



落地实战笔记

IAST融入DevSecOps的最佳实践

- 金融科技 | 移动社交 | O2O生活服务 | 传统零售
- 在线旅行服务 | 互联网医疗平台 | SaaS协同平台

目 录

序 言	1
先导篇	2
实践案例	8
同程旅行——需要主动去适应敏捷开发和 DevOps 的安全工具	8
去哪儿旅行——将 IAST 与自身 Q-SDL 体系适配	14
好大夫在线——通过请求管理，解决消息队列堆积的问题	23
好大夫在线——IAST 融入 SDL，包含原理、改造、融合和集成方案	27
上海知名某 SaaS 平台——IAST 部分功能解读&开发洞态 Python-SDK	37
深圳某大型集团公司——利用 IAST 做开源组件的安全治理	42
陌陌——问题驱动的 IAST 落地、改造之旅	48
陌陌——可适配自研 RPC 框架的 IAST	63
58 集团——与洞态 IAST 团队联合开发熔断降级功能避免对业务产生影响	66
自如——一直在寻求能实现全链路、跨服务、跨语言的安全检测工具	75
知乎——初尝试	83
某互联网金融科技公司——使用高自由度的自定义规则能力提高漏洞检出效率	86
聚水潭——如何使得安全工具的误报率为几乎为 0	89
了解 IAST:	95

序 言

随着敏捷开发和 DevOps 在企业软件开发上的应用，软件开发明显提效增速，但也给安全部门带来了较大压力。安全需要具备“简单、快捷、持续”的特性，主动去适应敏捷开发和 DevOps。于是，安全左移的理念——DevSecOps 应运而生。

本册从甲方视角出发，收录了 11 家企业如何将 IAST 落地到 DevSecOps 流程中，共 13 篇文章，涉及在线旅行服务、互联网医疗平台、SaaS 协同平台、传统零售、移动社交、O2O 生活服务、金融科技等多个行业，**大多数文章是用户自己所写**。

记录了甲方企业在安全左移实践过程中的一些思路，以及整体的安全建设方案。

也包含如何更好地使用 IAST 产品，结合自身业务做出优化，更好地适应本公司。

更记录了一个个实战 tips，从甲方视角来看，怎么才能把 IAST 推广下去；踩过的坑，如何克服；安全建设过程遇到的一个个问题及解决方法。

在此，特别感谢同程旅行、去哪儿旅行、好大夫在线、陌陌、58 集团、自如、知乎、聚水潭、深圳某大型集团公司以及某互联网金融科技公司的用户朋友们的支持，在百忙之中总结出实践经验供大家参考。

注：本手册内容具有一定的时效性，由于产品不断更新迭代，最新最全的产品功能请关注洞态 IAST 官网：<https://dongtai.io/>

先导篇

近几年，伴随云计算、容器技术以及 DevOps 的普及，DevSecOps 作为糅合了开发、安全及运营理念的全新方法，其关注热度持续上升，并在全球范围内得到广泛应用。目前 IAST 被部分业内人士看作一种“更适合 DevSecOps 流程构建”的应用程序安全检测技术，受到行业的更多关注。那么 IAST 是否真的更适合 DevSecOps 流程构建？它能够提供哪些核心能力和关键技术，以及有哪些局限性，未来前景如何？对此，安全牛特别邀请到火线安全洞态 IAST 产品负责人，就 IAST 和 DevSecOps 的相关话题展开探讨。

01

安全牛：

在您看来，目前 DevSecOps 的痛点和难点是什么？

洞态 IAST 产品负责人：

要了解 DevSecOps 的痛点和难点，首先要弄明白 DevSecOps 到底是什么。根据 Gartner 定义，DevSecOps（即 Development、Security 和 Operations）是指在不减少敏捷度和开发者效率，或在不要求开发者离开现有工具链的情况下，将安全尽可能无缝、无感知地集成进 IT 和 DevOps 开发中。

DevSecOps 有三个核心点：一是便于集成，安全工具可以很方便的与现有的 IT 或 DevOps 流程对接和打通，这也是实现 DevSecOps 的前提条件；二是无感知，要求安全工具对已有的 DevOps 流程不能产生任何的影响和干扰；三是在研发阶段解决安全问题，而不是像传统开发流程一样，在软件上线后由安全人员检测问题，再反馈给研发人员来解决问题。**问题越早的检测和修复，企业的整体修复成本就越低，这也是 DevSecOps 的核心目的之一。**目前来看，DevSecOps 在落地时遇到的主要痛点和难点也体现在这三个点上。

在 IAST 技术出现之前，我们熟知的安全技术是动态应用程序安全测试技术（DAST）和静态应用程序安全测试技术（SAST），这两种技术在 DevSecOps 流程构建中有其独特优势，但也有各自的不足。

DAST 的优点是检测结果准确，因为它拿的是真实 Payload（有效载荷），在运行的应用程序上直接做漏洞验证。DAST 发现问题后，没有代码层的相关信息，这可能会给研发人员解决问题带来一定的成本，同时其检测时间较长、会产生脏数据等，不能满足 DevSecOps 对无感的要求；SAST 的检测结果对于开发人员来说比较友好，但由于工具无法直接理解代码，尤其是开发人员在写代码时，引用的各种设计模式和新奇的技术，这些原因都会导致 SAST 漏洞检测的误报率存在挑战，给到研发人员时，不能保证报告的准确性，影响使用。

02

安全牛：

IAST 可以帮助 DevSecOps 进行哪些应对呢？

洞态 IAST 产品负责人：

IAST 是交互式应用程序安全测试（Interactive Application Security Testing），是一种新的应用程序安全测试方案，通过在服务端部署 Agent，收集、监控应用程序运行时的函数执行、数据传输等信息，然后根据污点跟踪算法、值传递算法等一系列算法进行漏洞的识别。

IAST 是一种应用程序运行时的漏洞检测技术，所以它具备了 DAST 中检测结果准确的特征；此外，IAST 采集到数据在方法内部的流动后，通过污点跟踪算法来进行漏洞检测，用算法来进行漏洞检测，所以检测结果也具备了 SAST 中全面性的特征。

同时因为 IAST 安装在应用程序内部，安全人员可以拿到类似于源码级漏洞报告，这种漏洞结果对于开发人员很友好，可以方便开发人员进行漏洞修复。综合来看，IAST 具有高检出率、低误报率、检测报告详细便于排查等一系列优势，可以很好地在 DevSecOps 流程中解决痛点和难点。

03

安全牛：

基于 IAST 来构建 DevSecOps 流程，所依靠的关键性技术有哪些？

洞态 IAST 产品负责人：

对于这个问题，我的理解是如何用 IAST 来构建 DevSecOps，或者说是构建 DevSecOps 流程时，IAST 必须具备哪些功能才能支撑这个流程的构建。我个人认为大概有三点。第一点，IAST 必须柔地嵌入 DevOps 流程，即十分便利地与 CI/CD 流程对接，包括与 Jenkins、Gitlab 等工具打通等；第二点，当 IAST 和 DevOps 流程对接时，需要做版本的控制，支持在 Agent 端直接指定项目名称和版本，进行后续的版本跟踪，以及版本的漏洞对比等；第三点，IAST 可通过漏洞复测与回归测试，验证此前发现的漏洞是否依旧存在。

04

安全牛：

相比较其他应用程序安全测试模式，您认为 IAST 的核心能力有哪些？其在具体的场景应用中又会存在哪些局限性？

洞态 IAST 产品负责人：

IAST 本质是做漏洞检测，其核心能力主要包括四点：一是实时的漏洞检测，保证不影响 DevOps 的原有效率；二是第三方组件的梳理和漏洞检测，保证应用避免供应链的攻击；三是灵活的漏洞检测逻辑，让用户在使用内置检测逻辑的同时，很方便地配置出具有业务属性的特定检测逻辑，来做业务层面的漏洞检测；四是极低的运营成本，IAST 在企业内部使用时，一定是需要持续运营的，当出现了 IAST 没有覆盖到的漏洞情况时，可以用最低的成本来完善检测策略和检测逻辑，保证漏洞的检出。

IAST 的局限性主要体现在 IAST 的内置漏洞策略有限、且无业务属性，无法保证检测所有的安全风险；推荐在上线前通过白盒、灰盒、黑盒、人工渗透测试一起来检测漏洞，然后将 IAST 没有覆盖到的漏洞策略补充进来；上线后可通过外部的众测、SRC 运营等手段，更全面地发现安全风险，同时将漏洞策略补充到 IAST 中，做后续的自动化测试。

05

安全牛：

目前国内 IAST 产品的代表类型有哪些？从应用的角度看其主要差异是什

么？

洞态 IAST 产品负责人：

根据 Gatner 定义，IAST 特指被动插桩的这种模式，但由于开发难度等一系列原因，在国内出现了一些临时性的解决方案，如：将黑盒改造成 IAST，另外也有将 RASP 与扫描器结合形成主动插桩的方案。

主动插桩的原理是在应用程序上安装 Agent，Agent 采集应用程序从外部获取数据的入口，以及最终触发漏洞的关键位置信息，然后联动外部扫描器，把流量数据发到扫描器上，扫描器根据漏洞库，或者根据主动式对应的 POC 库，来做一些流量的重组、重放，实现对漏洞的检测。它在检测漏洞的时候，是看外部扫描器端重组的 POC 有没有到达上次出现危险的位置。

主动式有很大的局限性：一，从整个行业来看，应用的安全性越来越高，比如验证码、数据包加密、防重放等一些安全措施越来越完善。在这样的背景下，主动式 IAST 依赖流量重放进行漏洞检测，比如：滑动验证码场景下，IAST 无法重放流量，此时，便无法检测对应位置的漏洞；二，主动式 IAST 需要进行流量重放，会产生大量的脏数据，影响功能测试结果；三，应用程序的技术架构整体趋势是向微服务、分布式等方向发展，在微服务中，服务间可能不用传统的 Http 请求进行通信，比如使用基于 TCP 协议的 RPC 请求。此时，主动式 IAST 无法发起 RPC 请求，也就无法进行漏洞检测。

被动式 IAST 的检测原理，是在应用程序上安装 Agent，安全人员进行正常的功能测试时，外部会有一定的流量进入。在这种模式下，所有进来的流量数据都会被标记为不可信，并分析不可信数据在内部应用程序中如何变化，如何流转，类似于生物学上的基因传递流程。它只需要分析不可信数据在应用程序内部的变化情况，重点分析数据的流向和传播，然后用“算法 + 数据流”进行漏洞检测，根据不可信数据未经任何有效处理直接到达危险函数的方法，来判定漏洞是否存在，无需做流量重放。前文所提到的验证码、数据包加密或者防重放场景，以及分布式、微服务的技术架构下都可以使用被动式 IAST 进行漏洞检测。

从整个行业趋势上来说，应用本身的安全性越来越高，只有被动式 IAST 才能兼容所有的场景，在实现漏洞检测的同时，满足 DevOps 流程下高效、准确等要求，所以最佳选择一定是被动式 IAST。

06

安全牛：

用户在选择 IAST 产品时，应从哪些维度进行评估？

洞态 IAST 产品负责人：

第一点是使用成本，它体现在几部分，其一是产品在 Server 端的部署成本，其二是在 Server 端的升级成本，其三是当把 Server 端部署和升级之后，Agent 在业务线上的推广成本，或者 Agent 在使用过程中的升级成本等。所以整体来看，需要综合考虑：Server 端的部署成本，Server 端的升级成本，Agent 端的部署升级成本以及 Agent 端的推广成本。

第二点是漏洞检测能力，建议直接把 IAST 部署到企业内部真实业务线上试运行两到三个月。根据“是否检测到漏洞，及漏洞检测的准确率”，对比哪款产品检测效果更佳，这是最实际的评估方法。

第三点是 IAST 的整体扩展性，在企业落地 IAST 时，需评估其是否能够便利地与已有业务系统较好结合。可通过查看其 API 接口是否完善、需要的数据是否都能获取。火线安全洞态 IAST 直接开放源代码，方便用户做二次开发，因此可扩展性更强。

第四点是前文提到的运营成本，当出现未检测到的漏洞时，如何将缺失的策略或检测规则加入到产品中，也会产生比较高的后期使用成本，不能忽视。

07

安全牛：

火线安全选择了开源 洞态 IAST 和商业洞态 IAST 产品模式齐头并进，其原因是什么？开源 IAST 产品的表现如何？我们未来的产品规划又是怎样的？

洞态 IAST 产品负责人：

IAST 是个非常不错的工具，可以高效地帮助企业在 DevOps 阶段解决相当多的漏洞。火线安全的理念是帮助整个行业提升安全能力，我们想让更多的企业使用 IAST 来防范安全风险。此外，IAST 本身是一个安全产品，其开发门槛比较高，倘若因为市场上没有开源的 IAST 产品，导致很多企业重复造轮子，就

会影响行业进步，因此我们选择了开源。洞态 IAST 是这一领域的后起之秀，产品进展很快，也得到了很多用户的 support 和认可。我们也会继续努力打磨产品，为用户带来更大的价值。

对 IAST 的未来规划：洞态 IAST 整体架构是利用 Agent 采集数据，在 Server 端进行漏洞检测。在这种架构下，在一定程度上将安全与开发进行分割，安全人员可专注于安全，开发人员可专注于开发。火线安全希望洞态 IAST 真正成为一款链接 Dev、Ops 和 Sec 团队的工具，让安全赋能开发和运维，并结合场景来满足更多 DevSecOps 流程下的安全需求。

安全牛评：

在代码安全与敏捷交付同样重要的时代，只有开发者主动接受安全测试，才能从“根”上解决代码安全问题。在提高开发人员安全意识的同时，将安全测试无感知地融入开发流程等等都是在想尽办法让开发者爱上测试。“以 IAST 为起点构建 DevSecOps 流程”的初衷是用开发思维拉近代码安全测试与代码开发者的距离。

而开源的代码安全工具，进一步推动开发者乐于进行安全测试，有利于应用开发行业代码安全整体水平的提高，也将成为推动代码安全市场良性循环的“加速剂”。火线安全开创了开源代码安全工具的元年——代码安全从代码开源做起。

实践案例

同程旅行

——需要主动去适应敏捷开发和 DevOps 的安全工具

本篇文章于 2021-11-10 发表在火线安全平台公众号

同程旅行是最先部署洞态 IAST 的企业之一。在未部署 IAST 前，同程旅行的漏洞检测修复速度一定程度上拖慢了应用更新迭代的进度，急需一款高效的自动化漏洞检测工具来提升安全能力。经过一系列的调研与考察，我们感叹于洞态 IAST 强大的检测能力和优越的兼容性，最终选定洞态 IAST 作为自动化漏洞检测的主力工具，整个部署调优的过程也得到了洞态团队的全力支持。以下为同程旅行的 IAST 落地实践：

01 安全困境

随着敏捷开发和 DevOps 在同程软件开发上的应用，软件开发明显提效增速，但也给安全部门带来了较大压力。在这一背景下，同程面临着以下问题：

△SAST、DAST、人工渗透测试、人工代码审计无法跟上软件开发的速度与规模

△针对框架结构复杂的接口，一般测试无法完全复现过程中的交互流量在这一困境下，安全如何更好地嵌入应用开发流程？

同程给出的答案是：安全需要具备“简单、快捷、持续”的特性，主动去适应敏捷开发和 DevOps。

02 自动化漏洞检测工具调研

在自动化漏洞检测工具调研过程中，我们首先对 SAST、DAST 和 IAST 进行了对比。综合比较来看，IAST 明显优于 SAST 和 DAST。

AST	SAST	DAST	IAST
误报率	高	低	低
检出率	高	中	高
第三方组件漏洞	静态扫描	依赖PoC、指纹	动态时支持
使用风险	无	产生脏数据	无
漏洞详情	一般	一般	详细
检出速率	根据代码量	根据PoC、URL	根据流量大小
漏洞种类	偏向代码漏洞	可发现运维、配置、运行漏洞	偏向应用本身

洞态 IAST 调研结果：

▷ 领先的技术架构

类型	架构	分析
一般IAST	重Agent端+轻服务端 数据监听和漏洞检测全部在Agent端完成。	1.需频繁升级Agent端； 2.未检测出漏洞的Agent端数据直接丢弃，若产品检测能力升级，需联系功能测试团队重新发起测试； 3.无法实现跨请求关联分析。
洞态IAST	轻Agent端+重服务端 Agent端仅实现数据监听，漏洞检测全部在服务端完成。	1.Agent端代码和逻辑简单，单点故障率更低，极少升级； 2.所有数据保存在服务端，可在服务端直接进行回归测试； 3.服务端可动态加载检测引擎，并可实现跨请求关联分析。

▷ 强大的检测能力

△ API 检测全面覆盖

The screenshot shows a user interface for API navigation. At the top, there are four tabs: '项目概况' (Project Overview), '项目漏洞' (Project Vulnerabilities), '项目组件' (Project Components), and 'API导航' (API Navigation). The 'API导航' tab is selected and highlighted in blue. Below the tabs, there are three dropdown menus: '请选择请求方法' (Select Request Method), '请选择覆盖状态' (Select Coverage Status), and a search bar '请输入API地址进行搜索' (Search API Address). To the right of the search bar, it says '覆盖率 27.59%'. The main content area displays a list of API endpoints with their methods and coverage status. The endpoints listed are:

- GET/POST /WebGoat/service/lessonplan.mvc (Coverage: 0%)
- GET/POST /WebGoat/service/cookie.mvc (Coverage: 0%)
- GET/POST /WebGoat/service/session.mvc (Coverage: 0%)
- GET /WebGoat/service/hint.mvc (Coverage: 100%)
- GET/POST /WebGoat/service/lessonmenu.mvc (Coverage: 0%)

△ 自动化漏洞验证

The screenshot shows a web-based application interface for vulnerability verification. At the top, there are tabs for '应用漏洞' (Application Vulnerabilities), '组件管理' (Component Management), '搜索' (Search), '系统配置' (System Configuration), '组织管理' (Organization Management), and '租户管理' (Tenant Management). A 'Add Agent' button and a user profile icon are also present. Below the header is a search bar with dropdowns for '请选择排序条件' (Select Sort Condition), '请选择开发语言' (Select Development Language), and '请选择漏洞状态' (Select Vulnerability Status), along with a general search input field and a magnifying glass icon. There are two buttons at the bottom of the search area: '批量验证' (Batch Verification) and '全量验证' (Full Verification). The main content area displays a table with one row. The first column contains a checkbox and the URL '/WebGoat/SqliInjection/attack4'. The second column shows the vulnerability details: 'POST/HEADER/PATH' and the code snippet 'org.springframework.web.method.support.HandlerMethodArgumentResolver.resolveArgument'. The third column indicates the severity as '高危' (High Risk) and the time as '1小时前' (1 hour ago). The fourth column lists tags: 'JAVA', 'Sql注入' (SQL Injection), and '验证中' (Verification In Progress, highlighted with a red border). The fifth column shows the source of the vulnerability as 'webgoat'.

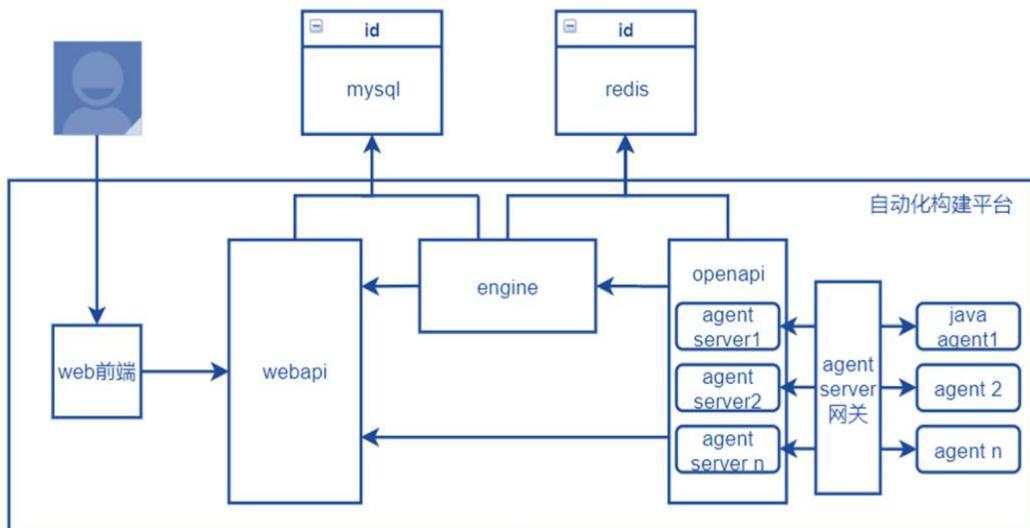
▷ 具有开源项目

- 活跃的开源社区，可持续贡献安全策略
- 企业方便进行二次开发，效率更高，且更贴合自身业务场景
- 部署使用成本低

03 同程 IAST 落地推广

▷ 部署架构

IAST 是基于同程内部的自动化构建平台进行的部署，这种部署不同于 K8s、CI/CD 集成部署。在容器平台上，对 Web、WebAPI、Engine、OpenAPI 四部分进行分开部署。而 OpenAPI 作为 Agent 的 Server，可在流量较大时，启动自适应功能，从而使容器自动扩容。



» Agent 安装

- 自动化部署平台：构建 dockerfile 中添加 Agent 部署的逻辑
- 非自动化部署平台：用户（测试人员）下载 Agent，根据 pid 主动安装

» IAST 测试

- 调研公司内部环境兼容性



» IAST 推广

当 IAST 的部署和测试的流程完毕后，安全部门的动作应是让业务接受并乐于使用 IAST，让 IAST 真正运行起来。

I. 发挥安全的主动性，主动贴合业务流程

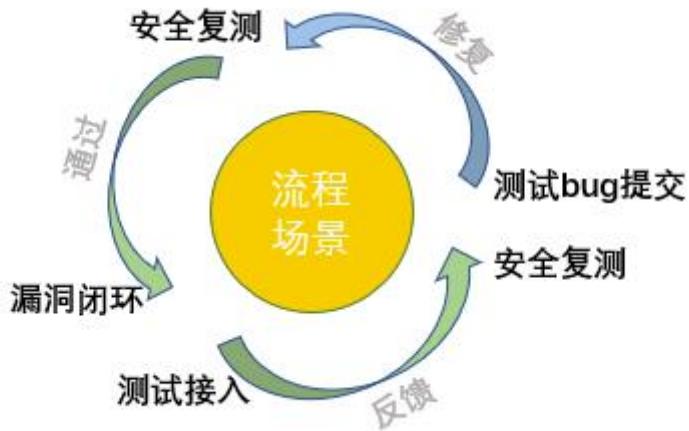
- 培训和文章推广：在公司内部开展周期性的安全培训和安全发文，介绍 IAST

- 根据发现的安全事件，主动推动和提供给业务线安全能力

- 与测试团队合作，推动 SDL 安全能力融入测试流程

II. IAST 场景应用探索——产品上线前的测试流程

- 测试接入 IAST，测试结果反馈安全对接人
- 安全部门复测检出漏洞，报告反馈测试对接人
- 测试提交漏洞至 bug 平台
- bug 修复后，测试反馈安全复测，复测通过，IAST 平台漏洞闭环



测试 bug 的闭环流程，将安全加入测试中，在测试和安全部门之间建立沟通，既利于解决传统测试过程中缺失安全报告的问题，也利于使安全更合理地融入开发流程，减少安全风险。

➤ 对 IAST 的未来规划

	目标	输出
自动化	接入自动化	统一公司内部应用部署平台，agent打入常用镜像，通过CI构建方式直接完全自动化部署
	流程自动化	与测试团队的合作流程，依赖IAST的自动化复测的成熟后和openapi接口的二次开发，做到流程的自动流转，降低流程中的人力成本
精准化	漏洞告警精准化	通过策略运营，让产出结果更贴合业务，输出的结果价值更高

实际使用感受：

1. 部署洞态 IAST 产生的价值：

- 检测漏洞更高效，覆盖的漏洞更全面
- 洞态 IAST 的漏洞详情十分详细，漏洞直接定位到代码行数，并可完整的还原漏洞触发流程，利于开发部门修复
- 误报率相比白盒低很多，复测漏洞所消耗的人力更少
- 高效、误报低的特点提高了安全部门的价值，其他部门对安全的认可度更高，也间接推动了安全部门与其他部门的沟通合作

2. 对原本服务的影响：在大规模推广 IAST，安装 Agent 后，对接口反应时长会有一定影响，但影响不大。

3. 缺点：部分中低危漏洞存在误报现象（洞态解释说明：因为同程之前部署的是 v1.0.3 版本，新版本 1.8.X 对误报现象已有很大改善）

洞态团队点评：

同程旅行的 IAST 实践突出亮点在于其安全理念。同程安全部门强调发挥安全的主动性，主动去适应业务的变化，主动培养同事的安全意识，让整个企业内部达成“安全不是流程的关卡而是齿轮，串联起应用整个生命周期”的安全共识，这对于 IAST 的推广使用能达到事半功倍的效果。此外，同程安全部门还能贴合使用场景，挖掘 IAST 的潜力，对 IAST 进行自动化部署、自动化复测、结果产出更贴合业务的改造，相信洞态 IAST 在同程旅行内能发挥其最大的价值。

去哪儿旅行

——将 IAST 与自身 Q-SDL 体系适配

本篇文章于 2021-12-16 发表在火线安全平台公众号

去哪儿是全球最大的中文在线旅行网站，创立于 2005 年。去哪儿网为消费者提供国内外特价机票、酒店、旅游度假、景点门票产品一站式预订服务，为旅游行业合作伙伴提供在线技术、移动技术解决方案。去哪儿近年来，接连发力大数据与人工智能，利用出游、住宿等领域的全量数据和人工智能，为用户打造智能化搜索、排序、推荐等服务。目前，去哪儿用户累计超 6 亿，平台年交易额超 1600 亿，且仍在快速发展中。

作为洞态 IAST 最早期版本的用户，去哪儿在 IAST 与其 Q-SDL 安全体系的融合适配上有着独到的见解，十分感谢去哪儿 Q-SDL 负责人耿朋敲对本次 IAST 实践的分享。

01 安全背景

新政策、新业务、新威胁

近年来，国家愈发重视信息安全，《网络安全法》、《数据安全法》、《个人信息保护法》相继出台并施行，对互联网企业提出了更严格的安全要求。在业务上，互联网企业的业务边界不断拓宽，用户和年交易额等均实现持续发展，业务保障需求更强。但技术研发成本不断攀升、盈利增速下降都促使着以业务为主的企业，更加注重在安全上的“高产出”。据可查数据，自疫情爆发以来，黑客针对国内互联网企业的攻击几乎呈现指数级增长，企业面临的安全威胁极大。

在这样的背景下，互联网企业的网络安全能力建设，尤其是应用开发期的安全能力建设显得格外重要。阿里、腾讯、字节、去哪儿、轻松筹等企业均在着力强化应用安全开发，试图在应用开发期便大幅降低安全风险。DevSecOps 流程、SDL 流程在各企业都已早早落地实践运行，并打造适配其业务场景的安全体系。

安全需求

去哪儿结合自身实际，打造了 Q-SDL 体系。Q-SDL 体系通过预防、检测和监控措施相结合的方式，减少设计、开发中的软件的漏洞数量和严重性问题，

降低应用安全开发和维护的总成本，保证系统的安全性。



未上线 IAST 前的 Q-SDL 体系架构

从上图可知，用于安全漏洞检测的自动化工具仅包括 SAST 和 DAST，但 SAST 和 DAST 均具有不可避免的严重缺陷。

我们在实际运行中发现，白盒测试存在误报率高、审计时策略规则失效明显、扫描效率低下严重阻碍开发节奏等问题，尤其是无法获取漏洞数据对安全部门造成了很大困扰。而黑盒覆盖范围有限，覆盖率依赖于 Explore 的结果，无法扫描 AJAX、CSRF Token、验证码等页面，无法测试 APP，无法定位漏洞的具体代码，需要花费较长时间与人力来进行漏洞定位与原因分析。

去哪儿急需一款能够弥补黑盒和白盒不足的产品来完善 Q-SDL 体系。

02 调研之路

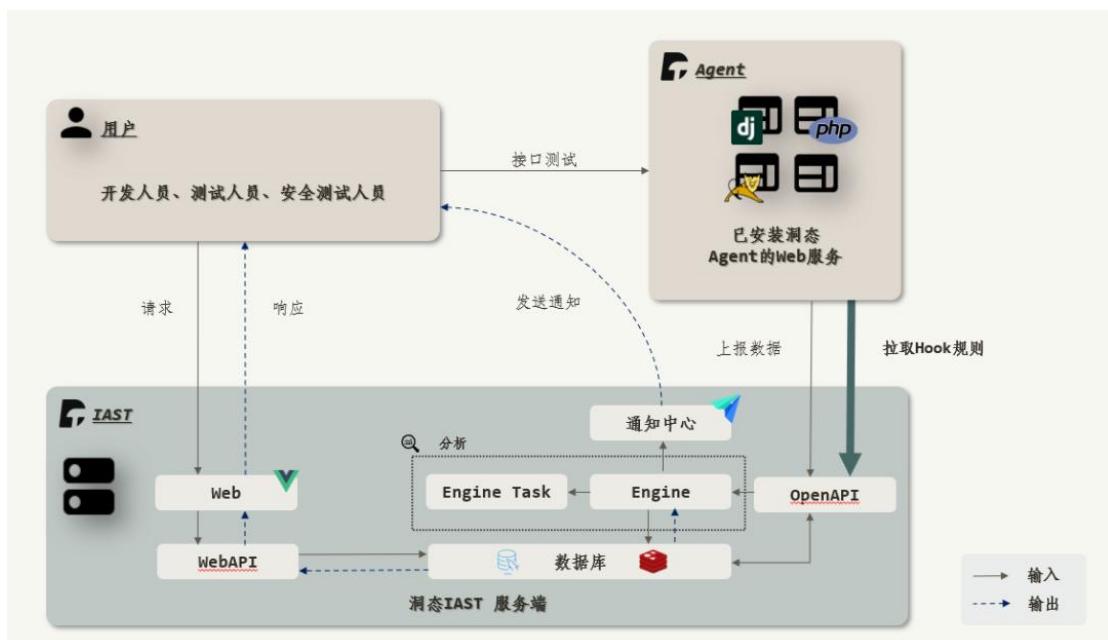
考虑到开源产品具备高扩展性、低使用成本的特性，我们首先将选择定位在开源产品上。由于当时市面上只有开源的 RASP，而没有开源的 IAST，因此首先调研的是 RASP。但调研后发现 RASP 存在严重影响服务器性能的问题，恰巧在了解 RASP 的过程中，发现火线安全正在研发 IAST，并打算开源发布，于是便进行了接触。以下为洞态 IAST 调研结果：

IAST 全称交互式应用程序安全测试，主要通过 Agent 来收集和监控应用程序运行时的函数执行及数据传输，并与服务端进行实时交互，进而更高效、更准确的识别应用软件的安全缺陷及漏洞。同时可准确定位漏洞所在的代码文件、行数、函数及参数，方便开发团队修复问题，还具备高低误报率、0 脏数据

的优势。但 IAST 在不同语言开发的 WEB 应用中需要有不同类型的 Agent，研发的技术难度和投入都非常巨大。

洞态 IAST 产品架构

在调研中我们发现洞态 IAST 在架构上完全不同于其他 IAST，其他 IAST 往往“重 Agent 端、轻服务端”，而洞态的 Agent 端仅用于实现数据监听，漏洞检测全部在服务端完成。这种方法的好处是 Agent 端代码和逻辑简单，单点故障率更低也极少需要升级，降低了维护成本；另外，传统 IAST 产品对于当时未检测的漏洞都在 Agent 端直接丢弃，产品出现新的检测策略后，需要重新发起应用的测试，而洞态 IAST 将检测数据保存在服务端，可轻松地在服务端进行回归测试。



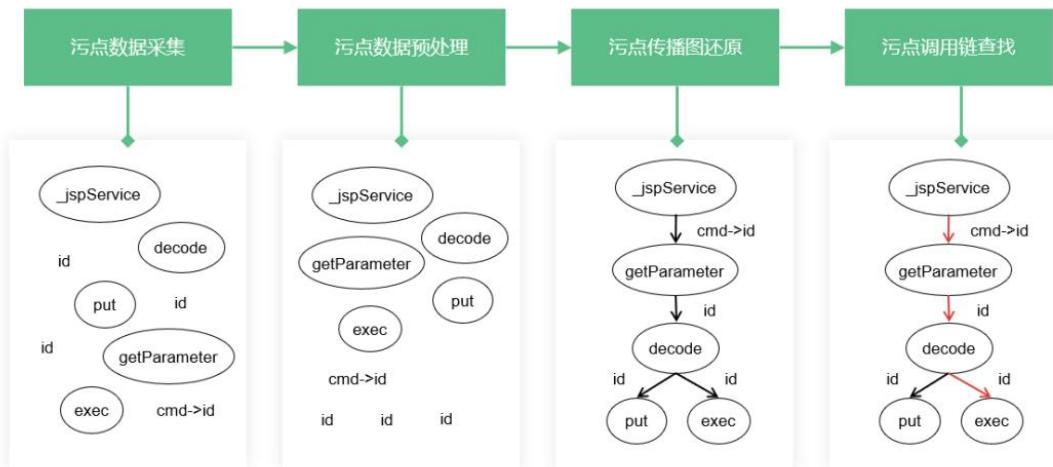
产品架构说明：首先，在服务器上安装 IAST Agent。当 IAST 启动，用户访问 Agent 服务后，Agent 便开始采集数据，并与 OpenAPI 服务通信，进行上报数据和 Hook 规则的拉取。OpenAPI 将数据存储到数据库中，包括 MySQL 和 Redis。然后，Agent 对 Engine 发送通知，Engine 便会来消费数据库中的数据，并在分析完毕后将漏洞信息回写到数据库中。最后，用户通过 WebAPI 查看数据库中漏洞的数据信息。

洞态 IAST 检测原理

洞态基于“值匹配算法”和“污点跟踪算法”对漏洞进行检测。这种算法检测准确率高，还无需采集和重放流量，可以适配如今各种场景下的漏洞检测（如

API 网关、分布式、微服务等架构下的后端服务漏洞检测），还不会产生脏数据，干扰正常的开发测试流程。

洞态污点跟踪算法



对于检测发现的漏洞，洞态根据外部可控数据的传播过程，完整的还原漏洞触发流程，帮助 DevOps 团队快速理解漏洞、定位漏洞，更好的解决漏洞。通过赋能研发人员，提高漏洞修复的效率。

洞态 IAST 产品优势

- 第三方开源组件漏洞分析
- 应用漏洞溯源、定位与分析
- 应用漏洞自动验证
- 全面的风险监测
- 低成本、高扩展、多策略和多场景

促使我们选择洞态 IAST 的原因，除以上产品优点外，还有洞态团队对 IAST 部署升级的全力支持。如果我们选择其他厂商的 RASP/IAST，不一定能得到全力支持。

03 IAST 实践

去哪儿部署洞态 IAST 已经有半年多的时间，特分享一下 IAST 实践经验。

IAST 应用于 Q-SDL 体系

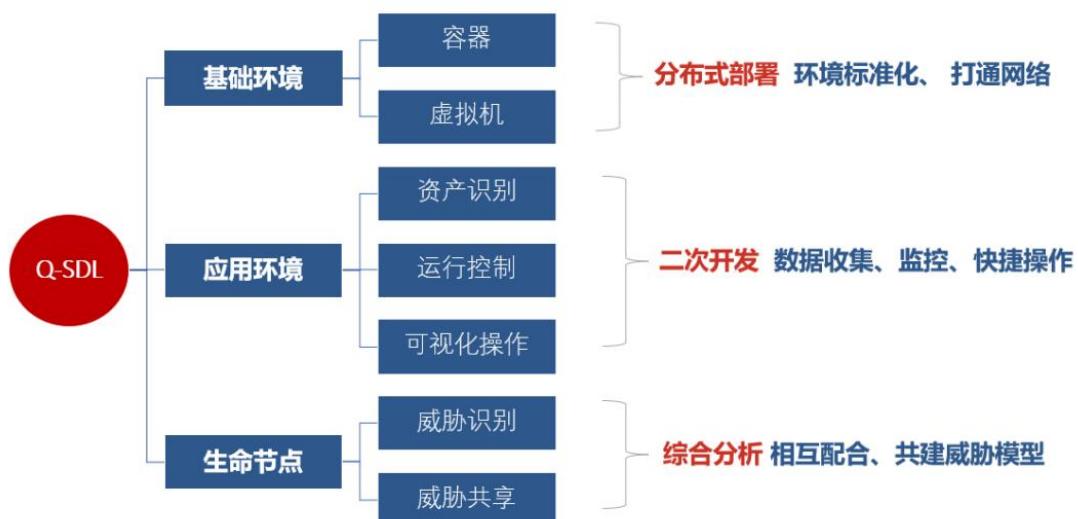
在 Q-SDL 体系中，IAST 承担测试的角色，并且打造了 IAST 平台收集漏洞信息数据。



上线 IAST 后的 Q-SDL 体系架构

部署适配

在 Q-SDL 体系中部署 IAST，主要是从基础环境、应用环境、生命节点上进行适配，并在具体的点上进一步优化。



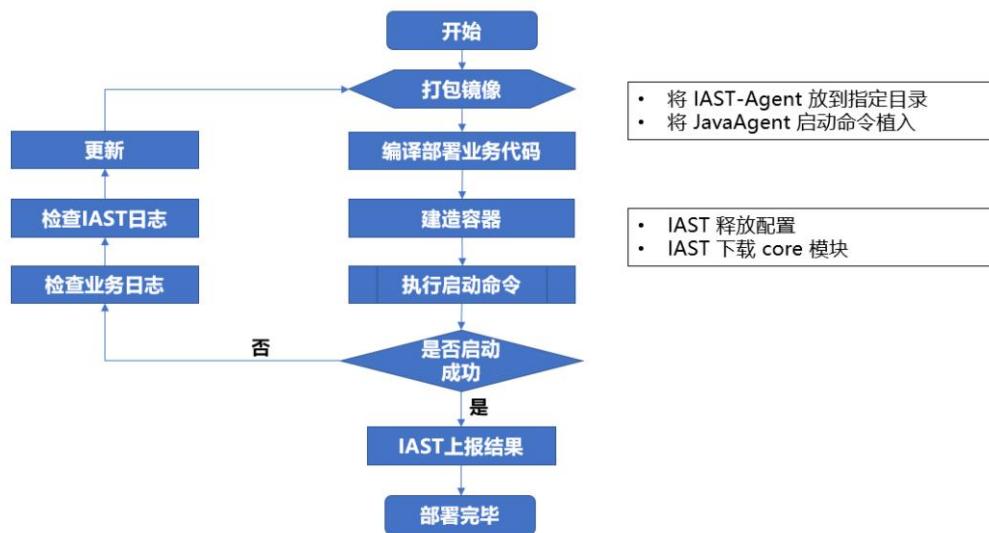
在去哪儿，IAST Agent 的部署环境包括容器与虚拟机二种。采用分布式部署的方式，提高系统的可靠性与可用性，并对二种不同环境开发一套标准流程，保证 Agent 在不同环境的标准化部署。部署完毕后，需要将安装 Agent 的机器所在的网络和 Agent 接收数据的 OpenAPI 之间的网络打通，使 OpenAPI 能接收数据。

在实际应用上，通过二次开发，扩展 IAST 的功能，并推动和去哪儿其他安全工具的融合。通过洞态 IAST Agent 收集识别企业资产，从而对整个资产进

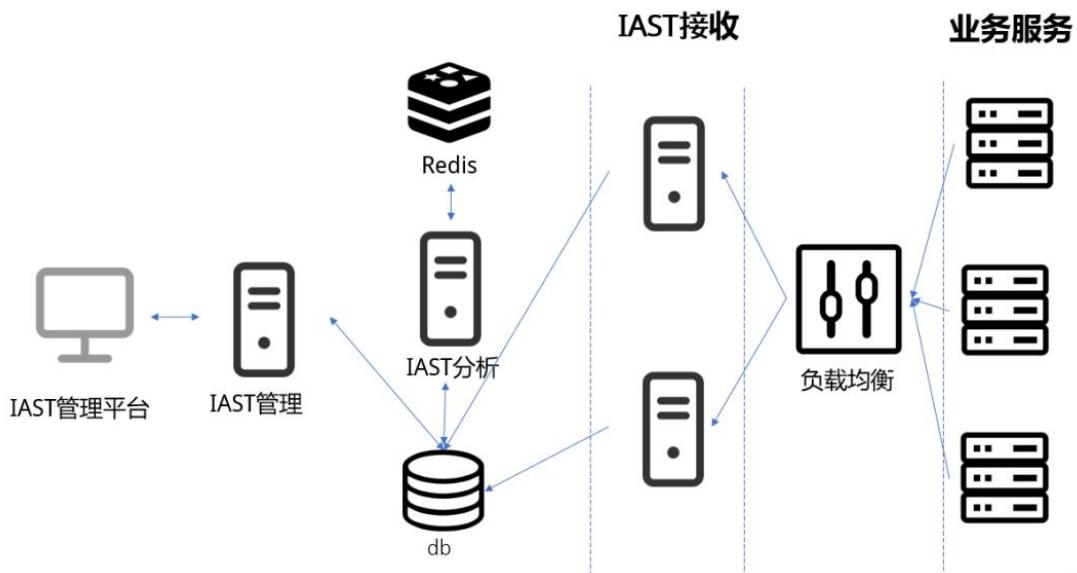
行漏洞检测。为应对 IAST 运行过程中可能出现影响业务等问题，我们提出了二套解决方案。一是“软着陆”，即通过洞态 IAST 自带的解决方案，直接删除核心模块；二是“硬着陆”，基于去哪儿强大的 API 直接将 IAST Agent 端去除。但我们使用这么长时间，还没出现过严重的问题。

在使用中，我们并不是将 IAST 孤立地用来检测。而是把威胁信息和项目信息，上传至去哪儿的威胁建模平台，然后将 **IAST、DAST 和 SAST** 做相应的配合，有针对性地检测某些漏洞，提高检测率。

私有云部署场景及方案



对于私有云部署场景，我们主要做了二点变动：第一点是在 IAST Agent 部署时，把 Agent 包直接放入镜像指定目录，以应对庞大的测试流量。另一点是对 Agent 端启动命令进行植入，以适配标准化环境，从而有序地执行，不去抢占其他环境变量的资源，发生冲突。



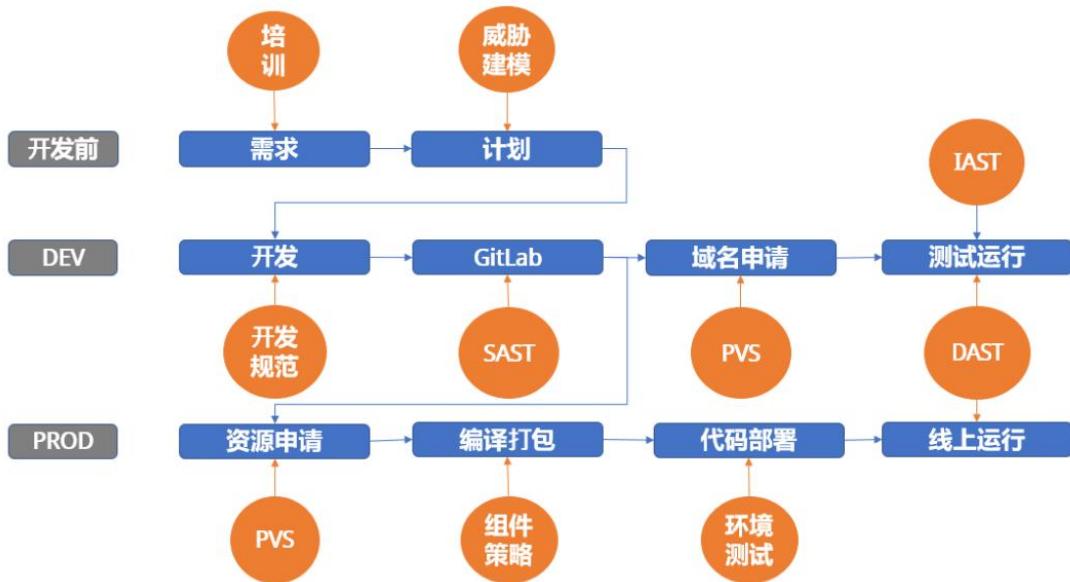
整体的部署方案是遵循洞态 IAST 的基础部署方案，我们只做了微调。业务端 Agent 没有做太大的改变，仅对 Java 包进行了二次开发。为防止应用高流量高并发导致 IAST 超载，针对性地开发了负载均衡功能。而在 IAST 分析模块上，我们将 Redis 独立出来，并做得更强大些以应对高流量高并发。

以下为我们去哪儿网络进出口的真实情况：



为控制突发性的高流量，我们采用了“自适应流量采集+分布式架构”的方案。自适应流量采集将无效的流量过滤，分布式架构则可灵活支撑高并发，减轻流量压力。

运行流程

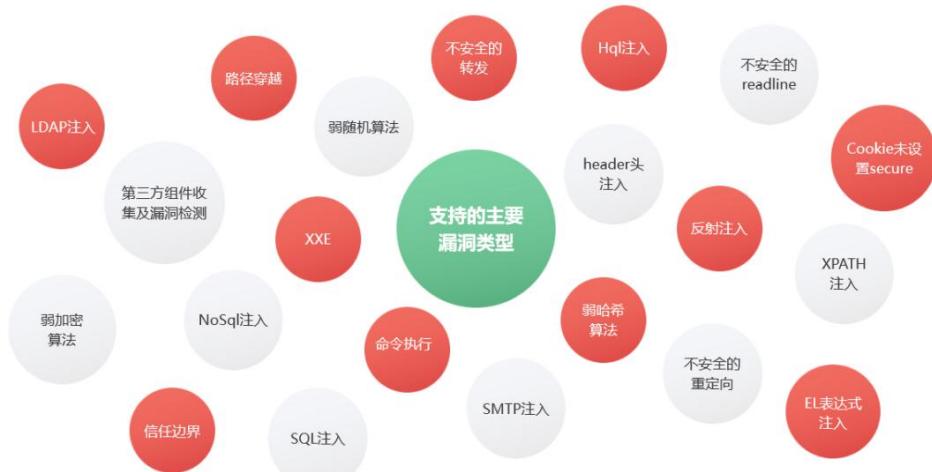


实际使用感受：

1. 值得称赞的点

- 应用威胁识别广泛

我们在上线前利用靶场做过了大量测试，才放心将洞态 IAST 上线。IAST 检测到的漏洞类型几乎覆盖所有的通用漏洞，包括（持续增加中）：



- 组件威胁识别率高

IAST 上线后发现了大量的组件威胁，具体数量等信息就不方便透露咯。

- 技术真材实料，服务尽心尽力

2. 吐槽

- 高危漏洞检出率较低

洞态IAST团队说明：因为去哪儿部署的是最早一版的洞态，出于稳定性考虑，还未进行版本更新，因此在检测能力上相对新版本会差许多。但洞态会加强研发力度，持续扩大可检测漏洞类型以及高危漏洞的检出率。

洞态团队点评：

去哪儿的 IAST 实践亮点在于擅长将 IAST 与自身 Q-SDL 体系的适配，也善于从自身所处互联网行业高流量高并发的特点出发，通过开发负载均衡功能、对 IAST 模块进行调整、扩展开发等手段，将 IAST 的作用发挥到极致，也希望大家都能够从去哪儿的安全智慧中找到灵感。

好大夫在线

——通过请求管理，解决消息队列堆积的问题

本篇文章于 2022-1-26 发表在火线安全平台公众号

好大夫在线创立于 2006 年，是中国领先的互联网医疗平台。好大夫在线通过智慧医疗技术，合理分配医疗资源，帮助患者找到好大夫，是中国最可信赖的院外医疗服务平台。基于对患者的需求分析，好大夫在线创建了国内首个实时更新的互联网医生数据库、首个专业的网上分诊系统、互联网院后疾病管理和线上复诊服务等。截至 2021 年 4 月，好大夫在线平台拥有 23 万实名注册的医生，其中 73% 来自三甲医院，已服务全国患者 7000 余万人，每日医患沟通次数超过 20 万次。

01 背景

某天，我司的洞态 Server 变得比较卡，CPU 一直处于高负荷的运行中。经过仔细排查，我们发现洞态的消息队列堆积了大量的未处理消息。

经推敲，此问题主要由两大原因导致：

1. 性能测试

测试的同学在测试环境中进行性能测试时，导致单个 Agent 发出上百万的数据包。

2. 服务健康检查

业务服务中存在心跳检测机制，单个系统对应多个 Agent，且心跳包发送频率为几秒钟一次。倘若公司内部有 15 个系统，每个系统平均部署 2 个 Agent。那么 Server 每天便需要额外处理心跳包 200 多万次。

分析可得：性能测试时产生的请求包和业务的心跳检测包都是不需要进行漏洞检测的，只有正常的 QA 功能测试才需要进行安全检测。

02 解决方案：请求管理

基于上述分析，我们决定对洞态 Java Agent 进行更新，增加请求管理的 Feature：针对性能测试和健康检查的 API 请求，业务方配置特定的 URL 或

Header 头, Java Agent 对这些请求进行 Bypass, 避免这类请求带来的性能损耗和洞态 Server 端计算的资源消耗。相关 PR:

<https://github.com/HXSecurity/DongTai-agent-java/pull/158>

<https://github.com/HXSecurity/DongTai-agent-java/pull/177>

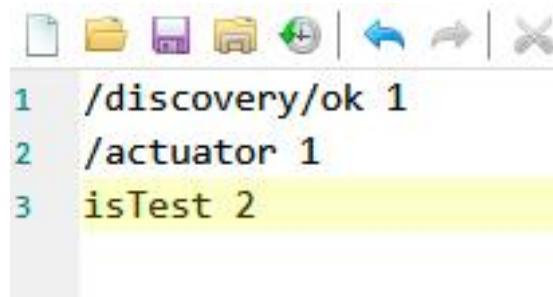
03 如何配置

1. 业务侧

分析心跳的 URL, 配置 URL 黑名单进行 Bypass; 针对性能测试, 由业务方配置特殊的 HTTP 请求头, 用于在 Java Agent 中进行 Bypass。

2. Java Agent

在 iast-core 模块的 resources 目录中增加 blackurl.txt 文件, 在其中配置上对应的规则, 规则格式: <Keyword> <type>, 如图:



- ▷ type 值为 1: 代表 URL 规则, keyword 填写完整的 URI (PATH)
- ▷ type 值为 2: 代表 HTTP 请求头规则, keyword 填写 header 头

04 效果展示

在业务心跳数据包正常发送的情况下, 没有增加 blackurl 时 Agent 端的消息队列, 如图:

方法池队列 (条)	重放队列 (条)	报告队列 (条)	
121	0	74	↑
0	0	0	↑
0	0	0	↑
0	0	0	↑
0	0	0	↑
4073	0	0	↑
0	0	0	↑
3879	0	4096	↑
8	0	91	↑
0	0	0	↑

在业务心跳数据包正常发送的情况下，增加 blackurl 时 Agent 端的消息队列，如图：

请求次数(次)	方法池队列(条)	重放队列(条)	报告队列(条)
1014	0	0	0
9818	0	0	0
0	0	0	0
0	0	0	0
2476	0	0	0
2031	0	0	0
211	0	0	0
16993	0	0	0
12560	0	0	0
108906	0	0	0

通过对比可以发现，解决了业务心跳带来的大量数据堆积问题。

洞态团队点评：

请求管理是一个足以让人眼前一亮的 Feature，既解决了 Agent 对业务需求的影响，也避免了业务需求对 Agent 检测能力的影响。请求管理功能极大地便利了 LAST 在企业中的落地使用，感谢伍雄师傅。

好大夫在线

——IAST 融入 SDL，包含原理、改造、融合和集成方案

本篇文章于 2022-2-15 发表在火线安全平台公众号

01 背景

随着《数据安全法》和《个人信息保护法》在 2021 年的相继出台并施行，整个社会对个人信息保护与数据安全的重视程度达到了前所未有的高度。好大夫在线收录了国内正规医院的 88 多万名医生信息（其中 24 万名医生实名注册），已向全国 7600 万患者提供了线上医疗服务，累积了大量的健康信息、病情描述和病历处方等医疗健康数据（这些数据类型在 GB/T 35273-2020《信息安全技术个人信息安全规范》被举例判定为个人敏感信息）。持续为用户提供安全、稳定的医疗服务，保护医生和患者的隐私，是好大夫在线运行和发展的基础，也是公司自成立以来就最为重视的事情之一。

根据 Verizon（威瑞森）发布的《2021 年数据泄露调查报告》来看，Web 应用攻击仍然是数据泄露的最常见手段。好大夫安全团队在应用安全方面持续做了实践尝试，结合好大夫项目管理流程及研发模式，打造了好大夫的 SDL 架构体系。从管理规范、安全活动、安全工具等几方面来提升安全工程能力，贯穿于项目开发流程中。

好大夫在线SDL架构体系



好大夫在线 SDL 架构体系，经历了安全人员徒手漏洞挖掘、静态应用程序安全测试、利用安全测试用例集将安全融入功能测试、基于轻量级安全评估开展半自动化黑盒测试、基于开源 IAST 打造 CI/CD 黄金管道流水线等各种措施。从适用场景、开发模式、准确度、覆盖度、投入产出等几方面因素综合考虑，使用开源 IAST 解决非逻辑安全漏洞是值得推崇的。本文将重点阐述针对开源 IAST 在应用过程中的容器环境适配、性能提升改造以及在应用系统 CI/CD 流水线的自动化集成部署实践经验，希望对此领域感兴趣的同学能够起到一定的借鉴作用。

02 IAST 探究

IAST：交互式应用程序安全测试（Interactive Application Security Testing），是一种实时动态交互的漏洞检测技术，通过在应用程序服务端部署 Agent 程序，收集、监控 Web 应用程序运行时函数执行、数据传输，并与扫描器端进行实时交互，高效、准确地识别安全缺陷及漏洞。IAST 最显著的特性是它使用插桩方式来收集安全相关信息，持续地从内部监控应用程序运行过程中的代码安全缺陷，在整个开发生命周期中实时地提供报警。

2.1 我们为什么需要 IAST？

好大夫在线产品开发采用小瀑布 + 部分敏捷实践模式，产品迭代速度很快。许多项目按周、天、甚至小时部署上线代码。SAST（静态源代码安全检测）、DAST（动态应用安全测试）、人工渗透测试、人工安全代码审计无法满足应用开发上线的速度和代码规模。同时 SAST 工具检测误报率高，DAST 工具覆盖率差，并且都需要大量人力进行二次验证，无法完美适配 CI/CD 自动化流水线。IAST 的以下几大优势则可以解决这些困局：

- **准确性：**传统黑盒/白盒扫描工具误报率非常高，因此需要投入大量人力去验证漏洞，IAST 的最大优势是降低漏洞误报率，提高项目中自动化漏洞发现的准确性；
- **敏捷性：**现代软件开发模型融合了敏捷模式和 DevOps 理念，迭代速度快。IAST 弥补了传统工具耗时久的不足，让开发和测试人员在执行功能测试的同时，无感知的完成安全测试；

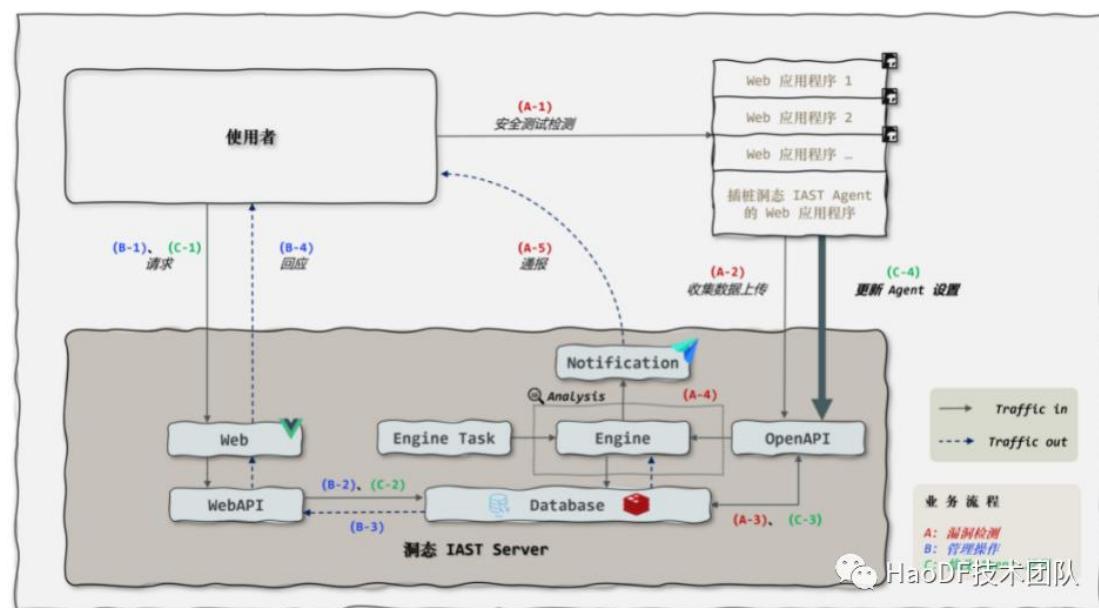
- 流程适合：IAST 多部署在测试环境中，在测试早期阶段，IAST 已经开始工作，在功能测试过程中发现安全问题，及时修复处理，实现安全左移；
- 覆盖度高：在做 API 接口安全测试时，很容易漏掉一些隐藏较深的接口，IAST 在服务端部署 Agent 程序收集测试数据，可以完全覆盖功能测试的所有接口，提高安全测试覆盖面。

经过调研，洞态 IAST 具有准确率高、易安装部署、完全开源等特点，使用应用程序运行时数据流分析，进而识别可被利用的安全漏洞。最终我们选择使用洞态 IAST 作为好大夫的灰盒测试工具。

2.2 洞态 IAST 架构及原理

洞态 IAST 采用轻代理，重服务端的技术架构。具体架构及原理如下：Web 服务器上安装 IAST Agent，用户（开发、测试或安全人员）对 web 服务发起功能性访问后，Agent 便开始采集访问数据，并与 OpenAPI 服务通信上报数据和 Hook 规则的拉取。OpenAPI 将数据存储到数据库中（MySQL 和 Redis），Engine 会在分析后这些数据后，识别漏洞信息并保存下来。安全人员可以通过 WebAPI 来查看漏洞信息。

洞态 IAST Agent 端仅用于实现数据监听，漏洞检测全部在 Server 端完成。这样的好处是 Agent 端的代码逻辑简单，单点故障率更低也极少需要升级，降低了维护成本。另外，将检测数据保存在 Server 端后，当产品出现新的检测策略后可以轻松在 Server 端进行回归测试，进而可以发现新的漏洞。



洞态 IAST 基于“值匹配算法”和“污点跟踪算法”对漏洞进行检测。这种算法检测准确率高，无需采集和重放流量，不会产生脏数据，可以适配各种场景下的漏洞检测（如 API 网关、分布式、微服务等架构下的后端服务漏洞检测）。

03 IAST 在好大夫的演进

3.1 IAST 改造优化

在好大夫容器环境中部署洞态 IAST 并不能直接使用，需要做一系列的改造优化，以解决兼容和性能问题。因为 IAST 的测试依赖测试的流量，如果测试的流量打的不全，那么会导致某些面覆盖不到，从而导致漏报。因为 IAST 采用的是插桩的串行方式，所以可能会出现性能等问题。

3.1.1 RPC 适配改造

好大夫在线容器环境使用了自定义的 RPC 协议，我们发现原生的 IAST Agent 不兼容 RPC 协议。通过排查发现环境中的 RPC 协议有一个自定义的 Host 头，在高版本的 JDK 中，如果不人工进行设置，这些头都不能进行重新赋值。

```
private static final String[] restrictedHeaders = {
    /* Restricted by XMLHttpRequest2 */
    // "Accept-Charset",
    // "Accept-Encoding",
    "Access-Control-Request-Headers",
    "Access-Control-Request-Method",
    "Connection", /* close is allowed */
    "Content-Length",
    // "Cookie",
    // "Cookie2",
    "Content-Transfer-Encoding",
    // "Date",
    // "Expect",
    // "Host",
    "Keep-Alive",
    "Origin",
    // "Referer",
    // "TE",
    "Trailer",
    "Transfer-Encoding",
    "Upgrade",
    // "User-Agent",
    "Via"
};
```

 HaoDF 技术团队

因为这块代码放在了静态块，静态块只会初始化一次。IAST Agent 在最开始会初始化 HttpURLConnection 类进行其他 Agent 包的下载，所以导致了 RPC 调用不通。

```

allowRestrictedHeaders = java.security.AccessController.doPrivileged(
    new sun.security.action.GetBooleanAction(
        theProp: "sun.net.http.allowRestrictedHeaders")).booleanValue();
if (!allowRestrictedHeaders) {
    restrictedHeaderSet = new HashSet<String>(restrictedHeaders.length);
    for (int i=0; i < restrictedHeaders.length; i++) {
        restrictedHeaderSet.add(restrictedHeaders[i].toLowerCase());
    }
} else {
    restrictedHeaderSet = null;
}

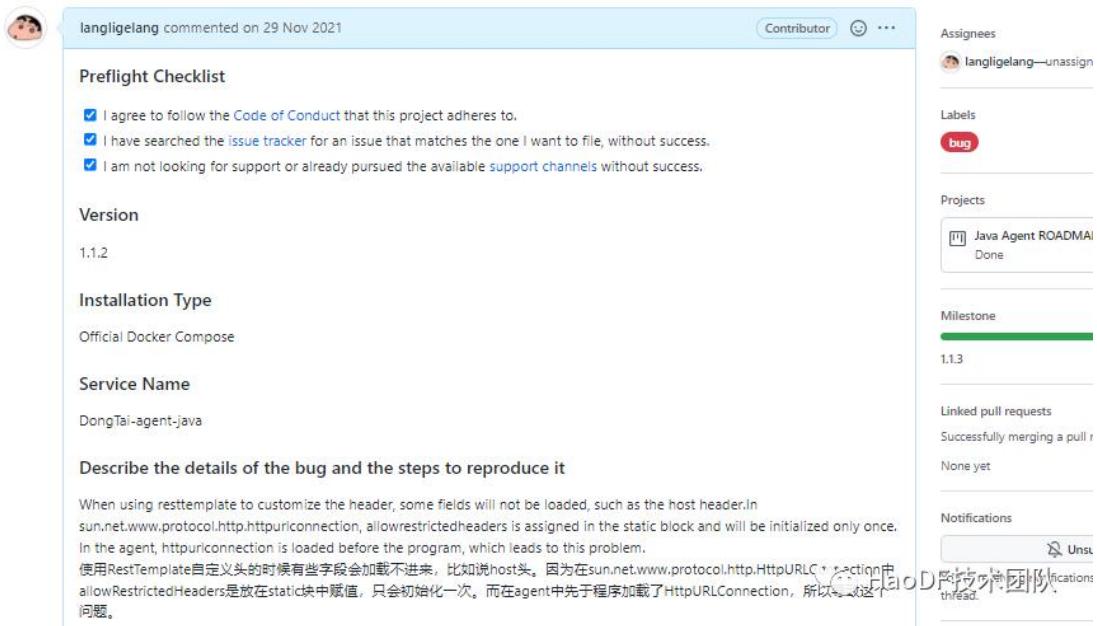
```

 HaoDF 技术团队

随后我们给洞态提交了 issue 后，修复此 bug。

When using resttemplate to customize the header, some fields will not be loaded, such as the host header. #153

 Closed  3 tasks done langligelang opened this issue on 29 Nov 2021 · 0 comments



langligelang commented on 29 Nov 2021

Preflight Checklist

- I agree to follow the [Code of Conduct](#) that this project adheres to.
- I have searched the [issue tracker](#) for an issue that matches the one I want to file, without success.
- I am not looking for support or already pursued the available [support channels](#) without success.

Version

1.1.2

Installation Type

Official Docker Compose

Service Name

DongTai-agent-java

Describe the details of the bug and the steps to reproduce it

When using resttemplate to customize the header, some fields will not be loaded, such as the host header. In sun.net.www.protocol.http.HttpURLConnection, allowrestrictedheaders is assigned in the static block and will be initialized only once. In the agent, HttpURLConnection is loaded before the program, which leads to this problem.
使用RestTemplate自定义头的时候有些字段会加载不进来，比如说host头。因为在sun.net.www.protocol.http.HttpURLConnection中allowRestrictedHeaders是在static块中赋值，只会初始化一次。而在agent中先于程序加载了HttpURLConnection，所以导致这个问题。

Assignees
 langligelang—unassign

Labels
 bug

Projects
 Java Agent ROADMAP
Done

Milestone
1.1.3

Linked pull requests
Successfully merging a pull request
None yet

Notifications
 Unsubscribed

3.1.2 Agent 性能优化

容器环境全量部署 IAST Agent 后，发现整体性能有一定下降，测试同学也反馈应用系统响应慢，无法满足功能和验收测试的要求。经过对 IAST Agent 的分析后，发现其心跳检测每秒钟就会发送一次，如果环境中存在大量的 Agent，IAST Server 就需要处理大量的心跳，导致报告队列经常出现积压，从而使 IAST Server 和应用服务响应速度变慢，整体拖慢了性能。心跳检测接口不存在安全漏洞，可以通过过滤心跳包来减轻 IAST Server 和应用服务的压力。与此同时我们也发现测试同学会经常进行性能测试，发送大量的测试报文，导致 IAST Server 非常慢，这些多余的性能测试包对漏洞分析没有帮助，所以也需要过滤掉性能测试的请求。下图为在没有过滤相关数据前的报告队列经常出现积压，导致 IAST

Server 和应用服务变慢。

请求次数(次)	方法池队列(条)	重放队列(条)	报告队列(条)
95034	4370	0	3857
1004	0	0	0
23434	727	0	265
45654	2663	0	22HaoDF技术团队

所以基于以上两点原因我们对 IAST Agent 更新了一个 feature。

Consider adding URL whitelist mechanism to agent(考虑对agent加入url白名单机制) #133

[Open](#) 2 tasks done langligelang opened this issue on 12 Nov 2021 · 0 comments

只需要维护更新 blackurl.txt，就可以做到对于心跳包和性能测试的数据包不检测。对于性能测试我们自定义了一个 http header 用于标识，只要带着这个 header，IAST Agent 就不会采集数据进行上报。

- 前面的 url 代表心跳的包，也就是 url 中存在此 url 不检测，标志位为 1；
- isTest 表示出现此头，我们就不检测，标志位为 2；

```
/discovery/ok 1
/actuator 1
isTest 2
```

HaoDF技术团队

洞态对 Java Agent 进行更新，增加请求管理的功能，针对性能测试和健康检查的 API 请求，业务方配置特定的 URL 或 Header 头，Java Agent 对这些请求进行 Bypass，避免这类请求带来的性能损耗和 IAST Server 端计算的资源消耗。优化后我们的 IAST Server 队列几乎没有出现积压的消息，IAST Server 端和

应用服务的速度也得到了显著的提高。

请求次数(次)	方法池队列(条)	重放队列(条)	报告队列(条)
51192	0	0	0
17539	0	0	0
47977	0	0	HaoDF技术团队

3.2 IAST 与 SDL 融合

为了进一步提升部署效率，实现自动化集成部署，我们将 IAST 与现有应用系统 CI/CD 流水线进行融合，同时考虑到 Agent 发生异常时，需要及时将 Agent 关停下线，避免对应用产生影响，我们制定了降级方案，在服务发布前设置了一个“开关”，可以及时进行降级。最终打造出应用与安全相结合的 CI/CD 黄金管道流水线。

3.2.1 自动化部署

为了提高部署效率，以及后续推广的便捷，我们将 IAST Agent 打包通过 gitlab 提交代码后生成如下展示版本号，以便 Jenkins 构建镜像时可直接调用。



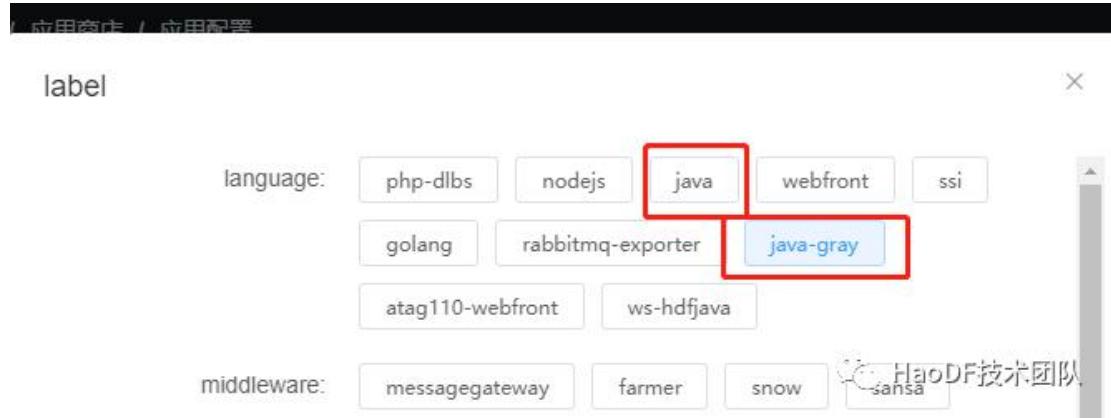
在测试容器环境进行服务发布时选择 java_gray 标签，Jenkins 在构建镜像时会自动带上 gitlab 上已经打包好的 Agent，实现自动化部署。



3.2.2 降级策略

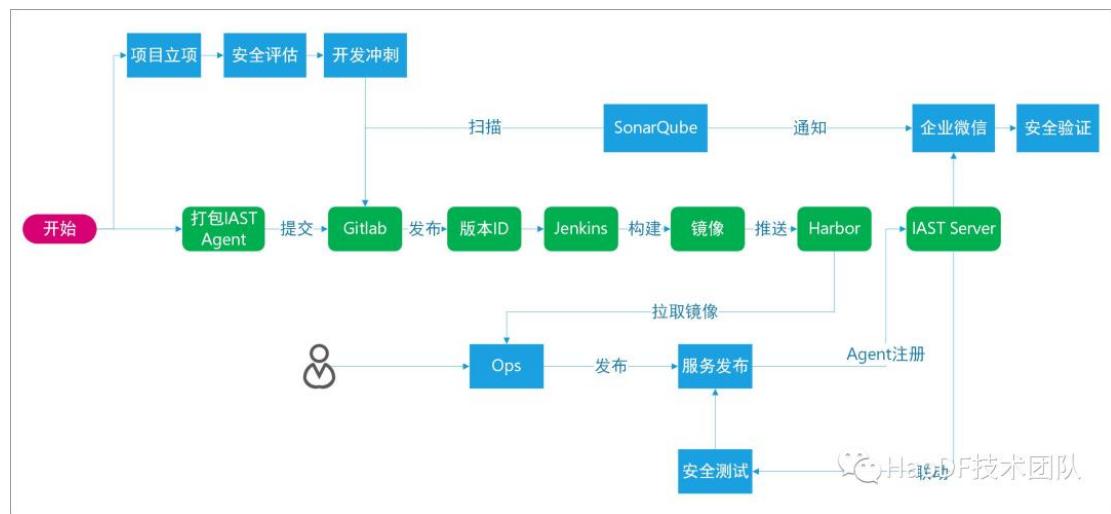
考虑到 IAST 可能出现的异常情况，基于应用环境稳定性考虑，在容器管理平台服务发布前设置了一个“开关”，如果 Agent 出现故障，应急处理人员可将

标签改为 java，重新发版，Jenkins 在构建镜像时将不再打包 IAST Agent，及时避免影响，实现彻底降级处理。



3.2.3 集成方案

下图为好大夫目前集成开源 IAST 的流程图。在项目立项以后进入需求评审，安全人员参与评审后决定是否需要评估此项目，然后开发人员进行编码，编码后提交 gitlab 会进行白盒扫描，结果反馈给安全人员，安全人员跟进检测结果。当测试人员通过 ops 系统进行测试环境的应用发版时候，IAST Agent 会与开发人员提交的应用代码合并打包镜像，应用发版后 IAST Agent 会自动进行注册，通过采集分析功能测试数据自动化测试安全漏洞，若发现漏洞会将结果发送给安全人员确认。至此 IAST 就融入了现有的 CI/CD 流程中。



3.2.4 运行效果

洞态 IAST 在好大夫测试环境部署完成后，经过一段时间的试点推广，已接入大部分 JAVA 应用系统，在此过程中没有发生因 IAST Agent 导致测试环境性

能瓶颈和其他应用系统故障。洞态 IAST 上线后，发现了一部分高质量安全漏洞，通过后台提供的污点流图中的污点来源和传播方法，结合源代码来确认定位安全漏洞。



04 IAST 技术展望

在 SDL 中加入开源 IAST 覆盖后端开发语言，提早融入测试环节，开展功能测试的同时完成安全扫描，极大提升了效率，洞态 IAST 在好大夫 SDL 架构体系中扮演着重要的角色。开源 IAST 技术已可以在企业应用安全建设中发挥作用，但还有很多功能需要完善，下面是对 IAST 的几点的技术展望。

4.1 与黑盒工具结合

IAST 与黑盒工具结合，进行联动测试，可以弥补各个 AST 的缺点，能够更加准确全面地发现漏洞。比如一些越权检测，对于一些中台的系统来说，本身的设计就无鉴权，鉴权在前端的 controller 层做，这时候使用 IAST 来做的话会明显出现弊端，所以对于 IAST 来说做到链路的跟踪，然后来联合黑盒一起做，能够取得不错的效果。

4.2 性能优化

对于一些庞大的应用系统，由于 IAST 采用的是插桩方式，会侵入代码内部，并且需要与 IAST Server 不断地通讯，会造成对 Server 端和 Agent 端的压力变大，使应用程序端变慢，所以这部分需要更加细致的优化。

4.3 隐私检测能力

IAST 应丰富敏感信息检测的规则模型，能够在应用开发过程中发现应用对个人敏感信息的传输、处理、存储方式并进行监测跟踪，辅助分析敏感信息暴露

面及信息泄露风险点。

4.4 减少误报

由于应用系统开发人员会针对特定的漏洞写一些安全过滤功能，这一点 IAST 还不能准确识别。安全人员针对这种情况需要结合内部使用的框架、程序员的编码习惯来开展定制化运营，过滤掉一些常见的规则减少误报。

上海某知名 SaaS 平台

——IAST 部分功能解读&开发洞态 Python-SDK

本篇文章于 2021-12-27 发表在火线安全平台公众号

01 引言

本文源自于 *FreeBuf* 作者 *SpenserCai*, 很感谢该作者对洞态 IAST 的关注与支持。

作为公司信息安全部的成员, 确保每一条业务线的应用安全, 是我工作的一部份, 那么如何完成这项使命呢? 我会在接下来的篇幅中一一说明。

02 应用安全测试

目前常见的应用安全测试有三种: SAST(静态应用程序安全测试), DAST(动态应用程序安全测试)、IAST(交互式应用程序安全测试)。

SAST

我们已经自研了一套应用于主要业务线的 SAST 平台, 在使用的过程中, 有不错的漏洞检出率, 但受限于编程语言, 以及其存在一定的误报率, 需要消耗较大的运营成本去审核检出的漏洞。

DAST

黑盒安全测试同样也应用于我们的日常安全运营中, 我们对一些扫描器实现了自动化平台化, 但扫出的漏洞依旧有限。

IAST

IAST 相当于 DAST 和 SAST 的组合, 是一种相互关联的运行时安全检测技术。它通过使用部署在 Web 应用程序上的 Agent 来监控运行时发送的流量并分析流量, 以实时识别安全漏洞。IAST 提供更高的测试准确性, 并详细地标注漏洞在应用程序代码中的确切位置, 从而帮助开发人员达到实时修复。

03 洞态 IAST

了解到洞态 IAST 还是从我的同事 @null-302 那儿, 当时给我的第一感觉

是：这么牛 X 的项目居然完全开源了。以下对现有功能进行大致说明，不足的地方还望大家指正。

通用漏洞扫描

洞态目前通过 Agent + Server 的方式，支持对 Java、Python、GoLang，PHP 等语言开发的应用进行交互式漏洞扫描，大致的原理是通过 Hook 技术结合污点传播，对程序中所有方法的参数、数据流转进行分析从而识别出漏洞。这种方式对漏洞的检出是不会存在误报的，由此就可以省去漏洞运营的成本。

举一个简单的例子 (python3 + Flask) :

```
1 @app.route("/test",methods=["POST"])
2 def Test():
3     data = json.loads(request.get_data())
4     argv1 = data["argv1"]
5     exec(argv1)
```

对于上述代码可以很明显的看到这是一个存在“代码执行漏洞”的接口，洞态的检测原理则是通过对 Flask 中的 request 下所有的方法以及 exec、eval 等可以执行命令的方法进行监控，通过污点传播的方式可以检测出当参数从 request 请求传入，流入命令执行函数时判定存在代码执行漏洞。

组件漏洞识别

对于组件的漏洞识别这一点也是洞态的亮点之一，可以很好的实现对项目组件安全性的监控，该模块本身具有组件漏洞检测的能力，同时对于一些 1day 的组件漏洞，可以结合情报进行识别、告警。

美中不足的是目前只支持 Java 应用。

组件名称	组件版本	路径	所属应用	语言	风险等级	漏洞数量
BOOT-INF/lib/bms-chargeManage-...	SNAPSHOT	[REDACTED]	[REDACTED]	JAVA	无风险	0
BOOT-INF/lib/joda-time-2.10.10.jar	2.10.10	[REDACTED]/j...	[REDACTED]	JAVA	无风险	0
BOOT-INF/lib/hutool-all-5.7.9.jar	5.7.9	[REDACTED]/j...	[REDACTED]	JAVA	无风险	0
BOOT-INF/lib/commons-codec-1.15.jar	1.4.2.Final	[REDACTED]/j...	[REDACTED]	JAVA	无风险	0
BOOT-INF/lib/commons-codec-1.15...	1.15	[REDACTED]/j...	[REDACTED]	JAVA	无风险	0
[REDACTED]	[REDACTED]	[REDACTED]/j...	[REDACTED]	JAVA	无风险	0
[REDACTED]	[REDACTED]	[REDACTED]/j...	[REDACTED]	JAVA	无风险	0
BOOT-INF/lib/fastjson-1.2.71.jar	1.2.71	[REDACTED]	[REDACTED]	JAVA	无风险	0
BOOT-INF/lib/bms-infrastructure-1....	SNAPSHOT	[REDACTED]	[REDACTED]	JAVA	无风险	0
BOOT-INF/lib/global-metrics-1.0.2....	SNAPSHOT	[REDACTED]	[REDACTED]	JAVA	无风险	0

敏感信息泄露

在最近的版本更新中添加了敏感信息泄露的检测功能，通过对应用的 Request、Response 进行正则匹配，实现了敏感信息进出站的检测。同时支持用户自定义正则检测敏感性息，在数据安全法落地的今天，这显得非常重要的。后面会单独写一篇文章来分享该模块的使用心得体会。

PS: 默认正则有待优化。

全流量及调用链

The screenshot shows the DongTai application monitoring interface. At the top, there are navigation tabs: 项目配置, 应用漏洞, 组件管理, 搜索 (highlighted in blue), 系统配置, and 组织管理. There are also buttons for + Add Agent, a bell icon, and a flame icon.

Below the tabs, there is a search bar with filters: 筛选 (dropdown), 包含 (dropdown), POST (selected), and a date range: 2021-12-19 00:49:30 ~ 2021-12-26 00:49:30. A search button is also present.

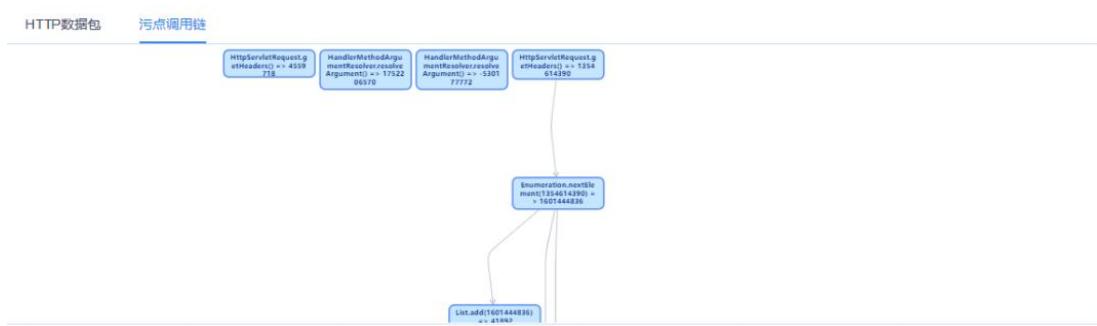
The main area displays a network traffic log entry. The log details are as follows:

- URL: http://[REDACTED] (timestamp: 2021-12-24 17:03:43)
- Agent: [REDACTED] (User-Agent: [REDACTED])
- User: admin
- Project: [REDACTED]
- Associated Vulnerability: 手机号码泄漏
- HTTP Request (POST):

 - HTTP/1.1
 - POST / [REDACTED] HTTP/1.1
 - content-length:36
 - host:jst-bms-charge-manage
 - content-type:application/json
 - connection:Keep-Alive
 - dt-traceid:[REDACTED]
 - accept:/*
 - user-agent:Apache-HttpClient/4.5.13 (Java/1.8.0_262)
 - Request body: [REDACTED]

- HTTP Response:

 - HTTP/1.1 200
 - DongTai:v1.1.3
 - Response body: [REDACTED]



官方定义这个功能叫搜索，我更愿意称其为全流量及调用链，如图所示，这个模块可以获取到应用完整流量，如果有漏洞则会标记。同时也能获取程序内部的完整调用关系，如果存在漏洞则会标记是哪一步存在漏洞。由于篇幅有限在这篇文章中就不详细描述使用场景了。

04 加入开源社区

在使用的过程中，我们也遇到了一些问题和 bug，洞态 IAST 团队在很快的时间内进行了修复，并且催促我们尽快测试。要知道在开源项目中能有这么快的 bug 修复速度，并且反向催促使用者尽快测试，是我闻所未闻的。

同时，需要将一个产品落地到实际场景中，至少在我看来，集成是必不可少的。洞态团队对于项目精神，加上，我们现有的需求，在这两个前提条件下，让我有了加入开源社区的想法。

洞态本身有着良好的 API 接口，可是需要完成集成，如果仅通过 API 就会出现如下不怎么好维护也不怎么优雅的代码：

```

1 data = {
2     "project_id":101,
3     ....
4 }
5 repData=
6 requests.post("http://xx.xxx.xx.xxxx/api/v1/project/version/add",data=data,
7 repData = json.load(repData.text)
8 if repData["status"] == 201:
9     version_id = repData["data"]["version_id"]
10    ....

```

每一次请求接口的时候都要对状态码做各种判断，都要通过 json 的 key

的方式（`repData["xx"]["yy"]`）去获取数据，同时每次都要去看文档才能知道要获取的数据是什么如何请求等等。

于是我便有了开发 SDK 的想法，洞态团队也非常欢迎我加入开源社区，于是就有了 Python 版的洞态 SDK，对于接口进行了封装，能以对象方法的形式请求，同时也能够直接通过对象属性的方式获取数据（IDE 本身会自动提示，使用过程中几乎不用看文档）

```
1 pip3 install dongtai-sdk
2
3 from dongtai_sdk.DongTai import DongTai
4 dongTaiSdk = DongTai("config.json")
```

具体可参看项目主页：

```
1 DongTai-SDK-Python
2 https://github.com/HXSecurity/DongTai-SDK-Python
```

深圳某大型集团公司

——利用 IAST 做开源组件的安全治理

本篇文章于 2022-2-10 发表在火线安全平台公众号

01 背景

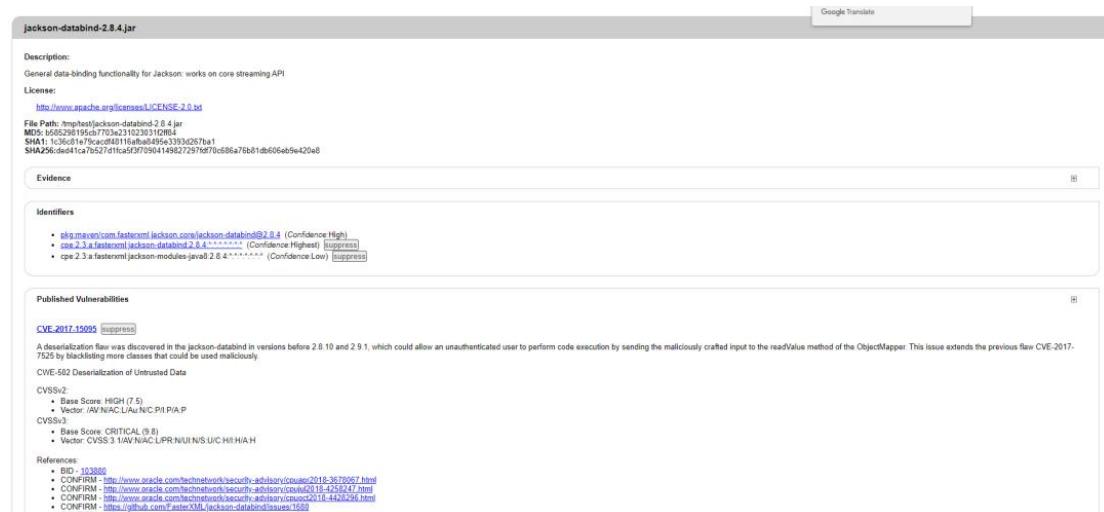
某大型集团在使用洞态一段时间后，发现其配套使用的组件管理的漏洞知识库内容存在滞后性和部分不准确性。倘若将这些内容引入到工单系统中作为安全组件整改任务，那么提供给开发部门的漏洞描述、漏洞危害和修复建议便会显得没有说服力。因此需要寻找实时、有效且免费的安全漏洞知识库的信息来源。

02 解决方案

目前业内开源组件的安全知识库的来源，主要有两种：

- 人工维护的商用漏洞数据库
- NVD 免费漏洞数据库

由于使用洞态提供的组件信息，最好还是集成免费信息来源。由于 dependency check 工具是通过调用 NVD 数据进行组件安全漏洞识别，因此一开始是想基于其扫描原理进行二次开发，从而配合洞态组件信息进行组件检测，但其工作量太高。



The screenshot shows the NVD search results for the file `jackson-databind-2.8.4.jar`. It includes the file path (`http://www.apache.org/licenses/LICENSE-2.0.txt`), SHA-256 hash (`SHA256:de4d1c7b527d1fc5f3799b04149827297d70c586a7b81db606e9e4209a8`), and a list of published vulnerabilities. One notable entry is `CVE-2017-15095`, which is marked as `Exploit` and `High` severity.

于是直接用 python 本地调取 NVD 数据进行识别。

```

1 def nvd_download():
2     for i in range(2002,2022):
3         url = "https://nvd.nist.gov/feeds/json/cve/1.1/nvdcve-1.1-{}.json.gz"
4         r = requests.get(url=url)
5         pathname = os.path.join("/opt/vul",url.split("/")[-1])
6         with open(pathname,"wb") as f:
7             f.write(r.content)
8         f.close()

```

但是在实际使用时，我并不知道如何去构造 CPE 语句进行查询，因此这种方案暂时被放弃了。

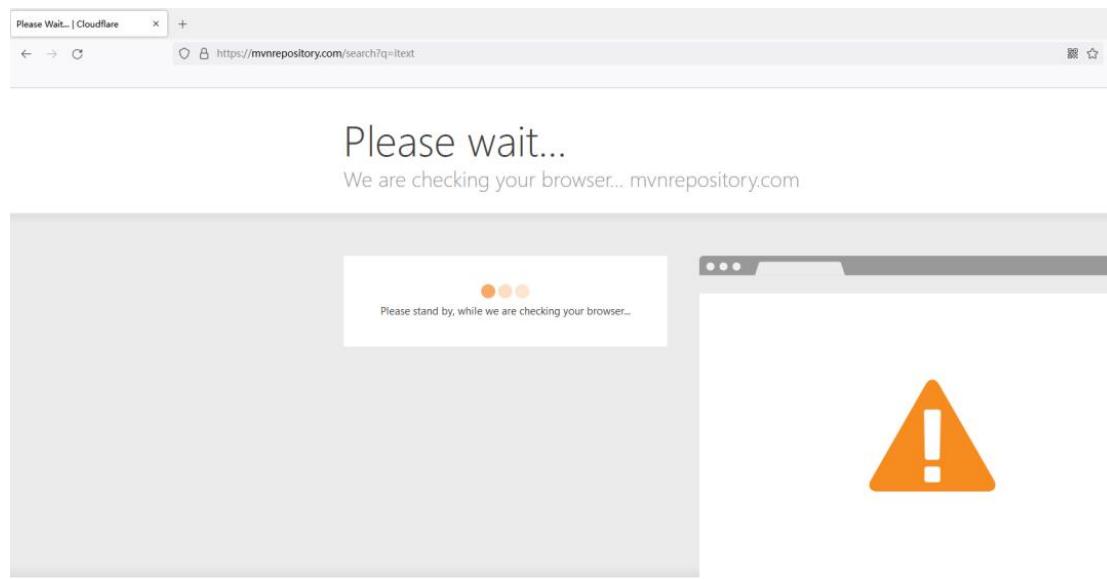


此后，偶然发现在 maven 仓库中居然有每个版本的漏洞信息。那么我是否可以根据自身项目与其做对比，获取安全漏洞信息？

2.10.0.pr1		Central	234	Jul, 20
2.9.10.8		Central	131	Jan, 20
2.9.10.7	13 vulnerabilities	Central	122	Dec, 20
2.9.10.6	15 vulnerabilities	Central	119	Aug, 20
2.9.10.5	17 vulnerabilities	Central	208	Jun, 20
2.9.10.4	21 vulnerabilities	Central	346	Apr, 20
2.9.10.3	33 vulnerabilities	Central	292	Feb, 20
2.9.10.2	34 vulnerabilities	Central	83	Jan, 20
2.9.10.1	35 vulnerabilities	Central	624	Oct, 20
2.9.10	38 vulnerabilities	Central	833	Sep, 20
2.9.9.3	43 vulnerabilities	Central	1,220	Aug, 20
2.9.9.2	43 vulnerabilities	Central	323	Jul, 20
2.9.9.1	45 vulnerabilities	Central	353	Jul, 20
2.9.9	47 vulnerabilities	Central	2,347	May, 20
2.9.x	48 vulnerabilities	Central	2,458	Dec, 20
2.9.8	51 vulnerabilities	Central	1,768	Sep, 20
2.9.7	55 vulnerabilities	Central	1,618	Jun, 20
2.9.6	59 vulnerabilities	Central	1,614	Mar, 20
2.9.5	60 vulnerabilities	Central	1,030	Jan, 20
2.9.4	63 vulnerabilities	Central	634	Dec, 20
2.9.3	63 vulnerabilities	Central	566	Oct, 20
2.9.2	63 vulnerabilities	Central	445	Sep, 20
2.9.1	63 vulnerabilities	Central	622	Jul, 20
2.9.0	31 vulnerabilities	Central	86	Jun, 20
2.9.0.pr4	31 vulnerabilities	Central	88	Apr, 20
2.9.0.pr3				

03 实际落地

当我想通过自动化方式去获取信息时，发现会触发 cloudflare 的五秒盾，进入人机识别。



后面花时间研究了一下绕过 CF 的手段——使用 selenium + chrome 有界

面请求一次，关闭浏览器，再请求一次，关闭浏览器，如此循环，即可绕过（外网 IP 访问很少会触发拦截）。

洞态开源组件 API 提供的组件名字内容如下：

```
1 components = [
2 'log4j-1.2.16.jar',
3 'log4j-1.2.17.jar',
4 'log4j-ecs-layout-0.1.3.jar',
5 'maven:com.fasterxml.jackson.core:jackson-databind:2.8.4:',
6 'maven:com.fasterxml.jackson.core:jackson-databind:2.9.10.6:',
7 'maven:mysql:mysql-connector-java:5.1.34:',
8 'maven:org.springframework.amqp:spring-amqp:1.1.3.RELEASE:',
9 'maven:org.springframework.amqp:spring-amqp:1.7.5.RELEASE:',
10 'maven:org.springframework.amqp:spring-rabbit:1.1.3.RELEASE:',
11 'maven:org.springframework.integration:spring-integration-ftp:4.3.7.RELEASE:
12 'maven:org.springframework.integration:spring-integration-kafka:2.2.0.RELEAS
13 'maven:org.springframework.kafka:spring-kafka:1.2.3.RELEASE:',
14 'maven:org.springframework.retry:spring-retry:1.1.3.RELEASE:',
15 'maven:org.springframework.retry:spring-retry:1.2.1.RELEASE:',
16 'maven:org.springframework:spring-aop:4.3.29.RELEASE:',
17 'mybatis-3.1.1.jar',
18 'mybatis-3.2.3.jar',
19 'mybatis-spring-1.2.0.jar',
20 'xstream-1.3.1.jar',
21 'xstream-1.4.14.jar',
22 'zkclient-0.1.jar',
23 'zookeeper-3.3.6.jar',
24 'abc-SNAPSHOT.jar'
25
26 ]
```

我将其划分为三种类型：

- a. 自带 maven 字符：这种类型可以直接拆分，作为参数内容可以传参；
- b. SNAPSHOT：这种类型定义为业务系统打包生成，忽略；
- c. 其他：这种类型要放到 maven 中搜索，找到其地址。

```
1 for c in components:
2     c = c.replace(".jar", '')
3     if "maven:" in c:
4         print(c+": "+str(maven_component(c)))
5     elif "SNAPSHOT" in c:
6         pass
7     else:
8         print(c+": "+str(search_component(c)))
```

maven 提供的安全漏洞信息分为 direct（直接影响）和 dependencies（当前组件的依赖组件中存在的漏洞），我们关注直接影响，忽略被受影响。

Home » com.thoughtworks.xstream » xstream » 1.4.16



XStream Core » 1.4.16

XStream Core

License	BSD 3-clause
Categories	XML Processing
Date	(Mar 13, 2021)
Files	jar (614 KB) View All
Repositories	Central
Used By	1,995 artifacts
Vulnerabilities	<p>Direct vulnerabilities:</p> <p>CVE-2021-39154</p> <p>CVE-2021-39153</p> <p>CVE-2021-39152</p> <p>View 12 more ...</p> <p>Vulnerabilities from dependencies:</p> <p>CVE-2021-33813</p> <p>CVE-2020-10683</p> <p>CVE-2018-1000632</p>

Note: There is a new version for this artifact

运行结果如下，包含四个字段内容：组件名字、CVE 列表（cvelist）、安全版本（safe_version）、组件名字组织（org）。

```

1 log4j-1.2.16:  ([], '', '')
2 log4j-1.2.17:  ([], '', '')
3 log4j-ecs-layout-0.1.3:  ([], '', '')
4 maven:com.fasterxml.jackson.core:jackson-databind:2.8.4:: ([ 'CVE-2021-20196'])
5 maven:com.fasterxml.jackson.core:jackson-databind:2.9.10.6:: ([ 'CVE-2021-20196'])
6 maven:mysql:mysql-connector-java:5.1.34:: ([ 'CVE-2019-2692'], '8.0.16', 'mysql')
7 maven:org.springframework.amqp:spring-amqp:1.1.3.RELEASE:: ([], '', 'org.springfram')
8 maven:org.springframework.amqp:spring-amqp:1.7.5.RELEASE:: ([], '', 'org.springfram')
9 maven:org.springframework.amqp:spring-rabbit:1.1.3.RELEASE:: ([], '', 'org.springfram')
10 maven:org.springframework.integration:spring-integration-ftp:4.3.7.RELEASE:
11 maven:org.springframework.integration:spring-integration-kafka:2.2.0.RELEASE:
12 maven:org.springframework.kafka:spring-kafka:1.2.3.RELEASE:: ([], '', 'org.springfram')
13 maven:org.springframework.retry:spring-retry:1.1.3.RELEASE:: ([], '', 'org.springfram')
14 maven:org.springframework.retry:spring-retry:1.2.1.RELEASE:: ([], '', 'org.springfram')

```

◀ ▶

最后将 CVE 放到 CVE search 中查找到漏洞描述和漏洞评分。

```

{
  "id": "CVE-2016-3333",
  "modified": "2018-10-12T22:12:00",
  "published": "2016-11-10T06:59:00",
  "access": {
    "assigner": "cve@mitre.org"
  },
  "impact": {
    "availability": "COMPLETE",
    "confidentiality": "COMPLETE",
    "integrity": "COMPLETE"
  },
  "last-modified": "2018-10-12T22:12:00",
  "references": [
    "https://www.acm孑nline.org/10.1145/2960000.2960001"
  ],
  "summary": "The Common Log File System (CLFS) driver in Microsoft Windows Vista SP2, Windows Server 2008 SP2 and R2 SP1, Windows 7 SP1, Windows 8.1, Windows Server 2012 Gold and K2, Windows RT 8.1, Windows 10 Gold, 1511, and 1607, and 1625 editions, contains a privilege elevation vulnerability. An attacker can exploit this vulnerability via a crafted application to gain system-level privileges. This is a different vulnerability than CVE-2016-0026, CVE-2016-3332, CVE-2016-3334, CVE-2016-3335, CVE-2016-3336, and CVE-2016-7284."
}

```

陌陌

——问题驱动的 IAST 落地、改造之旅

本篇文章于 2022-4-22 发表在火线安全平台公众号

本篇文章由陌陌企业安全工程师的角度阐述：

- 1、为什么 IAST 是最佳选择
- 2、为什么选择洞态 IAST
- 3、洞态 IAST 在落地实践中遇到的问题及解决方法

01 背景 我们为何需要 IAST?

在回答这个问题之前，我们首先得知道 IAST 是什么？

常见的 Web 应用安全测试技术：

DAST

动态应用程序安全测试（Dynamic Application Security Testing）技术在测试或运行阶段分析应用程序的动态运行状态。它模拟黑客行为对应用程序进行动态攻击，分析应用程序的反应，从而确定该 Web 应用是否易受攻击。（应用检测）

SAST

静态应用程序安全测试（Static Application Security Testing）技术通常在编码阶段分析应用程序的源代码或二进制文件的语法、结构、过程、接口等来发现程序代码存在的安全漏洞。（源码层面检测）

IAST

交互式应用程序安全测试（Interactive Application Security Testing）是 2012 年 Gartner 公司提出的一种新的应用程序安全测试方案，通过代理、VPN 或者在服务端部署 Agent 程序，收集、监控 Web 应用程序运行时函数执行、数据传输，并与扫描器端进行实时交互，高效、准确的识别安全缺陷及漏洞，同时可准确确定漏洞所在的代码文件、行数、函数及参数。IAST 相当于是 DAST 和 SAST 结合的一种互相关联运行时安全检测技术。

仅凭上述文字描述不够直观，下面我将结合漏洞样例来讲解三种安全测试技术的区别。

现有如下 Java Web 应用，它对外暴露的 URL 是：<http://www.immomo.com/exec?cmd=ls>

```

1 public class CommandExec extends GenericServlet {
2
3     @Override
4
5     public void service(ServletRequest servletRequest, ServletResponse ser
6
7         String cmd = servletRequest.getParameter("cmd");
8
9         String cmd2 = change(cmd);
10
11         Runtime.getRuntime().exec(cmd2);
12
13     }
14
15
16     public String change(String str){
17
18         return str+"xxxx";
19
20     }
21
22 }
```

示例代码 1-命令执行漏洞

DAST 之困

DAST 技术的实现通常就是一个漏洞扫描器，漏扫通过向目标发送大量 payload 来进行漏洞检测，如下：

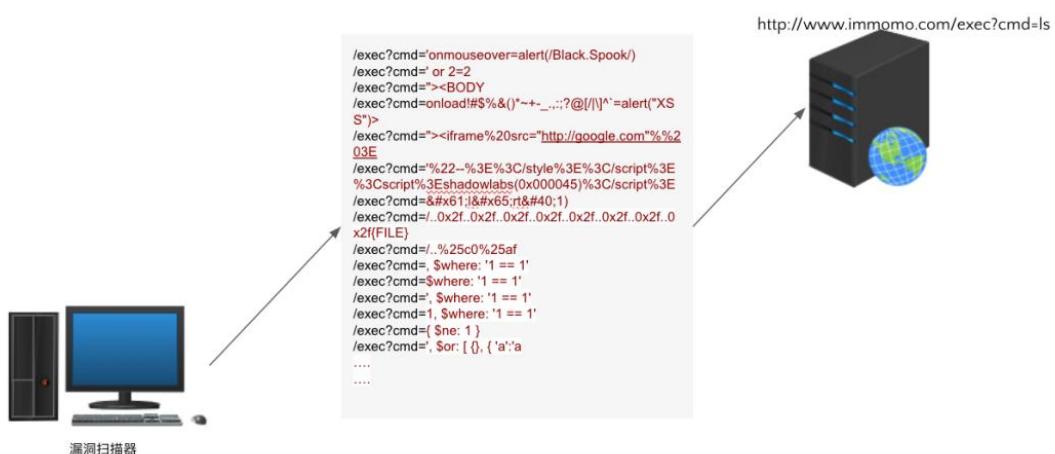


图 1-DAST 检测原理

那么它的缺点也就很明显了：

1、流量大

2、脏数据多

在 DAST 视角下，应用程序是一个黑盒子，覆盖面难免会低一点。

SAST 之困

在 SAST 技术的视角下，目标应用则一览无遗，毕竟是源码层面的检测，覆盖面大大提高，但是静态代码检测最大的缺点可能就藏在它的名字里——静态，这种检测方式是脱离了应用程序运行上下文的，一个简单的 if 判断可能就会引入一个误报，例如如下伪代码：

```
1 Boolean flag = false;
2 String cmd = servletRequest.getParameter("cmd");
3 if(flag){
4     Runtime.getRuntime().exec(cmd);
5 }
```

无论工程师们是使用关键词匹配还是抽象语法树(AST)进行自动化代码审计，这个问题都比较头疼。

如上所述，SAST 覆盖面虽广，但是误报太高，安全人员需要花费大量精力处理这些误报，而目前利用 QL 进行代码审计的方式我了解不深，就不予置评了。

建设企业安全没有一站式的解决方案，也没有完美的工具，只有适合解决某个问题的工具，上述两种技术虽有其局限性，但是在特定的领域依旧可以发光发热。

例如漏洞扫描器可以在 0day 出现时帮助安全人员快速发现风险点，静态代码审计工具可以嵌入 CI/CD 流程中快速进行高危漏洞筛查等等.....

在过去的时间里，陌陌在 DAST 以及 SAST 上已经有些成果，但是还不能满足我们的需求，我们希望拥有一个准确度高覆盖面广的安全测试工具，在项目上线前对应用进行安全检测，而 IAST 正好可以满足。

IAST——新的视角

我们先来看看 IAST 的检测原理：

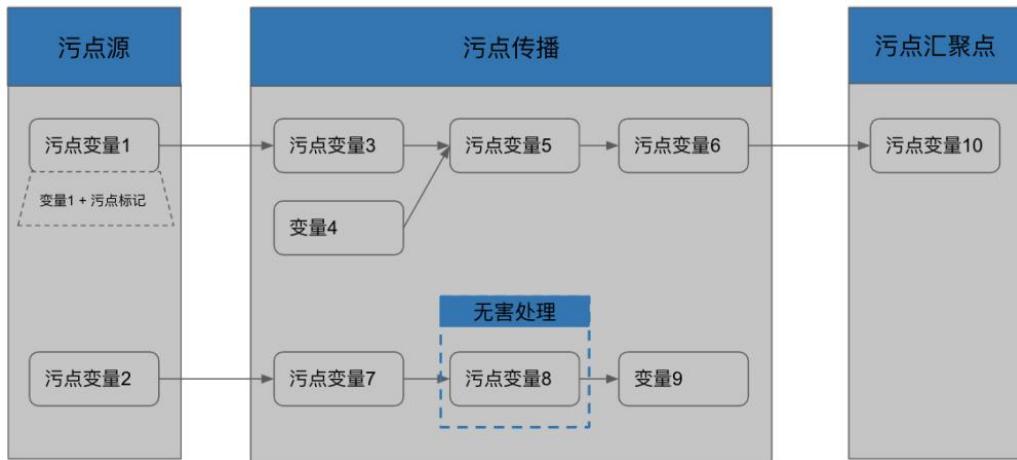


图 2-IAST 检测原理

IAST 会把一次完整请求中所涉及到的方法分为一个三元组，或者说是分为三类：**污点来源节点、污点传播节点、污点汇聚节点**。

对于这三类方法节点中所涉及到的变量，有不同的处理策略，例如：

- 一个来自污点来源节点的返回值变量，我们认为它是一个污点变量。
- 一个污点变量作为入参流经传播节点，那么该传播节点方法的返回值也将被标记为污点变量。
- 如果一个污点变量作为入参流入了汇聚节点，那么我们就认为一个漏洞產生了。

还是以示例代码 1-命令执行漏洞 为例，我们姑且认为本样例只涉及如下三个方法：

- servletRequest.getParameter()，污点来源
- CommandExec.change()，污点传播
- Runtime.getRuntime().exec()，污点汇聚

那么，在 IAST 的视角下，它们分别就是污点来源节点、污点传播节点、污点汇聚节点。此时当一个用户请求了该接口就会执行如下逻辑：

由于 `servletRequest.getParameter()`是污点来源节点，所以它的返回值会被加入到污点池中（污点池可以是一个 `Set` 类型的对象）

当方法 `CommandExec.change()`执行时，IAST 会获取到其入参并检查它是否在污点池中，如果在，那么 IAST 会获取该方法的返回值并将其也加入到污点池中

最后 Runtime.getRuntime().exec() 执行时，IAST 同样会获取其入参并判断它是否在污点池中，如果在，则认为此处存在漏洞

ps：上述流程为阐述方便而省略了细节，实际上 IAST 还有诸多问题需要解决。

那有人可能就会问了——IAST 是依靠什么执行上述策略的呢？

依靠的就是 hook 技术，IAST 会通过附加一个 Agent 到目标应用从而 hook 一些关键函数，对于 java 来说，就是重写这些函数的字节码从而在方法执行前后插入我们需要的逻辑。

通过上述原理，我们可以看出来：

IAST 的误报是很低的，这满足了我们对准确度的需求。

而覆盖面则取决于我们的 Agent 部署了多少业务，部署的多覆盖的多，这满足了我们的覆盖面广的需求。

02 选择 为何是洞态 IAST?

技术选型确定了，我们还得做产品选型，国内 IAST 产品也不多，于是我们联系了国内所有的 IAST 产品厂商，联系方式基本就是在官网填问卷，然后等着厂商给你打电话。

在与厂商的交流中，主要就是他们做产品介绍，然后我们提一些关注的问题，对于商业的 IAST 可以申请部署试用，但是我们还没进行到那一步，因为陌陌内部使用了自研的 RPC 框架，商业的 IAST 产品都无法支持（这也是可以预见的，当然，这些产品都支持定制，是否需要额外收费我并不清楚）。

对于洞态 IAST，得知其开源后我们就开始折腾了，对于一个工程师来说，省去了一道沟通的环节，能直接使用产品，了解其工作细节这一点就很友好了。

一番调研下来，陌陌最终选择了洞态，原因如下：

可二次开发。

陌陌内部使用了自研的 RPC 框架，所以我们需要定制化开发。

开源。

社区可反馈安全策略，保证漏洞覆盖的广度和深度。

轻 Agent 重服务端。

Agent 端只负责数据采集，对业务影响相对较小。

产品选型确定了，并不代表调研工作结束了，后来我们花时间通读了洞态 IAST Agent 的代码，对其有了更细致的了解，一半是因为对其实现感兴趣，一半是为了后续的改造工作。

还记得我前面提到的陌陌内部使用了自研的 RPC 框架吗，我们开始调研时，洞态 IAST 还处于 1.0.6 版本，还没有支持 RPC 漏洞的检测，而我本人也没相关经验，于是我们面临了第一个问题——如何做内部 RPC 框架下的漏洞检测？

03 实践 改造之路

如何做内部 RPC 框架下的漏洞检测？

在讲如何适配之前，我们需要先了解洞态 IAST 目前是怎么检测漏洞的。

洞态 IAST Agent 会把一次请求中涉及到的每个 Method 都封装到一个 MethodEvent 对象里，这个对象存储着一个方法调用的入参、返回值、方法签名等信息，然后这些 MethodEvent 会被加入一个全局的 TRACK_MAP 对象里，并最终在一次请求结束后被 Agent 上传到云端，一次请求中涉及到的所有 method 就组成了一个 methodpool，如下：

```

method_pool = [list: 0] [{"args": "", "source": False, "invokeid": 15, "className": "java.lang.String", "signature": "java.lang.String.getBytes()", "interfaces": ["java.lang.CharSequence", "java.lang.Comparable", "java.lang.String", "java.io.Serializable"], "methodName": "getBytes", "sourceHash": 10460619853, "targetHash": 10460619853}, {"args": "", "source": False, "invokeid": 15, "className": "java.lang.String", "signature": "java.lang.String.getBytes()", "interfaces": ["java.lang.CharSequence", "java.lang.Comparable", "java.lang.String", "java.io.Serializable"], "methodName": "getBytes", "sourceHash": 10460619853, "targetHash": 10460619853}, {"args": "", "source": False, "invokeid": 14, "className": "java.lang.ProcessBuilder", "signature": "java.lang.ProcessBuilder", "interfaces": ["java.lang.ProcessBuilder"], "methodName": "c", "sourceHash": 10460619853, "targetHash": 10460619853}, {"args": "", "source": False, "invokeid": 14, "className": "java.lang.String", "signature": "java.lang.String.substring(int,int)", "interfaces": ["java.lang.CharSequence", "java.lang.Comparable", "java.lang.String", "java.io.Serializable"], "methodName": "substring", "sourceHash": 10460619853, "targetHash": 10460619853}, {"args": "", "source": False, "invokeid": 13, "className": "java.lang.Runtime", "signature": "java.lang.Runtime.exec(java.lang.String)", "interfaces": ["java.lang.Runtime"], "methodName": "exec", "sourceHash": 10460619853, "targetHash": 10460619853}, {"args": "", "source": True, "invokeid": 11, "className": "javax.servlet.ServletRequest", "signature": "javax.servlet.ServletRequest.getParameter(java.lang.String)", "interfaces": [], "methodName": "getParameter", "sourceHash": 10460619853, "targetHash": 10460619853}], "source": "local", "target": "remote", "invokeid": 11]
    
```

图 3-MethodPool

云端的漏洞检测就是针对这个 methodpool 进行的，大致逻辑如下：

- 通过 invokeid 对 methodpool 进行逆序排序
- 遍历 methodpool，找到危险 sink 点，比如 Runtime.getRuntime().exec()
- 拿到 sink 点的入参，并和 methodpool 中的其他方法的返回值进行比较，匹配上了，则说明找到了 sink 点的上游 method

- 拿到上游 method 的入参，继续和 methodpool 中的其他方法返回值进行比较

-

- 按照上述逻辑，回溯到 source 点则结束

上述逻辑执行完后，不仅漏洞检出了，漏洞调用链也成功梳理出来了。值得一提的是，上面的入参和返回值比较都是比较的 hash 值。

可以看到，最初的洞态 IAST 只能在一个 methodpool 里进行漏洞检测，但是 RPC 调用是涉及到多个服务的，也就是需要串联多个 methodpool 发现漏洞。

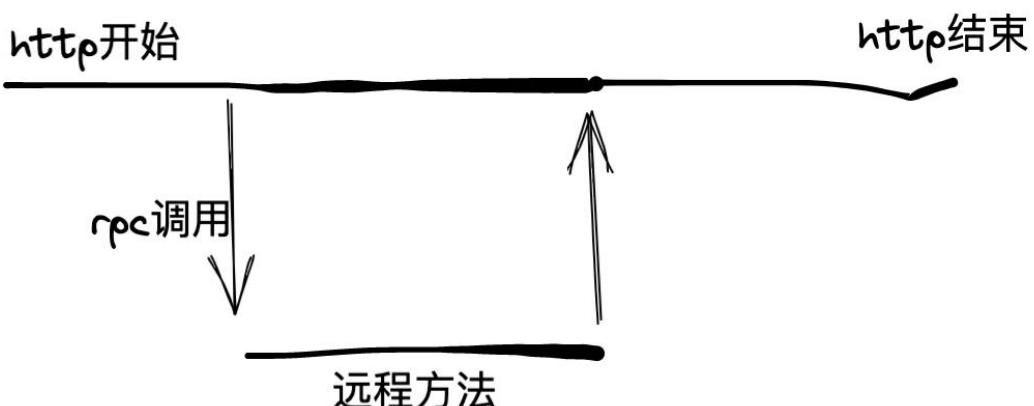


图 4-RPC 调用

于是我们的问题变成了——如何关联多个 methodpool?

一个经验丰富的 java 工程师应该一下就可以想到使用 traceid 来进行方法跟踪，但是我并不是，于是最初就诞生了一些奇奇怪怪的方案，当然都被自己否定了，最终还是使用了 traceid。

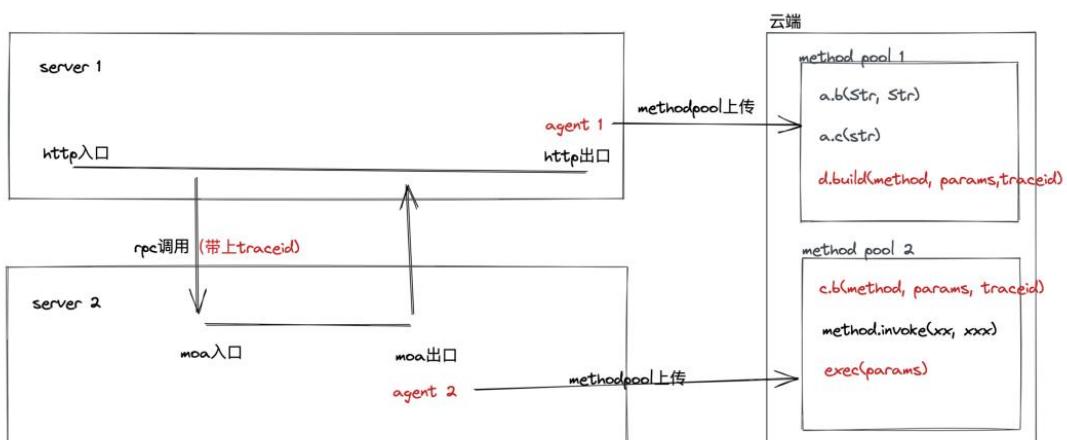


图 5-使用 traceid 关联多个 methodpool

上图是一个使用 traceid 关联多个 methodpool 的概念图，其中 server1 是对外暴露接口的 http 服务，server2 部署着 RPC 服务（陌陌内部 RPC 框架简称 MOA）。

大多数的 Java RPC 框架都是使用动态代理技术实现的，无论我在 server1 调用什么方法，RPC 框架最终都会通过网络把本次方法调用的信息（方法名、参数）发送到 server2，server2 才是真正执行方法的一方，而将方法调用信息发送到远端的这个操作肯定都是通过同一个或者某几个方法实现的，只要我们 hook 了这个方法，并且将 server1 生成的 traceid 一并发送到 server2，两个 methodpool 不就产生联系了吗？

我再具体点，MOA 框架下，当 consumer 端要调用一个远程方法时，每次都会调用到 `com.immomo.moa.service.consumer.invocationhandler.AbstractInvocationHandler.invoke(com.immomo.moa.api.Request)` 这个方法，其中 `com.immomo.moa.api.Request` 就封装了本次远程调用的相关信息，于是我 hook 了这个方法，并通过反射的方式向 Request 对象里写入了 agent 生成的 traceid。

consumer 端作为消费者发起请求，那么 provider 端就有一个负责接收请求的逻辑在，provider 端每次接收到请求都会调用 `com.immomo.moa.service.provider.AbstractProcessor.invoke(java.lang.reflect.Method,java.lang.Object[],com.immomo.moa.api.Request,com.immomo.moa.api.Response)` 这个方法，简单解释下几个参数的意义：

- 第一个参数：本次远程调用的具体方法
- 第二个参数：本次远程调用的方法的参数
- 第三个参数：consumer 端发送过来的 Request 对象
- 第四个参数：将要返回给 consumer 端的 Response 对象，里面会封装本次远程调用的结果

同样的，我只需要 hook 这个方法，并从 request 对象中取出 traceid 就行

现在关联 methodpool 的方案倒是有了，那要怎么实现多个 methodpool 之间的污点追踪呢？还记得我前面提到的吗，洞态 IAST 在同一个 methodpool 中进行污点追踪是通过比较 hash 值实现的，而现在两个 methodpool 中的 hash 值可就不一样了啊

我们最终采用了值比较的方式，当然不是两个 methodpool 的所有方法都通过值比较进行串联，而只是两个 methodpool 的连接点处进行值比较。

其实，单纯的想要检测 RPC 场景下的漏洞，也可以不进行 methodpool 的串联，而只是简单地将 AbstractProcessor.invoke() 方法的第二个参数全部加入污点池中就行。

但是很显然这个方法有个弊端，那就是会提高误报率，把第二个参数全部加入污点池意味着我们认为 consumer 端传过来的参数全部都是来自用户可控的外部输入，而实际上，consumer 传过来的参数有一部分是内部变量，而不是外部输入，如果这些内部变量流入了 sink 点，iast 也会认为它是一个漏洞，这不就误报了吗。所以，陌陌内部采用的方案还是使用值比较筛掉内部变量可能导致的误报。

现在，我们能够很轻松地串联起两个节点的 RPC 调用链了，那如果 RPC 调用不止两个服务呢？我们又将面临什么问题呢？

3 层、4 层甚至更多层的 RPC 服务间调用，IAST 怎么才能知道他们的调用顺序并进行正确的调用链拼接。

如果 RPC 方法调用就一个 consumer 和一个 provider，很容易区分出他们的调用顺序，但是 RPC 调用层级一多，我们又该怎么确定呢？

实际上，还是使用 traceid 解决，在阅读了 skywalking 的 traceid 设计方案后，我们学会了用 traceid 中的 spanid 进行节点顺序标记：

traceid 用.号分成了几段，最后一段就是 spanid，它会随着在节点间的传递而递增，这样我们在 IAST 服务端进行调用链拼接的时候就可以根据 spanid 确定他们正确的调用顺序。

当我们在做内部 RPC 适配的时候，洞态 IAST 开源版本也支持了 RPC 漏洞的检测，自然，他们已经有一套 traceid 生成方案了，于是，我们就直接当了一波“源码小偷”，高呼“开源万岁”。

现在已经能够检测内部 RPC 框架下的漏洞了，那么在前端该怎么展示这种类型的漏洞呢？

对于多节点的 RPC 调用，怎样展示污点流图？

在具体开始开发这块功能之前，我们和洞态 IAST 团队进行了深入的沟通交

流，他们也向我们展示了洞态 IAST 商业版本的解决方案，当发现 RPC 类型的漏洞时，商业版洞态 IAST 展示方式如下：



图 6-商业版本 RPC 漏洞污点流图

商业版本里通过在前端新增了一个页面的方式来比较清晰的展示 RPC 调用链，当时看到这个展示方案，我觉得很是清晰。

但是囿于我和小伙伴儿都没有 Vue 的开发经验，所以很难照搬其解决方案（洞态的同学表示可以支持，但是由于各种原因，我们还是自己弄了，向洞态 IAST 团队 respect），所以我们选择曲线救国，直接在开源版本现有的污点流图中展示整个 RPC 污点流图（见图 7），这样既确保了能够展示完整污点流图，又保证了项目进度稳步推进。

污点流图



图 7-陌陌内部 RPC 漏洞污点流图

图 7 展示的是一个跨越了 3 个节点的命令执行漏洞的污点流转过程，每两个节点之间是通过 `AbstractInvocationHandler.invoke()` 以及 `AbstractProcessor.invoke()` 进行拼接的，这在上图中有所体现。

拼接逻辑很简单，一图胜千言：

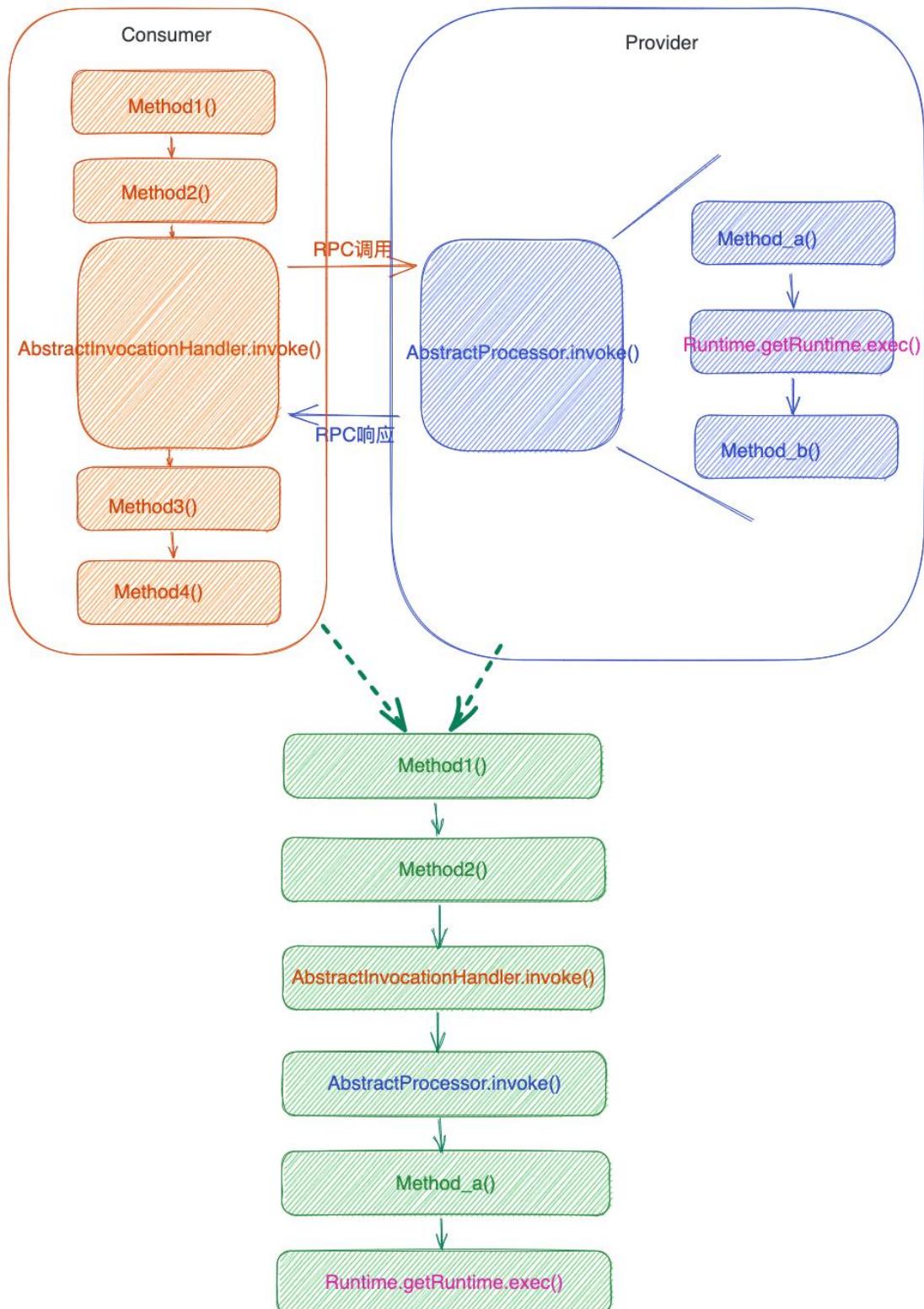


图 8-MethodPool 拼接方式

其中 `AbstractProcessor.invoke()` 和 `Method_a()`、`Runtime.getRuntime.exec()` 本身是包含关系，但是为了处理方便，在 methodpool 里他们被处理成了并列关系。

一切看起来都那么美好，而实际上还有问题需要处理。

远程方法调用的参数是字符串时，可以很容易地通过值比较的方式进行污点跟踪，可是如果参数是 Map、List 以及其他自定义对象时该怎么追踪？

试想一下这样一个场景：consumer 端将用户输入的一个参数通过 Map.put()方法放入了 map 中，然后这个 map 作为远程方法调用的参数传到了 provider 端，provider 通过 Map.get()方法取出了其中一个值，最终这个值流入了 sink 点。

如果这是一个漏洞，你要怎么检测它？

很粗暴的方案：

consumer 端： 污点流入了 map，所以 map 也是污点

provider 端： 获取到远程传过来的 map 对象，取出 map 中的每一个成员，并将其放入污点池，这样 map.get()的返回值就都是污点了。

然后比较 consumer 端的 map 和 provider 端的 map 是不是同一个，比较的方式是比较两个 map toString 过后的值是否一样。如果一样，consumer 和 provider 的 methodpool 就串起来了，认为漏洞存在。

当然，这种方案存在误报，因为通过值比较只可以大致确定 map 是同一个，而不能确定 map.get()的值是不是 consumer 端 map.put()放进去的值。

list 的处理方式和 map 类似，而自定义对象的处理要特殊一点，自定义对象需要通过反射的方式取出其属性，并且在进行值比较的时候也不能直接比较对象 toString 后的结果，因为自定义对象 toString 过后通常会有如下格式：

obj@随机字符串

在不同的 jvm 中，这个随机字符串是不一样的，这儿需要单独处理下。

改造的差不多了，我们准备上线了，上线不久就遇到了一个问题。

在部署了 IAST Agent 过后，某些接口出现超时的情况，这是为什么？

先说原因，是因为这些接口中处理了大量的 map 类型的数据，而 Map.get()方法是 agent 的一个 hook 点，agent 在 hook 的时候会去计算该 Map 的 hashCode，而 map 类型的 hashCode 的耗时取决于 map 的长度，长度一大，耗时就上升了。

我们对此逻辑进行了优化：之前是无论有没有污点流入 Map，都会计算该 Map 的 hashCode，现在只有当污点流入了该 Map 才计算，大大地减少了 agent 的计算量。

ps：这也是一个让我们挺头疼的问题，因为问题出现在业务方的环境下，我

们本地很难模仿业务环境并复现问题，而有问题的 Agent 又不能一直在业务方的机器上跑着，因为还有 QA 同学需要测试，不能影响他们的工作，只能趁着业务方的同事空闲的时候介入排查，最终在业务方同事及洞态 IAST 团队人员的帮助下，我们定位到了问题，再次向洞态 IAST 团队同学 respect。

在 1.4.1 版本中，洞态 IAST 新增了性能管理功能，可以很好的解决上述超时问题，而我们也是在第一时间着手引入该功能到我们的内部版本。

04 洞态 IAST 落地实践

目前，洞态 IAST 在陌陌的落地还处于初期，可能没有特别多的经验分享给大家，勿怪。下面我还是会以自我提问的方式来分享洞态 IAST 在陌陌部署过程中，我们的考虑以及相关行动。

1、如何开展部署工作？

方案：根据业务划分，把 Agent 部署到对应业务的所有服务上（API 以及 RPC 服务）。

因为陌陌内部大量使用 RPC，如果我们随性部署 Agent，无异于没有部署。

2、如何让部署工作更规范？

方案：请求陌陌发布系统同事支持，在陌陌发布系统的容器配置处增加了「接入 IAST」功能，使 IAST 真正融入了发布流程中。

这并不是我们在季度初（2022Q1）就想要做的一项工作，我们原本的打算是完成 IAST 对陌陌内部 RPC 框架的适配后手动部署几个业务线试试水，但是得益于适配工作进展顺利，我们有充足的时间来推进部署，于是想着让我们的部署工作变得更“Smart”，当然，在发布系统功能开发的过程中，我们还是经历了一段时间的手动部署，此时的部署十分保守，主要是为了观察 Agent 的稳定性。

发布系统的功能上线后，我们的部署效率有了提升，一周内部署了好几个业务，但是我们在推进过程中，发现仅仅是部署某个业务自己的 API 和 RPC 服务是很难发现问题的，某个业务自己项目结构通常就是一个 API 对应着好几个 RPC 服务，只有这么简单的一层，而实际上，一个 RPC 服务还可能依赖其他基础的 RPC 服务，例如账号、礼物、支付等服务，要想实现 IAST 的最大收益，我们必须把一个入口所涉及的所有 RPC 服务都给部署上 Agent，这就引出了下一个问题。

3、如何梳理业务背后所依赖的 RPC 服务？

方案：陌陌发布系统有这样两个功能

- 1.根据 appkey 查看某个项目依赖的所有 service uri。
- 2.根据 service uri 查看该 uri 属于哪个 appkey。

于是我们写了一个脚本，通过这两个接口梳理出了一棵依赖树

4、如何开展部署工作？

方案：和对应业务团队的领导沟通，取得部署支持。

这里的友好是对各个业务团队的友好，虽然我们一开始在部署 Agent 时就是通过先沟通再部署的方式，但是在经历了之前提到的某业务出现的 Agent 性能问题之后，我们对部署这一工作更加谨慎，虽然 Agent 是部署在预发布环境，但是如果由于 Agent 出现问题导致服务不可用，对于 QA 以及研发人员都会带来时间的损失，所以我们希望各个业务线都知道我们部署了这个 Agent，以便出现问题可以立即回退，所以和各业务线领导沟通是比较好的解决方案。

05 最后 总结

本文是洞态 IAST 在陌陌落地的一个阶段性回顾，全文都是以问题驱动的，这也是我们在实践过程中最真实的状态，发现问题->解决问题->发现新的问题->解决新的问题->...，回过头看，其实很多问题对于熟手来说算不上问题，只因我们缺少相关经验，所以走了很多弯路，所以希望本文中的经验能够帮助到大家！

END

首先感谢陌陌企业的信任与支持，洞态 IAST 自发布以来一直实现集自动化测试、人工测试、自动化检测于一体，多语言多场景覆盖，采用动态污点追踪被动插桩模式，解决传统应用安全测试过程中重放数据包产生的脏数据问题。

陌陌

——可适配自研 RPC 框架的 IAST

本篇文章于 2022-6-10 发表在火线安全平台公众号

关于陌陌

MOMO 是挚文集团推出的一款基于地理位置的移动社交应用，是中国领先的开放式社交平台之一。

在 **MOMO**，你可以通过视频、文字、语音、图片来展示自己，基于地理位置发现附近的人，加入附近的群组，建立真实、有效、健康的社交关系。

挚文集团的使命愿景是“连接人，连接生活”。

和洞态的结缘

为什么上线前做 IAST 安全检测？

陌陌是去年十月份开始接触 IAST 技术，因为陌陌现有的自动化安全测试体系，并不能满足我们的需求。目前除了 IAST 技术之外，SAST、DAST 这两项技术在陌陌都已经有了落地的实践，我们在上线前有针对源码层面的白盒扫描器，也有对线上业务进行扫描的黑盒扫描器，这两项技术都有自己的用武之地，当然也有一些弊端。

白盒扫描器最让人诟病的一点是高误报，在一个企业当中，可能安全人员的占比本来就不高，这种高误报让安全人员没有精力一条一条处理。

黑盒扫描器的实验原理，是对同一个接口发送大量 Payload，根据响应来判断是否存在漏洞，从实现原理可以看出，第一个弊端就是流量大；第二个弊端，有些接口会把测试用的 Payload 代入数据库中，这就导致可能会有脏数据的产生。正是由于这两个弊端，导致黑盒扫描器不能覆盖到比较敏感的业务，就导致它的覆盖率会低一点。

所以我们就想有没有一个解决方案，既可以保证漏洞的高检出率，而且覆盖面还不错，并且没有脏数据，所以我们把矛头瞄准在 IAST 这项技术。

陌陌为什么选择洞态？

了解到洞态，最开始是通过朋友圈的口口相传，洞态的开源间接促进了陌陌

对 IAST 技术的落地实践，因为陌陌内部大量使用了自研的 RPC 框架，这就导致无论我们选择哪一款 IAST 产品，都得自己动手改造来适配我们的框架（**小编点评：洞态 IAST 商业版支持适配企业自研框架，帮助企业节省手动改造的时间哈**）。

所以洞态的开源优势一下就体现出来了，我们可以直接拿到代码，直接可以开始改造，从而省去多余的沟通环节。

开始因为洞态的开源选择了它，但是在我们使用一段时间过后，我们发现洞态的优势不仅仅是代码开源。

开源社区很活跃

洞态开源社区很活跃，我们有时候在群里反馈一个 bug 或者是一个新的特性，洞态团队在下一版本就可以立马修复这个问题，然后帮我们实现新的功能，这一点我特别喜欢，点赞！

轻 Agent 重服务端设计

我还特别喜欢洞态轻 Agent 重服务端的设计，优势是如果 Agent 运算量过大，会对业务有一个比较明显的影响，所以轻 Agent 重服务端，让我们在 Agent 的性能上更好一点。

对于安全人员来说，推进部署会方便一点，如果你的 Agent 有问题，影响业务的正常运行，业务不会愿意进行部署。

洞态在陌陌落地实践

洞态在陌陌已经应用到哪个阶段？

因为我们前期花了一些工作在内部 RPC 框架的适配上，所以我们的部署工作进展比较缓慢，由于陌陌安全在整个陌陌公司的属性，我们推进部署 Agent 的方式并不是找运维同学批量接入，而是由安全同学对业务团队对接，进行部署，所以我们的部署工作是比较麻烦缓慢的，目前部署了几十台。

您觉得洞态哪个功能实用性最高，与陌陌实际业务场景更匹配？

在一段时间的使用下来，我最喜欢的是 Agent 管理相关的功能，无论是自动降级还是 Agent 热更新，我都特别喜欢，我觉得这两个功能特别赞。

自动降级功能解决了我们之前在业务上遇到的，部署 Agent 导致业务接口超

时的问题。

Agent 热更新主要是方便了更新 Agent，因为我们改动比较多，更新频繁，所以我觉得热更新很赞。

对洞态社区的贡献 or 想法

基于公司业务实现需要，在开发过程中，对洞态做了哪些升级和改造？

陌陌内部使用自研的 RPC 框架，所以我们对洞态做了适配，在这个过程中，我们一直和洞态团队的成员保持着密切的沟通，我们反馈了自己遇到的问题和一些想要的新特性，洞态也都很好的支持了我们。

在部署洞态 IAST 这件事上，开发人员是否难以接受导致推动困难，您是怎么克服的？

在部署的过程当中，可能经常会遇到来自业务方同学的友好发问，比如说 Agent 对代码是有侵入性的，出了问题怎么办？

像这种时候，你就得耐心的给他阐述，我们为了 Agent 的稳定性都做哪些工作，以及如果真的出了问题，有哪些措施可以解决这个问题。

对洞态的期待

根据公司实际应用场景，你最期待洞态 IAST 未来会开发的新功能是什么？

我一直想要一个功能，就是能否在服务端提供一个开关，就是配置项【是否开机即注册引擎】。现在 Agent Jar 包只负责装载 Core 以及其他 Jar 包，有了这个开关之后，我们就可以在 Agent 服务启动的时候，不立马注册 Core 的以及字节码的重写，而是由安全人员手动进行 Core 注册以及字节码的重写，这样安全人员对于 Agent 控制感更强一点。

还有一个比较期待的功能，如果 Agent 部署对业务方造成了影响，是否可以直接卸载掉这个 Agent，卸载过后还能还原业务方的代码字节码，这就要求我们 Agent 在启动的时候要保存原字节码。当然我后来想了一下，就放弃了这个想法，因为可能会带来一定的性能影响。当然如果洞态的同学有比较好的解决方案，我还是特别期待这个功能。

58 集团

——与洞态 IAST 团队联合开发熔断降级功能避免对业务产生影响

本篇文章于 2022-5-25 发表在火线安全平台公众号

前言

本文主要记录了 58 集团和火线洞态联合共建 58 云原生应用场景下安全产品部署前设计监控降级方案全过程，帮助 58 保证安全检测的可用性及稳定性。

No 1 58 集团简介

58 集团是国民服务大平台，是国内专业的“本地、免费、真实、高效”生活服务平台，为了保障企业产品和用户安全，58 集团信息安全部从公司安全架构层面开始逐步实践 SDL 落地“安全左移”，努力在应用发布到线上之前就将安全漏洞发现并修复，用最低的成本解决安全漏洞的问题。

No 2 58 为什么选择洞态

目前，58 集团先后完成了 DAST、SAST 的建设，为了实现更细粒度的漏洞检测和纵深检测，21 年 Q4 开始着手发起 IAST 项目。

DAST 是效果比较直观成本相对较低的检测手段，但缺点很明显：覆盖率不足且会产生脏数据引起业务问题。

SAST 相对来说成本高一些：因为国内没有运营成本比较低的白盒产品；从事静态分析的人员也比较少，很难培养合适的安全人员；最重要的是白盒会产生很多误报，这类误报的过滤也会带来一些额外的人力、信任成本。

58 安全部门希望拥有一个准确度高覆盖面广，方便测试融入安全的安全测试工具，在项目上线前对应用进行安全检测，而 IAST 正好可以满足。

IAST 是介于黑白盒之间的产品，依赖测试请求，同时又使用污点分析进行漏洞建模。优点是误报率相对较低，不会产生脏数据。

伴随上述问题，58 开始对 IAST 类产品进行了相关的调研。通过调研发现洞态比较符合 58 集团业务的需求。

1. 洞态被动插桩式检测。 不需要额外的 *fuzz* 流量，获取应用中的数据流和控制流进行污点分析。

2. 开源。 可自主使用，社区可反馈安全策略，保证漏洞覆盖的广度和深度。

3. 轻 Agent 重服务端。 *Agent* 端只负责数据采集，对业务影响相对较小。

4. 联合开发定制化功能。 58 内网测试服务对稳定性要求极高，安全产品的部署需优先保证业务方服务的可用性及稳定性，因而需要设计监控降级方案。

经过双方多次沟通，最终决定联合开发解决在 58 场景里最关心的稳定性相关问题。

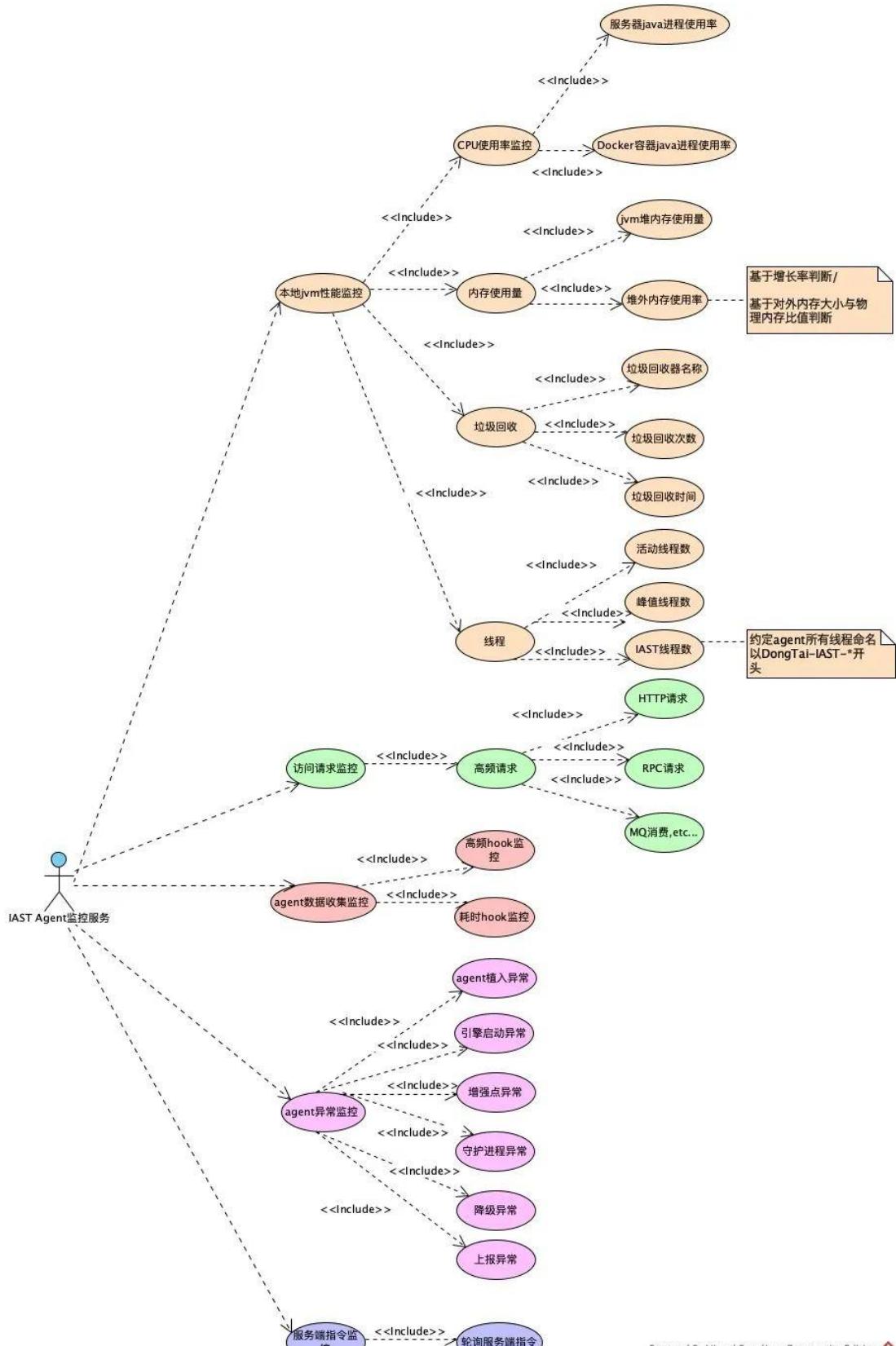
No 3 联合共建全过程

上面提到了 58 最核心的需求，所以双方联合开发的内容主要是对洞态开源内容的稳定性、降级方案进行了设计和改造。

1、业务分析

双方根据 58 的应用场景，确定需求范围(同样用例分析也是)，58 希望系统提供什么样的服务，以及 58 需要为系统提供的服务，以便容易理解这些元素的用途，也便于 58 软件开发人员最终实现这些元素。

58 经过和洞态团队的深入沟通，花了 3 个月制作了容灾降级方案，这里只做简单的描述，下图是结合 58 业务场景设计图。



△agent 监控用例图

Powered By Visual Paradigm Community Edition

//监控上报

///
NOCITCE**1.本地JVM性能监控**

未触发熔断，随PerformanceMonitor执行周期(默认60s)进行上报；触发熔断，放入。

REPORT_THREAD线程池排队上报。

2.访问请求监控

触发被判断为高频请求限流时，放入

REPORT_THREAD线程池排队上报。

3.agent数据收集监控

hook点命中被判断为高频hook限流/超时hook错误率达到阈值时，放入

REPORT_THREAD线程池排队上报。

4.agent异常监控

异常触发时，放入

REPORT_THREAD线程池排队上报。

2、业务流程

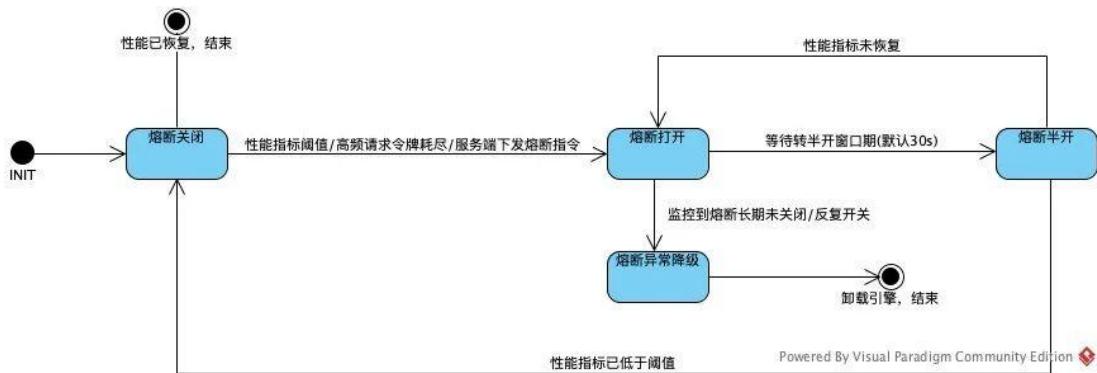
活动之间不仅有严格的先后顺序限定，而且活动的内容、方式、责任等也都必须有明确的安排和界定，以使不同活动在不同岗位角色之间进行转手交接成为可能。活动与活动之间在时间和空间上的转移可以有较大的跨度。

双方根据 58 业务实际需求，针对下方 6 种执行流程进行了重新设计。

- 洞态守护线程监控器执行流程部署配置
- 高频 hook 限流执行流程
- 高频请求限流执行流程
- 性能监控熔断执行流程
- 异常监控降级执行流程
- 服务端下发指令降级执行流程

3、业务实体状态图

状态机用于对模型元素的动态行为进行建模，更具体地说，就是对系统行为中受事件驱动的方面进行建模。状态机专门用于定义依赖于状态的行为（即根据模型元素所处的状态而有所变化的行为）。



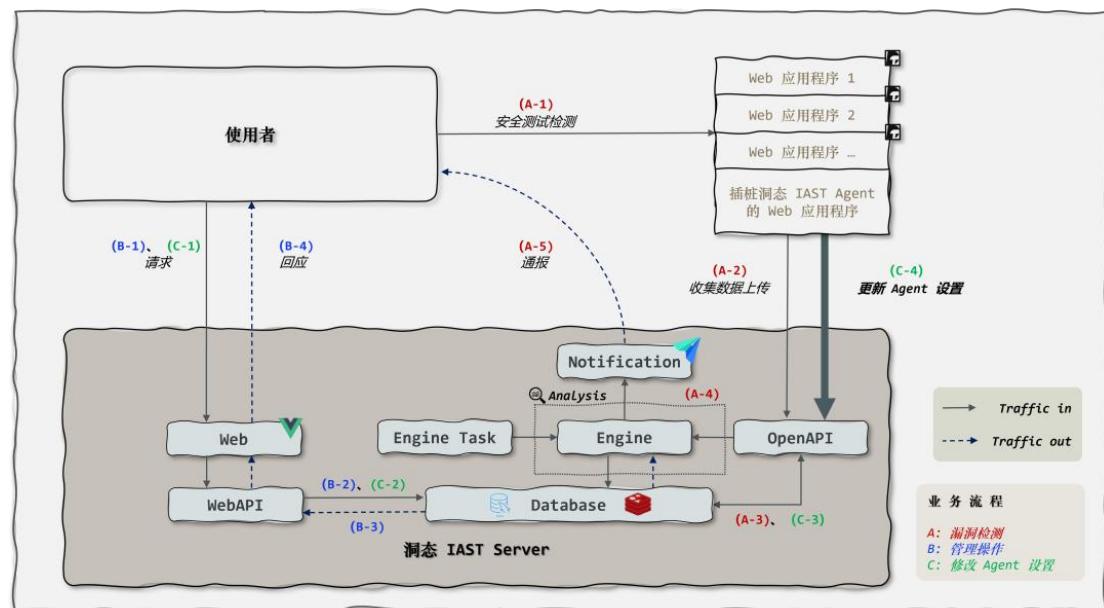
△性能熔断开关状态机

其行为不会随着元素状态发生变化，模型元素不需要用状态机来描述其行为（这些元素通常是主要负责管理数据的被动类）。

4、58 场景系统设计

1) 应用容器架构

应用容器架构定义了单个应用由哪些组件或服务构成，是关于应用容器内组件的逻辑分组。目的是将职责进行分解，分配给不同的组件，保证每个组件职责单一，进而控制和管理整体的复杂度。

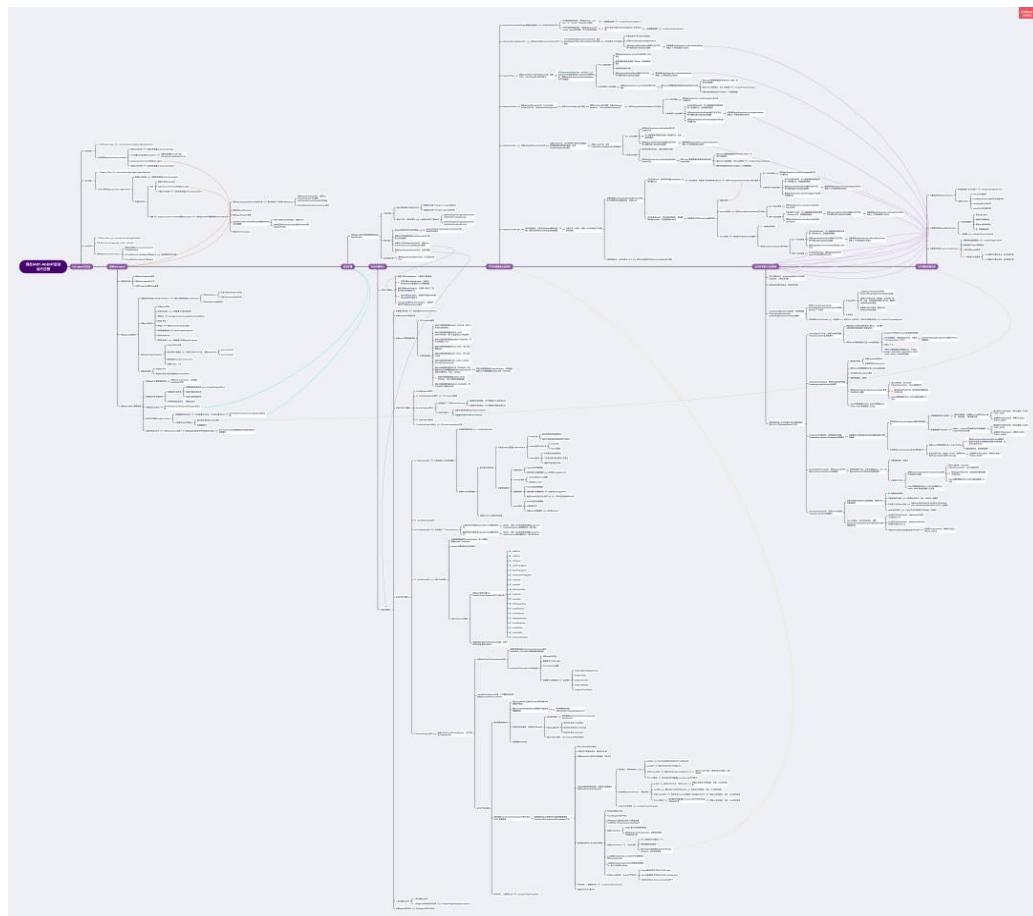


△洞态系统架构

2) 应用交互时序图

应用间交互时序图是一种流程建模，描述了应用之间交互的顺序，将交互行为建模为消息传递，通过描述消息是如何在应用间发送和接收，来动态展示应用

之间的交互。相对于其他 UML 图，时序图更强调交互的时间顺序，可以直观的描述交互的过程。



△洞态 JavaAgent(v1.3.0)启动流程图

除了针对 58 应用场景的容器架构和交互顺序流程建模设计，此外还对数据库、ES 等存储方式进行了设计。

58 和洞态经历了 4 个月研发，核对了 agent 端 5 千行代码，调整服务端 3 千多行代码，双方经过 50 次讨论，形成了 20 篇技术文档，最后完成符合 58 应用场景的设计。

当然，过程中也遇到了困难。测试的过程中，各种 agent 降级失败，服务端无法收到对应数据，经过双方联调测试后，最后成功部署，进而又做了大规模部署的高性能方案。

No 4 部署后测试效果

基于以上所考虑的问题点，在经过部署后，进行深入调研测试。

1) 测试环境

Agent 版本	1.7.0
靶场	DongTai-JavaAgent-Benchmark
软件配置	MySQL v5.7, Redis v6.2.5, Jmeter v5.4.1
操作系统	Ubuntu
CPU	16 核
内存	32 G

2) 测试场景

测试用例: DongTai-JavaAgent-Benchmark

此用例包含 Spring Boot, Spring MVC, JDBC Template 和 Redis Template。

测试接口会单次查询 MySQL 和多次查询并插入 Redis 数据。

并发量: 1, 25, 50, 100, 500, 1000, 5000

测试场景: 未插桩/插桩代理的应用运行

测试标准: 平均响应时间、响应中位数、吞吐量、CPU 占用、内存使用

3) 熔断降级配置

单请求 hook 限流: 1000 次

每秒处理请求数 (QPS) : 3000 次

系统 CPU 阈值: 90 %

JVM 内存阈值: 90 %

单请求 hook 限流: 限制单个请求内每秒 hook 数量

高频流量限流: 每秒限制处理请求数量 (并发量)

4) 测试结果

- 平均响应时间 (ms)

	插桩前	插桩后	开启熔断	Skywalking	Contrast
并发数1	0.69	2.23	2.02	0.78	1.05
并发数25	1.11	2.47	2.43	1.3	1.67
并发数50	1.25	1.62	1.22	1.46	1.76
并发数100	1.25	4.36	4.3	2.37	2.28
并发数500	20.55	26.59	20.48	26.12	23.97
并发数1000	43.92	59.07	44.01	58.32	56.37
并发数5000	109.55	142	110.53	137.92	136.63

- 响应中位数 (ms)

	插桩前	插桩后	开启熔断	Skywalking	Contrast
并发数1	1	2	2	1	1
并发数25	1	2	2	1	1
并发数50	1	4	3	1	1
并发数100	2	2	1	2	2
并发数500	21	26	21	25	22
并发数1000	42	70	48	62	60
并发数5000	240	284	214	284	265

- 每秒事务数(TPS)

	插桩前	插桩后	开启熔断	Skywalking	Contrast
并发数1	92.71	81.06	82.52	91.96	89.69
并发数25	2224.91	1984.65	1990.22	2188.47	2119.05
并发数50	4382.17	3434.96	3450.01	4300.24	4191.37
并发数100	8045.39	7845.18	8460.07	7916.05	7941.68
并发数500	15135.66	12547.89	15154.3	12842.05	13536.68
并发数1000	15966.55	12393.24	15915.75	12622.83	12928.48
并发数5000	15516.3	12154.15	15359.37	12603.18	12600.68

- CPU 使用率 (%)

	插桩前	插桩后	开启熔断	Skywalking	Contrast
并发数1	0.9	1.92	1.8	1.32	1.36
并发数25	5.7	20.18	20.58	8.06	9.05
并发数50	9.92	51.2	46.32	13.72	17.15
并发数100	17.65	32.8	19.8	23.8	35.58
并发数500	38.3	53.33	37.95	39.86	63
并发数1000	34.7	51.6	38.17	39.66	57.68
并发数5000	36.53	51.43	41.4	40.9	59.04

- 内存使用率 (%)

	插桩前	插桩后	开启熔断	Skywalking	Contrast
并发数1	0.044	0.048	0.048	0.047	0.168
并发数25	0.045	0.052	0.051	0.048	0.185
并发数50	0.046	0.053	0.053	0.049	0.17
并发数100	0.047	0.052	0.051	0.05	0.175
并发数500	0.081	0.163	0.074	0.081	0.263
并发数1000	0.073	0.115	0.077	0.08	0.261
并发数5000	0.092	0.144	0.092	0.1	0.175

No 5 58 生产环境效果

在实际生产环境部署上，经过测试总结出洞态在 58 应用场景下的优势主要有 3 点，很好的帮助 58 解决业务的安全问题。

1. 洞态轻代理、重服务器的架构

代理只负责采集和发送数据，服务器则负责分析与识别漏洞，每当并发量增高时，由于洞态优秀的架构设计和可预先添加熔断策略与阈值，即保证了 Agent 的稳定运行又不会影响到测试服务的正常运行。

2. 洞态熔断降级功能避免对业务产生影响

在测试环境中，依然会存在对应用进行压力测试的情景，如果在压测时应用依然安装了洞态的 agent，会导致压测结果严重失真。使用“熔断降级”功能，设置“高频流量限流”可以避免压力测试时 agent 对业务产生影响。

3. 洞态熔断降级功能可针对接口进行限流控制

高频访问数据库的接口在安装了 agent 后导致响应时间过长，是因为对 hook 点高频访问造成的。使用“熔断降级”功能，设置“单请求 hook 限流”可以针对接口进行限流控制。

洞态已经成为 58 集团众多的业务场景下研发和测试阶段不可缺少的检测工具，未来洞态将提供开发标准化，灵活的服务端数据接口和跨应用问题的解决方案。

自如

——一直在寻求能实现全链路、跨服务、跨语言的安全检测工具

本篇文章于 2022-5-27 发表在火线安全平台公众号

01 背景介绍

自如是一家提供高品质居住产品与生活服务的科技公司。租客可以享受在线看房、WiFi 覆盖、双周保洁等服务；业主可享受更增值、出租快、租务省的资产管理服务。

自如不管是在服务上，还是在产品安全上都希望给用户最大的保障，自如信息安全部从公司安全架构层面逐步实践“安全左移”，在云原生应用安全建设中，从单点到系统安全推进，构建了一个贯穿了从研发到运维过程的安全模型。

自如产品安全架构采用的是 Spring Cloud、Dubbo 等微服务分布式架构，整个公司的微服务的数量大概在 2000 个。由于这种微服务架构的特殊性，导致应用安全漏洞检测工作量加大，检测难度升级。

所以，自如安全部门一直寻求一种项目覆盖度全、误报率较低、不产生脏数据，且能实现全链路、跨服务、跨语言的安全检测工具。

02 自如为什么选择洞态？

IAST 是最近几年，在安全领域比较新型的技术和产品，自如安全也一直在跟踪研究，但是其他家的产品在跨服务之间检测有一定的缺失。

自如最先了解到的是火线的众测服务，后来通过一些安全技术公众号，看到洞态和一些互联网公司的落地合作介绍案例，并开始详细调研洞态。跟洞态技术人员深入了解和探讨后，觉得洞态的技术非常先进，主要功能有高可用，全链路，集成等，这非常符合自如目前的需求。

由于自如采用的微服务分布式架构，所以传统 IAST 在检测过程中无法跨服务检测，正好洞态的全链路功能可以很好的解决跨服务、跨语言检测难题，另外

有强大的开源社区支持也能很好的满足自如对 IAST 产品快速迭代，快速响应的需求，所以最终选择了洞态。

03 洞态在自如落地实践

洞态是如何在自如落地实践的呢？我们对自如的信息安全负责人进行了采访。

1、洞态帮助自如解决了哪些痛点？

洞态团队人员协助我们进行了分布式部署，目前洞态已经嵌入了我们内部的项目部署发布平台，在 QA 环境应用服务挂载集成洞态 agent，实现全链路、跨服务、跨语言的安全检测。较好的解决了自如目前安全测试资源紧张，项目覆盖度不完全，检测工具误报率较高和脏数据的问题。

2、部署洞态 agent 时，自如开发人员是否难以接受？又是怎么克服的？

我们的开发和测试人员都还是比较配合的，我们主要是集成在了部署发布平台里，进行自动化的 agent 挂载，大大减少了落地运营的成本，也非常感谢我们基础架构同事的支持和配合。

3、对于洞态的使用和推进，目前还存在哪些难点，是否有需要洞态团队协助的地方？

在现阶段的实际使用过程中，我们发现在实际的漏洞闭环运营上还有一定的缺失，例如版本的对应、漏洞联动推送及管理等。希望洞态团队能协助我们开发更适合自如业务的配置方案与良好的熔断降级效果。

4、自如对于洞态的下一步推动计划是什么？

目前我们已经引入了洞态优秀的全链路追踪功能，帮助我们在全链路，多语言，跨服务安全检测的同时，解决了很多安全问题，也很好的提高了工作效率，后续我们也会继续推进更多的项目和业务线接入洞态，这不管是对研发业务方，还是安全团队，洞态都起到了很大的帮助。

5、根据自如实际应用场景，最期待洞态未来会研发的新功能是什么？

我认为洞态的技术在全球都是比较先进的，在云原生应用场景下，它能够帮助很多企业发现并解决问题，期待把洞态的熔断降级，版本管理等功能集成到我们的流程中，帮助自如安全团队更好的实现推进落地，运营和高效产出。

04 落地实践测试

为了检验洞态的漏洞检测效果和漏洞检测能力，自如使用了多个测试用例，通过持续的功能测试和性能测试，发现洞态能覆盖常见高危漏洞，初步符合自如的预期。

这里为了保护自如业务的安全性，本文只展示其中部分测试过程和结果。

1. 测试环境

Agent 版本：

DongTai-java-agent v1.5

IAST Sever 端硬件配置：

4 Intel Core, 8 GB Memory

测试项目搭建平台\Agent 部署平台：

Windows11、centos7

2. 测试用例

文件名	适用 服务器	说明
vulns.war	Java	主要测试用例，包含十几种高危漏洞
S2-016.war	Java	Struts S2-016 漏洞
fastjson.war	Java	FastJson REC 漏洞
CVE-2019-10173.war	Java	xstream 反序列化漏洞
CVE-2019-12384.war	Java	jackson-databind 反序列化漏洞
Webgoat7	Java	web 漏洞实验应用平台（主要用于复测）
效能部门用例	Java	后端服务，提供 api 接口，查看组件扫描情况

策略管理：开启全策略检测

3. 测试过程

▪ 应用漏洞

在进行请求验证时，洞态报告一个高危漏洞：SSRF 服务端请求伪造。

http://127.0.0.1:7001(userInfo/queryUserRoleByUserName 的GET 出现服务器端请求伪造漏洞,位置:HEADER/PATH)

验证

导出

基本信息

服务器 IP: 127.0.0.1

客户端 IP: 127.0.0.1

中间件: JavaApplication

语言: JAVA

项目名称: SHAPSHOT

项目版本: V1.0

攻击参数: sec-fetch-mode,

攻击向量: 0

首次出现: 2021.12.30 14:11:02

出现次数: 1

危害等级: 高危

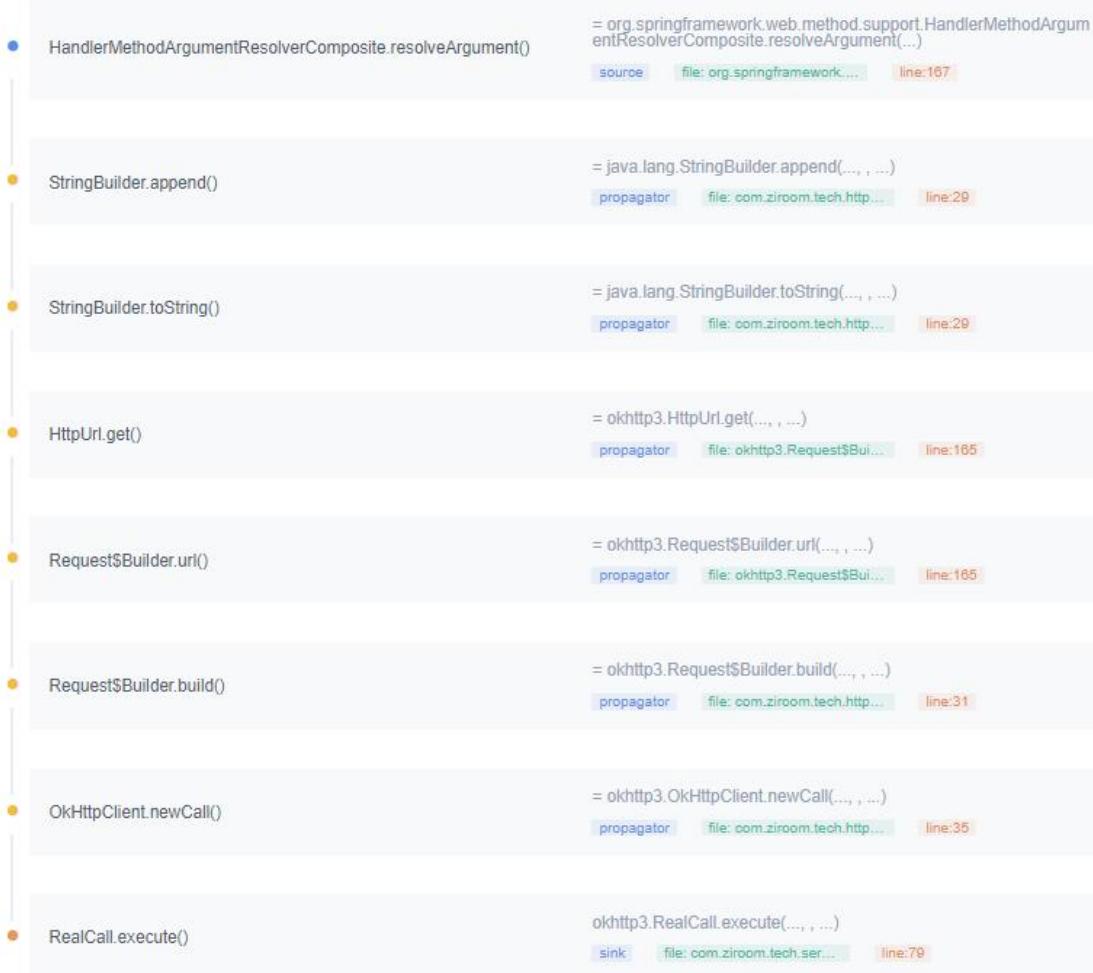
状态:

验证中

Agent: Windows 10-LAPTOP-TRUU0GC6-v1.1.4-4d4311bc954548b73e84dbd8b53ec7

污点流图

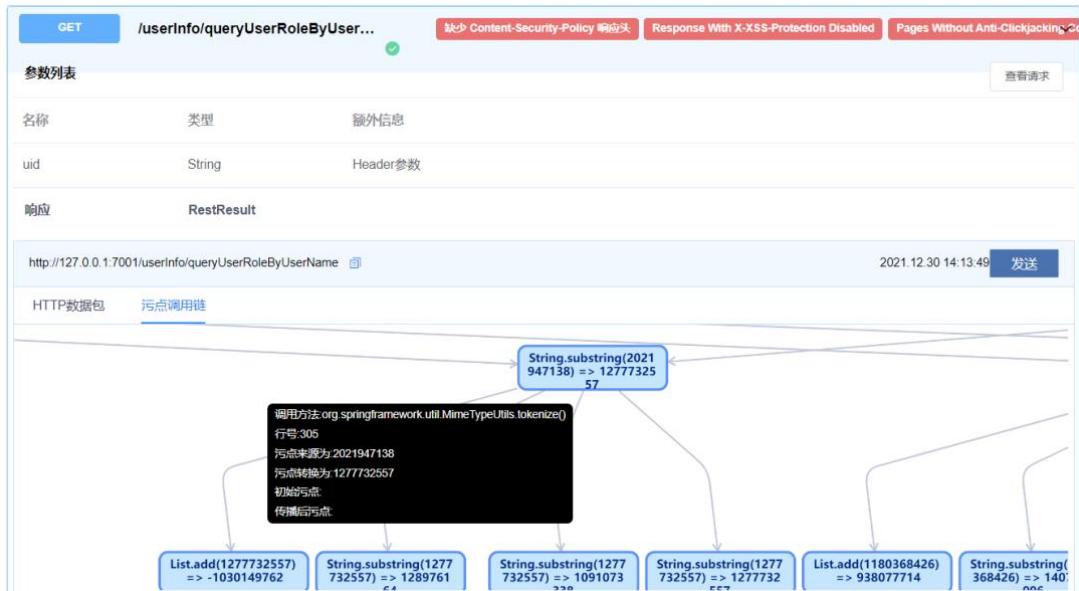
● 污点来源 ● 传播方法 ● 危险方法



△漏洞污点流图

▪ api 导航功能

展示来源样例为：SSRF 请求伪造漏洞的污点调用链和函数调用的详细流程导图。



在下图中可以看到，展示了已经经过测试的 API 接口。

方法	URL	状态	漏洞类型	覆盖率
POST	/r3AlarmNotification/queryR3AlarmNotificationBy...	缺少 Content-Security-Policy 响应头	Response With X-XSS-Protection Disabled	Pages Without Anti-Clickjacking Controls
参数列表				
名称	类型	额外信息		
pagination	Pagination	POST请求的body参数		
响应	RestResult			

请选择请求方法 请选择覆盖状态 请输入API地址进行搜索 覆盖率 80.56%

方法	URL	状态	漏洞类型	覆盖率
POST	/r3AlarmNotification/sendDailyReport...	缺少 Content-Security-Policy 响应头	Response With X-XSS-Protection Disabled	Pages Without Anti-Clickjacking Controls
DELETE	/task/delete/imei			
GET	/task/saveEhrAllJobGroupByCityCode			
POST	/test/kafkaMock			
GET	/order/getDeptTyp...	缺少 Content-Security-Policy 响应头	Response With X-XSS-Protection Disabled	Pages Without Anti-Clickjacking Controls
GET	/order/searchOrderListByOrder...	缺少 Content-Security-Policy 响应头	Response With X-XSS-Protection Disabled	Pages Without Anti-Clickjacking Controls

▪ 性能测试

洞态灰度测试，选取企业信息平台，共 6 个应用进行灰度测试，在 QA 环境下开启检测。

项目列表

项目名称	漏洞	存活Agent数量(个)	负责人	最近更新时间	管理
	●高危 3 ●提示 9576	1	admin	2022.05.10 20:41:48	 
	●高危 1 ●提示 7438	1	admin	2022.05.10 20:38:58	 
	●提示 1	1	admin	2022.05.10 17:48:32	 
		1	admin	2022.05.10 16:53:33	 
	●提示 1445	1	admin	2022.05.10 15:06:54	 
	●高危 1 ●提示 6969	0	admin	2022.05.10 12:05:47	 
	●提示 420	1	admin	2022.05.10 11:56:20	 
	●提示 40	0	admin	2022.05.09 11:05:28	 

- agent 对应用的性能损耗

选取安全中台应用，工具 jmeter，模拟在不同并发数量下进行性能测试，得到应用在安装 agent 前后的响应时间。

A	B	C	D
1 biz-security-service		插桩前 (ms 毫秒)	插桩后 (ms 毫秒)
2 单条请求	平均响应时间	104	123
	响应中位数	97	107
	99% Line	308	341
5 25条并发请求	平均响应时间	403	662
	响应中位数	383	637
	99% Line	896	1755
8 50条并发请求	平均响应时间	1238	1117
	响应中位数	1278	1203
	99% Line	2125	2150
11 100条并发请求	平均响应时间	2617	2295
	响应中位数	2549	2227
	99% Line	4815	4216

- agent 主动熔断降级

agent 检测到应用的内存或是 CPU 占用率超过预先设定的阈值，会进行自动降级熔断，停止工作。 agent 在自动熔断降级后，会在 40 秒（该值可自行设定）后进行自检，如果应用内存和 cpu 占用率数值降低到阈值以内，agent 会重新启动，继续检测；如果应用数值持续在高位运行，agnet 保持休眠，等待下次自检判断。

内存超过设定的阈值，agent 会自动熔断降级 agent.log 日志展示

```
1 Java
2 1 2022-04-25 18:16:41 [io.dongtai.iast.agent] [INFO] Engine performance fa
3 2 2022-04-25 18:16:41 [io.dongtai.iast.agent] [INFO] performance is over r
```

CPU 超过设定的阈值，agent 自动降级熔断 agent.log 日志展示

```
1 Java
2 1 2022-04-25 13:44:21 [io.dongtai.iast.agent] [INFO] Engine performance fa
3 2 2022-04-26 13:44:21 [io.dongtai.iast.agent] [INFO] The current CPU usage
```

05 结果总结

经过多轮的功能测试、性能测试，洞态在对常规漏洞的检测和组件风险检测上都有及时的漏洞告警，对小部分代码缺陷检测和不安全的请求参数配置检测可以做到针对性的覆盖。

- 1) 根据实际测试效果，洞态可以解决通过在应用测试阶段发现应用中存在的漏洞或缺陷，进行 实时检测，可覆盖 web 漏洞、三方组件漏洞、敏感信息泄露、API 漏洞等维度，可以更早的反馈研发修复，安全左移，降低安全成本，整体测试结果符合预期。
- 2) 现阶段 sever 端部署，在测试环境三台虚拟中部署，进行分布式部署，数据库、检测引擎、前端展示都是独立模块部署，并支持扩容机制。
- 3) 检测功能主要基于数据流的检测，后续再进行自定义的规则配置和产品的优化，能够有效的覆盖全部的信息检测。
- 4) SCA 能够有效的发现组件可能存在的风险，并对其进行识别和管理。
- 5) 主要覆盖在 QA 环境，检测目标是利用功能测试用例产生的流量进行安全检测分析，不会影响生产环境应用。
- 6) 根据性能测试结果，agent 对应用服务的性能影响可控，符合预期。agent 对应用的请求响应的性能损耗很小，损耗控制在 10% 左右，测试人员在进行功能测试时基本无感知。若质量同事对应用服务进行高并发等压测作业，洞态支持自



动熔断降级机制,同时支持手动停止/卸载 agent 服务,也可在 omega 上关闭 IAST 检测 agent 自动挂载,故原则上不会影响质量作业,并且最小成本入侵测试环境应用。

知乎

——初尝试

本篇文章于 2022-6-16 发表在火线安全平台公众号

01 关于知乎

知乎是中文互联网知名的可信赖问答社区，致力于构建一个任何人都可以便捷接入的知识分享网络，让人们便捷地与世界分享知识、经验和见解，发现更大的世界。

知乎以「让每个人高效获得可信赖的解答」为北极星，凭借认真、专业、友善的社区氛围和独特的产品机制，聚集了中国互联网上科技、商业、文化等领域里最具创造力的人群。

02 和洞态的缘

1、为什么在上线前做 IAST 安全检测？

近几年随着云计算、微服务和容器技术的快速普及，传统 SDLC 开发模式向 DevOps 敏捷开发和持续交付模式迁移。

在业务应用交付规模不断扩大、交付速度不断提高、开发运营场景一体化的大环境下，如何保障业务安全成了安全部门最大的难题，DevSecOps 为此应运而生，安全左移也是 DevSecOps 理念中重要的一环。

目前我们会在产品设计之初增加安全需求评审，在 CI/CD 流程中集成了自动化的黑白盒安全扫描，通过安全评审的输出结果决定是否需要人工介入进行渗透测试。

但是白盒扫描商业产品的昂贵的价格和相对较高的误报率和黑盒扫描覆盖不完全的情况，都需要人工的测试的介入，并不能完全高效的保证上线后的系统的绝对安全。

2、知乎为什么选择洞态？

交互式应用安全测试（也就是我们常说的 IAST）相当于白盒和黑盒的结合，是一种相互关联的运行时安全检测技术。它通过使用部署在 Web 应用程序上的

Agent 来监控运行时发送的流量并分析流量流以实时识别安全漏洞。

这种技术也被提出来有几年时间了，但市面上这类产品还是比较少，之前公司接入百度安全团队开源的 OpenRASP 的时候也测试过附带的 IAST 功能。

由于我们公司技术栈主要以 Go 为主，有少部分的 Java 项目，所以就只接入了 OpenRASP，可能由于 Java 项目少并且对外的业务更少的原因，OpenRASP 并没有拦截到什么漏洞。

后来和火线合作众测项目的时候了解到了火线的开源项目洞态 IAST，通过调研和本地测试，**这种被动式 IAST 通过插桩或者 Hook 的方式，利用污点传播分析，基于规则的漏洞监测技术，还是很适合部署在测试环境，通过 QA 来触发流量检测漏洞，并且部署方式可以透明的集成在 CI/CD 流程中作为 DevSecOps 工具链中的一环，做到安全左移在测试阶段及时发现安全问题。**

当时在测试洞态 IAST 的时候，Go Agent 还在开发中，只是针对 Java 项目进行了测试，也通过火线的师傅们的迭代开发，Java Agent 的性能也有了很大的提升，我们也通过了内部自动化部署发布流程的测试，已经开始推广接入洞态 IAST 了。

03 洞态在知乎的落地实践

洞态是如何在知乎落地实践的呢？

我们也对知乎的基础安全工程师陈浩若进行了采访。

1、洞态在知乎已经应用到哪个阶段？

目前我们已经开始推动两条以 Java 技术栈为基础的业务线在部署洞态 IAST，由于是推广前期并且有一些阻力，目前部署的 Agent 还不是很多。

但在使用过程中也发现了一些中高危的应用漏洞和组件漏洞，通过洞态 IAST 强大的污点传播分析也很快的定位到漏洞代码，及时发现问题修复问题。

接下来我们的计划是先完全覆盖目前正在推进接入的两条业务线，通过漏洞检测效果，优化调整规则，同时打通内部的安全平台系统如：漏洞管理平台，做到漏洞检测最大化并实现漏洞闭环。

04 对洞态社区的贡献 or 想法

1、基于公司业务实现需要，在开发过程中，对洞态做了哪些升级和改造？

前边也说到了由于公司技术栈以 Go 语言为主，目前洞态 IAST 也支持了 Go Agent，并且我们在适配内部常用的框架上也提交了 PR，但在我们内部测试过程中发现，由于 Go Agent 采用的是 Go Hook 的方式，在部署过程中有一些问题：例如如何区分不同的部署环境、还有一些性能和效率的问题，私下也和洞态开发师傅有一些沟通，也对洞态 IAST 对 Go 的支持充满了期许。

2、在部署洞态 IAST 这件事上，开发人员是否难以接受导致推动困难，您是怎么克服的？

推动和接入的问题，相对来说还是比较头疼的，技术的问题都是可以解决的，但在跨部门协作这块相对还是挺难的，尤其是对于安全部门来说，一方面是开发有开发的工作，配合安全修改部署脚本，发现了漏洞还要自己去修，都会多少有点排斥。

另一方面就是基于 IAST 的技术原理，无论是插桩还是 Hook，开发的同学多少会担心性能影响的问题，虽然只是在测试环境部署，但在早期的测试过程也会发现有一些类似内存溢出的问题，也辛苦洞态的师傅后来的优化和升级。

针对这些问题，我们也会进行多方面的测试，简化接入流程，同时也会在新员工技术培训上对安全意识和技术的宣贯。

05 对洞态的期待

1、根据公司实际应用场景，你最期待洞态 IAST 未来会开发的新功能是什么？

根据我们公司目前的应用场景，还是最期待洞态 IAST 能完善对 Go 的支持，还有就是如果能支持对访问控制类的漏洞检测就更好了。

某互联网金融科技公司

——使用高自由度的自定义规则能力提高漏洞检出效率

本篇文章于 2022-6-29 发表在火线安全平台公众号

洞态距正式开源已有 11 个月，用户已超过 200 家企业，覆盖互联网、汽车、金融等多个重要行业。

洞态是如何在互联网金融科技企业落地实践的呢？我们本期采访了安全架构师 PK，听听他是怎么说的吧。

01 个人介绍

大家好，我是 pk，目前在一家互联网金融科技企业，负责公司业务安全和数据安全。

02 和洞态的缘

1、上线前检测的优势，目前正在使用哪些安全检测工具，以及优劣势？

应用安全作为业务安全的核心，是我们目前一项重点工作。

在上线前设立安全卡点可以“短平快”地发现大部分安全风险，也是安全检测的最佳落地实践。

我司已经具备相对完善的 DevOps 流水线，也采用了传统安全检测手段，比如编码阶段白盒（SAST）+测试阶段黑盒（DAST）。

但传统检测手段的通病一样无法避免，比如误报率过高、无法精准定位并复现等等，严重影响日常安全运营效率。

2、为什么选择洞态？有哪些独有的优势？

因为机缘巧合，在火线成立初期就第一时间接触到了洞态 IAST。

相较于传统的 SAST+DAST 安全检测，洞态 IAST 除了可以明显提高漏洞准确度，还具备可快速融入公司 DevOps 流水线的优势。

对于业务同学使用体验较好，不会产生明显割裂感，达成了“业务与安全并进”的安全目标。

03 洞态的落地实践

1、洞态在贵公司已经应用到哪个阶段？

在技术选型初期我们已经充分考虑到业务使用的便利性，所以我们已经将洞态 agent 集成到了 CI/CD 构建流程中，应用发布时会自动集成洞态 agent，基本实现了安全接入业务无感知，所以推广初期还算顺利。

目前仍处于推广期，已经覆盖大约 50% 业务。

2、在使用过程中，洞态 IAST 帮助我们及时检测到了多少开发漏洞？

我们的检测场景分为传统 web 漏洞 + 敏感数据检测 2 个方向。

从实际效果来看，2 个月的时间收获了 2000+ 敏感数据传输，60+ 高危 web 漏洞，200+ 高危开源组件漏洞。

（此处与公司业务形态强相关，互金行业自带较强的个人信息属性）

3、洞态 IAST 的哪个功能实用性最高，与公司的实际业务场景更匹配

洞态 IAST 高自由度的自定义规则十分强大，从污点源函数、污点传播函数，到过滤函数、危险函数，甚至 header 白名单，能自由组合以满足我司“千奇百怪”的应用场景。

04 对洞态社区的贡献 or 想法

1、基于公司业务实现需要，在开发过程中，对洞态做了哪些升级和改造？

在使用过程中，也遇到过一些小问题，如发现 response 长度参数 bug、结果数据无法导出等。

洞态 IAST 拥有良好的开源社区生态，bug 和需求只要合理都会得到反馈，帮忙解决了很多问题。

个人自身也会力所能及地解答社区中的使用问题，和大家共同探讨学习成长很快。

05 对洞态的期待

1、根据公司实际应用场景，你最期待洞态未来会开发的新功能是什么？

希望洞态 IAST 未来能在 web 平台的管理功能上继续优化，如多维度统计分析、URL 白名单等。



对于大多数甲方安全团队来说，安全运营效率提升需要长期的统计分析数据支持，如果能做到可以直接用平台数据做安全考核指标那就更好啦！

聚水潭

——如何使得安全工具的误报率为几乎为 0

本篇文章于 2022-7-20 发表在火线安全平台公众号

亮点锦集

我个人在实际使用过程中体验到，对于像命令执行和 sql 注入这类漏洞，洞态能够做到百分百的检测。

——Spenser

对于一个开源项目而言，更多人参与就意味着可以完成更多的内容或者是发现更多的问题。我正是因为看中了洞态的开源生态，所以说我们才选择使用洞态。

——Spenser

滴水穿石，聚水成潭。聚水潭成立于 2014 年，创建之初，以电商 SaaS ERP 切入市场，凭借出色的产品和服务，快速获得市场的肯定。随着客户需求的不断变化，如今聚水潭已经发展成为以 SaaS ERP 为核心，集多种商家服务为一体的知名 SaaS 协同平台。

聚水潭不仅是洞态的早期用户，其相关负责人 Spenser 也是洞态开源社区的 Contributor，为洞态项目提交不少 issue 和策略。本篇文章记录了对聚水潭 Spenser 师傅所做的采访。

Q1

洞态小助手：请师傅先做一下自我介绍，包括公司职位，还有负责的具体事务。

Spenser:我是 Saas ERP 公司——聚水潭的高级安全工程师 Spenser，负责公司内部安全项目的研发以及安全架构的设计。

Q2

洞态小助手：好的~我们知道 IAST 产品一般是在产品上线前使用的，那您为什么选择在上线前做安全检测呢？

Spenser: 因为我们需要确保项目在上线的时候不会出现任何的安全隐患。洞态 IAST 通过内存 hook 的方式，可以帮我们非常准确地找出系统中可能存在的一些问题或者安全隐患。我个人在实际使用过程中体验到，对于像命令执行和 sql 注入这类漏洞，洞态能够做到百分百的检测。

Q3

洞态小助手: 上线前的安全检测工具有很多，那您为什么会选择了洞态 IAST 呢？

Spenser: 因为首先洞态是完全开源的 IAST 产品，这在中国甚至说是在全球都是没有先例的。洞态将整个生态和技术推到了大众面前，大家都可以参与进来。对于一个开源项目而言，更多人参与就意味着可以完成更多的内容或者是发现更多的问题。我正是因为看中了洞态的这样一个生态，所以说我们才选择使用洞态。

Q4

洞态小助手: 那您最一开始是从哪个渠道了解到洞态的呀？

Spenser: 我的同事野烟，在某一天加班的时候告诉我说，他发现了一个 IAST 产品，他自己测试下来效果是比较好，我是从那个时候开始了解的。

Q5

洞态小助手: 我知道洞态在聚水潭已经上了一部分业务线了。那目前为止有多少条业务线部署了洞态 IAST 呢？

Spenser: 大致上说，我们现在有三到四条业务线部署了洞态 IAST。

Q6

洞态小助手: 那在使用的这段时间里，洞态帮助咱们检测出了多少的开发漏洞，有具体的数字吗？

Spenser: 漏洞数量大概是在 50 个左右，中危的漏洞偏多。我们之前在测试的情况下，故意创造了一些高危漏洞，也是可以非常准确地识别。

Q7

洞态小助手: 嗯呢，那既然说到这儿，洞态误报的情况怎么样啊？

Spenser:除了之前关于敏感信息检测这一块有误报，其它基本没有。不过这块儿经过优化之后误报也几乎消失了。

洞态小助手: 您是通过什么样的方式来优化的呢？

Spenser:这个是我向洞态提了一条正则表达式，这个正则表达式是根据我的个人经验进行设计的。属于双方一起完成的，我贡献表达式，洞态团队把这条表达式设置成了敏感信息检测的默认表达式。

对于敏感数据检测的误报，我们会自己优化正则。漏洞检测，我们会根据自身的安全标准选择性开启规则，同时我们会编写一些 demo 来验证规则的漏检率和误报率，规则本身存在问题的会提 issue。同时我们也通过自研的 DongTai-SDK，对扫描结果进行验证，过滤掉一些无意义漏洞。

洞态小助手: 您也给我们提了策略，然后我们做了优化。

Spenser: 对。

Q8

洞态小助手: 公司对于洞态 IAST 的下一步推动计划是什么？预计什么时候在公司内部实现大面积推广呢？

Spenser:关于这一块的话，我们的主营业务线是由 C# 进行开发的，我们后期有计划参与到洞态 C# Agent 的开发中，这是为社区做贡献，也是为我们主要业务线能够使用上洞态做出一个铺垫嘛。

Q9

洞态小助手: 接下来这个问题其实和我刚才问的有点相似了，您这边基于公司的业务实现需要，在内部开发的过程中对开源版本的洞态做了哪些升级和改造？有没有和洞态开源社区进行共创共建？啊，这个这肯定是有。

Spenser: 关于我在开源社区的共创共建方面做的事情，一方面是我做了洞态 IAST SDK 的开发，Python 的 SDK 开发，通过 SDK 就不需要考虑底层的一些接口的调度，或者是一些错误的处理，可以更关注于用 IAST 进行业务的开发。

关于接口这一块，我提出了很多的 issue，然后洞态这边对于 issue 的处理速度也是非常得快。

Q10

洞态小助手: 谢谢师父！在公司部署和推广 IAST 这件事情上，您那边开发人员是否较难接受，导致推动存在困难，然后您是怎么克服的？

Spenser: 我们会先在安全部门自己的项目中上线部署，消除掉一些坑后，有些存在阻力的业务线，我们会找他们拉一份代码，然后自己部署一套完全隔离的测试环境，测试没有问题后，让他们部署到正在使用的测试环境。

之前我们在某一条业务线上进行推动的时候，由于 IAST 对于健康检测这块的识别有一定的不足，导致我们的测试环境使用了洞态一段时间之后变得非常的卡顿，是因为我们的 redis 库被写满了，健康检测的频率是非常高的，会产生大量的扫描任务，这些扫描任务都会被写进 redis，量比较大。消费端 worker 只有一个，无法及时地消费，就导致 redis 爆满之后测试环境业务比较卡顿。解决的方案是洞态开源社区的一位同学，提供了对于健康检测加白的功能，于是我们就解决这个问题。

洞态小助手: 那这个开源社区真的还是挺有帮助的。

Q11

洞态小助手: 对于聚水潭来说，洞态还需要进一步的推广使用，您预计还存在哪些难点，是否有需要洞态团队协助的地方？

Spenser: 我们的主营业务线用的是 C#，C#-Agent 开发完成之后，可能会存在一些漏洞，就需要洞态团队帮忙一起联动测试一下。另外，在开发的过程中可能会有涉及到协议的问题，就是上报的一些协议的问题，那也需要跟洞态团队进行交流。

Q12

洞态小助手: 好的。那您在使用的过程中有没有一些有趣的事情和我们分享一下？

Spenser:有趣的事情的话，我这个人有个习惯，就是说任何东西拿过来之后我会先对它进行验证，我要看它靠可不可靠嘛，毕竟东西大家都没有用过。公司的安全系统是用 Python 开发的。我会在系统中故意制造一些漏洞，然后看看洞态能不能识别。第一次没有识别出来，后来我同事去跟洞态团队的人进行了反馈。洞态团队经过大概一两天的时间，很快修复完了，修好之后还过来催着我的同事说：“你赶紧测一下，我现在在开车，我到家之后，就帮你一起看，看看效果怎么样，如果还是有问题的话，我当场改”。我觉得这个精神是非常好的。

洞态小助手: 哈哈，洞态技术人员对技术的热情程度是跟你一样的。

Q13

洞态小助手: 最后一部分的问题是关于洞态未来展望的，您希望洞态未来会开发的功能是什么？

Spenser:我希望洞态未来可以有一些攻击阻断的功能吧。我希望它可以有两个模块，一个是检测的，检测的话会消耗大量的性能，是在线上测试环境。拦截的话，像那些常见的漏洞，比如命令注入、sql 注入，使用带有拦截功能 agent 的时候，它可以在线上非常稳定的运行，但是同时它也可以有效地告警或者阻断。因为从技术层面上来说，sql 注入本质上是对数据库驱动的拦截。那如果可以在数据库驱动层面进行 hook 识别的话，那么我觉得所有的 sql 注入都可以防掉，那同时命令执行也是。对命令执行的检测，洞态其实已经实现。对于操作系统的环境不管是 Windows 还是 Linux，任何环境下去执行命令的时候，一定会调到系统的某一个函数。那么只要对这个函数进行拦截，就可以识别出是否是攻击，就可以把命令攻击拦截掉。

还有一方面，我觉得对于一些新型的攻击手法，或者是一些情报，也可以进行一个识别，能够更加智能化。像杀毒软件的话有智能查杀引擎，其实我也希望洞态可以有这种智能化发现漏洞的功能，通过人工智能或者深度学习这种技术能够识别出漏洞，甚至可以通过这种方式识别出业务逻辑层面的漏洞。

比如某平台是需要消耗积分进行查询业务，之前我挖到过这么一个漏洞，通过对查询语句的一些处理，可以使得用户不用消耗积分去查询。像这种其实就属

于业务型漏洞。可能一开始没法识别，但是通过训练，可能是我们自己发现漏洞，然后把它录制进去。在有了这样的数据源的情况下，进行训练之后，能够识别出业务型的漏洞。

洞态小助手：嗯。您提的这点可能需要和用户一起来开发，这种类型漏洞的检测，如果单靠我们来做的话，可能做得不会很到位。

Spenser:其实是你们提供了引擎，然后让用户提供数据源，数据源是在用户本地，那么你们的神经网络或者深度学习的模型可以通过对于这些数据源的训练，去识别出这套系统里面的业务型漏洞。每个系统逻辑都会不一样，但是肯定会有某种通用的写法，有大量的原始数据之后其实就可以训练出一种能够用于识别业务漏洞的神经网络或者是智能化引擎。

对于师傅的建议和期望，我们会认真考虑。关于以上对话，屏幕前的你是否有想进一步了解的内容（说不定我们会出个续集哈）？或者你在使用洞态的过程中，有哪些问题或者期望？都可以给我们留言讨论喔。

了解 IAST:

洞态 IAST 采用被动插桩的模式，可兼容所有场景。具有无脏数据、低误报率、高检出率的特点。更有高自由度的自定义规则能力，方便企业根据自身情况做规则调整，经过长期运营，检出效果会不断趋近最优值。

洞态 IAST 产品部分能力介绍如下：

产品功能	功能描述	开源版	商业版
Web应用漏洞检测	支持Owasp Top 10、CWE/SANS Top 25、PCI DSS 等类型的漏洞检测	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
多语言检测支持	支持Java、Python、PHP、Golang等语言的Web应用漏洞检测	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
漏洞详情与修复建议	支持详细的漏洞HTTP数据流及代码调用数据流，漏洞提供漏洞修复建议及代码示例	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
漏洞有效性验证	支持全局漏洞主动验证及漏洞验证功能，确保漏洞有效性和修复后漏洞验证	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
公开组件漏洞库	原始英文字段的NVD公开漏洞库数据	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
运行时组件识别及安全版本推荐	运行时依赖组件识别，确保真实调用版本；漏洞组件支持最小修复版本推荐	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
黑盒对接能力	支持对接黑盒，进行交叉验证工作	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
全链路漏洞检测	支持Spring Cloud、Dubbo、gRPC等微服务框架，Java、Python、Golang、PHP等多全链路污点传播过程追踪及漏洞检测	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
商业级组件漏洞库	支持中文版本的商业组件漏洞库及内部0day漏洞库	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
开源许可证分析	支持依赖组件开源许可证商用合规分析	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
组件漏洞可利用性分析及修复优先级推荐	组件漏洞可利用性分析，帮助企业修复已经产生危害的漏洞组件	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
漏洞报告导出	支持PDF、WORD、XLS等格式的报告导出功能	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
多账号管理	支持多角色、多账号及自定义权限的账号管理功能	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
CI/CD系统集成	支持编码阶段、CI/CD阶段、缺陷管理阶段、漏洞处置阶段等全流程DevOps工具集成	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
远程技术支持服务	提供原厂的技术响应和支持服务	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
部署及培训服务（1次）	提供1次线下的产品部署及使用培训服务	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
自研框架适配服务（2个）	支持标准产品外的自研框架、二开框架的适配性服务	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Agent运营支持服务、规则调优服务	提供原厂的Agent运营支持，确保iast产品部署后的价值产出	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
检测能力	新漏洞商业版会提前更新，开源版漏洞检测能力滞后6个月	滞后	<input checked="" type="checkbox"/>

相比较开源版本，洞态 IAST 商业版具有微服务下全链路漏洞检测能力、商业级别的漏洞库、开箱即用的 CI/CD 集成能力，运行时三方依赖组件风险管理，以及一对一的技术支持服务等等。依靠于开源社区，洞态团队结合众多甲方用户的真实使用场景下带来的反馈，在产品上做了很多设计和突破，欢迎大家来尝鲜。

扫描下方二维码，填写申请表单，可免费申请洞态 IAST 商业版试用，前往火线安全官网www.huoxian.cn，获取洞态IAST白皮书等更多信息。



扫一扫

了解火线

火线安全是一家社区驱动的应用安全创新企业，通过自主研发的自动化测试工具，结合超过20000名的实名白帽安全专家，帮助企业解决应用安全的各类风险。火线安全主要运营的产品和服务有安全众测、洞态IAST等。公司已与自然资源部、中国银行、腾讯、字节跳动、美团、京东、快手等众多知名机构和企业建立深度合作关系，并先后获得了奇绩创坛、经纬中国、五源资本等知名投资机构能力加持。

“洞态”是全球首个开源 IAST 产品，专注于 DevSecOps，具备高检出率、低误报率、0 脏数据的特点，帮助企业发现并解决应用上线前的安全风险。

“火线安全平台”是全球首个社区原生的安全众测平台，注册有超过20000名白帽安全专家，为企业提供可信的安全众测服务。

了解产品，长期关注安全行业动态，可关注以下公众号或添加以下工作人员进入应用安全攻防社群：



火线安全公众号



工作人员企微

火线安全北京总部

北京市海淀区上地东路35号颐泉汇C座3层

火线安全上海办事处

上海市黄浦区河南南路33号新上海城市广场19层

火线安全深圳办事处

深圳市南山区高新南一道008号创维大厦C座8层



火线安全平台



火线 Zone 安全社区