Parspec-Assignment

Launch and Secure the EC2 Instance

1. **Navigate to EC2**: Log in to your AWS Console and go to the EC2 Dashboard. Click "Launch instance".
2. **Choose an AMI**: Select the **Ubuntu** Server image (e.g., Ubuntu 24.04 LTS). This will be our operating system.
3. **Instance Type**: Select **t2.micro**.
4. **Key Pair**: Create a new key pair or select an existing one. You **must** have this `.pem` file to access your server.
5. **Network Settings (Security Group)**: This is a critical step for security. Click "Edit" in the Network settings section.
   - Create a new security group.
   - **Inbound Rule 1 (SSH)**:
     - **Type**: `SSH`
     - **Port**: `22`
     - **Source**: Select `My IP`. AWS will automatically detect your current IP address. This ensures only you can remotely manage the server.
   - **Inbound Rule 2 (HTTP)**:
     - Click "Add security group rule".
     - **Type**: `HTTP`
     - **Port**: `80`
     - **Source**: Select `Anywhere` (`0.0.0.0/0`). This allows anyone to access our web application.
6. **Launch**: Review the settings and click "Launch instance".
7. **Connect**: Once the instance is running, connect to it using SSH. Replace `your-key.pem` and `your-instance-ip` with your own values.

```
ssh -i "your-key.pem" ubuntu@your-instance-ip
```

2. Install Apache and ModSecurity

Next, we'll install the Apache web server and the ModSecurity WAF module.

1. **Update System Packages**:

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

2. **Install Apache and ModSecurity**:

   ```
   sudo apt install apache2 libapache2-mod-security2 -y
   ```

3. **Enable Necessary Apache Modules**: We need to enable the proxy module to forward traffic to our Python app and the security module.

   ```
   sudo a2enmod proxy proxy_http security2
   sudo systemctl restart apache2
   ```

3. Install Flask and Gunicorn

Now, let's set up the Python environment and our web application framework.

1. **Install Python and Venv**:

   ```
   sudo apt install python3-pip python3-venv -y
   ```

2. **Create Project Directory**:

   ```
   mkdir ~/sqli-lab
   cd ~/sqli-lab
   ```

3. **Create a Virtual Environment**:

```
python3 -m venv venv
source venv/bin/activate
```

4. **Install Flask and Gunicorn**:

```
pip install Flask gunicorn
```

5. **Create the Flask App**: Create a file named `app.py` with the following code. This app has a vulnerable endpoint that uses unsafe string formatting and a secure one that uses parameterized queries.

```
vim ~/sqli-lab/app.py
```

Paste the following code into the file:

```
import sqlite3
from flask import Flask, request, render_template_string

app = Flask(__name__)
DATABASE = 'users.db'

def init_db():
    """A helper function to create and populate the database."""
    conn = sqlite3.connect(DATABASE)
    cursor = conn.cursor()
    cursor.execute("DROP TABLE IF EXISTS users")
    cursor.execute("""
        CREATE TABLE users (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            username TEXT NOT NULL,
            password TEXT NOT NULL
        )
    """)
    cursor.execute("INSERT INTO users (username, password) VALUES (?,
?)", ('alpha03', 'password@123'))
```

```
    conn.commit()
    conn.close()


HTML_TEMPLATE_VULN = """
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Vulnerable Login</title>
    <style>
        body { font-family: sans-serif; background-color: #f4f4f9; display: gri
d; place-content: center; min-height: 100vh; margin: 0; }
        .container { background: white; padding: 2rem; border-radius: 8px; b
ox-shadow: 0 4px 6px rgba(0,0,0,0.1); text-align: center; }
        h2 { color: #333; }
        input[type="text"] { font-size: 1rem; padding: 8px; width: 250px; mar
gin-bottom: 1rem; border: 1px solid #ccc; border-radius: 4px; }
        input[type="submit"] { font-size: 1rem; padding: 10px 20px; border: n
one; background-color: #007bff; color: white; border-radius: 4px; cursor:
pointer; }
        input[type="submit"]:hover { background-color: #0056b3; }
        .result { margin-top: 1.5rem; padding: 1rem; border: 1px solid #ddd; ba
ckground-color: #e9e9e9; font-family: monospace; text-align: left; }
    </style>
</head>
<body>
    <div class="container">
        <h2>Login Portal</h2>
        <p>Enter your username to proceed.</p>
        <form action="/vulnerable" method="POST">
            <label for="username">Username:</label><br>
            <input type="text" id="username" name="username" autofocus>
            <br>
            <input type="submit" value="Submit">
        </form>
```

```
        <div class="result">
            {{ result | safe }}
        </div>
    </div>
</body>
</html>
"""

HTML_TEMPLATE_SECURE = """
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Secure Login</title>
    <style>
        body { font-family: sans-serif; background-color: #f4f4f9; display: gri
d; place-content: center; min-height: 100vh; margin: 0; }
        .container { background: white; padding: 2rem; border-radius: 8px; b
ox-shadow: 0 4px 6px rgba(0,0,0,0.1); text-align: center; }
        h2 { color: #333; }
        input[type="text"] { font-size: 1rem; padding: 8px; width: 250px; mar
gin-bottom: 1rem; border: 1px solid #ccc; border-radius: 4px; }
        input[type="submit"] { font-size: 1rem; padding: 10px 20px; border: n
one; background-color: #007bff; color: white; border-radius: 4px; cursor:
pointer; }
        input[type="submit"]:hover { background-color: #0056b3; }
        .result { margin-top: 1.5rem; padding: 1rem; border: 1px solid #ddd; ba
ckground-color: #e9e9e9; font-family: monospace; text-align: left; }
    </style>
</head>
<body>
    <div class="container">
        <h2>Login Portal</h2>
        <p>Enter your username to proceed.</p>
        <form action="/secure" method="POST">
```

```
            <label for="username">Username:</label><br>
            <input type="text" id="username" name="username" autofocus>
            <br>
            <input type="submit" value="Submit">
        </form>

        <div class="result">
            {{ result | safe }}
        </div>
    </div>
</body>
</html>
"""


@app.route('/vulnerable', methods=['GET', 'POST'])
def vuln_login():
    """Handles both displaying the form and processing the login."""
    result_message = "Awaiting submission..."

    if request.method == 'POST':
        username = request.form['username']
        query = f"SELECT * FROM users WHERE username = '{username}'"
        conn = sqlite3.connect(DATABASE)
        cursor = conn.cursor()

        try:
            cursor.execute(query)
            user = cursor.fetchone()
        except sqlite3.Error as e:
            user = None
            result_message = f"An SQL error occurred: {e}"

        conn.close()

        result_html = f"<p><strong>Executed Query:</strong><br>{query}
```

```
</p><hr>"
        if user:
            result_html += f"<h1>✅ Access Granted</h1><p>Welcome, {user
[1]}!</p>"
        else:
            if "SQL error" not in result_message:
                result_html += "<h1>❌ Access Denied</h1><p>Invalid userna
me.</p>"
            else:
                result_html += f"<h1>❌ Query Failed</h1><p>{result_messag
e}</p>"

        result_message = result_html

    return render_template_string(HTML_TEMPLATE_VULN, result=result_m
essage)


@app.route('/secure', methods=['GET', 'POST'])
def secure_login():
    """Handles both displaying the form and processing the login."""
    result_message = "Awaiting submission..."

    if request.method == 'POST':
        username = request.form['username']
        query = f"SELECT * FROM users WHERE username = ?"
        conn = sqlite3.connect(DATABASE)
        cursor = conn.cursor()

        try:
            cursor.execute(query, (username,))
            user = cursor.fetchone()
        except sqlite3.Error as e:
            user = None
            result_message = f"An SQL error occurred: {e}"
```

```
        conn.close()

        result_html = f"<p><strong>Executed Query:</strong><br>{query}
</p><hr>"
        if user:
            result_html += f"<h1>✅ Access Granted</h1><p>Welcome, {user
[1]}!</p>"
        else:
            if "SQL error" not in result_message:
                result_html += "<h1>❌ Access Denied</h1><p>Invalid userna
me.</p>"
            else:
                result_html += f"<h1>❌ Query Failed</h1><p>{result_messag
e}</p>"

        result_message = result_html

    return render_template_string(HTML_TEMPLATE_SECURE, result=result
_message)


if __name__ == '__main__':
    # print("Initializing the database...")
    init_db()
    # print("Database 'users.db' created with sample data.")
    # print("Starting Flask server at http://127.0.0.1:5000")
    app.run()
```

6. **Initialize the Database**: Run the script once to create the `users.db` file.

```
python3 app.py
```

Press `Ctrl+C` to exit the Flask server.

4. Configure Gunicorn and Apache Reverse Proxy

We will run the Flask app with Gunicorn and use Apache as a reverse proxy to handle incoming traffic and apply ModSecurity rules.

1. **Create a Gunicorn Service**:

   ```
   sudo vim /etc/systemd/system/gunicorn.service
   ```

   Paste the following configuration:

   ```
   [Unit]
   Description=Gunicorn instance for SQLi Lab
   After=network.target

   [Service]
   User=ubuntu
   Group=www-data
   WorkingDirectory=/home/ubuntu/sqli-lab
   Environment="PATH=/home/ubuntu/sqli-lab/venv/bin"
   ExecStart=/home/ubuntu/sqli-lab/venv/bin/gunicorn --workers 3 --bind 12
   7.0.0.1:8000 wsgi:app

   [Install]
   WantedBy=multi-user.target
   ```

2. **Create a `wsgi.py` file**:

   ```
   vim ~/sqli-lab/wsgi.py
   ```

   Paste this line:

   ```
   from app import app

   if __name__ == "__main__":
       app.run()
   ```

3. **Start the Gunicorn Service**:

```
sudo systemctl daemon-reload
sudo systemctl start gunicorn
sudo systemctl enable gunicorn
```

4. **Configure Apache**:

```
sudo vim /etc/apache2/sites-available/000-default.conf
```

Replace the entire file content with this:

```
<VirtualHost *:80>
    ProxyPreserveHost On
    ProxyPass / http://127.0.0.1:5000/
    ProxyPassReverse / http://127.0.0.1:5000/

    <IfModule security2_module>
        SecRuleEngine On
    </IfModule>

    <Location /vulnerable>
        <IfModule security2_module>
            SecRuleEngine Off
        </IfModule>
    </Location>
</VirtualHost>
```

5. Finalize ModSecurity Configuration

1. **Enable Core Rule Set (CRS)**:

```
sudo mv /etc/modsecurity/modsecurity.conf-recommended /etc/modsecu
rity/modsecurity.conf
```

2. **Set to Blocking Mode**:

```
sudo vim /etc/modsecurity/modsecurity.conf
```

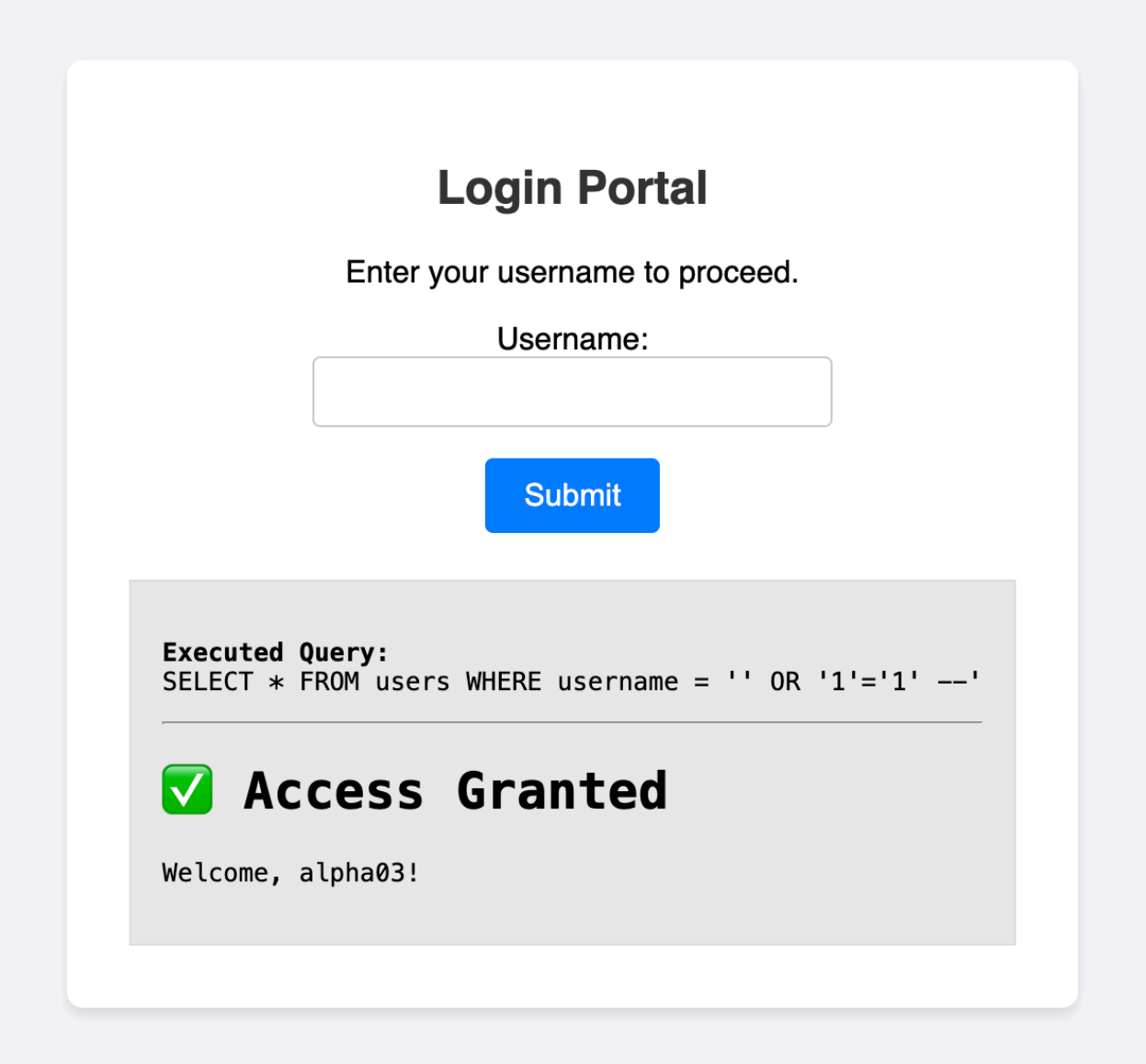
Find the line `SecRuleEngine DetectionOnly` and change it to `SecRuleEngine On`.

3. **Restart Apache**:Bash

```
sudo systemctl restart apache2
```

6. Testing the SQL Injection

Test 1: The Vulnerable Endpoint (ModSecurity OFF)



✅ **Expected Result:** An "Access Granted" message.

Test 2: The Secure Endpoint (ModSecurity ON)

❌ **Expected Result:** A **403 Forbidden** error from Apache.