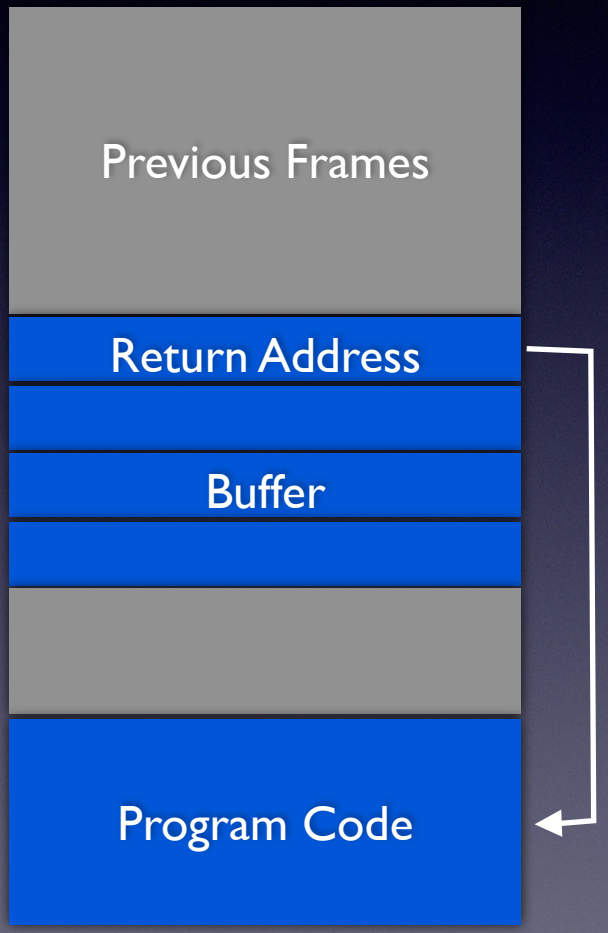
Stack-Based Buffer Overflow Overview

CMSC 426/626 - Computer Security
Fall 2014

The Stack

- Grows downward
- Organized in *frames* - each frame is associated with an active procedure call
- Frame includes *return address* and storage for parameters and local variables



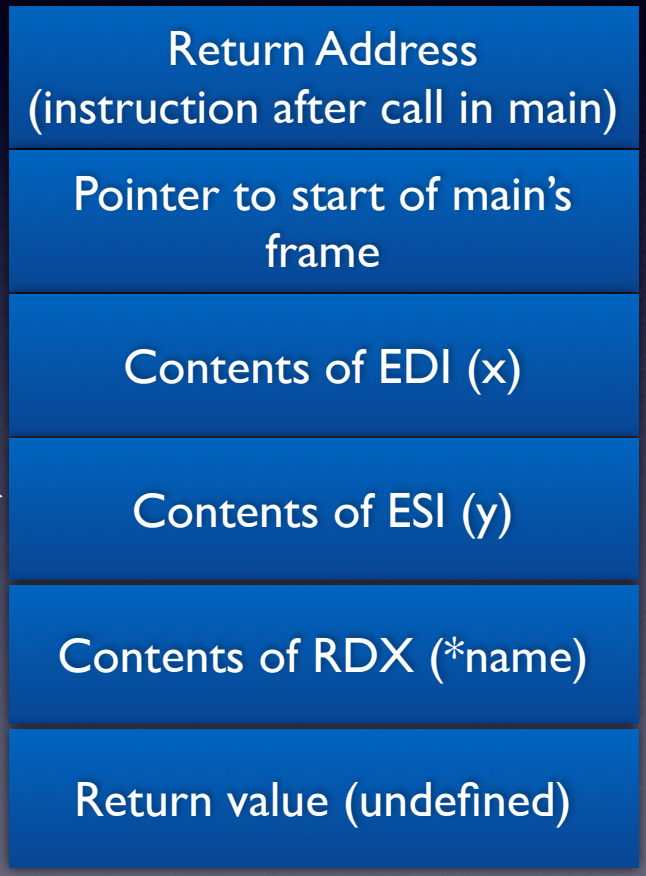
Stack Example

```
int my_func(int x, int y, char* name)
{
  int xx, yy, zz;
  float ans;
}
main()
{
  char name[6] = "chris";
  my_func(1, 2, name);
}
```

Stack Example

`x` in EDI, `y` in ESI, `*name` in RDX

```
_my_func:
    pushq   %rbp
 movq %rsp, %rbp
 movl %edi, -4(%rbp)
 movl %esi, -8(%rbp)
 movq %rdx, -16(%rbp)
 movl -20(%rbp), %eax
 popq %rbp
 ret
```

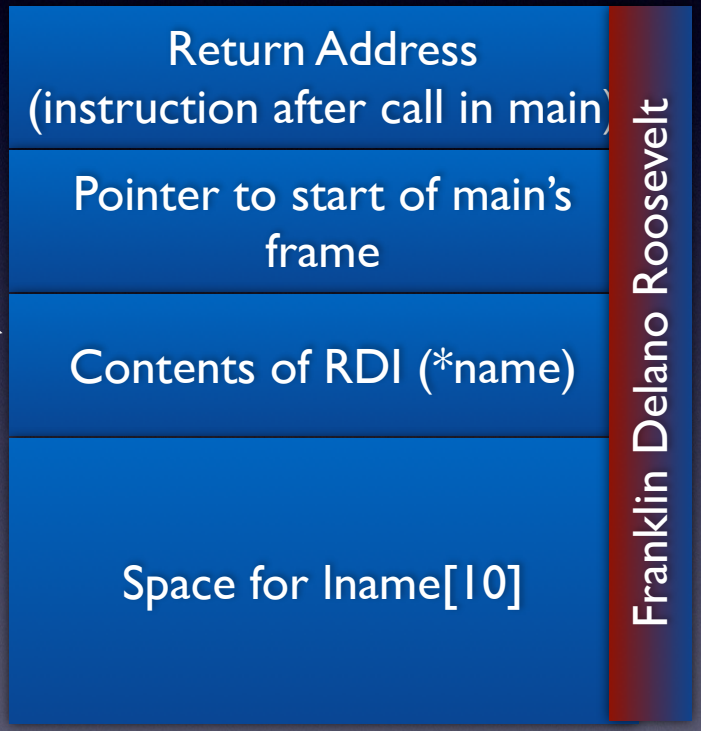


What will go wrong?

```
#include <string.h>
int my_func(char *name)
{
  char lname[10];
  strcpy(lname, name);
  return(0);
}
main()
{
  char name[50] = "franklin delano roosevelt";
  my_func(name);
}
```

The Assembly

```
_my_func:
 pushq%rbp
 movq  %rsp, %rbp
 subq  $48, %rsp
 movq  %rdi, -8(%rbp)
 leaq  -34(%rbp), %rax
 movq  -8(%rbp), %rcx
 movabsq $10, %rdx
 movq  %rax, %rdi
 movq  %rcx, %rsi
 callq___strcpy_chk
    ...code omitted...
 ret
```



Stack Smashing

- Use the buffer overflow to change execution flow
- Clever construction of overflow to replace return address with address of malicious code
- When function returns, will jump to malicious code

More on Buffer Overflows Later!