Web security

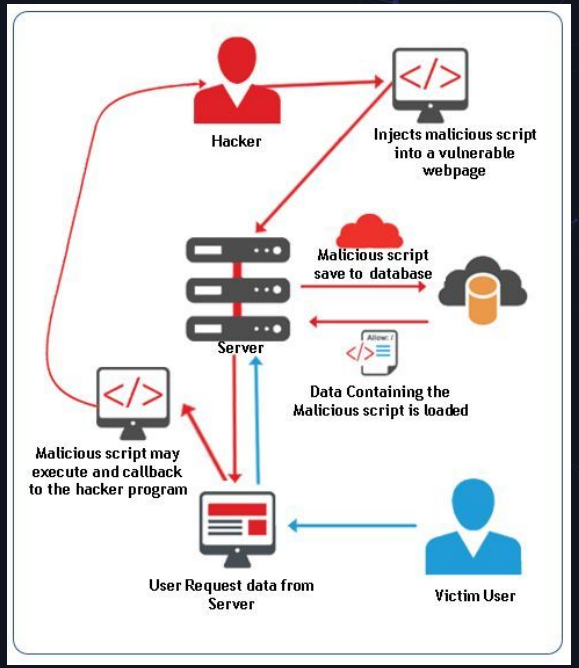
XSS Injections

XSS injections
The most common of all

"Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted websites. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user.

Flaws that allow these attacks to succeed are quite widespread and occur anywhere a web application uses input from a user within the output it generates without validating or encoding it."

Source: owasp.org



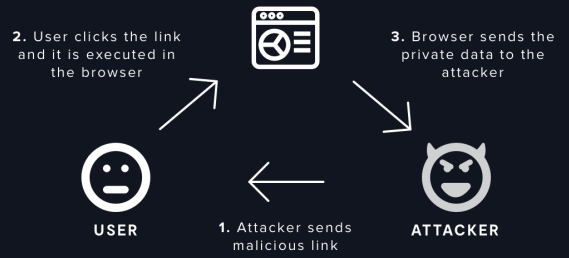
XSS injections
Reflected XSS Attacks

Reflected attacks are those where the injected script is reflected off the web server, such as in an error message, search result, or any other response that includes some of the input sent to the server as part of the request.

Reflected attacks can be delivered to victims like an e-mail, or a redirection from another website.

When a user is tricked into clicking on a malicious link, the injected code travels to the vulnerable web site, which reflects the attack back to the user’s browser.

The browser then executes the code because it came from a “trusted” server.



XSS injections
Reflected XSS Attacks - Example

```
npm install
node server.js
```

Now go there: http://localhost:3000

It should be working.

As an attacker, you also notice that you can customize the message:

http://localhost:3000/?title=Hey!&message=I-am-a-custom-message

XSS injections
Reflected XSS Attacks - Example

Let’s get serious:

http://localhost:3000/?title=Hey!&message=<script>alert('You have been hacked')</script>

Oopsie!

XSS injections
Reflected XSS Attacks - Example

That was nice, let’s continue:

http://localhost:3000/?title=Hey!&message=<script>alert(document.cookie)</script>

That’s even worse, you accessed the user’s cookies!

XSS injections
Reflected XSS Attacks - Example

NEVER trust user input.

XSS injections
Reflected XSS Attacks - Example

Ok, how could we fix that?

XSS injections
Reflected XSS Attacks - Example

Before displaying it anywhere, escape your data.

HTML escaping is removing traces of offending characters that could be wrongfully interpreted as markup. The following characters are reserved in HTML and must be replaced with their corresponding HTML entities:

" is replaced with "
& is replaced with &
< is replaced with <
> is replaced with >

There is a npm package just for that:
https://www.npmjs.com/package/escape-html

XSS injections
Reflected XSS Attacks - Example

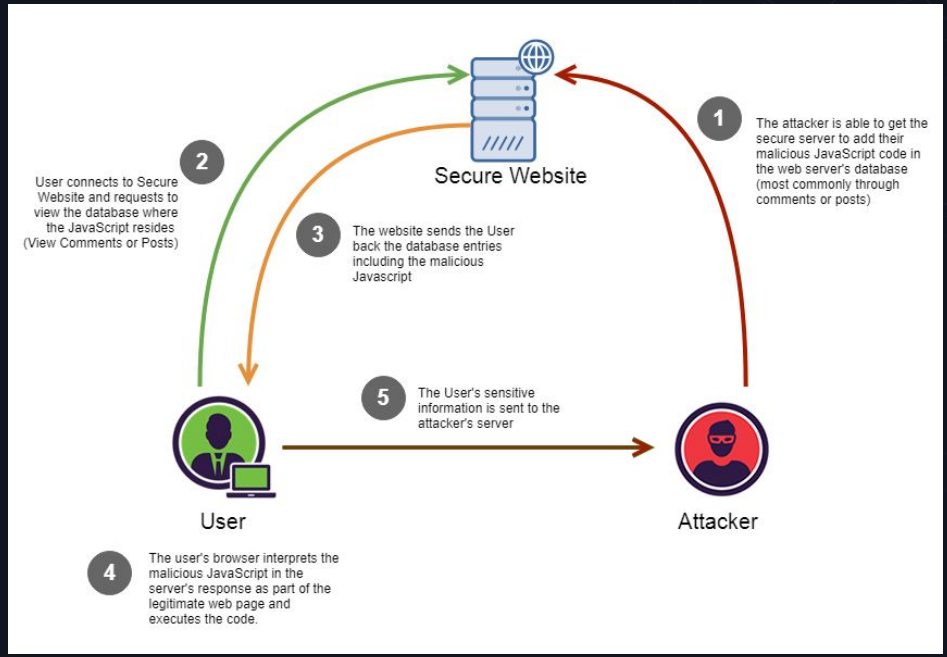
Practical work

Only modifying the server.js file, fix the security flaw.

You can install and use the following package to help you:
https://www.npmjs.com/package/escape-html

XSS injections
Persistent XSS Attacks

Persistent attacks are those where the injected script is permanently stored on the target servers, such as in a database, in a message forum, visitor log, comment field, etc. The victim then retrieves the malicious script from the server when it requests the stored information. Persistent XSS is also sometimes referred to as Stored XSS.



XSS injections
Persistent XSS Attacks - Example

```
npm install
node server.js
```

Now go there: http://localhost:3000

It should be working.

Now try to attack it.

XSS injections
Persistent XSS Attacks - Example

NEVER trust user input.

XSS injections
Persistent XSS Attacks - Example

Practical work

Only modifying the `server.js` file, fix the security flaw.

You still can install and use the following package to help you:
https://www.npmjs.com/package/escape-html

XSS injections
Conclusion

- Never trust user input
- When you need to display user input, escape it
- When you need to store user input, sanitize it
- Escaping is native in most framework, use it !
- Re-developing core features is usually a bad idea

```
# views/welcome.mustache

<html>
  <body>
    <h1>{{ title }}</h1>
    <p>{{{ message }}}</p>
  </body>
</html>
```

There should be only two brackets here