




 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... 🔍

 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

Encrypting data is security-sensitive

Analyze your code

Security Hotspot

Critical

Encrypting data is security-sensitive. It has led in the past to the following vulnerabilities:

- [CVE-2017-7902](#)
- [CVE-2006-1378](#)
- [CVE-2003-1376](#)

Proper encryption requires both the encryption algorithm and the key to be strong. Obviously the private key needs to remain secret and be renewed regularly. However these are not the only means to defeat or weaken an encryption.

This rule flags function calls that initiate encryption/decryption.

Ask Yourself Whether

- the private key might not be random, strong enough or the same key is reused for a long long time.
- the private key might be compromised. It can happen when it is stored in an unsafe place or when it was transferred in an unsafe manner.
- the key exchange is made without properly authenticating the receiver.
- the encryption algorithm is not strong enough for the level of protection required. Note that encryption algorithms strength decreases as time passes.
- the chosen encryption library is deemed unsafe.
- a nonce is used, and the same value is reused multiple times, or the nonce is not random.
- the RSA algorithm is used, and it does not incorporate an Optimal Asymmetric Encryption Padding (OAEP), which might weaken the encryption.
- the CBC (Cypher Block Chaining) algorithm is used for encryption, and it's IV (Initialization Vector) is not generated using a secure random algorithm, or it is reused.
- the Advanced Encryption Standard (AES) encryption algorithm is used with an unsecure mode. See the recommended practices for more information.





You are at risk if you answered yes to any of those questions.

Recommended Secure Coding Practices

- Generate encryption keys using secure random algorithms.
- When generating cryptographic keys (or key pairs), it is important to use a key length that provides enough entropy against brute-force attacks. For the Blowfish algorithm the key should be at least 128 bits long, while for the RSA algorithm it should be at least 2048 bits long.
- Regenerate the keys regularly.
- Always store the keys in a safe location and transfer them only over safe channels.
- If there is an exchange of cryptographic keys, check first the identity of the receiver.
- Only use strong encryption algorithms. Check regularly that the algorithm is still deemed secure. It is also imperative that they are implemented correctly. Use only encryption libraries which are deemed secure. Do not define your own encryption algorithms as they will most probably have flaws.
- When a nonce is used, generate it randomly every time.

https://rules.sonarsource.com/typescript/RSPEC-4787

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

- When using the RSA algorithm, incorporate an Optimal Asymmetric Encryption Padding (OAEP).
- When CBC is used for encryption, the IV must be random and unpredictable. Otherwise it exposes the encrypted value to crypto-analysis attacks like "Chosen-Plaintext Attacks". Thus a secure random algorithm should be used. An IV value should be associated to one and only one encryption cycle, because the IV's purpose is to ensure that the same plaintext encrypted twice will yield two different ciphertexts.
- The Advanced Encryption Standard (AES) encryption algorithm can be used with various modes. Galois/Counter Mode (GCM) with no padding should be preferred to the following combinations which are not secured:
 - Electronic Codebook (ECB) mode: Under a given key, any given plaintext block always gets encrypted to the same ciphertext block. Thus, it does not hide data patterns well. In some senses, it doesn't provide serious message confidentiality, and it is not recommended for use in cryptographic protocols at all.
 - Cipher Block Chaining (CBC) with PKCS#5 padding (or PKCS#7) is susceptible to padding oracle attacks.

Sensitive Code Example

```
// === Client side ===
crypto.subtle.encrypt(algo, key, plainData); // Sensitive
crypto.subtle.decrypt(algo, key, encData); // Sensitive

// === Server side ===
const crypto = require("crypto");
const cipher = crypto.createCipher(algo, key); // Sensitive
const cipheriv = crypto.createCipheriv(algo, key, iv); // Sensitive
const decipher = crypto.createDecipher(algo, key); // Sensitive
const decipheriv = crypto.createDecipheriv(algo, key, iv); // Sensitive
const pubEnc = crypto.publicEncrypt(key, buf); // Sensitive
const privDec = crypto.privateDecrypt({ key: key, passphrase: passphrase }); // Sensitive
const privEnc = crypto.privateEncrypt({ key: key, passphrase: passphrase }); // Sensitive
const pubDec = crypto.publicDecrypt(key, privEnc); // Sensitive
```

See

- [OWASP Top 10 2017 Category A3](#) - Sensitive Data Exposure
- [OWASP Top 10 2017 Category A6](#) - Security Misconfiguration
- [MITRE, CWE-321](#) - Use of Hard-coded Cryptographic Key
- [MITRE, CWE-322](#) - Key Exchange without Entity Authentication
- [MITRE, CWE-323](#) - Reusing a Nonce, Key Pair in Encryption
- [MITRE, CWE-324](#) - Use of a Key Past its Expiration Date
- [MITRE, CWE-325](#) - Missing Required Cryptographic Step
- [MITRE, CWE-326](#) - Inadequate Encryption Strength
- [MITRE, CWE-327](#) - Use of a Broken or Risky Cryptographic Algorithm
- [SANS Top 25](#) - Porous Defenses

Deprecated

This rule is deprecated; use {rule:javascript:S4426}, {rule:javascript:S5542}, {rule:javascript:S5547} instead.

Available In:
 