Lesson Guide

Cybersecurity Professional Program

Introduction to Python for Security

Network Communication

Course Path

- PY-01: Introduction to Programming
- PY-02: Data Types & Conditions
- PY-03: Loops
- PY-04: File System & Error Handling
- PY-05: Functions
- **PY-06: Network Communication**
- PY-07: Python for Security

Lesson Objectives

This lesson focuses on how Python is used for network communication, how sockets are created via Python, and how Python facilitates communication with servers.

Lesson Overview

The lesson focuses on sockets, what they are, and how Python is used to create and work with sockets. It also teaches how **netcat** is used to test server-client socket connections.

Learner Level

Learners are not familiar with sockets yet and will be introduced to them in this lesson.

Lesson Notes

During the lesson, when explaining sockets (client, server), let learners practice the commands on their own laptops.

Environment & Tools

- Windows, macOS, Linux
  - Python3
  - PyCharm
  - netcat

Creating a Server Socket

This part of the lesson is an introduction to socket creation, which functions will be used on the server-side, and how the socket can be tested using different tools.

Sockets
Explain what socket modules are used for. Python sockets can be used by attackers to create malicious code that acts as a reverse shell from the client to the attacker (listener). Mention that to work with Python sockets; the ***socket*** library needs to be imported.

Import Socket Library
Demonstrate how to import the socket module and show its contents using the ***dir()*** command.

Creating a Socket: Server-Side
Explain the parameters that the method can accept. Point out that the method's parameters have the default values ***AF_INET*** and ***SOCK_STREAM***.

Setting a Server IP Address and Port
Demonstrate how to use ***bind*** with a specific IP and explain address 0.0.0.0, which listens for all interfaces. Point out that the IP address is a string, and the port is an integer. Note that since a server waits for clients to connect, client IPs are not known beforehand.

Server Connection Limit
Emphasize that the user can specify any number of connections in the ***listen()*** function. Once an IP address and port are set, the next step is to set the number of connections the server can receive. Since it may be difficult for learners to understand the socket creation steps at this point, you can build the socket with comments for now and continue with the lesson.

Accepting Connections
Explain how the returned data from ***accept()*** is saved in variables and used later to send and receive information. In the example, ***conn*** is used to get socket connection details that can be relayed. ***addr*** is used to get the client IP and port. Explain that this is how a simple server-side socket is written in Python.

Connection Check
Explain that we are now running **netcat** on the client-side and will try to connect to our server. Since the server will accept connections from any interface, no special configuration is required. In the next slide, we will use **netcat** to connect to the server and test it. Demonstrate the ***nc –v*** command and explain it. Use ***nc –v [server ip] [port]*** to test the connection and verify that the server is configured properly. If the socket server was not configured to send data, **netcat** will accept the connection and then close it.

Example:
Create a socket in PyCharm that will accept all IP addresses on port 3434. Open Linux and run **netcat** to connect to your socket. Check the connection.

Creating a Client Socket

This part of the lesson is an introduction to socket creation and the functions that are used on the client-side to work with them. It also covers how sockets can be tested.

Establishing a Connection

Demonstrate how to use the ***connect()*** function. Point out that the IP address will be the target of the socket (the IP address to connect to). The port will be the port of the target server to which the user wants to connect.

Connection Error Handling

Explain that it is essential to keep the program's operational flow by handling potential errors. Point out that the potential exception is ***ConnectionRefusedError***.

Client to netcat: netcat Side

Demonstrate how to run ***nc –lvp*** and explain the command. Note that **netcat** needs to be enabled, and the client IP address must be set to the target **netcat** machine. Run ***nc –lvp [port]*** to establish a listener. The client socket will be operational when **netcat** receives the connection.

Example:

Set a socket to a loopback IP on port 4444 via **netcat**. Set a **netcat** listener and check the response when a client socket is established.

PY-06-L1: What Are Sockets?

This TDX Arena lab should be completed in class.

Sending Data

This part of the lesson explains how to send data between two sockets, how to send encoded data, and how to receive the data and decode it. It also discusses how to send large data via sockets.

How Data Is Sent

Demonstrate how to use the ***send()*** function. Explain the ***encode()*** method and the need for data to be in byte form for it to be sent.

Sending Data to netcat

Explain the ***decode()*** function performed by **netcat**. Note that in the example, **netcat** gets the message and closes the connection.

Example:

Create a client socket in PyCharm that sends the following data: "Hello, I'm a client who is sending you data."

Create a **netcat** listener that will receive the data. Check the data.

Receiving Data: Server Side

Demonstrate how to use the ***recv()*** function. Explain its main purpose and buffers. Explain how ***recv()*** is used in a ***while*** loop.

Receiving Data: Client Side

Review the ***decode()*** function again and make sure learners understand it. Review ***recv()*** and **buffers** again and make sure learners understand them. Point out that the data receiving process for the client is the same as that for the server.

Sending Large Data

Explain the condition that checks if a message is longer than the buffer size. If the message is longer, the condition splits it into smaller groups. If it is less than the buffer size, it is sent as is. Using an ***if*** condition, the code can check the package size and notify when the data is transferred. Since the data was encoded, it needs to be decoded on the server-side.

Example:
Build a client-side socket that will send "Hello, I'm a client." Build a server-side socket that will get the data and check the buffer. Have the socket print the message accordingly. If the information exceeds the size of the buffer's limit, notify the user that there is still information on the way and then print it again. When the complete information arrives, print a notification. For POC, create a string that is larger than the buffer size, send it to the server, and check the server's response.

PY-06-L2: Network Protocol Communications
This TDX Arena lab should be completed in class.

Echo Communication

This section will demonstrate how to create echo server communication between the server and the client and how to create a chat.

Echo Server Communication

The server will send a message and wait for a response from the client. The ***if*** condition in the example checks if the client sent an ***exit*** message. If so, it closes the connection and breaks the loop. Note the hierarchy between the client and the server. If the server sends information to the client first, the client should receive it and only then send information back.

Echo Client Communication

Demonstrate how to use loops to send and receive data between the client and server. If there is a connection, the client will wait for a message from the server and will then send a response to the server. The ***if*** condition in the example checks if the client sent ***exit*** or the data received from the server was ***exit***. If so, the client closes the connection and breaks the loop.

Example:

Build a chat between a client and server. Make sure everything is working properly. When one side sends ***exit*** or ***bye***, note that both sides will close the connection.

Use a socket to gather information:

When a client sends ***ifconfig*** or ***ipconfig*** to the server (depending on the OS system), the server will execute the command and send the output back to the client. Hint: You will need to increase the buffer's limit via the OS module.

TDX Arena

Learners will need to navigate to the TDX Arena practice arena as described in the slide deck.

TDX Arena Challenge: Fireflies

TDX Arena challenges can be done at home for extra practice. Make sure learners remember how to log in. Mention that the TDX Arena guide is provided in the course's LMS shell.