Spring Security

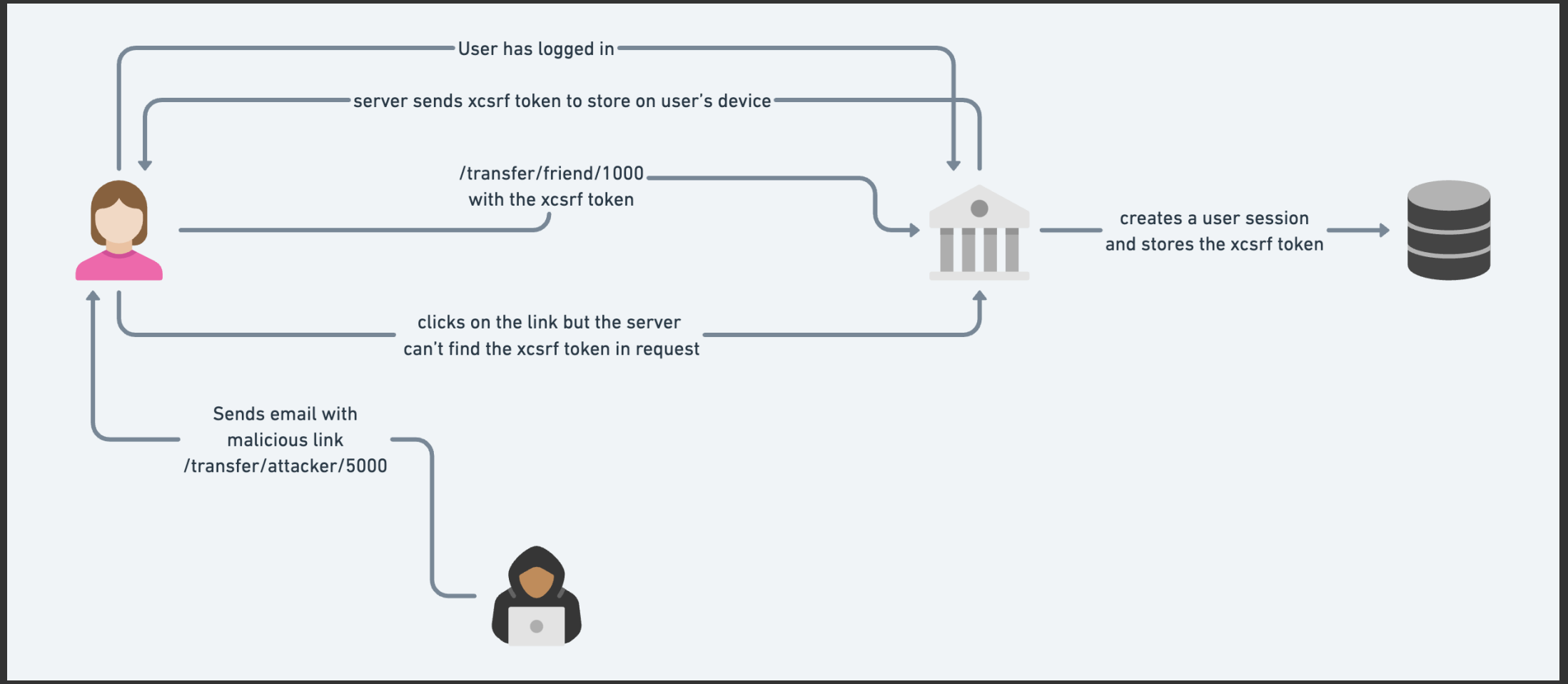
Security Attacks

Cross-Site Request Forgery (CSRF)

Cross-Site Request Forgery (CSRF) is an attack that tricks a user into performing actions on a web application in which they are authenticated, without their consent. This is achieved by exploiting the web application's trust in the user's browser.

For example, if a banking site doesn't protect against CSRF, an attacker could trick a logged-in user into making a bank transfer without their knowledge.

Cross-Site Request Forgery (CSRF)



Cross-Site Request Forgery (CSRF)

How to avoid:

1. Maintain user session on server with the xcsrf token. This token should be very unpredictable.
2. Go STATELESS and manage authentication with the JWT token.

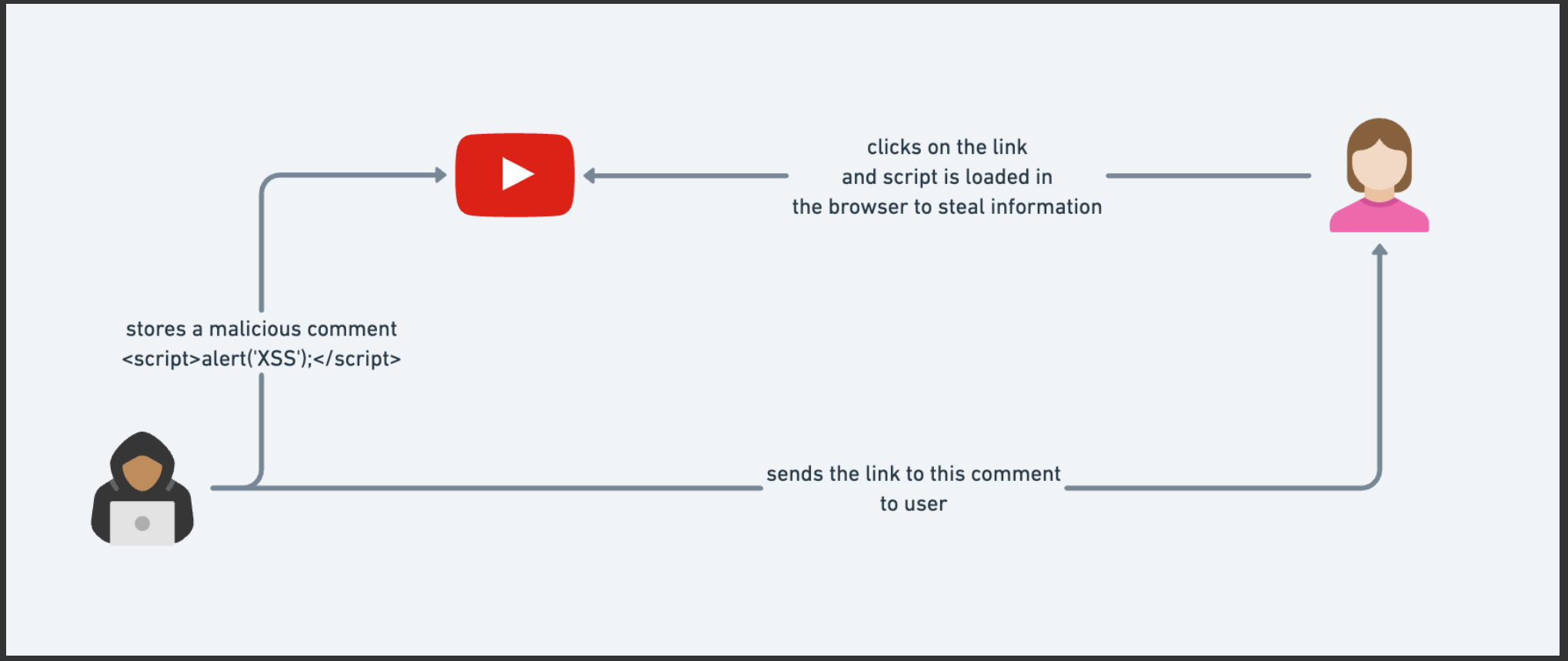
Cross-Site Scripting (XSS)

Cross-Site Scripting (XSS) is a type of security vulnerability typically found in web applications. XSS attacks occur when an attacker is able to inject malicious scripts into web pages viewed by other users. These scripts can steal user data, hijack user sessions, deface websites, or perform other malicious actions.

e.g. This script could be stored by a user as comments

<script>alert('XSS');</script>

Cross-Site Scripting (XSS)



Cross-Site Scripting (XSS)

How to avoid:

1. Input Validation:
Validate and sanitize all user inputs to ensure they do not contain malicious code. Use frameworks or libraries that automatically handle input validation.

2. Output Encoding:
Encode all outputs to ensure that any potentially dangerous characters are rendered harmless in the browser. This prevents injected scripts from being executed.

```
HtmlUtils.htmlEscape(userInput);
```

SQL Injection

SQL Injection is a type of attack that allows attackers to execute arbitrary SQL queries on a database by injecting malicious SQL code into an application's input fields.

The attacker manipulates this input to include SQL code that changes the intended query, often to retrieve or manipulate data in unauthorized ways.

Server query: *SELECT * FROM users WHERE username = 'input';*
Attacker input: *' OR '1'='1*
Resulting query:
*SELECT * FROM users WHERE username = '' OR '1'='1';*

SQL Injection

How to avoid:

- Use prepared statements and parameterized queries.
- Employ ORM frameworks that abstract direct SQL queries.
- Validate and sanitize inputs.