CSCI 476: Computer Security

Buffer Overflow Attack (Part 1)

The stack, stack frames, function prologue and epilogue

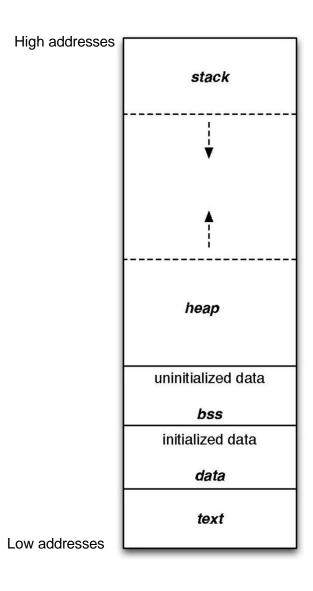
Reese Pearsall Spring 2023

Announcements

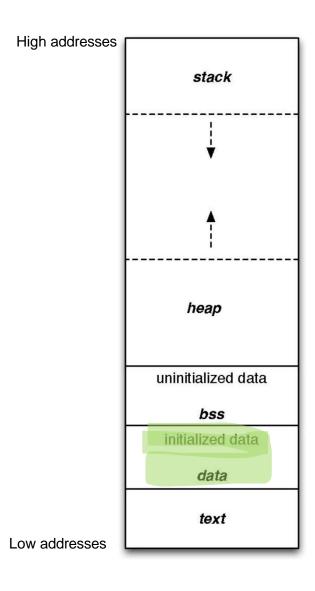
Lab 2 (Shellshock) due on Sunday 2/19

VM Issues

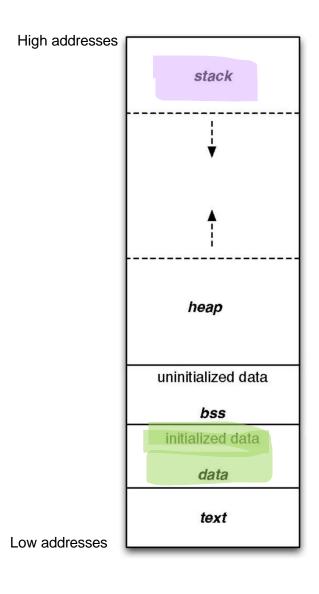
- Often times, the fastest solution is to create a brand new VM
- Crank up video memory



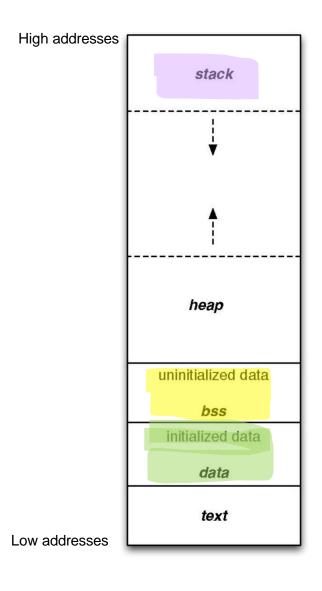
```
int x = 100;
int main()
       int a = 2;
       float b = 2.5;
       static int y;
       int *ptr = (int *) malloc(2*sizeof(int));
      ptr[0] = 5;
      ptr[1] = 6;
       free (ptr)
       return 1;
```



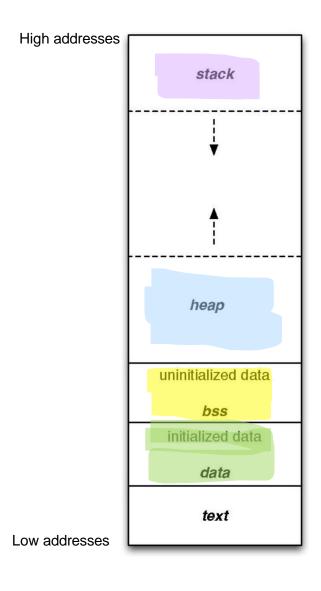
```
int x = 100;
int main()
       int a = 2;
       float b = 2.5;
       static int y;
       int *ptr = (int *) malloc(2*sizeof(int));
      ptr[0] = 5;
      ptr[1] = 6;
       free (ptr)
       return 1;
```



```
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;
}
```

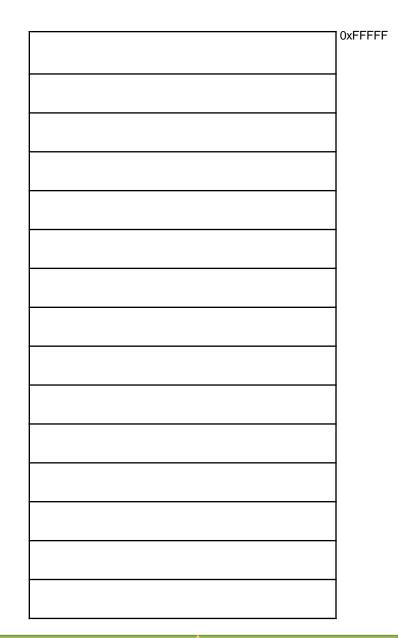
The Stack

```
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;
```

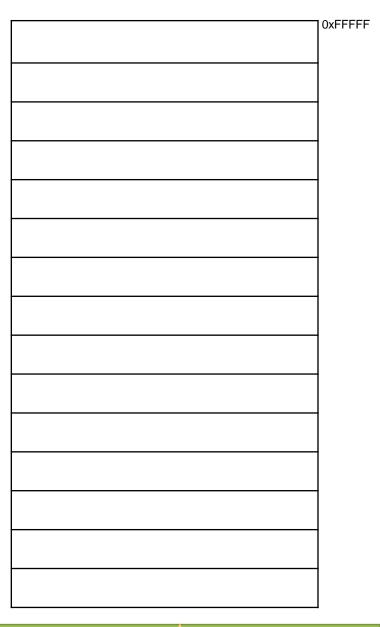
```
int foo(x,y) {
    printf(x);
    printf(y);
    int z = 1;
    foo2(z)
    return 0;
}
```

```
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	OXITITI
Previous Frame Pointer	
X = 3	
Y = 3	

int foo2(p){

printf(p);

return 0;

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

    int z = 1;

    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
```

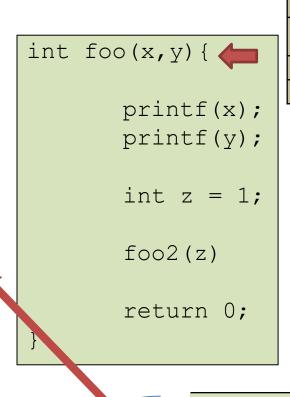
The Stack

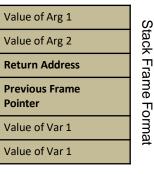
Stack frame for main()

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;
```

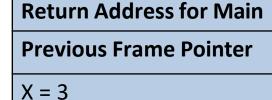




Stack Frame Format

The Stack

Stack **frame** for main()



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

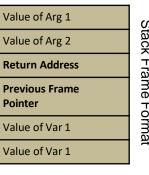
Y = 3
Return Address for foo()
Previous Frame Pointer

X = 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;
```



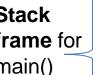
Stack Frame Format

The Stack

Return Address for Main

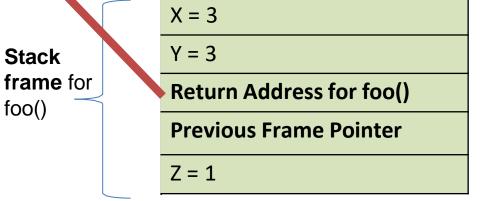
Previous Frame Pointer

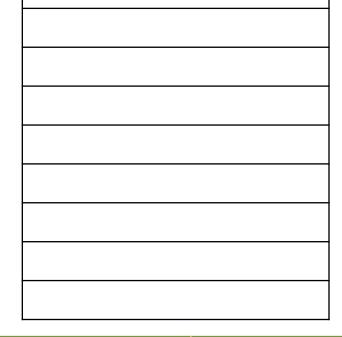
Stack **frame** for main()





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
```

```
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

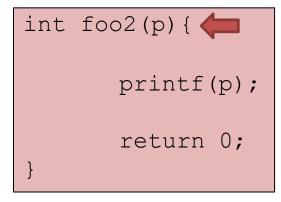
Previous Frame
Pointer

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
```



Stack frame for foo2()

p = 1

Return Address for foo2

Previous Frame Pointer

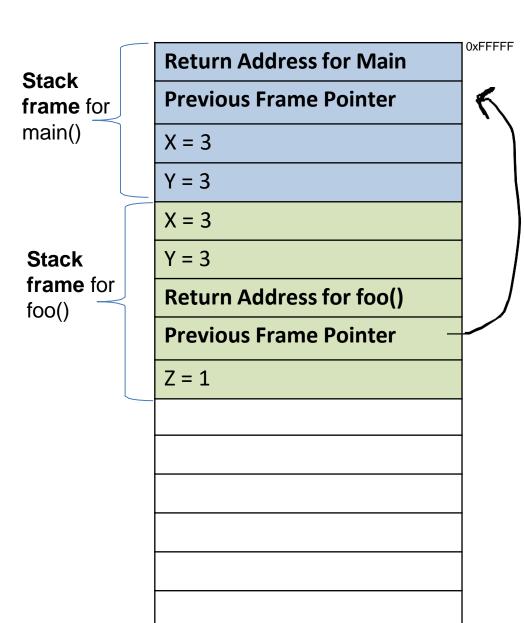
```
int main(){
       int x = 3;
       int y = 3;
       foo(x, y)
      int a = 0;
       foo2(a);
       return 0;
```

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

```
Stack Frame
                             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 **Return Address**



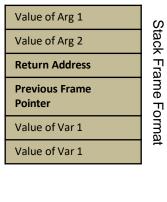
```
int main() {
    int x = 3;
    int y = 3;
    foo(x,y)
    int a = 0;
    foo2(a);
    return 0;
}
```

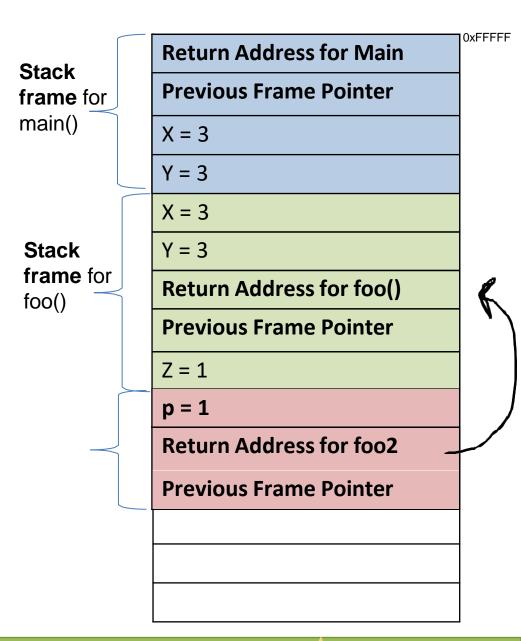
```
int foo2(p){
    printf(p);
    return 0;
}
```

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

int z = 1;

foo2(z)
    return 0;
}
```





```
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

Format

The Stack

Stack frame for main()

Stack

foo()

frame for

Return Address for Main
Previous Frame Pointer

X = 3

Y = 3

X = 3

Y = 3

Return Address for foo()

Previous Frame Pointer

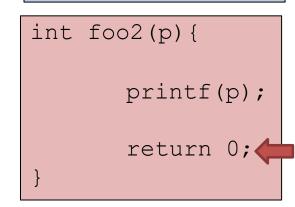
Z = 1

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

Stack **frame** for main()

Stack

foo()

frame for

```
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

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

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!

```
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;
```

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

Value of Arg 1

Value of Arg 2

Return Address

Stack

Return back to 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!

The Stack

Return Address for Main



X = 3Y = 3

X = 3

Y = 3

Return Address for foo()

Previous Frame Pointer

Z = 1

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;
}
```

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

Value of Arg 1

Value of Arg 2

Return Address

Previous Frame

Value of Var 1

Value of Var 1

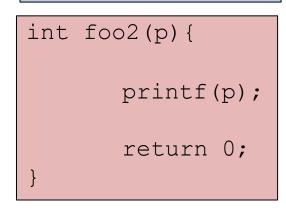
Pointer

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 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

Stack Frame

Format

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

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 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

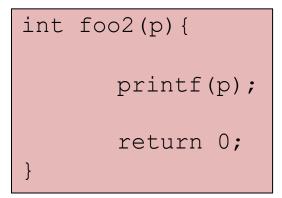
Value of Var 1

Value of Var 1
```

The Stack

Stack frame for main()

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



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
                               Stack Frame Format
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 foo(x,y) {
    printf(x);
    printf(y);

int z = 1;

foo2(z)

return 0;
}
```

The Stack

Return Address for Main

```
Stack
frame for
main()

X = 3

Y = 3

a = 0

p = 0

Return Address for foo2

Previous Frame Pointer

Previous Frame Pointer
```

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

return 0;
}
```

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

Stack Frame

Format

```
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 Return Address for Ma

```
Return Address for Main
Format
     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()
```

```
Surn back
```

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

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; int foo2(p){ When foo2() is

finished, it will return back

to main()

printf(p);

return 0;

```
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

Stack frame for main()

Stack Frame

Format

```
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()

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;
}
```

Program done!

Value of Arg 1

Value of Arg 2

Return Address

Previous Frame

Value of Var 1

Value of Var 1

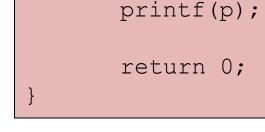
Pointer

The Stack

Stack frame for main()

Stack Frame Format

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



int foo2(p){

```
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!
```

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

Value of Var 1

0xFFFFF
1
1
J

```
#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;
```

0xFFFFF

```
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;
```

cstring.h>

```
#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;
```

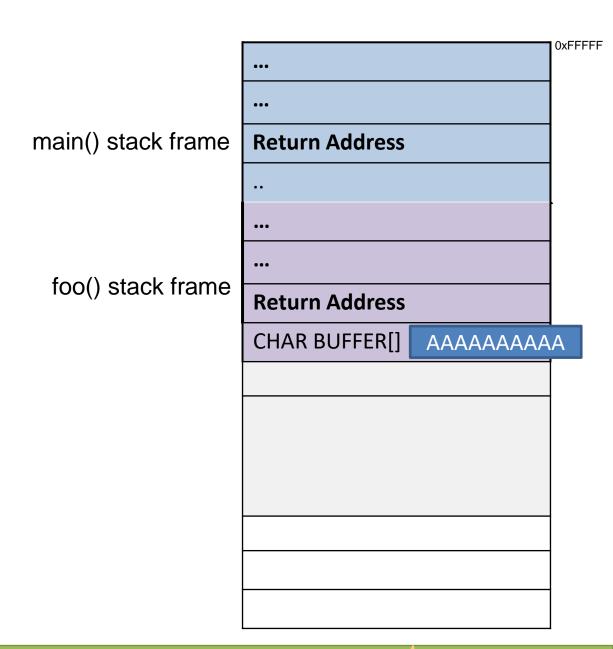
0xFFFFF main() stack frame **Return Address** foo() stack frame **Return Address** CHAR BUFFER[] AAAAAAAA

buffer[] can only hold 10 characters, right?

#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



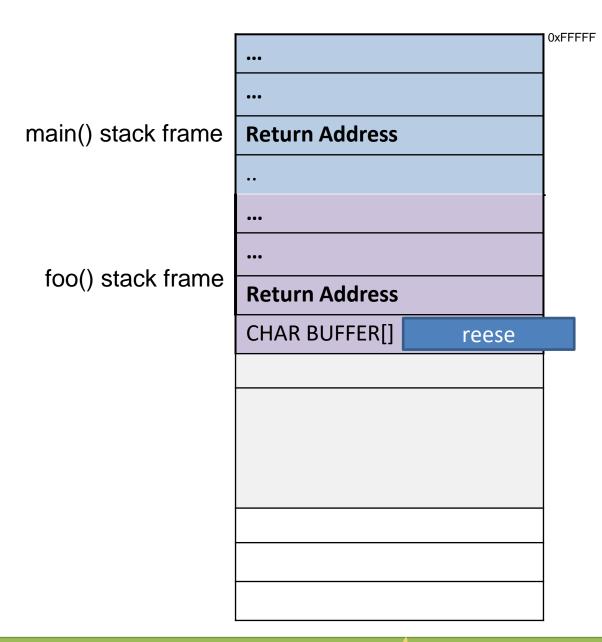
#include <string.h> #include <stdio.h> #include <stdlib.h> void foo(char *str) char buffer[10]; strcpy(buffer, str); int main(int argc, char *argv[]) foo(argv[1]); printf("Returned Properly\n"); return 0;

C doesn't care.

Instead of ./myprogram reese

What if we did.....

The Stack

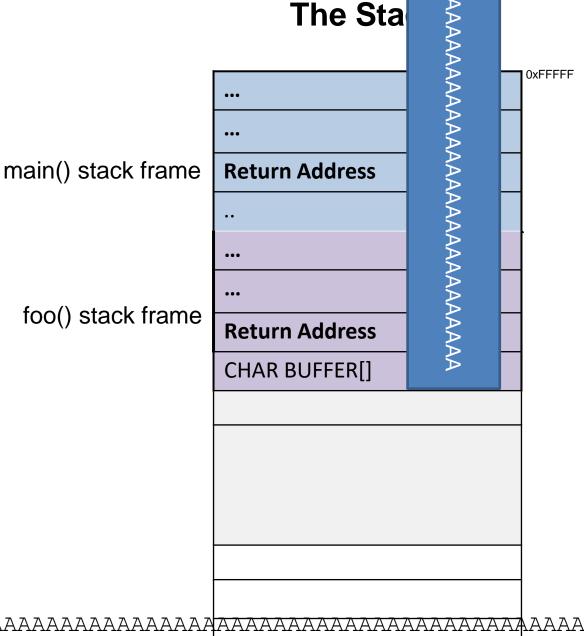


```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
void foo(char *str)
        char buffer[10];
        strcpy(buffer, str);
int main(int argc, char *argv[])
        foo(argv[1]);
        printf("Returned Properly\n");
        return 0;
```

C doesn't care.

Instead of ./myprogram reese

What if we did.....

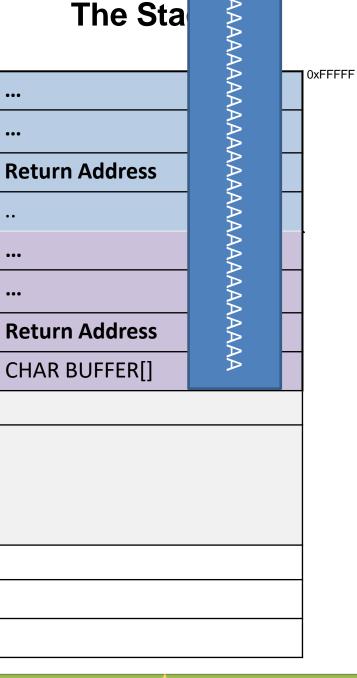


```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
void foo(char *str)
        char buffer[10];
        strcpy(buffer, str);
int main(int argc, char *argv[])
        foo(argv[1]);
        printf("Returned Properly\n");
        return 0;
```

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

Return Address Return Address CHAR BUFFER[]

The Sta

We can **overflow** this buffer!

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

What can our input control?

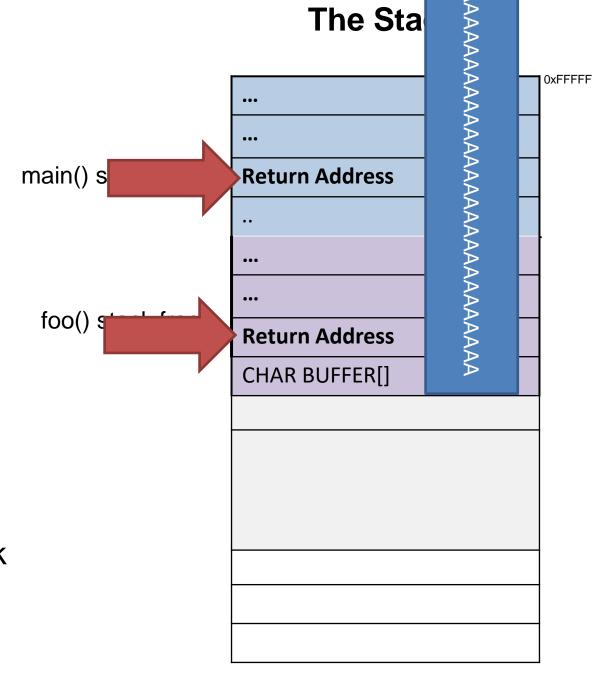
0xFFFFF

```
#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;
```

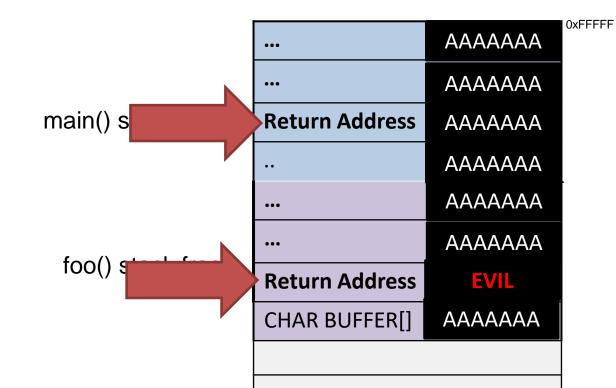
We can **overflow** this buffer!

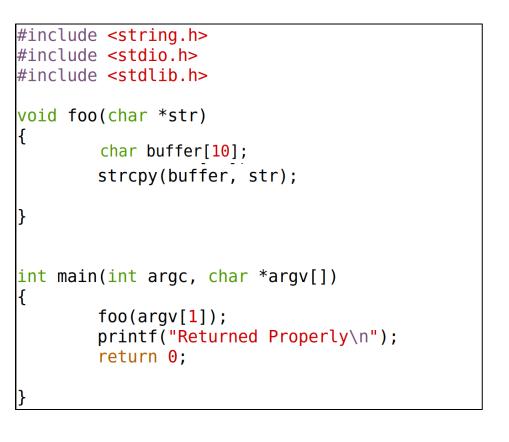
This will **overwrite** other values on the Stack

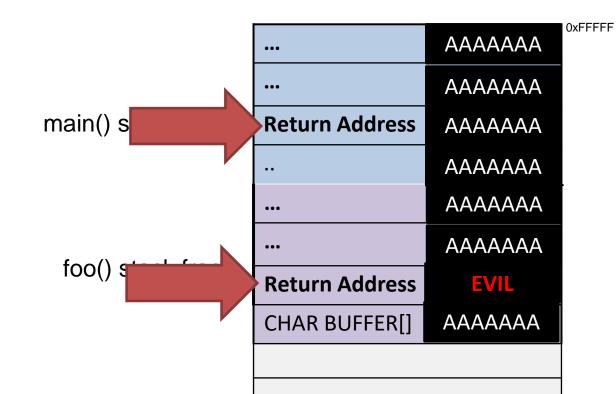
Our input can overwrite values on the stack, specifically, the **return address**



```
#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;
```

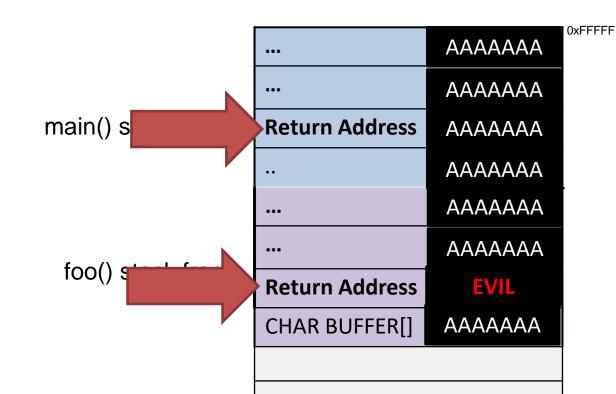






Instead of **EVIL**, what could we overwrite it with?





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]

•

٠

•

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



... 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 b
                                          push
                                                  $0x3
                                                  $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;
                                          add
                                                  %edx, %eax.
    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

```
$masp()
                                                  $0x3
                                          push
                                                              ; push b
                                                  $0x2
                                          push
                                                              ; push a
    foo(2,3);
                                                  .... <foo> ; push RA
                                          call
    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

```
void main()
                                                  $0x3
                                                              ; push b
                                          push
                                                  $0x2
                                          push
                                                              ; push a
    f$.esp3);
                                                  .... <foo> ; push RA
                                          call
    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;
                                          add
                                                  %edx, %eax.
    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
                                                    $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()
                                                    $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

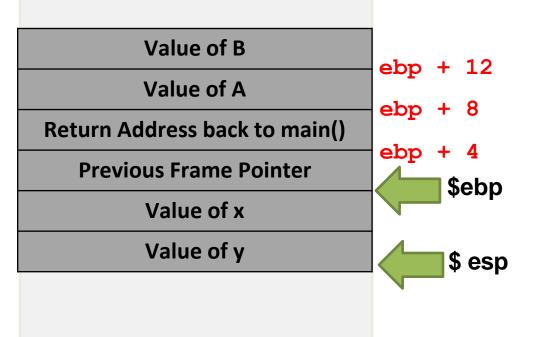
Value of y

```
void main()
                                           push
                                                  $0x3
                                                              ; push b
                                                  $0x2
                                           push
                                                              ; push a
    foo(2,3);
                                           call
                                                  .... <foo> ; push RA
    return 0;
                                           . . .
    $ebp
                                                  %ebp
                                           push
                                                                    ; 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{leax} = 0x8(ext{lebp}) ; 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

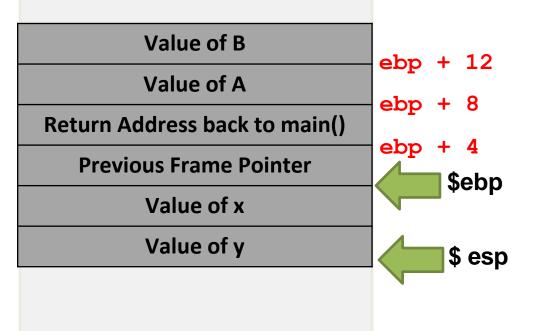
Why is this helpful knowledge?

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

