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                    function prologue
     push    %esi
     push    %ebx
     mov     0x8(%ebp),%ebx
     mov     0xc(%ebp),%esi          looks like we take 4 arguments
     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                    function epilogue
     pop     %ebp
     ret
```

```
      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
```

function prologue

looks like we take 4 arguments

initialize counter

loop thing

exit condition

function epilogue

```
      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
```

function prologue

ebx = dest
esi = src1
edi = src2
ecx = length

initialize counter (eax)

loop thing

exit condition

always null terminate

function epilogue

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!