

# 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

```
1 /**
2  * @name cmd_injection_wangminwei091
3  * @description 命令注入.
4  * @kind path-problem
5  * @problem.severity error
6  * @precision high
7  * @id java/cmd-injection-wangminwei091
8  * @tags security
9  *       external/cwe/cwe-089
10 */
11
12
13
14 import semmle.code.java.dataflow.FlowSources
15 import semmle.code.java.security.ExternalProcess
16 import DataFlow::PathGraph
17 // import DataFlow::PartialPathGraph
18 import semmle.code.java.StringFormat
19 import semmle.code.java.Member
20 import semmle.code.java.JDK
21 import semmle.code.java.Collections
22
23 class WConfigToExec extends TaintTracking::Configuration {
24   WConfigToExec() { this = "cmd::cmdTrackingTainted" }
25
26   override predicate isSource(DataFlow::Node source) {
27     source instanceof RemoteFlowSource
28   }
29
30   override predicate isSink(DataFlow::Node sink) {
31     sink.asExpr() instanceof ArgumentToExec
32   }
33
34 }
```

```

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

alerts
3 results
Show results in Problems view

Message

> comes input . CubeController.java:1043:56

> comes input . DiagnosisController.java:80:42

Path

1 project : String DiagnosisController.java:80:42

2 checkParameter(...) : String DiagnosisController.java:83:66

3 project : String DiagnosisService.java:83:44

4 args : String[] DiagnosisService.java:86:25

5 args : String[] DiagnosisService.java:97:34

6 diagCmd : String DiagnosisService.java:111:60

7 command : String CliCommandExecutor.java:82:42

8 command : String CliCommandExecutor.java:83:24

9 command : String CliCommandExecutor.java:86:42

10 command : String CliCommandExecutor.java:89:34

11 command : String CliCommandExecutor.java:119:52

12 cmd CliCommandExecutor.java:131:53

Path

1 project : String DiagnosisController.java:80:42

2 project : String DiagnosisController.java:83:100

3 checkParameter(...) : String DiagnosisController.java:83:66

4 project : String DiagnosisService.java:83:44

5 args : String[] DiagnosisService.java:86:25

6 args : String[] DiagnosisService.java:97:34

7 diagCmd : String DiagnosisService.java:111:60

8 command : String CliCommandExecutor.java:82:42

9 command : String CliCommandExecutor.java:83:24

10 command : String CliCommandExecutor.java:86:42

11 command : String CliCommandExecutor.java:89:34

12 command : String CliCommandExecutor.java:119:52

13 cmd CliCommandExecutor.java:131:53

> comes input . DiagnosisController.java:96:38

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。

```

1 /**
2 @kind path-problem
3 */

```

3

```

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
21         hes("%Mapping%") and
22         this.getAParameter().getAnAnnotation().toString().matches("Pa
23         thVariable")
24     }
25 }
26
27 class UserInputSource extends DataFlow::Node {
28     UserInputSource() {
29         exists(DumpProjectDiagnosisInfoMethod dumpProjectDiagnosi
30         sInfoMethod |
31         // this.asParameter() = dumpProjectDiagnosisInfoMetho
32         d.getParameter(0) and this instanceof RemoteFlowSource
33         this.asParameter().getCallable()
34         instanceof DumpProjectDiagnosisInfoMethod
35     )
36 }
37
38 class ProcessBuilderConstructorCall extends Call {
39     ProcessBuilderConstructorCall() {
40         this
41         .(ConstructorCall)

```

```

40         .getConstructedType()
41         .getSourceDeclaration()
42         .hasQualifiedName("java.lang", "ProcessBuilder")
43     }
44 }
45
46 class TaintConfig extends TaintTracking::Configuration {
47     TaintConfig() { this = "TaintConfig" }
48
49     override predicate isSource(DataFlow::Node source) { source instanceof UserInputSource }
50     override predicate isSanitizer(DataFlow::Node san){ //拉黑checkxxx
51         san.asExpr().(MethodAccess).getMethod().getName().matches
52         ("checkParameter")
53     }
54     override predicate isSink(DataFlow::Node sink) {
55         sink.asExpr() instanceof ArgumentToExec
56         // exists(Call call |
57         //     call instanceof ProcessBuilderConstructorCall and
58         //     sink.asExpr() = call.getAnArgument()
59         // )
60     }
61
62     override int explorationLimit() { result = 12 }
63
64 }
65
66
67 class CallTaintStep extends TaintTracking::AdditionalTaintStep {
68     override predicate step(DataFlow::Node n1, DataFlow::Node n2)
69     {
70         exists(Call call |
71             n1.asExpr() = call.getAnArgument() and
72             n2.asExpr() = call
73         )
74     }
75 }

```

```
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