SecureAssist Rulepack Configurator User Guide

August 2016

Codiscope, LLC
20 Park Plaza, Suite 1400
Boston, MA 02116
Phone: + 1 (617) 804-5428
info@codiscope.com

www.codiscope.com

Table of Contents

1 Introduction and Setup

Rulepacks are Java Archive (JAR) files that contain the files used by SecureAssist to analyze source code against a set of rules and detect vulnerabilities in real time as you code. A rulepack contains filters, guidance, propagators, rules, and sources that customize how and which vulnerabilities are reported.

Codiscope provides a default rulepack. However, in many cases, an organization may wish to scan for issues not included in the default Codiscope rulepacks. For example, rules enforcing a company's specific cryptography standards could be used in addition to the standard SecureAssist cryptography rules.

Using Rulepack Configurator, you can customize the default SecureAssist rulepack to meet your unique needs. You can even create your own rulepacks, save them, then import them into SecureAssist.

System and User Requirements

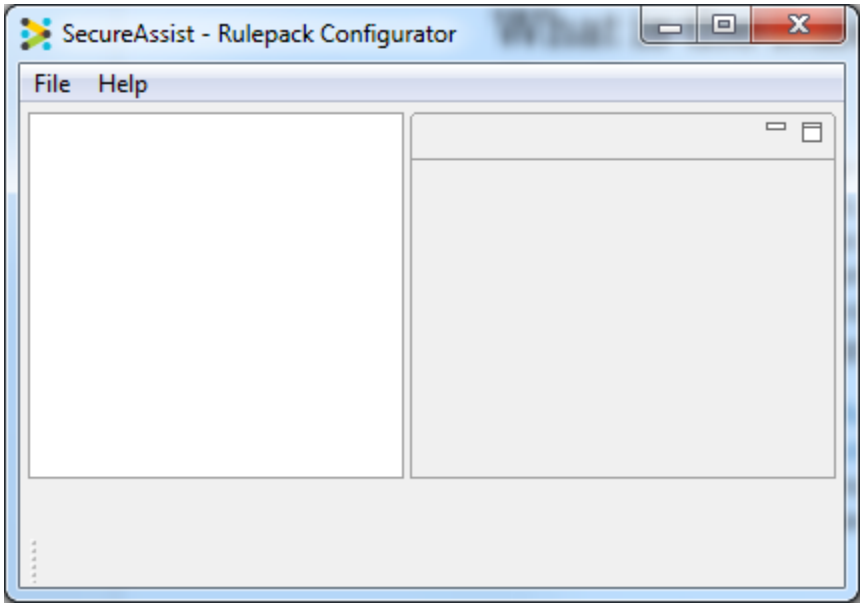
Developing rules requires an understanding of the SecureAssist rule schema, a basic knowledge of regular expressions (regex,) and a programming background. Those developing xml rules need knowledge of XPath. Deciding what the rule itself entails (what code needs to be flagged, what vulnerability the rule is for, and user guidance to help remediate the vulnerability) requires an understanding of software security.

Opening Rulepack Configurator the First Time

A Java Development Kit (JDK) needs to be available to run Rulepack Configurator. The Rulepack Configurator prompts for the JDK path the first time it is opened.

1. Open Rulepack.exe. A dialog will prompt you to enter the JDK path. (Rulepack Configurator prepopulates with the JDK path, if JDK is available in the program files folder.)

2. Browse to the JDK path (if necessary) and click **OK**. The dialog closes and Rulepack Configurator opens.



3. Exit the program by choosing **Exit** from the **File** menu or clicking the  . A popup will ask whether you want to continue.

4. Click **OK**. A second dialog will ask whether you want to save any new or extracted files in the archive.

5. Click **Yes**. The JDK path will now be saved, and you won't be prompted to enter it again.

2 Rulepack Configurator Quick Start

What are Rulepacks?

Rulepacks are Java Archive (JAR) files that contain the files used by SecureAssist to analyze source code against a set of rules and detect vulnerabilities in real time as you code. A rulepack contains filters, guidance, propagators, rules, and sources that customize how and which vulnerabilities are reported.

When you open a rulepack in Rulepack Configurator, you will see that it consists of two folders: Default and Custom.

- The Default folder is read-only and contains all the default elements in the rulepack.
- The Custom folder is where you can add or edit custom rules, sources, propagators, filters, and guidances. The individual custom settings you create in Custom will override the corresponding settings in Default. The Custom folder tree is empty until you add custom files to it.

To be usable, a rulepack should contain at least one rule, one filter, and one guidance. These three elements reference one another. Propagators and sources, conversely, do not reference any other elements.

Item	Description
Rules	Instruct SecureAssist on what vulnerabilities to identify in your code. *Linked to Filter by Rule ID.* *Links to Guidance by Rule Standard (guidance filename).*
Filters	Modify the way rules behave. Organize rules into categories, determine which rules are visible in the editor window of the IDE, and determine rules' severity levels. A rule must be added to a filter and activated for SecureAssist to display the rule's results. Custom rules that you create are disabled by default and must be enabled via a filter. *Links to one or more rules by Rule ID.*
Guidances	Provide advice to a SecureAssist user about how to fix the vulnerability. When code triggers a finding, guidance supplies the developer with the language-specific information necessary to understand and remediate the vulnerability. *Linked to in one or more rules by Rule Standard (guidance filename).*
Propagators	Track how taint travels through a system and describe conditions on which taint is passed between objects. *Feeds information to the Call Chain in SecureAssist.*

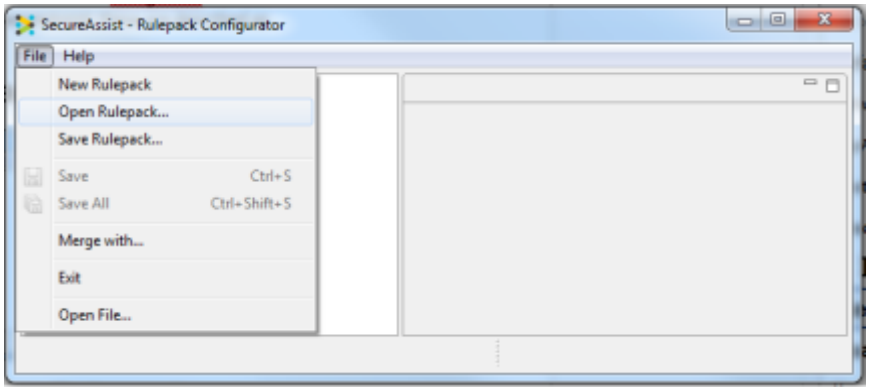
Item	Description
Sources	Inform SecureAssist where untrusted data can potentially enter an application. When SecureAssist finds a source, it is marked as such, and used to determine which lines of code are insecure during taint flow analysis. *Feeds information to the Call Chain in SecureAssist.*

Open and View an Existing Rulepack

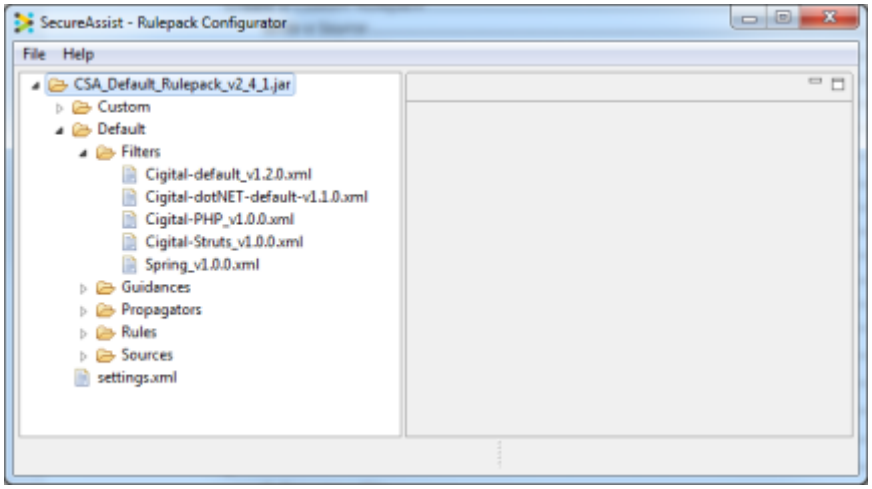
When you first open Rulepack Configurator, you'll be confronted with a blank screen. To populate it, follow these instructions.

Open a Rulepack

1. From the **File** menu, choose **Open Rulepack**.



2. From the **Open File** dialog, browse to and select the desired JAR file, then click **Open**. Rulepack Configurator displays the selected rulepack in a folder tree.



3. Browse the folders to see the different component files. The Default folder contains the default files for the rulepack. These files cannot be edited. To make changes, you can add and edit files in the Custom folder.

Open and View Files in a Rulepack

1. In an open rulepack, browse the navigation tree to locate the desired file.

2. Double-click an individual file. The file opens in a tab in the editing window.

3. To close the file, click the on the file's tab at the top of the editing window.

Remember! Files in the Default folder will be read-only.

Customize a Rulepack

Your company may have rules you'd like to activate in SecureAssist that don't appear in the default rulepack. To do this, you can customize the rulepack. There are two ways to customize a rulepack: add new unique custom files OR revise default files by editing copies of them.

Rulepack Settings

The settings for your rulepack appear in the settings.xml file at the bottom of the navigation tree. To edit them, double-click the file to display it in the editing window.

By default, the setting Custom filters have priority is selected. This means that when the rulepack is run in SecureAssist, any default rulepack files that you've customized will take precedence over the default files.

The Filter Settings is where you can choose to activate or deactivate all the filters contained in the rulepack. By default, all custom filters are activated, but any

Views

When you open a custom file in a rulepack, you can choose to view or edit it in one of two views, which you select by clicking on the tabs on the bottom of the editing window:

- ***Design.*** The **Design** tab presents the file's xml elements in a collapsible, graphical hierarchy.
- ***Source.*** The **Source** tab presents the file code for direct editing.

The exception to this is guidance files. By default, a guidance file is displayed in a graphical wizard. However, you can choose to view guidance files using the **Design** and **Source** tabs by right-clicking the file and choosing **View Source** from the popup menu. This will open a second tab displaying the file in the format.

Working with Files

Customize a Default Guidance File

Custom guidance files created this way will override the rulepack's default files.

1. In the navigation tree, right-click on either of the Guidances folder and select **Customize Default Guidance**. A dialog lists all the existing default guidance files.
2. Type the desired file name in the **Select Guidance** box OR select the desired file from the list, then click **OK**. The file is copied to the Custom Guidance folder with the same file name as the default file. The new file is opened automatically in the editing window.
3. If needed, resize the editing window so that the **Overwrite** checkboxes to the right of the data entry fields are displayed.
4. Check **Overwrite** next to the sections you wish to change, then make the desired changes to those sections. (Each section is read-only until you click its **Overwrite** box.)
5. To revert a section to the original content from the default guidance file, deselect **Overwrite**. Your edits will be reversed, and the section contents will be restored.
6. Choose **Save** from the **File** menu.
7. To close the file, click the ☒ on the file's tab.

Important! When you save a rulepack file, the changes will be retained for that session only UNLESS you additionally choose **Save Rulepack** from the **File** menu.

Customize a Default Filter, Propagator, Rule, or Source File

Custom files created this way will override the rulepack's default files.

1. In the navigation tree, open the default file you wish to customize.
2. Copy the contents of the default file, then close its tab.
3. Right-click on any of the Custom subfolders, select **New**, then select the type of file you wish to create. A dialog prompts you to enter a new name for the file you're creating, plus any other required fields for the file type.
4. Enter the same file name as the original default file. ***Important!*** The file name must be identical in order to ensure the custom file overrides the default file.
5. Enter values in any other fields, then click **Create**. The file is created in the appropriate folder and opened in a tab in the editing window.
6. Enter data in fields as desired.
7. Choose **Save** from the **File** menu.
8. To close the file, click the ☒ on the file's tab.

Create a New Custom File

Custom files created this way will run in addition to the rulepack's default files.

1. In the navigation tree, right-click on any of the Custom subfolders, select **New**, then select the type of file you wish to create. A dialog prompts you to enter a new name for the file you're creating, plus any other required fields for the file type.
2. Enter a unique file name. ***Important!*** If you want this custom file to run *in addition* to the default files, it must have a unique file name.
3. Enter values in the fields, then click **Create**. The file is created in the appropriate folder and opened in a tab in the editing window.
4. Enter data in fields as desired.
5. Choose **Save** from the **File** menu.
6. To close the file, click the ☒ on the file's tab.

Validate a File

1. Right-click the desired file and select **Validate** from the popup menu. Validation is run on the file. A popup will confirm successful validation.

   If validation fails, an error message will list any problems you must resolve before validation can be successful.

Preview Guidance

This feature allows you to view how your guidance page will appear in SecureAssist.

1. Right-click the desired guidance file and select **Preview** from the popup menu. A read-only tab opens in the editing window, displaying the guidance as it will appear in SecureAssist.

2. To close the preview, click the ☒ on the tab.

Rename a File

1. In the navigation tree, right-click the desired file and select **Rename**. The File Rename dialog opens.

2. Edit the file name as desired and click **Rename**.

Save Current File

1. To save the current file, choose **Save** from the **File** menu OR press **Ctrl+S**.

Save All Open Files

1. To save all open files in a rulepack, choose **Save All** from the **File** menu OR press **Ctrl+Shift+S**.

Delete a Custom File

Default files cannot be deleted.

1. In the navigation tree, right-click the desired file and select **Delete**.

Rulepacks

Validate Rulepack

1. Right-click on the rulepack (the top folder) in the navigation tree and select **Validate Rulepack**. Validation is run on the entire rulepack. A popup will confirm successful validation.

   If validation fails, an error message will list any problems you must resolve before validation can be successful.

Save Rulepack

1. Choose **Save Rulepack** from the **File** menu. The rulepack is validated before saving. If validation fails, an error message will list any problems you must resolve before validation can be successful.

3 Creating and Editing Custom Rulepack Files

This section describes the details of each type of custom rulepack file in the SecureAssist Rulepack Configurator.

Custom Filter

Filters allow the grouping of rules into problems. Each rule can be assigned a level of importance, be activated or deactivated, or have a flag set to show a marker in the IDE.

Example: Filter XML File Without Rules

```
<Filter desc="" name="" version="" ide="">
  <Problems>
    <Problem name="">
      <Rules>
        <Rule id="" active="true"
importance="" markerShow=""/>
        ...
      </Rules>
    </Problem>
    ...
  </Problems>
</Filter>
```

The following table describes the elements that are part of the example shown above.

Filter XML Elements

Item	Description
Filter	Contains information describing filter. Attributes: ▪ **desc** – filter's description ▪ **name** – filter's name ▪ **version** – filter's version ▪ **ide** – "eclipse" if filter contains rules to be used by eclipse plug-in. "dotNET" if filter contains rules to be used by dotNET addin.
Problem	Contains list of rules that should belong to this problem grouping. Attributes: ▪ **name** – name of the problem in human readable form.
Rules	Contains list of rules that belong to the problem.

Item	Description
Rule	Defines rules that are part of the filter. Attributes: ▪ **id** – id of the rule that is defined in the filter ▪ **active** – "true" if rule should be used for the scan, otherwise "false" ▪ **importance** – there are three levels of importance that can be assigned to the rule: low, medium and high. ▪ **markerShow** – "true" if marker should be created on the line of code where rule has fired, otherwise "false". Even if this argument is set to "false" and rule fires, user is able to find the result in WBSA Issues List view.

Custom Guidance

Guidance consists of two major parts. The first part contains generic guidance about the security problem while the second part of the guidance contains technology specific guidance. For example, guidance for SQL Injection will have a short and long description describing the problem in general terms but will also have specific guidance for different contexts such as Java or .NET.

Example: Guidance XML File

```
<Standard>
  <Title>Title</Title>

<ShortDescription>ShortDescription</ShortDescription>
  <LongDescription>LongDescription</LongDescription>
  <Classifications>
    <Classification source="">
      <URL>URL</URL>
      <Name>Name</Name>
    </Classification>
  </Classifications>
  <Contexts>
    <Context name=" " version="">
      <CodeSamples>
        <CodeSample type="">
          <Description>Description</Description>
        </CodeSample>
      </CodeSamples>
      <Recommendations>

<Recommendation>Recommendation</Recommendation>
      </Recommendations>
      <References>
        <Reference type=" ">
          <URL>URL</URL>
          <Description>Description</Description>
        </Reference>
      </References>
    </Context>
  </Contexts>
</Standard>
```

The following table describes the elements that are part of the example shown above.

XML Elements in Guidance XML File

Item	Description
Title	Title of the guidance
ShortDescription	A short description about the security problem this guidance relates to. This element is used as a tooltip when you hover over the marker in IDE.
LongDescription	A long description about the security problem this guidance relates to.
Classifications	List of industry standard classifications related to this guidance.
Classification	Classification name and URL related to guidance Attributes: ▪ **Source** – source of the classification (e.g., CWE, OWASP, etc.)
Contexts	List of contexts to which this guidance relates to
Context	Contains context related code samples and recommendations Attributes: ▪ **Name** – name of the context (e.g., J2EE, .NET, Hibernate, etc.) ▪ **Version** – Version of the context (optional field)
CodeSample	Sample code showing the vulnerability (Negative code sample) or a safe technique that does not pose a security risk (Positive code sample). Attributes: ▪ **Type** – type of the code sample (Positive, Negative)
Recommendations	List of recommendations specific to the context
Recommendation	Text describing the recommendation
References	List of references
Reference	Reference regarding security weakness in specified context Attributes: ▪ **Type** – type of the reference (Internal, External, RelatedStandard)

Guidance can be created using the guidance editor available in the Whitebox SecureAssist™ Rulepack Configurator tool.

Customizing Default Guidance

Any default guidance that comes as a part of Cigital SecureAssist Rulepack can be customized. Any customized changes made to the guidance will persist, even if default guidance is updated in a future Rulepack. For detailed instructions on how to customize default guidance please refer to Cigital SecureAssist Rulepack Configurator User's Guide.

Creating Custom Guidance

Customized guidance can be created using the Cigital SecureAssist™ Rulepack Configurator tool. Custom guidance has to conform to the XSD provided in the Rulepack. For detailed instructions on how to create custom guidance please refer to Whitebox SecureAssist™ Rulepack Configurator User's Guide.

Custom Source

Sources are used to identify parts of the code where tainted data is entering the system. Currently there are four types of taint:

- **FILE** – to identify data passed to application from file
- **DB** – to identify data passed to application from database
- **WEB** – to identify data passed to application from web
- **PRIVATE** – to identify data passed to application as private data

Example: Sources XML File

```
<Sources>
  <Source lang="" taint="">

<QualifiedName>class_qualified_name</QualifiedName>
    <Method>method_name<Method>
    <Arguments>
      <Argument>
        <Value>

<ComparatorOperator>equals</ComparatorOperator>

<ExpectedValue>argument_index</ExpectedValue>
          <ComparatorType>String</ComparatorType>
        </Value>
        <Index>0</Index>
      </Argument>
    </Arguments>
  </Source>
</Sources>
```

The following table describes the elements that are part of the example shown above.

Sources XML Elements

Item	Description
Source	Attributes: ▪ **lang** – language for which this source is defined - "java" for java files - "cs" for C# files - "vb" for VB.NET files - "cs|vb" for C# and VB.NET files ▪ **taint** – type of the taint that the source is introducing
QualifiedName	Qualified name of the object for which source is being defined
Method	Method name for which returning value is being tainted. For C# and VB.NET it can be property name.
Arguments	Optional element. Arguments are specified using the same syntax for sources, propagators, and rules. See the Rules section for instructions on writing Arguments elements.

Custom Propagator

Propagators are method invocations or constructor invocations that do not affect the taint associated with a variable and hence the variable remains tainted in the subsequent statements, too.

Example: Propagators XML File

```
<PropagationRule lang="java"
ruleID="JAVA_PROPAGATOR_027">

<QualifiedName>java.lang.StringBuilder</QualifiedName>
    <Method>append</Method>
    <Arguments>
      <Argument type="java.lang.Object">0</Argument>
        <Type>java.lang.Object</Type>
        <Index>0</Index>
      </Argument>
    </Arguments>
    <Propagate>
      <ReturnValue>
       <Caller>true</Caller>
       <Argument>0</Argument>
      </ReturnValue>
      <Caller>
       <Argument>0</Argument>
      </Caller>
    </Propagate>
</PropagationRule>
```

The following table describes all the XML elements and attributes that are possible in a propagator. The values of XML elements and attributes are treated as strings except wherever mentioned that values are treated as regular expression.

Propagator XML Elements

Item	Description
PropagationRule	Represents one propagator Attributes: ▪ **lang** (Required) – language for which this propagator is defined. - "java" for java files - "cs" for C# files - "vb" for VB.NET files - "cs|vb" for C# and VB.NET files ▪ **ruleID** (Required) – unique ID to identify the propagator. This has to be unique within a rulepack (jar file). One may use the same ID in a different rulepack, though it is not recommended.
QualifiedName	Fully qualified name of the class to which the method or constructor belongs. Required and exactly one instance of this XML element. The value is treated as a regular expression. Attributes: ▪ **extends** (Optional) – Can be true or false. By default it is false. If true, invocation of overridden methods/constructors in subclasses of this fully qualified name would be treated as propagator too.

Item	Description
Method	Maximum one instance of this XML element. Method name which has to be treated as a propagator. If the element is missing, this propagator would be treated as a constructor propagator with class name from QualifiedName element. The value of this element is treated as a regular expression.
Arguments	Optional element. Arguments are specified the same way for sources, propagators, and rules. See the Rules section for instructions on specifying arguments.
Argument (Optional child element of Arguments and mandatory child of ReturnValue and Caller)	Argument is specified the same way for sources, propagators, and rules. See the Rules section for instructions on specifying arguments.
Propagate	Required and exactly one occurrence. This element has the tainting criteria for the propagator. The scenarios and the resulting actions are provided in child elements of this element. Possible child elements are ReturnValue, Caller and ReferencedArgument. At least of these elements have to be present.
ReturnValue (child element of Propagate)	Optional element with one maximum occurrence. This element tells the CSA engine what further needs to be tainted if the return value of the method invocation is already tainted thereby propagating the taint. The possible child elements are "Argument" and "Caller" (explained below). At least one of the have to be there, both can be there too.
Caller (child element of Propagate)	Optional element with one maximum occurrence. This element tells the CSA engine what further needs to be tainted if the "caller" (the variable on which the method is invoked) of the method invocation is already tainted thereby propagating the taint. The required child element is "Argument" (explained below).
ReferencedArgument (child element of Propagate)	Optional element with maximum occurrences not exceeding the number of arguments to the method/constructor. This element tells the CSA engine that if a particular argument is already tainted then what further should be tainted thereby propagating the taint. The possible child elements are "Argument" and "Caller" (explained below). At least one of the have to be there, both can be there too. Attributes: ▪ **argumentNumber** (Required) – A non-negative index of the argument which is already tainted.

Item	Description
Caller (child element of ReturnValue and ReferencedArguemnt)	Optional element with one maximum occurrence. Can only have value "true". Means the "caller" (the variable on which the method is invoked) should be tainted too with the same taint type as of the return value of the method invocation (if this element is a child of ReturnValue) or the taint type of the argument indexed at the "argumentNumber" (if this element is a child of ReferencedArgument).
Argument (child element of ReturnValue and ReferencedArguemnt)	Optional element. Argument is specified the same way for sources, propagators, and rules. See the Rules section for instructions on specifying arguments.

Custom Rule

Rules are used to identify parts of the code that can lead to a security vulnerability and link the code to specific guidance. There are different rules that can be created for different file types (e.g., Java, JSP, XML, C#, etc). Even though rules differ from each other there are a set of elements that are shared between different rules.

Rule Structure and Common Elements

All the rules in Cigital SecureAssist™ share a common structure and some of the elements. The follow example presents empty rule structure with common elements.

Example: Rules XML File

```
<Rules>
  <Rule id="" lang="">
    <Category>Category</Category>
    <Title>Title</Title>
    <Description>Description</Description>
    <Match>
       ...
    </Match>
    <Standards>
       <Standard file="standard file name">
         <Context
version="version">context_name</Context>
       </Standard>
    </Standards>
  </Rule>
  ...
</Rules>
```

The following table describes the elements that are part of the example shown above.

Common Elements of Rules XML File

Item	Description
Rule	Attributes: ▪ **id** – rule id ▪ **lang** – language for which rule is defined (java, jsp, xml, cs, vb, aspx, config, xsd, porperties)
Category	Rule's category
Title	Rule's title
Description	Rule's description

Item	Description
Match	Contains information that are being used to perform scan. Since content of this element differs based on the rule, a detailed description of this element is available for the different rules described in the following chapters.
Standards	List of standards related to the rule
Standard	Guidance that relates to the rule Attributes: ▪ **file** – standard file name
Context	Context of the guidance that should be displayed if rule fires. See the Guidance description for possible Context values. Attributes: ▪ **version** – context version that should be displayed

Semantic and Taint Flow Rules

Semantic rules allows SecureAssist™ to identify code that can lead to security weaknesses like banned APIs, misuse of API calls, or use of APIs in the wrong block of code.

Example: Match XML Element for Semantic Rules

```
<Match block="" >
  <QualifiedName extends="" ></QualifiedName>
  <Method></Method>
  <Arguments>
    <Argument>
      <Index>0</Index>
      <Value>

<ComparatorOperator></ComparatorOperator>
        <ExpectedValue></ExpectedValue>
        <ComparatorType></ComparatorType>
      </Value>
    </Argument>
  </Arguments>
</Match>
```

Data flow rules identify code that is using data in an inappropriate way. This could lead to security weaknesses like SQL Injection, Log Injection, or Command Injection.

Example: Empty Match Element Used in Data Flow Rules

```
<Match block="" >
  <QualifiedName extends="" ></QualifiedName>
  <Method></Method>
  <Condition type="OR" >
    <Caller>

<Taint><Type>WEB|DB|FILE|PRIVATE</Type></Taint>
    </Caller>
    <Argument>
      <Index>0</Index>

<Taint><Type>WEB|DB|FILE|PRIVATE</Type></Taint>
    </Argument>
  </Condition>
</Match>
```

The following table describes the elements that are part of the examples shown above.

Match XML Elements Description for Semantic Rules

Item	Description
Match	Attributes: ▪ **block** – block in which API call should be located
QualifiedName	Qualified name of the object that the rule matches (interpreted as a regular expression). Attributes: ▪ **extends** – "true" if rule should fire on any object that extends class identified by qualified name, otherwise "false"
Method	The name of the method that the rule matches (interpreted as regular expression).
Arguments	Holds Argument elements that express restrictions based upon an Argument's type, index, or value. size – optional attribute specifying the number of arguments in the method invocation or constructor.
Argument	Argument elements can hold between 0 and 1 of each of the following elements: Index, Value, Taint, and Type.
Index (optional child of Argument)	Holds the index or indicies of the argument that is being matched. This can contain single argument, multiple arguments or range of arguments, e.g.: -Single argument <Argument><Index>0</Index></Argument> -Multiple arguments <Argument><Index>0,1,4</Index></Argument> -Range of arguments <Argument><Index>0-4</Index></Argument>

Item	Description
Value (optional child of Argument)	
ComparatorOperator (mandatory child of Value)	The type of operator to use in evaluating whether the actual value is a match for the Expected Value. A list of valid ComparatorOperators ordered by ComparatorType is provided in table 6.
ExpectedValue (mandatory child of Value)	The value the argument should be compared to in evaluating a match.
ComparatorType (mandatory child of Value)	The type of the value being evaluated. The possible values are String and int.
Taint (optional child of Argument and mandatory child of Caller)	Provides information regarding the taint the argument must contain for it to be a match. Contains one mandatory Type element. Taint elements are only applicable to arguments in Rules and is not valid for Source and Propagator elements.
Type (mandatory child of Taint)	Holds the type of the taint the argument possesses. The possible values are FILE, DB, WEB, PRIVATE, or UNTRUSTED. These values could be ORed using the pipe | symbol. UNTRUSTED is equivalent to WEB|DB|FILE|PRIVATE.
Type (optional child of Argument)	Specifies the fully qualified name of the type of the Argument. This will match on either an exact match, a superclass of the argument, or for an interface the argument implements.
Caller	Used when the object/variable on which the method is invoked needs to be checked for taint. Requires the taint type to be specified within a child Taint element.
Condition (optional child of Match and mandatory child of Conditions)	An optional element that allows boolean logic. (For example, if a method call has 3 arguments or the first argument has taint.). Has a mandatory attribute of type that can be OR or AND depending on how its children aught to be evaluated. Can have one or more of the following as children: Conditions, Caller, Argument, or Arguments.
Conditions (optional child of Condition)	An element that allows greater complexity to boolean logic than Conditon by allowing Condition elements to be grouped together. Has a mandatory attribute of type that can be OR or AND depending on how its children aught to be evaluated. Has one or more Condition elements as children.

The following table presents possible values for type and comparator in rule Match XML.

Argument Type and Comparators

Type Value	Comparator
String	Equals notEquals startsWith endsWith regex
Int	Equals notEquals greaterThan greaterThanOrEquals lessThan lessThanOrEquals

Java Examples

Rule identifies code where user generated Cipher object that implements DES transformation.

Rule in this example could be described as follows: "Find getInstance method call of object type javax.crypto.Cipher and if first parameter passed to it has string value equal to DES mark this part of code."

Example: Java Semantic Rule Match XML Element

```
<Match>

<QualifiedName>javax.crypto.Cipher</QualifiedName>
  <Method>getInstance</Method>
  <Arguments>
    <Argument>
      <Index>0<Index>
      <Value>

<ComparatorOperator>equals</ComparatorOperator>
        <ExpectedValue>DES</ExpectedValue>
        <ComparatorType>String</ComparatorType>
      </Value>
    <Argument>
  </Arguments>
</Match>
```

Rule identifies code where insecure API call is being used. The following example presents rule to identify any part of the code where the executeQuery or executeUpdate method of an java.sql.Statement object is called.

Example: Java Semantic Rule Match XML Element

```
<Match>

<QualifiedName>java.sql.Statement</QualifiedName>
  <Method>(executeQuery|executeUpdate)</Method>
</Match>
```

C# Examples

Rule identifies code where weak cryptographic hashing function are being used.

Example: C# Semantic Rule Match XML Element

```
<Match>
   <QualifiedName>
      System.Security.Cryptography.MD5
   </QualifiedName>
   <Method>

ComputeHash|HashCore|TransformBlock|TransformBlock
   </Method>
</Match>
```

Rule identifies code where weak cryptographic algorithm is used with weak key size.

Example: C# Semantic Rule Match XML Element

```
<Match>
   <QualifiedName>
      System.Security.Cryptography.RSA
   </QualifiedName>
   <Arguments>
    <Argument>
      <Index>0<Index>
      <Value>

<ComparatorOperator>lessThanOrEqual
        </ComparatorOperator>

<ExpectedValue>512</ExpectedValue>

<ComparatorType>int</ComparatorType>
      </Value>
    <Argument>
  </Arguments>
</Match>
```

Sample Data Flow Rules

Java Examples

Identify call to preparedStatement, where first argument or the caller (object/variable on which the method is invoked) of the method/constructor comes from an untrusted source.

This example will match all method calls to method preparedStatment of object java.sql.Connection where the first argument contains taint.

Example: Java Data Flow Rule Match XML Element

```
<Match>

<QualifiedName>java.sql.Connection</QualifiedName>
  <Method><![CDATA[prepareStatement]]></Method>
  <Condition type="OR”>
    <Arguments>
      <Argument>
        <Index>0</Index>
        <Taint><Type>FILE|DB|WEB</Type></Taint>
      </Argument>
    </Arguments>
    <Caller>
      <Taint><Type>FILE|DB|WEB</Type></Taint>
    </Caller>
  </Condition>
</Match>
```

C# Examples

Rule identifies part of the code where data provided by malicious user or attacker can manipulate a SQL query This rule does so when the SqlCommand is executed with a tainted parameter as well as when the SqlCommand itself is tainted.

Example: C# Data Flow Rule Match XML Element

```
<Match>
<QualifiedName>System.Data.SqlClient.SqlCommand</QualifiedName>
  <Method><![CDATA[Text]]></Method>
  <Arguments>
    <Argument>
      <Index>0</Index>
      <Taint><Type>FILE|DB|WEB</Type></Taint>
    </Argument>
  </Arguments>
  <Caller>
    <Taint><Type>FILE|DB|WEB</Type></Taint>
  </Caller>
</Match>
```

Custom RegEx Rules

This section describes step-by-step how to create regex custom rules that will identify part of the code that can lead to security weakness.

Example: Empty Match Element Used in Regex Rule

```
<Match>
  <Pattern ignoreCase=" " mark=" "></Pattern>
</Match>
```

The following table describes the match element that is part of the regex rule XML presented in the above example.

Match XML Elements Description for Regex Type Rules

Item	Description
Match	Match element
Pattern	RegEx pattern that will cause rule to fire. Attributes: ▪ **ignoreCase** – "true" if case is not important (default), otherwise false ▪ **mark** – "true" if part of the code identified by rule should be marked (default), otherwise "false".

The following example presents empty Match element used in regex rule with conditional element.

Example: Empty Match Element Used in Regex Rules with Conditional

```
<Match>
  <Conditional type="">
    <Pattern ignoreCase=" " mark="
"></Pattern>
    <Pattern ignoreCase=" " mark="
"></Pattern>
    <Conditional type="">
    ...
    </Conditional>
  </Conditional>
</Match>
```

The following table describes the Match element that is part of the regex rule with conditional element presented in the above example.

Match XML Elements Description for Regex Type Rules with Conditional

Item	Description
Match	Match element
Conditional	Conditional element used to evaluate if rule should fire Attributes: ▪ **type** – type of the conditional check to perform. Currently there are three types of conditional values: AND, OR, NOT
Pattern	RegEx pattern that is used to evaluate conditionals. (See table 9 for further details on Pattern Conditionals)

Examples

Rule searching for JSP output tag in JSP files is shown in the following example.

Example: Regex Rule Match XML Element

```
<Match>
```

```
  <Pattern ignoreCase="false" mark="true">
    <![CDATA[\<\%\=]]>
  </Pattern>
</Match>
```

Rule shown in the following example is searching for error page defined at the top of the JSP file. The rule fires if any one of the patterns match.

Example: Regex Rule Match XML Element with Conditional

```
<Match>
  <Conditional type="OR">
    <Pattern ignoreCase="true"
mark="true"><![CDATA[\<\%\@ page
errorPage=[\"|\'][\w\W
]*[\"|\'][\s]*\%\>]]></Pattern>
    <Pattern ignoreCase="true"
mark="true"><![CDATA[errorPage\=]]></Pattern>
  </Conditional>
</Match>
```

Custom XML and Config Rules

This section described step-by-step how to create XPath custom rules that will identify parts of configuration files that can lead to security weakness.

Example: Empty Match Element Used in XPath / Config XML Rules

```
<Match>
      <Prefix>Prefix</Prefix>
      <Namespace>Namespace</Namespace>
      <RootElement>RootElement</RootElement>
      <XPath>XPath</XPath>
      <Value>Value</Value>
</Match>
```

The following table presents description of Match element that is part of the XML / Config / XSD rule presented in the example above.

Match XML Elements Description for XML/Config Type Rules

XML	Description
Match	Match element
Prefix	Namespace prefix
Namespace	Namespace
RootElement	Root element of XML file
XPath	XPath to be evaluated
Value	Expected value returned by XPath

XML Examples

Rule looking for session-timeout set in web.xml file is shown below

Example: XPath Rule Match XML Element

```
<Match>
  <RootElement>web-app</RootElement>
  <XPath><![CDATA[/*[name()='web-
app']/*[name()='session-
config']/*[name()='session-
timeout']]]></XPath>
</Match>
```

Rule looking for action that attribute validate is not set to true is shown below.

Example: XPath Rule Match XML Element

```
<Match>
  <RootElement><![CDATA[struts-
config]]></RootElement>
  <XPath><![CDATA[/*[name()='struts-
config']/*[name()='action-
mappings']/*[name()='action']/@validate]]></XPath>
  <Value><![CDATA[true]]></Value>
</Match>
```

Config Examples

Rule verifies a session timeout value has been configured in web.config.

Example: Config Rule Match XML Element

```
<Match>
   <RootElement>configuration</RootElement>
   <XPath>
<![CDATA[//configuration/system.web/sessionState[@timeout>30]]]>
   </XPath>
</Match>
```

XSD Examples

Custom rule looking for presence of an element in sample xsd file is shown below.

Example: XPath Rule Match XML Element

```
<Match>
      <RootElement><![CDATA[xs:schema]]></RootElement>

<XPath><![CDATA[//xs:element[@name='Root']/xs:complexType
/xs:sequence/xs:element[@name='Customers']/xs:complexType
/xs:sequence/xs:element[@name='Customer']]]></XPath>
    </Match>
```

Note: For VS, you will need to select language as config to write custom xsd rules; for Eclipse/IntelliJ, you will need to select language as xml for xsd rules.

Custom Properties Rules

This section describes step-by-step how to create a custom properties rule.

Example: Empty Match Element Used Properties Rules

```
<Match>
   <Key>Key</Key>
   <Value comparator=" ">Value</Value>
</Match>
```

The following table presents description of Match element that is part of the properties rule presented in the example above.

Match XML Elements Description for Properties Type Rules

XML	Description
Match	Match element
Key	Name of the parameter (key)
Value	Value of the key Attributes: ▪ **Comparator** – type of the comparator to be used during match of parameter against expected value. Available comparators : contains, endsWith, equal, notEquals, startsWith

Examples

Rule checks if hibernate SQL logging is turned on in hibernate properties file.

Example: Properties Rule Match XML Element

```
<Match>
   <Key>hibernate.show_sql</Key>
   <Value comparator="equals">true</Value>
</Match>
```

Custom ASPX Rules

This section describes step-by-step how to create a custom ASPX rule.

Example: Empty Match Element Used in ASPX Rules

```
<Match>
   <Element>input</Element>
   <Attribute>type</Attribute>
   <Value comparator=" " type=" " value="
"></Value>
</Match>
```

The following table presents description of Match element that is part of the ASPX rule presented in the example above.

Match XML Elements Description for ASPX Type Rules

XML	Description
Element	Element's name
Attribute	Attribute's name
Value	Element represents value being searched. Attributes: ▪ **type** – type of expected value : Int, String, Boolean ▪ **value** – value to be used in comparison. If value is not present rule can match value with other element/attribute pair. ▪ **comparator** – type of the comparator to be used during match of parameter against expected value. Available comparators : - for type String : contains, endsWith, equal, notEquals, startsWith - for type Int : equals, greaterThan, greaterThanOrEqual, lessThan, lessThanOrEqual - for type Boolean : equals, notEquals

Examples

Rule identifies hidden fields in ASPX page.

Example: ASPX Rule Match XML Element

```
<Match>
   <Element>input</Element>
   <Attribute>type</Attribute>
   <Value comparator="equals" type="String"
value="hidden"/>
</Match>
```

Rule verifies that range validator has been specified for every TextBox control.

Example: ASPX Rule Match XML Element

```
<Match>
   <Element>asp:TextBox</Element>
   <Attribute>id</Attribute>
   <Value comparator="equals" type="String">
     <Element>asp:RangeValidator</Element>
     <Attribute>ControlToValidate</Attribute>
   </Value>
</Match>
```

4 More Support

We hope this document has helped you get started with SecureAssist. You can submit a support request at support.codiscope.com. You will also find other manuals, release notes, system requirements, and more.

Thanks for using Codiscope SecureAssist!

www.codiscope.com
20 Park Plaza, Suite 1400
Boston, MA 02116
Phone: + 1 (617) 804-5428
info@codiscope.com