WRITING LESS INSECURE JAVASCRIPT

FOSDEM

6th February 2022

Video

rg/2022/schedule/event/writing_less_insecure_javascript

and Subtitles (./talk.srt)

```
console.log((function whoAmI(){}).name)
```

André Jaenisch

Matrix: @Ryuno-Ki:matrix.org

Twitter: @AndreJaenisch (https://twitter.com/AndreJaenisch)

Others: jaenis.ch/about (https://jaenis.ch/about/)

WHAT WILL YOU LEARN TODAY?

1. What can you do as developer?
2. What can you do as team?
3. What can you do as company?
4. What can you do as community?

MOTIVATION

MOTIVATION

“Focus on the people because everything else is an implementation detail”

MOTIVATION

“Focus on the people because everything else is an implementation detail”

Source: similar from Ship It! (https://changelog.com/shipit)

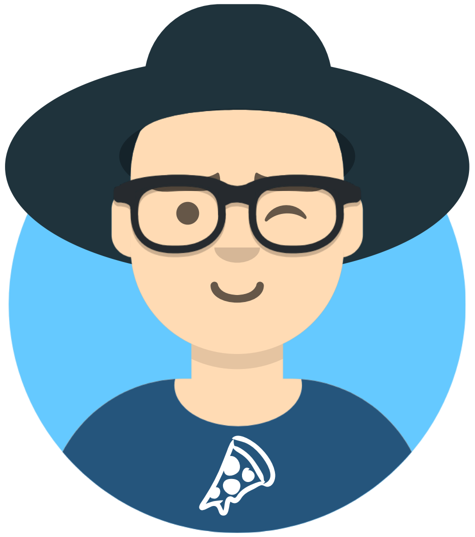
Don't try these code snippets on other people's machines. You might risk legal liability.

Don't try these code snippets on other people's machines. You might risk legal liability.

There is no 100 % security. That means not, that efforts are futile, though.

CHOOSE YOUR AVATAR

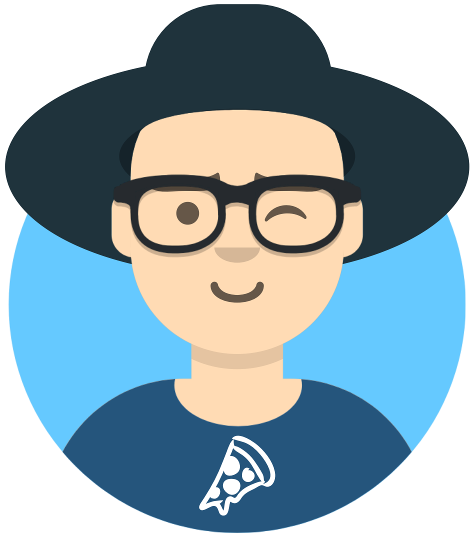
Alice



Bob

CHOOSE YOUR AVATAR

Alice



Bob

WHAT CAN YOU DO AS DEVELOPER?

It's dangerous to go alone! Take this.

If you want to see code, hit ⬇️

If you want to skip to tooling, use ➡️

ASSUMPTIONS

The presentations assumes JavaScript. You might get different results with TypeScript. However, even that is not a silver bullet.

You are expected to be at least somewhat familiar with the language. Terms like prototype chain are not unknown to you.

OWASP (OPEN WEB APPLICATION SECURITY PROJECT) TOP TEN

Read in full (https://owasp.org/Top10/)

1. Broken Access Control
2. Cryptographic Failures
3. Injection
4. Insecure Design
5. Security Misconfiguration
6. Vulnerable and Outdated Components
7. Identification and Authentication Failures
8. Software and Data Integrity Failures
9. Security Logging and Monitoring Failures
10. Server-Side Request Forgery

ATTACK VECTORS

1. Naming variables
2. Cross-site scripting (XSS)
3. Malicious package (Typo-squatting)
4. Prototype pollution
5. Reverse tabnapping
6. Directory traversal
7. Exploiting `postMessage`
8. ReDOS attacks

NAMING VARIABLES

Use `dangerously`, `raw` or `unsafe` for not validated or sanitized data. Being longer to write and defaulting to a secure state is a plus.

Back to attack vectors

CROSS-SITE SCRIPTING (XSS)

Quoting Snyk

A cross-site scripting attack occurs when the attacker tricks a legitimate web-based application or site to accept a request as originating from a trusted source.

This is done by escaping the context of the web application; the web application then delivers that data to its users along with other trusted dynamic content, without validating it. The browser unknowingly executes malicious script on the client side in order to perform actions that are otherwise typically blocked by the browser's Same Origin Policy.

Counter measures:

MALICIOUS PACKAGE (TYPO-SQUATTING)

Problem: Using typos for registering package with malicious code.

Counter measures:

PROTOTYPE POLLUTION

Prototype Pollution is a vulnerability affecting JavaScript. Prototype Pollution refers to the ability to inject properties into existing JavaScript language construct prototypes, such as objects. JavaScript allows all Object attributes to be altered, including their magical attributes such as `_proto_`*,* `constructor` *and* `prototype`*.*

An attacker manipulates these attributes to overwrite, or pollute, a JavaScript application object prototype of the base object by injecting other values. Properties on the `Object.prototype` are then inherited by all the JavaScript objects through the prototype chain.

When that happens, this leads to either denial of service by triggering JavaScript exceptions, or it tampers with the application source code to force the code path that the attacker injects, thereby leading to remote code execution.

Example code

```
function isObject(obj) {
  return ['function', 'object'].includes(typeof obj);
}
function merge(target, source) {
  for (let key in source) {
    if (isObject(target[key]) && isObject(source[key])) {
      merge(target[key], source[key]);
    } else {
      target[key] = source[key];
    }
  }
  return target;
}
function clone(target) { return merge({}, target); }
```

Example code

```
var o = {};
console.log(o.isAdmin);
// => undefined
clone({'constructor': {'prototype': {isAdmin: true}}})
console.log(o.isAdmin);
// => true
```

Inspired by Carlos Prolop' NodeJS - `__proto__` & `prototype` Pollution

(https://github.com/carlospolop/hacktricks/blob/maste

Example code

```
var o = {};
console.log(o.isAdmin);
// => undefined
clone({'constructor': {'prototype': {isAdmin: true}}})
console.log(o.isAdmin);
// => true
```

Inspired by Carlos Prolop' NodeJS - `__proto__` & `prototype` Pollution

(https://github.com/carlospolop/hacktricks/blob/maste

Example code

```
var o = {};
console.log(o.isAdmin);
// => undefined
clone({'constructor': {'prototype': {isAdmin: true}}})
console.log(o.isAdmin);
// => true
```

Inspired by Carlos Prolop' NodeJS - `__proto__` & `prototype` Pollution (https://github.com/carlospolop/hacktricks/blob/maste

Counter measures:

REVERSE TABNAPPING

Problem: Links with `target="_blank"` used to expose `window.opener`.

This allowed the target site to manipulate the source web page.

Counter measure

Use `rel="noopener noreferrer"` on links opening in a new tab.

Read About rel=noopener (https://mathiasbynens.github.io/rel-noopener/)

Back to attack vectors

DIRECTORY TRAVERSAL

Problem: Exploit HTTP to gain unauthorized access to restricted files or directories.

Main characteristic is „dot-dot-slash” (or `../`).

Example call

```
curl --path-as-is https://jaenis.ch/hobbies/../../
# yields a bad request, so don't try it
```

Counter measure

- Don't rely on user input when accessing the file system if possible.
- Normalize path information (i.e. URL-encoding) before using it. Check prefix matches directory for which user has access rights.
- Avoid requests to file system via URL.
- Don't store sensitive files on web server storage.

Read curl ootw: –path-as-is (https://daniel.haxx.se/blog/2020/07/29/curl-ootw-path-as-is/).

EXPLOITING `postMessage`

Problem: Lack of validation of origin for incoming messages allows arbitrary code execution.

Example code

```
window.postMessage({name: 'sync-ready'}, '*');
window.addEventListener('message', function (ev) {
  if (ev.data === 'page-ready'){
    // ...
  } else {
    chrome.runtime.sendMessage(ev.data, function(response){
    });
  }
}, false);
```

Example code

```
window.postMessage({name: 'sync-ready'}, '*');
window.addEventListener('message', function (ev) {
  if (ev.data === 'page-ready'){
    // ...
  } else {
    chrome.runtime.sendMessage(ev.data, function(response){
    });
  }
}, false);
```

Example code

```
window.postMessage({name: 'sync-ready'}, '*');
window.addEventListener('message', function (ev) {
  if (ev.data === 'page-ready'){
    // ...
  } else {
    chrome.runtime.sendMessage(ev.data, function(response){
    });
  }
}, false);
```

```
// Content scripts are only executed on top-level windows
var w = window.open(
  'https://example.com/according/to/web-extension',
  '_blank', 'width=100,height=100');
window.setTimeout(function () {
  w.postMessage({
    method: 'not-page-ready-but-another-allowed-method',
    data: {
      name: 'Key is expected by Web Extension.'
          + 'Inject XSS payload here if used via innerHTML',
    },
  });
  window.setTimeout(function () { w.close(); }, 0);
}, 1000);
```

Counter measures

- It is the developer’s responsibility to check the origin attribute of any messages received to ensure that they only accept messages from origins they expect.
- Be careful with Regular Expressions

`/https*:\/\/www.example.com$/i` vs.

`https://wwwXexample.com`

`/https*:\/\/www.example.com$/i` vs.

`https://wwwXexample.com`

`/https*:\/\/www\.example\.com/i` vs.

`https://www.example.com.attacker.com`

`/https*:\/\/www.example.com$/i` vs. `https://wwwXexample.com`

`/https*:\/\/www\.example\.com/i` vs. `https://www.example.com.attacker.com`

`event.origin.indexOf('https://www.examp`
`> -1` vs. `https://www.example.com.attacker`

The requirement for validating origin holds true for iFrames, Web Extensions or Web Sockets, too.

Read Exploiting xdLocalStorage (localStorage and postMessage) (https://grimhacker.com/2020/04/02/exploiting-xdlocalstorage-localstorage-and-postmessage/).

Read Wladimir Palant's Blog (https://palant.info/).

Read OWASP HTML5 Security Cheat Sheet (https://cheatsheetseries.owasp.org/cheatsheets/HTML5

REDOS (REGULAR EXPRESSION DENIAL OF SERVICE) ATTACKS

Explain video on YouTube (https://www.youtube.com/watch?v=iihd7lo6Ui8) and my remarks (https://twitter.com/AndreJaenisch/status/13900310792

1. Don't use a RegEx (Regular Expression). Sometimes using `startsWith()` or `endsWith()` are just fine. Or applying some Functional Programming. This sidesteps a whole can of worms :-)
2. Limit input size. Checking the length of a string is comparably cheap. Error early if the input is too large.

3. Pick into before RegEx. I assume that a `find()` or `indexOf()` isn't too expensive. If an e-mail string does not contain a @ your RegEx will fail anyway.
4. Anchor your RegEx. `^`, `$` and `\b` can go a long way.

5. Break it up. Similar to 1. having dedicated checks might be more readable (and thus maintainable) than one RegEx to rule them all. The syntax feels arcane anyway, so not add up on it or otherwise nobody will dare to touch it.

6. Choose your RegEx engine wisely. I only heard about it last year, but it seems that there are several engines out there. Some making guarantees. Rust's RegEx would have saved Cloudflare from CPU exhaustion in 2019 instead of the Lua one.

Back to attack vectors

TOOLING

ASSUMPTIONS

Use of ESLint

Use of VS Code (Visual Studio Code)

Use of npm or yarn as package manager

Use of Express.js or at least in communication with Back-end team for HTML generation and server configuration

Tools

- ESLint plugins
- VS Code extensions
- Package installation
- Test data
- Define Content Security Policy

ESLINT PLUGINS

- ESLint plugin anti trojan source (https://github.com/lirantal/eslint-plugin-anti-trojan-source)
- ESLint plugin for Node.js security rules (https://github.com/gkouziik/eslint-plugin-security-node)

Back to tools

VS CODE EXTENSIONS

PACKAGE INSTALLATION

USING NPM

```
# Bad
npm install
# Good
npm ci
# Better, but sometimes fail
npm ci --ignore-scripts
```

Back to tools

USING NPM

```
# Bad
npm install
# Good
npm ci
# Better, but sometimes fail
npm ci --ignore-scripts
```

USING YARN

```
# Bad
yarn
# Good
yarn install --frozen-lockfile
# Better, but sometimes fail
yarn install --frozen-lockfile --ignore-scripts
```

Back to tools

TEST DATA

- Use the Big List of Naughty Strings (https://github.com/minimaxir/big-list-of-naughty-strings) as test data for user controlled input.
- Test your forms using Bug Magnet (https://bugmagnet.org/) web extension.

Back to tools

DEFINE CONTENT SECURITY POLICY

Deploy Content Security Policy (https://snyk.io/blog/how-can-a-content-security-policy-prevent-xss-and-other-vulnerabilities/) (Laboratory (https://addons.mozilla.org/en-GB/firefox/addon/laboratory-by-mozilla/) might help)

Back to tools

WHAT CAN YOU DO AS TEAM?

Following items focus more on coordinates efforts within a single project.

- Culture
- Software architecture
- Auth
- Testing
- Paperwork

CULTURE

- Implement a no-blame-culture
- Treat incidents as process failures

Back to efforts

SOFTWARE ARCHITECTURE

- Learn about 12-Factor app (https://12factor.net/)
- Consider using Hexagonal architecture () or Domain Driven Design () to outline the boundaries of your application.
- Use immutable data structures (Immutable.js (https://immutable-js.com/), Immer (https://immerjs.github.io/immer/)).
- Familiarize yourself with AuthN (Authentication) and AuthZ (Authorisation).

- Separate customer data from app ones for stricter security measures
- Track the flow of data (monitoring as well as architecture diagram)

Back to efforts

AUTH

- Look into login with Magic Links. Alternatively use OTPs (One-Time Passwords)
- Study JWTs (JSON Web Tokens) (Read about Best Practices (https://blog.bitsrc.io/best-practices-for-using-jwt-df3788433fd3))
- Apply consistent auth strategies
- Allow for long time limit on revocable refresh tokens
- Define short limit for session tokens
- Store environment variables and secrets in a Vault
- Different privileges require different grades of security. Adapt accordingly

Back to efforts

TESTING

PAPERWORK

- Maintain a SBOM (Software Bill of Materials)
- Write ADRs (Architectural Decision Records)
- Prepare run-books in case of emergency (Get inspired by Magento incident response plan (https://github.com/talesh/response), c.f. Episode 22 of We Hack Purple podcast (https://wehackpurple.com/podcast/episode-22-with-guest-talesh-seeparsan/))
- Print run-books in case of ransomware
- Think about how to inform customers

WHAT CAN YO DO AS COMPANY?

- Security Champions
- Implement 2FA (Two-Factor Authentification) for developers and operators
- Match passwords against HIBP (Have I Been Pwned) (https://haveibeenpwned.com/)
- Check your backups regularly
- Monitoring and Intrusion Detection Systems (https://en.wikipedia.org/wiki/Intrusion_detection_sy
- Chaos engineering

- Contract penetration tester.
- Publish security audits.
- Add SLAs (Service Level Agreements) to your service p
(including Marketing).
- Offer a Bug Bounty program and outline boundaries.
- Provide an e-mail address for reporting security issues
process them.
- Understand defense in depth
(https://en.wikipedia.org/wiki/Defense_in_depth_(co
- Segment your network. Scan for open ports and close
you don't need them.

- Handle responsible disclosures (https://policymaker.disclose.io/policymaker/introduc
- Define your threat models (https://owasp.org/www-community/Threat_Modeling).
- Apply extra measures like VPN for distributed teams (S attack vector on compromised machine).
- Check your security certificates for expiration.
- Check your security certificates for revoke.

- Familiarize with ISO 27001 norm (https://www.iso.org/isoiec-27001-information-security.html) on security.
- Register additional domains to guard against domain squatting.
- Raise awareness for boss fraud mails.
- Scan for tokens in your codebases.
- Train on OWASP Juice Shop (https://owasp.org/www-project-juice-shop/).

WHAT CAN YO DO AS COMMUNITY?

- Encourage use of SECURITY.md (https://snyk.io/blog/ten-git-hub-security-best-practices/)

WHERE TO GO NOW?

1. Set up Snyk (https://snyk.io/product/snyk-code/).
2. Subscribe to the We Hack Purple podcast (https://wehackpurple.com/podcasts/).
3. Subscribe to Troy Hunt's podcast (https://www.troyhunt.com/my-weekly-updates-are-now-available-as-an-audio-podcast/).
4. Study OWASP Cheat Sheet Series (https://cheatsheetseries.owasp.org/).

IMAGE CREDITS

- Alice and Bob Learn Application Security (https://shehackspurple.ca/books/)
- Snyk logo on solid background by Snyk Press-Kit (https://snyk.io/press-kit/)
- Water drop photo by Herbert Goetsch (https://unsplash.com/@hg_photo) on Unsplash (https://unsplash.com/s/photos/water-drop)
- Super Tux! (https://flic.kr/p/rm79j) by Adam Piontek (https://www.flickr.com/photos/damek/) on Flickr (CC (https://creativecommons.org/licenses/by/2.0/))

ACHIEVEMENTS

Share (https://twitter.com/intent/tweet?
text=I%20attended%20Writing%20Less%20Insecure%20
2022-writing-less-insecure-
javascript%2F&hashtags=FOSDEM,wlij&via=AndreJaenis

THANK YOU

Writing less insecure JavaScript (https://jaenis.ch/hobbies/speaking/fosdem-2022-writing-less-insecure-javascript/) by André Jaenisch