Internship Report – Week 5

Title: Ethical Hacking & Exploiting Vulnerabilities
Intern Name: Rayyan Chaaran
Submission Date: July 24, 2025

🎯 Objective of Week 5:

The purpose of Week 5 was to gain hands-on experience with ethical hacking techniques and simulate common web vulnerabilities like SQL Injection and CSRF. This helped us learn how attackers exploit weak code — and more importantly — how to fix those weaknesses to secure real applications.

Task 1: Ethical Hacking Basics

Ethical hacking is the legal and authorized process of identifying and fixing system vulnerabilities. It mirrors the strategies of real attackers but is used to improve security.

Penetration Testing Phases:

- **Reconnaissance** – Identify IPs, domains, open ports
- **Scanning** – Use tools like Nmap to find vulnerabilities
- **Enumeration** – Collect service version info, login endpoints
- **Exploitation** – Simulate attack to confirm vulnerabilities
- **Reporting** – Document findings and fixes

Tools Used: Kali Linux, Nmap, Burp Suite, OWASP ZAP

🛠 Practical Work:

- Launched test app on local server
- Performed Nmap port scan
- Intercepted login request using Burp Suite
- Mapped login endpoints for testing

```
nmap -sV -T4 testphp.vulnweb.com
```

```
PORT     STATE SERVICE VERSION
80/tcp   open  http    Apache httpd 2.4.7
443/tcp  open  ssl/http Apache httpd 2.4.7
```

*SQL.txt - Notepad

File Edit Format View Help

```
sqlmap -u "http://testphp.vulnweb.com/listproducts.php?cat=1" --dbs
```

Task 2: SQL Injection & Exploitation

SQL Injection (SQLi) is a technique where attackers manipulate SQL queries using unsanitized inputs. This can give unauthorized access to sensitive data.

Example:

```sql
SELECT * FROM users WHERE username = '$user';
```

If input = `' OR '1'='1`, the condition becomes true and login bypasses.

🛡️ **Prevention:**

- Prepared Statements (parameterized queries)
- Input validation
- Least privilege DB accounts

Tool Used: SQLMap

🛠 Practical Work:

- Tested login and search fields
- Detected vulnerability with SQLMap
- Fixed code using parameterized queries
- Re-ran SQLMap to confirm fix

```
Database: acuart
[1 table found]
+---------+
| users   |
+---------+
```

```
nmap -sV -T4 testphp.vulnweb.com
```

```javascript
const mysql = require('mysql2');
const express = require('express');
const app = express();

const db = mysql.createConnection({
  host: 'localhost',
  user: 'root',
  password: '',
  database: 'test'
});

app.get('/user', (req, res) => {
  const id = req.query.id;
  db.execute('SELECT * FROM users WHERE id = ?', [id], (err, result) => {
    if (err) return res.send('Error');
    res.json(result);
  });
});

app.listen(3001, () => console.log('Secure SQL API running...'));
```

Task 3: CSRF Protection

CSRF (Cross-Site Request Forgery) tricks users into submitting unauthorized actions. It uses the user's browser and cookies to perform malicious actions.

Protection Methods:

- CSRF tokens with each form
- Verify tokens on server
- Use `csurf` middleware in Node.js

Practical Work:

- Used Burp Suite to create CSRF attack

- Added `csurf` middleware in Express
- Tested CSRF token in form submission
- Blocked unauthorized CSRF requests

```
const csrfProtection = csrf({ cookie: true });
app.use(express.urlencoded({ extended: true }));
```

```
const express = require('express');
const mysql = require('mysql2');
const app = express();

// MySQL connection
const db = mysql.createConnection({
  host: 'localhost',
  user: 'root',
  password: '',   // use your own password if needed
  database: 'test'
});

// Connect to DB
db.connect(err => {
  if (err) throw err;
  console.log('Connected to database');
});

// SQL Injection Protected Route
app.get('/user', (req, res) => {
  const id = req.query.id;
  db.execute('SELECT * FROM users WHERE id = ?', [id], (err, result) => {
    if (err) return res.send('Database Error');
    res.json(result);
  });
});

app.listen(3001, () => {
  console.log('Server running on http://localhost:3001');
});
```

```javascript
const express = require('express');
const cookieParser = require('cookie-parser');
const csrf = require('csurf');

const app = express();
app.use(cookieParser());

// Setup CSRF middleware using cookies
const csrfProtection = csrf({ cookie: true });
app.use(express.urlencoded({ extended: true }));

// Route to serve a sample form with CSRF token
app.get('/form', csrfProtection, (req, res) => {
  res.send(`
    <form action="/process" method="POST">
      <input type="hidden" name="_csrf" value="${req.csrfToken()}">
      <button type="submit">Submit</button>
    </form>
  `);
});

// Route that handles form submission
app.post('/process', csrfProtection, (req, res) => {
  res.send('✅ CSRF token verified. Safe request.');
});

app.listen(3002, () => {
  console.log('CSRF-Protected server running on http://localhost:3002');
});
```

```
C:\Users\hp\Documents\Feedback-app>cd ethical hacking week5

C:\Users\hp\Documents\Feedback-app\Ethical Hacking Week5>npm install express cookie-parse
npm warn deprecated csurf@1.11.0: This package is archived and no longer maintained. For
om/expressjs/express/discussions

added 14 packages, and audited 114 packages in 2s

15 packages are looking for funding
  run `npm fund` for details

2 low severity vulnerabilities

To address all issues (including breaking changes), run:
  npm audit fix --force

Run `npm audit` for details.
```


```
C:\Users\hp\Documents\Feedback-app>cd ethical hacking week5

C:\Users\hp\Documents\Feedback-app\Ethical Hacking Week5>npm install express cookie-parse
npm warn deprecated csurf@1.11.0: This package is archived and no longer maintained. For
om/expressjs/express/discussions

added 14 packages, and audited 114 packages in 2s

15 packages are looking for funding
  run `npm fund` for details

2 low severity vulnerabilities

To address all issues (including breaking changes), run:
  npm audit fix --force

Run `npm audit` for details.
```


```
const csrfProtection = csrf({ cookie: true });
app.use(express.urlencoded({ extended: true }));
```

Summary:

This week gave me real-world insight into how attackers exploit apps and how to prevent such attacks. I learned:

- How to perform ethical scans
- How SQL Injection works and how to fix it
- How to implement CSRF protection using tokens

Github repository:

https://github.com/SanataChaaran786