The code is written in **Java**. The security issue and concerns in the code are:

1. **Storing Sensitive Data in Cleartext**:

   - In the first block of code, user credentials (username and password) are stored in cleartext form in a cookie. This exposes the credentials to anyone who can access the cookie. This is a major security risk as cookies can be intercepted, especially if transmitted over an insecure connection.

   ```java
   data = credentials.getUserName() + ":" + new String(credentials.getPassword());
   // BAD: store data in a cookie in cleartext form
   response.addCookie(new Cookie("auth", data));
   ```

2. **Use of Weak Password Hashing**:

   - In the second block, the code hashes the credentials using SHA-512 with a hardcoded salt. While SHA-512 is a strong cryptographic hash function, using it for password hashing is not recommended because it's designed to be fast. Password hashing algorithms should be slow to make brute-force attacks less effective. Cryptographic algorithms like bcrypt, scrypt, or Argon2 are better choices for password hashing.

3. **Use of Hardcoded Salt**:

   - The code uses a hardcoded salt (`"ThisIsMySalt"`) for hashing. Salts should be unique for each user and credential set. Using a hardcoded salt across all hashed data negates many of the benefits of salting.

   ```java
   String salt = "ThisIsMySalt";
   ```

4. **Ambiguity in "Encrypted" Comment**:

   - The comment mentions storing data in encrypted form, but the data is actually being hashed, not encrypted. This can cause confusion.

   ```java
   ```

```
// GOOD: store data in a cookie in encrypted form
response.addCookie(new Cookie("auth", data));
```

Recommendations:

1. **Avoid Storing Credentials in Cookies**:
   - Even in hashed form, storing credentials in cookies is risky. Use session IDs or tokens instead, and keep the actual credentials server-side.

2. **Use Proper Password Hashing Algorithms**:
   - Use bcrypt, scrypt, or Argon2 for password hashing. These algorithms are designed for securely hashing passwords and include features like built-in salting and key stretching.

3. **Use Dynamic Salts**:
   - Use a unique salt for each user or set of credentials. Store the salt alongside the hashed data so it can be used during the verification process.

4. **Use HTTPS**:
   - Ensure your application uses HTTPS to prevent the interception of data, including cookies, during transmission.

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