
Writing x86_64 Assembly
0xC0FFEE Virtual Workshop
17 April 2020

Meeting Logistics

- Meet the moderator — Michael Higgo
- Question breaks
- Raise hand to ask a question — click on “Participants”
- Meeting being recorded
- Chat at https://discord.gg/c8P23U

Don’t be a dick

Please

- Locking room in 1 .. 2 .. 3

Prerequisites

```
> docker run -it singelet/x86_64_workshop
```

Or

```
> git clone https://github.com/singe/x86_64_workshop
```


0xC0FFEE
HACKER


@singe
Dominic White
dominic@sensepost.com

Why?

- Know
  - what's actually executing
  - how it works — mostly
  - how to change it
  - how to write your own
- Optimise — the compiler isn't always right
- Understand 1337 shellcodez - maybe
- Background for reversing

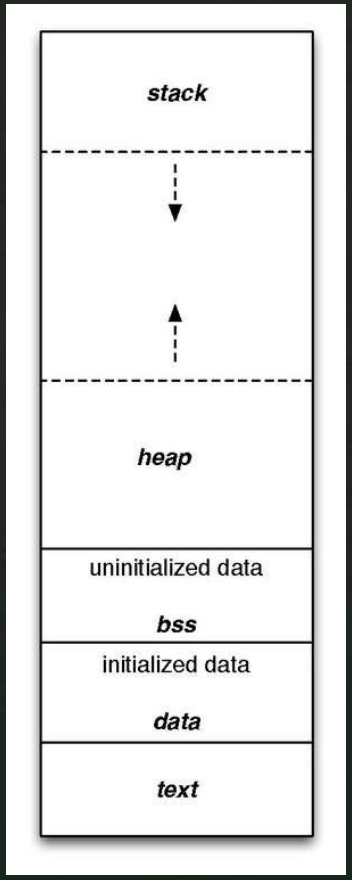

Screw slides — let’s get dirty
You’ll need your docker container

Sections

.text — code

.data — initialised data

.bss — uninitialized data



Registers

rsp — stack pointer

rip — instruction pointer

rax — return value

rdi, rsi, rcx, rdx, r8, r9 - args

Instructions

push, pop — put something on or take something off the stack

mov — move a value from one place to another

lea — load the contents of the memory pointed at

call — call a function

syscall — call a syscall

ret — return from a function

Building

```
as -o output.o input.s

ld -o binary output.o
```

Registers

Register	Conventional use	Low 32-bits	Low 16-bits	Low 8-bits
`%rax`	Return value, callee-owned	`%eax`	`%ax`	`%al`
`%rdi`	1st argument, callee-owned	`%edi`	`%di`	`%dil`
`%rsi`	2nd argument, callee-owned	`%esi`	`%si`	`%sil`
`%rdx`	3rd argument, callee-owned	`%edx`	`%dx`	`%dl`
`%rcx`	4th argument, callee-owned	`%ecx`	`%cx`	`%cl`
`%r8`	5th argument, callee-owned	`%r8d`	`%r8w`	`%r8b`
`%r9`	6th argument, callee-owned	`%r9d`	`%r9w`	`%r9b`
`%r10`	Scratch/temporary, callee-owned	`%r10d`	`%r10w`	`%r10b`
`%r11`	Scratch/temporary, callee-owned	`%r11d`	`%r11w`	`%r11b`
`%rsp`	Stack pointer, caller-owned	`%esp`	`%sp`	`%spl`
`%rbx`	Local variable, caller-owned	`%ebx`	`%bx`	`%bl`
`%rbp`	Local variable, caller-owned	`%ebp`	`%bp`	`%bpl`
`%r12`	Local variable, caller-owned	`%r12d`	`%r12w`	`%r12b`
`%r13`	Local variable, caller-owned	`%r13d`	`%r13w`	`%r13b`
`%r14`	Local variable, caller-owned	`%r14d`	`%r14w`	`%r14b`
`%r15`	Local variable, caller-owned	`%r15d`	`%r15w`	`%r15b`
`%rip`	Instruction pointer			
`%eflags`	Status/condition code bits			

https://web.stanford.edu/class/cs107/guide/x86-64.html

Instructions

push, pop — put something on or take something off the stack

mov — move a value from one place to another

lea — load the contents of the memory pointed at

call — call a function

syscall — call a syscall

ret — return from a function

jmp — jump somewhere

cmp — compare two things

jne — jump if not equal (used after cmp)

add, sub — like it says

624 others - http://ref.x86asm.net/coder64-abc.html

Instruction Suffix

- `b` for 1 byte (8 bits)
- `w` for a word (2 bytes, 16 bits)
- `l` for a long or double word (4 bytes, 32 bits)
- `q` for a quad word (8 bytes, 64 bits)

Building

Simplest:

```
as -o output.o input.s
ld -o binary output.o
```

Linker

```
ld -dynamic-linker <ld.so> -lc -e <function> -o binary output.o
```

More Building

~/build/Makefile

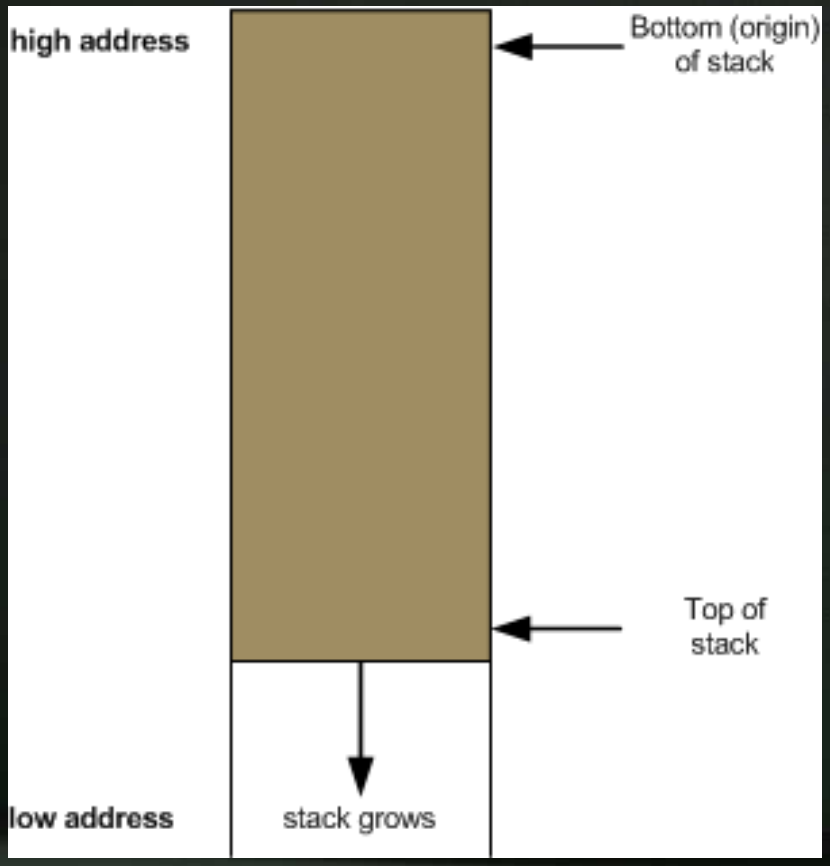
```
  > make
  > make clean
```

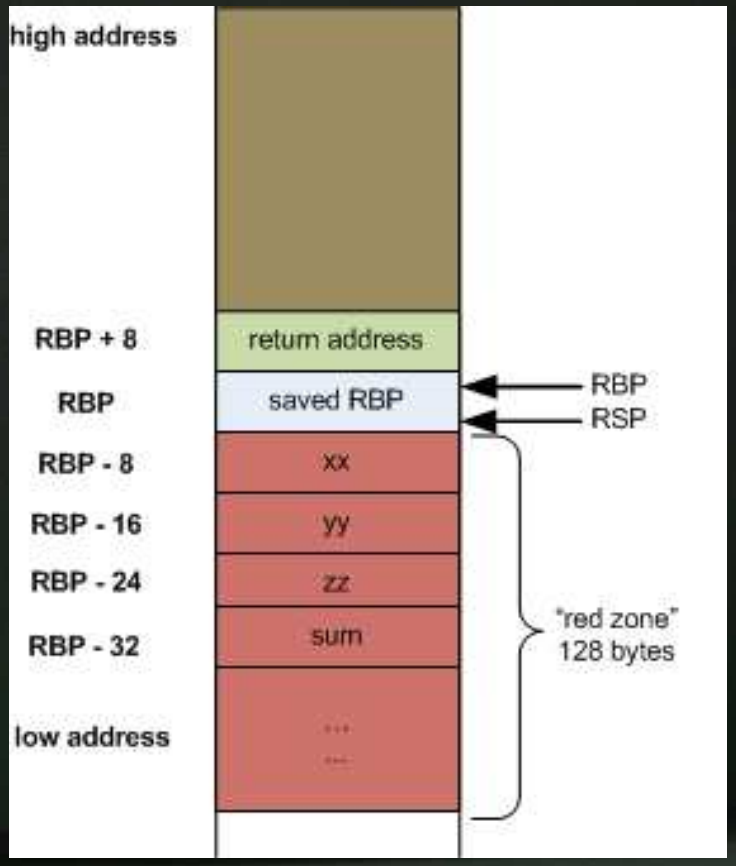
~/build/compile.sh

```
      -h help This help
      -a <arch> Specify the architecture default: x86_64
      -i <yes|no> Enable dynamic linking and libc import (Default: yes)
      -r run Run the program after building it
      -c cleanup Delete the .o and binary afterwards
      -x xxd Run the output through xxd when running it
      -s strace Run the program with strace

  > ./compile.sh -rc -i no <asm.s>
```

The Stack


high address
Bottom (origin)
of stack
Top of
stack
low address
stack grows


high address
RBP + 8
return address
RBP
saved RBP
RBP
RSP
RBP - 8
xx
RBP - 16
yy
RBP - 24
zz
RBP - 32
sum
"red zone"
128 bytes
low address


syscalls
60 - exit
1 - write
436 more at:
https://github.com/torvalds/linux/blob/master/arch/x86/entry/syscalls/syscall_64.tbl

Using the Compiler

```
gcc -S <code.c>
```

Optimisations:

```
-fomit-frame-pointer
-fno-stack-canary
-O3
```

GodBolt Compiler Explorer

https://godbolt.org/z/QpNjNF

Examining Files

objdump

- -d disassemble
- -j <section>
- -h list section headers

> objdump -d -j .text <binary>

readelf

- -a everything

gdb

break *<address or function name> - set a breakpoint

  break *_start

  break *main

info file - find entrypoint mostly

run — like it says

nexti — next instruction (step over)

stepi — next instruction

x/16cb $rsi — eXamine the memory pointed at by $rsi 16 bytes (b) at a time in ASCII