YASA 检测 CVE-2024-43202 路径穿越漏洞报告

背景介绍

本次扫描利用 YASA 静态分析工具检测 Java 项目中存在的路径穿越漏洞 CVE-2024-43202。该漏洞出现在 Apache DolphinScheduler 3.2.1 版本中，由不安全地加载 YAML 配置文件引发。最初尝试使用 YASA 默认内置的规则包（taint-flow-java-default）进行扫描，但并未成功检测出漏洞。因此编写了自定义的规则配置文件（rule_config_java_cve43202.json），专门标记了漏洞相关的 Source 和 Sink，最终成功识别出了此路径穿越漏洞。

CVE 编号与 Source/Sink 信息

CVE 编号：CVE-2024-43202。

Source：漏洞的源头是 HttpTaskDefinitionParser.parse(@NonNull String yamlConfigFile) 方法中的参数 yamlConfigFile。这个参数代表一个 YAML 配置文件的路径，由外部输入提供，并在后续流程中直接使用。

Sink：漏洞的敏感操作发生在 HttpTaskDefinitionParser.parseYamlConfigFile(String yamlConfigFile) 方法内部。当调用 parseYamlConfigFile(yamlConfigFile) 时，代码使用 new FileReader(yamlConfigFile) 来读取该路径指定的文件。由于 yamlConfigFile 来源可控，未经安全校验就直接传入文件读取操作，导致了路径穿越风险。下面摘录 HttpTaskDefinitionParser.java 于相关代码片段：

```java
public @NonNull HttpLoopTaskDefinition parse(@NonNull String yamlConfigFile) {
    LoopTaskYamlDefinition loopTaskYamlDefinition;
    try {
        loopTaskYamlDefinition = parseYamlConfigFile(yamlConfigFile);
    } catch (IOException ex) {
        throw new IllegalArgumentException(String.format("Parse yaml file: %s error", yamlConfigFile), ex);
    }
    validateYamlDefinition(loopTaskYamlDefinition);

    LoopTaskYamlDefinition.LoopTaskServiceYamlDefinition service = loopTaskYamlDefinition.getService();
    LoopTaskYamlDefinition.LoopTaskAPIYamlDefinition api = service.getApi();
    HttpLoopTaskSubmitTaskMethodDefinition submitTaskMethod =
            new SubmitTemplateMethodTransformer().transform(api.getSubmit());
    HttpLoopTaskQueryStatusMethodDefinition queryTaskStateMethod =
            new QueryStateTemplateMethodTransformer().transform(api.getQueryState());
    HttpLoopTaskCancelTaskMethodDefinition cancelTaskMethod =
            new CancelTemplateMethodTransformer().transform(api.getCancel());
    return new HttpLoopTaskDefinition(service.getName(), submitTaskMethod, queryTaskStateMethod, cancelTaskMethod);
}

protected @NonNull LoopTaskYamlDefinition parseYamlConfigFile(@NonNull String yamlConfigFile) throws IOException {
    Yaml yaml = new Yaml(new Constructor(LoopTaskYamlDefinition.class));
    try (FileReader fileReader = new FileReader(yamlConfigFile)) {
        return yaml.load(fileReader);
    }
}
```

上述代码中，yamlConfigFile 作为文件路径参数传递给 parseYamlConfigFile，并在其中通过 FileReader 直接打开文件。这种未经充分校验的文件路径操作会导致路径穿越漏洞。

YASA 配置

扫描命令：我们使用以下命令运行 YASA，对目标源码文件进行污点分析检测。本次扫描指定使用默认的 Java 污点规则包并加载自定义规则文件，同时定位待扫描的源代码路径和报告输出路径：

```
./yasa-engine-linux-x64 \
  --checkerPackIds "taint-flow-java-default" \
  --analyzer "JavaAnalyzer" \
  --ruleConfigFile "/home/fzk/yasa-bundle/example-rule-config/rule_config_java_cve43202.json" \
```

--sourcePath "/home/fzk/Downloads/dolphinscheduler-3.2.1/dolphinscheduler-task-plugin/dolphinscheduler-task-api/src/main/java/org/apache/dolphinscheduler/plugin/task/api/loop/template/http/parser/HttpTaskDefinitionParser.java" \

--report "/home/fzk/yasa-bundle/report/"

上述命令指定了目标源文件为 HttpTaskDefinitionParser.java（即包含漏洞的文件），使用了默认规则包 taint-flow-java-default 以及我们编写的自定义规则配置文件 rule_config_java_cve43202.json。

自定义规则文件：在 rule_config_java_cve43202.json 中，我们显式定义了漏洞对应的 Source 和 Sink 规则，使 YASA 能够识别该特定的污点传播模式。例如，在规则文件中将 yamlConfigFile 参数标记为 TaintSource（来源），限定于 HttpTaskDefinitionParser.java 文件的 parse 函数中；同时将对 parseYamlConfigFile 方法的调用标记为 TaintSink（敏感点），如下面的规则片段所示：


```
[
  {
    "checkerIds": ["taint_flow_java_input"],
    "sources": {
      "TaintSource": [
        {
          "path": "yamlConfigFile",
          "className": "*",
          "scopeFile": "/HttpTaskDefinitionParser.java",
          "scopeFunc": "parse",
          "attribute": "CVE-2024-43202-PathTraversal"
        }
      ]
    },
    "sinks": {
      "FuncCallTaintSink": [
        {
          "args": ["0"],
          "attribute": "CVE-2024-43202-PathTraversal",
          "calleeType":
          "org.apache.dolphinscheduler.plugin.task.api.loop.template.http.parser.HttpTaskDefinitionParser",
          "fsig": "parseYamlConfigFile",
          "scopeFile": "/HttpTaskDefinitionParser.java",
          "scopeFunc": "parse"
        }
      ]
    },
    "sanitizers": [],
    "entrypoints": []
  }
]
```


通过上述配置，我们告诉 YASA：凡是在 parse 方法中出现的参数 yamlConfigFile

都视为不可信输入；而该方法内对 parseYamlConfigFile(...) 的调用即为敏感操作，需要跟踪其参数（第 0 号参数即 yamlConfigFile）的污点流。由于默认规则包未涵盖这一特殊情况，自定义规则的加入弥补了检测规则的不足，从而能够发现 CVE-2024-43202 漏洞。

检测结果

漏洞检测结论：经过以上配置运行，YASA 成功检测出了路径穿越漏洞 CVE-2024-43202。工具在报告中给出了详细的污点传播调用链，说明了源头数据如何流经各函数最终到达敏感操作点。关键的调用链如下所示：

```
SOURCE: parse(@NonNull String yamlConfigFile)
SINK: loopTaskYamlDefinition = parseYamlConfigFile(yamlConfigFile)
ARG PASS: parseYamlConfigFile(@NonNull String yamlConfigFile)
Var Pass: FileReader fileReader = new FileReader(yamlConfigFile)
Return Value: return yaml.load(fileReader)
CALL RETURN: loopTaskYamlDefinition = parseYamlConfigFile(yamlConfigFile)
Var Pass: loopTaskYamlDefinition = parseYamlConfigFile(yamlConfigFile)
SINK: validateYamlDefinition(loopTaskYamlDefinition)
```

扫描输出结果如图：

```
═══════════════════  Analysis Overview  ═══════════════════
Language                            : java
Files analyzed                      : 1
Lines of code                       : 78
Total time                          : 316ms
Total instruction                   : 158
Executed instruction                : 158
Execution count                     : 284
Sources configured                  : 1
Sinks configured                    : 1
Valid entrypoints                   : 1
Avg execution time per instruction : 0.00ms
Avg instruction execution count     : 1.80
Execution time 70%/99%/100%         : 0.00ms/0.00ms/0.00ms
Execution times 70%/99%/100%        : 2.00/2.00/4.00
═══════════════════════════════════════════════════════════


═══════════════════  Performance Statistics  ═══════════════════
total cost: 316ms
preProcess cost: 254ms
startAnalyze cost: 50ms
makeFullCallGraph(BySymbolInterpret) cost: 18ms
symbolInterpret cost: 11ms
═══════════════════════════════════════════════════════════════

Found 3 potential output strategy files
Registered strategy: callgraph from callgraph-output-strategy.js
Registered strategy: interactive from interactive-output-strategy.js
Registered strategy: taintflow from taint-output-strategy.js
Successfully registered 3 output strategies

═══════════════════  outputFindings  ═══════════════════

═══════════════════ Findings ═══════════════════

──────────  1 :  taint_flow_java_input ──────────
Description: Java污点分析checker，会使用CallGraph边界制作entrypoint
File: /HttpTaskDefinitionParser.java
Line 45: parseYamlConfigFile(yamlConfigFile)
SINK RULE: parseYamlConfigFile
SINK Attribute: CVE-2024-43202-PathTraversal
entrypoint:
{
  filePath: '/HttpTaskDefinitionParser.java',
  functionName: 'parse',
  attribute: 'fullCallGraphMade',
  type: 'functionCall',
  packageName: undefined,
  funcReceiverType: ''
}
Trace:
 /HttpTaskDefinitionParser.java
  AffectedNodeName: yamlConfigFile
  42:  SOURCE:      public @NonNull HttpLoopTaskDefinition parse(@NonNull String yamlCon
 /HttpTaskDefinitionParser.java
  AffectedNodeName: parseYamlConfigFile
  45:  SINK:              loopTaskYamlDefinition = parseYamlConfigFile(yamlConfigFile);

──────────  2 :  taint_flow_java_input ──────────
Description: Java污点分析checker，会使用CallGraph边界制作entrypoint
File: /HttpTaskDefinitionParser.java
Line 49: validateYamlDefinition(loopTaskYamlDefinition)
SINK RULE: parseYamlConfigFile
SINK Attribute: CVE-2024-43202-PathTraversal
entrypoint:
{
  filePath: '/HttpTaskDefinitionParser.java',
  functionName: 'parse',
  attribute: 'fullCallGraphMade',
  type: 'functionCall',
  packageName: undefined,
  funcReceiverType: ''
}
Trace:
 /HttpTaskDefinitionParser.java
  AffectedNodeName: yamlConfigFile
  42:  SOURCE:      public @NonNull HttpLoopTaskDefinition parse(@NonNull String yamlCon
 /HttpTaskDefinitionParser.java
  AffectedNodeName: parseYamlConfigFile
  45:  CALL:              loopTaskYamlDefinition = parseYamlConfigFile(yamlConfigFile);
 /HttpTaskDefinitionParser.java
  AffectedNodeName: yamlConfigFile
  62:  ARG PASS:      protected @NonNull LoopTaskYamlDefinition parseYamlConfigFile(@Non
 /HttpTaskDefinitionParser.java
  AffectedNodeName: fileReader
  64:  Var Pass:          try (FileReader fileReader = new FileReader(yamlConfigFile)) {
 /HttpTaskDefinitionParser.java
  AffectedNodeName: [return value]
  65:  Return Value:              return yaml.load(fileReader);
 /HttpTaskDefinitionParser.java
  AffectedNodeName: parseYamlConfigFile
  45:  CALL RETURN:             loopTaskYamlDefinition = parseYamlConfigFile(yamlConfigF
 /HttpTaskDefinitionParser.java
  AffectedNodeName: loopTaskYamlDefinition
  45:  Var Pass:            loopTaskYamlDefinition = parseYamlConfigFile(yamlConfigFile
 /HttpTaskDefinitionParser.java
  AffectedNodeName: validateYamlDefinition
  49:  SINK:          validateYamlDefinition(loopTaskYamlDefinition);
═══════════════════════════════════════════════════════
  # Total-findings : 2
═══════════════════════════════════════════════════════

report is write to /home/fzk/yasa-bundle/report/report.sarif
```

上面结果表明：来源（SOURCE）为 parse 方法的参数 yamlConfigFile，随后该污点参数被传递给 Sink 函数 parseYamlConfigFile 并用于创建 FileReader。YASA 跟踪到了此污点流向并在调用链中标记出 FileReader fileReader = new FileReader(yamlConfigFile) 这一步，证明了未经验证的文件路径被直接用于文件读取操作，即存在路径穿越漏洞。最后，污点数据在 validateYamlDefinition 方法被使用（在报告中作为终点列出）。基于以上分析与工具报告，可确认 CVE-2024-43202 漏洞已成功被 YASA 工具检测和验证。