

Secrets

ABAP

Apex

C

C++

CloudFormation

COBOL

C#

CSS

Flex

Go

HTML

Java

JavaScript

Kotlin

Objective C

PHP

PL/I

PL/SQL

Python

RPG

Ruby

Scala

Swift

Terraform

Text

TypeScript

T-SQL

VB.NET

VB6

XML

TS

TypeScript static code analysis

Unique rules to find Bugs, Vulnerabilities, Security Hotspots, and Code Smells in your TYPESCRIPT code

All rules279

Vulnerability27

Bug51

Security Hotspot43

Code Smell158

Quick Fix50

Tags

Search by name...

Code Smell

Primitive types should be omitted from initialized or defaulted declarations

Code Smell

Non-null assertions should not be used

Code Smell

"undefined" should not be assigned

Code Smell

Trailing commas should not be used

Code Smell

Array constructors should not be used

Code Smell

Quotes for string literals should be used consistently

Code Smell

Statements should end with semicolons

Code Smell

Comments should not be located at the end of lines of code

Code Smell

Loops should not contain more than a single "break" or "continue" statement

Code Smell

Variable, property and parameter names should comply with a naming convention

Code Smell

Lines should not end with trailing whitespaces

Code Smell

Having a permissive Cross-Origin Resource Sharing policy is security-sensitive

Analyze your code

Security Hotspot

Minor

cwe owasp sans-top25

express.js

Having a permissive Cross-Origin Resource Sharing policy is security-sensitive. It has led in the past to the following vulnerabilities:

CVE-2018-0269

CVE-2017-14460

Same origin policy

in browsers prevents, by default and for security-reasons, a javascript frontend to perform a cross-origin HTTP request to a resource that has a different origin (domain, protocol, or port) from its own. The requested target can append additional HTTP headers in response, called CORS, that act like directives for the browser and change the access control policy / relax the same origin policy.

Ask Yourself Whether

You don't trust the origin specified, example: Access-Control-Allow-Origin: untrustedwebsite.com.

Access control policy is entirely disabled: Access-Control-Allow-Origin: \*

Your access control policy is dynamically defined by a user-controlled input like origin header.

There is a risk if you answered yes to any of those questions.

Recommended Secure Coding Practices

The Access-Control-Allow-Origin header should be set only for a trusted origin and for specific resources.

Allow only selected, trusted domains in the Access-Control-Allow-Origin header. Prefer whitelisting domains over blacklisting or allowing any domain (do not use \* wildcard nor blindly return the Origin header content without any checks).

Sensitive Code Example

nodejs http built-in module:

```
const http = require('http');
const srv = http.createServer((req, res) => {
  res.writeHead(200, { 'Access-Control-Allow-Origin': '*' });
  res.end('ok');
});
srv.listen(3000);
```







Express.js framework with cors middleware:

```
const cors = require('cors');

let app1 = express();
app1.use(cors()); // Sensitive: by default origin is set to
```

https://rules.sonarsource.com/typescript/RSPEC-5122

1/2

<div>Files should contain an empty newline at the end</div> <div> Code Smell</div>	<pre>let corsOptions = {   origin: '*' // Sensitive };  let app2 = express(); app2.use(cors(corsOptions));</pre>
<div>An open curly brace should be located at the end of a line</div> <div> Code Smell</div>	User-controlled origin:
<div>Tabulation characters should not be used</div> <div> Code Smell</div>	<pre>function (req, res) {   const origin = req.header('Origin');   res.setHeader('Access-Control-Allow-Origin', origin); // S };</pre>
<div>Function and method names should comply with a naming convention</div> <div> Code Smell</div>	<div>Compliant Solution</div> <div>nodejs <a href="#">http</a> built-in module:</div> <pre>const http = require('http'); const srv = http.createServer((req, res) =&gt; {   res.writeHead(200, { 'Access-Control-Allow-Origin': 'trust   res.end('ok'); }); srv.listen(3000);</pre> <div>Express.js framework with <a href="#">cors middleware</a>:</div> <pre>const cors = require('cors');  let corsOptions = {   origin: 'trustedwebsite.com' // Compliant };  let app = express(); app.use(cors(corsOptions));</pre> User-controlled origin validated with an allow-list: <pre>function (req, res) {   const origin = req.header('Origin');    if (trustedOrigins.indexOf(origin) &gt;= 0) {     res.setHeader('Access-Control-Allow-Origin', origin);   } };</pre> <div>See</div> <ul style="list-style-type: none"><li>• <a href="#">OWASP Top 10 2021 Category A5</a> - Security Misconfiguration</li><li>• <a href="#">OWASP Top 10 2021 Category A7</a> - Identification and Authentication Failures</li><li>• <a href="#">developer.mozilla.org</a> - CORS</li><li>• <a href="#">developer.mozilla.org</a> - Same origin policy</li><li>• <a href="#">OWASP Top 10 2017 Category A6</a> - Security Misconfiguration</li><li>• <a href="#">OWASP HTML5 Security Cheat Sheet</a> - Cross Origin Resource Sharing</li><li>• <a href="#">MITRE, CWE-346</a> - Origin Validation Error</li><li>• <a href="#">MITRE, CWE-942</a> - Overly Permissive Cross-domain Whitelist</li><li>• <a href="#">SANS Top 25</a> - Porous Defenses</li></ul> <div>Available In:</div> <div> </div>