Memory Corruption

Roy Xu
OSIRIS Lab Hack Night

OSIRIS

Objectives

- What is Memory Corruption?
- Buffer Overflow
- Pwntools
- Shellcode
- Format String

Memory Corruption

- Modifying a binary’s memory in unintended ways
- Typically results in segmentation faults
- Violates memory safety

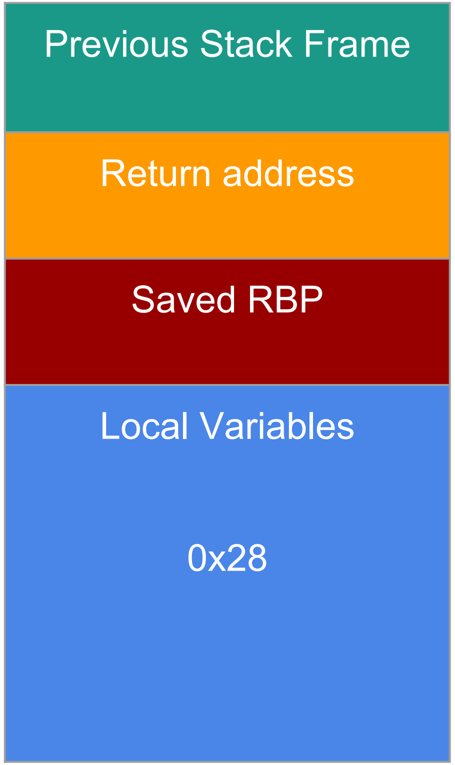
Types of Memory Corruption

- Buffer overflows
- Format string
- Out of bounds array access
- Using uninitialized memory
- Dangling pointers
- Heap corruption

Memory Corruption Goals

- Typically we want the binary to give us a shell
- Why?
  - Binaries are often run with elevated privileges
  - The setuid bit indicates that a binary should be run as root
- Root shell == win

Stack Layout



Previous Stack Frame

Return address

Saved RBP

Local Variables

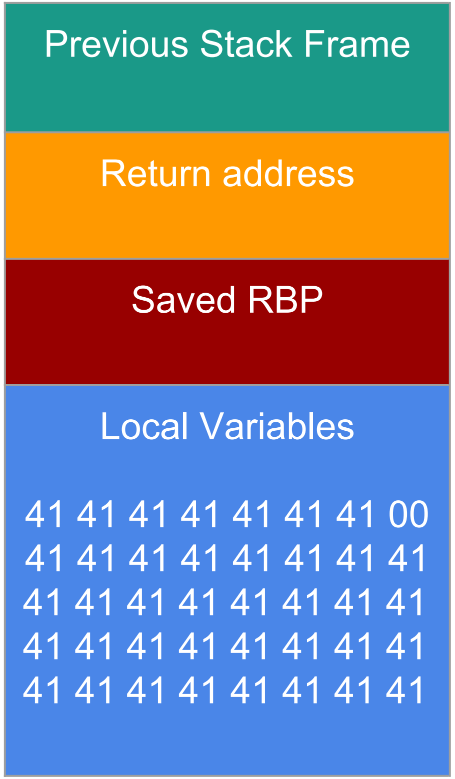
0x28

call
(push rip, jmp)

push rbp
mov rbp, rsp

sub rsp, 0x18

Buffer Overflow



Previous Stack Frame

Return address

Saved RBP

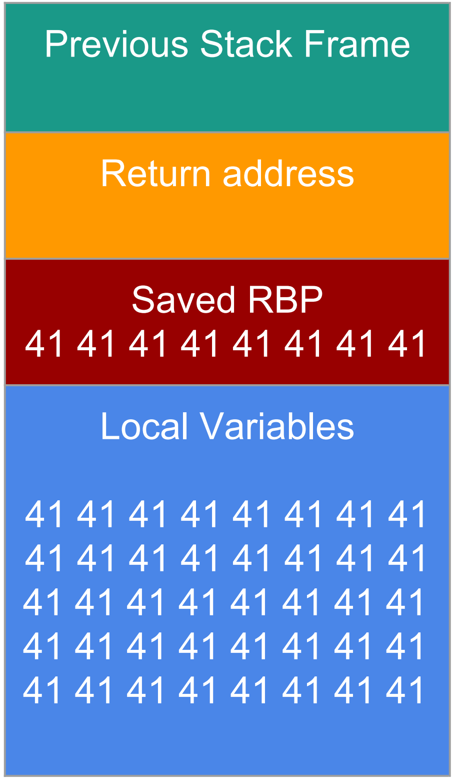
Local Variables

```
41 41 41 41 41 41 41 00
41 41 41 41 41 41 41 41
41 41 41 41 41 41 41 41
41 41 41 41 41 41 41 41
41 41 41 41 41 41 41 41
```

Fill local buffer
with 39 ‘A’s

Null byte to
terminate string

Buffer Overflow



Previous Stack Frame

Return address

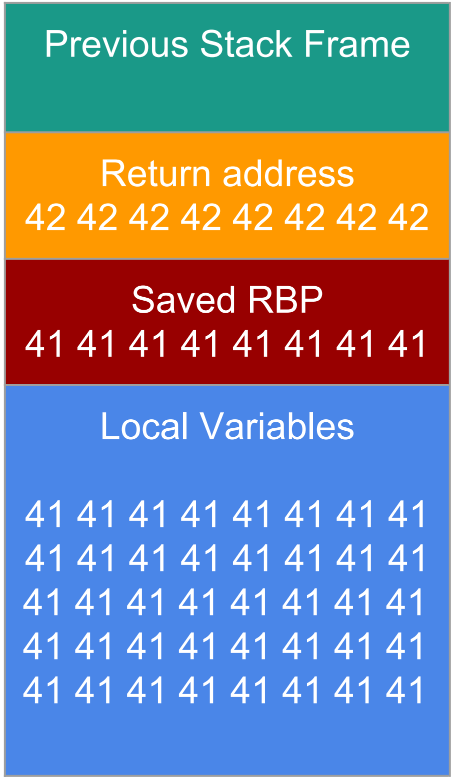
Saved RBP
41 41 41 41 41 41 41 41

Local Variables

41 41 41 41 41 41 41 41
41 41 41 41 41 41 41 41
41 41 41 41 41 41 41 41
41 41 41 41 41 41 41 41
41 41 41 41 41 41 41 41

What happens if we can input 48 ‘A’s

Buffer Overflow



Previous Stack Frame

Return address
42 42 42 42 42 42 42 42

Saved RBP
41 41 41 41 41 41 41 41

Local Variables

41 41 41 41 41 41 41 41
41 41 41 41 41 41 41 41
41 41 41 41 41 41 41 41
41 41 41 41 41 41 41 41
41 41 41 41 41 41 41 41

What happens if we can input 48 ‘A’s and 8 ‘B’s

Buffer Overflow

Function cleanup

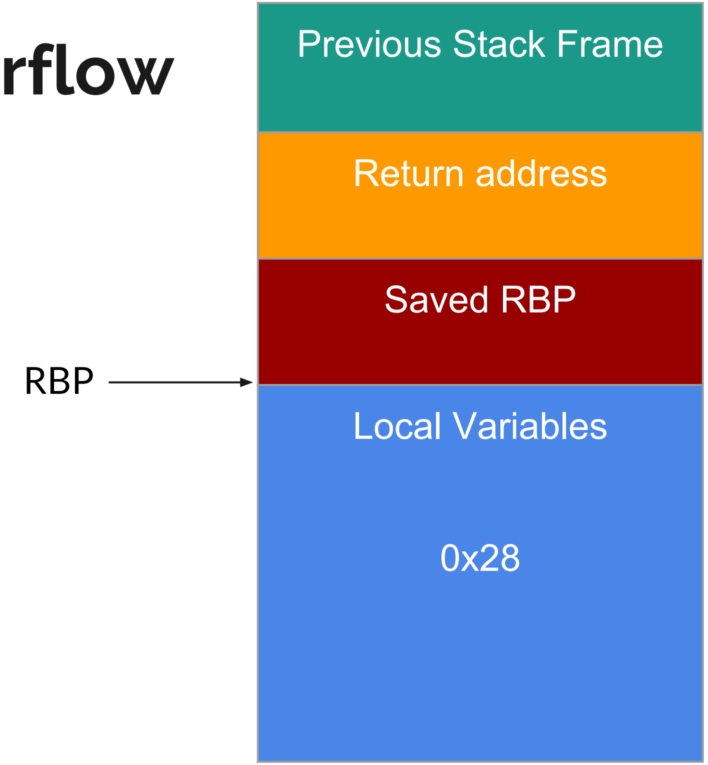
leave:

```
    mov rsp, rbp
    pop rbp
```

ret:

```
    pop rip
```

Buffer Overflow



call
(push rip, jmp)

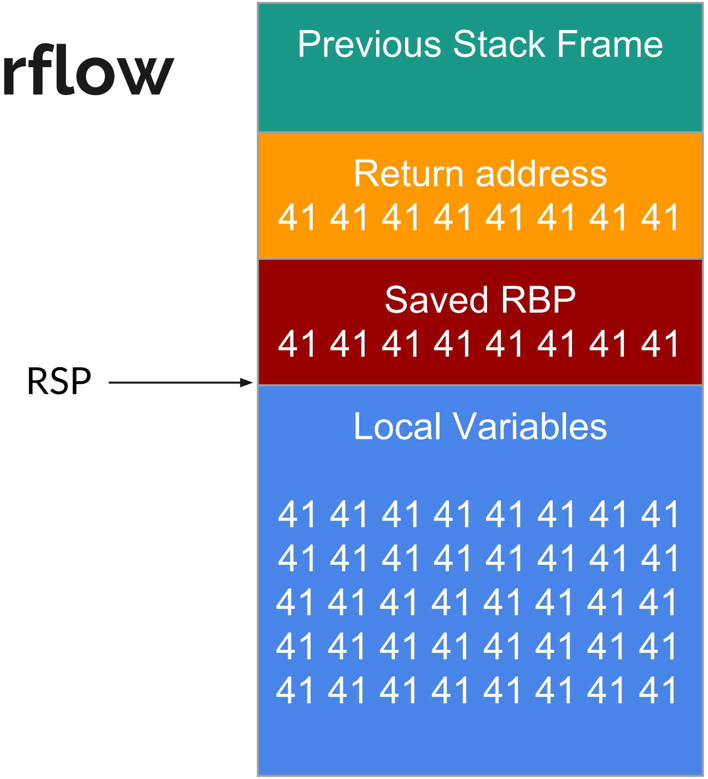
push rbp
mov rbp, rsp

sub rsp, 0x18

Buffer Overflow

leave:

mov rsp, rbp
pop rbp



RSP

Previous Stack Frame

Return address
41 41 41 41 41 41 41 41

Saved RBP
41 41 41 41 41 41 41 41

Local Variables

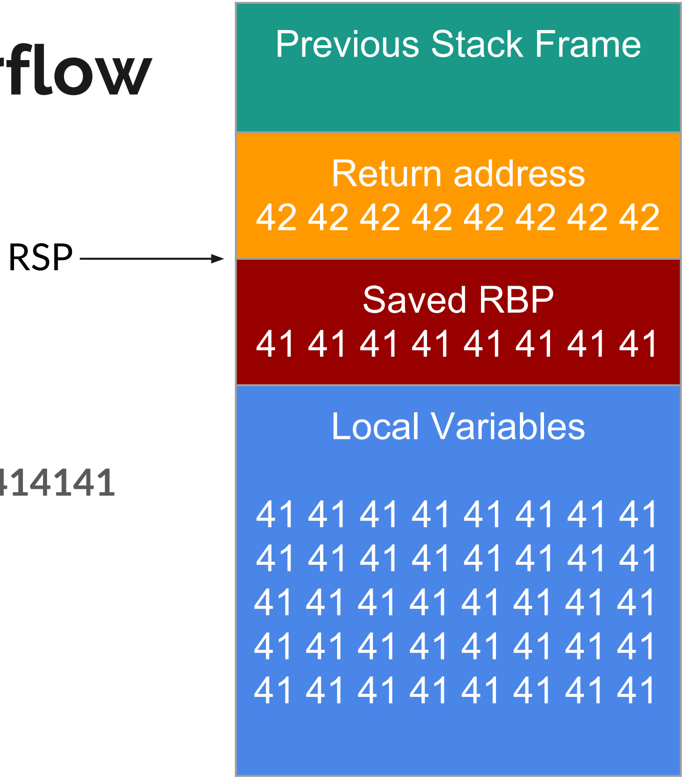
41 41 41 41 41 41 41 41
41 41 41 41 41 41 41 41
41 41 41 41 41 41 41 41
41 41 41 41 41 41 41 41
41 41 41 41 41 41 41 41

Buffer Overflow

leave:

mov rsp, rbp
pop rbp

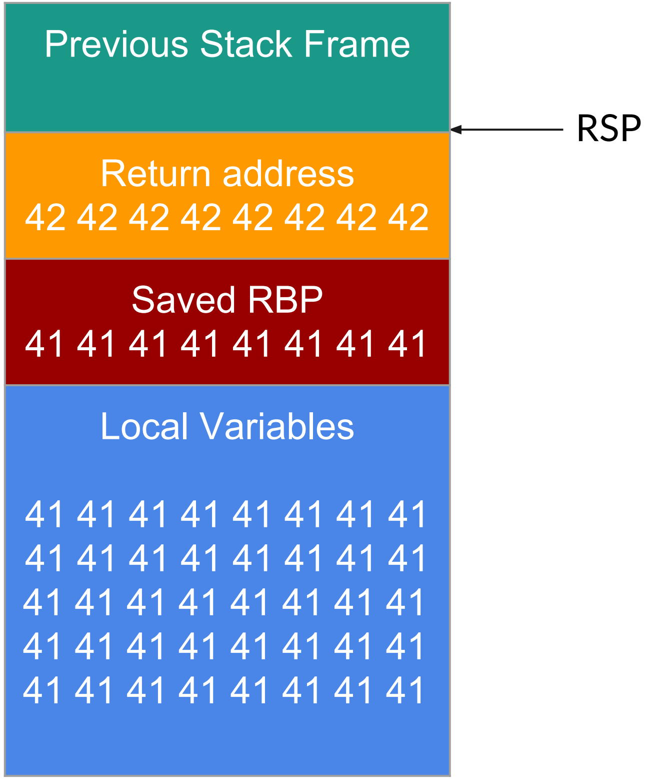
RBP = 0x4141414141414141



Buffer Overflow

```
ret:
        pop rip
```

RIP = 0x4242424242424242



Previous Stack Frame

RSP

Return address
42 42 42 42 42 42 42 42

Saved RBP
41 41 41 41 41 41 41 41

Local Variables

41 41 41 41 41 41 41 41
41 41 41 41 41 41 41 41
41 41 41 41 41 41 41 41
41 41 41 41 41 41 41 41
41 41 41 41 41 41 41 41

Pwntools

- Library written in python 2.7 for exploit development
- Sending raw bytes is hard, let pwntools do the work for you

https://github.com/Gallopsled/pwntools

Pwntools

```
from pwn import *
p = process(‘./binary’)                             # Opens a connection to a binary
p = remote(‘wargames.osiris.cyber.nyu.edu’, 1337)   # Opens a connection to a binary running remotely
p.recvline()                                        # Receives data until a newline
p.recvuntil(‘the’)                                  # Receives data until specified string
p.send(‘hello’)                                     # Sends data
p.sendline(‘hello’)                                 # Sends data with newline appended
p.interactive()                                     # Allows you to manually enter data and also prints
                                                      out all received data
```

Pwntools

```
context.log_level = ‘debug’              # prints out all bytes (useful for debugging)
p64(0x41414141)                          # converts an integer to string equivalent
u64(‘AAAAAAAA’)                          # converts string to integer equivalent
pause()                                  # pauses execution
gdb.attach(p, '''break main              # creates a gdb session
                 continue''')
```

Demo

wargames.osiris.cyber.nyu.edu

Shellcode

- What do we do without a convenient give shell function?

- Instructions are just values in memory
- Can’t we just write our own instructions?

Shellcode

```
xor eax, eax
mov rbx, 0xFF978CD091969DD1
neg rbx
push rbx
push rsp
pop rdi
cdq
push rdx
push rdi
push rsp
pop rsi
mov al, 0x3b
syscall
```

```
"\x31\xc0\x48\xbb\xd1\x9d\x96\x91\xd0\x8c\x97\xff\x48\xf7
 \xdb\x53\x54\x5f\x99\x52\x57\x54\x5e\xb0\x3b\x0f\x05"
```

The instructions in hex string format

Syscalls

- Used by userland programs to ask kernel to do something
  - Mostly wrapped in libc functions
- syscall / int 0x80

Syscall #	Param 1	Param 2	Param 3	Param 4	Param 5	Param 6
RAX	RDI	RSI	RDX	R10	R9	R8

- http://blog.rchapman.org/posts/Linux_System_Call_Table_for_x86_64/

Execve

- Tells the kernel to execute a program

Syscall #	RDI	RSI	RDX
0x3b	/bin/bash	0	0

Shellcode

- Where’s our shellcode though?

- The address of the stack changes every time we run the program
- The address of global variables stay consistent (the data section)

Nop Slide

- If we only know approximate location of our shellcode, we can guess
- nop (0x90)
  - Perform no operation
- Have a bunch of nops before our shellcode
  - Only have to get close, we slide down the nops until our shellcode

Demo

wargames.osiris.cyber.nyu.edu

Format String

- Strings with indicators that specify where data should be converted to characters
  - printf(‘this is an int: %d’, 100); → ‘this is an int: 100’

Character	Usage
d	signed integer
u	unsigned integer
x	hex number
s	string
c	char

Character	Size
hh	1 byte
h	2 byte
l	4 byte
ll	8 byte

Format String

- Variable number of arguments
- Arguments to format string functions taken from stack
- What happens if we control the format string?

Format String

- printf(user_input)
    - Does not expect any arguments (no extra room made on stack)
- user_input = ‘%s%s%s’
    - Starts printing values off the stack

Demo

wargames.osiris.cyber.nyu.edu

Format String

- We can also write with printf but with limitations
  - Can only write to addresses that exist on the stack
- %n writes the number of characters

- %xd will get an integer but with a minimum width of x
  - Useful for large number of characters
- %n$d will get the nth position argument
  - Allows you to more easily control where to write

Pwntools Auto Format

- http://docs.pwntools.com/en/stable/fmtstr.html
- Specify values and corresponding addresses
- Does not work if your input is not stored on the stack
  - Addresses have to exist on the stack
  - Autoformat writes them in and then uses them
- Only seems to work on 32 bit binaries

Demo

wargames.osiris.cyber.nyu.edu

Tips for Identifying Vulnerabilities

- Track how input is used
- Look at all available functions
    - A lot of the functions you'll see are setup or libc functions
        - You'll learn to recognize these
- Watch for potentially vulnerable functions (gets, printf, scanf, etc.)
- Pay attention to size checks (off by 1s)

Workshop

wargames.osiris.cyber.nyu.edu