Java源代码安全审计使用说明

**—— 安全团队**

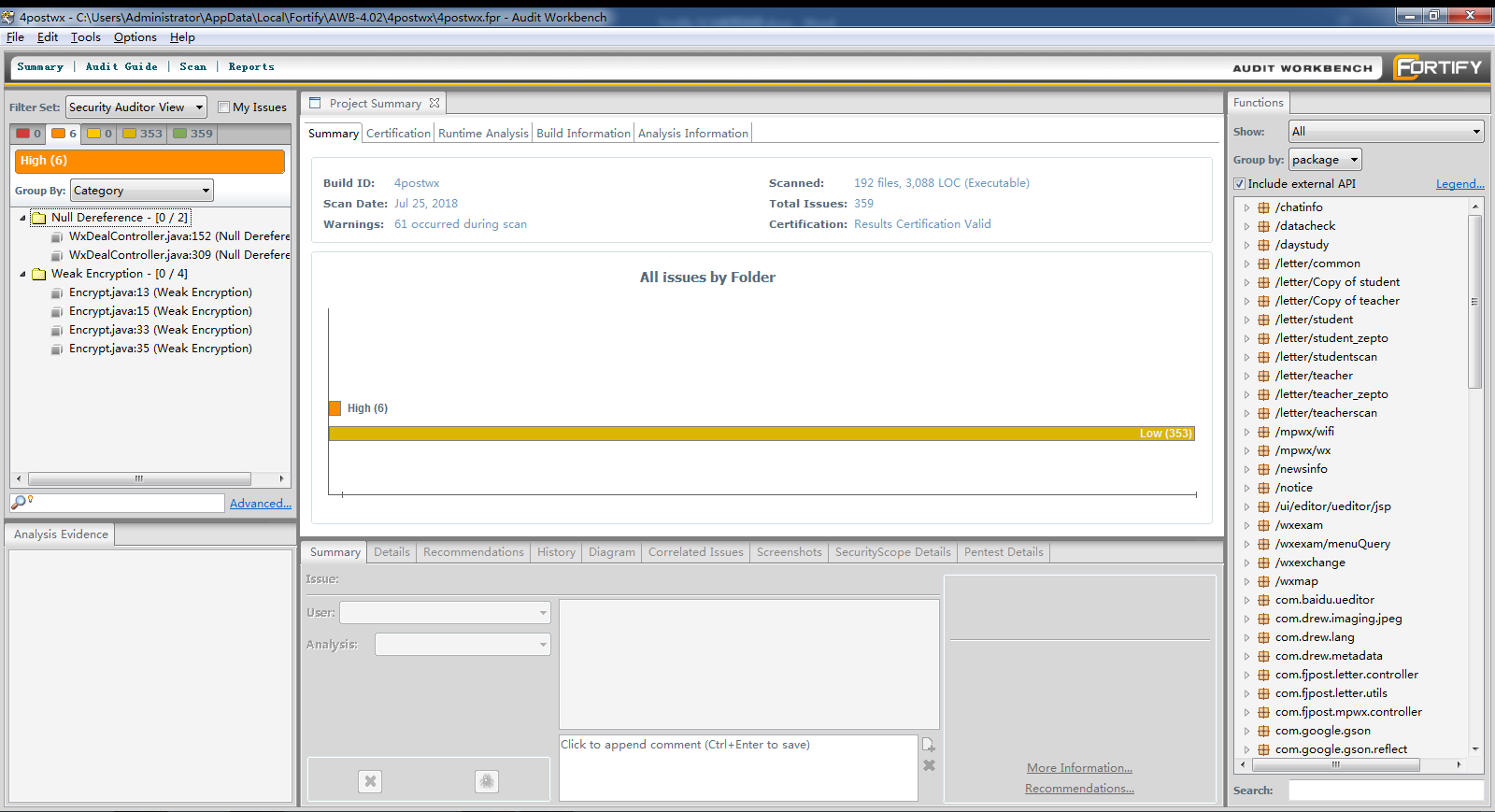
# Fortify SCA

## Fortify SCA简介

Fortify SCA 是一个静态的、白盒的软件源代码安全测试工具。它通过内置的五大主要分析引擎：数据流、语义、结构、控制流、配置流等对应用软件的源代码进行静态的分析，分析的过程中与它特有的软件安全漏洞规则集进行全面地匹配、查找，从而将源代码中存在的安全漏洞扫描出来，并给予整理报告。扫描的结果中不但包括详细的安全漏洞的信息，还会有相关的安全知识的说明，以及修复意见的提供。



Foritfy SCA 首先通过调用语言的编译器或者解释器把前端的语言代码（如JAVA，C/C++源代码）转换成一种中间媒体文件NST（Normal Syntax Tree）将其源代码之间的调用关系，执行环境，上下文等分析清楚。然后再通过上述的五大分析引擎从五个切面来分析这个NST，匹配所有规则库中的漏洞特征，一旦发现漏洞就抓取出来。最后形成包含详细漏洞信息的FPR结果文件，用AWB（Audit Work Bench）打开查看。

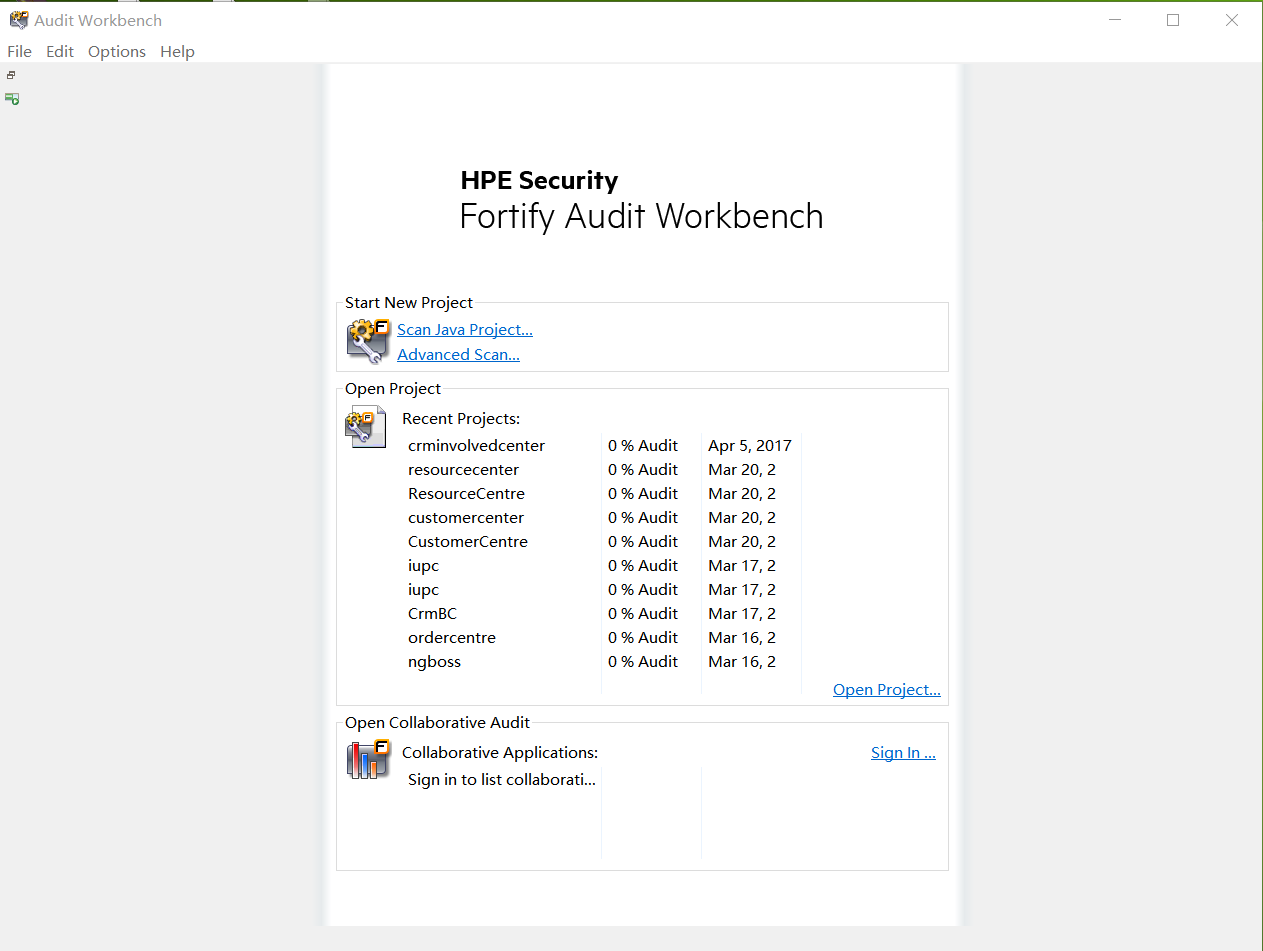


## 扫描项目

### Fortify界面扫描方式

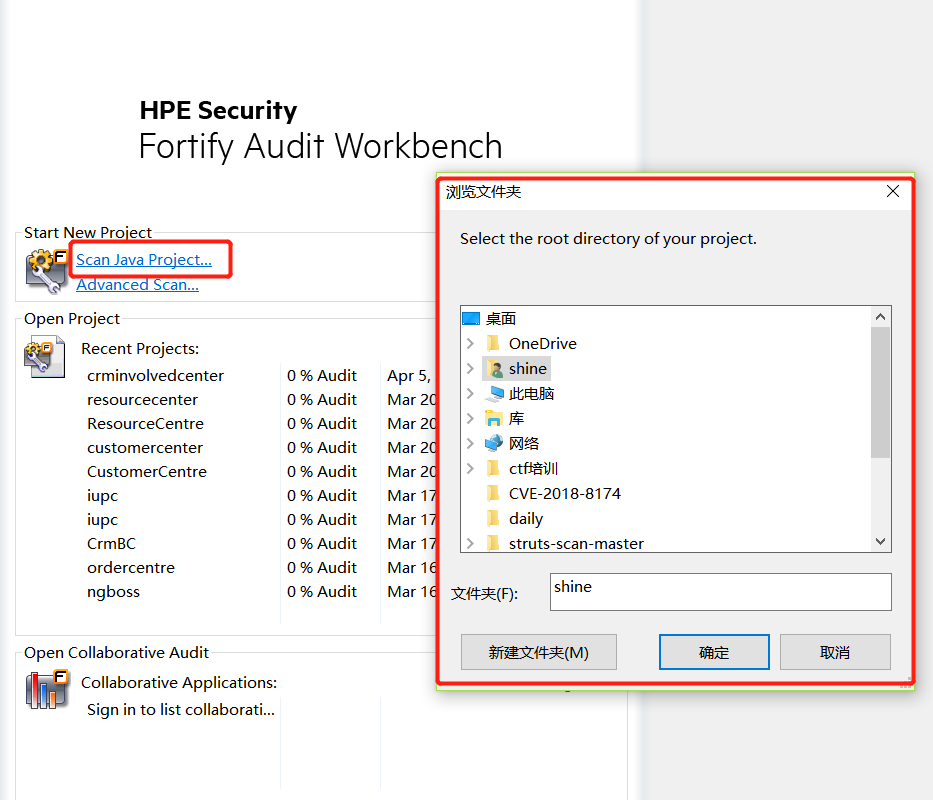
#### 启动Fortify界面

1.Fortify安装完成后，除开始菜单中的应用打开外，还可运行${Fortify\_Home}/bin/auditworkbench.cmd启动Fortify界面。

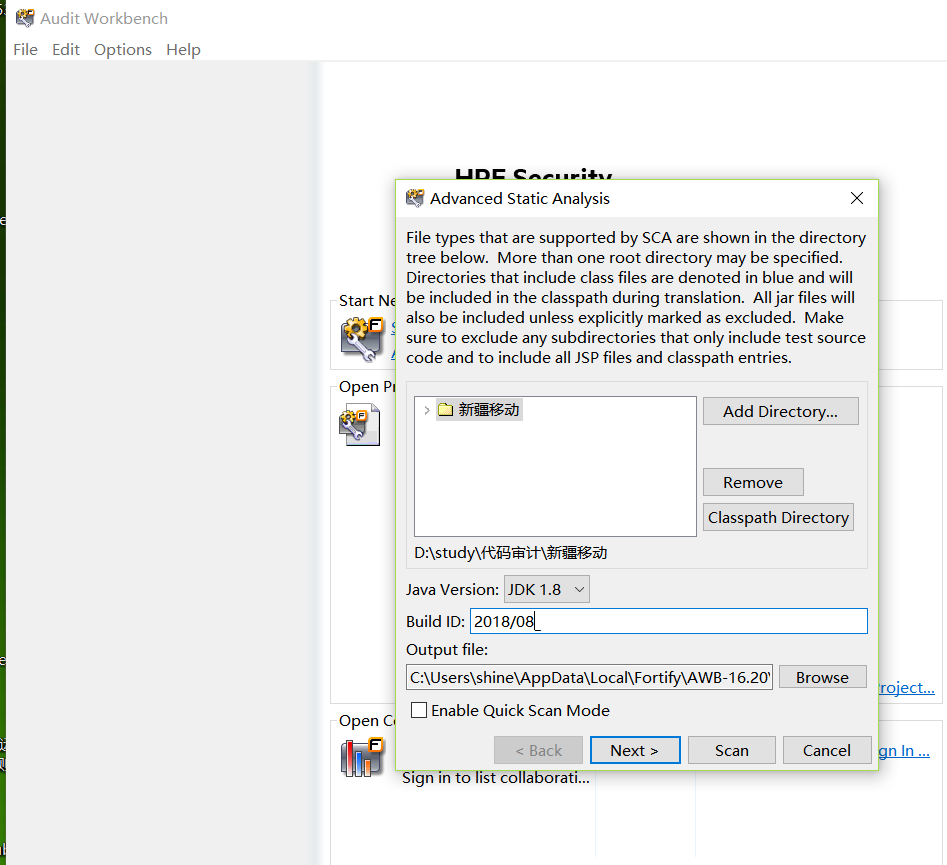


2.在“Start New Projects（新项目）”中，单击 Scan Java Project（扫描 Java 项目）。

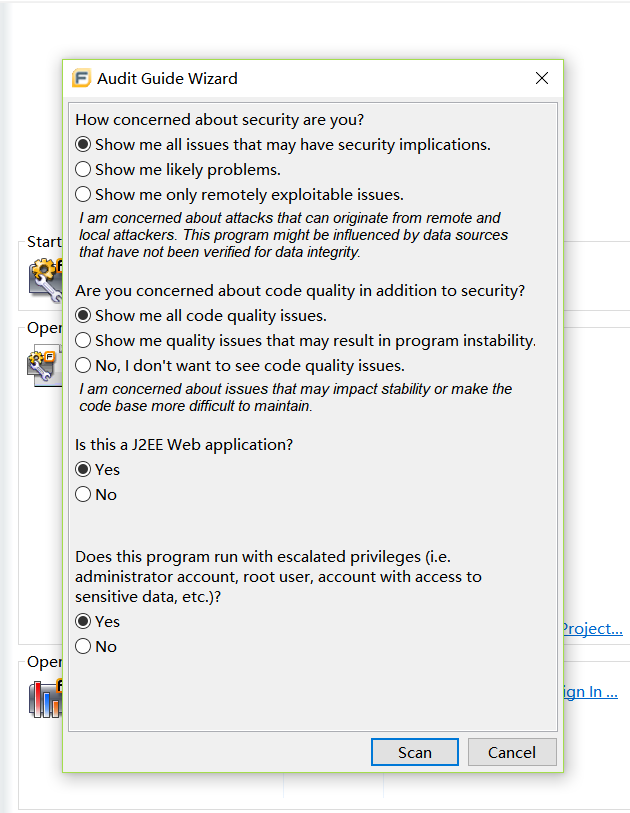
3.系统会显示 Browse for Folder（浏览文件夹）对话框。



4.选择包含所有需要分析的源代码的文件夹，然后单击 OK（确定）。注意：Fortify SCA 将 Build ID 设为文件夹名称。



5.系统会显示 AuditGuide Wizard（AuditGuide 向导）。

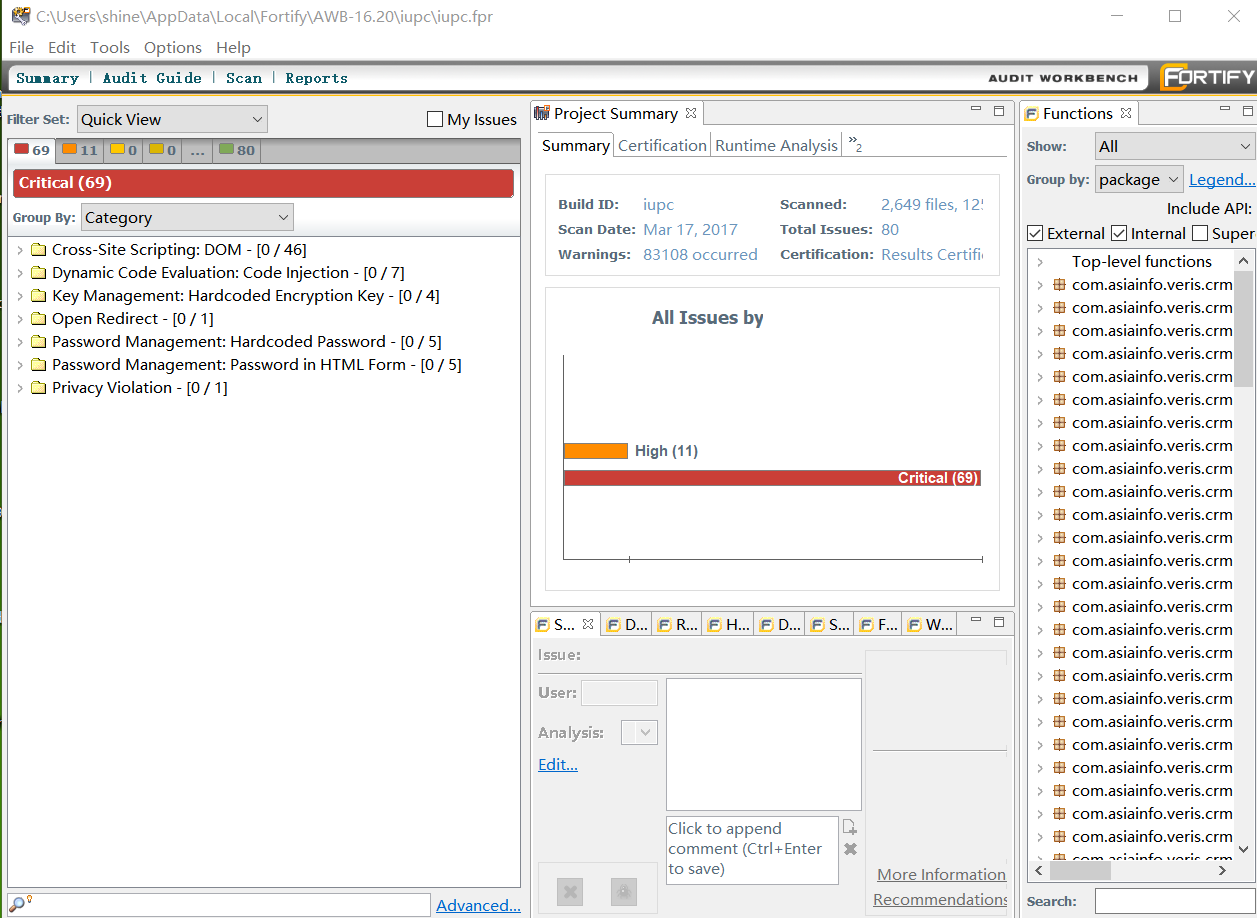


6.为需要审计的问题类型选择相关设置，然后单击Scan（运行扫描）。

7.如果 Fortify SCA 在扫描源代码时遇到任何问题，即会显示 Warning（警告）对话框。

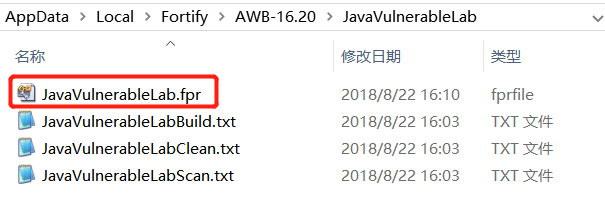
8.单击 OK（确定） 继续。

9.Fortify SCA 会分析源代码。完成该过程之后，Audit Workbench 会显示 FPR 文件。如下图所示：

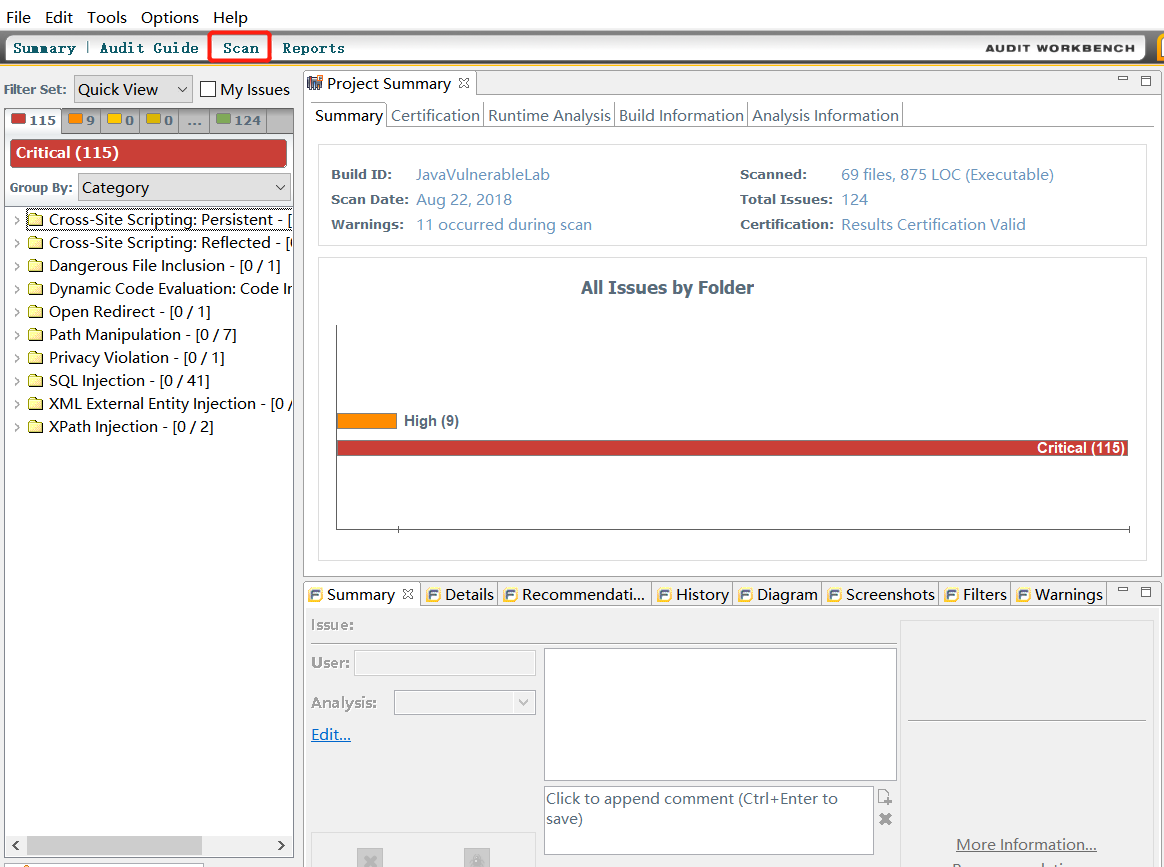


#### 重新扫描项目

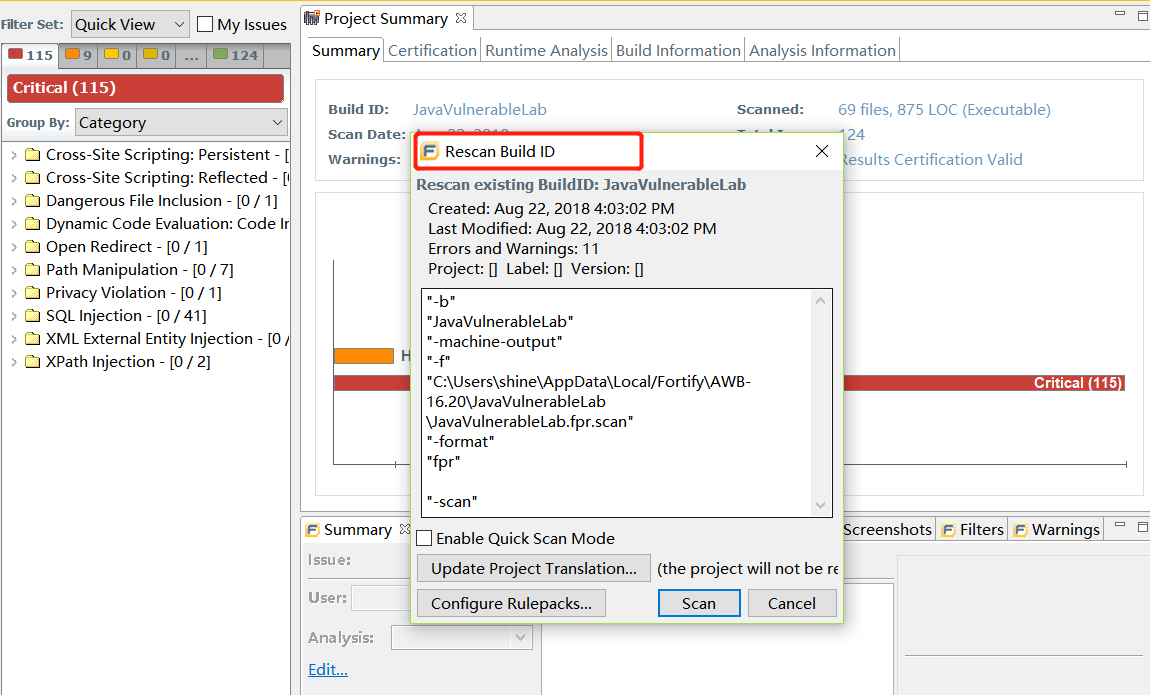
1.打开要扫描应用的FPR文件，双击打开。



2.单击 Scan（扫描）。



3.系统会显示 Rescan BuildID（重新扫描 Build ID）对话框。



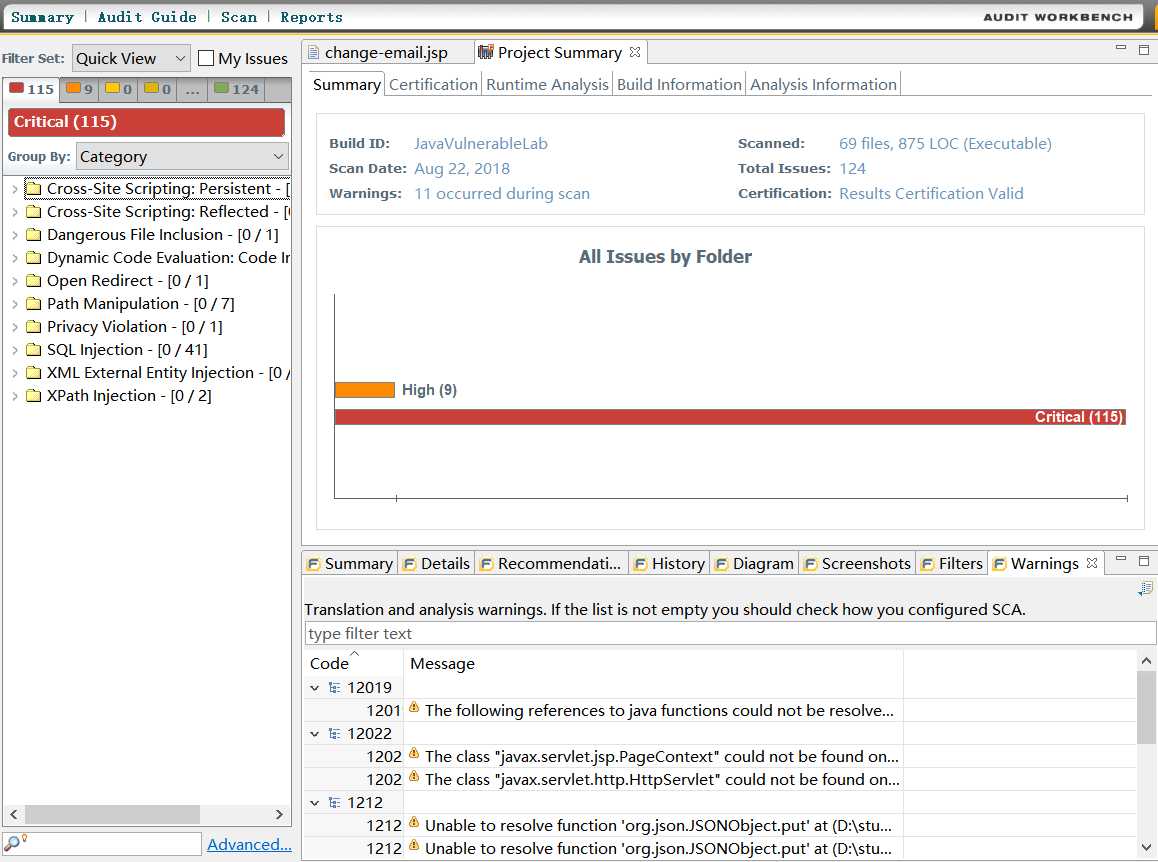
4.（可选）单击 Update Project Translation（更新项目转换）以重新转换该项目。

5.（可选）修改 Fortify SCA 扫描阶段的命令行选项。

6.（可选）单击 Manage Rulepacks（管理规则包）可以更改用来分析该项目的规则包：通过选中或清除复选框来添加或删除规则包。要使用未列出的自定义规则包，单击 Add Custom Rulepack（添加自定义规则包），浏览到规则包文件并选中，然后单击 OK（确定）。

7.单击Scan（扫描）。

8.扫描完成后，将显示 FPR。



### 命令行扫描方式

Fortify扫描工具UI客户端扫描性能不如命令行，同时，用UI扫描的时候还会卡在某个进度的，而cmd下出现上述问题的几率会减少。命令行扫描分为三步，进入到源代码目录下，然后翻译为中间文件，最后扫描中间文件，以下就以java源代码为例。如下所示：

#### 切换到源代码目录下

要扫描的网站源代码目录为：D:\study\代码审\JavaVulnerableLab\JavaVulnerableLab-master，cmd切换到该目录下：

CD D:\study\代码审计\JavaVulnerableLab\JavaVulnerableLab-master

#### 把源代码进行翻译为中间文件

命令格式如下：

D:\tools\fortify\fotify.license\bin\sourceanalyzer.exe -b test111 src\\*\*\\*

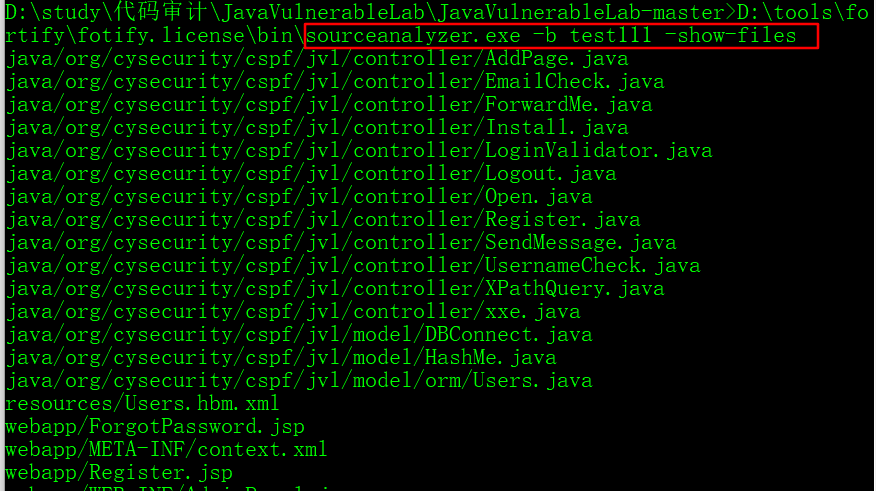
参数说明：

-b 后面的参数值是build\_id，也就是扫描id，自定义的。

Src\\*\*\\* 指的是src目录下所有的文件

编译完之后查看生成的中间文件，命令如下：

D:\tools\fortify\fotify.license\bin\sourceanalyzer.exe -b test111 -show-files



#### 扫描并生成结果

命令如下：

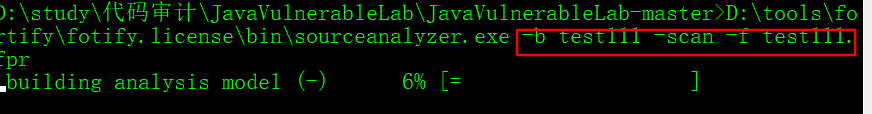
sourceanalyzer.exe -b test111 -scan -f test111.fpr

参数说明：

-scan：扫描

-f： 生成的fpr文件名，自定义

结果如下图所示：



#### 重新扫描

sourceanalyzer -b <build\_id> -clean 清除上一次扫描缓存

然后进行重新的扫描。

## 源代码安全审计

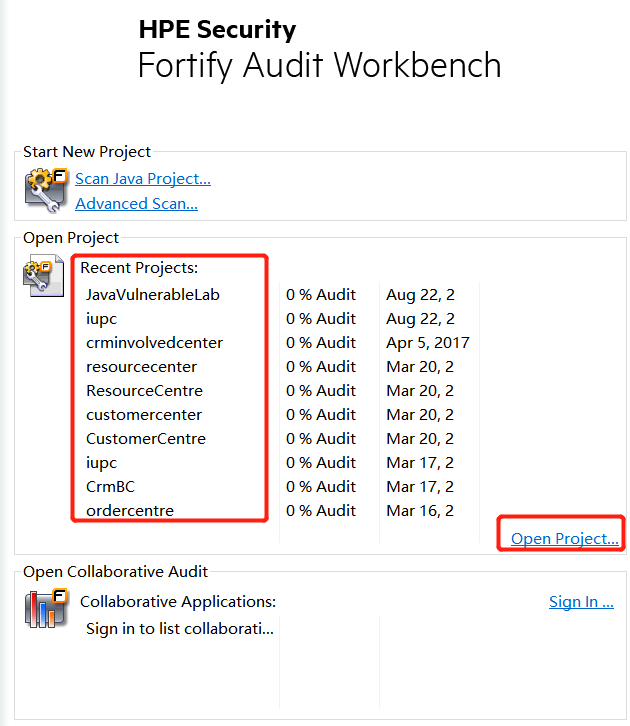
### 查看审计结果

启动 Audit Workbench。在 Windows 中：${Fortify\_Home}/bin/auditworkbench.cmd启动Fortify界面。在其他平台上：在终端或命令提示界面中，运行 auditworkbench。在闪现屏幕中会显示 Audit Workbench，然后会显示启动页。

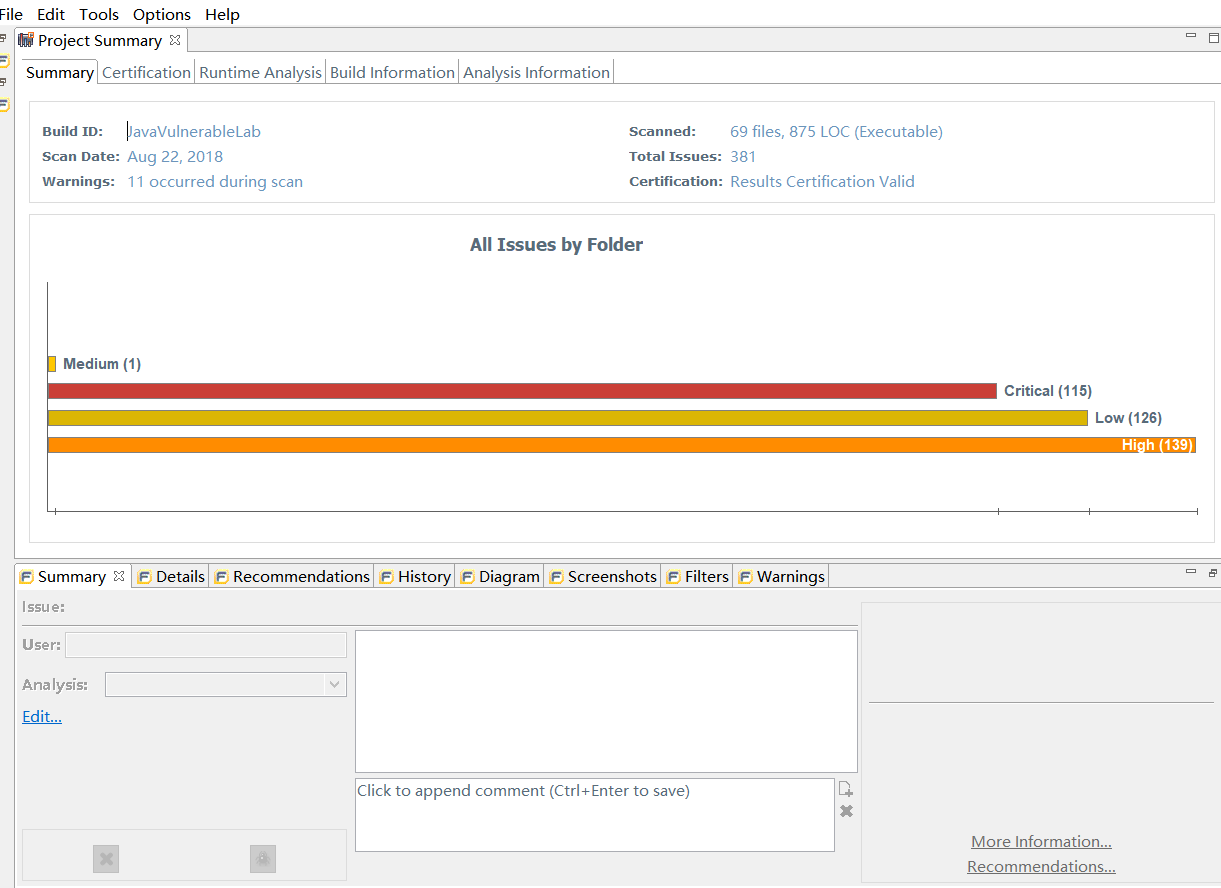
1.加载审计结果

选择 Open Project（打开项目）。

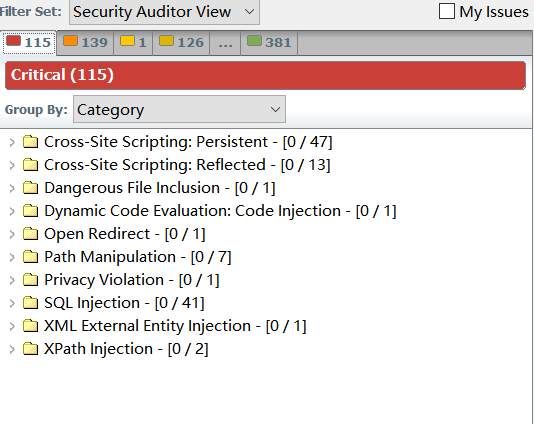
选择文件并单击 Open（打开）。



2.查看 Project Summary（项目汇总）面板中显示的信息。Project Summary（项目汇总）提供已加载的分析结果的总体概述，如已分析的项目的大小。



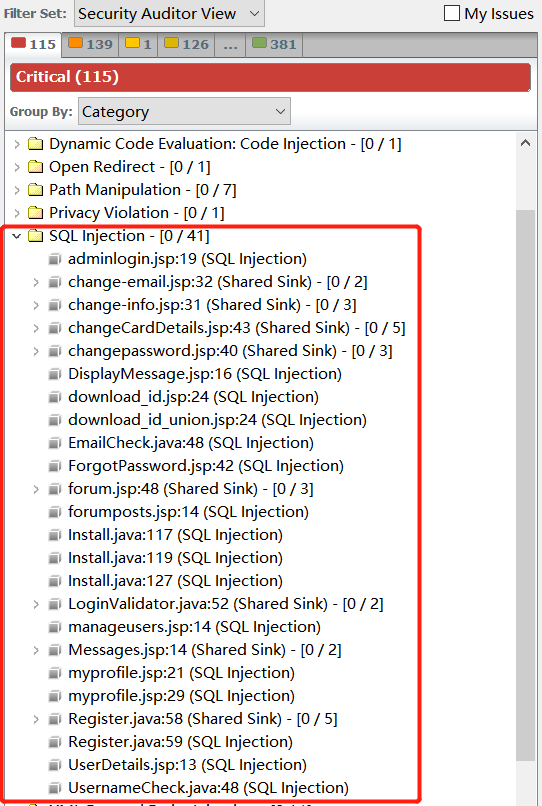
3.使用 Issues（问题）面板查看问题是如何分组和定义优先级的。



4.Critical（严重）文件夹包含了优先级最高的问题，接下来是High（高危），然后是 Medium（中危），最后是Low（低危）它包含了优先级最低的问题。All（全部）文件夹包含了所有 Critical（严重）、High（高危） Medium（中危）和Low（低危）问题。也可以按不同的标准对每个列表中的问题进行分组。本示例中根据 Category（风险类别）进行分组。

5.单击Critical（严重）、High（高危） Medium（中危）和Low（低危）可查看是如何对问题进行分组的。

6.单击 Hot（严重），然后在导航树中展开一个类别来查看单个问题，这里以 SQL Injection 为例。



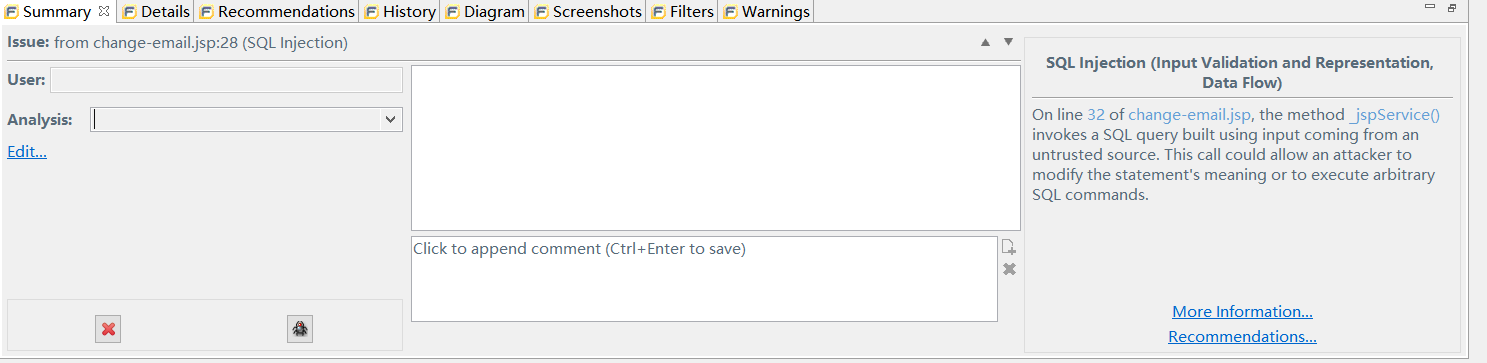
7.查看某个问题摘要。

在 Issues（问题）面板中找到并选中以下 SQL injection 问题：Change-email.jsp:32

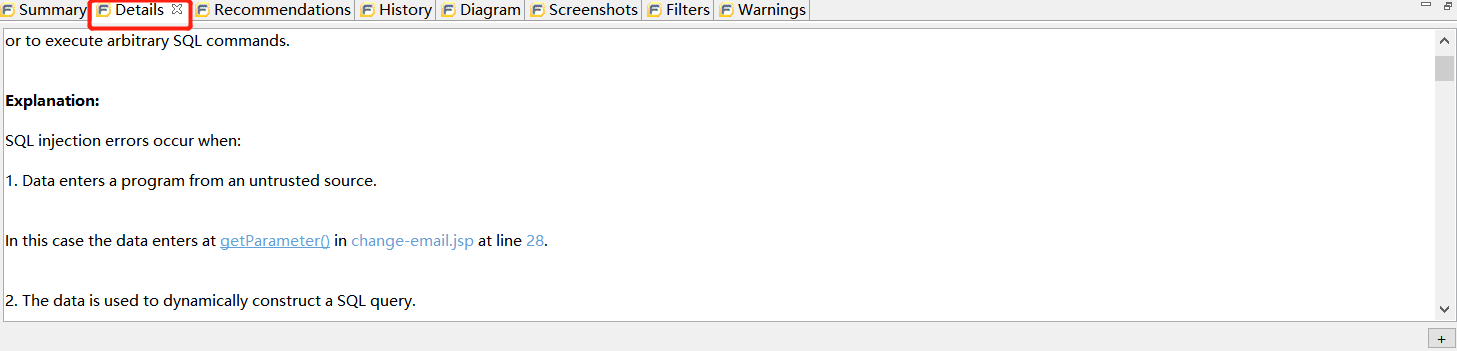
通过这个摘要我们查看源代码，可以明显看到获取的email和id参数直接带入到数据库进行update操作。存在注入。



在“Issue Auditing（问题审计）”面板的 Summary（总结）选项卡中，仔细阅读审计员对该问题的注释，并注意审计员针对这一问题所选择的分析选项。还要注意 Category（类别）字段表明这一问题所代表的漏洞类别，同时包括的内容还有该漏洞所属的漏洞系列以及发现这个漏洞的分析器的详细信息。

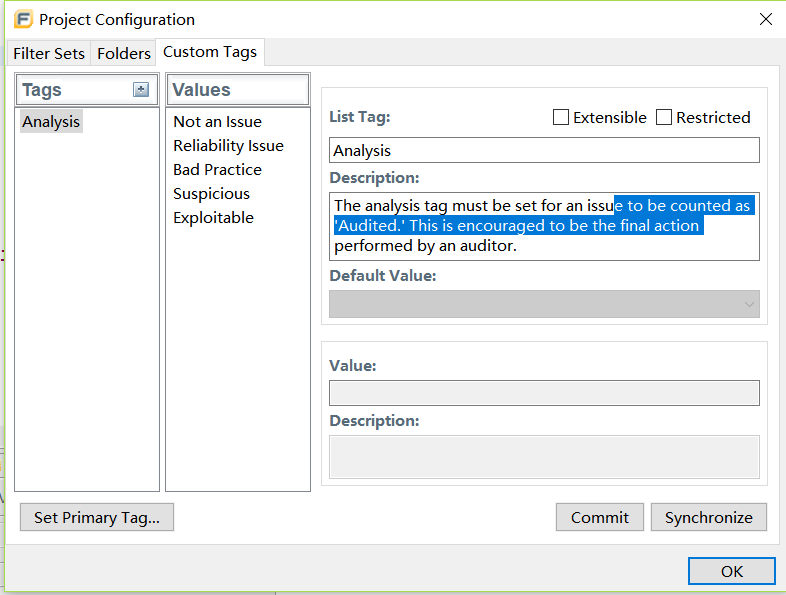


选择 Details（详细信息）选项卡来读取有关此类漏洞的更多信息。

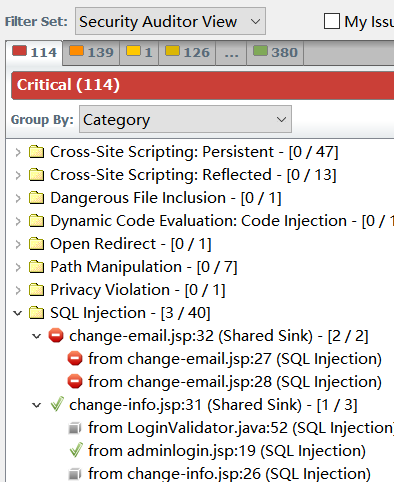


8.对某个问题进行标记

以上面的注入为例，在 summary面板中找到edit标签，单击，打开：



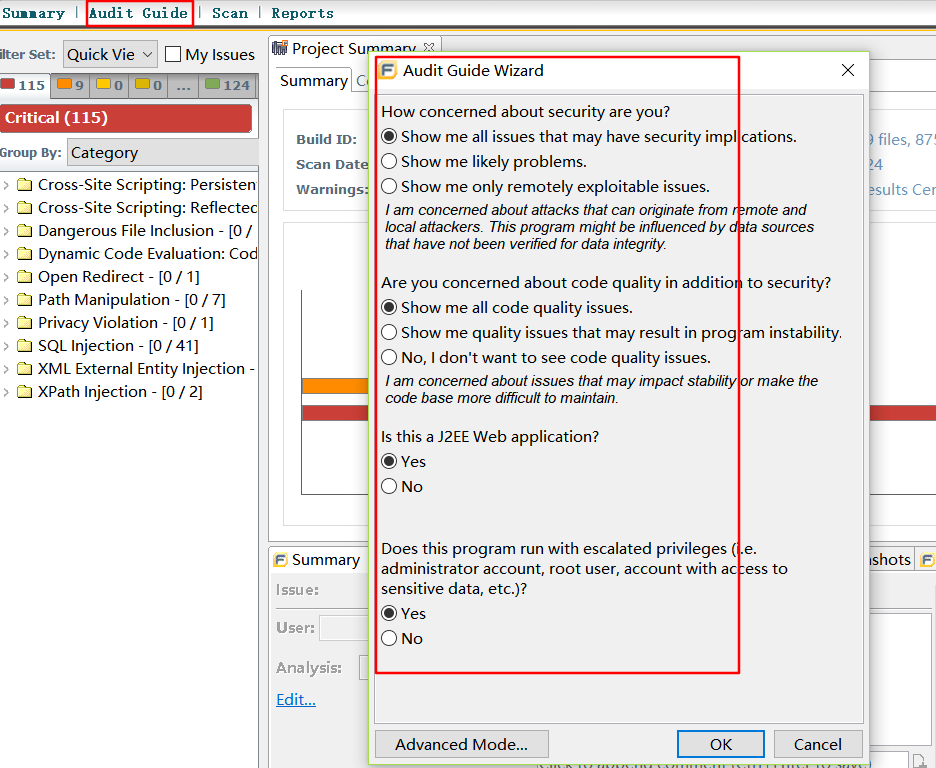
这里可以把该该问题进行分析然后标记，分析完标记的结果有五种，分别为Not an Issue(不是一个问题)，Rediability Issue（确认的问题），Bad Practice（不好的行为），Suspicious（可疑的问题），Exploitable（可利用的）。标记完之后发现左侧图标会发生改变，如下图:



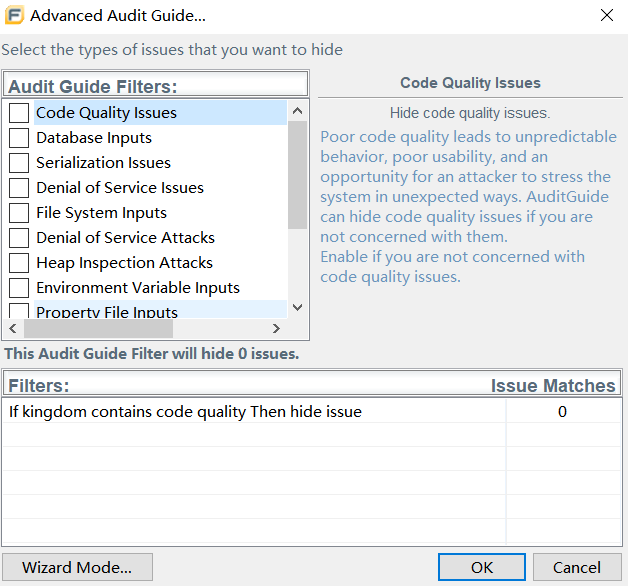
### 调整审计结果

调整过程使不同的审计员可以根据各自的需要来配置 Fortify SCA。步骤如下：

1.选择 Tools（工具）菜单中的 Audit Guide（审计指南），或单击 Audit Guide（审计指南）按钮。系统会显示“Audit Guide（审计指南）”设置面板。

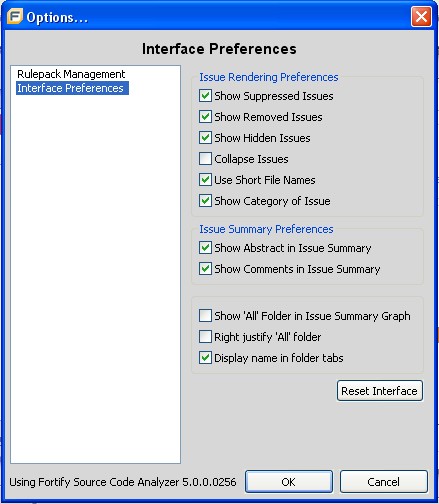


2.根据需求选择适当的选项，”Advanced Mode”中还有一些设置，如下图所示：



### 自定义问题视图

可以从 Options（选项）- Options（选项）- Interface Preferences（界面首选项）菜单中设置以下选项，以自定义“Issues（问题）”视图：



* Show Entry Points（显示入口点）：显示用户输入进入程序的所有位置。
* Show Suppressed Items（显示废除项）：显示您已废除的所有项。默认情况下，会禁用该选项。
* Show Removed Items（显示删除项）：如果您已经执行了“（Import New SCA Analysis 导入新 SCA 分析）”，则会显示自上次分析以来已删除的所有项。
* Use Short Filenames（使用短文件名）：仅通过文件名而非相对路径来参照“Issues（问题）”视图中的问题。默认情况下，已启用该选项。
* Show View（显示视图）：允许显示 Analysis Trace（分析跟踪）、Issues（问题）以及 Summary（总结）面板。它还提供了 Other（其他）选项，其中包含标准的Eclipse 导航特性。

### 审计一个结果

1. 检查总结信息并对问题进行注释。

“Issues Auditing（问题审计）”面板的 Summary（总结）和 Details（详细信息）选项卡既提供了有关在 Issues（问题）面板中选中的问题的附加信息，同时允许用户对该问题添加注释。在面板右边的粗体文本提供了问题的类别，该类别跟在括号之中，根据漏洞所属类型进行划分，并提供了检测出该问题的具体分析器。在该信息之下，有对该漏洞的简单描述，More Information（更多信息）链接可打开 Details（详细信息）选项卡，“Recommendation（建议）”链接可打开“Recommendation（建议）”选项卡。

选择 Details（详细信息）选项卡，获取有关该问题的详细描述，包括带有示例的问题的解释。选择 Recommendations（建议）选项卡，获取有关如何解决问题的建议，在审计类似问题上的有用提示，以及作为深入了解该主题的参考。

在底部的 Comment（注释）字段中，填写问题的简要描述以及为什么认为它容易受到攻击。完成对该问题的注释后，按 Enter 键。该注释会被贴上时间戳，且被添加至注释列表中。注释应包括在问题审计期间所得到的信息，如引发问题的数据类型（如用户输入），执行验证时该怎么办，和问题在上下文中出现的重要意义（如果这个代码出现在生产环境中会很严重）等。此外，当检查问题或该问题生成的报告时，还要考虑 Details（详细信息）选项卡中的部分相关描述。

在 Summary（总结）选项卡的左边有一个下拉菜单和两个按钮。下拉菜单用来记录审计的结果。从“Analysis（分析）”菜单中选择审计状态。

如果在审计问题后，经判断它实际上是安全的，可以通过单击 Suppress（废除）按钮来废除该问题。然而，如果问题需要由开发者来解决，可以单击 File Bug（报告错误）按钮来自动生成一个需要进行配置的错误报告。

2. 保存工作。

在“File（文件）”菜单中选择 Save Project（保存项目），将项目保存。Audit Workbench 项目文件存储最初的问题信息和在审计中所做的更改。

## 导出报告

### 打开报告模板

1.在 Audit Workbench 工具栏中单击 Reports（报告）。

2.系统会显示“Generate Reports（生成报告）”窗口。

3.从“Report（报告）”的下拉列表中选择报告模板。

4.“Generate Report（生成报告）”窗口中会显示报告模板设置。

### 生成报告

选择了所需报告模板和报告设置后，即可生成报告以查看其结果。您可以将报告结果另存为 PDF、RTF 和 XML 文件。

步骤如下：

1.在 Audit Workbench 工具栏中单击 Reports（报告）。

2.系统会显示“Generate Reports（生成报告）”窗口。

3.从“Report（报告）”下拉菜单中选择报告模板。

4.另外，还可以更改报告一节的设置。

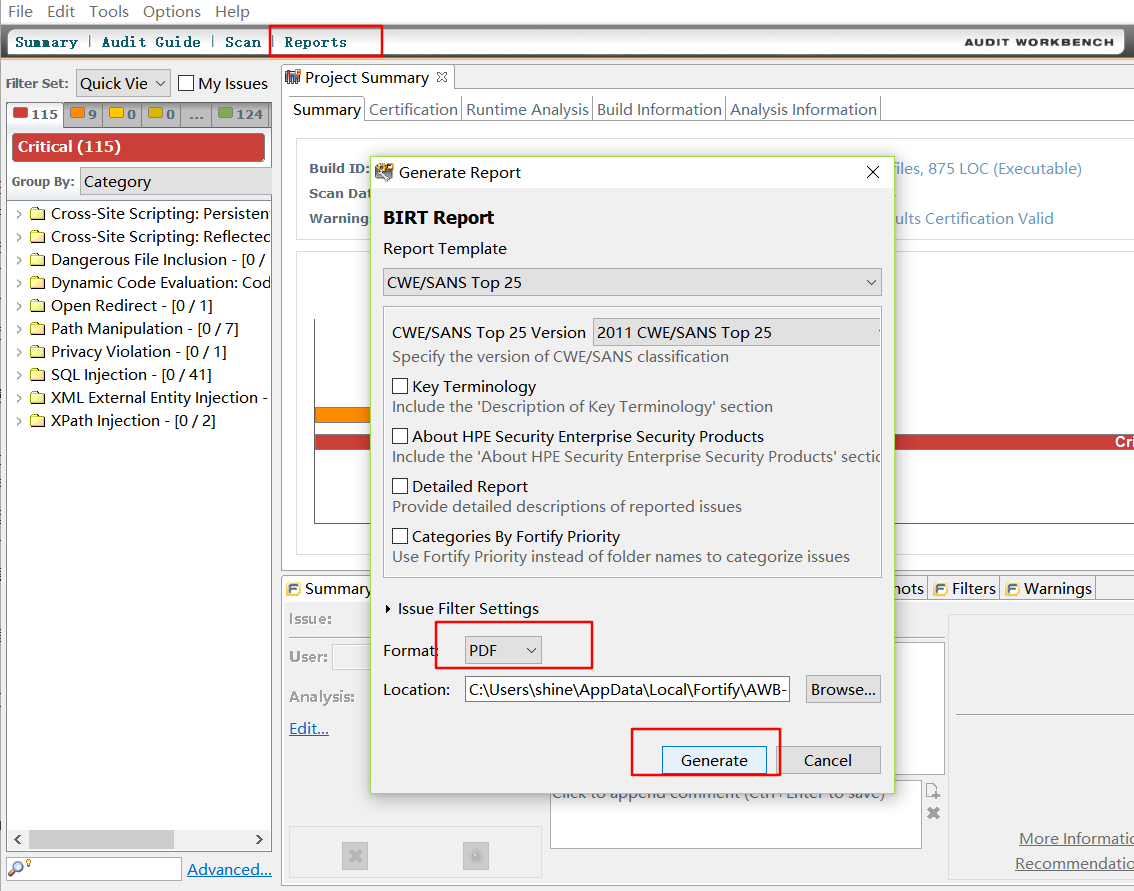
5.单击 Print Report（打印报告）。会显示保存报告的对话框。

6.指定文件名称和保存报告的位置。

7.选择报告的文件类型。提供的选项有 PDF、RTF 或 XML。

8.单击 Save（保存）。

已生成报告并另存为一个文件。如下图：



## 自定义规则指南

### 新建自定义规则包

本节介绍了如何新建一个自定义规则包。新规则包是空的。当您创建好规则包后，可以编写自定义规则并将这些规则添加到规则包中。注意16.20版本是没有这个选项的。

1.创建一个自定义规则包：

2.在“Audit（审计）”窗口中，选择 File（文件）- Open Custom Rules Editor（打开自定义规则编辑器）。

3.屏幕上将显示 Fortify Audit Workbench Custom Rules Editor（Fortify Audit Workbench 自定义规则编辑器）窗口。

4.选择 File（文件）- New Rule Pack（新规则包）。

5.屏幕上将显示 Create a New Folder（新建规则包）对话框。

6.在文件名栏，输入规则包的名称。

7.单击 Save（保存）。

在“Table View（表格视图）”中将显示一个新的自定义规则包。

### 打开“Custom Rulepack（自定义规则包）”

可以在不打开新规则的情况下将它添加到现有的规则包中。

打开一个现有的自定义规则包：

1.在“Audit（审计）”窗口中，选择 File（文件）- Open Custom Rules Editor（打开自定义规则编辑器）。

2.屏幕上将显示 Fortify Audit Workbench Custom Rules Editor（Fortify Audit Workbench 自定义规则编辑器）窗口。

3.选择 File（文件）- Open Rulepack（打开规则包）。

4.屏幕上将显示 Choose Rules File（选择规则文件）对话框。

5.选择所需的规则包，然后单击 Open（打开）。在“Table View（表格视图）”中将打开一个新的自定义规则包。

### 配置规则包的细节

规则包信息可以定义自定义规则。默认的规则包名称是文件名。Audit Workbench、Fortify SCA Team Server 和 Fortify Manager 可以通过显示“Rulepack/Name（规则/名称）”元素帮助用户定义特定的规则包。设置自定义规则包的细节：

1.在“Audit（审计）”窗口中，选择 File（文件） - Open Rulepack（打开规则包）。

2.屏幕上将显示 Choose Rules File（选择规则文件）对话框。

3.选择所需的文件，然后单击 Open（打开）。在 Table View（表格视图）中将显示一个新的自定义规则。

4.单击 XML View（XML 视图）选项卡。

5.设置规则包细节，如下所示：

6.在 Name（名称）元素中的 CDATA 的括号内输入定义规则包的名称。

7.在 Description（描述）元素中的 CDATA 的括号内输入对规则包的简短描述。

8.单击 File（文件）- Save Rulepack（保存规则包）。

### 安装自定义规则包

本节介绍了如何安装一个自定义规则包副本。默认情况下，Fortify SCA 通过使用自定义规则目录中的所有可以找得到的自定义规则包来分析代码。自定义规则目录的位置：

<install\_directory>/Core/config/ssm.properties

创建一个自定义规则包：

1.在“Audit（审计）”窗口中，选择 Options（选项）- Options（选项）。

2.屏幕上将显示 Options（选项）窗口。

3.选择 Rulepack Management（规则包管理）。屏幕上将显示 Ruelpack Management（规则包管理）面板。

4.单击 Import Rulepacks（导入规则包）。

5.屏幕上将显示 Select Rulepack（选择规则包）对话框。

6.选择所需的自定义规则包，然后单击 Open（打开）。

7.自定义规则包的副本会保存在自定义规则目录中。

### 编辑规则

1.选择需要编辑的规则。

2.单击 Details（详细信息）。

3.屏幕上会显示“Rule Details Editor（规则细节编辑器）”。

4.更改“Rule Parameters and Rule Description（规则参数和规则描述）”面板所需要的信息。

5.单击 OK（确定）。

### 删除规则

1.选择需要删除的规则。

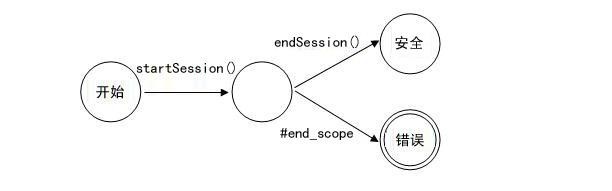
2.单击 Delete（删除）。屏幕上会出现一个对话框，询问您是否删除。

3.单击 Yes（是）删除该规则，或单击 No（否）取消删除。

### 编写自定义规则举例及标签解释

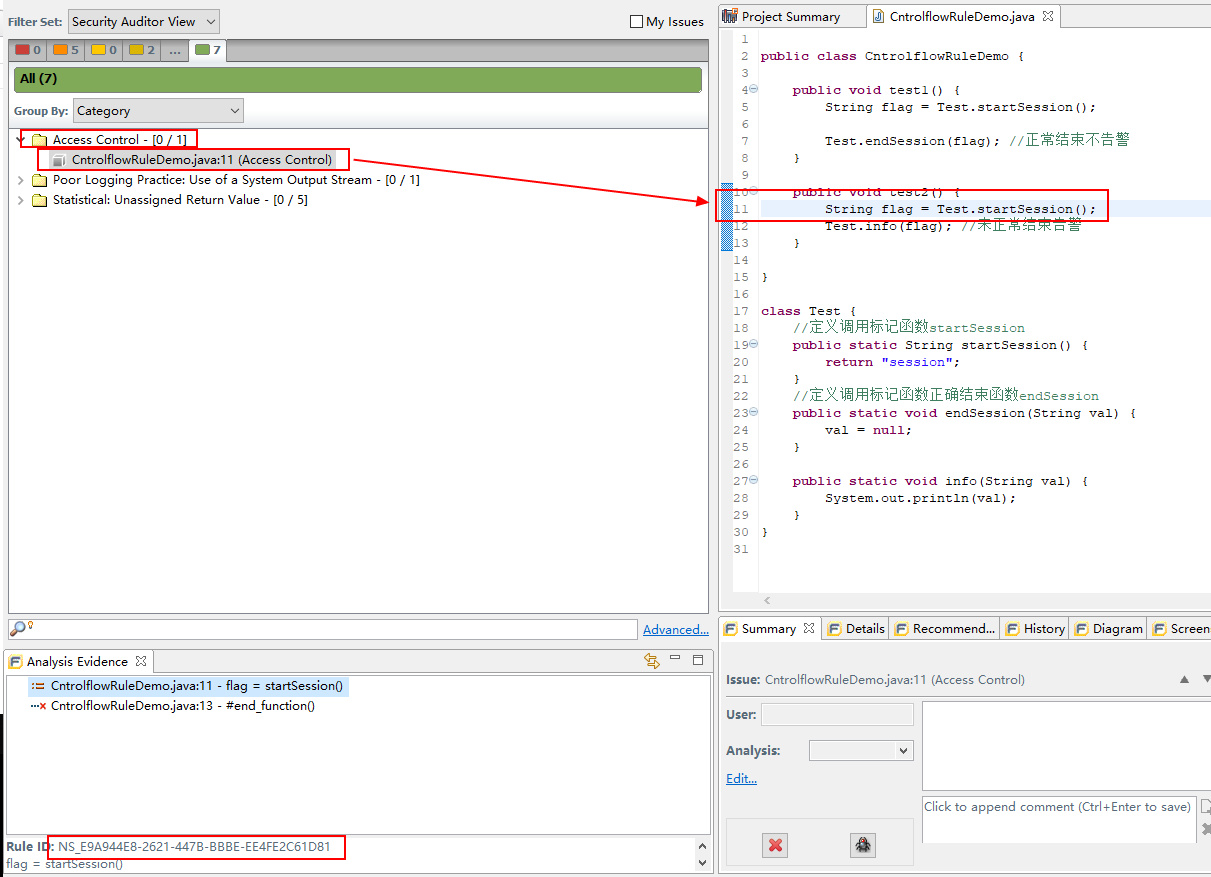
#### 控制流规则编写举例

将所有定义在包 .\*中的类 Test中的函数 startSession() 的调用的所有进行案例标记，在当前函数结束之前是不被定义在包 .\* 中的类 Test中的函数 endSession ()的调用所允许的。下面这个图表显示了整个流程。



|  |
| --- |
| <ControlflowRule formatVersion="16.20" language="java">  <RuleID>NS\_E9A944E8-2621-447B-BBBE-EE4FE2C61D81</RuleID>  <Description/>  <VulnCategory>Access Control</VulnCategory>  <DefaultSeverity>5.0</DefaultSeverity>  <FunctionIdentifier id="start">  <NamespaceName><Pattern>.\*</Pattern></NamespaceName>  <ClassName><Value>Test</Value></ClassName>  <FunctionName><Value>startSession</Value></FunctionName>  <ApplyTo implements="false" overrides="true"  extends="true"/>  </FunctionIdentifier>  <FunctionIdentifier id="end">  <NamespaceName><Pattern>.\*</Pattern></NamespaceName>  <ClassName><Value>Test</Value></ClassName>  <FunctionName><Value>endSession</Value></FunctionName>  </FunctionIdentifier>  <Definition><![CDATA[  state start startState; ——定义起始状态  state startSession; ——定义开始状态  state endSession; ——定义结束状态  state errorState(error); ——定义错误状态  var st;  startState -> startSession{st=$start()}  —— 起始->开始  startSession -> endSession{$end(st)}  ——开始->结束  startSession -> errorState{#end\_function()}  ——开始->错误，错误告警  ]]></Definition>  </ControlflowRule> |

扫描结果：

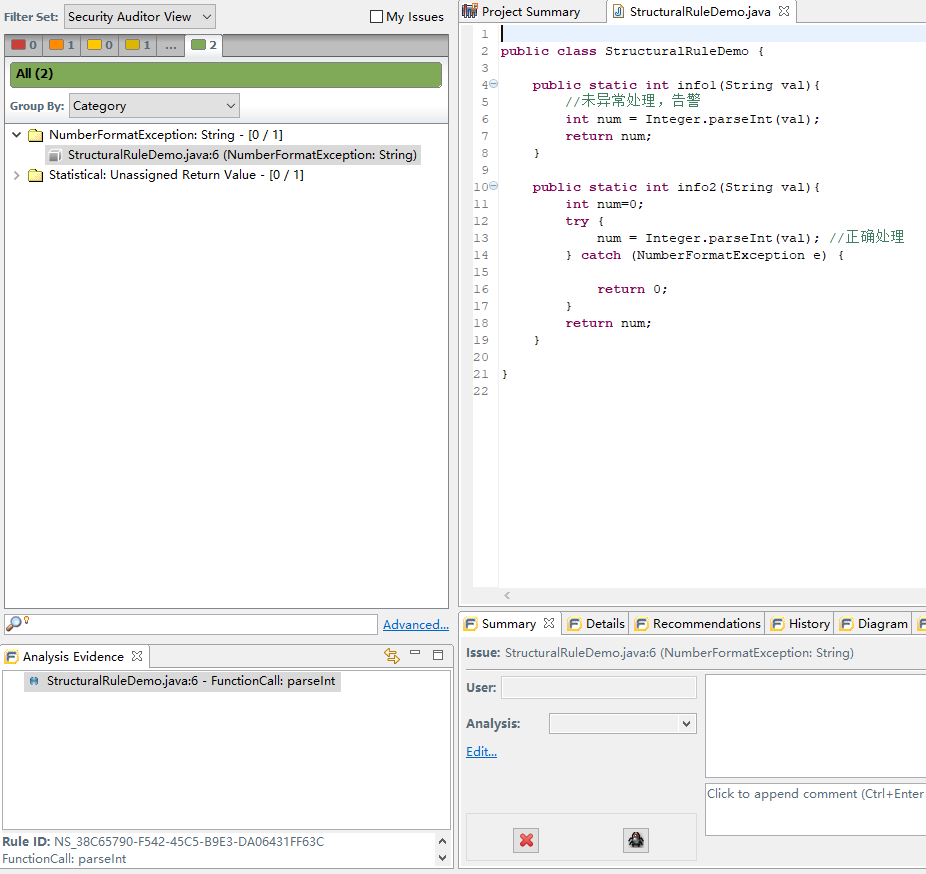


#### 结构规则举例

在 JSP 页面中标记所有的 HTML 格式的注释的实例。

|  |
| --- |
| <StructuralRule formatVersion="16.20" language="java">  <RuleID>NS\_38C65790-F542-45C5-B9E3-DA06431FF63F</RuleID>  <VulnKingdom></VulnKingdom>  <VulnCategory>NumberFormatException</VulnCategory>  <VulnSubcategory>String</VulnSubcategory>  <DefaultSeverity>3.0</DefaultSeverity>  <Description/>  <Predicate><![CDATA[  FunctionCall fc : fc.function.name == "parseInt"  and fc.function.enclosingClass.supers contains  [Class c : c.name == "java.lang.Integer"] and (not in [TryBlock:]  or not in [TryBlock:catchBlocks contains  [exception.type.name == "java.lang.NumberFormatException"]])  ]]></Predicate>——定义所有函数，某一个函数名=="parseInt"，并且类结构== “java.lang.Integer“ 并且不在Try块中或者Try块中不包含，异常类型名称=="java.lang.NumberFormatException"，进行告警  </StructuralRule> |

效果如下图所示：



### 验证自定义规则包

使用新的自定义规则扫描源代码：

1.创建简单的源文件，其中的代码与其文件夹中的规则匹配。

2.打开 Audit Workbench，单击 Advanced Scan（高级扫描）。

3.屏幕上会显示 Browse for Root Directory（浏览根目录）窗口。

4.选择包含您源文件的文件夹，然后单击 OK（确定）。屏幕上会显示 Commandline Builder（命令行构建器） 对话框。

5.如果是 Java 项目，选择根文件夹，然后单击 Classpath Directory（类路径目录）。

6.单击 Next（下一步）。

7.Commandline Builder: Stages of Fortify SCA Analysis（命令行构建器：Fortify SCA Analysis 操作步骤）窗口会显示在屏幕上。

9.单击 Manage Rulepacks（管理规则包）。

10.如果未安装测试自定义规则包，单击 Add Custom Rulepack（添加自定义规则包），然后安装规则包。

11.清除所有安全编码规则包并自定义规则包（测试包除外）。

注意：在测试清晰的规则时，使用安全编码规则包和自定义规则包进行扫描。

12.单击 OK（确定）。

13.单击 Run Scan（运行扫描）。

14.FPR 文件会显示在“Audit（审计）”窗口中。

# Findsecbugs

## 工具介绍

### SpotBugs

SpotBugs是一个使用静态分析来查找Java代码中的错误的程序。SpotBugs需要运行JRE（或JDK）1.8.0或更高版本。但是，它可以分析为任何版本的Java编译的程序，从1.0到1.9。

### Find Sec Bugs

SpotBugs插件，用于Java Web应用程序的安全审核。

特征：

1.131种错误模式

它可以检测131种不同的漏洞类型，其中包含超过811个唯一的API签名。

2.支持框架和库

涵盖流行的框架，包括Spring-MVC，Struts，Tapestry等等。

3.与IDE集成

插件可用于Eclipse、IntelliJ、Android Studio、NetBeans。Ant 和Maven可以使用命令行集成。

4.持续集成

可以与Jenkins和 SonarQube等系统一起使用。

5.OWASP TOP 10和CWE报道

通过引用OWASP Top 10和CWE，为每种错误模式提供了广泛的参考。

## SpotBugs使用方法

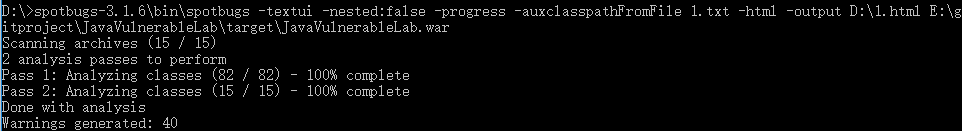
### SpotBugs命令行使用方法

在SpotBugs二进制包中/bin目录下，使用-textui参数，执行spotbugs命令运行命令行界面：

|  |
| --- |
| spotbugs [general options] -textui [command line options...] [jar/zip/class files, directories...] |

举例：

|  |
| --- |
| spotbugs -textui -nested:false -auxclasspathFromFile <依赖包列表文件> -html -output <输出HTML报告的文件路径> <要扫描的目录>  cmd>D:\spotbugs-3.1.6\bin\spotbugs -textui -nested:false -progress -auxclasspathFromFile C:\Users\nsfocus\Desktop\1.txt -html -output D:\1.html E:\gitproject\JavaVulnerableLab\target\JavaVulnerableLab.war |



**命令行配置选项：**

|  |  |
| --- | --- |
| **参数** | **说明** |
| -jvmArgs args | 指定要传递给JVM的参数。 |
| -maxHeap size | 指定以兆字节为单位的最大Java堆大小。默认值是768。分析非常大的程序或库可能需要更多的内存。 |
| -javahome <dir> | 指定JRE的位置（Java运行时环境） |
| -project <project> | 指定要分析的项目。指定的项目文件应该是使用GUI界面时创建的，它通常会以扩展名.fb或.fbp结束。 |
| -home <home directory> | 指定SpotBugs主目录 |
| -pluginList <jar1[;jar2...]> | 指定要加载的插件Jar文件列表 |
| -debug | 打印一系列检测器运行，并将类分析为标准输出。用于排除意外分析失败。 |
| -effort[:min|less|default|more|max] | 设置分析精度，提高精度也会增加内存消耗和时间。 |
| -adjustExperimental | 低优先级的实验性Bug模式 |
| -workHard | 确保分析工作至少是“默认的” |
| -conserveSpace | 与-effort:min一样 |
| -showPlugins | 显示可用的检测器插件列表 |
| -userPrefs <filename> | 用户首选项文件，例如Eclipse项目中的/path/to/project/.settings/edu.umd.cs.findbugs.core.prefs |
| -timestampNow | 设置结果的时间戳为当前时间 |
| -quiet | 抑制错误消息 |
| -longBugCodes | 报告长错误代码 |
| -progress | 在终端窗口中显示进度 |
| -release <release name> | 设置分析应用程序的发布名称 |
| -experimental | 任何置信水平的报告，包括实验性错误模式 |
| -low | 报告所有bug |
| -medium | 报告中优先级和优先级错误。这是默认设置 |
| -high | 只报告高优先级的bug |
| -maxRank <rank> | 只报告指定错误级别的问题 |
| -dontCombineWarnings | 不要组合仅在行号上不同的警告 |
| -sortByClass | 按类名对报告的错误实例进行排序 |
| -xml[:withMessages] | 以XML格式生成错误报告。稍后可以在GUI中查看生成的XML数据。您还可以将此选项指定为-xml:withMessages;当使用此选项的变体时，XML输出将包含人类可读的消息，描述文件中包含的警告。以这种方式生成的XML文件很容易转换为报表 |
| -xdocs | 以xdoc XML格式生成bug报告，以便与Apache Maven一起使用 |
| -html[:stylesheet] | 生成HTML输出。默认情况下，SpotBugs将使用默认值。生成HTML的xsl XSLT样式表:您可以在spotbugs中找到这个文件。jar，或者在SpotBugs源代码或二进制发行版中。这个选项的变体包括-html:plain。xsl,html:花哨。xsl和html:fancy-hist.xsl。平原。xsl样式表不使用Javascript或DOM，而且可以更好地用于旧的web浏览器或打印。花哨的。xsl样式表使用DOM和Javascript进行导航，使用CSS进行可视化表示。fancy-hist。xsl幻想的演变。xsl样式表。它广泛使用了DOM和Javascript来动态过滤bug列表。如果您想指定您自己的XSLT样式表来执行到HTML的转换，请将选项指定为-html:myStylesheet（我的样式表）。xsl,myStylesheet。xsl是要使用的样式表的文件名。 |
| -emacs | 以Emacs格式生成bug报告 |
| -relaxed | 放松的报告模式。对于许多检测器，这个选项抑制了用于避免报告误报的启发式方法。产生更多的误报 |
| -train[:outputDir] | 保存训练数据（实验），输出的目录默认为当前目录 |
| -useTraining[:inputDir] | 使用训练数据（实验），输入的目录默认为当前目录 |
| -redoAnalysis <filename> | 使用先前分析中的配置进行重做分析 |
| -sourceInfo <filename> | 指定源信息文件（字段/类的行号） |
| -projectName <project name> | 项目描述名称 |
| -reanalyze <filename> | 在提供的文件中重新做分析 |
| -output <filename> | 在指定的文件中生成输出 |
| -nested[:true|false] | 此选项允许或禁止扫描在要分析的文件和目录列表中找到的嵌套jar和zip文件。默认情况下，可以扫描嵌套的jar/zip文件。要禁用它，在命令行参数中添加-nested:false |
| -bugCategories <cat1[,cat2...]> | 只报告给定类别中的bug |
| -onlyAnalyze <classes/packages> | 只分析给定的类和包，以.\*结束，表示包中的类，以.-结束，表示包前缀 |
| -excludeBugs <baseline bugs> | 排除在基线XML输出中也报告的错误 |
| -exclude <filter file> | 排除匹配给定过滤器的bug |
| -include <filter file> | 只报告与filterFile.xml指定的过滤器匹配的bug实例 |
| -applySuppression | 排除任何与fbp文件中加载的抑制过滤器匹配的bug |
| -visitors <v1[,v2...]> | 只运行指定的检测器 |
| -omitVisitors <v1[,v2...]> | 省略指定的检测器 |
| -chooseVisitors <+v1,-v2,...> | 选择性地启用/禁用检测器 |
| -choosePlugins <+p1,-p2,...> | 选择性地启用/禁用插件 |
| -adjustPriority <v1=(raise|lower)[,...]> | 提高/降低对特定检测器的警告优先级 |
| -auxclasspath <classpath> | 设置辅助类路径以进行分析。此类路径应包括所有jar文件和目录，其中包含属于所分析程序一部分的类，但您不希望分析错误。 |
| -auxclasspathFromInput | 从标准输入读取辅助类路径以进行分析，每行向辅助类路径添加新条目以进行分析 |
| -auxclasspathFromFile <filepath> | 从文件中读取辅助类路径以进行分析，每行向辅助类路径添加新条目以进行分析 |

**项目配置选项**

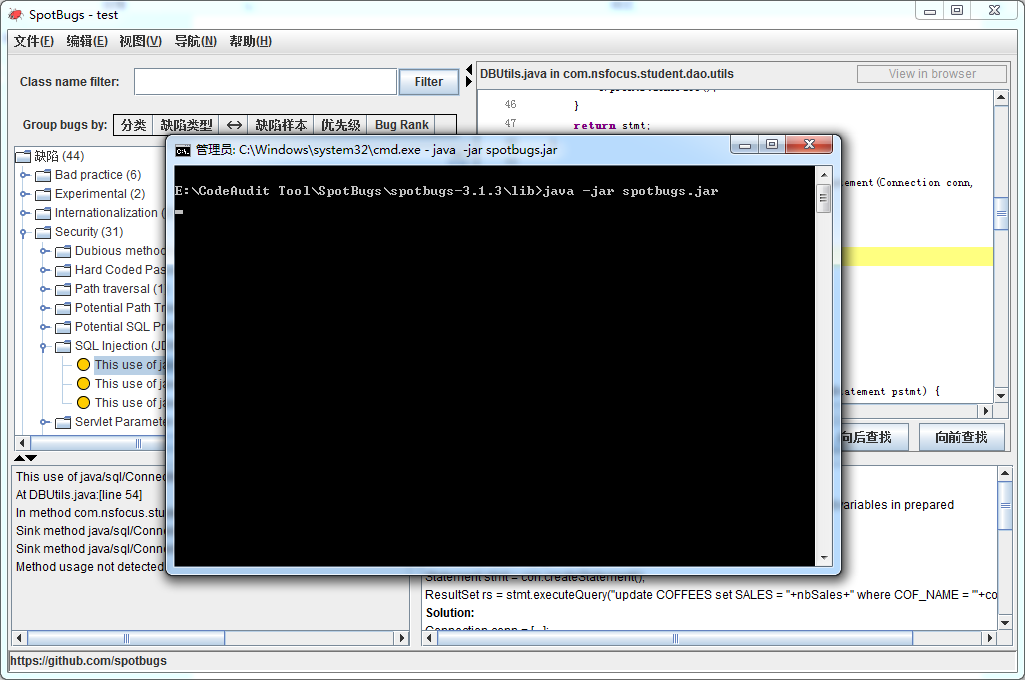
|  |  |
| --- | --- |
| **参数** | **说明** |
| -sourcepath <source path> | 设置分析类的源路径 |
| -exitcode | 设置进程的退出代码 |
| -noClassOk | 如果未指定类，则输出空警告文件 |
| -xargs | 从标准输入而不是命令行获取类文件/ jar文件列表 |
| -analyzeFromFile <filepath> | 从指定的文件中获取类/jar文件列表 |
| -bugReporters <name,name2,-name3> | 显式启用/禁用bug报告装饰器 |
| -printConfiguration | 打印配置并退出，无需运行分析 |
| -version | 打印版本，检查更新和退出，不运行分析 |

### SpotBugs GUI界面使用

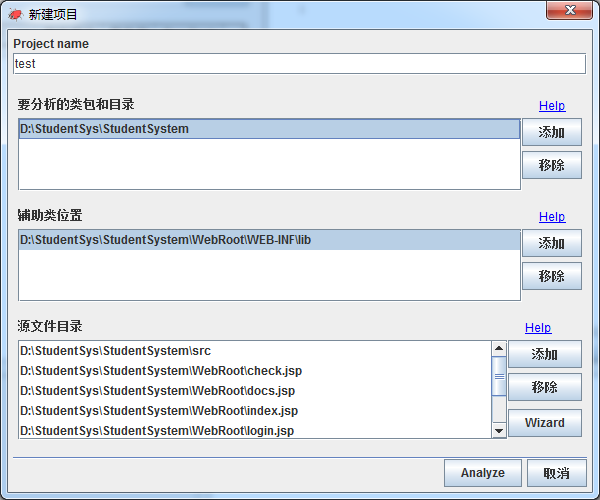
#### SpotBugs代码分析

1.将下载到的二进制版本解压到指定路径，如：E:\CodeAudit Tool\SpotBugs\spotbugs-3.1.3；

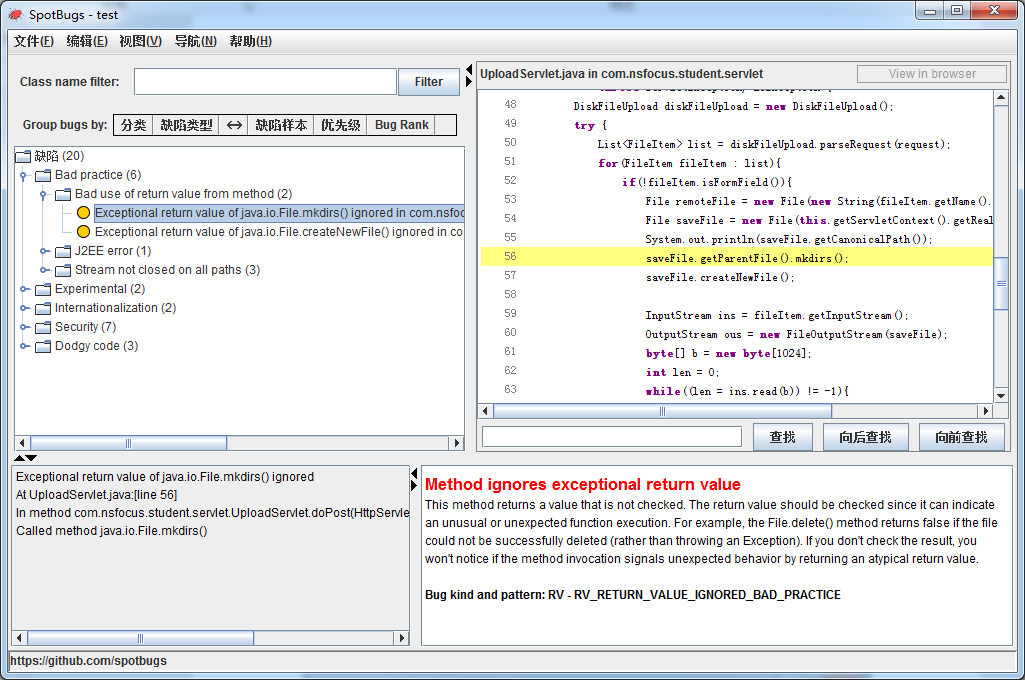
2.SpotBugs路径下的lib为主要程序文件，包括SpotBugs功能实现及其依赖包。进入lib目录，使用java –jar spotbugs.jar命令启动SpotBugs。



3.点击“文件”→“新建”，新建项目，指定待分析的字节码路径、依赖包路径、源码路径。



分析结果如下图所示。

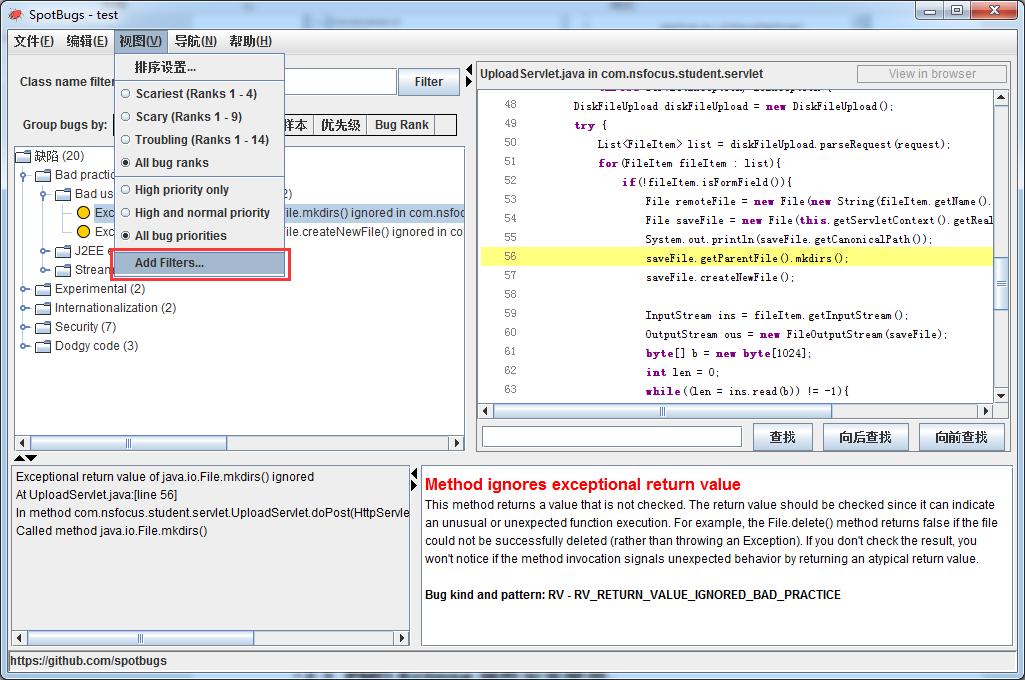


#### Find-Sec-Bugs插件安装

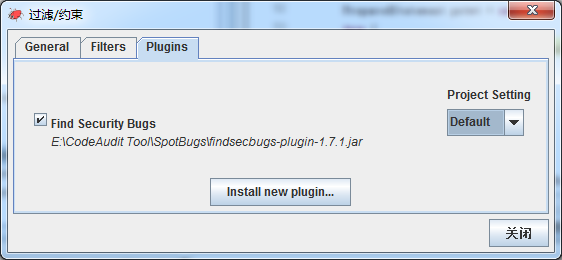
Find-Sec-Bugs插件访问地址：http://find-sec-bugs.github.io/

下载的文件为：findsecbugs-plugin-1.7.1.jar，可在SpotBugs界面中安装。过程如下：

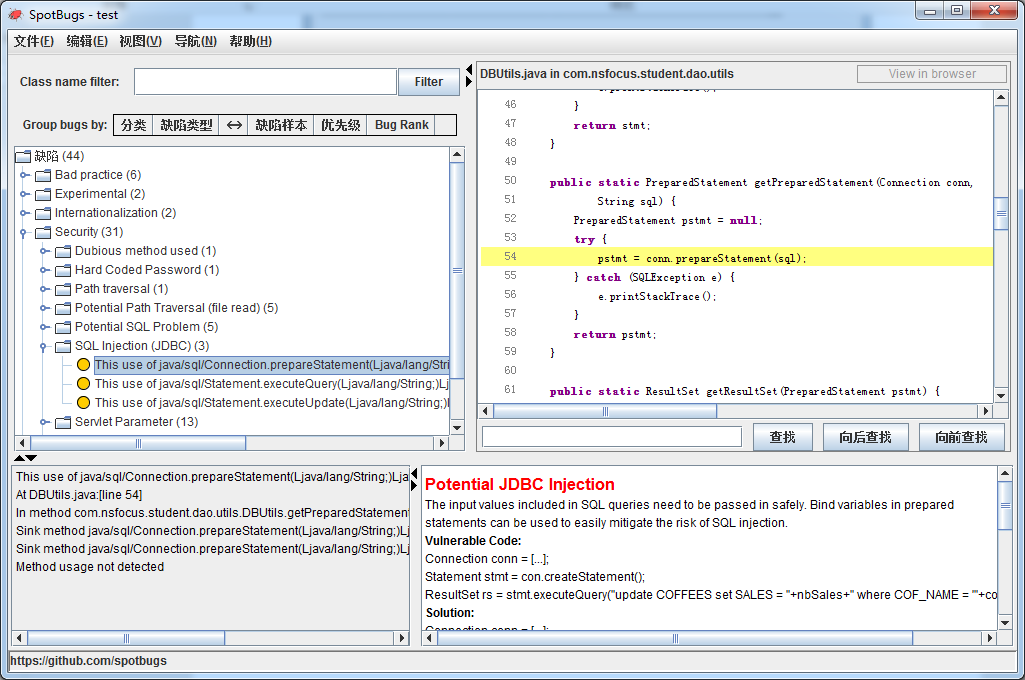
1.点击“视图”→“Add filters”，进入插件管理界面。



2.在Plugins选项卡中，点击“Install new plugin”按钮，选择下载的findsecbugs-plugin-1.7.1.jar文件。如下图：



3.重新对项目扫描，可看到结果中增加了较多的Security结果项。



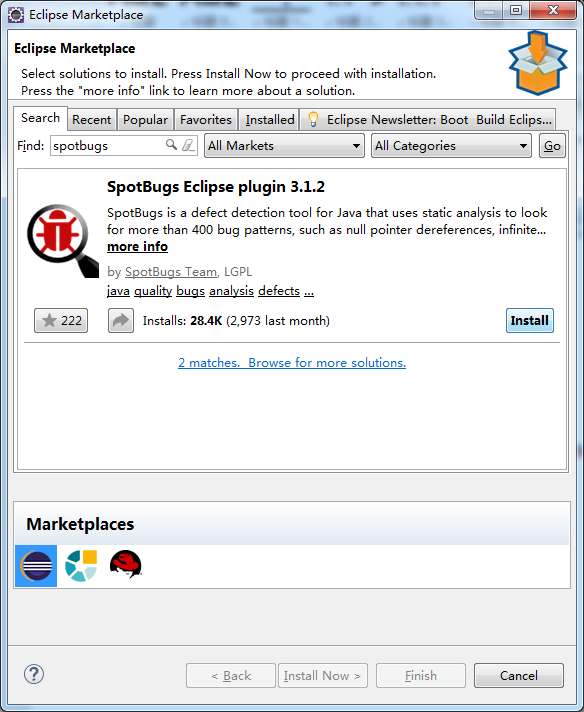
4.或将findsecbugs-plugin-1.8.0.jar直接复制的/%spotbugs解压目录%/plugin/文件夹下，spotbugs将自动调用该文件夹下的插件。

### SpotBugs Eclipse插件安装使用

#### SpotBugs插件安装

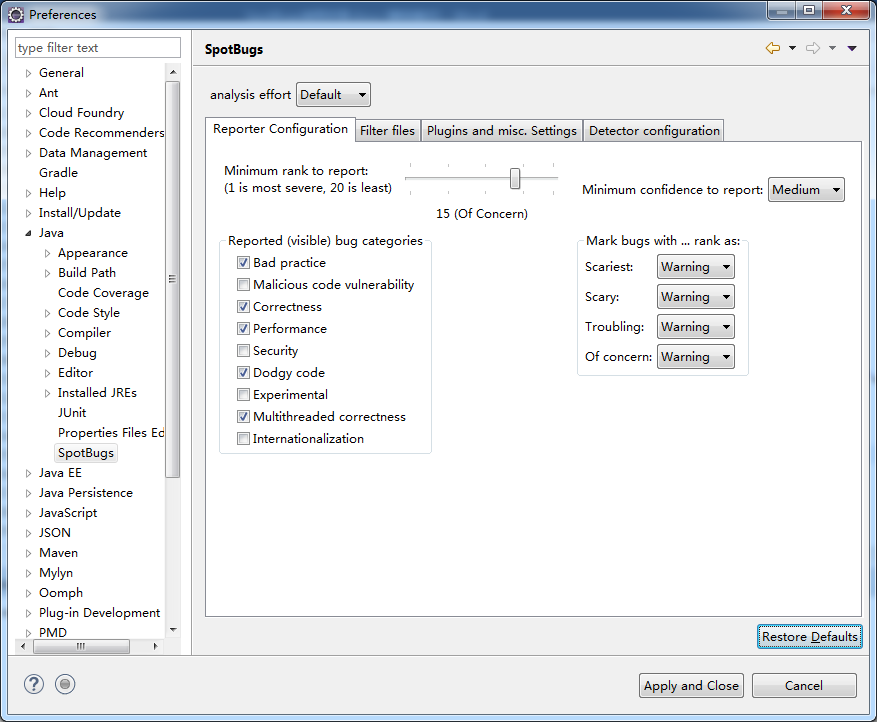
1.启动Eclipse；

2.选择Help→Eclipse Marketplace，在search界面搜索“spotbugs”。

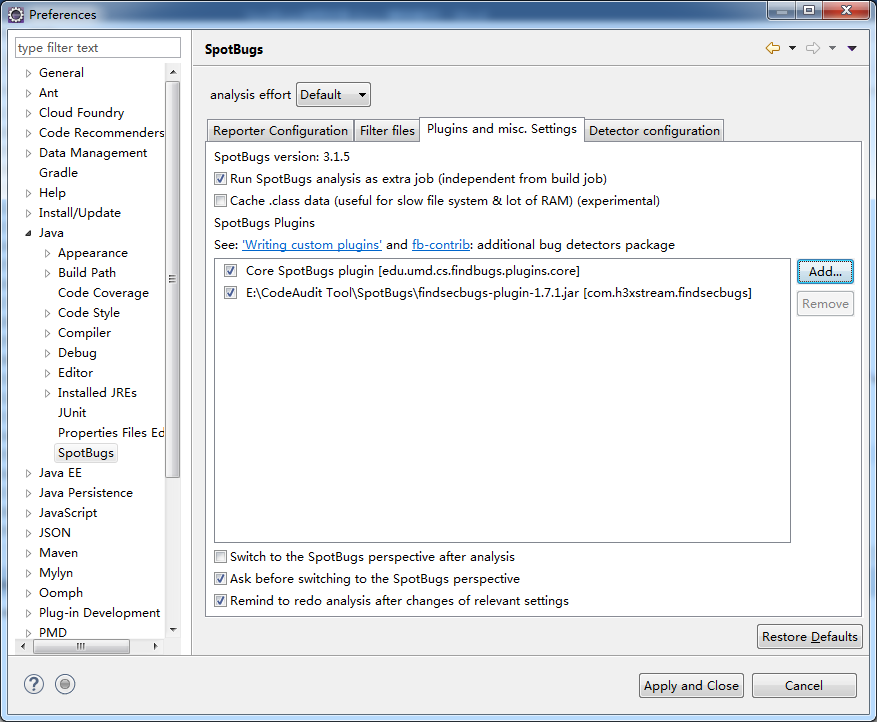


3.点击安装，安装成功后，将重启Eclipse；

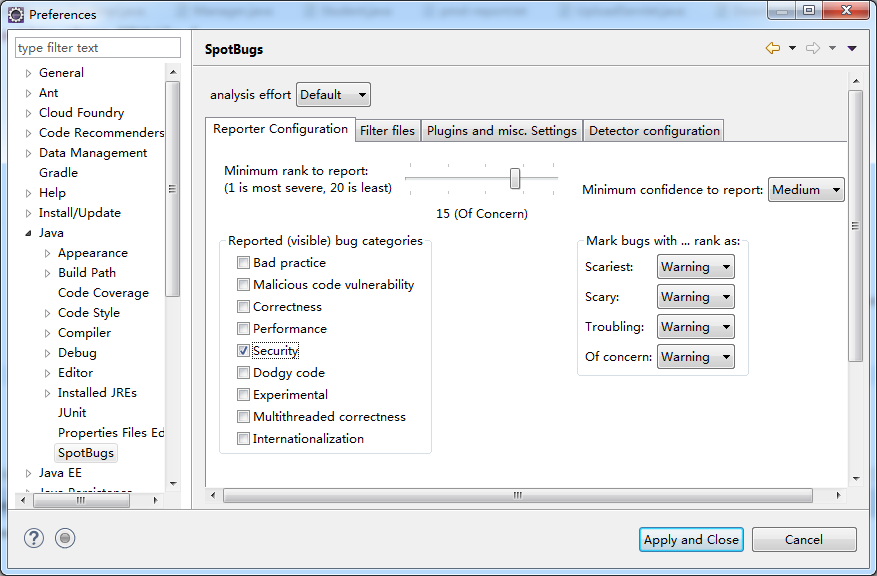
4.安装成功后，点击“window”“Preferences”菜单，在Java选项中，进入Spotbugs插件配置；如下图



5.在Plugins and misc， Settings选项卡中安装find-sec-bugs插件，如下图：

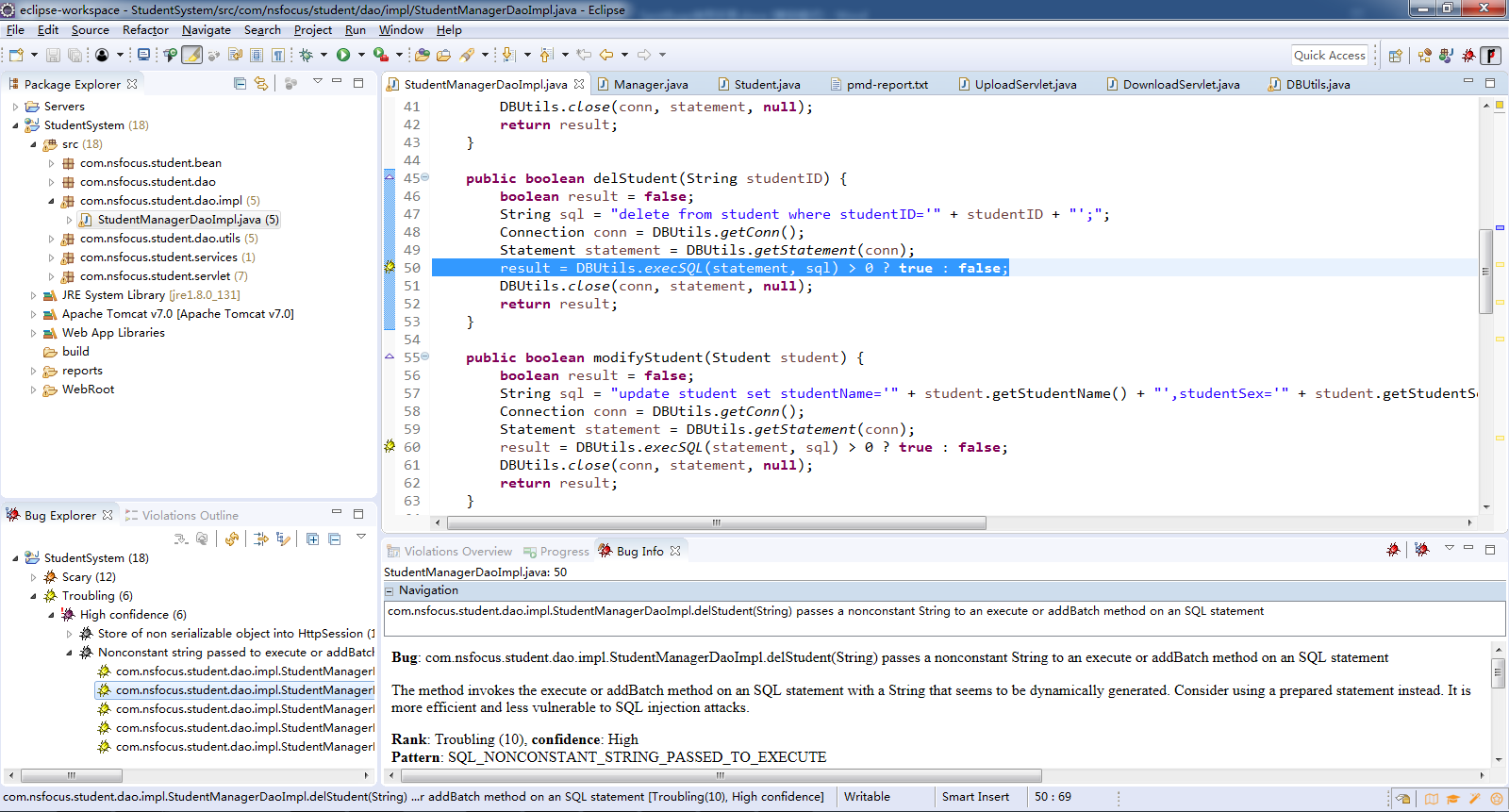


6.在Reporter Configuration选项卡中，可配置关注的缺陷，如只关注安全。



2.3.2 SpotBugs插件使用

在Project Explorer区域内右击项目→SpotBugs→Find Bugs，对代码进行检查，检查结果如下图：



# Dependency-check

## 工具简介

Dependency-Check是一个实用程序，用于识别项目依赖项并检查是否存在任何已知的、公开披露的漏洞。目前，支持Java和.NET，为Ruby，Node.js，Python添加了额外的实验支持，并为C / C ++构建系统（autoconf和cmake）提供了有限的支持。

该工具用于发现由于开发周期过程，项目所依赖的组件存在已知安全问题而未及时更新所造成的漏洞。该工具可以成为OWASP Top 10 2017 A9：2017使用具有已知漏洞的组件的解决方案的一部分。

Dependency-Check有一个命令行界面，一个Maven插件，一个Ant任务和一个Jenkins插件。核心引擎包含一系列分析器，用于检查项目依赖关系，收集有关依赖关系的信息（在工具中称为证据）。然后，该证据用于标识给定依赖项的公共平台枚举（CPE）。如果标识了CPE，则会在报告中列出关联的常见漏洞和披露（CVE）条目的列表。

依赖关系检查使用NIST托管的NVD数据源自动更新自身。

Dependency-Check支持多种格式的文件：Zip archive format（\*.zip，\*.ear，\*.war，\*.jar，\*.sar，\*.apk，.nupkg）；Tape Archive Format（.tar）；Gzip format（\*.gz，.tgz）；Bzip2 format（.bz2，\*.tbz2）。

## 使用方法

Dependency Check官网：

https://www.owasp.org/index.php/OWASP\_Dependency\_Check

Dependency Check官方文档：

https://jeremylong.github.io/DependencyCheck/index.html

Dependency Check有三种使用方式：

从命令行使用：此时Dependency Check作为一个单独的软件，与项目无关，使用时只需指定需要扫描的项目位置即可。在业务上线流程中推荐使用此方式。

作为插件在项目中使用：此时需要在项目的配置文件中做相关内容添加，只对当前项目有效。别的项目需要使用时，需要重新修改配置文件。

作为Ant Task使用：这种方式的使用介于以上两者之间，可以在多个项目中使用，但是需要安装，并且需要在项目的build.xml中添加相关配置。

### 命令行工具

从官网下载Command Line工具，下载后解压即可使用。如果是Windows平台，运行bin目录下Dependency-Check.bat；如果是Linux，运行Dependency-Check.sh；如果是Mac平台，使用brew install Dependency-Check即可安装。

Windows使用Dependency-Check.bat --project "My App Name"--scan "c:\java\application\lib" --format HTML --out D:\myapp.html

**--project 用于指定这个扫描项目的名字（不是要扫描的项目，而是扫描这个任务的名字，可以随便指定）**

**--scan 指定要扫描的压缩文件或者目录。**

**--out 指定输出文件位置及文件名**

**--format 指定报告输出格式**

**\*第一次使用会很慢，因为要下载相应的NVD库。**

**使用-n/--noupdate参数可禁用CPE数据的自动更新，使用--updateonly参数，则仅执行依赖检查的更新阶段，不会执行扫描，也不会生成报告。**

**使用--help参数获取帮助信息，使用--advancedHelp参数获得高级选项帮助信息。**

参考链接：https://jeremylong.github.io/DependencyCheck/dependency-check-cli/arguments.html

### 基于Maven的插件

这种方式需要Maven版本在3.1或者之后。需要在maven的项目依赖中添加Dependency-Check的依赖如下：

<dependency>

<groupId>org.owasp</groupId>

<artifactId>dependency-check-maven</artifactId>

<version>3.3.1</version>

<dependency>

然后添加插件申明：

<project>

...

<build>

...

<plugins>

...

<plugin>

<groupId>org.owasp</groupId>

<artifactId>Dependency-Check-maven</artifactId>

<version>3.3.1</version>

</plugin>

...

</plugins>

...

</build>

...

</project>

参考链接：https://jeremylong.github.io/DependencyCheck/dependency-check-maven/index.html

### Ant Task

在这种使用模式下， 需要首先从官网下载dependencyCheck-ant二进制包。解压后在build.xml中配置如下：

<!-- 此处指定Dependency-Check的解压后目录 -->

<property name="dependency-check.home"value="C:/tools/dependency-check-ant"/>

<path id="dependency-check.path">

<pathelement location="${dependency-check.home}/dependency-check-ant.jar"/>

<fileset dir="${dependency-check.home}/lib">

<include name="\*.jar"/>

</fileset>

</path>

<taskdef resource="dependency-check-taskdefs.properties">

<classpath refid="dependency-check.path"/>

</taskdef>

然后就可以执行Ant Task了。

使用dependency-check命令开始扫描，检查项目依赖关系，使用dependency-check-purge删除本地NVD库。使用dependency-check-update更新NVD库而不开启扫描。

当使用其他项目的时候，只需要复制build.xml配置过去，而不用重新下载NVD数据库。

参考链接：https://jeremylong.github.io/DependencyCheck/dependency-check-ant/index.html

### 报告输出

**Dependency-Check支持多种输出格式，默认是HTML格式。在命令式方式使用时，指定参数--format来选定格式，可以有XML，HTML，CSV，JSON，VULN，ALL这些选项（必须是大写）。**

HTML格式将会生成一个报告文档，有关于各项的统计总结，每个依赖对应的CVE，以及简单说明，是最通俗易读的方式。

而CSV这些格式的报告将会生成一个列表，结构如下：



结构很清晰，但是信息比较精简，关于证据的来源，可信度等信息都被忽略了，只保留了最重要的漏洞编号及简单描述，没有树形结构信息。

XML与JSON生成的报告很类似，只是语法结构不一样，内部的字段以及信息结构都是一样的。以下是JSON报告片段截取：

{ "scanInfo": {

"engineVersion": "2.1.0",

"dataSource": [

{

"name": "NVD CVE 2002",

"timestamp": "01/08/2017 15:14:30"

},

{

"name": "NVD CVE 2003",

"timestamp": "29/07/2017 15:24:56"},

......

]

},

"projectInfo": {

"name": "Testing",

"version": "2.1.0",

"reportDate": "2017-08-07T15:14:04.886+0800",

"credits": {

"NVD": "This report contains data retrieved fromthe National Vulnerability Database: http://nvd.nist.gov",

"NSP": "This report may contain data retrieved fromthe Node Security Platform: https://nodesecurity.io"

}

},

"dependencies": [

{

"fileName": "blanka-mis-1.0-SNAPSHOT.jar",

"filePath": "C:\\Users\\10076\_000\\Desktop\\blanka-mis-1.0-SNAPSHOT.jar",

"md5": "e0a00ba23c2238523437ee58239020b0",

"sha1": "24b9238959576d84fc39dd180e6f253af66950c0",

"evidenceCollected": {

"vendorEvidence": [

{

"type": "vendor",

"confidence": "HIGH",

"source": "file",

"name": "name",

"value": "blanka-mis"

},

{

"type": "vendor",

"confidence": "LOW",

"source": "Manifest",

"name": "spring-boot-classes",

"value": "BOOT-INF/classes/"

}

},

"productEvidence": [

{

"type": "product",

"confidence": "HIGH",

"source": "file",

"name": "name",

"value": "blanka-mis"

},

{

"type": "product",

"confidence": "LOW",

"source": "jar","name": "package name",

"value": "classes"

},

......

}

如果是人工审阅，html格式是最好的选择，如果是需要自动化处理，json与xml都差不多，而CSV格式信息比较少，并且可能出现编码问题，不做推荐。

### 通过PIP安装

通过pip install dependency-check的方式通过python安装Dependency-Check，会安装在当前用户家目录下

### 脱机下的使用方法

Dependency-Check工具所使用的NVD数据库，需要通过互联网进行下载。对于内网中无法访问互联网的开发集成环境，可通过互联网将需要的数据下载下来，再拷贝到对应的主机中。Dependency-Check的NVD数据文件存放在{Dependency-Check安装目录}/data/下。

## 基于Jenkins的插件配置及使用

### 插件安装

Dependency-Check Jenkins插件具有执行依赖关系分析构建和以后查看构建后结果的功能。该插件使用analysis-core构建，具有Jenkins静态分析插件提供的许多相同功能，包括阈值，图表以及在依赖关系已识别的情况下查看漏洞信息的能力。

系统管理->插件管理->可选插件，搜索Dependency-Check，点击直接安装。



图 2.1 插件管理界面



图 2.2 插件安装过程界面

### 全局配置

在Jenkins的系统管理->系统设置中，可对Dependency-Check插件进行全局配置。设置好Global Data Directory路径，在其他构建中，即可使用默认的全局路径，避免在构建过程中给，重复下载Dependency-Check所依赖的NVD数据库，造成构建时间过长。

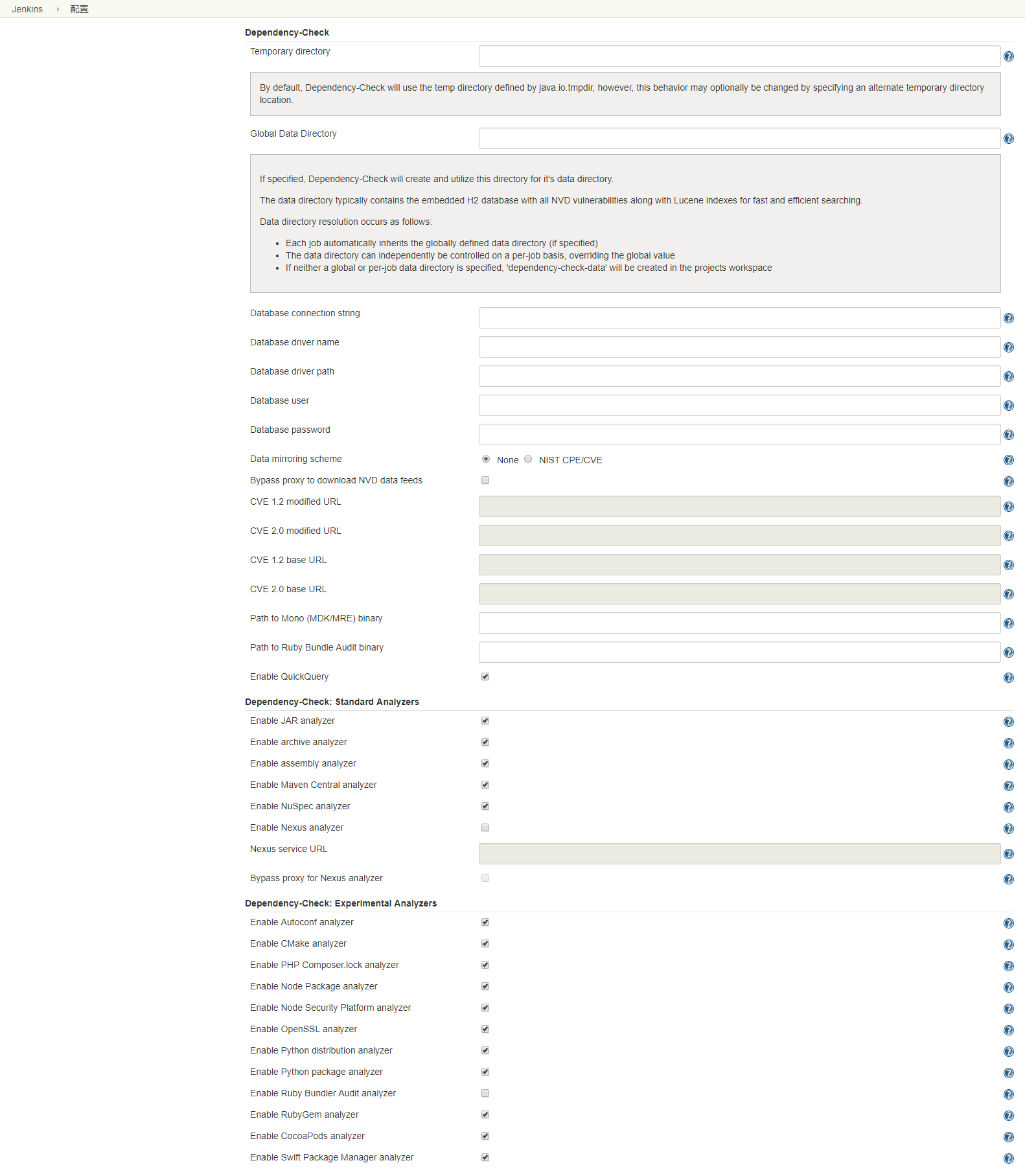


图 2.3 Dependency-Check全局配置界面

### 项目配置使用

新建或在现有Job中，在“Post Steps”栏目中，点击“Add post-build step”，插入“Invoke Dependency-Check analysis”模块。

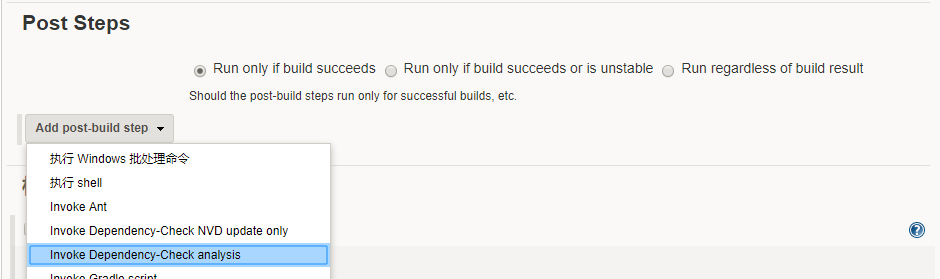


图 2.4 Post Steps

Path to scan为空时，会直接从当前空间开始扫描，即该Job所在位置。

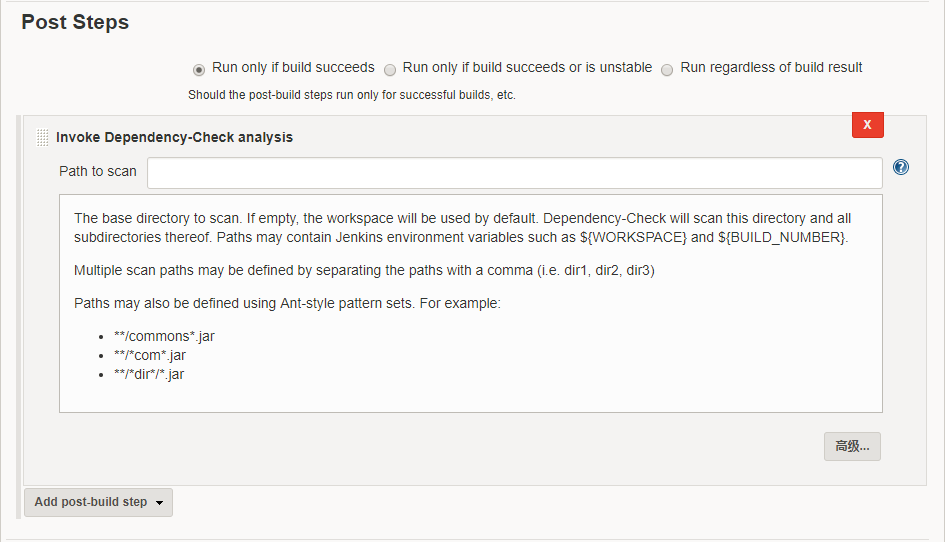


图 2.5 Dependency-Check模块

点击“高级”按钮，可配置更多内容，选择所需要的数据进行输出。

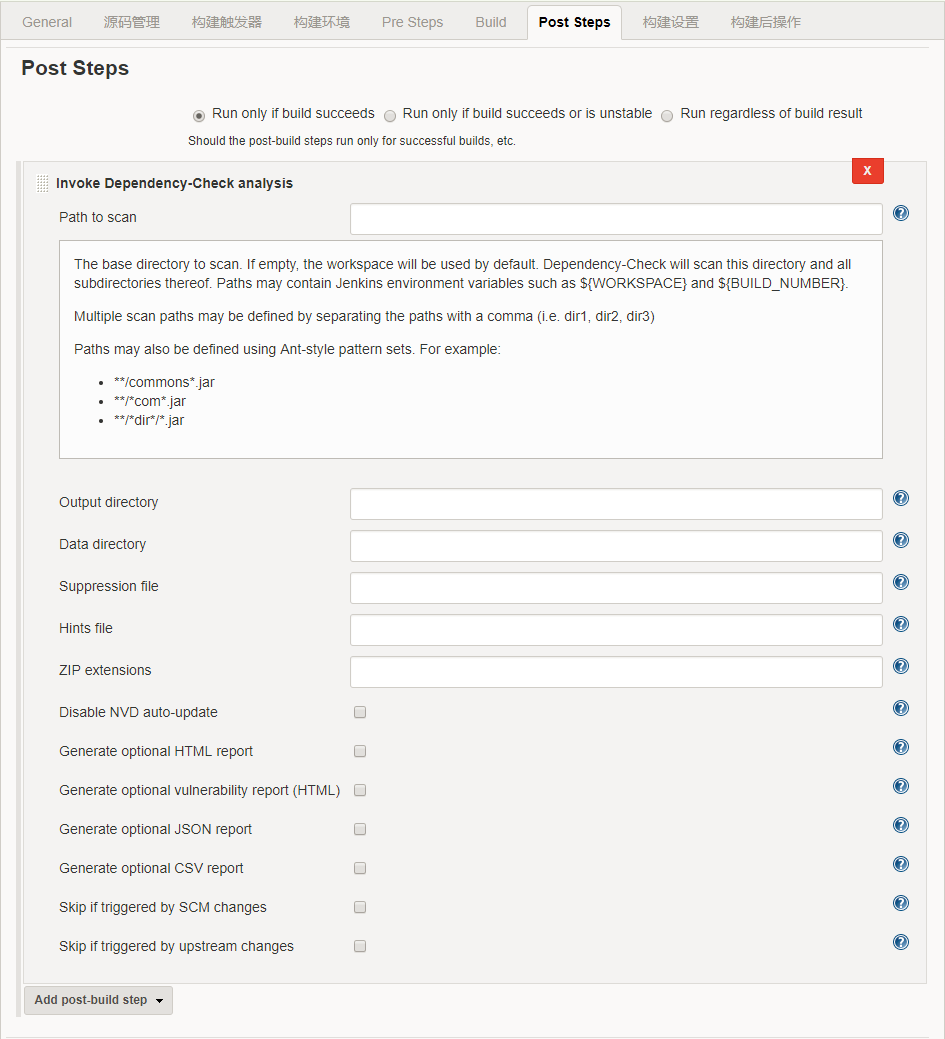


图 2.6 Dependency-Check模块高级选项

在“构建后操作”栏目中，可增加“Publish OWASP Dependency-Check analysis results”步骤，设置指标，对扫描结果进行控制，不满足指标的构建，直接失败。

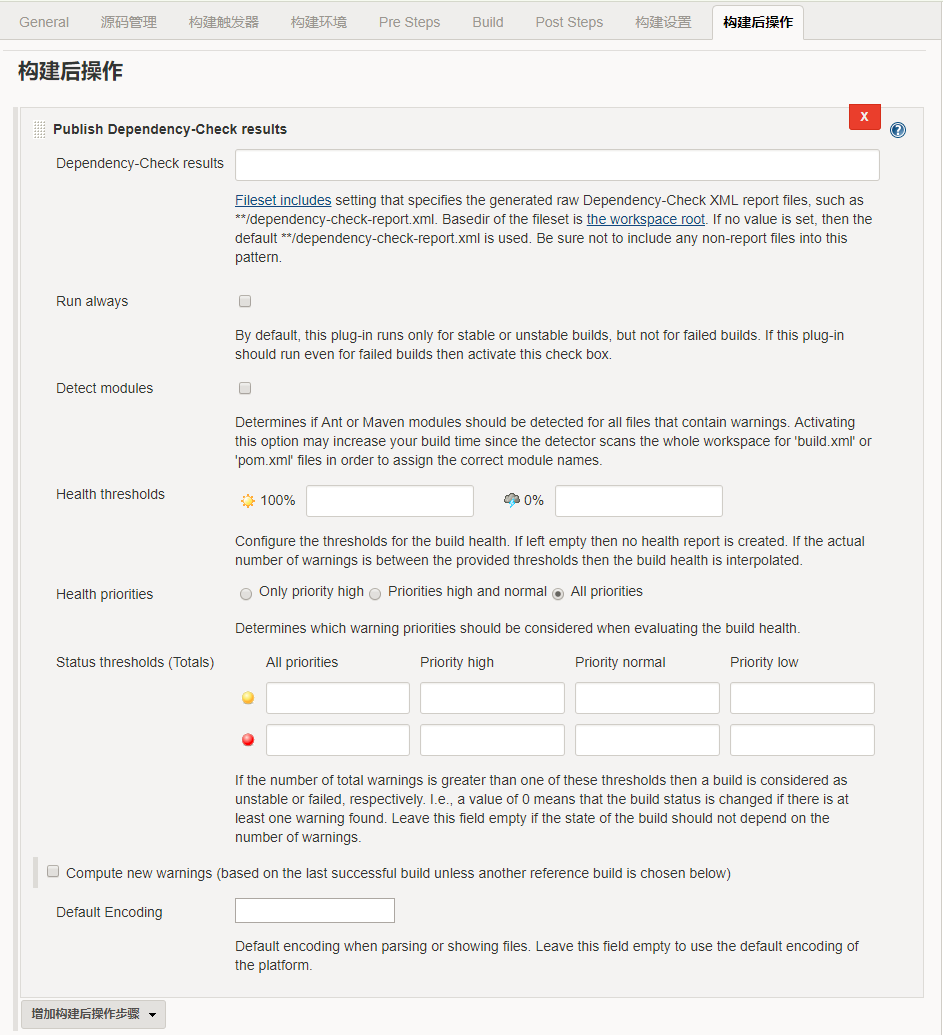


图 2.7 Dependency-Check模块结果处理

### 注意事项

避免每次构建都更新Dependency-Check所依赖的NVD数据库，造成构建时间过长，可在配置中，勾选“Disable NVD auto-update”选项。可以建一个自由风格的项目，插入“Invoke OWASP Dependency-Check NVD update”模块作为构建的步骤，该项目仅仅是为了定期去更新NVD数据库。

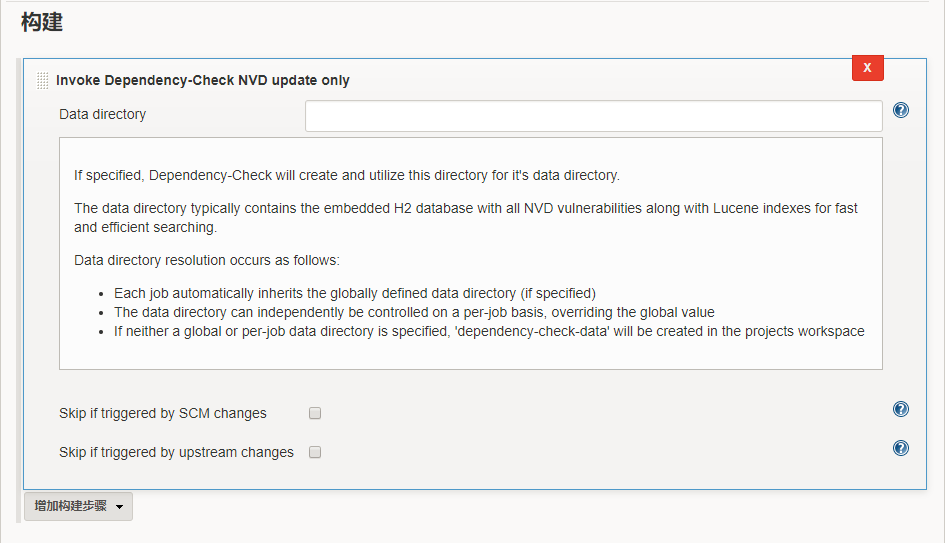


图 2.8 Dependency-Check模块update only功能

Dependency-Check功能，需要通过maven构建，在目标目录下生成构建完成的war包或完整项目才能完成对项目代码的扫描。

### Pipeline片段编写

通过“Pipeline Syntax”->“Snippet Generator”可生成OWASP Dependency-Check插件的语法片段。

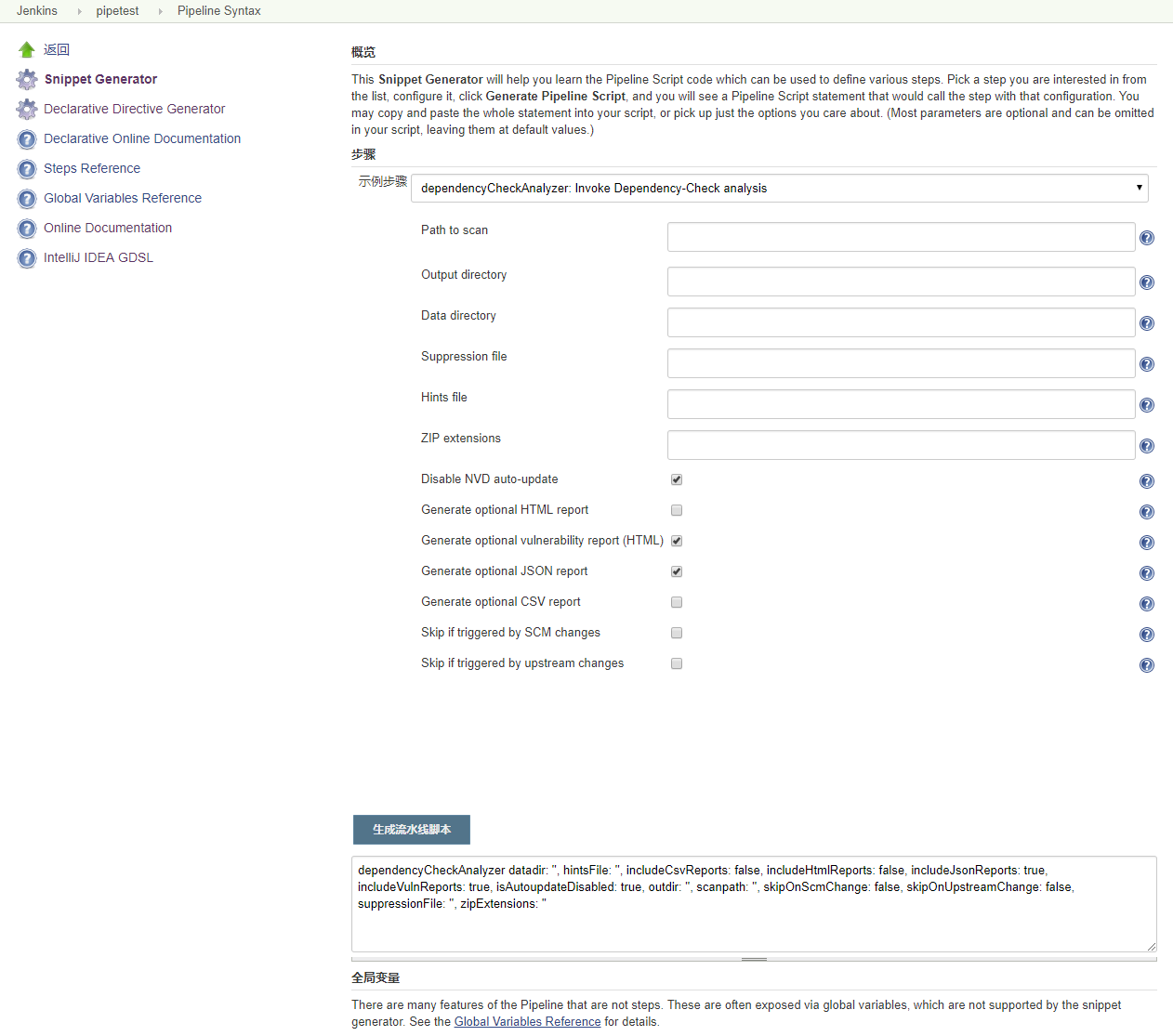


图 2.9 Dependency-Check插件pipeline编写器

stage('dependencyCheck') {

dependencyCheckAnalyzer datadir: '', hintsFile: '', includeCsvReports: false, includeHtmlReports: false, includeJsonReports: true, includeVulnReports: true, isAutoupdateDisabled: true, outdir: '', scanpath: '', skipOnScmChange: false, skipOnUpstreamChange: false, suppressionFile: '', zipExtensions: ''

}