Unit 11

Web Development in Python

(Practical)

In this exercise, we we introduced to the use of Flask as a simple web server. Together with Flask, exposure was made to Jing templates.

Security Considerations for Web Apps

Browsershave implemented many new security features ove the years to aid securing web applications. For example, `HttpOnly` that ensures cookies are inaccessible from JavaScript, `X-Frame-Options` to prevent pages being hosted in `IFRAME` and the introduction of JSON functionality, such as `JSON.parse()` that safely parses JSON content without executing JavaScript code.

- ***Cookies***. This is a small piece of text consisting of key-value pairs, that is persisted on the user's computer by a browser and were used as the main method of authenticating users and storing the credentials for use at some later point. (Tang et al., 2011). Since cookies often store sensitive information, these are often targets for attackers using Cross Site Scripting (XSS) and if successful, allow an attacker to impersonate the user.
- ***Cross-site scripting (XSS).*** XSS refers to client-side code injection that usually leverages client side scripting engines like JavaScript, ActiveX or Flash. Attackers either inject a payload that is then persisted on the target machine, or they send the payload script as part of the request sent to a webserver and "reflected" back in the HTTP response.
- ***Transport Layer Security (TLS)/Secure Socket Layer (SSL).*** These transport protocols are designed to facilitate secure transmission of data between client and server using authentication and encryption of the data. The biggest weakest here is the use of weak ciphers. Thankfully,
- ***SQL injection.*** SQL injection is still encountered in modern software due to poor development practices. Attackers are able to inject SQL commands into form fields which are then sent to the webserver and executed. This effectively allows an attacker to execute SQL statements directly, unless the application makes extra checks to

validate and “escape” the form inputs, such that a SQL syntax error occurs with the hacked field values.

- ***Cross Site Request Forgery (CSRF).*** CSRF is a roged request that originates from another site (“cross site”). It occurs frequently where an attacker causes a user’s browser to perform an unwanted action, together with any session cookies, while the victim is authenticated.
- ***Unrestricted URL access.*** Insufficient access checks are performed in the view templates to ensure the logged on user indeed has permissible access to buttons, links or other content.

The Open Web Application Security Project (OWASP) is an community of engineers and IT security professionals. They developed a list of the top 10 web application vulnerabilities used by companies to change their development culture and foster secure code.

Issue	**Description**
SQL Injection	Untrusted data is sent and executed without proper authorisation.
Broken Authentication	Authentication and session management are not handled correctly.
Sensitive Data Exposure	Sensitive data is not properly protected.
XML External Entities	Poorly configured XML processors evaluate external entity references within XML documents.
Broken Access Control	Resource restrictions are not propery enforced.
Security Misconfiguration	Insecure setup, HTTP headers or error messages with sensitive information.
Cross-Site Scripting (XSS)	Untrusted data is included in web pages without proper validation or escaping.
Insecure Deserialization	Deserialization tools can be used to perform remote code execution.
Using components with known vulnerabilities.	Usually, components and frameworks execute with the same privileges as the application.
Insufficient Logging/Monitoring	Logging and monitoring is required to trap attack early on. The average length of time to detect a breach is 200 days.

Security Recommendations

Since it is not possible to eradicate all forms of web app attacks, the following are a few recommendations. Srivani et al., (2017) recommend

1. CAPTCHA integration;
2. Brute force protection that limits retries;
3. file upload scans;
4. redirection away from HTTP to HTTPS.

Holik & Neradova (2017) recommend

1. Disable database ports for remote access;
2. Disable password autocompletion in input fields;
3. Utilise `X-Frame-Options` in reply headers;
4. Utilise the `HttpOnly` flag and enable secure cookies;
5. Parameterise and validate all SQL queries;
6. Implement a hidden authorisation token to aid against CSRF;
7. Use HTTPS connections.

In common, comparing against OWASP, the techniques that help mitigate insecure web apps concern SQL queries, HTTPS connections and validation of all input fields.

Web Server

```
from flask import Flask, render_template
from flask_mysqldb import MySQL

app = Flask(__name__)
app.config['MYSQL_HOST'] = 'localhost'
app.config['MYSQL_USER'] = 'root'
app.config['MYSQL_PASSWORD'] = 'codio'
app.config['MYSQL_DB'] = 'students'

mysql = MySQL(app)

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

def index():
```

```
    query = "SELECT score from grades;"

    cursor = mysql.connection.cursor()
    cursor.execute(query)
    results = cursor.fetchall()
    cursor.close()

    return render_template('index.html', title='Home', scores=results)

if __name__ == '__main__':
    app.run(host='0.0.0.0')
```

Data for Web Server

```
-- Table structure for table `grades`
DROP TABLE IF EXISTS `grades`;

CREATE TABLE `grades` (
  `surname` varchar(30) DEFAULT NULL,
  `forename` varchar(30) DEFAULT NULL,
  `module_code` varchar(8) DEFAULT NULL,
  `score` int(11) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;


-- Dumping data for table `grades`
LOCK TABLES `grades` WRITE;

INSERT INTO `grades` VALUES
('Smith','John','MATH101',85),
('Jones','Dave','COMP101',93),
('Bob','Alice','MATH101',50),
('Alice','Bob','MATH101',66);

UNLOCK TABLES;

DROP TABLE IF EXISTS `modules`;
CREATE TABLE modules (
        code varchar(10) not null,
        name varchar(20) not null);
INSERT INTO modules VALUE
('MOD1', 'Main Module'),
('MOD2', "Second Module"),
('MOD3', 'Third Module');

DROP TABLE IF EXISTS students;
CREATE TABLE students(
        id int(4) not null auto_increment,
        forename varchar(20) not null,
        surname varchar(20) not null,
        primary key(id));

INSERT INTO students VALUES
('Mary', 'Bary'),
("Sam", "Jam"),
('Flipp', 'Pilf');

DROP TABLE IF EXISTS student_grades;

CREATE TABLE student_grades(
        student_id int(4) not null,
)
```

Basic Template for Web Server

```
<html>
  {% block content %}
    <h1>Scores</h1>
    <ul>
      {% for score in scores %}
      <li>
        {{score[0]}}
      </li>
      {% endfor %}
    </ul>
  {% endblock %}
</html>
```

References

Holík, F. & Neradova, S. (2017) Vulnerabilities of modern web applications. *40th International Convention on Information and Communication Technology, Electronics and Microelectronics* 1256-1261. IEEE.

OWASP (2021) Who is the OWASP Foundation? Available from https://owasp.org/ [Accessed 23 Jul 2021]

Srivani, P., Ramachandram, S. & Sridevi, R. (2017). A survey on client side and server side approaches to secure web applications. 2017 International conference of Electronics, Communication and Aerospace Technology 1:22-27. IEEE.

Tang, S., Dautenhahn, N. & King, S.T. (2011) Fortifying web-based applications automatically. *Proceedings of the 18th ACM conference on Computer and communications security*:615-626.