SIGPwny

General FA2026 • 2026-09-17

Reverse Engineering I

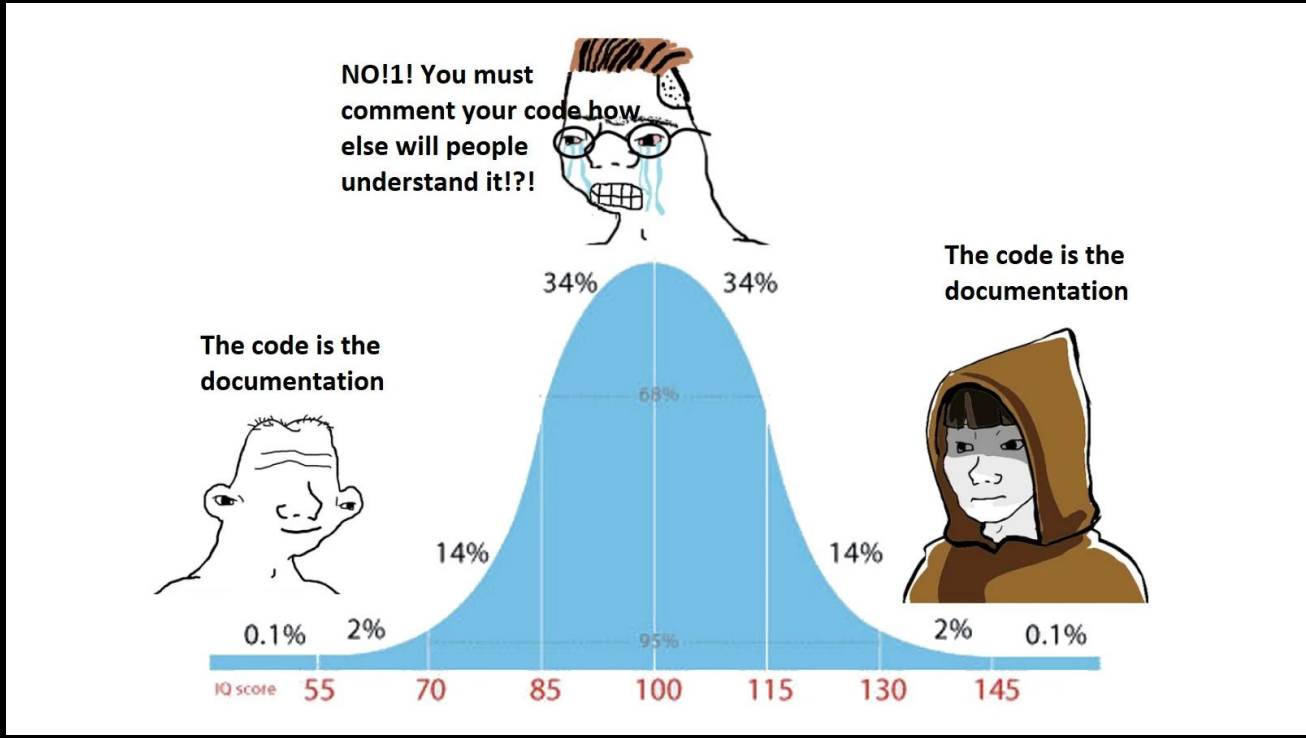
Juniper Peng

Announcements

- CSAW CTF Quals
  - Tomorrow, September 18th
  - All experience levels encouraged to join us, there will be free food!
- FallCTF 2026
  - Intro hacking competition run by SIGPwny!
  - September 27th, 12-6PM

ctf.sigpwny.com

sigpwny{its_open_source_if_you_try_hard_enough}



Which is easier to understand?

```python
def fibonacci(n):
    if n <= 0:
        return []
    elif n == 1:
        return [0]
    elif n == 2:
        return [0, 1]
    else:
        fib_sequence = [0, 1]
        while len(fib_sequence) < n:
            next_num = fib_sequence[-1] + fib_sequence[-2]
            fib_sequence.append(next_num)
        return fib_sequence
```

```python
def aaoaaaa04922(aa27619):
    aaoaaaoa20551 = -1
    aa276l9 = aa27619 + 1
    aa27618 = -aa27619
    if aa27618 > 0:
       return []
    elif not bool(aa276l9 - 2):
        return [] * aa27618
    elif aa27619 == 1:
        return [aa276l9-1]
    else:
        aaoaaaoa32021 = [0, 1]
        while True:
            if not (len(aaoaaaoa32021) < aa27619):
               break
            aaoaaaoa21049 = aaoaaaoa32021[-aaoaaaoa32021[1]]
            aaoaaaoa21049 += aaoaaaoa32021[aaoaaaoa20551**2 - 3]
            aaoaaaoa32021.append(aaoaaaoa21049)
        else:
            aaoaaa3322 = 23
            return [aaoaaa3322 + i for i in aaoaaaoa32021]
    return aaoaaaoa32021
```

Overview

- Basics
  - Motivation
  - Types of analysis
  - Abstraction levels
- Techniques
  - Common patterns
  - Tools
- Examples

Basics

What is reverse engineering?

Motivation

- Reverse engineering: reading other people's code
- Goal is to understand the code
  - The code is never "wrong" — it is the ultimate "documentation"
- Not all code is easy to read or well-documented
- Sometimes code is intentionally hard to understand (i.e. obfuscated)

Static vs Dynamic Analysis

- Static Analysis
  - Reading code
  - Using tools to understand code

- Dynamic Analysis
  - Running code
  - Inspecting program state while it is running

More helpful if...
- Code is simple - Code is hard to run

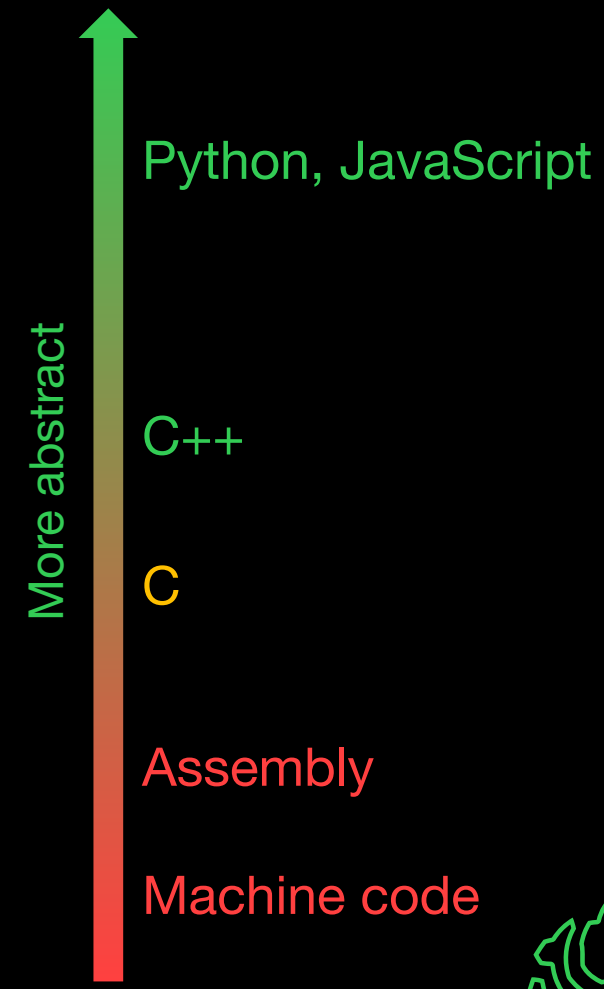
More helpful if...
- Code is complex - Useful data in memory

Static and and dynamic analysis are not a dichotomy! Use them together!

Abstraction Levels

- High level
  - Python, JavaScript, etc.
  - Easy to analyze

- Low level
  - C, assembly, etc.
  - Harder to analyze
  - More details about machine-specific behavior
  - Everything is ran as machine code at some point



Techniques

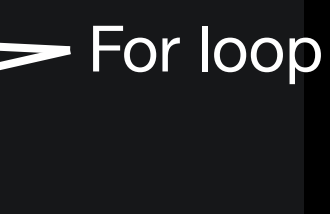
How to reverse engineer?

Static Analysis

- Function rewriting
  - Simplify complex portions of code
- Find known algorithms/patterns
- Decompilers
  - Automatically extract abstractions from low level programs
  - Turn assembly into more readable C
  - Will be covered in depth in Reverse Engineering II meeting
- Deobfuscation
  - Renaming functions and variables with more descriptive names
  - Using heuristics to figure out what was originally meant

Example: Common patterns

```
arr = [0] * 10
i = 9
while i >= 0:
  arr[i] = i * 2
  i -= 1
```



Can we simplify this code?

Example: Common patterns

```python
arr = [0] * 10
for i in range(9, 0 - 1, -1):
  arr[i] = i * 2
```

Order doesn't matter here

Simplify even more?

Example: Common patterns

```
arr = [0] * 10
for i in range(0, 10):
  arr[i] = i * 2
```

Even simpler? Use list comprehension

Example: Common patterns

```
arr = [i*2 for i in range(0, 10)]
```

Example: Common patterns

Which level of simplification is the most useful? Depends

```python
arr = [0] * 10
i = 9
while i >= 0:
  arr[i] = i * 2
  i -= 1
```

```python
arr = [0] * 10
for i in range(9, 0 - 1, -1):
  arr[i] = i * 2
```

```python
arr = [0] * 10
for i in range(0, 10):
  arr[i] = i * 2
```

```python
arr = [i*2 for i in range(0, 10)]
```

Dynamic Analysis

- Partial evaluation
  - Evaluate small portions of the code to reduce complexity
  - Observe behavior, such as variables

- Modifying programs
  - Add or remove code
  - Add print statements
  - "Patching" binaries

Advanced Dynamic Analysis

- Debuggers (Reverse Engineering II)
  - `gdb`, `pdb`

- Side channels (Reverse Engineering III)
  - Instruction counting, time counting

Example: Modifying Code

```python
f = input('What is the flag? ')

va = [1751, 1649, 1836, 1734]
arr = ''.join([chr(va[len(va)-1-i]//17) for i in range(len(va))])

if f[0] != 'f':
    print("That's definitely wrong.")
else:
    for it in range(len(arr)-1, -1, -1):
        if arr[it] != f[it]:
            print('Wrong flag!')
            exit(1)
```

Example: Modifying Code

```python
f = input('What is the flag? ')

va = [1751, 1649, 1836, 1734]
arr = ''.join([chr(va[len(va)-1-i]//17) for i in range(len(va))])

print(arr)

if f[0] != 'f':
    print("That's definitely wrong.")
else:
    for it in range(len(arr)-1, -1, -1):
        if arr[it] != f[it]:
            print('Wrong flag!')
            exit(1)
```

Patch in a print

```
$ python3 test.py
What is the flag? aaaa
flag
That's definitely wrong.
```

Example: Reverse Evaluation

```
q = input('What letter am I thinking of? ')
q = ord(q)
q *= 7
w = list(str(q))
w.reverse()
for i in range(len(w)):
    w[i] += str(i)
if w == ['30', '91', '62']:
    print('Success')
else:
    print('Wrong')
```

q = 'c'

q = 99

q = 693

w = ['6', '9', '3']

w = ['3', '9', '6']

w = ['30', '91', '62']

Example: Modifying Code 2

```
import string
for l in string.ascii_letters:
    q = l
    q = ord(q)
    q *= 7
    w = list(str(q))
    w.reverse()
    for i in range(len(w)):
        w[i] += str(i)
    if w == ['30', '91', '62']:
        print('Success')
        print(l)
```

What if we tried them all?

```
Success
c
```

Go try challenges!

- Go to **ctf.sigpwny.com**
- Start with **Python RE 1: Easy rev**

- If you don't have Python installed, see slides from setup meeting (Intro to Terminal and Setup)

Next Meetings

`2026-09-20` • **This Sunday**

- x86-64 Assembly
- Learn the fundamentals of x86-64 Assembly, including the stack, memory, registers, instructions, and syscalls.
- **Very important** meeting to prepare for Rev II.

`2026-09-24` • **Next Thursday**

- Reverse Engineering II
- Learn to reverse engineering x86-64 binaries!

ctf.sigpwny.com

sigpwny{its_open_source_if_you_try_hard_enough}

Meeting content can be found at sigpwny.com/meetings.

SIGPwny