Vulnerability Report

1. Unauthorized Access to Staff Accounts via Profile Edit

Discovery

- **Vulnerability Type:** Authentication Bypass / Business logic
- **Affected Endpoint:** `http://localhost:8888/edit-profile.php`
- **How Found:** Modifying the `username` field to an existing staff account.

Procedure

1. Navigated to `edit-profile.php` as a normal user.
2. Modified the `username` field to an existing staff username.
3. Entered my own password in the "old password" field and submitted the request.
4. Gained access to the staff account without knowing the actual password.

Impact

- Any user can **take over any other account**, including administrators.
- No validation is performed to check if the user actually owns the account they are modifying.
- **Complete account takeover** is possible by exploiting this vulnerability.

Mitigation

- **Require Verification of Old Passwords**: The actual old password must be validated before changing credentials.
- **Restrict Username Changes**: Users should not be able to modify usernames freely.
- **Implement Role-Based Access Controls (RBAC)**: Ensure only authorized users can edit sensitive information.

---

2. User Deletion via Unauthenticated Endpoint

Discovery

- **Vulnerability Type:** Insecure Direct Object Reference (IDOR)

- **Affected Endpoint:** `GET /delete-user.php?id=<id_to_delete>&continue=http://localhost:8888/list-profiles.php?who=student`
- **How Found:** Using Burp Suite to inspect HTTP deletion requests after logging into staff account using the 1st vulnerability

Procedure

1. Logged in as a staff account using the first vulnerability
2. Deleted the account and stored the endpoint using burp suite interceptor.
3. Used Burp Suite to send a `GET` request to `/delete-user.php` with another user's `id` with the session token of a normal user.
4. The server processed the request and deleted the specified user without proper authentication or authorization.

Impact

- Any authenticated user can **delete any other user**, including staff or administrators.
- Can be used for **denial of service** (mass deletion of user accounts).
- No additional authentication or confirmation is required, making this vulnerability severe.

Mitigation

- **Require Proper Authorization**: Validate user roles before allowing any delete action.
- **Use POST Requests with CSRF Protection**: Prevent attackers from exploiting the endpoint via GET requests.
- **Log All Deletion Attempts**: Alert admins if multiple deletion attempts are made.

---

3. Arbitrary File Upload and Remote Code Execution

Discovery

- **Vulnerability Type:** Unrestricted File Upload
- **Affected Feature:** File upload functionality
- **How Found:** Uploading a `.pdf.php` file and executing remote commands.

Procedure

1. Uploaded a PHP file named `shell.pdf.php`, bypassing the server's naive extension check. The actual payload is uploaded in gradescope.

2. Accessed the file via `http://localhost:8888/handins/shell.pdf.php` (The hash of the filename)
3. Executed remote commands using: http://localhost:8888/uploads/shell.pdf.php?cmd=whoami
4. Verified remote code execution by running system commands like `ls` and `whoami`.

Impact

- **Full server compromise**: Attackers can execute arbitrary commands.
- **Data theft, database manipulation, and persistent malware installation**.
- Can be leveraged for further **privilege escalation** within the system.

Mitigation

- **Whitelist File Types**: Allow only specific file types (`.jpg`, `.png`, `.pdf`), and verify contents.
- **Store Files Outside Webroot**: Move uploaded files to a directory where execution is not possible.
- **Use MIME Type Checking**: Verify file contents instead of relying on extensions.
- **Disable Execution in Uploads Directory**: Use `.htaccess` or server config to prevent `.php` execution.

---

4. Stored XSS via Profile Comments

Discovery

- **Vulnerability Type:** Stored Cross-Site Scripting (XSS)
- **Affected Feature:** Profile comment system
- **How Found:** Injecting JavaScript payload into the comment field via intercepted requests.

Procedure

1. Used Burp Suite to modify a comment submission request.
2. Injected the following payload (URL encoding of the payload): <svg onload=alert('XSS')>
3. Comment was stored in the database and executed when viewed by any user.
4. Verified that **all users** visiting the affected profile saw the JavaScript execute.

Impact

- **Account Hijacking**: Attackers can steal session cookies.

- **Phishing & Data Theft**: Inject malicious redirects to steal credentials.
- **Persistent Exploitability**: Code is stored in the database and affects every user who views the page.

Mitigation

- **Sanitize User Input**: Remove or encode `<script>`, `<svg>`, and other HTML elements.
- **Use a Content Security Policy (CSP)**: Restrict inline JavaScript execution.
- **Validate Input on Server-Side**: Ensure JavaScript is filtered before storing comments.

---

5. SQL Injection on Login

Discovery

- **Vulnerability Type:** SQL Injection
- **Affected Endpoint:** Login form
- **How Found:** Using `' OR 1=1 --` as the username.

Procedure

1. Used Burp Suite to intercept the login request.
2. Replaced the username field with:
   ' OR 1=1 --
3. Submitted the request and successfully logged in as `id=1` (a staff account named `lhoneycutt`).
4. Verified that **any password** works due to the `OR 1=1` clause making the WHERE condition always true.

Impact

- **Complete system compromise**: Can log in as any user, including admins.
- **Data breach**: Attackers can extract all user credentials from the database.
- **Site-wide takeover**: Can modify or delete critical data.

Mitigation

- **Use Prepared Statements**: Never concatenate user input directly into SQL queries.
- **Escape User Input Properly**: Sanitize input using PHP's `PDO::prepare()`.
- **Implement Least Privilege**: Restrict database permissions to prevent mass data exposure.

Conclusion

This report highlights five **critical** security vulnerabilities found in the FLAG Portal:

1. **User Deletion via Unauthenticated Endpoint**
2. **Unauthorized Account Takeover via Profile Edit**
3. **Arbitrary File Upload and Remote Code Execution**
4. **Stored XSS via Profile Comments**
5. **SQL Injection on Login**

Each vulnerability poses a significant risk, from **full server takeover** to **compromising user accounts** and **stealing sensitive data**. The recommended mitigations include **proper authentication**, **input validation**, **prepared statements**, and **secure file handling** to protect the application from exploitation. Addressing these issues will significantly improve the security posture of the FLAG Portal.

Appendix: Exploit Code Samples

- **Proof-of-Concept Payloads** for each vulnerability are included in the supporting files.
- Video demonstrations show the exploitation process and expected impact.
