

# **Intro to Reversing**



# Before we begin...

- There will be a CTF this weekend
  - Codegate, a large Korean competition
  - starts Friday at 7am, lasts 36 hours
  - will post details on piazza
  - same deal as before (points on scoreboard + free food)
  - very good way to learn!

# Possible Reversing Goals:

- In order of difficulty:
  - Figure out which parts of code are important
  - Know roughly what each function is doing
  - Be able to re-implement functions
  - Generate accepting inputs for functions
  - Spot subtle bugs in functions

# Tools:

- IDA: free trial available
  - <https://www.hex-rays.com/products/ida/support/download.shtml>
  - It is quite expensive to buy....
- Radare2
  - Open source, somewhat similar to IDA
  - I've never really used it...
- Objdump
  - Only useful for short code segments
- Google!

# Reversing Strategy:

1. Figure out "type signature"
  - a. what are the args, how many are there, etc
  - b. gets easier as you identify more functions in a program
2. Identify local variables
  - a. what is in each stack/register slot
  - b. which variables are important?
  - c. **try to find magic numbers!**
3. Look at overall layout
  - a. what branches based on what values
  - b. which details are important
4. Sketch pseudocode

```

push    %ebp
mov     %esp,%ebp
push    %edi
push    %esi
push    %ebx
mov     0x8(%ebp),%ebx
mov     0xc(%ebp),%esi
mov     0x10(%ebp),%edi
mov     0x14(%ebp),%ecx
test    %ecx,%ecx
jle     0x2c
mov     $0x0,%eax
1b: movzbl (%edi,%eax,1),%edx
xor     (%esi,%eax,1),%dl
mov     %dl, (%ebx,%eax,1)
add     $0x1,%eax
cmp     %ecx,%eax
jne     0x1b
2c: movb   $0x0, (%ebx,%ecx,1)
pop     %ebx
pop     %esi
pop     %edi
pop     %ebp
ret

```

```
push    %ebp
mov     %esp,%ebp
push    %edi
push    %esi
push    %ebx
```

} function prologue

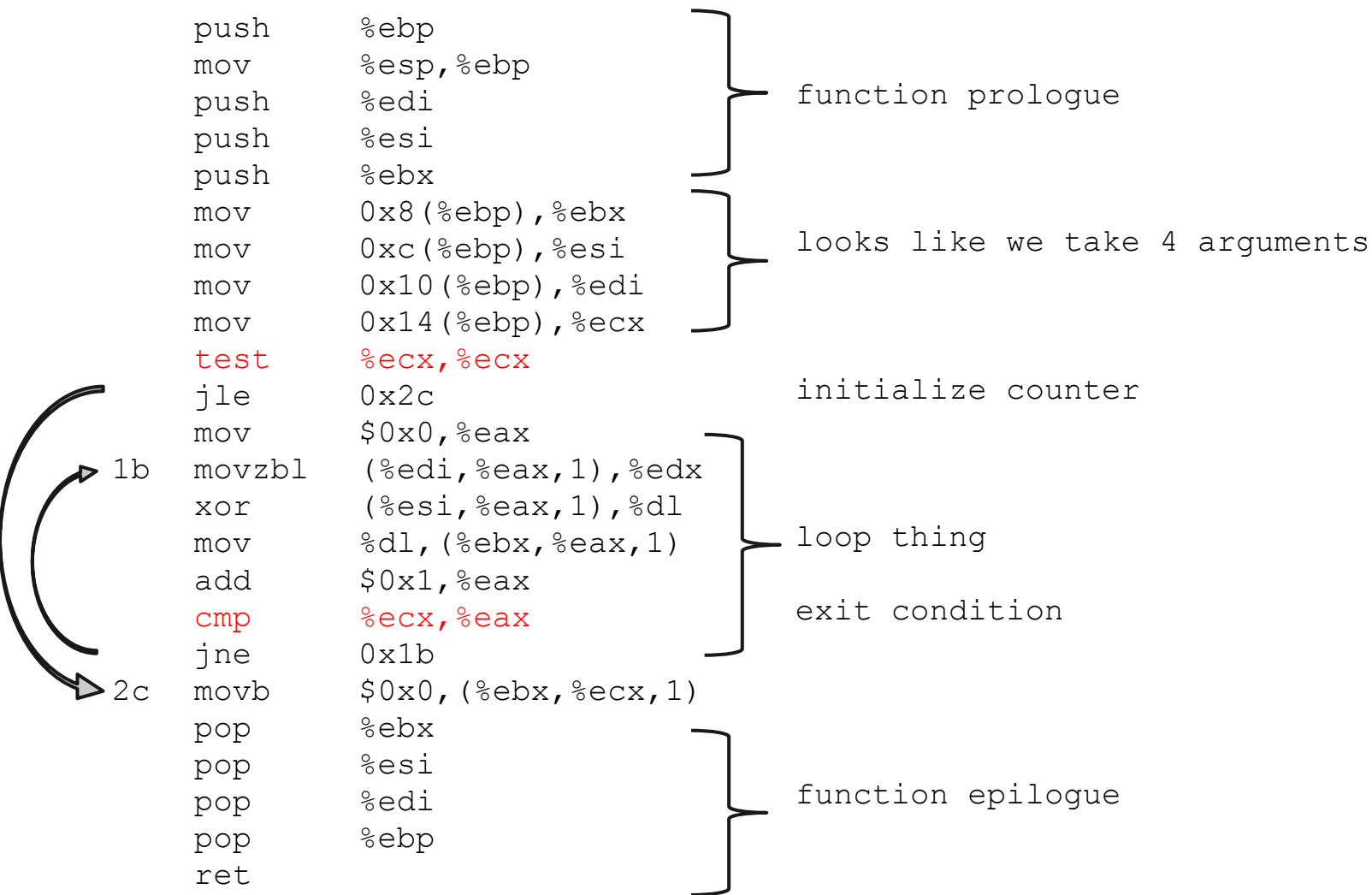
```
mov     0x8(%ebp),%ebx
mov     0xc(%ebp),%esi
mov     0x10(%ebp),%edi
mov     0x14(%ebp),%ecx
test    %ecx,%ecx
```

} looks like we take 4 arguments

```
jle     0x2c
mov     $0x0,%eax
1b: movzbl (%edi,%eax,1),%edx
xor     (%esi,%eax,1),%dl
mov     %dl, (%ebx,%eax,1)
add     $0x1,%eax
cmp     %ecx,%eax
```

```
jne     0x1b
2c: movb   $0x0, (%ebx,%ecx,1)
pop     %ebx
pop     %esi
pop     %edi
pop     %ebp
ret
```

} function epilogue



Assembly code with annotations and control flow arrows:

```

push    %ebp
mov     %esp,%ebp
push    %edi
push    %esi
push    %ebx
mov     0x8(%ebp),%ebx
mov     0xc(%ebp),%esi
mov     0x10(%ebp),%edi
mov     0x14(%ebp),%ecx
test    %ecx,%ecx
jle     0x2c
mov     $0x0,%eax
1b: movzbl (%edi,%eax,1),%edx
xor     (%esi,%eax,1),%dl
mov     %dl, (%ebx,%eax,1)
add     $0x1,%eax
cmp     %ecx,%eax
jne     0x1b
2c: movb   $0x0, (%ebx,%ecx,1)
pop     %ebx
pop     %esi
pop     %edi
pop     %ebp
ret

```

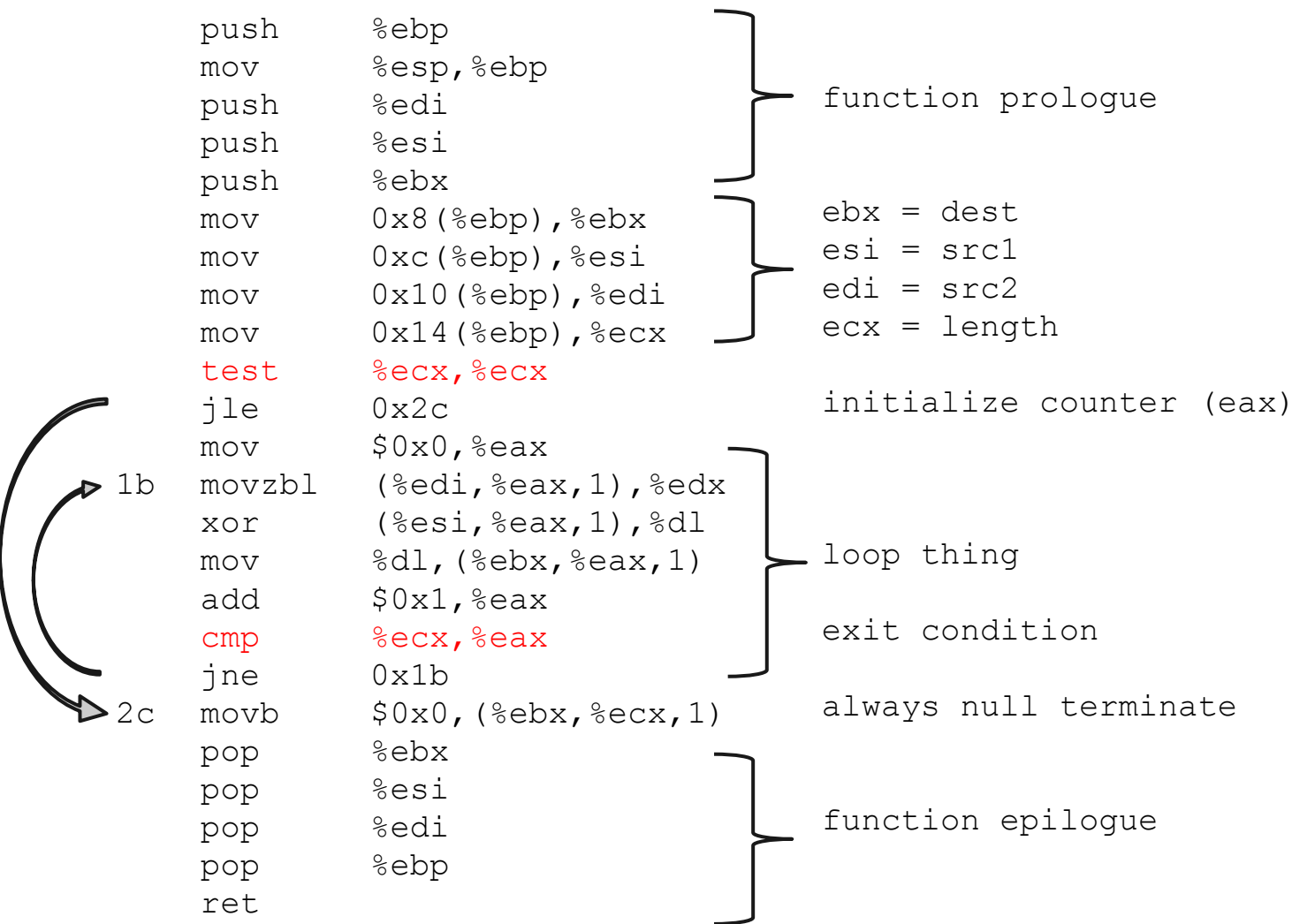
Annotations and Control Flow:

- function prologue**: Includes `push %ebp`, `mov %esp,%ebp`, `push %edi`, `push %esi`, and `push %ebx`.
- looks like we take 4 arguments**: Includes `mov 0x8(%ebp),%ebx`, `mov 0xc(%ebp),%esi`, `mov 0x10(%ebp),%edi`, and `mov 0x14(%ebp),%ecx`.
- initialize counter**: Includes `test %ecx,%ecx` and `jle 0x2c`.
- loop thing**: Includes `movzbl (%edi,%eax,1),%edx`, `xor (%esi,%eax,1),%dl`, and `mov %dl, (%ebx,%eax,1)`.
- exit condition**: Includes `add $0x1,%eax`, `cmp %ecx,%eax`, and `jne 0x1b`.
- function epilogue**: Includes `movb $0x0, (%ebx,%ecx,1)`, `pop %ebx`, `pop %esi`, `pop %edi`, `pop %ebp`, and `ret`.

Control flow arrows:

- An arrow from the `jne 0x1b` instruction to the `movzbl` instruction (labeled `1b`).
- An arrow from the `movb` instruction to the `ret` instruction (labeled `2c`).





Assembly code with annotations and control flow arrows:

```

push    %ebp
mov     %esp,%ebp
push    %edi
push    %esi
push    %ebx
mov     0x8(%ebp),%ebx
mov     0xc(%ebp),%esi
mov     0x10(%ebp),%edi
mov     0x14(%ebp),%ecx
test    %ecx,%ecx
jle     0x2c
mov     $0x0,%eax
1b: movzbl (%edi,%eax,1),%edx
xor     (%esi,%eax,1),%dl
mov     %dl, (%ebx,%eax,1)
add     $0x1,%eax
cmp     %ecx,%eax
jne     0x1b
2c: movb   $0x0, (%ebx,%ecx,1)
pop     %ebx
pop     %esi
pop     %edi
pop     %ebp
ret

```

Annotations and Control Flow:

- function prologue** (lines 1-4):
  - push %ebp
  - mov %esp,%ebp
  - push %edi
  - push %esi
- ebx = dest** (line 5):
  - mov 0x8(%ebp),%ebx
- esi = src1** (line 6):
  - mov 0xc(%ebp),%esi
- edi = src2** (line 7):
  - mov 0x10(%ebp),%edi
- ecx = length** (line 8):
  - mov 0x14(%ebp),%ecx
- initialize counter (eax)** (line 9):
  - test %ecx,%ecx
  - jle 0x2c
- loop thing** (lines 10-12):
  - mov \$0x0,%eax
  - 1b: movzbl (%edi,%eax,1),%edx
  - xor (%esi,%eax,1),%dl
  - mov %dl, (%ebx,%eax,1)
- exit condition** (line 13):
  - add \$0x1,%eax
  - cmp %ecx,%eax
- always null terminate** (line 14):
  - jne 0x1b
  - 2c: movb \$0x0, (%ebx,%ecx,1)
- function epilogue** (lines 15-18):
  - pop %ebx
  - pop %esi
  - pop %edi
  - pop %ebp
  - ret

Control flow arrows:

- From **1b** to **2c** (loop back)
- From **2c** to **1b** (loop back)

# Googling and reversing

- People copy code *a lot*
- Crypto code is rarely novel
- Strategy:
  - Find immutable details in the code
  - Google for them
  - Try to find something that implements your asm

**Your turn!**