Preventing Code Injection Attacks

Marcin Hoppe

@marcin_hoppe marcinhoppe.com

Overview

Dynamic code execution

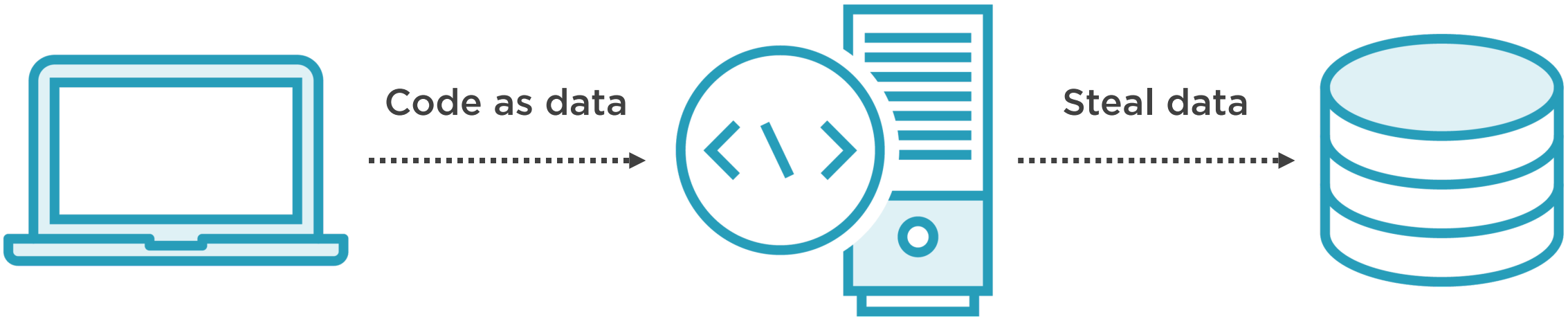
Unsafe functions

Impact of remote code execution

- Denial of service
- Server takeover

Safe coding patterns

Code Injection Attacks

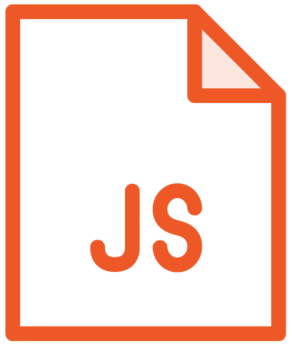


Parse data

Execute malicious code

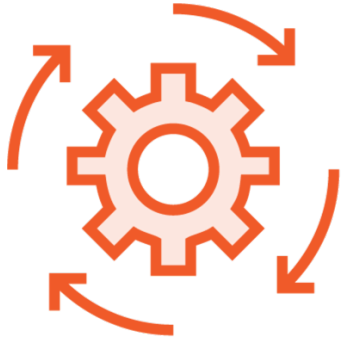
Dynamic Code Execution

JavaScript code can be loaded from Web, files, or user input



Parse

Transform source code
into abstract syntax tree



Compile

Generate bytecode
just-in-time

Execute

Run bytecode
instructions

JavaScript can generate and execute new code at runtime

Evaluate Arithmetic Expression

```
const expression = "( 1 + 1 ) * 2";  // User input

const result = eval(expression);     // Parse, compile, execute

console.log(result);                 // 4

console.log(typeof result);          // number
```

Unsafe Functions

eval	Function constructor
`eval(code)`	`f = new Function('param', code)` `f('argument')`
Direct and indirect invocation	Invoke like a function
Global and current scope	Only global scope

Unsafe Browser API

setTimeout

Execute provided code after a specified delay

setInterval

Repeatedly execute provided code with a specified delay between executions

How to Exploit the Bug?

Inspect original HTTP request

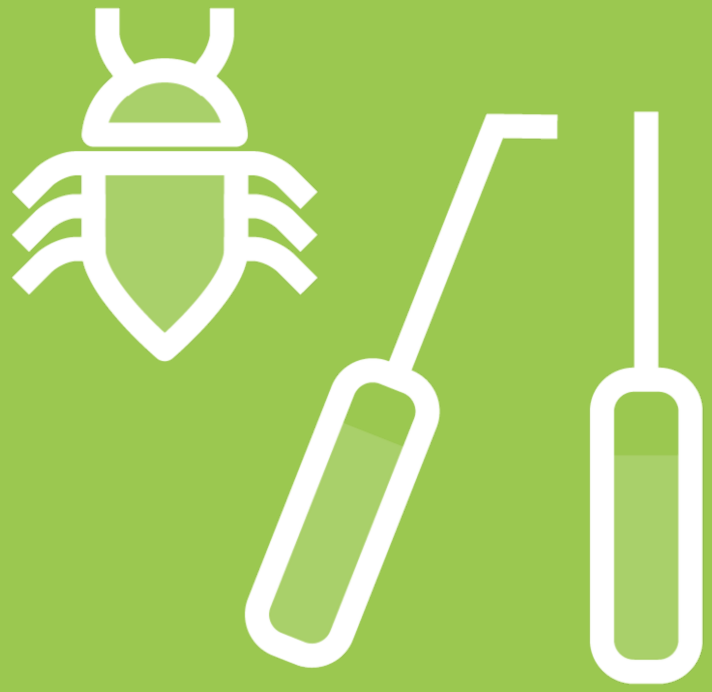
Inject malicious payload using browser development tools

Deliver it to the application

Modify data to inject the code

- Track input data
- Taint analysis
- Transformations

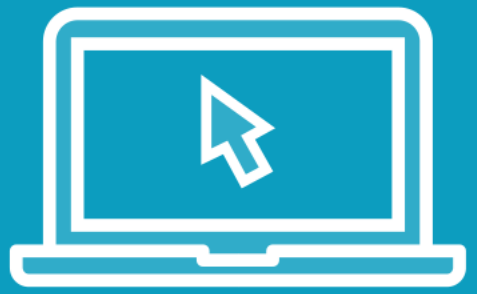
Work backwards from the code to build payload



Injection Attacks

Passing untrusted input data to any interpreter without input validation and sanitization may be exploitable!

Demo



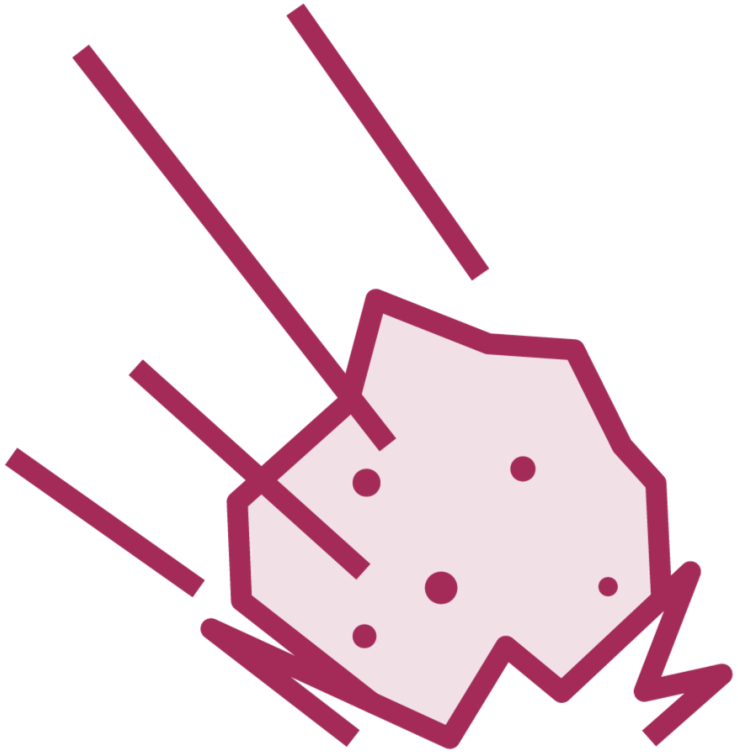
Login screen return URL

Attack

- Hijack
- Inject
- Deliver

Denial of Service (DoS)

Sensitive data leak



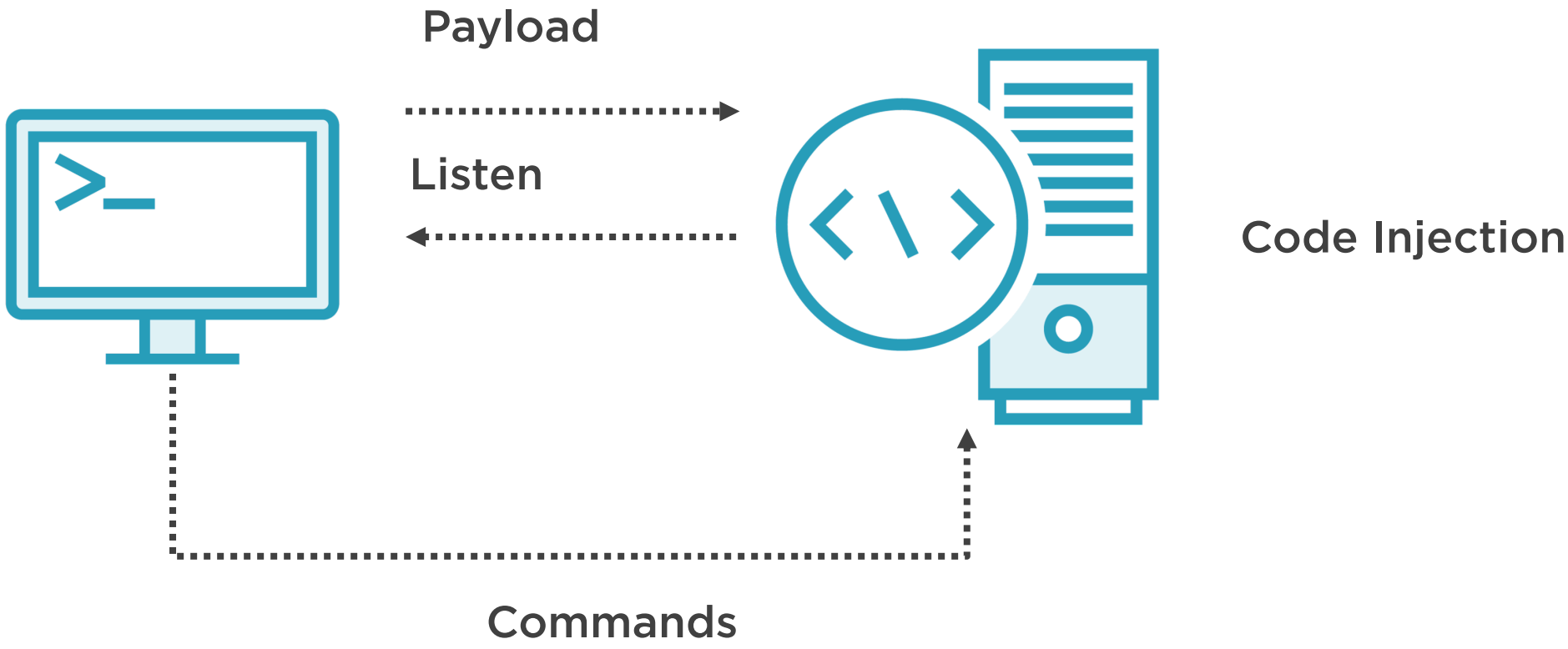
Denial of service

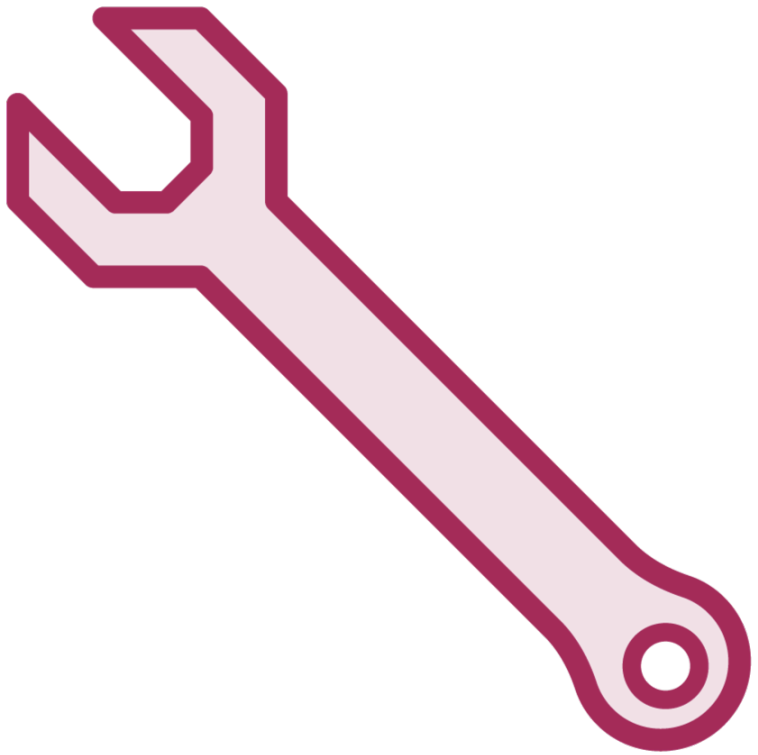
Modify application logic

- Bypass access control
- Compromise data integrity
- Steal sensitive data

Server takeover

Web Shell





Avoid unsafe functions

Validate input

- Prefer allow lists over block lists

Sanitize data passed to interpreters

Apply principle of least authority

Third-party Code

npm: the JavaScript package manager

Third-party packages may be prone to code injection

Validate input data before passing them to libraries

External packages need to be audited for use of unsafe functions

Code Injection through Math.js

```
const math = require('mathjs');      // Vulnerable library

math.eval('(1+1) * 2');              // Arithmetic expressions

math.eval('sqrt(-4)');               // Access to functions

math.eval('sqrt.constructor("return process.env")()');
```

Summary

Avoid passing untrusted data to JavaScript engine

Identify suspected code

- eval
- new Function
- setTimeout and setInterval

Audit third-party libraries for use of unsafe code