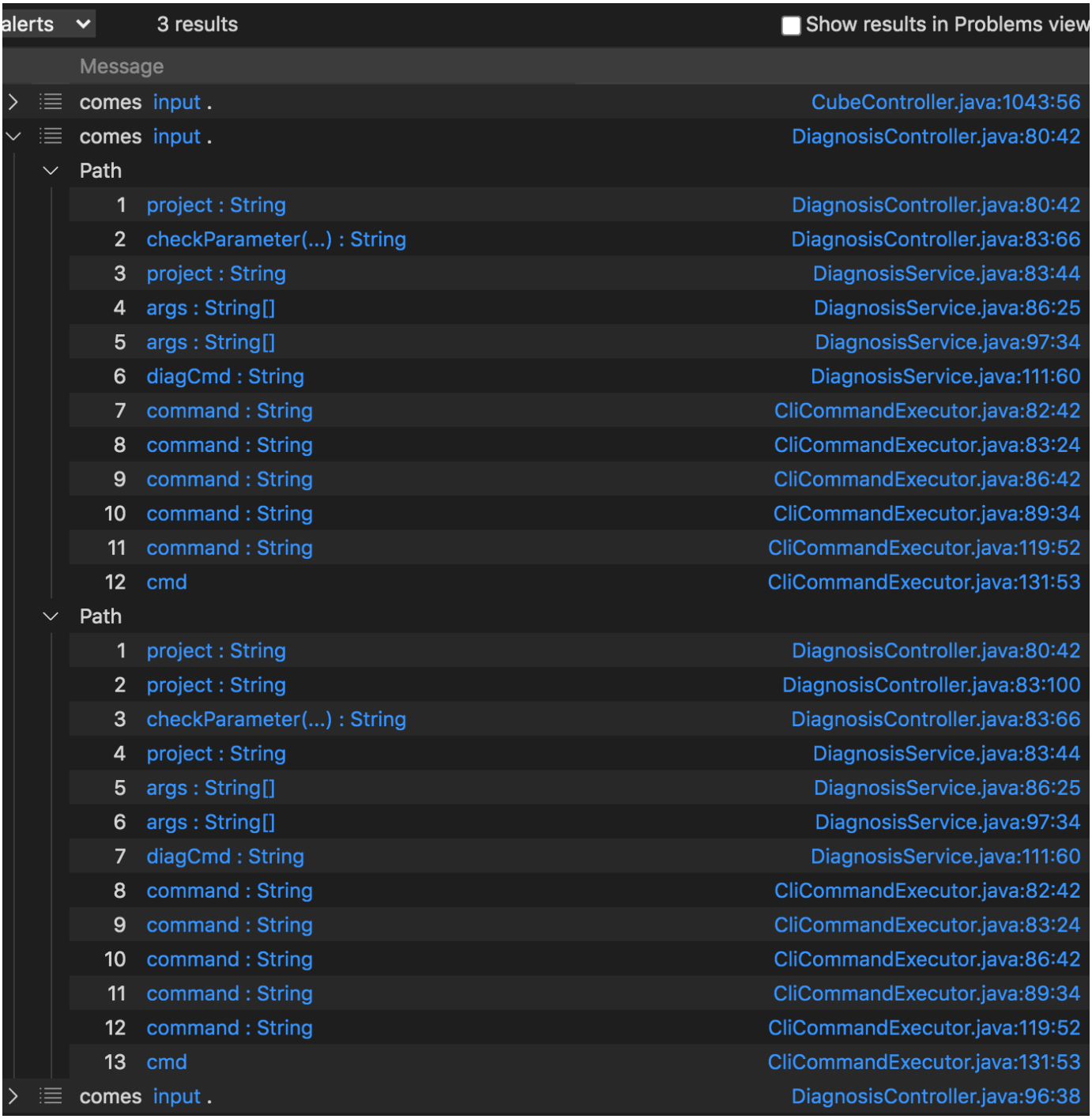
以Apache kylin为例编写命令执行检查规则

根据上一篇文章我们编写 一个有source,sanitizer,sink的规则，首先这样编写

```
/**
 * @name cmd_injection_wangminwei091
 * @description 命令注入.
 * @kind path-problem
 * @problem.severity error
 * @precision high
 * @id java/cmd-injection-wangminwei091
 * @tags security
 *       external/cwe/cwe-089
 */



import semmle.code.java.dataflow.FlowSources
import semmle.code.java.security.ExternalProcess
import DataFlow::PathGraph
// import DataFlow::PartialPathGraph
import semmle.code.java.StringFormat
import semmle.code.java.Member
import semmle.code.java.JDK
import semmle.code.java.Collections

class WConfigToExec extends TaintTracking::Configuration {
  WConfigToExec() { this = "cmd::cmdTrackingTainted" }

  override predicate isSource(DataFlow::Node source) {
    source instanceof RemoteFlowSource
   }

  override predicate isSink(DataFlow::Node sink) {
    sink.asExpr() instanceof ArgumentToExec
  }

}
```

```
35
36 class CallTaintStep extends TaintTracking::AdditionalTaintStep {
37   override predicate step(DataFlow::Node n1, DataFlow::Node n2) {
38     exists(Call call |
39       n1.asExpr() = call.getAnArgument() and
40       n2.asExpr() = call
41     )
42   }
43 }
44
45
46 from  DataFlow::PathNode source,DataFlow::PathNode  sink,WConfigT
   oExec c
47 where c.hasFlowPath(source, sink)
48 select source.getNode(), source, sink, "comes $@.", source.getNod
   e(), "input"
```

其中additionalTaintStep是用来描述node之间的调用关系的。
代码中的call taint step中描述了2个节点之间的关系，意思是调用 关系，一个是方法名(n2)，一个是参数(n1)。
即描述了函数调用。
但是这样的 结果并不完善，如下图



我们可以清晰的看到，checkParameter被扫描出来了，这是一个没有漏洞的数据流，说明我们没有写好。

我们继续把规则修改一下，把checkParameter拉黑。

```
/**
@kind path-problem
*/
```

```
4
5  import java
6  import semmle.code.java.dataflow.DataFlow
7  import semmle.code.java.dataflow.FlowSources
8  import semmle.code.java.dataflow.TaintTracking
9  import DataFlow::PathGraph
10 import semmle.code.java.security.ExternalProcess
11 import DataFlow::PathGraph
12 import semmle.code.java.StringFormat
13 import semmle.code.java.Member
14 import semmle.code.java.JDK
15
16 class DumpProjectDiagnosisInfoMethod extends Method {
17     DumpProjectDiagnosisInfoMethod() {
18     //   this.hasName("dumpProjectDiagnosisInfo")
19
20     this.getSourceDeclaration().getAnAnnotation().toString().matc
   hes("%Mapping%") and
21     this.getAParameter().getAnAnnotation().toString().matches("Pa
   thVariable")
22     }
23 }
24
25
26 class UserInputSource extends DataFlow::Node {
27     UserInputSource() {
28         exists(DumpProjectDiagnosisInfoMethod dumpProjectDiagnosi
   sInfoMethod |
29             // this.asParameter() = dumpProjectDiagnosisInfoMetho
   d.getParameter(0) and this instanceof RemoteFlowSource
30             this.asParameter().getCallable()
31             instanceof DumpProjectDiagnosisInfoMethod
32         )
33     }
34 }
35
36 class ProcessBuilderConstructorCall extends Call {
37     ProcessBuilderConstructorCall() {
38       this
39           .(ConstructorCall)
```

```
            .getConstructedType()
            .getSourceDeclaration()
            .hasQualifiedName("java.lang", "ProcessBuilder")
    }
}

class TaintConfig extends TaintTracking::Configuration {
    TaintConfig() { this = "TaintConfig" }

    override predicate isSource(DataFlow::Node source) { source instanceof UserInputSource }
    override predicate isSanitizer(DataFlow::Node san){ //拉黑checkxxx
        san.asExpr().(MethodAccess).getMethod().getName().matches("checkParameter")
        // none()
    }
    override predicate isSink(DataFlow::Node sink) {
        sink.asExpr() instanceof ArgumentToExec
    //   exists(Call call |
    //     call instanceof ProcessBuilderConstructorCall and
    //     sink.asExpr() = call.getAnArgument()
    //   )
    }

    override int explorationLimit() { result = 12 }

}


class CallTaintStep extends TaintTracking::AdditionalTaintStep {
    override predicate step(DataFlow::Node n1, DataFlow::Node n2) {
      exists(Call call |
        n1.asExpr() = call.getAnArgument() and
        n2.asExpr() = call
      )
    }
}

```

```
76 from TaintConfig cfg, DataFlow::PathNode source, DataFlow::PathNo
   de sink
77 where cfg.hasFlowPath(source, sink)
78 select sink, source, sink, "Custom constraint error message conta
   ins unsanitized user data"
```

代码第50行。即可完善结果