Reverse Engineering for CTFs 101

Some assembly required

whoami

Twitter @savinojossi

Email sj@l0dge.org

What to expect

and what not to expect

Linux binaries

No binary exploitation

Outline

Assembly

Static analysis

Dynamic analysis

Misconceptions

Assembly is hard

Misconceptions

I’m not a coder

Assembly

Basics

CPU instructions in human-readable form

01001000 11000111 11000000 00110111 00010011 00000000 00000000

mov eax, 0x1337

Assembly

Basics

Platform-dependent

- Intel
- ARM
- MIPS
- RISC-V

Assembly

Basics

Platform-dependent

- Intel
- ARM
- MIPS
- RISC-V

Assembly

Syntax

Intel

```
add esp, 4
```

AT&T

```
addl $4, %esp
```

Assembly

Syntax

Intel

```
add esp, 4
```

AT&T

```
addl $4, %esp
```

Assembly

General-purpose registers

RAX

RBX

RCX

R8

R9

...

Assembly

Index and pointer registers

`RSP`

Stack pointer

`RBP`

Base pointer

`RIP`

Instruction pointer

Memory

Endianness

280 = 2 * 100 + 8 * 10 + 0 * 1

Little Endian

```
110 = 1*4 + 1*2 + 0*1
```

Big Endian

```
110 = 1*1 + 1*2 + 0*4
```

Memory

Endianness

280 = 2 * 100 + 8 * 10 + 0 * 1

Little Endian

```
110 = 1*4 + 1*2 + 0*1
```

Big Endian

```
110 = 1*1 + 1*2 + 1*4
```

Assembly

Basic Instructions

Operations

Assembly

Basic Instructions

add 0x12 to RAX

```
add rax, 0x12
```

subtract 0x12 from RBX

```
sub rbx, 0x12
```

Assembly

Basic Instructions

Logical AND operation

```
and r8, 0x10111111
```

Logical XOR operation

```
xor r9, 0x00000010
```

Assembly

Basic Instructions

Store the value 0x1 into RAX

```
mov rax, 0x1
```

Copy the value of RAX into R9

```
mov r9, rax
```

Assembly

Dereference

Operate on address stored in RAX

```
add [rax], 0x4
```

```
RAX: 0x1004

0x1000     00000000
0x1004     00000004
0x1008     00000000
```

Assembly

Dereference

Operate on address stored in RAX

```
add [rax], 0x4
```

```
RAX: 0x1004

0x1000     00000000
0x1004     00000008
0x1008     00000000
```

Assembly

Flow control

Jump to an address

```
jmp 0x118b
```

Assembly

Flow control

Jump if RAX contains 0x4

```
cmp rax, 0x4
jeq 0x118b
```

Assembly

Flow control

Jump if RAX does not contain 0x4

```
cmp rax, 0x4
jne 0x118b
```

Assembly

Flow control

Jump if RAX is less than 0x4

```
cmp rax, 0x4
jl  0x118b
```

Assembly

Flow control

Jump if RAX is less or equal than 0x4

```
cmp rax, 0x4
jle 0x118b
```

Assembly

Flow control

Jump if RAX is greater than 0x4

```
cmp rax, 0x4
jg 0x118b
```

Assembly

Flow control

Jump if RAX is greater or equal than 0x4

```
cmp rax, 0x4
jge 0x118b
```

Assembly

Functions

Call a function

```
call 0x800117e <foo>
```

Assembly

Functions

Function prologue

```
push    rbp
mov     rbp, rsp
sub     rsp, 0x20
```

Assembly

Functions

Function epilogue

```
leave
ret
```

Static analysis

CTF question

RTFQ

Static analysis

CTF question

Take the hints!

Static analysis

strings

Extract readable text

`strings`

strings

```
$ strings ./binary

/lib64/ld-linux-x86-64.so.2
l +xX
libc.so.6
printf
malloc
__cxa_finalize
__libc_start_main
GLIBC_2.2.5
_ITM_deregisterTMCloneTable
__gmon_start__
_ITM_registerTMCloneTable
u/UH
[]A\A]A^A_
%c%c%c%c%c%s%c
;*3$"
GCC: (Debian 9.3.0-10) 9.3.0
/usr/lib/gcc/x86_64-linux-gnu/9/include
/usr/include/x86_64-linux-gnu/bits
/usr/include/x86_64-linux-gnu/bits/types
/usr/include
main.c
```

Static analysis

Symbols

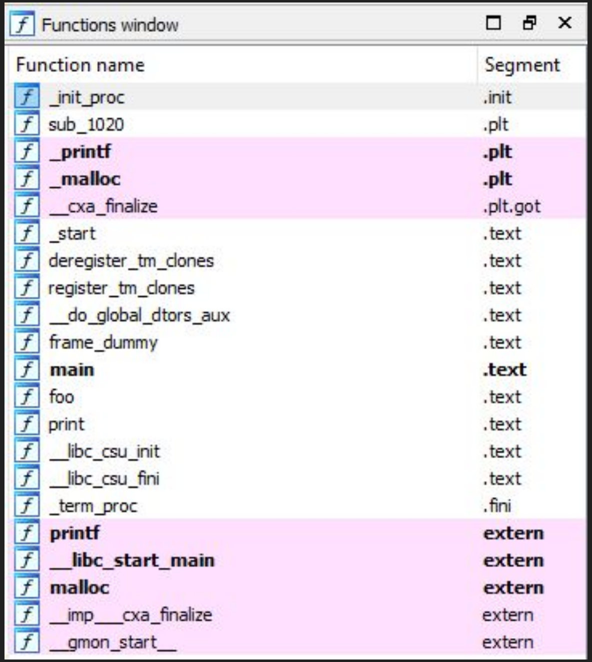
Function names

Variable names

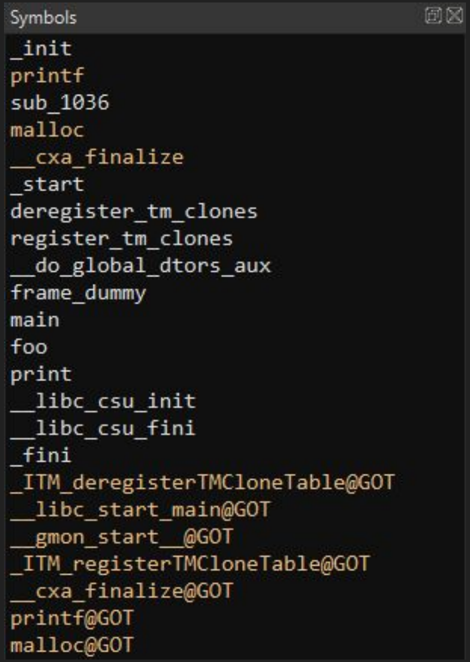
symbols

```
$ objdump -x ./binary

...
0000000000001000 l  F .init  0000000000000000              _init
0000000000001320 g  F .text  0000000000000001              __libc_csu_fini
0000000000000000  w       *UND*  0000000000000000                _ITM_deregisterTMCloneTable
0000000000004028  w       .data  0000000000000000                data_start
0000000000004038 g        .data  0000000000000000                _edata
0000000000001324 g  F .fini  0000000000000000              .hidden _fini
0000000000000000    F *UND*  0000000000000000              printf@@GLIBC_2.2.5
00000000000011a9 g  F .text  000000000000010c              print
0000000000000000    F *UND*  0000000000000000              __libc_start_main@@GLIBC_2.2.5
0000000000004028 g        .data  0000000000000000                __data_start
0000000000000000  w       *UND*  0000000000000000                __gmon_start__
0000000000004030 g  O .data  0000000000000000              .hidden __dso_handle
0000000000002000 g  O .rodata    0000000000000004          _IO_stdin_used
00000000000012c0 g  F .text  000000000000005d              __libc_csu_init
0000000000000000    F *UND*  0000000000000000              malloc@@GLIBC_2.2.5
000000000000117e g  F .text  000000000000002b              foo
0000000000004040 g        .bss   0000000000000000                _end
...
```

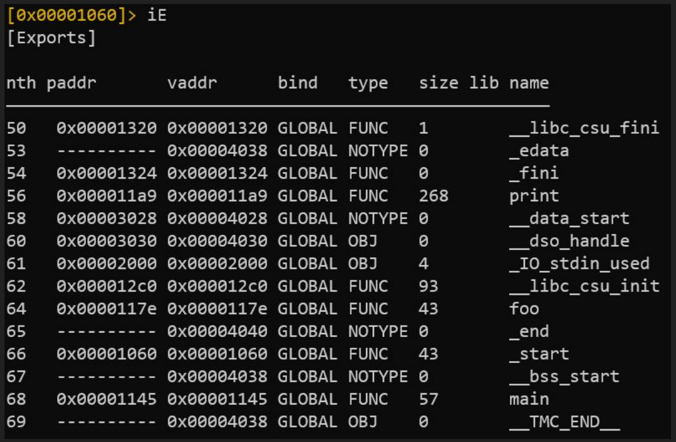
symbols



Function name	Segment
_init_proc	.init
sub_1020	.plt
_printf	**.plt**
_malloc	**.plt**
__cxa_finalize	.plt.got
_start	.text
deregister_tm_clones	.text
register_tm_clones	.text
__do_global_dtors_aux	.text
frame_dummy	.text
main	**.text**
foo	.text
print	.text
__libc_csu_init	.text
__libc_csu_fini	.text
_term_proc	.fini
printf	**extern**
__libc_start_main	**extern**
malloc	**extern**
__imp___cxa_finalize	extern
__gmon_start__	extern



```
_init
printf
sub_1036
malloc
__cxa_finalize
_start
deregister_tm_clones
register_tm_clones
__do_global_dtors_aux
frame_dummy
main
foo
print
__libc_csu_init
__libc_csu_fini
_fini
_ITM_deregisterTMCloneTable@GOT
__libc_start_main@GOT
__gmon_start__@GOT
_ITM_registerTMCloneTable@GOT
__cxa_finalize@GOT
printf@GOT
malloc@GOT
```



```
[0x00001060]> iE
[Exports]

nth paddr      vaddr      bind   type   size lib name
―――――――――――――――――――――――――――――――――――――――――――――――――――――
50  0x00001320 0x00001320 GLOBAL FUNC   1        __libc_csu_fini
53  ---------- 0x00004038 GLOBAL NOTYPE 0        _edata
54  0x00001324 0x00001324 GLOBAL FUNC   0        _fini
56  0x000011a9 0x000011a9 GLOBAL FUNC   268      print
58  0x00003028 0x00004028 GLOBAL NOTYPE 0        __data_start
60  0x00003030 0x00004030 GLOBAL OBJ    0        __dso_handle
61  0x00002000 0x00002000 GLOBAL OBJ    4        _IO_stdin_used
62  0x000012c0 0x000012c0 GLOBAL FUNC   93       __libc_csu_init
64  0x0000117e 0x0000117e GLOBAL FUNC   43       foo
65  ---------- 0x00004040 GLOBAL NOTYPE 0        _end
66  0x00001060 0x00001060 GLOBAL FUNC   43       _start
67  ---------- 0x00004038 GLOBAL NOTYPE 0        __bss_start
68  0x00001145 0x00001145 GLOBAL FUNC   57       main
69  ---------- 0x00004038 GLOBAL OBJ    0        __TMC_END__
```

Static analysis

POI

```
main()
```

Static analysis

```
(gdb) disassemble main
Dump of assembler code for function main:
   ...
   0x000000000000115f <+26>: jg  0x116d <main+40>
   0x0000000000001161 <+28>: mov eax,0x0
   0x0000000000001166 <+33>: call   0x11a9 <print>
   0x000000000000116b <+38>: jmp 0x1177 <main+50>
   0x000000000000116d <+40>: mov eax,0x0
   0x0000000000001172 <+45>: call   0x117e <foo>
   ...
```

Static analysis

POI

```
printf()
```

Static analysis

POI

Uncalled functions

Dynamic analysis

First execution

Run it

Dynamic analysis

```
$ ./binary

Segmentation fault (core dumped)
```

Dynamic analysis

gdb

gdb start

```
$ gdb ./binary
Reading symbols from binary...
(gdb) start
Temporary breakpoint 1 at 0x1154: file src/main.c, line 10.
Starting program: ./binary

Temporary breakpoint 1, main (argc=1, argv=0x7fffffffee268) at
src/main.c:10
```

gdb start

```
(gdb) disas main
Dump of assembler code for function main:
   0x0000000000001145 <+0>:   push   rbp
   0x0000000000001146 <+1>:   mov rbp,rsp
(gdb) start
...
(gdb) disas main
Dump of assembler code for function main:
   0x0000000008001145 <+0>:   push   rbp
   0x0000000008001146 <+1>:   mov rbp,rsp
```

gdb continue

```
(gdb) continue
Continuing.

Program received signal SIGSEGV, Segmentation fault.
```

gdb disassemble

```
(gdb) disassemble
Dump of assembler code for function foo:
...
   0x0000000008001195 <+23>:  mov rax,QWORD PTR [rbp-0x10]
   0x0000000008001199 <+27>:  add rax,rdx
=> 0x000000000800119c <+30>:  mov BYTE PTR [rax],0x63
   0x000000000800119f <+33>:  cmp DWORD PTR [rbp-0x4],0x66
   0x00000000080011a3 <+37>:  jle 0x800118b <foo+13>
...
```

gdb Breakpoints

```
(gdb) disassemble main
Dump of assembler code for function main:
   ...
   0x000000000800115b <+22>:  cmp DWORD PTR [rbp-0x4],0x0
   0x000000000800115f <+26>:  jg  0x116d <main+40>
   0x0000000008001161 <+28>:  mov eax,0x0
   ...
```

gdb Breakpoints

```
(gdb) break *0x000000000800115b
Breakpoint 1 at 0x115b: file src/main.c, line 11.

(gdb) b *0x800115b
Breakpoint 1 at 0x115b: file src/main.c, line 11.
```

Binary patching

```
(gdb) disassemble main
Dump of assembler code for function main:
   ...
   0x000000000000115f <+26>: jg  0x116d <main+40>
   0x0000000000001161 <+28>: mov eax,0x0
   0x0000000000001166 <+33>: call   0x11a9 <print>
   0x000000000000116b <+38>: jmp 0x1177 <main+50>
   0x000000000000116d <+40>: mov eax,0x0
   0x0000000000001172 <+45>: call   0x117e <foo>
   ...
```

Binary patching

```
(gdb) disassemble /r
Dump of assembler code for function main:
   ...
   0x0800115f <+26>:  7f 0c    jg     0x800116d <main+40>
   0x08001161 <+28>:  b8 00 00 00 00  mov    eax,0x0
   0x08001166 <+33>:  e8 3e 00 00 00  call   0x80011a9 <print>
   0x0800116b <+38>:  eb 0a    jmp    0x8001177 <main+50>
   0x0800116d <+40>:  b8 00 00 00 00  mov    eax,0x0
   0x08001172 <+45>:  e8 07 00 00 00  call   0x800117e <foo>
   ...
```

Binary patching

```
74 cb     JE rel8
7F cb     JG rel8
7D cb     JGE rel8
7C cb     JL rel8
7E cb     JLE rel8
```

Binary patching

```
00001150  48 89 75 e0 c7 45 fc 79  H.u..E.y
00001158  00 00 00 83 7d fc 00 7f  ....}...
00001160  0c b8 00 00 00 00 e8 3e  .......>
00001168  00 00 00 eb 0a b8 00 00  ........
```

```
00001150  48 89 75 e0 c7 45 fc 79  H.u..E.y
00001158  00 00 00 83 7d fc 00 7e  ....}..~
00001160  0c b8 00 00 00 00 e8 3e  .......>
00001168  00 00 00 eb 0a b8 00 00  ........
```

gdb disassemble

```
(gdb) disassemble /r main
Dump of assembler code for function main:
   ...
   0x0000115f <+26>:  7e 0c   jle    0x116d <main+40>
   0x00001161 <+28>:  b8 00 00 00 00  mov   eax,0x0
   0x00001166 <+33>:  e8 3e 00 00 00  call   0x11a9 <print>
   0x0000116b <+38>:  eb 0a   jmp    0x1177 <main+50>
   0x0000116d <+40>:  b8 00 00 00 00  mov   eax,0x0
   0x00001172 <+45>:  e8 07 00 00 00  call   0x117e <foo>
```

Experience

The secret sauce

```
strings, strings all day
```

Experience

The secret sauce

Windows

Linux

Embedded Devices

Malware

Exploitation

Experience

The secret sauce

Resources

Hex editors

Windows
- HxD
- UltraEdit

Mac
- Hex Fiend

Linux
- dhex
- radare2

Resources

Disassemblers

gdb / edb

radare2 / Cutter

IDA Free / Pro / Home

Binary Ninja

Hopper

Deep dive

Memory regions

```
(gdb) info proc map
Mapped address spaces:
        Start Addr     End Addr       Size      Offset objfile
        0x8000000      0x8001000      0x1000    0x0    ./binary
        0x8001000      0x8002000      0x1000    0x0    ./binary
        0x8002000      0x8003000      0x1000    0x0    ./binary
        ...
        0x7fffff7ef000 0x7ffffffef000 0x800000  0x0    [stack]
```

Assembly

Basic Instructions

Push a value onto the stack

```
push 0x491c
```

Pop a value from the stack

```
pop  rax
```

How the stack works

	Address	Value
RSP: 0x0010	0x0000	00000000
	0x0004	00000000
	0x0008	00000000
	0x000C	00000000
	0x0010	41424344

How the stack works

```
push 0x1337        RSP: 0x0010
```

Address	Value
0x0000	00000000
0x0004	00000000
0x0008	00000000
0x000C	00000000
0x0010	41424344

How the stack works

```
push 0x1337        RSP: 0x000C
```

Address	Value
0x0000	00000000
0x0004	00000000
0x0008	00000000
0x000C	00000000
0x0010	41424344

How the stack works

```
push 0x1337        RSP: 0x000C
```

Address	Value
0x0000	00000000
0x0004	00000000
0x0008	00000000
0x000C	00001337
0x0010	41424344

How the stack works

```
push 0x1337
```

RSP: 0x000C

RAX: 0x0000

Address	Value
0x0000	00000000
0x0004	00000000
0x0008	00000000
0x000C	00001337
0x0010	41424344

How the stack works

```
push 0x1337
pop  RAX
```

```
RSP: 0x000C
RAX: 0x0000
```

Address	Value
0x0000	00000000
0x0004	00000000
0x0008	00000000
0x000C	00001337
0x0010	41424344

How the stack works

```
push 0x1337        RSP: 0x000C
pop  RAX           RAX: 0x1337
```

Address	Value
0x0000	00000000
0x0004	00000000
0x0008	00000000
0x000C	00001337
0x0010	41424344

How the stack works

```
push 0x1337      RSP: 0x0010
pop  RAX         RAX: 0x1337
```

Address	Value
0x0000	00000000
0x0004	00000000
0x0008	00000000
0x000C	00001337
0x0010	41424344

Static analysis

Arguments

Passed in order

- `rdi`
- `rsi`
- `rdx`
- `rcx`
- `r8`
- `r9`
- `stack`

Static analysis

Arguments

```
lea     rdi, 0x8002004
mov     eax, 0x0
call    0x1030 <printf@plt>
```

Dynamic analysis

Return Code

```
lea     rdi, 0x2004
mov     eax, 0x0
call    0x1030 <printf@plt>
# breakpoint, read RAX
```

Shoutouts

@BHinfoSecurity and @corelight_inc for making this possible

@Fox0x01 for the ARM cheat sheet and azeria-labs.com in general

@TimMedin for his limitless amount of tricks and being a great teacher

@jmpalk1 for the cheat sheet

@nemesis09 for being noisy on Twitter

@the_ghosteh and Coen without Twitter for being awesome

Thank you!

Bonus slides

Why JLE?

```
(gdb) disassemble /r
Dump of assembler code for function main:
   ...
   0x0800115f <+26>:  7f 0c   jg     0x800116d <main+40>
   0x08001161 <+28>:  b8 00 00 00 00  mov   eax,0x0
   0x08001166 <+33>:  e8 3e 00 00 00  call   0x80011a9 <print>
   0x0800116b <+38>:  eb 0a   jmp    0x8001177 <main+50>
   0x0800116d <+40>:  b8 00 00 00 00  mov   eax,0x0
   0x08001172 <+45>:  e8 07 00 00 00  call   0x800117e <foo>
   ...
```

Why JLE?

Lazyness is an option

Close in documentation

No alignment issues

Why JLE?

Safest alternative

```
90  NOP
```

Every byte!

Hex editors

vim

```
vim ./binary
```

```
:%!xxd
```

Make your changes in ASCII

```
:%!xxd -r
```

```
:wq
```

Why AT&T syntax is insane

Assembly

Syntax

Intel

```
mov eax, [ebx + ecx*4 + mem_location]
```

Assembly

Syntax

Intel

```
mov eax, [ebx + ecx*4 + mem_location]
```

AT&T

```
movl mem_location(%ebx,%ecx,4), %eax
```

Visualization of register width

Registers

Width

```
... 0000 0000 0000 0000 0000
```

AL

AH

AX

EAX

RAX

Registers

Width

```
... 0000 0000 0000 0000 0000
```

AL

AH

AX

EAX

RAX

Registers

Width

```
... 0000 0000 0000 0000 0000
```

AL

AH

AX

EAX

RAX

Registers

Width

... 0000 0000 0000 0000 0000

AL

AH

AX

EAX

RAX

Registers

Width

... 0000 0000 0000 0000 0000

AL

AH

AX

EAX

RAX

Registers

Width

... 0000 0000 0000 0000 0000

AL

AH

AX

EAX

RAX

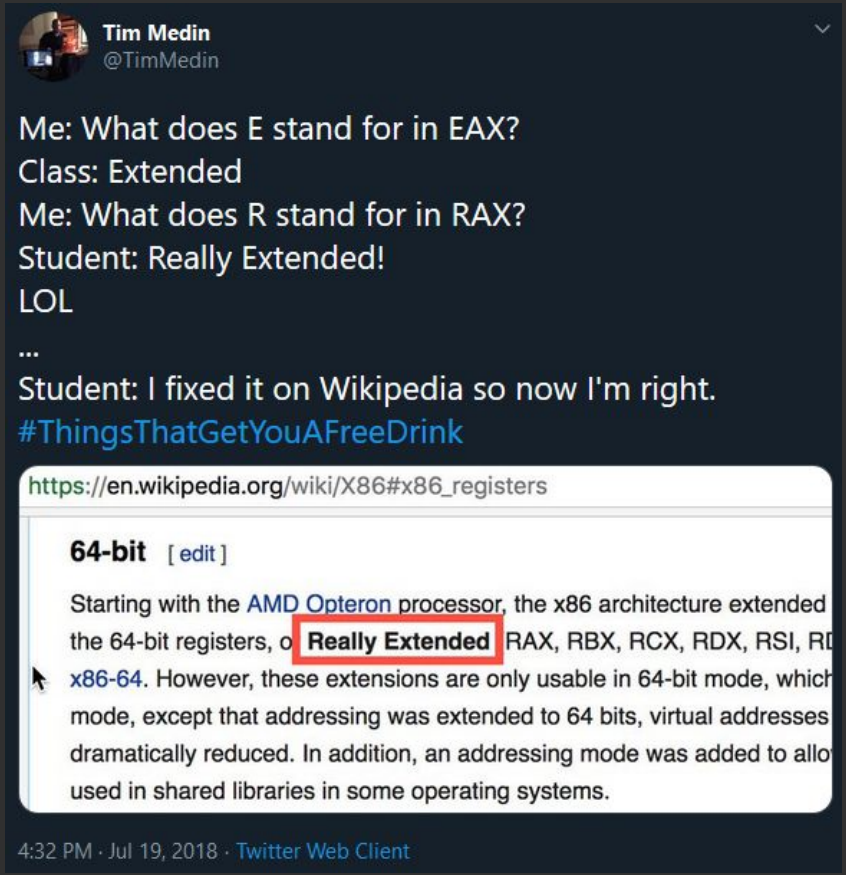
Registers

Memory pegs

```
... 0000 0000 0000 0000 0000

AL      A low

AH      A high

AX      A full

EAX     A Extended

RAX     A Register
```

Registers

Width



FLAGS register

```
15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0

 0 NT  IOPL OF DF IF TF SF ZF  0 AF  0  PF 1 CF
```

Assembly

Flow control

Jump if RAX is greater than 0x4

```
cmp rax, 0x4
jg 0x118b
```

Flow control

FLAGS

```
cmp rax, 0x4

rax - 0x4

je 0x118b
    if ZF = 1
```