Fortify规则教程2

作者：xsser

与其他白盒不一样的概念：

- Taint EntryPoint
- Taint Flag

其中Taint Characterization Rule应该是在属于在dataflow rule中的(在官方规则手册中)

其中我重点讲解关于漏洞相关的内容，我言简意赅的介绍，不多说废话。深入看下这个教程，将会对你理解fortify如何编写规则有一个大的理解提升。

Concept	Dataflow Rule	Taint Characterization Rule
Taint Source	DataflowSourceRule	TaintSource TaintWrite
Taint Entrypoint	DataflowEntryPointRule	TaintEntrypoint
Taint Passthrough	DataflowPassthroughRule	TaintTransfer
Taint Sink	DataflowSinkRule	TaintSink
Taint Cleanse	DataflowCleanseRule	TaintCleanse

Fortify Dataflow rule的结构解释

```
<DataflowXYZRule formatVersion="17.10" language="...">
  <RuleID>...</RuleID>
  <MetaInfo>...</MetaInfo> <!-- DataflowSinkRule only -->
  <VulnKingdom>...</VulnKingdom> <!-- DataflowSinkRule only -->
  <VulnCategory>...</VulnCategory> <!-- DataflowSinkRule only -->
  <VulnSubCategory>...</VulnSubcategory> <!-- DataflowSinkRule only -->
  <Description>...</Description> <!-- DataflowSinkRule only -->
  <FunctionIdentifier>
      ...
  </FunctionIdentifier>
  <InArguments>...</InArguments> <!-- Optional -->
  <OutArguments>...</OutArguments> <!-- Optional -->
  <TaintFlags>...</TaintFlags> <!-- Optional -->
</DataflowXYZRule>
```

其中的ruleid是唯一的，禁止重复，你可以使用官方提供的Custom Rules editor来创建一个rule。 然后他会自动给你创建一个ruleid，
vulnXXX都是介绍这个漏洞信息的，你可以简单的写一下，描述也可以简单些 或者抄袭下官方提供的rule里的内容，重要的是FunctionXXX，这里描述的是source属于哪个包的哪个class的哪个method。如下图

```
<FunctionIdentifier>
  <NamespaceName>
    <Pattern>javax\.servlet</Pattern>
  </NamespaceName>
  <ClassName>
    <Value>ServletRequest</Value>
  </ClassName>
  <FunctionName>
    <Value>getParameter</Value>
  </FunctionName>
  <ApplyTo implements="true" overrides="true" extends="true"/>
</FunctionIdentifier>
```

Applyto描述的是 这个方法可以是继承、重写等。看到这里你就明白了，那些点点点，通过官方custom rules edtior生成的规则多么简单，就是简单的制定了包下的method。那么这样的情况不用想，就会有很多误报和情况，接下来的规则里我会介绍这个情况如何解决。

写到这里我就想说fortify真垃圾，还不如codeql方便，codeql一条select就解决了这些问题。

Dataflow rule下有很多规则，比如sink, source, entryPoint等等，这些你们看下官方的手册基本上就懂他在说什么了，描述的xml规则结构也是package->class->method这个模式

Taint Characterization Rule

重点来了，这个规则才是核心，没想到吧，dataflow什么的都是太简单了，这个规则是一个超集。

这个规则就2部分

```
<CharacterizationRule formatVersion="17.10" language="...">
  <RuleID>...</RuleID>
  <MetaInfo>...</MetaInfo> <!-- TaintSink only -->
  <VulnKingdom>...</VulnKingdom> <!-- TaintSink only -->
  <VulnCategory>...</VulnCategory> <!-- TaintSink only -->
  <VulnSubCategory>...</VulnSubcategory> <!-- TaintSink only -->
  <Description>...</Description> <!-- TaintSink only -->
  <StructuralMatch><[CDATA[
    ...
  ]]></StructuralMatch>
  <Definition><![CDATA[
    ...
  ]]><Definition>
</CharacterizationRule>
```

其中StructuralMatch是描述这个上下文，Definition描述这个在哪些场景出发，比如TaintSource, TaintWrite, TaintEntrypoint, TaintSink, TaintTransfer, or TaintCleanse

这个规则有点那个味儿了，就是codeql的关系型查询的味道。主要通过逻辑符来连接多个条件来查询一定的结构树

```
<StructuralMatch><![CDATA[
FunctionCall fc:
    function is [Function:
        name == "of"
        and enclosingClass.supers contains [Class:
            name == "java.net.http.HttpHeaders"
        ]
    ]
    and arguments[0] is [Expression arg:]
    and arguments[1] is [Expression lambda:]
</StructuralMatch>
```

这表示寻找一个方法调用fc,他的原型的名字是以'of'为名字的方法，且属于java.net.http.HttpHeaders类下，且调用方法第1个参数是普通的参数，第2个参数是一个lambda表达式。

在CharacterizationRule中，Structural规则是描述一个规则树对象的；而Definition是描述这个这个规则可以用到的场景，比如TaintSource,TaintSink write等，且在这里可以访问的到structural里定义的变量，且可以用一些语法访问和遍历上面定义的变量。

这种规则可以用来优化一些误报的场景 比如数据流的时候，经过的点参数是一些复杂的结构，比如map set 或者自定义类型的，我们仅仅判断string是无法扫描出全部的漏洞。

或者是在一些满足某些状态的，比如while循环或者for循环，某个表达式返回的true or false来执行拼接，那么就会用到这个StructuralRule和CharacterizationRule。

如何调试规则

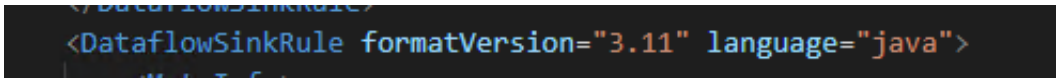
我们可以在cmd下执行命令

```
                    'd/code_review
$ sourceanalyzer -b example -scan -no-default-rules -rules example.xml -debug -verbose -analyzers structural
Fortify Static Code Analyzer 20.1.1.0007 (using JRE 1.8.0_181)
[warning]: Cannot read from "example.xml" (file not found)
[error]: No rules files found
exit(1)
```

其中-no-default-rules屏蔽原生垃圾规则，structural就是规则的类型，你可以定义dataflow来调试数据流的规则。

优缺点

1. 规则文件过大，10+M的文件是常见的。而且规则文件从古老的版本的fortify开始，可以看到3.11 的规则都在里面，整个 xml文件无比巨大，且里面还有一些错误的规则，可能是不小心写的，官方也太不小心了。



2. 在手册里，对于规则的CharacterizationRule的Definition没有详细的说明，让人摸不透。

鬼知道你的规则根据版本还支持了annotation，但是手册只能搜到17.X的，里面也没提到什么annotation。也不知道CallsMethod，calls区别是啥。官方问答和死了一样，关键的问题一问三不知或者不回答。没有完整的手册

3. 好处是，规则全，但是质量真的不可保证。但是部分支持不到位，比如list的索引,set map等，fortify就只支持Method的参数，不知道是不是我看的规则的版本问题