




 **sonar** RULES


 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



# JavaScript static code analysis

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

**All rules** 285

 Vulnerability 29


 Bug 62













 Security Hotspot 43

 Code Smell 151

 Quick Fix 41

Tags ▾

Search by name... 

Tests should include assertions
 Code Smell
"future reserved words" should not be used as identifiers
 Code Smell
Octal values should not be used
 Code Smell
Switch cases should end with an unconditional "break" statement
 Code Smell
"switch" statements should not contain non-case labels
 Code Smell
A new session should be created during user authentication
 Vulnerability
JWT should be signed and verified with strong cipher algorithms
 Vulnerability
Cipher algorithms should be robust
 Vulnerability
Encryption algorithms should be used with secure mode and padding scheme
 Vulnerability
Server hostnames should be verified during SSL/TLS connections
 Vulnerability
Server certificates should be verified during SSL/TLS connections
 Vulnerability
Cryptographic keys should be robust
 Vulnerability

## I/O function calls should not be vulnerable to path injection attacks

Analyze your code

 Vulnerability  Blocker  injection cwe owasp sans-top25

User-provided data, such as URL parameters, POST data payloads, or cookies, should always be considered untrusted and tainted. Constructing file system paths directly from tainted data could enable an attacker to inject specially crafted values, such as `'../'`, that change the initial path and, when accessed, resolve to a path on the filesystem where the user should normally not have access.

A successful attack might give an attacker the ability to read, modify, or delete sensitive information from the file system and sometimes even execute arbitrary operating system commands. This is often referred to as a "path traversal" or "directory traversal" attack.

The mitigation strategy should be based on the whitelisting of allowed paths or characters.

### Noncompliant Code Example

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

function (req, res) {
  const reqPath = __dirname + req.query.filename; // user-co

  let data = fs.readFileSync(reqPath, { encoding: 'utf8', fl
}
```

### Compliant Solution








```
const fs = require('fs');
const pathmodule = require('path');

function (req, res) {
  const reqPath = __dirname + req.query.filename; // user-co
  const resolvedPath = pathmodule.resolve(reqPath); // resol

  if (resolvedPath.startsWith(__dirname + '/uploads')) { //
    let data = fs.readFileSync(resolvedPath, { encoding: 'ut
  }
}
```

### See

- OWASP Top 10 2021 Category A1 - Broken Access Control
- OWASP Top 10 2021 Category A3 - Injection
- OWASP Top 10 2017 Category A1 - Injection
- OWASP Top 10 2017 Category A5 - Broken Access Control
- MITRE, CWE-20 - Improper Input Validation
- MITRE, CWE-22 - Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')
- MITRE, CWE-99 - Improper Control of Resource Identifiers ('Resource Injection')

<p><b>Weak SSL/TLS protocols should not be used</b></p> <p> Vulnerability</p>	<ul style="list-style-type: none"><li>• <a href="#">MITRE, CWE-641</a> - Improper Restriction of Names for Files and Other Resources</li><li>• <a href="#">SANS Top 25</a> - Risky Resource Management</li></ul> <p>Available In:</p> <div>  </div>
<p><b>Origins should be verified during cross-origin communications</b></p> <p> Vulnerability</p>	
<p><b>Regular expressions should not be vulnerable to Denial of Service attacks</b></p> <p> Vulnerability</p>	
<p><b>File uploads should be restricted</b></p> <p> Vulnerability</p>	
<p><b>Function calls should not pass extra</b></p>	

© 2008-2022 SonarSource S.A., Switzerland. All content is copyright protected. SONAR, SONARSOURCE, SONARLINT, SONARQUBE and SONARCLOUD are trademarks of SonarSource S.A. All other trademarks and copyrights are the property of their respective owners. All rights are expressly reserved.

[Privacy Policy](#)