Experiment No. 7

Aim: To develop secure session management using secure cookies and token-based authentication.

Tools:

- Python 3
- Flask (pip install Flask)
- Flask-JWT-Extended (pip install Flask-JWT-Extended)
- Web Browser (with Developer Tools)
- API Client (like Postman or curl)

Learning Objective: To understand and implement secure session management techniques using both HttpOnly/Secure cookies and JSON Web Tokens (JWT) to protect against session-based attacks.

Theory:

Session management is the process of maintaining a user's state across multiple requests in the stateless HTTP protocol. When a user logs in, the server creates a session and issues a unique identifier (session ID or token) to the client. This identifier is sent back with every subsequent request to authenticate the user.

Insecure session management can lead to severe vulnerabilities like **Session Hijacking**, where an attacker steals a valid session identifier to impersonate a user. The two primary approaches to secure session management are:

Secure Cookies: This is the traditional stateful approach. The server stores session data and sends a session ID to the client in a cookie. Security is enforced by setting specific cookie attributes:

- HttpOnly: Prevents client-side scripts (JavaScript) from accessing the cookie, mitigating XSS-based session theft.
- Secure: Ensures the cookie is only transmitted over an encrypted HTTPS connection.

- SameSite: Restricts the cookie from being sent with cross-site requests, mitigating Cross-Site Request Forgery (CSRF).

Token-Based Authentication (JWT): This is a modern, stateless approach. Upon login, the server generates a signed JSON Web Token (JWT) that contains user information (claims) and an expiration date. This token is sent to the client, which stores it and sends it back in the Authorization header of every request to a protected resource. The server verifies the token's signature to authenticate the request without needing to store session state.

Procedure / Implementation:

This experiment is divided into two parts: first, implementing secure cookie-based sessions, and second, using JWT for stateless authentication.

Part 1: Implementing Secure Session Management with Cookies

1. Install Flask: **pip install flask**
2. Create a Flask application named **secure_cookie_app.py**. This app will configure session cookies with security attributes.

```
from flask import Flask, session, request, redirect, url_for
from datetime import timedelta

app = Flask(__name__)
# This key must be kept secret in a real application
app.secret_key = "a_very_long_and_random_secret_key"

# Configure secure cookie attributes
app.config.update(
    SESSION_COOKIE_HTTPONLY=True,   # Protect against client-side script access
    SESSION_COOKIE_SECURE=True,     # Only send cookie over HTTPS
    SESSION_COOKIE_SAMESITE='Lax',  # Mitigates CSRF. 'Strict' is also an option.
      PERMANENT_SESSION_LIFETIME=timedelta(minutes=30) # Session expires after 30 minutes
)

@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        # In a real app, you would validate credentials against a database
        if request.form.get('username') == 'admin':
            session['user'] = 'admin'
            session.permanent = True # Use the configured lifetime
            return redirect(url_for('dashboard'))
        return '<form method="post">Username: <input name="username"><button>Login</button></form>'

@app.route('/dashboard')
def dashboard():
    if 'user' in session:
        return f"<h1>Welcome to the Dashboard, {session['user']}!</h1>"
    return redirect(url_for('login'))

if __name__ == '__main__':
    # The ssl_context='adhoc' is for local testing to enable HTTPS
    # In production, use a proper SSL/TLS certificate
    app.run(debug=True, ssl_context='adhoc')
```

3. **Run the application** and test it.

- Run the script: python secure_cookie_app.py
- Open your browser and navigate to https://127.0.0.1:5000/login.
- Log in and use your browser's Developer Tools (Application -> Cookies) to inspect the session cookie. You will see that the Secure and HttpOnly flags are enabled.

Part 2: Implementing Secure Session Management with JWT

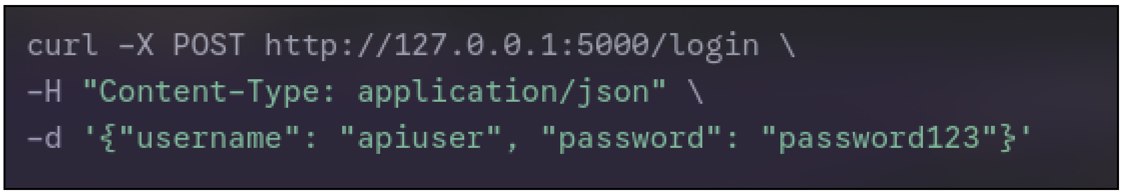
1. Install required libraries: **pip install Flask Flask-JWT-Extended**
2. **Create a Flask application** named jwt_app.py. This app will function as an API that uses JWT for authentication.

```
from flask import Flask, jsonify, request
from flask_jwt_extended import create_access_token, jwt_required, get_jwt_identity,
JWTManager
from datetime import timedelta
app = Flask(__name__)
# This key must be kept secret in a real application
app.config["JWT_SECRET_KEY"] = "another_super_secret_jwt_key"
app.config["JWT_ACCESS_TOKEN_EXPIRES"] = timedelta(minutes=15)
jwt = JWTManager(app)
@app.route('/login', methods=['POST'])
def login():
    # This endpoint expects JSON data
    username = request.json.get("username", None)
    password = request.json.get("password", None)
    # In a real app, validate credentials against a database
    if username != 'apiuser' or password != 'password123':
        return jsonify({"msg": "Bad username or password"}), 401
    # Create the access token
    access_token = create_access_token(identity=username)
    return jsonify(access_token=access_token)

@app.route('/profile')
@jwt_required() # This decorator protects the endpoint
def profile():
    current_user = get_jwt_identity()
    return jsonify(logged_in_as=current_user), 200

if __name__ == '__main__':
    app.run(debug=True)
```

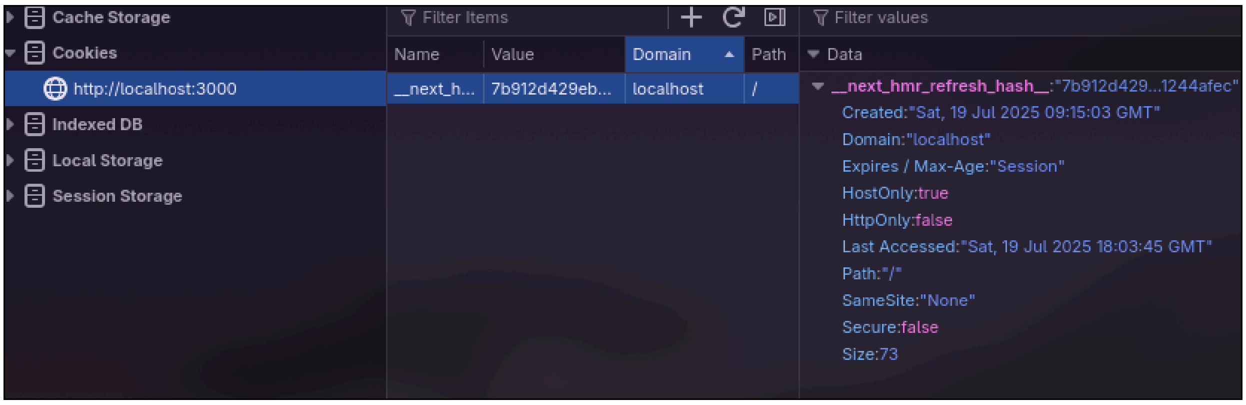
3. **Test the API** using a tool like curl or Postman.

Run the script: python jwt_app.py

Login to get a token:



```
curl -X POST http://127.0.0.1:5000/login \
-H "Content-Type: application/json" \
-d '{"username": "apiuser", "password": "password123"}'
```



```
Cache Storage
Cookies
    http://localhost:3000
Indexed DB
Local Storage
Session Storage

Name        Value           Domain      Path
__next_h... 7b912d429eb...  localhost   /

Data
__next_hmr_refresh_hash__:"7b912d429...1244afec"
    Created:"Sat, 19 Jul 2025 09:15:03 GMT"
    Domain:"localhost"
    Expires / Max-Age:"Session"
    HostOnly:true
    HttpOnly:false
    Last Accessed:"Sat, 19 Jul 2025 18:03:45 GMT"
    Path:"/"
    SameSite:"None"
    Secure:false
    Size:73
```

Learning Outcome: Performed an experiment to configure and deploy secure, stateful session management using secure cookie attributes (HttpOnly, Secure, SameSite) in Flask. Additionally, implemented a stateless session mechanism using JSON Web Tokens (JWT), demonstrating how to protect API endpoints and manage authentication without server-side session storage.

Conclusion: This experiment demonstrated two robust methods for securing user sessions. Configuring cookies with HttpOnly, Secure, and SameSite attributes is a critical defense for traditional web applications to prevent session theft. For modern APIs and single-page applications, stateless JWTs provide a flexible and secure alternative. Implementing these techniques correctly is essential for preventing common session-based attacks like session hijacking and CSRF, thereby ensuring the integrity and confidentiality of user data.

Name:
Class: BE-CSE
Roll No.:

For Faculty Use

Correction Parameters	Formative Assessment [40%]	Timely completion of Practical [40%]	Attendance / Learning Attitude [20%]	
Marks Obtained				