CS 201P Computer Security Winter 2020

# The buffer overflow attack: Exploring the stack and the attack strategy

7 February 2020

Aftab Hussain
University of California, Irvine

# the playfield

S

H

BSS

DS

TS

# buffer

S

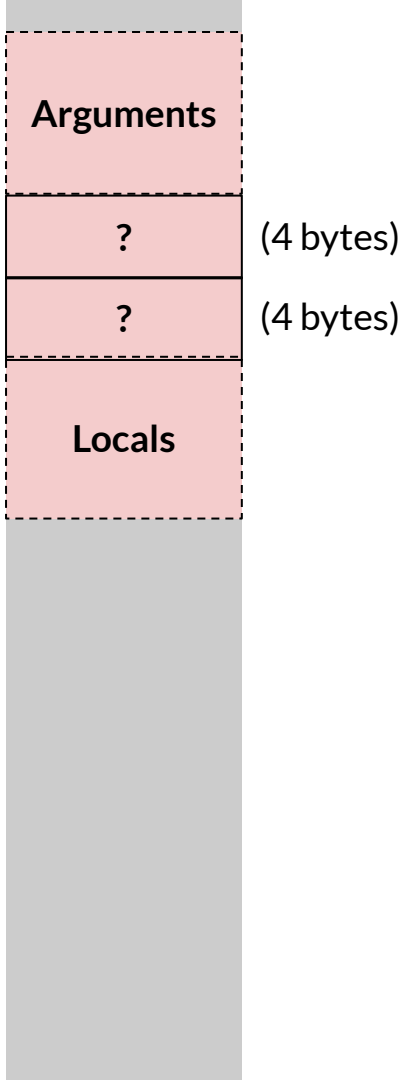
H

BSS

DS

TS

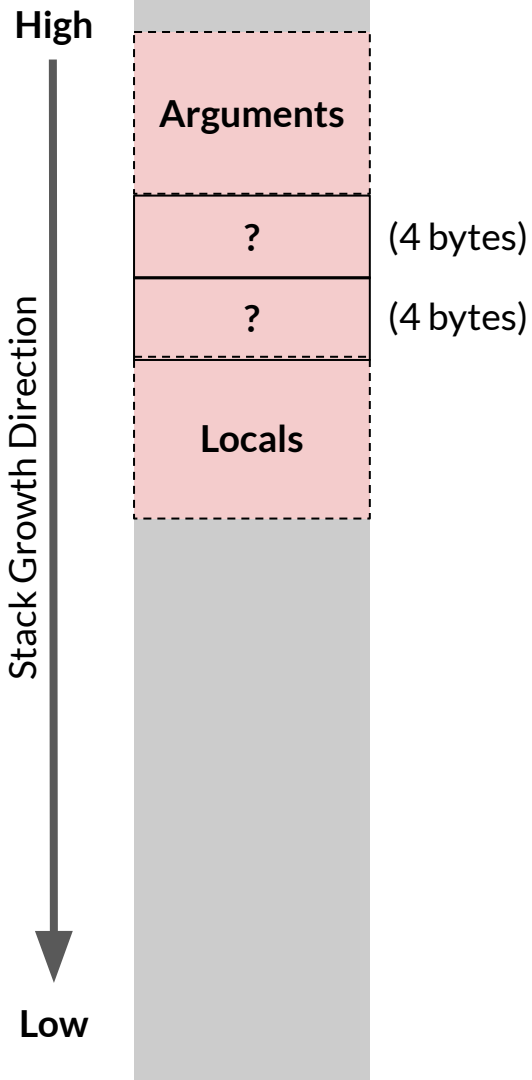
# buffer




# stack

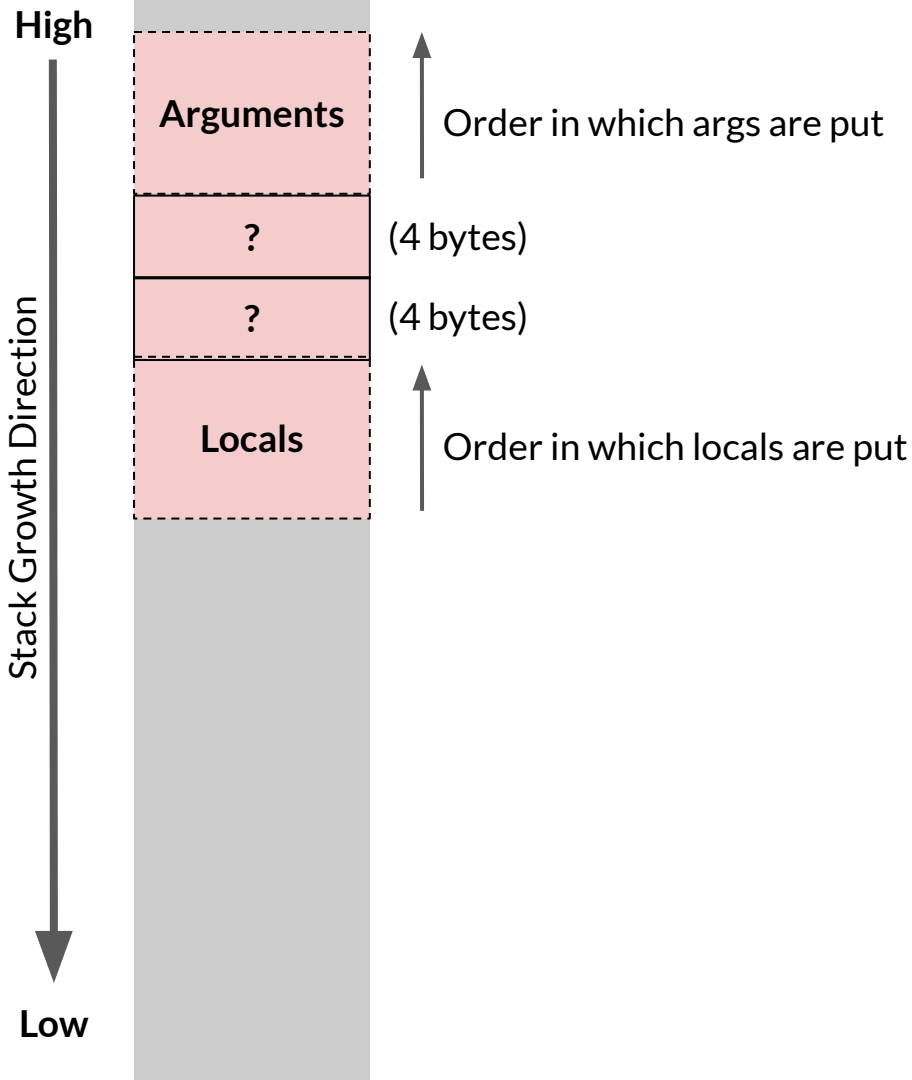
(32-bit architecture)

*after any function invocation*




# stack

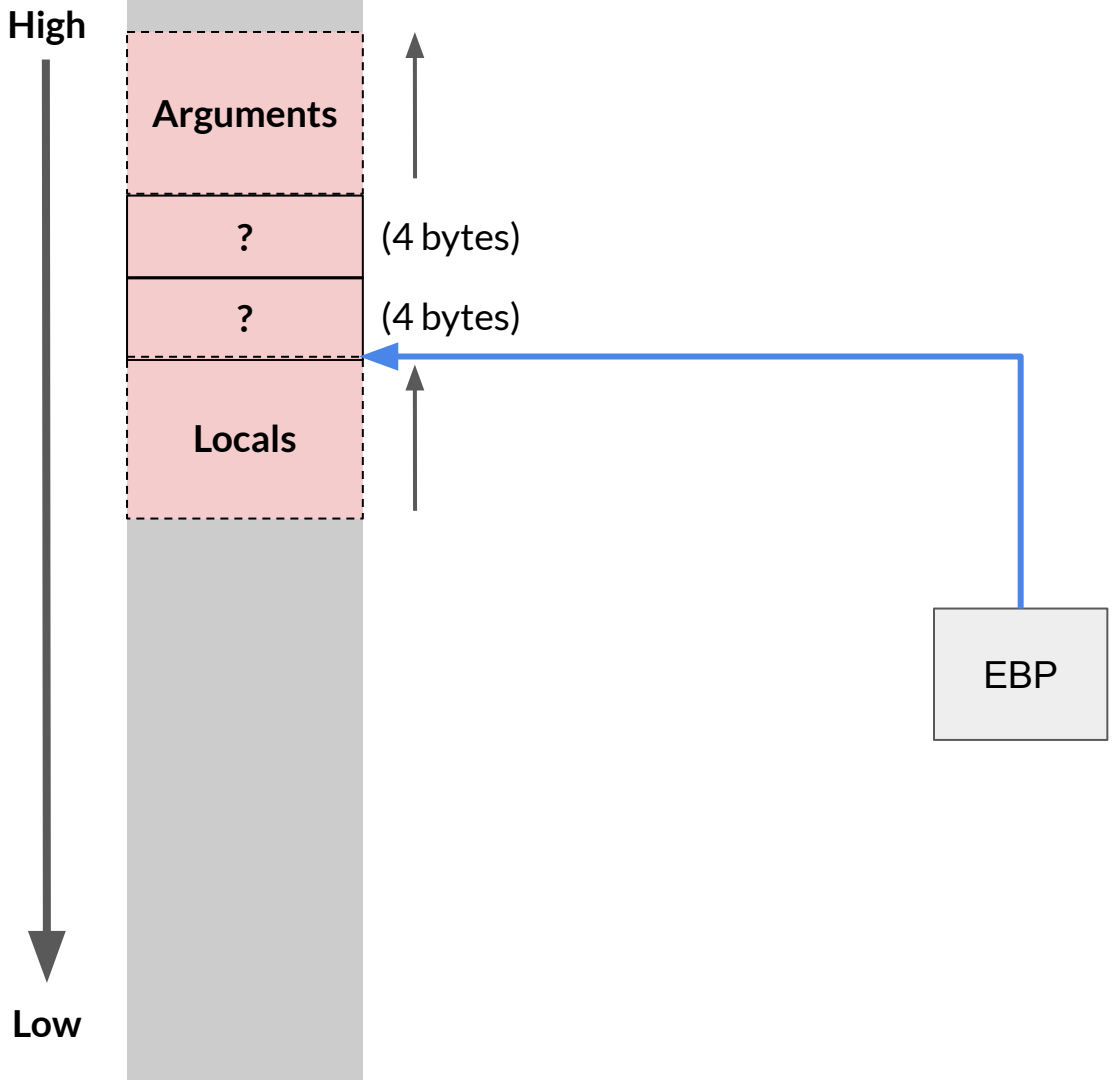
(32-bit architecture)




# stack

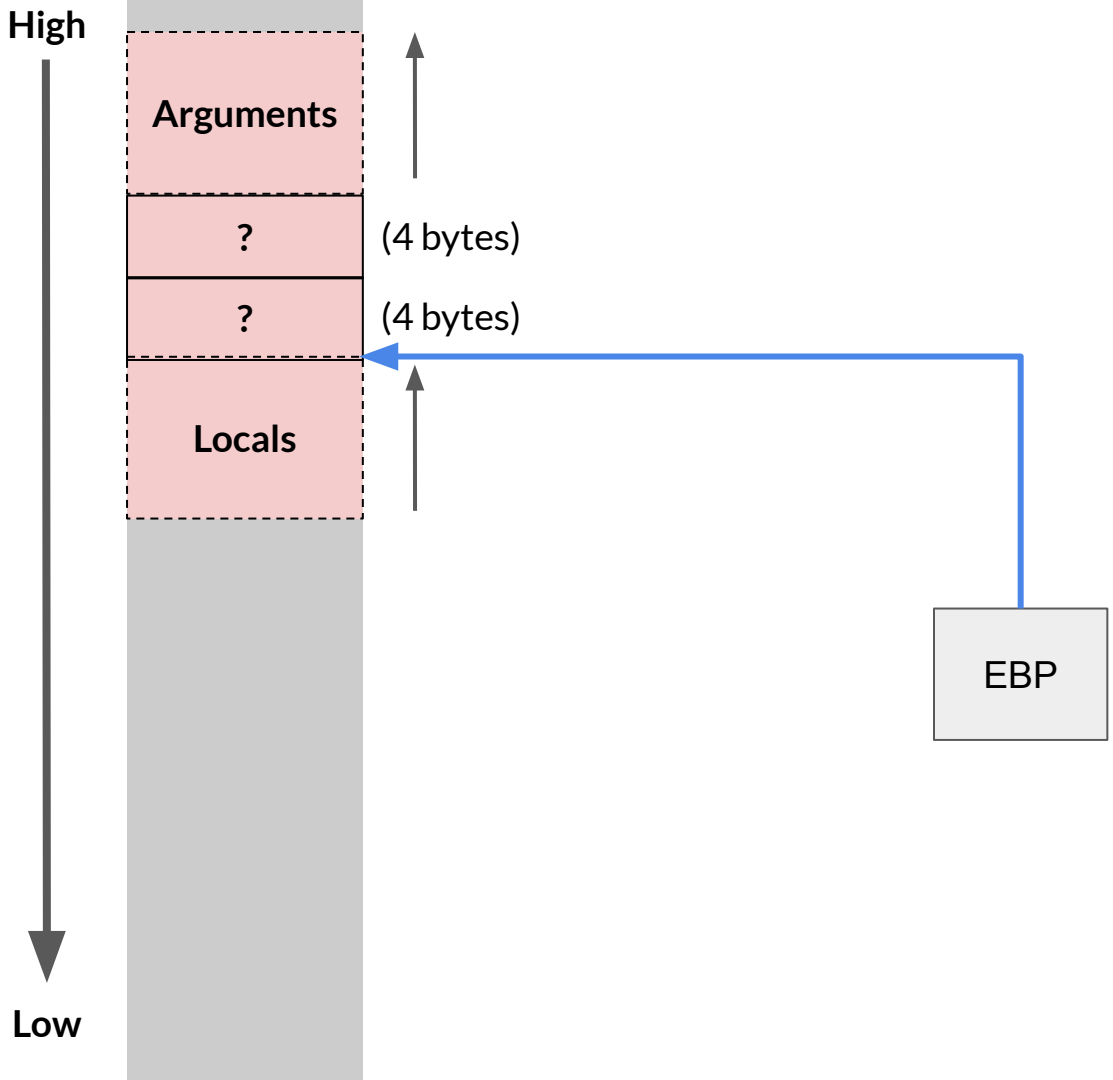
(32-bit architecture)

*how to refer to the variables?*




# stack

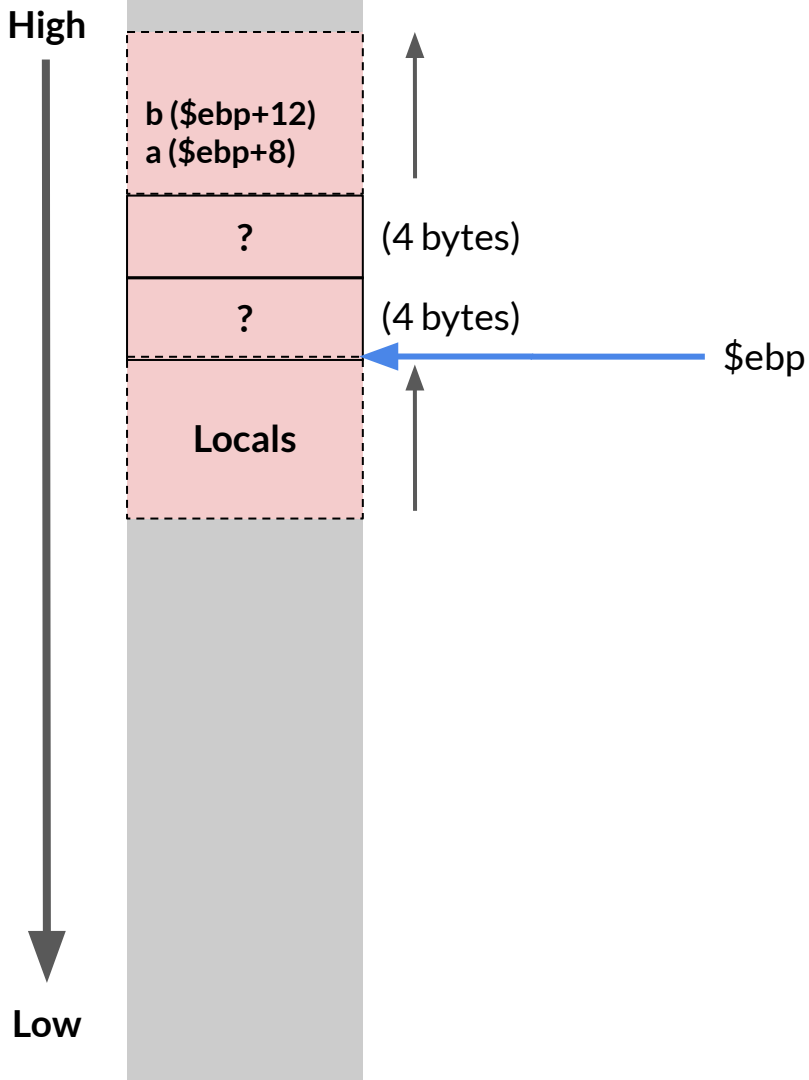
(32-bit architecture)




# stack

(32-bit architecture)

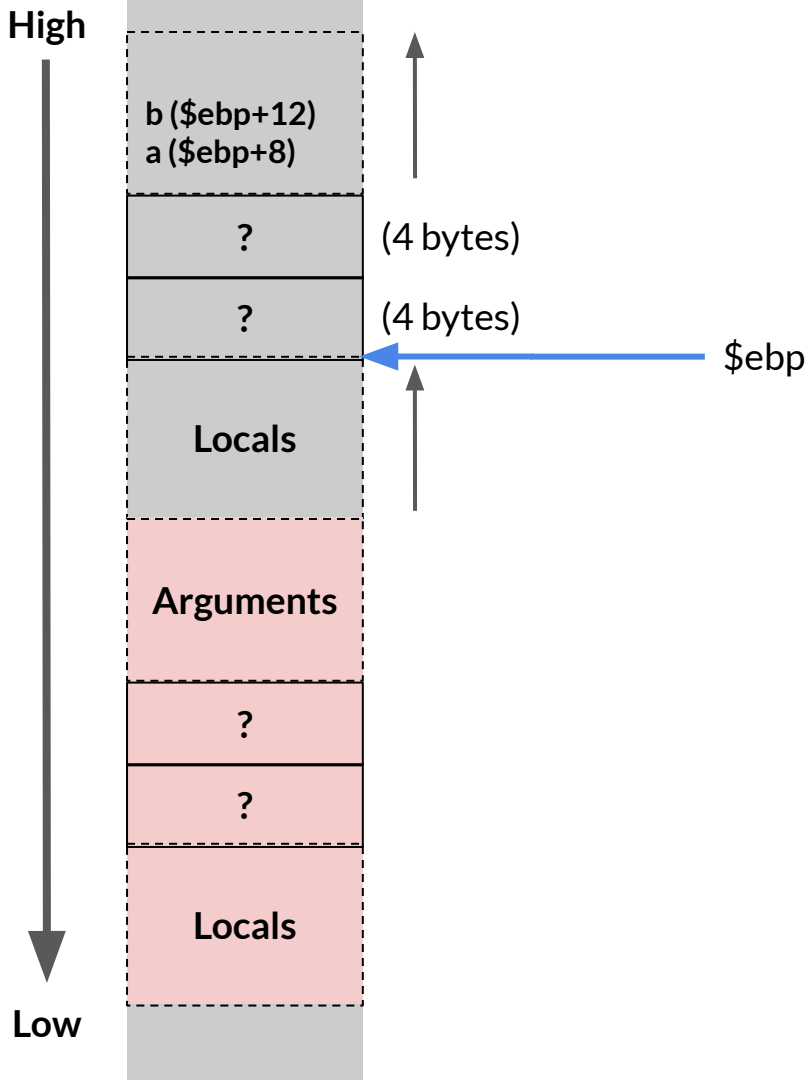
```
int main(int a, int b){
        ...
```




# stack

(32-bit architecture)

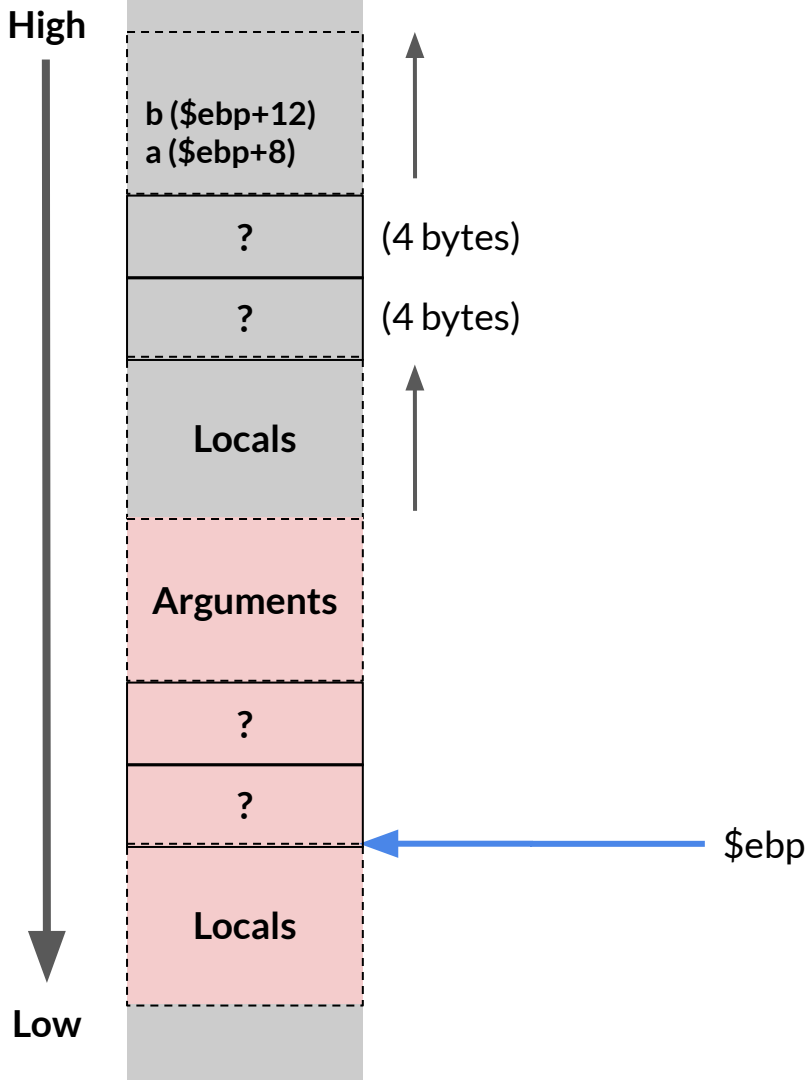
*our function calls another function*




# stack

(32-bit architecture)

*our function calls another function*

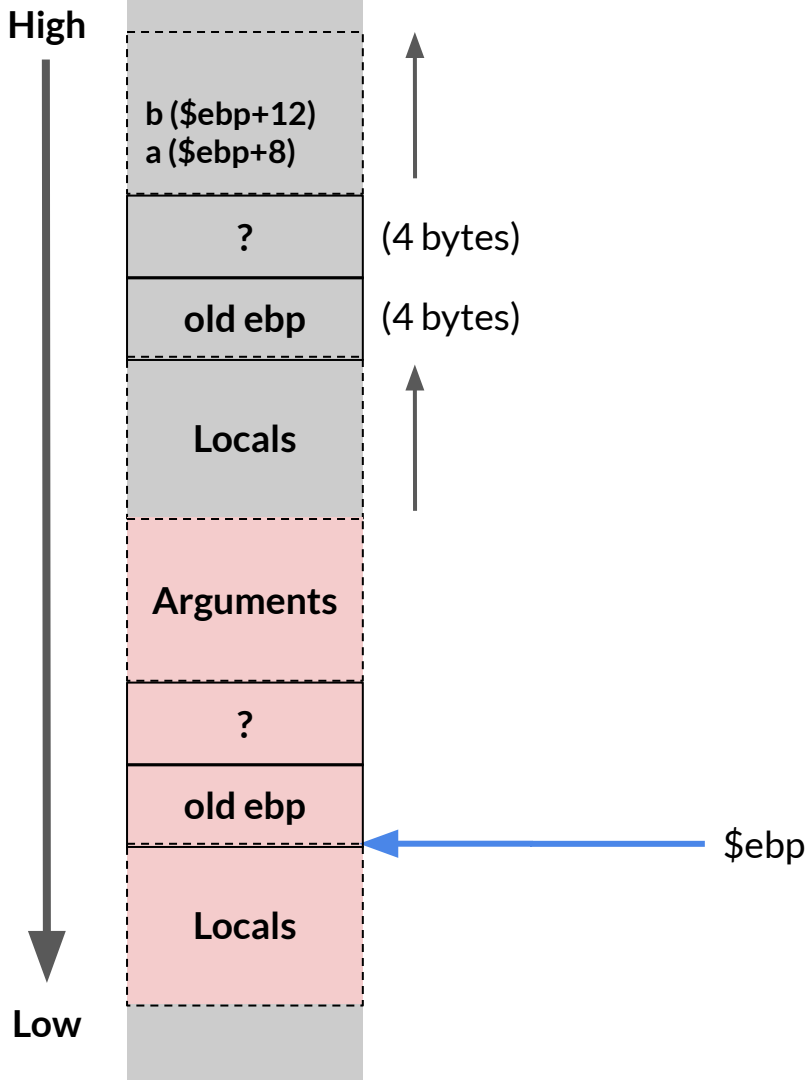



# stack

(32-bit architecture)

*our function calls another function*

*The old $ebp value?*



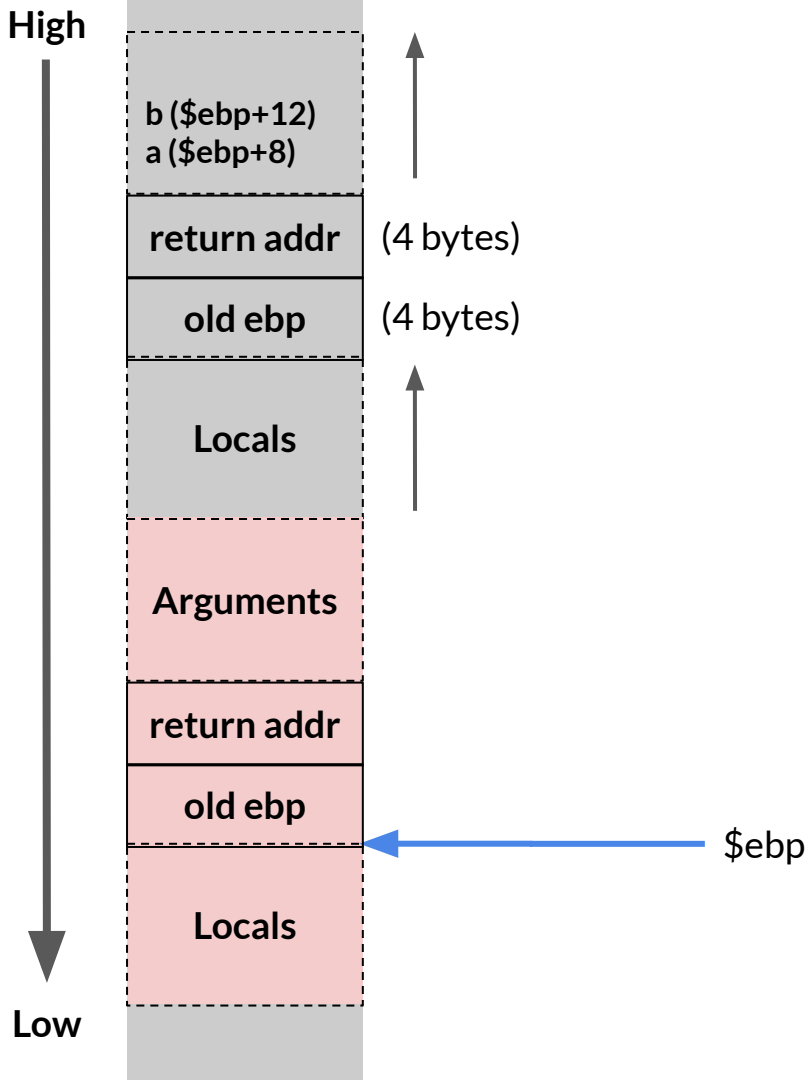

# stack

(32-bit architecture)

*our function calls another function*

*Which instruction do we execute after we are done with the current function?*

*The instruction right after the instruction that called the function.*

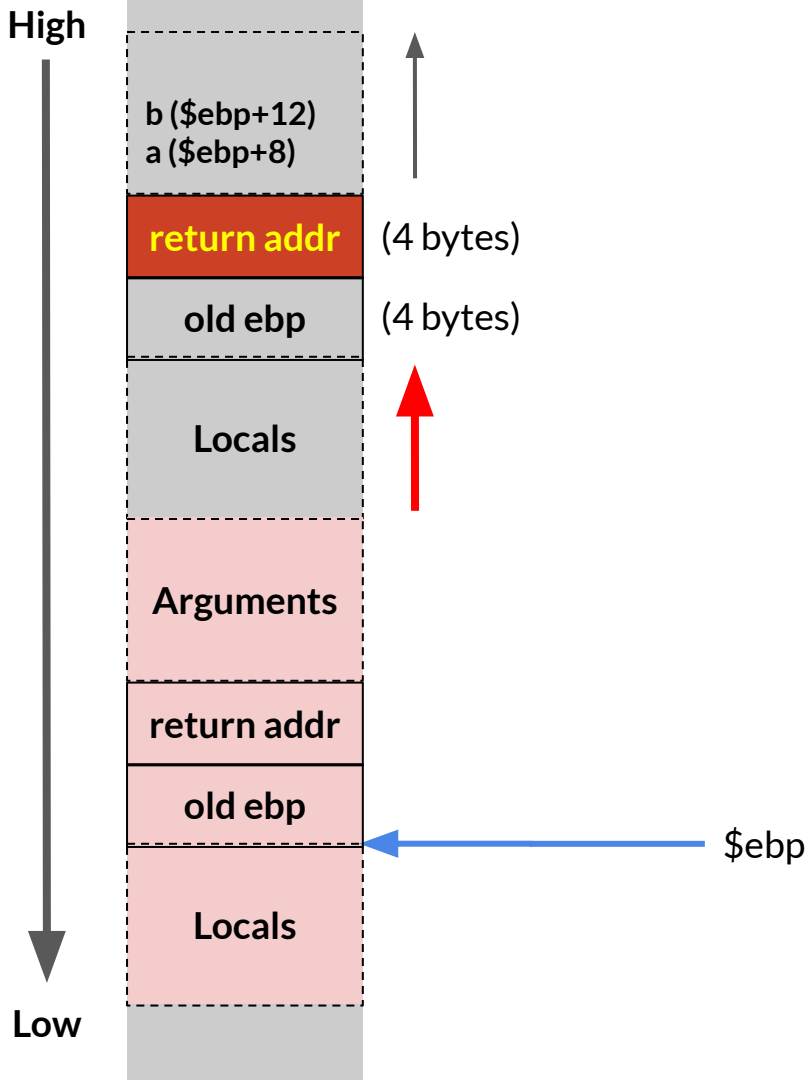



# stack

(32-bit architecture)

*our function calls another function*

*So what's interesting?*




# stack

(32-bit architecture)

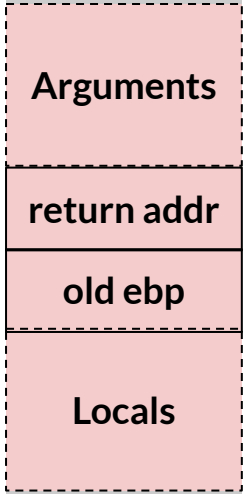
*our function calls another function*

*If we can make return address point to our code…*

*But we need to be careful of what we put in return addresss…*

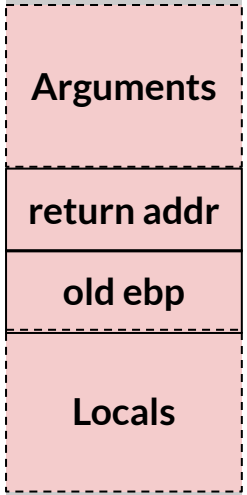
# our strategy

# taking aim
# (address calculation)



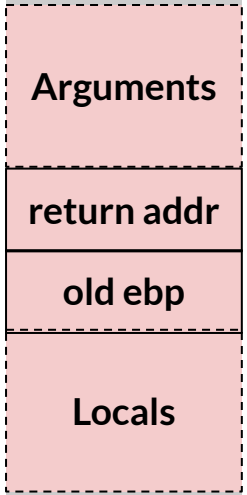

**Vulnerable Program**

```
int bof(char *str)
{
  char buffer[24];
  strcpy(buffer, str);
  return 1;
}

int main(int argc, char **argv)
{
  char str[517];
  FILE *badfile;
  badfile = fopen("badfile", "r");
  fread(str, sizeof(char), 517, badfile);
  bof(str);
  printf("Returned Properly\n");
  return 1;
}
```



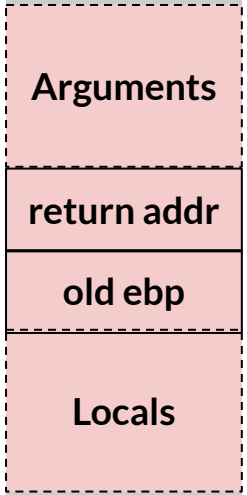

**Vulnerable Program**

```
int bof(char *str)
{
  char buffer[24];
  strcpy(buffer, str);
  return 1;
}

int main(int argc, char **argv)
{
  char str[517];
  FILE *badfile;
  badfile = fopen("badfile", "r");
  fread(str, sizeof(char), 517, badfile);
  bof(str);
  printf("Returned Properly\n");
  return 1;
}
```




**Vulnerable Program**

```
int bof(char *str)
{
  char buffer[24];
  strcpy(buffer, str);    //loads your exploit code to the memory
  return 1;
}

int main(int argc, char **argv)
{
  char str[517];
  FILE *badfile;
  badfile = fopen("badfile", "r");
  fread(str, sizeof(char), 517, badfile);
  bof(str);
  printf("Returned Properly\n");
  return 1;
}
```



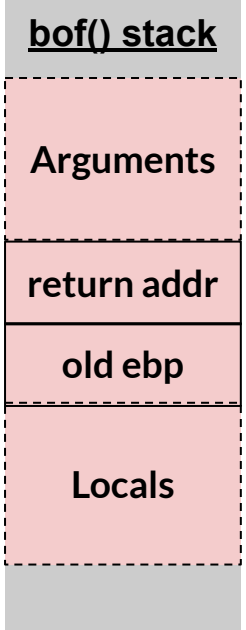

**Vulnerable Program**

```
int bof(char *str)
{
  char buffer[24];
  strcpy(buffer, str);    //loads your exploit code to the memory
  return 1;               //just need to set up return address of this fn
}                         //after we are done with the strcpy instruction

int main(int argc, char **argv)
{
  char str[517];
  FILE *badfile;
  badfile = fopen("badfile", "r");
  fread(str, sizeof(char), 517, badfile);
  bof(str);
  printf("Returned Properly\n");
  return 1;
}
```

$gcc -z execstack -fno-stack-protector -g -o stack_dbg stack.c
$touch badfile

$gdb -q stack_dbg

**Enter GDB**

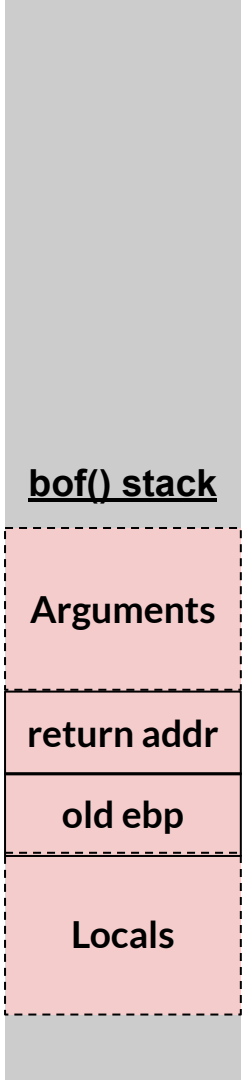


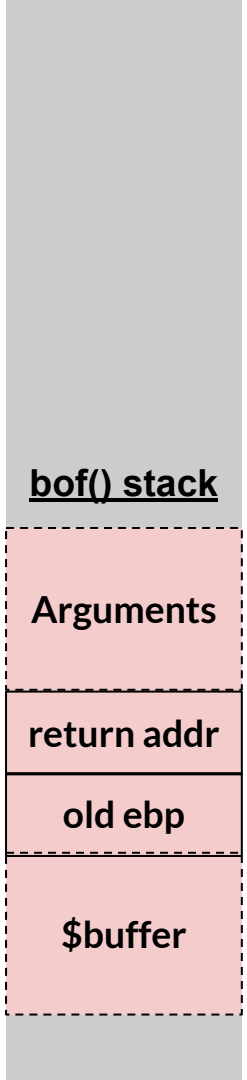
$gcc -z execstack -fno-stack-protector -g -o stack_dbg stack.c
$touch badfile

$gdb -q stack_dbg

**Enter GDB**

break at bof() function
gdb$ b bof()
gdb$ run






$gcc -z execstack -fno-stack-protector -g -o stack_dbg stack.c
$touch badfile

$gdb -q stack_dbg

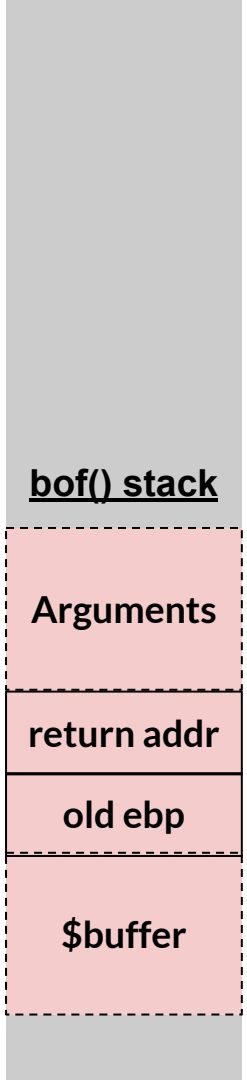
**Enter GDB**

break at bof() function
gdb$ b bof()
gdb$ run

The program will stop after bof() is entered.
Find the location of the buffer variable.

gdb$ p/x &buffer

[you get the hex address of buffer]




$gcc -z execstack -fno-stack-protector -g -o stack_dbg stack.c
$touch badfile

$gdb -q stack_dbg

**Enter GDB**

break at bof() function
gdb$ b bof()
gdb$ run

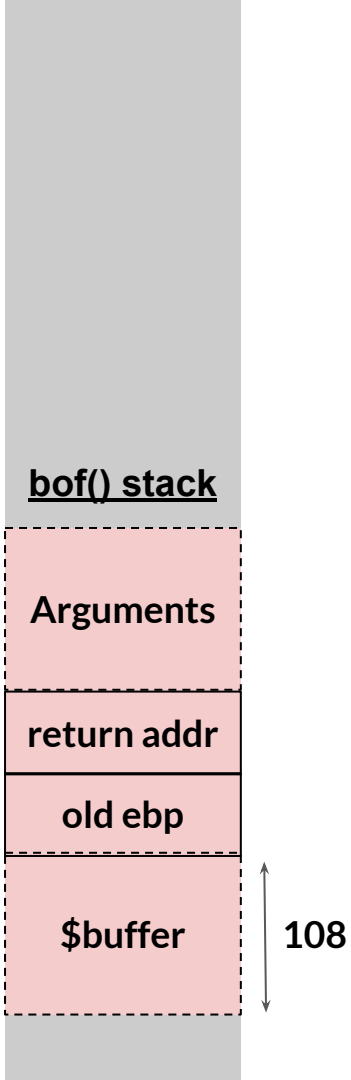
The program will stop after bof() is entered.
Find the location of the buffer variable.

gdb$ p/x &buffer

[you get the hex address of buffer]

Get address of $ebp value

gdb$ p/d addr(ebp) - addr(buffer)




$gcc -z execstack -fno-stack-protector -g -o stack_dbg stack.c
$touch badfile

$gdb -q stack_dbg

**Enter GDB**

break at bof() function
gdb$ b bof()
gdb$ run

The program will stop after bof() is entered.
Find the location of the buffer variable.

gdb$ p/x &buffer

[you get the hex address of buffer]

Get address of $ebp value

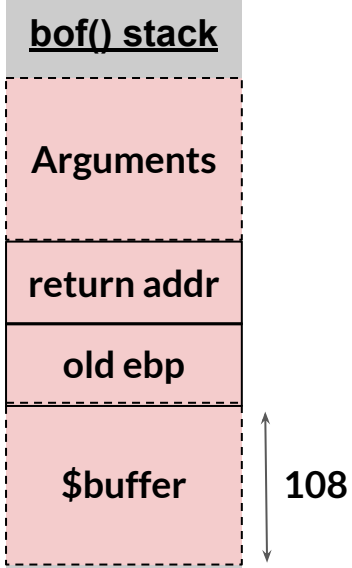
gdb$ p/d addr(ebp) - addr(buffer)

Let's say it's 108

Use the offsets you get to set up the return address value.

NOTE:

Regarding the content of the return address, as discussed, it would be $ebp + [some value, like 12]. Make sure the resulting hexadecimal address from the sum does not contain any sequence like “00”: This sequence would resemble a null byte. When using strcpy, the copy process would stop on encountering the null byte, causing the attack to fail.



Thank you

Ref. Computer Security: A Hands on Approach
Wenliang Du