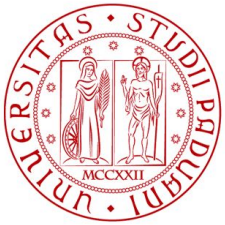
# CyberSecurity: Principle and Practice

*BSc Degree in Computer Science*
*2021-2022*

## Lesson 17: Return Oriented Programming


Prof. Mauro Conti
Department of Mathematics
University of Padua
conti@math.unipd.it
http://www.math.unipd.it/~conti/

Teaching Assistants
Luca Pajola
pajola@math.unipd.it.
Pier Paolo Tricomi
pierpaolo.tricomi@phd.unipd.it


UNIVERSITÀ DEGLI STUDI DI PADOVA

SPRITZ SECURITY & PRIVACY RESEARCH GROUP

DIPARTIMENTO MATEMATICA

# Disclaimer



All information presented here has the only purpose of teaching how reverse engineering works.

Use your mad skillz only in CTFs or other situations in which you are legally allowed to do so.

Do not hack the new Playstation. Or maybe do, but be prepared to get legal troubles 😊

# WRITE⊕EXECUTE

**W⊕X**

also referred as

No-Execute (NX, original Linux name)

Data Execution Prevention (DEP, later in Windows)

is a **mitigation to prevent code injection**

No memory mapping is writable and executable at the same time.

CPU will fault when trying to execute from NX memory :(

# BYPASSING

Can’t inject our own code... **let’s use what’s already there**!

This is called a ***code reuse attack***.

Applications don’t normally contain malicious code, so how can we do bad stuff?

# BYPASSING


noon AT AgREEd
LOCATION FIVE
mILLIOn dOLLARS
And nOBOdy
gETS HuRT

# CODE REUSE

Idea: isolate **small pieces of code (*gadgets*)** that do simple things and use them as **building blocks**.

Gadgets typically do small operations (e.g., set a register, write to memory, …)

By ***chaining* them** we can build complex payloads that do whatever we want (assuming Turing-complete gadget sets)

But **how to chain them?**

# Return-Oriented Programming (ROP)

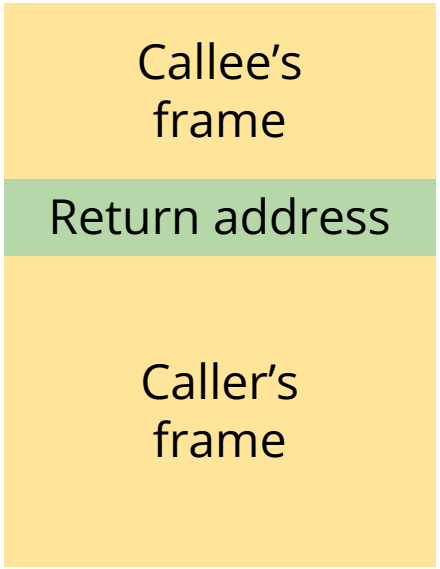

Most common code reuse technique, build on:

- Identify ROP gadgets that end with a **`ret`** instruction
- Controlling the stack, chain gadgets one after the other

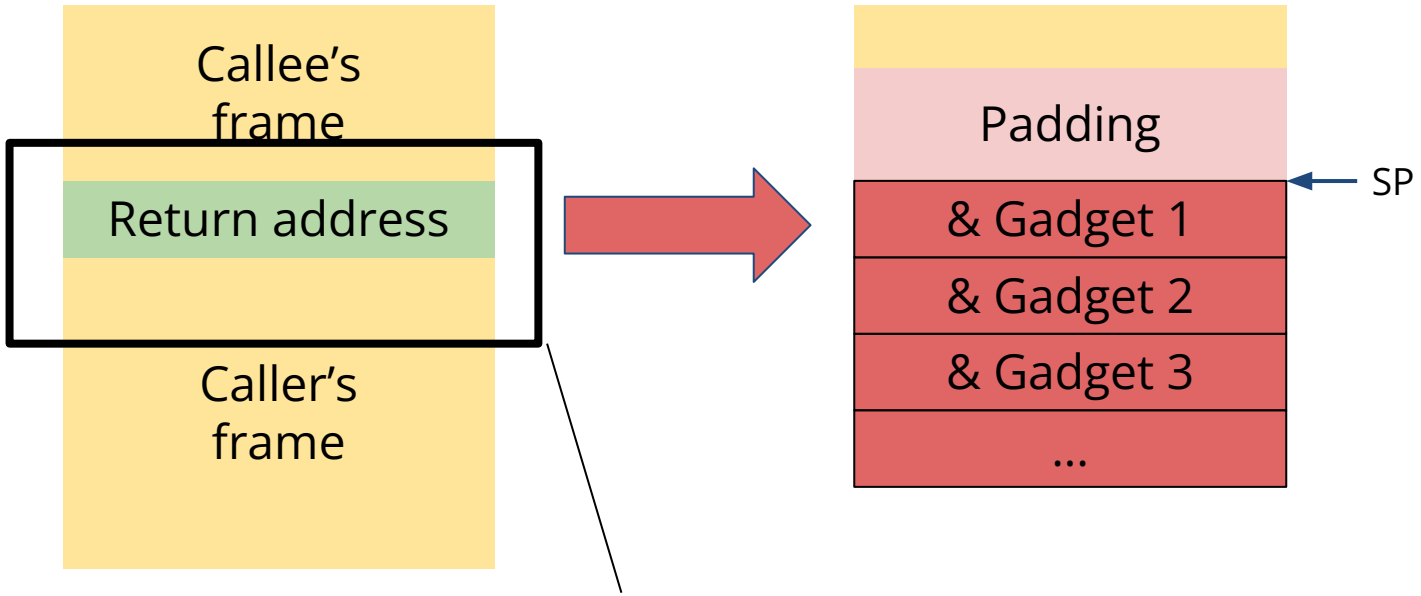

- Solar Designer, *Getting around non-executable stack (and fix)*, Bugtraq 1997
- Nergal, *The advanced return-into-lib(c) exploits: PaX case study*, Phrack 2001
- H. Shacham, *The Geometry of Innocent Flesh on the Bone: Return-into-libc without Function Calls (on the x86)*, CCS 2007

# Return-Oriented Programming (ROP)



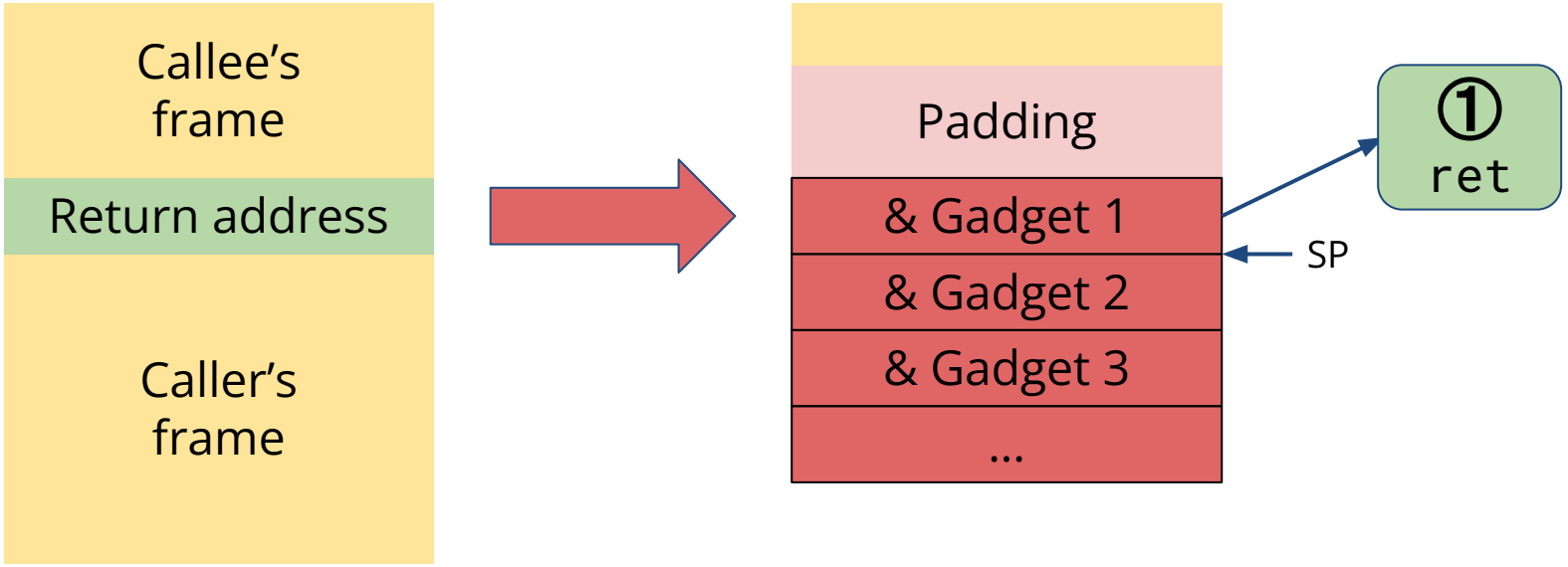


# Return-Oriented Programming (ROP)

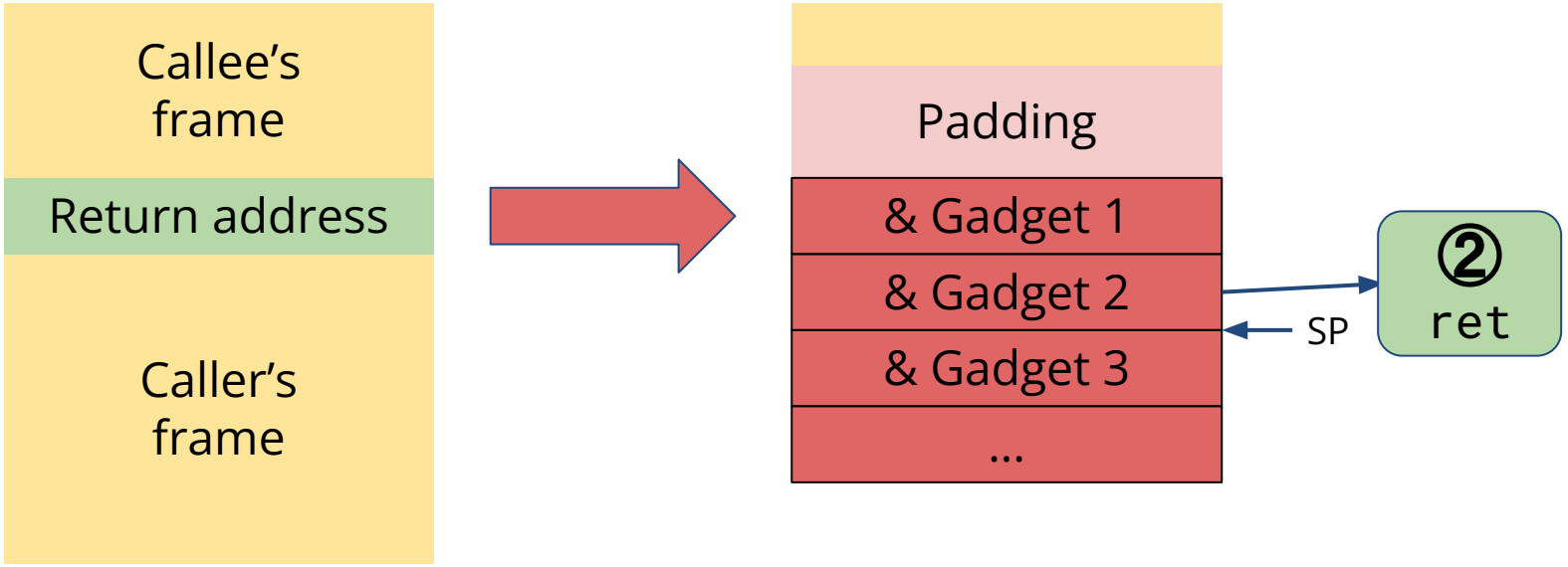



We might need a **space which is bigger than just the one of return address**… we overwrite **caller's** frame too

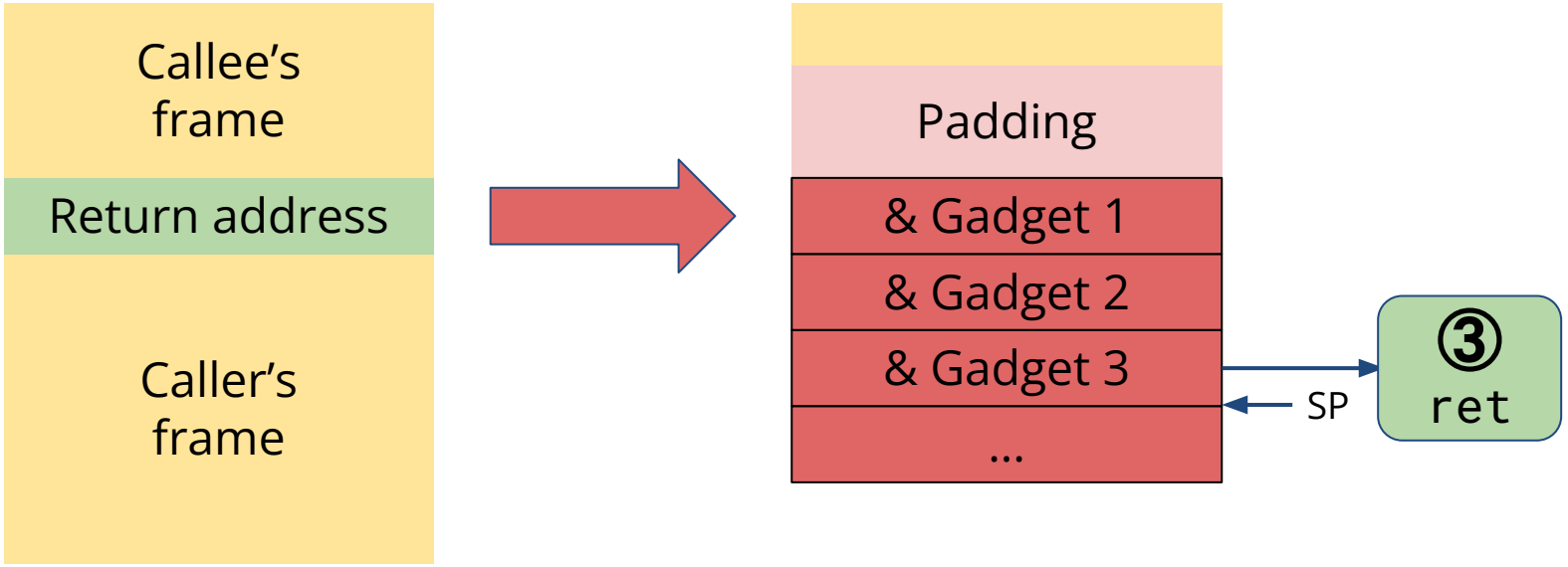
# Return-Oriented Programming (ROP)



# Return-Oriented Programming (ROP)

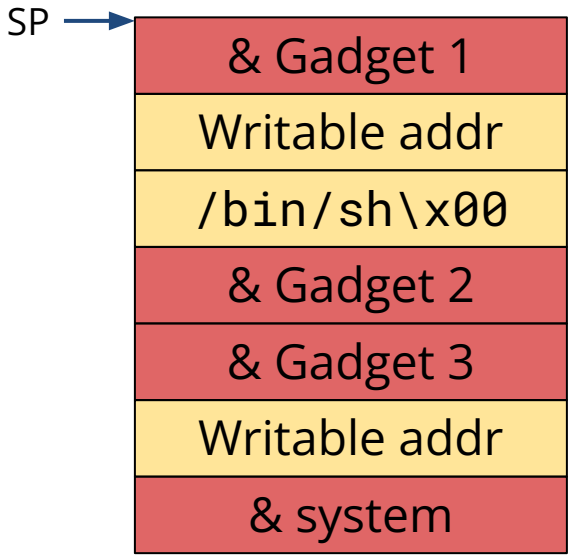


# Return-Oriented Programming (ROP)



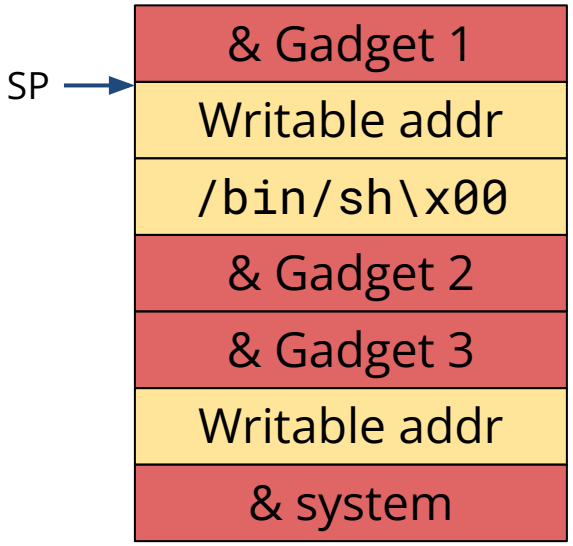
# ROP Chain Example

Goal: call `system("/bin/sh")`



# ROP Chain Example

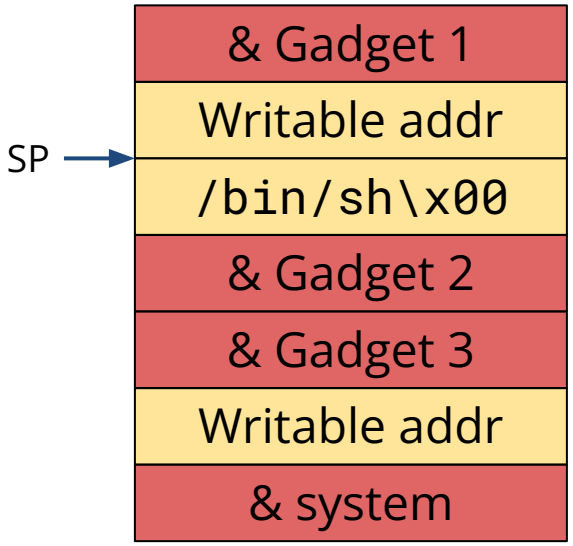
Goal: call `system("/bin/sh")`




① `pop rax; pop rbx; ret`

# ROP Chain Example

Goal: call `system("/bin/sh")`

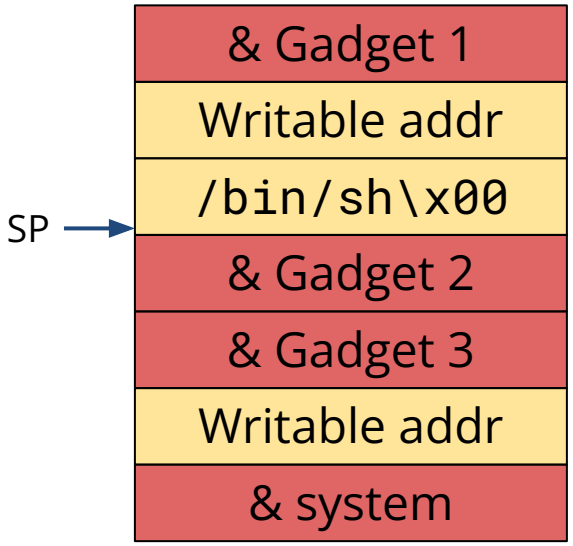



① `pop rax; pop rbx; ret`

`rax = WADDR`

# ROP Chain Example

Goal: call `system("/bin/sh")`



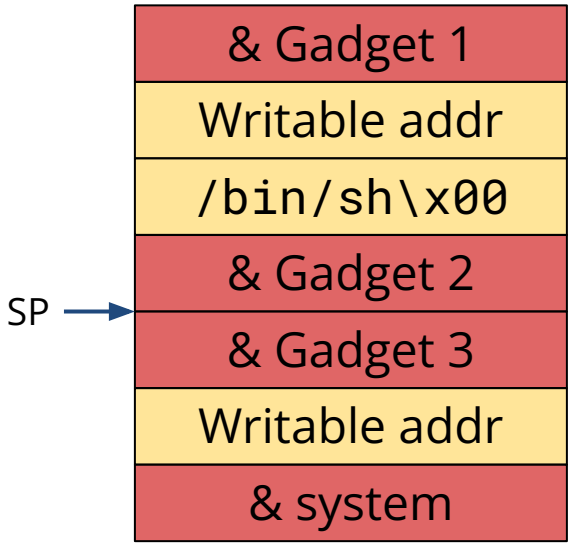

① `pop rax; pop rbx; ret`

`rax = WADDR`

`rbx = '/bin/sh\x00'`

# ROP Chain Example

Goal: call `system("/bin/sh")`

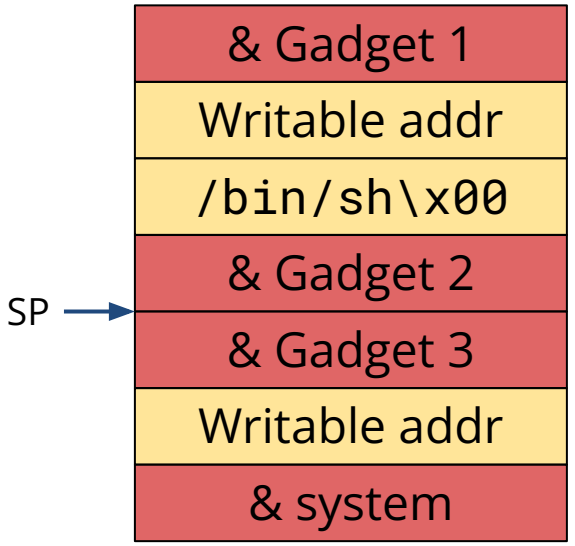



② `mov [rax], rbx; ret`

```
rax = WADDR
rbx = '/bin/sh\x00'
```

# ROP Chain Example

Goal: call `system("/bin/sh")`

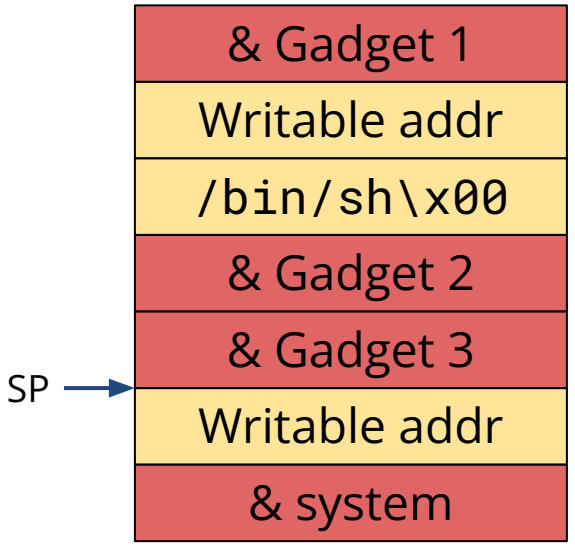



② `mov [rax], rbx; ret`

```
rax = WADDR
rbx = '/bin/sh\x00'
*WADDR = '/bin/sh\x00'
```

# ROP Chain Example

Goal: call `system("/bin/sh")`

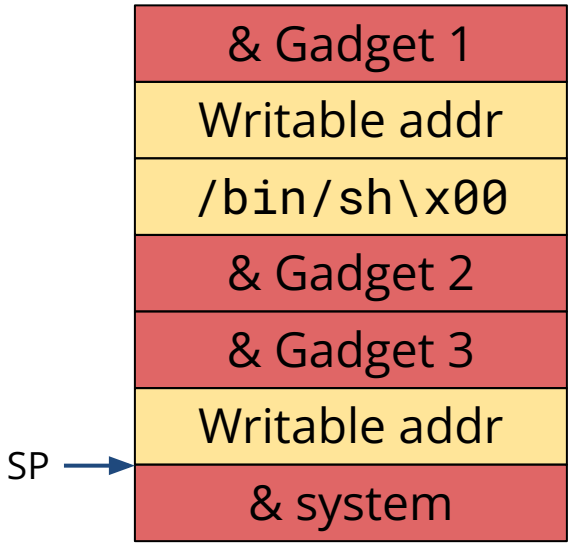



③ `pop rdi; ret`

```
rax = WADDR
rbx = '/bin/sh\x00'
*WADDR = '/bin/sh\x00'
```

# ROP Chain Example

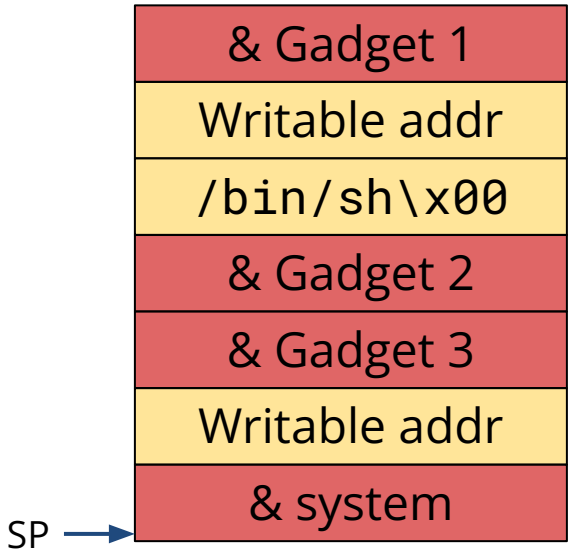
Goal: call `system("/bin/sh")`




```
rax = WADDR
rbx = '/bin/sh\x00'
*WADDR = '/bin/sh\x00'
rdi = WADDR
```

# ROP Chain Example

Goal: call `system("/bin/sh")`




④ `system("/bin/sh")`

```
rax = WADDR
rbx = '/bin/sh\x00'
*WADDR = '/bin/sh\x00'
rdi = WADDR
```

# BE CAREFUL!



In creating ROP Chains, we are playing with the Stack Pointer.
**Some ASM instructions (e.g., MOVAPS) require the stack pointer to be 16-bytes aligned to work.**

E.g., MOVAPS is used by **`system`** function

- before calling **`system`**, we should be sure to have the stack pointer (value of RSP) 16-bytes aligned (last digit should be 0, in hex).

If our stack pointer is not aligned (last digit is 8), we need to align it.
...How?

**We need something acting as a NOP** in reverse.
In PWN, a NOP **could be a gadget containing just RET instruction**.
(since an address in x64 is 8 bytes long, and a RET will increase the stack pointer by 8 bytes, we can align RSP again)

# How to find Gadgets?



ROPgadget -- binary [binary] | grep “what you need”

ROPgadget is a tool usually provided with radare2 (https://github.com/JonathanSalwan/ROPgadget)

Use grep to filter the output for what you want to search

E.g. to find gadgets using rax, you can do:
ROPgadget --binary ./a.out | grep “rax”

Other tools:

- ropper (https://scoding.de/ropper/)
- Pwntools rop (https://docs.pwntools.com/en/stable/rop/rop.html)
- Many others...

# Exercises



1) I'll let you in on a secret; a useful string "/bin/cat flag.txt" is present in this binary, as is a call to system(). It's just a case of finding them and chaining them together to make the magic happen.
2) How do you make consecutive calls to a function from your ROP chain that won't crash afterwards?
3) Our favourite string "/bin/cat flag.txt" is not present this time. Can you write it in the binary?

# Questions? Feedback? Suggestions?