The code is written in **JavaScript** and uses the **Express.js** framework, which is a web application framework for **Node.js**. It also uses the `xpath` and `xmldom` libraries to parse and query XML data.

The primary security concern in this code is:

1. **XML External Entity (XXE) Attack**:

   - The XML parser used (`xmldom`) can be susceptible to XXE attacks if not configured properly. An attacker can exploit this vulnerability by sending malicious XML content, potentially leading to the disclosure of internal files, denial of service, or server-side request forgery.

   - The application reads an XML file from `/var/tmp/export.xml`, and then the content is parsed with `new dom().parseFromString(xml);`. If this XML contains a definition for an external entity, it can lead to the aforementioned issues.

2. **XPath Injection**:

   - The application constructs an XPath query by directly concatenating user input from `req.query.team` into the query string:

     ```javascript
     const nodes = xpath.select("/teams/team[name='" + teamName + "']/members/member/name/text()", doc);
     ```

     This can be exploited by an attacker to inject malicious XPath content, potentially leading to information disclosure or denial of service.

3. **Error Disclosure**:

   - The application sends raw error messages to the client:

     ```javascript
     res.send(e.message);
     ```

     This can disclose internal implementation details to the user, which can be used by an attacker to further understand the system and craft more precise attacks.

Recommendations:

1. **Prevent XXE**:

   - Configure the XML parser to disallow the definition and usage of external entities. Check the documentation of your parser and ensure you're using a secure configuration.

2. **Parameterize XPath Queries**:

   - Avoid constructing XPath queries using string concatenation with user input. Instead, use parameterized queries or validate and sanitize the user input to prevent XPath injection.

3. **Limit Error Disclosure**:

   - Instead of sending raw error messages to the client, log the error details server-side and send a generic error message to the client. This prevents revealing potentially sensitive information.

4. **Input Validation**:

   - Always validate and sanitize user input to ensure it meets expected patterns and values.

By addressing these concerns, the security of the code can be significantly improved.