Labs 5 to 6. Reverse Engineering

Dan Bruce, David Clark and Hector D. Menendez

Department of Computer Science
University College London

October 30, 2017

Modified from Xeno Kovah slides of Open Security Training

Introduction

This Lectures aims to teach a practical tool of Reverse Engineering.

Reverse Engineering

Processes of extracting knowledge or design information from anything man-made and re-producing it or reproducing anything based on the extracted information.

Reverse Engineering in Software

Three different possibilities:

- Communication sniffers.
- Disassembly.
- Decompilation.

Tools for Reverse Engineering

Radare: Portable reversing framework for Windows, Linux or Mac OS.

Olly debugger: 32-bit assembler level analysing debugger for Microsoft Windows.

IDAPro: Windows, Linux or Mac OS X hosted multi-processor disassembler and debugger that offers so many RE features.

Radare Features

Disassemble (and assemble for) many different architectures.

Debug with local native and remote debuggers (gdb, rap, webui, r2pipe, winedbg, windbg).

Run on Linux, *BSD, Windows, OSX, Android, iOS, Solaris and Haiku.

Perform forensics on filesystems and data carving.

Be scripted in Python, Javascript, Go and more.

Support collaborative analysis using the embedded webserver.

Visualize data structures of several file types.

Patch programs to uncover new features or fix vulnerabilities.

Use powerful analysis capabilities to speed up reversing.

Aid in software exploitation.

Olly Features

Intuitive user interface, no cryptical commands.

Code analysis - traces registers, recognizes procedures, loops, API calls, switches, tables, constants and strings.

Directly loads and debugs DLLs.

Object file scanning - locates routines from object files and libraries.

Allows for user-defined labels, comments and function descriptions.

Understands debugging information in Borland format.

Saves patches between sessions, writes them back to executable file and updates fixups.

Open architecture - many third-party plugins are available.

No installation - no trash in registry or system directories.

Debugs multithread applications.

Radare2

Reverse Engineering Framework.

Library.

Shell commands.

Accessible from JS/Python/Ruby.

More than 8 years working.

Applications

Malware Classification.

Drivers analysis.

Vulnerabilities Detection.

Generate cracks.

Industrial espionage.

Computer Forensics.

Internal tools

Multi-platform Debugger.

Multi-platform disassembler.

Binary Parsers.

Code Analyser.

Web interfaces.

Advantages

Open source.

Modular.

Active Community.

Under Development.

Multi-language.

Installing Radare

From git:

`https://github.com/radare/radare2`

Install it:

```
cd radare2
sh sys/install.sh
```

Starting with Radare

To start Radare just go to a shell and run:

```
> r2 file.bin
```

Basic Commands

s: move.

px: hexdump.

aa: analyse.

pdf: disassemble.

afl: function list.

ii: imports.

iz: strings.

Commands Logic

Each command is related to a letter, therefore the following commands are related:

px: print hex.

pd: print disassembly.

w: write.

q: quit.

s: seek.

i: info.

Prefix

?: conditional.

!: run shell command.

#: comment.

#!: hashbang.

.: interpreter.

(: macro.

$: alias.

”: quote?.

`: evaluation and insertion.

Suffix

~: internal grep/cut.

|: pipe to process.

>: pipe to file.

>>: cat to file.

@: temporal seek.

@@: iterator.

*: command output.

j: json output.

?: help.

Help Menu

?: main help.

suffix ?: command help.

???.

?@?.

?$?.

First Step

Get as much information of the binary as possible:

- File type
- Libraries
- Entrypoints
- Imports
- Symbols
- Strings

Rabin2

This tool analyse the file and provide information about it:

- Multi-platform
- Different output format.
- Allows tiny-bins
- API access.

Code Analysis

Describe Opcodes.

Pseudo Disassembly.

Flags.

Convert.

Analyse functions.

Graphs

They are helpful for analysing the code flow

- Graphviz
- Ascii ART
- Web Interface

Debugger

Run

$> r2 -d

Stepping / Tracing / Breakpoints / Continue.

Syscalls / signals / maps.

Emulator

Allows the emulation of specific code fragments.

Describe each instruction with text.

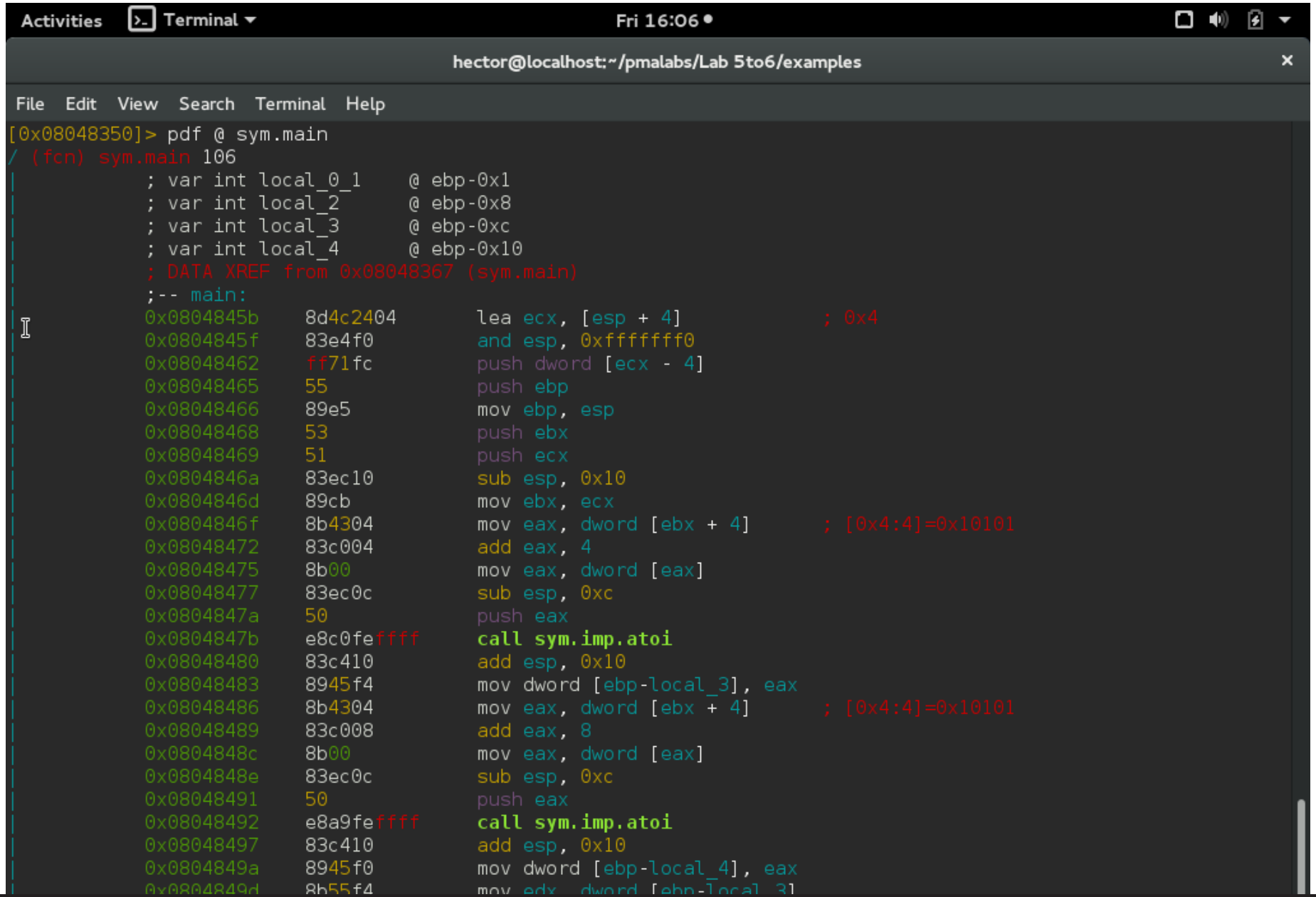
Example 1

```
1  #include<stdio.h>
2  #include<string.h>
3  #include<stdlib.h>
4  int main(int argc, char * argv[])
5  {
6
7      int a,b;
8      a=atoi(argv[1]);
9      b=atoi(argv[2]);
10     printf("A+B=%d",a+b);
11 }
```

Example 1

Introduce this in Radare for analysing and showing the disassembly code:

```
> aa
> pdf
> pdf @ sym.main
> iz
```


Activities
Terminal
Fri 16:06
hector@localhost:~/pmalabs/Lab 5to6/examples
File Edit View Search Terminal Help
[0x08048350]> pdf @ sym.main
/ (fcn) sym.main 106
| ; var int local_0_1 @ ebp-0x1
| ; var int local_2 @ ebp-0x8
| ; var int local_3 @ ebp-0xc
| ; var int local_4 @ ebp-0x10
| ; DATA XREF from 0x08048367 (sym.main)
| ;-- main:
| 0x0804845b 8d4c2404 lea ecx, [esp + 4] ; 0x4
| 0x0804845f 83e4f0 and esp, 0xfffffff0
| 0x08048462 ff71fc push dword [ecx - 4]
| 0x08048465 55 push ebp
| 0x08048466 89e5 mov ebp, esp
| 0x08048468 53 push ebx
| 0x08048469 51 push ecx
| 0x0804846a 83ec10 sub esp, 0x10
| 0x0804846d 89cb mov ebx, ecx
| 0x0804846f 8b4304 mov eax, dword [ebx + 4] ; [0x4:4]=0x10101
| 0x08048472 83c004 add eax, 4
| 0x08048475 8b00 mov eax, dword [eax]
| 0x08048477 83ec0c sub esp, 0xc
| 0x0804847a 50 push eax
| 0x0804847b e8c0feffff call sym.imp.atoi
| 0x08048480 83c410 add esp, 0x10
| 0x08048483 8945f4 mov dword [ebp-local_3], eax
| 0x08048486 8b4304 mov eax, dword [ebx + 4] ; [0x4:4]=0x10101
| 0x08048489 83c008 add eax, 8
| 0x0804848c 8b00 mov eax, dword [eax]
| 0x0804848e 83ec0c sub esp, 0xc
| 0x08048491 50 push eax
| 0x08048492 e8a9feffff call sym.imp.atoi
| 0x08048497 83c410 add esp, 0x10
| 0x0804849a 8945f0 mov dword [ebp-local_4], eax

Example 2

```
 1  #include<stdio.h>
 2  #include<string.h>
 3  #include<stdlib.h>
 4  int add(int a, int b)
 5  {
 6      return a+b;
 7  }
 8
 9  int main(int argc, char * argv[])
10  {
11      int a,b;
12      a=atoi(argv[1]);
13      b=atoi(argv[2]);
14      printf("A+B=%d",add(a,b));
15  }
```

Example 2

Introduce this in Radare for analysing and showing the disassembly code:

```
> aa
> pdf
> pdf @ sym.main
> pdf @ sym.add
```

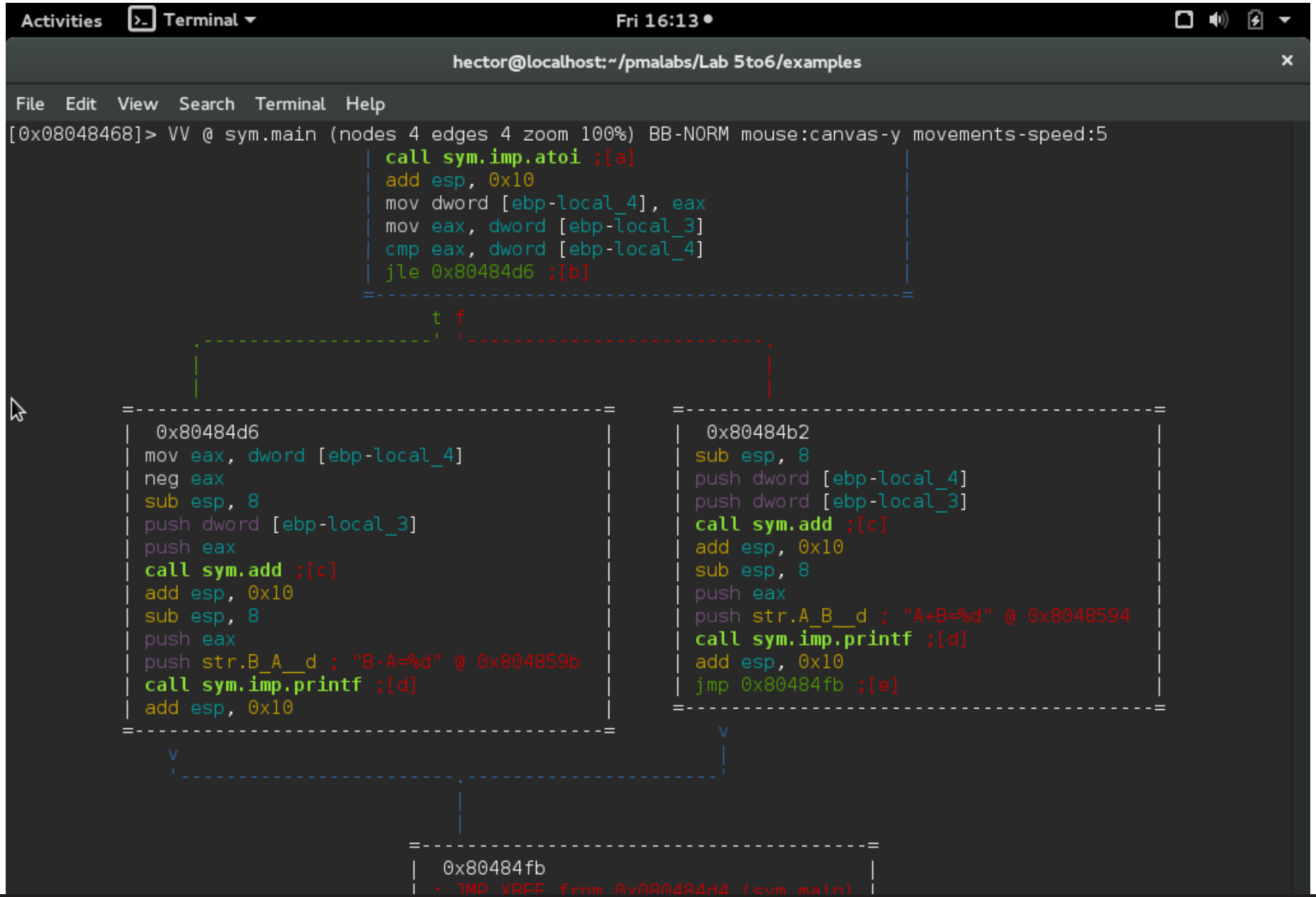
Example 3

```
1  #include<stdio.h>
2  #include<string.h>
3  #include<stdlib.h>
4  int add(int a, int b)
5  {
6      return a+b;
7  }
8  int main(int argc, char * argv[])
9  {
10     int a,b;
11     a=atoi(argv[1]);
12     b=atoi(argv[2]);
13     if(a > b)
14       printf("A+B=%d",add(a,b));
15     else
16       printf("B-A=%d",add(-a,b));
17 }
```

Example 3

Introduce this in Radare for analysing and showing the disassembly code:

```
> aa
> pdf
> pdf @ sym.main
> pdf @ sym.add
> V
>> q
> V @ sym.main
>> V
```


Activities
Terminal
Fri 16:13
hector@localhost:~/pmalabs/Lab 5to6/examples
File Edit View Search Terminal Help
[0x08048468]> VV @ sym.main (nodes 4 edges 4 zoom 100%) BB-NORM mouse:canvas-y movements-speed:5
call sym.imp.atoi ;[a]
add esp, 0x10
mov dword [ebp-local_4], eax
mov eax, dword [ebp-local_3]
cmp eax, dword [ebp-local_4]
jle 0x80484d6 ;[b]
t f
0x80484d6
mov eax, dword [ebp-local_4]
neg eax
sub esp, 8
push dword [ebp-local_3]
push eax
call sym.add ;[c]
add esp, 0x10
sub esp, 8
push eax
push str.B_A__d ; "B-A=%d" @ 0x804859b
call sym.imp.printf ;[d]
add esp, 0x10
0x80484b2
sub esp, 8
push dword [ebp-local_4]
push dword [ebp-local_3]
call sym.add ;[c]
add esp, 0x10
sub esp, 8
push eax
push str.A_B__d ; "A+B=%d" @ 0x8048594
call sym.imp.printf ;[d]
add esp, 0x10
jmp 0x80484fb ;[e]
0x80484fb

Exercises

Time to Crack files:

You have to find the passwords of the following files using Radare2:

- Crack0
- Crack1

Explain the solutions.