Berkeley SCHOOL OF INFORMATION

CTF 2

Full Name: Jason Chow

Date: 9/22/2022

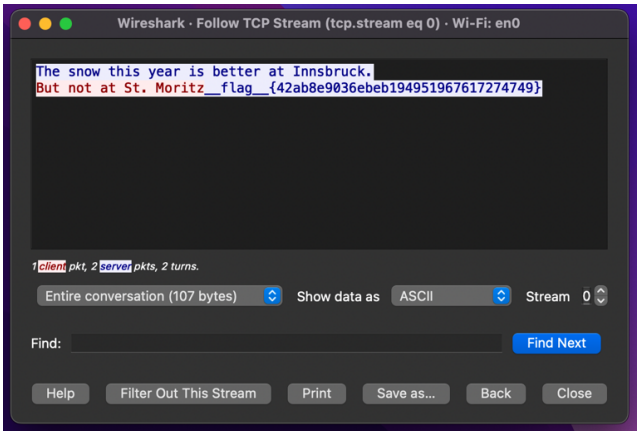
Challenge Name: For Your Eyes Only

Sockets enable bi-directional communication between a client (my computer) and a server (the CTF system). For this challenge, a socket connection is established from my computer to the CTF system, in order to send a secret message - "But not at St. Moritz". If, and only if, the CTF system receives the correct secret message, the system respond with a flag.

Figure 1: Source code to send message

```
In [28]:
####################
# For Your Eyes Only
####################

# This stanza imports the socket library and
# establishes an IPv4 socket connection with
# the CTF system over port 8001.
from socket import *
s = socket(AF_INET, SOCK_STREAM)
s.connect(("w210.network",8001))

# This stanza receives data from the CTF system,
# then sends secret message before closing socket
while True:
    data = s.recv(1024).decode()
    print(data)
    if not data:
        quit()
    else:
        s.send(b"But not at St. Moritz") # Secret message encoded in byte stream
        break # Break out of loop

s.close() # Closes socket

The snow this year is better at Innsbruck.
```

Figure 2: TCP trace showing response/flag



SCHOOL OF INFORMATION, UNIVERSITY OF CALIFORNIA
102 SOUTH HALL #4600, BERKELEY, CA 94720-4600
PH 510.642.1464 FAX 510.642.5814
ISCHOOL.BERKELEY.EDU

Challenge Name: Roman Reverse Captcha

The Roman Reverse Captcha challenge builds on the prior challenge. However, rather than sending one secret message, the client (my computer) must receive, process and respond to multiple dynamically generated messages (math equations) from the CTF system. If, and only if, the CTF system receives the correct answers for each set of equations, the system respond with a flag.

Figure 1: TCP trace showing equations/flags

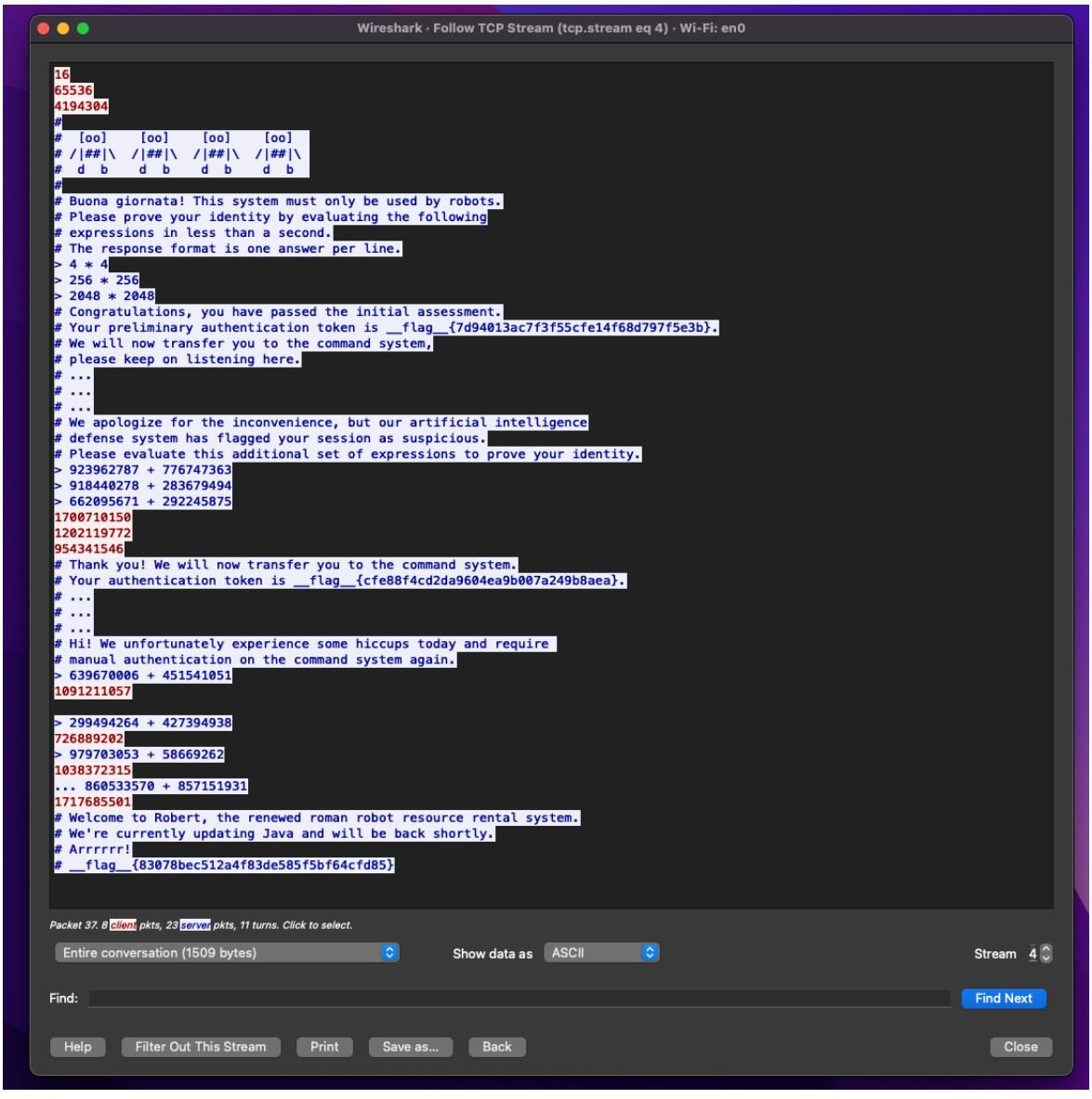


Figure 2: Source code

In [24]:

```
########################
# Roman Reverse Captcha
########################

# This stanza imports the socket / regular expression
# libraries, and establishes an IPv4 socket
# connection with the CTF system over port 8002.
import re
from socket import *
s = socket(AF_INET, SOCK_STREAM)
s.connect(("w210.network", 8002))

# This stanza assumes you've seen first set of equations
# and sends answers back to CTF system.
s.send(b"16\n")
s.send(b"65536\n")
s.send(b"4194304\n")

while True:

    # This stanza receives data from CTF system and searches
    # for instances of the word 'identity' and 'hiccups', which
    # are located in the second and third set of equations.
    # The index design is used for subsequent code to properly
    # capture set of equations, perform calculations, then send
    # answer back to CTF system.
    data = s.recv(1024).decode()
    identity_index = data.find("identity")
    hiccup_index = data.find("hiccups")
    print(data)

    # Executes code to handle second set of equations.
    # The word 'identity' only appears in second challenge.
    if identity_index > 193:

        # Searches for + operator in equation.
        digit = re.findall(r'\d+', data)

        # Initializes variables with elements of equation.
        element1 = digit[0]
        element2 = digit[1]
        element3 = digit[2]
        element4 = digit[3]
        element5 = digit[4]
        element6 = digit[5]

        # Transform element to integers, then adds them together.
        equation1 = int(element1) + int(element2)
        equation2 = int(element3) + int(element4)
        equation3 = int(element5) + int(element6)

        # Appends \n character to each equation.
        message4 = str(equation1) + '\n'
        message5 = str(equation2) + '\n'
        message6 = str(equation3) + '\n'

        # Sends formatted answers back to CTF system.
        s.send(message4.encode())
        s.send(message5.encode())
        s.send(message6.encode())

    # Executes code to handle third set of equations.
    # The word 'hiccups' only appears in third challenge.
    if hiccup_index > 38:
```

```
# Executes code to handle third set of equations.
# The word 'hiccups' only appears in third challenge.
if hiccup_index > 38:

    # This code segment handles the first equation.
    # The code gets chunks of data from CTF system,
    # formats the data so the digits can be added,
    # then sends the answer back to CTF system.
    fragments = []
    done = False

    while (not done):
        chunk = s.recv(1024).decode()
        if ('\n' in chunk): # Removes \n character
            done = True
            break
        else:
            fragments.append(chunk) # Appends chunk to list

    message = "".join(fragments) # Re-assembles digits
    words = message.split()
    element1 = words[1]
    element2 = words[3]
    equation4 = int(element1) + int(element2) # Adds numbers
    message1 = str(equation4) + '\n' # Adds \n character
    s.send(message1.encode()) # Sends answer to CTF system

    # This code segment handles the second equation.
    # The code performs the same logic as described
    # for equation one above.
    fragments = []
    done = False
    chunk = ""
    message = ""
    words = ""
    element1 = ""
    element2 = ""
    message1 = ""

    while (not done):
        chunk = s.recv(1024).decode()
        print('')
        if ('\n' in chunk):
            done = True
            break
        else:
            fragments.append(chunk)

    message = chunk
    words = message.split()
    element1 = words[1]
    element2 = words[3]
    equation4 = int(element1) + int(element2)
    message1 = str(equation4) + '\n'
    s.send(message1.encode())

    # This code segment handles the third equation.
    # The code performs the same logic as described
    # for equation one above.
    fragments = []
    done = False
    chunk = ""
    message = ""
    words = ""
    element1 = ""
    element2 = ""
    message1 = ""
```

```
        while (not done):
            chunk = s.recv(1024).decode()
            print('')
            if ('\n' in chunk):
                done = True
                break
            else:
                fragments.append(chunk)

        message = chunk
        words = message.split()
        element1 = words[1]
        element2 = words[3]
        equation4 = int(element1) + int(element2)
        message1 = str(equation4) + '\n'
        s.send(message1.encode())

        # This code segment handles the fourth equation.
        # The code performs the same logic as described
        # for equation one above.
        fragments = []
        done = False
        chunk = ""
        message = ""
        words = ""
        element1 = ""
        element2 = ""
        message1 = ""

        while (not done):
            chunk = s.recv(1024).decode()
            if ('\n' in chunk):
                done = True
                break
            else:
                fragments.append(chunk)

        message = chunk
        words = message.split()
        element1 = words[1]
        element2 = words[3]
        equation4 = int(element1) + int(element2)
        message1 = str(equation4) + '\n'
        s.send(message1.encode())

s.close() # Closes socket connection.
```