Command and Control

5062CEM: Programming and Algorithms 2

https://github.coventry.ac.uk/ivanovn/C2

Student ID: 10223282
Date: 03/12/2021

Table of Contents

User Documentation

Installation

Installing C2 with virtual enviremont

To install C2.py with a virtual environment, you can take advantage of the "Makefile". The following commands will set the environment and install the required modules.

```
~/$ make venv
~/$ . ./venv/bin/activate
~/$ make prereqs
```

The alternative python commands are:

```
~/$ python3 -m venv venv
~/$ . ./venv/bin/activate
~/$ pip3 install -r requirements.txt
```

Installing C2 on local basis

In case you want to install C2.py on a local basis, you can use just the pip3 command, which will make sure that all required modules are installed.

```
~/$ pip3 install -r requirements.txt
```

Installing the client

The installation of the client sonsista of simply pasting it and running it.

Configuration

The configuration of the server consists of setting a key, certificate, password for the key and port on which the server is supposed to work. The server comes with a default key and certificate located in the same directory as the server. The default port is 9090. To change them, you can either open the server.py file and set new ones in the Server class or change them from the GUI.

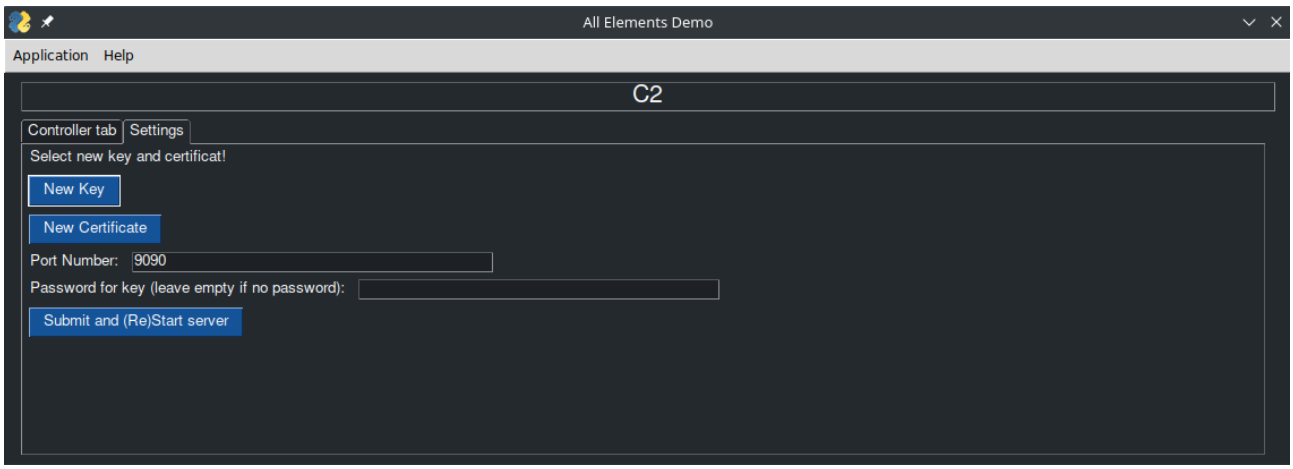
Changing the default settings

Open the server.py file and change the following line to the new values:

```
...
def __init__(self, port=9090, cert="./cert.pem", key="./key.pem",
paswd="bleh", clients={}):
...
```

Changing the setting from the GUI

If you want to change the setting from the GUI you simple need to go on the Setting tab, put the wanted values and click on “Submit and (Re)Start server”.



This will start or restart the server and set the new values.

FYI:
It is a demo feature. To not break the server, all the values need to be set.

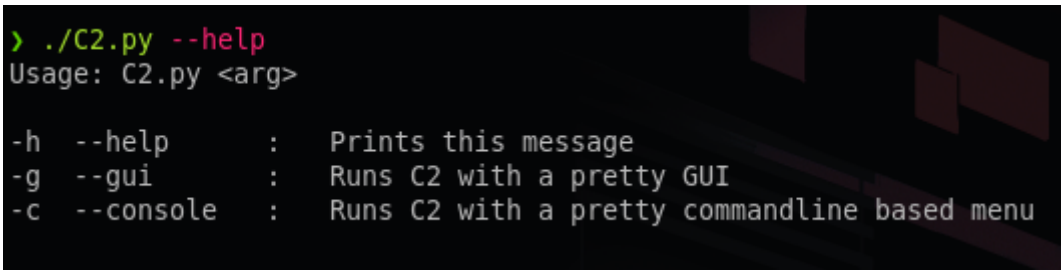
Changing the client settings

Similarly to changing the default server setting, to change the client setting, you need to open the source code and set the new server IP address, port and certificate. The default values are in the initial class function.

```
...
def __init__(self, host = '127.0.0.1', port = 9090, cert = './cert.pem'):
...
```

Interfaces

There are two interfaces: a text-based menu that can be used in the terminal and a graphical user interface (GUI). To select an interface, an argument needs to be passed to the C2.py file.
“-g” or “--gui” will open the GUI and “-c” or “--console” will prompt the text-based menu.

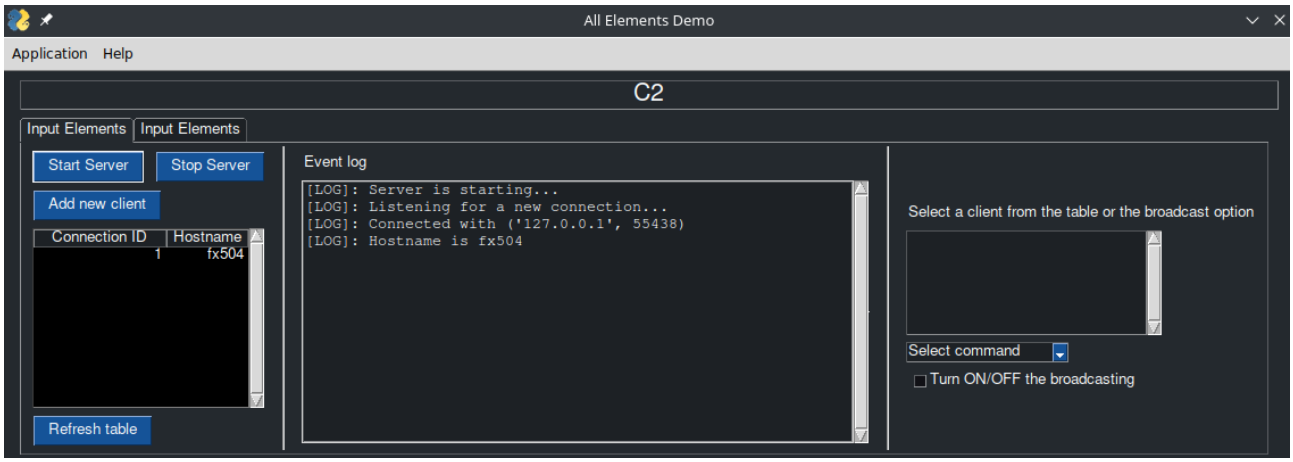


GUI

The GUI has two main tabs. One is for the setting and one for the application programming interface (API). The setting tab was discussed in the previous topic. The API tab has three columns. The left one is for controlling the server, the right one for controlling the clients and the central one is for logs and output of the commands.

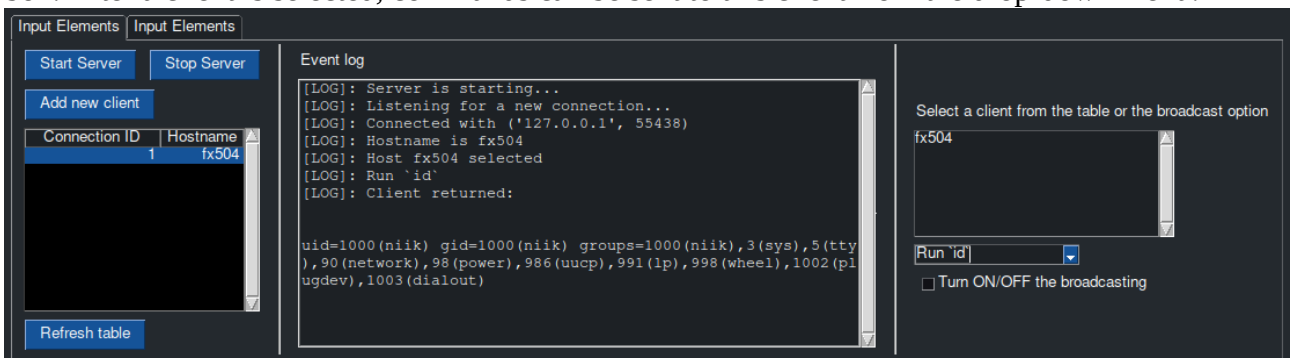
Left Column

The left column is self-explanatory. It has four buttons and one table. The first two buttons are for running or stopping the server. The third one is for adding a new connection. The last button refreshes the table in which all the client's ID's and hostnames are displayed. Once you have any clients on the table, you can select the required one.

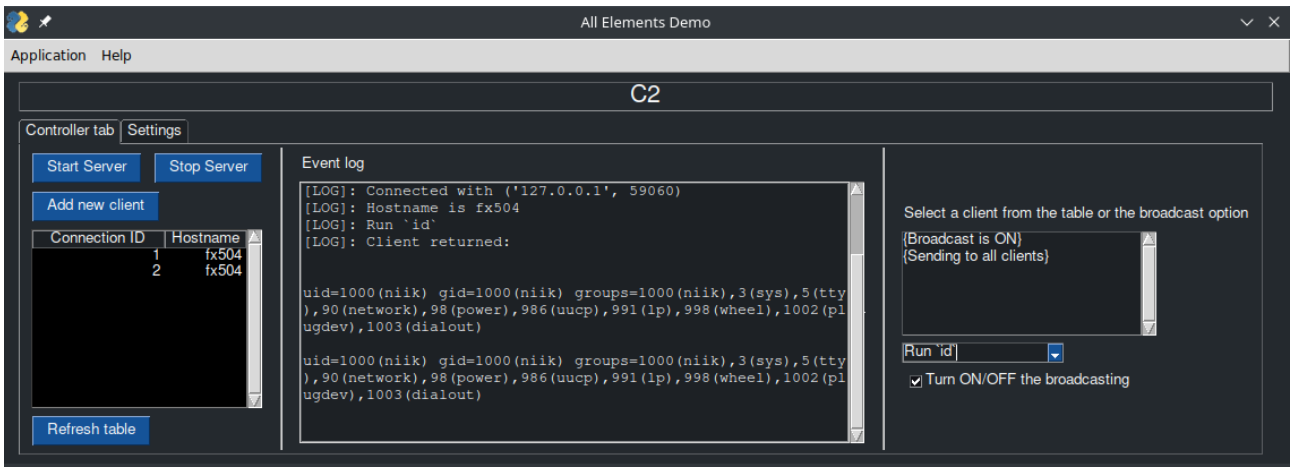


Right Column

After selecting a client from the table from the left column, the client will be displayed in the text box. After a client is selected, commands can be sent to this client from the drop-down menu.



If needed, the broadcast checkbox can be ticked, and commands will be sent to all clients.



Text-based Menu

The menu has the same functions as the GUI, except for the settings.The options are listed in the following order:

[s] Start server

[a] Add new client

[l] List connected clients

[i] Run `id`

[p] Run `ls`

[p] Run `pwd`

[p] Run `ip a`

[d] Disconnect client

[b] Set broadcast ON/OFF

[k] Stop server

[q] Quit

Every option is self-explanatory and very similar to the options from the GUI.

```
❯ ./C2.py -c                                                                    at ⊙ 04:14:12 pm
[LOG]: Server is starting...
[LOG]: Listening for a new connection...
[LOG]: Connected with ('127.0.0.1', 59052)
[LOG]: Hostname is fx504
[LOG]: Listening for a new connection...
[LOG]: Connected with ('127.0.0.1', 59054)
[LOG]: Hostname is fx504
----------
[CON]: 1: fx504
[CON]: 2: fx504
----------
[LOG]: Running `id` on [1] fx504
uid=1000(niik) gid=1000(niik) groups=1000(niik),3(sys),5(tty),90(network),98(power),986(uucp),991(lp),998(wheel),1002(p
lugdev),1003(dialout)

[LOG]: Broadcast is ON
[LOG]: Running `id` on all clients
uid=1000(niik) gid=1000(niik) groups=1000(niik),3(sys),5(tty),90(network),98(power),986(uucp),991(lp),998(wheel),1002(p
lugdev),1003(dialout)

uid=1000(niik) gid=1000(niik) groups=1000(niik),3(sys),5(tty),90(network),98(power),986(uucp),991(lp),998(wheel),1002(p
lugdev),1003(dialout)

[LOG]: fx504 disconnected!
[LOG]: fx504 disconnected!
[LOG]: Server is stopping...
> [s] Start server
  [a] Add new client
  [l] List connected clients
  [i] Run `id`
  [p] Run `ls`
  [p] Run `pwd`
  [p] Run `ip a`
  [d] Disconnect client
  [b] Set broadcast ON/OFF
  [k] Stop server
  [q] Quit
```

A small difference in the way of running commands on the client. Unlike the GUI, in the menu, you first select the command which then prompts you to s submenu of all the connected clients. If no clients are connected the submenu is empty and the command cannot be sent.

```
❯ ./C2.py -c
[LOG]: Server is starting...
[LOG]: Listening for a new connection...
[LOG]: Connected with ('127.0.0.1', 34312)
[LOG]: Hostname is fx504
[LOG]: Listening for a new connection...
[LOG]: Connected with ('127.0.0.1', 34314)
[LOG]: Hostname is fx504
> [1] fx504
  [2] fx504
```

A few more words

Do not worry about stopping the server while having connected clients. The API will disconnect all clients before stopping the server. Furthermore, in case the broadcast is one and you select the disconnect command, all clients will get disconnected.

Developer Documentation

C2.py

If new interfaces are implemented, just import the interface, add a message for it and include a new “elif” statement.

gui.py

The PySimpleGUI module is used for building the graphical interface. The layout is in the function “gui_layout”. If new commands are added to the server and the controller, simply add the description of the command in the commands list in the function and then add a new “elif” statement in the “-COMMAND-” event in the GUI function.

If new tab layouts are being added, simply add them in the layout variable:

```
...
layout += [[sg.TabGroup([[ sg.Tab('Controller tab', main_layout),
sg.Tab('Settings', ssl_layout)]], key='-TAB GROUP-')]]
...
```

menu.py

The menu is located in the menu function. In case new commands are being added, simply add a tuple in the options list. The tuple needs to consist of the command description as well as the command. After this, just edit the “menuIndex” number of the options in the loop.

server.py

In order to add a new command to the server, simply add the command to this list in the API function in the server class. If the command is added to the client, the client will execute it.

```
...
elif message in ["ID", "LS", "PWD", "IP"]:
if destination == "":
print("[ERR]: No client specified!")
else:
d = destination.get()
if d == "ALL":
self.broadcast(message)
else:
d[1].send(message.encode('ascii'))
...
```

client.py

To add a new command in the client, you need to add the alias that is going to be received from the server and then link it to the actual command that needs to be executed.

```
...
# Sending result from list of commands
elif message in ['ID', 'LS', 'PWD', 'IP']:
command = {'ID': 'id', 'LS': 'ls', 'PWD': 'pwd', 'IP': 'ip a'}
msg = self.run_command(command[message])
client.send(msg.encode('ascii'))
...
```

Reference List

Seitz. J. (2014). *Black Hat Python: Python Programming for Hackers and Pentesters*. No starch press

Appendix

Server-Side

C2.py

```
#!/bin/python3
#!/usr/bin/env python3
#import socket
#import ssl
#import sys
#import threading
#from queue import Queue
#from server import *
#from time import sleep
#from simple_term_menu import TerminalMenu
#import PySimpleGUI as sg
import sys
from menu import *
from gui import *

def arg_choose():
USAGE="""Usage: C2.py <arg>

-h --help : Prints this message
-g --gui : Runs C2 with a pretty GUI
-c --console : Runs C2 with a pretty commandline based menu
"""

try:
argv = sys.argv[1]
except:
print("ERROR!")
print(USAGE)
sys.exit(2)
if argv == '-h' or argv == "--help": print(USAGE)
elif argv == '-g' or argv == "--gui": gui_main()
elif argv == '-c' or argv == "--console": menu_main()
else: print(USAGE)

if __name__ == '__main__':
arg_choose()
```

gui.py

```
#//#!/usr/bin/env python3
#import socket
#import ssl
#import sys
#import threading
#from queue import Queue
from server import *
from time import sleep
import PySimpleGUI as sg


# Returns a list of all hostnames from all clients
def host_list():
hosts=[]
ls = Server().client_ls()
for i in ls:
hosts.append([i, ls[i][0]])
return hosts

# Returns a list of all clients
def cl_list():
hosts=[]
ls = Server().client_ls()
for i in ls:
hosts.append([i, ls[i][1]])
return hosts

# Sends a command to the client from the server and reads the received
output
def run_command(msg, command, client, q_dest, q_msg, q_data):
print(f"[LOG]: {msg}")
q_dest.put(client)
q_msg.put(command)
sleep(0.1)
print(f"[LOG]: Client returned:\n\n")
Server().read_data(q_data)

def gui_layout():
data=[['', '']]
commands=['Run `id`', 'Run `ls`', 'Run `pwd`', 'Run `ip a`', 'Disconnect
client']
sg.theme("DarkGrey14")
headings = ["Connection ID", "Hostname"]

menu_def = [['Application', ['&Exit']],
['Help', ['&About']] ]
```

```
right_click_menu_def = [[], ['Exit']]

controll_col = [[sg.Menu(menu_def, key='-MENU-')],
[sg.Button('Start Server'), sg.Button('Stop Server'),],
[sg.Button('Add new client')],
[sg.Table(values=data,
select_mode=sg.TABLE_SELECT_MODE_BROWSE,
headings=headings,
background_color='black',
justification='right',
auto_size_columns=True,
num_rows=10,
enable_events=True,
key='-TABLE-')],
[sg.Button('Refresh table')]]
logging_col = [[sg.Text("Event log")], [sg.Output(size=(60,15),
font='Courier 10')]]

client_col = [[sg.Text("Select a client from the table or the broadcast
option")],
[sg.Listbox(values=[], key="-CLIENT-", size=(30, 6))],
[sg.Combo(values=commands, default_value="Select command",
enable_events=True, key="-COMMAND-")],
[sg.Checkbox("Turn ON/OFF the broadcasting", enable_events=True, key="-
BROADCAST-")]]

main_layout = [[sg.Column(controll_col), sg.VSeperator(),
sg.Column(logging_col), sg.VSeperator(), sg.Column(client_col)]]

ssl_layout = [[sg.Text("Select new key and certificat!")],
[sg.Button("New Key")],
[sg.Button("New Certificate")],
[sg.Text("Port Number:"), sg.Input(enable_events=True, default_text=9090,
key='-PORT-')],
[sg.Text("Password for key (leave empty if no password):"),
sg.Input(password_char="*", enable_events=True, key='-PASS-')],
[sg.Button('Submit and (Re)Start server')]]


layout = [[sg.Text('C2', justification='center', size=(92, 1),
font=("Helvetica", 16), relief=sg.RELIEF_RIDGE, k='-TEXT HEADING-',
enable_events=True)]]
layout += [[sg.TabGroup([[ sg.Tab('Controller tab', main_layout),
sg.Tab('Settings', ssl_layout)]], key='-TAB GROUP-')]]

return sg.Window('C2', layout, right_click_menu=right_click_menu_def)

def gui(q_data, q_msg, q_dest):
port=9090
```

```
cert="./cert.pem"
key="key.pem"
paswd="bleh"

server=""
data = host_list()
client = ""
window = gui_layout()
while True:
event, values = window.read()

# Exiting options
if event in (None, 'Exit') or event == sg.WIN_CLOSED:
if server != "":
q_dest.put(server)
q_msg.put("KILLSURV")
server=""
q_msg.put("EXIT")
print("[LOG]: Quitting...")
break

# Shows an about page
elif event == 'About':
print("[LOG]: Clicked About!")
sg.popup("""BLEH""")

# Starts the server
elif event == "Start Server":
if server != "":
print("[INFO]: Server is already running!")
else:
server = Server(port, cert, key, paswd).start()

# Stops the server
elif event == "Stop Server":
if server != "":
q_dest.put(server)
q_msg.put("KILLSURV")
server=""

# Starts listenning for a new client
elif event == "Add new client":
if server != "":
q_dest.put(server)
q_msg.put("NEWCL")
else: print("[INFO]: Server is not running!")

# Refreshes the table in case of new connection, or disconnecting
elif event == 'Refresh table':
```

```
data = host_list()
window.Element('-TABLE-').update(values=data)
if client == "ALL":
window.Element('-CLIENT-').update(values=[["Broadcast is ON"], ["Sending to
all clients"]])
else:
window.Element('-CLIENT-').update(values=[])
client = ""

# Setting a target client as long as the broadcast is OFF
elif event == "-TABLE-" and client != "ALL":
data_selected = values[event]
if len(data_selected) > 0:
ds = data_selected[0]
client = cl_list()
client = client[ds]
print(f"[LOG]: Host {data[ds][1]} selected")
window.Element('-CLIENT-').update(values=[data[ds][1]])

# Setting broadcast to ON/OFF
elif event == "-BROADCAST-":
if values[event] == True:
client="ALL"
window.Element('-CLIENT-').update(values=[["Broadcast is ON"], ["Sending to
all clients"]])
elif values[event] == False:
client=""
window.Element('-CLIENT-').update(values=[])

# Running a selected command
elif event == "-COMMAND-":
if server != "":
msg = values[event]
if client != "":
if msg == 'Run `id`':
command = "ID"
run_command(msg, command, client, q_dest, q_msg, q_data)
elif msg == "Run `ls`":
command = "LS"
run_command(msg, command, client, q_dest, q_msg, q_data)
elif msg == "Run `pwd`":
command = "PWD"
run_command(msg, command, client, q_dest, q_msg, q_data)
elif msg == "Run `ip a`":
command = "IP"
run_command(msg, command, client, q_dest, q_msg, q_data)
elif msg == "Disconnect client":
command = "DISSCL"
run_command(msg, command, client, q_dest, q_msg, q_data)
```

```
client = ""

# Selecting a new key
elif event == "New Key":
key = sg.popup_get_file('Choose new key')
key = str(key)

# Selecting a new certificate
elif event == "New Certificate":
cert = sg.popup_get_file('Choose new certificat')
cert = str(cert)

# (Re)Starts the server after new key, cert, port and password are given
elif event == "Submit and (Re)Start server":
print(f"[LOG]: New key: {key}\n")
print(f"[LOG]: New cert: {cert}\n")
port = int(values['-PORT-'])
print(f"[LOG]: New port: {port}\n")
paswd = str(values['-PASS-'])
print(f"[LOG]: New pass: {print('*'*len(paswd))}")

if server != "":
q_dest.put(server)
q_msg.put("KILLSURV")
server = Server(port, cert, key, paswd).start()

window.close()
exit(0)

# Creates the required queues, starts an API thread and runs the GUI
def gui_main():
q_data, q_msg, q_dest = Queue(), Queue(), Queue()
API_thread = threading.Thread(target=Server().API, name="API", args=(q_msg,
q_data, q_dest))
API_thread.start()

gui(q_data, q_msg, q_dest)


if __name__ == '__main__':
gui_main()
```

menu.py

```
#!/usr/bin/env python3
#import socket
#import ssl
#import sys
#import threading
#from queue import Queue
from server import *
from time import sleep
from simple_term_menu import TerminalMenu

# Quits the program.
def quit():
print("[LOG]: Quitting...")
sys.exit(0)

# Gets all the clients listed on the server and returns them in appropriate
format
def cl_list():
options=[]
ls = Server().client_ls()
for i in ls:
options.append((f"[{i}] {ls[i][0]}", i, ls[i][1]))
return options


def menu(q_data, q_msg, q_dest):
# Menu layout
options = [("[s] Start server",),
("[a] Add new client",),
("[l] List connected clients",),
("[i] Run `id`", "ID"),
("[p] Run `ls`", "LS"),
("[p] Run `pwd`", "PWD"),
("[p] Run `ip a`", "IP"),
("[d] Disconnect client", "DISSCL"),
("[b] Set broadcast ON/OFF",),
("[k] Stop server",),
("[q] Quit",)]


tm = TerminalMenu([i[0] for i in options])
server=""
broadcast=False
while True:
menuIndex = tm.show()
# Option for starting the server
if menuIndex==0:
```

```
if server != "":
print("[INFO]: Server is already running!")
else:
server = Server().start()

# Adding a new client option
elif menuIndex==1:
if server != "":
q_dest.put(server)
q_msg.put("NEWCL")
else: print("[INFO]: Server is not running!")

# Option for listing all connected clients
elif menuIndex==2:
if server != "":
q_msg.put("CONS")
else: print("[INFO]: Server is not running!")

# API commands option
elif menuIndex in [3, 4, 5, 6, 7]:
log = {3: "Running `id` on",
4: "Running `ls` on",
5: "Running `pwd` on",
6: "Running `ip a` on",
7: "Disconnecting from"}

# Submenu for all clients (not displaied in case the broadcast is on)
if server != "":
if broadcast == False:
opt = cl_list()
si = TerminalMenu([i[0] for i in opt])
subIndex = si.show()
if subIndex!=None:
print(f"[LOG]: {log[menuIndex]} {opt[subIndex][0]}")
q_dest.put((opt[subIndex][1], opt[subIndex][2]))
q_msg.put(options[menuIndex][1])
sleep(0.1)
Server().read_data(q_data)
elif broadcast == True:
print(f"[LOG]: {log[menuIndex]} all clients")
q_dest.put("ALL")
q_msg.put(options[menuIndex][1])
sleep(0.1)
Server().read_data(q_data)

else: print("[INFO]: Server is not running!")

# Broadcast option (sets a variable to True or False in order to turn ON or
OFF the broadcast)
```

```
elif menuIndex == 8:
if broadcast == False:
broadcast = True
br = "ON"
elif broadcast == True:
broadcast = False
br = "OFF"
print(f"[LOG]: Broadcast is {br}")

# Stop server option
elif menuIndex == 9:
if server != "":
q_dest.put(server)
q_msg.put("KILLSURV")
server=""
else: print("[INFO]: Server is not running!")

# Exit option
elif menuIndex==10:
if server != "":
q_dest.put(server)
q_msg.put("KILLSURV")
server=""
q_msg.put("EXIT")
quit()

# Creates the required queues, starts an API thread and runs the menu
def menu_main():
q_data, q_msg, q_dest = Queue(), Queue(), Queue()
API_thread = threading.Thread(target=Server().API, name="API", args=(q_msg,
q_data, q_dest))
API_thread.start()

menu(q_data, q_msg, q_dest)


if __name__ == '__main__':
menu_main()
```

server.py

```
#!/usr/bin/env python3
import socket
import ssl
import sys
import threading
from queue import Queue

class Server():
'''Server class managing the connection on the controller-side.

Funcs:
__init__ : Initialising the class with default values port and cert, key,
password, and clients.
start : Opens a secure socket and starts listenning for new connections.
broadcast : Sends a message to all clients
close_srv : Disconnects all clients and stops the server
close_con : Disconnects a client
rm_client : Based on a argument send, it uses close_con to disconnect a
single client, or all of them
handle : Handles the result from the client
receive : Accepts new connection trying to connect to the server
client_ls : Returns a dictinary of all clients
read_data : Reads the data from the handler
API : Sends commands to the client
'''

def __init__(self, port=9090, cert="./cert.pem", key="key.pem",
paswd="bleh", clients={}):
self.port = port
self.cert = cert
self.key = key
self.paswd = paswd
self.clients = clients


def start(self):
#Needs cert and key generated:
#openssl req -x509 -newkey rsa:4096 -keyout key.pem -out cert.pem -sha256 -
days 365
try:
context = ssl.SSLContext()
context.load_cert_chain(certfile=self.cert, keyfile=self.key,
password=self.paswd)
self.server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
self.server.settimeout(10)
self.server.bind(('127.0.0.1', self.port))
self.server.listen()
```

```
self.server = context.wrap_socket(self.server, server_side=True)
print("[LOG]: Server is starting...")
return self.server

except OSError:
print("[ERR]: SERVER IS BUSY!")
return ''

# Sending Messages To All Connected Clients
def broadcast(self, message):
for client in self.clients:
client = self.clients[client][1]
message = f"{message}"
client.send((message.encode('ascii')))


def close_srv(self, server):
try:
server.shutdown(socket.SHUT_RDWR)
server.close()
print("[LOG]: Server is stopping...")
except AttributeError:
print("[ERR]: Server can't be stopped!")

def close_con(self, index):
self.clients[index][1].shutdown(socket.SHUT_RDWR)
self.clients[index][1].close()
print(f'[LOG]: {self.clients[index][0]} disconnected!')


def rm_client(self, index):
if index == "ALL":
for i in self.clients:
self.close_con(i)
self.clients.clear()

else:
self.close_con(index)
del self.clients[index]


# Handling Messages From Clients
def handle(self, index, client, data):
while True:
try:
client[1].settimeout(100)
data.put(client[1].recv(1024).decode('ascii'))

# Removes client from the list if the client is dead
```

```
if data.queue[0] == "":
data.get()
self.rm_client(index)
break

except:
#self.rm_client(index)
break

# Receiving / Listening Function
def receive(self, server, data):
try:
# Accept Connection
client, address = server.accept()
print(f"[LOG]: Connected with {(str(address))}")

# Request And Store Nickname
client.send('HOST'.encode('ascii'))
hostname = client.recv(1024).decode('ascii')

# Print And Broadcast Hostnames
print("[LOG]: Hostname is {}".format(hostname))
client.send('Connected to server!'.encode('ascii'))

# Start Handling Thread For Client
index = len(self.clients)+1
self.clients |= {index: (hostname, client)}
self.handle(index, self.clients[index], data)

except socket.timeout:
print("[ERR]: Timeout | No connected client")

def client_ls(self):
return self.clients
def read_data(self ,data):
if data.empty() is False:
while True:
print(data.get())
if data.empty() is True:
break
else:
print("[LOG]: No data has been received!")

def API(self, q="", data="", destination=""):
while True:
message=""
if q.empty() is True: pass
#msg = input("# ")
#message = q.put(msg)
```

```
else: message = q.get()

if message == "CONS":
print("-"*10)
for i in self.clients:
print(f"[CON]: {i}: {self.clients[i][0]}")
print("-"*10)

elif message == "KILLSURV":
client = "ALL"
if destination == "":
print("[ERR]: Missing a Server object")
else:
d = destination.get()
self.rm_client(client)
self.close_srv(d)
#break

elif message == "DISSCL":
if destination == "":
print("[ERR]: No client specified!")
else:
d = destination.get()
if d == "ALL":
self.broadcast(message)
self.rm_client(d)
else:
d[1].send(message.encode('ascii'))
self.rm_client(d[0])

elif message == "NEWCL":
if data == "" or destination == "":
print("[ERR]: Missing a Queue object")
else:
d = destination.get()
print("[LOG]: Listening for a new connection...")
listen_thread = threading.Thread(target=self.receive, args=(d, data))
listen_thread.start()
elif message in ["ID", "LS", "PWD", "IP"]:
if destination == "":
print("[ERR]: No client specified!")
else:
d = destination.get()
if d == "ALL":
self.broadcast(message)
else:
d[1].send(message.encode('ascii'))

elif message == "READD":
```

```
if data.empty() is False:
print(data.get())
else:
print("No data!")
elif message == "EXIT":
break
elif message == "":
pass

else:
print("[DEBUG]: Unknown message passed to the API!")




if __name__ == '__main__':
server = Server().start()

data = Queue()
message = Queue()
destination = Queue()
API_thread = threading.Thread(target=Server().API, name="API",
args=(message, data, destination))
API_thread.start()
```

Client-Side

client.py

```
#!/usr/bin/env python3
import socket
import ssl
import threading
import os
import sys
from time import sleep


class Client():
'''Client class managing the connection on the client-side,
as well as running the tasks given by the controller.

Funcs:
__init__ : Initialising the class with default values host, port and cert.
start : Opens a secure socket and tries to connect to the server.
run_command : Once the server is connected the receiver can pass system
commands which will be run from this function.
disconnect : If the controller passes the command for disconnection, this
function will make sure that the socket is closed appropriately.
receive : Once the client has connected with the controller, this function
listens for new commands.
'''

def __init__(self, host = '127.0.0.1', port = 9090, cert = './cert.pem'):
self.host = host
self.port = port
self.cert = cert

def start(self):
'''Attempting to connect to a controller. Going through an infinite loop
with an exception for
ConnectionRefusedError and ConnectionResetError in case the controller is
down.

Return:
client : An ssl class of the connection with the controller.
'''

while True:
try:
context = ssl.SSLContext() #Defaults to TLS
context.verify_mode = ssl.CERT_REQUIRED
context.load_verify_locations(self.cert)
```

```
client = socket.create_connection((self.host, self.port))

#Create secure socket
client=context.wrap_socket(client, server_hostname=self.host)
return client

except (ConnectionRefusedError, ConnectionResetError):
pass


def run_command(self, command):
'''A function that runs past system commands.

Args:
command : A command to run.

Return:
msg : A result message from the run command.
'''

msg = str(os.popen(command).read())
return msg


def disconnect(self, client):
'''A function that disconnects the client from the controller. It stops for
0.5 seconds in order for the server to close the connection first,
Then shutdowns and closes the socket.

Args:
client : A connected socket.
'''

sleep(0.5)
client.shutdown(socket.SHUT_RDWR)
client.close()
print("Disconnecting!")

def receive(self, client):
'''Listens for messages from the controller and do the appropriate actions
for given messages.

Args:
client : A connected socket.
'''
print("Connected to server!")
while True:
try:
# Receive message from controller
```

```
message = client.recv(1024).decode('ascii')

# Sending hostname
if message == 'HOST':
client.send(socket.gethostname().encode('ascii'))

# Sending result from list of commands
elif message in ['ID', 'LS', 'PWD', 'IP']:
command = {'ID': 'id', 'LS': 'ls', 'PWD': 'pwd', 'IP': 'ip a'}
msg = self.run_command(command[message])
client.send(msg.encode('ascii'))

# Disconnecting from controller
elif message == 'DISSCL' or not len(message):
try:
self.disconnect(client)
break
except:
break

except:
self.disconnect(client)
break


def start_client():
'''Starts the client from the client class and passes it to a thread
for the receiving function.
'''

while True:
cl = Client().start()
client_thread = threading.Thread(target=Client().receive, args=(cl,))
client_thread.start()
client_thread.join()
print("-"*10)


if __name__ == '__main__':
start_client()
```