No eXecute (NX)

WHAT IS IT?

Defines areas of memory as either instructions

Writable XOR Executable

WHAT IS IT?

Shellcode is useless
:(

Is it over?

Return-Oriented Programming

WHAT IS IT?

Chaining a bunch of code already present in the binary itself

WHAT IS IT?

In fact, you've already done it twice just now!

Ret Gadget
Jmp RSP Gadget

Passing Parameters

x86

Params are pushed onto the stack

x86_64

Parameters are stored in registers

Sample Code

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

int main() {
    func(1, 2, 3, 4, 5, 6);
}
```

x86

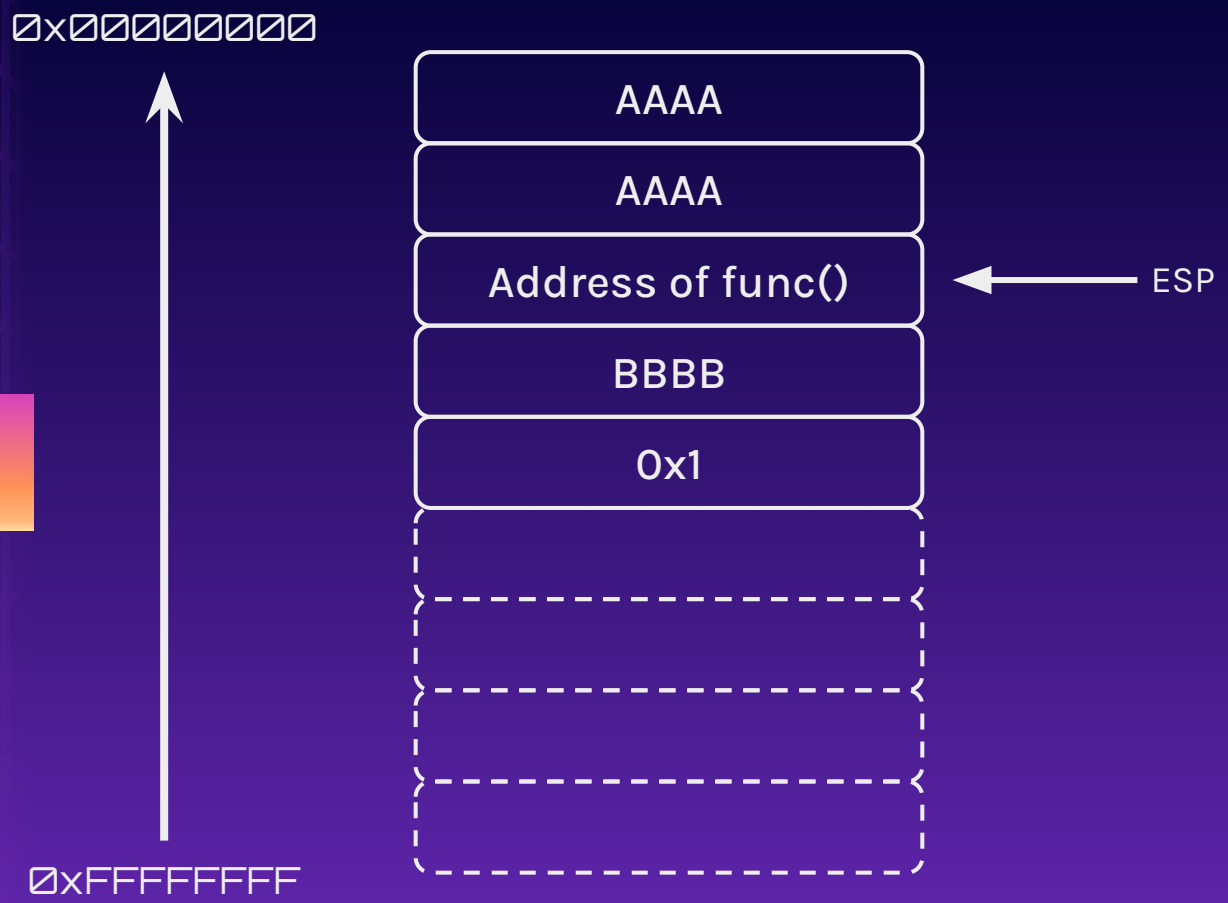
ESP →	0x1
	0x2
	0x3
	0x4
	0x5
EBP →	0x6

x86_64

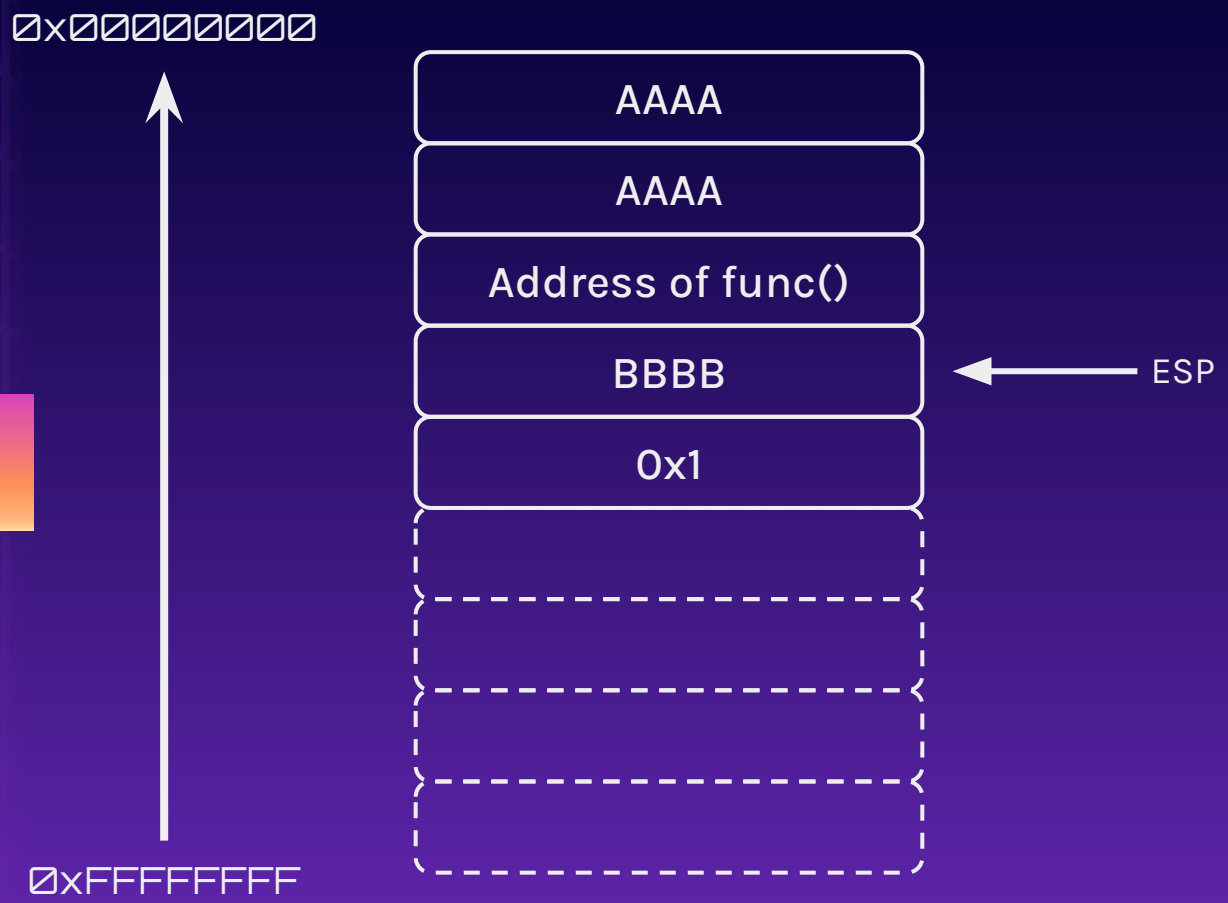
RDI	0x1
RSI	0x2
RDX	0x3
RCX	0x4
R8	0x5
R9	0x6

Sample Code

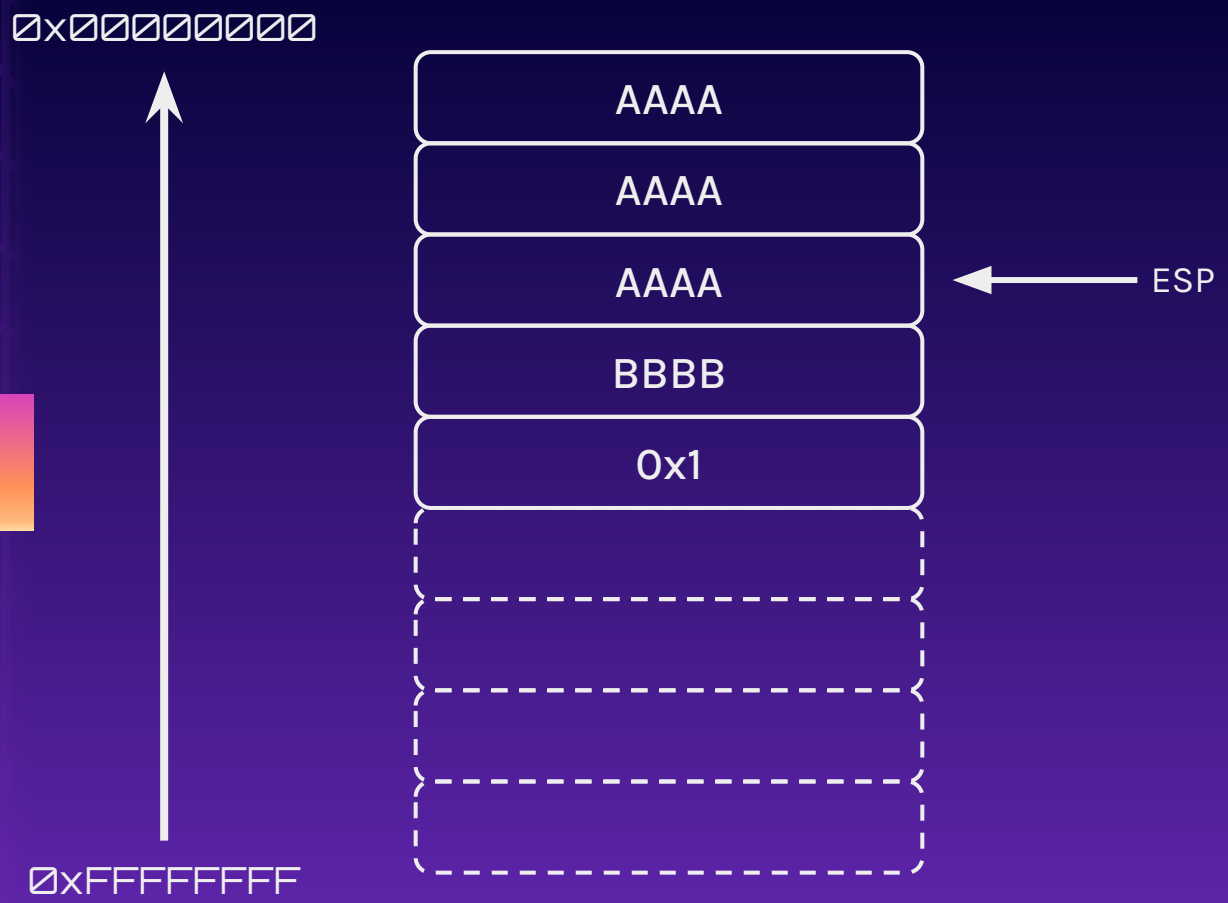
```
int func(int arg1) {
    return arg1;
}

int main() {
    char buffer[256];
    gets(buffer);
}
```



Next instruction
ret

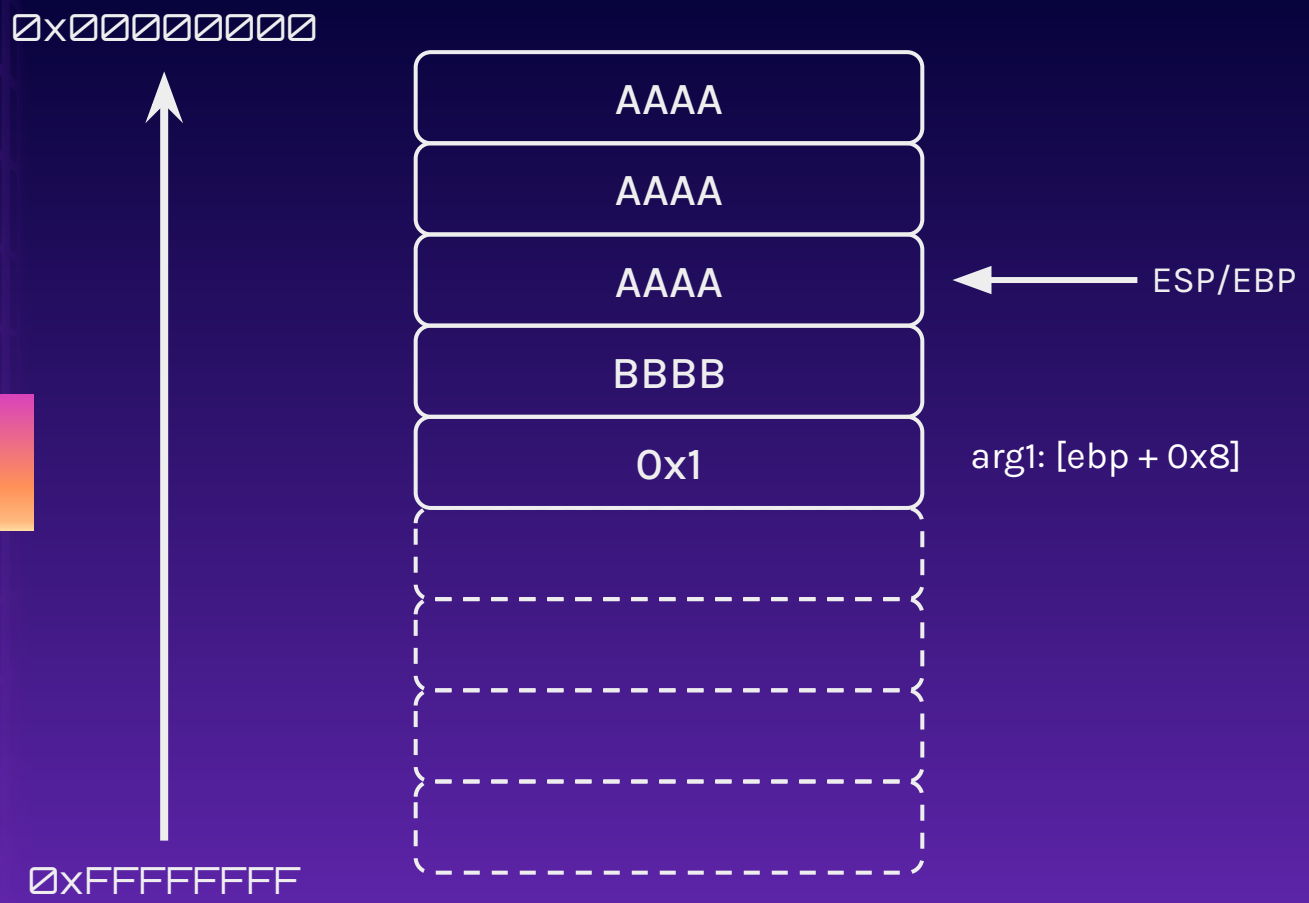


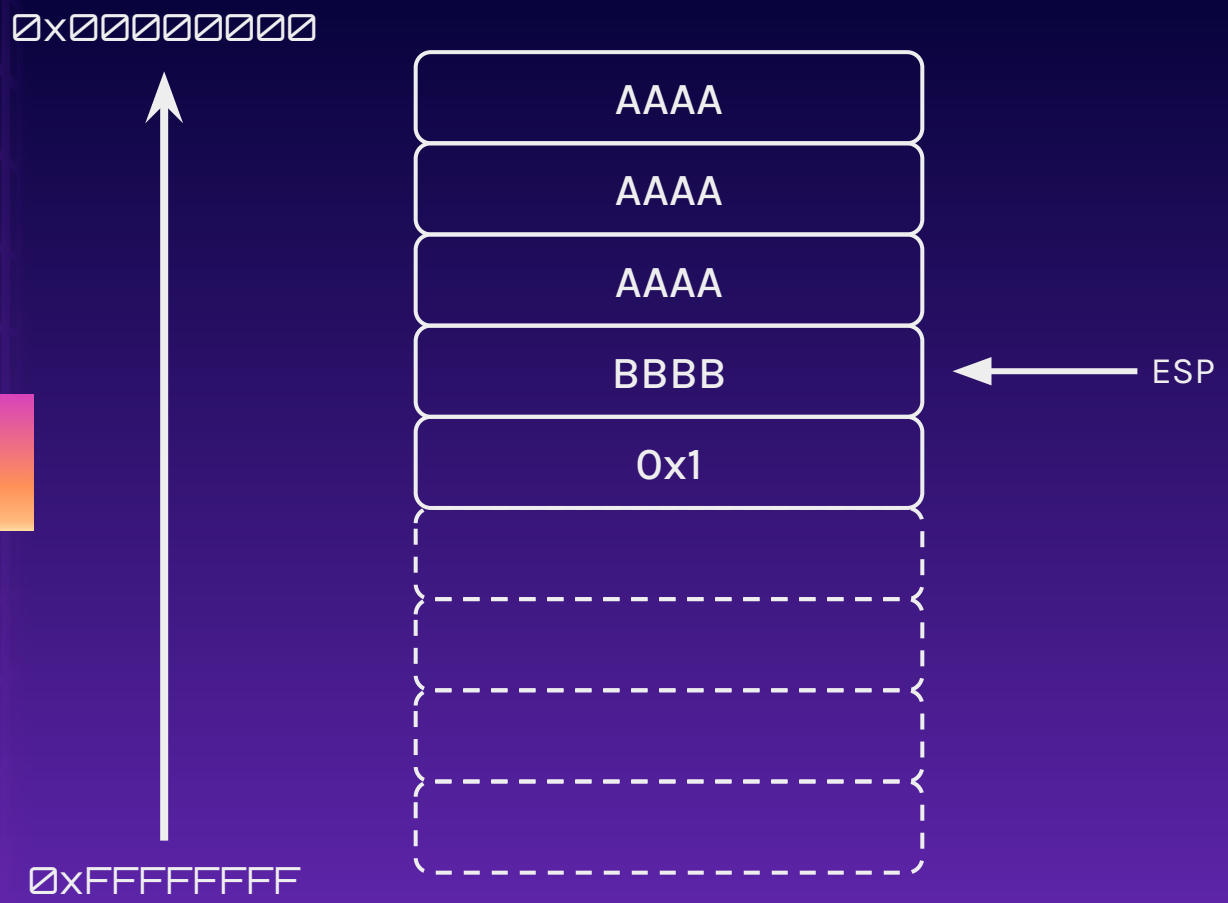
Next instruction
push ebp



Next instruction
mov ebp, esp

Next instruction
pop ebp





Next instruction
ret

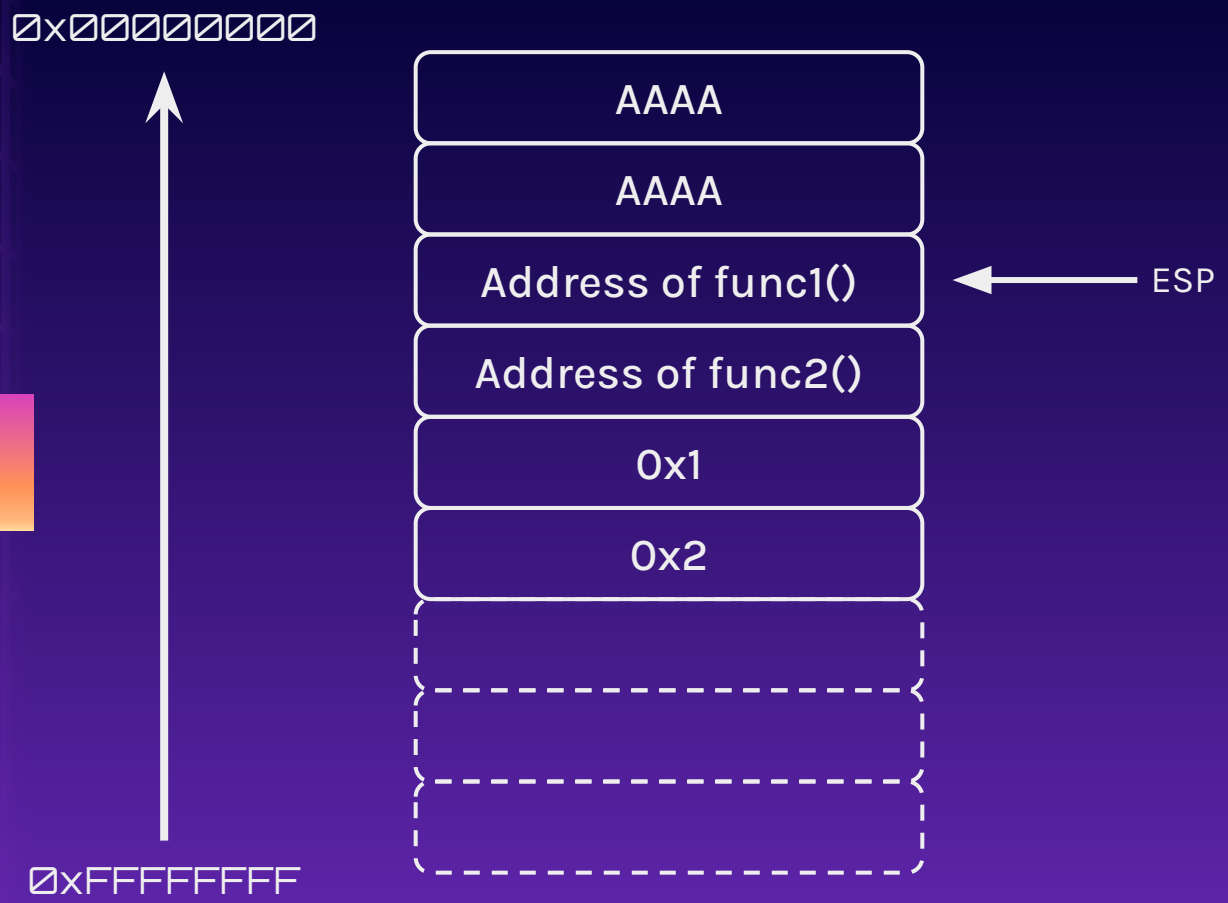
WHAT HAPPENS?

EIP goes to 0x42424242

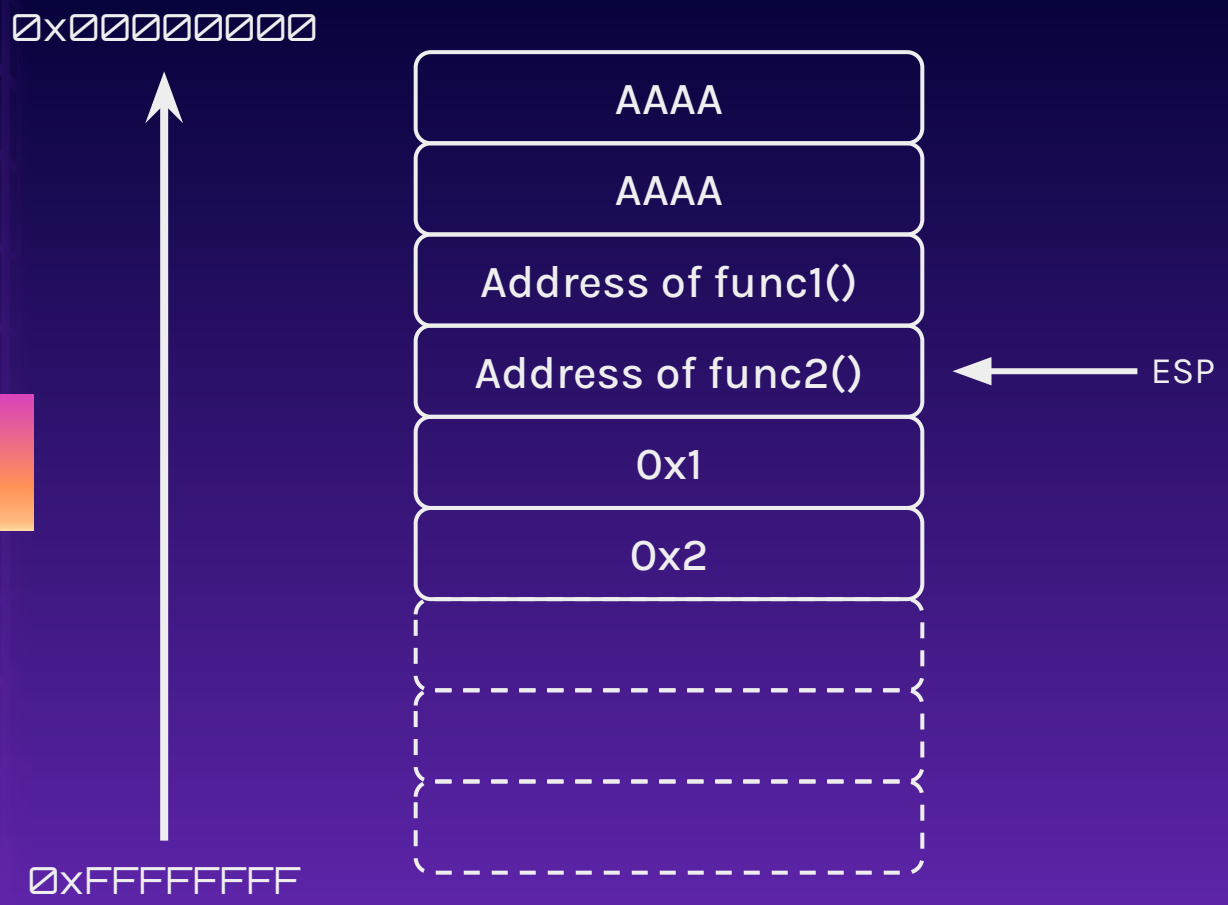
We can chain multiple return addresses

Sample Code

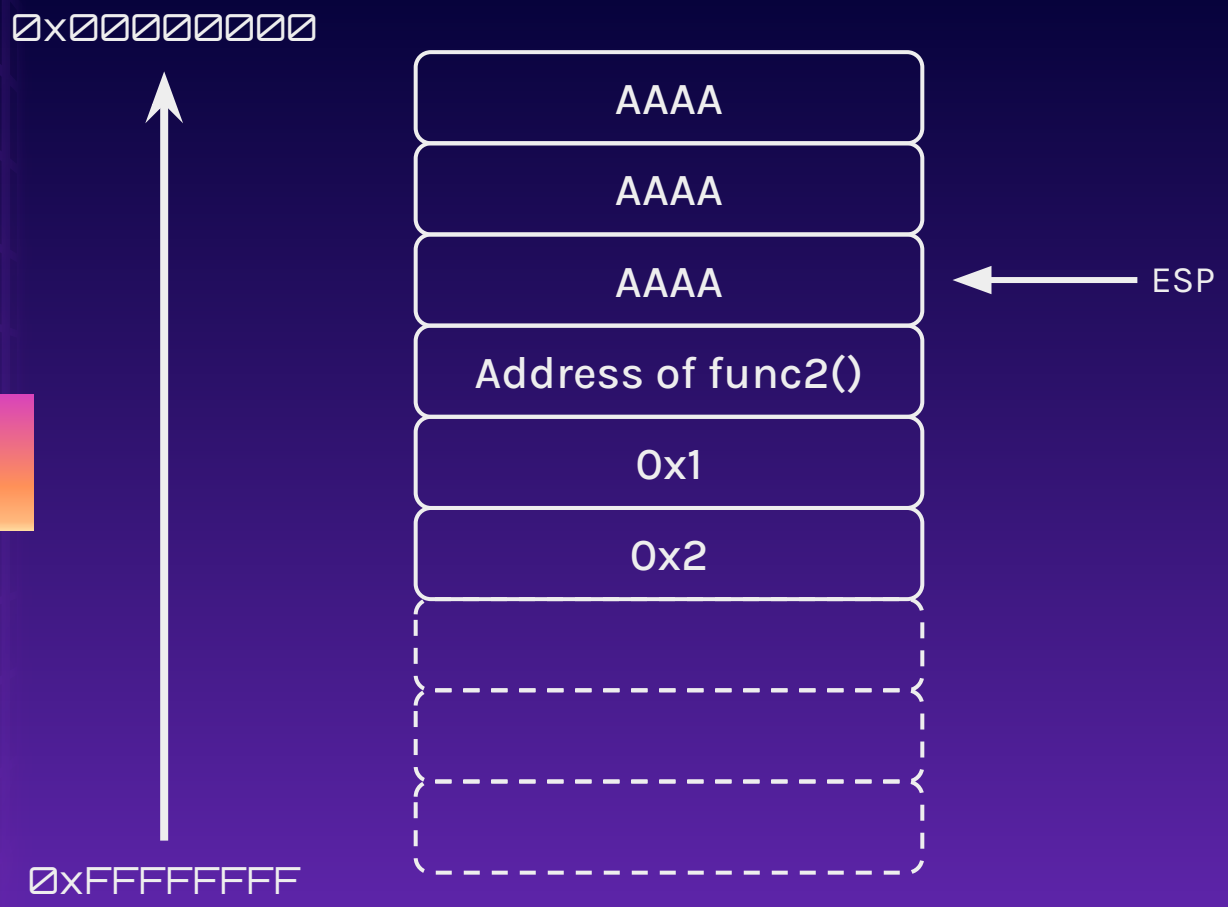
```
int func1(int arg1) {
    return arg1;
}

int func2(int arg2) {
    return arg2;
}

int main() {
    char buffer[256];
    gets(buffer);
}
```



Next instruction
ret

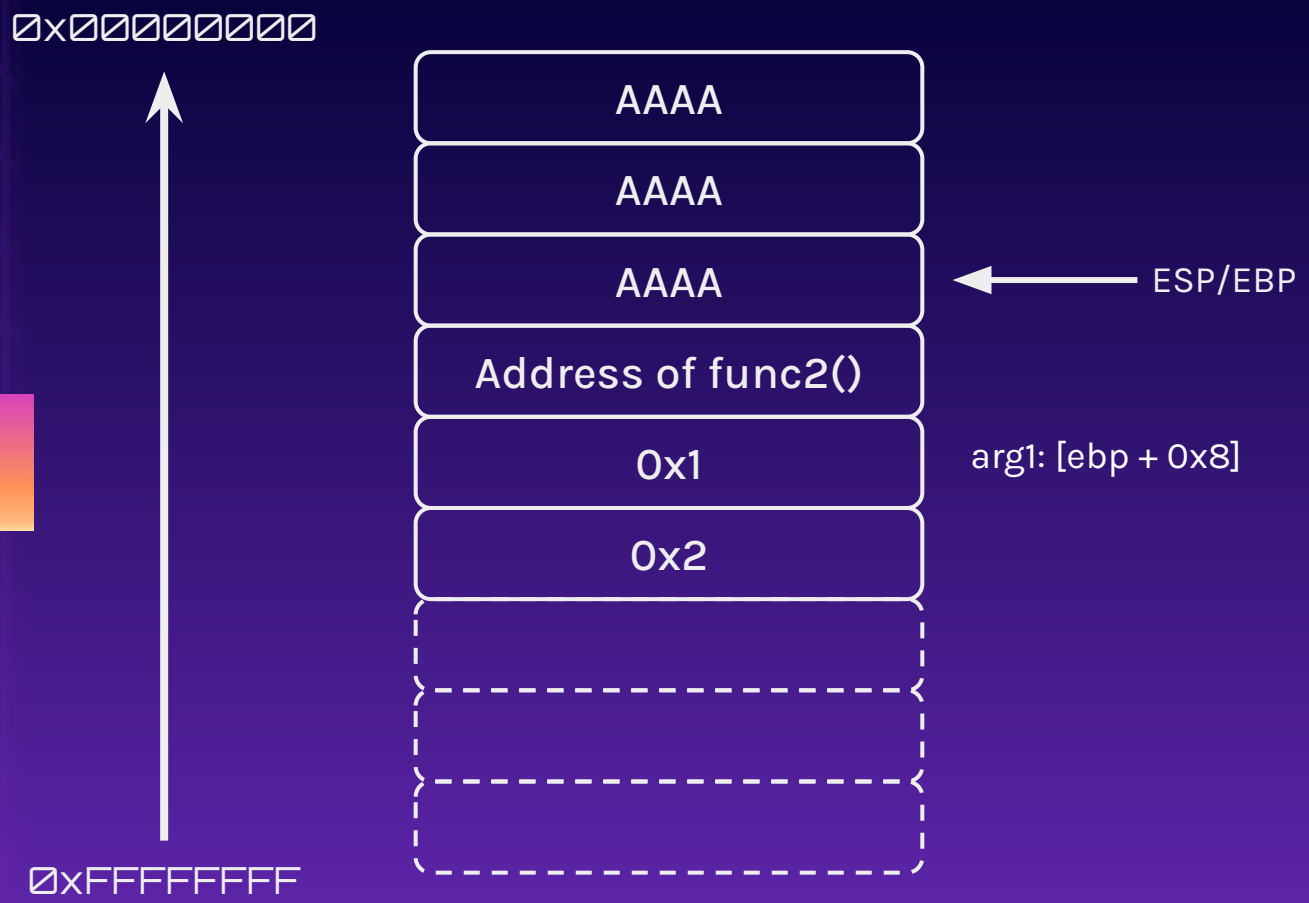


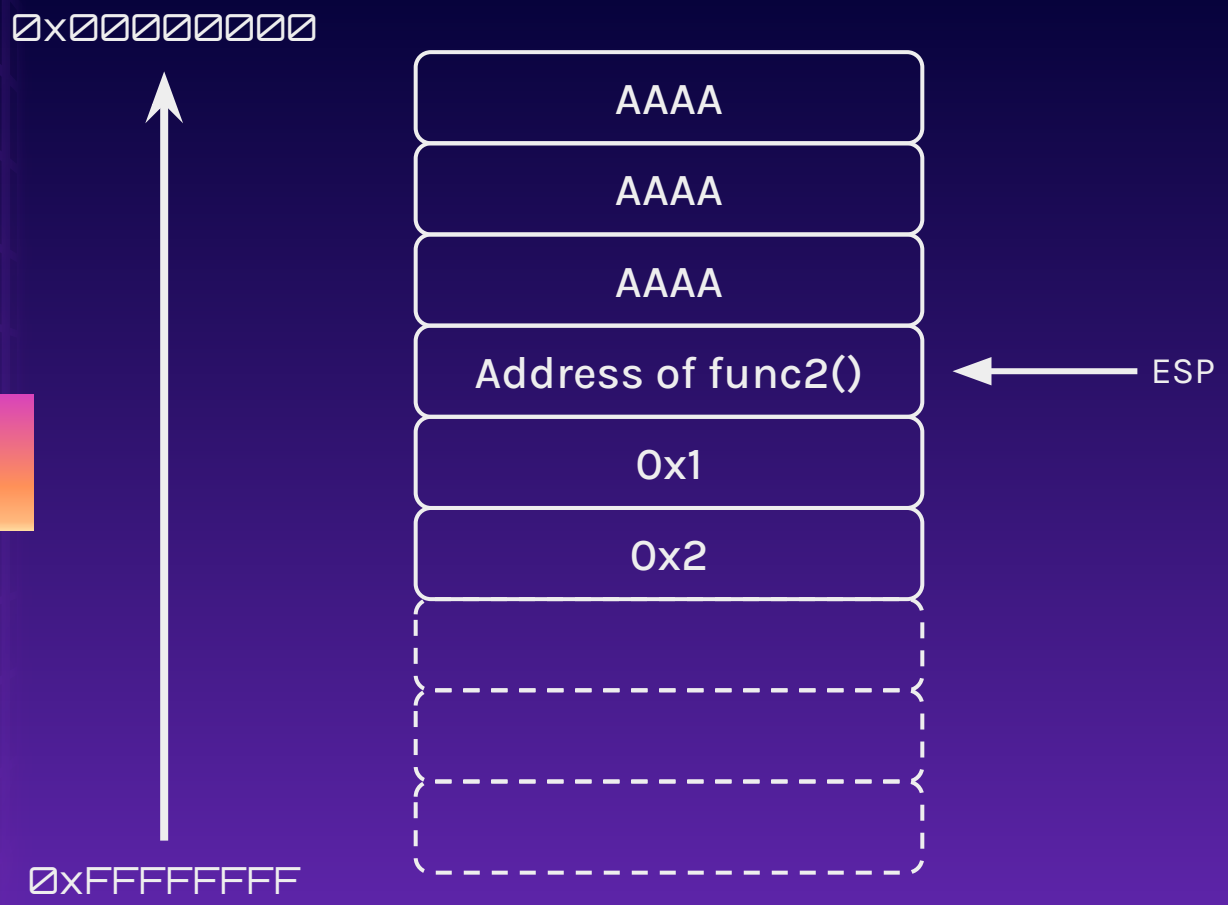
Next instruction
push ebp



Next instruction
mov ebp, esp

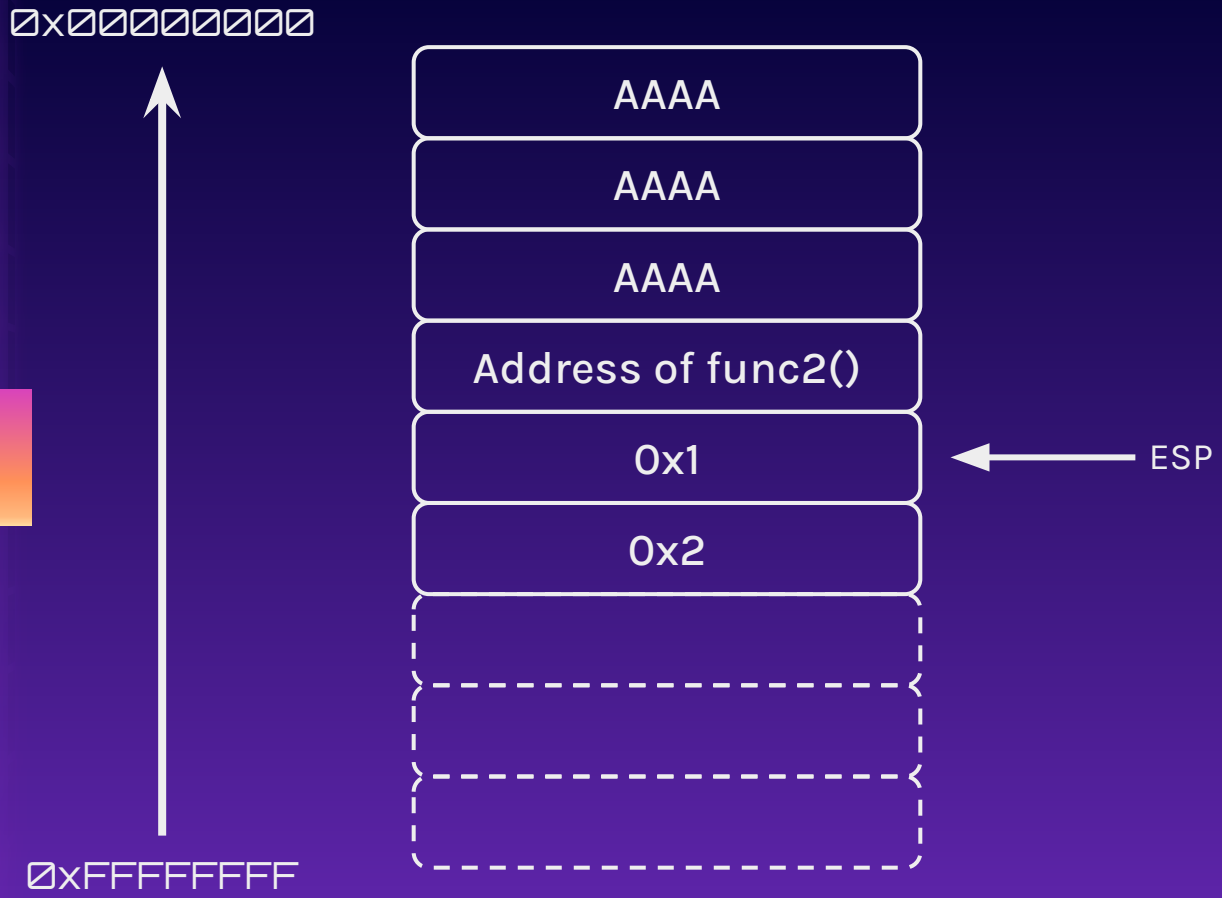
Next instruction
pop ebp



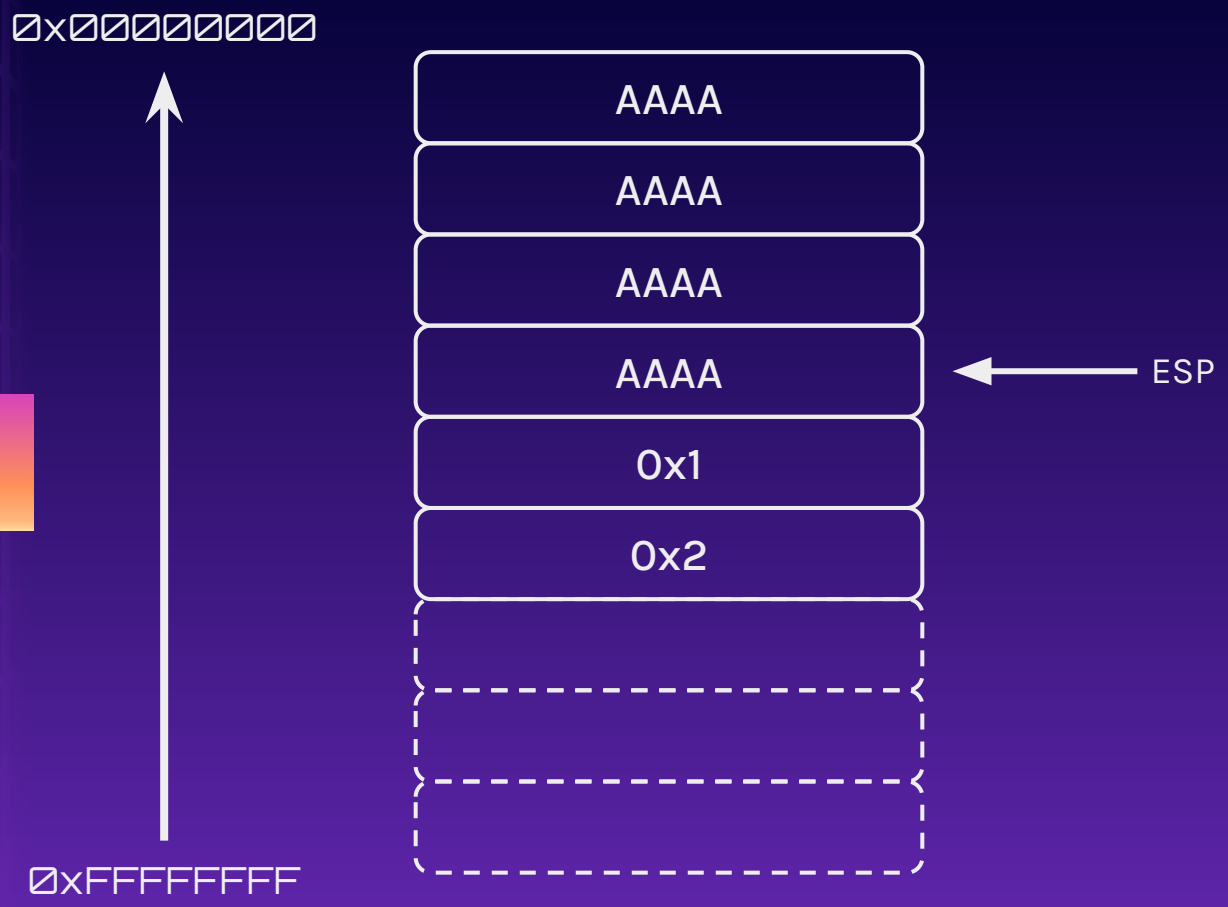


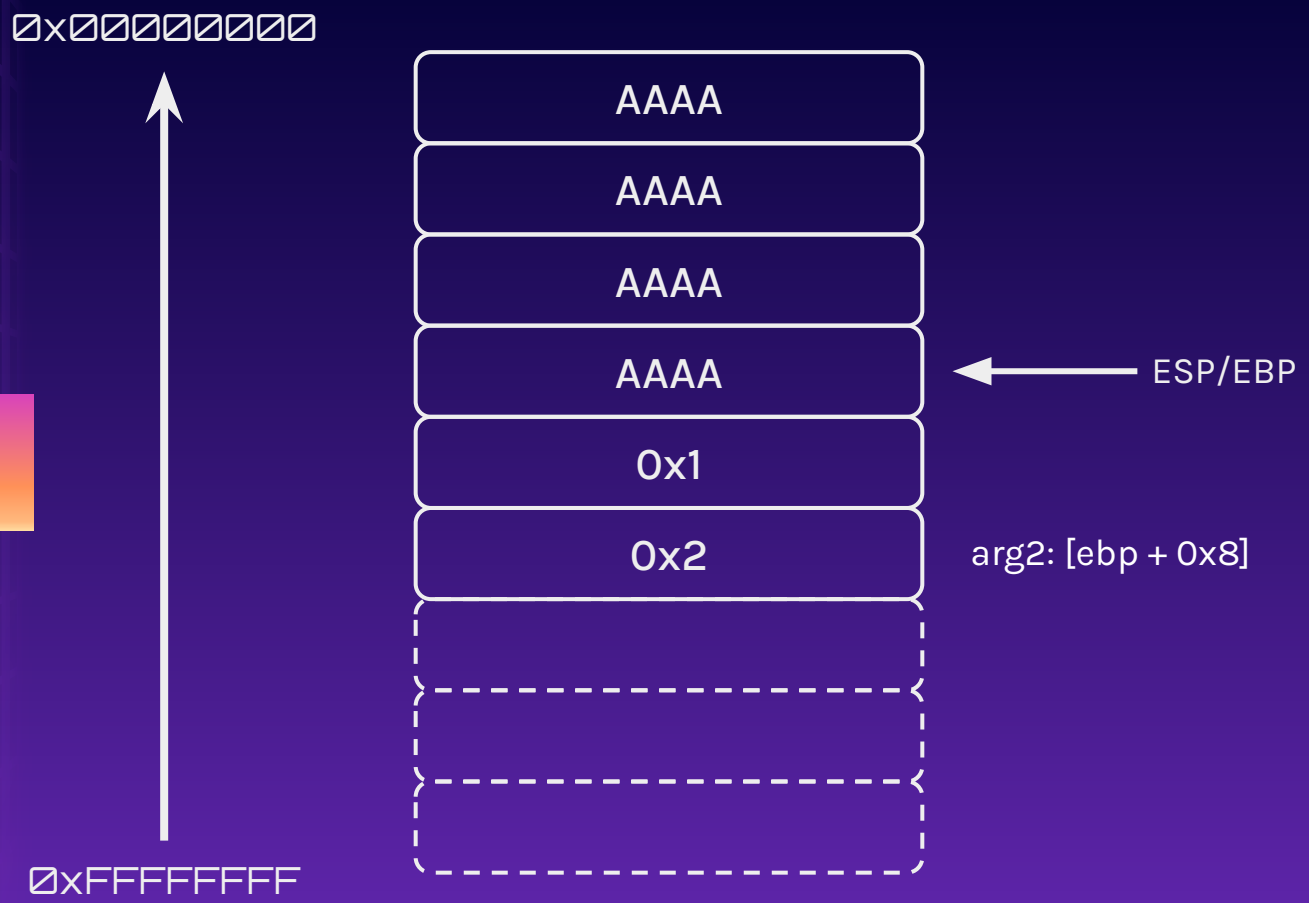
Next instruction
ret

Next instruction
push ebp



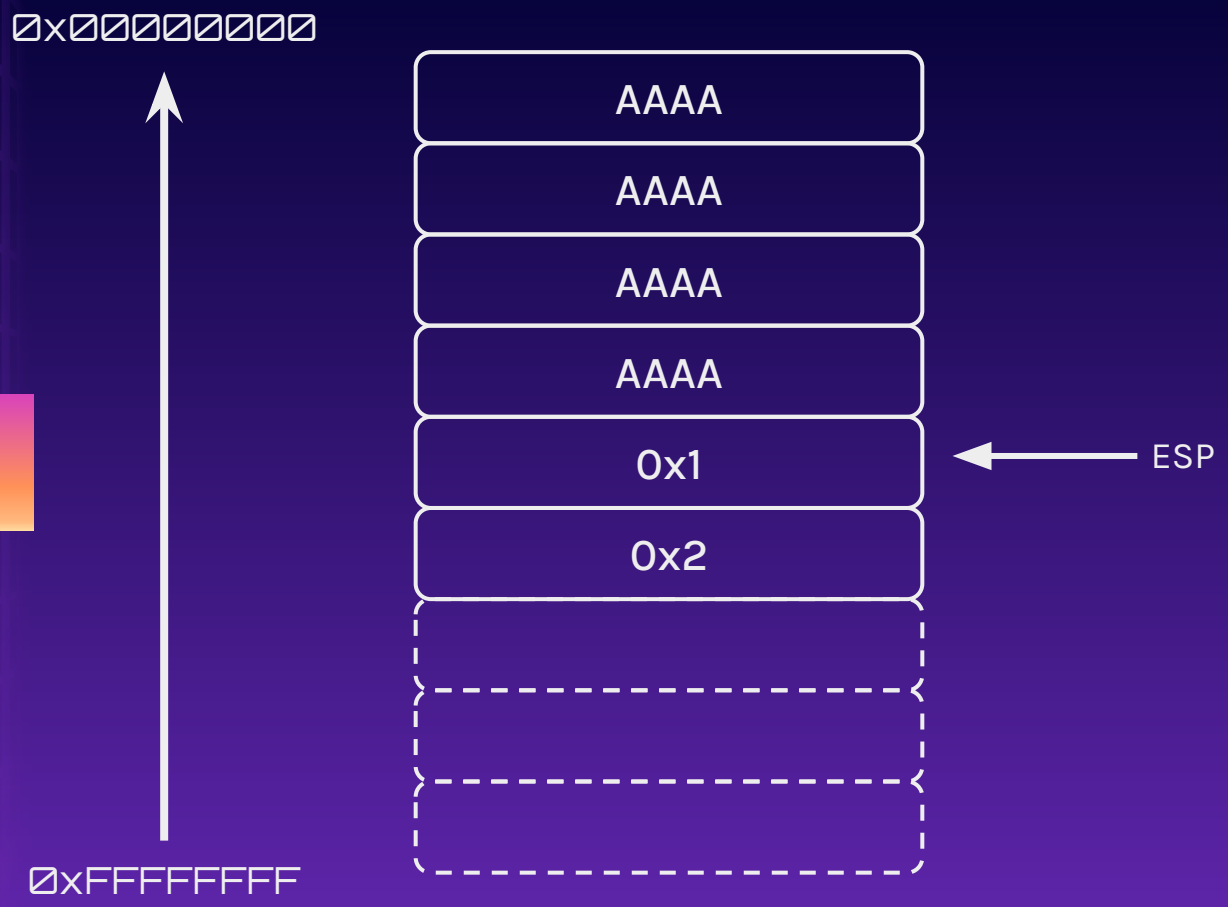
Next instruction
mov ebp, esp





Next instruction
pop ebp

Next instruction
ret



PROBLEM

Arguments taken
are [ebp + 0x8]

Max of 2 functions chained

PROBLEM

What if function needs more than one argument?

What if more than 2 chained functions are required?

ROP Gadgets

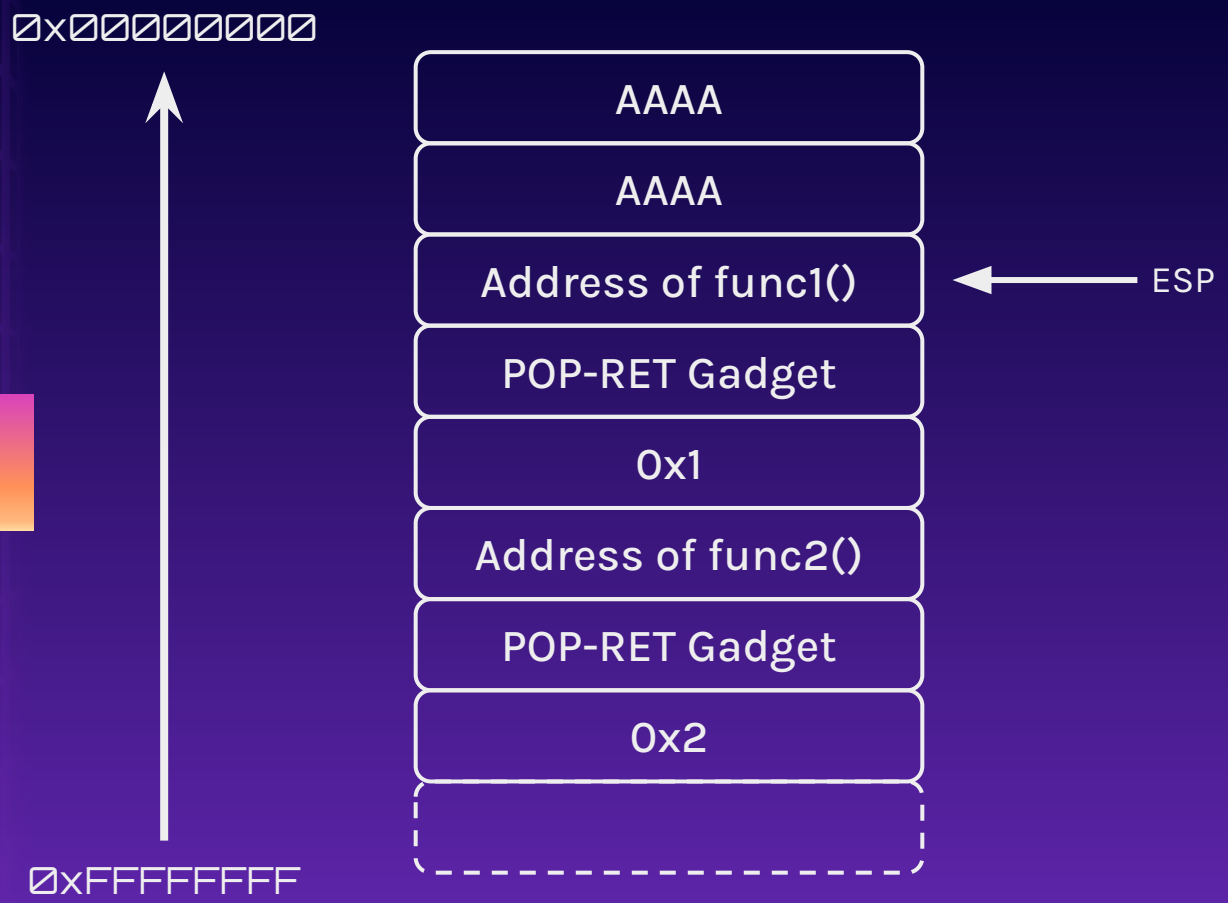
Machine instructions that are already present in the binary.

~Wikipedia-Kun

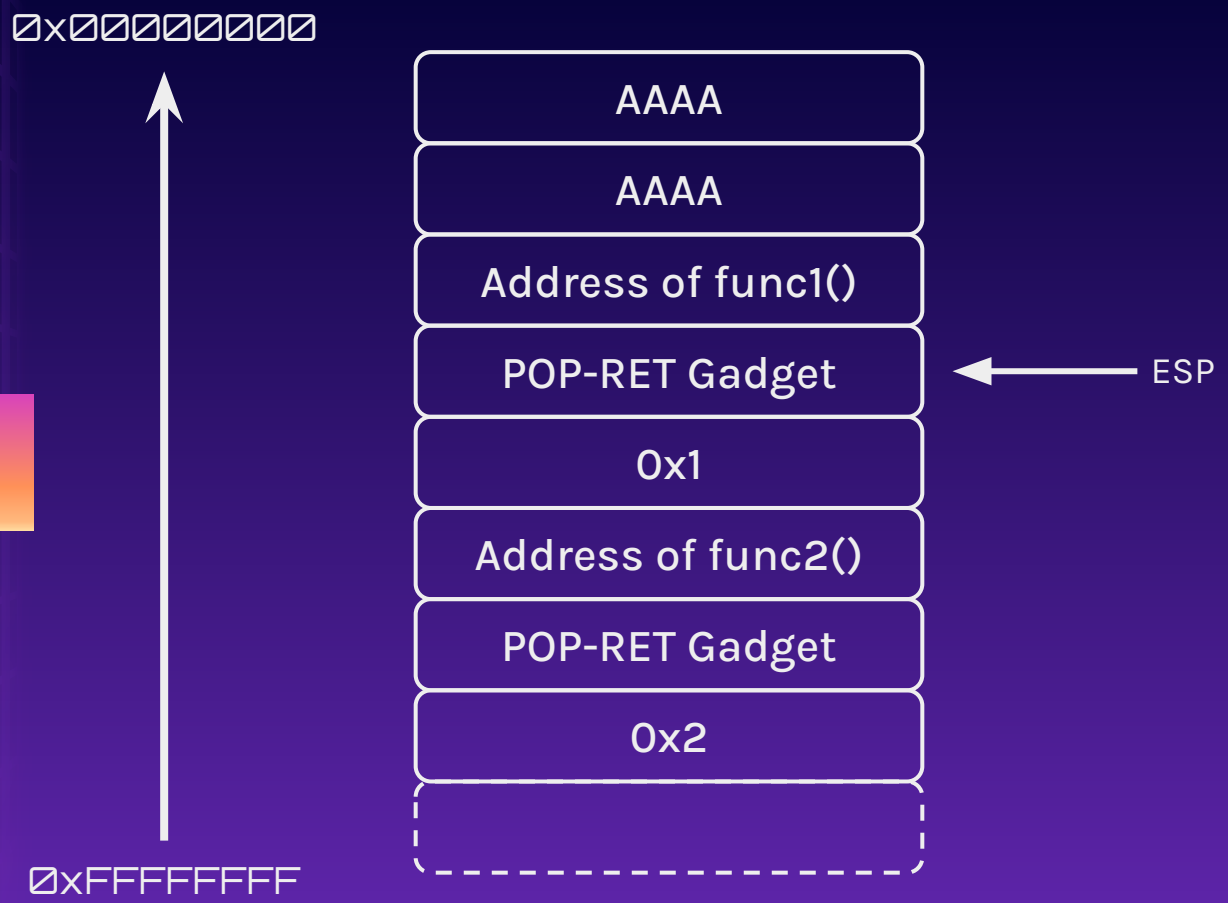
POP-RET Gadget

A pop instruction followed by a ret instruction

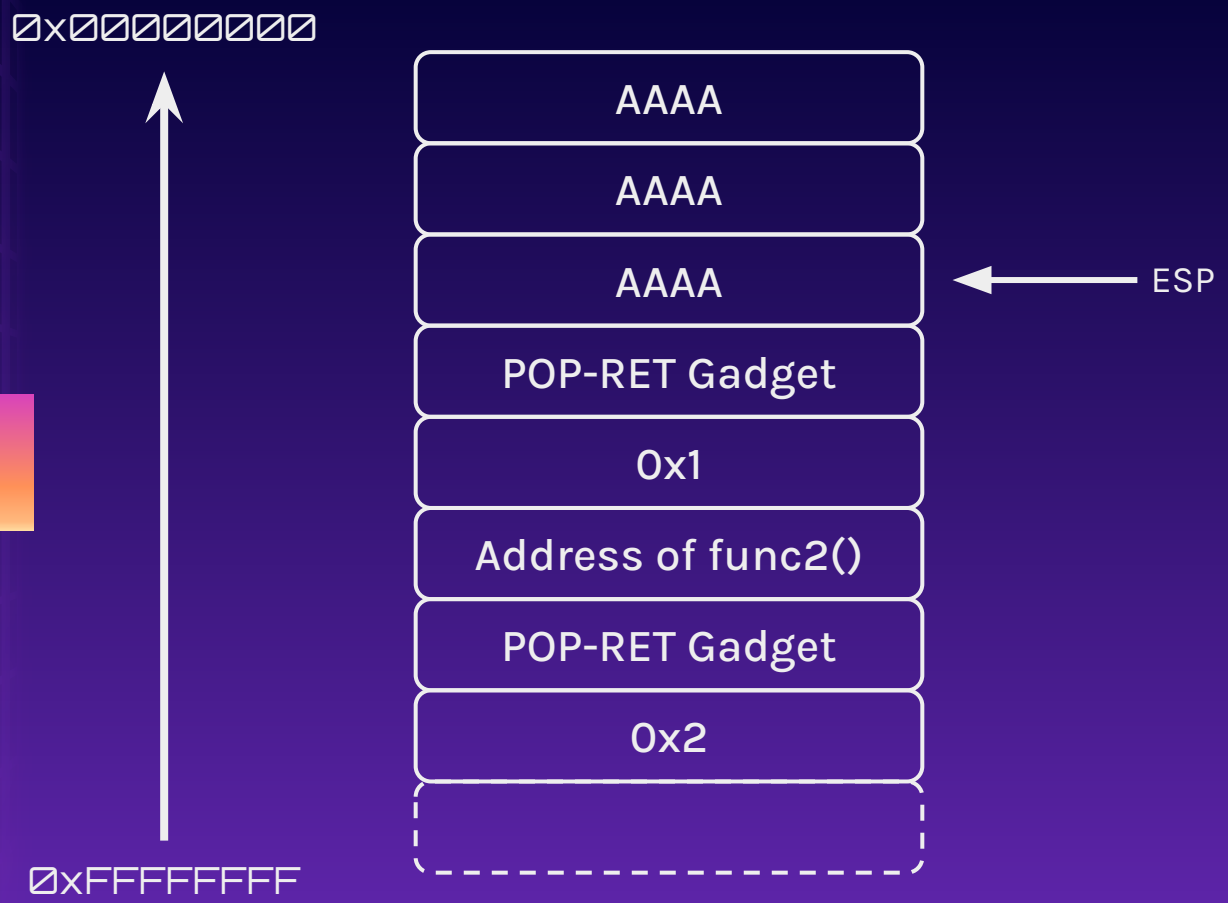
Place in between function address and arguments



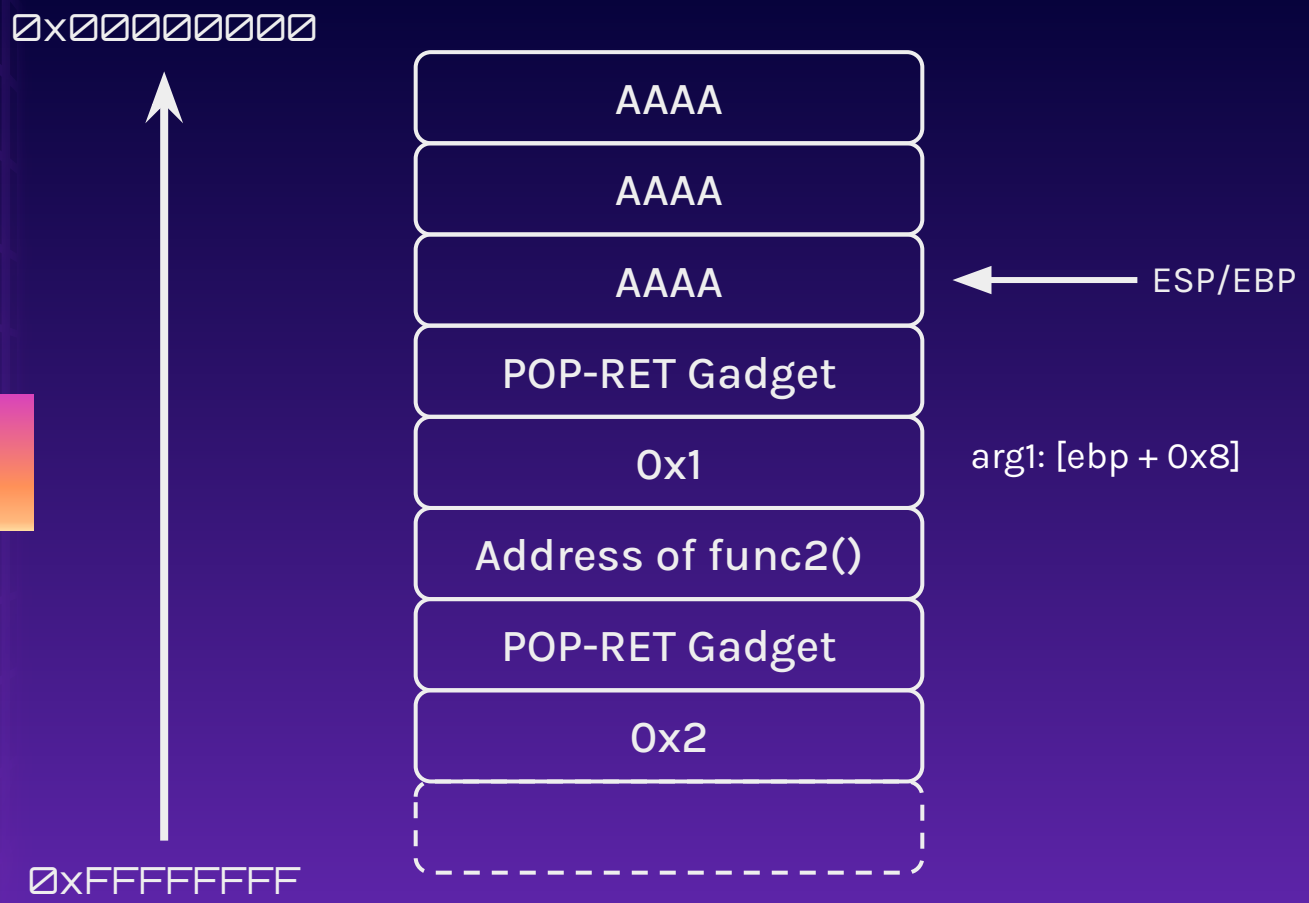
Next instruction
ret



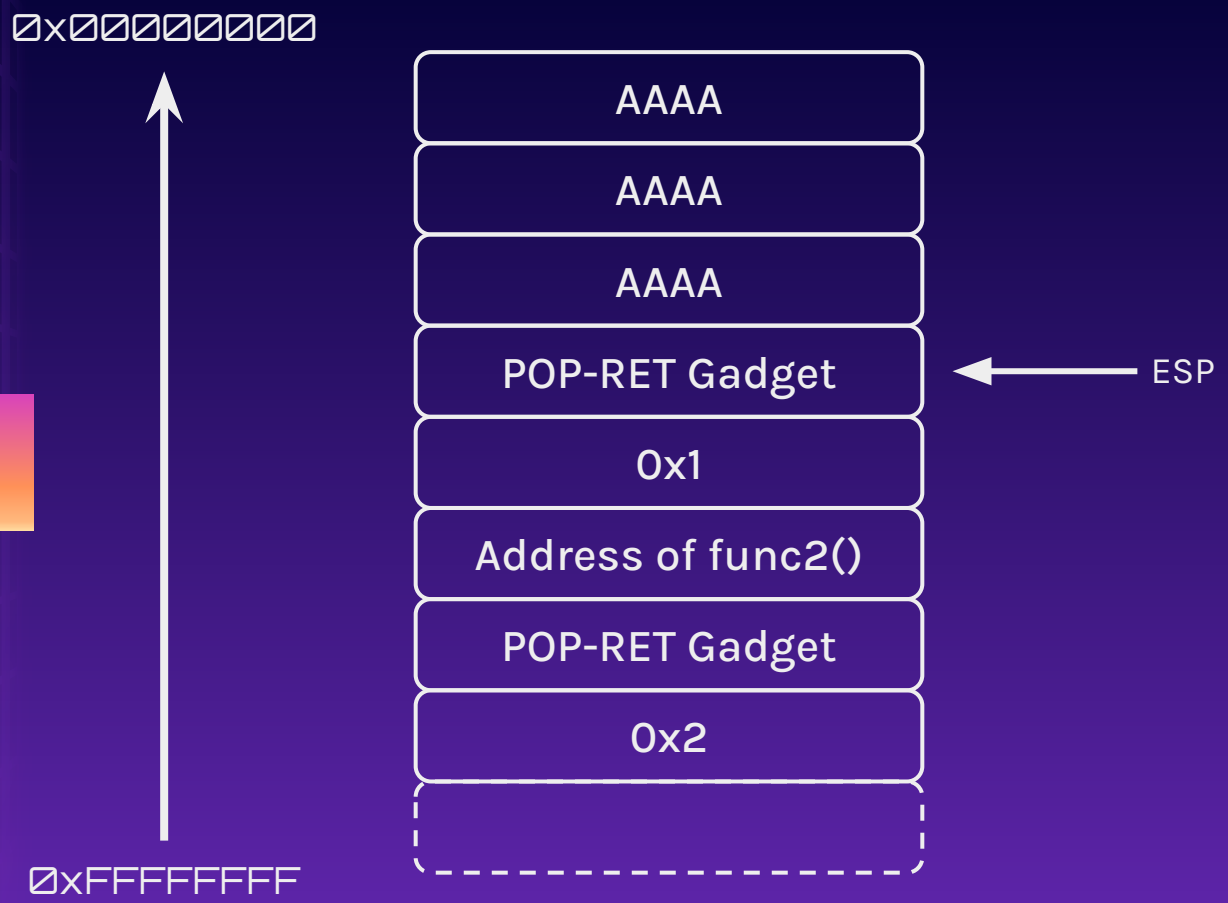
Next instruction
push ebp



Next instruction
mov ebp, esp

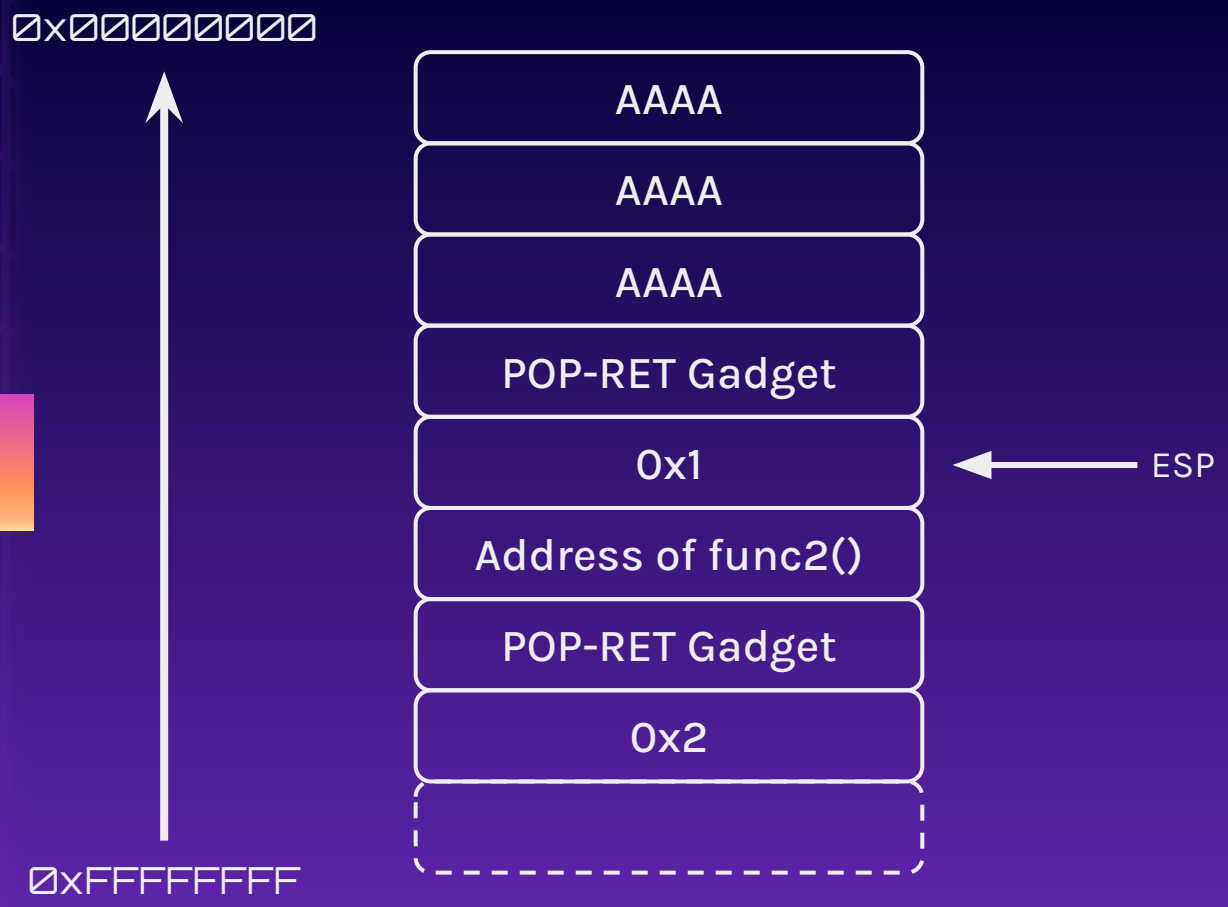


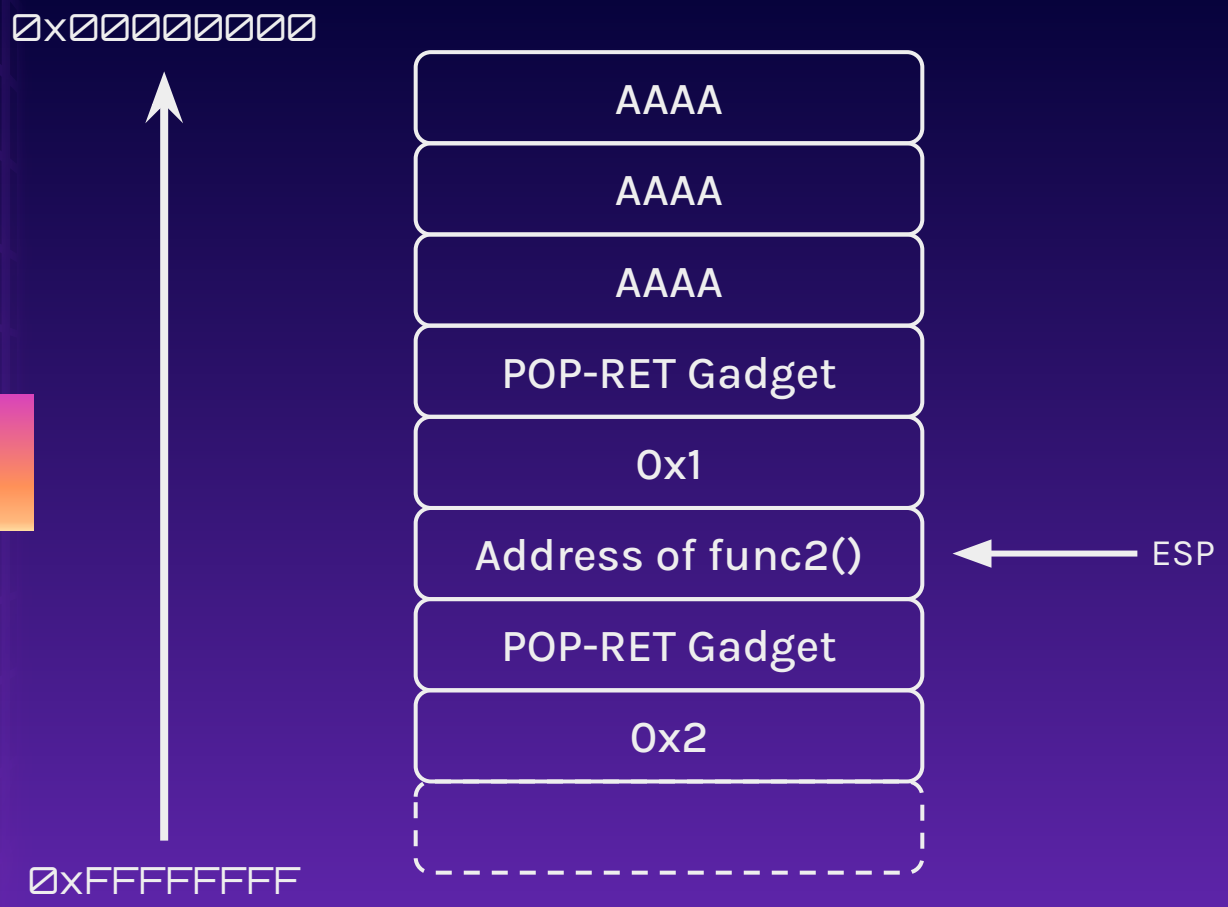
Next instruction
pop ebp



Next instruction
ret

Next instruction
pop e??





Next instruction
ret

WHAT HAPPENED

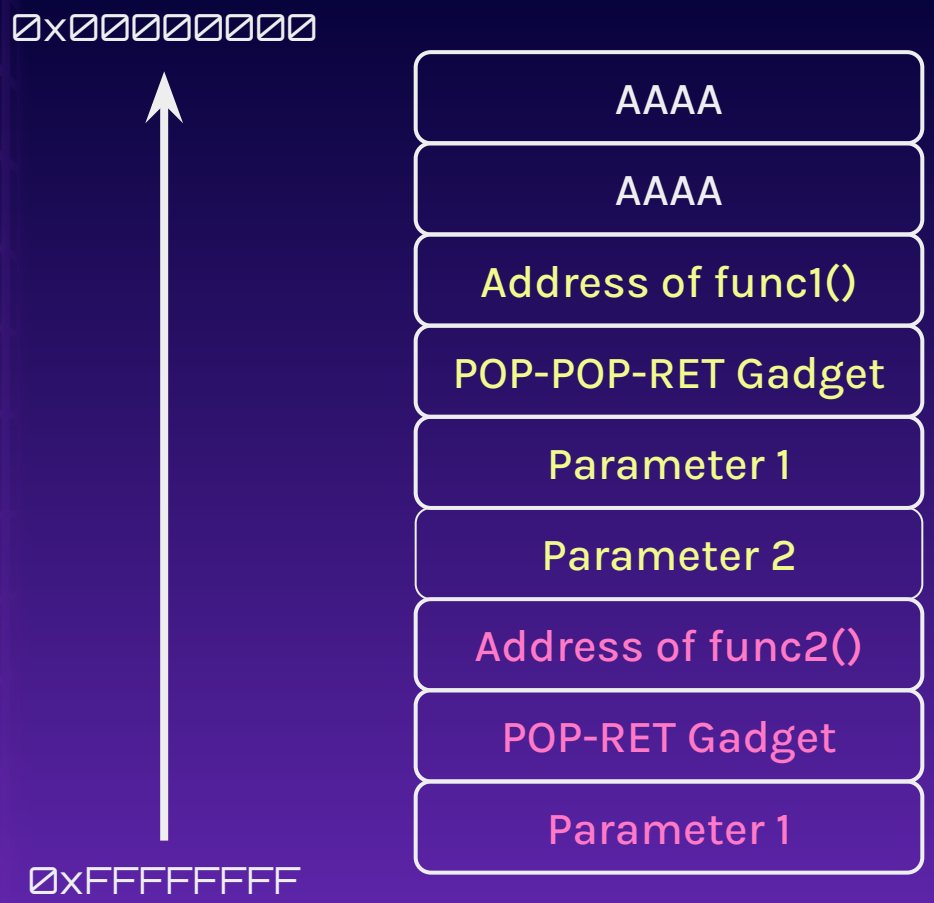
Notice how the stack layout is exactly how we started

Now we can chain as many functions as we like

HOW?

What if there are 2 or more arguments?

Just find a POP-POP-RET Gadget!


0x00000000
AAAA
AAAA
Address of func1()
POP-POP-RET Gadget
Parameter 1
Parameter 2
Address of func2()
POP-RET Gadget
Parameter 1
0xFFFFFFFF

HOW?

What about strings?

Recall that C strings are just character arrays

HOW?

Reuse strings already present in the binary

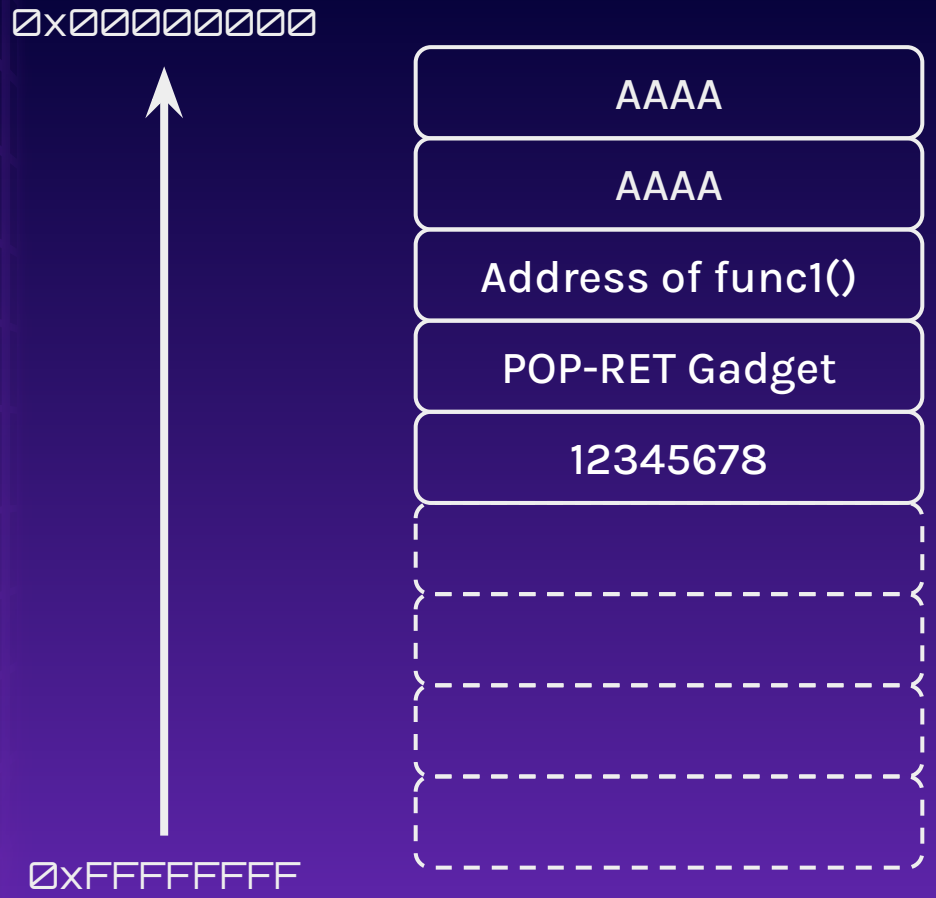
Pass the address of the string as the parameter

Finding Gadgets

```
pwndbg> ropper -- --search "pop e??;
ret;"
0x08048540: pop ebp; ret;
0x080483a5: pop ebx; ret;
```

Finding String

```
>>> from pwn import *
>>> elf = context.binary =
ELF("./binaryfile")

# Finding Address of String
>>> next(elf.search(b"somestring\x00"))
12345678
```


0x00000000
AAAA
AAAA
Address of func1()
POP-RET Gadget
12345678
0xFFFFFFFF

ROP Chain

Chaining a bunch of gadgets to chain multiple functions together

ret2func.c

25 mins to pwn ret2func32

Download files at:
http://ctfd.platypew.social

nc pwn.platypew.social 30004

```c
bool win1 = false;
bool win2 = false;

void func1(int arg1) {
    if (arg1 == 0xdeadbeef)
        win1 = true;
}

void func2(int arg2) {
    if (arg2 == 0xcafebabe)
        win2 = true;
}

void win(char* secret) {
    if (!(win1 && win2)) {
        return;
    }

    if (!strncmp(secret, "magicman", 8))
        system("/bin/sh");
}

void vuln() {
    char buffer[64];
    gets(buffer);
}
```