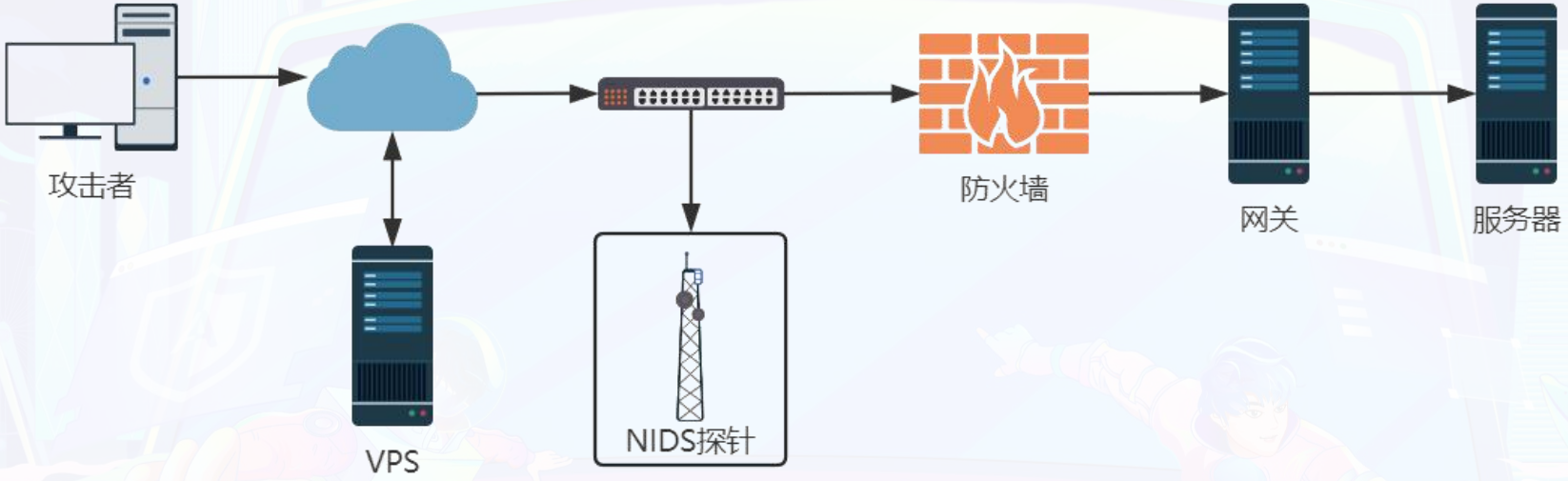
RASP检测能力提升的思考与实践

许纬地 @turn1tup

目 录

1.想法由来与发展-蓝军攻防演练



参与了公司多次蓝军演练，相关安全产品的检测能力未能尽如人意。

1.想法由来与发展-回顾WAF类产品

从传统防火墙到RASP，产品安全检测能力（上限）不断在提升，当然，他们也各有定位。

而在云原生场景下，甲方可能会愈发重视RASP的建设。

1.想法由来与发展-不成熟的想法

是否能做到这些?

- 避免单一黑名单策略
- 0-DAY漏洞检测
- 规则策略修改更简单
- RASP策略简单，主要上报日志

实践过程中:

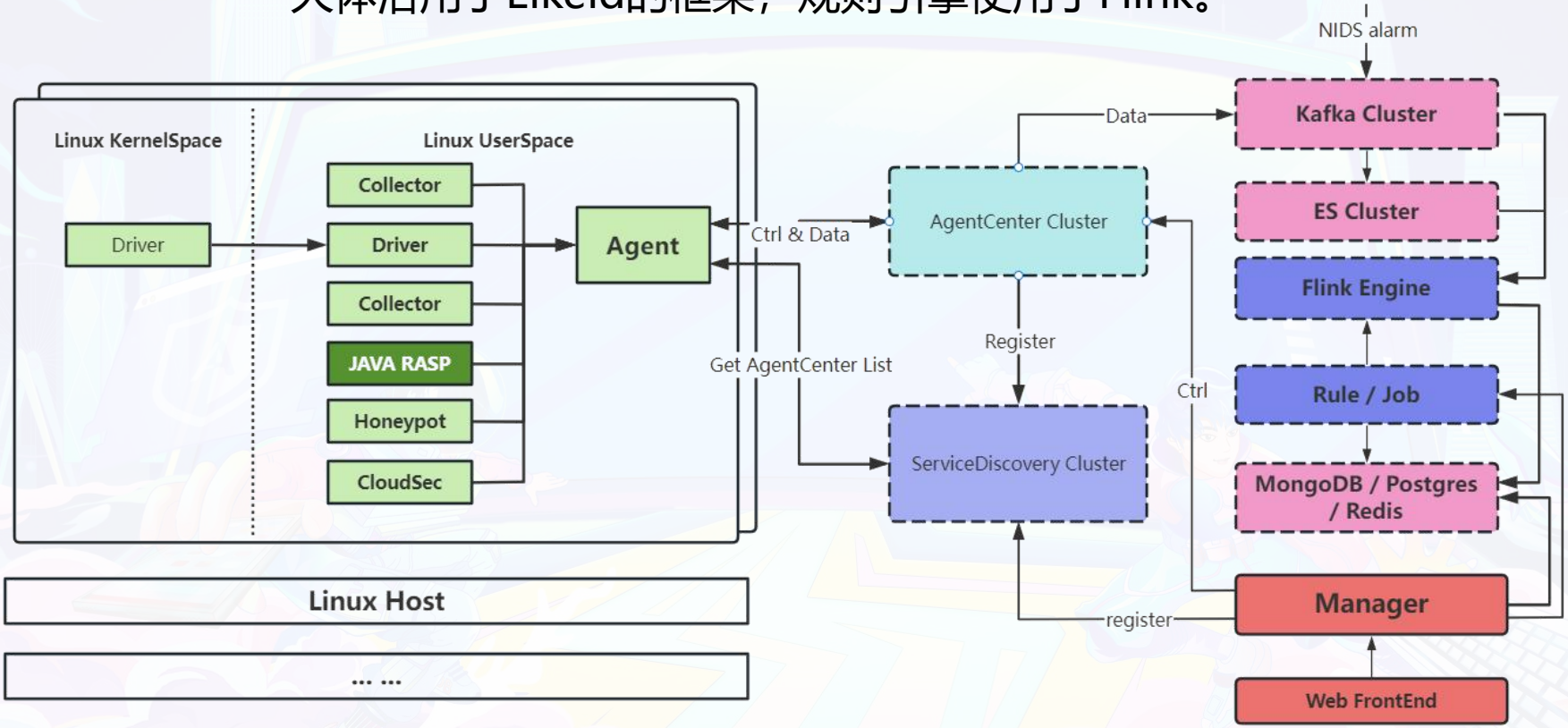
- 系统复杂度高，个人精力有限
- 建设成本提升，后台机器、带宽
- 对运营方的能力要求更高
- 非一蹴而就，部分策略放在RASP
- 优化日志上报过滤策略 ...

二、整体架构和思路

我们先从HIDS架构说起，再到RASP的架构策略

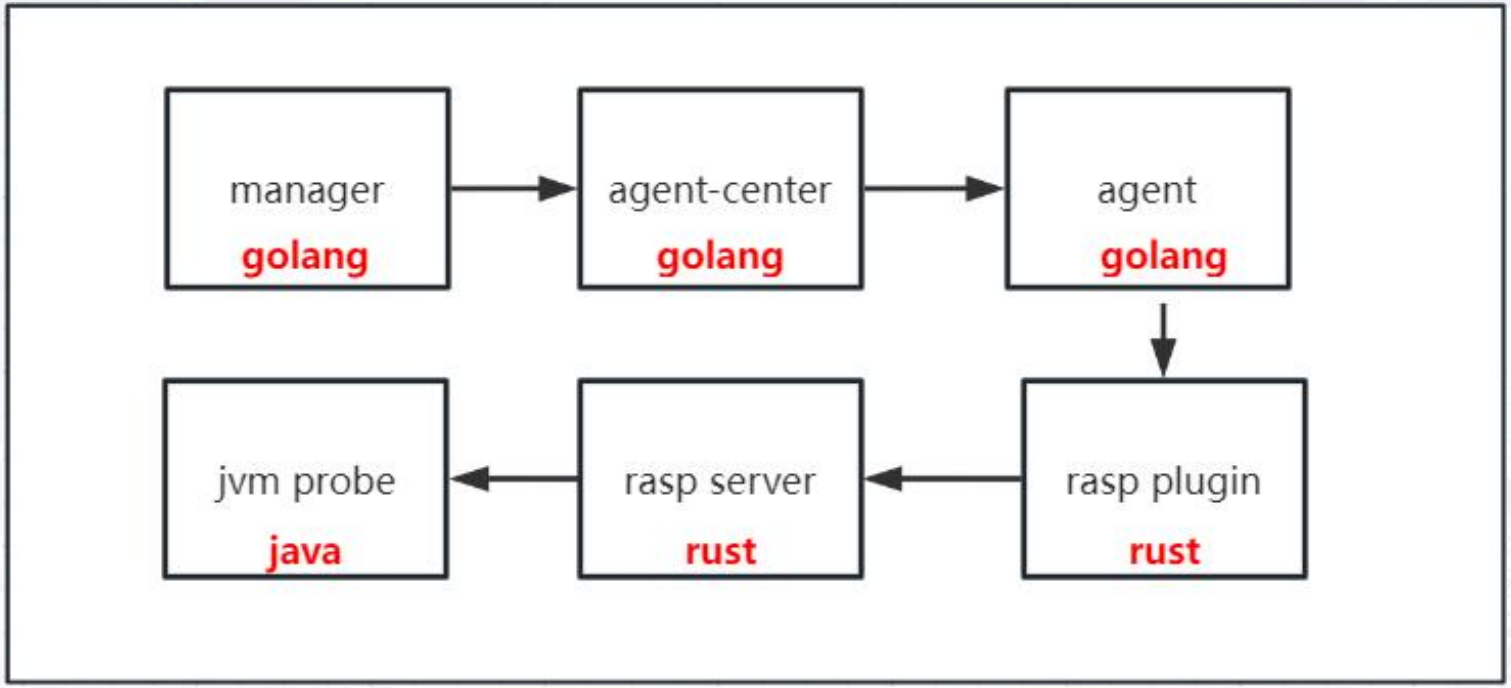
2.整体架构与思路-HIDS架构

大体沿用了Elkeid的框架，规则引擎使用了Flink。

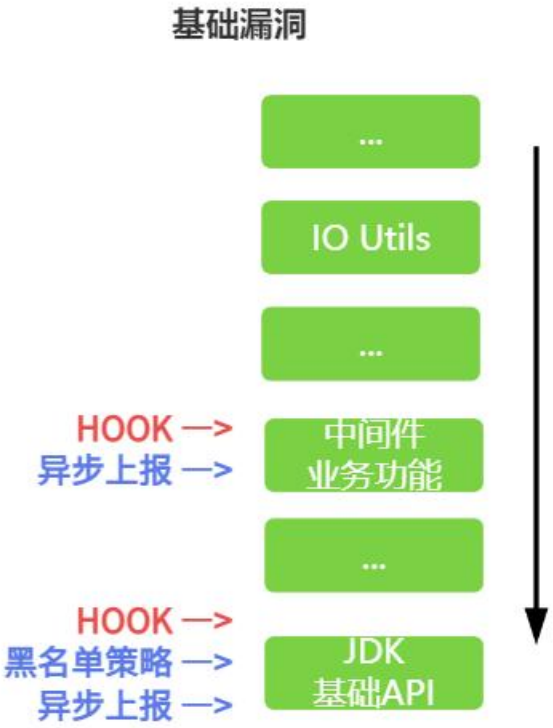


2.整体架构与思路-HIDS架构

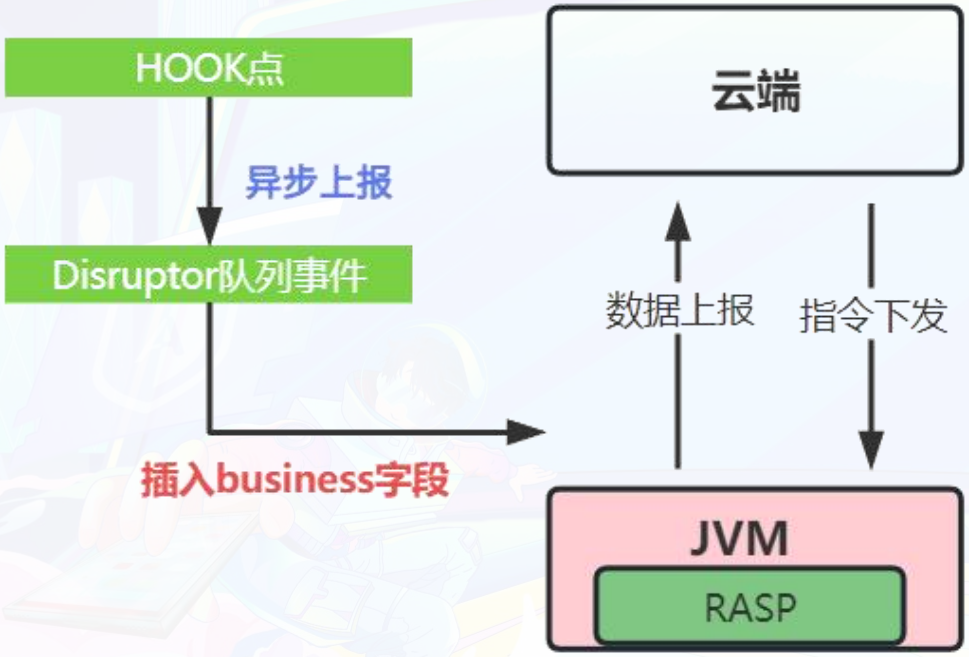
系统复杂得让人抓狂，后端到RASP：



2.整体架构与思路-RASP钩挂策略



2.整体架构与思路-RASP双端策略



RASP端上报钩挂点数据到云端

云端可下发各种指令

2.整体架构与思路-双端策略

端侧：

- HOOK点：
  - 流量数据
  - 组件漏洞
  - 中间件业务功能
  - JDK基础API

策略：

- 本地黑名单、SQL语法引擎
- 判断是否异常类加载
- HOOK点联动
- 合理上报HOOK点数据
- RASP自身防护

2.整体架构与思路-RASP双端策略

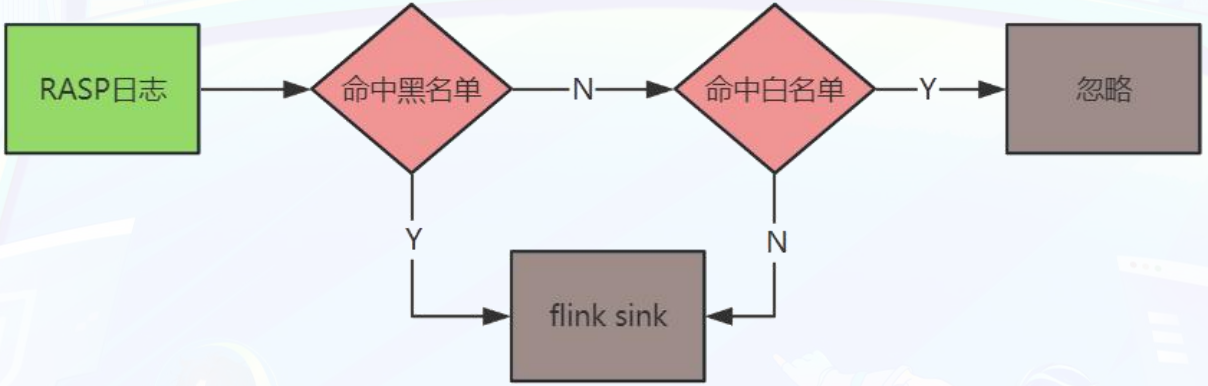
云端：

- 漏洞检测
  - 黑名单规则
  - 黑白灰名单规则
- 后门检测：
  - 业务类后门行为
  - 异常加载的类执行敏感行为
  - so、远程jar、黑名单反射、异常类加载
  - 后渗透命令行为检测（未完成）
- 误报过滤
- 指令下发
  - RASP开关
  - 下发、调整HOOK点
  - 调整端侧黑名单策略、拦截开关
  - 热补丁
  - ...

三、主要检测策略

介绍几条特别的规则策略：黑白灰名单、异常类加载、后门检测、反射黑名单，最后简要介绍整体安全能力。

3.检测策略-黑白灰名单

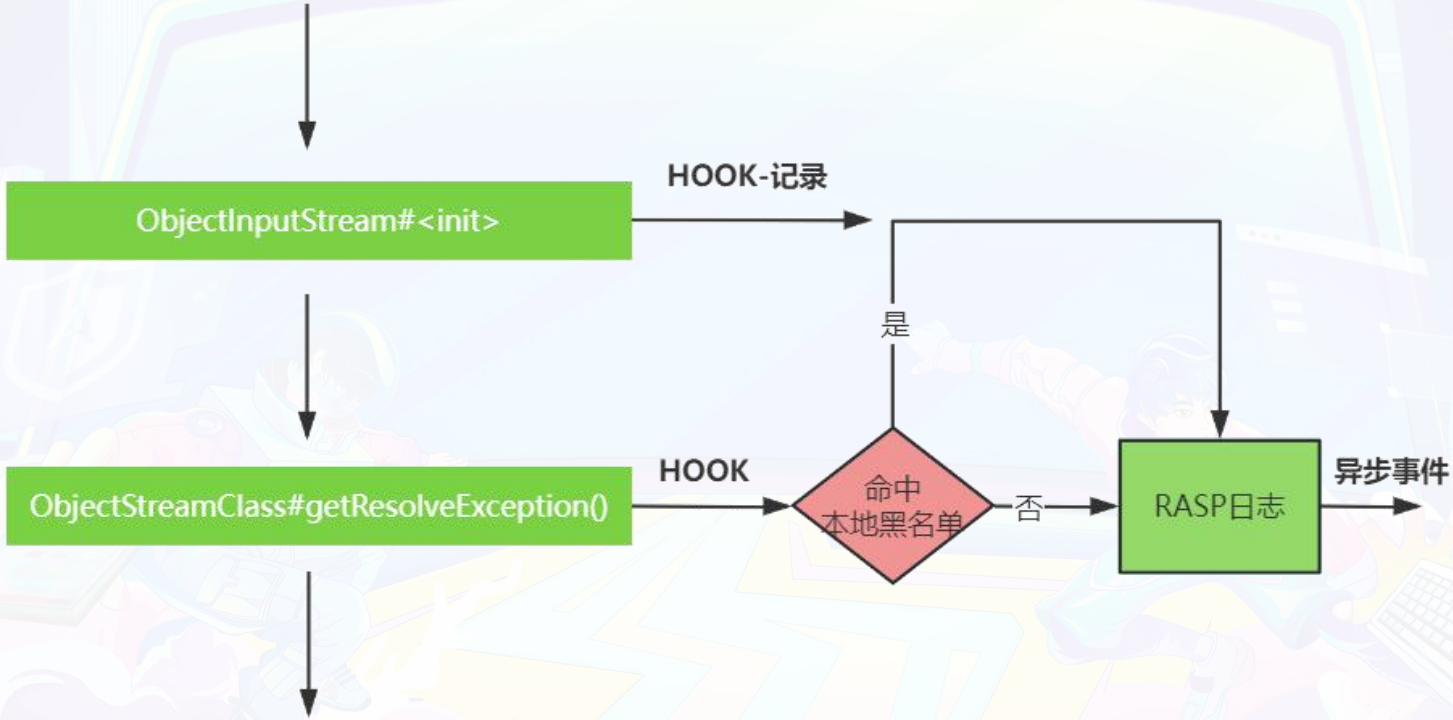


数据上报到云端后，对于未命中黑名单也未命中白名单的日志判“灰”，判灰的直接告警。

通过这种规则策略，安全运营方可以应对反序列化类的零日漏洞。

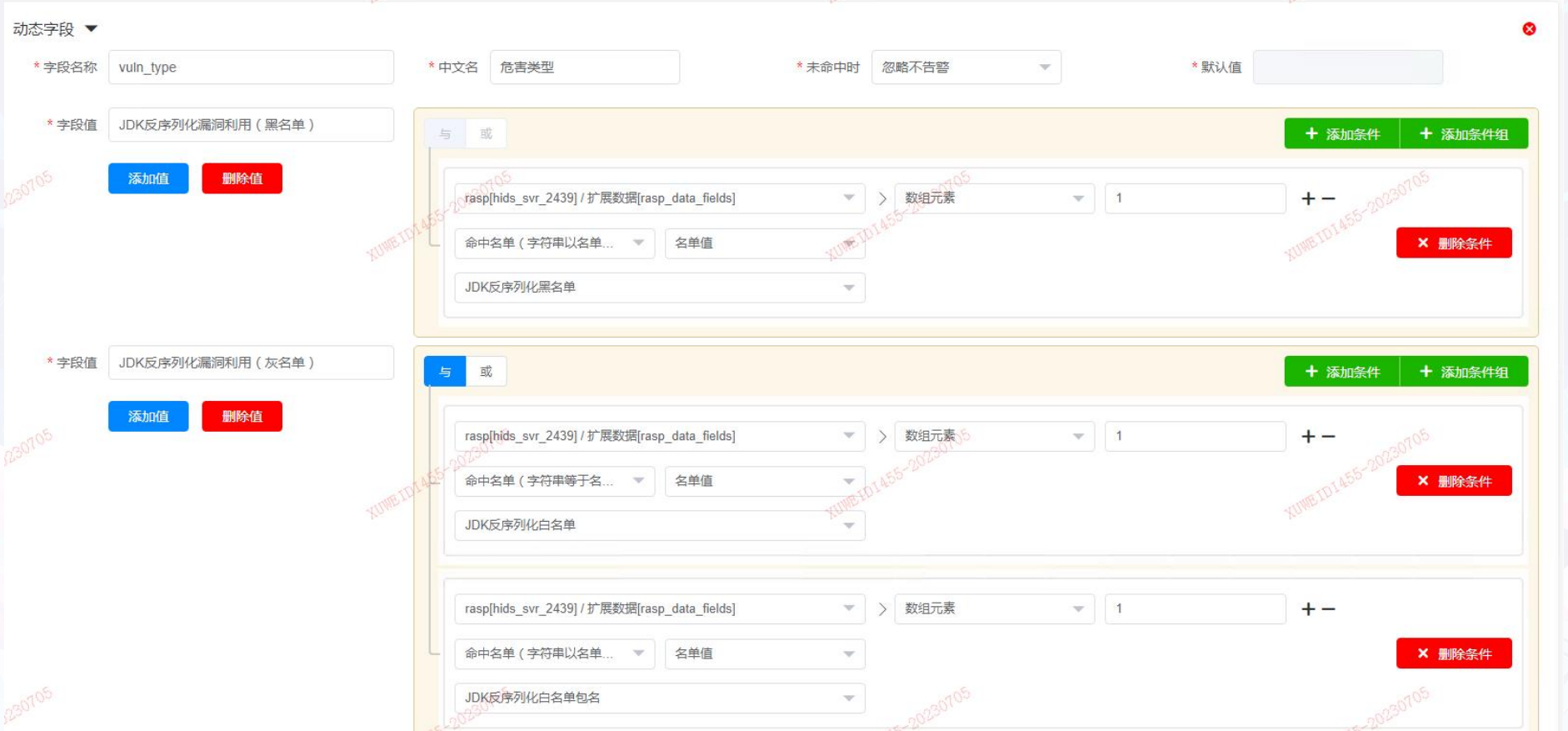
3.检测策略-黑白灰名单

JDK反序列化相关钩挂点逻辑抽象如图



3.检测策略-黑白灰名单

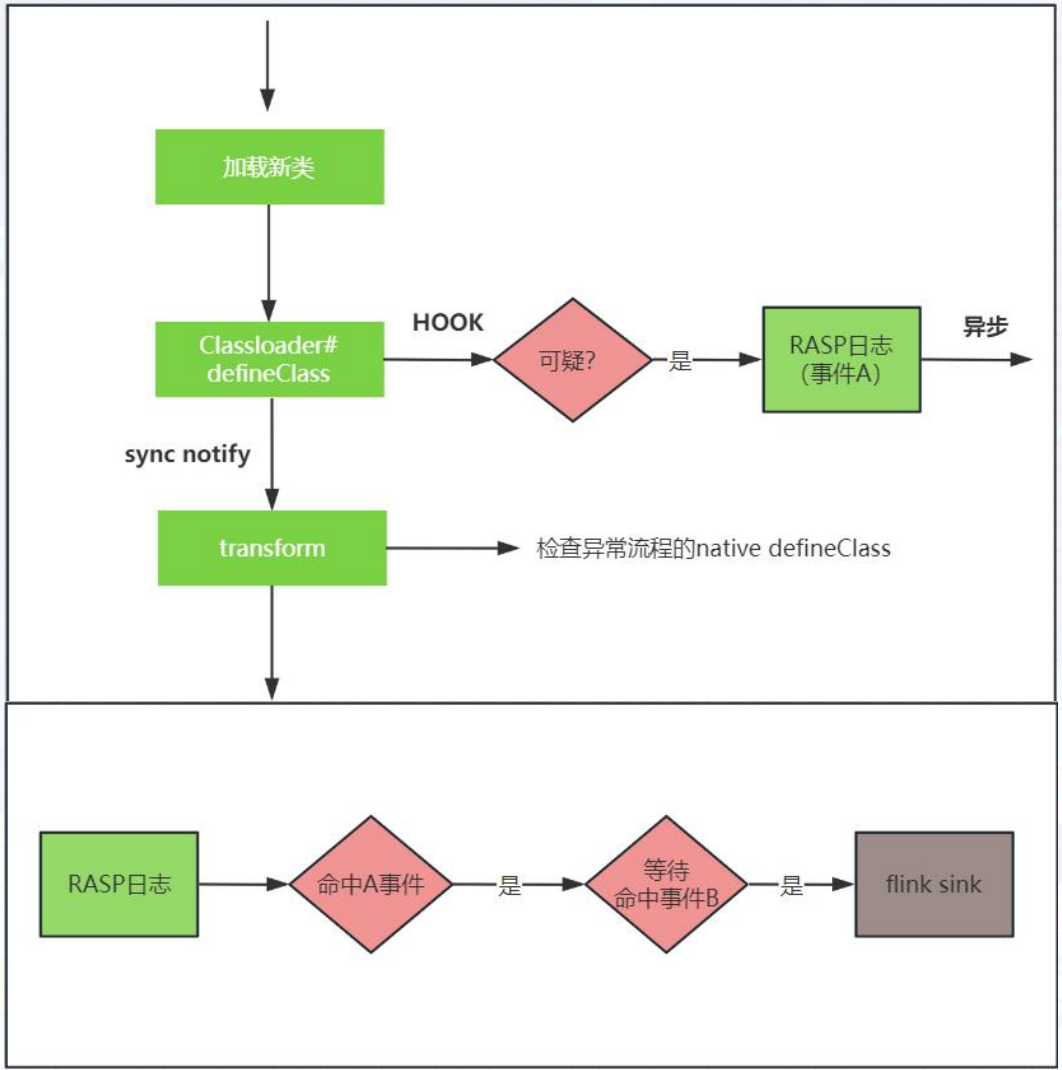
JDK反序列化防护规则展示



3.检测策略-异常类执行敏感方法

JAVA进程加载新类时RASP会判断本次加载是否异常行为，如是则会上报云端。

云端注意到这个异常类后，会持续监控这个异常类是否有其他敏感行为。



3.检测策略-异常类执行敏感方法

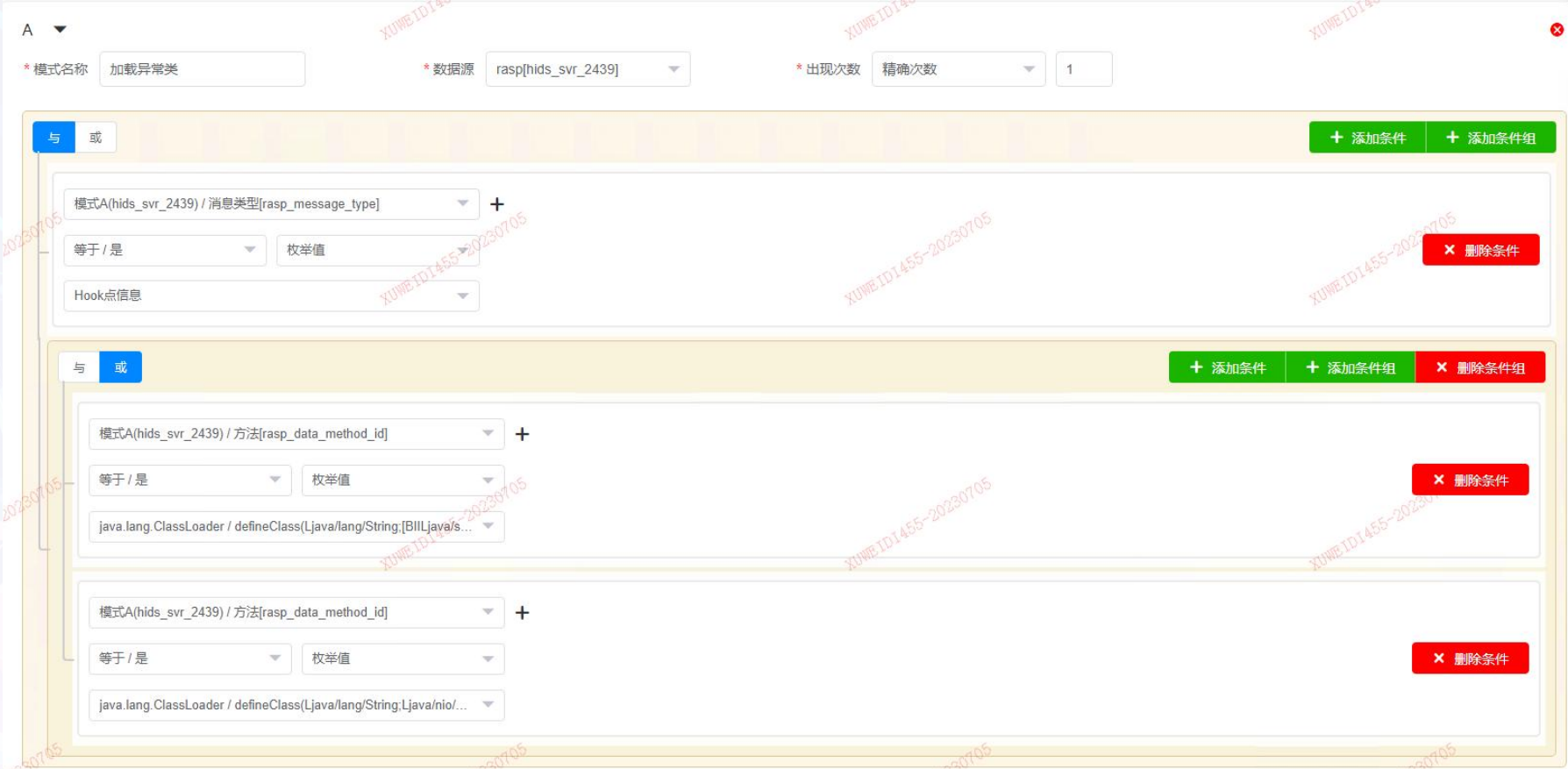
新类的类加载行为是否异常：

- 是否存在其他类加载器
- 类加载器是否异常、是否特殊类加载器
- 特殊类加载器 白名单检查调用流程
- 远程加载的类、类来源上传的JAR
- 自定义的类加载器
- 其他 ...

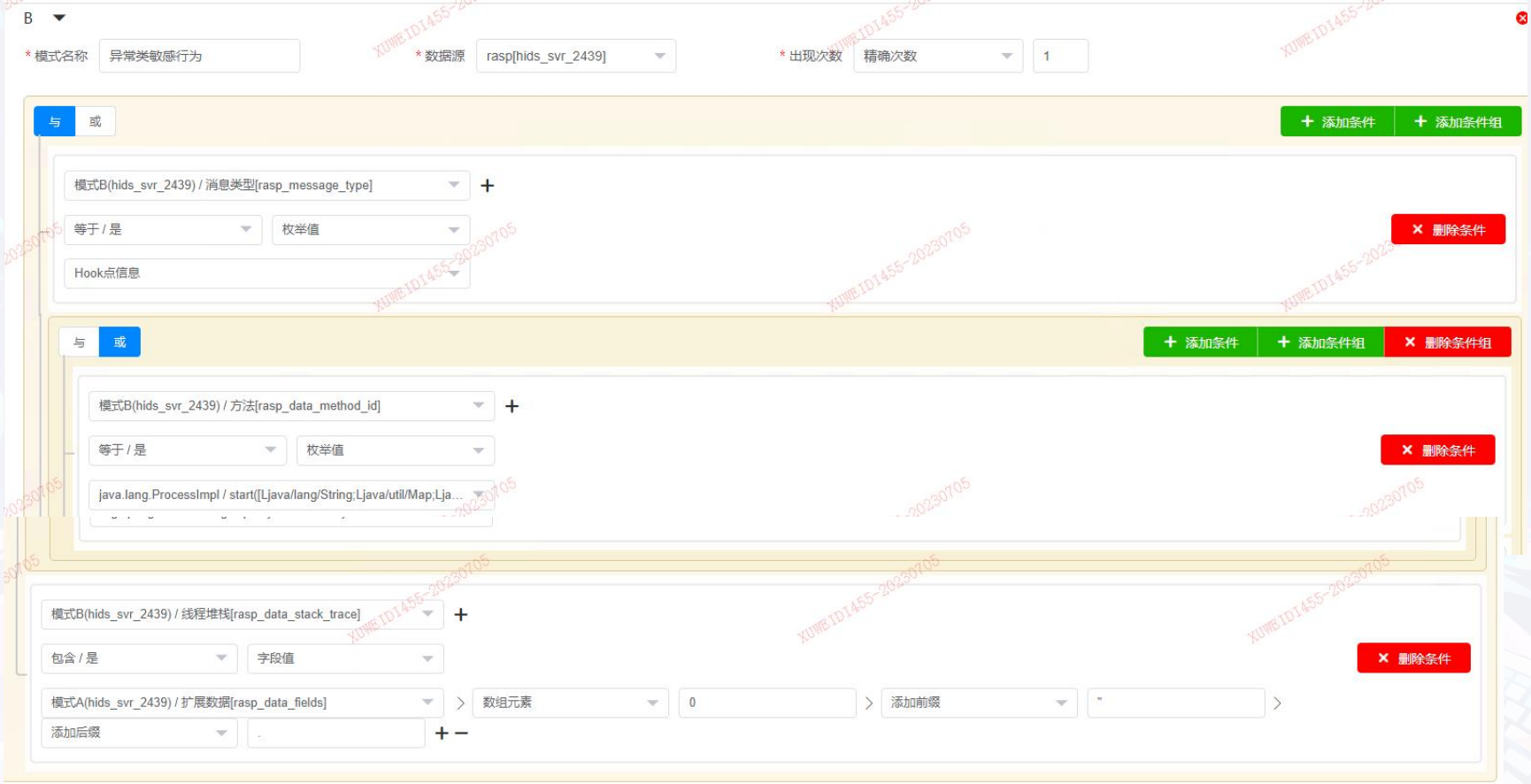
敏感方法包括：

- 文件读写
- 命令执行
- 类加载
- 等等 ...

3.检测策略-异常类执行敏感方法

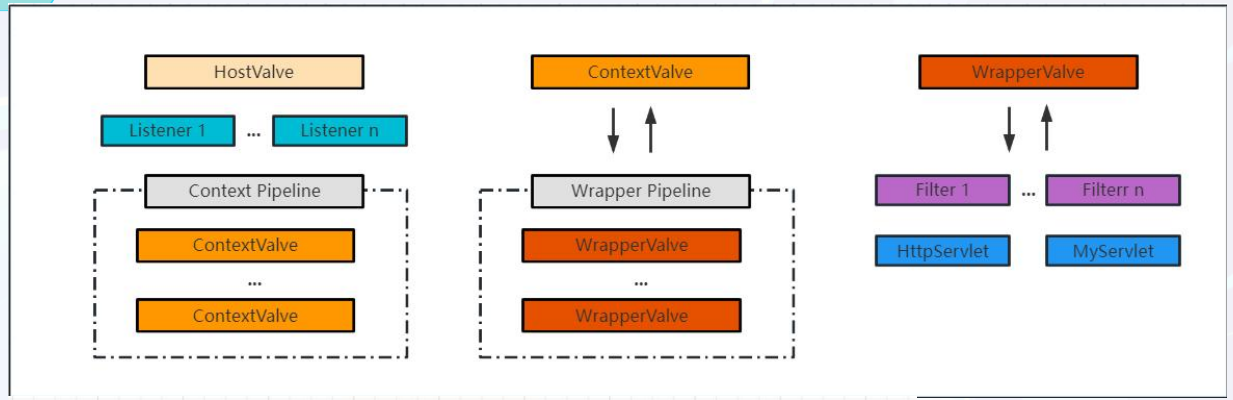


3.检测策略-异常类执行敏感方法



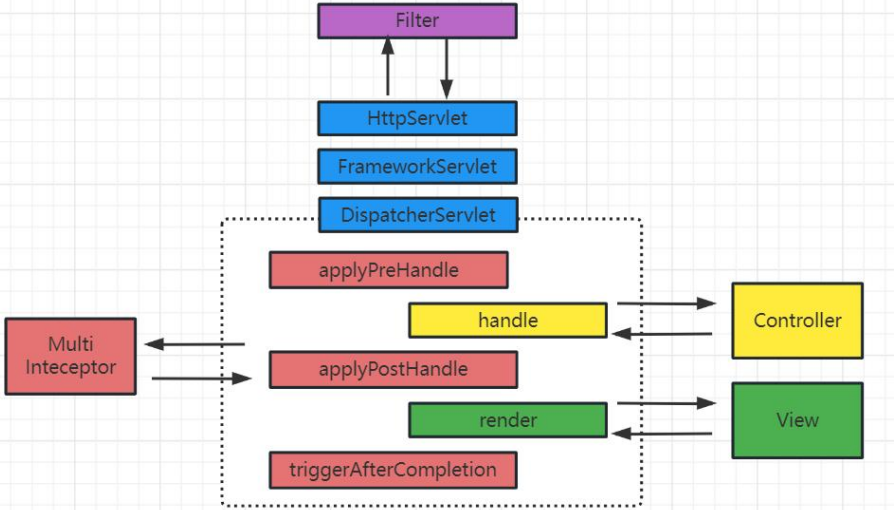
3.检测策略-后门行为

spring-web的流程可抽象为右边两张图，其间可穿插各类内存马...

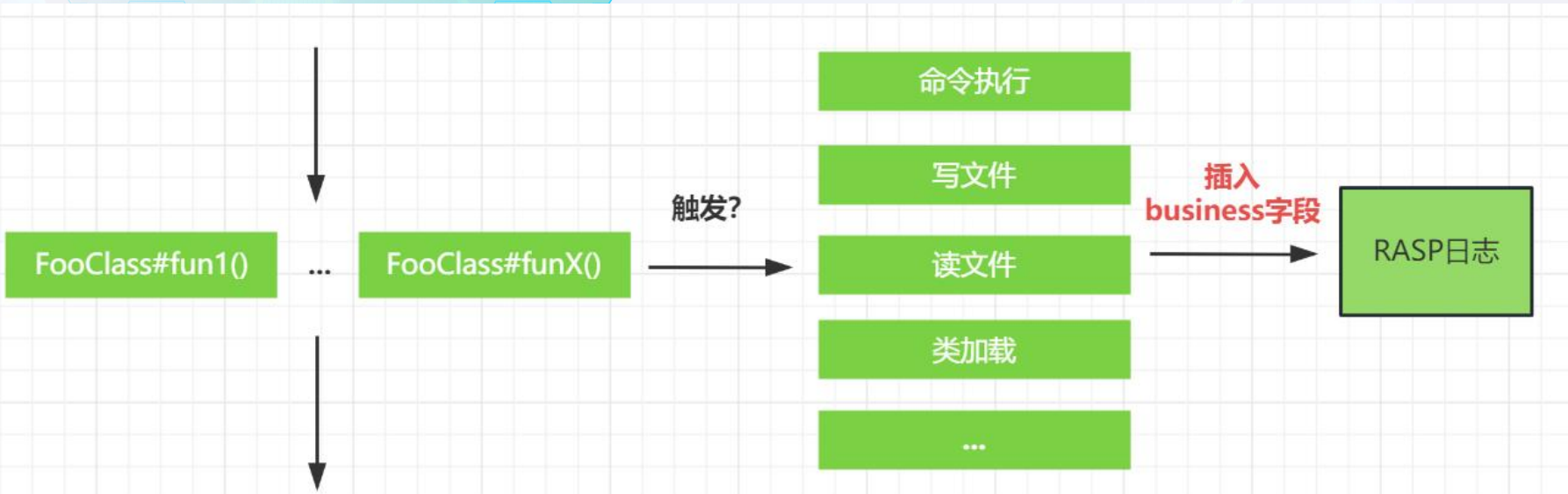


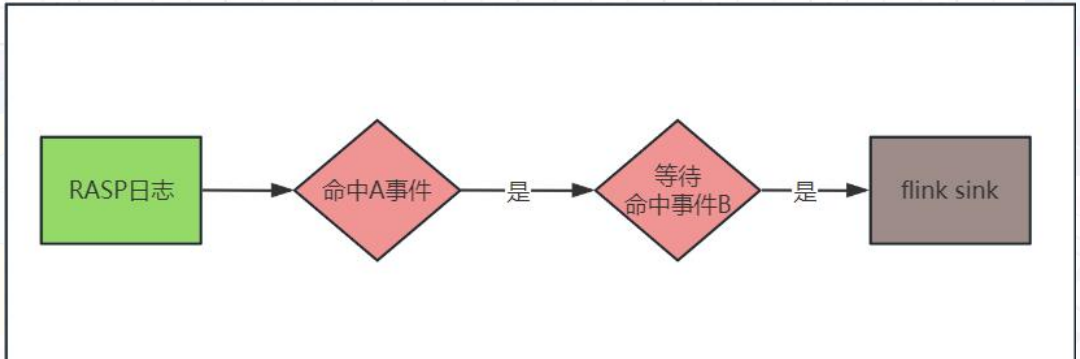
我们目前并不针对内存马做“订制”规则。

一方面，除了前文的异常类检测，这里的“后门行为检测”也可有所覆盖。

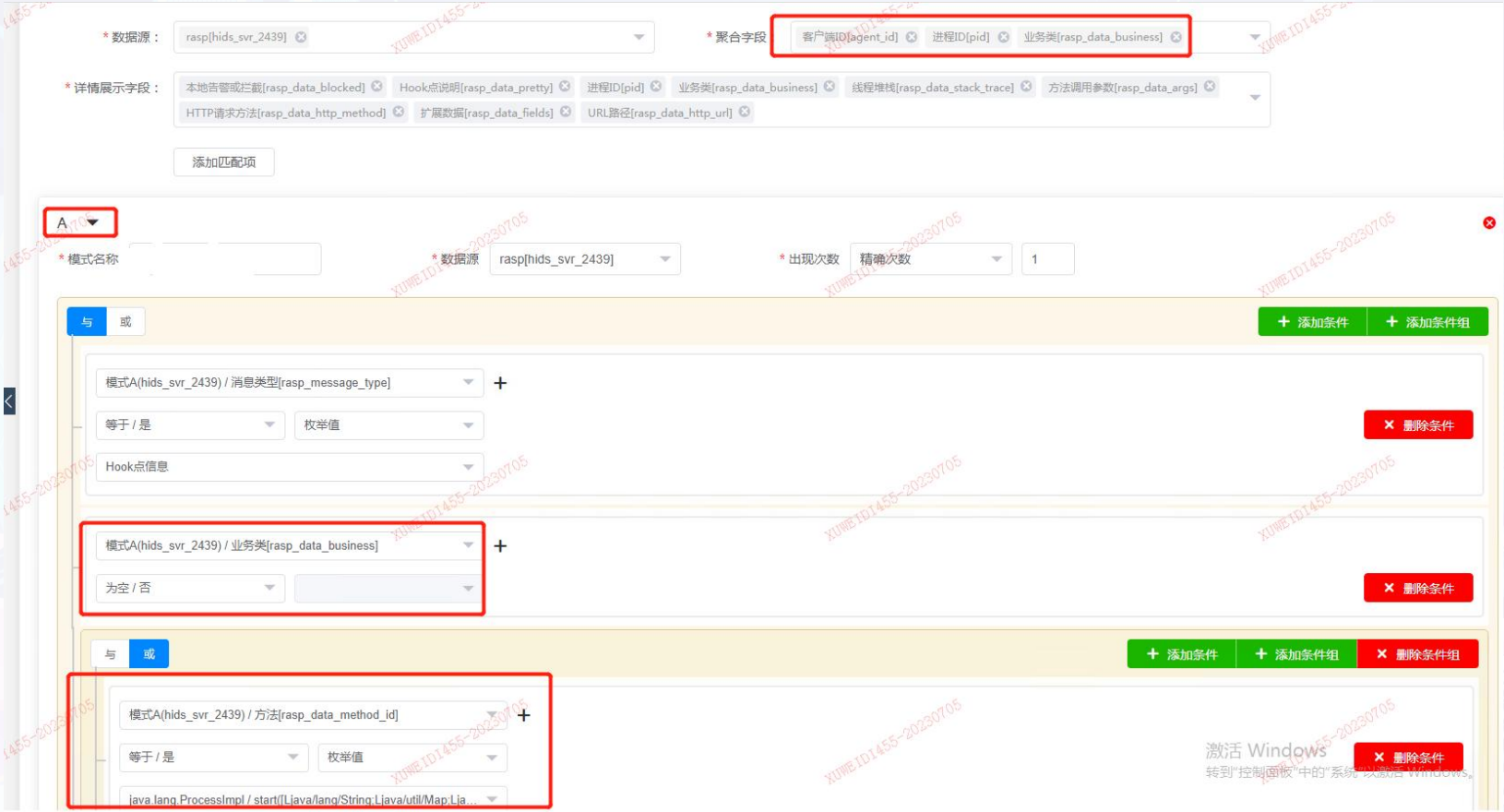


3.检测策略-后门行为

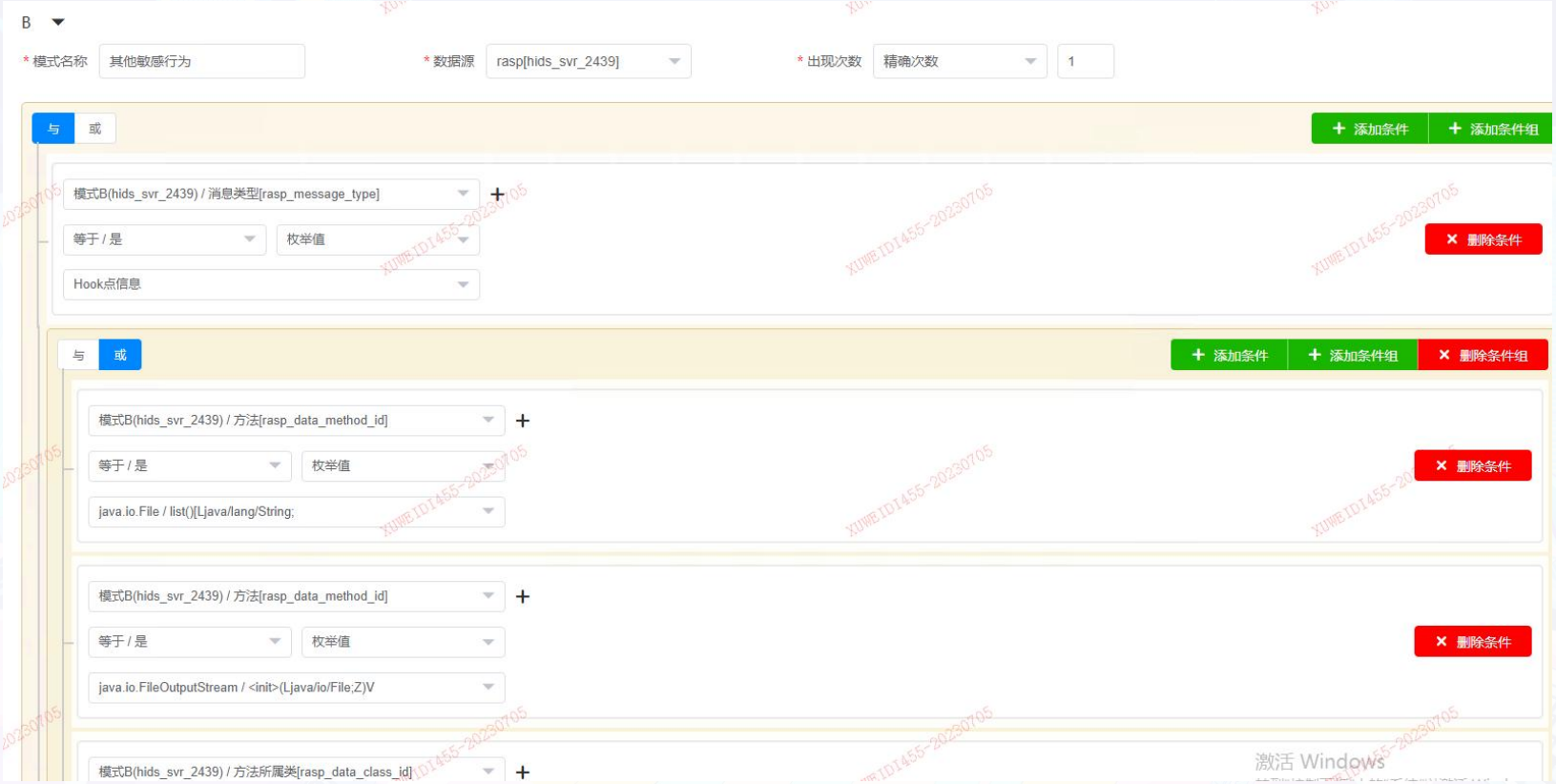

FooClass#fun1()
...
FooClass#funX()
触发?
命令执行
写文件
读文件
类加载
...
插入
business字段
RASP日志


RASP日志
命中A事件
是
等待
命中事件B
是
flink sink

3.检测策略-后门行为


* 数据源：
rasp[hids_svr_2439]
* 聚合字段
客户端ID[agent_id]
进程ID[pid]
业务类[rasp_data_business]
* 详情展示字段：
本地告警或拦截[rasp_data_blocked]
Hook点说明[rasp_data_pretty]
进程ID[pid]
业务类[rasp_data_business]
线程堆栈[rasp_data_stack_trace]
方法调用参数[rasp_data_args]
HTTP请求方法[rasp_data_http_method]
扩展数据[rasp_data_fields]
URL路径[rasp_data_http_url]
添加匹配项
A
* 模式名称
* 数据源
rasp[hids_svr_2439]
* 出现次数
精确次数
1
与
或
添加条件
添加条件组
模式A(hids_svr_2439) / 消息类型[rasp_message_type]
等于 / 是
枚举值
Hook点信息
删除条件
模式A(hids_svr_2439) / 业务类[rasp_data_business]
为空 / 否
删除条件
与
或
添加条件
添加条件组
删除条件组
模式A(hids_svr_2439) / 方法[rasp_data_method_id]
等于 / 是
枚举值
java.lang.ProcessImpl / start([Ljava/lang/String;Ljava/util/Map;Lja...
删除条件

3.检测策略-后门行为


B
* 模式名称
其他敏感行为
* 数据源
rasp[hids_svr_2439]
* 出现次数
精确次数
1
与
或
添加条件
添加条件组
模式B(hids_svr_2439) / 消息类型[rasp_message_type]
等于 / 是
枚举值
Hook点信息
删除条件
与
或
添加条件
添加条件组
删除条件组
模式B(hids_svr_2439) / 方法[rasp_data_method_id]
等于 / 是
枚举值
java.io.File / list()[Ljava/lang/String;
删除条件
模式B(hids_svr_2439) / 方法[rasp_data_method_id]
等于 / 是
枚举值
java.io.FileOutputStream / <init>(Ljava/io/File;Z)V
删除条件
模式B(hids_svr_2439) / 方法所属类[rasp_data_class_id]

3.检测策略-反射调用黑名单

指定黑名单，当关注的方法、字段被调用、设置、获取时上报云端。触发告警可能的原因有：

- 攻击者尝试绕过RASP
- webshell行为
- 表达式注入

```yaml
---
name: java.lang.ClassLoader
includes:
  methods:
    - "*"
excludes:
  methods: <3 items>
---
name: sun.reflect.NativeMethodAccessorImpl
includes:
  methods:
    - invoke0
---
name: sun.misc.Unsafe
includes:
  methods:
    - "*"
---
name: java.lang.ClassLoader$NativeLibrary
includes:
  methods:
    - load
---
name: com.
includes:
  methods:
    - "*"
  fields:
    - "*"
```

3.检测策略

RASP安全能力分为漏洞、事前、事后

云端可从基础钩挂点覆盖掉部分组件漏洞，对于特殊漏洞（SSRF）会有额外的主机行为检测规则。

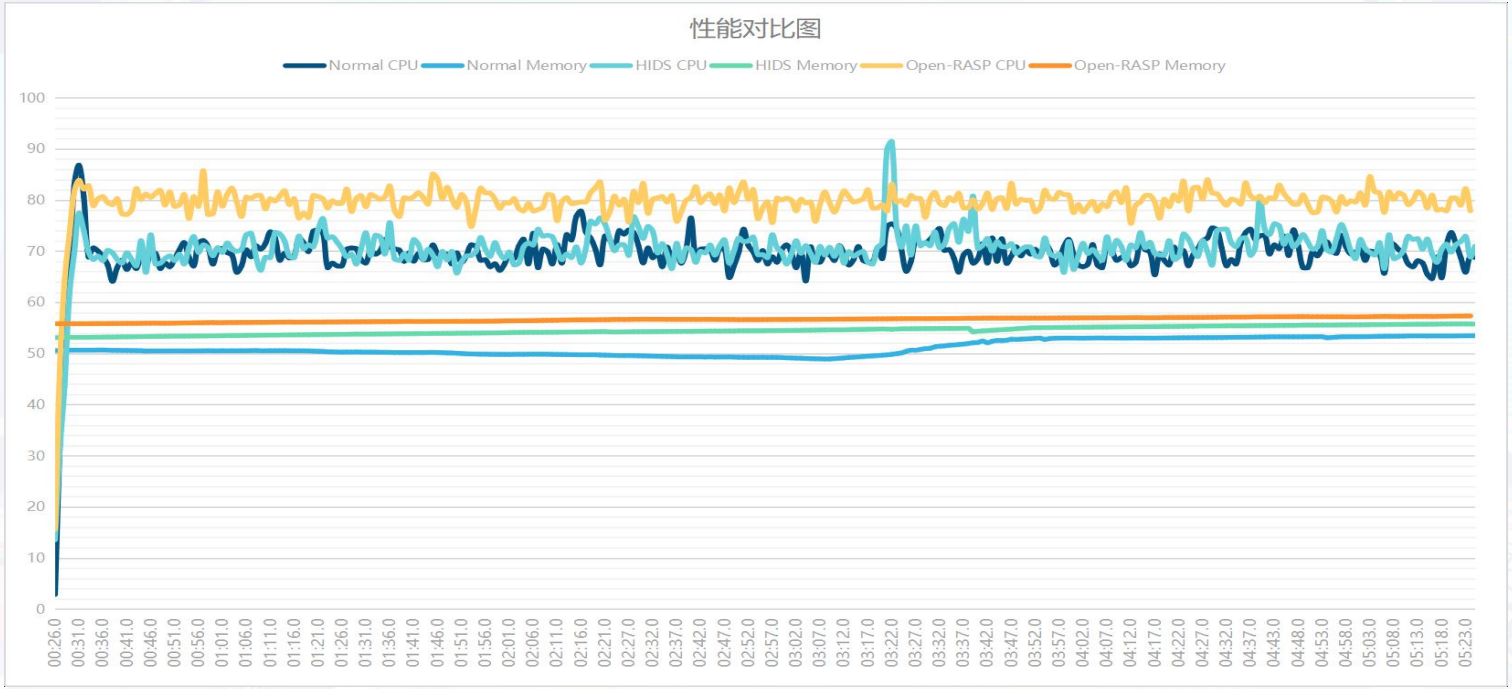
漏洞	注入	SQL注入 命令注入 JNDI	XXE ... ...
	模板注入 /表达式执行	Spring SPEL Freemarker	OGNL Velocity
	敏感数据泄露	任意文件下载 任意文件读取	
	失效的身份认证 和会话管理	越权 后台爆破	
	失效的访问控制	文件上传 SSRF	CSRF
	跨站脚本	XSS	
	不安全的 反序列化	JDK反序列化 Fastjson xstream xmldecoder	Hessian1/2 Yaml ... ...
	漏洞跟踪 /组件漏洞	Spring4Shell Apache CommonsText	Log4Shell ... ...
事前	使用含有漏洞的 组件	三方库版本收集	热补丁
	不安全的配置	基线检查	组件检查
	应用加固	修改HTTP响应头部	
后渗透	不足的日志记录 和监控	web后门 RASP绕过	

四、性能

漏洞靶场下多种类型接口测试出来的情况

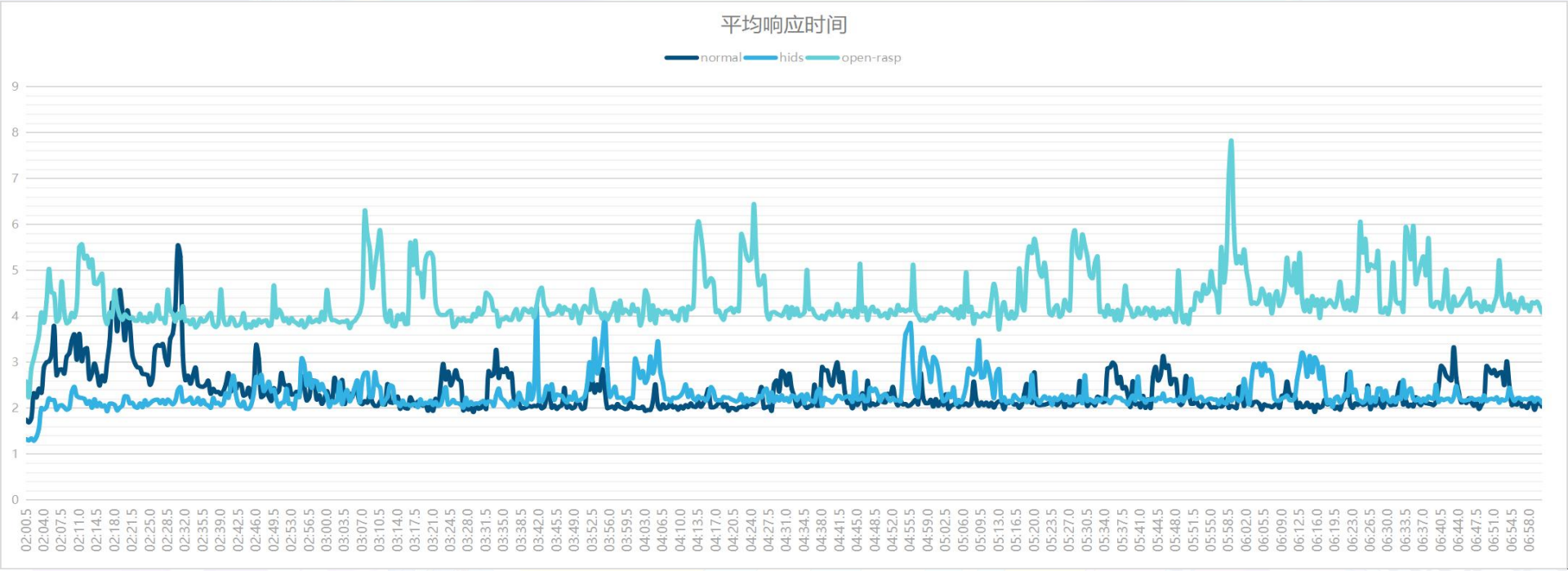
4.性能（漏洞靶场）-请求打满

HTTP请求打满后，此间相关钩挂点仅黑名单生效，不上报数据
在一定周期内有性能抖动（浅蓝色线条）



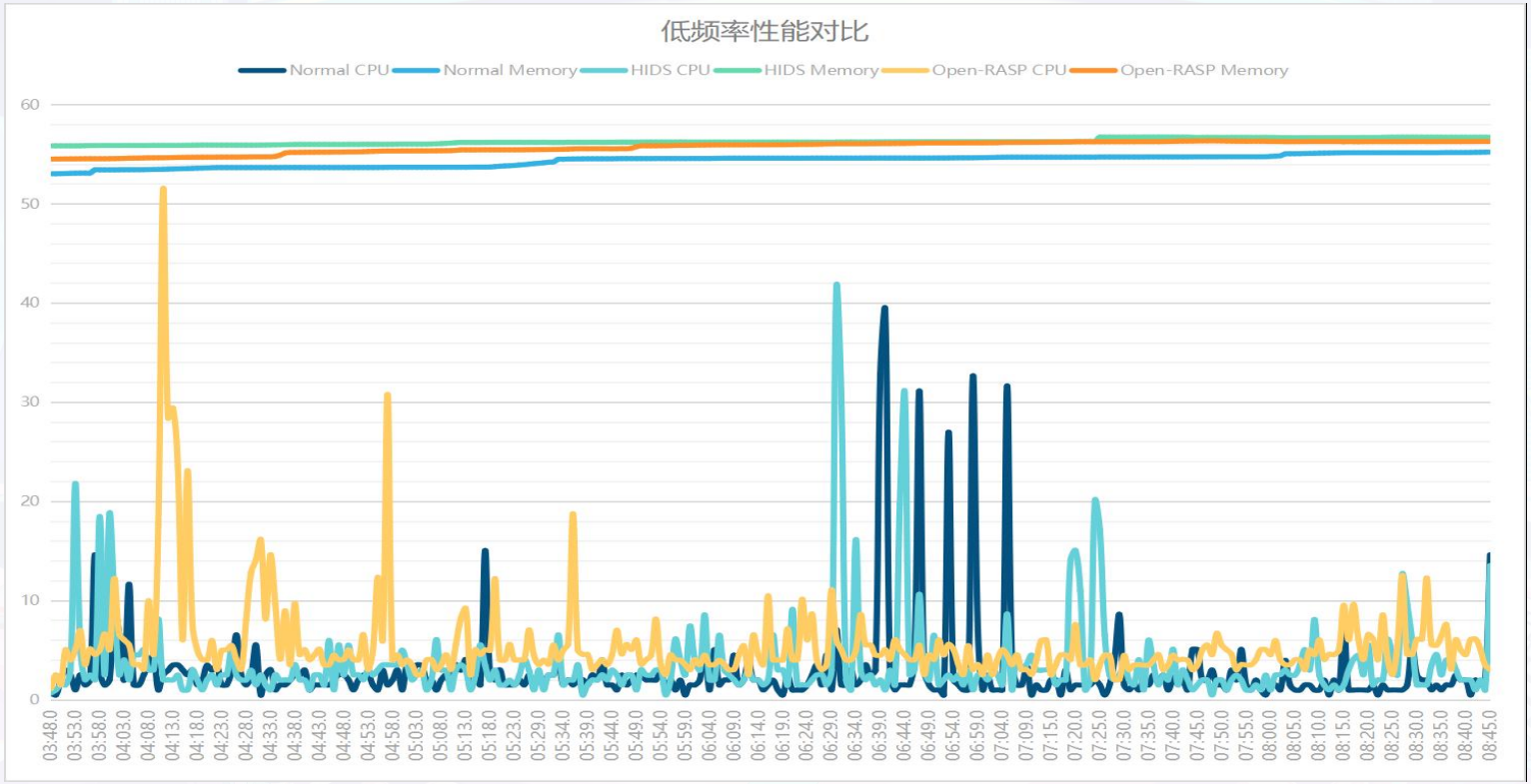
4.性能（漏洞靶场）-请求打满

HTTP延迟基本不变（浅蓝色线），发包上限为原来的99.36%



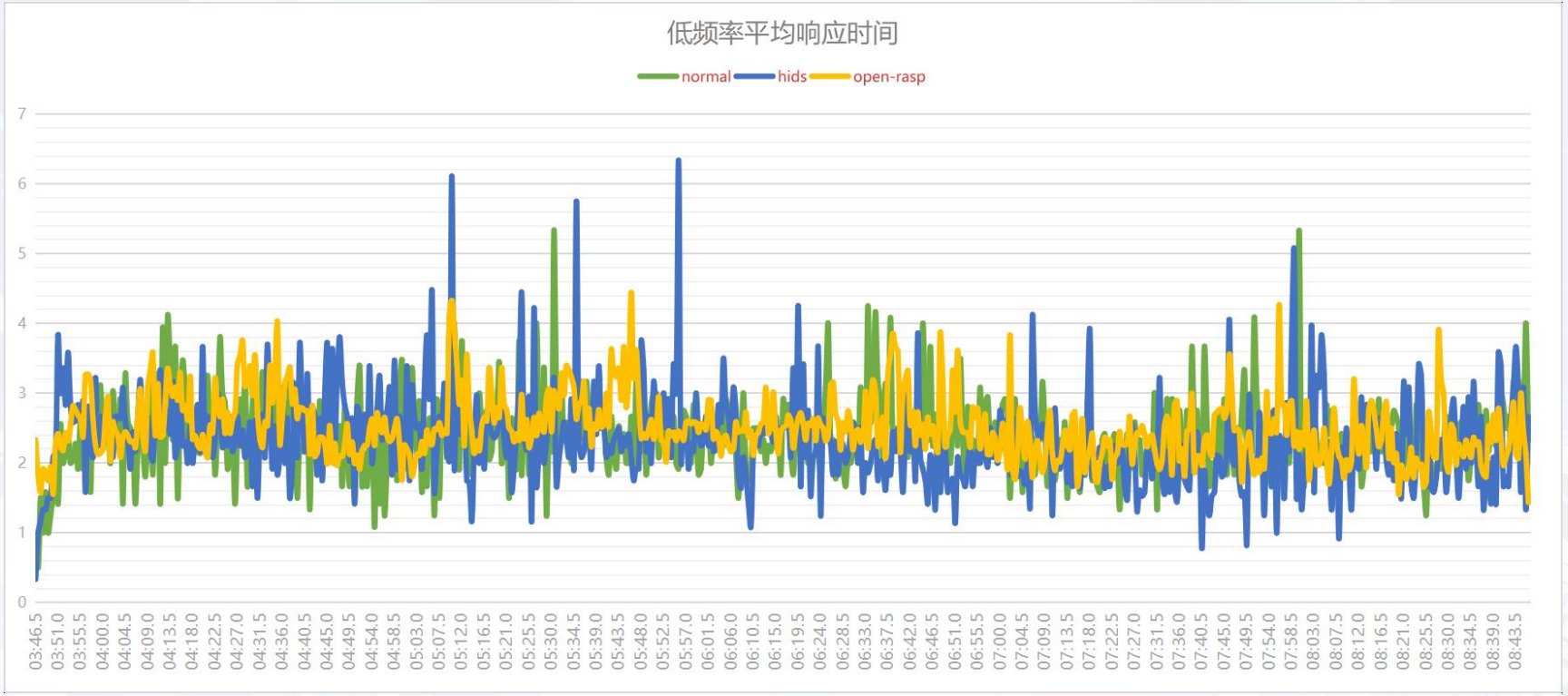
4.性能（漏洞靶场）-低频率请求

流量低的场景作为对比，性能基本没有变化（浅蓝色线）。



4.性能（漏洞靶场）-低频率请求

HTTP延迟（蓝色线）也基本没有变化



五、结语

通过RASP端与云端结合的这种方式，我们让应用防护能力得到了很大的提升；其中的某些安全策略在运行时为了性能考量，我们也不得不降低预期，这也体现了安全是一种成本。

当然，普通RASP维护起来比较简单，而本次演讲所说的RASP本身也不复杂，但一套下来可谓是十分复杂，例如flink sql规则的落地，从demo编写验证到前后端落地，也是耗费许久，整个系统十分依靠整个团队的协作。


THANKS