LAB MANUAL

ON

CYBER SECURITY ESSENTIALS

(R22A6281)

B.TECH II YEAR – II SEM (R22)

(2023-2024)

DEPARTMENT OF EMERGING TECHNOLOGIES

MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY

(Autonomous Institution – UGC, Govt. of India)

Recognized under 2(f) and 12 (B) of UGC ACT 1956

(Affiliated to JNTUH, Hyderabad, Approved by AICTE - Accredited by NBA & NAAC – 'A' Grade - ISO 9001:2015 Certified)

Maisammaguda, Dhulapally (Post Via. Hakimpet), Secunderabad – 500100, Telangana State, India

IALLA REDDY COLLEGE OF ENGINEERING ANDTECHNOLOGY

II Year B.Tech CSE(CyS) - II Sem (R22) L/T/P/C

0/-/2/1

(R22A6281) - CYBER SECURITY ESSENTIALS LAB

Course objectives:

1. To understand various types of cyber-attacks and cyber-crimes
2. To learn threats and risks within context of the cyber security
3. To have an overview of the cyber laws & concepts of cyber forensics
4. To study the defensive techniques against these attacks
5. To understand various cyber security privacy issues

WEEK - 1

Writing simple Python scripts for tasks like string manipulation, reading from and writing to files, basic network communication.

WEEK - 2

Implementing basic encryption and decryption algorithms in Python Caesar cipher, AES, DES

WEEK - 3

Using python to generate and verify hashes (MD5, SHA-256) for files and messages.

WEEK - 4

Building a simple Python Client-Server application, understanding sockets.

WEEK - 5

Writing a python script to capture and analyze network packets(using libraries like Scapy or PySpark

WEEK - 6

Creating a web scraper in Python to gather data from websites(using BeautifulSoup, Selenium)

WEEK - 7

Simple penetration testing tasks using Python (Eg: port scanning, vulnerability scanning with tools like Nmap in Python.

WEEK - 8

Using python to interact with security-related APIs (eg. VirusTotal, Shodan)

WEEK - 9

Writing python scripts for basic static malware analysis (file signature analysis, string extraction).

WEEK - 10

Developing a simple IDS using Python

Week - 1: writing simple Python scripts that include string manipulation, reading from and writing to files, and basic network communication.

Objective:

- Perform string manipulation
- Read from and write to a file
- Implement a basic example of network communication

Python Code:

```
import socket

# String Manipulation Function
def reverse_string(s):
    return s[::-1]

# File Read and Write Function
def read_and_reverse_write(input_file, output_file):
    with open(input_file, 'r') as file:
        content = file.read()
    reversed_content = reverse_string(content)
    with open(output_file, 'w') as file:
        file.write(reversed_content)

# Basic Network Communication Function
def simple_server(port):
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.bind(('localhost', port))
    server_socket.listen(1)
    print(f"Server listening on port {port}...")
    conn, addr = server_socket.accept()
    print(f"Connected by {addr}")
    while True:
        data = conn.recv(1024)
        if not data:
            break
        conn.sendall(data)
    conn.close()

# Main Execution
if __name___== "__main__":
    # String Manipulation
    original_string = "Hello, World!"
    reversed_str = reverse_string(original_string)
    print(f"Original String: {original_string}")
    print(f"Reversed String: {reversed_str}")
```

```
# File Read and Write
read_and_reverse_write('input.txt', 'output.txt')

# Simple Network Communication (Uncomment to run the server)
# Note: Running the server will require a client to connect and send data. #
simple_server(65432)
```

Instructions for Execution:

1. String Manipulation: This part of the script reverses a given string.
2. File Read and Write:
    a. Create a file named input.txt in the same directory as the script with some content.
    b. The script will read this file, reverse its content, and write it to output.txt.
3. Basic Network Communication:
    a. This is a simple server that echoes received messages.
    b. To test this, you will need to write a separate client script or use a network tool to send data to the server.
    c. Uncomment the simple_server(65432) line to run the server.

Week - 2: the focus is on implementing basic encryption and decryption algorithms in python.

Objective:

- Implement the Caesar Cipher encryption and decryption
- Implement AES and DES encryption and decryption

Python Code:

```
from Crypto.Cipher import AES, DES
from Crypto.Random import get_random_bytes
from Crypto.Util.Padding import pad, unpad
import base64

# Caesar Cipher
def caesar_cipher_encrypt(text, shift):
    result = ""
    for i in range(len(text)):
        char = text[i]
        if char.isupper():
            result += chr((ord(char) + shift - 65) % 26 +
        65) else:
            result += chr((ord(char) + shift - 97) % 26 +
    97) return result

def caesar_cipher_decrypt(text, shift):
    return caesar_cipher_encrypt(text, -shift)

# AES Encryption/Decryption
def aes_encrypt(plain_text, key):
    cipher = AES.new(key, AES.MODE_CBC)
    ct_bytes = cipher.encrypt(pad(plain_text.encode('utf-8'), AES.block_size))
    iv = base64.b64encode(cipher.iv).decode('utf-8')
    ct = base64.b64encode(ct_bytes).decode('utf-8')
    return iv, ct

def aes_decrypt(iv, ct, key):
    iv = base64.b64decode(iv)
    ct = base64.b64decode(ct)
    cipher = AES.new(key, AES.MODE_CBC, iv)
    pt = unpad(cipher.decrypt(ct), AES.block_size)
    return pt.decode('utf-8')

# DES Encryption/Decryption
def des_encrypt(plain_text, key):
    cipher = DES.new(key, DES.MODE_CBC)
    ct_bytes = cipher.encrypt(pad(plain_text.encode('utf-8'), DES.block_size))
```

```
    iv = base64.b64encode(cipher.iv).decode('utf-8')
    ct = base64.b64encode(ct_bytes).decode('utf-8')
    return iv, ct

def des_decrypt(iv, ct, key):
    iv = base64.b64decode(iv)
    ct = base64.b64decode(ct)
    cipher = DES.new(key, DES.MODE_CBC, iv)
    pt = unpad(cipher.decrypt(ct), DES.block_size)
    return pt.decode('utf-8')

# Main Execution
if ___name___== "___main
    ":    #    Caesar    Cipher
    Example shift = 4
    original_text = "HelloWorld"
    encrypted = caesar_cipher_encrypt(original_text, shift)
    decrypted = caesar_cipher_decrypt(encrypted, shift)
    print(f"Caesar Cipher: {original_text} -> {encrypted} -> {decrypted}")

    # AES Example
    aes_key = get_random_bytes(16)  # AES key must be either 16, 24, or 32 bytes long iv,
    encrypted = aes_encrypt(original_text, aes_key)
    decrypted = aes_decrypt(iv, encrypted, aes_key)
    print(f"AES: {original_text} -> {encrypted} ->
    {decrypted}")

    # DES Example
    des_key = get_random_bytes(8)  # DES key must be 8 bytes long iv,
    encrypted = des_encrypt(original_text, des_key)
    decrypted = des_decrypt(iv, encrypted, des_key)
    print(f"DES: {original_text} -> {encrypted} ->
    {decrypted}")
```

Instructions for Execution:

1. Caesar Cipher: Demonstrates basic shift-based encryption and decryption.
2. AES and DES:
    a. These sections use the **pycryptodome** library for AES and DES encryption.
    b. Install the library using **"pip install pycryptodome"** if not already installed.
    c. The script demonstrates encryption and decryption with randomly generated keys.

Week - 3: the focus is on using python to generate and verify hashes for files and messages, utilizing hashing algorithms like MD5 and SHA-256.

Objective:

- Generate MD5 and SHA-256 hashes for strings.
- Verify hashes of strings

Python Code:

```
import hashlib

# Function to generate MD5 hash
def generate_md5_hash(input_string):
    md5_hash = hashlib.md5()
    md5_hash.update(input_string.encode())
    return md5_hash.hexdigest()

# Function to generate SHA-256 hash
def generate_sha256_hash(input_string):
    sha256_hash = hashlib.sha256()
    sha256_hash.update(input_string.encode())
    return sha256_hash.hexdigest()

# Function to verify a hash
def verify_hash(input_string, known_hash):
    # Generate hash for the input string
    generated_hash = generate_md5_hash(input_string) if len(known_hash) == 32 else
generate_sha256_hash(input_string)

    # Compare the generated hash with the known hash return
    generated_hash == known_hash

# Main Execution
if __name___== "__main__":
    # Example strings
    example_string1 = "HelloWorld"
    example_string2 = "HelloWorld!"

    # Generate hashes
    md5_hash_example1 = generate_md5_hash(example_string1)
    sha256_hash_example1 = generate_sha256_hash(example_string1)

    print(f"MD5 Hash of '{example_string1}': {md5_hash_example1}")
    print(f"SHA-256 Hash of '{example_string1}':
    {sha256_hash_example1}")
```

```
    # Verifying hashes
    print(f"Verifying MD5 Hash of '{example_string1}': {verify_hash(example_string1, md5_hash_example1)}")
    print(f"Verifying SHA-256 Hash of '{example_string1}': {verify_hash(example_string1,
sha256_hash_example1)}")

    # Verifying incorrect hash
    print(f"Verifying SHA-256 Hash of '{example_string2}' with hash of '{example_string1}':
{verify_hash(example_string2, sha256_hash_example1)}")
```

Instructions for Execution:

1. Generate MD5 and SHA-256 hashes: The script generates hashes for input strings using MD5 and SHA-256 algorithms.
2. Verify hashes: The script checks if a given string matches a known hash, demonstrating hash verification.

Week - 4: the focus is on building a simple python client-server application to understand the basics of socket programming.

Objective:

- Create a simple TCP Server and Client.
- Understand the basics of socket programming in Python.

Python Codes:

server.py (Server Script)

```
import socket

def start_server(host='localhost', port=65432):
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        s.bind((host, port))
        s.listen()
        print(f"Server listening on {host}:{port}")
        conn, addr = s.accept()
        with conn:
            print(f"Connected by {addr}")
            while True:
                data = conn.recv(1024)
                if not data:
                    break
                conn.sendall(data)

if __name___== "__main__":
    start_server()
```

Client.py (Client Script)

```
import socket

def start_client(host='localhost', port=65432):
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        s.connect((host, port))
        s.sendall(b'Hello, server')
        data = s.recv(1024)
        print(f"Received {data.decode()}")

if __name___== "__main__":
    start_client()
```

Instructions for Execution:

1. Run the Server Script: Start the server first. It will listen for incoming connections on localhost (127.0.0.1) and port 65432.
2. Run the Client Script: Once the server is running, run the client script. The client will connect to the server, send a message, and receive an echo back from the server.
3. Socket Programming: This demonstrates a basic TCP/IP socket connection where the server listens for connections and the client sends a message.

Week - 5: the focus is on writing a python script to capture and analyze network packets.

Objective:

- Capture network packets using **Scapy.**
- Analyze and print the details of captured packets.

Python Code:

```
from scapy.all import sniff

# Packet processing function
def process_packet(packet):
    print(packet.summary())
    # Add more analysis as needed, e.g., checking for specific protocols, ports, etc.

# Start packet sniffing
def start_sniffing():
    print("Starting packet sniffing...")
    sniff(prn=process_packet, count=10)  # Capturing 10 packets for demonstration

if __name__ == "__main__":
    start_sniffing()
```

Instructions for Execution:

1. **Install Scapy:** If not already installed, you can install Scapy using pip install scapy.
2. **Run the Script:** This script will start capturing packets and process 10 packets for demonstration. Each packet's summary information will be printed.
3. **Customize Packet Analysis:** You can extend the process_packet function to perform more detailed analysis, such as filtering specific types of packets, analyzing packet contents, etc.

Week - 6: the focus is on creating a web scraper in python to gather data from websites.

Objective:

- Scrape data from a web page using BeautifulSoap.
- Extract and print specific elements from the web page.

Python Code:

```
import requests
from bs4 import BeautifulSoup

# Function to scrape a web page
def scrape_website(url):
    # Send an HTTP request to the URL
    response = requests.get(url)
    # Parse the HTML content of the page
    soup = BeautifulSoup(response.text, 'html.parser')

    # Example: Extract and print all paragraph texts
    paragraphs = soup.find_all('p')
    for para in paragraphs:
        print(para.get_text())

# Main Execution
if __name__== "__main__":
    url = "http://example.com"  # Replace with the URL of the website you want to scrape
    scrape_website(url)
```

Instructions for Execution:

1. **Install BeautifulSoup and Requests:** If not already installed, install them using **pip install beautifulsoup4** requests.
2. **Run the Script:** The script sends a request to the specified URL, parses the HTML content, and prints the text of all paragraphs. You can modify the script to scrape different elements as needed.
3. **Customize for Different Websites:** Change the URL and the elements you want to scrape according to your requirements.

Week - 7: the focus is on simple penetration testing tasks using python.

Objective:

- Create a simple port scanner using Python.
- Scan a target host to identify open ports.

Python Code:

```
import socket

# Function to scan a single port
def scan_port(ip, port):
    try:
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
            s.settimeout(1)  # Timeout of 1 second
            result = s.connect_ex((ip, port))
            if result == 0:
                return True
            else:
                return False
    except socket.error:
        return False

# Main function to scan ports on a host
def port_scanner(target_ip, port_range):
    print(f"Starting scan on host: {target_ip}")
    for port in range(*port_range):
        if scan_port(target_ip, port):
            print(f"Port {port} is open")

# Main Execution
if __name__== "__main__":
    target_ip = "192.168.1.1"  # Replace with the target IP address
    port_range = (1, 1025)     # Scanning the first 1024 ports
    port_scanner(target_ip, port_range)
```

Instructions for Execution:

1. **Set Target IP and Port Range:** Replace target_ip with the IP address of the target host. Adjust the **port_range** as needed (the current range is 1 to 1024).
2. **Run the Script:** The script will scan the specified port range on the target IP and report open ports.

Week - 8: the focus is on using python to interact with security related APIs.(VirusTotal API).

[VirusTotal API : https://www.virustotal.com/gui/home/upload]

Objective:

- Using python to query the VirusTotal API.
- Analyze a URL or File for Potential Security Threats.

Python Code:

```
import requests

# Function to check a URL using the VirusTotal API
def check_url_virustotal(api_key, url):
    params = {'apikey': api_key, 'resource': url}
    response = requests.post('https://www.virustotal.com/vtapi/v2/url/report', params=params)
    return response.json()

# Main Execution
if __name__== "__main__":
    api_key = "YOUR_VIRUSTOTAL_API_KEY"  # Replace with your VirusTotal API key
    url_to_check = "http://example.com"  # Replace with the URL you want to analyze

    result = check_url_virustotal(api_key, url_to_check)
    if result.get('positives', 0) > 0:
        print(f"URL {url_to_check} detected as potentially malicious.")
        print("Details:", result)
    else:
        print(f"URL {url_to_check} appears to be safe.")
```

Instructions for Execution:

1. **Get a VirusTotal API Key:** Sign up on VirusTotal and obtain your API key.
2. **Set the API Key and URL:** Replace YOUR_VIRUSTOTAL_API_KEY with your own key and http://example.com with the URL you wish to analyze.
3. **Run the Script:** The script will send the URL to VirusTotal and print the analysis results.

Week - 9: the focus is on writing python scripts for basic static malware analysis.

Objective:

- Perform basic static analysis on files to identify potential malware.
- Calculate file hashes, extract strings, and analyze file headers.

Python Code:

Before you begin, you might need to install additional libraries, like `pefile` for PE file analysis and `hashlib` for hashing (included in Python Standard Library).

```
import pefile
import hashlib
import re
import sys

# Function to calculate a file's hash
def calculate_hash(filename):
    hasher = hashlib.sha256()
    with open(filename, 'rb') as file:
        buf = file.read()
        hasher.update(buf)
    return hasher.hexdigest()

# Function to extract printable strings from the file
def extract_strings(filename):
    with open(filename, 'rb') as file:
        content = file.read()
    strings = re.findall(b'[\\x20-\\x7E]{4,}', content)
    return [s.decode('utf-8') for s in strings]

# Function to analyze PE file headers
def analyze_pe_file(filename):
    try:
        pe = pefile.PE(filename)
        return True, pe.dump_info()
    except pefile.PEFormatError:
        return False, "Not a valid PE file."

# Main Execution
if __name__== "__main__":
    filename = sys.argv[1]  # Replace with the file you want to analyze
```

```
print(f"Analyzing file: {filename}")
print("\n[+] Calculating file hash...")
file_hash = calculate_hash(filename)
print(f"SHA-256 Hash: {file_hash}")

print("\n[+] Extracting strings...")
strings = extract_strings(filename)
print(f"Extracted strings: {strings[:5]}...")  # Print first 5 strings for brevity

print("\n[+] Analyzing PE file...")
is_pe, pe_info = analyze_pe_file(filename)
if is_pe:
    print(pe_info)
else:
    print(pe_info)
```

Instructions for Execution:

1. **Install pefile:** Use **pip install pefile** if not already installed.
2. **Run the Script:** Pass the file you want to analyze as a command-line argument. Example: python script.py sample.exe

3. Analyze the Output:

a. The script calculates the SHA-256 hash of the file.
b. It extracts printable strings that could be of interest.
c. For PE (Portable Executable) files, it analyzes and prints file headers

Week - 10: the task is to develop a simple Intrusion Detection System(IDS) using Python.

Objective:

- Create a basic network-based Intrusion Detection System(IDS).
- Monitor and analyze network packets for suspicious patterns.

Python Code:

This script uses the Scapy library for packet capturing and analysis. We'll look for a simple pattern - for example, detecting a large number of HTTP requests to a specific server, which might indicate a DoS attack.

```
from scapy.all import sniff
from collections import Counter
import time

# Configuration
MONITOR_DURATION = 60  # Time in seconds to monitor the traffic
THRESHOLD_REQUESTS = 100  # Threshold for number of requests to trigger an alert
TARGET_IP = "192.168.1.1"  # IP of the target server to monitor

# Global counter for requests
request_counter = Counter()

# Packet processing function
def process_packet(packet):
    if packet.haslayer("IP") and packet.haslayer("TCP"):
        ip_src = packet["IP"].src
        ip_dst = packet["IP"].dst
        tcp_dport = packet["TCP"].dport

        # Example: Detect multiple requests to a specific server (HTTP port 80)
        if ip_dst == TARGET_IP and tcp_dport == 80:
            request_counter[ip_src] += 1

# Intrusion Detection Function
def detect_intrusion():
    print("Starting network monitoring...")
    sniff(prn=process_packet,
    timeout=MONITOR_DURATION)

    # Check if any source IP exceeded the threshold
    for ip, count in request_counter.items():
        if count > THRESHOLD_REQUESTS:
            print(f"Potential intrusion detected from {ip}. Total requests: {count}")

if __name___== "__main__":
```

detect_intrusion()

Instructions for Execution:

1. **Install Scapy:** If not already installed, use **pip install scapy**.
2. **Run the Script:** Execute the script to start monitoring network traffic for the specified duration.
3. **Review Alerts:** The script will report any source IPs that exceed the threshold for requests to the target IP.