实验八：污点分析

2024.6.27

实验四：死代码检测

- 实验平台配置
- 实验内容
- API介绍
- 作业提交

实验平台配置——Tai-e

- **按照 https://tai-e.pascal-lab.net/intro/overview.html 配置**
  - Download assignments at https://github.com/pascal-lab/Tai-e-assignments

实验内容

污点分析

- 在本次作业中的污点分析，会把一些特定的方法（通常是产生数据的 API）视作污染源，调用这些方法的语句会返回污点数据；为了和指针分析中的对象对应，这些污点数据也被叫做污点对象（taint objects）。然后本次作业也把一些特定方法的某些参数视作 taint sinks。为了达到更高的精度，大家需要实现一个上下文敏感的污点分析。

实验内容

污点分析

类型	语句	规则（在上下文 c 下）
Call (source)	`l: r = x.k(a1,...,an)`	$\dfrac{c : l \to c^t : m \in CG \quad \langle m, u \rangle \in \mathit{Sources}}{[\,] : t_l^u \in pt(c : r)}$
Call (sink)	`l: r = x.k(a1,...,an)`	$\dfrac{c : l \to c^t : m \in CG \quad \langle m, i \rangle \in \mathit{Sinks} \quad [\,] : t_j^u \in pt(c : ai)}{\langle j, l, i \rangle \in \mathit{TaintFlows}}$

实验内容

污点分析——污点传播

- 污点分析和指针分析看起来很相似，因为他们本质上都在跟踪程序中数据的流动——指针分析跟踪的是抽象的对象，污点分析跟踪的是污点对象。但是他们又有些不同：污点分析中的污点是一个更加抽象的概念——它与数据的内容相关联，因此它可以在不同的对象之间传播。我们把这样的现象叫做污点传播。

```java
String taint = getSecret(); // source
StringBuilder sb = new StringBuilder();
sb.append("abc");
sb.append(taint); // taint is transferred to sb
sb.append("xyz");
String s = sb.toString(); // taint is transferred to s
leak(s); // sink
```

实验内容

污点分析——污点传播

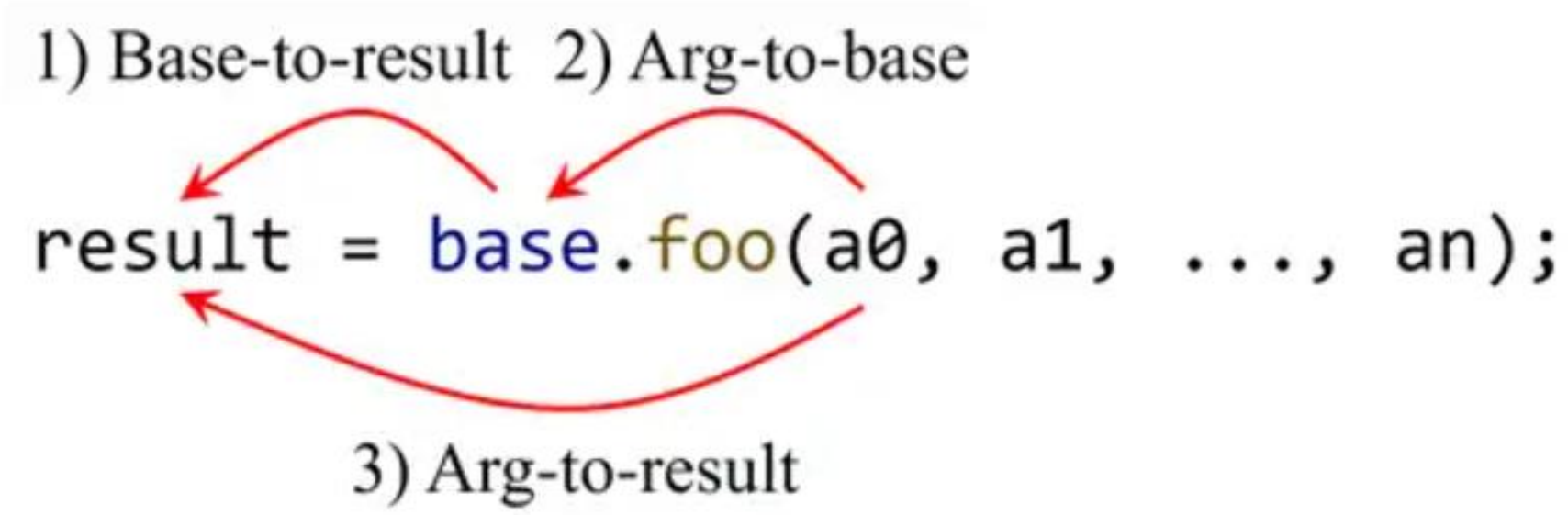
```java
String taint = getSecret(); // source
StringBuilder sb = new StringBuilder();
sb.append("abc");
sb.append(taint); // taint is transferred to sb
sb.append("xyz");
String s = sb.toString(); // taint is transferred to s
leak(s); // sink
```

课上讲解的污点分析算法由于无从知道程序中 API 的含义，例如它不知道上面例子中的 append() 和 toString() 方法可以在不同的对象之间传播内容（特别是敏感数据），因而它不会像先前所描述的那样顺利地传播污点。结果就是，很多安全漏洞都会被遗漏。又因为上述模式在现实中的代码中很常见，所以我们需要对这些方法进行额外处理。

实验内容

污点分析——污点传播

- 解决方法是告诉污点分析哪些方法会引发污点传播以及它们是如何传播污点的。在这次作业中，当一个引发污点传播的方法 foo 被调用时，有以下三种污点传播的模式：



实验内容

污点分析——处理污点传播

本质上，污点传播指的是某个方法调用会在调用点把污点从一些变量传播到另外一些变量中。在这样的传播过程中，我们把污点的来源称为 **from 变量**，把目标称为 **to 变量**。例如，对于一个 base-to-result 的污点传播，from 变量指的是调用点中的 base 变量，to 变量指的是调用点中接收返回值的变量。

为了处理污点传播，我们为污点分析定义一种新的输入，叫做 $TaintTranfers$。它是由四元组 $\langle m,from,to,u\rangle$ 所构成的集合，其中 m 表示会引发污点传播的方法，而污点会从 $from$ 所表示的变量中传播到 to 所表示的变量中。u 表示传播后的污点（由 to 指向）的类型。

实验内容

污点分析——处理污点传播

类型	语句	规则（在上下文 c 下）
Call (base-to-result)	`l: r = x.k(a1,...,an)`	$\dfrac{c : l \to c^t : m \in CG \quad \langle m, \text{“base”}, \text{“result”}, u\rangle \in TaintTransfers \quad [\,] : t_j^{u'} \in pt(c : x)}{[\,] : t_j^{u} \in pt(c : r)}$
Call (arg-to-base)	`l: r = x.k(a1,...,an)`	$\dfrac{c : l \to c^t : m \in CG \quad \langle m, i, \text{“base”}, u\rangle \in TaintTransfers \quad [\,] : t_j^{u'} \in pt(c : ai)}{[\,] : t_j^{u} \in pt(c : x)}$
Call (arg-to-result)	`l: r = x.k(a1,...,an)`	$\dfrac{c : l \to c^t : m \in CG \quad \langle m, i, \text{“result”}, u\rangle \in TaintTransfers \quad [\,] : t_j^{u'} \in pt(c : ai)}{[\,] : t_j^{u} \in pt(c : r)}$

实验内容

污点分析——配置污点分析

- 为了让污点分析用起来灵活，项目提供了可配置的污点分析框架。该框架支持使用 YAML 格式来配置 sources、sinks和污点传播。作业包中路径为 src/test/resources/pta/taint/taint-config.yml 的文件可以作为参考样例。

- 配置文件由一条条数据构成。表示 source 的数据格式是：

```yml
{ method: <METHOD_SIGNATURE>, type: <TYPE_NAME> }
```

实验内容

污点分析——配置污点分析

- 表示 sink 的数据格式是：

```yml
{ method: <METHOD_SIGNATURE>, index: <INDEX> }
```

- 表示污点传播的数据格式是：

```yml
{ method: <METHOD_SIGNATURE>, from: <INDEX>, to: <INDEX>, type: <TYPE_NAME> }
```

实验内容

具体任务

在这次作业中，你需要补全下面列出的两个类中的方法：

pascal.taie.analysis.pta.cs.Solver:

void addReachable(CSMethod)
void addPFGEdge(Pointer,Pointer)
void analyze()
PointsToSet propagate(Pointer,PointsToSet)
void processCall(CSVar,CSObj)

pascal.taie.analysis.pta.plugin.taint.TaintAnalysiss:

Set<TaintFlow> collectTaintFlows()

实验内容

具体任务

```java
public class Solver {

    private static final Logger logger = LogManager.getLogger(Solver.class);

    private final AnalysisOptions options;

    private final HeapModel heapModel;

    private final ContextSelector contextSelector;

    private CSManager csManager;

    private CSCallGraph callGraph;

    private PointerFlowGraph pointerFlowGraph;

    private WorkList workList;

    private boolean enableTaintAnalysis;

    private TaintAnalysiss taintAnalysis;

    private PointerAnalysisResult result;
```

```java
public class TaintAnalysiss {

    private static final Logger logger = LogManager.getLogger(TaintAnalysiss.class);

    private final TaintManager manager;

    /**
     * Map from method (which is source method) to set of types of
     * taint objects returned by the method calls.
     */
    private final MultiMap<JMethod, Type> sources = Maps.newMultiMap();

    /**
     * Map from method (which causes taint transfer) to set of relevant
     * {@link TaintTransfer}.
     */
    private final MultiMap<JMethod, TaintTransfer> transfers = Maps.newMultiMap();

    /**
     * Map from variable to taint transfer information.
     * The taint objects pointed to by the "key" variable are supposed
     * to be transferred to "value" variable with specified type.
     */
    private final MultiMap<Var, Pair<Var, Type>> varTransfers = Maps.newMultiMap();

    private final TaintConfig config;

    private final Solver solver;
```

API介绍

- **pascal.taie.analysis.pta.plugin.taint.Source**

  这个record类表示sources

- **pascal.taie.analysis.pta.plugin.taint.Sink**

  这个record类表示sinks

- **pascal.taie.analysis.pta.plugin.taint.TaintTransfer**

  这个record类表示污点传播。在这个类中，我们用整数来表示污点传播被对应方法引发时的 from 变量和 to 变量。

API介绍

- **pascal.taie.analysis.pta.plugin.taint.TaintConfig**

  这个类表示污点分析的配置信息。它提供了解析配置文件以及获取 sources、sinks 和污点传播信息的 API。

- **pascal.taie.analysis.pta.plugin.taint.TaintManager**

  这个类被用来管理污点分析中的污点对象。

- **pascal.taie.analysis.pta.plugin.taint.TaintFlow**

  这个 record 类表示分析算法检测到的 taint flows（由 source 的调用点和 sink 的调用点组成），也就是污点分析算法的结果。

API介绍

- **pascal.taie.analysis.pta.plugin.taint.TaintAnalysiss**

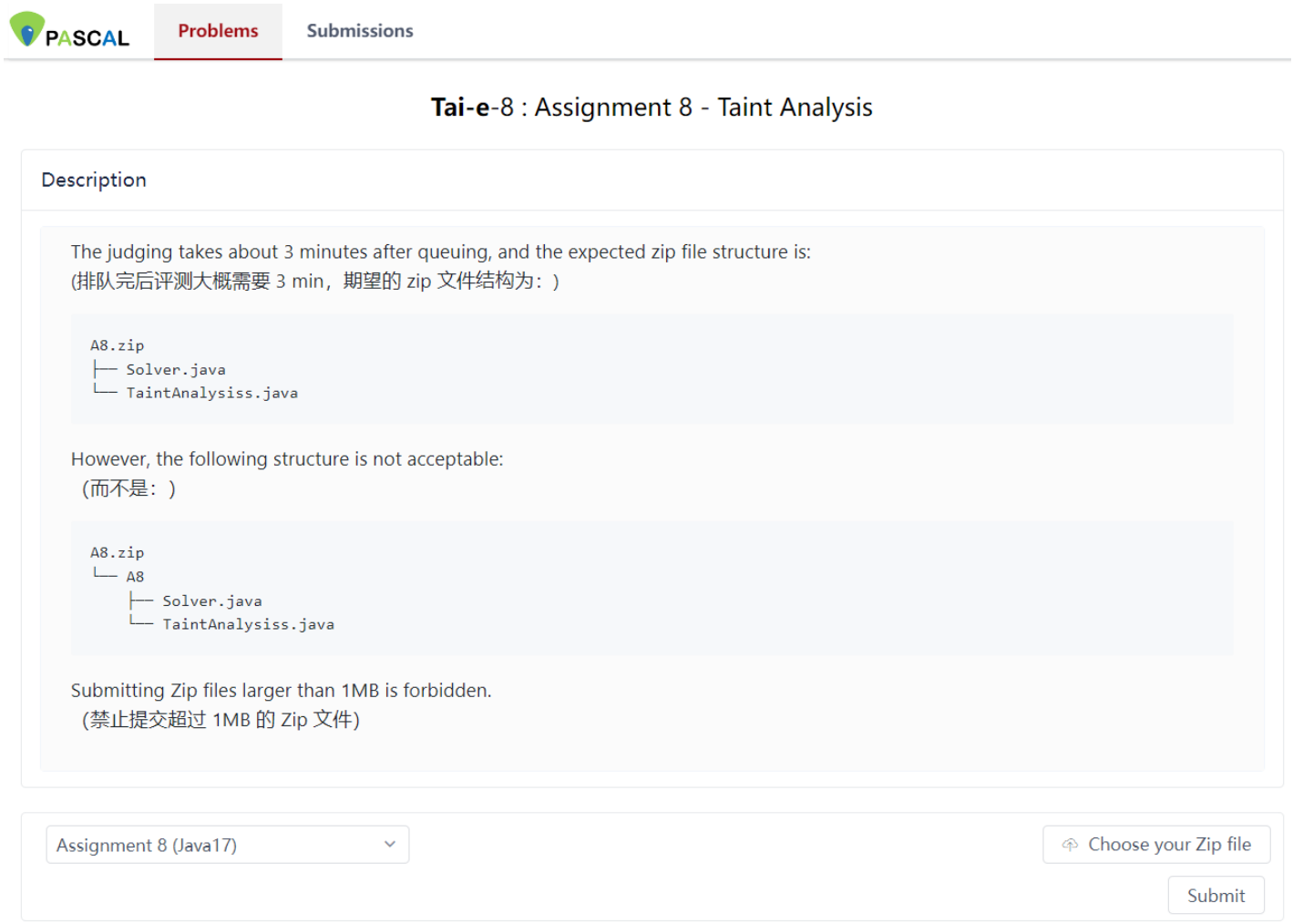
  这个类实现了污点分析，是本次实验需要补全的类。

作业提交

在线测试平台：https://oj.pascal-lab.net/problem

提交一个**zip文件**，包括实现好的以下类：

- TaintAnalysiss.java
- Solver.java



作业测试与提交

最终提交内容

- 实验报告
- 代码（zip文件），注意只需要提交本次实验的几个java文件，不要提交整个项目。

将实验报告与A8.zip文件放在同一个文件夹下，压缩后提交到下面地址中，注意实验报告包含测试截图，有未通过的测试样例也需要提交结果截图

截止时间：2024-7-7 23:59

提交地址：https://send2me.cn/ll1Dhc5l/QWGlZsuKtrrnzg