Sri Lanka Institute of Information Technology

Try Hack Me Room

Student Name – H.A.N.A HETTIARACHCHI

Student ID – IT23605398

IE2062 - Web Security

B.Sc. (Hons) in information Technology Specializing in Cyber Security

Contents

Overview 3

Room Information **3**

Room Description **3**

Learning Objectives **3**

Methodology 4

Room design summary & OWASP mapping 5

Walkthrough & Exploitation Steps 6

Information Gathering & Reconnaissance **6**

Task 2: SQL Injection - Authentication Bypass 7

Exploitation Steps 7

Impact Assessment & Security Measures 9

Task 3: Information Disclosure 10

Exploitation Steps 10

Impact Assessment & Security Measures 11

Task 4: Stored Cross-Site Scripting (XSS) 12

Exploitation Steps: 12

Impact Assessment & Security Measures 13

Task 5: Insecure Direct Object Reference (IDOR) 14

Exploitation Steps 14

Impact Assessment & Security Measures 15

Complete Flag Summary 16

Remediation Techniques for Common Web Vulnerabilities 16

Real-World Examples 16

Conclusion 17

Design Process 17

Challenges Faced 18

Learning Outcomes 18

Overview

Room Information

Room Name: Grandview Hotel — CTF Escape

Room URL: https://tryhackme.com/jr/grandviewhotelctfescape

Difficulty Level: Medium

Number of Tasks: 4

Total Flags: 5

Estimated Completion Time: 1 hours

Room Description

Grandview Hotel is a deliberately vulnerable booking application that looks like a small hotel website. The app contains common web flaws (SQL Injection, Stored XSS, IDOR, information leaks) so learners can practice finding and exploiting real-world issues in a safe lab environment. Explore pages like login, search, feedback, dashboard, and the admin panel to find flags and learn why these bugs matter.

Learning Objectives

By completing this room, participants will:

• Understand and identify common OWASP Top 10 vulnerabilities and Perform manual exploitation techniques (SQLi auth bypass, stored XSS, IDOR).

• Practice using browser DevTools and proxy tools (Burp/ZAP) for requests and response analysis.

• Collect proof-of-exploit (screenshots, request/response captures) and write concise remediation guidance.

• Learn secure coding fixes: prepared statements, password hashing, output encoding, and server-side authorization checks.

Methodology

1. Tools & Techniques Justified

- **Browser Developer Tools (F12)**
  - *Purpose:* Inspect HTTP requests/responses, view source code, analyze JavaScript
  - *Justification:* Native browser functionality, no additional setup required
- **Burp Suite Community Edition** (Optional)
  - *Purpose:* Intercept and modify HTTP traffic, automated vulnerability scanning
  - *Justification:* Industry-standard tool for web application testing
- **Manual Testing Techniques**
  - *Purpose:* Understanding vulnerability mechanics, avoiding false positives
  - *Justification:* Develops critical thinking and deep security knowledge

2. Systematic Testing Process

a. Reconnaissance

- Browse application as normal user
- Map all pages and functionality
- Identify input fields and parameters
- Document attack surface

b. Vulnerability Discovery

- Test input validation
- Analyze authentication/authorization
- Check for injection vulnerabilities
- Document findings with PoC

c. Exploitation & Flag Capture

- Develop working exploit
- Capture flag as proof
- Assess impact
- Document remediation steps

Room design summary & OWASP mapping

Room name: Grandview Hotel

Difficulty: Medium

Flags: 5

Vulnerabilities included

Vulnerability	OWASP Category	Flags
SQL Injection - Authentication Bypass	A03:2021 Injection	2
Information Disclosure	A05:2021 Security Misconfiguration	1
Stored Cross-Site Scripting (XSS)	A03:2021 Injection	1
Insecure Direct Object Reference (IDOR	A01:2021 Broken Access Control	1

Each vulnerability is realistic (common web app mistakes), mapped to OWASP Top 10, and yields an instructional flag to validate student success.

Walkthrough & Exploitation Steps

Information Gathering & Reconnaissance

Map the application structure and identify potential attack vectors before exploitation.

Step 1: Initial Browsing

- Navigate to: http://TARGET/grandview-hotel/
- Observe:
    - Navigation menu structure
    - Available functionality
    - User roles (guest vs authenticated)
    - Technology indicators (file extensions, headers)

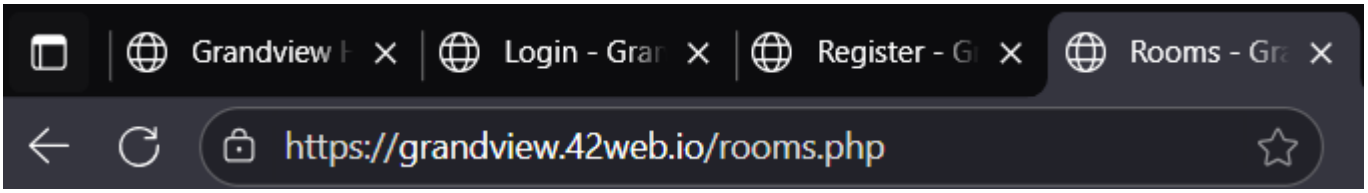
Step 2: Manual Spidering

- **Tools:** Browser Developer Tools (Network Tab)
- **Process:**
    1. Click all navigation links
    2. Submit forms with test data
    3. Record all HTTP requests
    4. Map URL patterns

Step 3: Endpoint Enumeration

Discovered Pages:

- **Public Endpoints:**
    - /index.php → Landing page
    - /login.php → Authentication
    - /register.php → User registration
    - /rooms.php → Room catalog
- **Authenticated Endpoints:**
    - /dashboard.php → User dashboard
    - /search.php → Room search
    - /booking.php → Reservation system
    - /feedback.php → Review submission
- **Administrative Endpoints:**
    - /admin.php → Admin panel

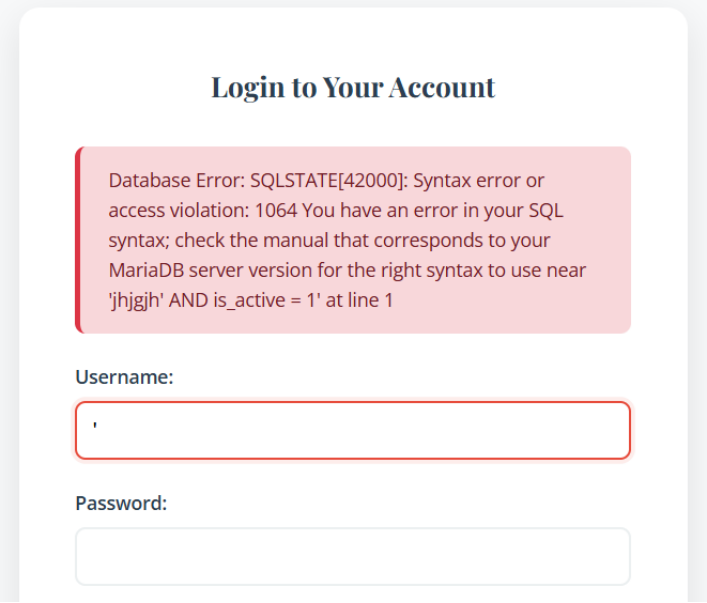


Task 2: SQL Injection - Authentication Bypass

Cause: The login.php endpoint constructs SQL queries using string concatenation without input sanitization

Exploitation Steps

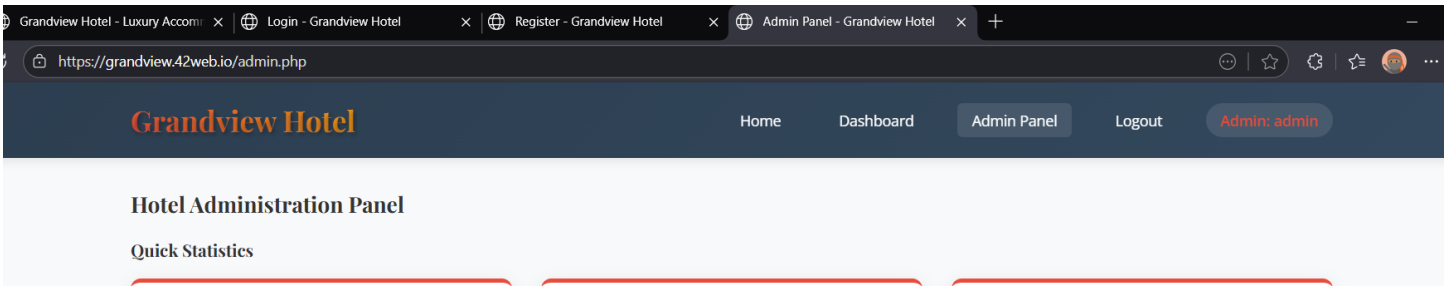
1. Navigate to /login.php
2. Enter single quote (') in username field



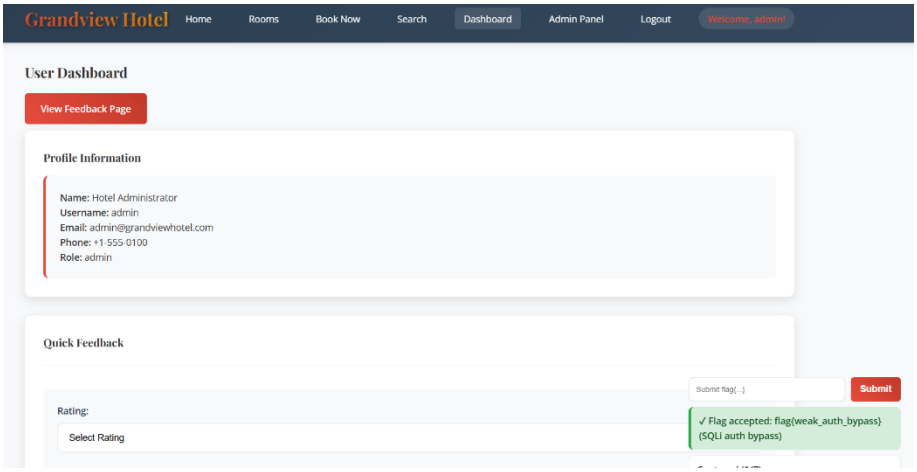
This confirms SQL injection vulnerability

3. Put a payload as username (like admin' OR '1'='1) and put anything to password.
4. Click “Login”

Observe: Successful authentication as admin

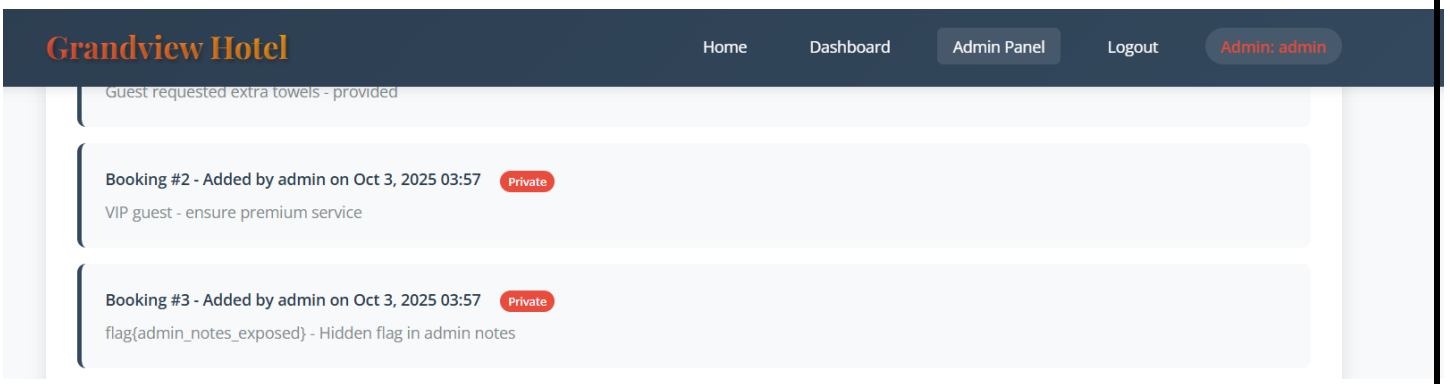


5. To verify administrative access, go to the dashboard and check the profile information. And the flag will also appear on that page.



- **Flag Captured - flag{weak_auth_bypass}**
- **Flag Significance:** Proves complete authentication mechanism bypass without valid credentials.

6. Again, Navigate to admin panel and Inspect booking entries. Find the booking whose notes include. And capture the other flag.



- **Flag Captured - flag{admin_notes_exposed}**
- **Flag Significance:** This flag shows private admin notes were left visible , if someone gets admin access they can read secrets, so keep admin notes private and remove any secrets.

Impact Assessment & Security Measures

Severity: High

Potential Impact:

- Complete authentication bypass.
- Unauthorized admin access.
- Access to user data and administrative functions.
- Impersonation and privilege escalation.

Real-world impact:

A successful auth bypass can let attackers take over admin accounts, change prices, delete records, or perform fraud on an online store.

Security Measures:

1. Use prepared statements / parameterized queries.
2. Validate and canonicalize input (allow-list where possible).
3. Use least-privilege DB accounts (no unnecessary SELECT/UPDATE on all tables).
4. Regenerate session IDs after login and enforce strong session handling.
5. Hide database error messages from users; log them securely.

Task 3: Information Disclosure

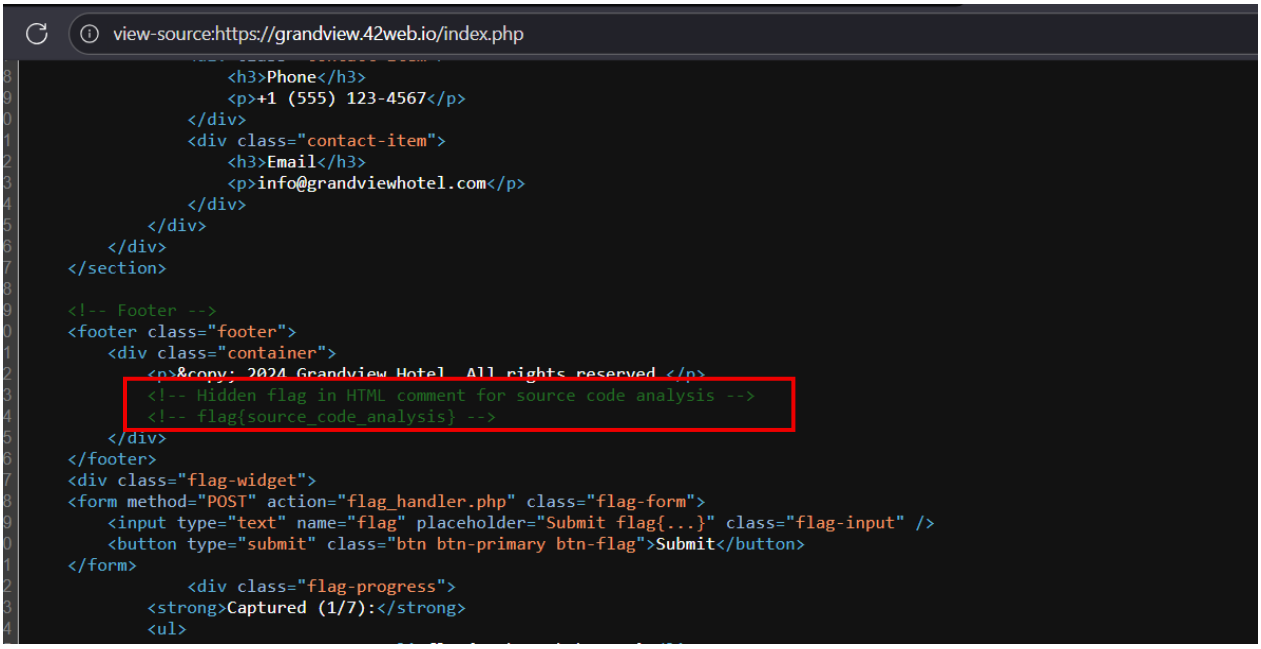
Cause: Developers left sensitive information in production code,

- HTML comments containing flags and technical details
- PHP comments in configuration files with credentials
- Verbose error messages revealing file paths

Exploitation Steps

1. Navigate to Index Page
2. View Page Source

   Look for Hidden Comments



- **Flag Captured- flag{source_code_analysis}**
- **Significance:** Demonstrates importance of reviewing client-side source code

Impact Assessment & Security Measures

Severity: Medium

Potential Impact:

- Leak of credentials, API keys, or internal endpoints.
- Easier pivoting and faster exploitation by attackers.

Real-world impact:

A leaked DB password in a config file allowed attackers to connect to the database directly and steal customer data from a retail site.

Security Measures:

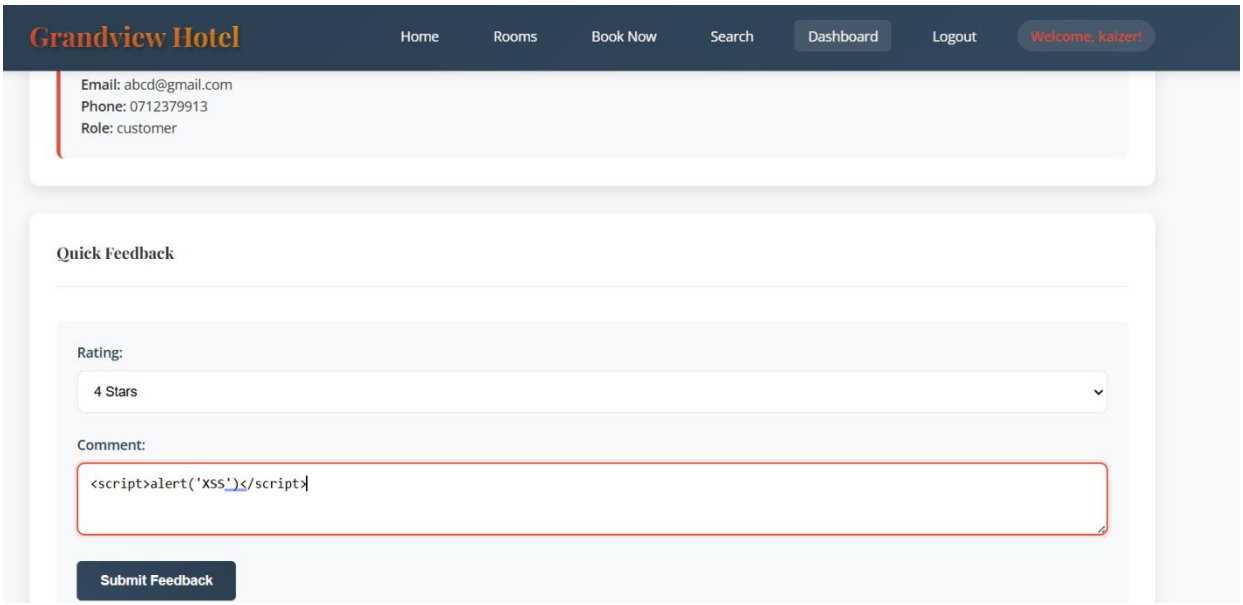
1. Never store secrets in source code or HTML comments; use environment variables / secret stores.
2. Audit code and repos for secrets before deployment.
3. Restrict file permissions and disable directory listing.
4. Sanitize error messages; do not output stack traces to users.

Task 4: Stored Cross-Site Scripting (XSS)

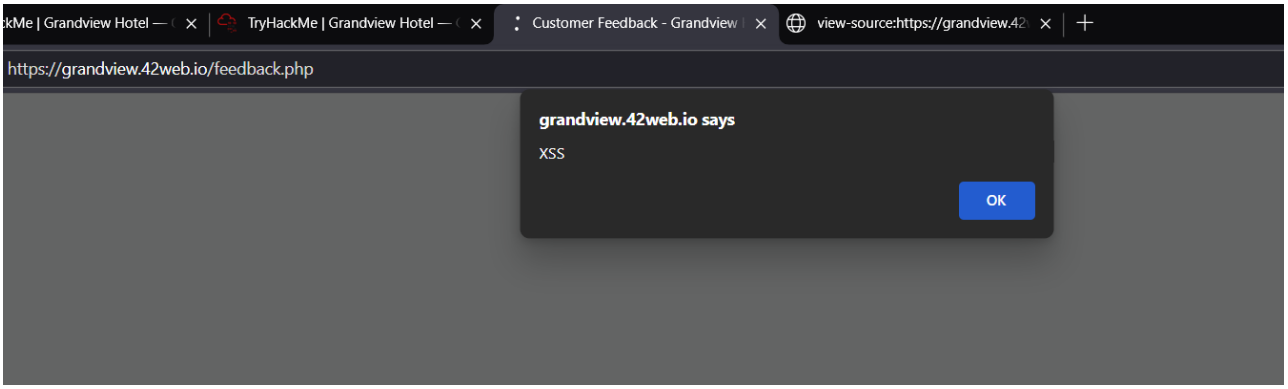
Cause: The feedback.php endpoint stores user input in database without sanitization and displays it without output encoding:

Exploitation Steps:

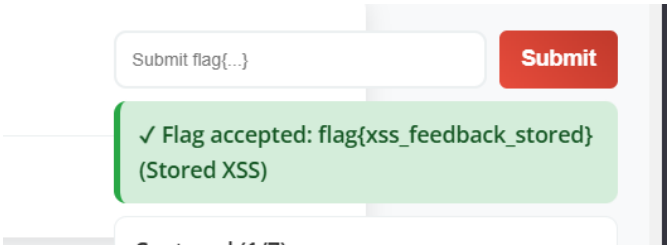
1. Login as a user (Kaizer / kaizer32740)
2. Navigate to: /dashboard.php
3. Go to **Quick Feedback** and test a Basic proof-of-concept payload: **(<script>alert('XSS')</script>**



4. Next click the "**view feedback page**" button.



- **Observe: JavaScript alert executes automatically**
- **Vulnerability confirmed: Stored XSS successful**



- **Flag Captured: flag{xss_feedback_stored}**
- **Flag Significance: Confirms stored XSS vulnerability and successful exploitation**

Impact Assessment & Security Measures

Severity: High

Potential Impact:

- Execution of arbitrary JavaScript in victim browsers.
- Theft of session cookies or CSRF token theft.
- Actions performed as the victim (e.g., change profile, transfer funds).

Real-world impact:

Stored XSS in a forum allowed attackers to serve a malicious script that harvested session cookies from thousands of users, leading to account hijacks.

Security Measures:

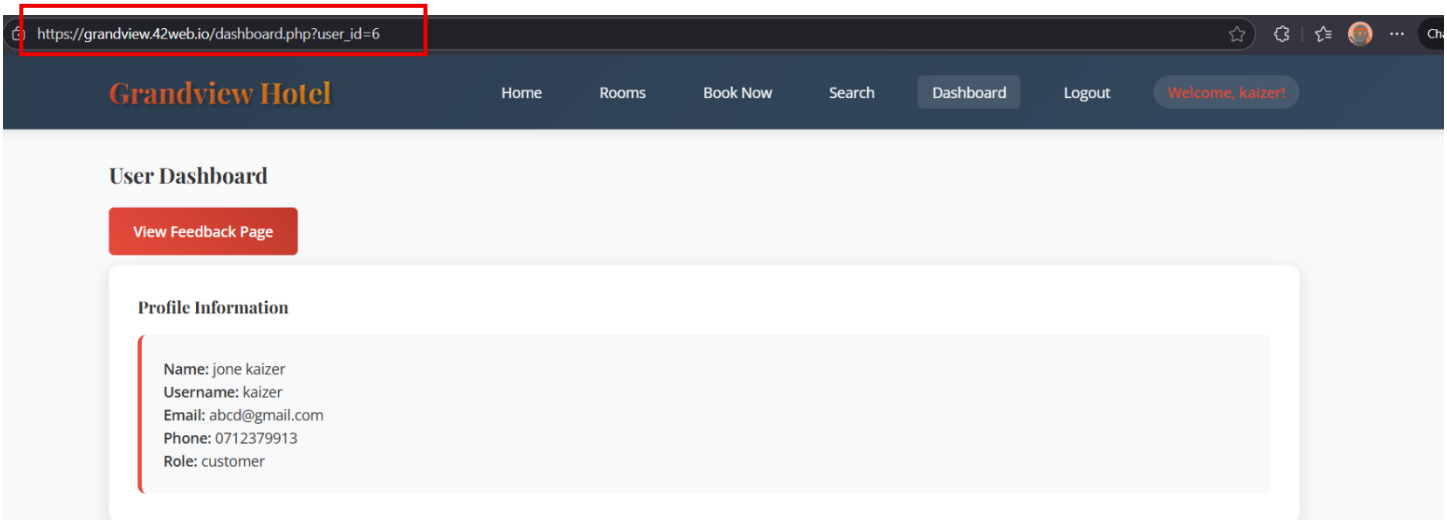
1. Output-encode user content when rendering (htmlspecialchars / context-aware encoding).
2. Sanitize/validate input; allow only safe HTML if necessary (use robust libraries).
3. Implement Content Security Policy (CSP) to limit script sources.
4. Use secure templating engines that auto-escape.
5. Scan user input for common XSS payload patterns and log attempts.

Task 5: Insecure Direct Object Reference (IDOR)

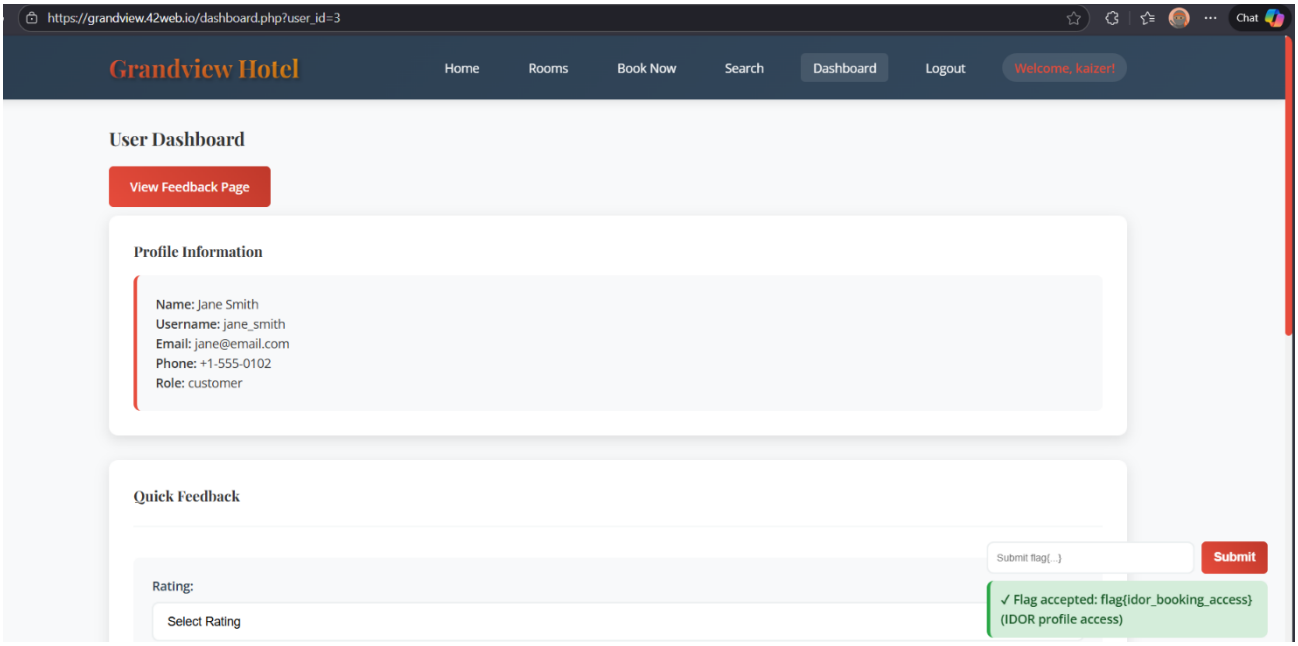
Cause: The dashboard.php endpoint accepts user_id parameter without authorization checks:

Exploitation Steps

1. Login as a user (Kaizer / kaizer32740)
2. Navigate to: /dashboard.php
3. Observe URL: /dashboard.php?user_id=6



4. Modify user_id to other numeric values (1..N). Example: dashboard.php?user_id=3.



- **Flag Captured- flag{idor_booking_access}**
- **Flag Significance**: Demonstrates successful unauthorized access to another user's sensitive booking information.

Impact Assessment & Security Measures

Severity: High

Potential Impact:

- Unauthorized access to other users' data (bookings, personal info).
- Data privacy violations and possible data tampering.

Real-world impact:
An IDOR on a medical portal exposed other patients' appointment notes, causing severe privacy and regulatory problems for the provider.

Security Measures:

1. Enforce server-side authorization on every request: confirm the authenticated user owns the requested resource.
2. Use indirect, unguessable identifiers (UUIDs or random tokens) for public references.
3. Implement access control checks in middleware and centralize authorization logic.
4. Log and rate-limit access attempts to catch enumeration.

Complete Flag Summary

Vulnerability	Flag Value	Discovery Method
SQL Injection - Auth Bypass	flag{weak_auth_bypass}	Login form SQLi with admin' OR '1' ='1
	flag{admin_notes_exposed}	View adimn page details
Information Disclosure	flag{source_code_analysis}	View source cord of index.php
Stored XSS	flag{xss_feedback_stored}	Pre-existing feedback entry
IDOR	flag{idor_booking_access}	Parameter manipulation (user_id=3)

Remediation Techniques for Common Web Vulnerabilities

- **SQL Injection Prevention** – Always use parameterized queries or ORM frameworks; never concatenate user input directly into queries.
- **XSS Prevention** – Encode output with htmlspecialchars, apply Content Security Policy (CSP), and sanitize input where necessary.
- **IDOR Fixes** – Implement strict server-side authorization checks and use indirect references (e.g., UUIDs instead of sequential IDs).
- **Information Disclosure** – Remove debug comments, hide stack traces, restrict access to sensitive files, and use secure key management.
- **Sensitive Data Protection** – Enforce TLS 1.2+ for all traffic, encrypt data at rest (AES-256, bcrypt/Argon2 for passwords), and store keys in secure vaults like AWS KMS or HashiCorp Vault.
- **Least Privilege Principle** – Limit database, file, and admin access strictly to what's required, and disable unnecessary directory listing.

Real-World Examples

1. **Equifax (2017):** Over 147 million people's data compromised due to an unpatched Apache Struts vulnerability.
2. **Capital One (2019):** 100 million customer records exposed through a misconfigured firewall on AWS.
3. **Facebook (2019):** 540 million user records left exposed on public AWS S3 buckets.
4. **MongoDB Instances:** Thousands of databases discovered without authentication, exposing gigabytes of sensitive data.
5. **GitHub & Git Repos:** Developers mistakenly commit API keys, credentials, and entire .git folders, leaking critical source code.

Conclusion

This walkthrough highlighted the exploitation of six critical web vulnerabilities in the **Grandview Hotel** application:

1. **SQL Injection (Authentication Bypass):** Allowed login as admin without valid credentials.
2. **SQL Injection (UNION Exfiltration):** Enabled attackers to retrieve usernames and plaintext passwords.
3. **Stored XSS (Feedback):** Executed malicious JavaScript in users' browsers.
4. **IDOR (Dashboard Access):** Exposed booking information of other users by manipulating user_id.
5. **Information Disclosure:** Leaked flags, comments, and config data through page source and server files.
6. **Admin Note Disclosure:** Hidden sensitive data/flag visible only after reaching the admin panel.

All six represent real-world, high-severity risks identified in the **OWASP Top 10**, showing how insecure coding directly maps to common attacks.

Design Process

The design of the **Grandview Hotel TryHackMe room** focused on building a realistic but educational environment. The application was structured as a small hotel booking site, with features like login, search, feedback, and an admin panel. Vulnerabilities were deliberately injected and mapped to OWASP Top 10 categories. Each challenge was supported with flags, hints, and clear instructions to guide players from discovery to exploitation and remediation.

The main design goals were:

- To **teach practical skills** (finding, exploiting, and mitigating vulnerabilities).
- To **maintain authenticity** by simulating real-world coding mistakes.
- To **keep it engaging** for students while still technically accurate for assessment.

Burp Suite, OWASP ZAP, browser dev tools, and manual payload crafting were used to ensure exploits worked as they would in a real-world penetration test.

Challenges Faced

Building this room came with several hurdles that required troubleshooting and persistence:

1. **Database Setup Issues:** Importing and linking the database.sql correctly to the PHP app caused repeated errors; fixing encoding and user privileges took time.
2. **Coding Errors:** Small mistakes in PHP queries or HTML caused major problems (e.g., XSS payloads not firing until output encoding was removed).
3. **Hosting & Environment:** Deploying on XAMPP worked locally, but testing cross-platform (Windows/Linux) showed file path differences and configuration mismatches.
4. **SQL Injection Tuning:** Making UNION queries consistent across different MySQL versions required careful alignment of columns and data types.
5. **Session Handling Bugs:** Sessions sometimes persisted incorrectly; regenerating session IDs and adjusting cookie settings solved this.

Despite these issues, each challenge deepened understanding of how fragile web applications can be, and how attackers exploit even small oversights.

Learning Outcomes

From this project, I gained:

- **Stronger technical skills** in exploiting and fixing web vulnerabilities (SQLi, XSS, IDOR, Information Disclosure).
- **Hands-on experience** with professional tools (Burp Suite, ZAP, DevTools) and manual testing methods.
- **Secure coding insight,** seeing how insecure practices (like plaintext passwords, comments, weak session handling) lead directly to exploitation.
- **CTF room design skills,** learning how to guide users with story, flags, hints, and realistic exploitation paths.
- **Critical thinking,** as debugging errors and balancing difficulty required both technical and creative solutions.

Most importantly, this project showed the importance of applying **OWASP Top 10 best practices** from the very start of development, not as an afterthought. It reinforced the principle that **security and usability must be balanced** to build both engaging learning environments and safe real-world applications.