SQL Injection

Injection is #1 on the 2010 OWASP Top Ten web security risks. SQL injection is when a user is able to manipulate a value which is used unsafely inside a SQL query. This can lead to data leaks, data loss, elevation of privilege, and other unpleasant outcomes.

Brakeman focuses on ActiveRecord methods dealing with building SQL statements.

A basic (Rails 2.x) example looks like this:

```
User.first(:conditions => "username = '#{params[:username]}'")
```

Brakeman would produce a warning like this:

```
Possible SQL injection near line 30: User.first(:conditions => ("username = '#{params[:username]}'"))
```

The safe way to do this query is to use a parameterized query:

```
User.first(:conditions => ["username = ?", params[:username]])
```

Brakeman also understands the new Rails 3.x way of doing things (and local variables and concatentation):

```
username = params[:user][:name].downcase
password = params[:user][:password]

User.first.where("username = '" + username + "' AND password = '" + password + "'")
```

This results in this kind of warning:

```
Possible SQL injection near line 37:
User.first.where(((("username = '" + params[:user][:name].downcase) + "' AND password = '") + params[:user][:password]) + "'"))
```

See the Ruby Security Guide for more information and Rails-SQLi.org for many examples of SQL injection in Rails.

MultipleVariableDeclarations

Description

Checks that each variable declaration is in its own statement and on its own line.

Rationale: the Java code conventions chapter 6.1 recommends that declarations should be one per line/statement.

Examples

Example:

```
public class Test {
  public void myTest() {
    int mid;
    int high;
    // ...

    int lower, higher; // violation
    // ...

    int value,
        index; // violation
    // ...

    int place = mid, number = high;  // violation
  }
}
```

postfixOperator - CWE 398

L Link

C Code

Description

Prefix ++/-- operators should be preferred for non-primitive types. Pre-increment/decrement can be more efficient than post-increment/decrement. Post-increment/decrement usually involves keeping a copy of the previous value around and adds a little extra code.

Example cpp file

C

```
void main(){
for(vector<int>::iterator iter=vector_database.begin(); vector_database!=veci.end(); iter++)
        if( *iter == 10)
                vector_database.erase(iter);
```

no-nonoctal-decimal-escape

Disallow `\8` and `\9` escape sequences in string literals

Although not being specified in the language until ECMAScript 2021, `\8` and `\9` escape sequences in string literals were allowed in most JavaScript engines, and treated as "useless" escapes:

```
"\8" === "8"; // true
"\9" === "9"; // true
```

Since ECMAScript 2021, these escape sequences are specified as **non-octal decimal escape sequences (https://tc39.es/ecma262/#prod-annexB-NonOctalDecimalEscapeSequence)**, retaining the same behavior.

Nevertheless, the ECMAScript specification treats `\8` and `\9` in string literals as a legacy feature. This syntax is optional if the ECMAScript host is not a web browser. Browsers still have to support it, but only in non-strict mode.

Regardless of your targeted environment, these escape sequences shouldn't be used when writing new code.

Rule Details

This rule disallows `\8` and `\9` escape sequences in string literals.

Examples of **incorrect** code for this rule:

```
/*eslint no-nonoctal-decimal-escape: "error"*/

"\8";

"\9";

var foo = "w\8less";

var bar = "December 1\9";

var baz = "Don't use \8 and \9 escapes.";

var quux = "\0\8";
```

Examples of **correct** code for this rule:

```
/*eslint no-nonoctal-decimal-escape: "error"*/

"8";

"9";

var foo = "w8less";

var bar = "December 19";

var baz = "Don't use \\8 and \\9 escapes.";

var quux = "\0\u0038";
```

Further Reading

ECMAScript® 2023 Language Specification

tc39.es

Function is too complex (C901)

Functions that are deemed too complex are functions that have too much branching logic. Branching logic includes `if` / `elif` / `else` and `for` / `while` loops.

Anti-pattern

The following example has a complexity score of 5, because there are five potential branches.

```
def post_comment(self):
    if self.success:
        comment = 'Build succeeded'
    elif self.warning:
        comment = 'Build had issues'
    elif self.failed:
        comment = 'Build failed'

    if self.success:
        self.post(comment, type='success')
    else:
        self.post(comment, type='error')
```

Best practice

To reduce the complexity of a function you should make the function do less. In the example above, the function actually does two things: formats a comment and posts the comment. Let's split that up into two specific functions that have only one task each.

```
def get_comment(self):
    comments = {
        'success': 'Build succeeded',
        'warning': 'Build had issues',
        'failed': 'Build failed'
    }
    return comments[self.type]

def post_comment(self, comment):
    self.post(comment, type=self.type)
```

These two functions now have a complexity of 1.

Additional links

- https://en.wikipedia.org/wiki/Cyclomatic_complexity

CompareWithEmptyStringEfficientlyRule

This rule will fire if a string is compared to **”“** or **String.Empty**. Instead use a **String.Length** test which should be a bit faster. Another possibility (with .NET 2.0) is to use the static **String.IsNullOrEmpty** method. **String.IsNullOrEmpty**.

Bad example:

```
public void SimpleMethod (string myString)
{
    if (myString.Equals (String.Empty)) {
    }
}
```

Good example:

```
public void SimpleMethod (string myString)
{
    if (myString.Length == 0) {
    }
}
```

ForLoopShouldBeWhileLoop ¶

Since: 0.6

Name: for loop should be while loop

Under certain circumstances, some `for` loops can be simplified to `while` loops to make code more concise.

This rule is defined by the following class: oclint-rules/rules/basic/ForLoopShouldBeWhileLoopRule.cpp (https://github.com/oclint/oclint/blob/master/oclint-rules/rules/basic/ForLoopShouldBeWhileLoopRule.cpp)

Example:

```
void example(int a)
{
    for (; a < 100;)
    {
        foo(a);
    }
}
```

v: stable

Rule no_short_bool_cast

Short cast `bool` using double exclamation mark should not be used.

L Link

C Code

Examples

Example #1

```
--- Original
+++ New
 <?php
-$a = !!$b;
+$a = (bool)$b;
```

consider-using-generator / R1728

Description:

If your container can be large using a generator will bring better performance.

Problematic code:

```
list([0 for y in list(range(10))])  # [consider-using-generator]
tuple([0 for y in list(range(10))])  # [consider-using-generator]
sum([y**2 for y in list(range(10))])  # [consider-using-generator]
max([y**2 for y in list(range(10))])  # [consider-using-generator]
min([y**2 for y in list(range(10))])  # [consider-using-generator]
```

Correct code:

```
list(0 for y in list(range(10)))
tuple(0 for y in list(range(10)))
sum(y**2 for y in list(range(10)))
max(y**2 for y in list(range(10)))
min(y**2 for y in list(range(10)))
```

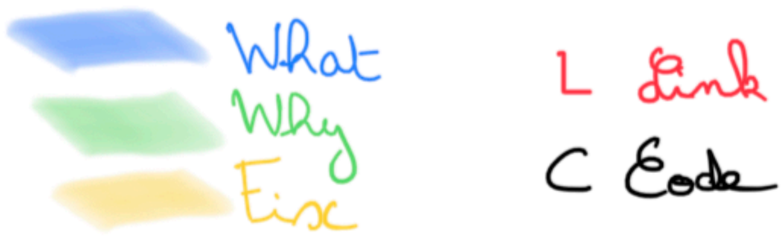
Additional details:

Removing `[]` inside calls that can use containers or generators should be considered for performance reasons since a generator will have an upfront cost to pay. The performance will be better if you are working with long lists or sets.

For `max`, `min` and `sum` using a generator is also recommended by pep289.

Related links:

- PEP 289
- Benchmark and discussion for any/all/list/tuple
- Benchmark and discussion for sum/max/min



csharp.lang.correctness.double.double-epsilon-equality.correctness-double-epsilon-equality

Double.Epsilon is defined by .NET as the smallest value that can be added to or subtracted from a zero-value Double. It is unsuitable for equality comparisons of non-zero Double values. Furthermore, the value of Double.Epsilon is framework and processor architecture dependent. Wherever possible, developers should prefer the framework Equals() method over custom equality implementations.

by r2c

TEST CODE

```
using System;

public class Example
{
   static bool IsApproximatelyEqual(double value1, double value2, double epsilon)
   {
      // If they are equal anyway, just return True.
      if (value1.Equals(value2))
         return true;

      // Handle zero to avoid division by zero
      double divisor = Math.Max(value1, value2);
      if (divisor.Equals(0))
         divisor = Math.Min(value1, value2);
      //ruleid: correctness-double-epsilon-equality
      return Math.Abs((value1 - value2) / divisor) <= Double.Epsilon;
   }

   static bool lazyEqualLeftCompare(double v1, double v2){
       //ruleid: correctness-double-epsilon-equality
       return Math.Abs(v1 - v2) <= Double.Epsilon;
   }
}

```

[0] Source for rule

[1] https://docs.microsoft.com/en-us/dotnet/api/system.double?view=net-6.0#testing-for-equality

[2] https://docs.microsoft.com/en-us/dotnet/api/system.double.epsilon?view=net-6.0#platform-notes

Accessing Android external storage is security-sensitive

Storing data locally is a common task for mobile applications. Such data includes files among other things. One convenient way to store files is to use the external file storage which usually offers a larger amount of disc space compared to internal storage.

Files created on the external storage are globally readable and writable. Therefore, a malicious application having the permissions `WRITE_EXTERNAL_STORAGE` or `READ_EXTERNAL_STORAGE` could try to read sensitive information from the files that other applications have stored on the external storage.

External storage can also be removed by the user (e.g when based on SD card) making the files unavailable to the application.

Ask Yourself Whether

Your application uses external storage to:

- store files that contain sensitive data.
- store files that are not meant to be shared with other application.
- store files that are critical for the application to work.

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

Recommended Secure Coding Practices

- Use internal storage whenever possible as the system prevents other apps from accessing this location.
- Only use external storage if you need to share non-sensitive files with other applications.
- If your application has to use the external storage to store sensitive data, make sure it encrypts the files using EncryptedFile.
- Data coming from external storage should always be considered untrusted and should be validated.
- As some external storage can be removed, make sure to never store files on it that are critical for the usability of your application.

Sensitive Code Example

```
import android.content.Context;

public class AccessExternalFiles {

    public void accessFiles(Context context) {
        context.getExternalFilesDir(null); // Sensitive
    }
}
```

Compliant Solution

```
import android.content.Context;

public class AccessExternalFiles {

    public void accessFiles(Context context) {
        context.getFilesDir();
    }
}
```

See

- OWASP Top 10 2021 Category A4 - Insecure Design
- Android Security tips on external file storage
- Mobile AppSec Verification Standard - Data Storage and Privacy Requirements
- OWASP Mobile Top 10 2016 Category M2 - Insecure Data Storage
- MITRE, CWE-312 - Cleartext Storage of Sensitive Information
- SANS Top 25 - Risky Resource Management
- SANS Top 25 - Porous Defenses

L Link
C Code

JWT should be signed and verified with strong cipher algorithms

If a JSON Web Token (JWT) is not signed with a strong cipher algorithm (or not signed at all) an attacker can forge it and impersonate user identities.

- Don't use `none` algorithm to sign or verify the validity of a token.
- Don't use a token without verifying its signature before.

Noncompliant Code Example

jsonwebtoken library:

C
```
const jwt = require('jsonwebtoken');

let token = jwt.sign({ foo: 'bar' }, key, { algorithm: 'none' }); // Noncompliant: 'none' cipher doesn't sign the JWT (no signature will be included)

jwt.verify(token, key, { expiresIn: 360000 * 5, algorithms: ['RS256', 'none'] }, callbackcheck); // Noncompliant: 'none' cipher should not be used when verifying JWT signature
```

Compliant Solution

jsonwebtoken library:

C
```
const jwt = require('jsonwebtoken');

let token = jwt.sign({ foo: 'bar' }, key, { algorithm: 'HS256' }); // Compliant

jwt.verify(token, key, { expiresIn: 360000 * 5, algorithms: ['HS256'] }, callbackcheck); // Compliant
```

See

L
- OWASP Top 10 2021 Category A2 - Cryptographic Failures
- OWASP Top 10 2017 Category A3 - Sensitive Data Exposure
- MITRE, CWE-347 - Improper Verification of Cryptographic Signature