Setup

- YOU NEED AN SSH CLIENT – DO THIS NOW
- If on Windows
  - Download PuTTY (google it)
- If on Linux
  - You probably already have an SSH client, so chill

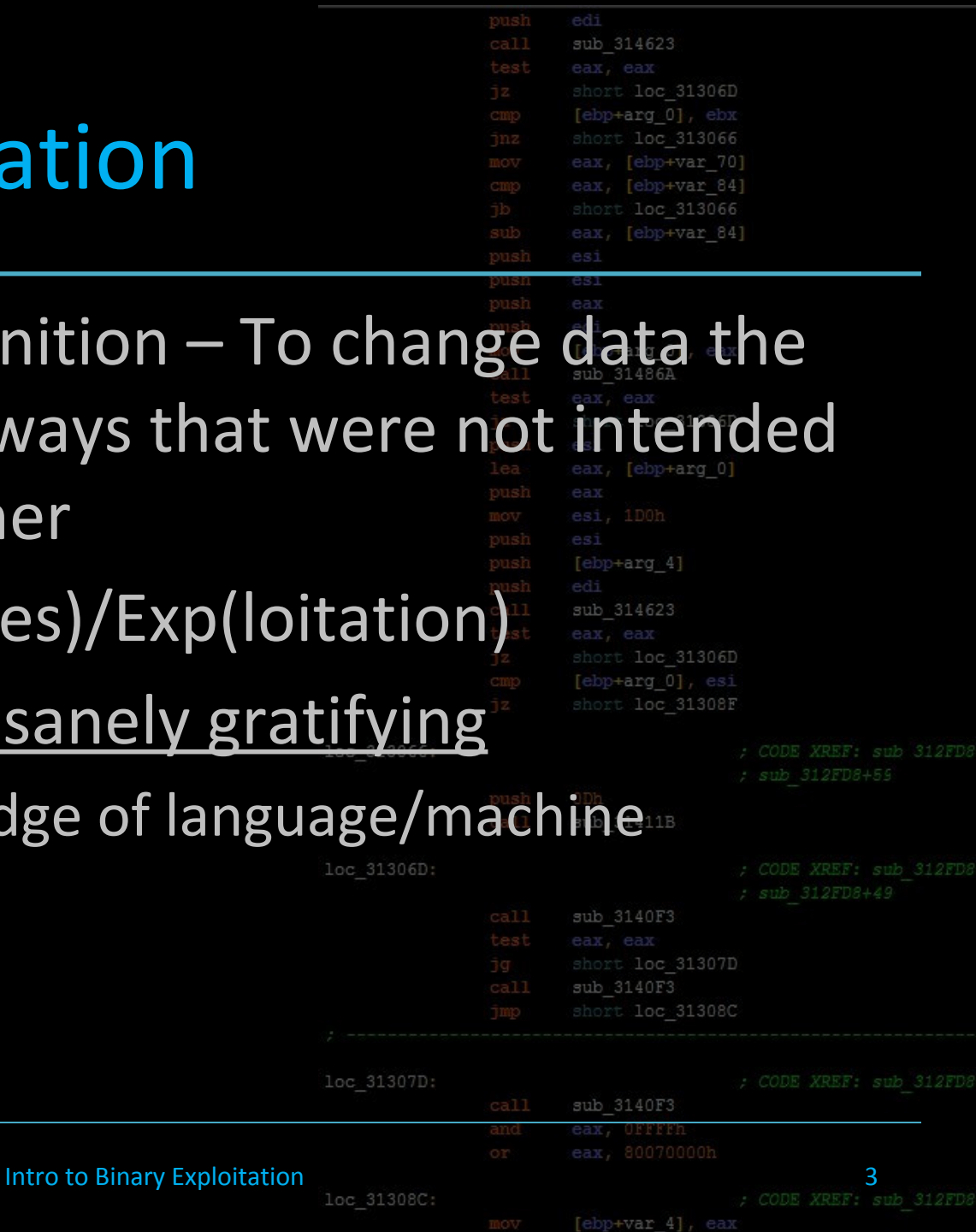
RPISEC

Intro to Binary Exploitation

Fall 2014

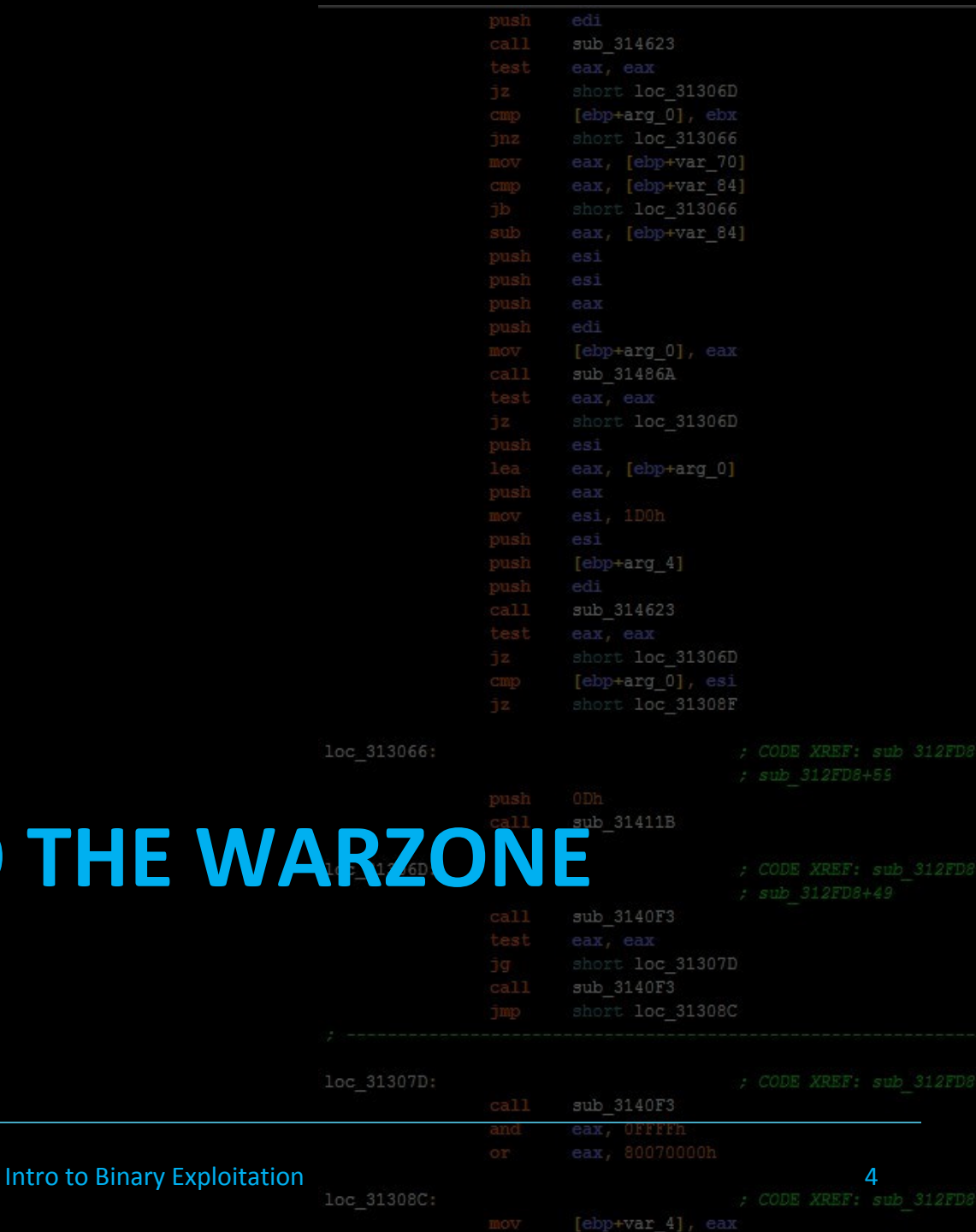
Binary Exploitation

- The simplest definition – To change data the program uses in ways that were not intended by the programmer
- In CTFs - Pwn(ables)/Exp(loitation)
- Very technical, insanely gratifying
  - Intimate knowledge of language/machine



let’s pwn some stuff

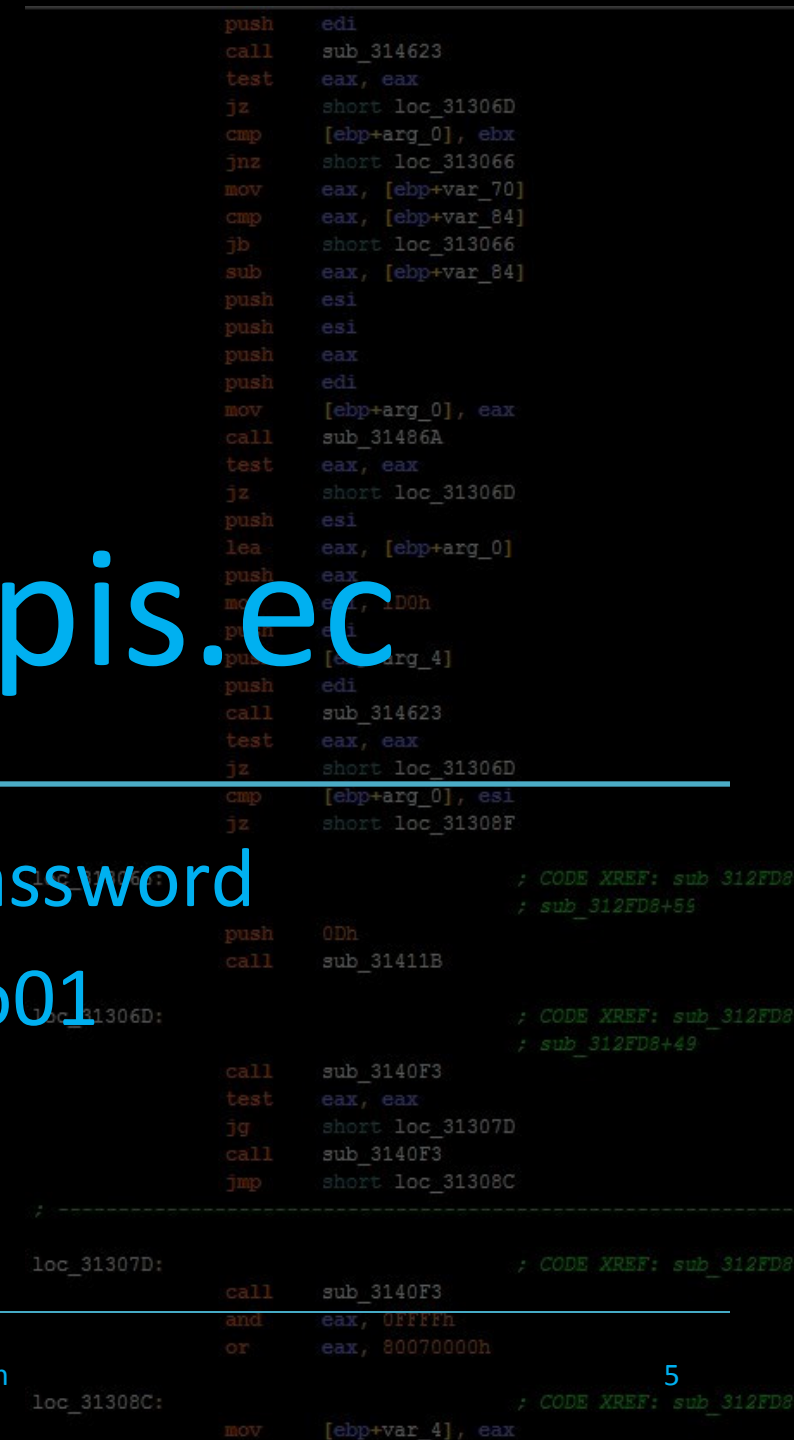
WELCOME TO THE WARZONE



warzone.rpis.ec

ssh username/password

intro01:intro01



Tips to get started

- cd /levels
- ./intro01
  - AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
- python –c ‘print “A”*20’
- gdb ./intro01
  - run

- In GDB:
  - Info functions
  - Info registers
    - i r
  - disassemble <function>
    - disas main
  - breakpoint <function>
    - b main
  - breakpoint * <address>
    - b * 0x08048455

Stack Overview

- The stack is a region of memory for a program to maintain function variables and stuff during execution
- This is main()'s stack ------->

0x00	0x00	0x00	0x00	
0x00	0x00	0x00	0x00	
0x41	0x41	0x41	0x41	<-- buffer[0] to buffer[3]
0x41	0x41	0x41	0x41	...
...	...	...	...	...
0x41	0x41	0x00	0x00	...
0x00	0x00	0x00	0x00	<-- buffer[60] to buffer[63]
0x00	0x00	0x00	0x00	<-- modified
0x00	0x00	0x00	0x00	<-- Saved EBP Address
0xd3	0x54	0xe4	0xb7	<-- Saved Return Address
...	...	...	...	<-- Previous stack frame
...	...	...	...	

Understanding the Stack

0x00	0x00	0x00	0x00	
0x00	0x00	0x00	0x00	
0x41	0x41	0x41	0x41	<-- buffer[0] to buffer[3]
0x41	0x41	0x41	0x41	...
...	...	...	...	...
0x41	0x41	0x00	0x00	...
0x00	0x00	0x00	0x00	<-- buffer[60] to buffer[63]
0x00	0x00	0x00	0x00	<-- modified
0x00	0x00	0x00	0x00	<-- Saved EBP Address
0xd3	0x54	0xe4	0xb7	<-- Saved Return Address
...	...	...	...	<-- Previous stack frame
...	...	...	...	

Understanding the Stack

0x00	0x00	0x00	0x00	
0x00	0x00	0x00	0x00	
0x41	0x41	0x41	0x41	<-- buffer[0] to buffer[3]
0x41	0x41	0x41	0x41	...
...	...	...	...	...
0x41	0x41	0x41	0x41	...
0x41	0x41	0x41	0x41	<-- buffer[60] to buffer[63]
0x00	0x00	0x00	0x00	<-- modified
0x00	0x00	0x00	0x00	<-- Saved EBP Address
0xd3	0x54	0xe4	0xb7	<-- Saved Return Address
...	...	...	...	<-- Previous stack frame
...	...	...	...	

Corrupting the Stack

0x00	0x00	0x00	0x00	
0x00	0x00	0x00	0x00	
0x41	0x41	0x41	0x41	<-- buffer[0] to buffer[3]
0x41	0x41	0x41	0x41	...
...	...	...	...	...
0x41	0x41	0x41	0x41	...
0x41	0x41	0x41	0x41	<-- buffer[60] to buffer[63]
0x41	0x00	0x00	0x00	<-- modified
0x00	0x00	0x00	0x00	<-- Saved EBP Address
0xd3	0x54	0xe4	0xb7	<-- Saved Return Address
...	...	...	...	<-- Previous stack frame
...	...	...	...	

PWNING the Stack

0x00	0x00	0x00	0x00	
0x00	0x00	0x00	0x00	
0x41	0x41	0x41	0x41	<-- buffer[0] to buffer[3]
0x41	0x41	0x41	0x41	...
...	...	...	...	...
0x41	0x41	0x41	0x41	...
0x41	0x41	0x41	0x41	<-- buffer[60] to buffer[63]
0xef	0xbe	0xad	0xde	<-- modified
0x00	0x00	0x00	0x00	<-- Saved EBP Address
0xd3	0x54	0xe4	0xb7	<-- Saved Return Address
...	...	...	...	<-- Previous stack frame
...	...	...	...	

Endianess – How data is stored in memory

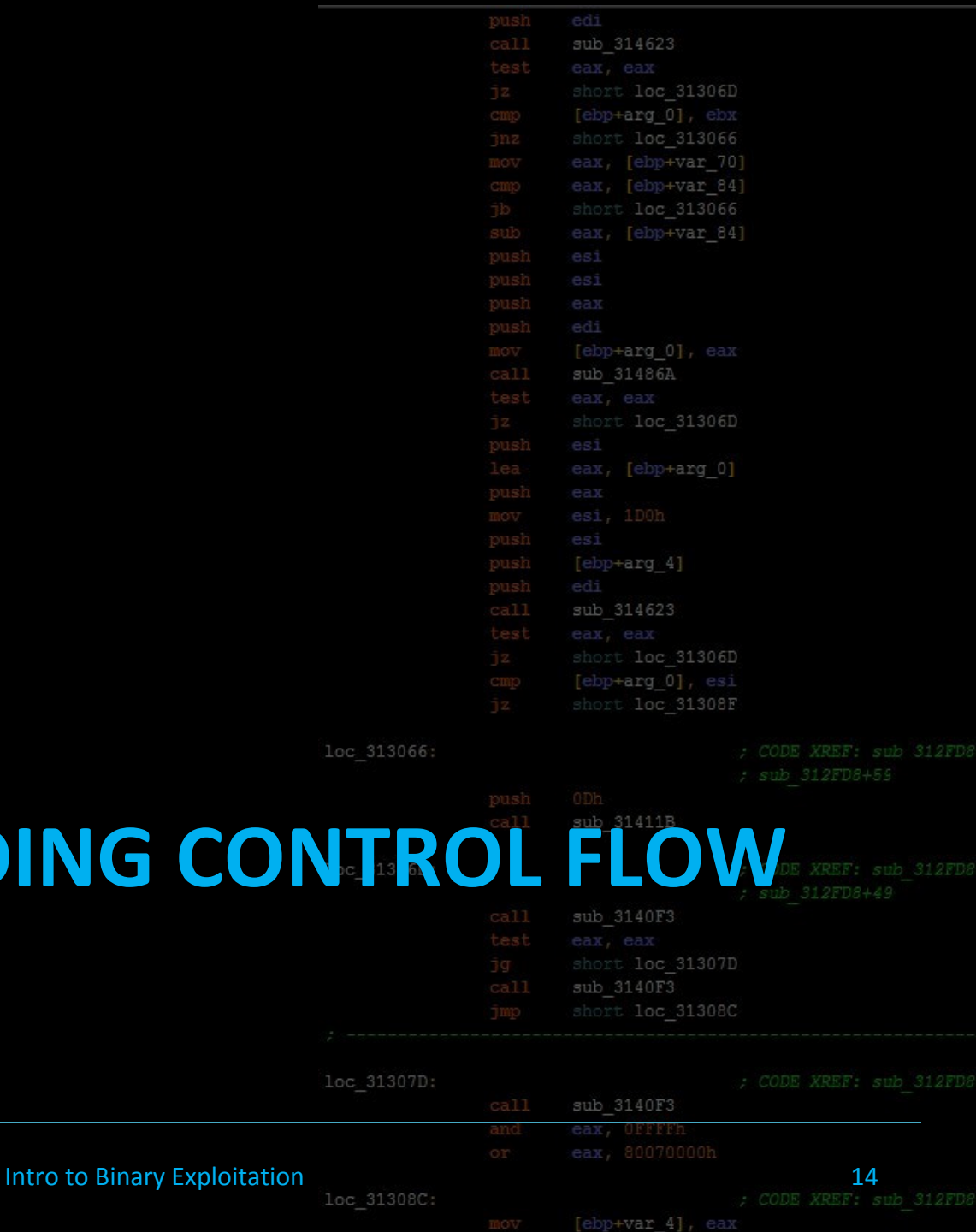
- Endianess – How data is stored in memory
- Modern computers are generally little endian
  - ‘little end in’
- Endianess can be confusing, and I don’t want to get into the details
  - 0x41424344 stored as 0x44, 0x43, 0x42, 0x41
  - 0xdeadbeef stored as 0xef, 0xbe, 0xad, 0xde

Intro01 Exploit

(python -c 'print "A"*64 + "\xef\xbe\xad\xde"'; cat) | ./intro01

Bend it like Beckham

UNDERSTANDING CONTROL FLOW



Example ELF / EXE in Memory

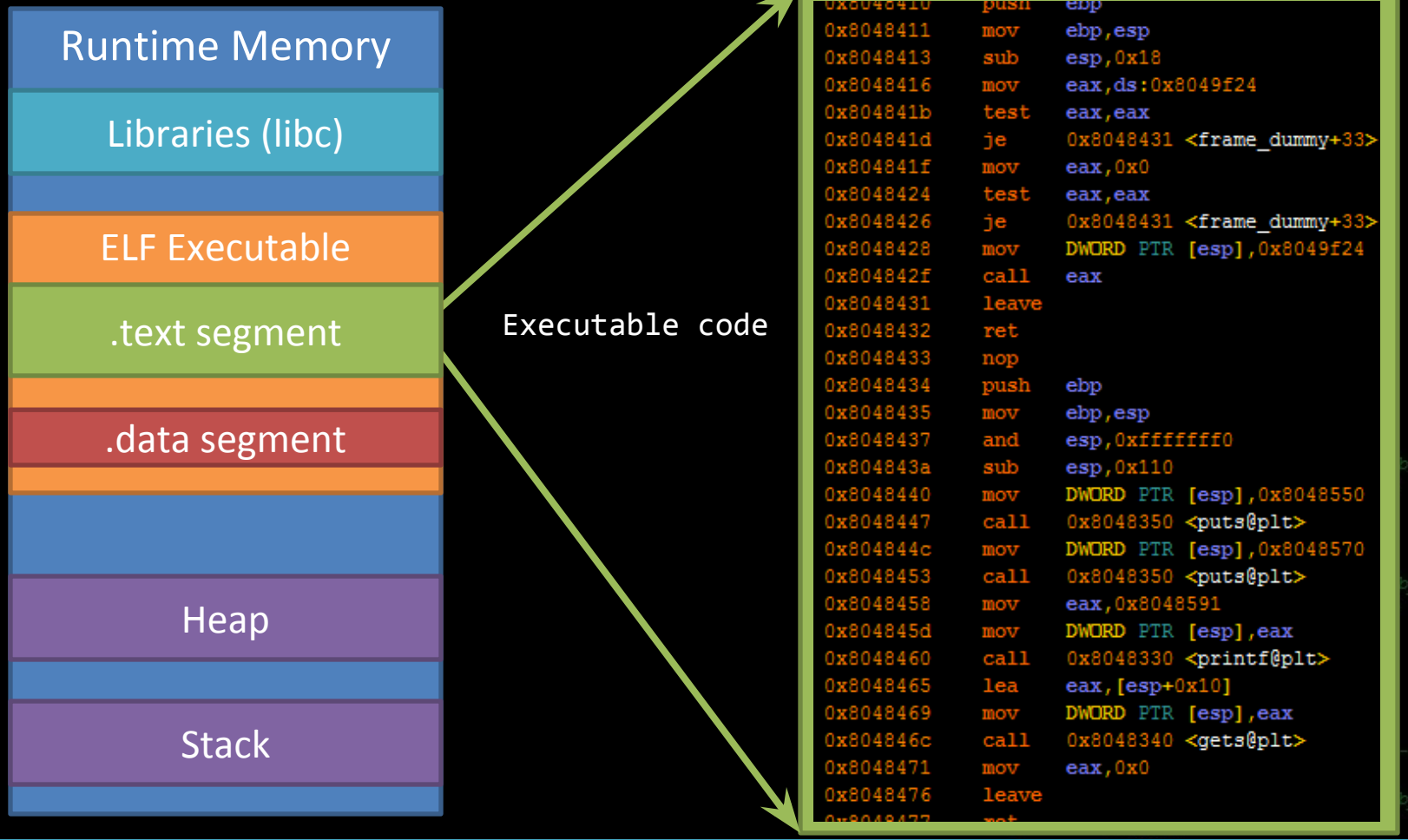
0x00000000 – Start of memory

0x08048000 – Start of .text Segment

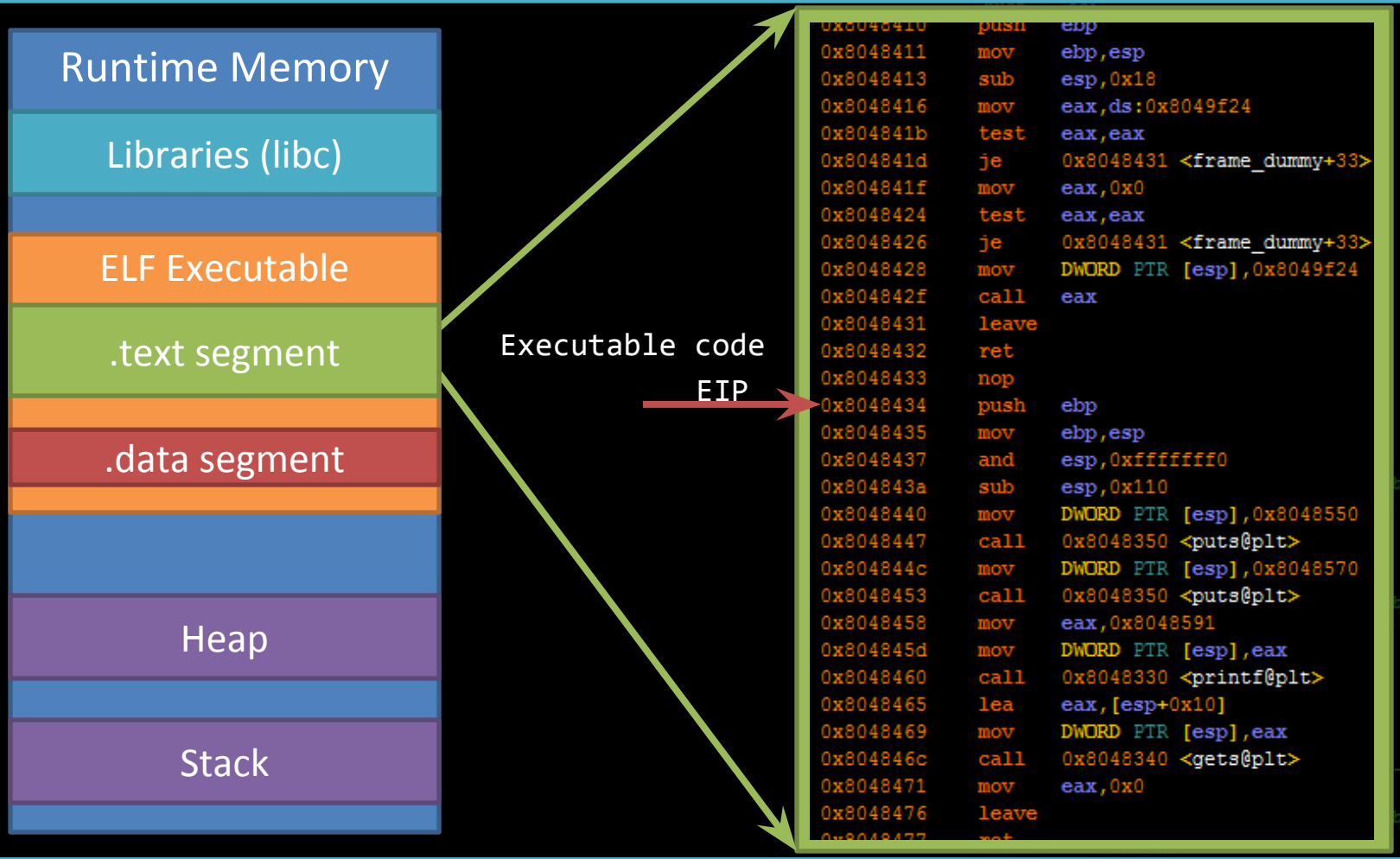
0xbfff0000 – Top of stack

0xFFFFFFFF – End of memory

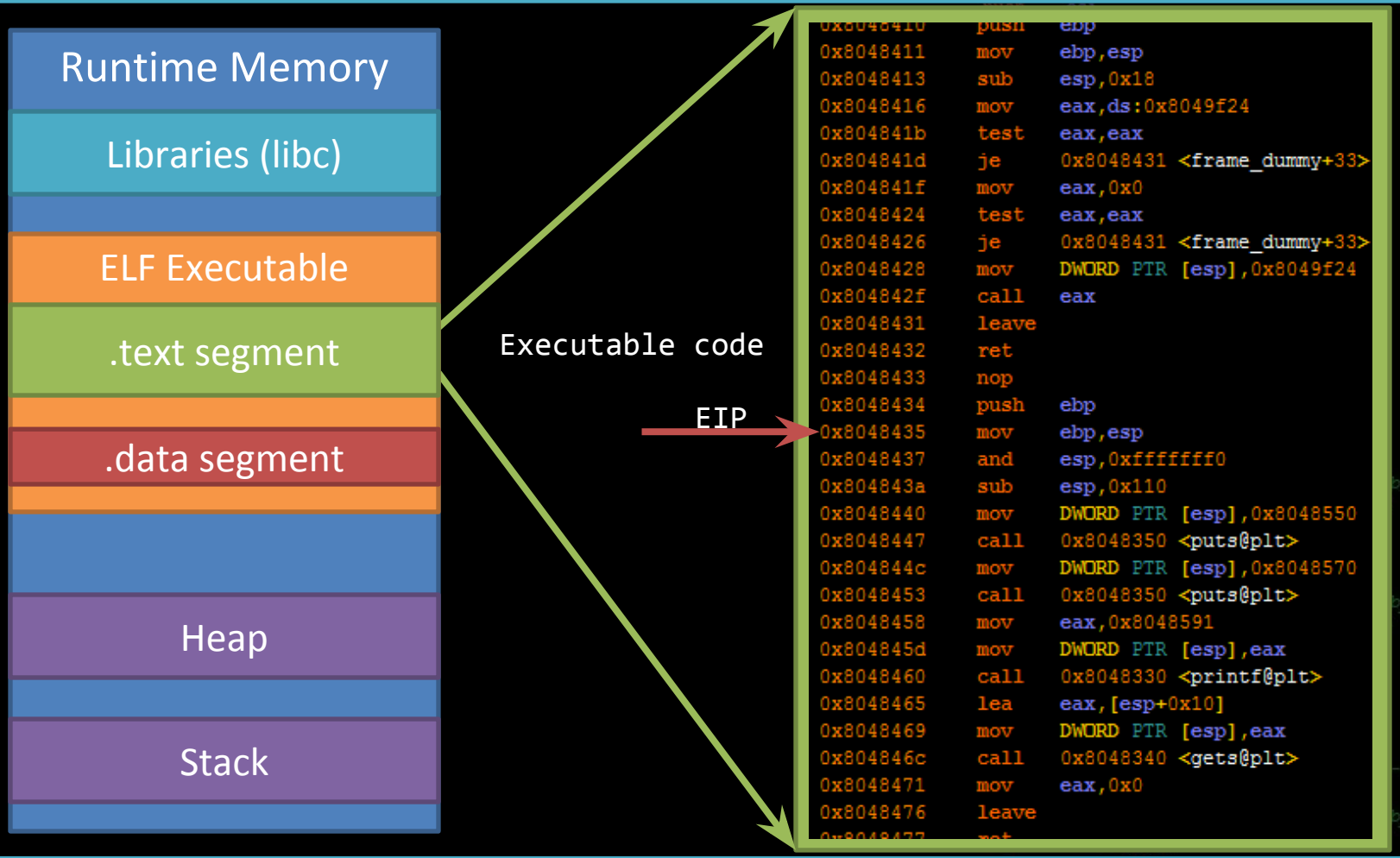
Example ELF / EXE in Memory



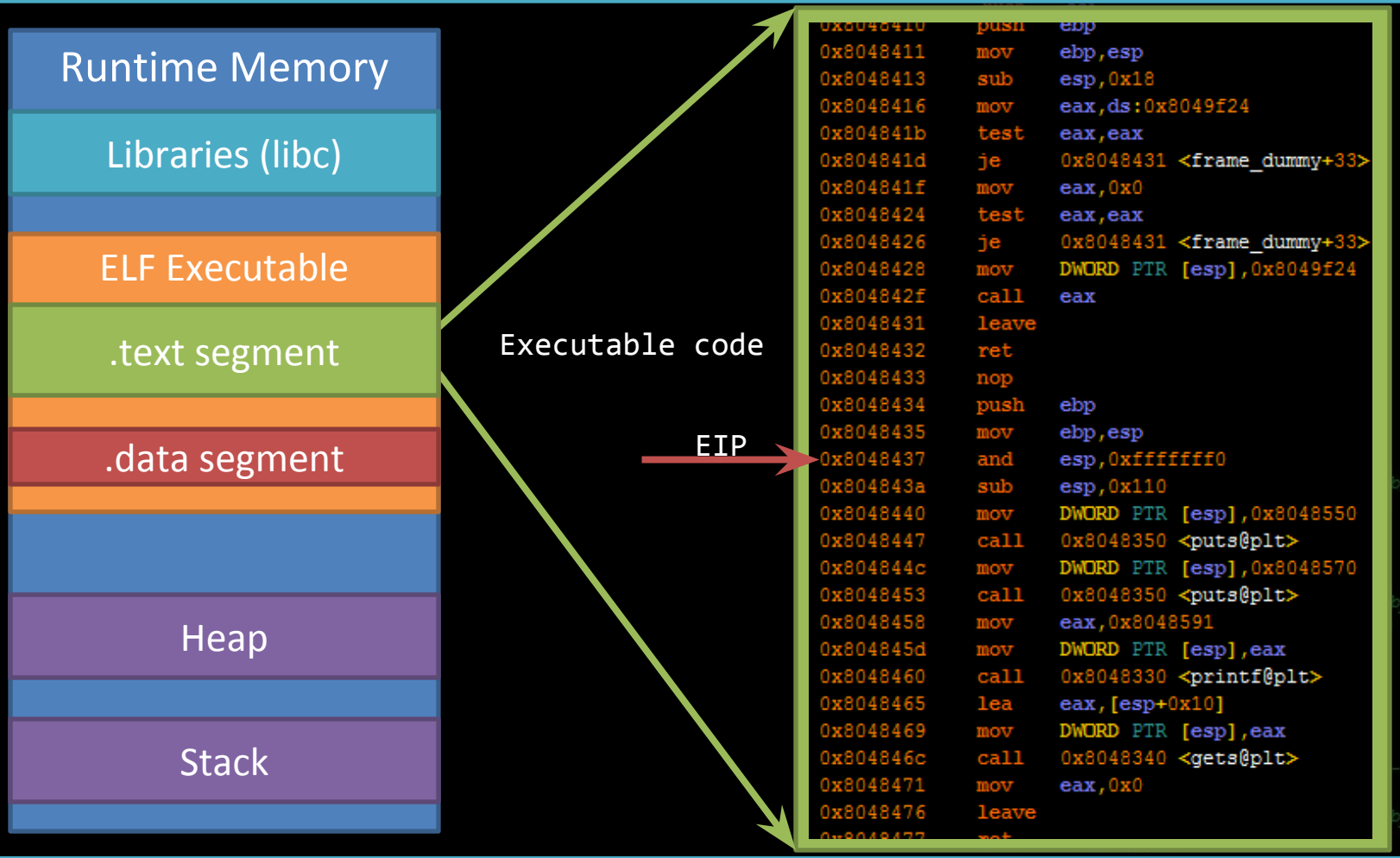
Example ELF / EXE in Memory



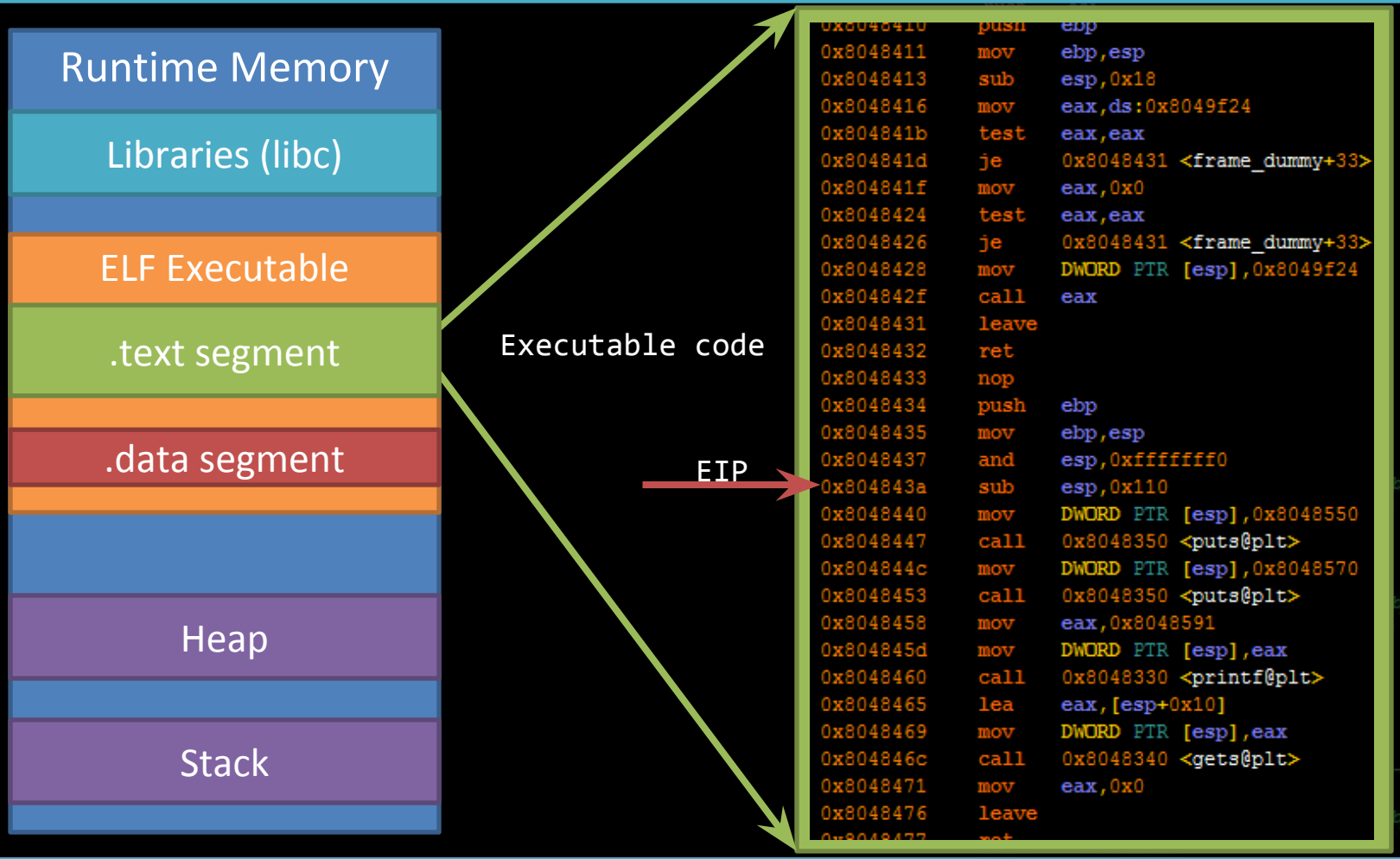
Example ELF / EXE in Memory



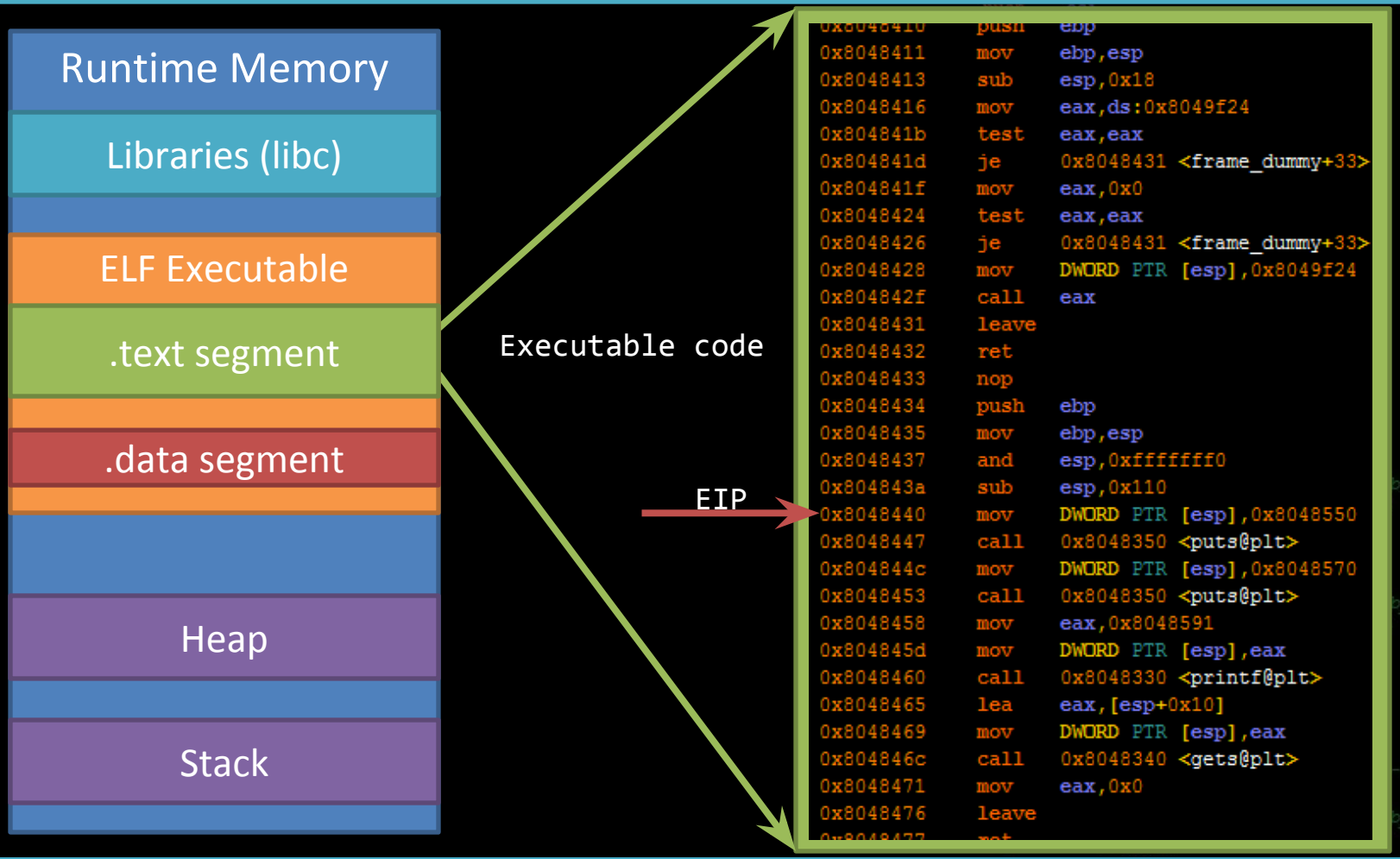
Example ELF / EXE in Memory



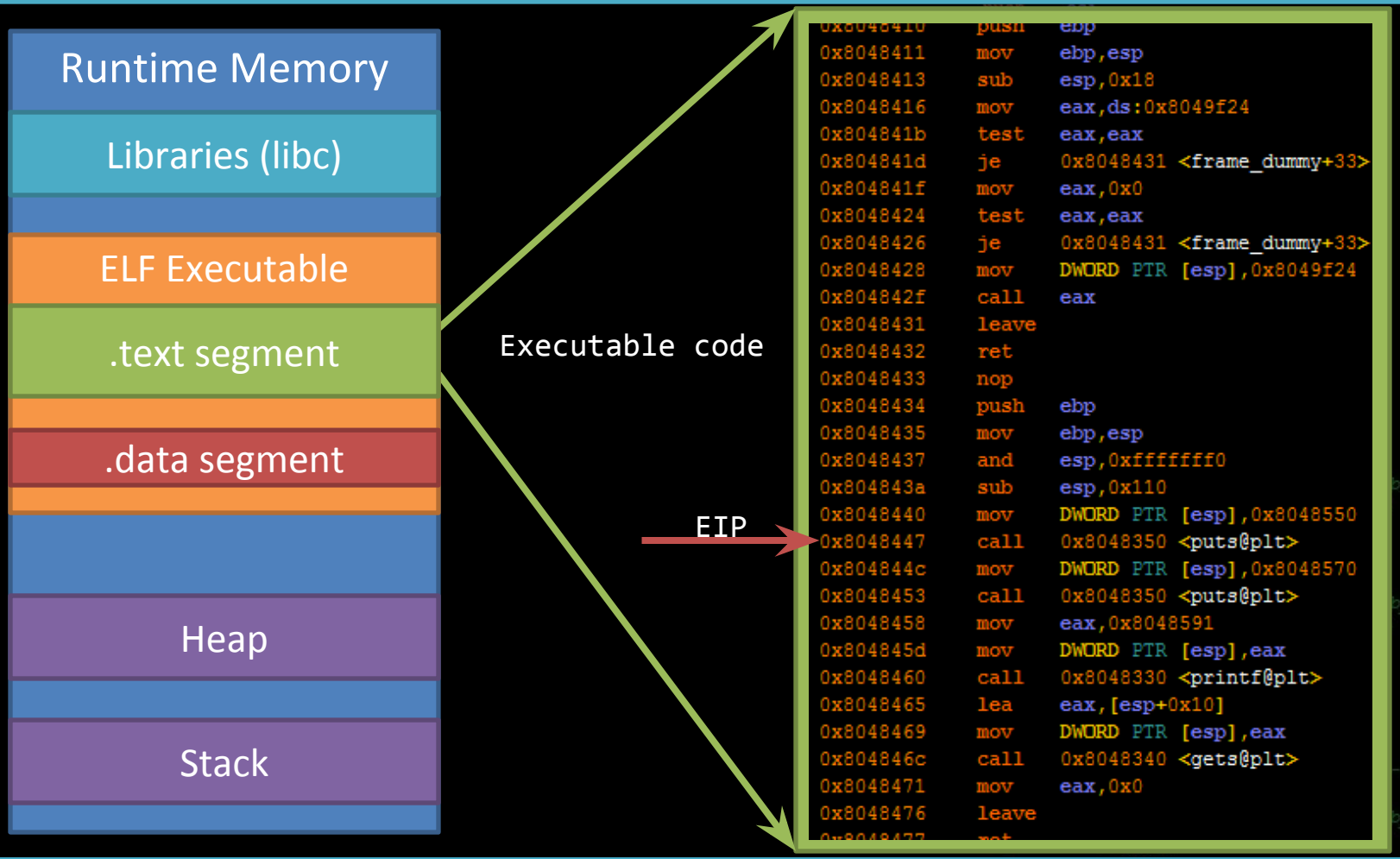
Example ELF / EXE in Memory



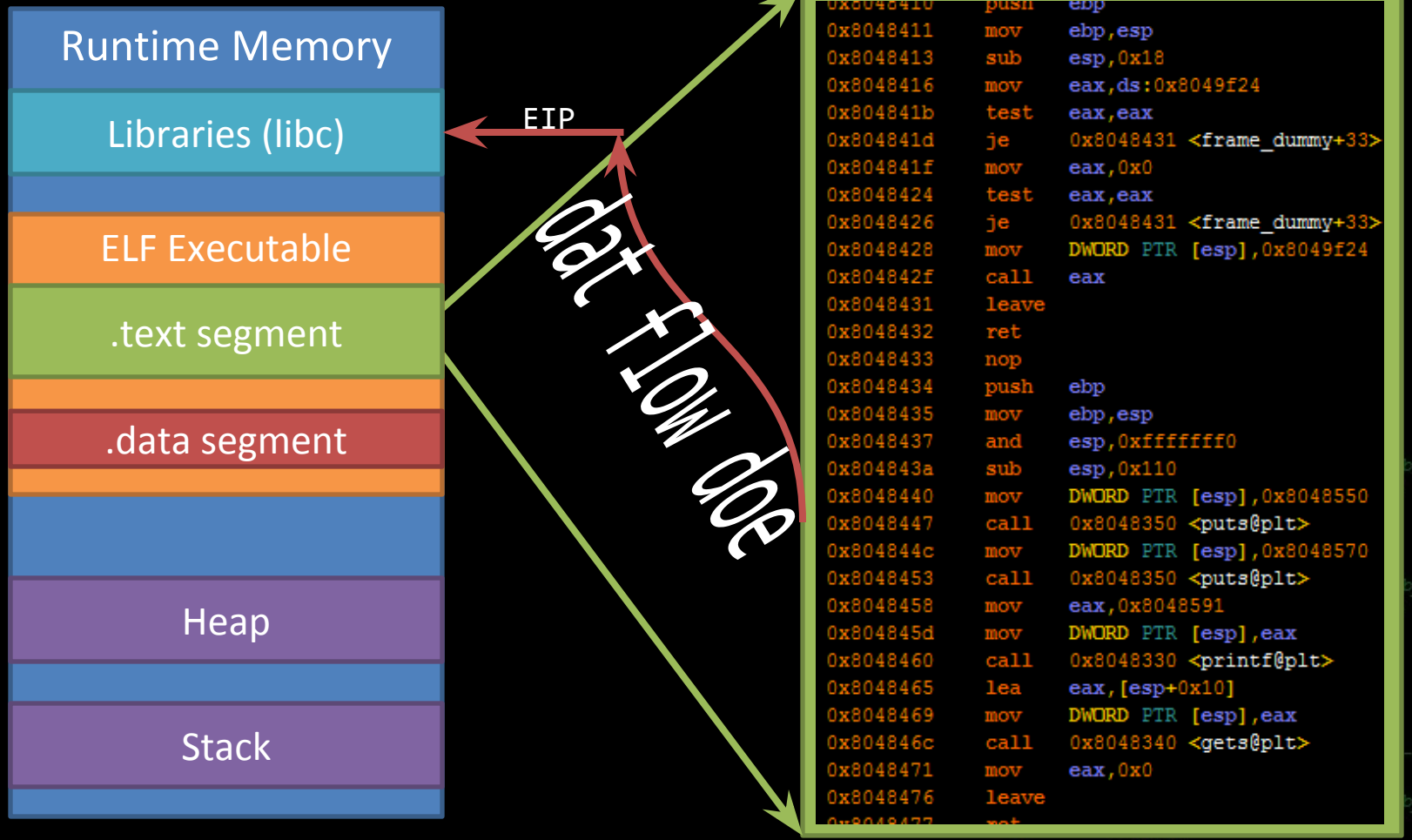
Example ELF / EXE in Memory



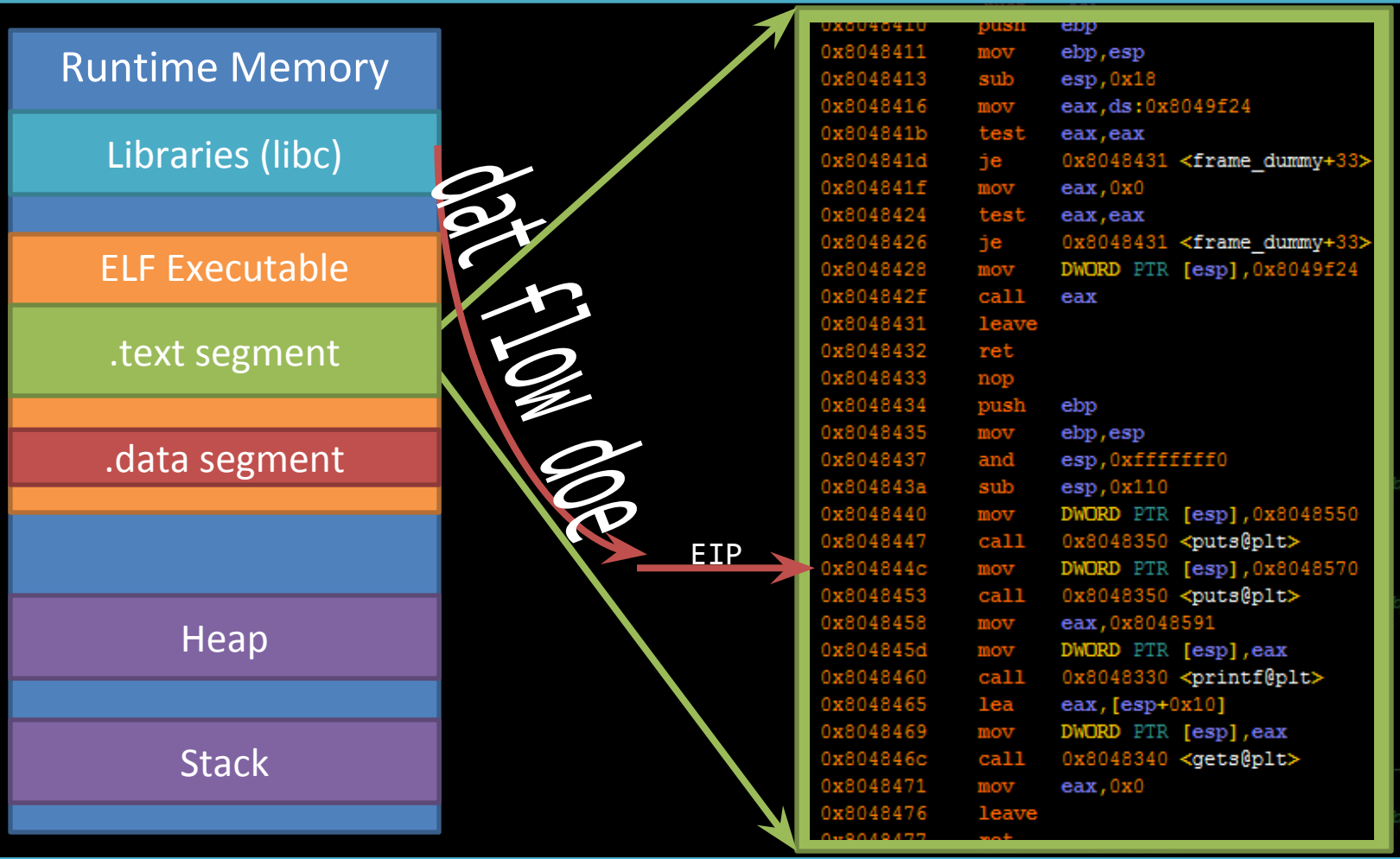
Example ELF / EXE in Memory



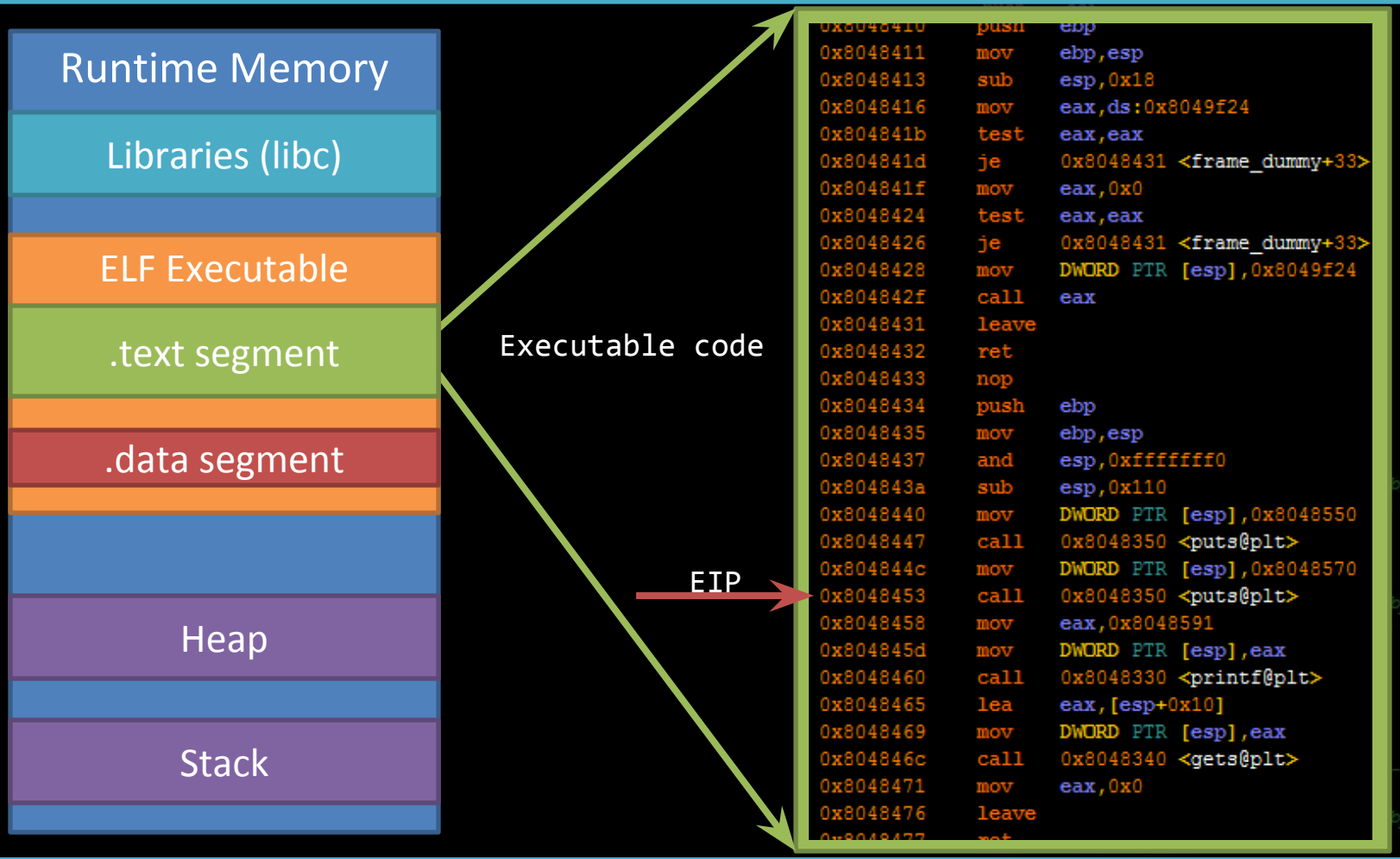
Example ELF / EXE in Memory



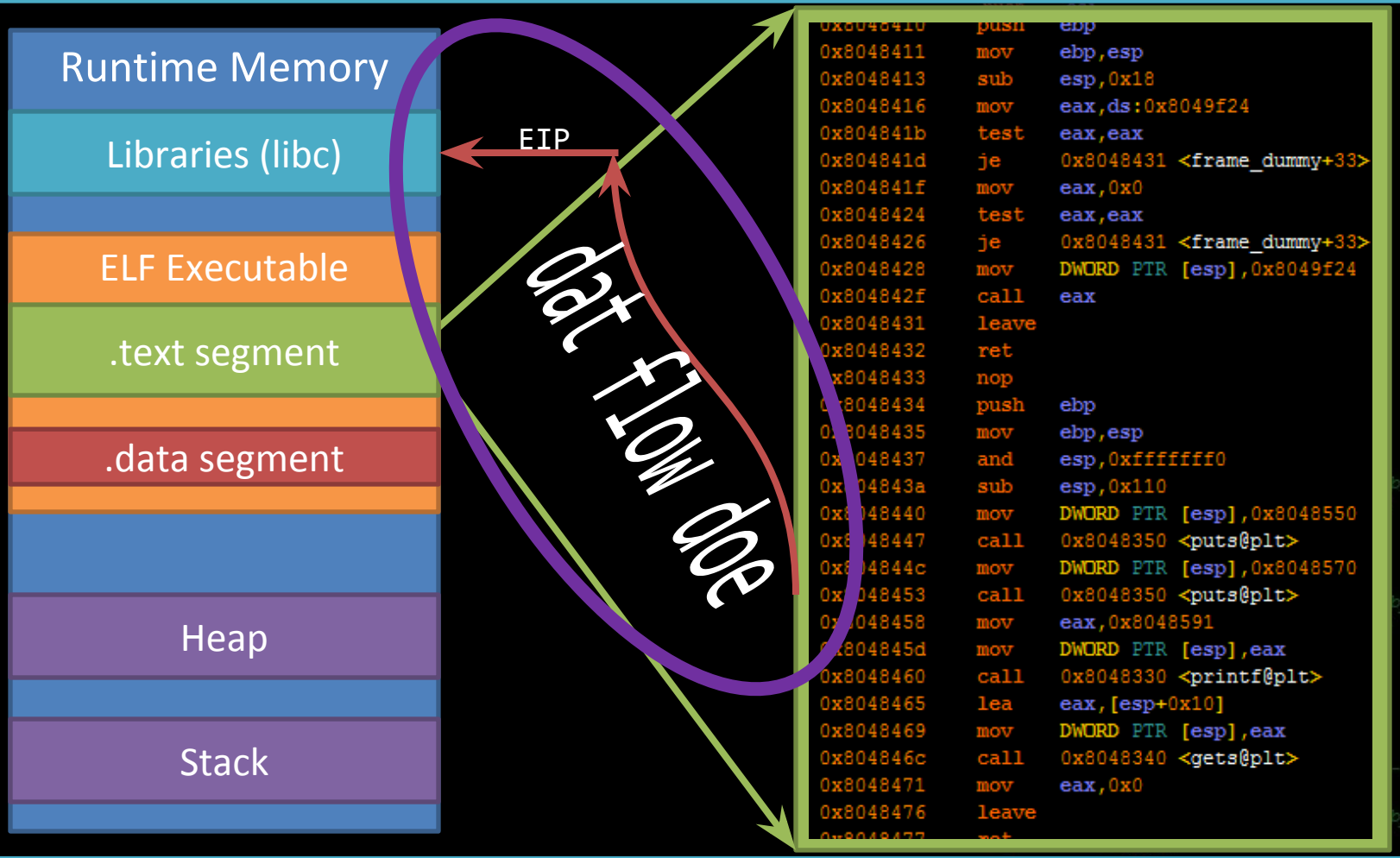
Example ELF / EXE in Memory



Example ELF / EXE in Memory



Example ELF / EXE in Memory



How Calling Works

```
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	
...	...	...	...	
...	...	...	...	

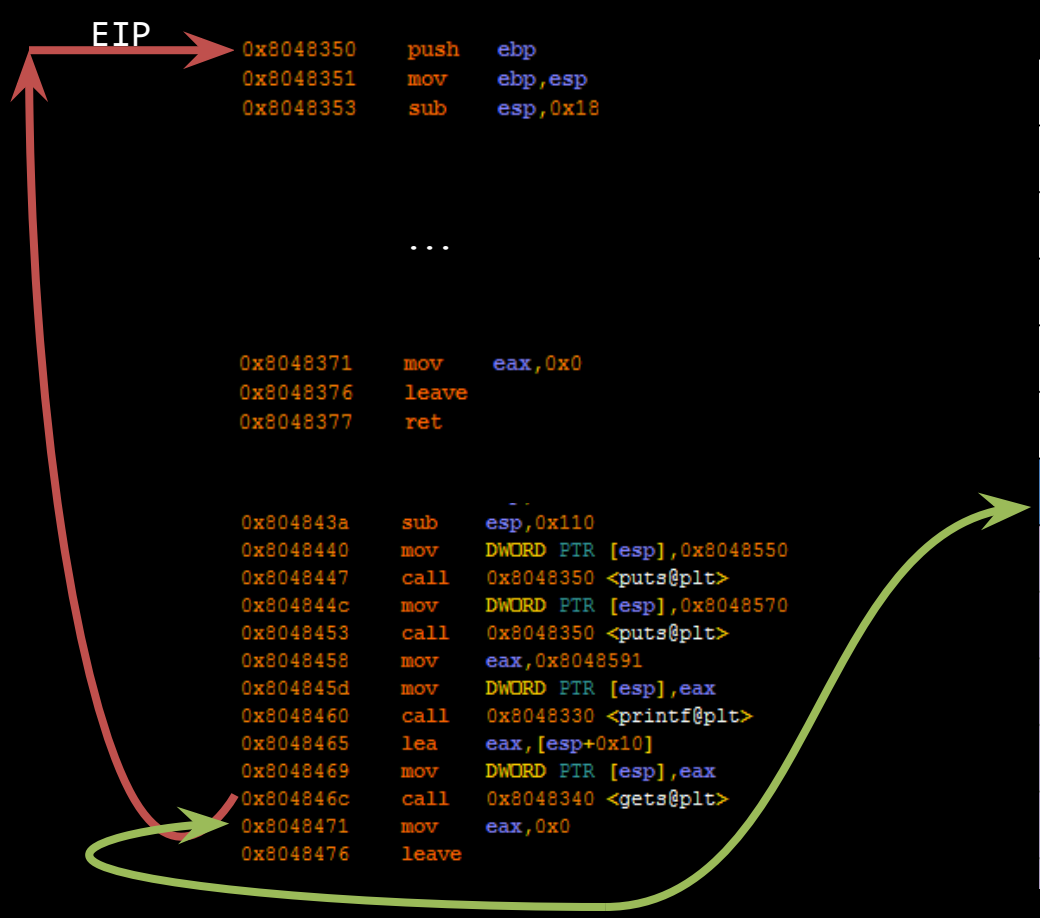
How Calling Works

```
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	
...	...	...	...	
...	...	...	...	

How Calling Works



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	
...	...	...	...	
...	...	...	...	

How Calling Works

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	
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	
...	...	...	...	
...	...	...	...	

How Calling Works

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	
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	
...	...	...	...	
...	...	...	...	

How Calling Works

```
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	
...	...	...	...	
...	...	...	...	

Returning

```
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	
...	...	...	...	
...	...	...	...	

Returning

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

             ...

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

EIP

```
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	
...	...	...	...	
...	...	...	...	

Returning

```
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	
...	...	...	...	
...	...	...	...	

Returning

```
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	
...	...	...	...	
...	...	...	...	

Returning

```
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	
...	...	...	...	
...	...	...	...	

Now that you know how it works …

OWNING CONTROL FLOW

Stack Smashing

```
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	
...	...	...	...	
...	...	...	...	

Stack Smashing

```
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	
...	...	...	...	
...	...	...	...	

Stack Smashing

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

             ...

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

EIP

```
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	
...	...	...	...	
...	...	...	...	

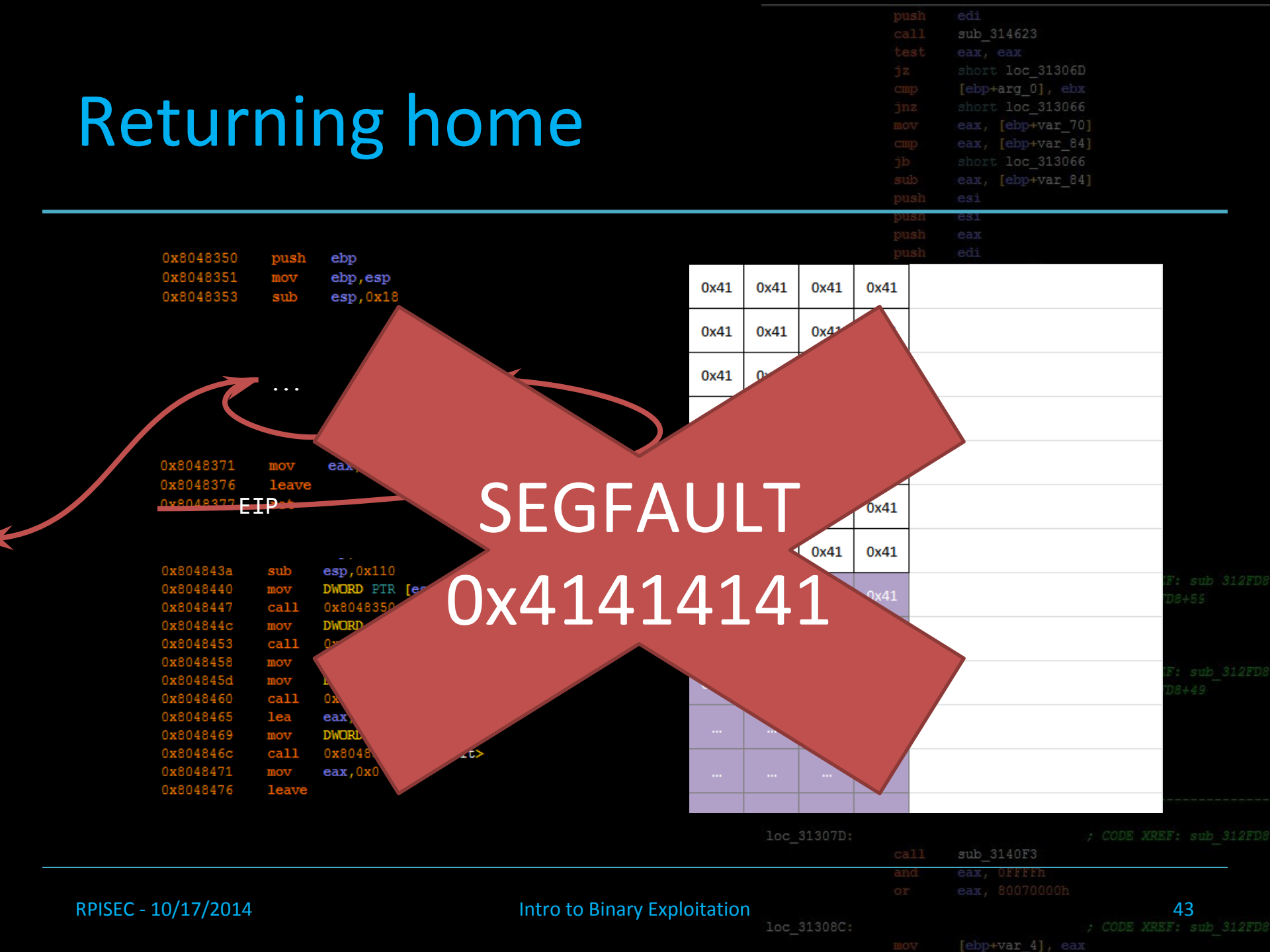
Returning

```
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

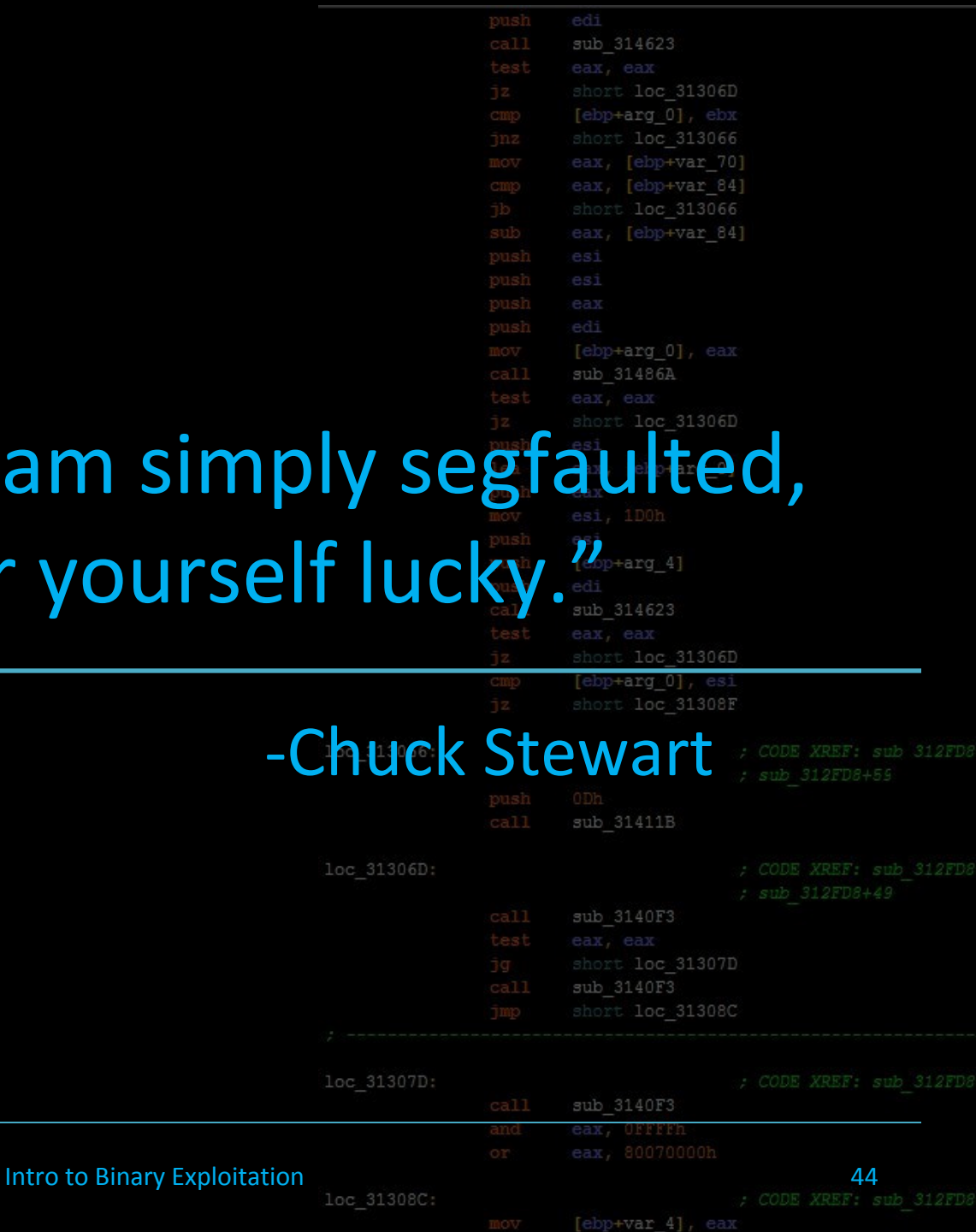
0x41	0x41	0x41	0x41	
0x41	0x41	0x41	0x41	
0x41	0x41	0x41	0x41	
0x41	0x41	0x41	0x41	
0x41	0x41	0x41	0x41	
0x41	0x41	0x41	0x41	
0x41	0x41	0x41	0x41	<----- Saved Return Address
0x41	0x41	0x41	0x41	<----- Argument One to gets()
0x41	0x41	0x41	0x41	
0x41	0x41	0x41	0x00	
...	...	...	...	
...	...	...	...	

Returning home



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

-Chuck Stewart



Redirecting Control Flow

```
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

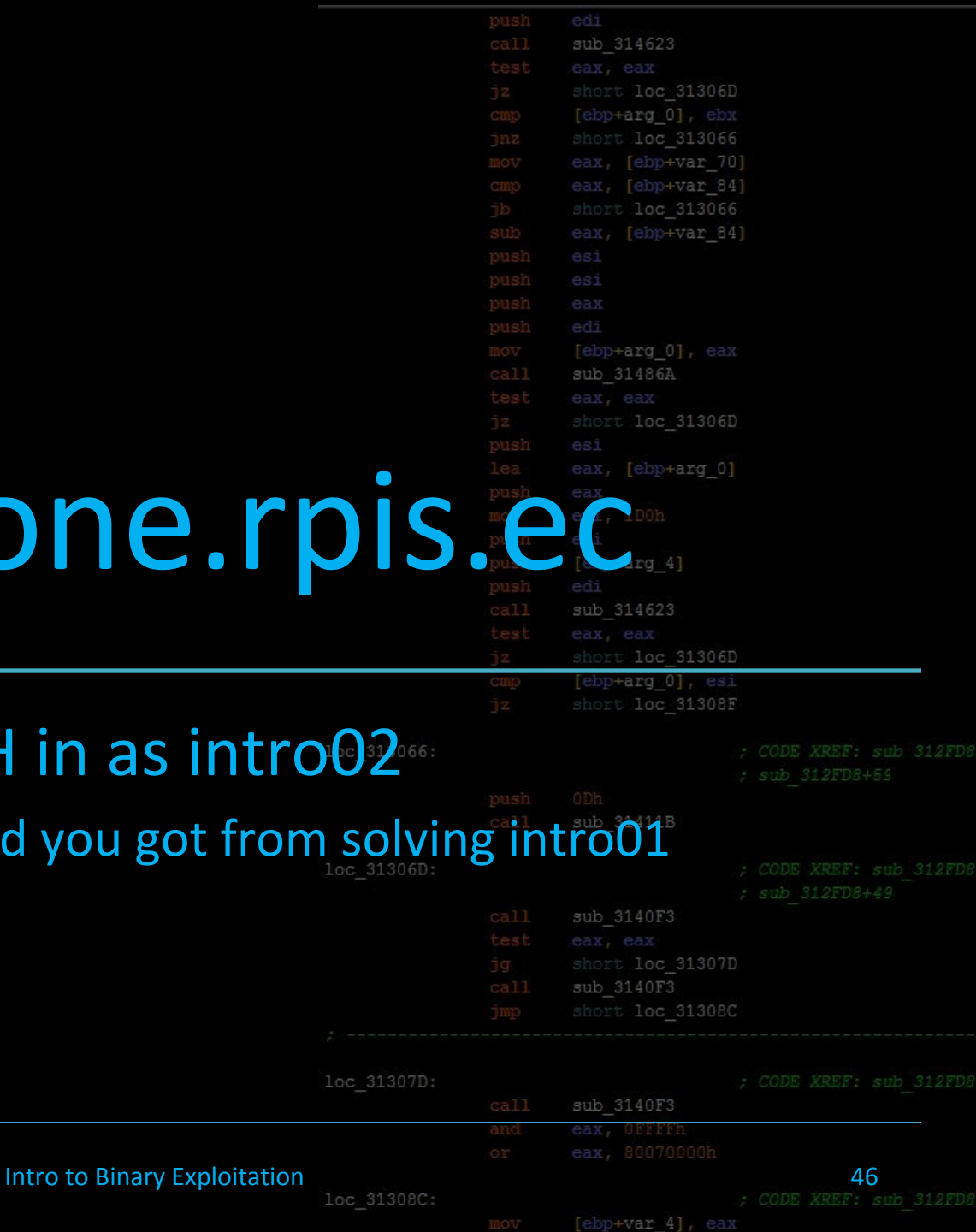
Overwrite with a code address

0x41	0x41	0x41	0x41	
0x41	0x41	0x41	0x41	
0x41	0x41	0x41	0x41	
0x41	0x41	0x41	0x41	
0x41	0x41	0x41	0x41	
0x41	0x41	0x41	0x41	
0x30	0x83	0x04	0x08	<-- Saved Return Address
0x41	0x41	0x41	0x41	<-- Argument One
0x41	0x41	0x41	0x41	
0x41	0x41	0x41	0x00	
...	...	...	...	
...	...	...	...	

warzone.rpis.ec

SSH in as intro02

use the password you got from solving intro01



Example ELF / EXE in Memory

- What if there's no easy function to pop a shell like intro02?
  - No easy 'win' function
- Make our own exec() function in a buffer on the stack, and redirect control flow to it!

Shellcode and other antics

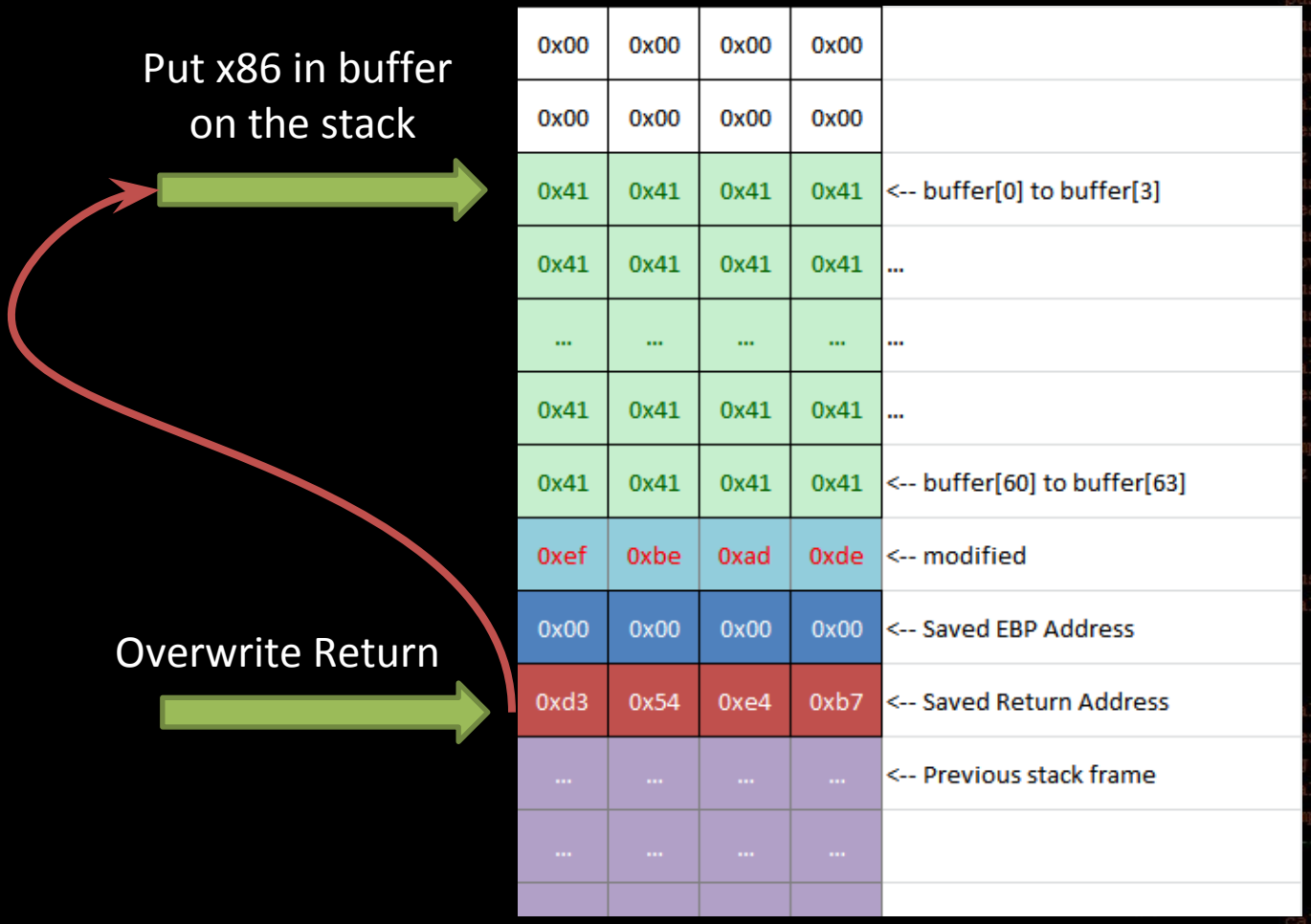
INJECTING CODE

```
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], ebx
                jnz     short loc_313066
                mov     eax, [ebp+var_70]
                cmp     eax, [ebp+var_84]
                jb      short loc_313066
                sub     eax, [ebp+var_84]
                push    esi
                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_312FD8
                                        ; sub_312FD8+55
                push    0Dh
                call    sub_31411B

loc_31306D:                             ; CODE XREF: sub_312FD8
                                        ; 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_312FD8
                call    sub_3140F3
                and     eax, 0FFFFh
                or      eax, 80070000h

loc_31308C:                             ; CODE XREF: sub_312FD8
                mov     [ebp+var_4], eax
```

PWNING the Stack

Put x86 in buffer on the stack

Overwrite Return



Intro03 & Additional Reading

- There are multiple ways to solve intro03, we would like to see you use shellcode to solve it
- http://insecure.org/stf/smashstack.html
- We'll cover writing shellcode & more advanced forms of exploitation later this year