CSCI 476: Computer Security

Buffer Overflow Attack (Part 1)

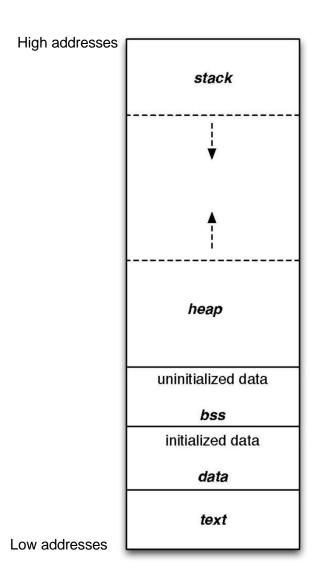
The stack, stack frames, function prologue and epilogue

Reese Pearsall Fall 2024

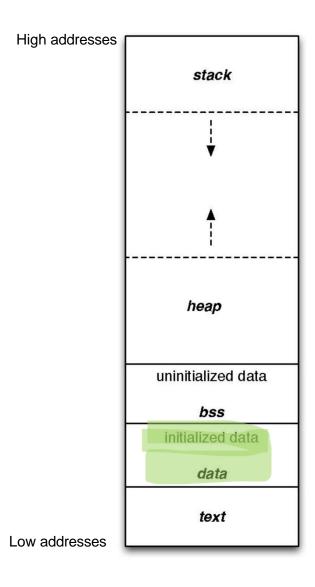
Announcements

Lab 2 (Shellshock) posted

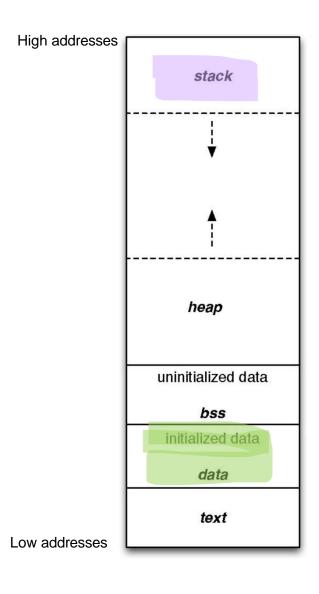
• Still ironing out some issues for the Apple M1/M2 people



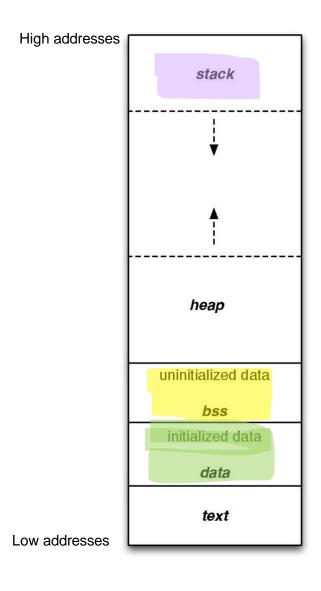
```
int x = 100;
int main()
       int a = 2;
       float b = 2.5;
       static int y;
       int *ptr = (int *) malloc(2*sizeof(int));
      ptr[0] = 5;
      ptr[1] = 6;
       free (ptr)
       return 1;
```



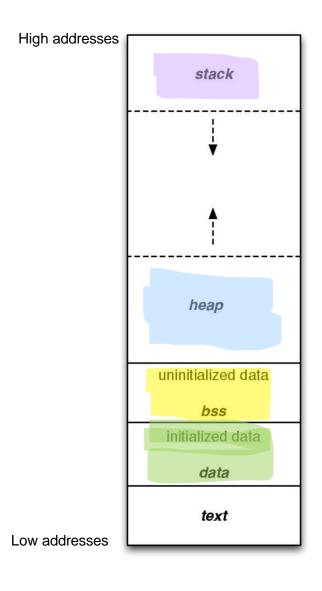
```
int x = 100;
int main()
       int a = 2;
       float b = 2.5;
       static int y;
       int *ptr = (int *) malloc(2*sizeof(int));
      ptr[0] = 5;
      ptr[1] = 6;
       free (ptr)
       return 1;
```



```
int x = 100;
int main()
      int a = 2;
      float b = 2.5;
       static int y;
       int *ptr = (int *) malloc(2*sizeof(int));
      ptr[0] = 5;
      ptr[1] = 6;
       free (ptr)
       return 1;
```



```
int x = 100;
int main()
      int a = 2;
      float b = 2.5;
      static int y;
      int *ptr = (int *) malloc(2*sizeof(int));
      ptr[0] = 5;
      ptr[1] = 6;
      free (ptr)
      return 1;
```



```
int x = 100;
int main()
      int a = 2;
      float b = 2.5;
      static int y;
      int *ptr = (int *) malloc(2*sizeof(int));
      ptr[0] = 5;
      ptr[1] = 6;
      free (ptr)
      return 1;
```

```
int main() {
    int x = 3;
    int y = 3;

    foo(x,y)

    int a = 0;
    foo2(a);

return 0;
}
```

```
int foo(x,y) {
    printf(x);
    printf(y);
    int z = 1;
    foo2(z)
    return 0;
}
```

```
int foo2(p){
     printf(p);
     return 0;
}
```

The Stack

```
int main(){
       int x = 3;
       int y = 3;
       foo(x, y)
       int a = 0;
       foo2(a);
       return 0;
```

```
int foo(x,y) {
    printf(x);
    printf(y);

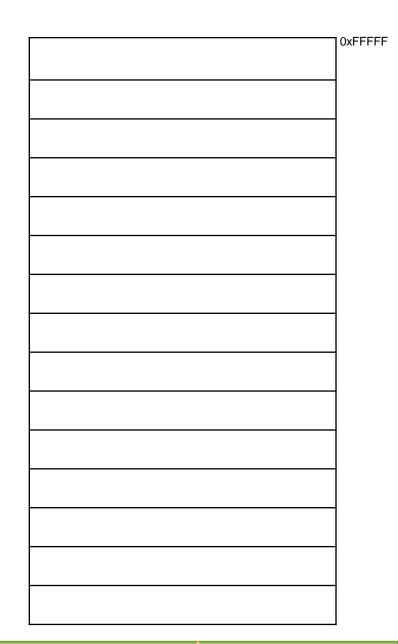
int z = 1;

foo2(z)

return 0;
}
```

```
int foo2(p){
    printf(p);
    return 0;
}
```

Every time a function is called, memory gets allocated on **the stack** to hold function values and information



The Stack

```
int main(){
       int x = 3;
       int y = 3;
       foo(x, y)
       int a = 0;
       foo2(a);
       return 0;
```

```
int foo(x,y) {
    printf(x);
    printf(y);
    int z = 1;
    foo2(z)
    return 0;
}
```

This memory on the stack is called a **stack frame**

```
int foo2(p){
    printf(p);
    return 0;
}
```

Every time a function is called, memory gets allocated on **the stack** to hold function values and information

The Stack

```
int main(){
       int x = 3;
       int y = 3;
       foo(x, y)
       int a = 0;
       foo2(a);
       return 0;
```

```
int foo(x,y) {
       printf(x);
       printf(y);
       int z = 1;
       foo2(z)
       return 0;
```

```
Stack Frame Format
```

```
Value of Arg 1
Value of Arg 2
Return Address
Previous Frame Pointer
Value of Var 1
Value of Var 1
```

```
int foo2(p){
       printf(p);
       return 0;
```

The stack frame consists of local variables, function arguments, and addresses

```
int main() {
    int x = 3;
    int y = 3;

    foo(x,y)

    int a = 0;
    foo2(a);

return 0;
}
```

```
int foo(x,y) {
    printf(x);
    printf(y);
    int z = 1;
    foo2(z)
    return 0;
}
```

```
Value of Arg 1

Value of Arg 2

Return Address

Previous Frame
Pointer

Value of Var 1

Value of Var 1
```

The Stack

Stack frame for main()

0xFFFFF **Return Address for Main Previous Frame Pointer** X = 3Y = 3

int foo2(p){

printf(p);

return 0;

```
int main() {
    int x = 3;
    int y = 3;
    foo(x,y)

    int a = 0;
    foo2(z)
    foo2(a);
    return 0;
}
```

```
Value of Arg 1

Value of Arg 2

Return Address

Previous Frame
Pointer

Value of Var 1

Value of Var 1
```

The Stack

Stack frame for main()

0xFFFFF **Return Address for Main Previous Frame Pointer** X = 3Y = 3

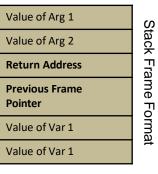
```
int main(){
       int x = 3;
       int y = 3;
       foo(x, y)
       int a = 0;
       foo2(a);
       return 0;
```

```
int foo2(p){
       printf(p);
       return 0;
```

```
int foo(x,y) {
      printf(x);
      printf(y);
      int z = 1;
      foo2(z)
      return 0;
```

Stack

foo()



Format

Return Address for Main Stack

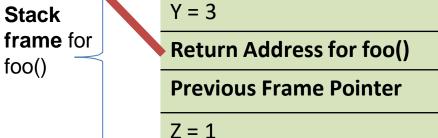
frame for

main()



X = 3

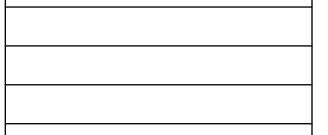
We need to know where to return to when this function finishes



X = 3

The Stack

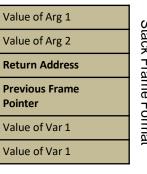




```
int main(){
       int x = 3;
       int y = 3;
       foo(x, y)
       int a = 0;
       foo2(a);
       return 0;
```

```
int foo2(p){
       printf(p);
       return 0;
```

```
int foo(x,y) {
      printf(x);
      printf(y);
      int z = 1;
      foo2(z)
      return 0;
```



Stack Frame Format

The Stack

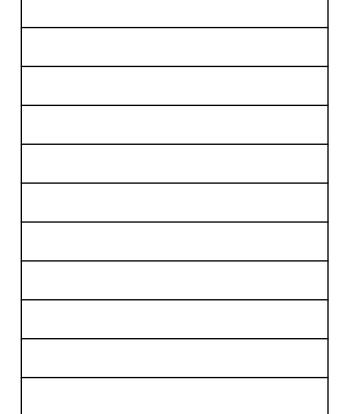
Return Address for Main

Stack **frame** for main()



X = 3

We need to know where to return to when this function finishes



```
int main() {
    int x = 3;
    int y = 3;

    foo(x,y)

    int a = 0;
    foo2(a);

return 0;
}
```

```
int foo(x,y) {
    printf(x);
    printf(y);

int z = 1;

foo2(z)

return 0;
}
```

```
Value of Arg 1

Value of Arg 2

Return Address

Previous Frame
Pointer

Value of Var 1

Value of Var 1
```

The Stack

```
0xFFFFF
              Return Address for Main
Stack
              Previous Frame Pointer
frame for
main()
              X = 3
              Y = 3
              X = 3
              Y = 3
Stack
frame for
              Return Address for foo()
foo()
              Previous Frame Pointer
              Z = 1
```

```
int main() {
    int x = 3;
    int y = 3;

    foo(x,y)

    int a = 0;
    foo2(a);

    return 0;
}
```

```
int foo(x,y) {
    printf(x);
    printf(y);

int z = 1;

foo2(z) 
return 0;
}
```

```
Value of Arg 1

Value of Arg 2

Return Address

Previous Frame
Pointer

Value of Var 1

Value of Var 1
```

The Stack

```
0xFFFFF
              Return Address for Main
Stack
              Previous Frame Pointer
frame for
main()
              X = 3
              Y = 3
              X = 3
              Y = 3
Stack
frame for
              Return Address for foo()
foo()
              Previous Frame Pointer
              Z = 1
```

```
int foo2(p){
    printf(p);

return 0;
}
```

Stack frame for foo2()

p = 1
Return Address for foo2

Previous Frame Pointer

```
int main(){
       int x = 3;
       int y = 3;
       foo(x, y)
      int a = 0;
       foo2(a);
       return 0;
```

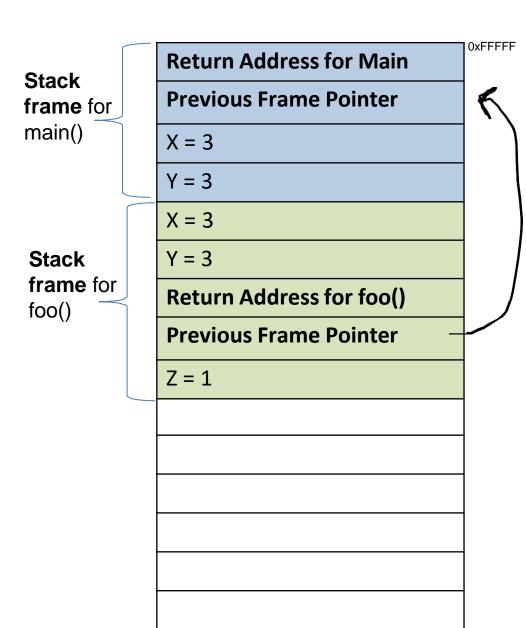
```
int foo2(p){
      printf(p);
      return 0;
```

```
Stack Frame
                              Return Address
                              Previous Frame
                                             Format
                              Pointer
int foo(x, y) {
                              Value of Var 1
                              Value of Var 1
          printf(x);
          printf(y);
          int z = 1;
          foo2(z)
           leturn 0;
              p = 1
 Stack
              Return Address for foo2
 frame for
 foo2()
              Previous Frame Pointer
```

Value of Arg 1

Value of Arg 2

The Stack



```
int main() {
    int x = 3;
    int y = 3;

    foo(x,y)

    int a = 0;
    foo2(a);

return 0;
}
```

```
int foo2(p){
    printf(p);
    return 0;
}
```

```
int foo(x,y) {
    printf(x);
    printf(y);

int z = 1;

foo2(z)
    return 0;
}
```

```
Value of Arg 1

Value of Arg 2

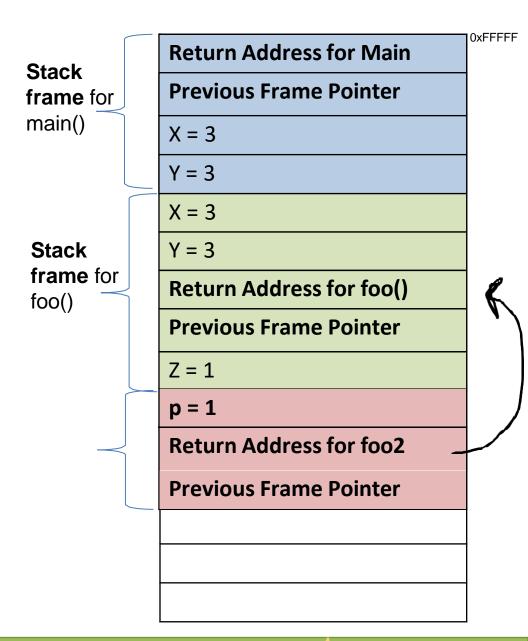
Return Address

Previous Frame
Pointer

Value of Var 1

Value of Var 1
```

The Stack



```
int main() {
    int x = 3;
    int y = 3;

    foo(x,y)

    int a = 0;
    foo2(a);

return 0;
}
```

```
int foo(x,y) {
    printf(x);
    printf(y);

int z = 1;

foo2(z)
    return 0;
}
```

The Stack

```
Value of Arg 2

Return Address

Previous Frame
Pointer

Value of Var 1

Value of Var 1
```

Value of Arg 1

Stack frame for main()

Stack

foo()

frame for

Return Address for Main
Previous Frame Pointer

```
Y = 3
```

X = 3

X = 3

Y = 3

Return Address for foo()

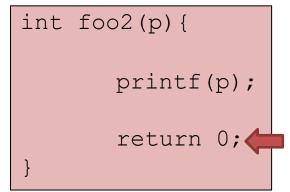
Previous Frame Pointer

Z = 1

p = 1

Return Address for foo2

Previous Frame Pointer



This function is finished, so we need to determine where the next instruction of the program is

```
int main(){
       int x = 3;
       int y = 3;
       foo(x, y)
       int a = 0;
       foo2(a);
       return 0;
```

```
int foo(x, y) {
      printf(x);
      printf(y);
      int z = 1;
       foo2(z)
       return 0;
```

```
Value of Arg 1
                             Stack
Value of Arg 2
                             Frame
Return Address
Previous Frame
                            Format
Pointer
Value of Var 1
Value of Var 1
```

The Stack

Return Address for Main

Previous Frame Pointer

0xFFFFF

```
Stack
frame for
main()
```

X = 3

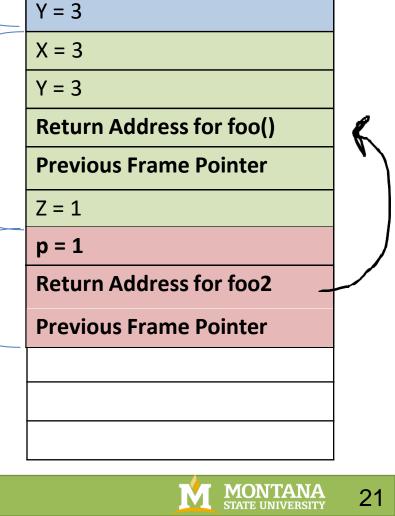
```
Stack
frame for
```

foo()

This function is finished, so we need to determine where the next instruction of the program is Look at the return address in the stack frame!

printf(p); return 0;

int foo2(p){



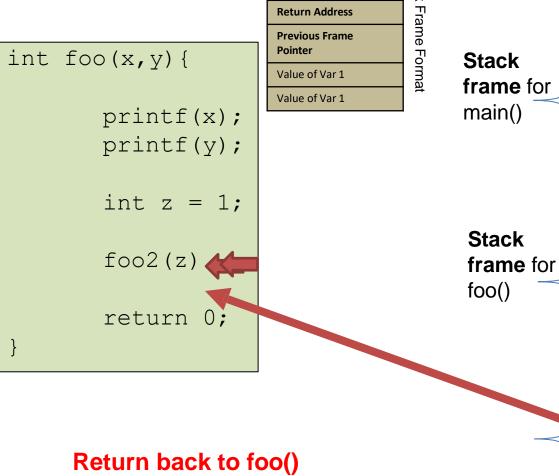
```
int main() {
    int x = 3;
    int y = 3;

    foo(x,y)

    int a = 0;
    foo2(a);

return 0;
}
```

```
int foo2(p){
    printf(p);
    return 0;
}
```



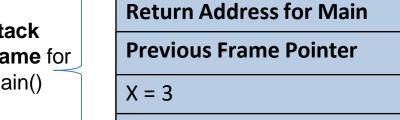
Value of Arg 1

Value of Arg 2

Stack

This function is finished, so we need to determine where the next instruction of the program is **Look at the return address in the stack frame!**

The Stack





Y = 3

$$Y = 3$$

Return Address for foo()

Previous Frame Pointer

$$p = 1$$

Return Address for foo2

Previous Frame Pointer

```
int main() {
    int x = 3;
    int y = 3;

    foo(x,y)

    int a = 0;
    foo2(a);

return 0;
}
```

```
Value of Arg 1

Value of Arg 2

Return Address

Previous Frame
Pointer

Value of Var 1

Value of Var 1
```

The Stack

```
Return Address for Main
Stack
              Previous Frame Pointer
frame for
main()
              X = 3
              Y = 3
              X = 3
              Y = 3
Stack
frame for
              Return Address for foo()
foo()
              Previous Frame Pointer
              Z = 1
```

```
int foo2(p){
    printf(p);
    return 0;
}
```

foo2 () is finished, so we can remove their information from the stack

```
int main() {
    int x = 3;
    int y = 3;
    foo(x,y)

    int z = 1;

    int a = 0;
    foo2(z)
    foo2(a);
    return 0;
}
```

foo() is done, we now need to return back to main!

Value of Arg 1

Value of Arg 2

Return Address

Previous Frame

Value of Var 1

Value of Var 1

Pointer

Stack Frame

Format

The Stack

0xFFFFF **Return Address for Main** Stack **Previous Frame Pointer frame** for main() X = 3Y = 3X = 3Y = 3Stack **frame** for **Return Address for foo()** foo() **Previous Frame Pointer** Z = 1

int foo2(p){

```
int main() {
    int x = 3;
    int y = 3;

    foo(x,y)

    int a = 0;
    foo2(a);

return 0;
}
```

```
int foo(x,y) {
    printf(x);
    printf(y);

int z = 1;

foo2(z)

return 0;
}
```

```
Value of Arg 2

Return Address

Previous Frame
Pointer

Value of Var 1

Value of Var 1
```

Value of Arg 1

The Stack

Stack frame for main()

```
0xFFFFF
Return Address for Main
Previous Frame Pointer
X = 3
Y = 3
```

```
int foo2(p){
    printf(p);
    return 0;
}
```

foo() is done, we now need to return back to main!

```
int main() {
    int x = 3;
    int y = 3;

    foo(x,y)

    int a = 0;
    foo2(a);

    return 0;
}
```

printf(p);

return 0;

int foo2(p){

```
int foo(x,y) {
    printf(x);
    printf(y);

int z = 1;

foo2(z)

return 0;
}
```

```
Value of Arg 1

Value of Arg 2

Return Address

Previous Frame
Pointer

Value of Var 1

Value of Var 1
```

The Stack

Stack frame for main()

0xFFFFF **Return Address for Main Previous Frame Pointer** X = 3Y = 3a = 0

```
int main(){
       int x = 3;
       int y = 3;
       foo(x, y)
       int a = 0;
       foo2(a);
       return 0;
```

int foo2(p){

printf(p);

return 0;

```
int foo(x, y) {
       printf(x);
       printf(y);
       int z = 1;
       foo2(z)
       return 0;
```

The Stack Stack Frame

```
Return Address for Main
Format
     Stack
                    Previous Frame Pointer
     frame for
     main()
                    X = 3
 Stack
 frame for
 foo2()
```

```
Y = 3
a = 0
0 = q
Return Address for foo2
```

Previous Frame Pointer

foo2() is called again, so a new stack frame is created and put onto the stack

Value of Arg 1

Value of Arg 2 Return Address

Previous Frame

Value of Var 1

Value of Var 1

Pointer

```
int main() {
    int x = 3;
    int y = 3;

    foo(x,y)

    int a = 0;
    foo2(a);

    return 0;
}
```

```
int foo(x,y) {
    printf(x);
    printf(y);

int z = 1;

foo2(z)

return 0;
}
```

The Stack

```
Return Address for Main
   Stack
                 Previous Frame Pointer
   frame for
   main()
                 X = 3
                 Y = 3
                a = 0
                 0 = q
Stack
                 Return Address for foo2
frame for
                 Previous Frame Pointer
foo2()
```

```
int foo2(p){
    printf(p);
    return 0;
}
```

When foo2() is finished, it will return back to main()

Value of Arg 1

Value of Arg 2

Return Address

Previous Frame

Value of Var 1

Value of Var 1

Pointer

Stack Frame

Format

The Stack Value of Arg 1 **Stack and Function Invocation** Stack Value of Arg 2 Frame Return Address 0xFFFFF **Previous Frame Return Address for Main** Format int main(){ Pointer int foo(x, y) { Stack Value of Var 1 **Previous Frame Pointer frame** for Value of Var 1 int x = 3; main() printf(x); X = 3int y = 3; printf(y); Y = 3a = 0foo(x, y)int z = 1; p = 0foo2(z)int a = 0; Ctaal **Return Address for foo2** foo2(a); **frame** for **Previous Frame Pointer** return 0; foo2() return 0;

When foo2() is

int foo2(p){

```
int main() {
    int x = 3;
    int y = 3;

    foo(x,y)

int a = 0;
    foo2(a);

return 0;
}
```

```
int foo(x,y) {
    printf(x);
    printf(y);

int z = 1;

foo2(z)

return 0;
}
```

```
Value of Arg 1

Value of Arg 2

Return Address

Previous Frame
Pointer

Value of Var 1

Value of Var 1
```

The Stack

Stack frame for main()

```
0xFFFFF
Return Address for Main
 Previous Frame Pointer
X = 3
Y = 3
a = 0
```

```
int foo2(p){
    printf(p);
    return 0;
}
```

When foo2() is finished, it will return back to main()

```
int main() {
    int x = 3;
    int y = 3;

    foo(x,y)

    int a = 0;
    foo2(a);

return 0;
}
```

printf(p);

return 0;

int foo2(p){

```
int foo(x,y) {
    printf(x);
    printf(y);

int z = 1;

foo2(z)

return 0;
}
```

Program done!

The Stack

Stack frame for main()

Stack Frame Format

```
Return Address for Main
Previous Frame Pointer
X = 3
Y = 3
a = 0
```

Value of Arg 1

Value of Arg 2

Return Address

Previous Frame

Value of Var 1

Value of Var 1

Pointer

```
int main() {
    int x = 3;
    int y = 3;

    foo(x,y)

    int a = 0;
    foo2(a);

return 0;
}
```

```
int foo(x,y) {
    printf(x);
    printf(y);

int z = 1;

foo2(z)

return 0;
}
```

```
int foo2(p){
    printf(p);
    return 0;
}
```

```
Program done!
```

Value of Arg 1

Value of Arg 2

Return Address

Previous Frame

Value of Var 1
Value of Var 1

Pointer

Stack Frame

Format

The Stack

0xFFFFF
1

Demo

is_c_good.c

```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
void foo(char *str)
        char buffer[10];
        strcpy(buffer, str);
int main(int argc, char *argv[])
        foo(argv[1]);
        printf("Returned Properly\n");
        return 0;
```

```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
void foo(char *str)
        char buffer[10];
        strcpy(buffer, str);
int main(int argc, char *argv[])
        foo(argv[1]);
        printf("Returned Properly\n");
        return 0;
```

The Stack

		□0xFFF
		UXFFF

The Stack

```
main() stack frame
```

```
0xFFFFF
Return Address
```

```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
void foo(char *str)
        char buffer[10];
        strcpy(buffer, str);
int main(int argc, char *argv[])
        foo(argv[1]);
        printf("Returned Properly\n");
        return 0;
```

0xFFFFF main() stack frame **Return Address** foo() stack frame **Return Address** CHAR BUFFER[]

```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
void foo(char *str)
        char buffer[10];
        strcpy(buffer, str);
int main(int argc, char *argv[])
        foo(argv[1]);
printf("Returned Properly\n");
        return 0;
```

0xFFFFF main() stack frame **Return Address** foo() stack frame **Return Address** CHAR BUFFER[]

```
#include <string.h>
#include <stdio.h>
                                                                                                0xFFFFF
#include <stdlib.h>
void foo(char *str)
                                                      main() stack frame
                                                                       Return Address
        char buffer[10];
         strcpy(buffer, str);
                                                       foo() stack frame
                                                                        Return Address
                                                                        CHAR BUFFER[]
int main(int argc, char *argv[])
        foo(argv[1]); ____
        printf("Returned Properly\n");
         return 0;
```

The input of this program eventually gets put on the stack!

```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
void foo(char *str)
        char buffer[10];
        strcpy(buffer, str);
int main(int argc, char *argv[])
        foo(argv[1]);
        printf("Returned Properly\n");
        return 0;
```

main() stack frame **Return Address** foo() stack frame **Return Address** CHAR BUFFER[] AAAAAAAA

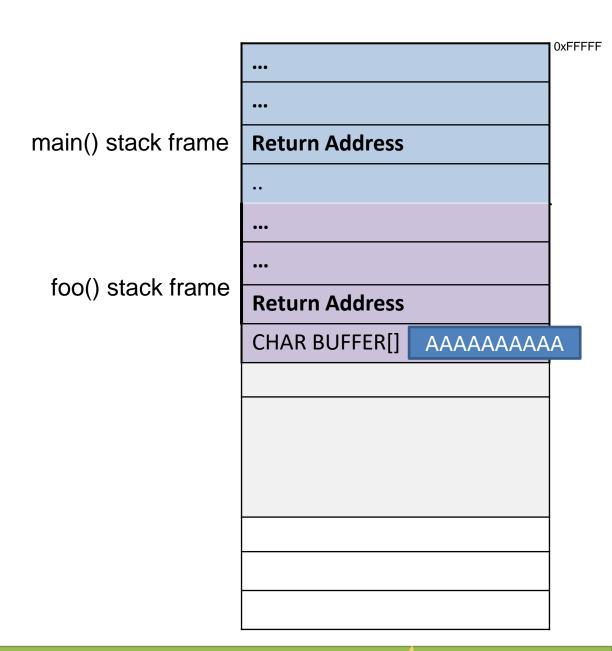
buffer[] can only hold 10 characters, right?

0xFFFFF

#include <string.h> #include <stdio.h> #include <stdlib.h> void foo(char *str) char buffer[10]; strcpy(buffer, str); int main(int argc, char *argv[]) foo(argv[1]); printf("Returned Properly\n"); return 0;

C doesn't care.

The Stack



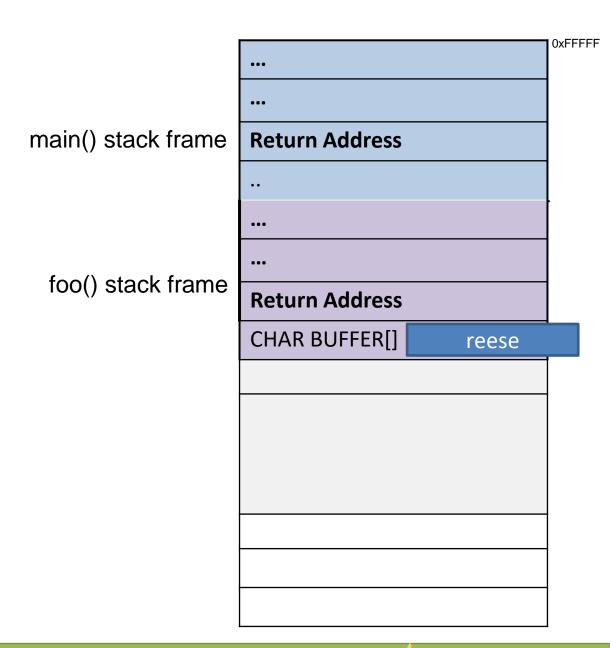
#include <string.h> #include <stdio.h> #include <stdlib.h> void foo(char *str) char buffer[10]; strcpy(buffer, str); int main(int argc, char *argv[]) foo(argv[1]); printf("Returned Properly\n"); return 0;

C doesn't care.

Instead of ./myprogram reese

What if we did.....

The Stack

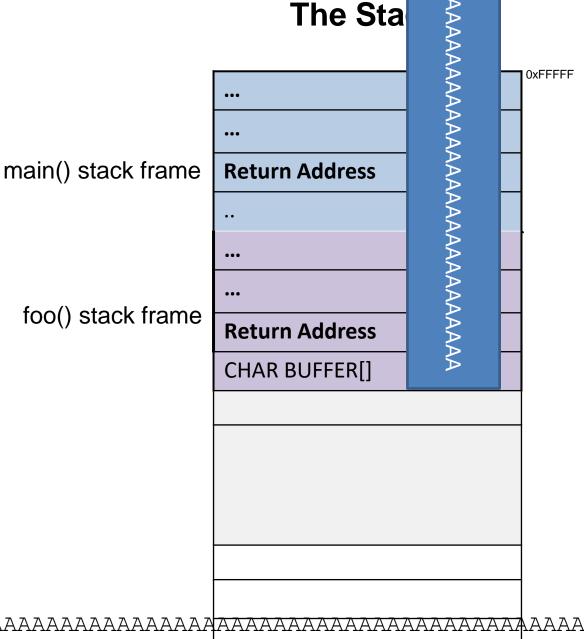


```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
void foo(char *str)
        char buffer[10];
        strcpy(buffer, str);
int main(int argc, char *argv[])
        foo(argv[1]);
        printf("Returned Properly\n");
        return 0;
```

C doesn't care.

Instead of ./myprogram reese

What if we did.....

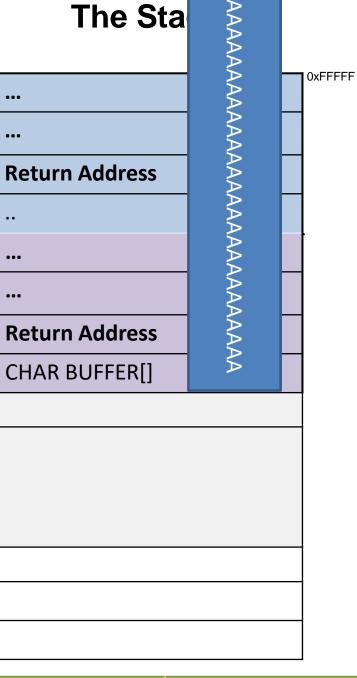


```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
void foo(char *str)
        char buffer[10];
        strcpy(buffer, str);
int main(int argc, char *argv[])
        foo(argv[1]);
        printf("Returned Properly\n");
        return 0;
```

main() stack frame foo() stack frame

We can **overflow** this buffer!

This will **overwrite** other values on the Stack



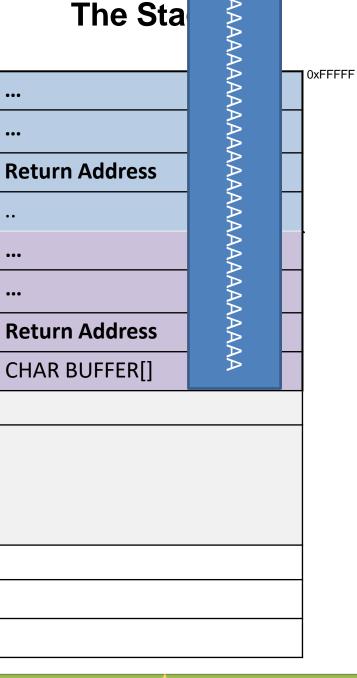
```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
void foo(char *str)
        char buffer[10];
        strcpy(buffer, str);
int main(int argc, char *argv[])
        foo(argv[1]);
        printf("Returned Properly\n");
        return 0;
```

main() stack frame foo() stack frame

We can **overflow** this buffer!

This will **overwrite** other values on the Stack

What can our input control?

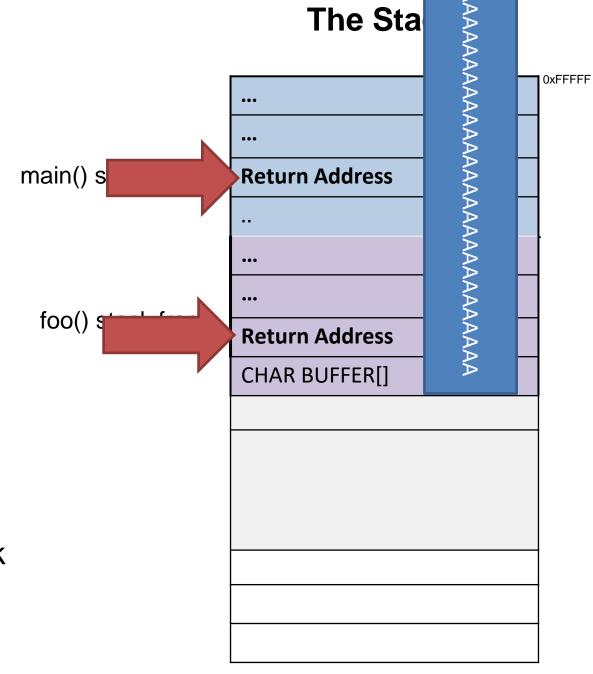


```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
void foo(char *str)
        char buffer[10];
        strcpy(buffer, str);
int main(int argc, char *argv[])
        foo(argv[1]);
        printf("Returned Properly\n");
        return 0;
```

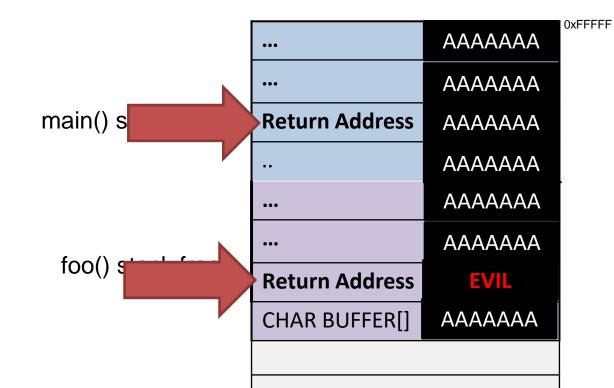
We can **overflow** this buffer!

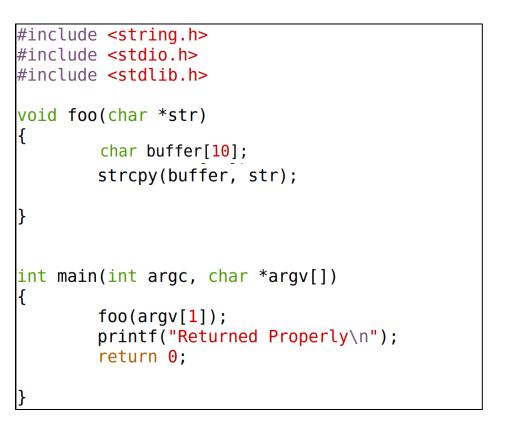
This will **overwrite** other values on the Stack

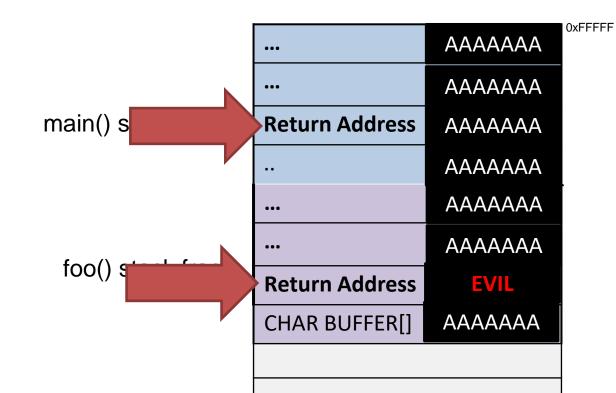
Our input can overwrite values on the stack, specifically, the **return address**



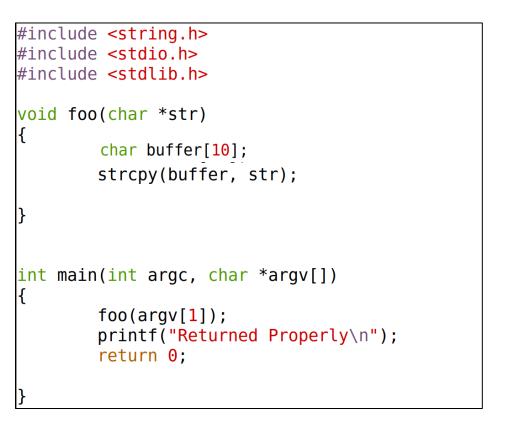
```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
void foo(char *str)
        char buffer[10];
        strcpy(buffer, str);
int main(int argc, char *argv[])
        foo(argv[1]);
        printf("Returned Properly\n");
        return 0;
```

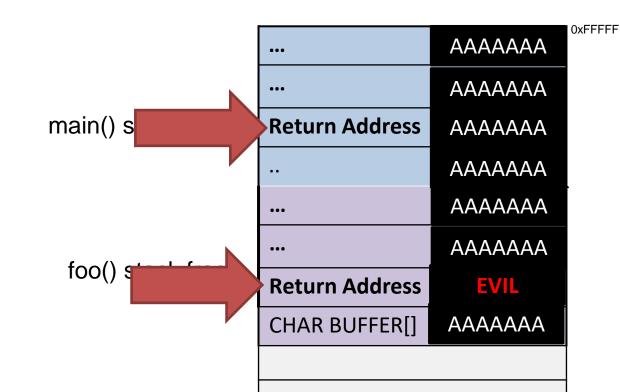






Instead of **EVIL**, what could we overwrite it with?





Instead of **EVIL**, what could we overwrite it with?



Our own malicious code

... previous stack frames...

Arguments Return Address Previous frame pointer buffer[99] buffer[0]

The CPU needs to keep track of two things:

1. The location of the top of stack

2. The location of the current stack frame we are executing

... previous stack frames...

Arguments

Return Address

Previous frame pointer

buffer[99]

•

•

•

.

buffer[0]

The CPU needs to keep track of two things:

1. The location of the top of stack

2. The location of the current stack frame we are executing

?????

... previous stack frames...

Arguments

Return Address

Previous frame pointer

buffer[99]

.
.
.
.
buffer[0]

The CPU needs to keep track of two things:

1. The location of the top of stack

The register **\$esp** points to the top of the **s**tack

2. The location of the current stack frame we are executing

\$esp

... previous stack frames... **Arguments** Return Address Previous frame pointer buffer[99] buffer[0]

The CPU needs to keep track of two things:

1. The location of the top of stack

The register **\$esp** points to the top of the **s**tack

\$esp

2. The location of the current stack frame we are executing

The register **\$ebp** points to the **b**ase of the current stack frame

... previous stack frames...

```
$ esp
void main()
                                          push
                                                  $0x3
                                                              ; push b
                                                  $0x2
                                          push
                                                             ; push a
    foo(2,3);
                                          call
                                                  .... <foo> ; push RA
    return 0;
                                          . . .
                                          push
                                                  %ebp
                                                                   ; save ebp
                                                  %esp, %ebp
                                                                   ; set ebp
                                          mov
                                          . . .
void foo(int a, int b)
                                                  0x8(%ebp), %edx
                                          mov
                                                                   ; a
                                                  0xc(%ebp), %eax
                                          mov
                                                                   ; b
    int x, y;
                                                  %edx, %eax.
                                          add
    x = a + b;
                                                  %eax,-0x8(%ebp); x=
                                          mov
    y = a - b;
                                                  0x8(%ebp), %eax ; etc.
                                          mov
                                          sub
                                                 0xc(%ebp), %eax
                                                  %eax, -0x4(%ebp)
                                          mov
                                          . . .
                                          leave ; set esp = ebp
                                                 ; pop ebp
                                                 ; pop RA
                                          ret
```



... previous stack frames...

Value of B

```
$masp()
                                                  $0x3
                                                              ; push b
                                          push
                                                  $0x2
                                          push
                                                              ; push a
    foo(2,3);
                                          call
                                                  .... <foo> ; push RA
    return 0;
                                           . . .
                                                                   ; save ebp
                                          push
                                                  %ebp
                                                  %esp, %ebp
                                                                   ; set ebp
                                          mov
                                           . . .
void foo(int a, int b)
                                                  0x8(%ebp), %edx
                                          mov
                                                                   ; a
                                                  0xc(%ebp), %eax
                                          mov
                                                                   ; b
    int x, y;
                                                  %edx, %eax.
                                          add
    x = a + b;
                                                  %eax,-0x8(%ebp); x=
                                          mov
    y = a - b;
                                                  0x8(%ebp), %eax ; etc.
                                          mov
                                          sub
                                                  0xc(%ebp), %eax
                                                  %eax, -0x4(%ebp)
                                          mov
                                          . . .
                                          leave ; set esp = ebp
                                                 ; pop ebp
                                                 ; pop RA
                                          ret
```

... previous stack frames...

Value of B

Value of A

```
void main()
                                                  $0x3
                                                              ; push b
                                          push
                                                  $0x2
                                          push
                                                              ; push a
    f$.esp3);
                                          call
                                                  .... <foo> ; push RA
    return 0;
                                           . . .
                                                                   ; save ebp
                                          push
                                                  %ebp
                                                  %esp, %ebp
                                                                   ; set ebp
                                          mov
                                           . . .
void foo(int a, int b)
                                                  0x8(%ebp), %edx
                                          mov
                                                                   ; a
                                                  0xc(%ebp), %eax
                                          mov
                                                                   ; b
    int x, y;
                                                  %edx, %eax.
                                          add
    x = a + b;
                                                  %eax,-0x8(%ebp); x=
                                          mov
    y = a - b;
                                                  0x8(%ebp), %eax ; etc.
                                          mov
                                          sub
                                                  0xc(%ebp), %eax
                                                  %eax, -0x4(%ebp)
                                          mov
                                          . . .
                                          leave ; set esp = ebp
                                                 ; pop ebp
                                          ret
                                                 ; pop RA
```

... previous stack frames...

Value of B

Value of A

Return Address back to main()

```
void main()
                                                    $0x3
                                            push
                                                                ; push b
                                           push
                                                    $0x2
                                                                ; push a
    foo(2,3);
                                                   .... <foo> ; push RA
                                            call
    return 0;
                                            . . .
   $ esp
                                            push
                                                    %ebp
                                                                      ; save ebp
                                                   %esp, %ebp
                                                                      ; set ebp
                                            mov
                                            . . .
void foo(int a, int b)
                                                   0x8(%ebp), %edx
                                            mov
                                                                      ; a
                                                   0xc(%ebp), %eax
                                            mov
                                                                      ; b
    int x, y;
                                            add
                                                   %edx, %eax.
    x = a + b;
                                                   ext{%eax} = 0x8(ext{%ebp}) ; x=
                                            mov
    y = a - b;
                                                   0x8(%ebp), %eax ; etc.
                                            mov
                                                   0xc(%ebp), %eax
                                            sub
                                                   %eax, -0x4(%ebp)
                                            mov
                                            . . .
                                            leave ; set esp = ebp
                                                   ; pop ebp
                                            ret
                                                   ; pop RA
```

... previous stack frames...

Value of B

Value of A

Return Address back to main()

```
void main()
                                                    $0x3
                                            push
                                                                ; push b
                                                    $0x2
                                            push
                                                                ; push a
    foo(2,3);
                                                    .... <foo> ; push RA
                                            call
    return 0;
                                            . . .
    $ esp
                                            push
                                                    %ebp
                                                                      ; save ebp
                                                   %esp, %ebp
                                                                      ; set ebp
                                            mov
                                            . . .
void foo(int a, int b)
                                                    0x8(%ebp), %edx
                                            mov
                                                                      ; a
                                                    0xc(%ebp), %eax
                                            mov
                                                                      ; b
    int x, y;
                                            add
                                                    %edx, %eax.
    x = a + b;
                                                    ext{%eax} = 0x8(ext{%ebp}) ; x=
                                            mov
    y = a - b;
                                                   0x8(%ebp), %eax ; etc.
                                            mov
                                                   0xc(%ebp), %eax
                                            sub
                                                   %eax, -0x4(%ebp)
                                            mov
                                            . . .
                                            leave ; set esp = ebp
                                                   ; pop ebp
                                            ret
                                                   ; pop RA
```

... previous stack frames...

Value of B

Value of A

Return Address back to main()

```
void main()
                                           push
                                                   $0x3
                                                               ; push b
                                           push
                                                   $0x2
                                                               ; push a
    foo(2,3);
                                            call
                                                   .... <foo> ; push RA
    return 0;
                                            . . .
   $ esp
                                                   %ebp
                                            push
                                                                     ; save ebp
                                                   %esp, %ebp
                                                                     ; set ebp
                                           mov
                                            . . .
void foo(int a, int b)
                                                   0x8(%ebp), %edx
                                                                     ; a
                                           mov
                                                   0xc(%ebp), %eax
                                           mov
                                                                     ; b
    int x, y;
                                            add
                                                   %edx, %eax.
    x = a + b;
                                                   ext{%eax} = 0x8(ext{%ebp}) ; x=
                                           mov
    y = a - b;
                                                   0x8(%ebp), %eax ; etc.
                                           mov
                                                   0xc(%ebp), %eax
                                            sub
                                                   %eax, -0x4(%ebp)
                                           mov
                                            . . .
                                            leave ; set esp = ebp
                                                   ; pop ebp
                                            ret
                                                  ; pop RA
```

Every time a function is called, the **function prologue** occurs

... previous stack frames...

\$ ebp

Value of B

Value of A

Return Address back to main()

Previous Frame Pointer

```
void main()
                                           push
                                                  $0x3
                                                              ; push b
                                           push
                                                  $0x2
                                                              ; push a
    foo(2,3);
                                           call
                                                  .... <foo> ; push RA
    return 0;
                                           . . .
    $ esp
                                                  %ebp
                                           push
                                                                    ; save ebp
                                                  %esp, %ebp
                                                                    ; set ebp
                                           mov
                                           . . .
void foo(int a, int b)
                                                  0x8(%ebp), %edx
                                           mov
                                                                    ; a
                                                  0xc(%ebp), %eax
                                           mov
                                                                    ; b
    int x, y;
                                           add
                                                  %edx, %eax.
                                                                    ; +
    x = a + b;
                                                  ext{%eax} = 0x8(ext{%ebp}) ; x=
                                           mov
    y = a - b;
                                                  0x8(%ebp), %eax ; etc.
                                           mov
                                                  0xc(%ebp), %eax
                                           sub
                                                  %eax, -0x4(%ebp)
                                           mov
                                           . . .
                                           leave ; set esp = ebp
                                                  ; pop ebp
                                           ret
                                                 ; pop RA
```

... previous stack frames...

Value of B

Value of A

Return Address back to main()

Previous Frame Pointer

```
void main()
                                          push
                                                  $0x3
                                                              ; push b
                                                  $0x2
                                          push
                                                             ; push a
    foo(2,3);
                                          call
                                                 .... <foo> ; push RA
    return 0;
                                           . . .
                 $ebp
    $ esp
                                                  %ebp
                                          push
                                                                   ; save ebp
                                                  %esp, %ebp
                                                                   ; set ebp
                                          mov
                                           . . .
void foo(int a, int b)
                                                  0x8(%ebp), %edx
                                          mov
                                                                   ; a
                                                  0xc(%ebp), %eax
                                          mov
                                                                   ; b
    int x, y;
                                          add
                                                  %edx, %eax.
    x = a + b;
                                                  ext{eax}, -0x8(ext{ebp}); x=
                                          mov
    y = a - b;
                                                  0x8(%ebp), %eax ; etc.
                                          mov
                                                 0xc(%ebp), %eax
                                          sub
                                                  %eax, -0x4(%ebp)
                                          mov
                                          . . .
                                          leave ; set esp = ebp
                                                 ; pop ebp
                                          ret
                                                 ; pop RA
```

... previous stack frames...

Value of B

Value of A

Return Address back to main()

Previous Frame Pointer

Value of x

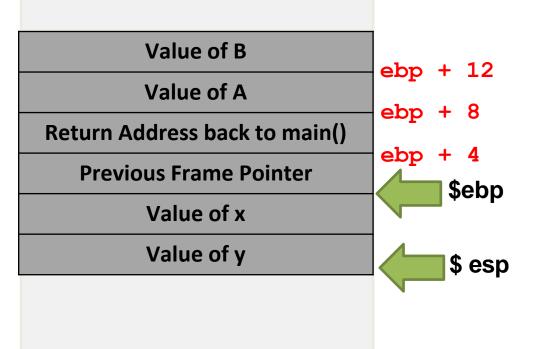
Value of y

```
void main()
                                           push
                                                  $0x3
                                                              ; push b
                                                  $0x2
                                           push
                                                              ; push a
    foo(2,3);
                                           call
                                                  .... <foo> ; push RA
    return 0;
                                           . . .
    $ebp
                                           push
                                                  %ebp
                                                                    ; save ebp
                                                  %esp, %ebp
                                                                    ; set ebp
                                           mov
   $fespint a, int b)
                                           . . .
                                                  0x8(%ebp), %edx
                                           mov
                                                                    ; a
                                                  0xc(%ebp), %eax
                                           mov
                                                                    ; b
    int x, y;
                                           add
                                                  %edx, %eax.
    x = a + b;
                                                  ext{%eax} = 0x8(ext{%ebp}) ; x=
                                           mov
    y = a - b;
                                                  0x8(%ebp), %eax ; etc.
                                           mov
                                                  0xc(%ebp), %eax
                                           sub
                                                  %eax, -0x4(%ebp)
                                           mov
                                           . . .
                                           leave ; set esp = ebp
                                                  ; pop ebp
                                           ret
                                                 ; pop RA
```

... previous stack frames... Value of B Value of A **Return Address back to main() Previous Frame Pointer** \$ebp Value of x Value of y \$ esp

Why is this helpful knowledge?

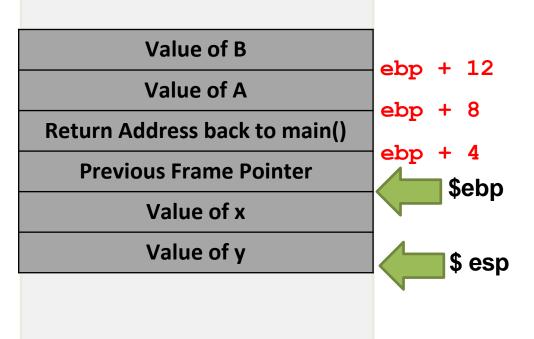
This tells us how the return address in put onto the stack, and how these important pointers are managed ... previous stack frames...



Why is this helpful knowledge?

This tells us how the return address in put onto the stack, and how these important pointers are managed

... previous stack frames...



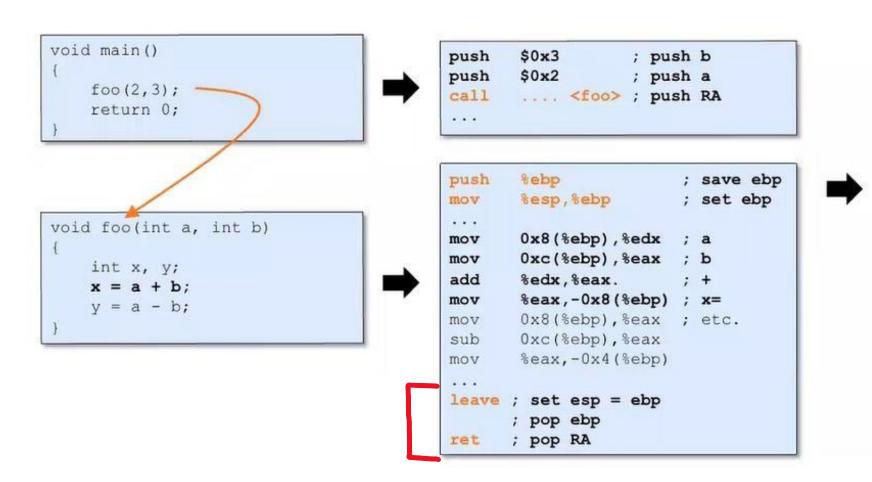
Why is this helpful knowledge?

This tells us how the return address in put onto the stack, and how these important pointers are managed

Every time a function is called, the **function prologue** occurs

... previous stack frames...

When a function finishes, a function epilogue occurs and cleans up the stack



```
int bof(char *str)
    char buffer[BUF SIZE];
    // potential buffer overflow!
    strcpy(buffer, str);
    return 1;
int main(int argc, char **argv)
    char str[517];
    FILE *badfile;
    badfile = fopen("badfile", "r");
    if (!badfile) {
        perror("Opening badfile"); exit(1);
    int length = fread(str, sizeof(char), 517, badfile);
    printf("Input size: %d\n", length);
    dummy function(str);
    fprintf(stdout, "==== Returned Properly ====\n");
    return 1;
// This function is used to insert a stack frame of size
// 1000 (approximately) between main's and bof's stack frames.
// The function itself does not do anything.
void dummy function(char *str)
    char dummy buffer[1000];
    memset(dummy buffer, 0, 1000);
    bof(str);
```

```
int bof(char *str)
    char buffer[BUF SIZE];
    // potential buffer overflow!
    strcpy(buffer, str);
    return 1;
int main(int argc, char **argv)
    char str[517];
    FILE *badfile;
    badfile = fopen("badfile", "r");
    if (!badfile) {
        perror("Opening badfile"); exit(1);
    int length = fread(str, sizeof(char), 517, badfile);
    printf("Input size: %d\n", length);
    dummy function(str);
    fprintf(stdout, "==== Returned Properly ====\n");
    return 1;
// This function is used to insert a stack frame of size
// 1000 (approximately) between main's and bof's stack frames.
// The function itself does not do anything.
void dummy function(char *str)
    char dummy buffer[1000];
    memset(dummy buffer, 0, 1000);
    bof(str);
```

Reads (up to) 517 bytes of data from badfile

```
int bof(char *str)
    char buffer[BUF SIZE];
    // potential buffer overflow!
    strcpy(buffer, str);
    return 1;
int main(int argc, char **argv)
    char str[517];
    FILE *badfile;
    badfile = fopen("badfile", "r");
    if (!badfile) {
        perror("Opening badfile"); exit(1);
    int length = fread(str, sizeof(char), 517, badfile);
    printf("Input size: %d\n", length);
    dummy function(str);
    fprintf(stdout, "==== Returned Properly ====\n");
    return 1;
// This function is used to insert a stack frame of size
// 1000 (approximately) between main's and bof's stack frames.
// The function itself does not do anything.
void dummy function(char *str)
    char dummy buffer[1000];
    memset(dummy buffer, 0, 1000);
    bof(str);
```

Reads (up to) 517 bytes of data from badfile

Storing the file contents into a str variable of size 517 bytes

```
int bof(char *str)
    char buffer[BUF SIZE];
    // potential buffer overflow!
    strcpy(buffer, str);
    return 1;
int main(int argc, char **argv)
    char str[517];
    FILE *badfile;
    badfile = fopen("badfile", "r");
    if (!badfile) {
        perror("Opening badfile"); exit(1);
    int length = fread(str, sizeof(char), 517, badfile);
    printf("Input size: %d\n", length);
    dummy function(str); -
    fprintf(stdout, "==== Returned Properly ====\n");
    return 1;
// This function is used to insert a stack frame of size
// 1000 (approximately) between main's and bof's stack frames.
// The function itself does not do anything.
void dummy function(char *str)
    char dummy buffer[1000];
    memset(dummy buffer, 0, 1000);
    bof(str);
```

Reads (up to) 517 bytes of data from badfile

Storing the file contents into a str variable of size 517 bytes

Calls the dummy_function() which calls bof()

```
int bof(char *str)
    char buffer[BUF SIZE];
    // potential buffer overflow!
    strcpy(buffer, str);
    return 1;
int main(int argc, char **argv)
    char str[517];
    FILE *badfile;
    badfile = fopen("badfile", "r");
    if (!badfile) {
        perror("Opening badfile"); exit(1);
    int length = fread(str, sizeof(char), 517, badfile);
    printf("Input size: %d\n", length);
    dummy function(str); -
    fprintf(stdout, "==== Returned Properly ====\n");
    return 1;
// This function is used to insert a stack frame of size
  1000 (approximately) between main's and bof's stack frames.
// The function itself does not do anything.
void dummy function(char *str)
    char dummy buffer[1000];
    memset(dummy buffer, 0, 1000);
    bof(str);
```

bof () function uses strcopy to copy function argument into buffer

```
BUF SIZE = 100
```

Reads (up to) 517 bytes of data from badfile

Storing the file contents into a str variable of size 517 bytes

Calls the dummy_function() which calls bof()

```
int bof(char *str)
    char buffer[BUF SIZE];
    // potential buffer overflow!
    strcpy(buffer, str);
    return 1;
int main(int argc, char **argv)
    char str[517];
    FILE *badfile;
    badfile = fopen("badfile", "r");
    if (!badfile) {
        perror("Opening badfile"); exit(1);
    int length = fread(str, sizeof(char), 517, badfile);
    printf("Input size: %d\n", length);
    dummy function(str); -
    fprintf(stdout, "==== Returned Properly ====\n");
    return 1;
// This function is used to insert a stack frame of size
  1000 (approximately) between main's and bof's stack frames.
// The function itself does not do anything.
void dummy function(char *str)
    char dummy buffer[1000];
    memset(dummy buffer, 0, 1000);
    bof(str);
```

bof () function uses strcopy to copy function argument into buffer

BUF SIZE = 100



There is no check if str is bigger than the buffer, so buffer overflow can occur!

Reads (up to) 517 bytes of data from badfile

Storing the file contents into a str variable of size 517 bytes

Calls the dummy_function() which calls bof()

```
int bof(char *str)
    char buffer[BUF SIZE];
    // potential buffer overflow!
    strcpy(buffer, str);
    return 1;
int main(int argc, char **argv)
    char str[517];
    FILE *badfile;
    badfile = fopen("badfile", "r");
    if (!badfile) {
        perror("Opening badfile"); exit(1);
    int length = fread(str, sizeof(char), 517, badfile);
    printf("Input size: %d\n", length);
    dummy function(str); -
    fprintf(stdout, "==== Returned Properly ====\n");
    return 1;
// This function is used to insert a stack frame of size
  1000 (approximately) between main's and bof's stack frames.
// The function itself does not do anything.
void dummy function(char *str)
    char dummy buffer[1000];
    memset(dummy buffer, 0, 1000);
    bof(str);
```

bof () function uses strcopy to copy function argument into buffer

BUF SIZE = 100



There is no check if str is bigger than the buffer, so buffer overflow can occur!

Reads (up to) 517 bytes of data from badfile

Storing the file contents into a str variable of size 517 bytes

> Calls the dummy_function() which calls bof()

buffer is a stack variable, so we can overwrite other values on the stack with a buffer overflow!

... previous stack frames... Arguments **Return Address** Previous frame pointer buffer[99] buffer[0]

Here is the current stack frame in bof()

We can control the contents of buffer[] with our badfile

... previous stack frames... **Arguments Return Address** Previous frame pointer buffer[99] buffer[0]

Here is the current stack frame in bof()

We can control the contents of buffer[] with our badfile

We can overflow this buffer and overwrite the contents above it



... previous stack frames...

Arguments

Return Address

Previous frame pointer

buffer[99]

- -
- .
- .
- .
- buffer[0]

The juicy piece of information here in the **return address**

The program will jump to that address and continue to execute code

... previous stack frames...

Arguments

Return Address

Previous frame pointer

buffer[99]

- .
- .
- •
- buffer[0]

The juicy piece of information here in the **return address**

The program will jump to that address and continue to execute code

Overwriting the return address with something else can lead to:

Non-existent address

→ CRASH

Access Violation

→ CRASH

Invalid Instruction

→ CRASH

Execution of attacker's code! → Oh no!!

... previous stack frames...

Arguments

Return Address

Previous frame pointer

buffer[99]

- .
- .
- .
- ٠
- buffer[0]

The juicy piece of information here in the **return address**

The program will jump to that address and continue to execute code

We can overwrite it, so if it points to the location of our own code we also inject, it will execute that code!

... previous stack frames...

Arguments

Return Address

Previous frame pointer

buffer[99]

- .
- .
- •
- •

buffer[0]

The juicy piece of information here in the **return address**

The program will jump to that address and continue to execute code

We can overwrite it, so if it points to the location of our own code we also inject, it will execute that code!

And our code will get a root shell

(there are many things our code can do, but we will be focused on getting a root shell)