)

The compilation process



Symbols

Compilers keep the names of functions and global variables as symbols, to facilitate debugging

The strip command removes all the symbols from an ELF file

```
08048a8c T _fini
08048aa8 R _fp_hw
080486b3 T from
080487cb T getdata
08049fc4 d _GLOBAL_OFFSET_TABLE_
         w __gmon_start__
080485e4 T helo
08048a5a T __i686.get_pc_thunk.bx
08048434 T _init
08049ed0 d __init_array_end
08049ed0 d __init_array_start
08048aac R _IO_stdin_used
         w _Jv_RegisterClasses
080489f0 T __libc_csu_fini
08048a00 T __libc_csu_init
         U __libc_start_main@@GLIBC_2.0
080489bd T main
08048899 T mainprocess
         U memcpy@@GLIBC_2.0
         U puts@@GLIBC_2.0
0804873e T rcptto
08048510 T _start
0804a020 B stdin@@GLIBC_2.0
0804a040 B stdout@@GLIBC_2.0
         U strlen@@GLIBC_2.0
         U strncat@@GLIBC_2.0
         U strncmp@@GLIBC_2.0
         U strncpy@@GLIBC_2.0
```

Output of the nm command

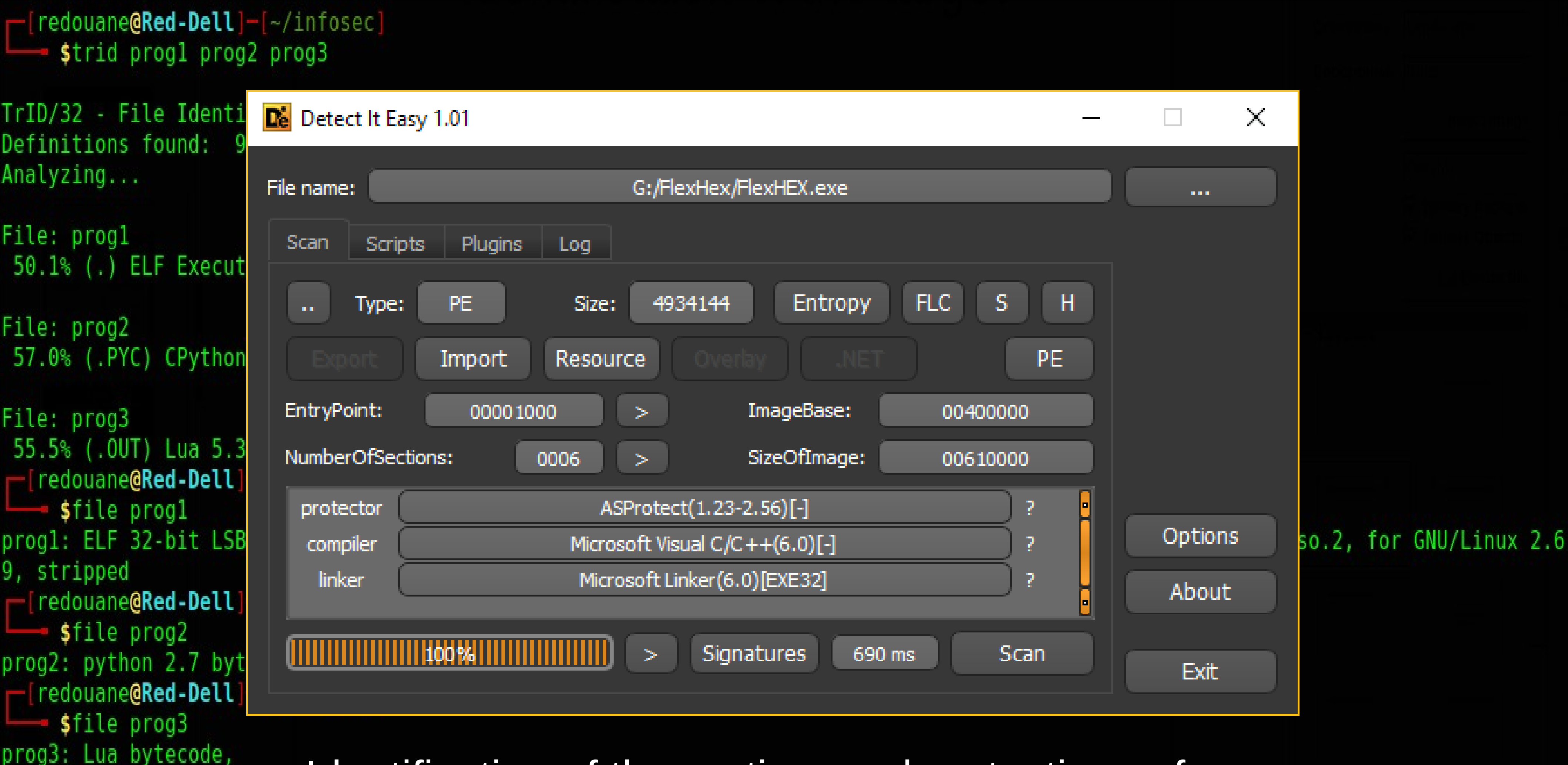
How is Reverse Engineering achieved

- Reverse Engineering of programs is done in two principal steps : Static and Dynamic Analysis
- The methods can differ depending on the program

Identification of the target

- We attempt to detect the runtime of the program, as well as any packers or publicly avaliable protections that it could be using
- For that, we use file identifiers and packer detectors (examples on Windows : DiE, PeiD, on Linux : trid), as well as hex editors

Identification of the target



Identification of the runtime and protections of a program on Windows

Identification of the format of three files using the commands trid and file

Disassembly

- Disassembly is the process of going from the machine code to the instructions that the processor will execute (inverse of assembly)

```
08048510 <.text>:
 8048510:       31 ed                   xor    ebp,ebp
 8048512:       5e                      pop    esi
 8048513:       89 e1                   mov    ecx,esp
 8048515:       83 e4 f0                and    esp,0xfffffff0
 8048518:       50                      push   eax
 8048519:       54                      push   esp
 804851a:       52                      push   edx
 804851b:       68 f0 89 04 08          push   0x80489f0
 8048520:       68 00 8a 04 08          push   0x8048a00
 8048525:       51                      push   ecx
 8048526:       56                      push   esi
 8048527:       68 bd 89 04 08          push   0x80489bd
 804852c:       e8 5b ff ff ff          call   804848c <__libc_start_main@plt>
 8048531:       f4                      hlt
 8048532:       90                      nop
 8048533:       90                      nop
 8048534:       55                      push   ebp
 8048535:       89 e5                   mov    ebp,esp
 8048537:       53                      push   ebx
 8048538:       83 ec 04                sub    esp,0x4
 804853b:       e8 00 00 00 00          call   8048540 <exit@plt+0x44>
 8048540:       5b                      pop    ebx
```

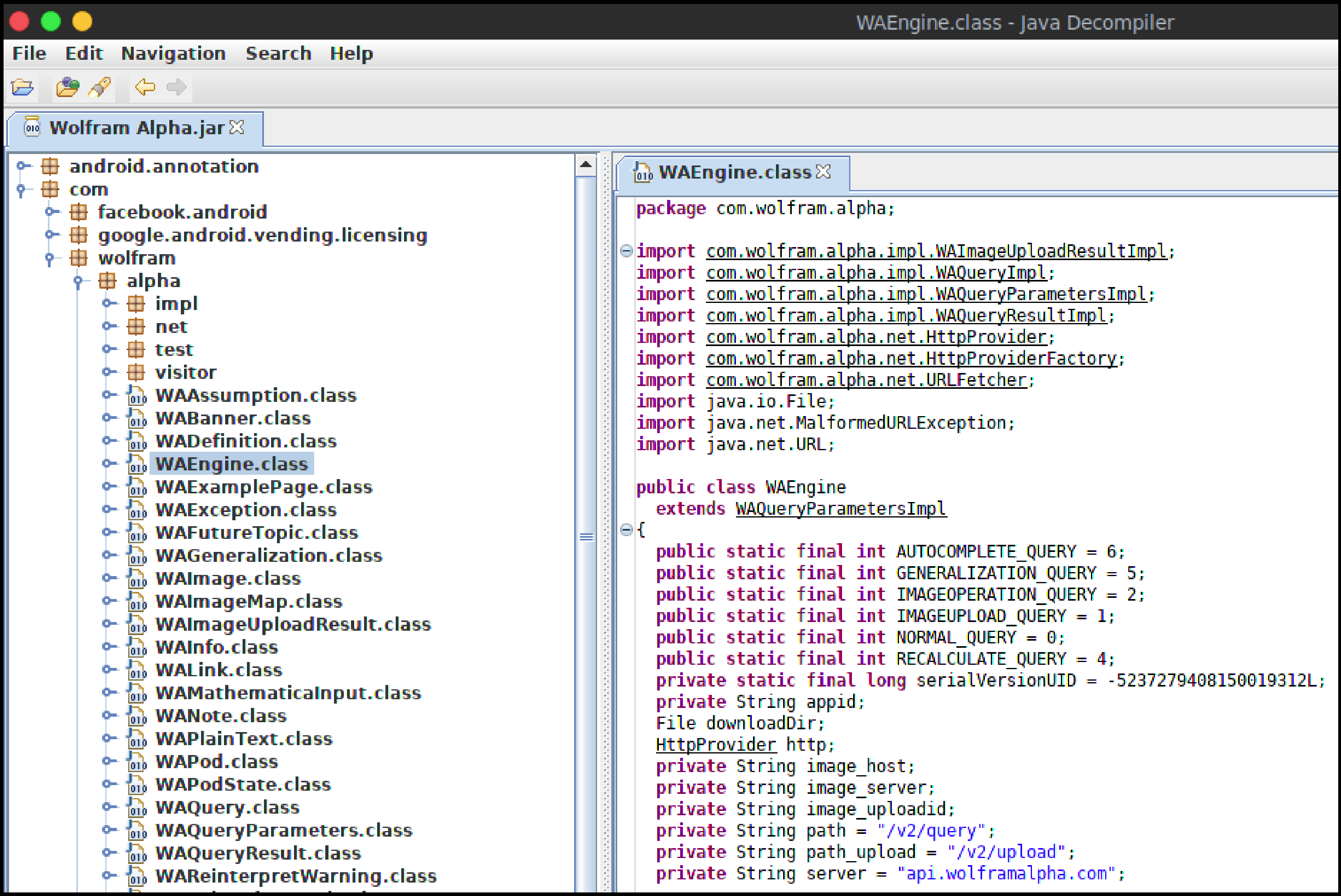
.text section of an ELF file

Decompilation

- Decompilation is the process of going from a compiled program to its source code
- Applicable when the compiled file contains enough informations to get back its source code

  (Java, .NET, Python, Lua etc.)
- Hard to apply on native executables

Decompilation



Decompilation of the Wolfram Alpha android app

Code profiling: tracing events

- Strace traces all the syscalls made by your program
- Ltrace traces the calls from your program to functions in dynamically linked libraries
- Other utilities like Valgrind can be used to do more advanced profiling

Debugging

- The analysis of the program at runtime
- The ability to pause and resume its execution, and to change its registers or memory at runtime

Debugging

```
RBP: 0x7fffffffe110 --> 0x400640 (<__libc_csu_init>:    push   r15)
RSP: 0x7ffffffb9f0 --> 0x0
RIP: 0x4005f1 (<main+58>:       mov    rdi,rax)
R8 : 0x8
R9 : 0x1
R10: 0x0
R11: 0x246
R12: 0x4004e0 (<_start>:        xor    ebp,ebp)
R13: 0x7fffffffe1f0 --> 0x1
R14: 0x0
R15: 0x0
EFLAGS: 0x246 (carry PARITY adjust ZERO sign trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0x4005e4 <main+45>:  mov    rcx,rdx
   0x4005e7 <main+48>:  mov    edx,0x1
   0x4005ec <main+53>:  mov    esi,0x2710
=> 0x4005f1 <main+58>:  mov    rdi,rax
   0x4005f4 <main+61>:  call   0x4004a0 <fread@plt>
   0x4005f9 <main+66>:  lea    rax,[rbp-0x2720]
   0x400600 <main+73>:  mov    rsi,rax
   0x400603 <main+76>:  lea    rdi,[rip+0xd0]        # 0x4006da
[------------------------------------stack-------------------------------------]
0000| 0x7ffffffb9f0 --> 0x0
0008| 0x7ffffffb9f8 --> 0x0
0016| 0x7ffffffba00 --> 0x0
0024| 0x7ffffffba08 --> 0x0
0032| 0x7ffffffba10 --> 0x0
0040| 0x7ffffffba18 --> 0x0
0048| 0x7ffffffba20 --> 0x0
0056| 0x7ffffffba28 --> 0x0
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
0x00000000004005f1      7               fread(data, 10000, 1, f);
gdb-peda$
```

Example debugging session under gdb (using peda)

Control Flow Graphs (CFG)

- Reverse Engineering complex programs can be hard and time consuming
- Control flow graphs allow us to know about the flow of execution of the program more easily
- Some Reverse Engineering frameworks like IDA, Radare2 and x64dbg support them

Control Flow Graphs (CFG)

Radare2 displaying the Control Flow Graph of a program

Assembly Language

- An Architecture-dependent programming language
- A bunch of architecture-dependent instructions that the processor executes sequentially
- These instructions have access to the memory, as well as the registers, which are used for the temporary storage of data

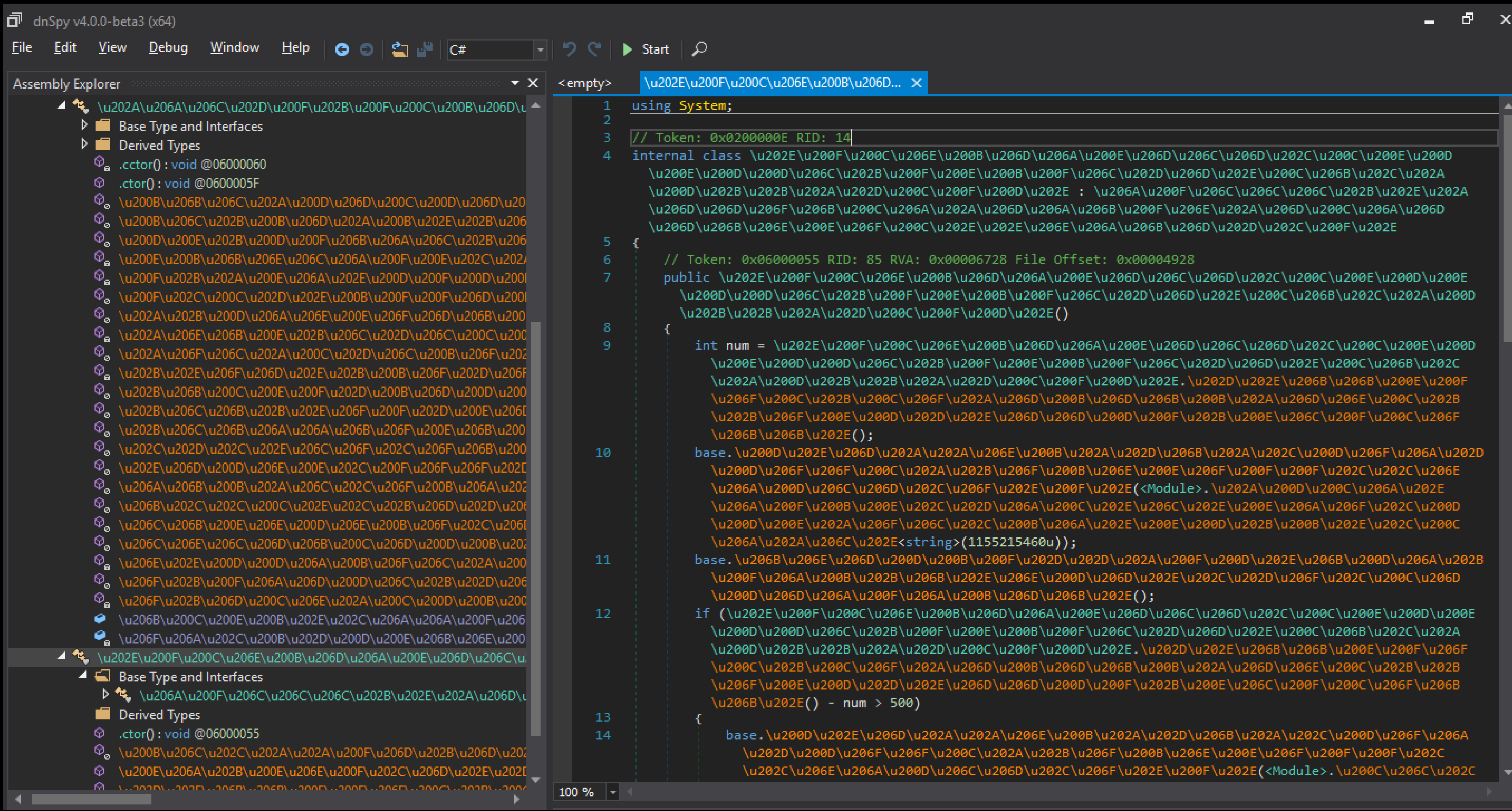
What can make Reverse Engineering more difficult

- Reverse Engineering is often applied on paid software (what we call piracy)
- For many paid software developers and malware authors, Reverse Engineering is a real threat
- A lot of techniques can be applied to make the task harder, but none of them can make it unapplicable

Obfuscation

- Obfuscation is when we try to make code hard to read and follow
- Many automatic obfuscators are avaliable for different programming langages
- Very good solution when the source code of the program is avaliable

Obfuscation



Example of an obfuscated .NET program

Anti-Debugging

- The goal is to detect the presence of a debugger, or to make it fail to attach
- Many OS-dependent tricks can be applied :
  - IsDebuggerPresent() or a similar API on Windows
  - ptrace on Linux
  - Checking the process list for the names of the debuggers
  - The attempt of the process to debug itself

Packers

- The role of packers is to output a program that has the given program embedded in a packed form, and when the output program is ran, the decompression happends and the main program runs

- Popular packers are : UPX, PECompact, ExeStealth

Integrity Checks

- Programs check if their state has been altered or not
- Checksums are frequently used to check for integrity
- This technique also detects software breakpoints, but not hardware ones

Integrity Checks

Example : H is a hash function

- H(protected_region) = 13be6f7e92f14bc09251

- If a single bit changes in protected_region, H(protected_region) will completely change, so it will be detected, and the program will quit

Virtual Machine emulation

- Code is translated to a langage that ressembles assembly langage, but that uses different instruction encodings
- The program emulates the execution (reads the bytecodes, disassembles them, and does the corresponding operations)
- Very effective and widely used (VMProtect, Themida, Denuvo etc.), but affects performance a lot

Thank you !

Any questions ?