




 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 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

Code Smell

Non-null assertions should not be used

Code Smell

Code Smell

"undefined" should not be assigned

Code Smell

Code Smell

Trailing commas should not be used

Code Smell

Code Smell

Array constructors should not be used

Code Smell

Code Smell

Quotes for string literals should be used consistently

Code Smell

Code Smell

Statements should end with semicolons

Code Smell

Code Smell

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

Code Smell

Code Smell

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

Code Smell

Code Smell

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

Code Smell

Code Smell

Lines should not end with trailing whitespaces

Code Smell

Allowing browsers to sniff MIME types is security-sensitive

Security HotspotMinor🔍owasp express.js

MIME confusion attacks occur when an attacker successfully tricks a web-browser to interpret a resource as a different type than the one expected. To correctly interpret a resource (script, image, stylesheet ...) web browsers look for the **Content-Type header** defined in the HTTP response received from the server, but often this header is not set or is set with an incorrect value. To avoid content-type mismatch and to provide the best user experience, web browsers try to deduce the right content-type, generally by inspecting the content of the resources (the first bytes). This "guess mechanism" is called **MIME type sniffing**.

Attackers can take advantage of this feature when a website ("example.com" here) allows to upload arbitrary files. In that case, an attacker can upload a malicious image *fakeimage.png* (containing malicious JavaScript code or a **polyglot content** file) such as:

```
<script>alert(document.cookie)</script>
```

When the victim will visit the website showing the uploaded image, the malicious script embedded into the image will be executed by web browsers performing MIME type sniffing.

Ask Yourself Whether

- Content-Type** header is not systematically set for all resources.
- Content of resources can be controlled by users.

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

Recommended Secure Coding Practices

Implement **X-Content-Type-Options** header with *nosniff* value (the only existing value for this header) which is supported by all modern browsers and will prevent browsers from performing MIME type sniffing, so that in case of Content-Type header mismatch, the resource is not interpreted. For example within a `<script>` object context, JavaScript MIME types are expected (like *application/javascript*) in the Content-Type header.

Sensitive Code Example

In Express.js application the code is sensitive if, when using **helmet**, the `noSniff` middleware is disabled:





```
const express = require('express');
const helmet = require('helmet');

let app = express();

app.use(
  helmet({
    noSniff: false, // Sensitive
  })
);
```

https://rules.sonarsource.com/typescript/RSPEC-5734

1/2

Files should contain an empty newline at the end
 Code Smell
An open curly brace should be located at the end of a line
 Code Smell
Tabulation characters should not be used
 Code Smell
Function and method names should comply with a naming convention
 Code Smell

Compliant Solution

When using `helmet` in an Express.js application, the `noSniff` middleware should be enabled (it is also done by default):

```
const express = require('express');
const helmet= require('helmet');

let app = express();

app.use(helmet.noSniff());
```

See

- [OWASP Top 10 2021 Category A5](#) - Security Misconfiguration
- [OWASP Top 10 2017 Category A6](#) - Security Misconfiguration
- [developer.mozilla.org](#) - X-Content-Type-Options
- [blog.mozilla.org](#) - Mitigating MIME Confusion Attacks in Firefox

Available In:

