Corso di Sicurezza dei Sistemi Informatici

Basics of Reverse Engineering for Security

Luigi Coppolino

Contact info

Prof. Luigi Coppolino
luigi.coppolino@uniparthenope.it

Prof. Salvatore D'Antonio
salvatore.dantonio@uniparthenope.it

Prof. Luigi Romano
luigi.romano@uniparthenope.it

Università degli Studi di Napoli "Parthenope"
Dipartimento di Ingegneria

Roadmap

- ...

What is RE?

- Reverse engineering is the process of extracting the knowledge or design blueprints from anything man-made
  - conducted to obtain missing knowledge, ideas, and design philosophy when such information is unavailable

Why Reversing

- Security-Related Reversing
  - Cryptographic algorithms
  - Vulnerability research
  - Malware analysis
  - Digital Right Management
- Reversing in Software Development
  - Interoperability with Proprietary Software
  - Competing Software
  - ***Evaluating Software Quality and Robustness***

The Reversing Process

- *System-level reversing:* determine the general structure of the program and sometimes even locate areas of interest within it
  - Look at interaction with external word, mainly the OS: networking activity, file accesses, registry access, …
  - determine areas of special interest
- Code-level reversing: extracting design concepts and algorithms from a program binary

- Static VS Dynamic Analysis

Understanding ELF

What about the file…

- The *file* command provides some info about the file

```
$ file somma.c
somma.c: C source, ASCII text, with CRLF line terminators
```

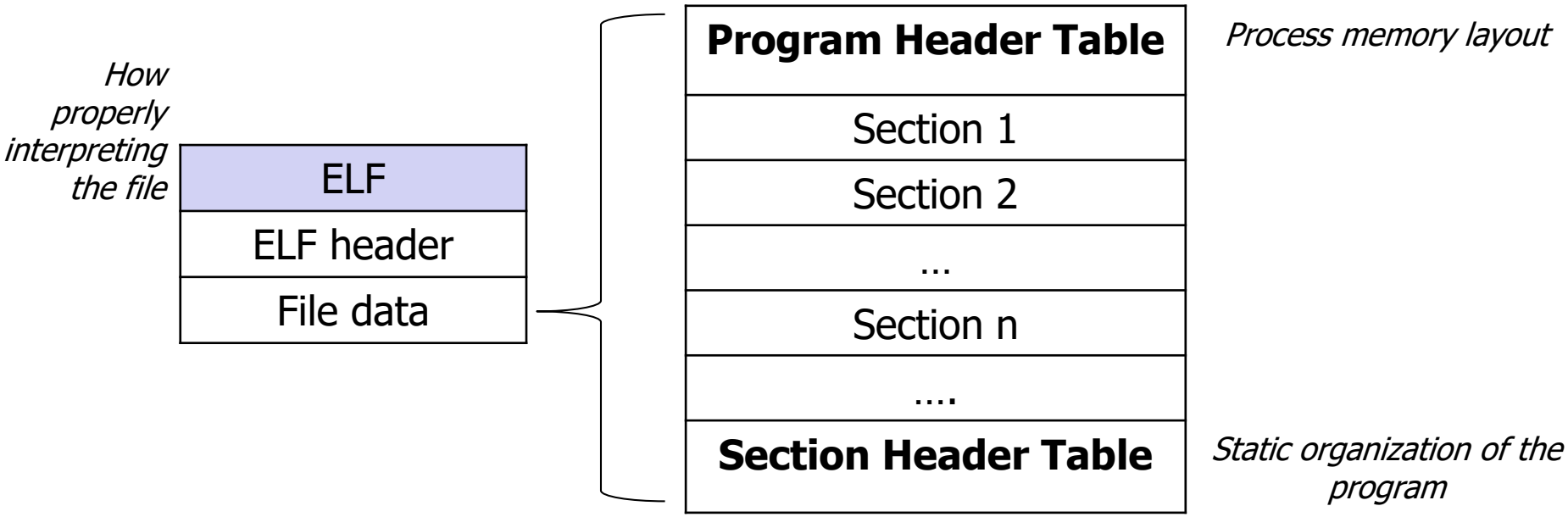
```
$ file somma
somma: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically
linked, interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 2.6.32, B
uildID[sha1]=853a47de5fb8e195468006aa7b4a12c43eff2bca, not stripped
```

```
$ cp somma somma.jpg
```

```
$ file somma.jpg
somma.jpg: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamica
lly linked, interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 2.6.3
2, BuildID[sha1]=853a47de5fb8e195468006aa7b4a12c43eff2bca, not stripped
```

The ELF File Format

- **Executable and Linkable Format:** Linux binary format
  - Executable
  - Object files (relocatable)
  - Shared Objects (.so)



ELF Header

- FILE ORGANIZATION
  - The magic number identifies somma as an ELF file
    - 7f followed by : 45=**E**,4c=**L**,46=**F**
  - the following byte is the architecture (01=32bit 02=64 bit)

```
$ readelf -h somma
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x4003e0
  Start of program headers:          64 (bytes into file)
  Start of section headers:          7568 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         9
  Size of section headers:           64 (bytes)
  Number of section headers:         36
  Section header string table index: 33
```

File data

Sections

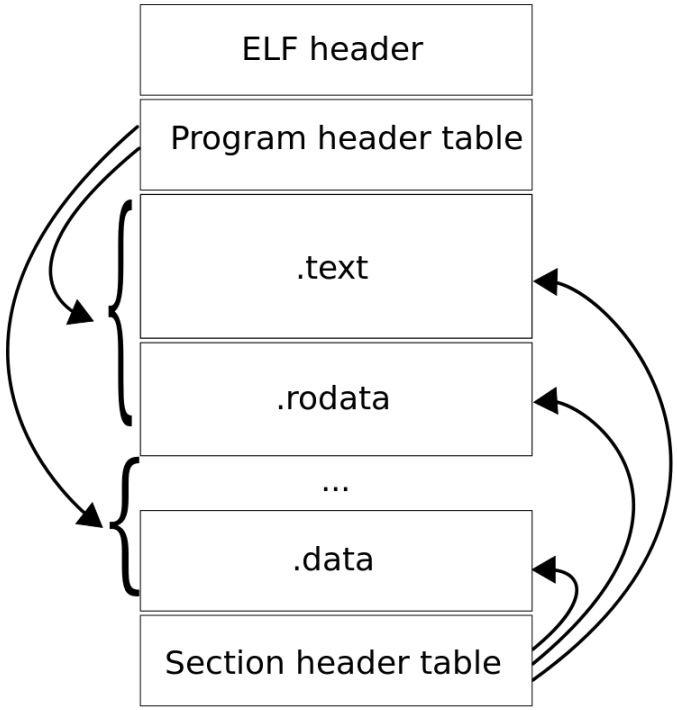
.text = code

.rodata = read only data (ex. strings)

.bss = uninitialized arrays and variables

.data = global tables, variables

Program header presents the program runtime layout in terms of **segments**: a segment can include one or more sections



Sections

readelf -S somma

There are 36 section headers, starting at offset 0x1d90:

Section Headers:

[Nr] Name	Type	Address	Offset	Size	EntSize	Flags	Link	Info	Align
[0]	NULL	0000000000000000	00000000	0000000000000000	0000000000000000		0	0	0
…									
[14] .text	PROGBITS	00000000004003e0	000003e0	00000000000001b2	0000000000000000	AX	0	0	16
[15] .fini	PROGBITS	0000000000400594	00000594	0000000000000009	0000000000000000	AX	0	0	4
[16] .rodata	PROGBITS	00000000004005a0	000005a0	0000000000000004	0000000000000004	AM	0	0	4
…									
[25] .data	PROGBITS	0000000000601020	00001020	0000000000000010	0000000000000000	WA	0	0	8
[26] .bss	NOBITS	0000000000601030	00001030	0000000000000008	0000000000000000	WA	0	0	1

…

Key to Flags:

W (write), A (alloc), X (execute), M (merge), S (strings), l (large) I (info), L (link order), G (group), T (TLS), E (exclude), x (unknown)

Segments

readelf -l somma

Elf file type is EXEC (Executable file)

Entry point 0x4003e0

There are 9 program headers, starting at offset 64

Program Headers:

Type	Offset	VirtAddr	PhysAddr	FileSiz	MemSiz	Flags	Align
PHDR	0x0000000000000040	0x0000000000400040	0x0000000000400040	0x00000000000001f8	0x00000000000001f8	R E	8
INTERP	0x0000000000000238	0x0000000000400238	0x0000000000400238	0x000000000000001c	0x000000000000001c	R	1
[Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]							
LOAD	0x0000000000000000	0x0000000000400000	0x0000000000400000	0x00000000000006f4	0x00000000000006f4	R E	200000
LOAD	0x0000000000000e10	0x0000000000600e10	0x0000000000600e10	0x0000000000000220	0x0000000000000228	RW	200000

…

Section to Segment mapping:

Segment Sections...

00

01 .interp

02 .interp .note.ABI-tag .note.gnu.build-id .gnu.hash .dynsym .dynstr .gnu.version .gnu.version_r .rela.dyn .rela.plt .init .plt .plt.got .text .fini .rodata …

03 .init_array .fini_array .jcr .dynamic .got .got.plt .data .bss

…

Symbols

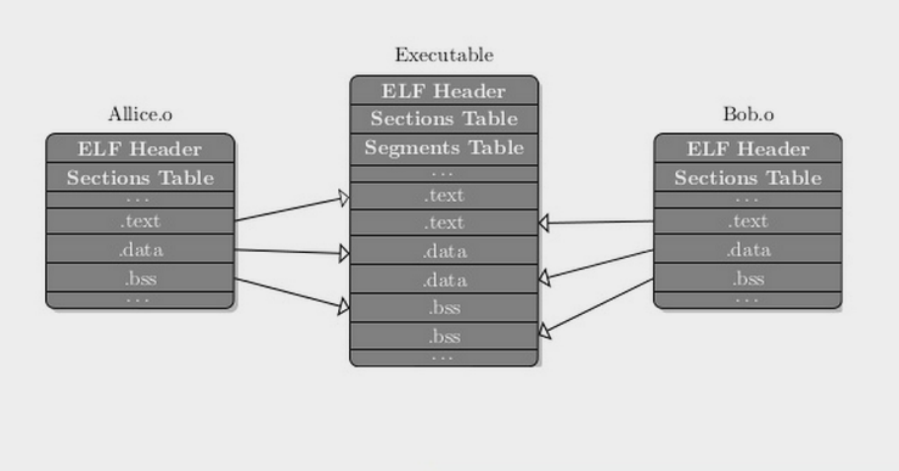
$ readelf -s somma | more

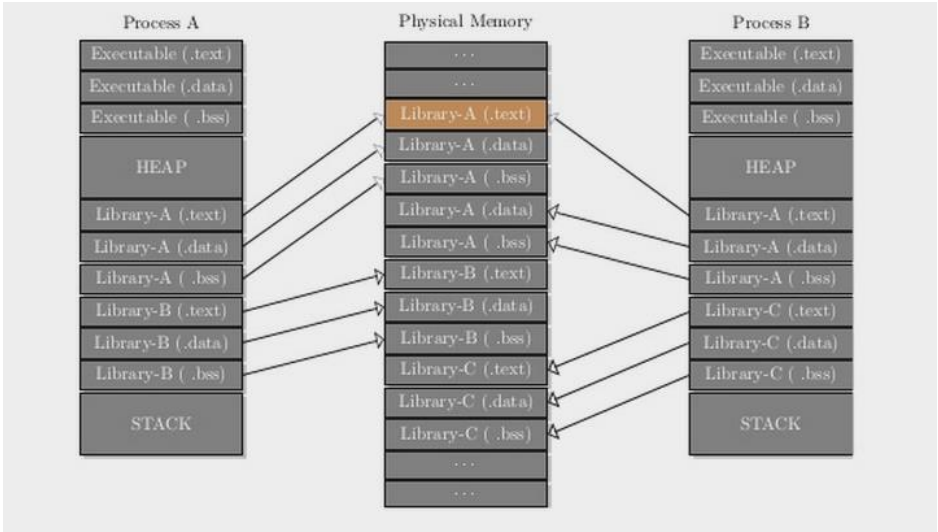
```
Symbol table '.dynsym' contains 4 entries:
   Num:    Value          Size Type    Bind   Vis      Ndx Name
     0: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND printf@GLIBC_2.2.5 (2)
     2: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND __libc_start_main@GLIBC_2.2.5 (2)
     3: 0000000000000000     0 NOTYPE  WEAK   DEFAULT  UND __gmon_start__

Symbol table '.symtab' contains 68 entries:
   Num:    Value          Size Type    Bind   Vis      Ndx Name
     0: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 0000000000400238     0 SECTION LOCAL  DEFAULT    1
     2: 0000000000400254     0 SECTION LOCAL  DEFAULT    2
    ...
    60: 0000000000400430    42 FUNC    GLOBAL DEFAULT   14 _start
    61: 0000000000601038     0 NOTYPE  GLOBAL DEFAULT   26 __bss_start
    62: 000000000040053a    67 FUNC    GLOBAL DEFAULT   14 main
    63: 0000000000000000     0 NOTYPE  WEAK   DEFAULT  UND _Jv_RegisterClasses
    64: 0000000000601038     0 OBJECT  GLOBAL HIDDEN    25 __TMC_END__
    65: 0000000000000000     0 NOTYPE  WEAK   DEFAULT  UND _ITM_registerTMCloneTable
    66: 0000000000400526    20 FUNC    GLOBAL DEFAULT   14 somma
    67: 00000000004003c8     0 FUNC    GLOBAL DEFAULT   11 _init
```

Dynamic Linking

Static vs Dynamic

- Static binaries: the linker includes in the final binary all of the necessary libraries
  - Bigger files, yet more portable
- Dynamic binaries: libraries are kept apart and loaded at runtime on request
  - Smaller files, libraries can be shared by more binaries





Dynamic libraries

```
$ file somma
somma: ELF 64-bit LSB executable, x86-64,
version 1 (SYSV), dynamically linked,
interpreter /lib64/ld-linux-x86-64.so.2, for
GNU/Linux 2.6.32,
BuildID[sha1]=a2a00f667f21ef963f1b896f8a5b3918
db15bbc5, not stripped
```

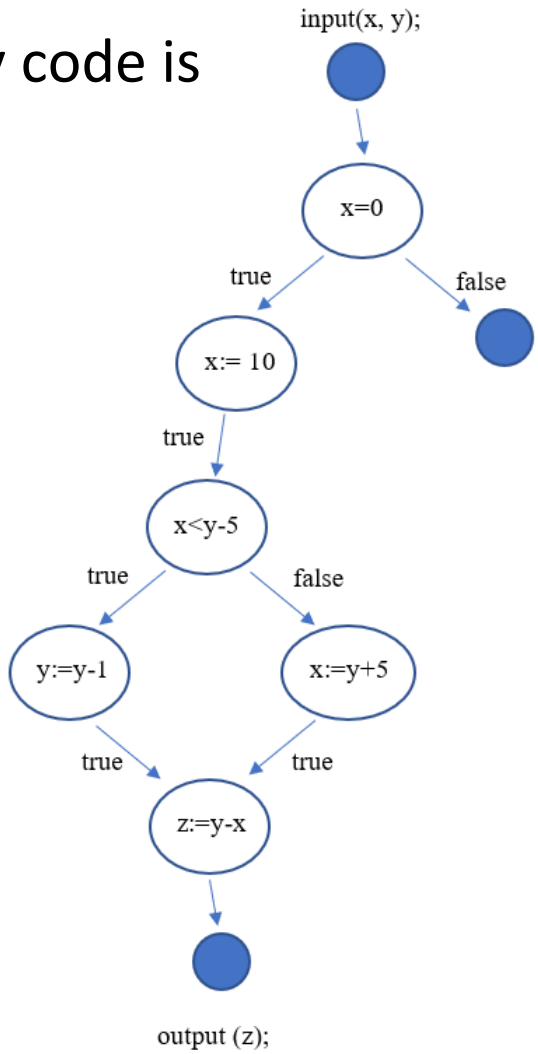
```
$ objdump -p somma | grep NEEDED
NEEDED               libc.so.6
```

PS **ldd somma** can provide additional info but may execute the program => @@@MALWARE!!!

Static Code Review: starting RE

The Compass of the Reverser

- The starting point for Static Analysis of binary code is reconstructing its **Control Flow**, that is
  - The ***order*** of instructions execution;
  - The ***logic*** behind control flow statements
- It is determined by **conditional blocks**:
  - If blocks
  - Switch blocks
  - Loops cycles (including foor)



Control Flow Statements: the Assembly view

```
//gcc -g -ocontrolFlow
//controlFlow.c
#include <stdio.h>

int main(){
   int j, i = 0;
   for(j=0;j < 2; j++)
      i=i+1;
   while(i < 10) i++;
   if (i < 10) i++;
   else i--;
```

```
   switch(i){
      case 1:
         i+=2;
         break;
      case 2:
         i+=3;
         break;
      default:
         i+=1;
   }
}
```

objdump -S -d -Mintel controlFlow

```
int main(){
  4004d6:       55                      push   rbp
  4004d7:       48 89 e5                mov    rbp,rsp
        int j, i = 0;
  4004da:       c7 45 fc 00 00 00 00    mov    DWORD PTR [rbp-0x4],0x0
        for(j=0;j < 2; j++)
  4004e1:       c7 45 f8 00 00 00 00    mov    DWORD PTR [rbp-0x8],0x0
  4004e8:       eb 08                   jmp    4004f2 <main+0x1c>
                i=i+1;
  4004ea:       83 45 fc 01             add    DWORD PTR [rbp-0x4],0x1
#include <stdio.h>

int main(){
        int j, i = 0;
        for(j=0;j < 2; j++)
  4004ee:       83 45 f8 01             add    DWORD PTR [rbp-0x8],0x1
  4004f2:       83 7d f8 01             cmp    DWORD PTR [rbp-0x8],0x1
  4004f6:       7e f2                   jle    4004ea <main+0x14>
                i=i+1;
        while(i < 10) i++;
  4004f8:       eb 04                   jmp    4004fe <main+0x28>
  4004fa:       83 45 fc 01             add    DWORD PTR [rbp-0x4],0x1
  4004fe:       83 7d fc 09             cmp    DWORD PTR [rbp-0x4],0x9
  400502:       7e f6                   jle    4004fa <main+0x24>
        if (i < 10)     i++;
  400504:       83 7d fc 09             cmp    DWORD PTR [rbp-0x4],0x9
  400508:       7f 06                   jg     400510 <main+0x3a>
  40050a:       83 45 fc 01             add    DWORD PTR [rbp-0x4],0x1
  40050e:       eb 04                   jmp    400514 <main+0x3e>
        else i--;
  400510:       83 6d fc 01             sub    DWORD PTR [rbp-0x4],0x1

        switch(i){
```

Information Flow and Program Slicing

- *Program slicing is a method used by experienced computer programmers for abstracting from programs. Starting from a subset of a program's behavior, slicing reduces that program to a minimal form which still produces that behavior. The reduced program, called a "slice", is an independent program guaranteed to faithfully represent the original program within the domain of the specified subset of behavior.* [PROGRAM SLICING, Mark Weiser]
- ***program slicing*** *is the computation of the set of program statements, the* ***program slice****, that may affect the values at some point of interest.* [Wikipedia]

Program Slicing Example

```
1    BEGIN
2    READ(X,Y)
3    TOTAL := 0.0
4    SUM := 0.0
5    IF X<=1
6        THEN SUM := Y
7        ELSE BEGIN
8            READ(Z)
9            TOTAL := X*Y
10   END
11   WRITE(TOTAL,SUM)
12   END.
```

Slice on Z at 12

```
BEGIN
READ(X,Y)
IF X < 1
        THEN
        ELSE READ(Z)
END.
```

Slice on X at 9

```
BEGIN
READ(X,Y)
END
```

Slicing in Reverse Engineering

- Focus only on a subset of the code
  - Identify a point of interest of the reverser
  - Go backward to retrieve the slice on the given PoI
  - Identify the point where to operate

Lab 1

Skip controls…

- understanding a program control flow

- Introduction to dynamic code analysis

The binary file: register

- Let's try to execute

```
$ ./register
Usage: register <activation code>
```

- Let's try a random input

```
$ ./register AAAA
Given: AAAA
Sorry, the given code is not valid!
```

“register” X-ray

- objdump –d register
  - gdb register
  - set disassembly-flavor intel
  - disassemble main

Building the “register” Control Flow

```
0000000004005b6 <main>:
  4005b6:       55                      push   %rbp
  4005b7:       48 89 e5                mov    %rsp,%rbp
  4005ba:       48 83 ec 10             sub    $0x10,%rsp
  4005be:       89 7d fc                mov    %edi,-0x4(%rbp)
  4005c1:       48 89 75 f0             mov    %rsi,-0x10(%rbp)
  4005c5:       83 7d fc 02             cmpl   $0x2,-0x4(%rbp)
  4005c9:       75 51                   jne    40061c <main+0x66>
  4005cb:       48 8b 45 f0             mov    -0x10(%rbp),%rax
  4005cf:       48 83 c0 08             add    $0x8,%rax
  4005d3:       48 8b 00                mov    (%rax),%rax
  4005d6:       48 89 c6                mov    %rax,%rsi
  4005d9:       bf b8 06 40 00          mov    $0x4006b8,%edi
  4005de:       b8 00 00 00 00          mov    $0x0,%eax
  4005e3:       e8 98 fe ff ff          callq  400480 <printf@plt>
  4005e8:       48 8b 45 f0             mov    -0x10(%rbp),%rax
  4005ec:       48 83 c0 08             add    $0x8,%rax
  4005f0:       48 8b 00                mov    (%rax),%rax
  4005f3:       be c3 06 40 00          mov    $0x4006c3,%esi
  4005f8:       48 89 c7                mov    %rax,%rdi
  4005fb:       e8 a0 fe ff ff          callq  4004a0 <strcmp@plt>
  400600:       85 c0                   test   %eax,%eax
  400602:       75 0c                   jne    400610 <main+0x5a>
  400604:       bf d5 06 40 00          mov    $0x4006d5,%edi
  400609:       e8 62 fe ff ff          callq  400470 <puts@plt>
  40060e:       eb 16                   jmp    400626 <main+0x70>
  400610:       bf e8 06 40 00          mov    $0x4006e8,%edi
  400615:       e8 56 fe ff ff          callq  400470 <puts@plt>
  40061a:       eb 0a                   jmp    400626 <main+0x70>
  40061c:       bf 10 07 40 00          mov    $0x400710,%edi
  400621:       e8 4a fe ff ff          callq  400470 <puts@plt>
  400626:       b8 00 00 00 00          mov    $0x0,%eax
  40062b:       c9                      leaveq
  40062c:       c3                      retq
  40062d:       0f 1f 00                nopl   (%rax)
```

Let's go Dynamic

- Let's start debugging: `gdb register`
- Go to first program instruction: `start`
- Execute instructions one by one: `si`

```
   0x4005c5 <main+15>:  cmp    DWORD PTR [rbp-0x4],0x2
=> 0x4005c9 <main+19>:  jne    0x40061c <main+102>
 | 0x4005cb <main+21>:  mov    rax,QWORD PTR [rbp-0x10]
 | 0x4005cf <main+25>:  add    rax,0x8
 | 0x4005d3 <main+29>:  mov    rax,QWORD PTR [rax]
 | 0x4005d6 <main+32>:  mov    rsi,rax
 |->   0x40061c <main+102>:    mov    edi,0x400710
       0x400621 <main+107>:    call   0x400470 <puts@plt>
       0x400626 <main+112>:    mov    eax,0x0
       0x40062b <main+117>:    leave
```

- There is a comparison of something with 0x2 if not equal a jump to a print and then exit
  - Let's execute (`si` … until the print and execute the print with a `next`)

```
gdb-peda$ next
Usage: register <activation code>
```

Building the “register” Control Flow

```
0000000004005b6 <main>:
  4005b6:       55                      push   %rbp
  4005b7:       48 89 e5                mov    %rsp,%rbp
  4005ba:       48 83 ec 10             sub    $0x10,%rsp
  4005be:       89 7d fc                mov    %edi,-0x4(%rbp)
  4005c1:       48 89 75 f0             mov    %rsi,-0x10(%rbp)
  4005c5:       83 7d fc 02             cmpl   $0x2,-0x4(%rbp)
  4005c9:       75 51                   jne    40061c <main+0x66>
  4005cb:       48 8b 45 f0             mov    -0x10(%rbp),%rax
  4005cf:       48 83 c0 08             add    $0x8,%rax
  4005d3:       48 8b 00                mov    (%rax),%rax
  4005d6:       48 89 c6                mov    %rax,%rsi
  4005d9:       bf b8 06 40 00          mov    $0x4006b8,%edi
  4005de:       b8 00 00 00 00          mov    $0x0,%eax
  4005e3:       e8 98 fe ff ff          callq  400480 <printf@plt>
  4005e8:       48 8b 45 f0             mov    -0x10(%rbp),%rax
  4005ec:       48 83 c0 08             add    $0x8,%rax
  4005f0:       48 8b 00                mov    (%rax),%rax
  4005f3:       be c3 06 40 00          mov    $0x4006c3,%esi
  4005f8:       48 89 c7                mov    %rax,%rdi
  4005fb:       e8 a0 fe ff ff          callq  4004a0 <strcmp@plt>
  400600:       85 c0                   test   %eax,%eax
  400602:       75 0c                   jne    400610 <main+0x5a>
  400604:       bf d5 06 40 00          mov    $0x4006d5,%edi
  400609:       e8 62 fe ff ff          callq  400470 <puts@plt>
  40060e:       eb 16                   jmp    400626 <main+0x70>
  400610:       bf e8 06 40 00          mov    $0x4006e8,%edi
  400615:       e8 56 fe ff ff          callq  400470 <puts@plt>
  40061a:       eb 0a                   jmp    400626 <main+0x70>
  40061c:       bf 10 07 40 00          mov    $0x400710,%edi
  400621:       e8 4a fe ff ff          callq  400470 <puts@plt>
  400626:       b8 00 00 00 00          mov    $0x0,%eax
  40062b:       c9                      leaveq
  40062c:       c3                      retq
  40062d:       0f 1f 00                nopl   (%rax)
```

Let's try a random code

- Restart again the program providing an input:

```
start AAA-BBB-CCC
```

- Let's go ahead until an output

```
   0x4005d9 <main+35>:  mov    edi,0x4006b8
   0x4005de <main+40>:  mov    eax,0x0
   0x4005e3 <main+45>:  call   0x400480 <printf@plt>
=> 0x4005e8 <main+50>:  mov    rax,QWORD PTR [rbp-0x10]
   0x4005ec <main+54>:  add    rax,0x8
   0x4005f0 <main+58>:  mov    rax,QWORD PTR [rax]
   0x4005f3 <main+61>:  mov    esi,0x4006c3
   0x4005f8 <main+66>:  mov    rdi,rax
```

```
gdb-peda$ next
Given: AAA-BBB-CCC
```

Let's try a random code

- After a comparison there is a jump on not equal to a print and then exit

```
   0x4005f8 <main+66>:  mov    rdi,rax
   0x4005fb <main+69>:  call   0x4004a0 <strcmp@plt>
   0x400600 <main+74>:  test   eax,eax
=> 0x400602 <main+76>:  jne    0x400610 <main+90>
 | 0x400604 <main+78>:  mov    edi,0x4006d5
 | 0x400609 <main+83>:  call   0x400470 <puts@plt>
 | 0x40060e <main+88>:  jmp    0x400626 <main+112>
 | 0x400610 <main+90>:  mov    edi,0x4006e8
 |->   0x400610 <main+90>:      mov    edi,0x4006e8
       0x400615 <main+95>:      call   0x400470 <puts@plt>
       0x40061a <main+100>:     jmp    0x400626 <main+112>
       0x40061c <main+102>:     mov    edi,0x400710
```

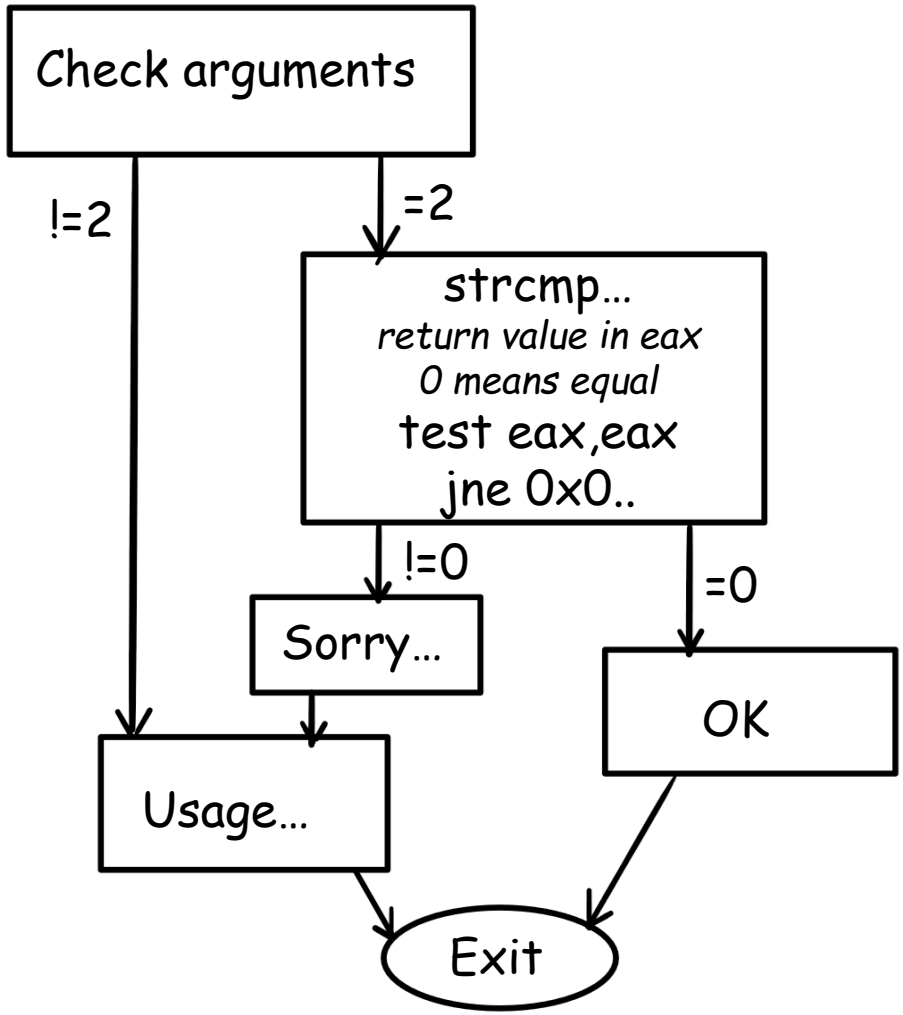
JUMP is taken

Building the “register” Control Flow

```
0000000004005b6 <main>:
  4005b6:       55                      push   %rbp
  4005b7:       48 89 e5                mov    %rsp,%rbp
  4005ba:       48 83 ec 10             sub    $0x10,%rsp
  4005be:       89 7d fc                mov    %edi,-0x4(%rbp)
  4005c1:       48 89 75 f0             mov    %rsi,-0x10(%rbp)
  4005c5:       83 7d fc 02             cmpl   $0x2,-0x4(%rbp)
  4005c9:       75 51                   jne    40061c <main+0x66>
  4005cb:       48 8b 45 f0             mov    -0x10(%rbp),%rax
  4005cf:       48 83 c0 08             add    $0x8,%rax
  4005d3:       48 8b 00                mov    (%rax),%rax
  4005d6:       48 89 c6                mov    %rax,%rsi
  4005d9:       bf b8 06 40 00          mov    $0x4006b8,%edi
  4005de:       b8 00 00 00 00          mov    $0x0,%eax
  4005e3:       e8 98 fe ff ff          callq  400480 <printf@plt>
  4005e8:       48 8b 45 f0             mov    -0x10(%rbp),%rax
  4005ec:       48 83 c0 08             add    $0x8,%rax
  4005f0:       48 8b 00                mov    (%rax),%rax
  4005f3:       be c3 06 40 00          mov    $0x4006c3,%esi
  4005f8:       48 89 c7                mov    %rax,%rdi
  4005fb:       e8 a0 fe ff ff          callq  4004a0 <strcmp@plt>
  400600:       85 c0                   test   %eax,%eax
  400602:       75 0c                   jne    400610 <main+0x5a>
  400604:       bf d5 06 40 00          mov    $0x4006d5,%edi
  400609:       e8 62 fe ff ff          callq  400470 <puts@plt>
  40060e:       eb 16                   jmp    400626 <main+0x70>
  400610:       bf e8 06 40 00          mov    $0x4006e8,%edi
  400615:       e8 56 fe ff ff          callq  400470 <puts@plt>
  40061a:       eb 0a                   jmp    400626 <main+0x70>
  40061c:       bf 10 07 40 00          mov    $0x400710,%edi
  400621:       e8 4a fe ff ff          callq  400470 <puts@plt>
  400626:       b8 00 00 00 00          mov    $0x0,%eax
  40062b:       c9                      leaveq
  40062c:       c3                      retq
  40062d:       0f 1f 00                nopl   (%rax)
```

Let's Bypass the Control…method 1

- We can break execution just after the strcmp
- Just change the result in eax before the test



Let's Bypass the Control…method 1

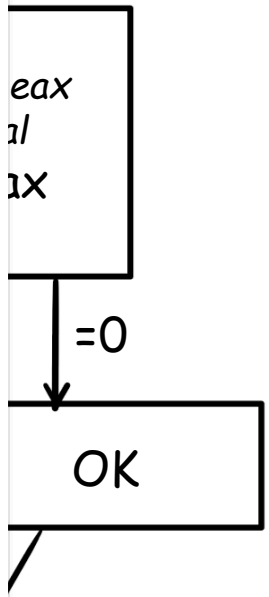
- We can
  after th
- Just cha
  before t

```
gdb-peda$ break *0x0000000000400600
Breakpoint 4 at 0x400600
gdb-peda$ continue
Continuing.
Given: AAA-BBB-CCC
```

```
   0x4005f8 <main+66>:  mov    rdi,rax
   0x4005fb <main+69>:  call   0x4004a0 <strcmp@plt>
=> 0x400600 <main+74>:  test   eax,eax
   0x400602 <main+76>:  jne    0x400610 <main+90>
   0x400604 <main+78>:  mov    edi,0x4006d5
```

```
Breakpoint 4, 0x0000000000400600 in main ()
gdb-peda$ set $eax=0
gdb-peda$ ni
```

```
gdb-peda$
License Activated!
```



Alternative method: 2

- Break @strcmp
- strcmp has to compare the provided string against the valid code
- The two arguments are provided as pointer to the strings
  - Throughout registers RSI, RDI
- Let's check the memory at the two addresses…

```
Breakpoint 2, 0x00000000004005fb in main ()
gdb-peda$ x/s $rsi
0x4006c3:       "LUIGI-COPPO-77-03"
gdb-peda$ x/s $rdi
0x7fffffffee681: "AAA-BBB-CCC"
gdb-peda$
```

More alternatives

1. $ strings register

```
colui@COLUI-SURFACE:/mnt/c/Users/colui/OneDrive - uniparthenope.it/teaching/SSI/labs$ strings register
/lib64/ld-linux-x86-64.so.2
libc.so.6
puts
printf
strcmp
__libc_start_main
__gmon_start__
GLIBC_2.2.5
UH-H
AWAVA
AUATL
[]A\A]A^A_
Given: %s
LUIGI-COPPO-77-03
License Activated!
Sorry, the given code is not valid!
Usage: register <activation code>
;*3$"
GCC: (Ubuntu 5.4.0-6ubuntu1~16.04.9) 5.4.0 20160609
```

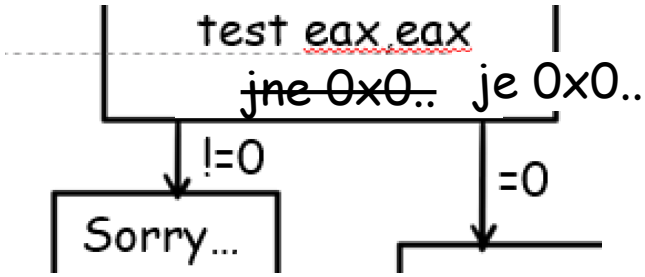
2. $ readelf -x .rodata register

```
Hex dump of section '.rodata':
  0x004006b0 01000200 00000000 47697665 6e3a2025 ........Given: %
  0x004006c0 730a004c 55494749 2d434f50 504f2d37 s..LUIGI-COPPO-7
  0x004006d0 372d3033 004c6963 656e7365 20416374 7-03.License Act
  0x004006e0 69766174 65642100 536f7272 792c2074 ivated!.Sorry, t
  0x004006f0 68652067 6976656e 20636f64 65206973 he given code is
  0x00400700 206e6f74 2076616c 69642100 00000000  not valid!.....
  0x00400710 55736167 653a2072 65676973 74657220 Usage: register
  0x00400720 3c616374 69766174 696f6e20 636f6465 <activation code
  0x00400730 3e00                                >.
```

Binary Patching

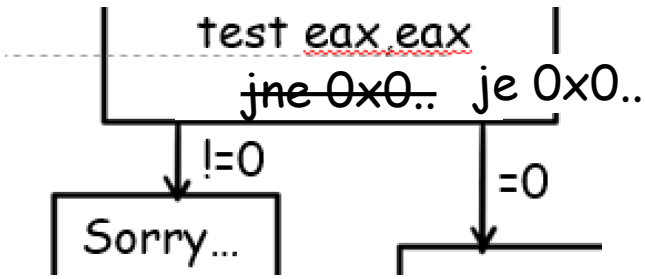
Flipping the controlflow

- Inverting the control flow statement condition would make it …
  - JNE->JE



Flipping the controlflow

- Inverting the control flow statement condition would make it …
  - JNE->JE
- Only 1 bit of distance…
  - Short Jump Opcode
    - JE= 0x74 JNE=0x75
    - JB=0x72 JNB=0x73
  - Near Jump Opcode
    - JNE=0F 84 JE=0F 85
    - …



Other useful opcode

- Relative Jump: EB xx
  - If xx in 0x00-0x7f => jump forward (2+xx byte)
  - If xx in 0x80-0xff => jump backward (2-2's cmpl xx byte)
- 0x90 (nop) : do nothing
- 0xC3 (ret) : return from current function
- 0xCC (int3) : trigger a sw breakpoint …more on this next classes

Patching

- Retrieve the control flow…
- Identify the point were to alterate the control flow
- Identify the control flow statement to change
- Patch it …

We need a tool…

plenty of them binary editors/hex editors (gdb –write; HxD: register_fix)

register_fix | register

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0
00000600	85	C0	75	0C	BF	D5	06	40	00	E8	62	F
00000610	BF	E8	06	40	00	E8	56	FE	FF	FF	EB	0
00000620	00	E8	4A	FE	FF	FF	B8	00	00	00	00	C

register_fix | register

Offset(h)	00	01	02	03	04	05	06	07	08	0
00000600	85	C0	74	0C	BF	D5	06	40	00	E
00000610	BF	E8	06	40	00	E8	56	FE	FF	F
00000620	00	E8	4A	FE	FF	FF	B8	00	00	0

$ ./register_fix AAA
Given: AAA
License Activated!

Lab 2

Fooling registerPlus binary file

- *Introduction to radare2*
- *Patching a binary file*

Installing radare2

- radare2 is a complete suite for reverse engineering completely open source and free
  - git clone https://github.com/radare/radare2
    cd radare2
    ./sys/install.sh
- It is an open source alternative to expensive alternatives such as IDA (Windows) or Hopper Disassembler (MacOS)

A Brief Intro to Radare2

- `r2 ./register` starts reversing the a.out binary
- "?" Allows to view a list of classes of commands
  - For every class typing its letter and ? Gives the list of subcommands

```
[0x004004c0]> ?
Usage: [.][times][cmd][~grep][@[@iter]addr!size]
Append '?' to any char command to get detailed H
Prefix with number to repeat command N times (f.
|%var =valueAlias for 'env' command
| *[?] off[=[0x]value]    Pointer read/write dat
| (macro arg0 arg1)       Manage scripting macro
| .[?] [-|(m)|f|!sh|cmd]  Define macro or load r
| =[?] [cmd]              Send/Listen for Remote
| <[...]                  Push escaped string ir
| /[?]                    Search for bytes, rege
| ![?] [cmd]              Run given command as i
| #[?] !lang [..]         Hashbang to run an rla
| a[?]                    Analysis commands
| b[?]                    Display or change the
```

```
[0x004004c0]> a?
|Usage: a[abdefFghoprxstc] [...]
| aa[?]             analyze all (fcns + bbs)
| a8 [hexpairs]     analyze bytes
| abb [len]         analyze N basic blocks i
| ac [cycles]       analyze which op could b
| ad[?]             analyze data trampoline
| ad [from] [to]    analyze data pointers to
| ae[?] [expr]      analyze opcode eval expr
| af[?]             analyze Functions
```

disassembling

- `aaa` (analyze all functions and symbols)
- `afl` (list functions)

```
[0x004004c0]> afl
0x00400438    3 26           sym._init
0x00400470    1 6            sym.imp.puts
0x00400480    1 6            sym.imp.printf
0x00400490    1 6            sym.imp.__libc_start_main
0x004004a0    1 6            sym.imp.strcmp
0x004004b0    1 6            sub.__gmon_start_4b0
0x004004c0    1 41           entry0
0x004004f0    4 50   -> 41    sym.deregister_tm_clones
0x00400530    3 53           sym.register_tm_clones
0x00400570    3 28           sym.__do_global_dtors_aux
0x00400590    4 38   -> 35    entry1.init
0x004005b6    6 119          main
0x00400630    4 101          sym.__libc_csu_init
```

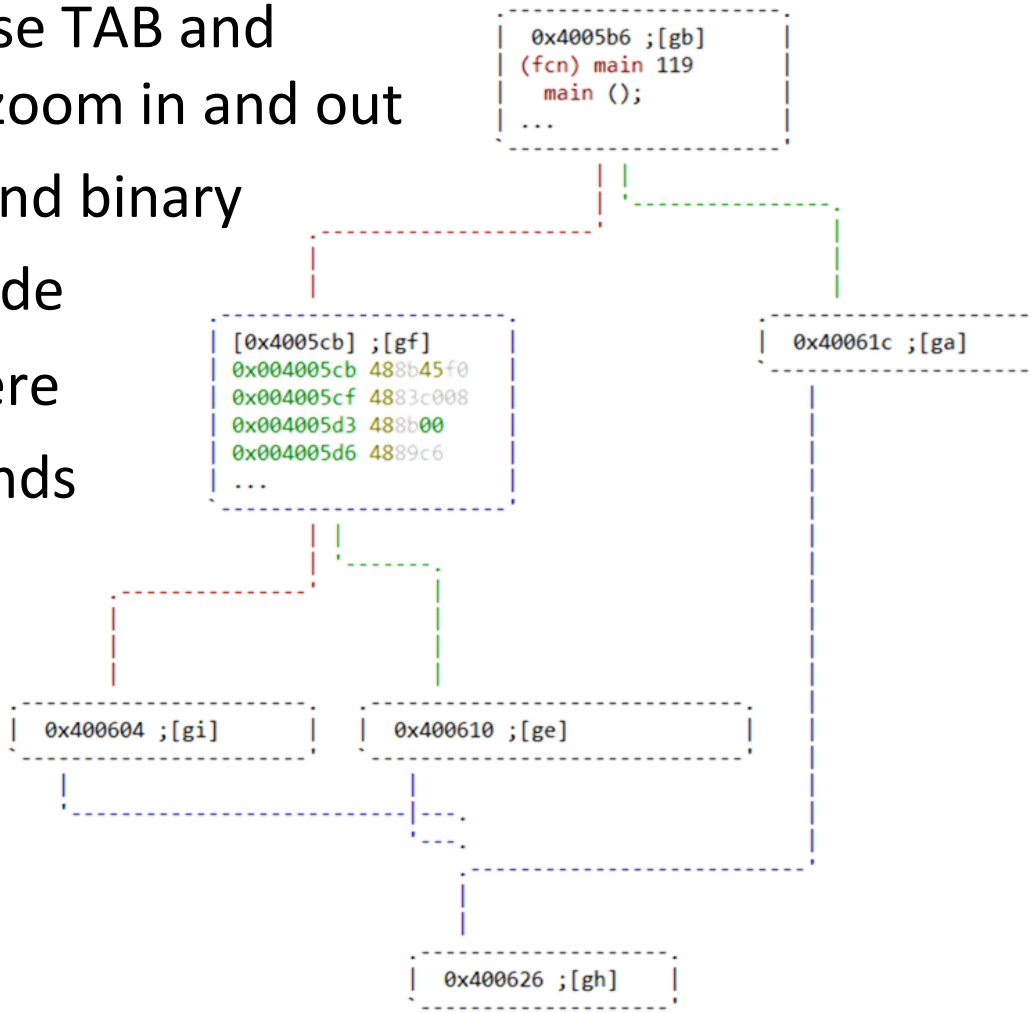
- `pdf` *`main`* to “print disassembled function” main
- `s` *`main`* to seek @ main

debugging

- `ood` to reopen in debug mode
- `d?` for debugging commands
- `db` *`ADDRESS`* places a breakpoint @ADDRESS
- `dc` continue to the breakpoint

Control Flow: Visual Mode

- Switch to visual mode: `VV` use TAB and arrows to navigate; `+/-` to zoom in and out
- press `p` to show addresses and binary
- `q` to return to non visual mode
- Note that in visual mode there is a different set of commands (try ?)
  - normal commands can still be used after a ":«
    - `:ood AAA` reloads with argument AAA
  - `s/S` step into/out

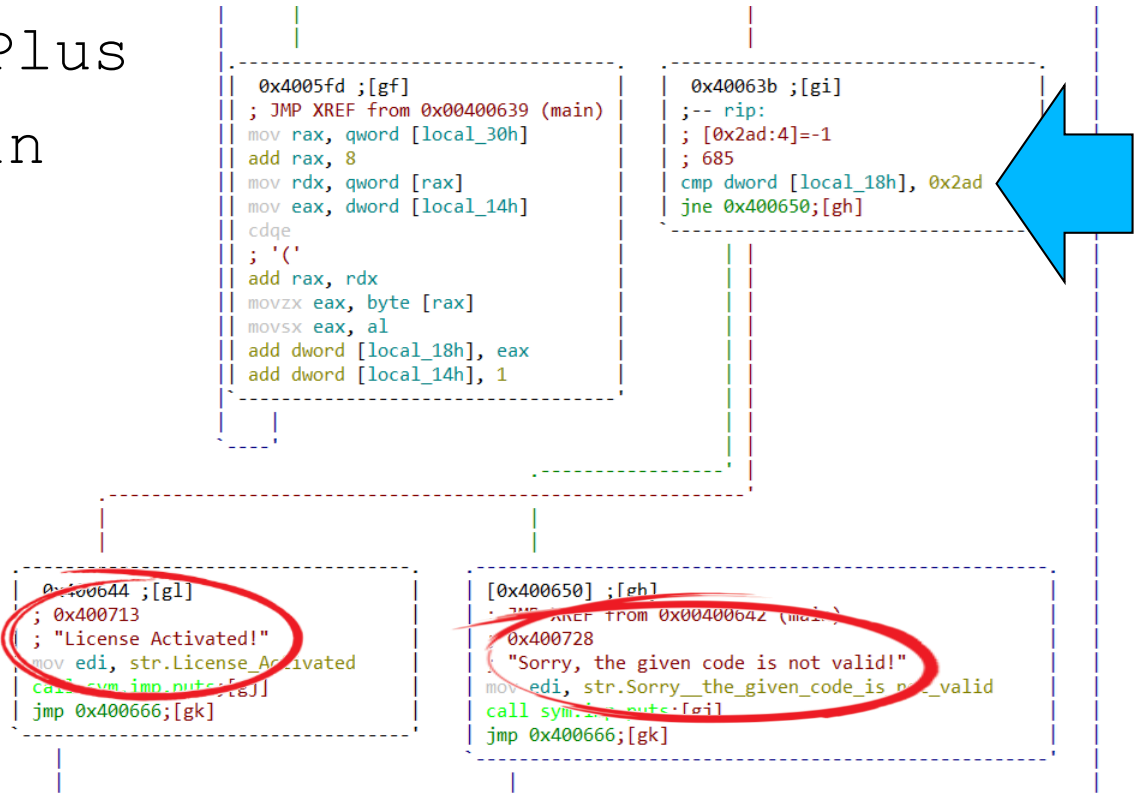


registerPlus

- 10 minutes to bypass the control yourself
- … at least describe the approach you would like to follow

Analyzing the ./registerPlus control flow

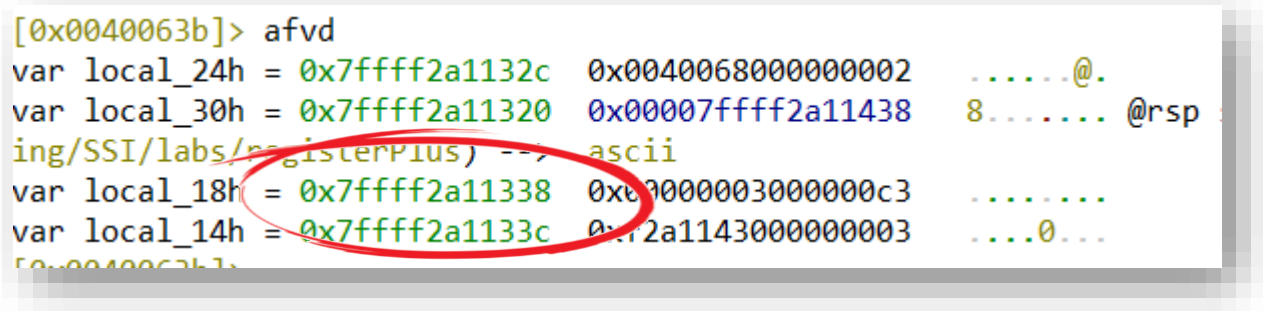
- `r2 ./registerPlus`
- `aaa/afl/s main`
- `VV /p`
- `:ood AAAA`
- `db 0x0…` (cmp)
- `dc`
- `q` -> rip@cmp



- The cmp compares the memory @local_18h with the immediate 0x2ad

Solution 1: set the control variable to the expected value

- `afvd` (analyze function variables and display)
  - lists the variables in the function



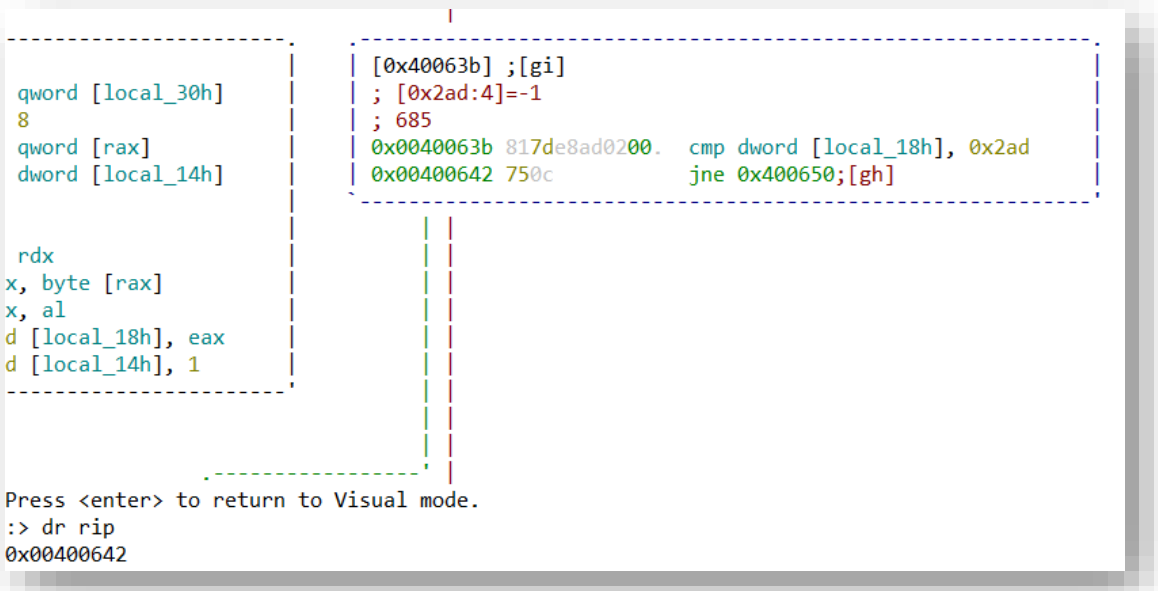
- `wv 0x2ad @0x7ffff2a11338` (write variable)
- check the results with afvd again
- `dc` to the success…

Solution 2: jump to activation

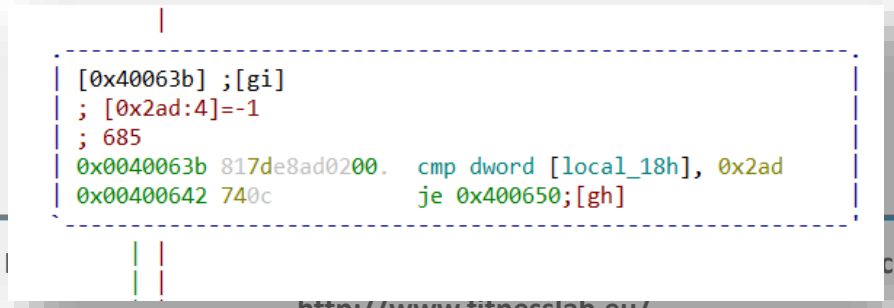
- go to the jne …
- `dr rip` (check the value of rip register)
- `dr rip=…` (set rip with the address of the activation branch)

Solution 3: invert the control flow statement

- go to the jump condition
- `dr rip` (show rip register pointing at the: jne 0x400650)



- `:wa je 0x400650 @0x00400642` (write opcode) and finally `dc` …



Solution 4: patch registerPlus

- `cp registerPlus registerPlus_fix` (make a copy)
- `r2 ./registerPlus_fix`
- `aaa /afl/s main`
- `pdf` and identify the jne address

```
|   0x00400639      72c2           jb 0x4005fd
|   0x0040063b      817de8ad0200.  cmp dword [local_18h], 0x2ad ; [0x2ad:4]=-1 ; 685
,==< 0x00400642      750c           jne 0x400650
||  0x00400644      bf13074000     mov edi, str.License_Activated ; 0x400713 ; "License Activated!"
||  0x00400649      e822feffff     call sym.imp.puts           ; int puts(const char *s)
```

- `s 0x00400642` (seek to the jne)
- `Vp` to switch to visual mode
- `oo+` to reload the current file in read-write mode
- `A` to invoke the Awasome assembly editor…write the new line «`je 0x400650`» and double click enter…`q` to quit the visual mode and `q` to quit r2…
- try to execute ./registerPlus_fix AAAA

References

- http://www.skyfree.org/linux/references/ELF_Format.pdf
- http://delivery.acm.org/10.1145/810000/802557/p439-weiser.pdf?ip=192.167.9.86&id=802557&acc=ACTIVE%20SERVICE&key=296E2ED678667973%2E3FE349642144B6A6%2E4D4702B0C3E38B35%2E4D4702B0C3E38B35&__acm__=1540198716_f46c5ae91721a10aef961a49b5fd85b8
- https://radare.gitbooks.io/radare2book/content/