JavaScript Security: Best Practices

UNDERSTANDING JAVASCRIPT SECURITY

Marcin Hoppe

@marcin_hoppe marcinhoppe.com

Overview

Role of JavaScript in Web security

How JavaScript code is executed

- Browsers
- Node.js

Dangerous JavaScript features

Sensitive data leak

Basic Notions of Web Security

ATTACKERS

Capable and motivated

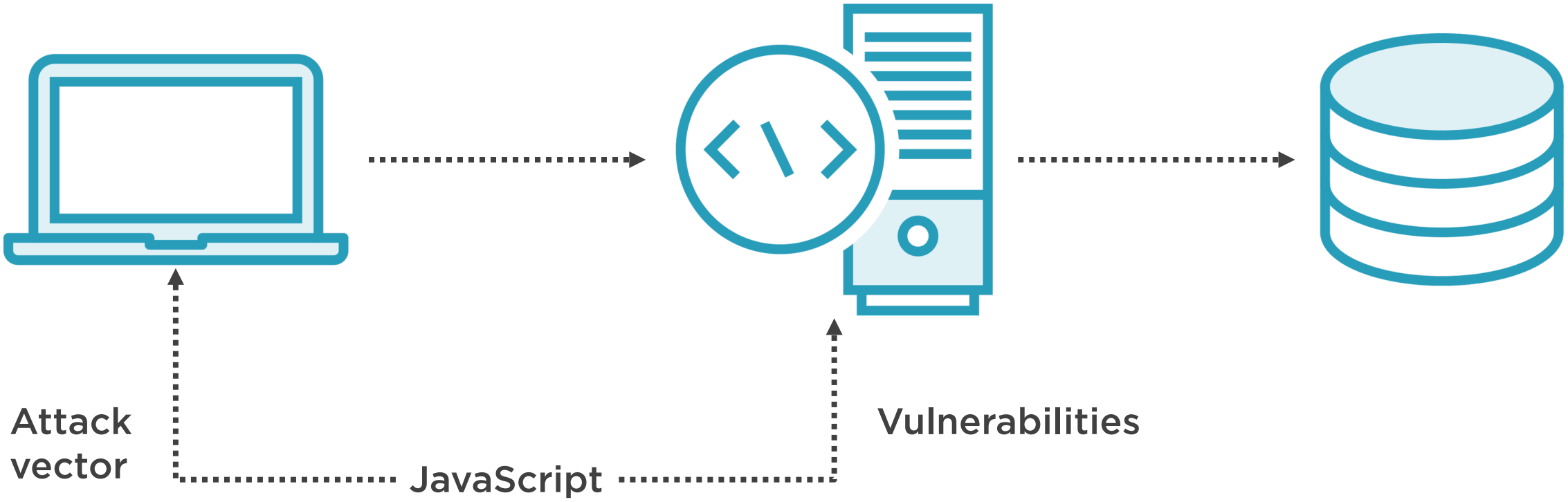
VULNERABILITIES

Flaws in code and configuration

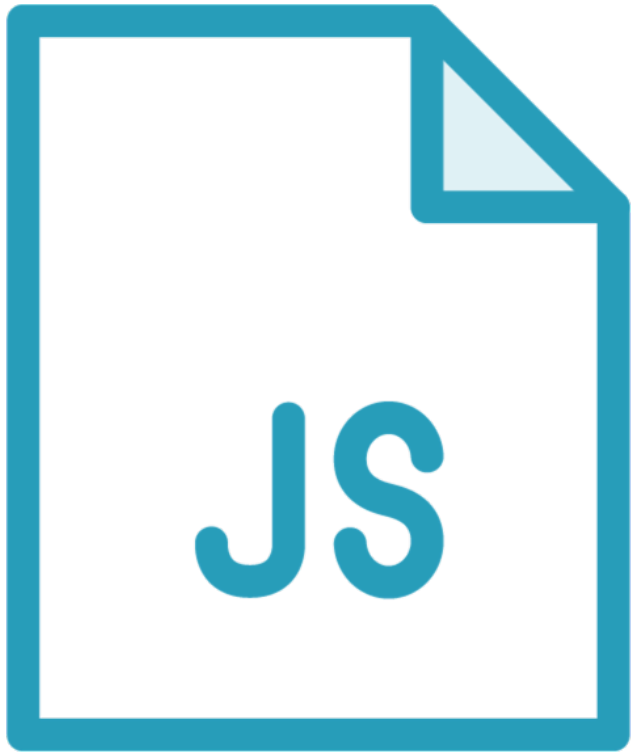
DATA BREACHES

Steal data or abuse functionality

JavaScript in Web Security



How Browsers Execute JavaScript Code



Code is downloaded

Each site gets a sandbox

Multiple security measures

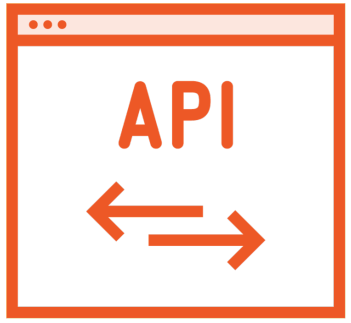
- OS process separation
- HTTPS
- Subresource Integrity (SRI)

Browser Sandbox

JavaScript code running in the browser is restricted

No local resources

No direct access to devices, files, and local network



Only browser APIs

Limited access to resources allowed by the user

Same origin only

Code and data from different sites cannot interact

Node.js vs Browser

JavaScript in the browser

Downloaded from the Web

Untrusted and highly restricted

Limited blast radius

JavaScript in Node.js

Loaded from local files

Trusted and highly privileged

May lead to server compromise

JavaScript Security Pitfalls

DYNAMIC TYPE SYSTEM

Abusing conversions and comparisons

DYNAMIC CODE EXECUTION

Interpreting untrusted data as code

PROTOTYPAL INHERITANCE

Modifying behaviour of child objects

Dynamic nature of JavaScript can lead to security bugs

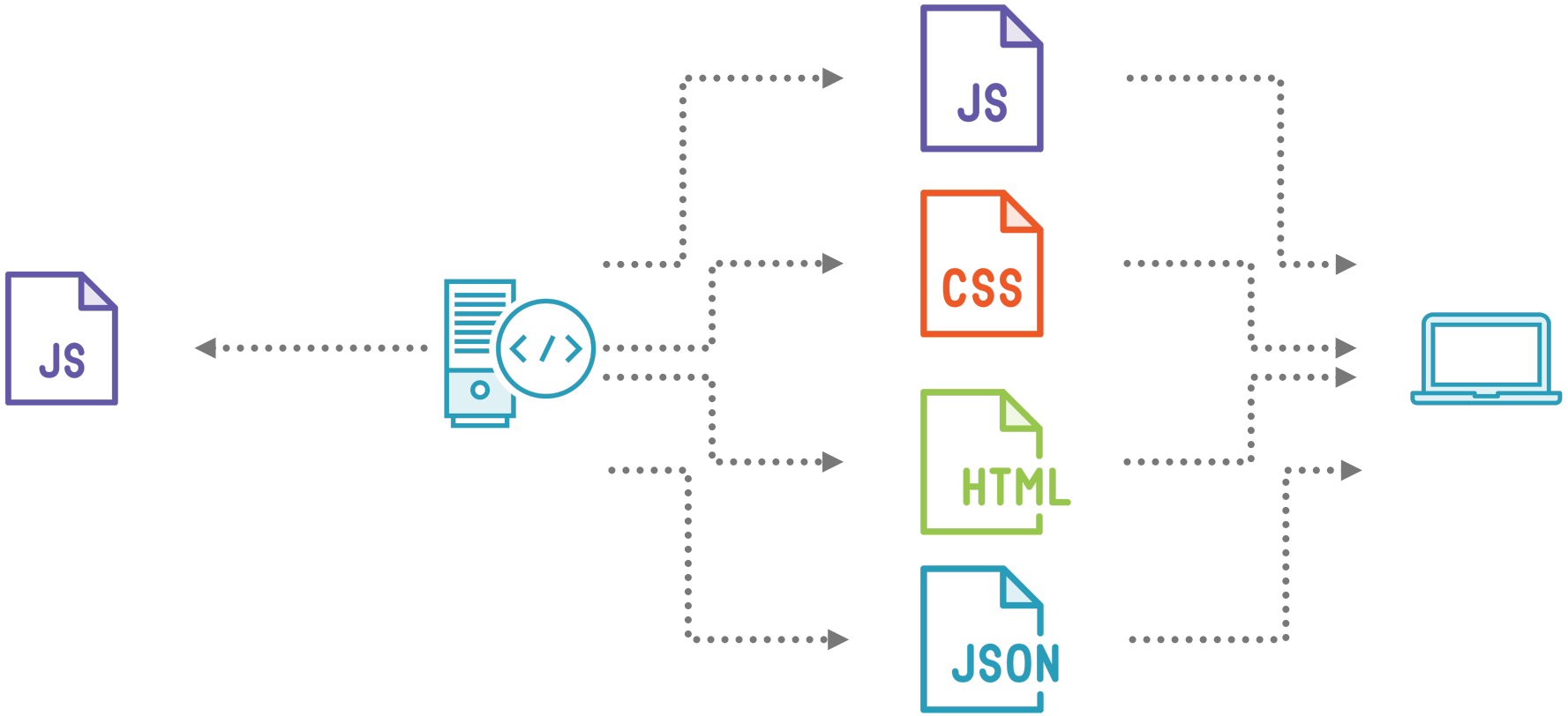
Demo

Wired Brain Coffee eCommerce

Login

User Profile Management

Wired Brain Coffee eCommerce



Dynamic Type System Pitfalls

Automatic conversions

Unexpected code may be executed

Loose comparisons

Security checks may be bypassed

Always “use strict” mode

How to Exploit the Bug?

Inspect original HTTP request

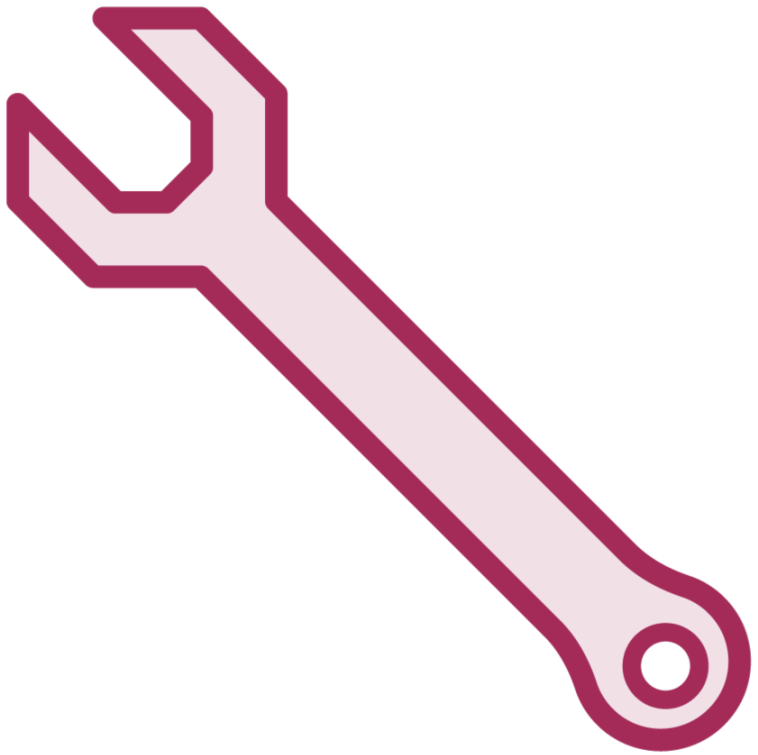
Inject malicious payload using browser development tools

Deliver it to the application

Modify data to confuse the type system

- Mixed data types
- Arrays and objects
- Missing properties

Work backwards from identified flaw in the code



Use strict mode

Do not use loose comparison (==)

- Use === instead
- Consider using `Object.is`

Verify types of untrusted data items

Summary

Dynamic nature of JavaScript code can lead to vulnerabilities

- Dynamic typing
- Dynamic code execution
- Prototypal inheritance

Security bugs in the browser may become an attack vector

Vulnerabilities in Node.js code can lead to serious data breaches