




 **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... 

	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
	Weak SSL/TLS protocols should not be used
	Vulnerability
	Origins should be verified during cross-origin communications
	Vulnerability

OS commands should not be vulnerable to command injection attacks

Analyze your code

 Vulnerability  Blocker  injection cwe owaspsans-top25

Applications that allow execution of operating system commands from user-controlled data should control the command to execute, otherwise an attacker can inject arbitrary commands that will compromise the underlying operating system.

The mitigation strategy can be based on a list of authorized and safe commands to execute and when a shell is spawned to sanitize shell meta-characters. Keep in mind that when a single argument to the command is user-controlled and shell-metachars are sanitized, it can still lead to vulnerabilities if the attacker can inject a dangerous option supported by the command, such as `-exec` available with `find`, in that case, mark end of option processing on the command line using `--` (double-dash) or restrict options to only trusted values.

Noncompliant Code Example

When using functions like `execSync` a shell is spawn and therefore shell metachars are available and allow attackers to execute additional arbitrary commands:

```
const cp = require('child_process');

function (req, res) {
  const cmd = 'ls '+req.query.arg;

  const out = cp.execSync(cmd); // Noncompliant: example of
}
```

Compliant Solution







Use functions like `execFileSync` with a defined command and user-controlled arguments put in a array:

```
const cp = require('child_process');

function (req, res) {
  const out = cp.execFileSync("ls", [req.query.arg]); // Compliant
}
```

- See
- OWASP Top 10 2021 Category A3 - Injection
 - OWASP OS Command Injection Defense [Cheat Sheet](#)
 - OWASP Top 10 2017 Category A1 - Injection
 - MITRE, [CWE-20](#) - Improper Input Validation
 - MITRE, [CWE-78](#) - Improper Neutralization of Special Elements used in an OS Command
 - SANS Top 25 - Insecure Interaction Between Components

Available In:  Developer

<div>Regular expressions should not be vulnerable to Denial of Service attacks</div> <div> Vulnerability</div>	<div>sonarcloud  sonarqube  Developer Edition</div> <div>© 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</div>
<div>File uploads should be restricted</div> <div> Vulnerability</div>	
<div>Function calls should not pass extra arguments</div> <div> Bug</div>	
<div>Regular expressions should be syntactically valid</div> <div> Bug</div>	
<div>Getters and setters should access the</div>	