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

TypeScript static code analysis

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

All rules 279 | Vulnerability 27 | Bug 51 | Security Hotspot 43 | Code Smell 158 | Quick Fix 50

Tags | Search by name...

Using hardcoded IP addresses is security-sensitive
Security Hotspot

Regular expression quantifiers and character classes should be used concisely
Code Smell

Regular expression literals should be used when possible
Code Smell

"await" should not be used redundantly
Code Smell

Redundant casts and non-null assertions should be avoided
Code Smell

Type aliases should be used
Code Smell

Type guards should be used
Code Smell

"module" should not be used
Code Smell

"for of" should be used with Iterables
Code Smell

Imports from the same modules should be merged
Code Smell

Jump statements should not be redundant
Code Smell

Default export names and file names should match
Code Smell

Constructing arguments of system commands from user input is security-sensitive

Analyze your code

Security Hotspot | Major | injection cwe owasp sans-top25

Constructing arguments of system commands from user input is security-sensitive. It has led in the past to the following vulnerabilities:

- CVE-2016-9920
- CVE-2021-29472

Arguments of system commands are processed by the executed program. The arguments are usually used to configure and influence the behavior of the programs. Control over a single argument might be enough for an attacker to trigger dangerous features like executing arbitrary commands or writing files into specific directories.

Ask Yourself Whether

- Malicious arguments can result in undesired behavior in the executed command.
- Passing user input to a system command is not necessary.

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

Recommended Secure Coding Practices

- Avoid constructing system commands from user input when possible.
- Ensure that no risky arguments can be injected for the given program, e.g., type-cast the argument to an integer.
- Use a more secure interface to communicate with other programs, e.g., the standard input stream (stdin).

Sensitive Code Example

Arguments like `-delete` or `-exec` for the `find` command can alter the expected behavior and result in vulnerabilities:

```
const { spawn } = require("child_process");
const input = req.query.input;
const proc = spawn("/usr/bin/find", [input]); // Sensitive
```

Compliant Solution

Use an allow-list to restrict the arguments to trusted values:

```
const { spawn } = require("child_process");
const input = req.query.input;
if (allowed.includes(input)) {
  const proc = spawn("/usr/bin/find", [input]);
}
```

See

The global "this" object should not be used

Code Smell

"catch" clauses should do more than rethrow

Code Smell

Boolean checks should not be inverted

Code Smell

Deprecated APIs should not be used

Code Smell

Wrapper objects should not be used

- OWASP Top 10 2021 Category A3 - Injection
- OWASP Top 10 2017 Category A1 - Injection
- MITRE, CWE-88 - Argument Injection or Modification
- SANS Top 25 - Insecure Interaction Between Components
- CVE-2021-29472 - PHP Supply Chain Attack on Composer

Available In:

sonarcloud | sonarqube Developer Edition