

Team 6

Helmy Akram Helmy	20221459753
George fady	20221454655
Mohamed tarek elsayed	20221440690
Adham ehab	2106124

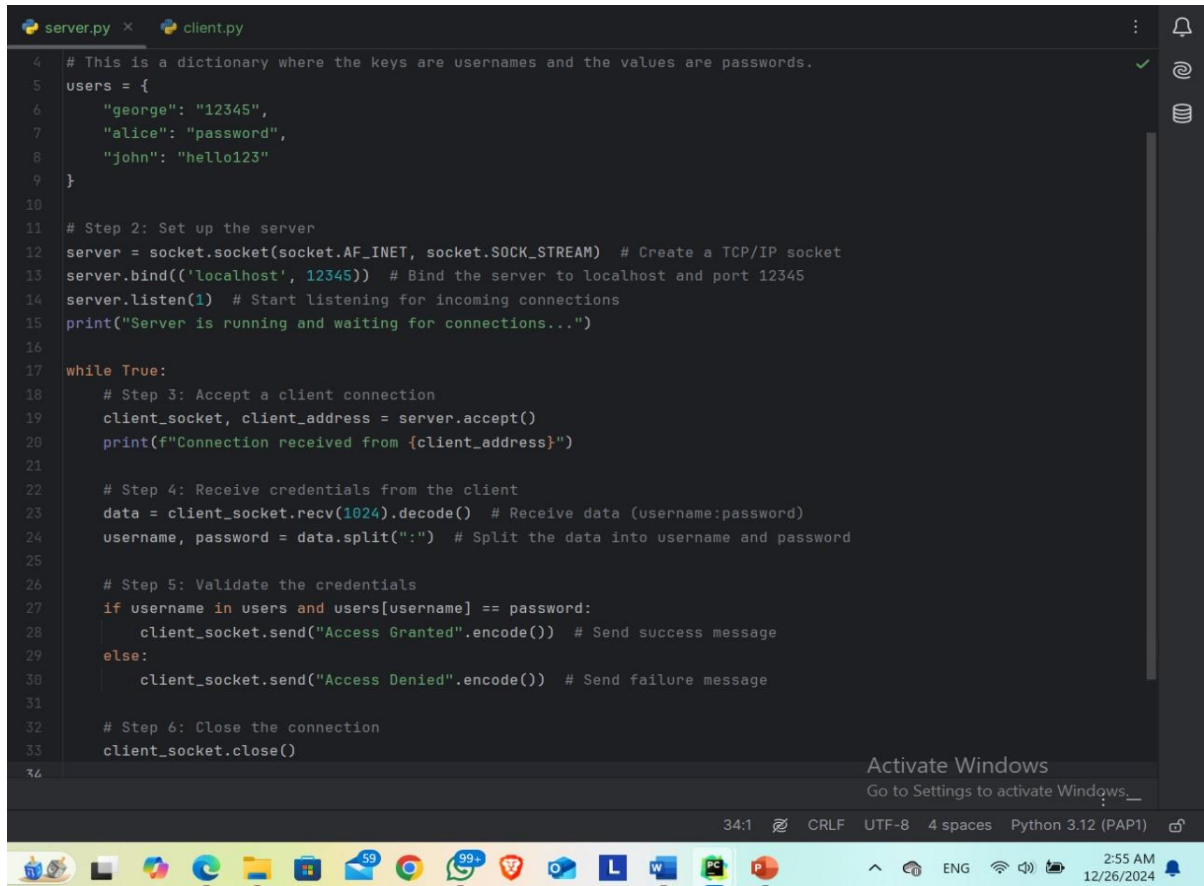
All the resources [here](#)

<https://github.com/gogofady/Remote-Command-Execution-Server>

Remote Command Execution Server With PAP protocol

PAP protocol

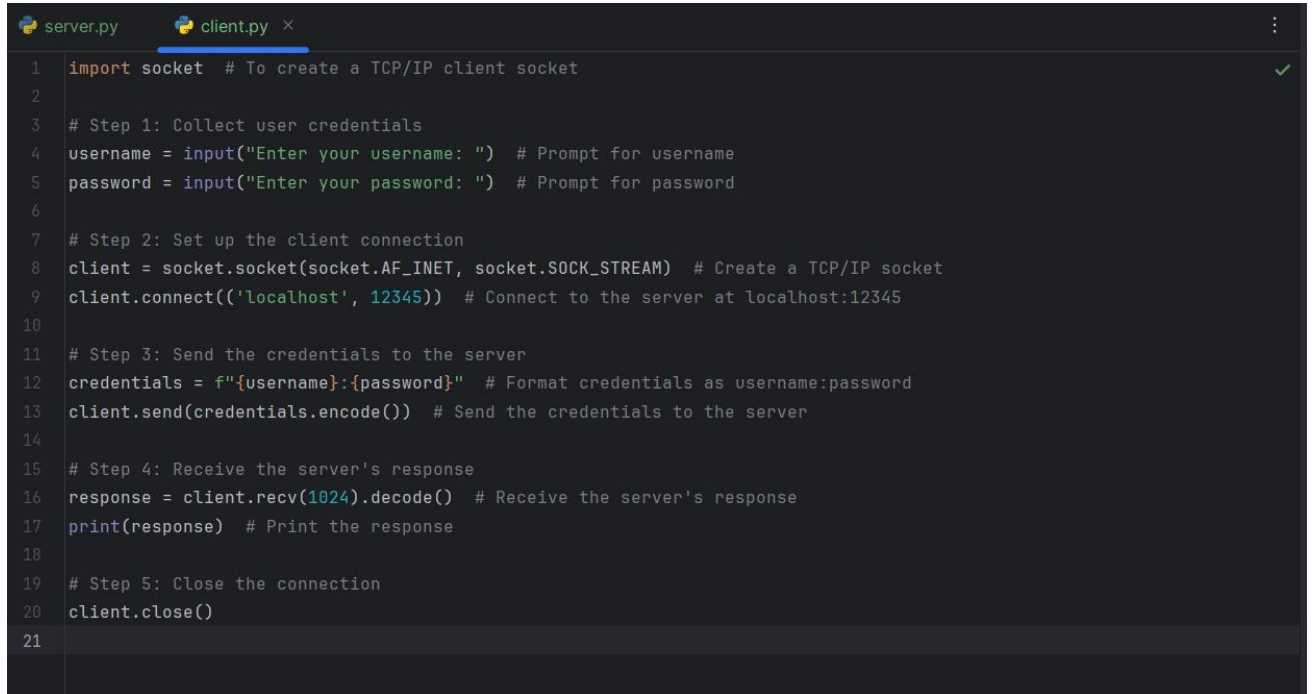
- Initial server (local host with port 12345) using TCP IP with python

A screenshot of a code editor showing a Python script for a PAP server. The script is named 'server.py' and is located in a file explorer. The code defines a dictionary of users and passwords, sets up a TCP/IP socket on localhost port 12345, and implements a loop to accept connections, receive credentials, validate them against the dictionary, and send success or failure messages. The script is written in Python 3.12 and uses the 'PAP1' protocol. The editor interface includes a dark theme, a sidebar with file explorer, and a status bar at the bottom showing the current line (34:1), encoding (CRLF), and other settings. The Windows taskbar is visible at the bottom of the screen.

```
server.py x client.py
4 # This is a dictionary where the keys are usernames and the values are passwords.
5 users = {
6     "george": "12345",
7     "alice": "password",
8     "john": "hello123"
9 }
10
11 # Step 2: Set up the server
12 server = socket.socket(socket.AF_INET, socket.SOCK_STREAM) # Create a TCP/IP socket
13 server.bind(('localhost', 12345)) # Bind the server to localhost and port 12345
14 server.listen(1) # Start listening for incoming connections
15 print("Server is running and waiting for connections...")
16
17 while True:
18     # Step 3: Accept a client connection
19     client_socket, client_address = server.accept()
20     print(f"Connection received from {client_address}")
21
22     # Step 4: Receive credentials from the client
23     data = client_socket.recv(1024).decode() # Receive data (username:password)
24     username, password = data.split(":") # Split the data into username and password
25
26     # Step 5: Validate the credentials
27     if username in users and users[username] == password:
28         client_socket.send("Access Granted".encode()) # Send success message
29     else:
30         client_socket.send("Access Denied".encode()) # Send failure message
31
32     # Step 6: Close the connection
33     client_socket.close()
34
```

With database to store credentials from user (username & password) without encryption or hashing (applying APA)

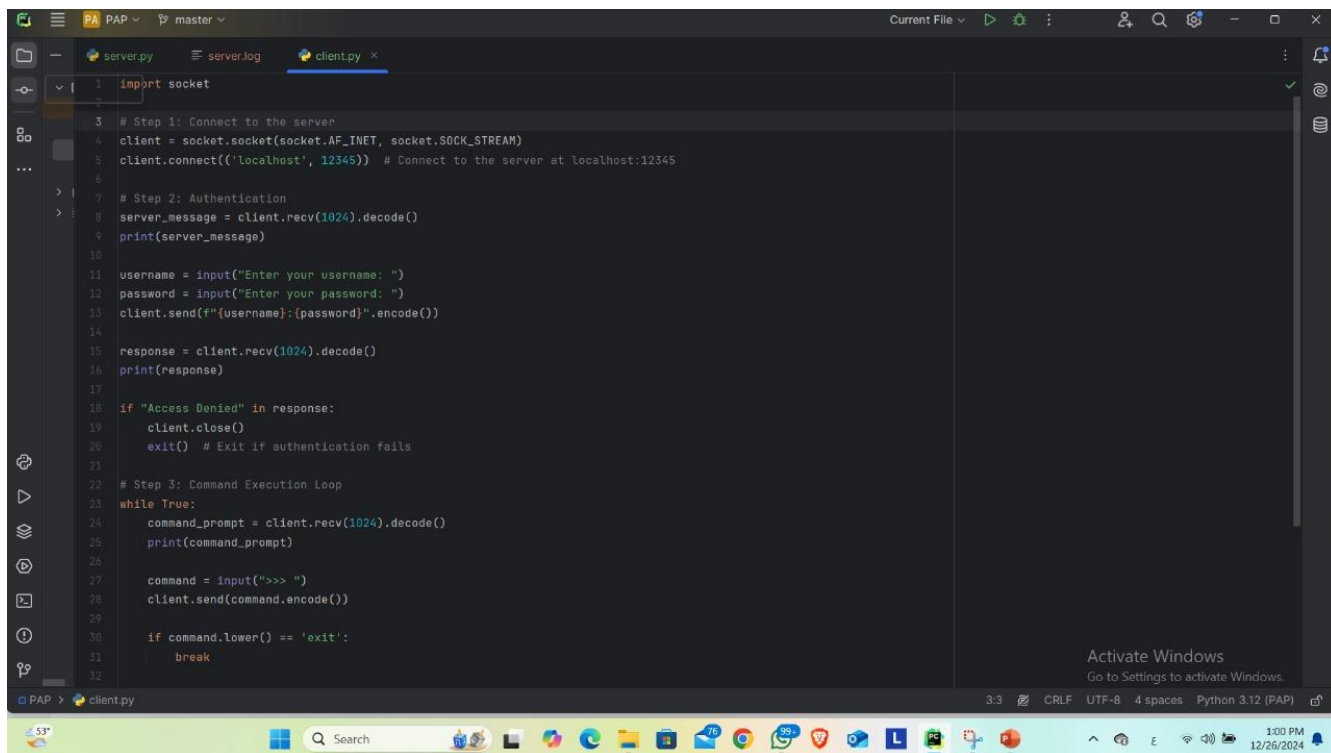
- initial client with python to connect to server (local host with port 12345)



```
1 import socket # To create a TCP/IP client socket
2
3 # Step 1: Collect user credentials
4 username = input("Enter your username: ") # Prompt for username
5 password = input("Enter your password: ") # Prompt for password
6
7 # Step 2: Set up the client connection
8 client = socket.socket(socket.AF_INET, socket.SOCK_STREAM) # Create a TCP/IP socket
9 client.connect(('localhost', 12345)) # Connect to the server at localhost:12345
10
11 # Step 3: Send the credentials to the server
12 credentials = f"{username}:{password}" # Format credentials as username:password
13 client.send(credentials.encode()) # Send the credentials to the server
14
15 # Step 4: Receive the server's response
16 response = client.recv(1024).decode() # Receive the server's response
17 print(response) # Print the response
18
19 # Step 5: Close the connection
20 client.close()
21
```

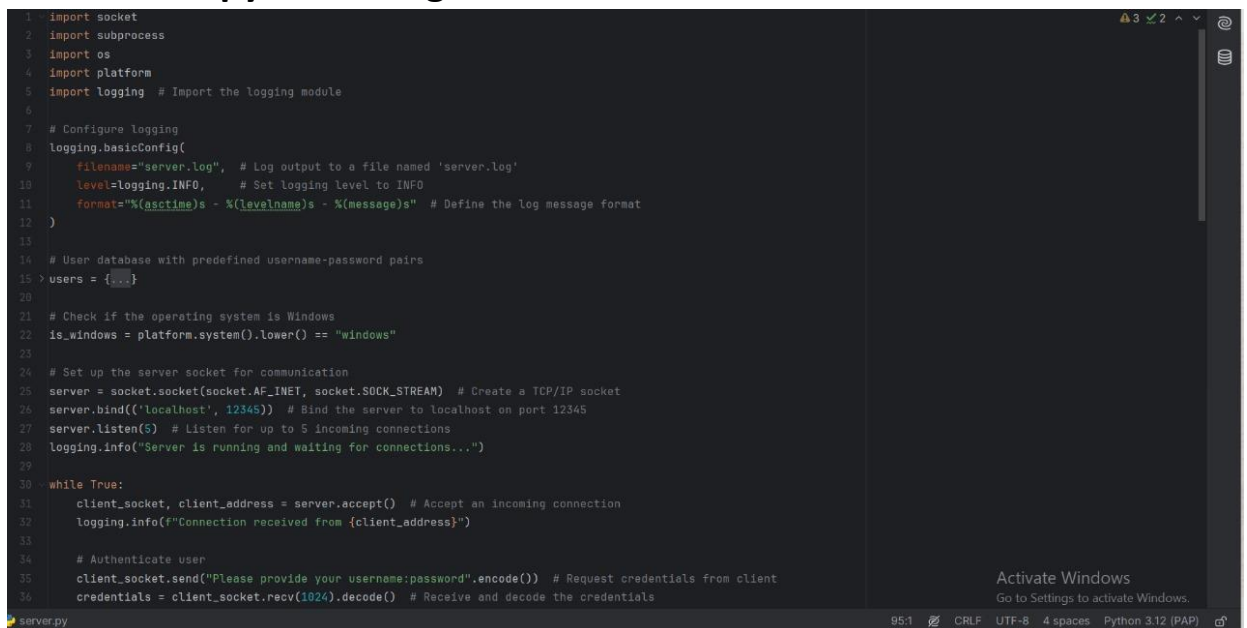
Integrated with C2 service

- Client:



```
1 import socket
2
3 # Step 1: Connect to the server
4 client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
5 client.connect(('localhost', 12345)) # Connect to the server at localhost:12345
6
7 # Step 2: Authentication
8 server_message = client.recv(1024).decode()
9 print(server_message)
10
11 username = input("Enter your username: ")
12 password = input("Enter your password: ")
13 client.send(f"{username}:{password}".encode())
14
15 response = client.recv(1024).decode()
16 print(response)
17
18 if "Access Denied" in response:
19     client.close()
20     exit() # Exit if authentication fails
21
22 # Step 3: Command Execution Loop
23 while True:
24     command_prompt = client.recv(1024).decode()
25     print(command_prompt)
26
27     command = input(">>> ")
28     client.send(command.encode())
29
30     if command.lower() == 'exit':
31         break
32
```

- Server with python integrated with C2 service

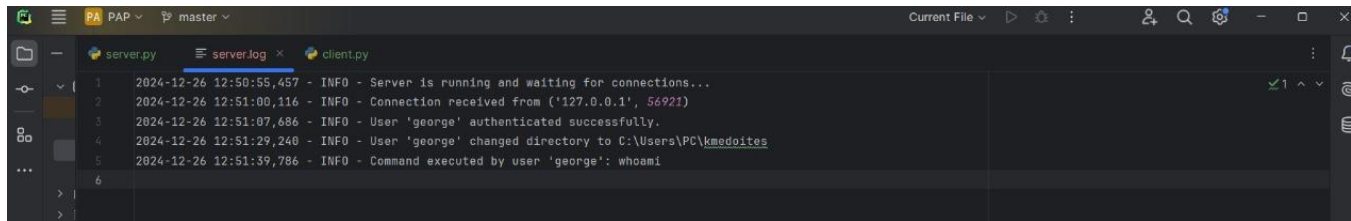


```
1 import socket
2 import subprocess
3 import os
4 import platform
5 import logging # Import the logging module
6
7 # Configure logging
8 logging.basicConfig(
9     filename="server.log", # Log output to a file named 'server.log'
10     level=logging.INFO, # Set logging level to INFO
11     format="%(asctime)s - %(levelname)s - %(message)s" # Define the log message format
12 )
13
14 # User database with predefined username-password pairs
15 users = {}
16
17 # Check if the operating system is Windows
18 is_windows = platform.system().lower() == "windows"
19
20 # Set up the server socket for communication
21 server = socket.socket(socket.AF_INET, socket.SOCK_STREAM) # Create a TCP/IP socket
22 server.bind(('localhost', 12345)) # Bind the server to localhost on port 12345
23 server.listen(5) # Listen for up to 5 incoming connections
24 logging.info("Server is running and waiting for connections...")
25
26 while True:
27     client_socket, client_address = server.accept() # Accept an incoming connection
28     logging.info(f"Connection received from {client_address}")
29
30     # Authenticate user
31     client_socket.send("Please provide your username:password".encode()) # Request credentials from client
32     credentials = client_socket.recv(1024).decode() # Receive and decode the credentials
33
```

```
server.py x server.log client.py
38
39 if username in users and users[username] == password: # Check if credentials are valid
40     logging.info(f"User '{username}' authenticated successfully.")
41     client_socket.send("Access Granted. You can now execute commands.".encode())
42 else:
43     logging.warning(f"Authentication failed for username: {username}")
44     client_socket.send("Access Denied. Disconnecting...".encode()) # Inform client of authentication failure
45     client_socket.close() # Close the connection
46     continue # Return to waiting for the next connection
47
48 current_dir = os.getcwd() # Store the initial working directory
49
50 while True:
51     # Prompt the user for a command
52     client_socket.send(f"Current Directory: {current_dir}\nEnter a command to execute or type 'exit' to disconnect:".encode())
53     command = client_socket.recv(1024).decode() # Receive the command from the client
54
55     if command.lower() == 'exit': # Check if the user wants to exit
56         client_socket.send("Goodbye!".encode()) # Send a farewell message
57         logging.info(f"User '{username}' disconnected.")
58         break # Exit the command loop
59
60     try:
61         if command.startswith('cd '): # Check if the command is to change directories
62             new_dir = command[3:].strip() # Extract the target directory
63             new_path = os.path.join(current_dir, new_dir) # Construct the absolute path
64
65             if os.path.isdir(new_path): # Verify if the directory exists
66                 current_dir = os.path.abspath(new_path) # Update the current directory
67                 client_socket.send(f"Changed directory to {current_dir}".encode())
68                 logging.info(f"User '{username}' changed directory to {current_dir}")
69             else:
70                 logging.info(f"User '{username}' changed directory to {current_dir}")
71                 client_socket.send(f"The directory '{new_dir}' does not exist.".encode()) # Notify about invalid directory
72                 logging.warning(f"Invalid directory change attempt by user '{username}': {new_dir}")
73             else:
74                 if is_windows and command.strip() == 'ls': # Map 'ls' to 'dir' for Windows compatibility
75                     command = 'dir'
76
77                 # Execute the command in a subprocess
78                 result = subprocess.run(command, shell=True, cwd=current_dir, capture_output=True, text=True)
79
80                 if result.stdout: # Send the command's standard output back to the client
81                     client_socket.send(result.stdout.encode())
82                     logging.info(f"Command executed by user '{username}': {command}")
83                 elif result.stderr: # Send any error messages from the command
84                     client_socket.send(result.stderr.encode())
85                     logging.error(f"Error during command execution by user '{username}': {result.stderr}")
86                 else: # Notify if the command executed successfully but had no output
87                     client_socket.send("Command executed successfully with no output.".encode())
88
89     except Exception as e:
90         # Handle any unexpected errors during command execution
91         client_socket.send(f"Error executing command: {str(e)}".encode())
92         logging.error(f"Exception during command execution by user '{username}': {str(e)}")
93
94 client_socket.close() # Close the connection after exiting the loop
95 logging.info(f"Connection closed with {client_address}")
```

Support commands like (whoami ,cd, ls,)

- All logs recording on server.log file



The screenshot shows a code editor with a dark theme. The top bar indicates the project is named 'PAP' and the current file is 'server.log'. The editor displays five lines of log data:

```
1 2024-12-26 12:50:55,457 - INFO - Server is running and waiting for connections...
2 2024-12-26 12:51:00,116 - INFO - Connection received from ('127.0.0.1', 56921)
3 2024-12-26 12:51:07,686 - INFO - User 'george' authenticated successfully.
4 2024-12-26 12:51:29,240 - INFO - User 'george' changed directory to C:\Users\PC\kmedoites
5 2024-12-26 12:51:39,786 - INFO - Command executed by user 'george': whoami
```

The interface includes a sidebar on the left with icons for file explorer, search, and other editor functions. The top right corner has standard window controls and a search icon.