Rafael Almeida Systems Exam 2 Sheet 04/15/24

x86-64 Basics

Registers: rsp: stack pointer

rip: instruction pointer arg 1-6:

Caller Saved r10 r11

rdi, rsi, rdx, rcx, r8, r9 ret: rax

Operand Types: imm $, reg %,

mem ( ) ← dereference addr in reg

Addressing Modes:

D(Rb, Ri, S) = D + Rb + S·Ri

mov: solve the above and then access

lea: only computes

Control Flow

Condition Codes

Carry flag: carry-out = 1

Zero: result = 0

Sign: top bit = 1

Overflow: signed overflow

cmp a, b: b - a

test a, b: a and b

arithmetic inst except lea read: set reg

Conditionals

if-statements are conditional goto label

using jmp instruction

Do-While

[prep code]
.LABEL:
[body]
[some test]
[jump]
ret

While Loop

Jump to Middle

goto MID
.LOOP:
[body]
.MID:
[conditional check]

Guarded Do (may not run in first iteration)

[prep]
[condition check]
[jmp done]
.LOOP
[body]
[condition check]

For Loop

for (init; test; update)

init
while test
update

Switch Statement

Jump Table

← this addr + 8x jumps to statement x

.L4
.quad .L8 ← addr of case 0
.quad .L3 1
⋮ ⋮

Assembly

[check if case is default]
[use default]
jmp *.L4(, rdi, 8)
↳ indirect jump

Procedure Call

Stack

bottom = large addr

grows down: by subtracting

rsp → ← top: small addr

Operations

push: decrement by 8, write to (%rsp) (more; to mem)

pop: read (%rsp), increment by 8 (less)

call: push ret addr which is +8 from %rip (-8)

ret: pop and jump (+8)

Stack Frame

arguments
ret addr ← old frame's caller frame
rbp → ← calle's current frame
local vars
rsp →

Caller and Callee Saved

fun 1
save current vars in stack
call fun 2
restore vars

or

fun 1
save stuff from caller
[body]
restore caller vars
[done]

Structures

Arrays

n: # of items, s: size of item

x, x+s, x+2s, x+3s, ..., x + n·s

access: (x, i, s) = x + i·s

Struct

Similar to array but can jump

by different sizes

struct
int a[4];
long val;
char* str;

addr
a | val | str
4·4 = 16 24 32

Buffer Overflow

Old frames ← ret addr
Buf
x, sub x from rsp then a ret
← rsp

Code Injection

Old frame
← ret addr A overridden by B
Pad
Code ← B

After sub rsp and then a ret, rip goes to stack and exec my code

because rip gets incremented, which is higher up in mem (bottom of stack), code appears backwards on stack

ROP

Exploit String

1. ~
2. ~
3. ~

Stack
Bottom +
Top -

old frame
← g4
← g3
← g2
← g1 ret addr
Pad

find addr of useful existing code

Exploit String

1. ~
2. ~ pad
3. ~
4. g1 addr
5. g2
6. g3

Stack Randomization

ASLR

allocate rand amount before, cannot know add of injected code

Non-Executable Stack

Canaries

adds random num to stack and ensures it is there. OP