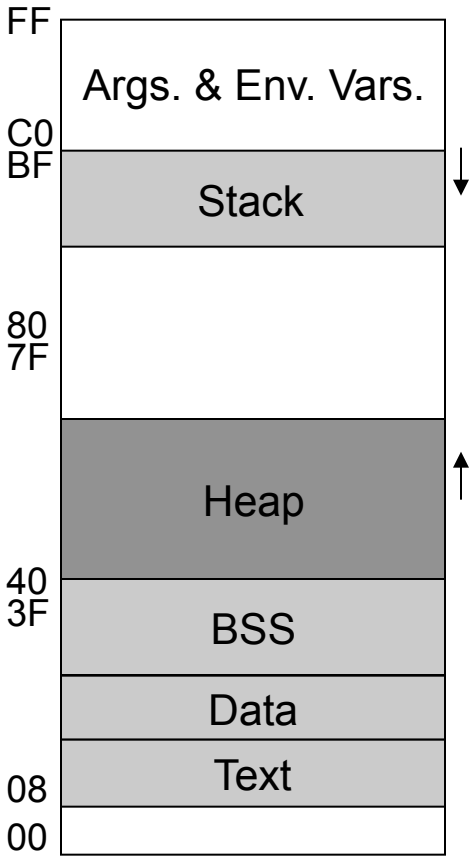
Buffer Overflows I

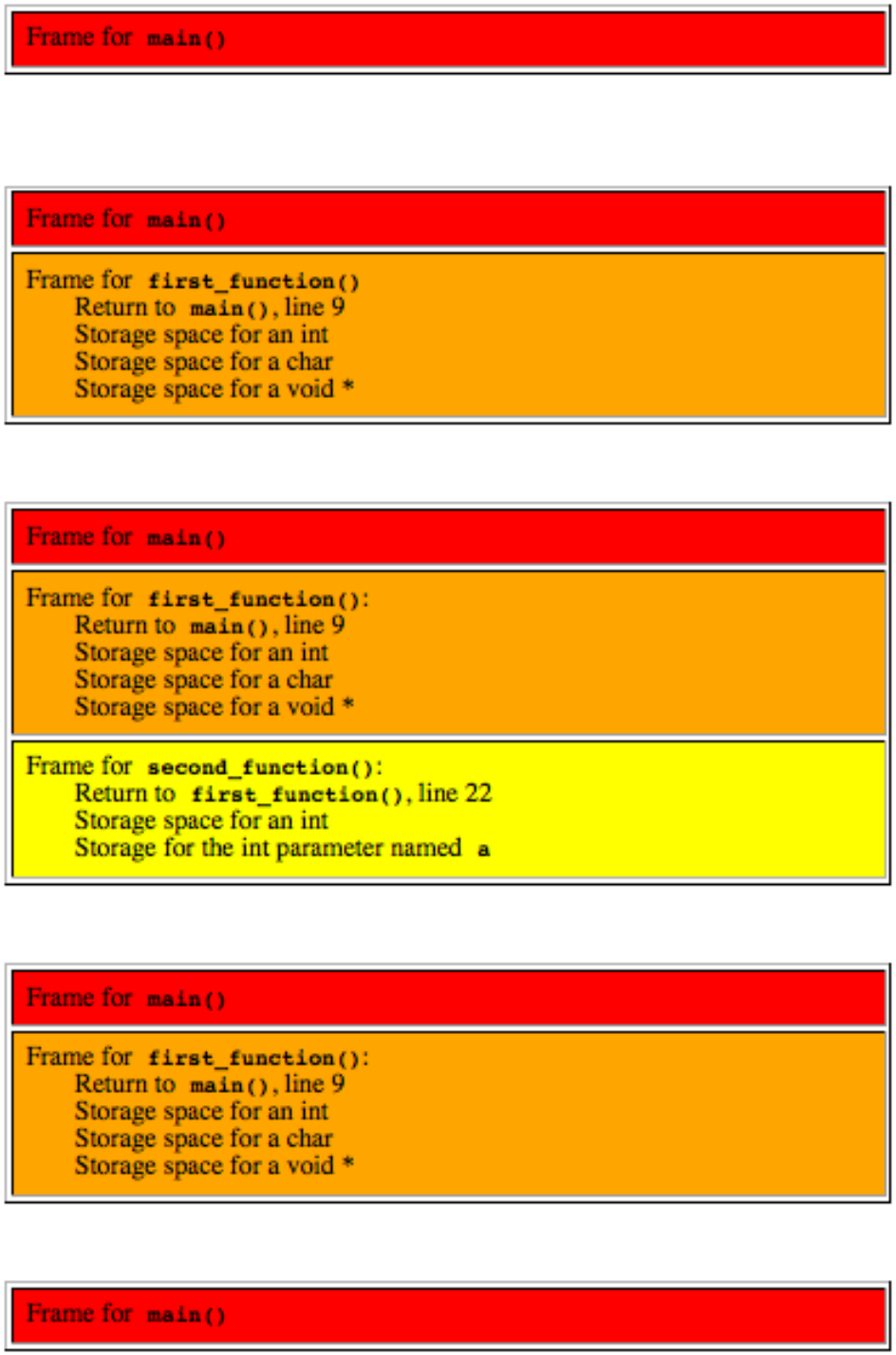
Smashing the Stack

Linux Memory Layout

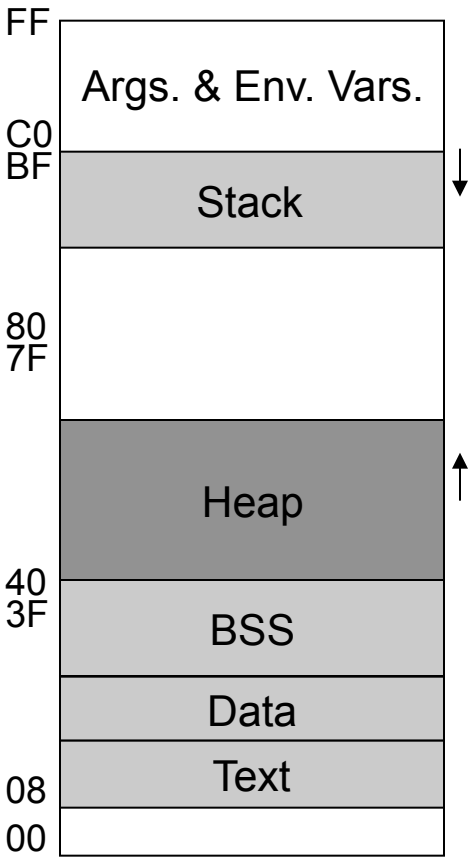


Stack Operation

- Frames
  - Function calls
  - Push and Pop



Stack Operation

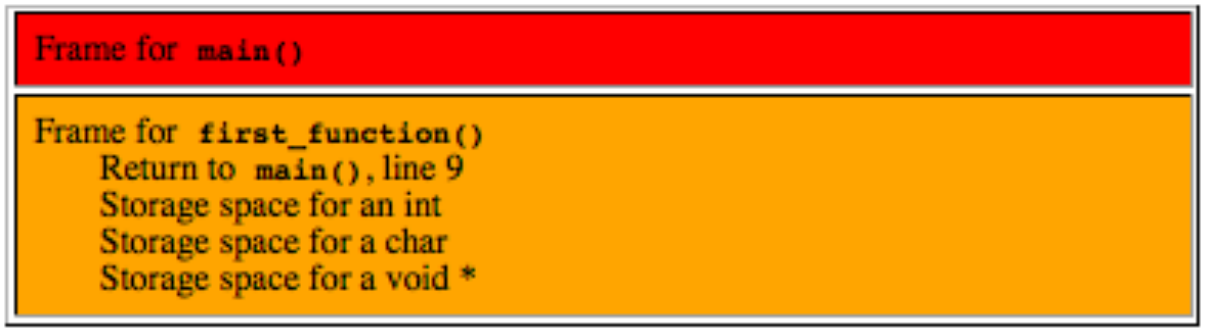


Registers:

ebp: Base Ptr.

eip: Instruction Ptr.

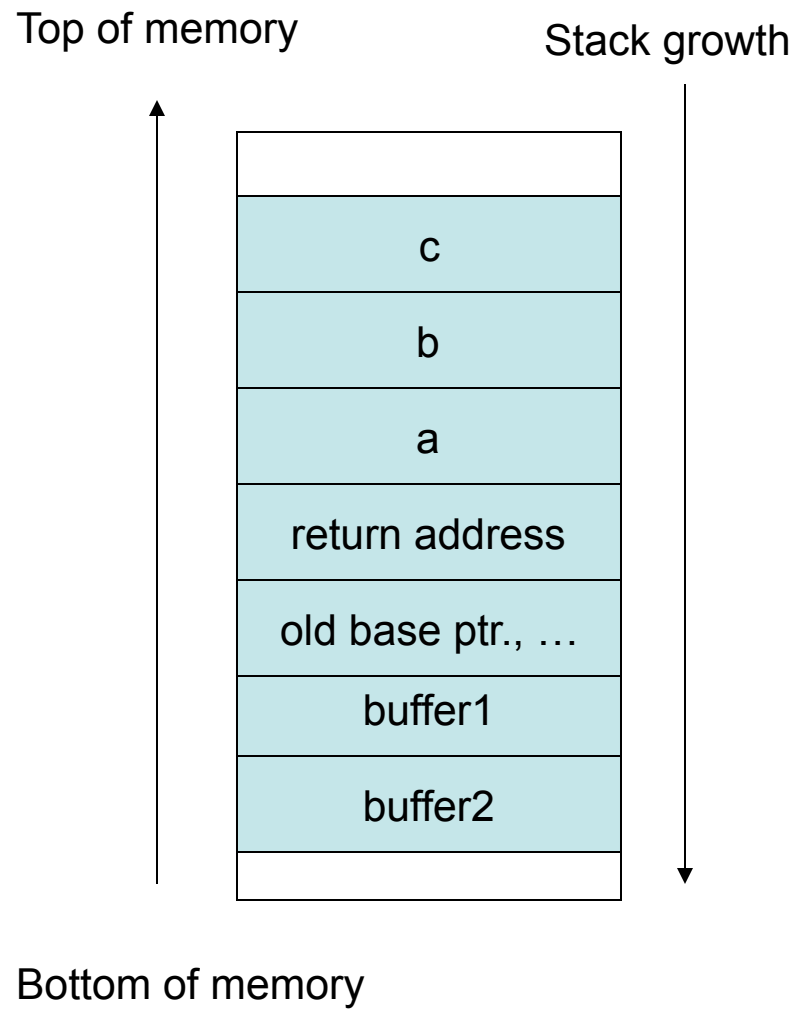
esp: Stack Ptr.



Stack Layout

```
void function (int a,
       int b, int c) {
    char buffer1[5];
    char buffer2[10];
}

int main() {
   function(1,2,3);
}
```



Registers:

ebp: Base Ptr.

eip: Instruction Ptr.

esp: Stack Ptr.

In Assembly

```
void function (int a,
       int b, int c) {
    int x,y,z;
    x = 5; z = 7;
}

int main() {
   function(1,2,3);
}
```

```
[main]
0x04 push 3
0x06 push 2
0x08 push 1
0x0a push eip + 2
0x0d jmp 0x2a
…
[function]
0x2a push ebp
0x2c mov ebp esp
0x2e sub esp 12
0x30 mov [ebp-4], 5
0x33 mov [ebp-12], 7
```

x, z

Registers:

ebp: Base Ptr.

eip: Instruction Ptr.

esp: Stack Ptr.

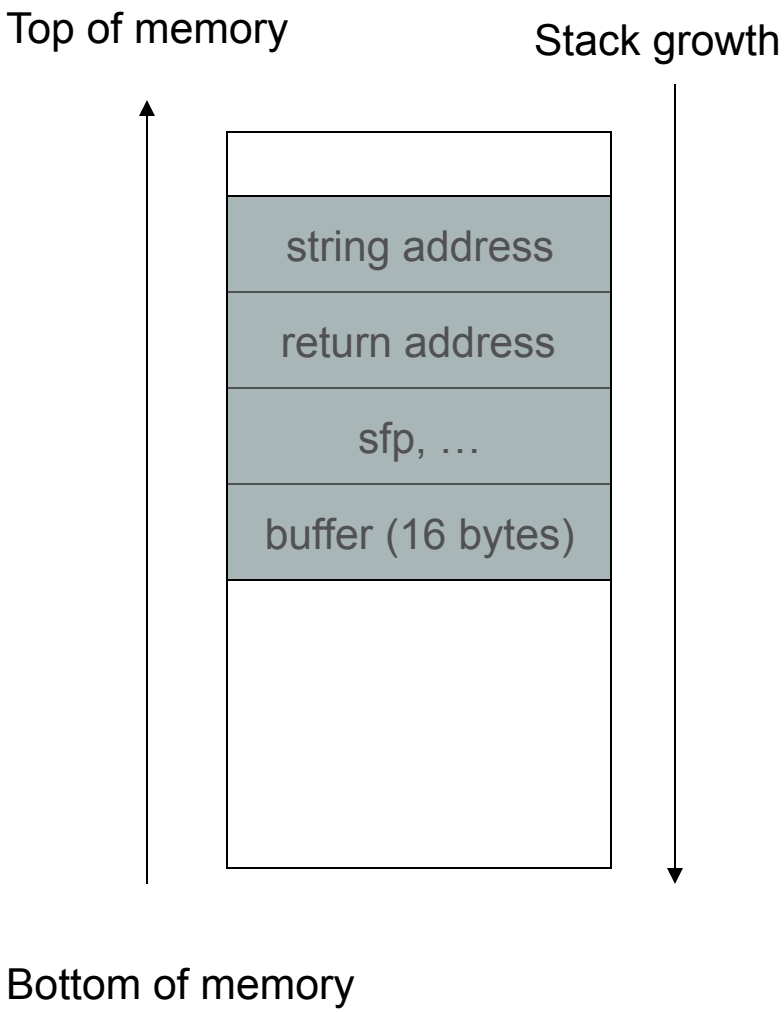
Buffer Overflow

- Buffer
  - A piece of _____
  - Capacity is _____
  - Size determined by _____
- TPS: too much?

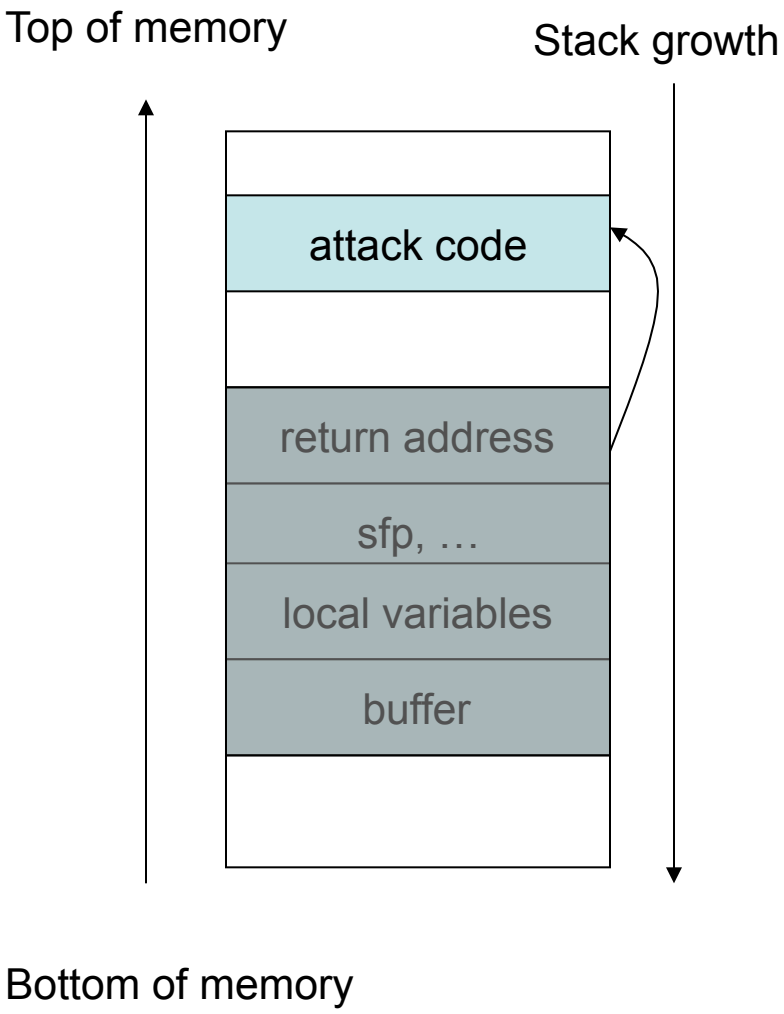
Buffer Overflow Problems

```
void function (char *str) {
    char buffer[16];
    strcpy(buffer,str);
}

int main() {
   char *str="I am greater
   than 16 bytes, and I am
   going to overflow your
   buffer";
   function(str);
}
```



Buffer Overflow Attacks

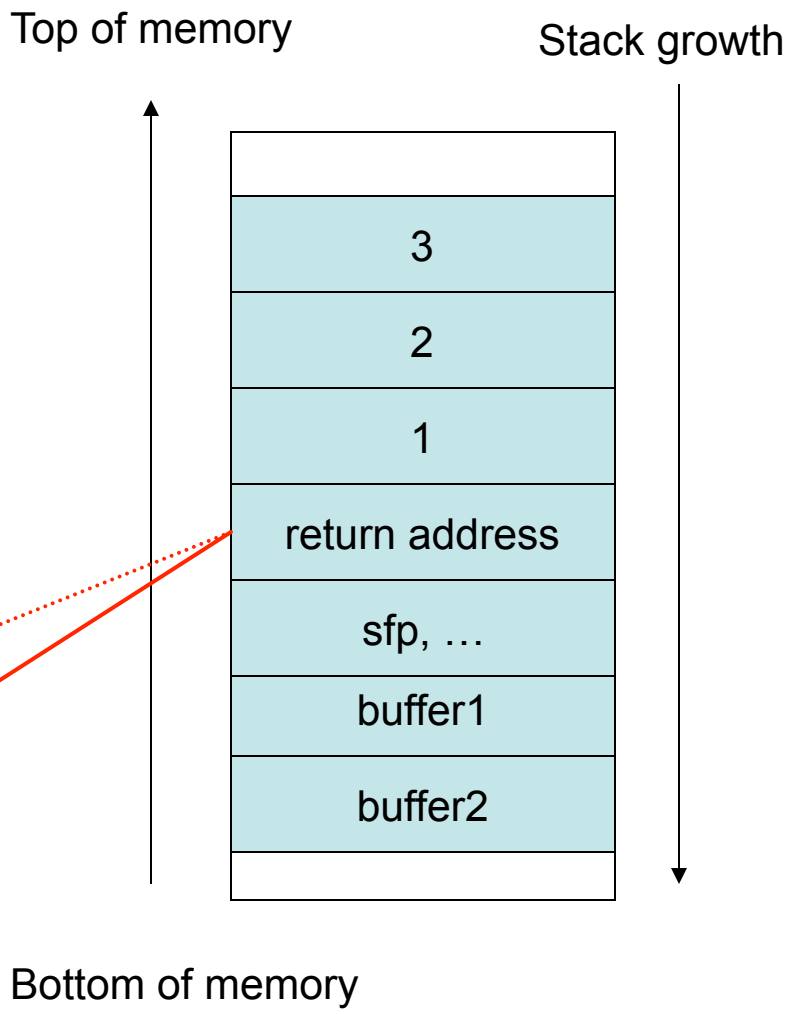


Launching an Attack

- Understand the target system
- Injecting the attack code
- Change the flow of execution
  - Overwrite the return address
  - Run the attack code

Overwrite the Return Address

```
void function (int a, int b,
   int c) {
     char buffer1[5];
     char buffer2[10];
     int *ret, i;
     ret=buffer1+[FOO?];
     (*ret)+=[BAR?];
}
int main() {
     int x;
     x=0;
     function(1,2,3);
     x=1;
     printf("%d\n",x)
}
```



```
void function (int a, int b, int c) {
    char buffer1[5];
    char buffer2[10];
    int *ret, i;
    ret=buffer1+1;
    (*ret)+=2;
}
```

Notes:
This was done on an Intel-based Mac.

Negatives would normally be really big (0xffffffff = -1).

```
(gdb) disassemble function
Dump of assembler code for function function:
0x00001f3a <function+0>:    push   %ebp
0x00001f3b <function+1>:    mov    %esp,%ebp
0x00001f3d <function+3>:    sub    $0x28,%esp
0x00001f40 <function+6>:    lea    -17(%ebp),%eax
0x00001f43 <function+9>:    add    $0x1,%eax
0x00001f46 <function+12>:   mov    %eax,-12(%ebp)
0x00001f49 <function+15>:   mov    -12(%ebp),%eax
0x00001f4c <function+18>:   mov    (%eax),%eax
0x00001f4e <function+20>:   lea    2(%eax),%edx
0x00001f51 <function+23>:   mov    -12(%ebp),%eax
0x00001f54 <function+26>:   mov    %edx,(%eax)
0x00001f56 <function+28>:   leave
0x00001f57 <function+29>:   ret
End of assembler dump.
```

Start new frame

Allocate space for local vars

Get addr for buf1

Add 1

Save to ret

Get ret's val

Load (2 + ret's val)

Save back to ret's val

Figuring Out Stack Layout

```
void function (int a, int b, int c) {
    char buffer1[5];
    char buffer2[10];
    int *ret, i;

    for(i=0;i<5;i++) buffer1[i]=i;
    for(i=0;i<10;i++) buffer2[i]=i;

    ret=(int *)buffer2;

    for(i=0;i<20;i++) printf("%X:%X\n",&
   (ret[i]),ret[i]);
}
int main() {
    function(1,2,3);
}
```

Unsigned hex

```
[dliu@omega stack]$ ./a.out
BFFFE560:3020100
BFFFE564:7060504
BFFFE568:B75F0908
BFFFE56C:0
BFFFE570:3020100
BFFFE574:8049504
BFFFE578:BFFFE588
BFFFE57C:8048265
BFFFE580:0
BFFFE584:0
BFFFE588:BFFFE5A8
BFFFE58C:80483F6
BFFFE590:1
BFFFE594:2
BFFFE598:3
BFFFE59C:859D78
BFFFE5A0:67E020
BFFFE5A4:80483FC
BFFFE5A8:BFFFE608
BFFFE5AC:73979A
```

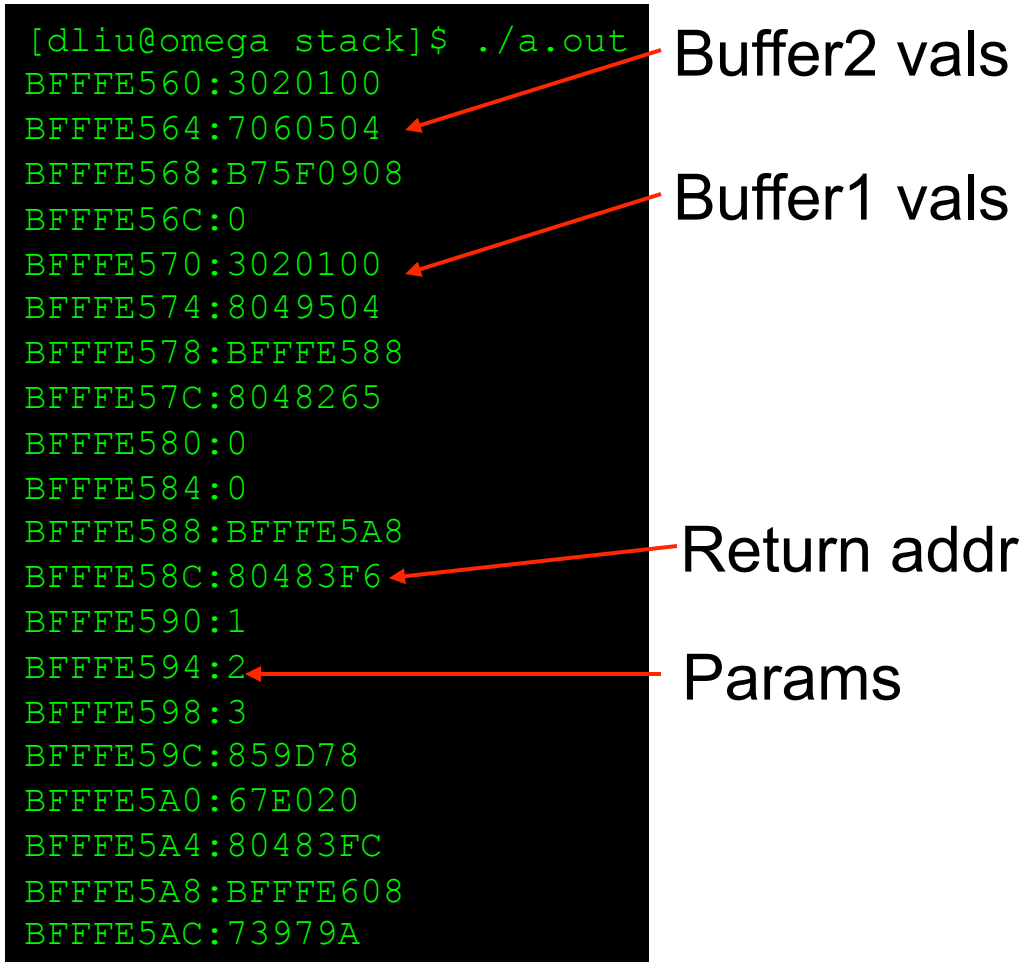
Figuring Out Stack Layout

```
void function (int a, int b, int c)
    {
     char buffer1[5];
     char buffer2[10];
     int *ret, i;

     for(i=0;i<5;i++) buffer1[i]=i;
     for(i=0;i<10;i++) buffer2[i]=i;

     ret=(int *)buffer2;

     for(i=0;i<20;i++) printf("%X:%X
    \n",&(ret[i]),ret[i]);
}
int main() {
     function(1,2,3);
}
```

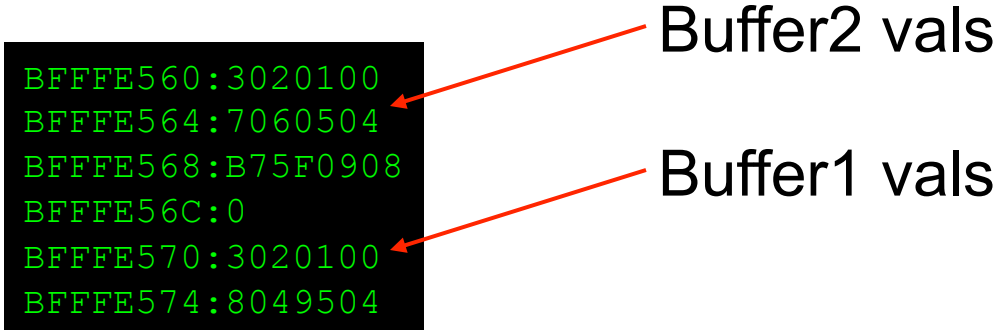


Finding the Return Address

- Pushed after the params

```
BFFFE58C:80483F6
BFFFE590:1
BFFFE594:2
BFFFE598:3
```

- Find the start of `buffer1`



Finding BAR

```
(gdb) disassemble main
Dump of assembler code for function main:
0x080483d8 <main+0>:    push   %ebp
0x080483d9 <main+1>:    mov    %esp,%ebp
0x080483db <main+3>:    sub    $0x8,%esp
0x080483de <main+6>:    and    $0xfffffff0,%esp
0x080483e1 <main+9>:    mov    $0x0,%eax
0x080483e6 <main+14>:   sub    %eax,%esp
0x080483e8 <main+16>:   movl   $0x0,0xfffffffc(%ebp)
0x080483ef <main+23>:   sub    $0x4,%esp
0x080483f2 <main+26>:   push   $0x3
0x080483f4 <main+28>:   push   $0x2
0x080483f6 <main+30>:   push   $0x1
0x080483f8 <main+32>:   call   0x8048344 <function>
0x080483fd <main+37>:   add    $0x10,%esp
0x08048400 <main+40>:   movl   $0x1,0xfffffffc(%ebp)
0x08048407 <main+47>:   sub    $0x8,%esp
0x0804840a <main+50>:   pushl  0xfffffffc(%ebp)
0x0804840d <main+53>:   push   $0x80484f7
0x08048412 <main+58>:   call   0x8048288
0x08048417 <main+63>:   add    $0x10,%esp
0x0804841a <main+66>:   leave
0x0804841b <main+67>:   ret
End of assembler dump.
```

Push params onto stack

Call function (pushes IP onto stack)

The return address

Skipping the Instruction

- Find the return address

```
0x080483f8 <main+32>:    call   0x8048344 <function>
0x080483fd <main+37>:    add    $0x10,%esp
```

- Skip over the assignment

```
0x080483f8 <main+32>:    call   0x8048344 <function>
0x080483fd <main+37>:    add    $0x10,%esp
0x08048400 <main+40>:    movl   $0x1,0xfffffffc(%ebp)
0x08048407 <main+47>:    sub    $0x8,%esp
```

Result

```
void function (int a, int b, int c) {
    char buffer1[5];
    char buffer2[10];
    int *ret;
    ret=buffer1+28;
    (*ret)+=10;
}
int main() {
    int x;
    x=0;
    function(1,2,3);
    x=1;
    printf("%d\n",x)
}
```

```
$ ./a.out
0
```

Exercise

- Dr. Wright's Mac
- Get the two variables
  - FOO and BAR

FOO = ?

Hint: My Mac's text addresses start with x0000….

```
void function (int a, int b, int c)
    {
     char buffer1[6];
     char buffer2[10];
     int *ret, i;

     for(i=0;i<5;i++) buffer1[i]=i;
     for(i=0;i<10;i++) buffer2[i]=i;

     ret=(int *)buffer2;

     for(i=0;i<20;i++) printf("%X:%X
    \n",&(ret[i]),ret[i]);
}
int main() {
     function(1,2,3);
}
```

```
[19:15:25] mkw@blacktide:~% ./buf2
BFFFF7F8:3020100
BFFFF7FC:7060504
BFFFF800:1000908
BFFFF804:40302
BFFFF808:BFFFF7F8
BFFFF80C:5
BFFFF810:0
BFFFF814:BFFFF8E4
BFFFF818:BFFFF838
BFFFF81C:1FF7
BFFFF820:1
BFFFF824:2
BFFFF828:3
BFFFF82C:1000
BFFFF830:0
BFFFF834:0
BFFFF838:BFFFF85C
BFFFF83C:1F16
BFFFF840:1
BFFFF844:BFFFF864
```

BAR = ?

```
(gdb) (gdb) disassemble main
Dump of assembler code for function main:
0x00001f58 <main+0>:    push   %ebp
0x00001f59 <main+1>:    mov    %esp,%ebp
0x00001f5b <main+3>:    push   %ebx
0x00001f5c <main+4>:    sub    $0x24,%esp
0x00001f5f <main+7>:    call   0x1ffc <__i686.get_pc_thunk.bx>
0x00001f64 <main+12>:   movl   $0x0,-12(%ebp)
0x00001f6b <main+19>:   movl   $0x3,8(%esp)
0x00001f73 <main+27>:   movl   $0x2,4(%esp)
0x00001f7b <main+35>:   movl   $0x1,(%esp)
0x00001f82 <main+42>:   call   0x1f3a <function>
0x00001f87 <main+47>:   movl   $0x1,-12(%ebp)
0x00001f8e <main+54>:   mov    -12(%ebp),%eax
0x00001f91 <main+57>:   mov    %eax,4(%esp)
0x00001f95 <main+61>:   lea    148(%ebx),%eax
0x00001f9b <main+67>:   mov    %eax,(%esp)
0x00001f9e <main+70>:   call   0x301b <dyld_stub_printf>
0x00001fa3 <main+75>:   add    $0x24,%esp
0x00001fa6 <main+78>:   pop    %ebx
0x00001fa7 <main+79>:   pop    %ebp
0x00001fa8 <main+80>:   ret
End of assembler dump.
```

Result

```
void function (int a, int b, int c) {
    char buffer1[5];
    char buffer2[10];
    int *ret, i;
    ret=buffer1+26;
    (*ret)+=7;
}
int main() {
    int x;
    x=0;
    function(1,2,3);
    x=1;
    printf("%d\n",x)
}
```

```
$ ./a.out
0
```