Ruby on Rails

Ruby on Rails and the Kiuwan Solutions

This guide explains how to use Kiuwan if your source code is written in Ruby on Rails.

Contents:

Kiuwan Code Security does not offer only source code analysis, but also has the feature of categorizing your rules, create models and Action Plans according to your needs, generate results and share results in the cloud. Although it is not possible to analyze Ruby code directly, it is possible to import the defects or vulnerabilities found with other engines into Kiuwan and make use of the other features.

In the following guide, we will use Brakeman to analyze the code and integrate it with Kiuwan Code Security.

Brakeman is an open source engine designed to find security vulnerabilities in Ruby on Rails applications.

Integrating an external engine in only three steps

Integrating an external engine in Kiuwan requires three steps:

1. Define the rules of your external engine in a Kiuwan format (ruledef) and import them in your model.
2. Convert the output result report of the external engine to a Kiuwan results report format.
3. Analyze the application and upload the results to Kiuwan.

As an example, we are going to use BRAKEMAN as an external engine, and integrate it with Kiuwan.

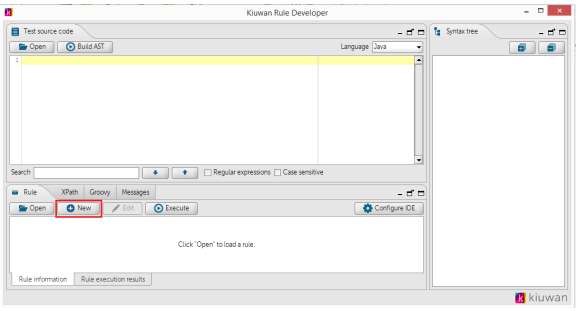
Rules definition

First, you need to import the Brakeman rules (checkpoints, in Brakeman jargon) in your Kiuwan library.

Brakeman's definitions can be found here.

Kiuwan's ruledefs are definition files in XML format. We can use the Kiuwan Rule Developer tool to create these documents, see the installation instructions here.

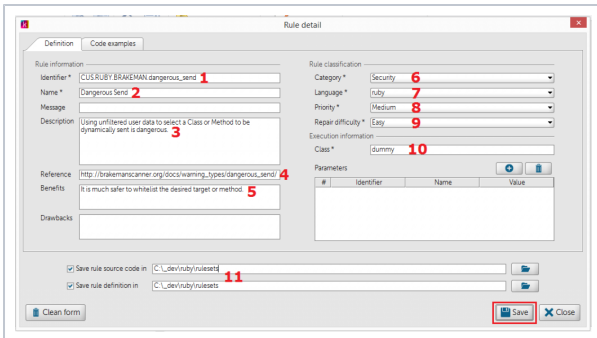
After logging in with your Kiuwan account, in the initial screen, click **New** to create a new rule:



The definition form appears:

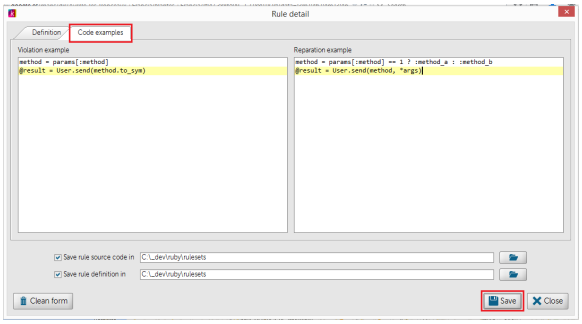
Here is an example on how to fill it, using Brakeman's Dangerous Send rule.

Image	Description



1. A specific identifier for the rule:
   a. namespace: **CUS.RUBY.BRAKEMAN**
   b. unique code for the rule (lowercase, no blank spaces): **dangerous_send**
2. The rule name to be displayed.
3. A simple description.
4. A URL where to find more information for this rule.
5. Benefits and drawbacks.
6. Choose one of the software characteristics that will be mostly affected when the rule finds defects.
7. The programming language, Ruby.
8. Choose the priority for this rule.
9. Choose the repair difficulty of defects found by the rule.
10. Class: This is an unused field when we are using 3rd party engines. Write a dummy value.
11. Choose a target folder where the definition file will be saved.

As extra help documentation, you can include some code examples.



Once we save the definition, the file *CUS.RUBY.BRAKEMAN.dangerous_send.rule.xml* appears in the target folder.

```
<?xml version="1.0" encoding="UTF-8"?>

<rule-definition xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://www.optimyth.com/schema/definitions/rule" version="1.0">

 <rule id="CUS.RUBY.BRAKEMAN.dangerous_send">
   <java-class>dummy</java-class>

   <java-version>1.6</java-version>

   <tags></tags>

   <reference><![CDATA[http://brakemanscanner.org/docs/warning_types
/dangerous_send/]]></reference>

   <parameters/>

   <code-examples>

     <code-example id="codeExample"/>

   </code-examples>

   <incompatibilities/>

   <related-rules/>

   <criteria-values>
     <criterium-value ref="OPT.CRITERIUM_VALUE.LANGUAGE_PARSER.RUBY"/>
     <criterium-value ref="OPT.CRITERIUM_VALUE.PRIORITY.MEDIUM"/>
     <criterium-value ref="OPT.CRITERIUM_VALUE.REPAIR_DIFFICULTY.LOW"/>
     <criterium-value ref="OPT.CRITERIUM_VALUE.CQM.SECURITY"/>
     <criterium-value ref="OPT.CRITERIUM_VALUE.ENGINE.OTHER"/>
   </criteria-values>

   <i18ns>

     <i18n ref="OPT.LANGUAGE.ENGLISH">
       <name><![CDATA[Dangerous Send]]></name>
       <message><![CDATA[]]></message>
       <description><![CDATA[Using unfiltered user data to select a Class
or Method to be dynamically sent is dangerous.]]></description>
       <benefits><![CDATA[It is much safer to whitelist the desired target
or method.]]></benefits>
       <drawbacks><![CDATA[]]></drawbacks>
       <parameters/>
       <code-examples>

         <code-example ref="codeExample">
           <ko><![CDATA[method = params[:method]
@result = User.send(method.to_sym)]]></ko>
           <ok><![CDATA[method = params[:method] == 1 ? :method_a : :
method_b
@result = User.send(method, *args)]]></ok>
         </code-example>

       </code-examples>

     </i18n>
   </i18ns>
 </rule>
</rule-definition>
```

IMPORTANT: You need to **manually edit** this file and replace *OPT.CRITERIUM_VALUE.ENGINE.**QAKING*** by *OPT.CRITERIUM_VALUE.ENGINE.**OTHER***.

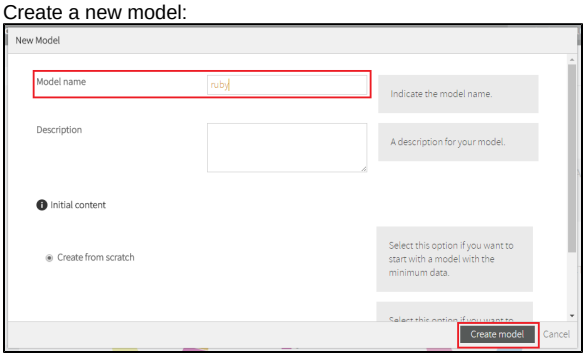
When you have many rule definitions to create, doing them all manually, as described above, can be inefficient. In this case, you can develop a simple program to automate the task.

That program should read the source where the 3rd party engine has the definitions and create an XML file for each rule with the above format (one file with all the ruledefs will work as well).

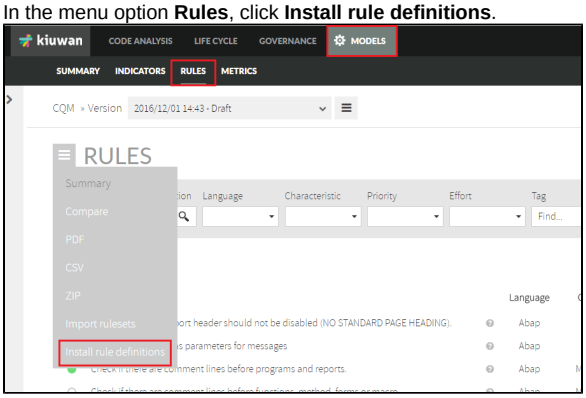
Import this definition to your Kiuwan rules library

To use these new rules in your models you have to import them to your library.

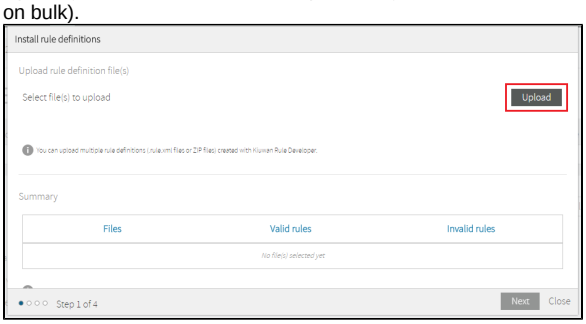
1. Open a new session in Kiuwan
2. Go to **Settings** > **Models Management**
3. Create a new model:



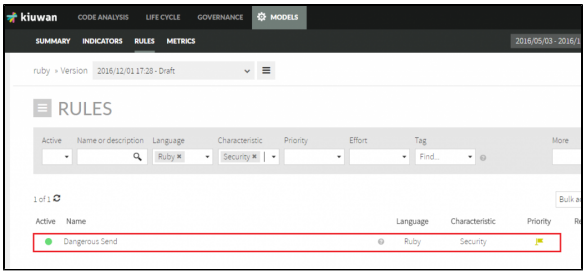
4. In the menu option **Rules**, click **Install rule definitions**.



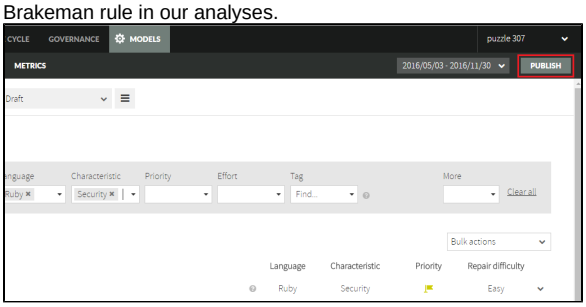
5. Upload your created definition file (or files if you have more than one and want to import them on bulk).



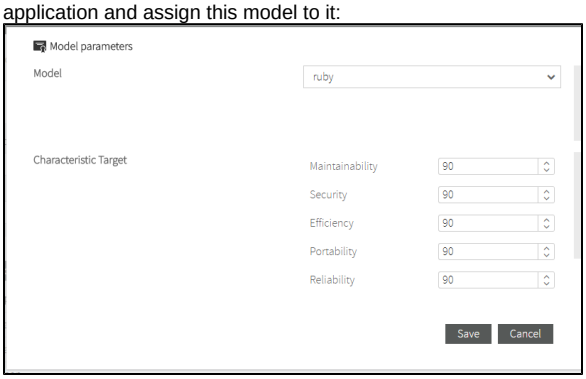
6. Choose the model where you want to include this rule (you can do this in a later step as well). The rule is automatically added to the library and in the selected model (if any was selected). From the library you can add it to any other existing models or the ones you create from now on. You don't need to do the import again since the rule definition is already in your library.



7. Now you can publish the model. This is **mandatory** if you want to use this model with the Brakeman rule in our analyses.



8. To use the model, go to **Settings** > **Application management**. You can create a new application and assign this model to it:



Format converter

Before the next steps, you need a converter program to transform the defects report generated by Brakeman to a format which Kiuwan understands. Find detailed information about this format in Third party analyzers.

For Brakeman, the Kiuwan Team has published a sample transformer application in the Kiuwan Github public account.

You can get the application here.

You need the *dist/kiuwan-thirdparty-report-importer-0.2.2.jar* file.

Analysis of an application

Once you have your converter, you can analyze a Ruby application and see the results in Kiuwan.

This analysis has 3 steps:

1. Analyze the application with Brakeman.
2. Convert the defects report generated by Brakerman to Kiuwan format.
3. Analyze with Kiuwan Local Analyzer to upload the report to Kiuwan.

Analysis with Brakeman

Assuming that the source code of your application is in c:\myapp\src, we should execute:

%RUBY_HOME%\bin\brakeman.bat --format json --out c:\brakeman_report.json c:\myapp\src

The output format of this report is a json file:

```
{
 "scan_info": {
   "app_path": "C:/_github/ruby/rails_admin-master",
   "rails_version": "4.2.0",
...
 },

 "warnings": [
   {
     "warning_type": "Dangerous Send",
     "warning_code": 23,
     "message": "User controlled method execution",
     "file": "C:/_github/ruby/rails_admin-master/app/controllers
/rails_admin/main_controller.rb",
     "line": 28,
...
}
```

Conversion

In this step, you convert the JSON file generated by Brakerman to the XML Kiuwan format. The output file must have 'kiuwan_' as prefix.

```
java -cp kiuwan-thirdparty-report-importer-0.2.2.jar com.kiuwan.importer.Main
Brakeman c:\brakeman_report.json c:\kiuwan_brakeman.xml -base-folder= c:
\myapp\src
```

This is the generated report in Kiuwan format:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>

<kiuwan>
   <defects>
       <violation>
           <file hashed="false" line="28" name="app/controllers/rails_admin
/main_controller.rb">send(params[:bulk_action])</file>
           <rule code="CUS.RUBY.BRAKEMAN.dangerous_send"/>
       </violation>
   </defects>
</kiuwan>
```

Analysis with Kiuwan

In this last step, you will use the Kiuwan Local Analyzer to upload the report to our server. At the same time, the Kiuwan Local Analyzer will calculate volumetric metrics and will run a duplicated code analysis, which are mandatory in any Kiuwan analysis.

You should include the parameter:

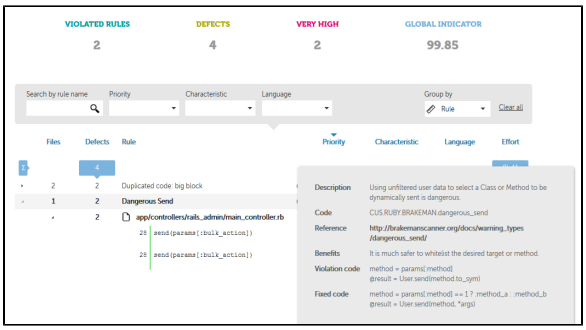
```
    -x, --extended-reports
    Directory where third party reports should be taken from
```

```
"%KIUWAN_LOCAL_ANALYZER%\agent.cmd" -s c:\myapp\src -n myrubyappname -c -x c:
\path_to_folder_with_kiuwan_brakeman_xml
```

Ready to get the most out of Kiuwan

In a few minutes we can see the results on Kiuwan.

Select your Ruby application in Kiuwan, and in the **Defects** screen, you can find the defects detected by Brakeman, in what file and line of code they are and help information (this is the meta information you put in the rule definition) for each imported Brakeman rule.



Additionally, all Kiuwan indicators are calculated taking the Brakeman defects into account. Now you can benefit from all Kiuwan features including the "what if" analysis to create an action plan that may include Brakeman defects, generate PDF reports, share the results with your colleagues, etc...