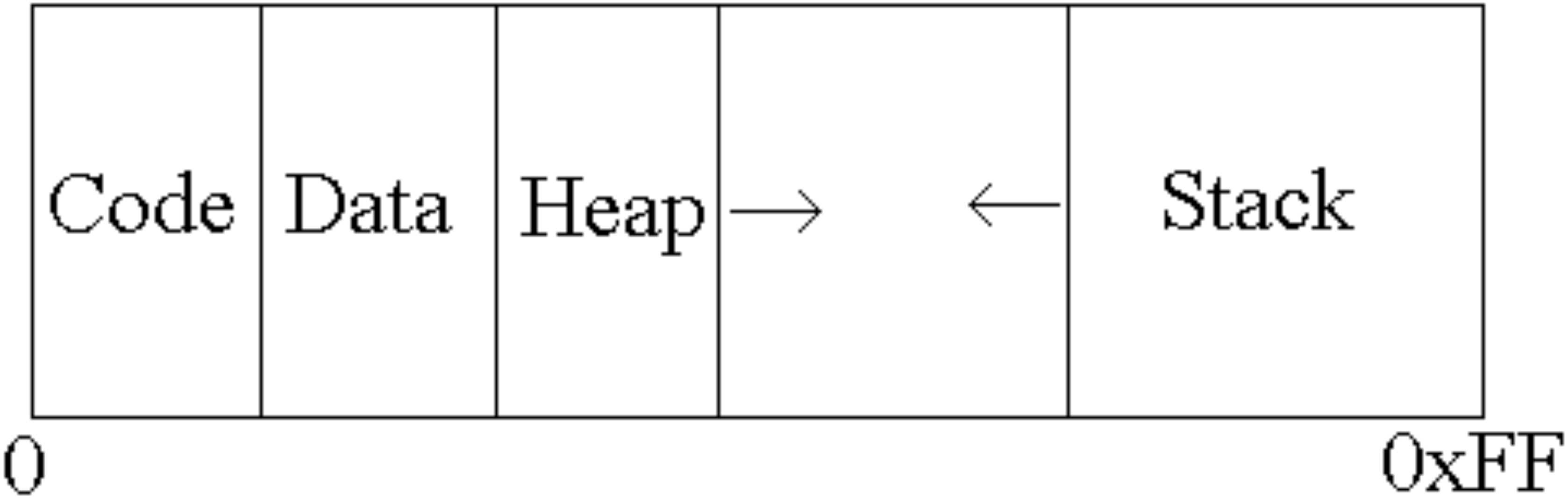
Stack Smashing, Part 1

CS4440/7440

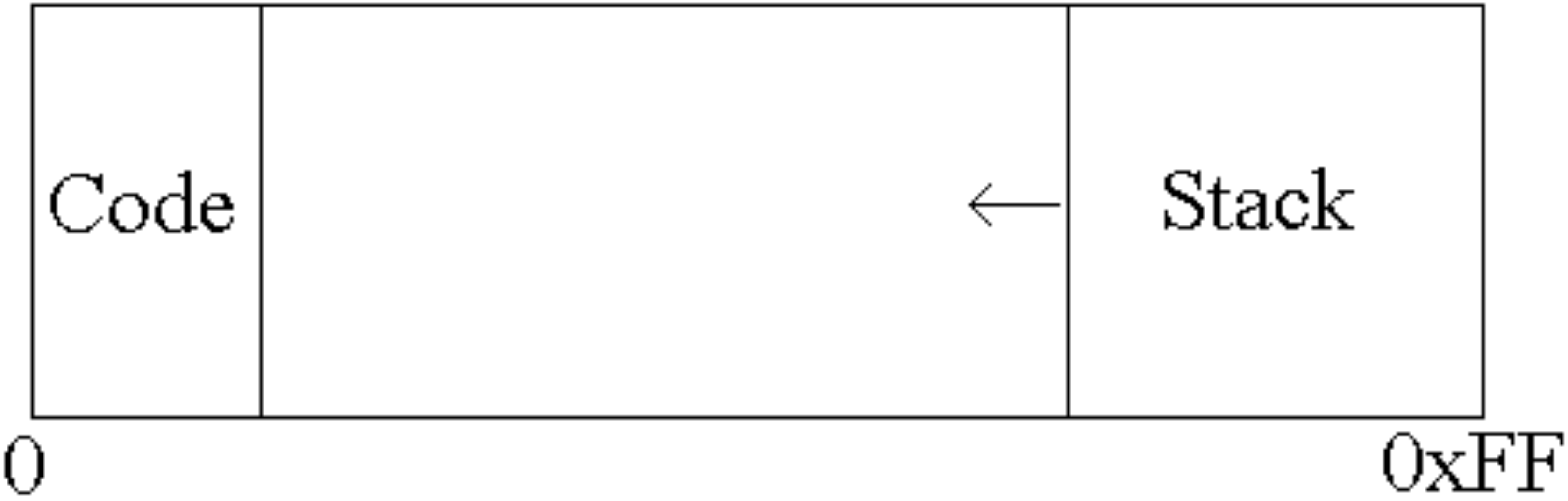
Smashing the Stack for Fun and Profit

- Review: Process memory organization
- The problem: Buffer overflows
- How to exploit the problem
- Implementing the Exploit
- Challenge

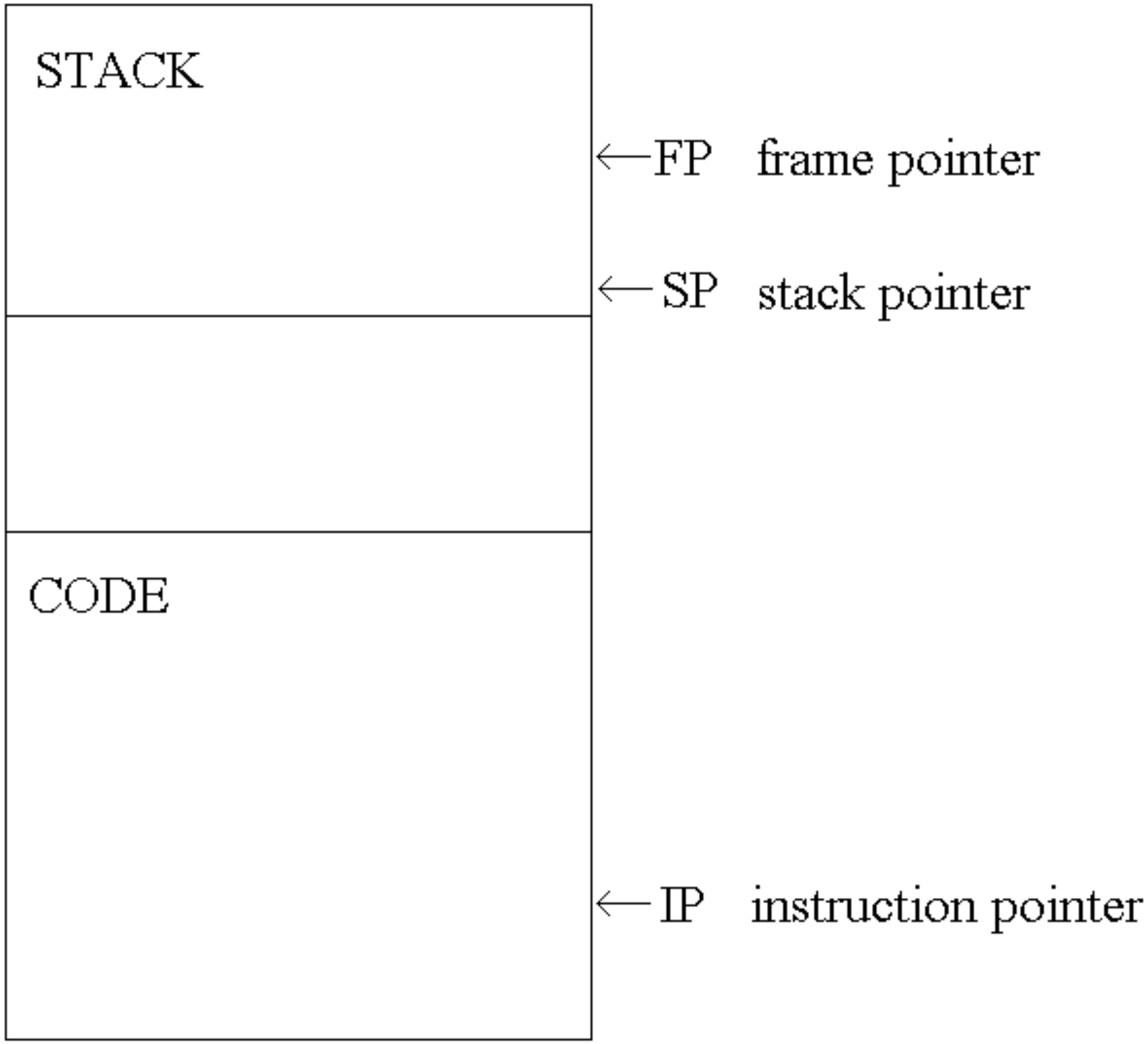
Process Memory Organization



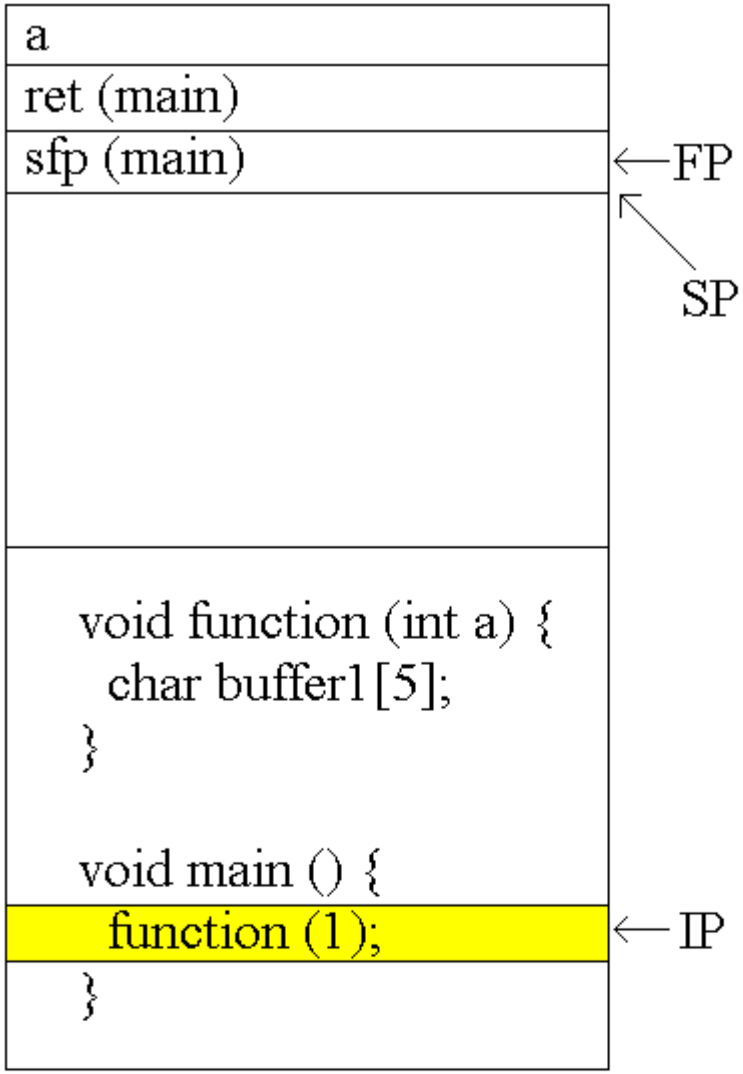
Process Memory Organization



Process Memory Organization

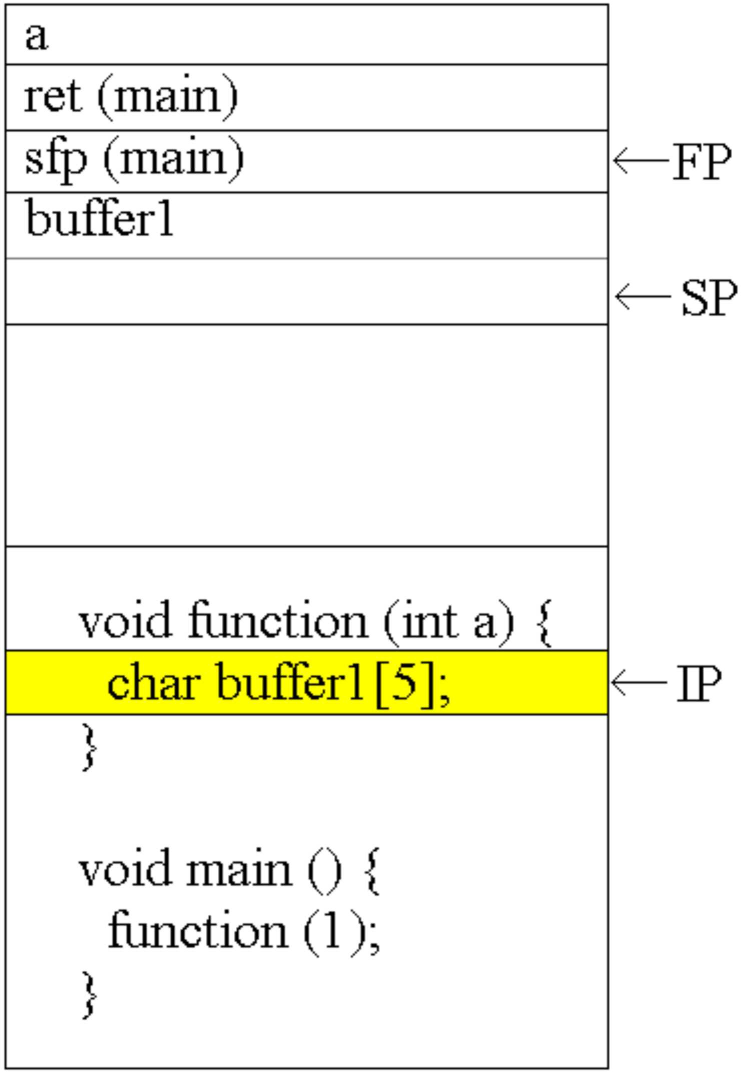


Function Calls



```
void function (int a) {
  char buffer1[5];
}

void main () {
  function (1);
}
```

Function Calls



a

ret (main)

sfp (main) ←FP

buffer1

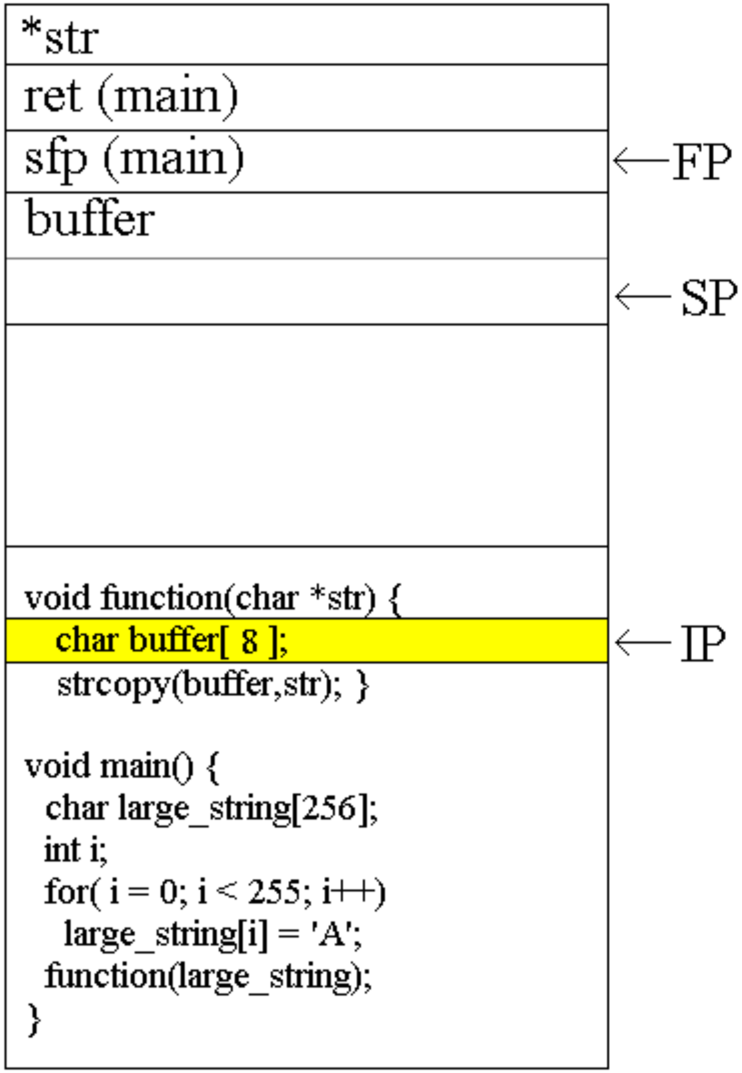
←SP

```
void function (int a) {
  char buffer1[5];    ←IP
}

void main () {
  function (1);
}
```

Buffer Overflows

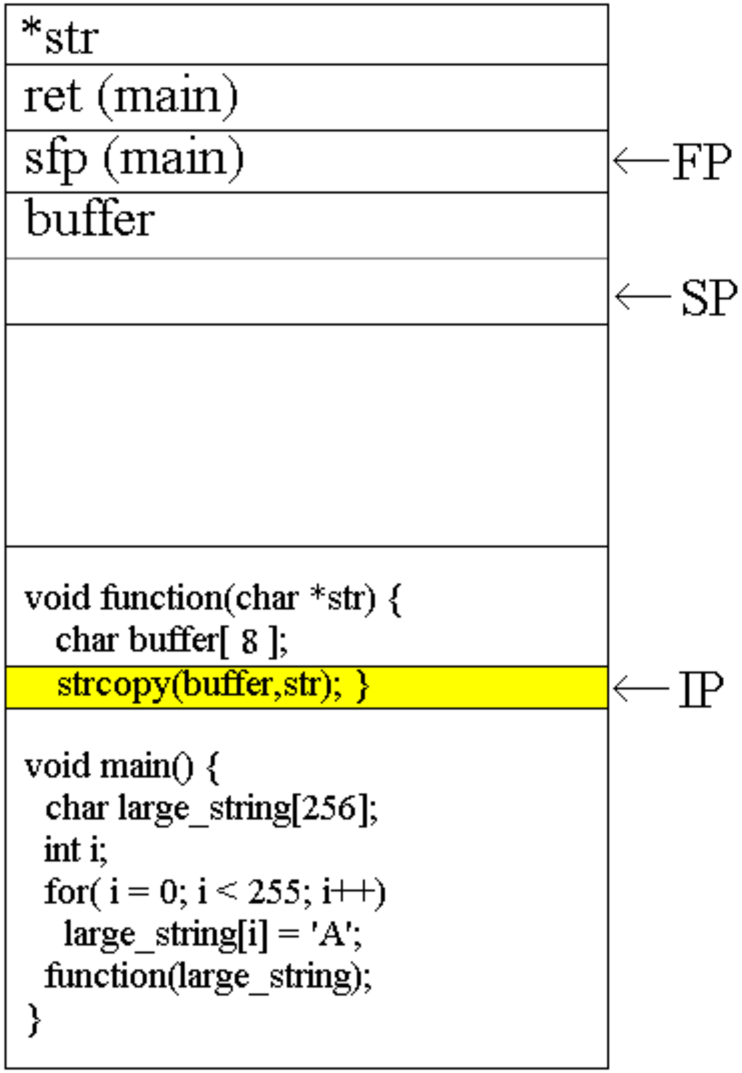
```
void function(char *str) {
   char buffer[8];
   strcpy(buffer,str); }

void main() {
   char large_string[256];
   int i;
   for( i = 0; i < 255; i++)
      large_string[i] = 'A';
 function(large_string); }
```

Buffer Overflows



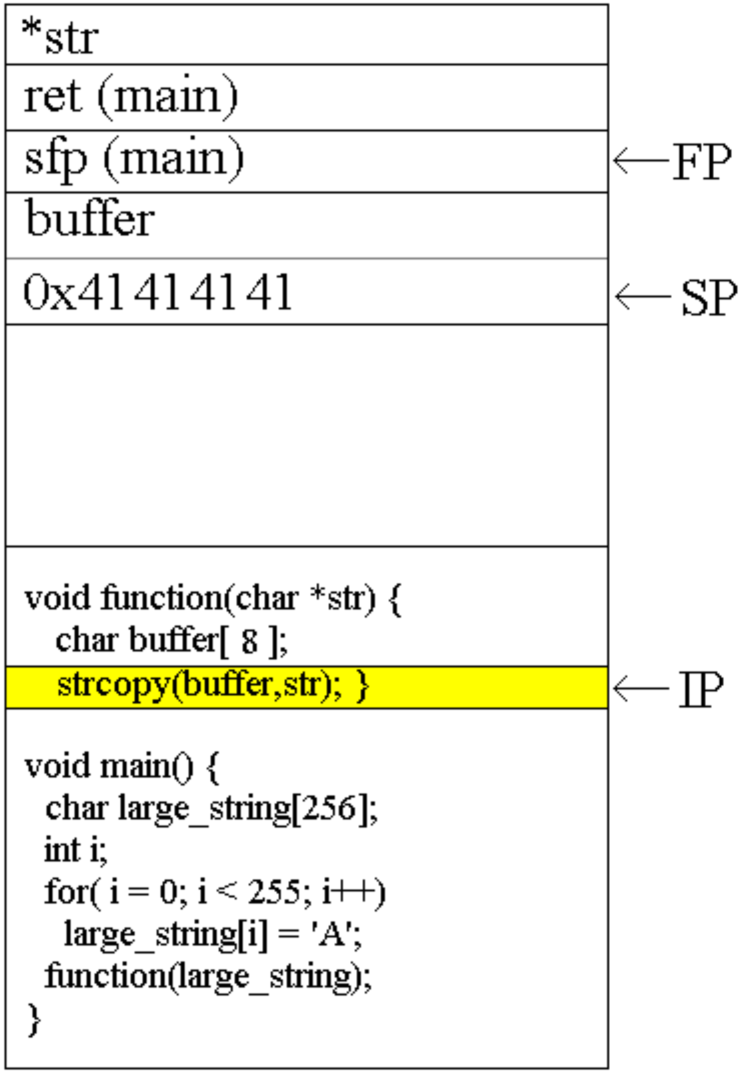
```
void function(char *str) {
   char buffer[ 8 ];
   strcopy(buffer,str); }

void main() {
  char large_string[256];
  int i;
  for( i = 0; i < 255; i++)
    large_string[i] = 'A';
  function(large_string);
}
```

Buffer Overflows



```
void function(char *str) {
   char buffer[ 8 ];
   strcopy(buffer,str); }

void main() {
  char large_string[256];
  int i;
  for( i = 0; i < 255; i++)
    large_string[i] = 'A';
  function(large_string);
}
```

Buffer Overflows



*str

ret (main)

sfp (main) ← FP

buffer

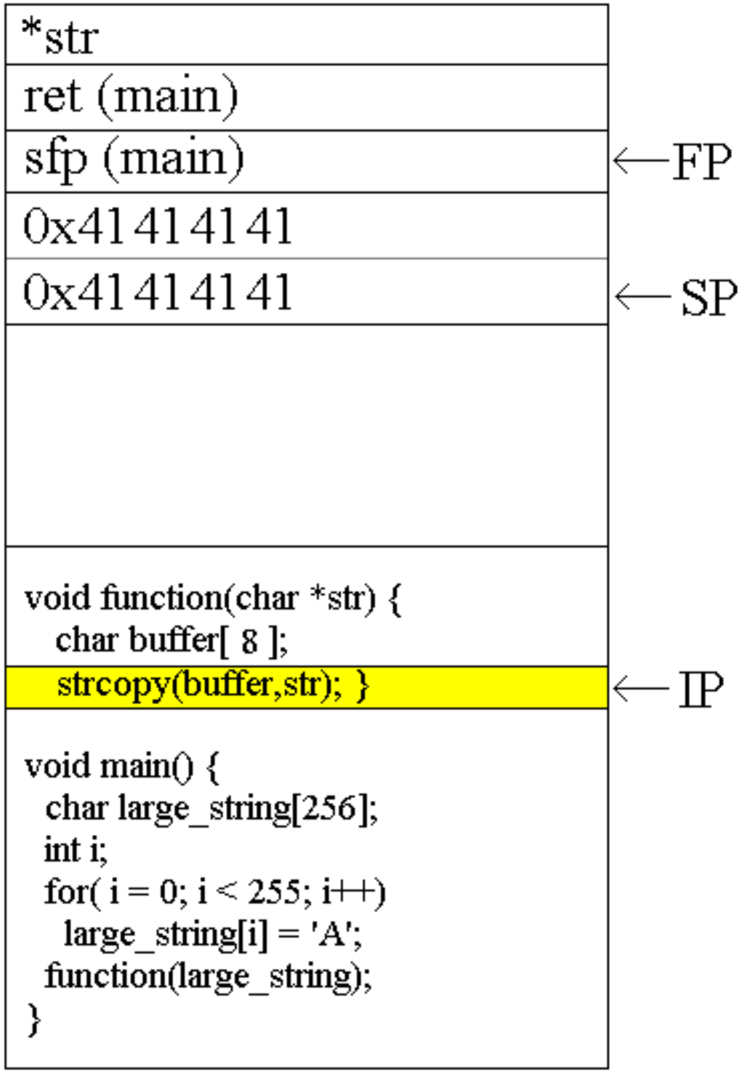
0x41414141 ← SP

```
void function(char *str) {
   char buffer[ 8 ];
   strcopy(buffer,str); }
```

← IP

```
void main() {
  char large_string[256];
  int i;
  for( i = 0; i < 255; i++)
    large_string[i] = 'A';
  function(large_string);
}
```

Buffer Overflows



*str

ret (main)

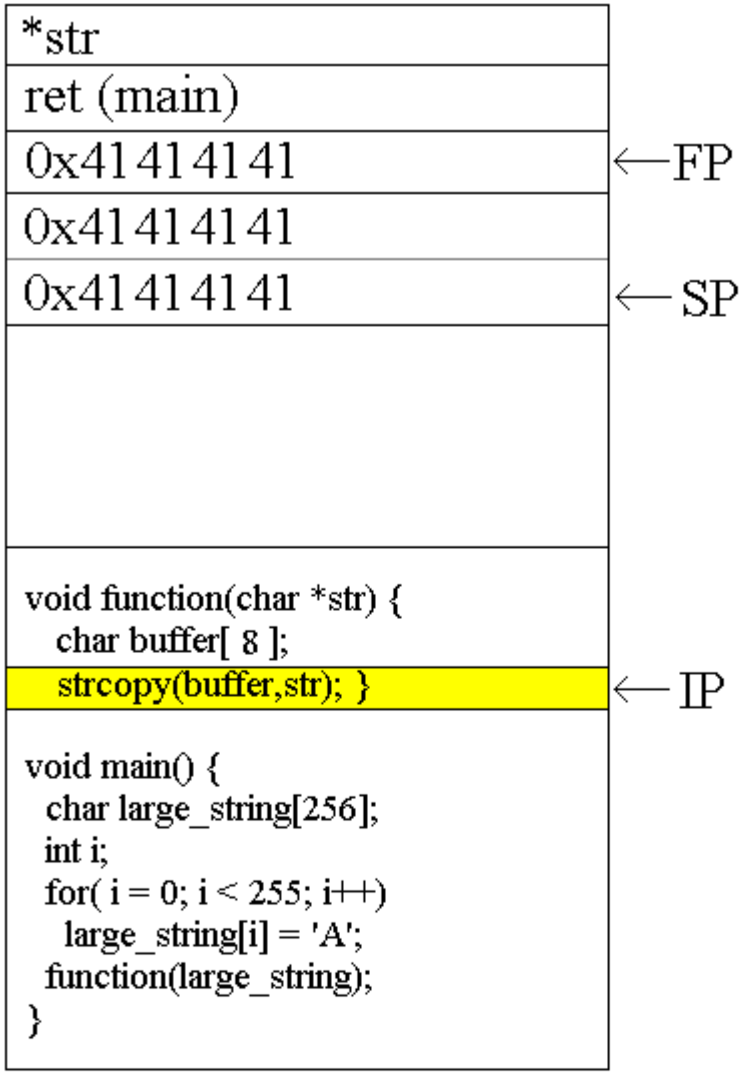
sfp (main)

0x41414141

0x41414141

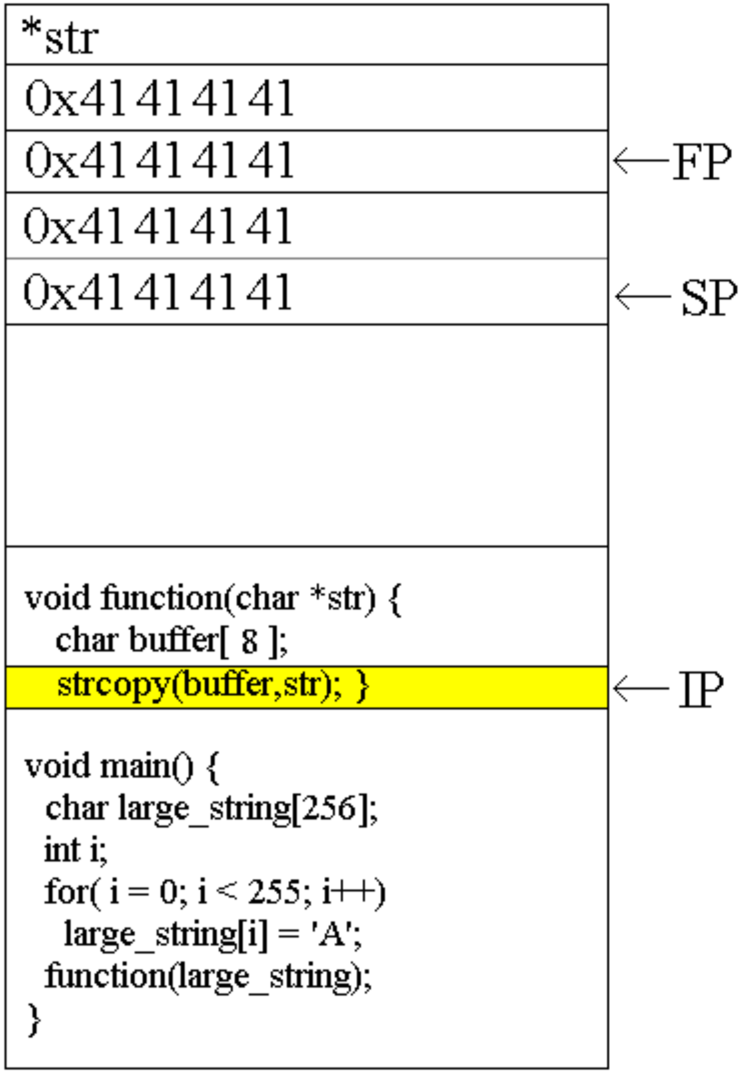
```
void function(char *str) {
  char buffer[ 8 ];
  strcopy(buffer,str); }

void main() {
 char large_string[256];
 int i;
 for( i = 0; i < 255; i++)
   large_string[i] = 'A';
 function(large_string);
}
```

Buffer Overflows



```
void function(char *str) {
   char buffer[ 8 ];
   strcopy(buffer,str); }

void main() {
  char large_string[256];
  int i;
  for( i = 0; i < 255; i++)
    large_string[i] = 'A';
  function(large_string);
}
```

Buffer Overflows



*str

0x41414141

0x41414141 ← FP

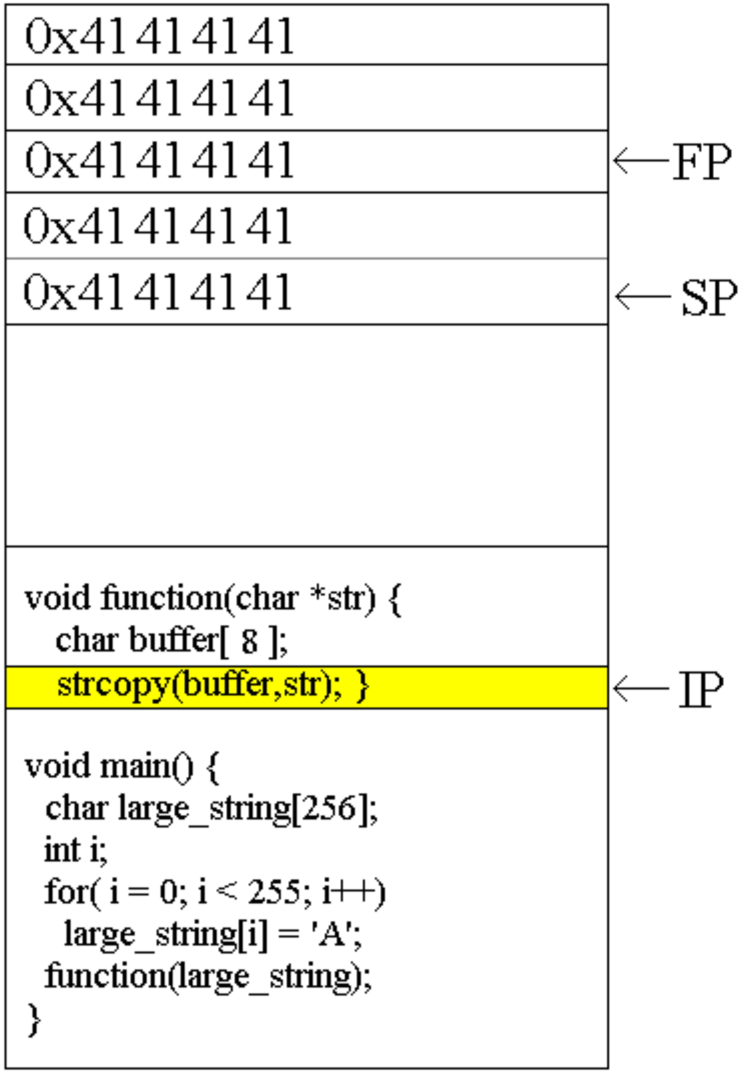
0x41414141

0x41414141 ← SP

```
void function(char *str) {
   char buffer[ 8 ];
   strcopy(buffer,str); }

void main() {
  char large_string[256];
  int i;
  for( i = 0; i < 255; i++)
    large_string[i] = 'A';
  function(large_string);
}
```

← IP

Buffer Overflows



0x41414141

0x41414141

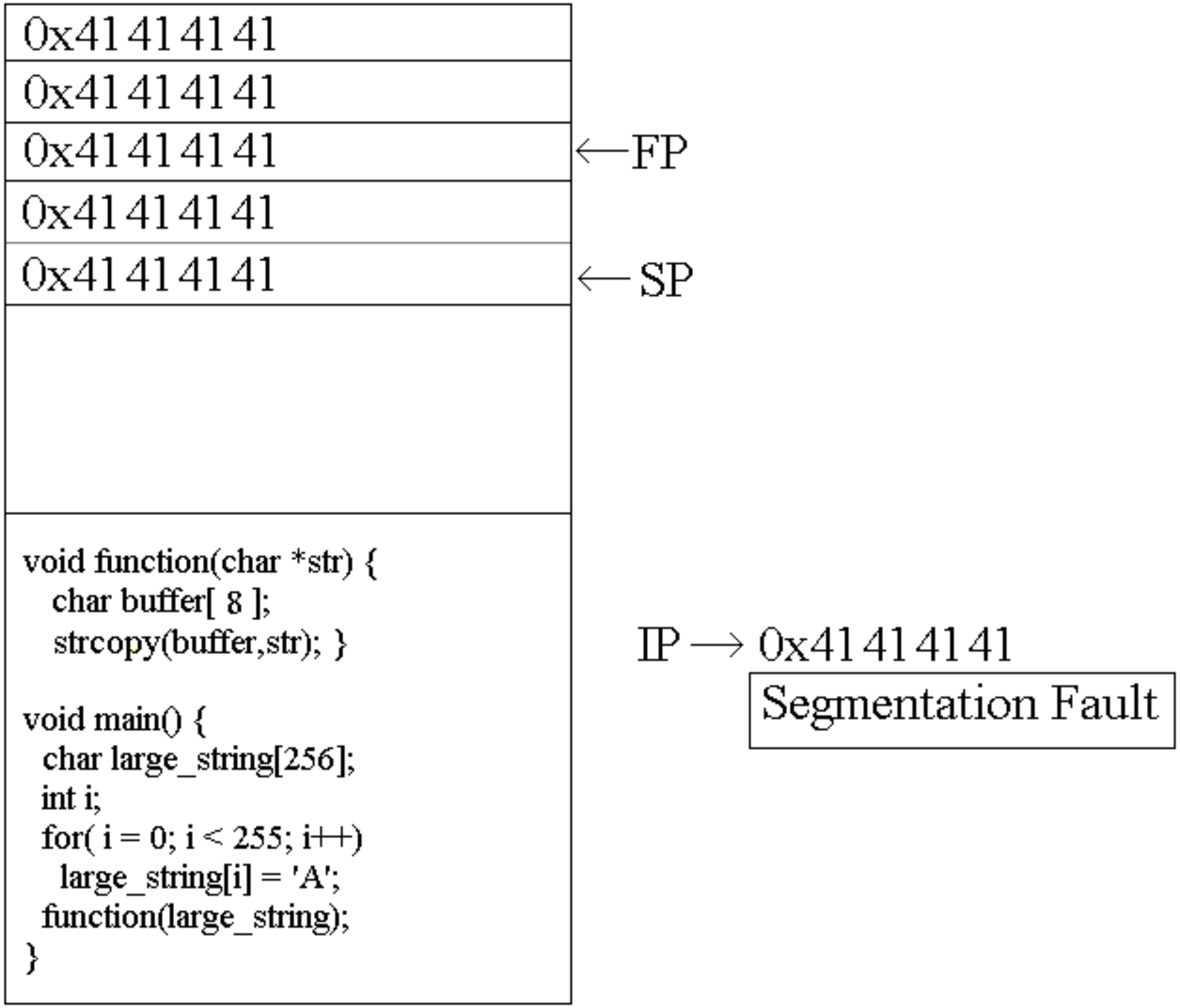
0x41414141

0x41414141

0x41414141

```
void function(char *str) {
   char buffer[ 8 ];
   strcopy(buffer,str); }

void main() {
  char large_string[256];
  int i;
  for( i = 0; i < 255; i++)
    large_string[i] = 'A';
  function(large_string);
}
```

Buffer Overflows

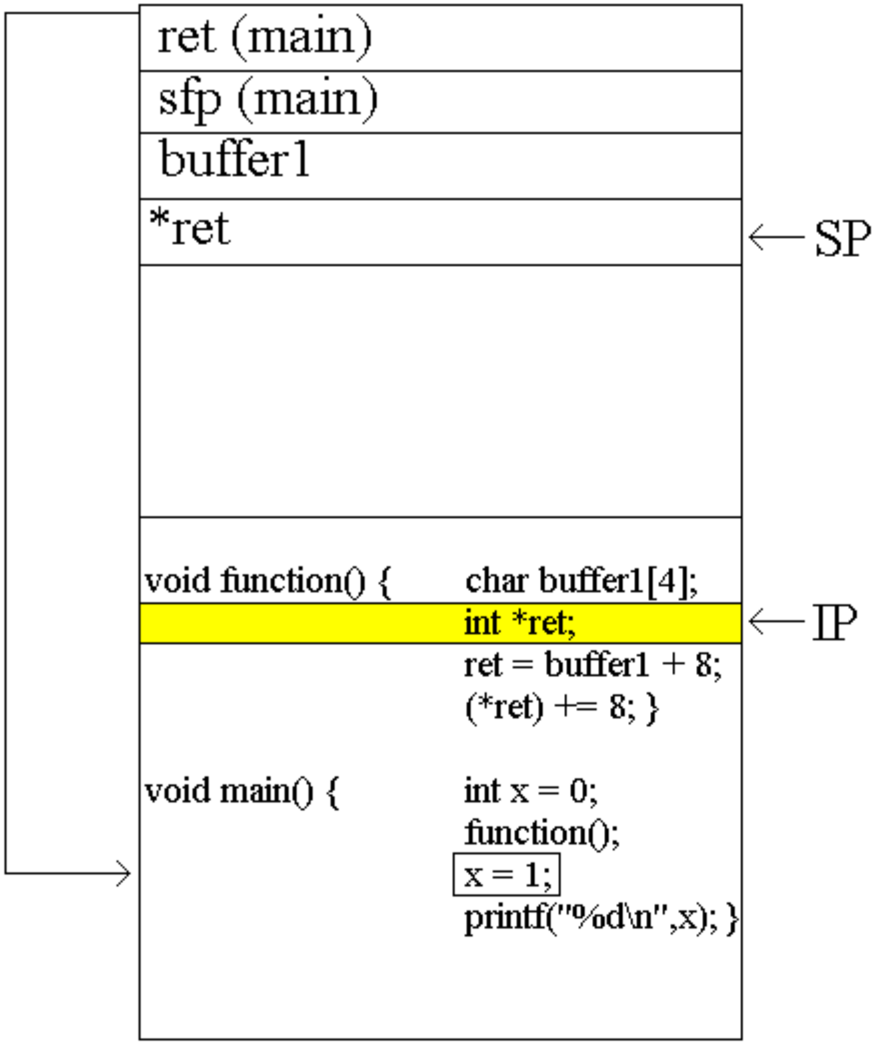


```
void function(char *str) {
   char buffer[ 8 ];
   strcopy(buffer,str); }

void main() {
  char large_string[256];
  int i;
  for( i = 0; i < 255; i++)
    large_string[i] = 'A';
  function(large_string);
}
```

Modifying the Execution Flow

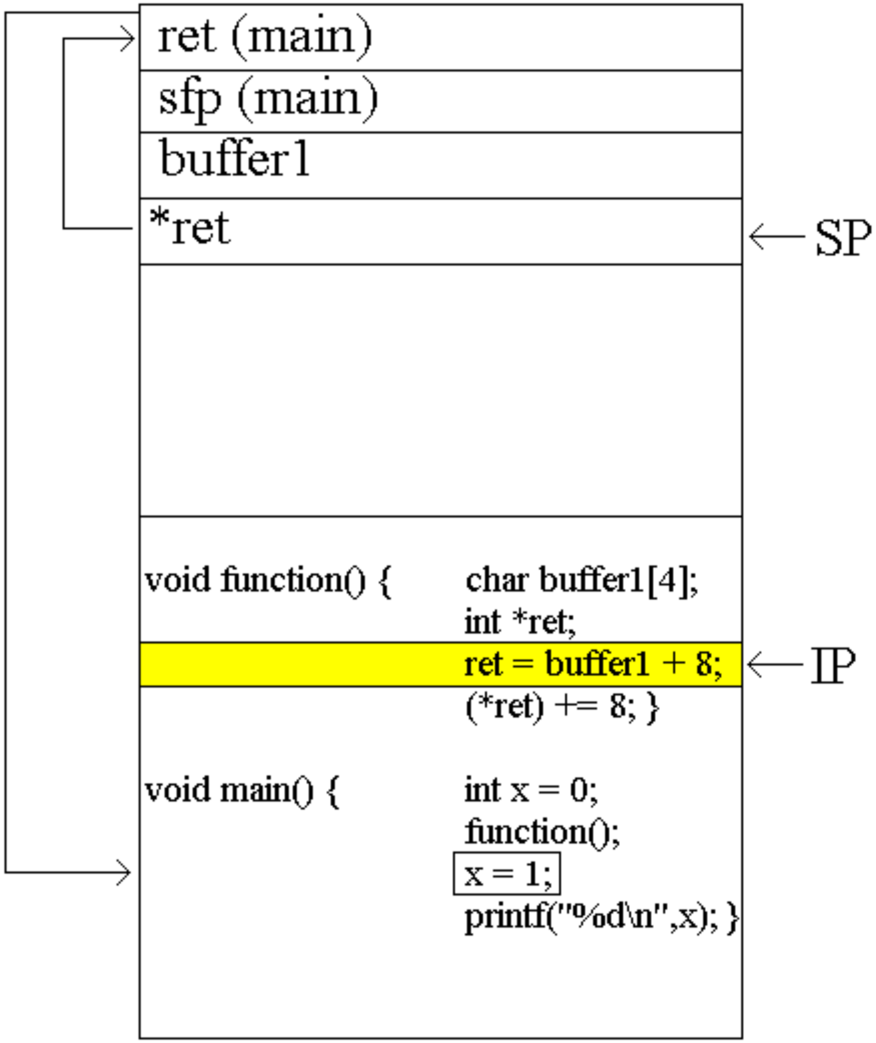
```
void function()
    {  char buffer1[4];
       int *ret;
       ret = buffer1 + 8;
       (*ret) += 8;    }

void main()
    {  int x = 0;
       function();
       x = 1;
       printf("%d\n",x);  }
```

Modifying the Execution Flow



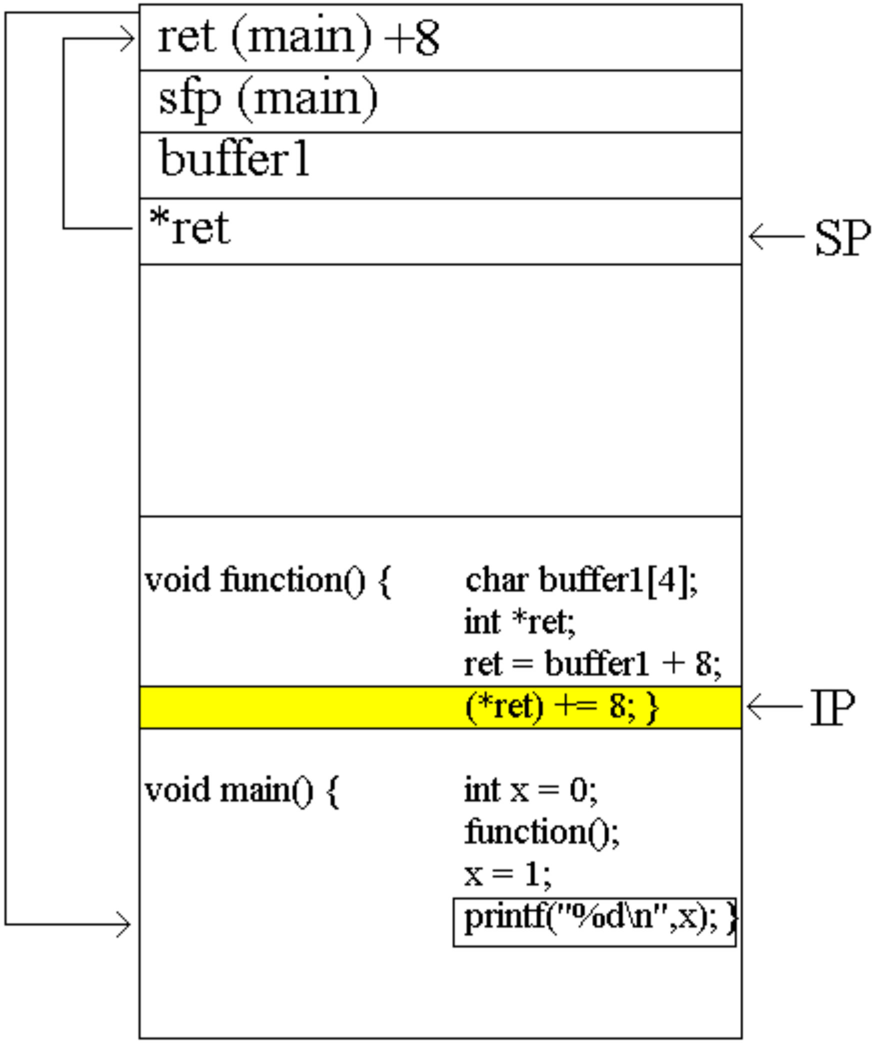
```
void function() {        char buffer1[4];
                         int *ret;
                         ret = buffer1 + 8;
                         (*ret) += 8; }

void main() {            int x = 0;
                         function();
                         x = 1;
                         printf("%d\n",x); }
```

Modifying the Execution Flow



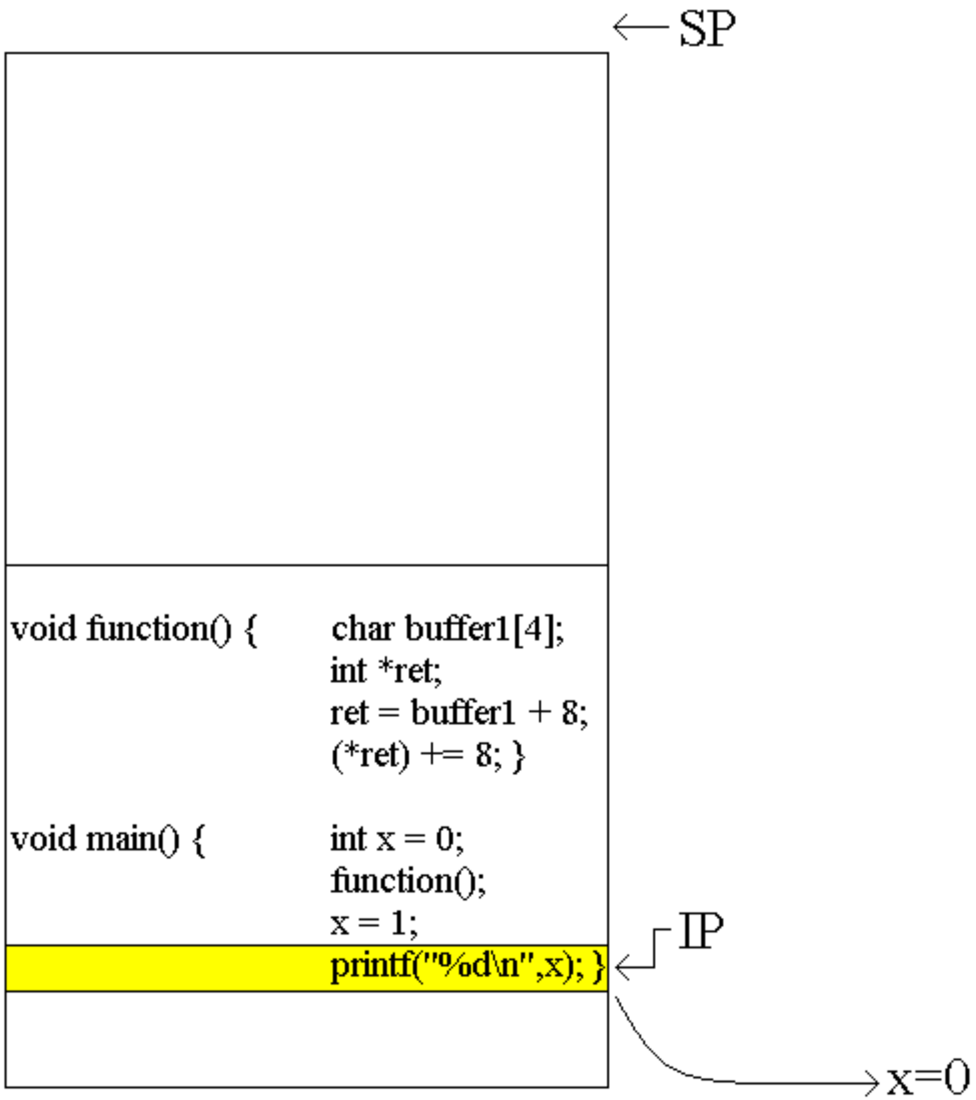
```
void function() {       char buffer1[4];
                        int *ret;
                        ret = buffer1 + 8;
                        (*ret) += 8; }

void main() {           int x = 0;
                        function();
                        x = 1;
                        printf("%d\n",x); }
```

Modifying the Execution Flow



```
void function() {      char buffer1[4];
                       int *ret;
                       ret = buffer1 + 8;
                       (*ret) += 8; }

void main() {          int x = 0;
                       function();
                       x = 1;
                       printf("%d\n",x); }
```

Modifying the Execution Flow



```
void function() {        char buffer1[4];
                         int *ret;
                         ret = buffer1 + 8;
                         (*ret) += 8; }

void main() {            int x = 0;
                         function();
                         x = 1;
                         printf("%d\n",x); }
```

Tools Used

- Compiler: gcc –fno-stack-protector
- Shell
  - Mac or Cygwin
- Windows/Linux: objdump -D
- Mac: otools –tv
- Your mileage may vary…

Challenge Problem

- Install these tools
- Try playing with the offset constants
  - I.e.,. the “8”s
    - are these correct?
  - Can you get the predicted behavior to work by altering the offsets?
  - Go ahead and use “brute force” search
- Try to figure out what the constants should be with otools or objdump
  - …and/or by looking at the slides for x86
  - Turn in your best answer(s)
    - i.e., the version(s) of stacksmash.c that alter control flow other than segmentation faults