



# Embedded Device Exploitation 101: An Introduction to Firmware Hacking

Benjamin Kollenda  
embo++ 2023, 23.03.2023



## Embedded RE - Recap

Deeply embedded devices

emBO++



# How to break things



1. Know it
2. Dump it
3. Analyze it
4. Break it

# How to break things

# emBO++

1. Know it
2. Dump it
3. Analyze it
4. Break it



nRF51802

Multiprotocol *Bluetooth*® low energy/2.4 GHz RF System on Chip

Preliminary Product Specification v0.7

Key Features

Applications

- RSSI (1 dB resolution)
- ARM® Cortex™-M0 32 bit processor

## ARM®v6-M Architecture Reference Manual

- S1
- M0

- 16 kB RAM

- Cortex-M0 Programmable Peripheral Interconnect (PPI)
- Quadrature Decoder (QDEC)
- AES HW encryption
- Real Time Counter (RTC)
- QFN48 package, 6 x 6 mm

# How to break things

# emBO++

1. Know it ✓
2. Dump it
3. Analyze it
4. Break it



## How to break things

1. Know it ✓
2. Dump it ✓
3. Analyze it
4. Break it

# emBO++

```
0000120 e7f0
0000130 e7e8
0000140 4825
0000150 290d
0000160 2113
0000170 2000
0000180 2000
0000190 2001
00001a0 2000
00001b0 490b
00001c0 d009
00001d0 07c0
00001e0 0500
00001f0 210f
0000200 e7fe
0000210 01a5
0000220 68e3
0000230 f7ff
0000240 1e64
0000250 bc30
0000260 2a00
0000270 2a00
0000280 0000
```



`rld!"`

```
func2(ptr_Hello);
```

```
}
```



# How to break things

1. Know it ✓
2. Dump it ✓
3. Analyze it ✓
4. Break it





# Source to Firmware

C bugs to assembly bugs

# A bug's tale

## 1. Write C bug

```
while (1) {  
    // (3) read the plaintext from a prom  
    char array[32] = {0};  
    if (prompt(UART0, array) != 0) {  
        break;  
    }  
}
```

# A bug's tale

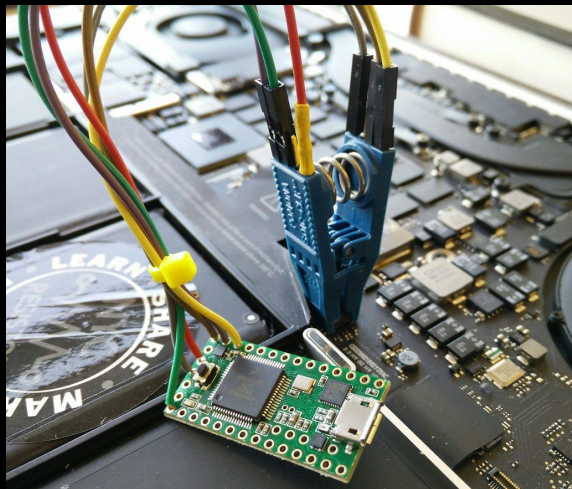


1. Write C bug
2. Compile

```
0000120 e7f0 2001 e7ee 2001 e7ec 2000 e7ea 2001
0000130 e7e8 2001 e7e6 2001 e7e4 2001 e7e2 b500
0000140 4825 69c0 b280 381d 0003 f000 f877 1718
0000150 290d 2929 2929 290f 2523 2929 1b19 111f
0000160 2113 1529 2729 291d 2001 bd00 2001 e7fc
0000170 2000 e7fa 2000 e7f8 2001 e7f6 2001 e7f4
0000180 2000 e7f2 2000 e7f0 2000 e7ee 2000 e7ec
0000190 2001 e7ea 2001 e7e8 2001 e7e6 2001 e7e4
00001a0 2000 e7e2 b510 f7ff ffca 2800 d005 480b
00001b0 490b 6048 13c8 490b 6188 f7ff ff8d 2800
00001c0 d009 2001 2105 0709 2207 0212 1889 67c8
00001d0 07c0 4905 60c8 bd10 0040 1000 ffd5 c007
00001e0 0500 4000 6c00 4000 0500 5000 4807 6802
00001f0 210f 430a 6002 4806 4780 4806 4700 e7fe
0000200 e7fe e7fe e7fe e7fe e7fe 0000 0524 4000
0000210 01a5 0000 00c1 0000 4c06 2501 4e06 e005
0000220 68e3 cc07 432b 3c0c 4798 3410 42b4 d3f7
0000230 f7ff ff4a 0278 0000 0288 0000 b430 4674
0000240 1e64 7825 1c64 42ab d304 5d63 005b 18e3
0000250 bc30 4718 461d e7f8 e002 c808 1f12 c108
0000260 2a00 d1fa 4770 4770 2000 e001 c101 1f12
0000270 2a00 d1fb 4770 0000 0288 0000 0000 2000
0000280 0000 0000 0000 0000 0000 0000 0000 0000
```

# A bug's tale

1. Write C bug
2. Compile
3. Program



# emBO++

# A bug's tale

1. Write C bug
2. Compile
3. Program
4. Ship



# emBO++

# A bug's tale

1. Write C bug
2. Compile
3. Program
4. Ship



1. Dump it
2. Analyze it
3. Find it
4. Exploit it

# A bug's tale

1. Write C bug
2. Compile
3. Program
4. Ship



1. Dump it
2. Analyze it
3. Find it
4. Exploit it



# A bug's tale

1. Write C bug
2. Compile
3. Program
4. Ship



1. Dump it ✓
2. Analyze it ✓
3. Find it
4. Exploit it

# A bug's tale

1. Write C bug
2. Compile
3. Program
4. Ship



1. Dump it ✓
2. Analyze it ✓
3. Find it
4. Exploit it



# Classic Vulnerabilities

# Classic vulnerabilities

- Buffer overflows
  - Stack

```
while (1) {  
    // (3) read the plaintext from a prompt  
    char array[32] = {0};  
    if (prompt(UART0, array) != 0) {  
        break;  
    }  
}
```

# Classic vulnerabilities

- Buffer overflows
  - Stack
  - Heap

```
void heap_overflow(char* in_buf) {  
    char* buf = malloc(32);  
    memcpy(buf, in_buf, 64);  
    free(buf);  
}
```

# Classic vulnerabilities



- Buffer overflows
  - Stack
  - Heap
- Integer overflows/underflows

```
void int_overflow(char* in_buf, size_t length) {  
    uint32_t full_size = 64 + length;  
    char* buf = malloc(512);  
    if (full_size < 512) {  
        memcpy(buf, in_buf, length + 64)  
    }  
    free(buf);  
}
```

## Classic vulnerabilities

- Buffer overflows
  - Stack
  - Heap
- Integer overflows/underflows
- Format strings

```
void format_string() {  
    char tmp_str[64] = {0};  
    char dest_buf[64] = {0};  
    uart_read(tmp_str, 64);  
    sprintf(dest_buf, tmp_str, strlen(tmp_str));  
}
```

# Classic vulnerabilities

- Buffer overflows
  - Stack
  - Heap
- Integer overflows/underflows
- Format strings
- Use after free

```
void user_after_free() {  
    struct A* a = malloc(sizeof(struct A));  
    free(a);  
    struct B* b = malloc(sizeof(struct B));  
    a->length = 512;  
    free(b);  
}
```



## Classic vulnerabilities

- Buffer overflows
  - Stack
  - Heap
- Integer overflows/underflows
- Format strings
- Use after free
- TOCTOU

```
void toctou(char* file_name) {  
    if (!is_symlink(file_name)) {  
        write_log(file_name);  
    }  
}
```

# Classic vulnerabilities

- Buffer overflows
  - Stack
  - Heap
- Integer overflows/underflows
- Format strings
- Use after free
- TOCTOU

emBO++

... and how to find them

# Finding vulnerabilities



- Code review
- Fuzzing
- Reverse engineering
- Try things

# Finding vulnerabilities

- Code review
- Fuzzing
- Reverse engineering
- Try things



# Exploiting things

1. Dump it ✓
2. Analyze it ✓
3. Find it

emBO++

# Exploiting things

1. Dump it ✓
2. Analyze it ✓
3. Find it ✓

emBO++

# Exploiting things



1. Dump it ✓
2. Analyze it ✓
3. Find it ✓
4. Exploit it

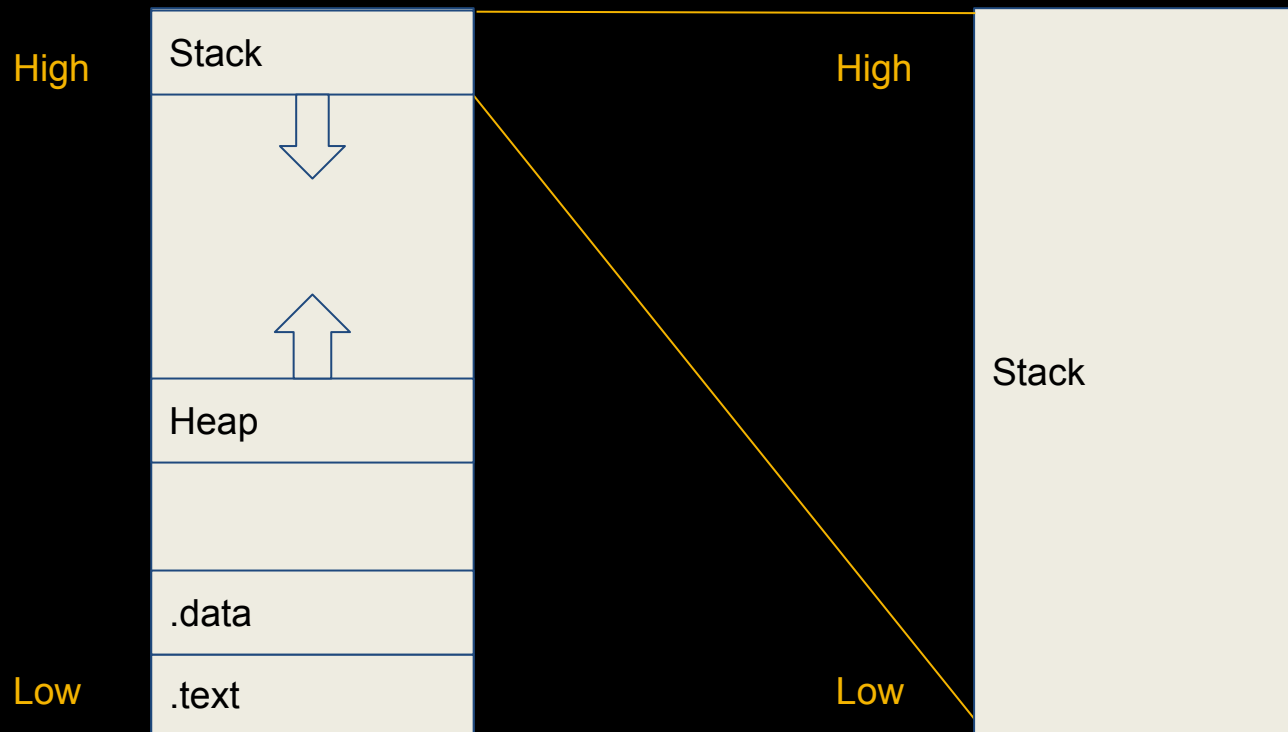




# Stack buffer overflow I

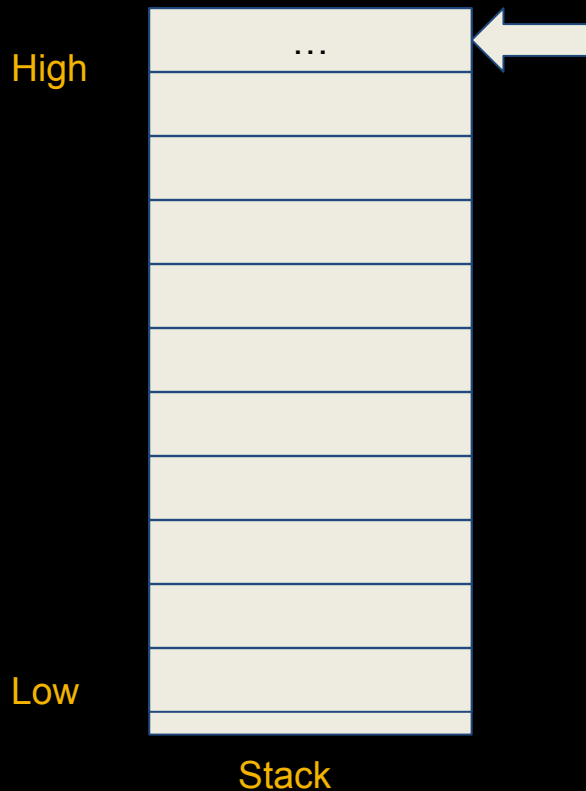
# Stacks 101

emBO++



# Stacks 101

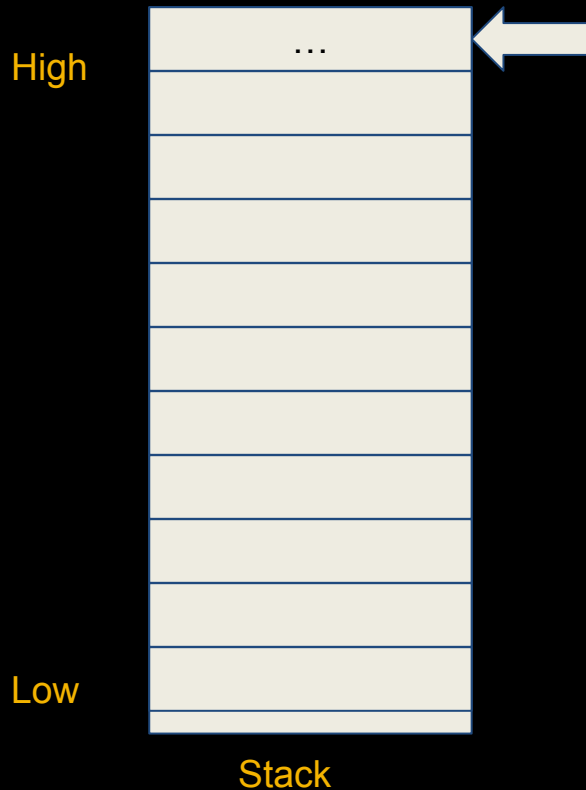
# emBO++



```
int callee(int a1, int a2, int a3, int a4, int a5) {  
    return a1 + a2 + a3 + a4 + a5;  
}  
  
int caller() {  
    int temp = 0;  
    temp = callee(1, 2, 3, 4, 5);  
    return temp;  
}
```

# Stacks 101

# emBO++

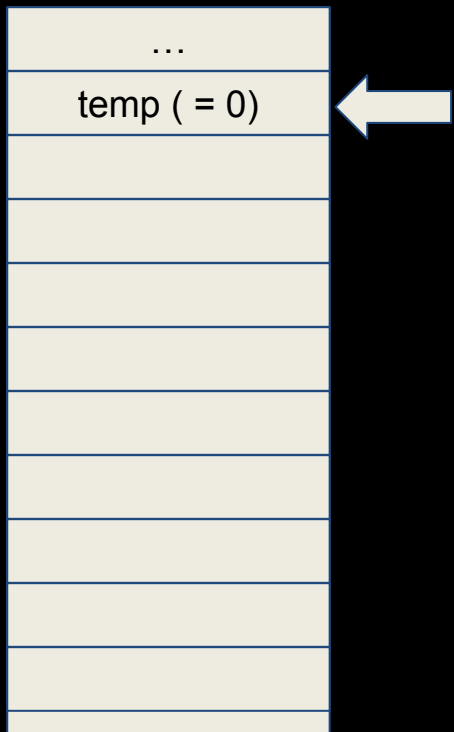


```
int callee(int a1, int a2, int a3, int a4, int a5) {  
    return a1 + a2 + a3 + a4 + a5;  
}  
  
int caller() {  
    int temp = 0; ←  
    temp = callee(1, 2, 3, 4, 5);  
    return temp;  
}
```

# Stacks 101

# emBO++

High



Low

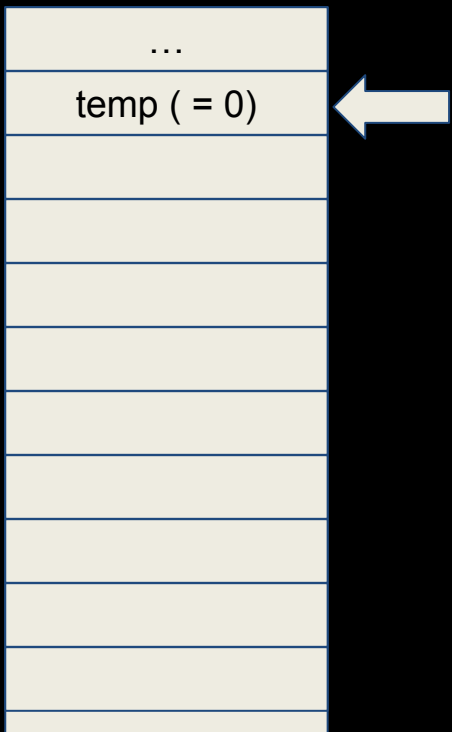
Stack

```
int callee(int a1, int a2, int a3, int a4, int a5) {  
    return a1 + a2 + a3 + a4 + a5;  
}  
  
int caller() {  
    int temp = 0;  
    temp = callee(1, 2, 3, 4, 5);  
    return temp;  
}
```

# Stacks 101

# emBO++

High



Low

Stack

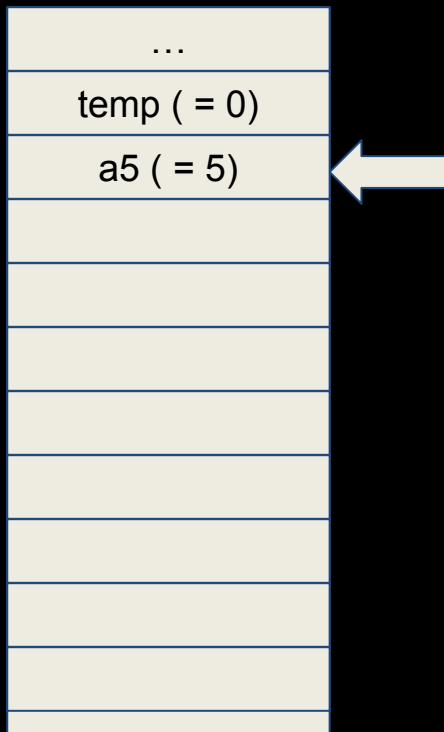
```
int callee(int a1, int a2, int a3, int a4, int a5) {  
    return a1 + a2 + a3 + a4 + a5;  
}  
  
int caller() {  
    int temp = 0;  
    temp = callee(1, 2, 3, 4, 5);  
    return temp;  
}
```

Two white arrows indicate the call sequence: one points from the 'temp = callee(1, 2, 3, 4, 5);' line in the caller function to the 'temp (= 0)' frame in the stack, and another points from the 'return temp;' line in the caller function to the 'temp (= 0)' frame in the stack.

# Stacks 101

# emBO++

High



Low

Stack

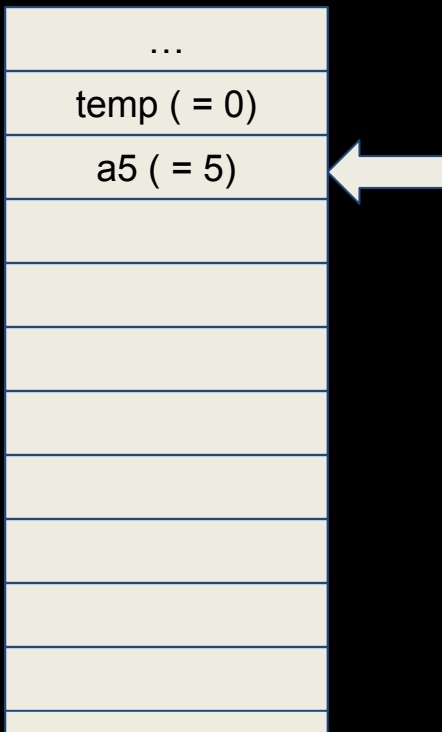
```
int callee(int a1, int a2, int a3, int a4, int a5) {  
    return a1 + a2 + a3 + a4 + a5;  
}  
  
int caller() {  
    int temp = 0;  
    temp = callee(1, 2, 3, 4, 5);  
    return temp;  
}
```

Two white arrows indicate execution flow. One arrow points from the right to the line "temp = callee(1, 2, 3, 4, 5);". Another arrow points from below to the same line.

# Stacks 101

# emBO++

High



Low

Stack

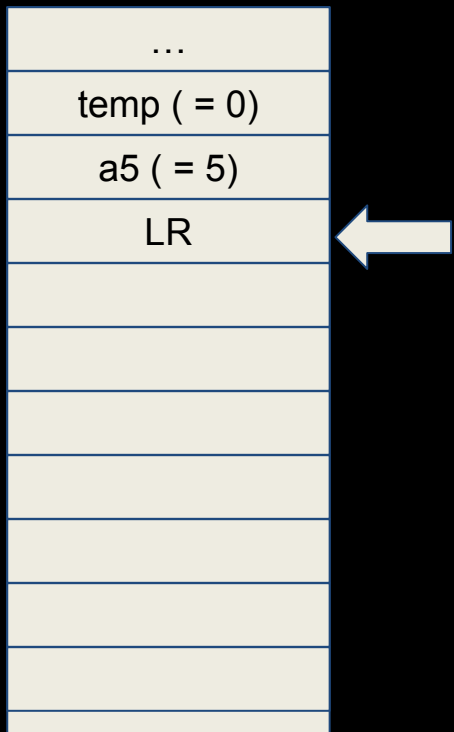
```
int callee(int a1, int a2, int a3, int a4, int a5) {  
    return a1 + a2 + a3 + a4 + a5;  
}  
  
int caller() {  
    int temp = 0;  
    temp = callee(1, 2, 3, 4, 5);  
    return temp;  
}
```



# Stacks 101

# emBO++

High



Low

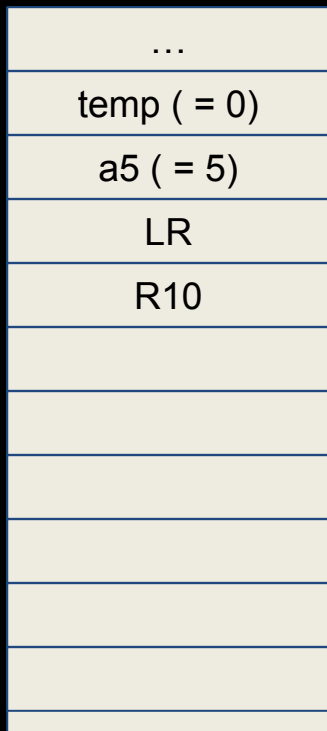
Stack

```
int callee(int a1, int a2, int a3, int a4, int a5) {  
    return a1 + a2 + a3 + a4 + a5;  
}  
  
int caller() {  
    int temp = 0;  
    temp = callee(1, 2, 3, 4, 5);  
    return temp;  
}
```

# Stacks 101

# emBO++

High



Low

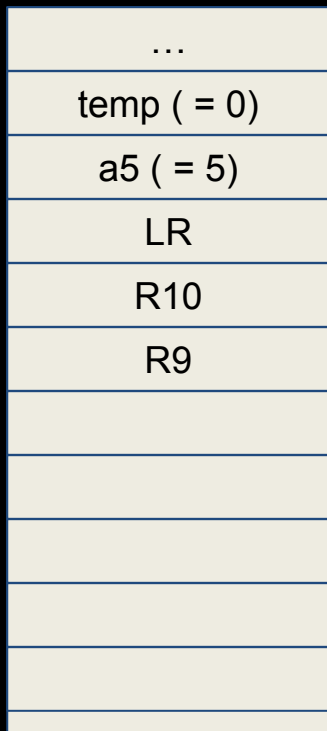
Stack

```
int callee(int a1, int a2, int a3, int a4, int a5) {  
    return a1 + a2 + a3 + a4 + a5;  
}  
  
int caller() {  
    int temp = 0;  
    temp = callee(1, 2, 3, 4, 5);  
    return temp;  
}
```

# Stacks 101

# emBO++

High



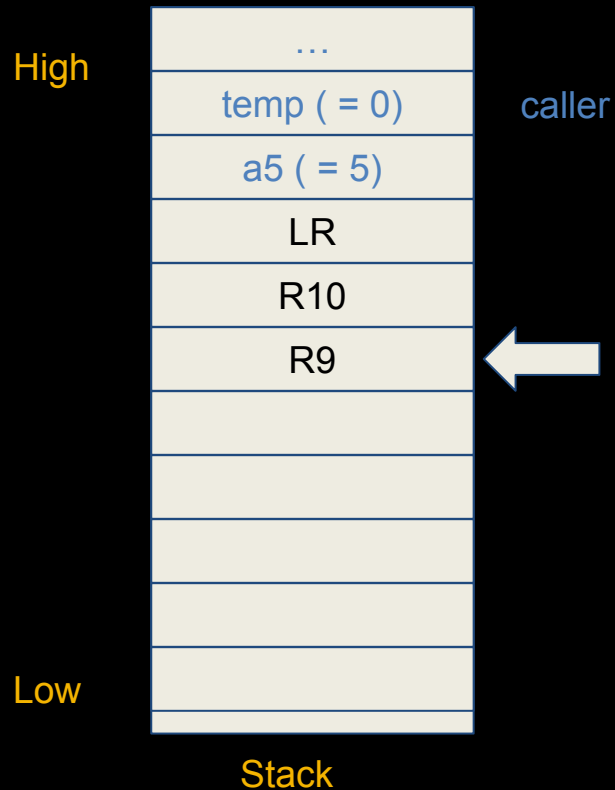
Low

Stack

```
int callee(int a1, int a2, int a3, int a4, int a5) {  
    return a1 + a2 + a3 + a4 + a5;  
}  
  
int caller() {  
    int temp = 0;  
    temp = callee(1, 2, 3, 4, 5);  
    return temp;  
}
```

# Stacks 101

# emBO++

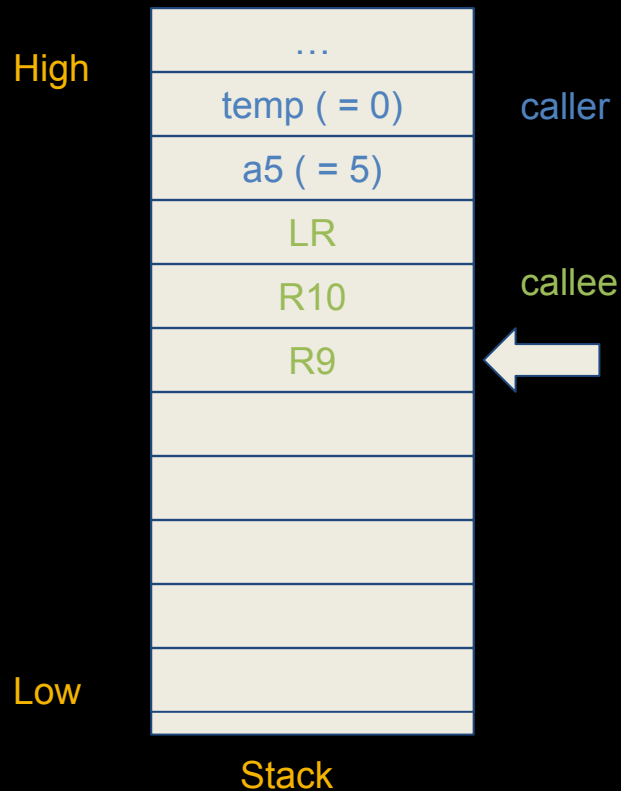


```
int callee(int a1, int a2, int a3, int a4, int a5) {  
    return a1 + a2 + a3 + a4 + a5;  
}  
  
int caller() {  
    int temp = 0;  
    temp = callee(1, 2, 3, 4, 5);  
    return temp;  
}
```

The code shows two functions. The 'callee' function takes five integer arguments and returns their sum. The 'caller' function declares a local 'temp' variable, calls 'callee' with arguments 1 through 5, and returns the result. A white arrow points to the opening curly brace of the 'callee' function.

# Stacks 101

# emBO++



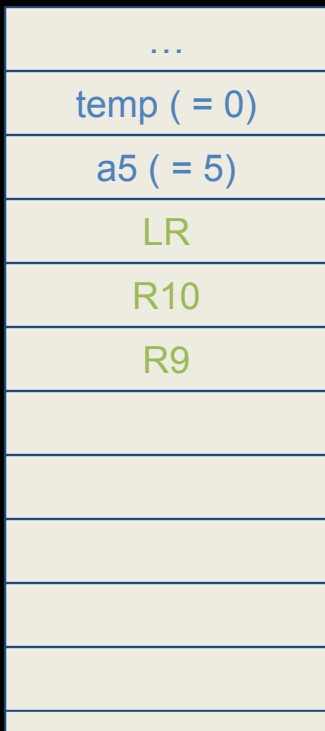
```
int callee(int a1, int a2, int a3, int a4, int a5) {  
    return a1 + a2 + a3 + a4 + a5;  
}  
  
int caller() {  
    int temp = 0;  
    temp = callee(1, 2, 3, 4, 5);  
    return temp;  
}
```

A white arrow points to the opening brace of the 'callee' function definition, indicating the start of the function's execution context.

# Stacks 101



High



Low

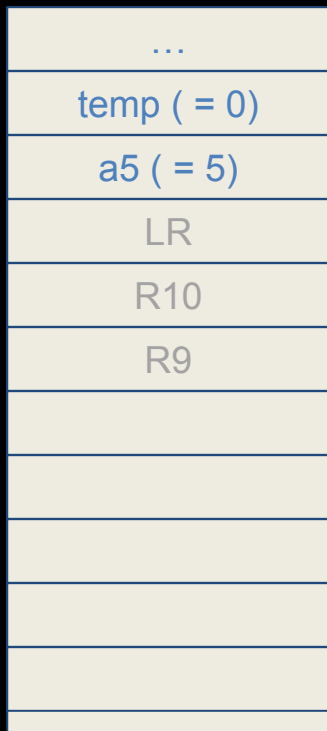
Stack

```
int callee(int a1, int a2, int a3, int a4, int a5) {  
    return a1 + a2 + a3 + a4 + a5;  
}  
  
int caller() {  
    int temp = 0;  
    temp = callee(1, 2, 3, 4, 5);  
    return temp;  
}
```

# Stacks 101

# emBO++

High



Low

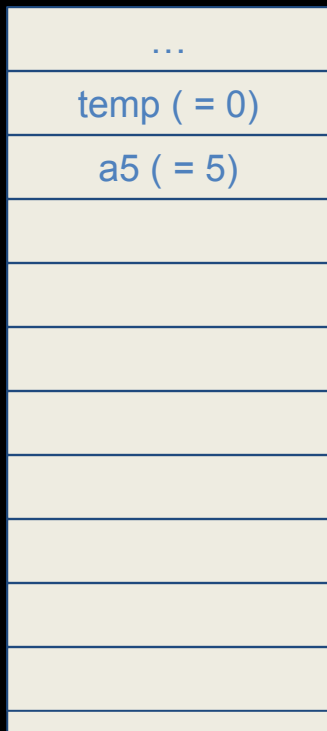
Stack

```
int callee(int a1, int a2, int a3, int a4, int a5) {  
    return a1 + a2 + a3 + a4 + a5;  
}  
  
int caller() {  
    int temp = 0;  
    temp = callee(1, 2, 3, 4, 5);  
    return temp;  
}
```

# Stacks 101



High



Low

Stack

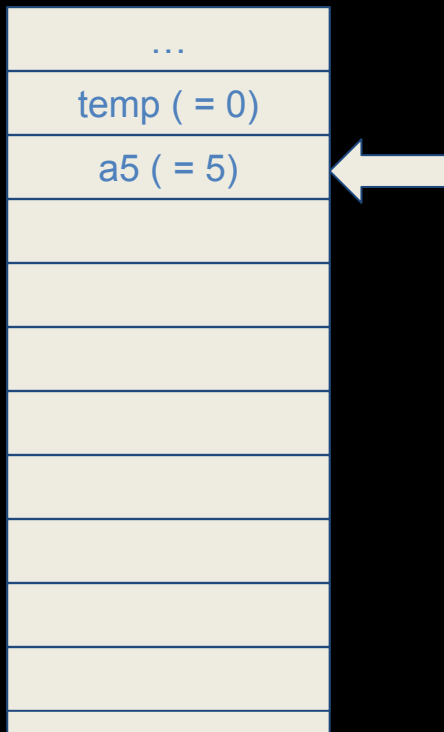
```
int callee(int a1, int a2, int a3, int a4, int a5) {  
    return a1 + a2 + a3 + a4 + a5;  
}  
  
int caller() {  
    int temp = 0;  
    temp = callee(1, 2, 3, 4, 5);  
    return temp;  
}
```



# Stacks 101

# emBO++

High



Low

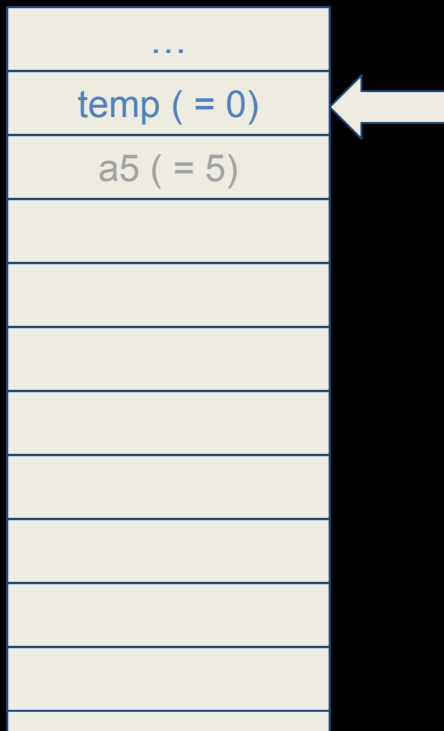
Stack

```
int callee(int a1, int a2, int a3, int a4, int a5) {  
    return a1 + a2 + a3 + a4 + a5;  
}  
  
int caller() {  
    int temp = 0;  
    temp = callee(1, 2, 3, 4, 5);  
    return temp;  
}
```

# Stacks 101



High



Low

Stack

```
int callee(int a1, int a2, int a3, int a4, int a5) {  
    return a1 + a2 + a3 + a4 + a5;  
}  
  
int caller() {  
    int temp = 0;  
    temp = callee(1, 2, 3, 4, 5);  
    return temp;  
}
```

A white arrow points to the line "temp = callee(1, 2, 3, 4, 5);" in the caller function, indicating the point of execution where the callee function is called.

# Stacks 101

# emBO++


High



Low

Stack

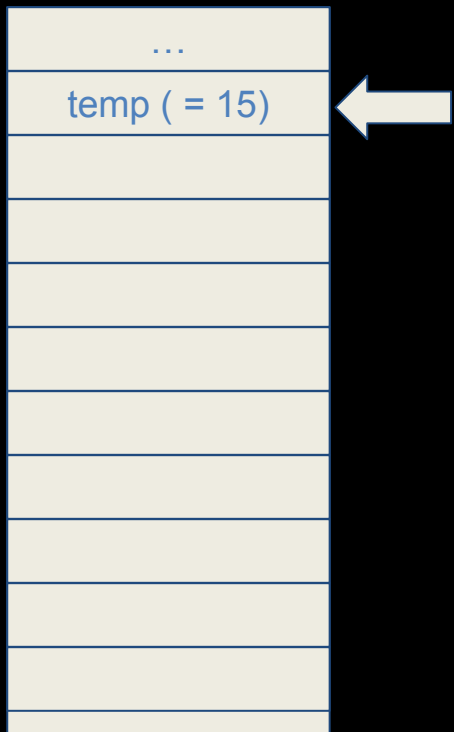
```
int callee(int a1, int a2, int a3, int a4, int a5) {  
    return a1 + a2 + a3 + a4 + a5;  
}  
  
int caller() {  
    int temp = 0;  
    temp = callee(1, 2, 3, 4, 5);  
    return temp;  
}
```



# Stacks 101

# emBO++

High



Low

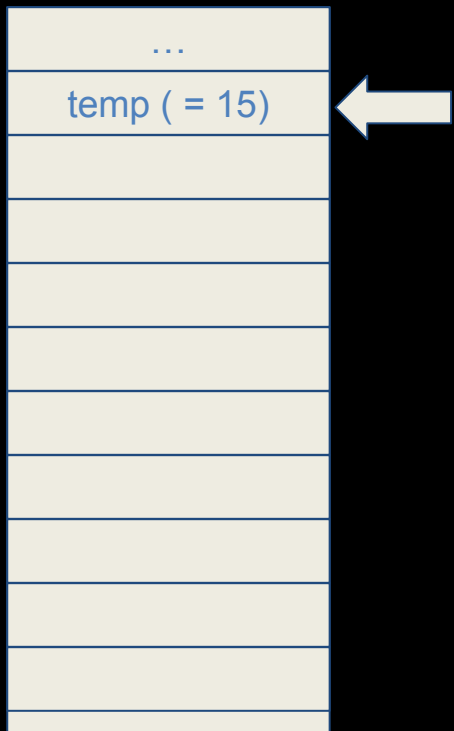
Stack

```
int callee(int a1, int a2, int a3, int a4, int a5) {  
    return a1 + a2 + a3 + a4 + a5;  
}  
  
int caller() {  
    int temp = 0;  
    temp = callee(1, 2, 3, 4, 5);  
    return temp;  
}
```

# Stacks 101

# emBO++


High



Low

Stack

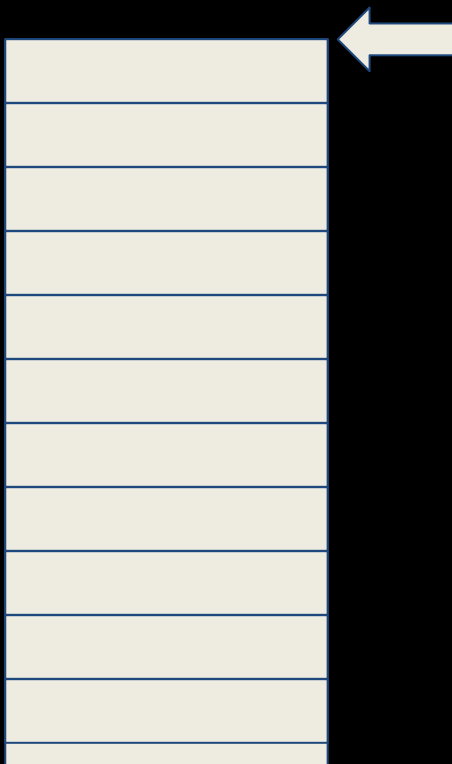
```
int callee(int a1, int a2, int a3, int a4, int a5) {  
    return a1 + a2 + a3 + a4 + a5;  
}  
  
int caller() {  
    int temp = 0;  
    temp = callee(1, 2, 3, 4, 5);  
    return temp;  
}
```



# Stacks 101

# emBO++

High



Low

Stack

```
int callee(int a1, int a2, int a3, int a4, int a5) {  
    return a1 + a2 + a3 + a4 + a5;  
}  
  
int caller() {  
    int temp = 0;  
    temp = callee(1, 2, 3, 4, 5);  
    return temp;  
}
```

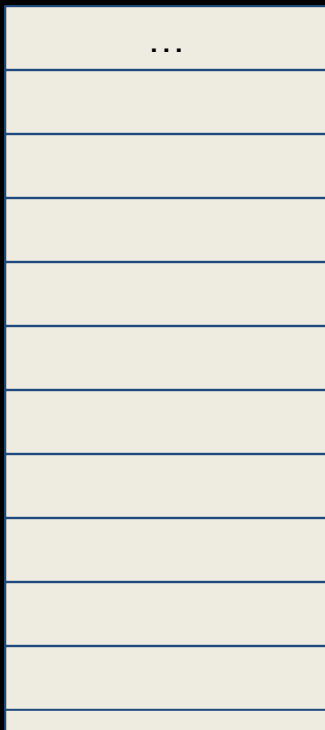
A white arrow points to the caller function, indicating the current function being executed.

emBO++



# Stack overflow

High



Low

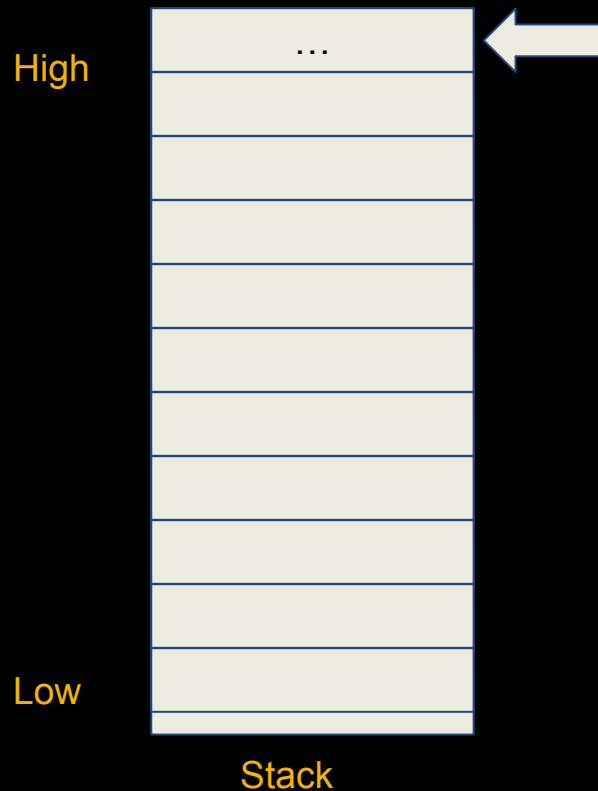
Stack

# emBO++

```
void callee(char* buf, size_t len) {  
    char my_buf[16] = {0};  
    memcpy(my_buf, buf, len);  
}  
  
void caller(char* aaa_buf) {  
    callee(aaa_buf, 8);  
    callee(aaa_buf, 16);  
    callee(aaa_buf, 20);  
    callee(aaa_buf, 24);  
    callee(aaa_buf, 28);  
}
```




# Stack overflow



# emBO++

```
void callee(char* buf, size_t len) {  
    char my_buf[16] = {0};  
    memcpy(my_buf, buf, len);  
}  
  
void caller(char* aaa_buf) {  
    callee(aaa_buf, 8);  
    callee(aaa_buf, 16);  
    callee(aaa_buf, 20);  
    callee(aaa_buf, 24);  
    callee(aaa_buf, 28);  
}
```



# Stack overflow

High



Low

Stack

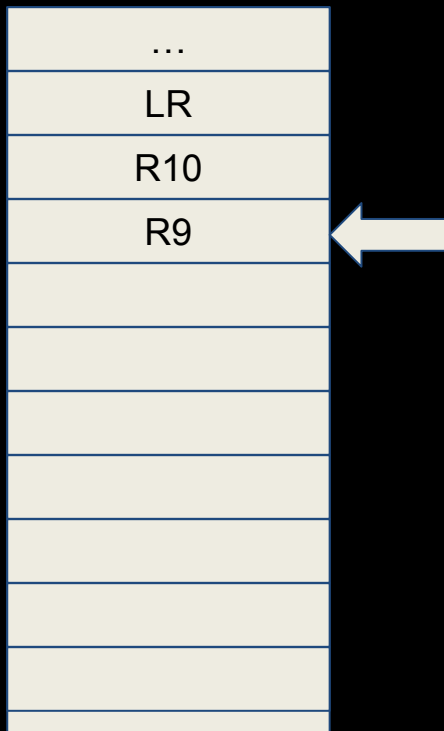
# emBO++

```
void callee(char* buf, size_t len) {  
    char my_buf[16] = {0};  
    memcpy(my_buf, buf, len);  
}
```

```
void caller(char* aaa_buf) {  
    callee(aaa_buf, 8);  
    callee(aaa_buf, 16);  
    callee(aaa_buf, 20);  
    callee(aaa_buf, 24);  
    callee(aaa_buf, 28);  
}
```

# Stack overflow

High



Low

Stack

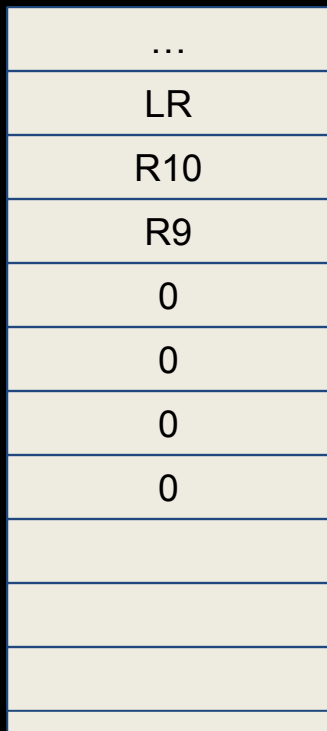
# emBO++

```
void callee(char* buf, size_t len) {  
    char my_buf[16] = {0};  
    memcpy(my_buf, buf, len);  
}
```

```
void caller(char* aaa_buf) {  
    callee(aaa_buf, 8);  
    callee(aaa_buf, 16);  
    callee(aaa_buf, 20);  
    callee(aaa_buf, 24);  
    callee(aaa_buf, 28);  
}
```

# Stack overflow

High



Low

Stack



# emBO++

```
void callee(char* buf, size_t len) {  
    char my_buf[16] = {0};  
    memcpy(my_buf, buf, len);  
}
```

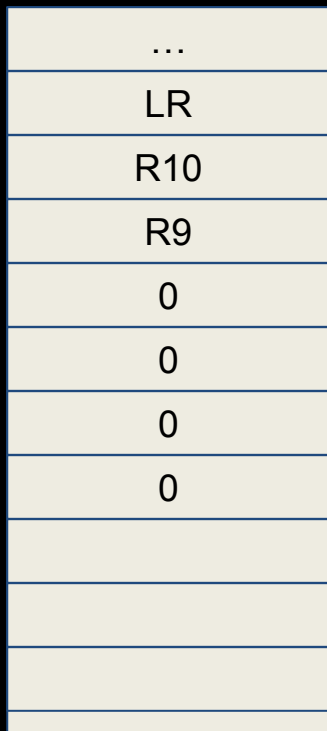


```
void caller(char* aaa_buf) {  
    callee(aaa_buf, 8);  
    callee(aaa_buf, 16);  
    callee(aaa_buf, 20);  
    callee(aaa_buf, 24);  
    callee(aaa_buf, 28);  
}
```



# Stack overflow

High



Low

Stack

# emBO++

```
void callee(char* buf, size_t len) {  
    char my_buf[16] = {0};  
    memcpy(my_buf, buf, len);  
}
```

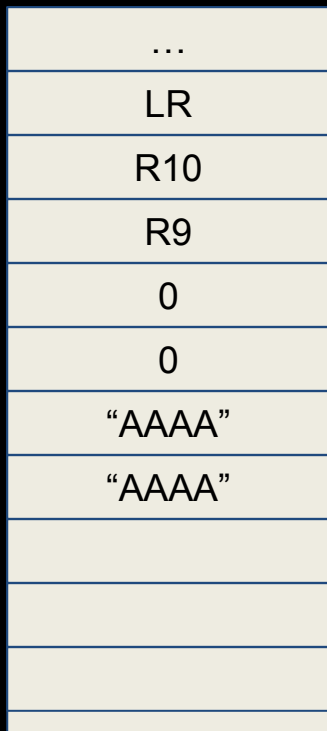


```
void caller(char* aaa_buf) {  
    callee(aaa_buf, 8);  
    callee(aaa_buf, 16);  
    callee(aaa_buf, 20);  
    callee(aaa_buf, 24);  
    callee(aaa_buf, 28);  
}
```



# Stack overflow

High



Low

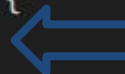
Stack

# emBO++

```
void callee(char* buf, size_t len) {  
    char my_buf[16] = {0};  
    memcpy(my_buf, buf, len);  
}
```

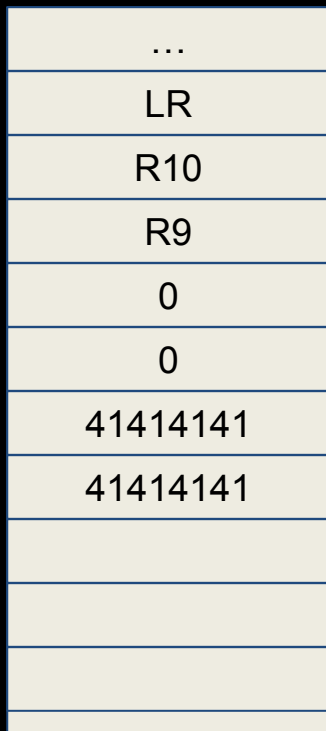


```
void caller(char* aaa_buf) {  
    callee(aaa_buf, 8);  
    callee(aaa_buf, 16);  
    callee(aaa_buf, 20);  
    callee(aaa_buf, 24);  
    callee(aaa_buf, 28);  
}
```



# Stack overflow

High



Low

Stack

# emBO++

```
void callee(char* buf, size_t len) {  
    char my_buf[16] = {0};  
    memcpy(my_buf, buf, len);  
}
```

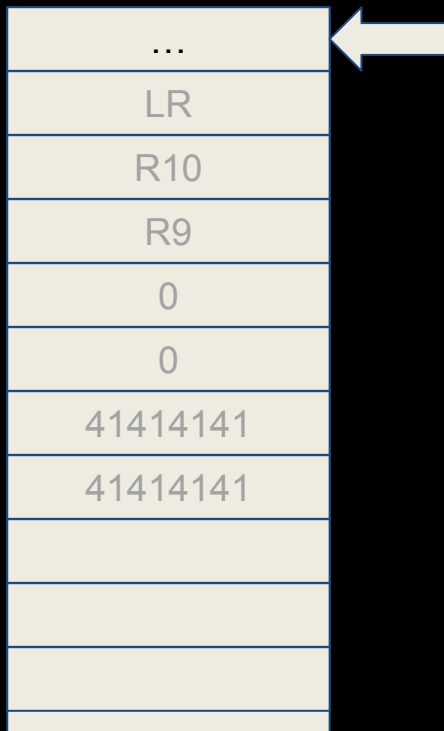


```
void caller(char* aaa_buf) {  
    callee(aaa_buf, 8);  
    callee(aaa_buf, 16);  
    callee(aaa_buf, 20);  
    callee(aaa_buf, 24);  
    callee(aaa_buf, 28);  
}
```



# Stack overflow

High



Low

Stack

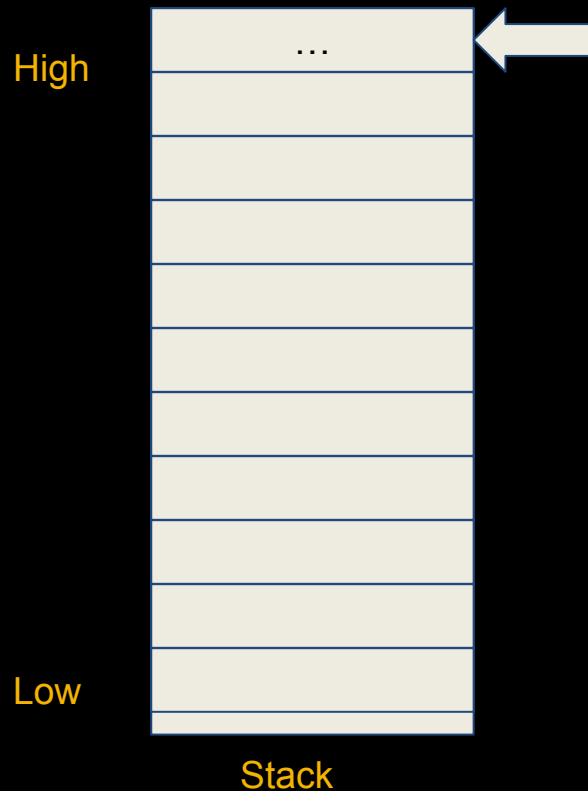
# emBO++

```
void callee(char* buf, size_t len) {  
    char my_buf[16] = {0};  
    memcpy(my_buf, buf, len);  
}
```

```
void caller(char* aaa_buf) {  
    callee(aaa_buf, 8);  
    callee(aaa_buf, 16);  
    callee(aaa_buf, 20);  
    callee(aaa_buf, 24);  
    callee(aaa_buf, 28);  
}
```



# Stack overflow

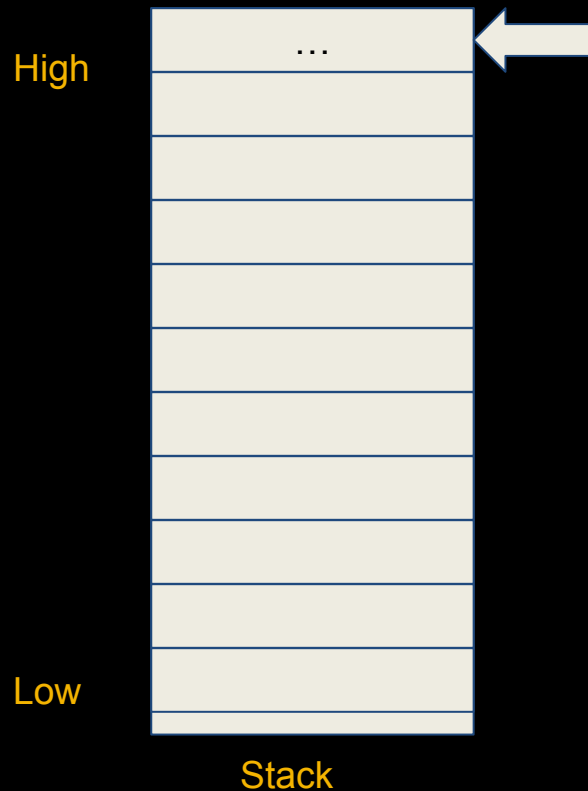


# emBO++

```
void callee(char* buf, size_t len) {  
    char my_buf[16] = {0};  
    memcpy(my_buf, buf, len);  
}  
  
void caller(char* aaa_buf) {  
    callee(aaa_buf, 8);  
    callee(aaa_buf, 16);  
    callee(aaa_buf, 20);  
    callee(aaa_buf, 24);  
    callee(aaa_buf, 28);  
}
```


A white arrow points to the `memcpy` line in the `callee` function. A blue arrow points to the `callee(aaa_buf, 28);` line in the `caller` function, indicating a recursive call that leads to a stack overflow.

# Stack overflow



# emBO++

```
void callee(char* buf, size_t len) {  
    char my_buf[16] = {0};  
    memcpy(my_buf, buf, len);  
}  
  
void caller(char* aaa_buf) {  
    callee(aaa_buf, 8);  
    callee(aaa_buf, 16);  
    callee(aaa_buf, 20);  
    callee(aaa_buf, 24);  
    callee(aaa_buf, 28);  
}
```



# Stack overflow

High



Low

Stack

# emBO++

```
void callee(char* buf, size_t len) {  
    char my_buf[16] = {0};  
    memcpy(my_buf, buf, len);  
}
```

```
void caller(char* aaa_buf) {  
    callee(aaa_buf, 8);  
    callee(aaa_buf, 16);  
    callee(aaa_buf, 20);  
    callee(aaa_buf, 24);  
    callee(aaa_buf, 28);  
}
```

# Stack overflow

High

...
LR
R10
R9
41414141
41414141
41414141
41414141

Low

Stack



# emBO++

```
void callee(char* buf, size_t len) {  
    char my_buf[16] = {0};  
    memcpy(my_buf, buf, len);  
}
```

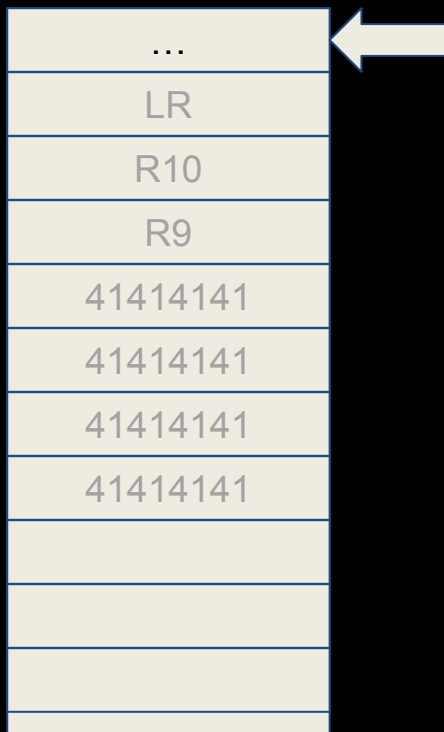


```
void caller(char* aaa_buf) {  
    callee(aaa_buf, 8);  
    callee(aaa_buf, 16);  
    callee(aaa_buf, 20);  
    callee(aaa_buf, 24);  
    callee(aaa_buf, 28);  
}
```



# Stack overflow

High



Low

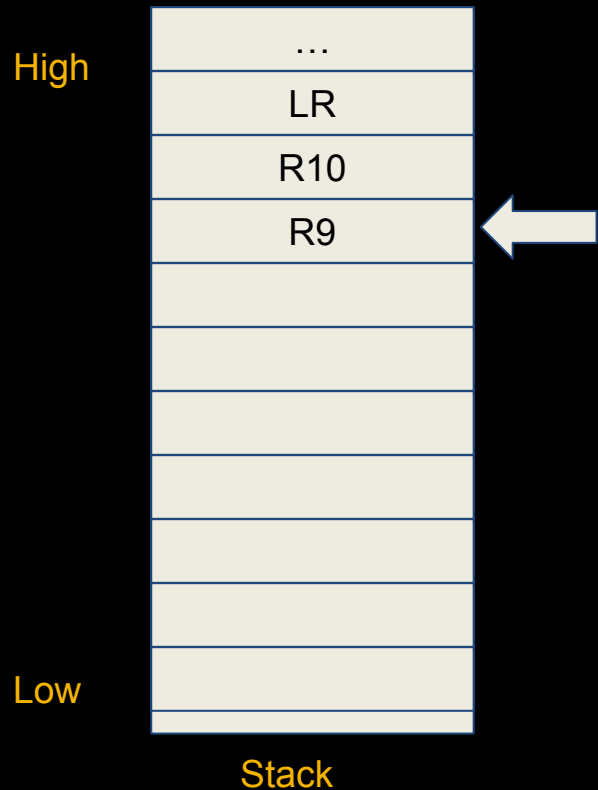
Stack

# emBO++

```
void callee(char* buf, size_t len) {  
    char my_buf[16] = {0};  
    memcpy(my_buf, buf, len);  
}
```

```
void caller(char* aaa_buf) {  
    callee(aaa_buf, 8);  
    callee(aaa_buf, 16);  
    callee(aaa_buf, 20);  
    callee(aaa_buf, 24);  
    callee(aaa_buf, 28);  
}
```

# Stack overflow



# emBO++

```
void callee(char* buf, size_t len) {  
    char my_buf[16] = {0};  
    memcpy(my_buf, buf, len);  
}  
  
void caller(char* aaa_buf) {  
    callee(aaa_buf, 8);  
    callee(aaa_buf, 16);  
    callee(aaa_buf, 20);  
    callee(aaa_buf, 24);  
    callee(aaa_buf, 28);  
}
```

# Stack overflow

High

...
LR
R10
41414141
41414141
41414141
41414141
41414141

Low

Stack



# emBO++

```
void callee(char* buf, size_t len) {  
    char my_buf[16] = {0};  
    memcpy(my_buf, buf, len);  
}
```

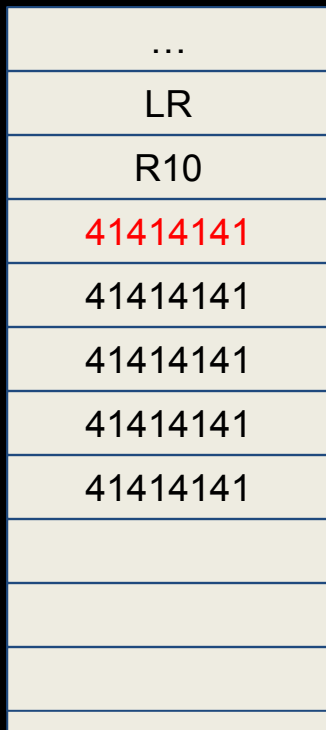


```
void caller(char* aaa_buf) {  
    callee(aaa_buf, 8);  
    callee(aaa_buf, 16);  
    callee(aaa_buf, 20);  
    callee(aaa_buf, 24);  
    callee(aaa_buf, 28);  
}
```



# Stack overflow

High



Low

Stack



# emBO++

```
void callee(char* buf, size_t len) {  
    char my_buf[16] = {0};  
    memcpy(my_buf, buf, len);  
}
```



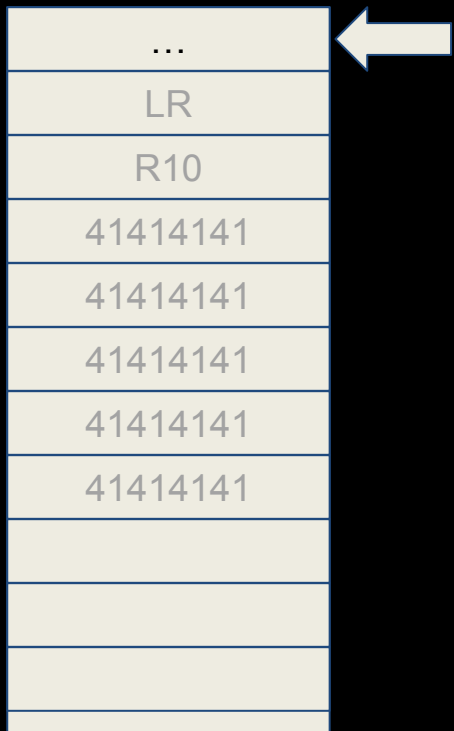
```
void caller(char* aaa_buf) {  
    callee(aaa_buf, 8);  
    callee(aaa_buf, 16);  
    callee(aaa_buf, 20);  
    callee(aaa_buf, 24);  
    callee(aaa_buf, 28);  
}
```





# Stack overflow

High



Low

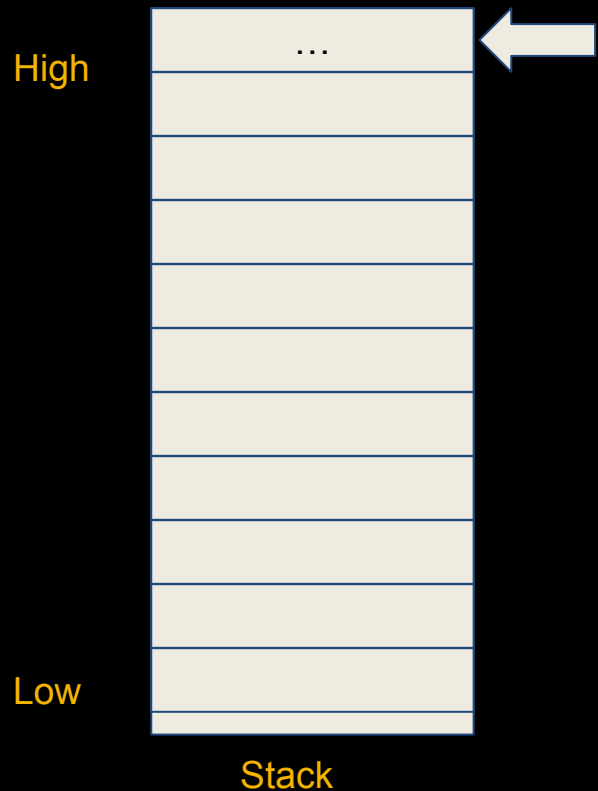
Stack

# emBO++

```
void callee(char* buf, size_t len) {  
    char my_buf[16] = {0};  
    memcpy(my_buf, buf, len);  
}
```


```
void caller(char* aaa_buf) {  
    callee(aaa_buf, 8);  
    callee(aaa_buf, 16);  
    callee(aaa_buf, 20);  
    callee(aaa_buf, 24);  
    callee(aaa_buf, 28);  
}
```

# Stack overflow



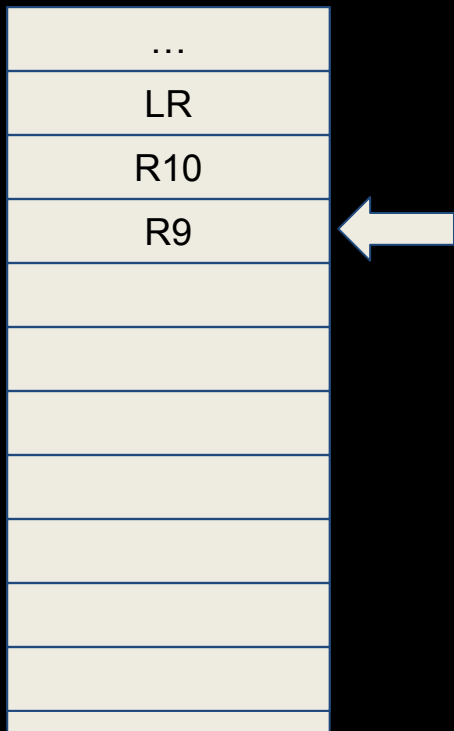
# emBO++

```
void callee(char* buf, size_t len) {  
    char my_buf[16] = {0};  
    memcpy(my_buf, buf, len);  
}  
  
void caller(char* aaa_buf) {  
    callee(aaa_buf, 8);  
    callee(aaa_buf, 16);  
    callee(aaa_buf, 20);  
    callee(aaa_buf, 24);  
    callee(aaa_buf, 28);  
}
```



# Stack overflow

High



Low

Stack

# emBO++

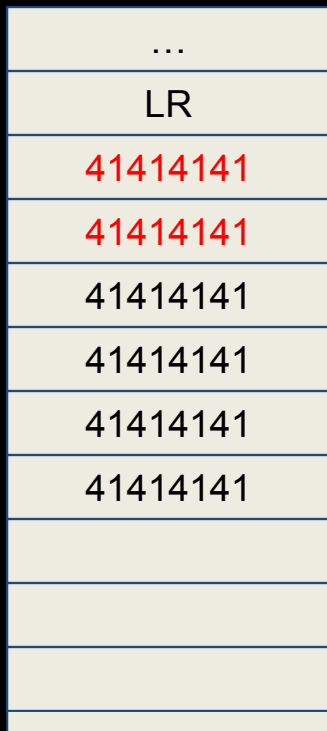
```
void callee(char* buf, size_t len) {  
    char my_buf[16] = {0};  
    memcpy(my_buf, buf, len);  
}
```

```
void caller(char* aaa_buf) {  
    callee(aaa_buf, 8);  
    callee(aaa_buf, 16);  
    callee(aaa_buf, 20);  
    callee(aaa_buf, 24);  
    callee(aaa_buf, 28);  
}
```



# Stack overflow

High



Low

Stack



# emBO++

```
void callee(char* buf, size_t len) {  
    char my_buf[16] = {0};  
    memcpy(my_buf, buf, len);  
}
```

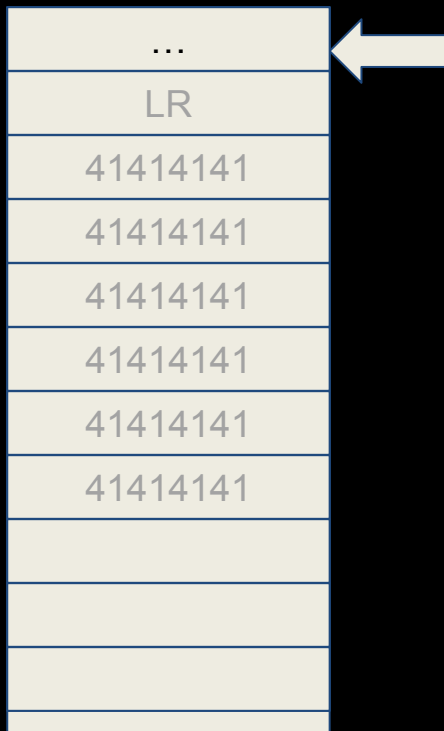


```
void caller(char* aaa_buf) {  
    callee(aaa_buf, 8);  
    callee(aaa_buf, 16);  
    callee(aaa_buf, 20);  
    callee(aaa_buf, 24);  
    callee(aaa_buf, 28);  
}
```



# Stack overflow

High



Low

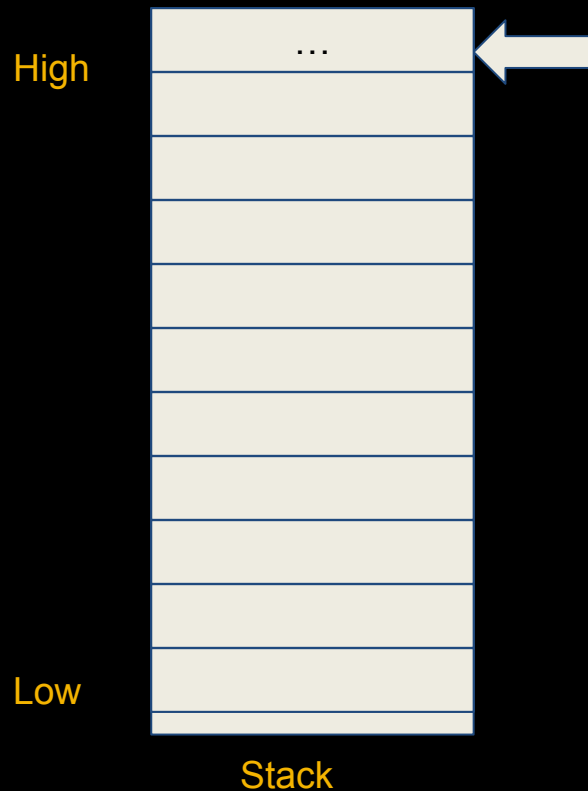
Stack

# emBO++

```
void callee(char* buf, size_t len) {  
    char my_buf[16] = {0};  
    memcpy(my_buf, buf, len);  
}  
  
void caller(char* aaa_buf) {  
    callee(aaa_buf, 8);  
    callee(aaa_buf, 16);  
    callee(aaa_buf, 20);  
    callee(aaa_buf, 24);  
    callee(aaa_buf, 28);  
}
```


A white arrow points to the 'memcpy' line in the 'callee' function. A blue arrow points to the 'callee(aaa\_buf, 28);' line in the 'caller' function, indicating that the caller is passing a buffer size that exceeds the callee's local buffer capacity, leading to a stack overflow.

# Stack overflow



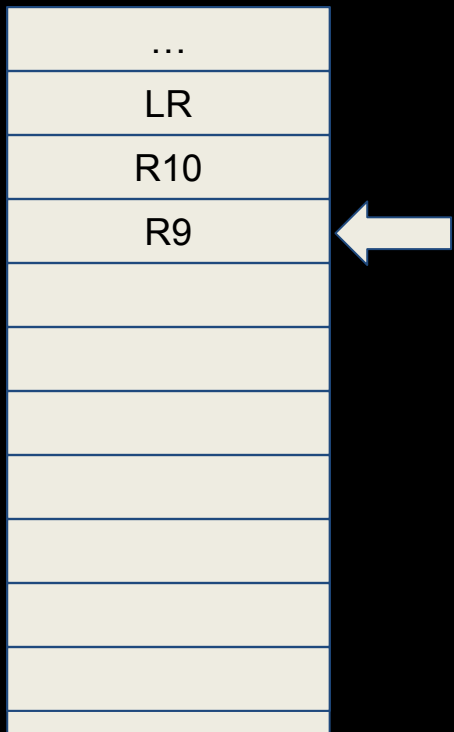
# emBO++

```
void callee(char* buf, size_t len) {  
    char my_buf[16] = {0};  
    memcpy(my_buf, buf, len);  
}  
  
void caller(char* aaa_buf) {  
    callee(aaa_buf, 8);  
    callee(aaa_buf, 16);  
    callee(aaa_buf, 20);  
    callee(aaa_buf, 24);  
    callee(aaa_buf, 28);  
}
```



# Stack overflow

High



Low

Stack

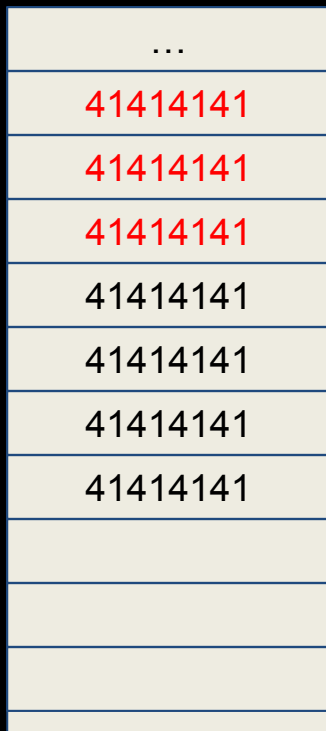
# emBO++

```
void callee(char* buf, size_t len) {  
    char my_buf[16] = {0};  
    memcpy(my_buf, buf, len);  
}
```

```
void caller(char* aaa_buf) {  
    callee(aaa_buf, 8);  
    callee(aaa_buf, 16);  
    callee(aaa_buf, 20);  
    callee(aaa_buf, 24);  
    callee(aaa_buf, 28);  
}
```

# Stack overflow

High



Low

Stack

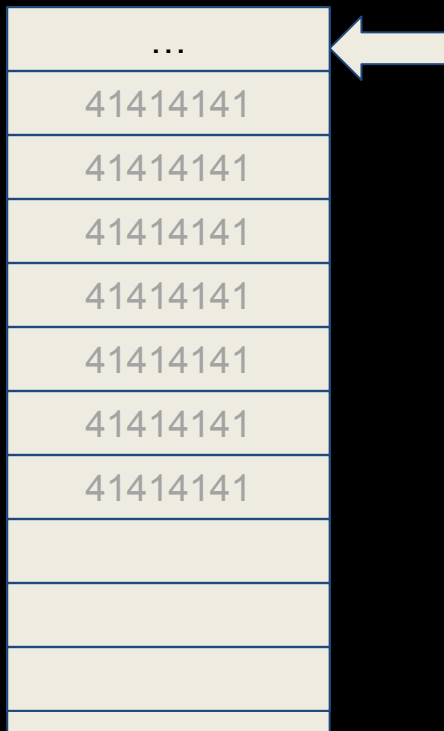
# emBO++

```
void callee(char* buf, size_t len) {  
    char my_buf[16] = {0};  
    memcpy(my_buf, buf, len);  
}  
  
void caller(char* aaa_buf) {  
    callee(aaa_buf, 8);  
    callee(aaa_buf, 16);  
    callee(aaa_buf, 20);  
    callee(aaa_buf, 24);  
    callee(aaa_buf, 28);  
}
```



# Stack overflow

High



Low

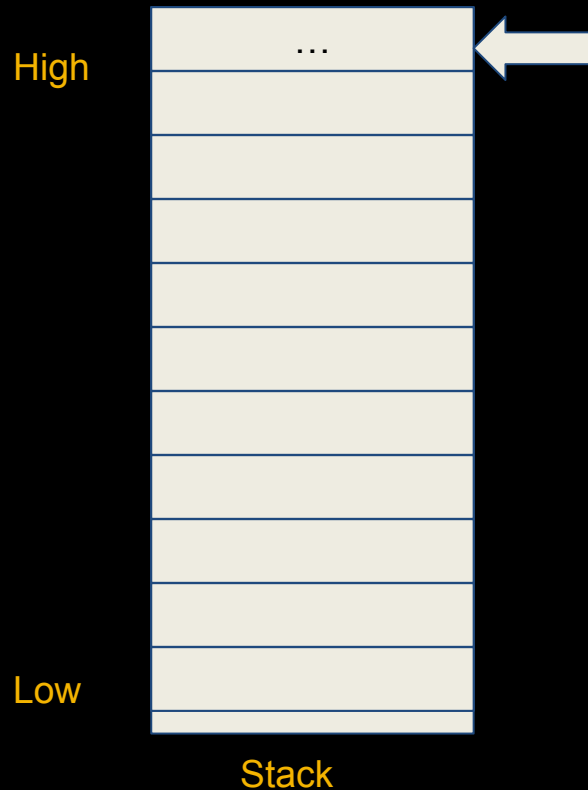
Stack

# emBO++

```
void callee(char* buf, size_t len) {  
    char my_buf[16] = {0};  
    memcpy(my_buf, buf, len);  
}
```


```
void caller(char* aaa_buf) {  
    callee(aaa_buf, 8);  
    callee(aaa_buf, 16);  
    callee(aaa_buf, 20);  
    callee(aaa_buf, 24);  
    callee(aaa_buf, 28);  
}
```

# Stack overflow

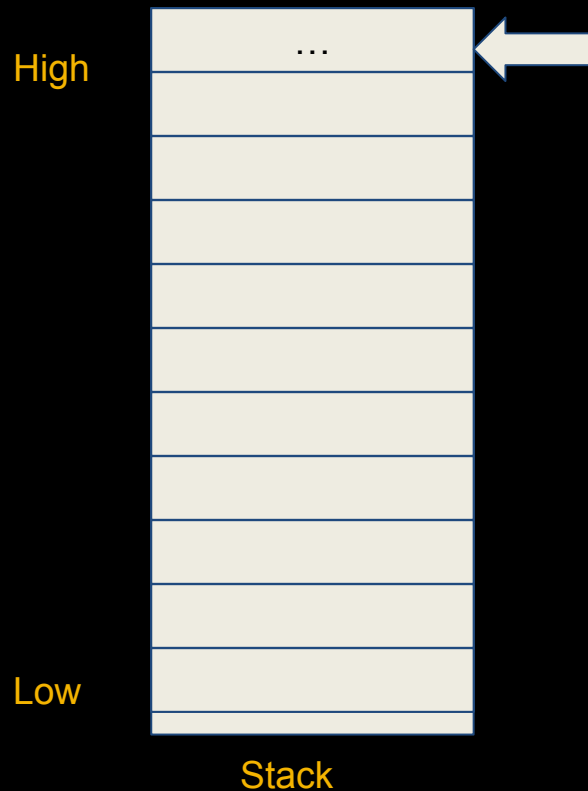


# emBO++

```
void callee(char* buf, size_t len) {  
    char my_buf[16] = {0};  
    memcpy(my_buf, buf, len);  
}  
  
void caller(char* aaa_buf) {  
    callee(aaa_buf, 8);  
    callee(aaa_buf, 16);  
    callee(aaa_buf, 20);  
    callee(aaa_buf, 24);  
    callee(aaa_buf, 28);  
}
```



# Stack overflow

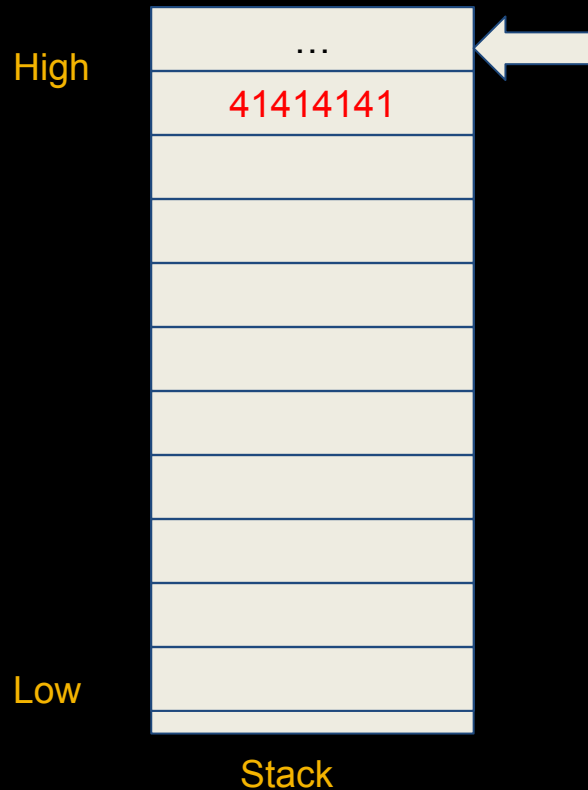


# emBO++

```
void callee(char* buf, size_t len) {  
    char my_buf[16] = {0};  
    memcpy(my_buf, buf, len);  
}  
  
void caller(char* aaa_buf) {  
    callee(aaa_buf, 8);  
    callee(aaa_buf, 16);  
    callee(aaa_buf, 20);  
    callee(aaa_buf, 24);  
    callee(aaa_buf, 28);  
}
```



# Stack overflow



# emBO++

```
void callee(char* buf, size_t len) {  
    char my_buf[16] = {0};  
    memcpy(my_buf, buf, len);  
}  
  
void caller(char* aaa_buf) {  
    callee(aaa_buf, 8);  
    callee(aaa_buf, 16);  
    callee(aaa_buf, 20);  
    callee(aaa_buf, 24);  
    callee(aaa_buf, 28);  
}
```



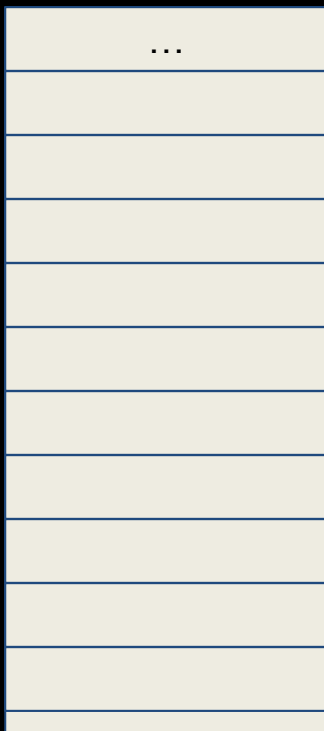
emBO++

Hmm....

# Controlled stack overflow



High

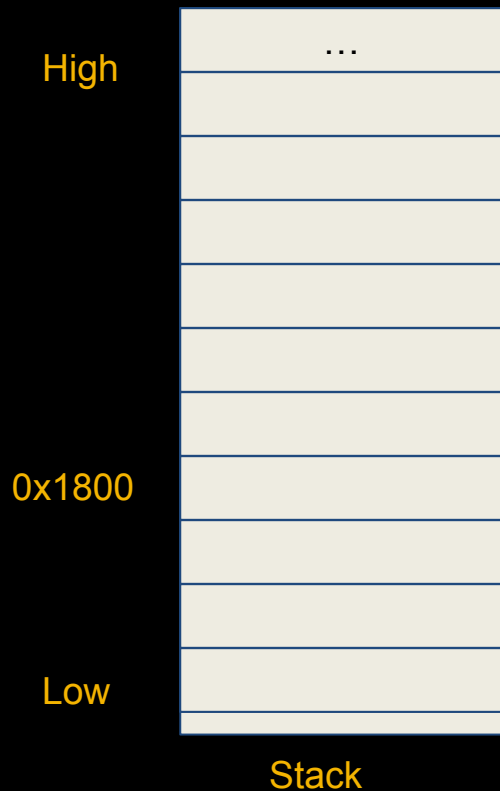


Low

Stack

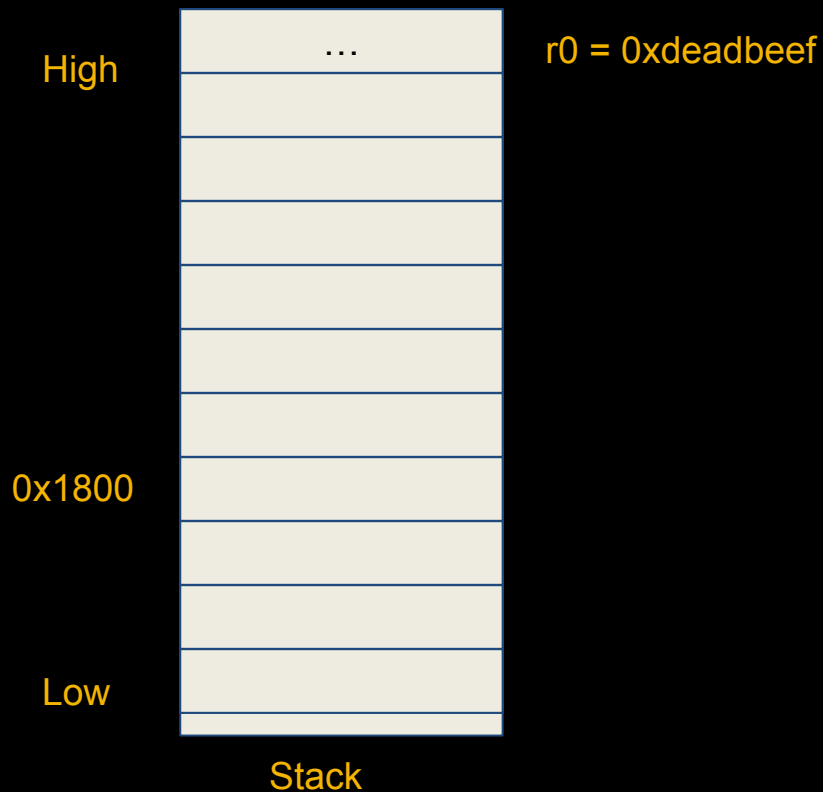
```
void callee(char* buf, size_t len) {  
    char my_buf[16] = {0};  
    memcpy(my_buf, buf, len);  
}  
  
void caller(char* aaa_buf) {  
    callee(aaa_buf, 8);  
    callee(aaa_buf, 16);  
    callee(aaa_buf, 20);  
    callee(aaa_buf, 24);  
    callee(aaa_buf, 28);  
}
```

# Controlled stack overflow



```
void callee(char* buf, size_t len) {  
    char my_buf[16] = {0};  
    memcpy(my_buf, buf, len);  
}  
  
void caller(char* aaa_buf) {  
    callee(aaa_buf, 8);  
    callee(aaa_buf, 16);  
    callee(aaa_buf, 20);  
    callee(aaa_buf, 24);  
    callee(aaa_buf, 28);  
}
```

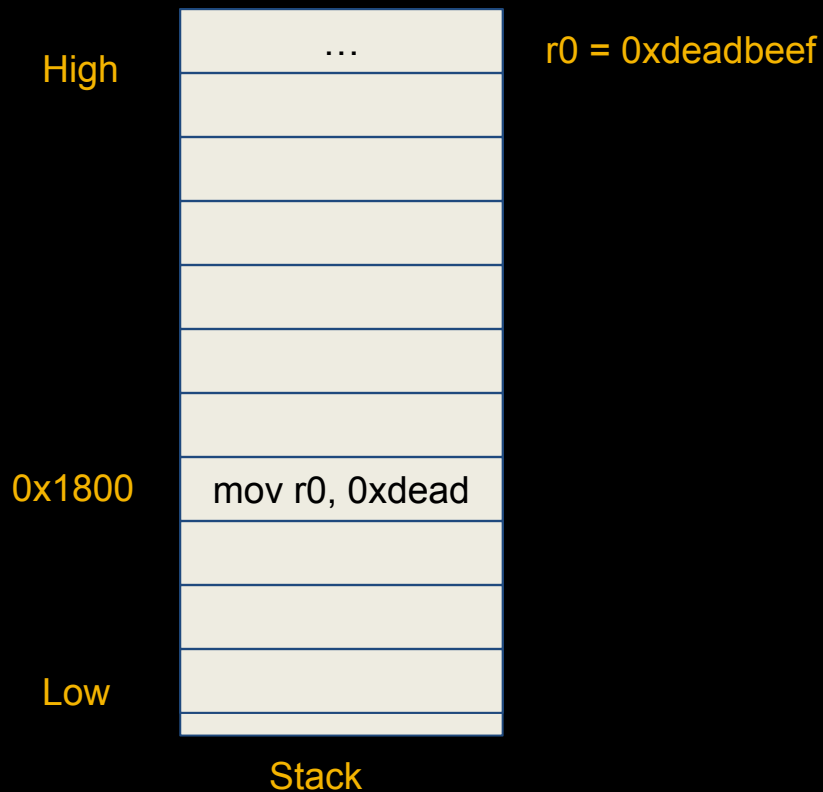
# Controlled stack overflow



```
void callee(char* buf, size_t len) {  
    char my_buf[16] = {0};  
    memcpy(my_buf, buf, len);  
}  
  
void caller(char* aaa_buf) {  
    callee(aaa_buf, 8);  
    callee(aaa_buf, 16);  
    callee(aaa_buf, 20);  
    callee(aaa_buf, 24);  
    callee(aaa_buf, 28);  
}
```

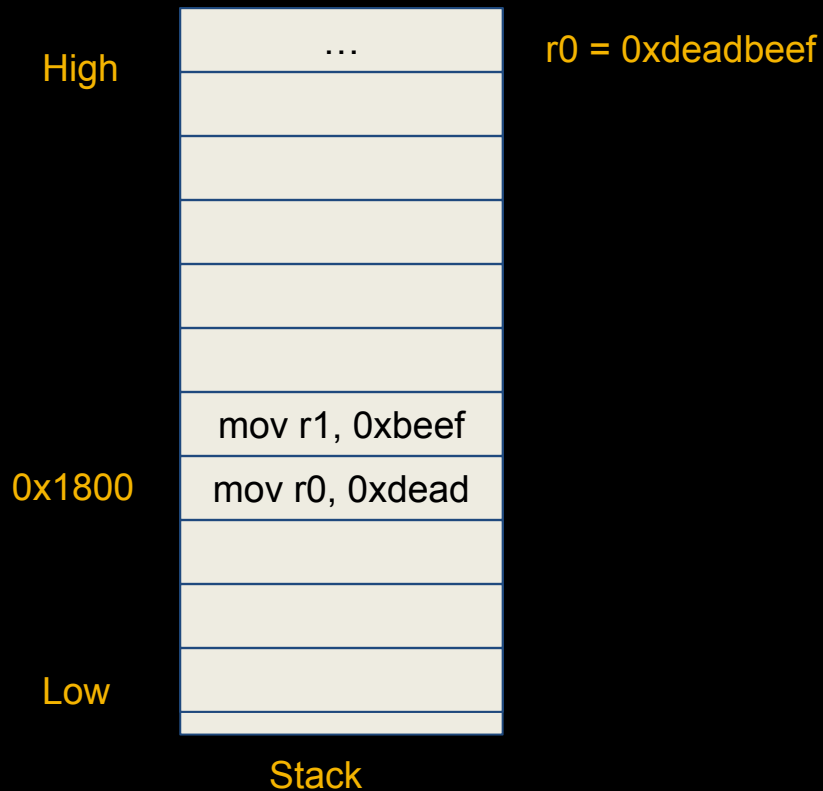


# Controlled stack overflow



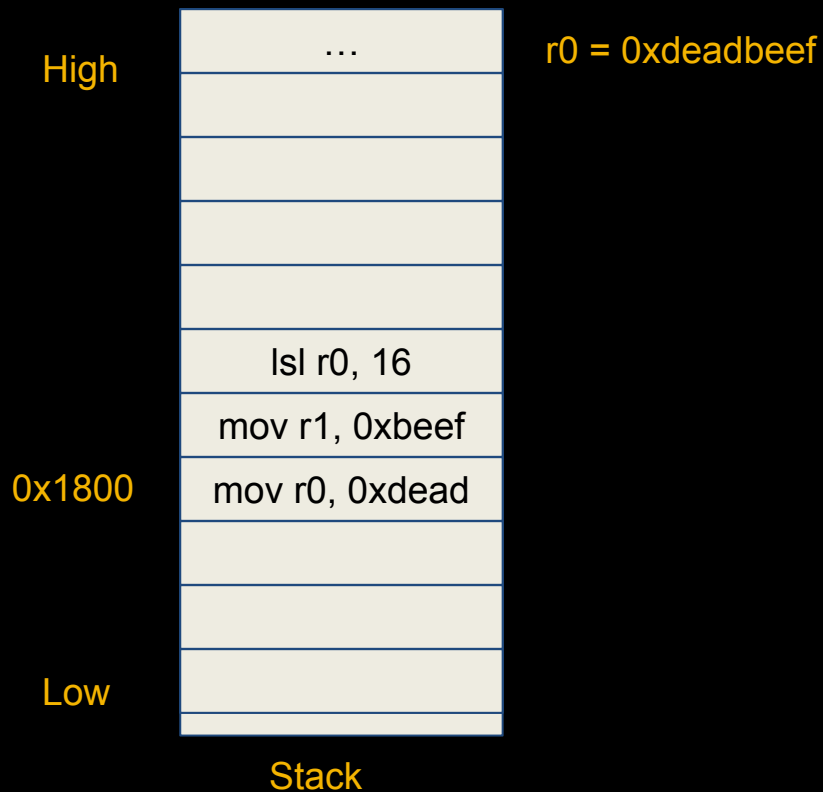
```
void callee(char* buf, size_t len) {  
    char my_buf[16] = {0};  
    memcpy(my_buf, buf, len);  
}  
  
void caller(char* aaa_buf) {  
    callee(aaa_buf, 8);  
    callee(aaa_buf, 16);  
    callee(aaa_buf, 20);  
    callee(aaa_buf, 24);  
    callee(aaa_buf, 28);  
}
```

# Controlled stack overflow



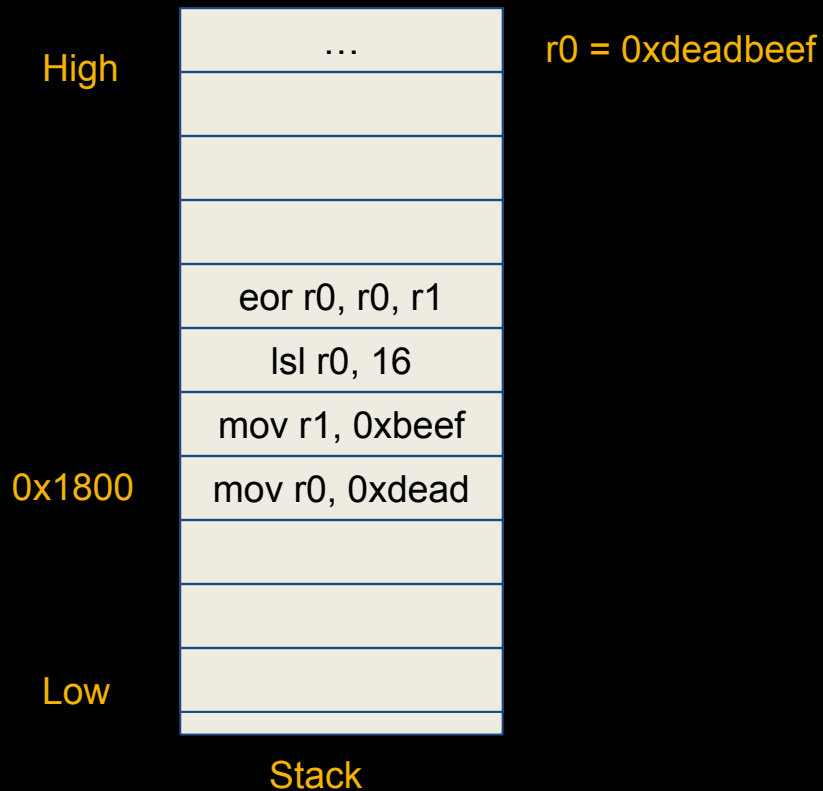
```
void callee(char* buf, size_t len) {  
    char my_buf[16] = {0};  
    memcpy(my_buf, buf, len);  
}  
  
void caller(char* aaa_buf) {  
    callee(aaa_buf, 8);  
    callee(aaa_buf, 16);  
    callee(aaa_buf, 20);  
    callee(aaa_buf, 24);  
    callee(aaa_buf, 28);  
}
```

# Controlled stack overflow



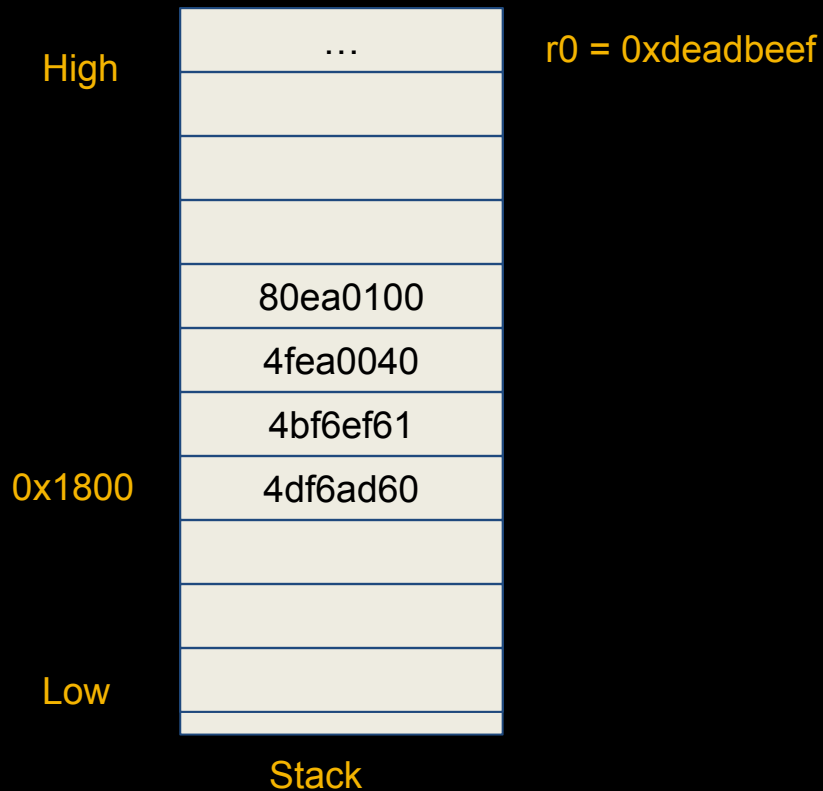
```
void callee(char* buf, size_t len) {  
    char my_buf[16] = {0};  
    memcpy(my_buf, buf, len);  
}  
  
void caller(char* aaa_buf) {  
    callee(aaa_buf, 8);  
    callee(aaa_buf, 16);  
    callee(aaa_buf, 20);  
    callee(aaa_buf, 24);  
    callee(aaa_buf, 28);  
}
```

# Controlled stack overflow



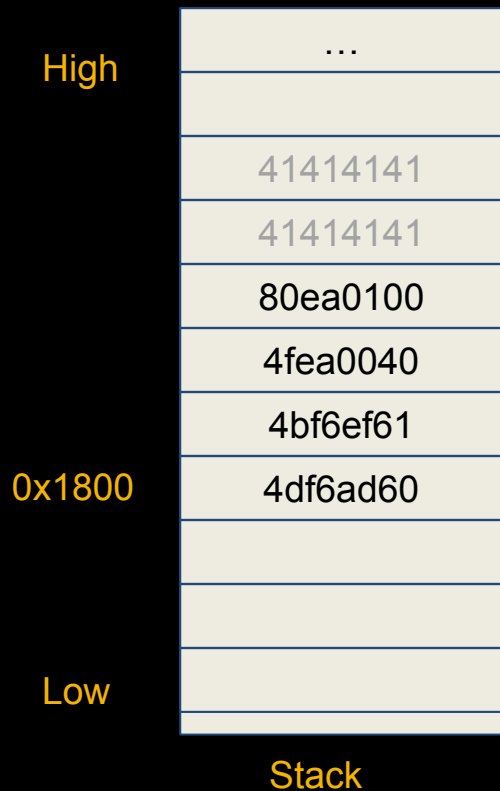
```
void callee(char* buf, size_t len) {  
    char my_buf[16] = {0};  
    memcpy(my_buf, buf, len);  
}  
  
void caller(char* aaa_buf) {  
    callee(aaa_buf, 8);  
    callee(aaa_buf, 16);  
    callee(aaa_buf, 20);  
    callee(aaa_buf, 24);  
    callee(aaa_buf, 28);  
}
```

# Controlled stack overflow



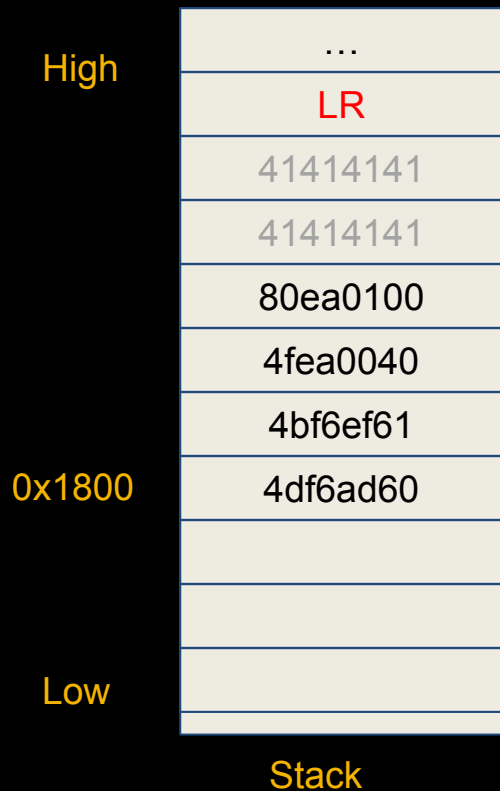
```
void callee(char* buf, size_t len) {  
    char my_buf[16] = {0};  
    memcpy(my_buf, buf, len);  
}  
  
void caller(char* aaa_buf) {  
    callee(aaa_buf, 8);  
    callee(aaa_buf, 16);  
    callee(aaa_buf, 20);  
    callee(aaa_buf, 24);  
    callee(aaa_buf, 28);  
}
```

# Controlled stack overflow



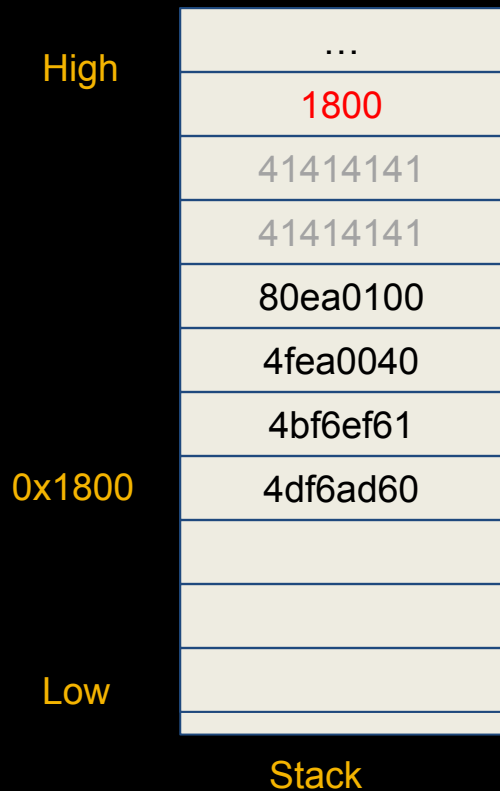
```
void callee(char* buf, size_t len) {  
    char my_buf[16] = {0};  
    memcpy(my_buf, buf, len);  
}  
  
void caller(char* aaa_buf) {  
    callee(aaa_buf, 8);  
    callee(aaa_buf, 16);  
    callee(aaa_buf, 20);  
    callee(aaa_buf, 24);  
    callee(aaa_buf, 28);  
}
```

# Controlled stack overflow



```
void callee(char* buf, size_t len) {  
    char my_buf[16] = {0};  
    memcpy(my_buf, buf, len);  
}  
  
void caller(char* aaa_buf) {  
    callee(aaa_buf, 8);  
    callee(aaa_buf, 16);  
    callee(aaa_buf, 20);  
    callee(aaa_buf, 24);  
    callee(aaa_buf, 28);  
}
```

# Controlled stack overflow

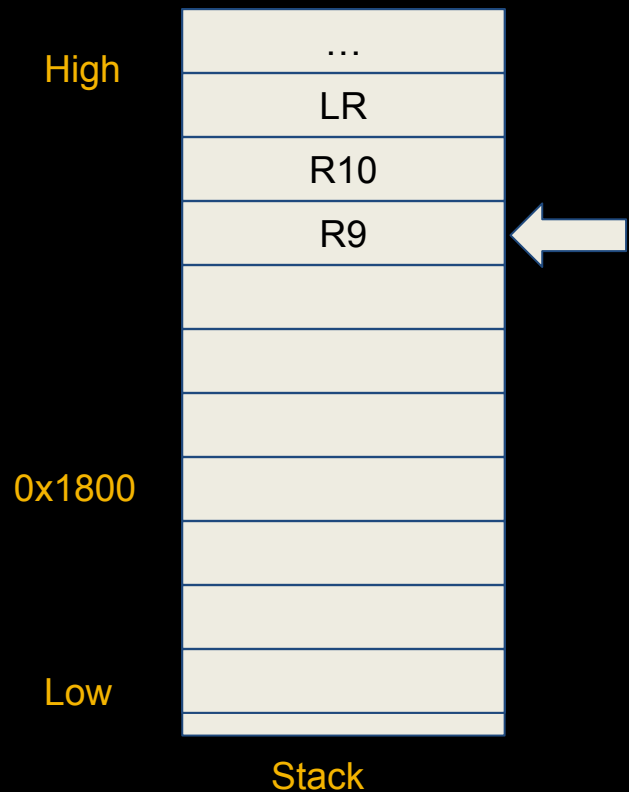


```
void callee(char* buf, size_t len) {  
    char my_buf[16] = {0};  
    memcpy(my_buf, buf, len);  
}  
  
void caller(char* aaa_buf) {  
    callee(aaa_buf, 8);  
    callee(aaa_buf, 16);  
    callee(aaa_buf, 20);  
    callee(aaa_buf, 24);  
    callee(aaa_buf, 28);  
}
```



# Controlled stack overflow

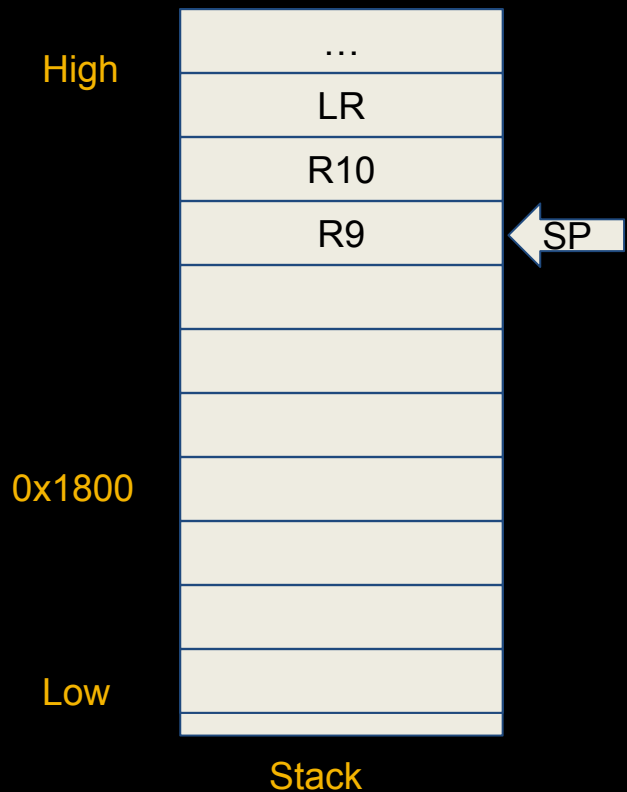
# emBO++



```
void callee(char* buf, size_t len) {  
    char my_buf[16] = {0};  
    memcpy(my_buf, buf, len);  
}  
  
void caller(char* aaa_buf) {  
    callee(aaa_buf, 8);  
    callee(aaa_buf, 16);  
    callee(aaa_buf, 20);  
    callee(aaa_buf, 24);  
    callee(aaa_buf, 28);  
}
```

# Controlled stack overflow

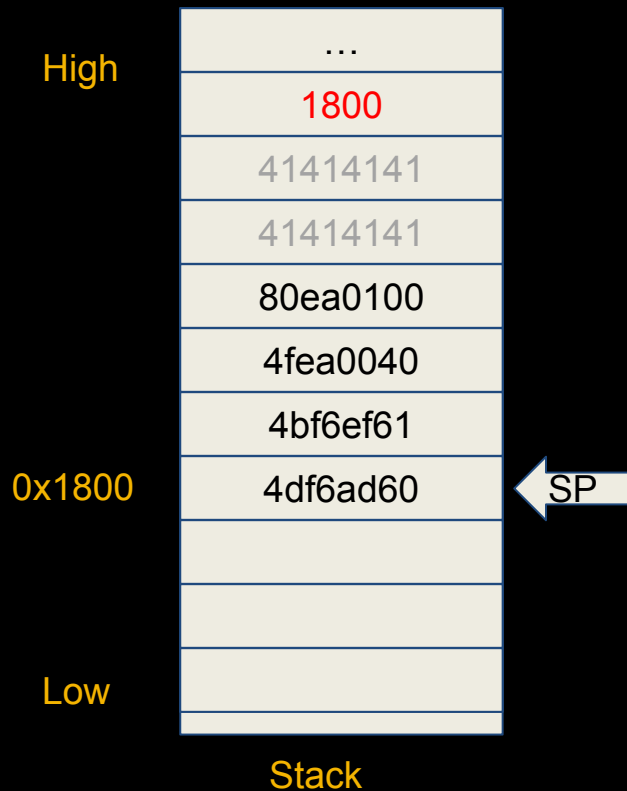
emBO++



```
void callee(char* buf, size_t len) { ← PC
    char my_buf[16] = {0};
    memcpy(my_buf, buf, len);
}

void caller(char* aaa_buf) {
    callee(aaa_buf, 8);
    callee(aaa_buf, 16);
    callee(aaa_buf, 20);
    callee(aaa_buf, 24);
    callee(aaa_buf, 28); ← LR
}
```

# Controlled stack overflow



```
void callee(char* buf, size_t len) {  
    char my_buf[16] = {0};  
    memcpy(my_buf, buf, len);  
}  
  
void caller(char* aaa_buf) {  
    callee(aaa_buf, 8);  
    callee(aaa_buf, 16);  
    callee(aaa_buf, 20);  
    callee(aaa_buf, 24);  
    callee(aaa_buf, 28);  
}
```

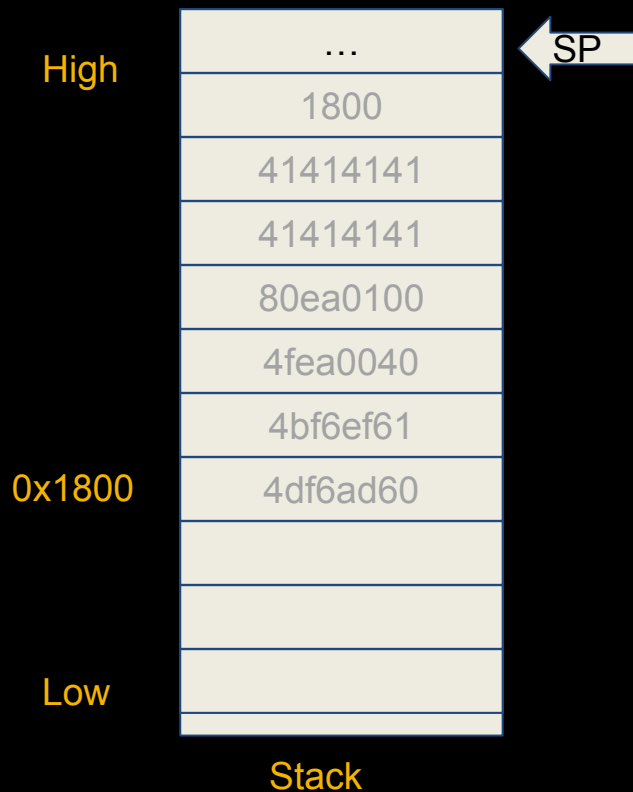
PC

LR

Two code snippets are shown. The first is a function 'callee' that takes a buffer and length, initializes a local buffer, and copies the input. A white arrow labeled 'PC' points to the 'memcpy' line. The second is a function 'caller' that calls 'callee' with different lengths. A blue arrow labeled 'LR' points to the last call to 'callee'.

# Controlled stack overflow

# emBO++



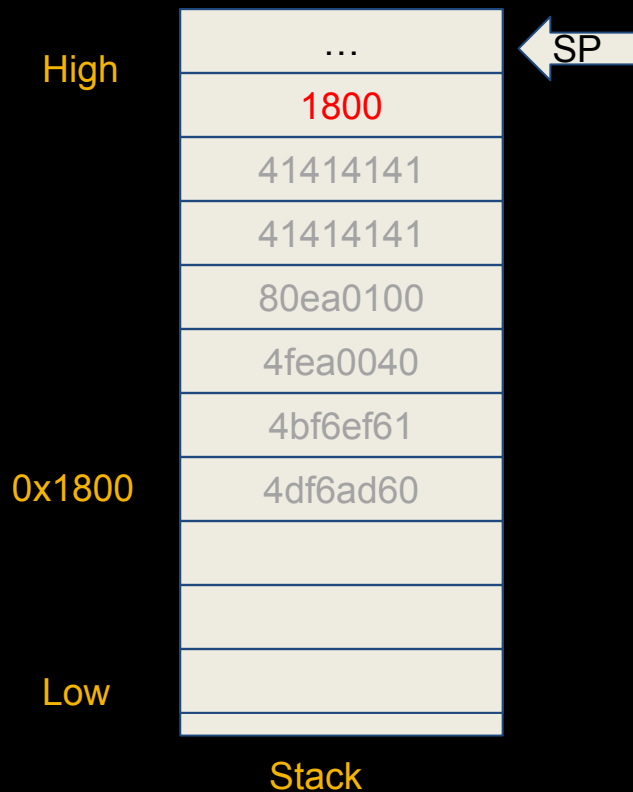
```
void callee(char* buf, size_t len) {  
    char my_buf[16] = {0};  
    memcpy(my_buf, buf, len);  
}  
  
void caller(char* aaa_buf) {  
    callee(aaa_buf, 8);  
    callee(aaa_buf, 16);  
    callee(aaa_buf, 20);  
    callee(aaa_buf, 24);  
    callee(aaa_buf, 28);  
}
```

← PC

← LR

# Controlled stack overflow

# emBO++



```
void callee(char* buf, size_t len) {  
    char my_buf[16] = {0};  
    memcpy(my_buf, buf, len);  
}
```

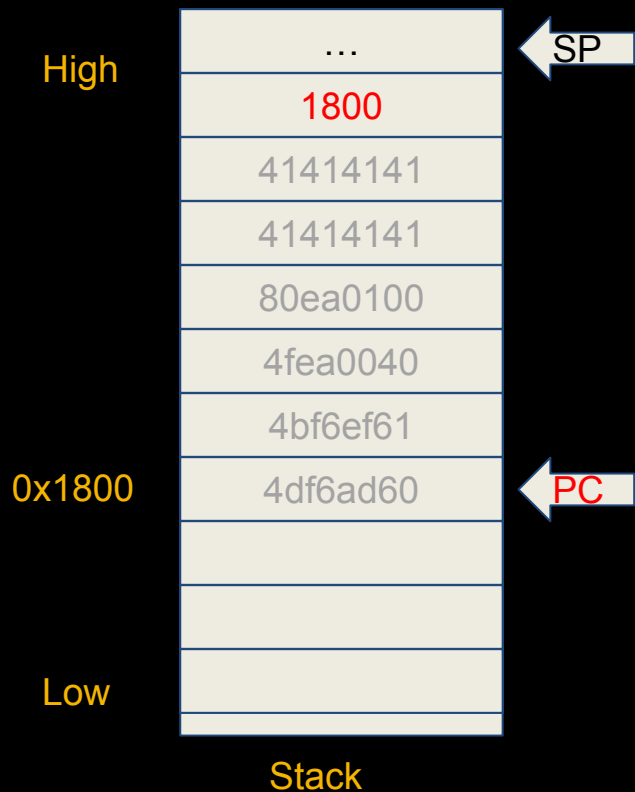
PC

```
void caller(char* aaa_buf) {  
    callee(aaa_buf, 8);  
    callee(aaa_buf, 16);  
    callee(aaa_buf, 20);  
    callee(aaa_buf, 24);  
    callee(aaa_buf, 28);  
}
```

LR

# Controlled stack overflow

# emBO++

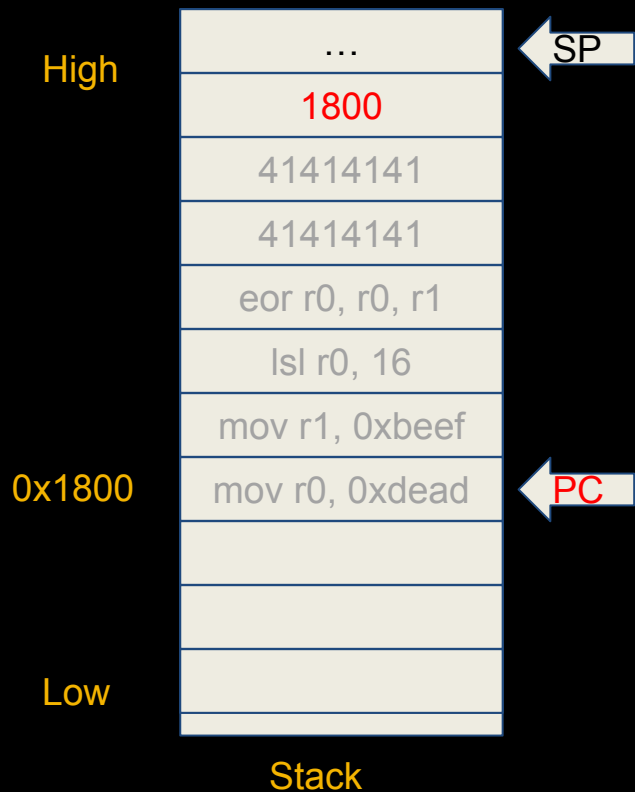


```
void callee(char* buf, size_t len) {  
    char my_buf[16] = {0};  
    memcpy(my_buf, buf, len);  
}  
  
void caller(char* aaa_buf) {  
    callee(aaa_buf, 8);  
    callee(aaa_buf, 16);  
    callee(aaa_buf, 20);  
    callee(aaa_buf, 24);  
    callee(aaa_buf, 28);  
}
```

LR

# Controlled stack overflow

# emBO++

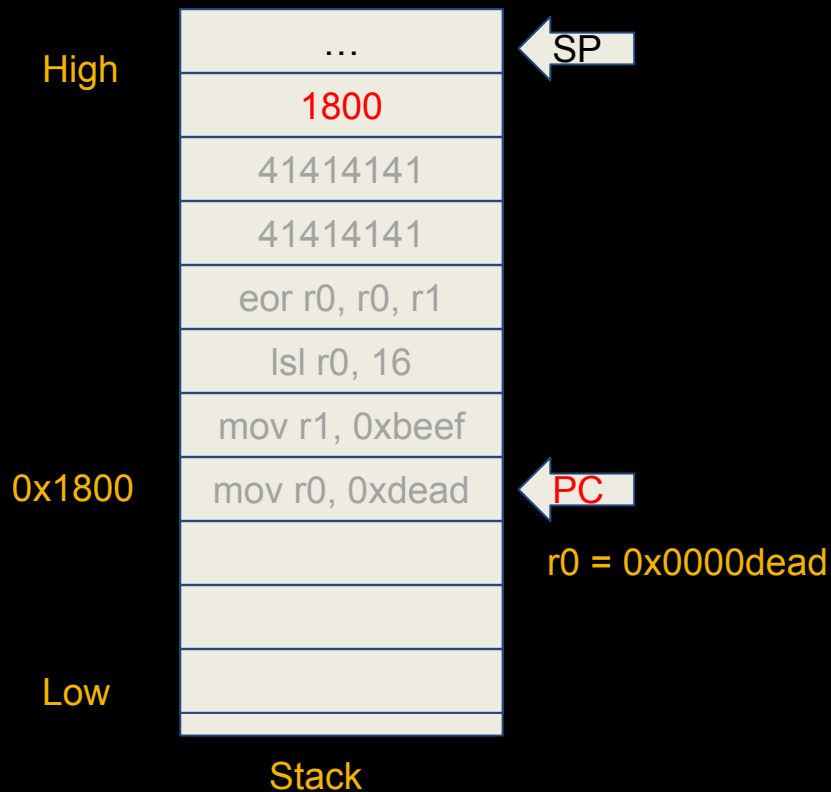


```
void callee(char* buf, size_t len) {  
    char my_buf[16] = {0};  
    memcpy(my_buf, buf, len);  
}  
  
void caller(char* aaa_buf) {  
    callee(aaa_buf, 8);  
    callee(aaa_buf, 16);  
    callee(aaa_buf, 20);  
    callee(aaa_buf, 24);  
    callee(aaa_buf, 28);  
}
```

← LR

# Controlled stack overflow

# emBO++

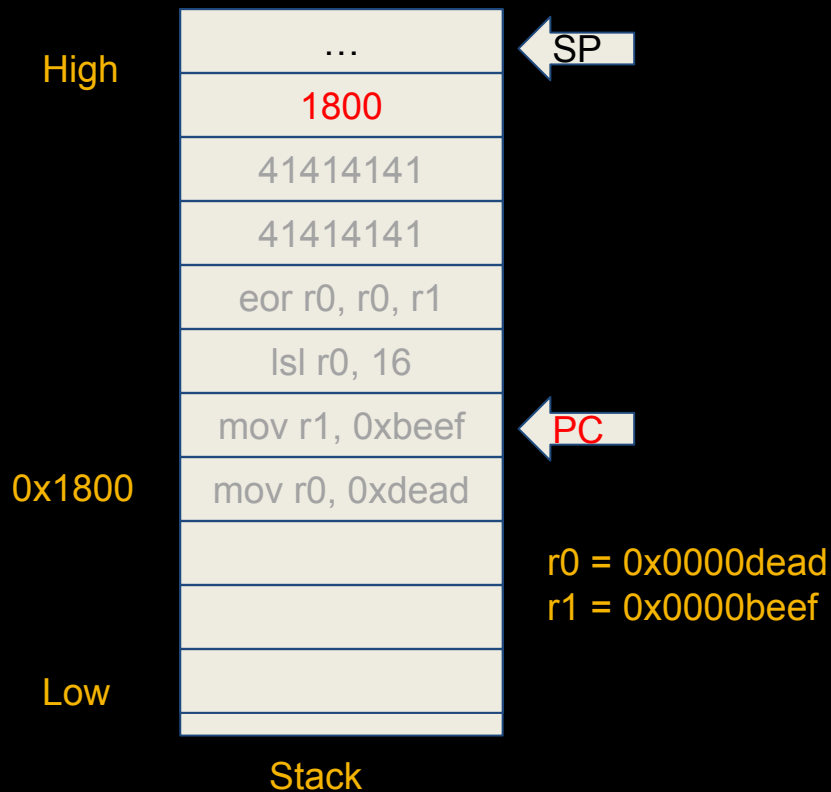


```
void callee(char* buf, size_t len) {  
    char my_buf[16] = {0};  
    memcpy(my_buf, buf, len);  
}  
  
void caller(char* aaa_buf) {  
    callee(aaa_buf, 8);  
    callee(aaa_buf, 16);  
    callee(aaa_buf, 20);  
    callee(aaa_buf, 24);  
    callee(aaa_buf, 28);  
}
```

LR



# Controlled stack overflow

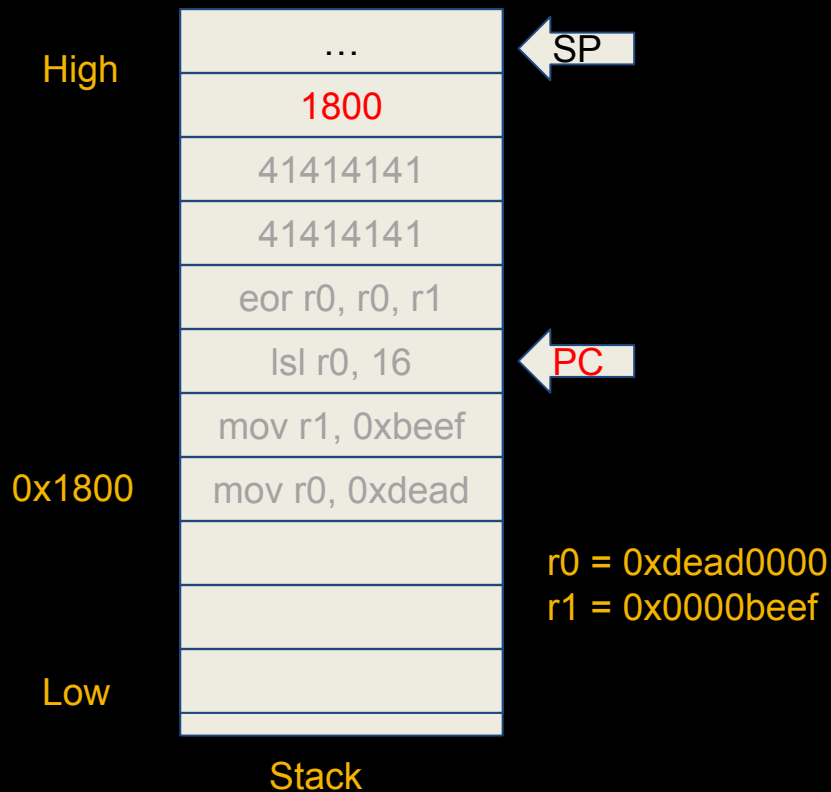


```
void callee(char* buf, size_t len) {  
    char my_buf[16] = {0};  
    memcpy(my_buf, buf, len);  
}  
  
void caller(char* aaa_buf) {  
    callee(aaa_buf, 8);  
    callee(aaa_buf, 16);  
    callee(aaa_buf, 20);  
    callee(aaa_buf, 24);  
    callee(aaa_buf, 28);  
}
```

LR

A code snippet showing two functions: 'callee' and 'caller'. 'callee' takes a character pointer 'buf' and a 'size\_t' 'len', declares a 16-byte array 'my\_buf', and copies 'buf' into it. 'caller' takes a character pointer 'aaa\_buf' and calls 'callee' five times with increasing offsets (8, 16, 20, 24, 28). A blue arrow labeled 'LR' points to the end of the 'caller' function.

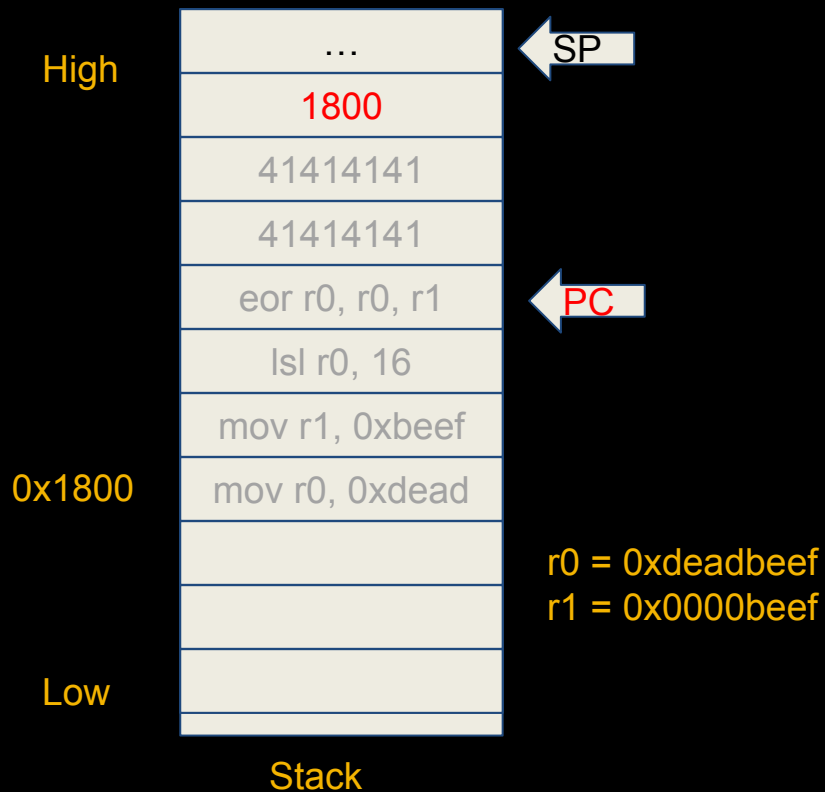
# Controlled stack overflow



```
void callee(char* buf, size_t len) {  
    char my_buf[16] = {0};  
    memcpy(my_buf, buf, len);  
}  
  
void caller(char* aaa_buf) {  
    callee(aaa_buf, 8);  
    callee(aaa_buf, 16);  
    callee(aaa_buf, 20);  
    callee(aaa_buf, 24);  
    callee(aaa_buf, 28);  
}
```

LR

# Controlled stack overflow

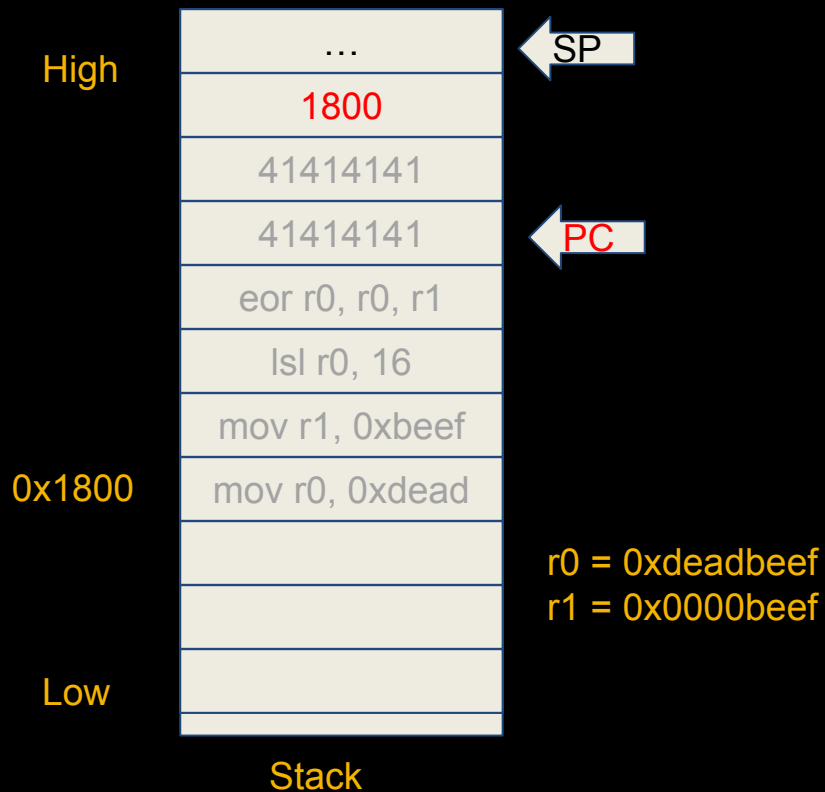


```
void callee(char* buf, size_t len) {  
    char my_buf[16] = {0};  
    memcpy(my_buf, buf, len);  
}  
  
void caller(char* aaa_buf) {  
    callee(aaa_buf, 8);  
    callee(aaa_buf, 16);  
    callee(aaa_buf, 20);  
    callee(aaa_buf, 24);  
    callee(aaa_buf, 28);  
}
```

← LR

A C code snippet is shown in a dark-themed editor. It defines two functions: 'callee' and 'caller'. 'callee' takes a character pointer 'buf' and a 'size\_t' 'len', declares a local array 'my\_buf' of size 16, and calls 'memcpy' to copy 'len' bytes from 'buf' to 'my\_buf'. 'caller' takes a character pointer 'aaa\_buf' and calls 'callee' five times with increasing offsets: 8, 16, 20, 24, and 28. A blue arrow labeled 'LR' points to the right, indicating the return path from the caller.

# Controlled stack overflow



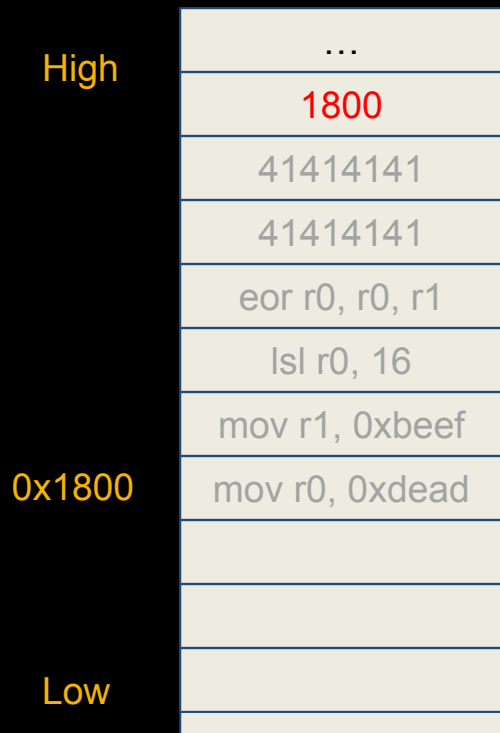
```
void callee(char* buf, size_t len) {  
    char my_buf[16] = {0};  
    memcpy(my_buf, buf, len);  
}  
  
void caller(char* aaa_buf) {  
    callee(aaa_buf, 8);  
    callee(aaa_buf, 16);  
    callee(aaa_buf, 20);  
    callee(aaa_buf, 24);  
    callee(aaa_buf, 28);  
}
```

LR

A C code snippet is shown. It defines two functions: 'callee' and 'caller'. 'callee' takes a character pointer 'buf' and a 'size\_t' 'len', declares a local array 'my\_buf' of size 16, and calls 'memcpy' to copy 'len' bytes from 'buf' to 'my\_buf'. 'caller' takes a character pointer 'aaa\_buf' and calls 'callee' five times with increasing offsets: 8, 16, 20, 24, and 28. To the right of the code, a blue arrow labeled 'LR' points to the right.

# Controlled stack overflow

# emBO++



Stack

← SP

BREAK ALL THE THINGS!



r0 = 0xdeadbeef  
r1 = 0x0000beef

```
char* buf, size_t len) {  
    buf[16] = {0};  
    y_buf, buf, len);  
  
char* aaa_buf) {  
    aa_buf, 8);  
    aa_buf, 16);  
    aa_buf, 20);  
    aa_buf, 24);  
    callee(aaa_buf, 28);  
}
```

← LR

# Exploiting things

1. Dump it ✓
2. Analyze it ✓
3. Find it ✓
4. Exploit it ✓



# Breaking Exploiting things

emBO++

1. Dump it ✓
2. Analyze it ✓
3. Find it ✓
4. Exploit it ✗

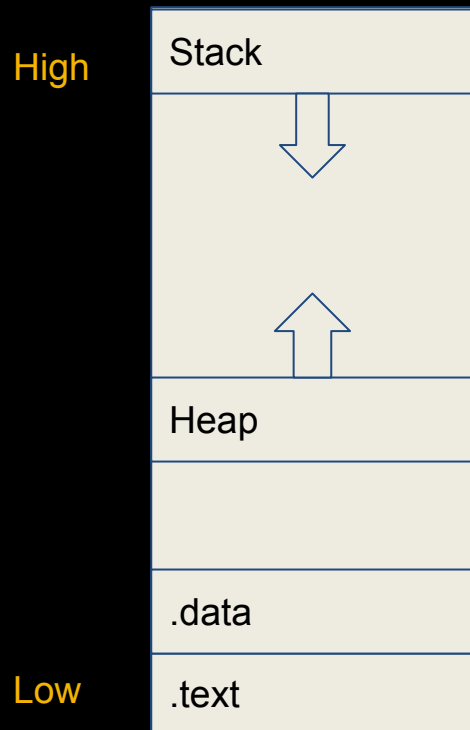


## Mitigation I: NX



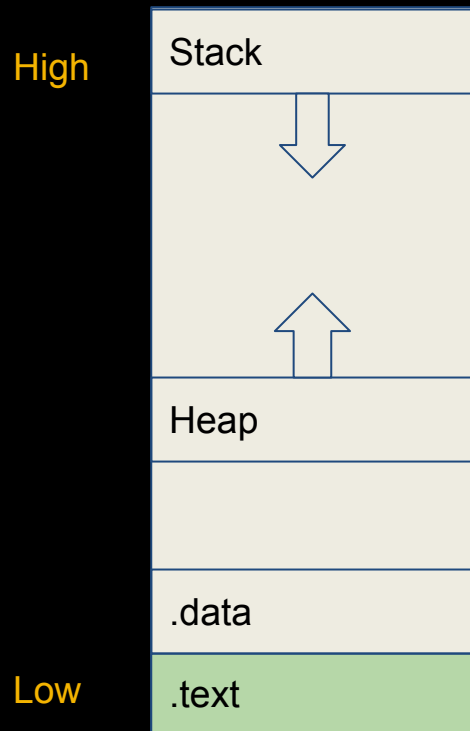
# No eXecute

# emBO++



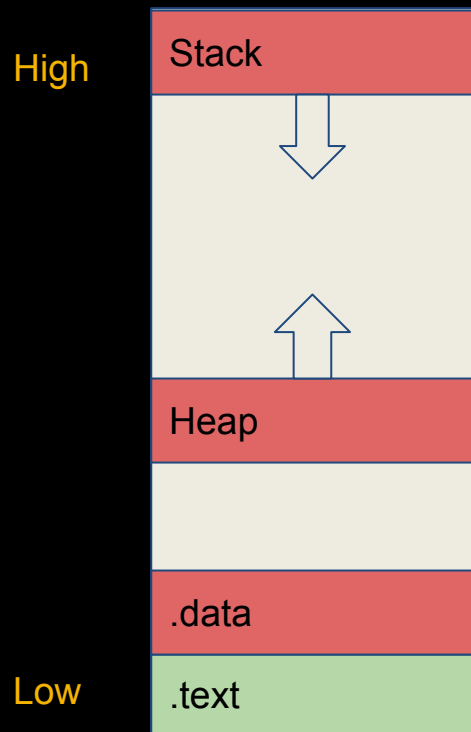
# No eXecute

# emBO++



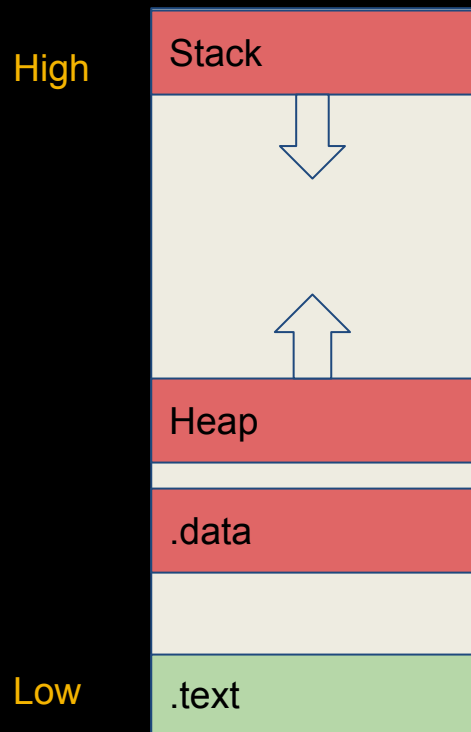
# No eXecute

# emBO++



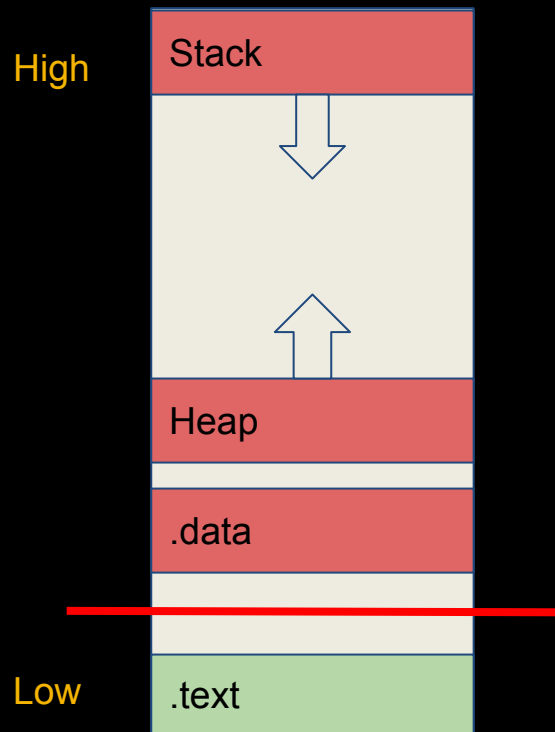
# No eXecute

# emBO++



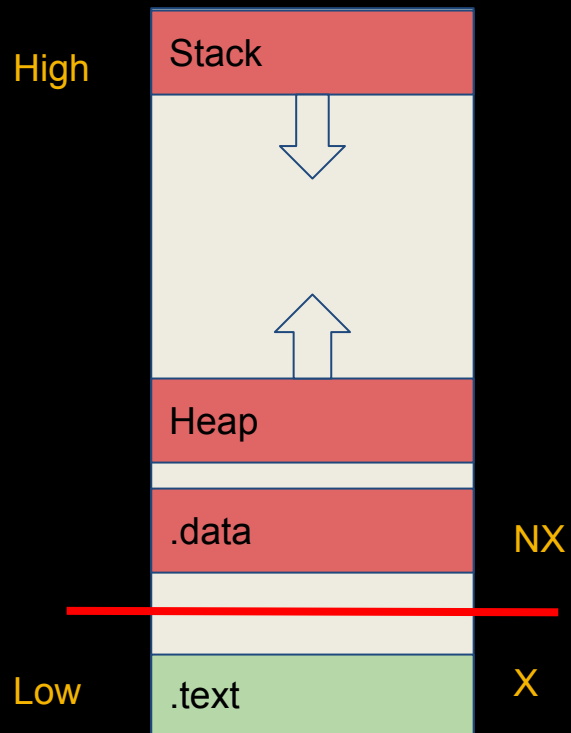
# No eXecute

# emBO++



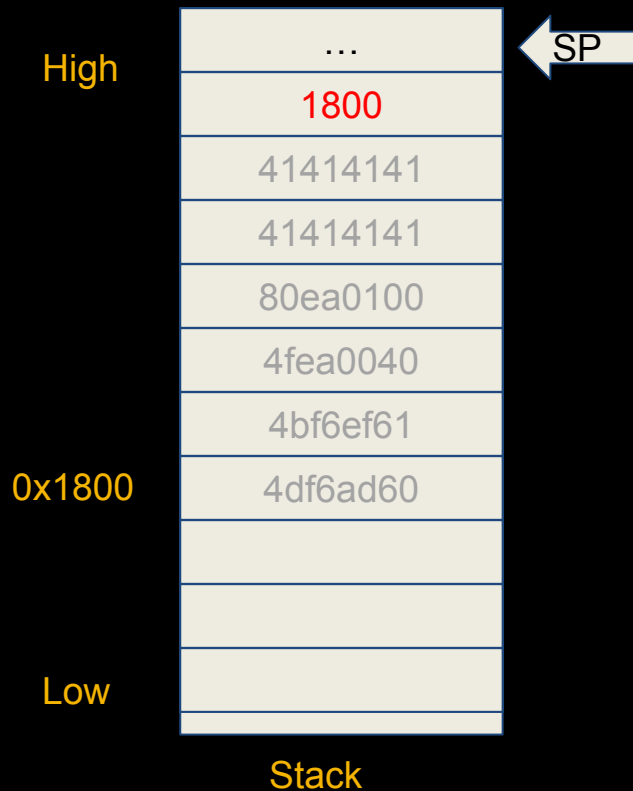
# No eXecute

# emBO++



# No eXecute

# emBO++



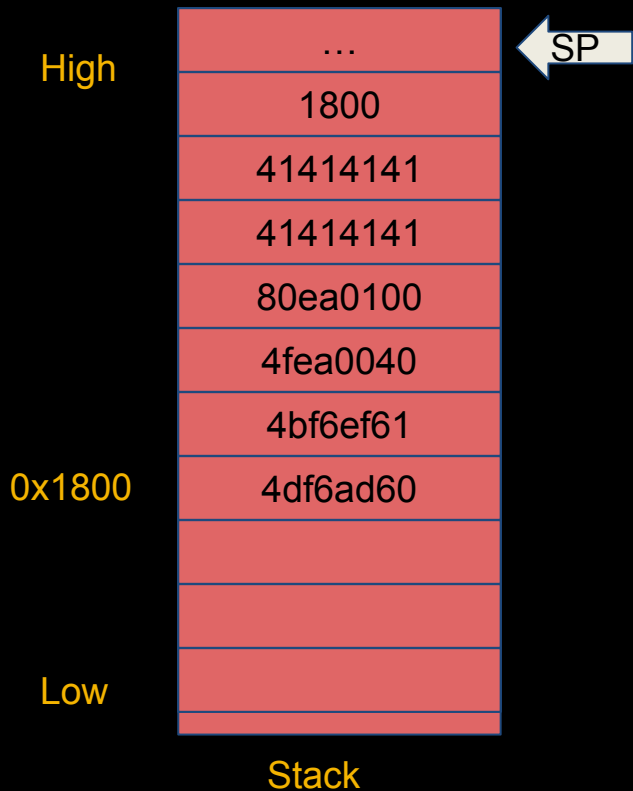
```
void callee(char* buf, size_t len) {  
    char my_buf[16] = {0};  
    memcpy(my_buf, buf, len);  
}  
  
void caller(char* aaa_buf) {  
    callee(aaa_buf, 8);  
    callee(aaa_buf, 16);  
    callee(aaa_buf, 20);  
    callee(aaa_buf, 24);  
    callee(aaa_buf, 28);  
}
```

← PC

← LR

# No eXecute

# emBO++



```
void callee(char* buf, size_t len) {  
    char my_buf[16] = {0};  
    memcpy(my_buf, buf, len);  
}  
  
void caller(char* aaa_buf) {  
    callee(aaa_buf, 8);  
    callee(aaa_buf, 16);  
    callee(aaa_buf, 20);  
    callee(aaa_buf, 24);  
    callee(aaa_buf, 28);  
}
```

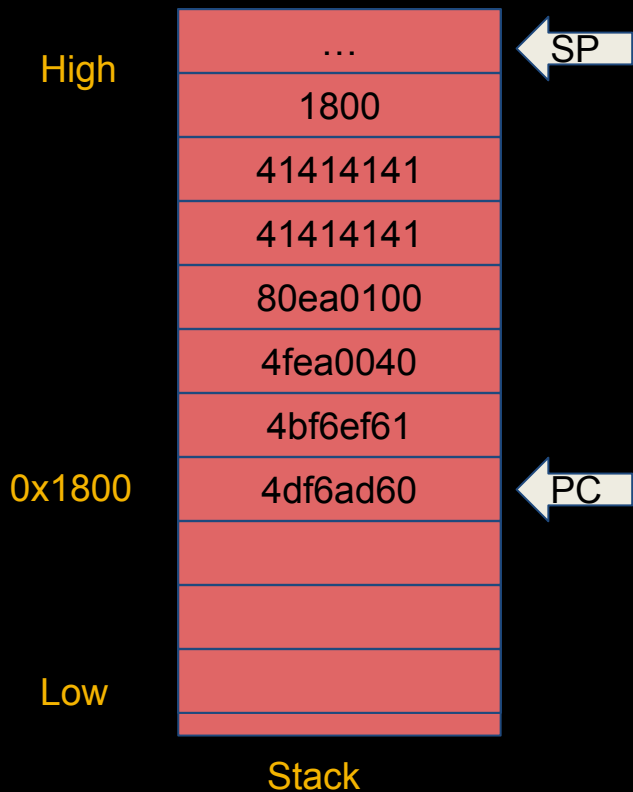
PC

LR



# No eXecute

# emBO++



```
void callee(char* buf, size_t len) {  
    char my_buf[16] = {0};  
    memcpy(my_buf, buf, len);  
}  
  
void caller(char* aaa_buf) {  
    callee(aaa_buf, 8);  
    callee(aaa_buf, 16);  
    callee(aaa_buf, 20);  
    callee(aaa_buf, 24);  
    callee(aaa_buf, 28);  
}
```

LR

# No eXecute

# emBO++



```
void callee(char* buf, size_t len) {  
    char my_buf[16] = {0};  
    memcpy(my_buf, buf, len);  
}  
  
void caller(char* aaa_buf) {  
    callee(aaa_buf, 8);  
    callee(aaa_buf, 16);  
    callee(aaa_buf, 20);  
    callee(aaa_buf, 24);  
    callee(aaa_buf, 28);  
}
```

LR

The diagram shows two code blocks. The first block is a function 'callee' that takes a buffer and length, initializes a local buffer, and copies the input. The second block is a function 'caller' that calls 'callee' with different offsets (8, 16, 20, 24, 28). A blue arrow labeled 'LR' points to the right, indicating the link register.

# Breaking Exploiting things

emBO++

1. Dump it ✓
2. Analyze it ✓
3. Find it ✓
4. Exploit it ✗



# Breaking Breaking



mBO++

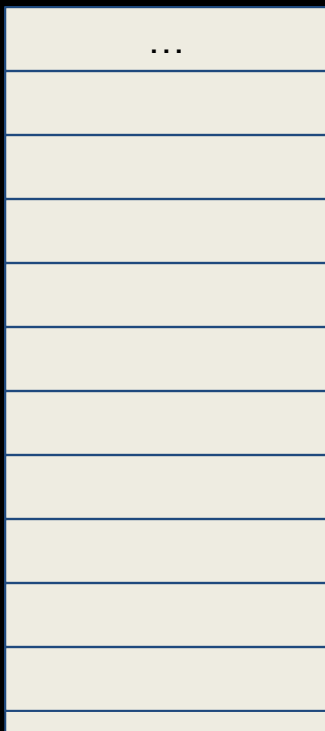
1. Dump it ✓
2. Analyze it ✓
3. Find it ✓
4. Exploit it ✓

## Stack buffer overflow II

# Return of the Operation

# emBO++

High



r0 = 0xdeadbeef

Low

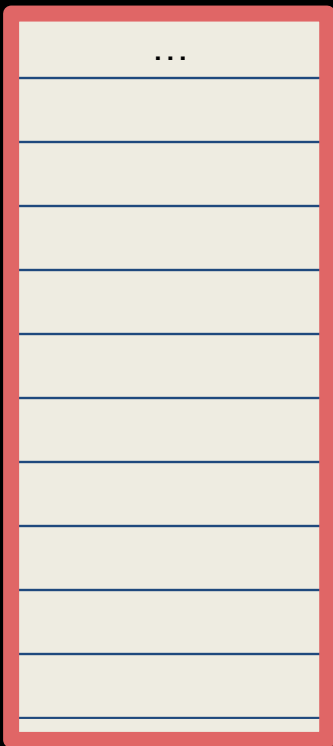
Stack

```
void callee(char* buf, size_t len) {  
    char my_buf[16] = {0};  
    memcpy(my_buf, buf, len);  
}  
  
void caller(char* aaa_buf) {  
    callee(aaa_buf, 28);  
}  
  
void return_deadbeef() {  
    return 0xdeadbeef;  
}
```

# Return of the Operation

# emBO++

High



r0 = 0xdeadbeef

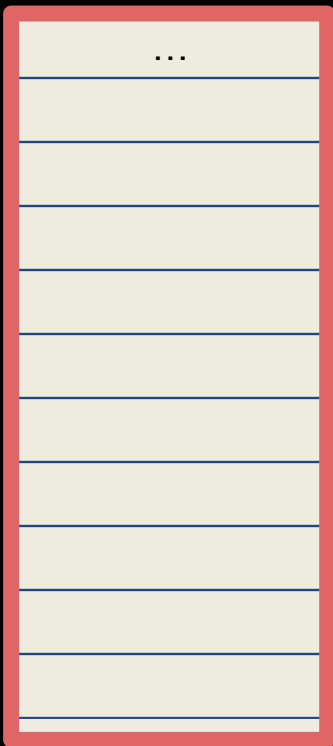
Low

```
void callee(char* buf, size_t len) {  
    char my_buf[16] = {0};  
    memcpy(my_buf, buf, len);  
}  
  
void caller(char* aaa_buf) {  
    callee(aaa_buf, 28);  
}  
  
void return_deadbeef() {  
    return 0xdeadbeef;  
}
```

# Return of the Operation

# emBO++

High



r0 = 0xdeadbeef

Low

Stack

0x800

```
void callee(char* buf, size_t len) {  
    char my_buf[16] = {0};  
    memcpy(my_buf, buf, len);  
}  
  
void caller(char* aaa_buf) {  
    callee(aaa_buf, 28);  
}  
  
void return_deadbeef() {  
    return 0xdeadbeef;  
}
```

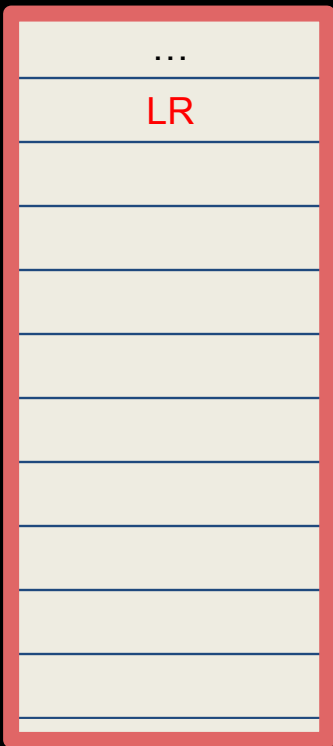
r0 = 0xdeadbeef



# Return of the Operation

# emBO++

High



Low

Stack

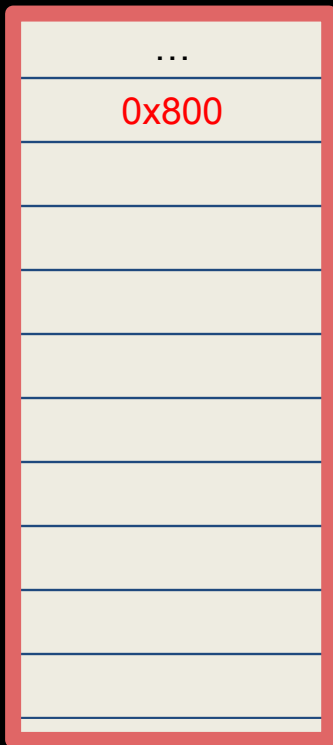
0x800

```
void callee(char* buf, size_t len) {  
    char my_buf[16] = {0};  
    memcpy(my_buf, buf, len);  
}  
  
void caller(char* aaa_buf) {  
    callee(aaa_buf, 28);  
}  
  
void return_deadbeef() {  
    return 0xdeadbeef;  
}
```

# Return of the Operation

# emBO++

High



0x800

Low

Stack

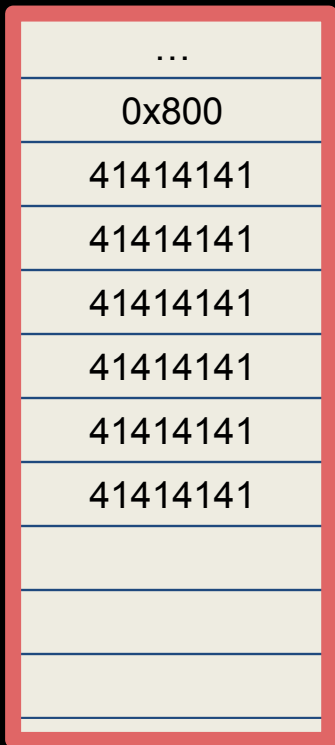
0x800

```
void callee(char* buf, size_t len) {  
    char my_buf[16] = {0};  
    memcpy(my_buf, buf, len);  
}  
  
void caller(char* aaa_buf) {  
    callee(aaa_buf, 28);  
}  
  
void return_deadbeef() {  
    return 0xdeadbeef;  
}
```

# Return of the Operation

# emBO++

High



Low

Stack

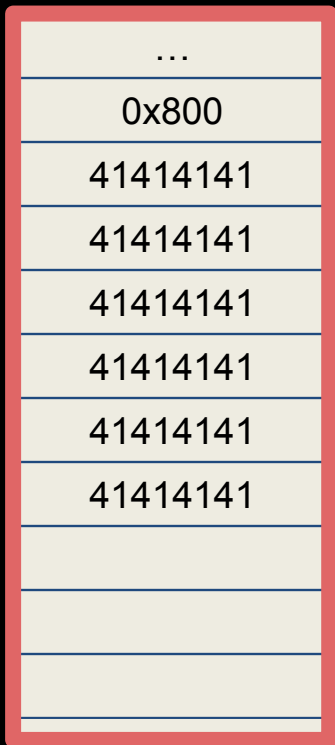
0x800

```
void callee(char* buf, size_t len) {  
    char my_buf[16] = {0};  
    memcpy(my_buf, buf, len);  
}  
  
void caller(char* aaa_buf) {  
    callee(aaa_buf, 28);  
}  
  
void return_deadbeef() {  
    return 0xdeadbeef;  
}
```

# Return of the Operation

# emBO++

High



Low

Stack

0x800

```
void callee(char* buf, size_t len) {  
    char my_buf[16] = {0};  
    memcpy(my_buf, buf, len);  
}  
  
void caller(char* aaa_buf) {  
    callee(aaa_buf, 28);  
}  
  
void return_deadbeef() {  
    return 0xdeadbeef;  
}
```

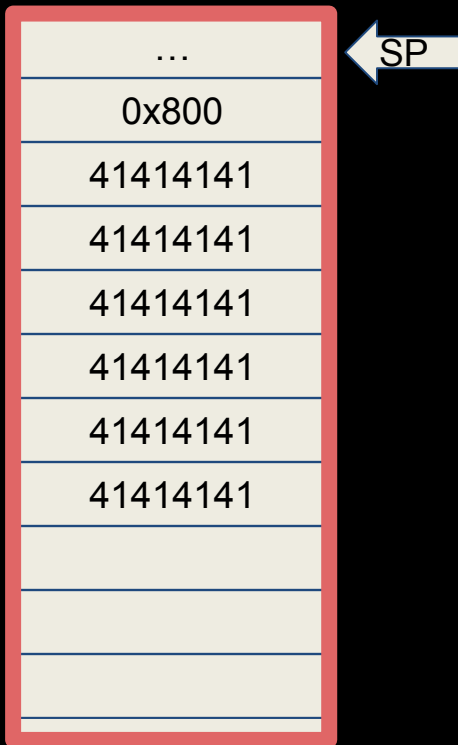
PC

LR

# Return of the Operation

# emBO++

High



Low

Stack

0x800

```
void callee(char* buf, size_t len) {  
    char my_buf[16] = {0};  
    memcpy(my_buf, buf, len);  
}  
  
void caller(char* aaa_buf) {  
    callee(aaa_buf, 28);  
}  
  
void return_deadbeef() {  
    return 0xdeadbeef;  
}
```

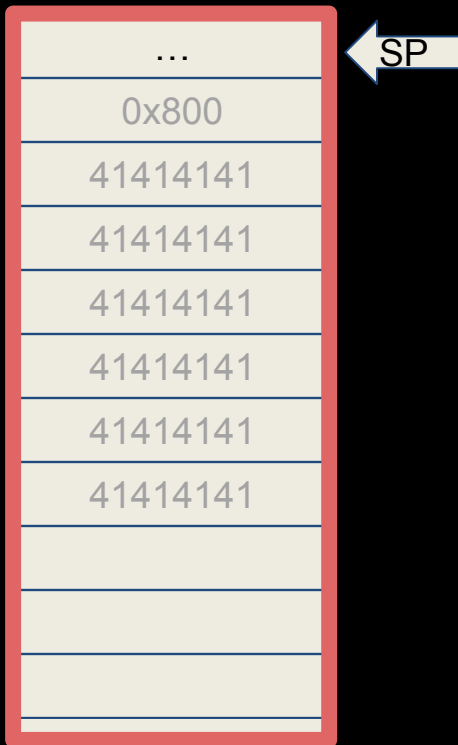
PC

LR

# Return of the Operation

# emBO++

High



Low

Stack

0x800

```
void callee(char* buf, size_t len) {  
    char my_buf[16] = {0};  
    memcpy(my_buf, buf, len);  
}  
  
void caller(char* aaa_buf) {  
    callee(aaa_buf, 28);  
}  
  
void return_deadbeef() {  
    return 0xdeadbeef;  
}
```

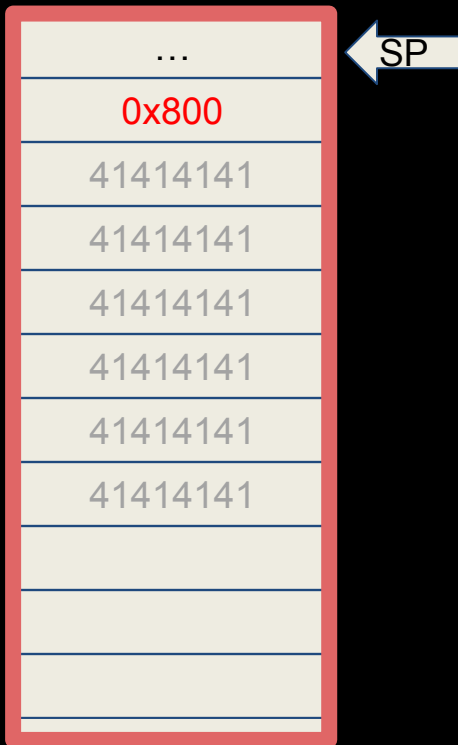
PC

LR

# Return of the Operation

# emBO++

High



Low

Stack

0x800

```
void callee(char* buf, size_t len) {  
    char my_buf[16] = {0};  
    memcpy(my_buf, buf, len);  
}  
  
void caller(char* aaa_buf) {  
    callee(aaa_buf, 28);  
}  
  
void return_deadbeef() {  
    return 0xdeadbeef;  
}
```

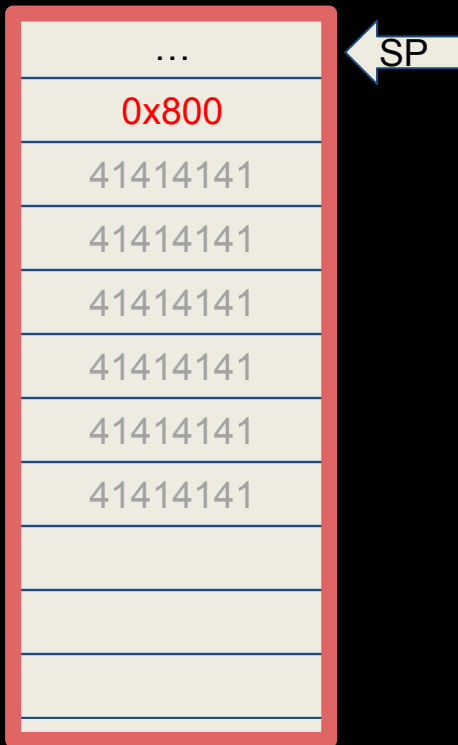
PC

LR

# Return of the Operation

# emBO++

High



Low

Stack

0x800

```
void callee(char* buf, size_t len) {  
    char my_buf[16] = {0};  
    memcpy(my_buf, buf, len);  
}  
  
void caller(char* aaa_buf) {  
    callee(aaa_buf, 28);  
}  
  
void return_deadbeef() {  
    return 0xdeadbeef;  
}
```

LR

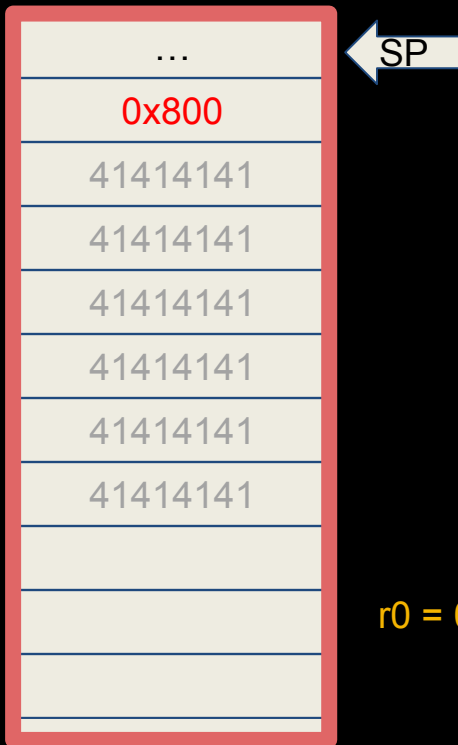
PC



# Return of the Operation

# emBO++

High



0x800

r0 = 0xdeadbeef

Low

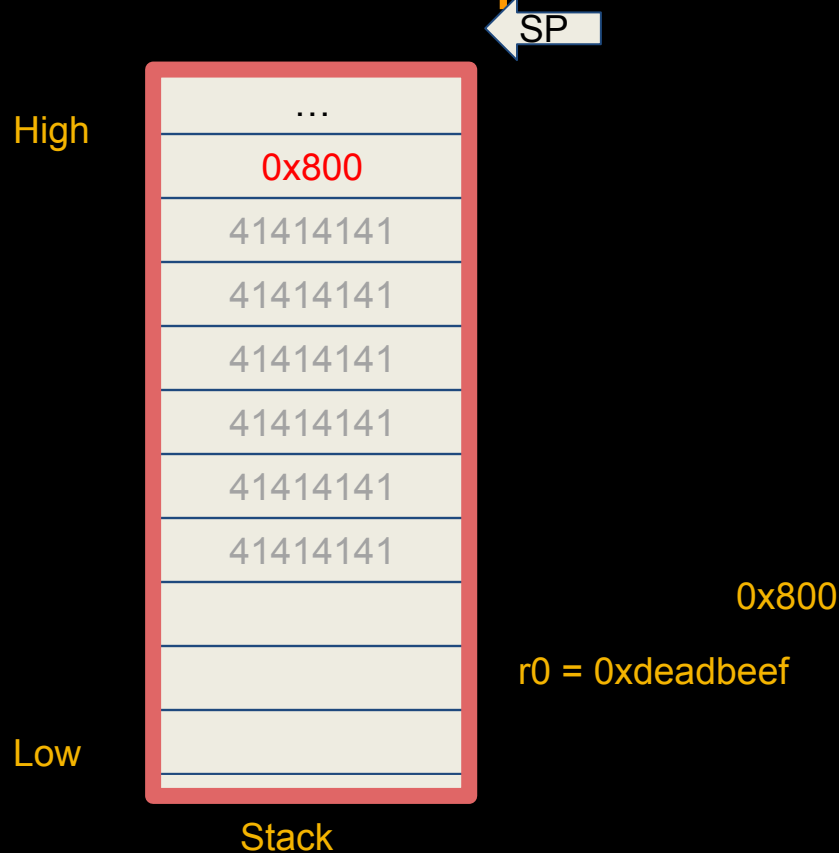
Stack

```
void callee(char* buf, size_t len) {  
    char my_buf[16] = {0};  
    memcpy(my_buf, buf, len);  
}  
  
void caller(char* aaa_buf) {  
    callee(aaa_buf, 28);  
}  
  
void return_deadbeef() {  
    return 0xdeadbeef;  
}
```

LR

PC

## Return of the Operation



```
void callee(char* buf, size_t len) {  
    char my_buf[16] = {0};  
    memcpy(my_buf, buf, len);  
}  
  
void caller(char* aaa_buf) {  
    callee(aaa_buf, 28);  
}  
  
void return_deadbeef() {  
    return 0xdeadbeef;  
}
```

LR

PC

# Breaking Breaking



mBO++

1. Dump it ✓
2. Analyze it ✓
3. Find it ✓
4. Exploit it ✓

# Breaking Breaking Exploiting things

1. Dump it ✓
2. Analyze it ✓
3. Find it ✓
4. Exploit it ✓✓✓

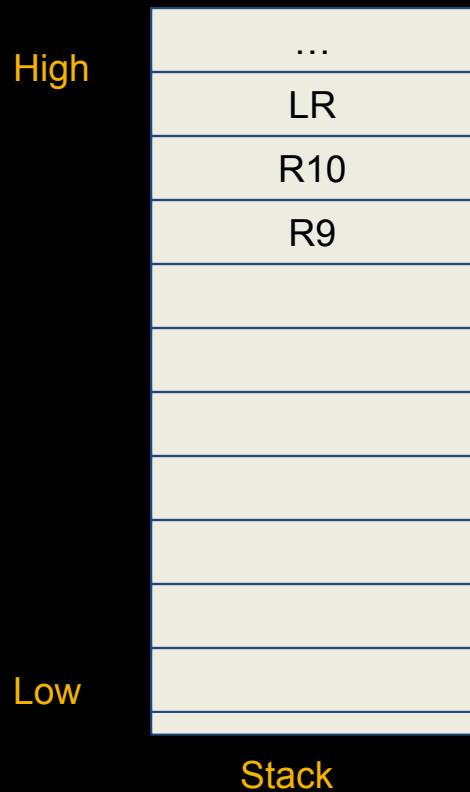


mBO++  
king Breaking

## Mitigations II

# Stack canaries

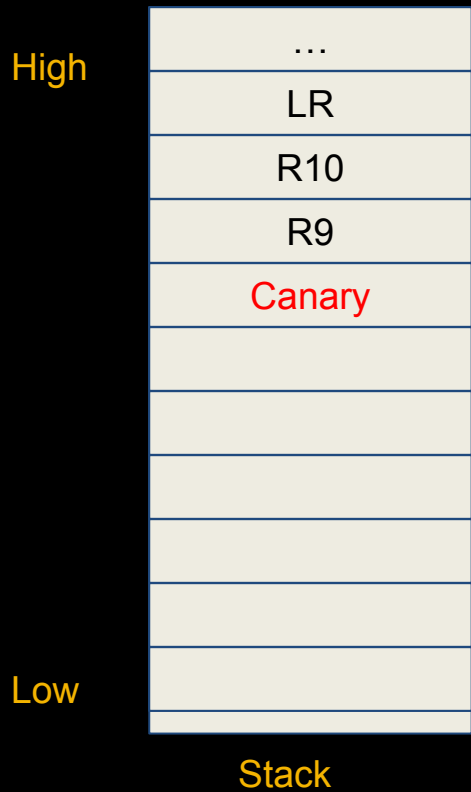
emBO++



# Stack

# Stack canaries

emBO++



# Stack canaries



High

...
LR
R10
R9
Canary
41414141
41414141
41414141
41414141

Low

Stack



# Stack canaries



High

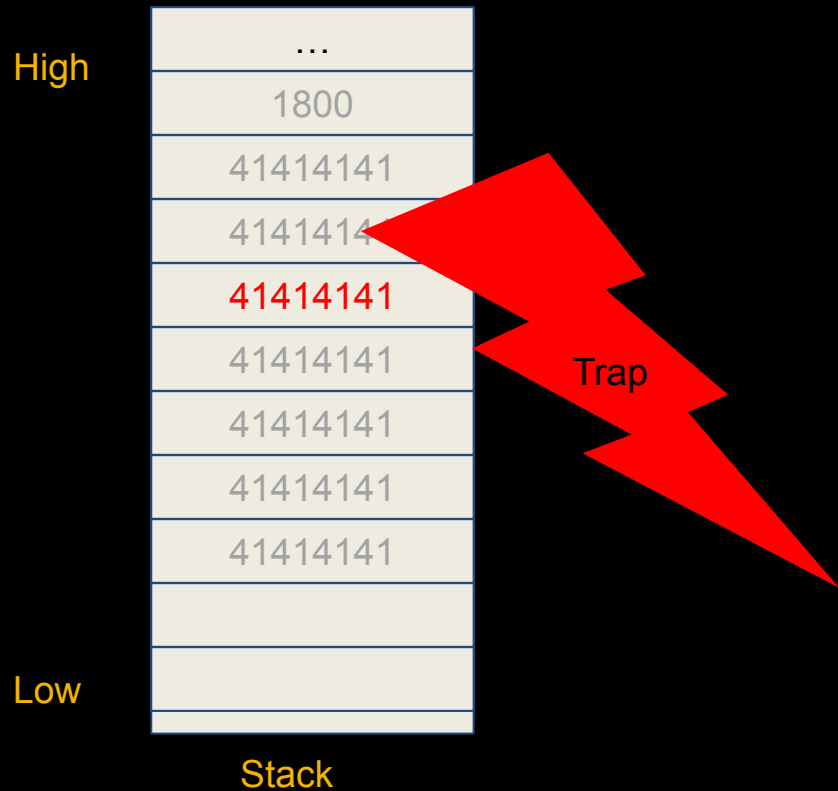
...
1800
41414141
41414141
41414141
41414141
41414141
41414141
41414141

Low

Stack

# Stack canaries

emBO++



# Control Flow Integrity - CFI



High

...
LR
R10
R9
Canary
41414141
41414141
41414141
41414141

Low

Stack

# Control Flow Integrity - CFI



High

...
LR
R10
R9
Canary
41414141
41414141
41414141
41414141

Low

Stack

# Control Flow Integrity - CFI



High

...
LR
R10
R9
Canary
41414141
41414141
41414141
41414141

Low

Stack

# Control Flow Integrity - CFI



High

...
1800
R10
R9
Canary
41414141
41414141
41414141
41414141

Low

Stack

# Control Flow Integrity - CFI



High

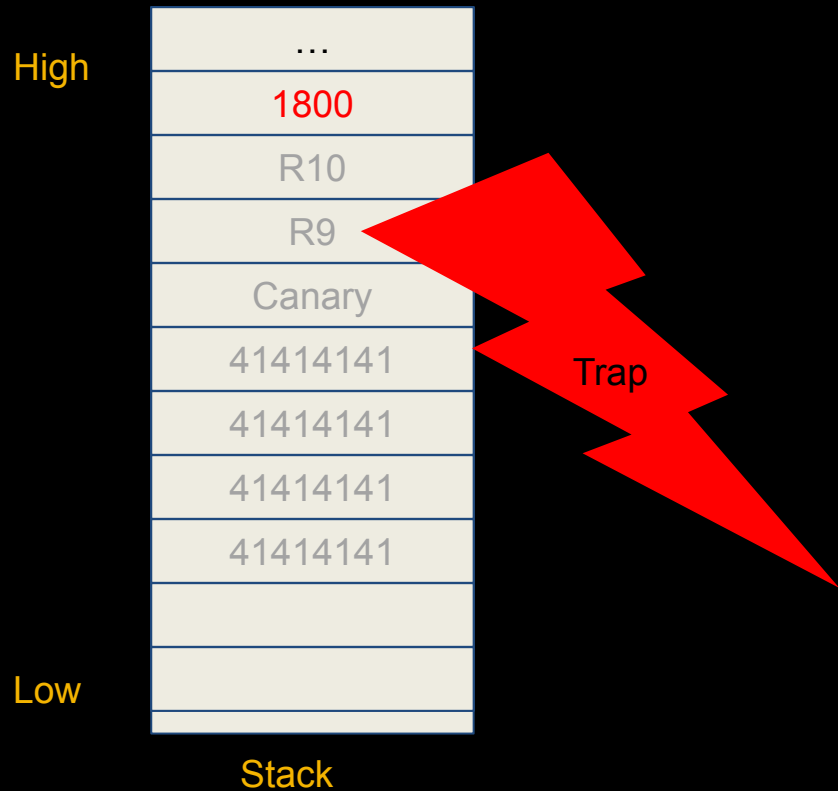
...
1800
R10
R9
Canary
41414141
41414141
41414141
41414141

Low

Stack

# Control Flow Integrity - CFI

emBO++





# Mitigations

- NX
- Stack canaries
- CFI
- ASLR
- Binary diversity
- Anti-RE/Anti-Dump

# Mitigations

- NX
- Stack canaries
- CFI
- ASLR
- Binary diversity
- Anti-RE/Anti-Dump

# Mitigations

- NX
- Stack canaries
- CFI
- ASLR
- Binary diversity
- Anti-RE/Anti-Dump

# Mitigations

- NX
- Stack canaries
- CFI
- ASLR
- Binary diversity
- Anti-RE/Anti-Dump

# Mitigations

- NX
- Stack canaries
- CFI
- ASLR
- Binary diversity
- Anti-RE/Anti-Dump

# Mitigations

- NX
- Stack canaries
- CFI
- ASLR
- Binary diversity
- Anti-RE/Anti-Dump

# Mitigations

- NX
- Stack canaries
- CFI
- ASLR
- Binary diversity
- Anti-RE/Anti-Dump

# Conclusion

- Many attacks, but also many defenses
- Try not to create bugs ;)
- ... but always consider what happens if you do



# Conclusion

emBO++

- Many attacks, but also many defenses
- Try not to create bugs ;)
- ... but always consider what happens if you do
- Have fun :)



[bkollenda@emproof.com](mailto:bkollenda@emproof.com)  
[emproof.com/blogs/](https://emproof.com/blogs/)