|  |  |  |
| --- | --- | --- |
| **产品名称** | | **密级** |
| **\*\*\*\*软件设计方案** | | **公开** |
| **拟制人：（签字）** | |  |
| **日期** | **2022.7.28** | **共12页** |

**\*\*\*\*软件设计方案**

**大连理工大学**

目 录

[1．问题与背景 4](#_Toc133310514)

[2. 需求分析 4](#_Toc133310515)

[2.1 用户需求分析 4](#_Toc133310516)

[2.2 功能需求分析 4](#_Toc133310517)

[2.3 非功能需求 5](#_Toc133310518)

[2.4运行需求 5](#_Toc133310519)

[3. 概要设计 5](#_Toc133310520)

[3.1 总体设计 5](#_Toc133310521)

[3.3 数据结构设计 6](#_Toc133310522)

[3.4 运行设计 6](#_Toc133310523)

[3.5 异常出错处理设计 6](#_Toc133310524)

[3.2 接口设计 6](#_Toc133310525)

[4. 问题汇总 7](#_Toc133310526)

[5.附录 7](#_Toc133310527)

[5.1术语及缩略语 7](#_Toc133310528)

[5.2参考资料 7](#_Toc133310529)

修订记录

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 修订版本 | 修改者 | 修改描述 |
| 2023.04.27 | 1.0 | 高鑫 |  |
|  |  |  |  |
|  |  |  |  |

# 1．问题与背景

表达式语言（Expression Language），又称EL表达式，是一种在JSP中内置的语言，可以作用于用户访问页面的上下文以及不同作用域的对象，取得对象属性值或者执行简单的运算和判断操作。

OGNL全称Object-Graph Navigation Language即对象导航图语言，一种功能强大的表达式语言。OGNL可以存取对象的任意属性并且调用对象的方法，还可以遍历整个对象的结构图，实现字段类型转化。webwork2 和 Struts2.x 中使用 OGNL 代替原来的 EL 来做界面数据绑定（就是把textfield.hidden和对象层某个类的某个属性绑定在一起，修改和现实自动同步）Struts2框架因为滥用OGNL表达式，所以漏洞较多。OGNL具有三要素: 表达式、ROOT对象、上下文环境（MAP结构）。处理OGNL的最顶层对象是一个Map对象，通常称这个Map对象为context map或者context，OGNL的root就在这个context map中，在表达式中可以直接引用root对象的属性。OGNL是一种功能强大。的语言，但在Apache Struts中将用户提供的输入视为OGNL会影响安全性。

Spring表达式语言（简称SpEl）是一个支持查询和操作运行时对象导航图功能的强大的表达式语言. 它的语法类似于传统EL, 但提供额外的功能, 最出色的就是函数调用和简单字符串的模板函数。Spel创建的初衷是了给Spring社区提供一种简单而高效的表达式语言, 一种可贯穿整个Spring产品组的语言, 这种语言的特性应基于Spring产品的需求而设计. 虽然SpEL引擎作为Spring组合里的表达式解析的基础, 但它不直接依赖于Spring, 可独立使用。

# 需求分析

## 2.1 用户需求分析

#### 2.2.1 OGNL注入检测

OGNL可以访问静态方法、属性以及对象方法等, 其中包含可以执行恶意操作如命令执行的类java.lang.Runtime等, 当OGNL表达式外部可控时, 攻击者就可以构造恶意的OGNL表达式来让程序执行恶意操作, 这就是OGNL表达式注入漏洞。比如这个具有代表性的注入语句：@java.lang.Runtime@getRuntime().exec("open -a Calculator")，它调用Runtime类来执行系统命令完成OGNL注入。

OGNL表达式是Struts2的默认表达式语言, 所以只针对Struts2标签有效。Struts2中的getValue函数和setValue函数都能成功解析恶意的OGNL表达式.由于不同版本的Struts2的漏洞成因不同，所以针对不同版本CVE的注入语句格式也不同。例如Struts2 2.0.0 - 2.0.8版本下，存在CVE漏洞S2-001，漏洞成因是当参数值是形如%{string}的形式的时候，ST2会把string这个值当做OGNL表达式去执行。

我们需要检测出已有的每个版本的CVE漏洞以及包含的注入语句，并且通过构建相关注入语句的文法可以够防御更多注入语句的组合。

#### 2.2.2 SPEL注入检测

SimpleEvaluationContext和StandardEvaluationContext是SpEL提供的两个EvaluationContext。SimpleEvaluationContext: 针对不需要SpEL语言语法的全部范围并且应该受到有意限制的表达式类别, 公开SpEL语言特性和配置选项的子集。StandardEvaluationContext: 公开全套SpEL语言功能和配置选项, 可以使用它来指定默认的根对象并配置每个可用的评估相关策略.SimpleEvaluationContext旨在仅支持SpEL语言语法的一个子集, 不包括Java类型引用、构造函数和bean引用; 而StandardEvaluationContext是支持全部SpEL语法的。

SpEL表达式是可以操作类及其方法的, 可以通过类类型表达式T(Type)来调用任意类方法. 这是因为在不指定EvaluationContext的情况下默认采用的是StandardEvaluationContext, 而它包含了SpEL的所有功能, 在允许用户控制输入的情况下可以成功造成任意命令执行。

## 2.2 功能需求分析

#### 2.2.1功能划分

#### 2.2.2功能描述

## 2.3 非功能需求

#### 2.3.1数据精度需求

友商检测数据精度对比如图2-1，图2-2：

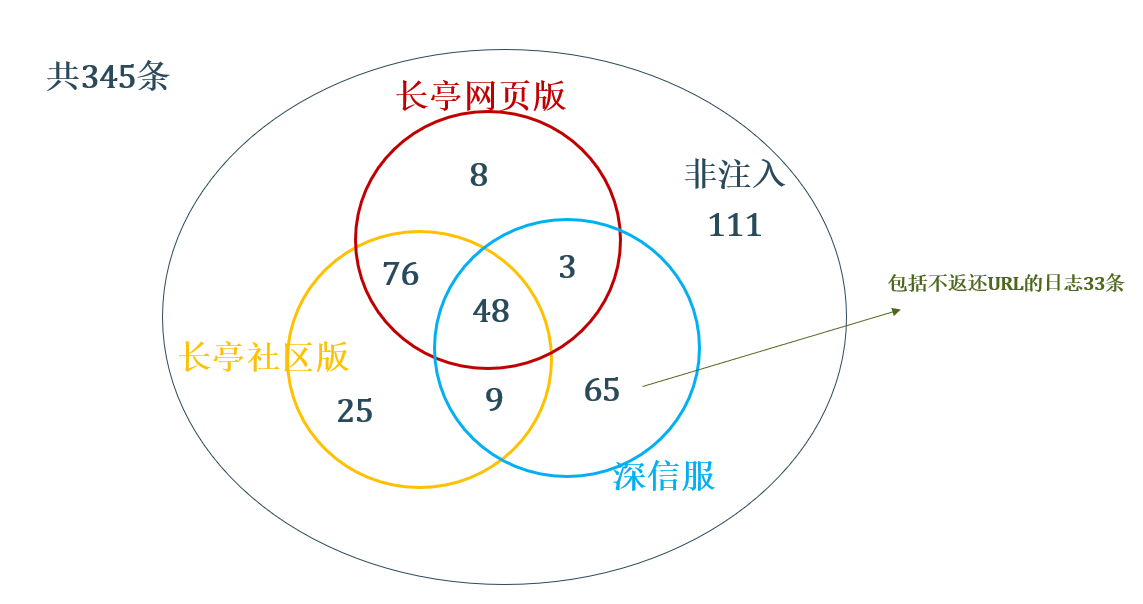


图2-1

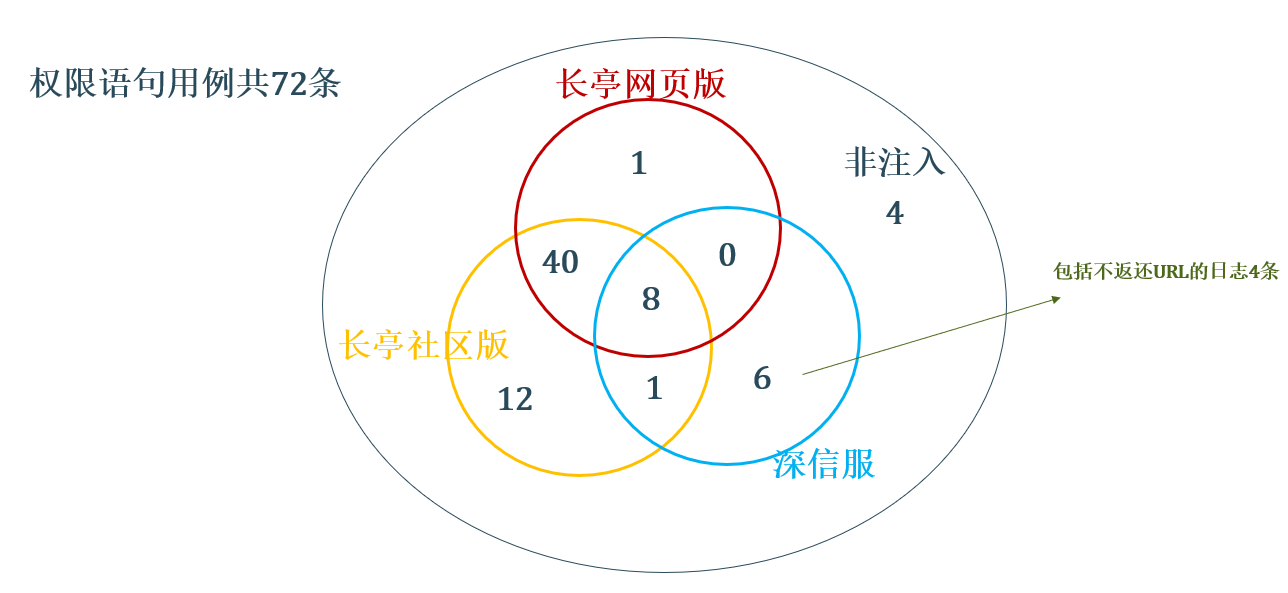


图2-2

长亭社区版demo语法弹性较大，检出的语句较多，但是同时也存在盲点，比如：java.io.File类下的一些使用。长亭网页版检测结果略微少于长亭社区版，但是大体上检测逻辑跟检出语句相同。深信服的检出语句较少，要求java类必须执行调用方法才能识别注入，不存在单独使用java类就能构成注入的情况，并且深信服对安全权限语句的要求严格。

#### 2.3.2性能需求

*【如响应时间、更新处理时间、数据转换与传输时间、运行时间等。】*

## 2.4运行需求

#### 2.4.1用户接口

测试单条输入语句。用户输入测试语句，检测模型返回语句测试结果。

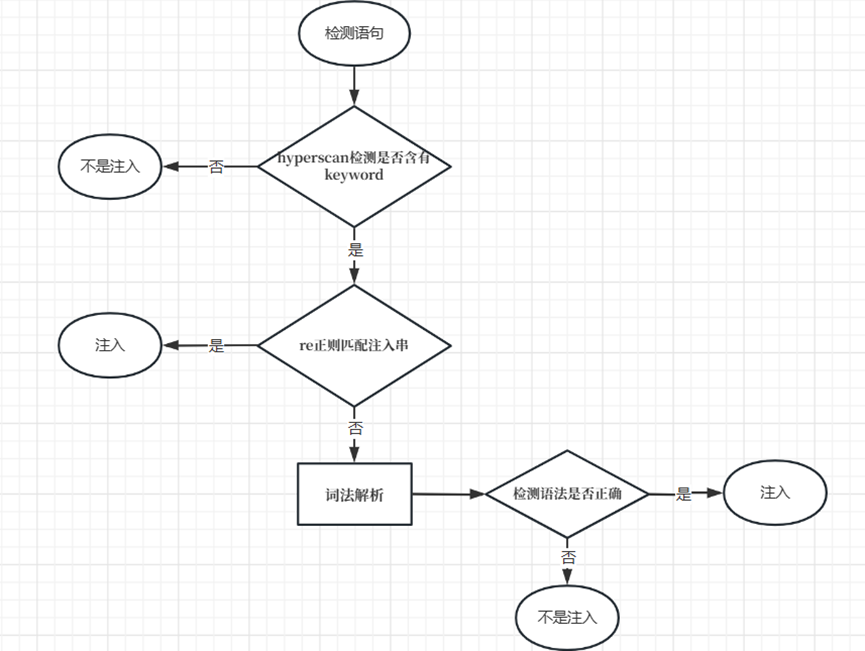
测试包含多条语句的txt文档。用户输入测试文档给检测模型，模型返回两个结果文档，一个注入语句组成的文档和一个非注入语句的文档。

# 概要设计

## 3.1 总体设计

#### 3.1.1处理流程

流程图如下图3-1



#### 3.1.2总体结构和模块外部设计

#### 3.1.3功能分配

Hyperscan：Hyperscan是一种高性能、多模式的正则表达式匹配引擎，能够在非常短的时间内快速地匹配输入数据中的多个正则表达式。在模型的第一阶段运用Hyperscan进行白噪音筛选，尽可能的筛选掉不可能构成注入的数据，避免大量的白噪音数据进入模型导致模型运行效率降低，降低白噪音对检测结果的影响。

Re正则匹配：我们将常见的注入语句中，影响注入权重较高的部分，提取出来作为注入成分库。在模型的第二阶段使用Re正则匹配，如果我们的检测语句与注入成分库中的注入片段成功匹配，确认检测语句可能完成注入或有注入倾向，判定该语句有注入风险，为注入语句。

词法分析：将输入语句转换为token流，token流用于语法解析。

语法分析：将词法分析阶段产生的token流，基于我们定义的语法文件进行语法分析。

## **3.2 接口设计**

#### 3.2.1外部接口

命令行：./test 测试单条输入语句。

命令行：./data.txt 测试包含多条语句的txt文档。

#### 3.2.2内部接口

Hyperscan(): Hyperscan白噪音筛选模块

reDetection(): Re正则匹配模块

ognlDetection(): OGNL语法检测模块

spelDetection(): SPEL语法检测模块

## 3.3 数据结构设计

#### 3.3.2数据结构设计

#### 3.3.3数据结构与程序的关系

## 3.4 运行设计

*运行时系统包括的进程，软件组件等动态对象及交互，例如用时序图描述。*

## 3.5 异常出错处理设计

#### 3.5.1出错输出信息

#### 3.5.2出错处理对策

*【如设置后备、性能降级、恢复及再启动等。】*

# 问题汇总

## 4.1 特殊用例

#### 4.1.1 长亭网页版demo检测不稳定的语句

有时候是nomal 有时候是high

%{@java.lang.Class}

%{@java.lang.ClassLoader}

#### 4.1.2 ognl.OgnlContext相关检测

%{@ognl.OgnlContext@DEFAULT\_MEMBER\_ACCESS, @a@a()} high

%{@ognl.OgnlContext} high

%{#a=@ognl.OgnlContext@DEFAULT\_MEMBER\_ACCESS,@a@a()} normal

%{#\_memberAccess=@ognl.OgnlContext,@sun.misc.Unsafe@getUnsafe()} normal

#### 4.1.3 ProcessBuilder相关检测

%{@java.lang.ProcessBuilder@command(List<String> command)} high

%{@a()@java.lang.ProcessBuilder} high

%{@ProcessBuilder@a()} high

%{@ProcessBuilder@a} high

%{@a@ProcessBuilder} high

%{@a()@ProcessBuilder} normal

## 4.2 数据统计

#### 4.2.1 注入相关的java类

注入相关的Java类是怎么提取的，怎么保证是与注入相关的？  
 注入相关的Java类是从已有的注入数据中提取的使用到的Java类，我们构造注入用例验证的能完成注入的Java和验证的conflunce的黑名单中提及到的危险类。

java.lang.ProcessBuilder

Java.lang.RuntimePermission

Java.lang.ClassLoader

Java.lang.System

java.lang.ThreadGroup

Java.lang.Compiler

java.lang.Thread

java.lang.Class

java.lang.ThreadLocal

java.lang.Package

java.lang.Runtime

java.lang.SecurityManager

Java.lang.Object

java.lang.InheritableThreadLocal

java.lang.Shutdown

java.lang.Process

sun.misc.Unsafe

Java.io.File

java.io.BufferedReader

java.io.DataInputStream

javax.script.ScriptEngineManage

javax.persistence.EntityManager

javax.servlet.ServletContext

Java.io.FileDescriptor

Java.io.FileInputStream

Java.io.FileOutputStream

Java.io.FilePermission

Java.io.FileReader

Java.io.FileWriter

java.util.Collections

java.util.Scanner

#### 4.2.2 注入成分库

注入成分库如何提取，如何保证是完备的？

注入成分库是用于re正则匹配模块，主要是payload中常见的字符串，例如：java.lang.ProcessBuilder这种不用与权限语句搭配使用，单独出现就被检测为注入的类，以及权限语句中常出现有功能作用的字符串。注入成分库里的字符串用于re正则匹配，如果检测语句中匹配到注入成分库中的字符串，即被检测为注入。

Re正则匹配用于语法分析之前，未被检出的语句还会进入语法阶段进行语法检测。

java.lang.ProcessBuilder

Java.lang.Runtime

Java.lang.ClassLoader

Java.lang.System

javax.script.ScriptEngineManager

java.lang.Shutdown

java.lang.Class

Java.io.FileDescriptor

Java.io.FileInputStream

Java.io.FileOutputStream

Java.io.FilePermission

Java.io.FileReader

Java.io.FileWriter

Java.io.File

ognl.OgnlContext

DEFAULT\_MEMBER\_ACCESS

#### 4.2.3 白噪音筛选的keyword

白噪声keyword有什么，如何提取？

主要是词法阶段定义的keyword以及注入语句必须存在的一些词，包括java类的类名跟包名，以及注入语句（安全权限语句）中常见的方法名，属性名。

# 5.附录

【列出文当中所用到的专门术语的定义和缩写词的原文。】

## 5.1术语及缩略语

Root对象:OGNL的Root对象可以理解为OGNL的操作对象. 当指定了一个表达式的时候, 需要指定这个表达式针对的是哪个具体的对象. 而这个具体的对象就是Root对象, 这就意味着, 如果有一个OGNL表达式, 则需要针对Root对象来进行OGNL表达式的计算并且返回结果.

上下文环境:在OGNL的内部, 所有的操作都会在一个特定的数据环境中运行. 这个数据环境就是上下文环境(Context).OGNL的上下文环境是一个Map结构, 称之为OgnlContext.Root对象也会被添加到上下文环境当中去, 简而言之, 上下文就是一个MAP结构, 它实现了java.utils.Map的接口.

OGNL:全称Object-Graph Navigation Language即对象导航图语言，一种功能强大的表达式语言.

CVE漏洞：Common Vulnerability and Exposures，CVE是一个公开已知的网络安全漏洞和暴露的列表。列表中的每一项都是基于在特定软件产品中发现的特定漏洞或暴露，而不是基于一般类别或类型的漏洞或暴露。

## 5.2参考资料

#### 5.2.1 参考网页

<https://www.jianshu.com/p/e3c77c053359>

<https://www.freebuf.com/articles/web/325700.html>

<https://mp.weixin.qq.com/s/nCMtSD7QH8ai6fpurJBXTg>

<https://blog.csdn.net/weixin_53972936/article/details/127590995>

https://blog.csdn.net/sinat\_35821285/article/details/78510184