U23 2016 - Reverse Engineering

Andy

andy@koeln.ccc.de

November 15, 2016

Introduction
- Static program analysis
- Dynamic program analysis

Tools
- strings
- objdump
- IDA
- Hopper
- gdb

Live Reversing

Exercises

Section 1

Introduction

Introduction

- **Reverse Engineering** is the process of analyzing a design to figure out exactly how it works and re-produce the analyzed design as a 1:1 copy - not only a functional copy which behaves like the original but is designed exactly like the original
- Not only applicable to software. Also all kinds of hardware: Chips, PCBs, Trains, Planes, Cars etc. Also maybe only parts of a whole system like the engine, suspension, certain control mechanisms etc.

Introduction

- **Reverse Engineering** is the process of analyzing a design to figure out exactly how it works and re-produce the analyzed design as a 1:1 copy - not only a functional copy which behaves like the original but is designed exactly like the original
- Not only applicable to software. Also all kinds of hardware: Chips, PCBs, Trains, Planes, Cars etc. Also maybe only parts of a whole system like the engine, suspension, certain control mechanisms etc.
- We are only interested in software here

When do we reverse engineer software?

- Malware analysis
- Lost source code of an original product
- Compatibility of file formats / network protocols
- Security analysis
- Debugging
- Curiosity (not 100% legal, but most of the time this can be twisted to be about security or compatibility - situation between EU and US is different!)

Static program analysis

- Look at the code, but don't execute it
- This is what you do when you disassemble a program and stare at the code
- Also possible on source code, but not important here

Dynamic program analysis

- Execute the code you want to analyze
- Instrument it while it's being executed
- Figure out what's going on either automatically using a tool or by hand
- This is what you do if you debug a program using gdb/MSVC or run it in an emulator with augmentation capabilities (Unicorn, PIN etc.)

Section 2

Tools

strings

- Easiest tool ever
- Just dumps all printable character sequences longer than n characters (default is 4)

Example:

```
$ strings test
/lib64/ld-linux-x86-64.so.2
libc.so.6
puts
__libc_start_main
__gmon_start__
GLIBC_2.2.5
UH-0
AWAVA
AUATL
[]A\A]A^A_
Hello, World!
;*3$"
GCC:␣(GNU)␣6.1.1␣20160802
[...]
```

objdump

- Very simple disassembler
- Installed everywhere because it's part of `binutils`
- `-d` disassembles a binary

Example:

```
$ objdump -d test
[...]
0804840b <main>:
 804840b:       8d 4c 24 04             lea    0x4(%esp),%ecx
 804840f:       83 e4 f0                and    $0xfffffff0,%esp
 8048412:       ff 71 fc                pushl  -0x4(%ecx)
 8048415:       55                      push   %ebp
 8048416:       89 e5                   mov    %esp,%ebp
 8048418:       51                      push   %ecx
 8048419:       83 ec 04                sub    $0x4,%esp
 804841c:       83 ec 0c                sub    $0xc,%esp
 804841f:       68 c0 84 04 08          push   $0x80484c0
 8048424:       e8 b7 fe ff ff          call   80482e0 <puts@plt>
 8048429:       83 c4 10                add    $0x10,%esp
 804842c:       b8 00 00 00 00          mov    $0x0,%eax
 8048431:       8b 4d fc                mov    -0x4(%ebp),%ecx
 8048434:       c9                      leave
 8048435:       8d 61 fc                lea    -0x4(%ecx),%esp
 8048438:       c3                      ret
 8048439:       66 90                   xchg   %ax,%ax
 804843b:       66 90                   xchg   %ax,%ax
 804843d:       66 90                   xchg   %ax,%ax
 804843f:       90                      nop
[...]
```

IDA

- The mother of disassemblers
- Very powerful tool
- Quite costly
- Free version is available for Windows but quite outdated. Rumors tell there will be an update sometime soon
- Runs in wine on Mac OS and Linux
- Not really much to know about, I'll show the basic features later
- If you have the money, it even has a decompiler for certain architectures
- Point to take away: It's **the** tool to do reversing
- https://www.hex-rays.com/products/ida/index.shtml

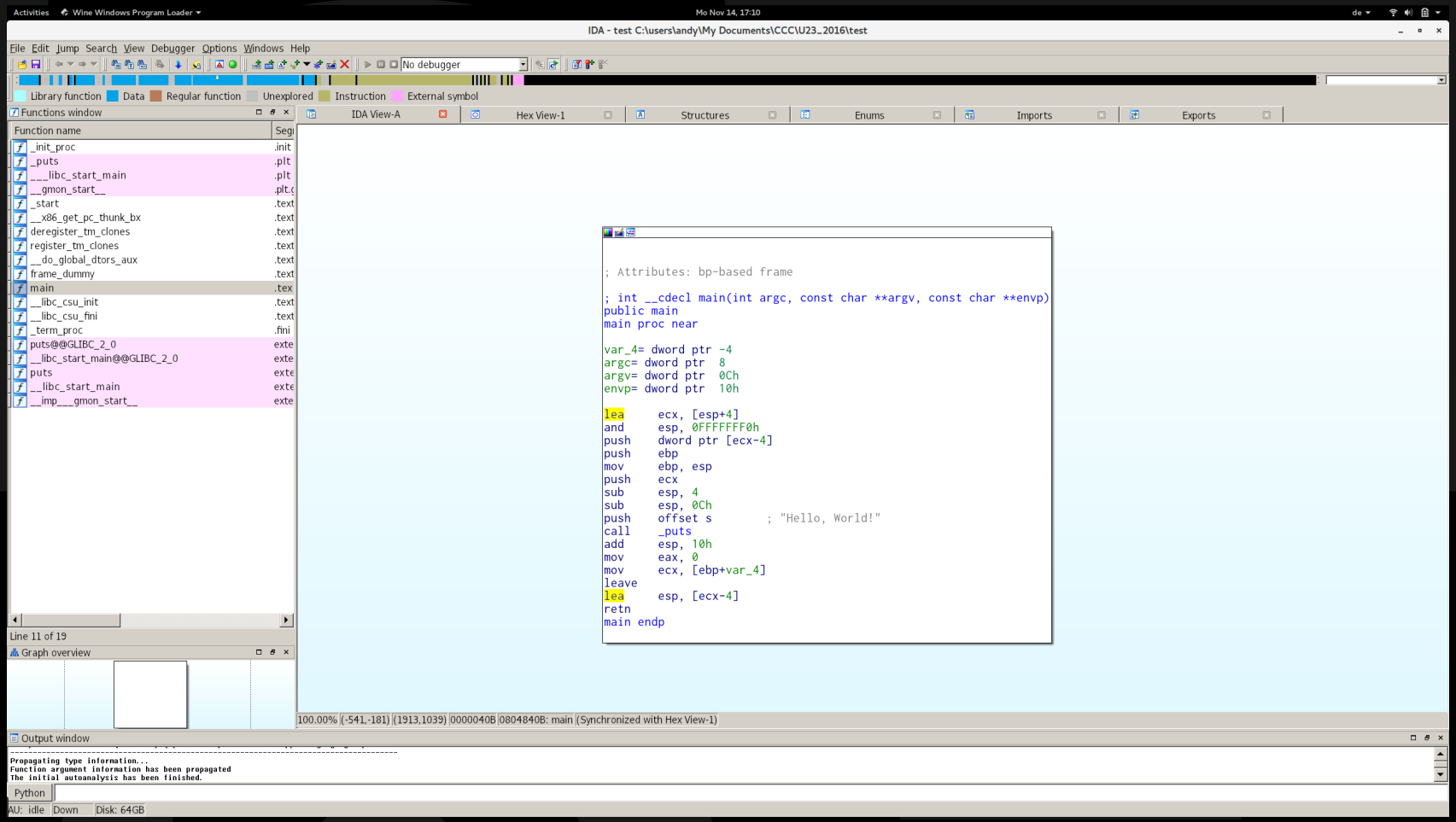
IDA - test C:\users\andy\My Documents\CCC\U23_2016\test
File Edit Jump Search View Debugger Options Windows Help
Functions window
Function name
_init_proc
_puts
___libc_start_main
__gmon_start__
_start
__x86_get_pc_thunk_bx
deregister_tm_clones
register_tm_clones
__do_global_dtors_aux
frame_dummy
main
__libc_csu_init
__libc_csu_fini
_term_proc
puts@@GLIBC_2_0
__libc_start_main@@GLIBC_2_0
puts
__libc_start_main
__imp___gmon_start__
Line 11 of 19
Graph overview
IDA View-A
Hex View-1
Structures
Enums
Imports
Exports
; Attributes: bp-based frame
; int __cdecl main(int argc, const char **argv, const char **envp)
public main
main proc near
var_4= dword ptr -4
argc= dword ptr 8
argv= dword ptr 0Ch
envp= dword ptr 10h
lea ecx, [esp+4]
and esp, 0FFFFFFF0h
push dword ptr [ecx-4]
push ebp
mov ebp, esp
push ecx
sub esp, 4
sub esp, 0Ch
push offset s ; "Hello, World!"
call _puts
add esp, 10h
mov eax, 0
mov ecx, [ebp+var_4]
leave
lea esp, [ecx-4]
retn
main endp
Output window
Propagating type information...
Function argument information has been propagated
The initial autoanalysis has been finished.
Python
AU: idle Down Disk: 64GB

Figure: IDA

Hopper

- A bit like IDA
- Demo version available
- Not quite that expensive
- Also not that powerful
- https://www.hopperapp.com/

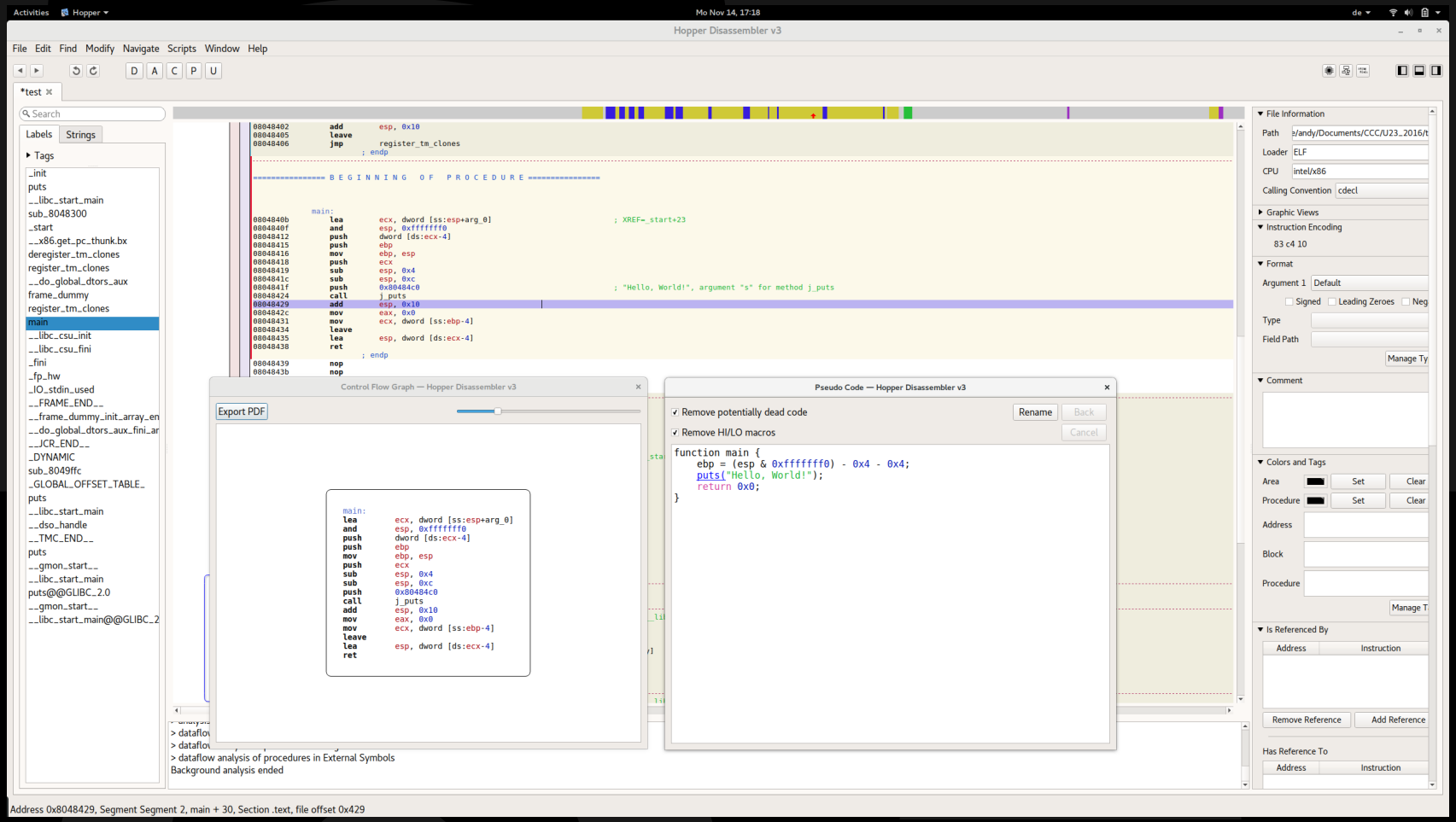

Figure: Hopper

gdb

- Standard Debugger for unix environments
- Has a textinterface
- Different addons available like peda (https://github.com/longld/peda) specifically for exploitation/reversing

gdb Cheat Sheet 1

- Start gdb: `gdb ./myfile`
- Running the binary: `run`
- Setting breakpoints on symbols: `break main`
- Setting breakpoints on addresses: `break *0x4004fa`
- Listing breakpoints: `info breakpoints`
- Delete breakpoints: `del <n>` (n = number from `info breakpoints`)
- Show registers: `info registers`
- Disassemble things: `disassemble main`
- Disassemble things without symbols: `disassemble 0x4004fa,+0x20` (disassemble 0x20 bytes starting from 0x4004fa)

gdb Cheat Sheet 2

- Show backtrace: `backtrace`
- Single step instruction: `si`
- Single step instruction while not entering subroutines: `ni`
- Forward to end of function: `finish`
- Continue until next breakpoint: `continue`
- Examine memory: `x` - Example: `x/32wx $esp` - "Look into a pointer at esp and dump 32 words in hexadecimal representation"
- Print strings: `print` - Example: `print (char*)0x80484c0` - "Take address `0x80484c0`, cast it into a char pointer and print the string it points to"
- Help: `help <command>`

gdb peda

- peda is a nice addon for GDB
- Get it from https://github.com/longld/peda
- See Readme.md on how to install it

Section 3

Live Reversing

Section 4

Exercises

Exercises

1. Solve the three exercises in `/u23/reversing`
2. Find out what they do
3. Reverse engineer
4. ???
5. Profit!

Hint: If you encounter unknown C-functions, check the man-pages!
Example: `man 3 strlen`

Copying files to/from the remote machine:

```
scp -P 8523 user@u23.labor.koeln.ccc.de:/u23/reversing/*.elf
/local/directory
```

Windows users: Use WinSCP