The code is written in JavaScript using the Express.js framework, a popular framework for building web applications in Node.js.

Here are the potential security issues in the provided code:

1. **Open Redirect Vulnerability**:

   - The code redirects the user to a URL provided in the `redirect_url` query parameter if the authentication is successful. This can be exploited by attackers to redirect users to malicious websites. Open redirects can be used for phishing attacks, stealing credentials, or tricking users into downloading malicious software.

2. **Lack of Body Parser**:

   - The code expects `username` and `password` to be present in `req.body`, but there's no mention of using a body parser middleware (like `express.json()` or `express.urlencoded()`). Without this middleware, `req.body` would be `undefined`.

3. **Potential for Verbose Error Messages**:

   - The code returns an 'Incorrect Credentials Supplied' message if authentication fails. While this specific message is generally safe, it's essential to be cautious about the verbosity of error messages, as overly specific messages can provide attackers with information about the application's internal workings.

4. **No Mention of HTTPS**:

   - The application listens on an HTTP server. For security, especially when handling sensitive operations like logins, HTTPS should be used to encrypt the traffic between the client and server.

5. **Lack of Input Validation and Sanitization**:

   - The code doesn't seem to validate or sanitize the `username` and `password` inputs before passing them to the `auth` function. Depending on the implementation of the `auth` function, this could lead to various vulnerabilities like SQL injection, if these inputs are used in constructing database queries.

6. **No Rate Limiting**:

   - The code doesn't implement rate limiting for the login endpoint. Without rate limiting, attackers can perform brute force attacks by trying many username and password combinations in a short amount of time.

7. **Session Management**:

   - The code doesn't provide details about how user sessions are managed after a successful login. Proper session management, including secure session cookie handling and session expiration, is critical for security.

To address these security issues:

- Use a whitelist approach for redirects, or validate the `redirect_url` to ensure it points to a trusted or known location.

- Ensure you use middleware like `express.json()` or `express.urlencoded()` to parse incoming request bodies.

- Use HTTPS for secure communication.

- Always validate and sanitize user inputs before processing them.

- Implement rate limiting to prevent brute force attacks.

- Ensure secure session management with features like HttpOnly and Secure flags for cookies, and consider using a well-established session management library.

Always consult security best practices and guidelines specific to the frameworks and libraries you're using.