Montrehack | ROP 101

A Hands-On Introduction to Return Oriented Programming

Twitter @alxbl_sec | **Keybase** @alxbl

Challenges (Head Start)

ctf.segfault.me

Details & Downloads on Port 80

X64 function_call(rdi, rsi, rdx, rcx, r8, r9)

Gadget Hunting https://github.com/JonathanSalwan/ROPgadget

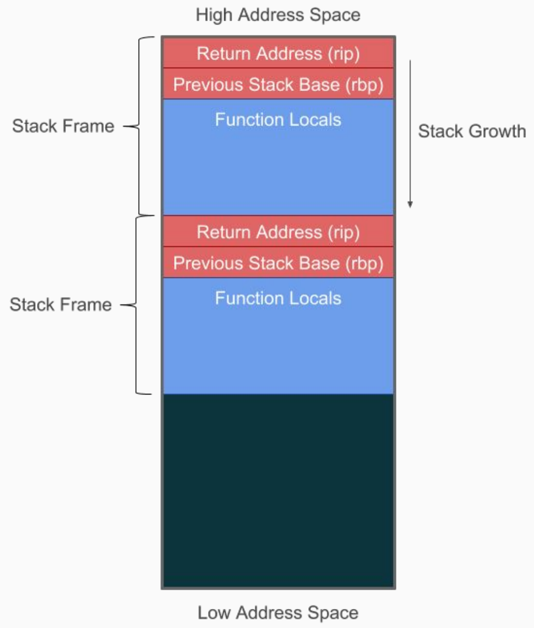
Introduction

- Classic Stack Smashing Recap
- Data Execution Prevention (NX)
- Return Oriented Programming

Classic Stack Smashing Recap

- Recall: Stack Grows Down (High -> Low)

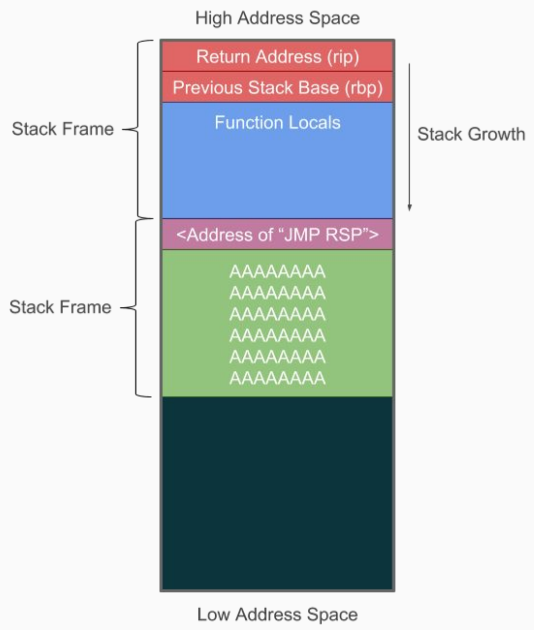
- Unbounded Read to Stack Buffer
- Overwrite the stored return address
- Find a way to JMP to stack
- ????
- Break Stuff



Classic Stack Smashing Recap

- Recall: Stack Grows Down (High -> Low)

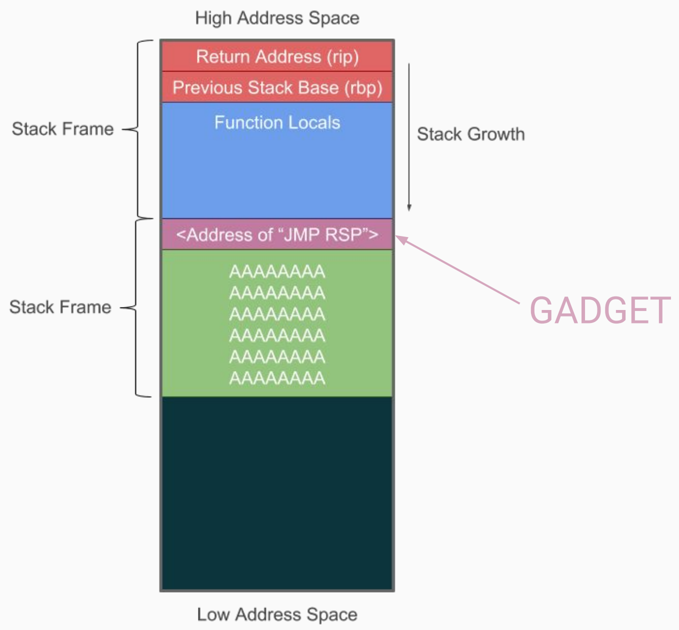
- Unbounded Read to Stack Buffer
- Overwrite the stored return address
- Find a way to JMP to stack
- ????
- Break Stuff



Classic Stack Smashing Recap

- Recall: Stack Grows Down (High -> Low)

- Unbounded Read to Stack Buffer
- Overwrite the stored return address
- Find a way to JMP to stack
- ????
- Break Stuff

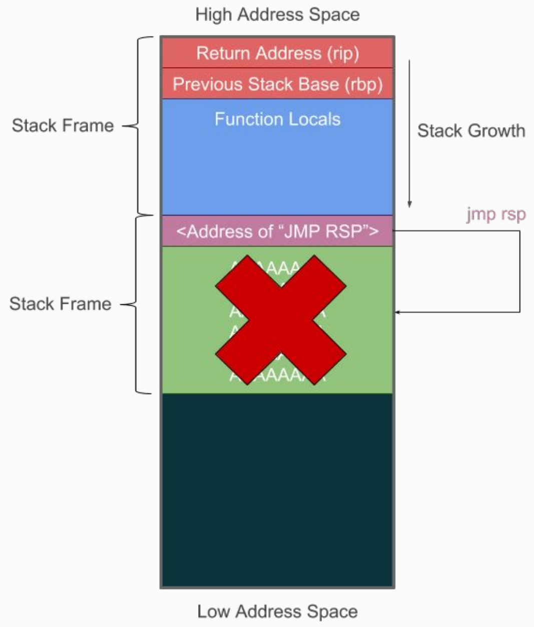


Introduction

- Classic Stack Smashing Recap
- Data Execution Prevention (NX)
- Return Oriented Programming

Data Execution Prevention (NX)

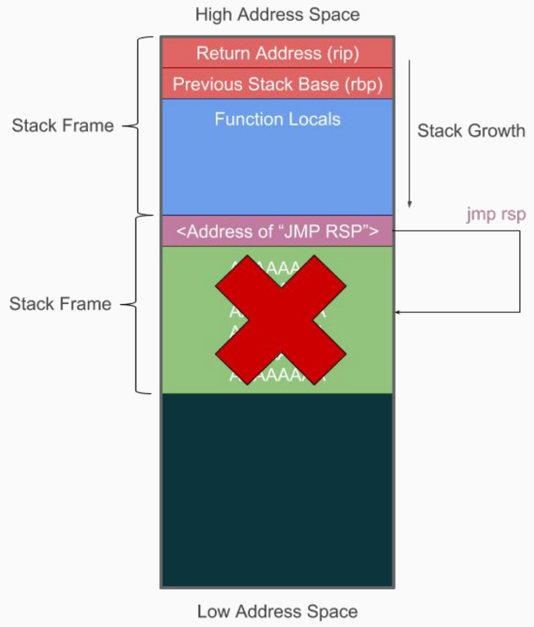
- Called No Execute (NX) on Linux
- Stack memory cannot be executed
- **Even** with “jmp rsp”
- Segmentation Fault



Data Execution Prevention (NX)

- Called No Execute (NX) on Linux
- Stack memory cannot be executed
- **Even** with “jmp rsp”
- Segmentation Fault

- => We only control return address(es)

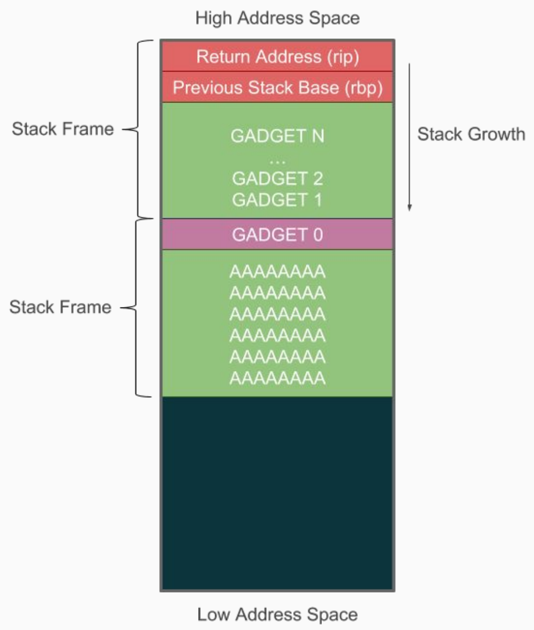


Introduction

- Classic Stack Smashing Recap
- Data Execution Prevention (NX)
- Return Oriented Programming

Return Oriented Programming

- Use existing code in the binary
- Leverage "**ret**" to control flow of execution
- Stack acts as a list of locations to execute
- … a list of **gadgets**.
- Okay, but more concretely?



Return Oriented Programming (Visualization)

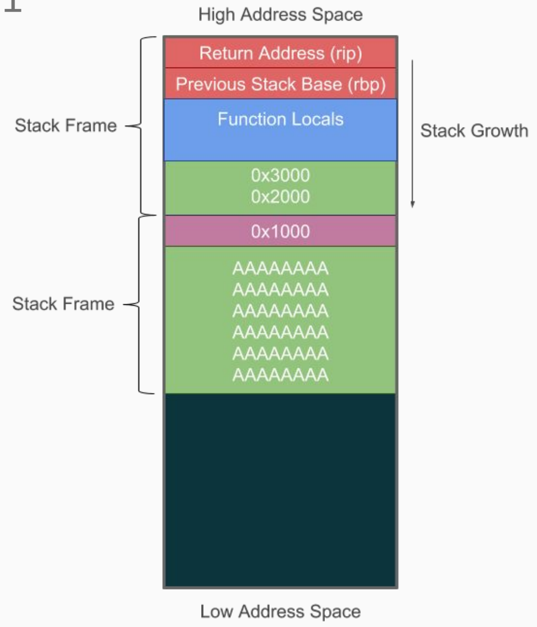
Gadgets

```
0x0100 xor rax, rax
0x0101 ret

0x0200 mov rdx, 0x40
0x0201 mov rax, 0x02
0x0202 ret
```

```
0x0300 mov rdi, rax
0x0301 call print_flag
0x0302 ret ; optional
```

Shellcode Desired

```
0x0000 xor rax, rax ; rax=0
0x0010 mov rdx, 0x40
0x0020 mov rax, 0x02
0x0030 add rax, rdx
0x0040 mov rdi, rax ; arg1=0x42
0x0050 call print_flag
```

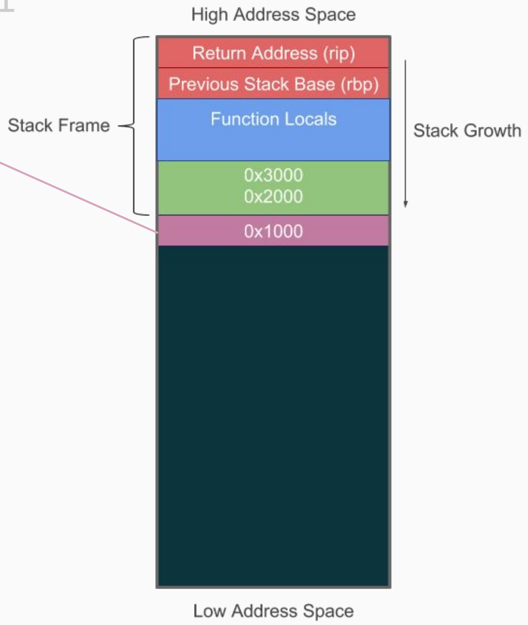


Return Oriented Programming (Visualization)

Gadgets

```
0x0100 xor rax, rax
0x0101 ret

0x0200 mov rdx, 0x40
0x0201 mov rax, 0x02
0x0202 ret
```

```
0x0300 mov rdi, rax
0x0301 call print_flag
0x0302 ret ; optional
```



Shellcode Desired

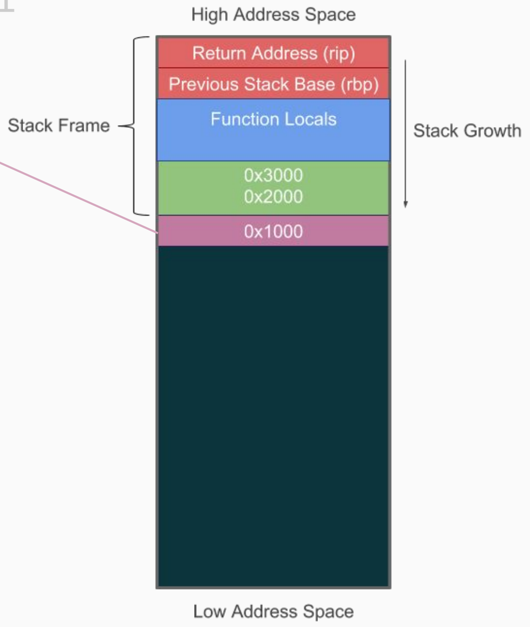
```
0x0000 xor rax, rax ; rax=0
0x0010 mov rdx, 0x40
0x0020 mov rax, 0x02
0x0030 add rax, rdx
0x0040 mov rdi, rax ; arg1=0x42
0x0050 call print_flag
```

Return Oriented Programming (Visualization)

Gadgets

```
0x0100 xor rax, rax
0x0101 ret

0x0200 mov rdx, 0x40
0x0201 mov rax, 0x02
0x0202 ret
```

```
0x0300 mov rdi, rax
0x0301 call print_flag
0x0302 ret ; optional
```



Shellcode Desired

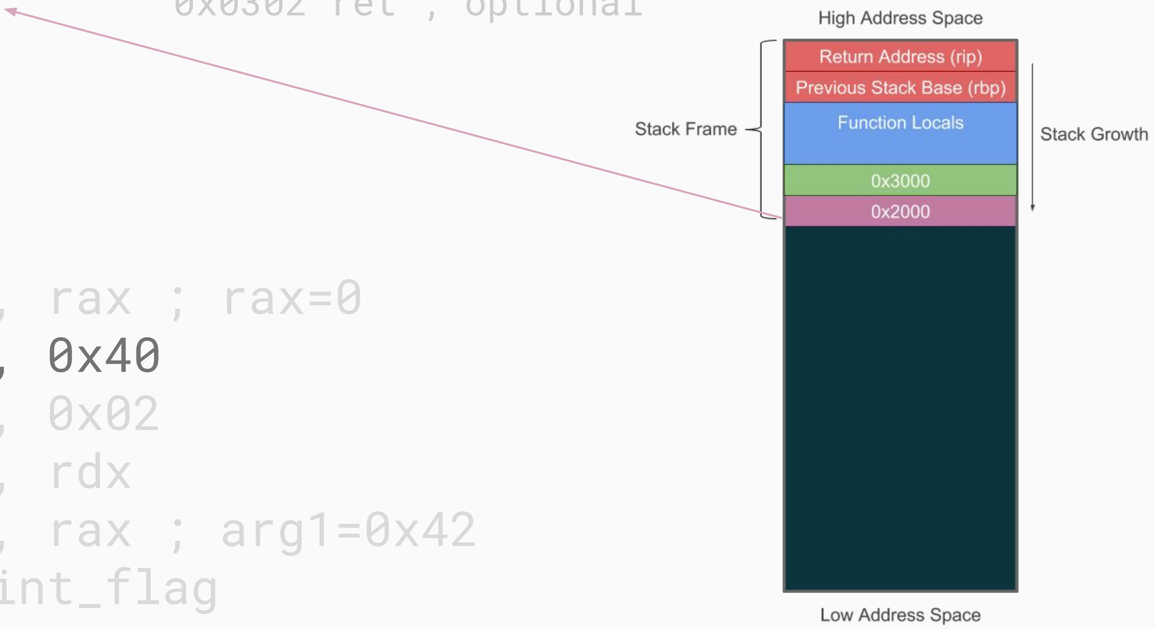
```
0x0000 xor rax, rax ; rax=0
0x0010 mov rdx, 0x40
0x0020 mov rax, 0x02
0x0030 add rax, rdx
0x0040 mov rdi, rax ; arg1=0x42
0x0050 call print_flag
```

Return Oriented Programming (Visualization)

Gadgets

```
0x0100 xor rax, rax
0x0101 ret

0x0200 mov rdx, 0x40
0x0201 mov rax, 0x02
0x0202 ret
```

```
0x0300 mov rdi, rax
0x0301 call print_flag
0x0302 ret ; optional
```



Shellcode Desired

```
0x0000 xor rax, rax ; rax=0
0x0010 mov rdx, 0x40
0x0020 mov rax, 0x02
0x0030 add rax, rdx
0x0040 mov rdi, rax ; arg1=0x42
0x0050 call print_flag
```

Return Oriented Programming (Visualization)

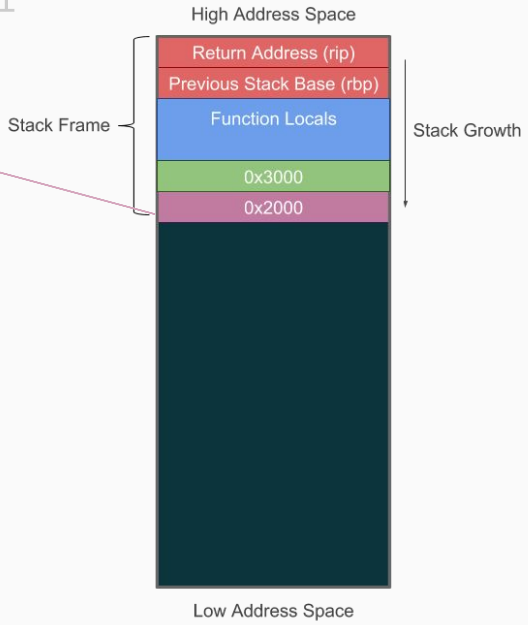
Gadgets

```
0x0100 xor rax, rax
0x0101 ret

0x0200 mov rdx, 0x40
0x0201 mov rax, 0x02
0x0202 ret
```

```
0x0300 mov rdi, rax
0x0301 call print_flag
0x0302 ret ; optional
```

Shellcode Desired

```
0x0000 xor rax, rax ; rax=0
0x0010 mov rdx, 0x40
0x0020 mov rax, 0x02
0x0030 add rax, rdx
0x0040 mov rdi, rax ; arg1=0x42
0x0050 call print_flag
```

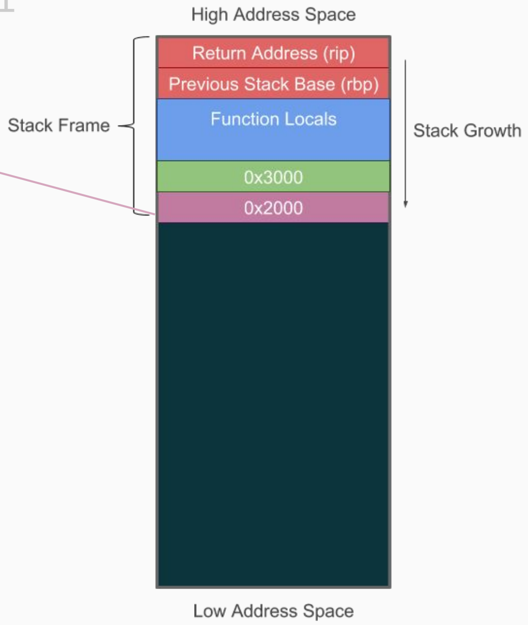


Return Oriented Programming (Visualization)

Gadgets

```
0x0100 xor rax, rax
0x0101 ret

0x0200 mov rdx, 0x40
0x0201 mov rax, 0x02
0x0202 ret
```

```
0x0300 mov rdi, rax
0x0301 call print_flag
0x0302 ret ; optional
```



Shellcode Desired

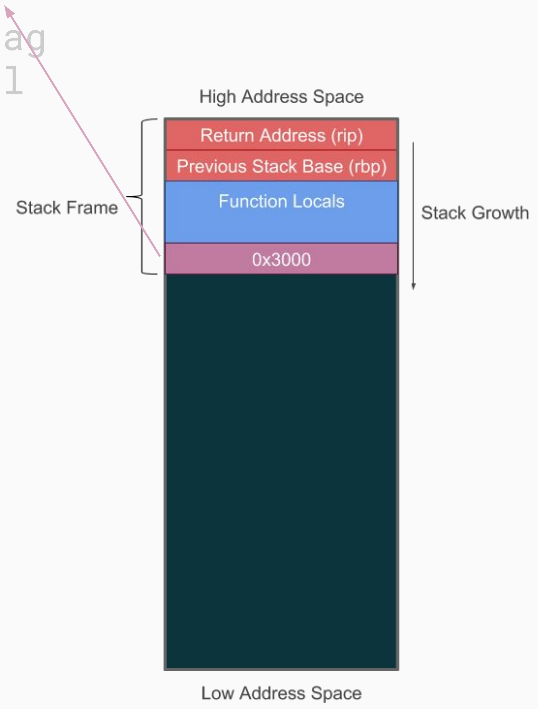
```
0x0000 xor rax, rax ; rax=0
0x0010 mov rdx, 0x40
0x0020 mov rax, 0x02
0x0030 add rax, rdx
0x0040 mov rdi, rax ; arg1=0x42
0x0050 call print_flag
```

Return Oriented Programming (Visualization)

Gadgets

```
0x0100 xor rax, rax
0x0101 ret

0x0200 mov rdx, 0x40
0x0201 mov rax, 0x02
0x0202 ret
```

```
0x0300 mov rdi, rax
0x0301 call print_flag
0x0302 ret ; optional
```



Shellcode Desired

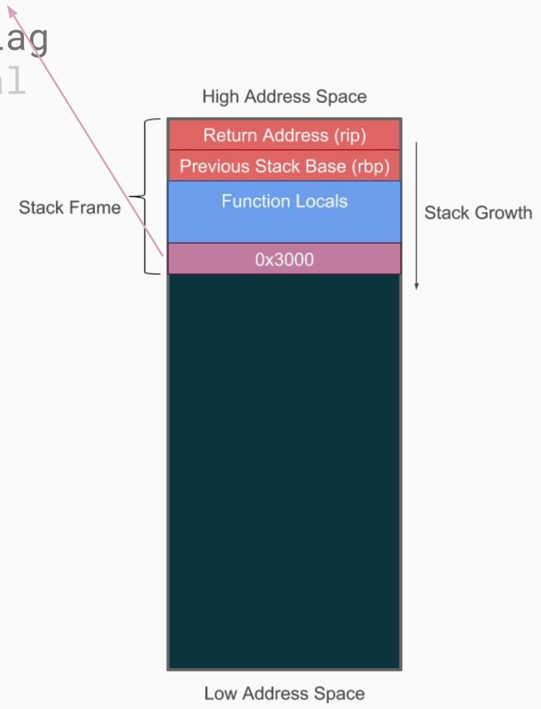
```
0x0000 xor rax, rax ; rax=0
0x0010 mov rdx, 0x40
0x0020 mov rax, 0x02
0x0030 add rax, rdx
0x0040 mov rdi, rax ; arg1=0x42
0x0050 call print_flag
```

Return Oriented Programming (Visualization)

Gadgets

```
0x0100 xor rax, rax
0x0101 ret

0x0200 mov rdx, 0x40
0x0201 mov rax, 0x02
0x0202 ret
```

```
0x0300 mov rdi, rax
0x0301 call print_flag
0x0302 ret ; optional
```



Shellcode Desired

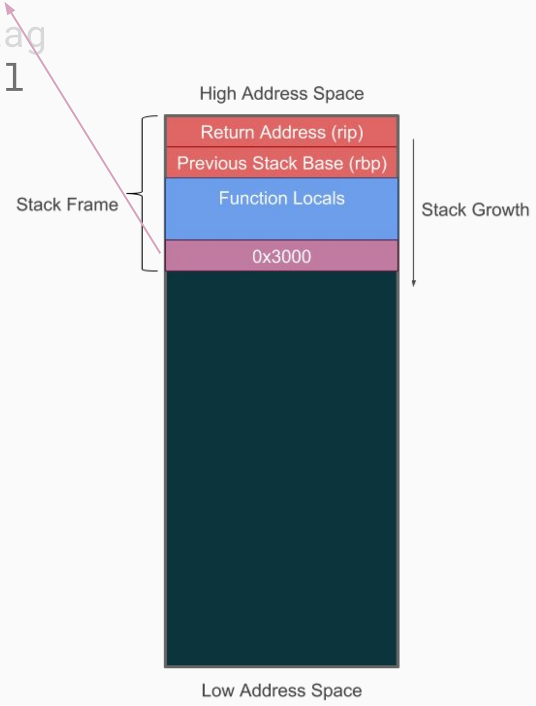
```
0x0000 xor rax, rax ; rax=0
0x0010 mov rdx, 0x40
0x0020 mov rax, 0x02
0x0030 add rax, rdx
0x0040 mov rdi, rax ; arg1=0x42
0x0050 call print_flag
```

Return Oriented Programming (Visualization)

Gadgets

```
0x0100 xor rax, rax
0x0101 ret

0x0200 mov rdx, 0x40
0x0201 mov rax, 0x02
0x0202 ret
```

```
0x0300 mov rdi, rax
0x0301 call print_flag
0x0302 ret ; optional
```



Shellcode Desired

```
0x0000 xor rax, rax ; rax=0
0x0010 mov rdx, 0x40
0x0020 mov rax, 0x02
0x0030 add rax, rdx
0x0040 mov rdi, rax ; arg1=0x42
0x0050 call print_flag
```

Return Oriented Programming (Visualization)

Gadgets

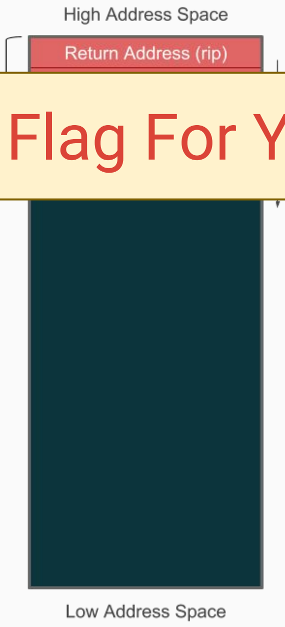
```
0x0100 xor rax, rax
0x0101 ret

0x0200 mov rdx, 0x40
0x0201 mov rax, 0x02
0x0202
```

```
0x0300 mov rdi, rax
0x0301 call print_flag
0x0302 ret ; optional
```

0x42: SEGV (si_error=ENOFLAG): No Flag For You!

Shellcode Desired

```
0x0000 xor rax, rax ; rax=0
0x0010 mov rdx, 0x40
0x0020 mov rax, 0x02
0x0030 add rax, rdx
0x0040 mov rdi, rax ; arg1=0x42
0x0050 call print_flag
```

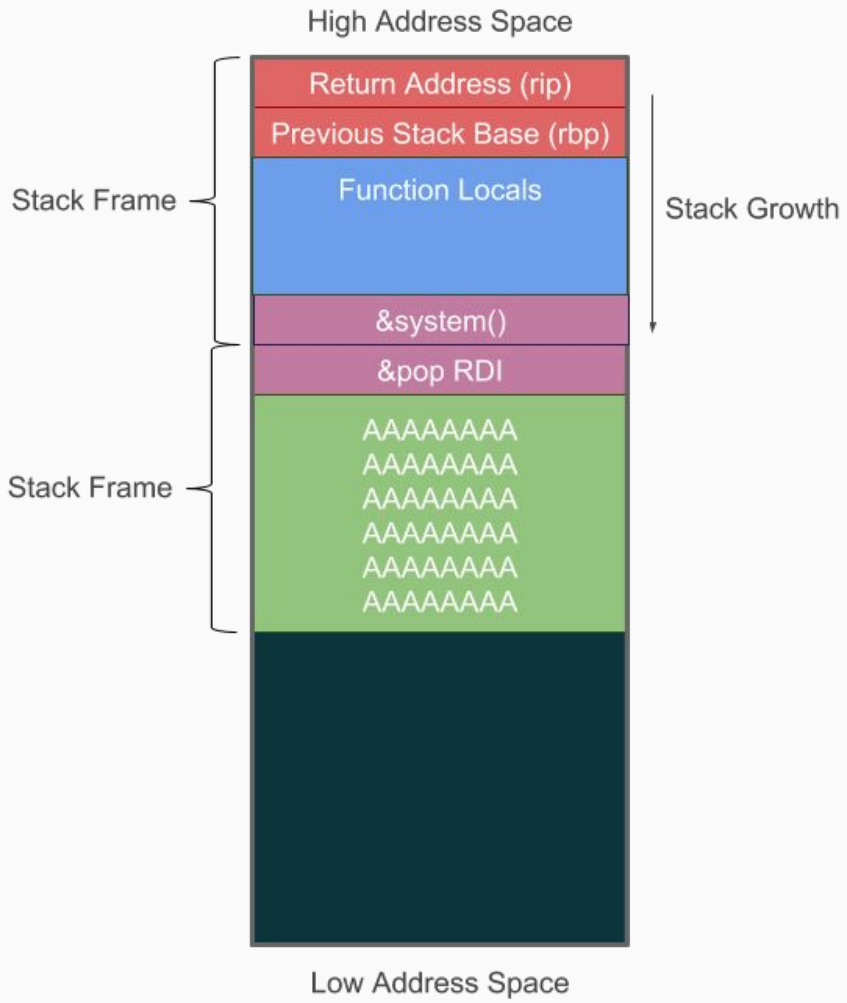


Techniques

- Return to libc (ret2libc)
- Stack Pivoting
- Padding

ret2libc

- **When:** libc imports are available
- Avoids complicated ROP chains
- Just call **system()**
- Interesting Gadgets
  - pop rdi (arg 1)
  - address of system()

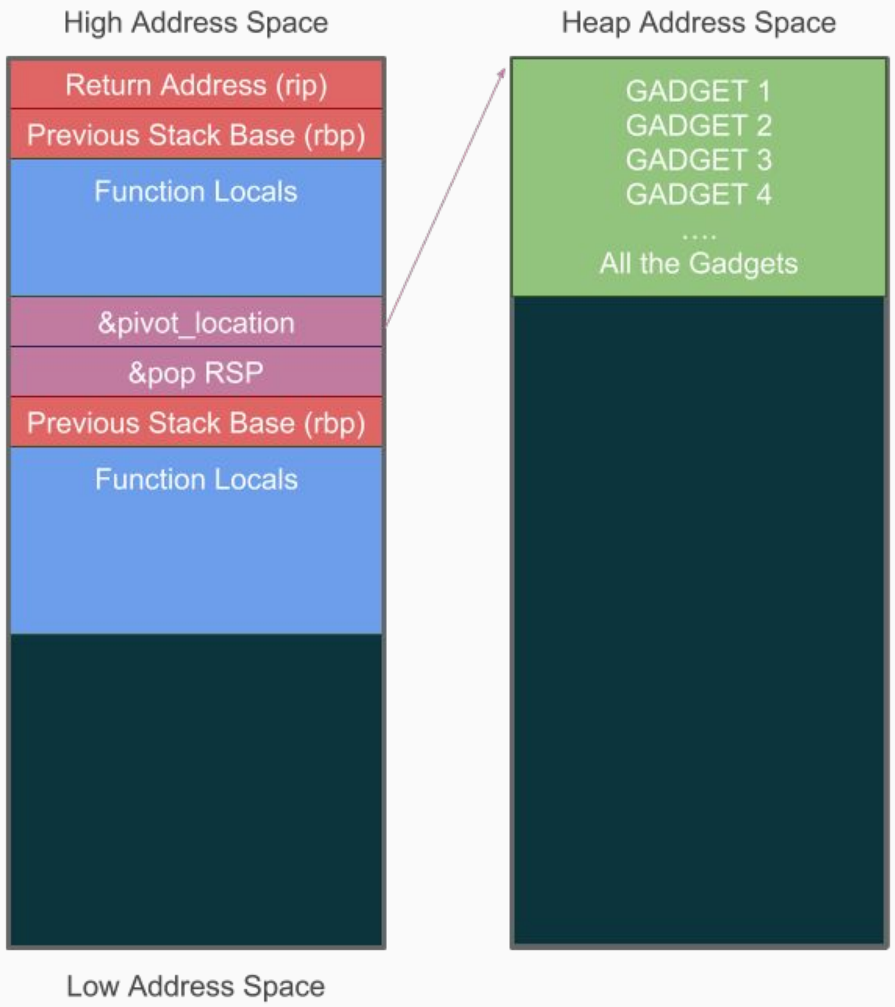


Techniques

- Return to libc (ret2libc)
- Stack Pivoting
- Padding

Stack Pivoting

- **When:** Not enough space on stack for full chain
- Put full chain **elsewhere**
  - Other Stack Frame
  - Heap
- Set stack pointer to **elsewhere**
- Chain continues from there
- Interesting Gadgets
  - pop rsp
  - sub rsp, N
  - add rsp, N



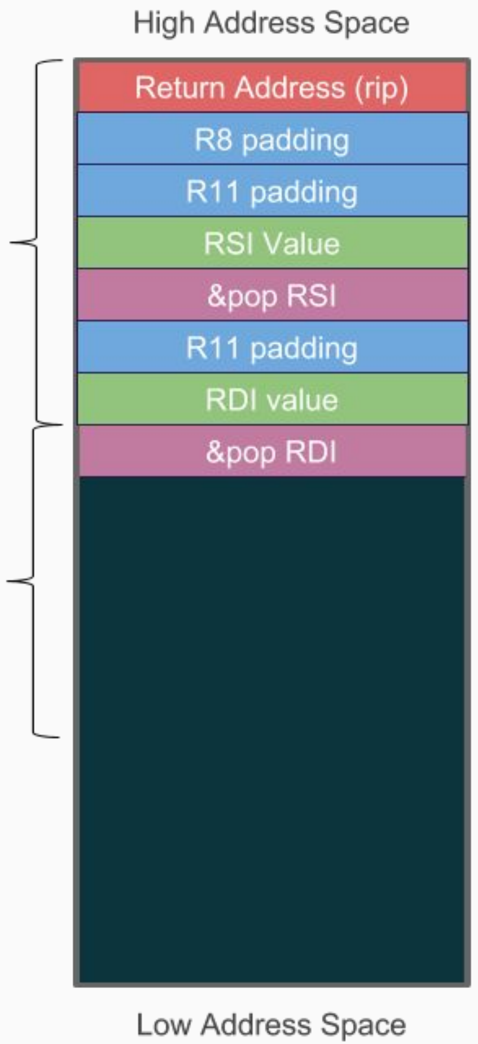
Techniques

- Return to libc (ret2libc)
- Stack Pivoting
- Padding

Padding

- **When:** Gadget has side effects
- The gadget might pop more than one register
- It doesn't mean you can't use it
- Add useless entries on the stack for those unnecessary registers

Want
rdi, rsi
Gadgets
pop rdi; pop r11; ret
pop rsi; pop r11; pop r8; ret



Challenges

What You’ve All Been Waiting For.

Challenges

ctf.segfault.me

Details & Downloads on Port 80

X64 function_call(rdi, rsi, rdx, rcx, r8, r9)

Gadget Hunting https://github.com/JonathanSalwan/ROPgadget

Hint #1

Look at the state of the process (registers) at the crash site

Hint #2

Look at the state of the process (registers) at the crash site

Look for gadgets that give you control over function parameters

Hint #3

Look at the state of the process (registers) at the crash site

Look for gadgets that give you control over function parameters

Allocate a buffer for your shellcode, read it in and jump to it.

Resources

- RPISEC’s Modern Binary Exploitation Course
  github.com/RPISEC/MBE
- pwntools
  docs.pwntools.com/
- pwndbg
  github.com/pwndbg/pwndbg
- radare2
  radare.gitbooks.io/radare2book
- me (solution write-ups soon™)
  segfault.me
  github.com/alxbl
- Sources and Solutions (soon™)
  github.com/montrehack
- ROPGadget
  github.com/JonathanSalwan/ROPgadget