The language used in the provided code is JavaScript, which is often used for web development in both the frontend and backend (using Node.js).

There are several potential security and design issues in this code:

1. **Prototype Pollution**:

   - The `merge` function is vulnerable to prototype pollution. If the `source` object contains properties like `__proto__` or `constructor`, an attacker could potentially add or modify properties on all objects, leading to unexpected behaviors or even code execution in some cases.

   - The `updateUser` function uses this `merge` function to merge the request body into the `userData` object. This opens up the possibility for an attacker to send a malicious payload in the request body to exploit the prototype pollution vulnerability.

2. **Potential Direct Object Reference**:

   - The `updateUser` function takes a `userId` and updates the corresponding user's data. Depending on how this function is called and how user authentication is implemented, there might be a potential for an attacker to update another user's data by manipulating the `userId`.

3. **Logging Sensitive Data**:

   - In the `updateUser` function, the entire `userData` is being logged with `log("Saving userData " + userData.toString());`. This might leak sensitive user information in the logs.

4. **Insecure Default Role**:

   - In the `getRole` function, the default role is set to "user". Depending on the application's design, it might be safer to default to a more restrictive role or to no role at all, ensuring that permissions are explicitly granted.

5. **Misuse of `__proto__`**:

   - Directly manipulating or accessing the `__proto__` property can be dangerous and is generally discouraged in modern JavaScript development. It's better to use safer alternatives like `Object.getPrototypeOf()` or `Object.setPrototypeOf()`, or avoid manipulating prototypes directly.

   - The first few examples in the code are demonstrating the potential pitfalls of direct prototype manipulation.

6. **Not Checking for Own Properties in `merge`**:

- In the `merge` function, there's a loop that iterates over properties of the `source` object. It's generally a good practice to check if the property belongs to the source object itself (and not its prototype) using `source.hasOwnProperty(attr)`.

To enhance security:

- Ensure that user input is validated and sanitized before being processed.
- Refactor the `merge` function to prevent prototype pollution.
- Be careful with logging. Avoid logging sensitive user data.
- Set more secure defaults.
- Avoid directly manipulating or accessing the `__proto__` property.
- Always check for own properties when iterating over an object's properties.

And, as always, ensure you're following best practices and guidelines specific to the frameworks and libraries you're using.