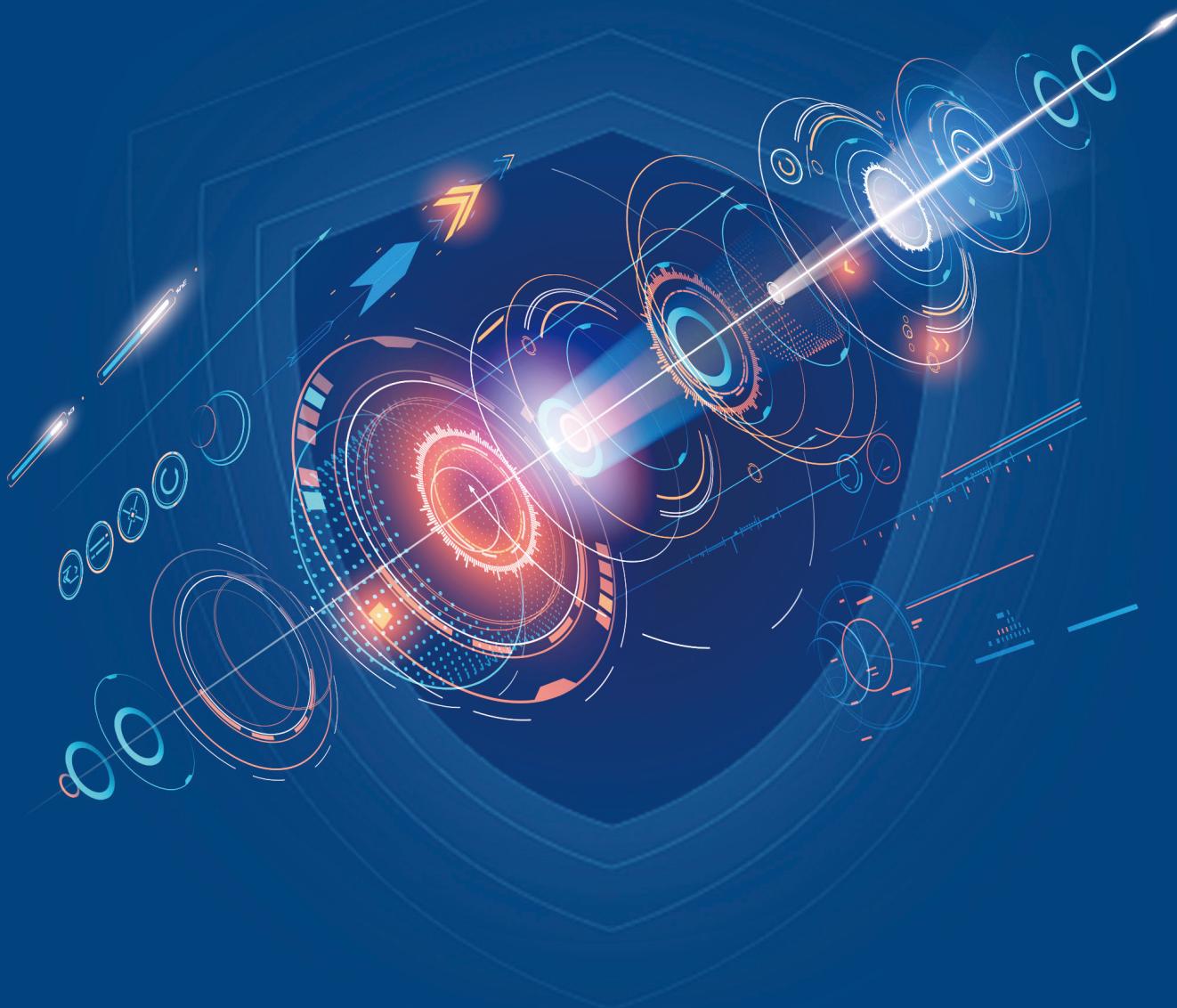


应用安全利器

IAST — 金融行业专刊



2024年7月

北京基调网络股份有限公司

关于 基调听云

北京基调网络股份有限公司（简称：基调听云）成立于 2007 年，是国内先行提出应用性能管理与用户体验优化解决方案的厂商，曾五次入选国际权威咨询机构 Gartner 的 APM 魔力象限。经过 17 年的持续探索，现在已经推出“听云、观云、安云”等覆盖应用性能监控、可观测性与应用安全等多个领域的产品与解决方案体系，致力于引领可观测性与应用安全领域发展。多年来，基调听云高度重视技术创新，针对企业数字化转型进程中面临的运维挑战，持续为企业提供一流的产品和服务。

在应用性能管理和监控领域，基调听云已经推出 APM、DEM 等多款自主研发的核心产品，拥有国内领先的数据采集与治理能力。为千行百业提供完整覆盖端到端的实时应用性能管理与智能分析平台，实现生产环境下的实时代码级应用性能监控、真实用户体验性能监控和智能分析，帮助企业提升系统性能表现，改善用户体验，加速业务创新。

在应用安全领域，基调听云推出的全新应用安全产品“安云”是一款应用安全态势管理（ASPM）产品，通过实时监测和分析应用程序的安全事件和异常行为，在运行态代码级别帮助企业识别、确定漏洞优先级并推动修复，保护敏感数据和业务流程的安全。

基调听云凭借卓越的技术实力和全场景的落地实践经验，已经成功帮助超过五千家知名企业构建完善的用户体验和应用性能监控平台。为政府、金融、运营商、云服务、互联网、交通运输、能源电力、科技制造、教育等各大行业客户的数字化转型和业务增长提供强大的支持，并赢得了广泛的信赖和认可。



关于 火线安全

北京安全共识科技有限公司，简称火线安全，是一家有攻防背景的应用安全公司，有全国首个“云安全知识库”、“云安全攻防期刊”。多次参加国家级、省级和行业攻防演练行动，成绩均在前 10%。主要为金融、互联网、汽车、O2O 平台及国央企提供众测、交互式应用安全测试产品 (IAST/ 灰盒)、SRC 渗透测试等多方面的信息安全服务。

参编人员

出品人

陈靖华、李甜甜、黎峰、
邬迪

编辑

李雅、庞伊良、卢中阳、
高鹏、郭梦飞、陈继成

设计

黄潇仪、李博惠

PREFACE

导语

根据 Gartner 的统计，应用安全领域近几年频繁出现大额的投资并购事件，市场规模已经超过了每年 100 亿美元。

如果把我们今天所有的应用加在一起，总量还不到 10 年后的 1%。随着数字化、IoT 和 AI 的发展，应用软件的范围会扩展到无数实体领域。特别是 AI 降低了开发门槛，让每个人都可以像程序员一样开发自己的应用，这会导致应用的爆发，安全问题将日益突出。

应用安全不仅关系个人隐私的泄露，还涉及到人身安全与整个世界的生产生活安全，例如曾经有黑客利用心脏起搏器实现暗杀；某些国家的基础设施遭受黑客攻击，水电、金融、交通、通信等领域都曾出现大面积的瘫痪。

随着数据安全及应用现代化发展，国内行业的应用安全建设进入深水区，最近几年国家监管和各行各业也开始愈加重视。我们看到，从 2021 年开始，国内陆续出台了很多相关的法律法规，比如《数据安全法》、《个人信息安全保护法》、《关键基础设施保护条例》等，里面的内容都和应用安全密不可分。

金融行业是国家代表性行业，是中国经济数字化转型中的排头兵，在应用开发、应用安全等领域切身践行着最新的应用实践。

2023 年 6 月，中国证券业协会（下称中证协）正式印发《证券公司网络和信息安全三年提升计划（2023–2025）》，提出合理加大科技资金投入。鼓励有条件的证券公司在 2023–2025 三个年度信息科技平均投入金额不少于上述三个年度平均净利润的 10% 或平均营业收入的 7%，并保持稳定的资金投入，相比征求意见稿中的 8% 与 6% 又有所提升。

在实际的业务运行过程中，越来越多的金融行业企业发现建设完整的 DevSecOps 流程是适应业务的敏捷快速迭代开发模式的最佳解决方案。IAST 作为一款适用于 DevSecOps 的优秀安全检测工具，目前已经在金融行业部分头部企业获得了初步应用，但因缺乏落地实践案例的指引，整体上在金融行业的全面推广还面临不小的难度。

安云 IAST 是基调听云推出的一款成熟的 IAST 产品，已在国内多个头部金融、证券行业客户成功落地。本专刊是基调听云联合证通股份有限公司、数禾科技、火线安全等单位共同编写的业界首个《IAST 金融行业专刊》，本期专刊将从金融行业技术发展背景、相关政策要求解读、IAST 技术分析、真实甲方落地实践以及 IAST 发展趋势探讨等方向，为金融行业系统性落地 IAST 提供参考。

期待通过本专刊，推动金融行业科技创新和信息安全工作交流、共享、提升，帮助金融行业企业构建更加安全的 DevSecOps 体系，更好地落地使用应用安全相关产品工具。



CONTENTS

目录

导语 PREFACE	04
目录 CONTENTS	06
背景分析 BACKGROUND ANALYSIS	07
证券期货业系统安全测试的政策背景研究分析	08
技术详解 TECHNICAL EXPLANATION	13
什么是 IAST	14
IAST 的分类与原理解析 ?	19
为什么 IAST 需要运营?	25
落地案例 CASE STUDY	28
安云 IAST 证券行业落地实践	29
IAST 在数禾的实践与探索	38
发展趋势 DEVELOPMENT TREND	47
IAST 与黑盒联动落地	48
IAST 未来的百种可能: IAST 未来主流技术路线浅析	69
结语 EPILOGUE	76
法律声明 DISCLAIMER	77

背景分析

BACKGROUND ANALYSIS

证券期货业系统安全测试的政策背景研究分析

证通股份有限公司 副总裁 黎峰

01 前言

我国资本市场的发展高度依赖信息技术，证券行业综合运用实时交易、网上交易等信息系统，保障了市场高效稳定运转，为资本市场的发展壮大和改革创新奠定了坚实基础。互联网技术的发展进一步拓展了行业机构获客边界，扩大了资本市场服务的覆盖面。随着新一轮科技革命和产业变更的加速演进，以人工智能、大数据、云计算、区块链为代表的新兴技术与传统金融深度融合，对业务的支撑和服务能力不断增强，进一步推动证券行业数字化转型，并助力高质量发展。业务与技术加速融合，网络和信息安全管理日趋复杂，信息系统建设任务明显增加，上线变更测试操作较为频繁，给行业网络和信息安全管理能力带来严峻挑战。

系统测试是对信息系统软硬件部署、配置等元素进行检验，以确保系统的功能、性能、安全等方面满足实际业务需求。系统测试是信息系统建设、运维的重要环节，是保证信息系统可靠性和稳定性，提高信息系统安全性、可维护性和用户体验的重要手段，也是降低信息系统故障率、提升行业机构生产效率和竞争力的必要措施。

02 案例

近年来，证券行业重要信息系统网络安全事件发生较为频繁，对资本市场的安全平稳运行造成较大冲击。据证监会官网显示，2022年5月，某券商因为系统设计与升级变更未经充分论证和测试，升级回退不完备等问题导致交易系统连续两次故障，被中国证监会采取出具警示函的监管措施；2022年9月，某券商因新建具有交易功能移动APP存在上线测试报告中缺少稳定性内容、安全测试报告不完整、压力测试报告缺少明确结论等问题，被中国证监会派出机构采取出具警示函的监管措施；2023年3月，某券商因存在信息系统升级论证测试不充分、未及时报告网络安全事件的问题，被中国证监会派出机构采取责令改正的监管措施；2023年7月，某券商因存在变更重要信息系统前未充分评估技术风险、未制定全面的测试方案等问题发生交易宕机网络安全事件，被中国证监会派出机构采取出具警示函的监管措施；2023年12月，某期货公司因重要信息系统上线验收前无安全测试相关报告，未开展重要信息系统压力测试，被中国证监会派出机构采取责令改正的监管措施；2024年4月，某期货公司次席交易系统上线前对系统功能测试不充分，未对信息技术系统服务操作及变更系统行为进行监控，未有效保护客户信息等，被中国证监会派出机构采取责令改正的监管措施；2024年5月，某期货公司存在修改防火墙安全策略后测试不充分即上线，被中国证监会派出机构采取出具警示函的监管措施。

由此可见，机构信息系统安全测试能力不足、自动化测试水平不高、测试不充分等问题已成为制约行业网络和信息安全发展的主要问题之一。

03 政策研究

国家对网络和信息安全的要求不断提高，陆续颁布了《网络安全法》《数据安全法》《个人信息保护法》《关键信息基础设施安全保护条例》等法律法规，明确政府、机构和个人等主体的网络和信息安全义务和责任，对网络运行安全、网络信息安全、应急管理、数据安全保护、个人信息保护、关键基础设施安全保护等作出了具体的规定。证券期货业高度重视网络和信息系统安全工作，从部门规章、规范性文件、行业自律规则到行业标准等多层级文件规定中对安全测试有着诸多具体、明确的要求，具体如下。

（一）部门规章及规范性文件

早在 2012 年中国证监会公布的《证券期货业信息安全保障管理办法》（证监会令第 82 号）中第二十二条明确规定要求“核心机构和经营机构开展信息系统新建、升级、变更、换代等建设项目，应当进行充分论证和测试。”

2018 年中国证监会公布的《证券基金经营机构信息技术管理办法》（证监会令第 152 号令，2021 年第 179 号令修订）第二十一条、第二十二条、第二十三条、第二十五条、第三十二条针对专用开发测试环境，重要信息系统上线或变更要求，重要信息系统定期压力测试和评估分析，业务测试审查、评估、留痕，测试岗位分离制衡等进行规范，明确“证券基金经营机构在生产环境开展重要信息系统技术或业务测试的，应对测试流程及结果进行审查”。

2021 年 6 月，中国证监会发布的《证券期货业网络安全事件报告与调查处理办法》（证监会公告〔2021〕12 号），在第二十八条中明确处罚依据，对研发、测试等系统管理过程中未严格执行行业相关法律法规和行业相关技术管理规定、技术规则、技术指引和技术标准，造成网络安全事件的，中国证监会及其派出机构依法采取监督管理措施或者实施行政处罚，并要求对相关责任人员进行内部责任追究。

2021 年中国证监会发布《证券期货业科技发展“十四五”规划》（证监办法〔2021〕70 号），在信息安全管理体系建设平台建设任务中，提出了“面向行业机构提供第三方软件安全准入测试服务，降低软件接入风险。”的工作任务。

2023年2月中国证监会公布《证券期货业网络和信息安全管理方法》（证监会令第218号，替换第82号令）中，进一步强化对重要信息系统变更和测试要求，在第十六条中明确要求“核心机构和经营机构在重要信息系统上线、变更前应当制定全面的测试方案，持续完善测试用例和测试数据，并保障测试的有效执行”，第二十一条明确要求“核心机构和经营机构应当每年至少开展一次重要信息系统压力测试；发现市场较大波动，重要信息系统的性能容量可能无法保障安全平稳运行的，应当及时对相关信息系统开展压力测试”，并明确“应当依照有关行业标准，根据系统技术特点和承载业务类型，制定压力测试方案，设定测试场景，从系统性能、网络负载、灾备建设等方面设置测试指标，有序组织测试工作，测试完成后形成压力测试报告存档备查，并保存五年以上。”

除此之外，中国证监会及其派出机构通过《机构监管情况通报》、《证券期货业网络和信息安全管理方法》等多种方式，专门通报相关信息系统安全事件案例，要求强化安全管理，完善系统测试工作，加强压力测试，定期开展系统健壮性评估，及时消除风险隐患，同时也要求证券期货基金经营机构加强信息系统建设的统筹规划，充分了解系统架构及内部运行机制，强化研发、测试、上线、升级变更及运维管理，完善应急预警、处置、报告机制，确保信息系统安全平稳运行。

（二）自律规则

2008年9月，中国证券业协会等印发了《证券期货经营机构信息技术治理工作指引（试行）》（中证协发[2009]113号），在第三十一条中明确要求“公司应为IT应用实现提供必需的财力和人力资源，并在制定工作计划时充分考虑软件开发、测试及部署实施所需要的时间周期。”

2023年6月，中国证券业协会、中国证券基金业协会、中国期货业协会分别印发了《证券公司网络和信息安全三年提升计划（2023—2025）》、《基金管理公司网络和信息安全三年提升计划（2023—2025）》、《期货公司网络和信息安全三年提升计划（2023—2025）》，阐明了未来三年全面提升证券、基金管理和期货公司网络和信息安全的指导思想、基本原则、总体目标、主要任务及实施路径。《提升计划》聚焦提升行业科技治理和信息系统架构掌控能力，聚焦网络和信息安全存在的基础性、深层次问题，从技术投入、系统架构、研发测试等进行深入调研，查找薄弱环节，明确提出系统研发测试管理提升方向和要求。《提升计划》鼓励行业机构增加信息技术投入，升级信息系统架构，健全信息安全防护体系，包括落实等保要求、提升安全攻击防控能力、加强态势感知能力、加强数据安全管理体系建设等措施。叠加金融行业信创推进，网络安全、信息安全能力将为金融机构业务平稳运行提供坚实底座。

以《证券公司网络和信息安全三年提升计划（2023—2025）》为例，中国证券业协会鼓励有条件的证券公司2023—2025年三个年度信息技术平均投入金额不少于上述三个年度平均净利润的10%或平均营业收入的7%，并保持稳定的资金投入。持续优化信息科技投入结构，加大研发类、网络和信息安全类以及信创建设等方面的投入，深化信息技术架构设计、系统测试、安全防护、数字化转型能力建设。

《提升计划》提出了要强化系统研发测试管理能力的工作任务，主要包括建立及完善需求设计及分析机制，持续提升代码开发效率及安全、制定并落实信息系统代码审计规范，全面加强信息系统测试质量管控等四方面具体要求，其中明确要求强化研发供应链安全管理，发布前必须通过安全测试；加强信息系统的功能测试和非功能测试与评估，推进测试左移、右移，提升自动化测试比例，全面提升测试效能与质量；严格控制系统变更实施，防控变更流程不严谨、测试不全面等风险；重要信息前应开展回归测试等，强化信息系统上线投产前的合规审核，加强安全防护检查校验，夯实网络和信息安全技术体系。

（三）行业标准

证券期货业发布多个推荐性行业标准，针对行业测试进行规范：

2010年发布的金融行业标准《证券期货业网络安全等级保护基本要求》（JR/T 0060—2010，2021年更新）、《证券期货业网络安全等级保护测评要求》（JR/T 0067—2010，2021年更新），规定了证券期货业网络安全等级保护的重要要求，以及保护对象的安全通用要求和安全扩展要求，其中对安全测试有大量细节指导。

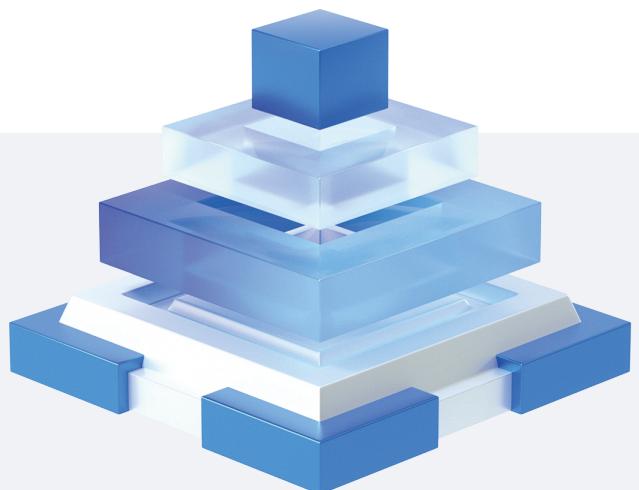
在2013年发布的金融行业标准《证券期货业信息系统运维管理规范（JR/T 0099—2012）》（证监会公告[2013]7号），对系统维护中的系统测试环境、资源配置、测试方案及测试、变更的全流程进行了细致阐述和规范。

在随后发布的金融行业标准《证券期货业信息系统审计规范》（JR/T 0112—2014）（证监会公告[2014]58号）、《证券期货业信息系统审计指南》（JR/T 0146—2016，系列标准）（证监会公告[2016]25号），其中对信息系统安全测试检查与审计要求进行规范。

在2019年发布的《证券期货业软件测试规范（JR/T 0175—2019）》（证监会公告[2019]20号，简称《测试规范》），对证券期货业信息系统建设过程中的总体要求、单元测试、集成测试、系统测试、系统集成测试、验收测试等测试活动的内容进行规范，进一步强化测试过程、结束的约束，提高行业规范化程度。

在 2020 年发布的《证券期货业软件测试指南 软件安全测试》(JR/T 0191-2020) (证监会公告〔2020〕40 号) 在《测试规范》中测试流程与内容的基础上，提供软件安全测试流程、技术和参考文档，从测试目的与流程、测试技术选择、测试方法等角度对如何进行软件安全测试给出指导意见，指出了在证券期货业行业针对每个具体系统需要完成的安全功能测试、代码安全测试、自动化漏洞扫描、渗透测试、模糊测试等测试方式，通过加强对软件产品的安全特性、潜在的漏洞与风险等内容的检测，指引机构系统软件的安全测试工作。

综上，证券期货业对信息系统安全测试有诸多明确规范要求，基于安全合规的前提下，行业机构各方正积极探索利用大模型等人工智能、大数据等先进技术，实施业技深度融合，推进系统安全测试的自动化、智能化建设，不断提升安全测试的质量与水平，有效降低业务风险。





技术详解

TECHNICAL EXPLANATION

什么是 IAST

基调听云专家顾问 郭梦飞；基调听云高级市场经理 庞伊良

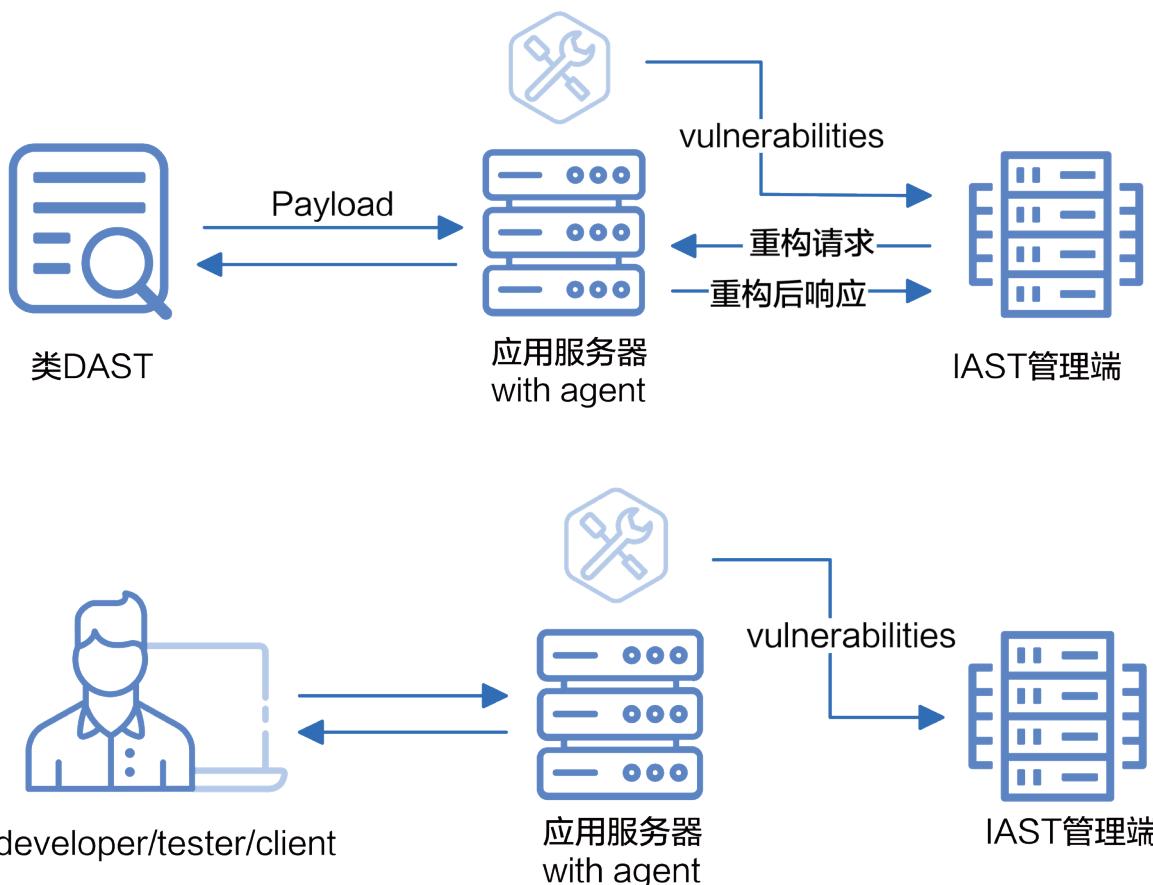
IAST 全称 Interactive Application Security Testing，即交互式应用程序安全测试，一般也叫灰盒，是一种在应用和 API 中自动化识别和诊断软件漏洞的应用程序安全测试技术。最早在 2012 年由 Gartner 首次提出，后来在国外得到广泛推广和应用。与传统的 SAST（静态应用程序安全测试）和 DAST（动态应用程序安全测试）方法相比，IAST 提供了更全面、准确和高效的安全测试解决方案。

传统的 SAST 工具主要关注静态代码分析，一般是通过检查源代码来发现潜在的漏洞和安全弱点。然而，由于无法准确模拟应用程序的实际运行环境和用户输入，因此 SAST 工具往往会产生大量误报。DAST 工具通过模拟实际应用程序的交互和攻击场景来测试应用程序的安全性，但尽管 DAST 工具可以提供更真实的测试结果以及 Payload，但由于无法准确确定漏洞的具体代码位置且漏报率较高，给开发人员的修复工作带来了挑战的同时也容易留下安全隐患。与此相比，IAST 将 SAST 和 DAST 的优点融合到一个解决方案中，通过 Agent 插桩技术（Instrument）收集、监控 Web 应用程序运行时的函数执行、数据传输，并与 IAST 服务端（Server）进行实时交互，高效、准确地识别安全缺陷及漏洞，可准确确定漏洞所在的代码文件、行数、函数及参数，能够准确地检测到漏洞的可利用性，并确定其在应用程序代码中的精确代码位置。

目前国内常见的 IAST 产品中，一般是通过代理模式、流量镜像模式、插桩模式等读取应用程序运行时的相关数据交互，其中插桩模式还可进一步分为主动插桩、被动插桩。

代理模式、流量镜像模式都是黑盒的一种变种，是在早期国内 IAST 技术不成熟的现实情况下，通过对黑盒的改造以充当 IAST 使用的产物。主动插桩模式在关键函数（危险方法）Hook 到流量后，会添加 Payload 进行扫描，这个过程仍然是类似黑盒的功能，主动对目标应用进行扫描，应用服务器的 IAST Agent 不会追踪整个污点数据流，只会收集存在危险方法调用的请求流量，然后发送验证数据包给 IAST 的 Server 端，Server 端会向应用服务器发送构造好的重放流量来验证风险是否存在。这种采集和重放流量的方式比爬虫、代理式黑盒是显著减少了的，但是其本质仍然是黑盒的技术和逻辑，因为只要存在流量重放，就一定会存在脏数据，当下的新场景也无法覆盖。

对 IAST 来说，被动插桩模式才是最重要、也是最符合 IAST 技术路线与原理定位的检测模式。最早推出 IAST 产品的美国厂商 Contrast，其 IAST 就只提供被动插桩一种方式。



被动插桩基于值匹配算法和污点跟踪算法进行漏洞检测，不会主动发送 Payload，不用采集和重放流量，对来自客户端的请求响应进行污点传播数据流监控，根据是否经过无害化处理来判断是否存在漏洞。只需要开启了业务测试，就会自动触发安全测试，通过测试流量就可以实时地进行漏洞检测，不会影响同时运行的其他测试活动，在这个过程中不会产生脏数据，并且可以解决比如微服务这样的新场景、新架构的检测问题。被动插桩 IAST 的技术原理，在提出之初主要是针对于 JAVA 而言，因为 JAVA 提供了原生的接口，它允许在程序运行起来以后，对编译后的代码进行操作或监听等操作。IAST 主要是对代码的调用链过程进行监听，监听的点主要分为三类：入口方法、传播方法、危险方法。

入口方法：主要是指接受用户传入数据的方法；

传播方法：对用户传入的参数进行处理的业务逻辑；

危险方法：通常指可能会造成危害的方法。

对被动式 IAST 来说，污点跟踪算法是其最核心的漏洞检测方式。上文提到，JAVA 应用提供了原生的接口，在 IAST 启动的时候，会通过 Agent 技术对上述三类方法进行监听。与零信任概念类似，IAST 不信任任何用户传入的值，所有由用户传入的值都会被打上污点标签。如果用户传入的数据携带着污点标签传入到了最终的危险方法，则会被判定为存在漏洞。如果研发人员在传播方法过程中，对传入的值进行了过滤或使用了安全方法，则会在经过这类方法的时候销毁污点标签，此时再传入到危险方法时就不再携带污点标签，也就不会判定为漏洞了。

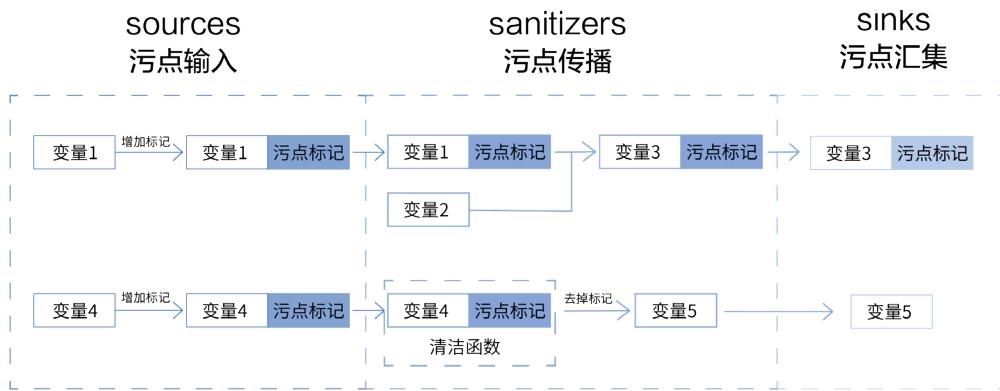
检测原理：

详细地来说，IAST 通过 JAVA 的 Instrument 技术将 Agent 运行在 JVM 中，在成功插桩以后 IAST 会对该应用程序进行 Hook。IAST 主要 Hook 的点分为三类：Source、Propagator、Sink。

其中 Source 主要是接受用户传入数据的点，IAST 将不会信任该 Source 的任何数据，因此 IAST 将用户传入的数据标记上污点 tag。

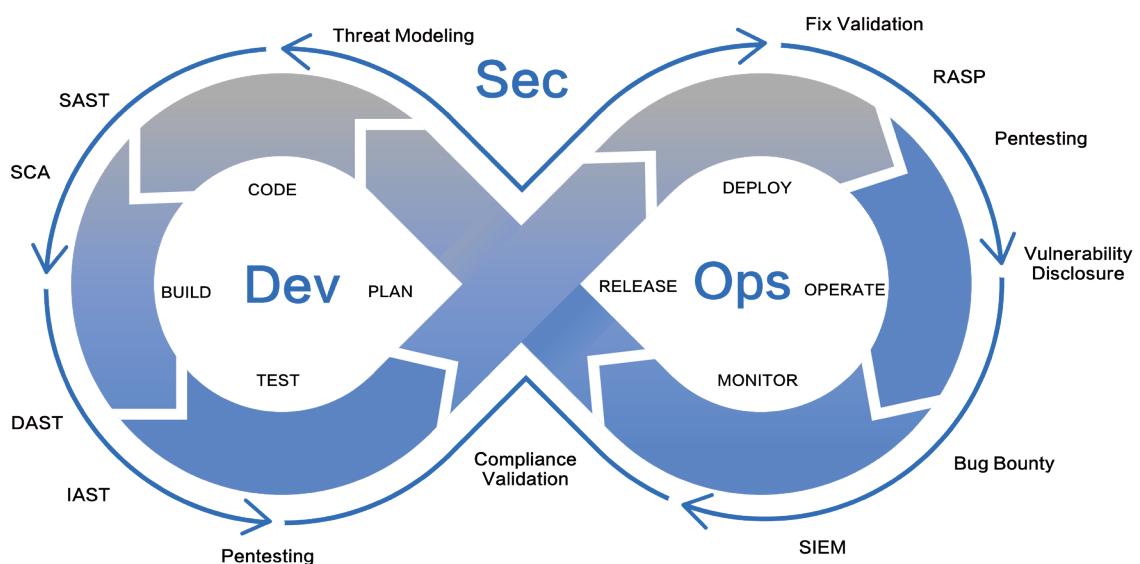
Propagator 顾名思义也就是传播方法，一般而言用户传进来的参数都会经过去程内部逻辑，比如字符串拼接等，这些就被称为 Propagator，不管该参数经过了多少 Propagator 点，污点 tag 始终都会携带。当然传播方法中会有一个特定的方法，也就是常说的清洁函数，当参数在传播过程中经过了安全方法或过滤方法（比如 XXE 的 setFeature、自研过滤方法等），会将污点 tag 进行清洗、摘除。

在经过上述的流转过程后，一部分数据就会进入 Sink 点，也称为危险方法执行处，如果用户传入的参数此时还携带着污点 tag，证明数据流转过程中没有经过任何安全方法和过滤方法，IAST 将标记为存在漏洞，并将该参数流转的调用链还原，展示相应的传播过程以及堆栈信息。如果污点 tag 此时已经不再存在，则不再判定为漏洞。



通过上述原理，可以得出结论：IAST 对于判定漏洞是否存在的条件要求比黑盒和白盒测试更为严格，因此其准确率更高。当发现漏洞，IAST 能够支持生成相应的报告，并提供有关漏洞的详细信息，包括漏洞的类型、代码具体位置、严重程度等。IAST 平台还能够提供漏洞修复建议，帮助开发团队更好地进行修复工作。同时，为了进一步提升修复效率，IAST 平台支持项目分组，将开发工作与安全工作协同。开发人员可以通过登录平台来查看自己所负责的项目概况。这在以前可能需要安全人员逐个分发安全报告，漏洞闭环周期慢且难管理。当下开发人员可以更加便捷地查看项目的安全状况，并采取相应的措施进行修复。这种集中管理的方式提高了团队之间的协作效率，使整个修复过程更加快速高效。

得益于基于污点标记跟踪算法的检测原理，IAST 能在用户手动或自动执行功能测试的同时触发流量，并实时检测安全漏洞，与开发人员原有的工作流程具有良好的集成性，可以与开发环境和持续集成 / 持续交付（CI/CD）流水线完美集成，是 DevSecOps 中的测试环节的不二之选。



IAST 在插桩时可以获取代码中的所有 API。在功能测试过程中，它会统计并记录测试中的 API 调用覆盖率。举个例子，一个系统总共有 100 个 API 接口，但 IAST 检测到测试的 API 覆盖率只有 95%。这可能是因为：测试人员没有覆盖到部分 API，安全人员可以进行复测；或 API 已经被废弃，但长期下来代码中没有删除，形成了常说的坏链和死链情况。这些未覆盖或已经废弃但没有删除的 API，由于长时间没有维护，在设计或实现中可能存在已知或未知的安全漏洞。一旦系统上线后，不法分子就有可能通过模糊测试等技术，发现这些遗留接口的存在。并且利用可能存在的安全问题进行非法入侵等恶意行为，导致隐私数据泄露等严重后果。通过梳理和报告 API 覆盖率和遗留接口的情况，IAST 可以提醒研发团队及时检查和修复，以避免因未知安全漏洞导致的风险。

微服务架构具有低耦合和强大的可扩展性等特点，随着信息化的发展，受到越来越多企业的青睐。然而，这些流行的架构也带来了安全方面的挑战。漏洞的产生不仅限于代码内部的调用。传统的 SAST 和 DAST 受限于检测方式，很难覆盖微服务架构中的所有情况。通过 TraceID 技术，IAST 能够跟踪微服务的上下游信息，弥补了流行的微服务架构的安全测试需求，更全面地检测和保护微服务架构的安全。

综上所述，IAST 作为一种主流的应用安全测试技术，它很好地结合和融合了 SAST 和 DAST 的优点，但这并不意味着 IAST 拥有替代 SAST 和 DAST 的能力。相反，IAST 与 SAST、DAST 作用在应用安全生命周期的不同阶段，实现了不同的安全能力，并且相互之间进行能力补充，共同组成了完备的应用安全测试体系。

IAST 采用被动插桩作为核心运行模式，基于污点跟踪算法进行检测。不仅准确率高，而且不会对正常业务产生影响，天然地融入了 DevSecOps 理念，实现了开发、安全、运维工作的深度集成。在安全测试准确率、报告质量以及与软件开发流程的集成性等多个维度都展现出了明显的优势。

IAST 的分类与原理解析

北京基调网络股份有限公司 安全研究员 高鹏

01 背景

在当今数字化浪潮的推动下，应用程序已经成为现代社会中不可或缺的一部分。无论是金融交易、医疗记录还是日常购物，我们都离不开各种各样的应用程序。然而，随着应用程序的普及，与之相关的安全风险也日益增加。在这个背景下，对于应用程序的安全性进行细致评估和测试显得尤为重要。交互式应用安全测试（IAST）作为一种高效且精确的安全评估方法，正在成为许多企业提升应用程序安全性的首选。

02 IAST 的分类

IAST 的分类主要分为主动式 IAST 与被动式 IAST，它们从实现原理，部署方式，检测能力上都有差别，下文会对这几点进行详细介绍。

（一）主动式 IAST



- 流程图解析 -

主动式的 IAST 顾名思义，相较于被动式的 IAST 会有流量和主动检测的能力，也分为 Agent 端和 Server 端，具体流程图如上。下文拿百度开源 OpenRasp 列举：<https://github.com/baidu-security/openrasp-iast>

实际上，IAST 的实质相当于将 DAST 与 RASP 合二为一。百度的这个项目在这方面做得十分出色，它利用 Python 来对 Agent 传递的流量进行处理，添加恶意载荷后进行重放。随后，通过 RASP 技术来判断是否到达最终的 Hook 点。

这一方法的优势在于能够通过精心构造的攻击载荷进行测试。这样一来，可以有效地模拟应用程序中的条件判断。举个例子，如果被动式的测试接收到一个 ID 参数，应用程序首先会在数据库中查询是否存在该值，然后再执行后续的流程。然而，被动式无法主动感知这一过程，这可能导致当后续触发了漏洞点时也会报告出来。与之相对，主动式方法通过重放并修改 ID 参数，可以明确地知道该值是否能够顺利通过后续流程，从而减少了误报的情况。

当然，这也引出了另一个问题，主动式可能会引入脏数据。虽然最终能够捕获漏洞触发点并进行 Hook，但不能保证在这一过程中是否存在一些安全的机制，将这些无效的测试数据存储了下来。

目标系统，也就是我们的应用服务（例如：Java 服务），需要在服务上安装 Javaagent，通过 -javaagent 在服务启动时一起启动安装 Agent。

也有另外一种方式，Java 服务在服务器上只是一个进程，或者容器，或者 POD，像百度的 OpenRasp 实现通过安装一个类似于HIDS的探针(二进制探针)部署在服务器上采集流量，再传输到 Server 端由服务器端进行扫描。

在探针启动成功后，会实时采集应用系统的流量，并发送到 IAST 服务端。

随后发包器会对探针传回来的请求进行解析，添加对应的 Payload，随后再次发送请求，探针采集再次采集数据，包括调用链数据，随后会判断返回的调用链数据中是否存在对应的 Payload，逻辑看下图。

openrasp_iast/plugin/scanner/sql_basic.py

```
03 class ScanPlugin(scan_plugin_base.ScanPluginBase):
04
05     plugin_info = {
06         "name": "sql_basic",
07         "show_name": "SQL注入检测插件",
08         "description": "基础sql注入漏洞检测插件"
09     }
10
11     # yinhuochong
12     def mutant(self, rasp_result_ins):
13         """
14             测试向量生成
15         """
16         if not rasp_result_ins.has_hook_type("sql"):
17             return
18
19         payload_list = [("1'openrasp", "1'openrasp"),
20                         ("1\"openrasp", "1\"openrasp"),
21                         ("a`openrasp", "a`openrasp")]
22
23         # 获取所有待测试参数
24         request_data_ins = self.new_request_data(rasp_result_ins)
25         test_params = self.mutant_helper.get_params_list(
26             request_data_ins, ["get", "post", "json", "headers", "cookies"])
27
28         for param in test_params:
29             if not request_data_ins.is_param_concat_in_hook("sql", param["value"]):
30                 continue
31             payload_seq = self.gen_payload_seq()
32             for payload in payload_list:
33                 request_data_ins = self.new_request_data(rasp_result_ins, payload_seq, payload[1])
34                 request_data_ins.set_param(param["type"], param["name"], payload[0])
35
36                 hook_filter = [{{
37                     "type": "sql",
38                     "filter": {
39                         "query": payload[1]
40                     }
41                 }]
42                 request_data_ins.set_filter(hook_filter)
43                 request_data_list = [request_data_ins]
44                 yield request_data_list
45
46     # yinhuochong
47     def check(self, request_data_list):
48         """
49             请求结果检测
50         """
51         request_data_ins = request_data_list[0]
52         feature = request_data_ins.get_payload_info()["feature"]
53         rasp_result_ins = request_data_ins.get_rasp_result()
54         if rasp_result_ins is None:
55             return None
56         if self.checker.check_concat_in_hook(rasp_result_ins, "sql", feature):
57             return "SQL语句逻辑可被用户输入控制"
58         else:
59             return None
```

首先对采集的请求进行解析，并将 Payload_list 中的 Payload 新增值测试参数中，随后发起请求。

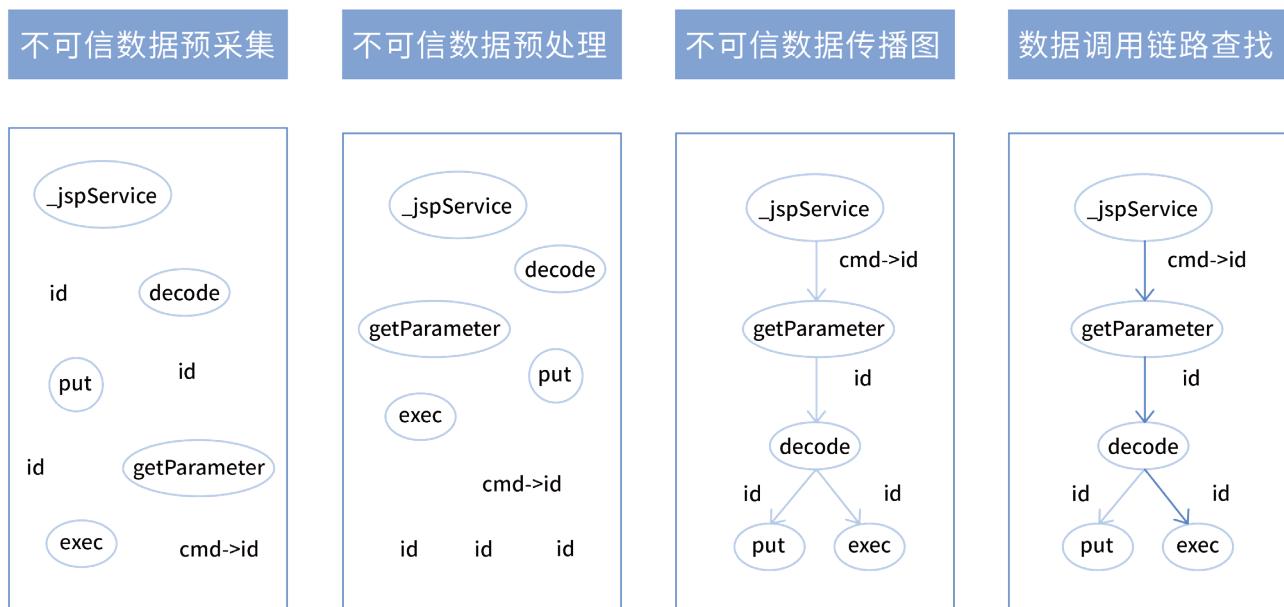
此时应用服务会收到这个请求进行处理，此时探针采集回来调用链数据，并上传至服务端，通过 openrasp_iast.plugin.scanner.sql_basic.ScanPlugin.check 判断调用链中是否存在传进去的 `payload_list` 字段，如果存在即为存在漏洞。

通过流量型检测来实现的 IAST 还是会存在一定的问题，如下图。

流量型 (被动式/代理式黑盒)	多方式采集测试流量(浏览器代理/ 客户端代理/流量镜像等) +流量重放+漏洞库	不区分开发语言	解决了黑盒爬虫无法覆盖部分场景的问题（如前后端分离） 但依然无法解决新场景新架构中无法检测的问题。 流量重放会存在较大延时，并生成脏数据。
主动插桩	Agent端采集特定流量+少量流量重放	区分开发语言	Agent端只采集存在危险方法调用的请求流量，并通过发送少量的验证数据包判断是否存在漏洞。采集和重放的流量较被动式黑盒显著减少，但依然需要进行流量重放，无法解决新场景和新架构的检测问题依然会产生脏数据和少量延时。

(二) 被动式 IAST

被动式 IAST 的检测需要有功能测试 / 自动化测试，也就是需要有测试人员去触发对应的功能点，紧接着 Agent 采集当前接口的调用链，上报至 Server 端并检测。



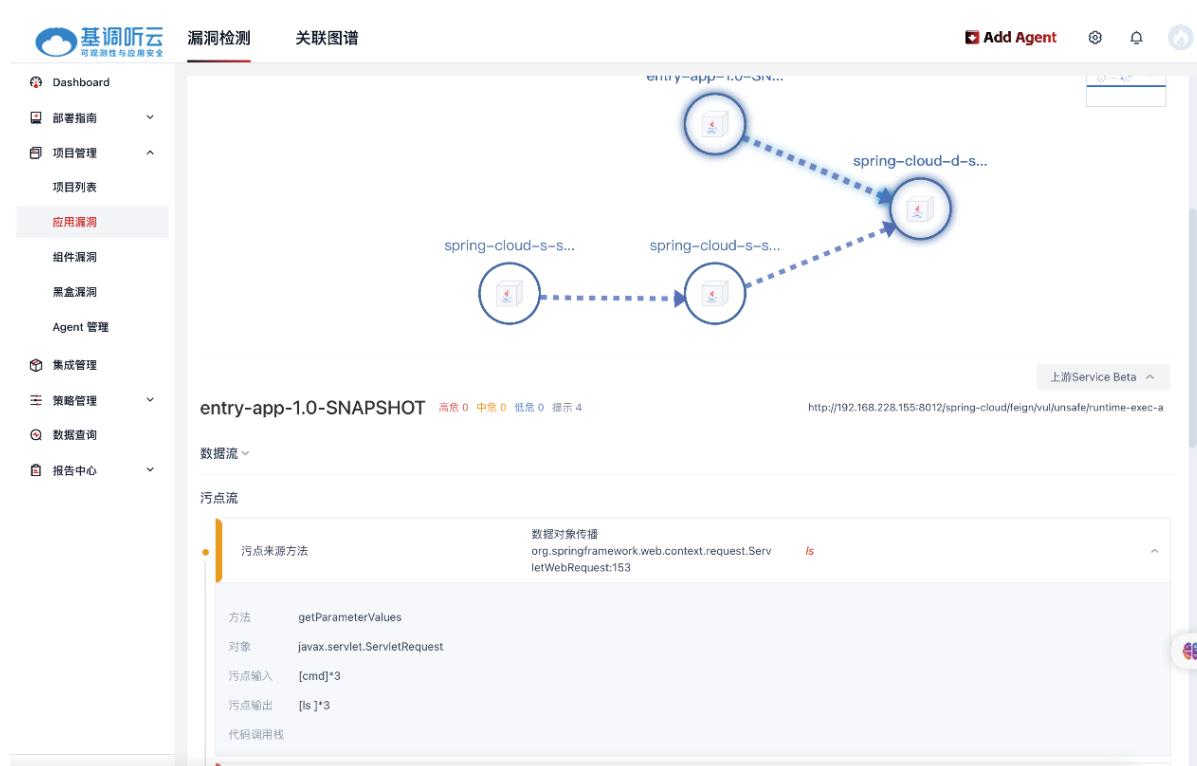
- 流程图解析 -

从图中可以看到，采集 `getParameter` 等参数入口点获取参数当做污点源，随后将采集的数据以及污点进行预处理，将采集到的污点和最终的危险方法进行还原，最后通过污点查找判断哪条链路可以通过外部参数可控直至最终危险方法，通过这样判断检测的漏洞能够实现实时的检测，无需等待扫描。

另外在微服务 /RPC 调用的场景下，由于被动式 Agent 是部署在不同的服务上，也就能够实现将不同的微服务调用进行串联，如下图。



This screenshot shows the Jiaduo Tingyun application security monitoring interface. The main title is "漏洞检测" (Vulnerability Detection) under the "关联图谱" (Relationship Map) tab. A specific vulnerability entry is highlighted: "http://192.168.228.155:8029/api/cmd-injection/unsafe/runtime-exec GET出现命令执行漏洞". The "基本信息" (Basic Information) section details the vulnerability as a "命令执行" (Command Execution) issue, first appearing on 2023.08.22 10:11:49, last active on 2023.08.22 10:12:06, with an "Unknown" intermediate and V1.0 project version. The "漏洞描述" (Vulnerability Description) section explains that command execution vulnerabilities occur when web applications do not filter user input strictly, allowing attackers to construct special command strings to execute external programs or system commands. Below this, a "Relationship Map" diagram illustrates the flow of data between four services: "entry-app-1.0-SNAPSHOT", "spring-cloud-d-service", and two instances of "spring-cloud-s-service". Arrows indicate the direction of data flow between these components.



This screenshot shows a detailed view of a vulnerability under the "关联图谱" (Relationship Map) tab. The main title is "entry-app-1.0-SNAPSHOT" with a count of 4 vulnerabilities. The "污点流" (Contamination Flow) section highlights a "污点来源方法" (Source Method) named "getParametersValues" from the class "org.springframework.web.context.request.ServletWebRequest" at line 153. It traces the flow of tainted data through methods like "getParameterValues", "getServletRequest", and "getCmd" (with input "[cmd]*3") to "getIs" (with output "[is]*3"). The "数据对象传播" (Data Object Propagation) section shows the propagation of the tainted object "ServletWebRequest" through the "getWebRequest" method. The "上游Service Beta" (Upstream Service Beta) section shows the URL "http://192.168.228.155:8012/spring-cloud/feign/vul/unsafe/runtime-exec-a".

并且能够实现展示污点在每一个微服务的数据流转过程，直至最终的漏洞触发，这样的优势非常明显的是能够让安全人员很清晰的看清漏洞最终触发在哪个微服务，减少排查和沟通的成本。

下图展示了被动 IAST 的优势项。

被动插桩	Agent端监听数据(流) +检测算法存储	区分开发语言	基于值匹配算法和污点跟踪算法进行漏洞检测，无需采集和重放流量，解决了新场景、新架构的检测问题，且 无脏数据、检测实时 。
------	-----------------------	--------	--

03 总结

主被动式 IAST 分类以及优势点：

IAST	主动式	被动式
检测原理	DAST+RASP	动态污点分析
误报	基本无	存在
漏报	存在	较少
加密场景	不支持	支持
覆盖场景	常规	更多
脏数据	可能存在	无
开发难度	中	高
分布式	支持	支持
复现难度	容易	一般

为什么 IAST 需要运营？

基调听云专家顾问 郭梦飞

截至 2022 年，CNNVD 合计发布漏洞信息 199465 条，仅 2022 年新增漏洞信息就达到 24801 条，为历史新高，并保持连年增长态势。超高危级漏洞占比呈持续上升趋势。面对不断“更新”的安全漏洞与安全风险，安全产品不可能停下运营的脚步。

比如企业在落地白盒时，要去运营相应的检测规则；黑盒需要定期去维护相应的 Payload；蜜罐同样需要定制一套能够符合当前企业场景的高仿真蜜罐等。IAST 同样属于以规则为主的安全产品，也需要运营。本文将通过安全技术发展的需要、应用环境的复杂性、行业标准的差异三点来详细剖析 IAST 运营的必要性。

01 安全技术发展的需要

安全技术发展日益迭代，各种各样的漏洞也是层出不穷，比如核弹级漏洞 Apache Log4j2 的覆盖面之广，FastJson 组件迭代了数个版本都没将漏洞彻底修复，在每一版迭代修复后，都会有各种各样的利用链绕过迭代。而网络安全的本质是人与人之间攻与防的过程，没有绝对的安全，安全是一个需要持续运营的结果。因此，企业需要定期地对行业内新爆出的漏洞进行自检，但这样往往会造成重复的、机械化的工作量。因此，我们可以借助 IAST 实现应用上线前的漏洞检测自动化。只需要将这些漏洞的规则特征沉淀到 IAST 自定义危险方法规则当中，当代码逻辑执行到该处时，IAST 就可以像卡点工具一样批量地将最新的漏洞拦截在上线前，实现对 IAST 的规则赋能。IAST 赋能的规则越全面、越广泛，IAST 越能发挥它强大的作用，使应用的安全风险降到最低。

02 各个企业应用环境的复杂性

IAST 是通过 Instrument 技术进行插桩检测的，从原理上来说，与 APM 类产品的插桩原理有些相同之处。APM 更加关注性能上的变化，只在一些关键函数处进行 Hook；而 IAST 更加关注整体的链路变化，以分析漏洞的存在与否。倘若 IAST 与 APM 都去 Hook 同一个函数，就可能会导致存在一些冲突的点。对于市面上常见的 APM 插桩，IAST 已经做到最大的兼容性。但在实际的企业场景中，往往不仅仅包含市面上常见的 APM 产品，还会存在自研的 APM、代码覆盖率、服务治理等其它类型的常驻 Agent。

在我们曾经服务过某行业客户中，发现了一次兼容性问题。由于 JSON 数据交给了自研的 Agent 进行处理，而在处理这些 JSON 数据时，自研的 Agent 对其加了同步锁，但 IAST 也会对这个自研的 Agent 进行 Hook，这样就会拖慢应用的请求与响应。安云 IAST 针对这种情况也进行了优化：在 Agent 包内忽略内部实现，跳过了一些耗时的方法。随着迭代，安云 IAST 将忽略内部实现方法的机制开放出来，用户也可以通过针对性的设置，提升可用性。

黑名单配置，是 IAST 的 Agent 为了避免出现兼容性问题的另一种解决方案。通过对一些不必要的类默认放入黑名单中，包括但不限于 APM 的类等，以避免这些类被 Hook(类的所有方法本身，不含类方法内部的其他方法调用)。这个黑名单的配置可以是一个前缀，比如 org/abc/def/*，此时会产生一个新需求：如果要对这个黑名单内的类或者前缀后面的某个类进行 Hook，就需要配置白名单了。将白名单优先于黑名单进行处理，可以适应更多企业场景。

即是经过了上面的兼容性处理，IAST 在面对自研和二开的框架时，仍然可能会遇到可用性及漏洞检测的问题。比如，有一些企业在二开或自研的过程中更改了 Servlet，没有使用官方版本，会导致 IAST 无法捕捉到 Source 点，根据 IAST 的检测原理，也就没有办法进行信息采集，就会出现 0 漏洞产出的情况。这需要针对具体的框架添加相应的入口方法。通常，这些问题在前期 POC 测试就会暴露出来，但难免也会在批量部署的过程中遇到非常规的业务。IAST 需要针对具体情况进行优化兼容，以保证 IAST 在当下环境的可用性。毕竟只有保证了“能用”，才有“用好”的基础。而“用好”的关键，就在于如何运营检测规则。比如，我们处理过很多用户的反馈：一个漏洞在入参的时候 Filter 做了过滤，这个漏洞其实可以算误报，但如何降低这种误报？这就在于过滤方法的规则赋能，也驱使 IAST 需要运营，针对当前企业场景，IAST 少误报，才能有信心将其作为上线前的卡点。

03 行业标准的差异

对于漏洞等级来说，市面上并没有统一的标准来指定哪一种漏洞是高危，哪一种是低危。行业里会有一些默认的规则，比如 SQL 注入定为高危漏洞，XSS 定为中低危漏洞。但是，相对于企业来说，漏洞的危害评判应该由该企业根据自身实际情况来自行判断。举个例子，某金融机构将敏感信息泄露（如手机号、身份证等）作为高危项，但其他行业的企业可能并不认为身份认证漏洞是高危。这实际上就是针对漏洞的评判等级不同所引发的区别。因此，IAST 需要在企业内部进行运营，制定适合企业的漏洞评级标准。

从代码规范治理的角度来看，修复方案应该由企业内部的安全部门和研发部门进行统一制定。安云 IAST 也会给出相应的修复建议，但修复建议也应该是定制化的。比如究竟应该去引用哪个过滤方法来完成参数的合法性验证？这都是偏向企业定制化的场景。

安全没有一劳永逸的安逸，IAST 也不是部署后就可以高枕无忧万事大吉的万能产品，不仅仅是 IAST，几乎每一款安全产品都要用心运营。只有针对企业自身实际业务与架构情况进行针对性的配置，并加以持续运营，才可将安全产品的作用发挥至最大。

落地实践

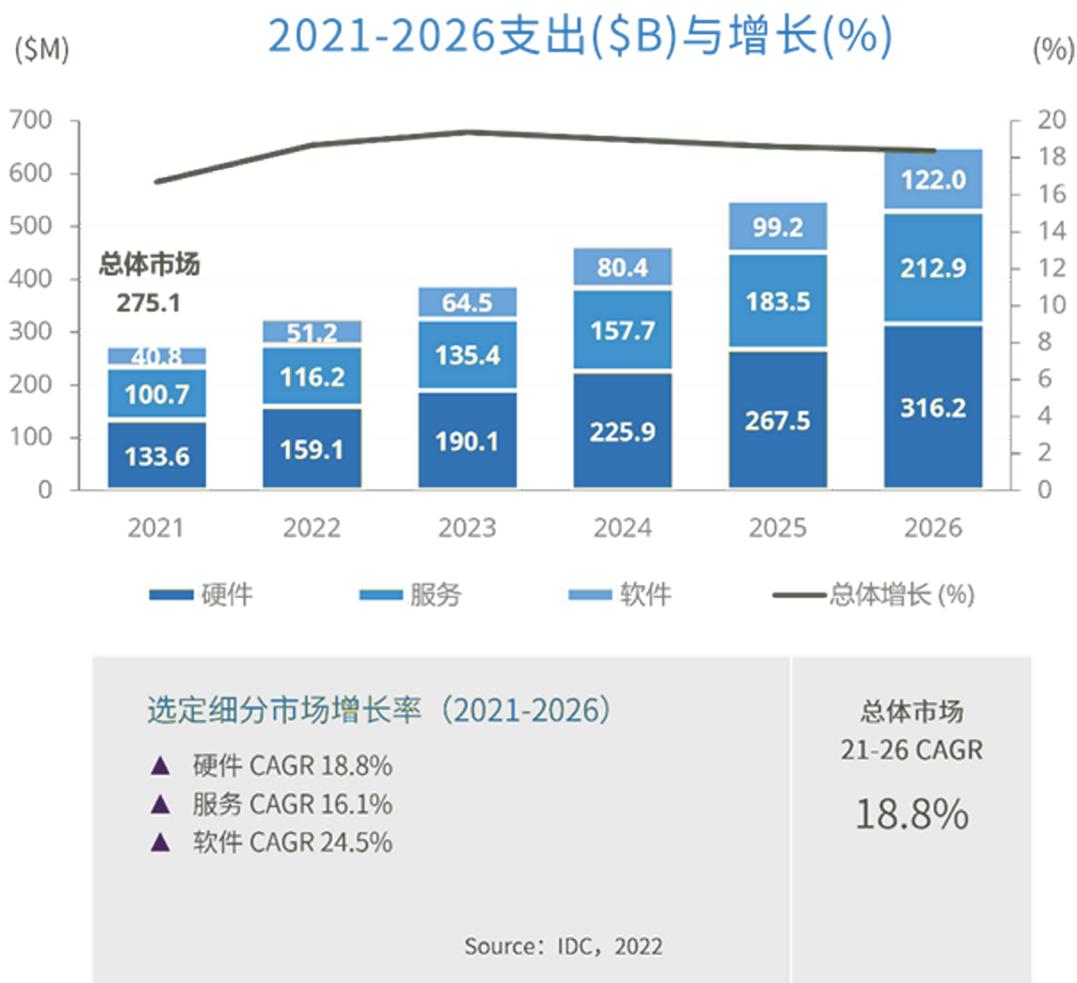
CASE STUDY

安云 IAST 证券行业落地实践

基调听云高级市场经理 庞伊良

01 前言

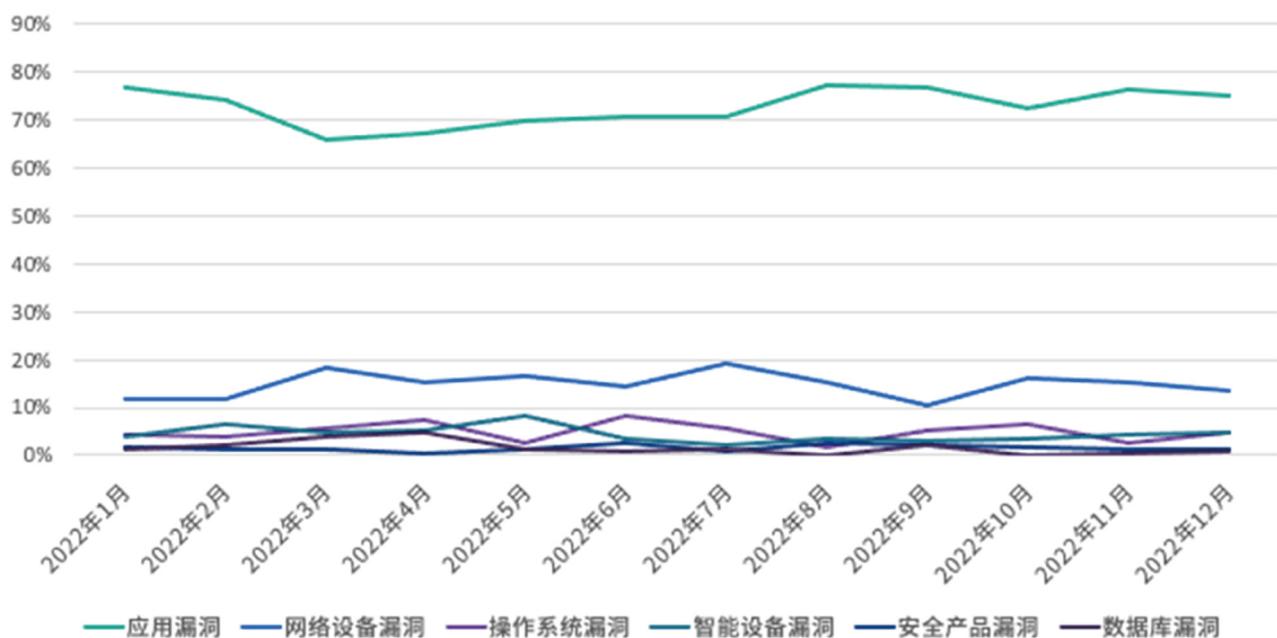
IDC 在《中国数字化转型市场预测，2021–2026：通过应用场景践行数字化优先策略》中预测，未来五年是数字化发展的黄金时期，企业在应用硬件、软件和服务上的投入将快速增长。证券行业作为金融行业的代表，是数字化转型浪潮中的领航员，在应用开发、应用安全方面面临着更多机遇与更大的挑战。



- 图注: IDC, 《中国数字化转型市场预测, 2021–2026：通过应用场景践行数字化优先策略》 -

伴随着数字化浪潮而来的，还有日益严重的应用安全问题。在 2022 年国家信息安全漏洞共享平台 CNVD 公布的漏洞趋势中，应用漏洞几乎始终占据 70% 以上的占比。应用安全问题不仅带来了个人隐私泄露风险，影响企业经营，在金融这样与国计民生息息相关的行业中，会直接影响社会经济正常运转。

2022年国家信息安全漏洞共享平台（CNVD）漏洞趋势



- 图注：2022 年国家信息安全漏洞共享平台（CNVD）漏洞趋势 -

2023 年 6 月，中国证券业协会(下称中证协)正式印发《证券公司网络和信息安全三年提升计划(2023—2025)》，提出合理加大科技资金投入。鼓励有条件的证券公司在 2023—2025 三个年度信息科技平均投入金额不少于上述三个年度平均净利润的 10% 或平均营业收入的 7%，并保持稳定的资金投入，相比征求意见稿中的 8% 与 6% 又有所提升。一些省份如江苏省也发布了省级的金融管理办法，要求将 IT 建设投入的 5% 以上投入在网络安全领域，足以看出国家与监管机构在要求企业保障网络安全方面的重视程度。

保障业务连续性无疑是证券行业进行网络安全建设时的首要目标。在证券行业中，安全部门需要保障业务运行中不出现安全事件、不出现安全事故，这仅仅依靠应用上线后的各种安全防护设备显然是不合理也不可行的。《三年提升计划》明确要求健全网络和信息安全防护体系，深化漏洞全生命周期管控。要求使用白盒检测、黑盒检测、灰盒检测和人工渗透相结合的技术手段，检测代码安全缺陷和引入的第三方组件漏洞，提升安全风险检测的全面性、准确性和效率，实现漏洞通报、修复的闭环管理，确保变更上线前已知风险全面收敛。为了弥补安全开发实践的欠缺、提前发现安全威胁、降低漏洞修复成本，许多证券行业企业开始探索建设 DevSecOps 流程。

02 证券行业应用安全测试思考

随着金融行业的网络安全风险不断累积，证券行业网络安全防护也面临着前所未有的威胁与挑战。以 XSS、SQL 注入、敏感信息泄露、未授权访问、弱口令、远程代码执行、任意文件读取、逻辑漏洞等为代表的 Web 安全漏洞依旧是最多发的漏洞类型。应用系统上线后漏洞修复往往要耗费较长时间和较高的成本。

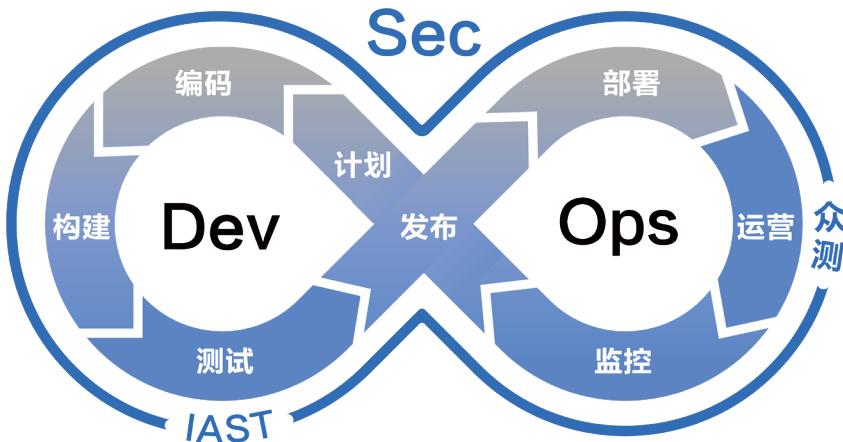
在实际的业务运行过程中，越来越多的证券行业企业发现原有的软件开发与安全测试流程过于复杂，工作繁琐，且对于人工投入要求较高，很难适应业务的敏捷快速迭代开发模式。推动安全左移，建设完整的 DevSecOps 流程将是证券行业的主流做法。

AST、IAST、DAST、SCA 依据不同的检测原理，都是推动企业客户安全左移、构建完整 SDLC 流程，助力 DevSecOps 概念落地不可或缺的一部分。

AST 是在软件开发生命周期 (SDLC) 的早期阶段（可以理解为编码阶段）发现代码漏洞，由于 AST 是通过扫描源代码发现漏洞，所以能够精确给出存在漏洞的代码行。

DAST 是在软件开发生命周期 (SDLC) 的中后阶段（可以理解为运行阶段）发现系统漏洞，由于 DAST 不扫描源代码，所以通常不能给出存在漏洞的具体代码位置。

IAST (Interactive Application Security Testing, 交互式应用程序安全测试, 也叫灰盒) 通过插桩技术 (Instrumented) 收集安全信息, 持续地从内部运行的代码中发现安全和逻辑问题, 提供实时的报警和展示, 从而帮助开发人员快速定位和修复相关问题, 在应用上线前发现安全漏洞, 降低漏洞修复成本。IAST 适用于软件开发生命周期 (SDLC) 的开发与测试阶段, 可以无缝集成至 DevSecOps 敏捷安全开发流程中。

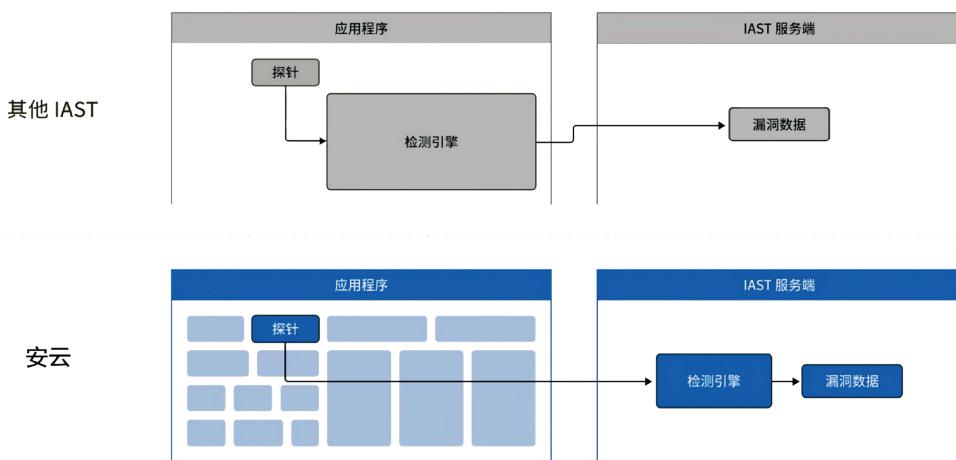


IAST 目前已经在证券行业部分头部企业获得了初步应用, 但因缺乏落地实践案例的指引, 整体上在证券行业的全面推广还面临不小的难度。

03 安云 IAST

安云 IAST 是由基调听云自主研发的 IAST 产品, 在开源社区共同的开发和维护下, 已在全球近 30 个国家和地区, 帮助 200 多家企业部署了数十万个节点, 拥有行业内最多的落地应用, 并形成了国内首个《IAST 落地实战笔记》, 是最能落地的 IAST 产品。

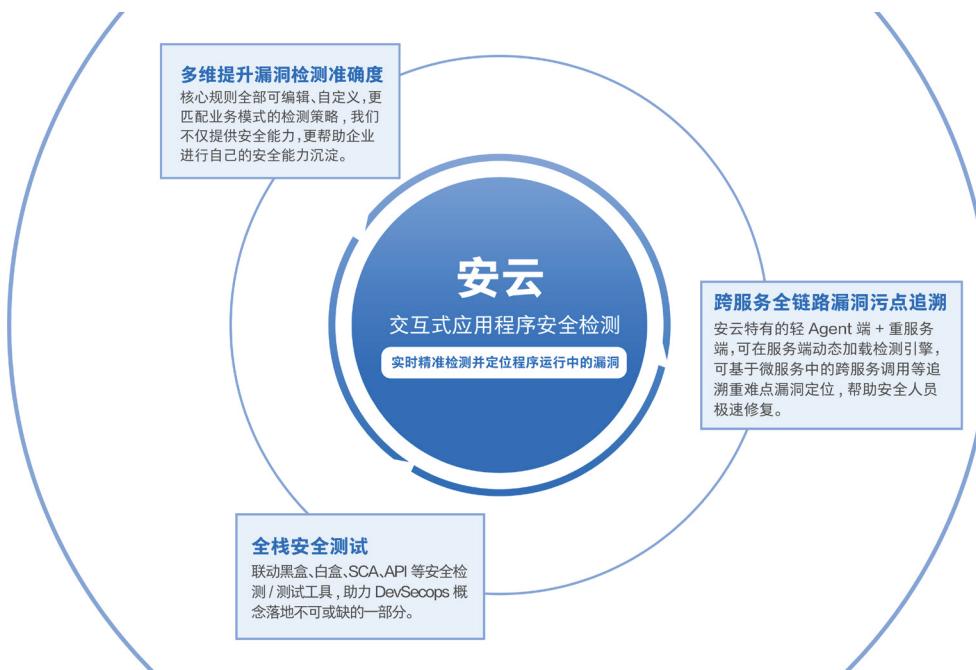
安云 IAST 专注用户体验, 独创数据采集、扫描引擎分离架构, 极大地提升了 IAST 的稳定性, 十万级部署, 万级并发, 仍能无感运行。



- 图注: 安云 IAST 独创数据采集、扫描引擎分离架构 -

安云 IAST 可以完美适配敏捷开发和 DevSecOps 流程，让开发人员和测试人员在执行功能测试时，实时、动态、同步进行漏洞检测，可精确确定漏洞所在代码行，在无感知的情况下完成安全测试。

安云 IAST 在程序运行过程中使用插桩技术监控和收集信息，根据这些信息来判断程序是否存在漏洞与安全风险。它对来自客户端产生的请求和响应进行分析，这点类似于 DAST，具备了黑盒中检测结果准确的特征；而它能够监控数据流信息，通过污点分析产生告警又类似于 SAST，检测结果也具备了白盒的全面性特点。



- 安云 IAST 在协助某证券行业客户安全左移、构建完整 SDL 流程过程中，发挥了独特的作用 -

04 安云 IAST 在证券行业的落地实践方案

我们以安云 IAST 服务过的某头部大型证券客户为例，介绍 IAST 在证券行业真实的落地实践方案。

(一) 证券行业客户现状

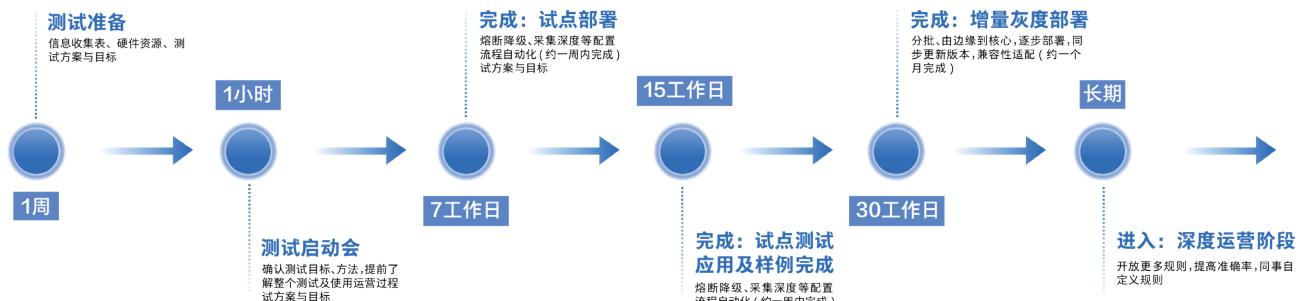
该证券行业客户是一家领先的科技驱动型证券集团，综合实力位居国内证券业第一方阵。得益于其在业务数字化转型与网络安全建设方面起步较早，内部已经拥有一套完整的安全评估与漏洞管理、修复流程，自身具备较强的安全实力，自研能力很强。通过自研的 SDL 全流程赋能平台，集成有威胁建模、自动化测试等功能，实现了自动化基于场景的轻量级威胁建模能力，支持从外部接入源代码分析、AST 等工具和渗透测试服务。

配合责任清晰的安全 BP 制度，已成功上线并运营 Contrast 多年，熟知 IAST 的技术原理和产品落地场景，因为国产化的推进，需要替换原工具。

安云 IAST 也在接入该 SDL 全流程赋能平台，融入客户 DevSecOps 流程的过程中进行了一些全新的探索。



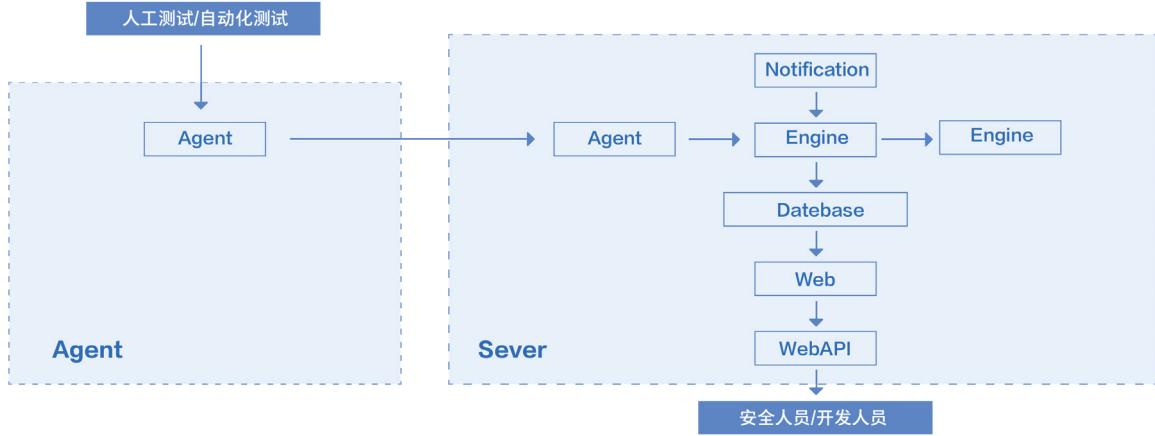
该证券行业客户体量庞大，资产、应用、数据众多，应用上线前依赖人工渗透测试寻找安全隐患，并且为了确保上线业务的安全性，还需要多人渗透并进行交叉验证。人工投入大、人员能力要求高，效率难以满足业务的敏捷快速迭代开发。那么是否可以通过接入 IAST 工具，实现应用上线前的漏洞自动化检测与发现，把人从重复性劳动中解放出来呢？答案是肯定的，但是在落地实施过程中还会面临不少现实问题。



- 图注：在证券行业中接入 IAST 实现自动化漏洞发现实施过程思路 -

(二) 安云 IAST 接入实践

接入自动化 IAST 的第一步是插桩，想要在如此庞大的体量中使用 IAST 进行全量覆盖，需要部署过万个节点，Agent 的部署是其中的重点也是难点。安云 IAST 采用了数据采集与扫描引擎分离的独特架构，Agent 端只负责采集数据，所有运算分析均通过 Server 端完成，有效地降低了资源占用，这让安云在支持数万个节点的部署与上万个节点并发的同时，仍能保持生产级别的 Agent 稳定性。



对于头部证券的业务部门来说，脏数据是无法接受的。上万个节点的并发检测如何能不产生任何脏数据？安云 IAST 的解决方案是真正的纯被动插桩模式。被动插桩模式不会主动发送 Payload，对来自客户端的请求响应进行污点传播数据流监控，根据是否经过无害化处理判断是否存在漏洞。只需要手动或自动进行业务测试，安全测试也会同步启动，通过测试流量即可实时地进行漏洞检测，并不会影响同时运行的其他测试活动，在此过程中不会产生脏数据。安全部门不再需要花费时间和精力处理脏数据，不再需要和研发沟通重放 Payload 等问题。



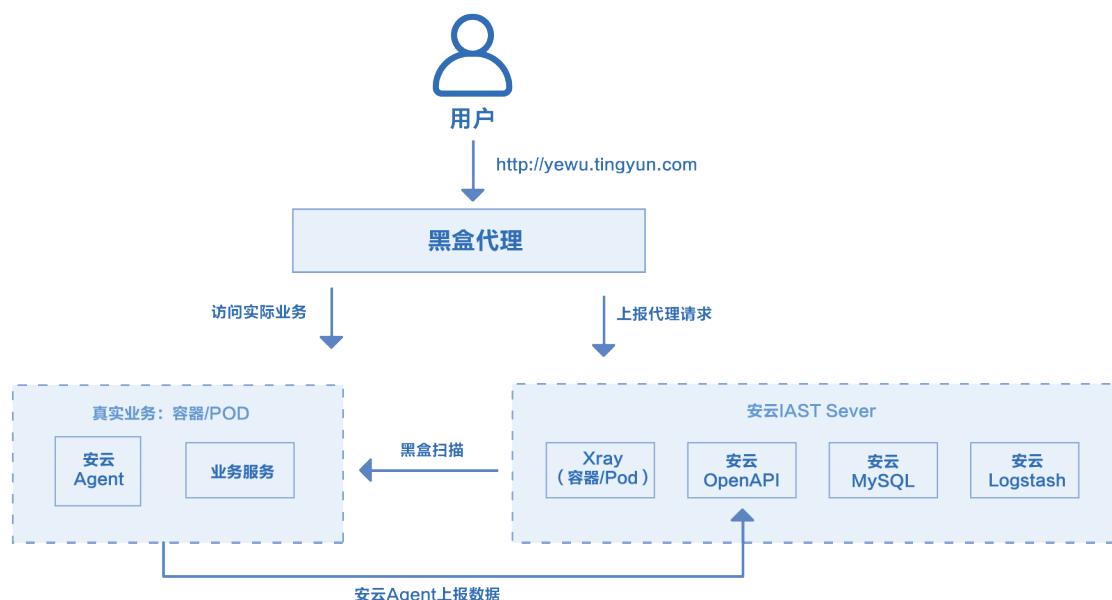
实现规模化 IAST 插桩检测后，检测的准确性是另一个关键问题。本身 IAST 作为结合了 SAST 与 DAST 部分优势特性的工具，就已经具备了检测结果准确、误报漏报少的优势。如何在 80 分的基础上更进一步，做到 90 分甚至 95 分？我们发现将 IAST 与白盒联动管理，进行风险展示和门禁管控，能够按照应用 - 单元 - 版本号等字段管理扫描结果。这样就实现了在测试验收阶段加入安全度量指标，更直观地展示 IAST 检查结果，便于后续进一步处理。

在与 SDL 全流程赋能平台对接的过程中我们发现，该平台作为一个 SDL 全流程赋能平台，接入了包括 IAST、DAST 等多种单点安全工具，各个单点安全工具均在应用安全测试中拥有着各自的独特价值。IAST 产品的特点决定了它可以提供更高的测试准确性，可详细地标注漏洞在应用程序代码中的确切位置，帮助开发人员修复。但同时 IAST 也存在一定的限制，对于部分复杂的漏洞场景，由于漏洞检测不依赖真实的漏洞 Payload，所以在验证漏洞的可利用性时，可能存在一定难度。

但是目前各单点工具之间仍然是独立工作，难以协同。如今，如同木桶效应中各块木板的长短一般，相较于追求某个单点安全工具的极致性能，探寻安全能力和安全数据之间联通的可能性或许是一个更具性价比也更高效可行的安全建设路径。在拥有可以满足核心安全需求的单点安全能力的条件下，通过联通性与安全效能的叠加性可以获得更大的经济收益与安全保障效果。

即使是拥有了该 SDL 全流程赋能平台提供的完整 SDL 能力，要实现各个单点工具之间的联动仍然是十分困难的，需要耗费极大的人力与时间。因此，我们开始思考如何通过 IAST 直接与其他应用安全测试工具结合起来，以更方便地验证漏洞，从而降低推进漏洞修复工作的难度。

目前，安云 IAST 已经实现了与黑盒 DAST 工具的一体化联动，黑盒工具可根据以下架构在实现请求头修改和数据同步之后对漏洞数据进行关联。



- 图注：安云 IAST 与黑盒 DAST 一体化联动架构 -

有了安云 IAST 成功接入 SDL 全流程赋能平台的经验，且安云 IAST 已实现与黑盒联动，相信在未来，单点工具之间、工具与平台之间互相联动协作，数据互通，必将帮助甲方更好地推进信息安全建设。

05 能落地，才有意义

安全部门的首要目标是保障业务连续，免受安全风险与威胁，安全产品与安全能力不能落地就没有意义。安全团队如何与开发、业务部门协作，是技术、产品、人员之外一个隐性的影响因素，特别是在 IAST 这种由安全部门采购，研发部门使用的安全工具，其中的影响表现得尤其明显。安全产品拥有良好的落地能力，高效、准确地发现与修复漏洞的过程提高了安全部门的价值，也间接推动了安全部门与其他部门的沟通合作，这对于甲方安全部门在内部推动整体安全建设是大有裨益的。

经过多年数字化转型的建设，证券行业信息化水平普遍较高，大多已经建立了较为完备的 DevSecOps 流程，在如今业务与监管的双重压力下，证券甲方企业需要补足在 DevSecOps 流程中各个节点的安全能力，更需要能落地、能联动的、好用易用的人性化安全产品。安云 IAST 凭借装机量大、稳定性强、检出率高、专注用户体验等优点使其正逐渐成为更多甲方客户的第一选择。

IAST 在数禾的实践与探索

数禾科技 应用安全工程师 李傲翔

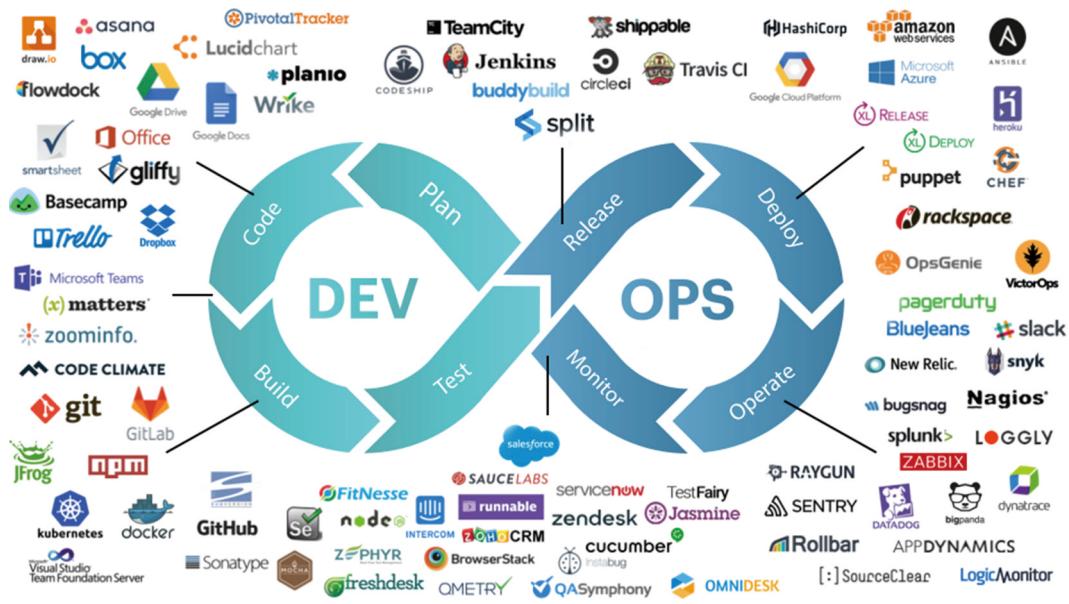
01 引言

近年来，随着互联网技术的快速发展，应用安全问题日益严峻。传统的安全测试方法，如人工渗透测试和代码审计，在应对快速迭代的软件开发模式下显得力不从心。IAST（Interactive Application Security Testing）灰盒安全测试技术作为一种新兴的自动化安全测试技术，具有高效、准确、实时等优势，受到了广泛关注。

02 背景

数禾科技依托自研的 DevOps 平台，实现了容器化部署和微服务架构的高效管理和自动化运维。在 2023 年之前，我司信息安全部主要依赖人工渗透测试保障应用系统自身的安全性。但随着我司 DevOps 体系的建设和完善，软件开发的规模和复杂性急剧增加，应用程序的迭代和部署愈加频繁，人工渗透测试面临着效率低下、测试覆盖率不足以及难以跟上快速迭代节奏的问题，采用自动化安全测试工具成为必然的选择。

在我司 DevOps 建设的阶段性成果的基础上，为了进一步优化开发、运营与安全的流程，将逐渐从 DevOps 向 DevSecOps 转变。DevSecOps 在 DevOps 的基础上全程引入了安全性，将安全性作为开发和运维的核心要素，通过自动化、CI/CD 等实践，将安全性纳入整个开发过程中。这样可以更早地发现和解决安全问题，提高软件系统的安全性和稳定性。



- 图 1 DevOps -

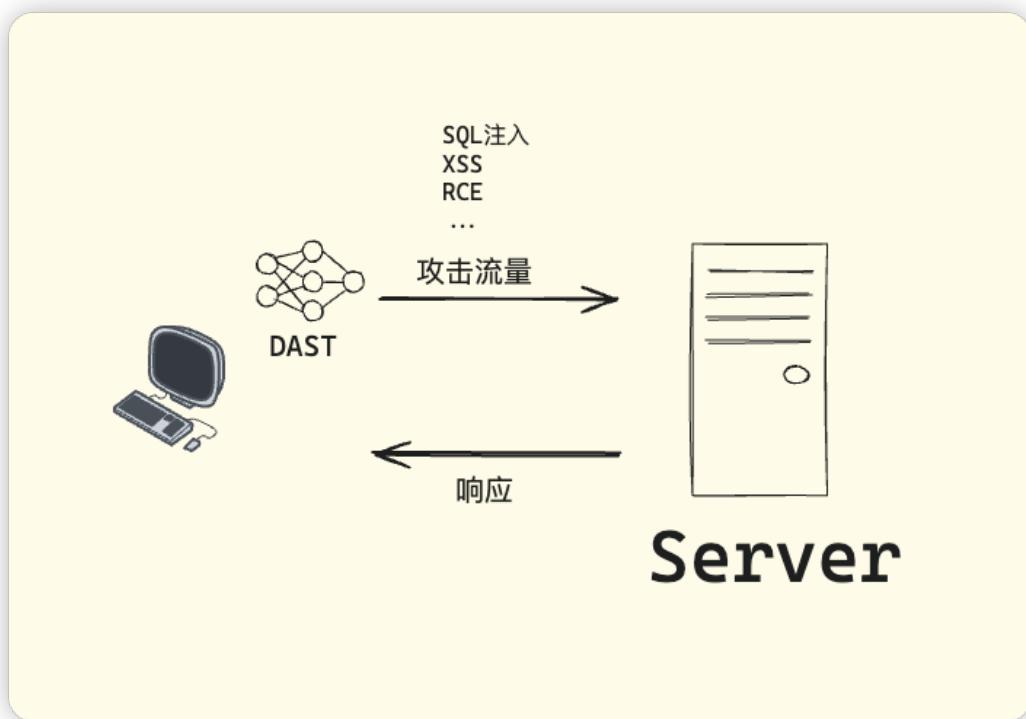
03 实践与挑战

(一) 技术选型

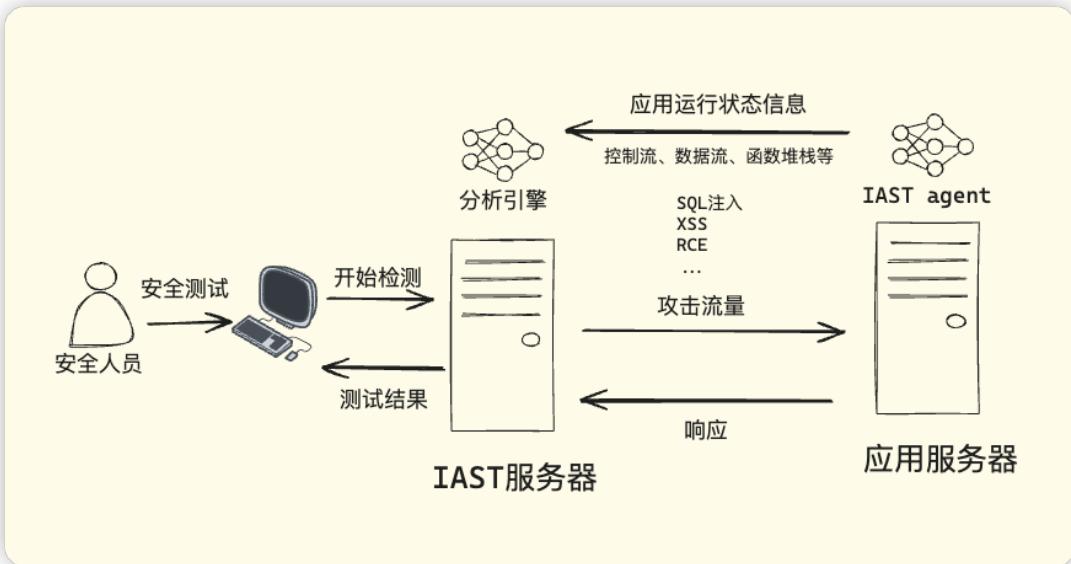
常见的自动化安全测试工具有静态应用安全测试 SAST (Static Application Security Testing) 、动态安全测试 DAST (Dynamic Application Security Testing) 以及交互式应用安全测试 IAST (Interactive Application Security Testing) , 三者的优缺点如下表。

	SAST	DAST	IAST
原理	通过分析应用程序的源代码、字节码或者二进制代码来检测安全漏洞。	通过模拟黑客攻击, 发送恶意请求和输入篡改数据, 通过分析应用程序的响应和行为判断是否存在漏洞, 如图2。	通过在运行时的应用程序中插桩来监控应用程序的运行状态, 从而检测出潜在的安全漏洞。 主动型IAST仅在Sink点（危险方法）进行相应的hook, 当检测到流量触发Sink点时, 插桩会通知服务端, 触发扫描器, 由服务端主动发送攻击流量来尝试触发安全漏洞, 工作流程如图3。

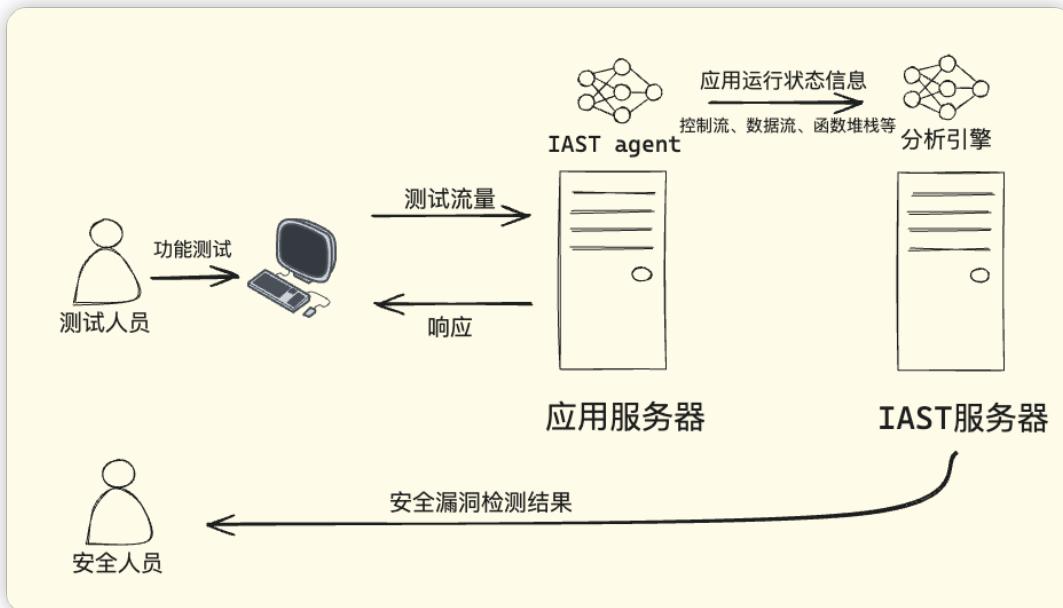
	SAST	DAST	IAST
优势	直接阅读源代码，漏报少，可以在代码研发阶段发现潜在的安全漏洞。	模拟真实攻击，可发现应用运行态漏洞，可进行自动化批量测试。	漏报误报低、检出率高；实时监测，可发现应用运行态漏洞；检出漏洞可定位到代码行；对应用性能影响小。
劣势	误报率过高、代码扫描时间过长等。	误报高，产生大量脏数据，覆盖的检测范围有限，难以定位检出漏洞的具体位置，验证繁琐。	需要插桩，插桩需要区分编程语言，安装部署较繁琐；对企业DevOps流程的完成度要求较高。
总结	难以满足我司DevSecOps快速迭代的开发需求。	主要作为人工安全测试的辅助工具。	更适合作为DevSecOps循环中的自动化测试工具。



- 图 2 DAST 工作原理 -



- 图 3 主动型 IAST 工作原理 -



- 图 4 被动型 IAST 工作原理 -

综上所述，尽管主动型 IAST 的准确率会更高一些，但存在注入脏数据、对应用并发压力大，以及新场景（加密、验签、防重放）无法覆盖等问题，而且侵入性较强，使用时容易对业务系统造成过高的干扰。而被动式 IAST 部署后可以实时监测漏洞，可随功能测试一同进行安全测试，并且完全不产生脏数据，将对业务系统的影响降到了最低，所以我司最终选择了被动型 IAST。

(二) POC 测试

为了测试 IAST 工具针对常见安全漏洞的检测效果，我司分别在内部应用和漏洞靶场上进行了 POC 测试。漏洞靶场选取了 OWASP 官方的 Benchmark Java 项目（项目地址：<https://owasp.org/www-project-benchmark/>），共测试了 2681 个测试用例，涉及 10 类常见的安全漏洞。下表是某厂商 IAST 工具在 BenchmarkJava 靶场上的测试数据，可以看到在靶场测试环境下，IAST 工具的误报率为 0。

漏洞类型	漏洞等级	测试用例数	检出漏洞数	误报数
SQL注入	高危	504	259	0
XPATH注入	中危	35	15	0
XXS	中危	455	189	0
不安全的 cookie	低危	67	36	0
不安全的 Hash 算法	低危	236	127	0
命令执行	高危	251	126	0
弱加密	低危	246	129	0
弱随机性	低危	493	193	0
总数	\	2681	1274	0
目录变量	中危	268	117	0
违反信任边界		2681	126	0

(三) 自动化集成

为了更好的将 IAST 工具自动化集成到我司现有的 CI/CD 流程中，需要实现以下需求：

- **高度自动化：** 无需人工干预，自动完成 IAST Agent 的安装和配置，并自动进行安全漏洞检测。
- **测试人员无感知：** 对测试人员完全透明，无需额外操作或学习成本，能够无缝融入测试流程。
- **应用版本全覆盖：** 覆盖测试环境所有部署的应用版本，不遗漏安全风险。

基于上述需求，我司采用了下列方案实现 IAST Agent 的自动化部署和自动化漏洞检测：

1. 修改基础镜像的 Dockerfile 文件，添加环境变量 JAVA_OPTS，指定 IAST 插桩 Agent 的路径，并传入应用名、应用版本号等变量；
2. 应用在测试环境进行版本发布时，触发 Jenkins 流水线（如图 5），并在“制作镜像”阶段设置容器的环境变量 JAVA_OPTS；
3. 在“部署环境”阶段完成后，容器启动，IAST Agent 上线，并自动在 IAST 服务器端关联对应的应用和版本号；
4. 在“执行自动化测试”阶段和后期的人工软件测试阶段，当测试流量访问该应用时，IAST 自动对该应用进行漏洞检测。



– 图 5 Jenkins 流水线 –

（四）系统对接

1. Webhook 对接

当 IAST 检出漏洞时，IAST 通过钉钉机器人发送漏洞信息，安全人员可点击链接跳转到漏洞详情，提高漏洞验证的效率。

2. JIRA 对接

当漏洞验证存在时，安全人员可直接在 IAST 系统上提交 BUG 工单到 JIRA 系统，且漏洞与 JIRA 相关联，便于推动漏洞修复的跟踪闭环。

3. DevOps 平台对接

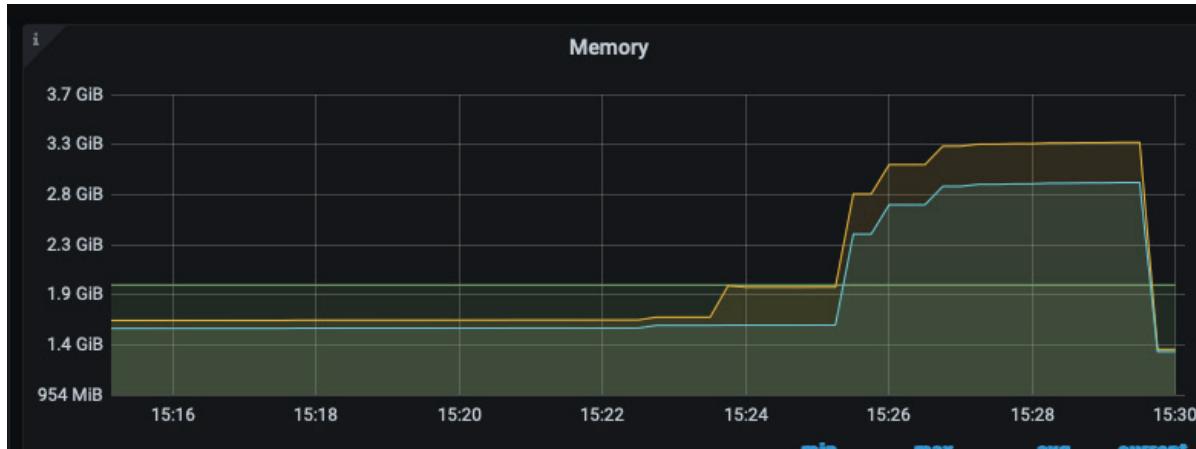
为了实现安全卡点的目的，可以将 IAST 与 DevOps 系统进行对接。当某个安全漏洞在 JIRA 系统关联的 BUG 工单未关闭时（修复完成、延期修复、转化需求、取消），该漏洞关联的应用版本将无法在 DevOps 平台进行发布。

(五) 运营优化

在 IAST 工具运营和使用的过程中，陆陆续续遇到了一些困难和挑战。为了解决这些问题，提高 IAST 工具的效果和稳定性，我们进行了一系列运营优化工作。

1. 应用重启

任何的插桩行为都必定会对系统性能产生一定的损耗，IAST Agent 也是如此，特别是在高并发场景下。在某些情况下甚至会发生容器重启的现象，如图 6。



- 图 6 容器重启 -

尽管应用重启发生在测试环境，还是会对测试同事的工作造成不便，违背了“测试人员无感知”的初衷。经过排查后发现，部署 IAST 后导致应用发生重启的原因主要是：

1. 部分接口请求流量过大，特别是一些心跳探活接口，导致 Pod 机器 CPU 和内存资源占用超过上限而重启；
2. 当应用服务器 CPU 和内存资源分配较为有限时，正常运行的应用在 IAST agent 插桩后可能会出现内存资源不足的情况，从而导致应用重启。

为了避免应用频繁重启，我司采取了以下规避风险的措施：

1. 事前解决方案

- 流量分析：在应用部署 IAST Agent 前，在 SLS 日志统计该应用各 URI 的请求量（采样 1 天的数据和 1 小时的数据），并进行分析。

- 黑名单策略：根据流量分析结果，将 HTTP 请求量较高的 URI（其中大部分为心跳接口）加入 IAST Agent 数据采集黑名单，并在后续运营期间做进一步处理（见 2.5.3 节）。

2. 事中解决方案

- 自动化监控：通过自动化脚本进行监测，当应用发生重启时，自动向服务器发送暂停 Agent 请求，并通过 Webhook 机器人进行告警。

- 人工服务：成立【IAST 服务台】工作群，开发和测试人员在测试过程中遇到疑似 IAST 导致的问题（1. 应用重启；2. 某个接口超时、无法访问）可在群里反馈，并及时获得技术支持。

通过上述措施，我们有效地降低了应用重启的发生率，规避了应用重启带来的不便之处，同时提升了测试同事的满意度。

2. 漏洞验证与修复

在 IAST 工具全面上线使用后，安全漏洞检出的速度超过了人工验证的速度，导致存量的待验证漏洞逐渐累积，有些漏洞之间可能还存在关联关系或共通性，导致安全团队无从下手，这不仅影响了安全团队的工作效率，还延缓了漏洞的修复进度。

为了解决上述问题，我们采用了以下解决方案：

- 漏洞优先级排序：根据应用等级、漏洞类型、影响范围等因素，对检出的漏洞进行优先级排序，按优先级对漏洞进行验证和修复。

- 公参过滤计划：部分注入类漏洞的污点源由公参传入，可以在网关处对公参进行统一过滤，在降低安全风险的同时，减少开发同事修复工作量。

3. 黑名单管理

在 IAST 运营过程中，为了提高数据采集的效率和准确性，需要将一些特定的流量加入 IAST agent 数据采集黑名单，从而避免 IAST Agent 对这些流量进行不必要的监控和分析。例如一些心跳接口、健康检查接口等，它们的请求频率较高，可能会对 IAST Agent 的性能产生影响。也有少数应用由于资源限制，需要将请求频率较高的业务流量加入黑名单中。

如何更新黑名单规则？如何保障黑名单中的正常业务接口的安全性？这些问题成为了 IAST 黑名单运营过程中的难题。

为了有效管理黑名单，我们采取了以下措施：

1. 建立黑名单管理机制：我们制定了一套明确的黑名单管理流程和规范，确保黑名单的添加、修改和删除都有明确的记录。

2. 定期审查和更新：我们定期对黑名单进行审查和更新，以确保黑名单的准确性和有效性。并及时进行清理和移除不再需要的黑名单项。

3. 自动化管理：我们通过自动化脚本每季度定期从 SLS 日志中提取需要加入黑名单的 URI 规则，以减少人工干预和错误的可能性。

4. 定期测试：针对黑名单中非心跳接口的部分接口，每季度规划安全测试计划对其进行安全测试，以确保即使在黑名单中的业务接口也不会因为缺乏 IAST 工具检测而增加安全风险。

通过黑名单管理，我们可以有效地减少 IAST 工具的误判和干扰，提高工具的准确性和稳定性。同时，黑名单管理也有助于保护敏感信息和降低 IAST 工具对系统性能的影响。

4. 逻辑漏洞检测

IAST 工具在逻辑漏洞检测方面存在显著的局限性。尽管 IAST 工具能够检测常见的注入类漏洞，例如 SQL 注入和跨站脚本攻击，但对于逻辑漏洞却无能为力。逻辑漏洞并非源于代码缺陷，而是代码逻辑设计上的缺陷，例如权限控制不当和业务逻辑错误等。这类问题往往需要深入理解业务场景和代码逻辑才能识别。

而逻辑漏洞在应用系统中普遍存在，且一旦被攻击者利用，可能造成严重的后果。攻击者可以利用逻辑漏洞进行越权访问，获取敏感数据、篡改业务数据或者执行恶意操作。因此，加强逻辑漏洞的检测和修复对于保障应用程序安全至关重要。

因此，我们提出了一种解决方案：借助 IAST 工具采集令牌，辅助人工测试逻辑漏洞。具体来说，IAST 工具在运行时会采集 HTTP 数据包，可以从这些数据包中提取未过期的令牌信息（Token、Session、Uid 等）。在人工安全测试的过程中，可以尝试使用不同的令牌组合，来模拟不同的用户行为和权限，从而发现潜在的逻辑漏洞。

04 总结与展望

IAST 工具自上线使用以来，已覆盖我司近 97%SpringBoot 应用，检出了大量高危漏洞，包括 SQL 注入、SSRF、不安全的 Json 反序列化、反射注入、路径穿越、SEPL 表达式注入等安全漏洞。

目前，IAST 工具对应用服务器的 CPU 和内存资源有一定的占用，这对于资源有限的应用场景来说，会造成一定的负担和限制。因此，如果 IAST 技术能够在降低 Agent 资源占用上有所突破，例如优化 Agent 的代码和算法，提升资源利用效率，或提供更灵活的 Agent 配置选项，能够满足不同应用场景的需求，我们将非常乐意进一步扩大 IAST 在我司的部署范围，在更大程度上融入 DevSecOps 流程。未来我们将继续优化 IAST 工具运营流程和策略，为我司应用安全提供更强有力的保障。

发展趋势

DEVELOPMENT TREND

IAST 与黑盒联动落地

基调听云高级安全顾问 高鹏

01 背景

金融行业一直以来都扮演着国家经济的重要支柱和引擎的角色。随着中国经济的持续数字化转型，金融行业作为其中的排头兵，正在积极探索和应用最新的科技，以适应日新月异的市场需求。然而，数字化转型的道路并非一帆风顺，信息安全问题在这个领域显得愈发突出。近年来，金融行业不断升级的科技应用不仅带来了效率的提升，也引发了对网络和信息安全的更深刻关切。

在过去的几年里，金融行业在应用开发和安全领域面临着一系列挑战。为了应对这些挑战，越来越多的金融企业开始意识到建设完整的 DevSecOps 流程是实现敏捷快速迭代开发模式的关键。在这一背景下，交互式应用安全测试（IAST）作为一种前沿的安全检测工具，正逐渐引起金融行业的关注。

IAST 不同于传统的静态和动态应用安全测试方法。它不仅能够在应用运行时检测潜在的安全漏洞，还能够提供与应用运行环境实际交互的情境，从而更准确地发现潜在问题。在实际应用中，IAST 可以与传统的黑盒测试方法相结合，形成一种强有力的安全防线。黑盒测试通过模拟攻击者的视角，评估应用程序的薄弱环节，而 IAST 则深入到应用内部，通过实际交互寻找漏洞，提供了更全面的安全覆盖。

一篇突出的案例是证券行业部分头部企业在实际应用中成功将 IAST 与黑盒测试相结合，构建了一套完善的安全评估体系。通过将两种方法的优势互补，这些企业不仅能够更早地发现潜在的漏洞，还能够更准确地定位问题所在，从而大幅度提升了应用的安全性。这种联动落地策略在信息安全领域的探索中具有重要的启示意义，它不仅为证券行业提供了更有效的安全保障，也为其他行业在数字化转型中的安全实践提供了有益的借鉴。

综上所述，金融行业在数字化转型的浪潮中，面临着机遇与挑战并存的复杂局面。IAST 作为一项创新的安全技术，为金融行业提供了有效的工具来更好地保护其应用程序。而与黑盒测试方法的联动应用，更是将安全策略推向了一个新的高度。在接下来的篇章中，我们将深入探讨 IAST 的技术原理、落地实践以及对金融行业信息安全的积极影响。

目前安云 IAST 是唯一一个支持将第三方黑盒数据集成联动至 IAST 的产品，除了使用开源常见的 Xray, AWVS, APPScan 等产品外，还有部分为业务系统定制的黑盒扫描器或通用扫描器，在业务每次提测或迭代时，都会将这些工具的检测结果指标加入到流程中，来保证业务上线前的安全建设。

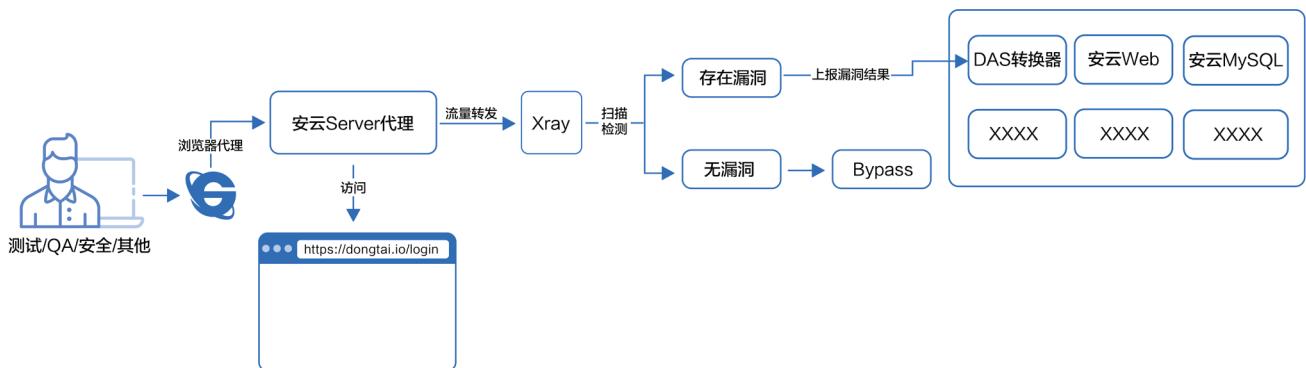
本文记录标准商业产品，自研黑盒扫描器结合 IAST 联动的过程与效果。

02 集成流程图

目前流程图有两种。

(一) 流程图一

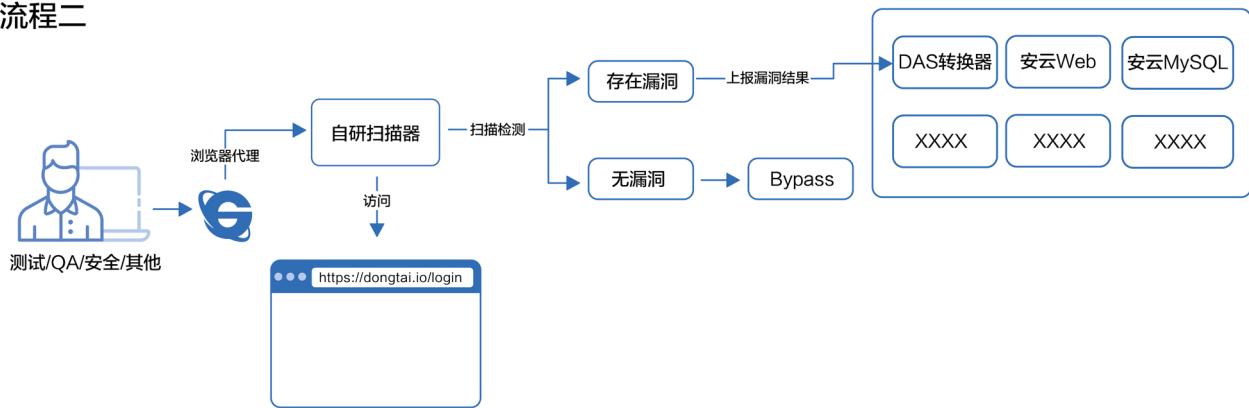
流程一



从流程图上来看，自研扫描器主要的工作原理是通过浏览器代理的方式（HTTP/HTTPS/Socket5），将测试人员 / 自动化测试 / QA / 安全 / 其他的测试流量经过安云 IAST 的代理，随后转发到自研扫描器，通过扫描检测判断是否存在漏洞，如果存在漏洞会则与 IAST 检测到的漏洞进行关联，或将扫描器漏洞上传至 IAST 进行聚合。

(二) 流程图二

流程二



从流程图上来看，自研扫描器主要的工作原理是通过浏览器代理的方式 (HTTP/HTTPS/Socket5)，将测试人员 / 自动化测试 /QA/ 安全 / 其他的测试流量经过自研扫描器，通过扫描检测判断是否存在漏洞，如果存在漏洞会将漏洞数据上传至 IAST。

(三) 那么这两种流程的区别在哪？

流程一的方式主要实现将自研 / 黑盒扫描器的扫描结果与安云 IAST 检测的漏洞结果进行关联。

流程二的方式主要实现将黑盒扫描器的扫描结果同步上传至同一个项目的漏洞列表中进行聚合。

03 集成步骤

(一) 流程图一实现

其实不难发现，上文流程图二并未将漏洞结果与安云 IAST 进行关联，只是将其他扫描器的漏洞同步至安云 IAST，下文中会对这一块内容进行实现。

1.Xray 开源版扫描

```
[root@iZbp14fqk9bl1v7rs4msZ Dongtai_USB]# docker-compose restart
[+] Running 3/3
  # Container dongtai_usb-xray-1      Started
  # Container dongtai_usb-usb-1       Started
  # Container dongtai_usb-mitmproxy-1 Started
[root@iZbp14fqk9bl1v7rs4msZ Dongtai_USB]# pwd
/root/Dongtai_USB
[root@iZbp14fqk9bl1v7rs4msZ Dongtai_USB]# cat config-tutorial.ini
[usb]
ip = 127.0.0.1,177.7.0.11,192.168.0.0/24

iast_url = http://127.0.0.1/api/v1/dast_webhook
dast_token = 837e62834b8423418c2a2a311c6

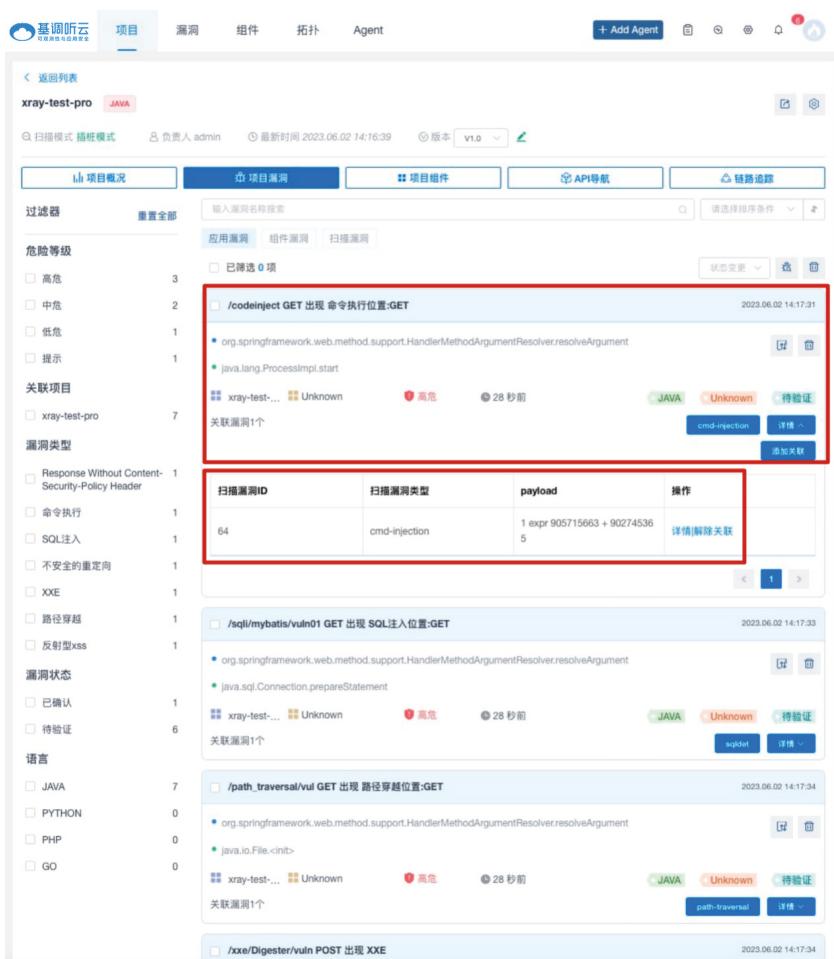
; xray_url = https://127.0.0.1/api/v2/vuln/filter/
; xray_token = xxxxx
```

注释掉的内容：商业版 Xray(洞鉴) Token

端口号	描述
10802	DongTai Proxy

需要开放端口

2. 测试与结果



The screenshot shows the Dongtai IAST platform interface. At the top, there's a navigation bar with '项目' (Project) selected. Below it, the project name 'xray-test-pro' is shown, along with its status as 'JAVA'. The main area displays a table of vulnerabilities. One specific row is highlighted with a red box:

扫描漏洞ID	扫描漏洞类型	payload	操作
64	cmd-injection	1 expr 905715663 + 90274536 5	详情 解除关联

This row corresponds to a vulnerability entry in the main list:

- 漏洞描述: /codeinject GET 出现 命令执行位置:GET
- 详细信息: org.springframework.web.method.support.HandlerMethodArgumentResolver.resolveArgument
java.lang.ProcessImpl.start
- 状态: 高危 (Red)
- 时间: 2023.06.02 14:17:31
- 操作: [编辑](#), [删除](#)

可以看到安云 IAST 的被动插桩 IAST 检出结果，与 Xray 的检测结果进行了关联，直观能够自动验证漏洞并且提供 Payload。

(二) 流程图二实现

1. 扫描器数据格式实现

在集成前首先需要实现 IAST 所支持的扫描器格式以及请求报文。

- 添加 Header 头部分

发送时需要携带以下请求头以标注接收格式：

```
X-Dongtai-Dast-Vul-Api-Version: v1
X-Dongtai-Dast-Vul-Api-Authorization: <40位uuid>
```

- 请求 Body

```
{
    "vul_name": "", #漏洞名 格式为 target+漏洞类型
    "detail": "", #漏洞详情
    "vul_level": "HIGH", #HIGH,MEDIUM,LOW,NOTE 漏洞等级，对应现在安云的4个等级
    "urls": [""], # 黑盒扫描发送的多个 url 地址
    "payload": "", # 黑盒扫描触发漏洞的 payload，可为空
    "create_time": 1679020853, # 时间戳(秒)
    "vul_type": "", #黑盒扫描的漏洞类型
    "request_messages": [{"# 一组扫描对应的所有请求和响应信息
        "request": "",
        "response": ""
    }
],
#以下为dongtai对接相关信息。
"dt_mark": [""], # dt-mark-header 的值
"target": "", # 原始请求地址
"dt_uuid_id": [""], # 可为多个，为插桩后Agent在响应头提供的uuid_id，需要在dt-request-id处拆分出来
"agent_id": [""],
"dongtai_vul_type": [""], # 安云的漏洞类型，多个类型，为空数组即对应所有调用链漏洞
"dst_tag": "", # 所集成的黑盒扫描器标识
}
```

○数据示例

```
{  
    'vul_name': 'http://127.0.0.1:8109/admin/userlist 越权(IDOR)漏洞',  
    'detail': '',  
    'vul_level': 'HIGH',  
    'urls': ['http://127.0.0.1:8109/admin/userlist'],  
    'payload': '',  
    'create_time': 1686807116,  
    'vul_type': 'SecurityTeam_IDORScanner',  
    'request_messages': [{  
        'request': '{\n            "Host": "127.0.0.1:8109",\n            "User-Agent":\n                "Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:109.0) Gecko/20100101\n                Firefox/114.0",\n            "Accept": "*/*",\n            "Accept-Language": "zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2",\n            "Accept-Encoding": "gzip, deflate,\n            br",\n            "X-Requested-with": "XMLHttpRequest",\n            "Connection": "keep-alive",\n            "Referer": "http://127.0.0.1:8109/usermanager",\n            "Cookie":\n                "JSESSIONID=A1DC8B8DB62CDC31E4EBD9C2AB1DF6C0; XSRF-TOKEN=12124f7c-fe25-4dea-8287-f2a01fce2e33;\n            session=eyJlc2VyaWQiojI3LCJ1c2Vycm9sZWlkIjo5ox0.ZIqTig.3Y2YZ26rGV96H3wI6ddbI5Ij0k0",\n            "Sec-Fetch-Dest": "empty",\n            "Sec-Fetch-Mode": "cors",\n            "Sec-Fetch-Site": "same-origin",\n            "cmd": "whoami"\n        }',  
        'response': '{\n            "code":200,\n            "data":[\n                {\n                    "createtime": "Fri, 16 Aug 2019 16:37:55\n                    GMT",\n                    "email": "xxxx",\n                    "id": 27,\n                    "idcard": 123,\n                    "imgurl": "/uploads/ECharts.png",\n                    "loginstatus": 1,\n                    "password": "123",\n                    "phone": 123,\n                    "role_id": 99,\n                    "truename": "123",\n                    "username": "admin"\n                },\n                {\n                    "createtime": "Fri, 16 Aug 2019 17:04:49\n                    GMT",\n                    "email": "123",\n                    "id": 40,\n                    "idcard": xxx,\n                    "imgurl": "/uploads/ios43.png",\n                    "loginstatus": 1,\n                    "password": "test",\n                    "phone": xxxx,\n                    "role_id": 2,\n                    "truename": "123",\n                    "username": "test"\n                }\n            ],\n            "msg": "success"\n        }\n    }],  
    'dt_mark': ['3c7ffcc7-c9d5-479c-afa3-50ce38c199f4'],  
    'target': 'http://127.0.0.1:8109/admin/userlist',  
    'dt_uuid_id': ['d1d3bad6-e5e4-4386-b21d-b3da80671edc'],  
    'agent_id': ['25'],  
    'dongtai_vul_type': ['sql-injection'],  
    'dast_tag': 'SecurityTeam_IDORScanner'  
}
```

2. 推送服务实现

安云 IAST 的 DAST 数据推送接口如下

```
/api/v1/dast_webhook
```

2. 推送服务实现

上文说到，需要在请求头中加入安云 IAST DAST Token: X-Dongtai-Dast-Vul-Api-Authorization

在部署安装包中找到: config-tutorial.ini 文件，从中可找到 X-Dongtai-Dast-Vul-Api-Authorization 的 token，具体实现如下：

```
headers = {
    "X-Dongtai-Dast-Vul-Api-Version": "v1",
    "X-Dongtai-Dast-Vul-Api-Authorization": iast_token,
    "Content-Type": "application/json"
}
```

```

68 ],
69     # 以下为dongtai对接相关信息。
70     "dt_mark": [f'{get_uuid()}'], # dt-mark-header 的值  guli
71     "target": target, # 原始请求地址
72     "dt_uuid_id": [f'{get_uuid()}'], # 可为空，可为多个，为插桩后agent在响应头提供的uuid_id，需要在dt-request-id处拆分出来saujiji
73     "agent_id": [agent_id],
74     "dongtai_vul_type": [dongtai_vul_type], # 洞态的漏洞类型，多个类型。为空数组即对应所有调用链漏洞
75     "dast_tag": dast_tag, # 所集成的黑白扫描器标识
76 }
77 logger.log("INFOR", "[+]Report DongTai IAST DAST Data Success!")
78
79 # 从配置文件中获取DongTai的服务端上报配置
80 config = Config().get_config()
81 if not Config().check_config():
82     exit(0)
83 logger.log("INFOR", "[+] Config Loader Success!")
84
85 headers = {
86     "X-Dongtai-Dast-Vul-Api-Version": "v1",
87     "X-Dongtai-Dast-Vul-Api-Authorization": config['DongTai_IAST'][2],
88     "Content-Type": "application/json"
89 }
90 print(data)
91 r = requests.post(url=config['DongTai_IAST'][0] + config['DongTai_IAST'][1],
92                     headers=headers,
93                     data=json.dumps(data))
94 print(r.text)
95 return data
96
5 usages ▾ CaCaGou*

```

由于将配置写入在 yml 文件中，为了扩展性与更改的便携性，所以将安云 IAST-Server 与安云 IAST-DastAPI 写入到 yml 中。完整实现代码如下：

```

def DongTai_IAST_DastScanner_Report(vul_name,
                                      detail,
                                      vul_level,
                                      urls,
                                      payload,
                                      create_time,
                                      vul_type,
                                      request,
                                      response,
                                      target,
                                      agent_id,
                                      dongtai_vul_type,
                                      dast_tag):
    ....

```

```
:param vul_name: 漏洞名 格式为 target+漏洞类型 示例: http://127.0.0.1:8080/users?id=1231231存在垂直越权漏洞
:param detail: 漏洞详情 示例: sql注入是XXXX
:param vul_level: HIGH,MEDIUM,LOW,NOTE 漏洞等级, 对应现在安云的4个等级 示例: HIGH
:param urls: 黑盒扫描发送的多个 url 地址 示例: http://127.0.0.1:8080/users?id=1231231
:param payload: 黑盒扫描触发漏洞的 payload, 可为空 示例: http://127.0.0.1:8080/users?id=1231231
:param create_time: 时间戳(秒)
:param vul_type: 黑盒扫描的漏洞类型
:param request: 一组扫描对应的所有请求信息
:param response: 一组扫描对应的所有响应信息
:param target: 原始请求地址
:param agent_id: 从IAST Agent管理页面查看AGENT id
:param dongtai_vul_type: 安云的漏洞类型, 多个类型, 为空数组即对应所有调用链漏洞
:param dast_tag: 所集成的黑盒扫描器标识
:return:
"""

data = {
    "vul_name": vul_name, # 漏洞名 格式为 target+漏洞类型
    "detail": detail, # 漏洞详情
    "vul_level": vul_level, # HIGH,MEDIUM,LOW,NOTE 漏洞等级, 对应现在安云的4个等级
    "urls": [urls], # 黑盒扫描发送的多个 url 地址
    "payload": payload, # 黑盒扫描触发漏洞的 payload, 可为空
    "create_time": GetTimeStampLimit(), # 时间戳(秒)
    "vul_type": vul_type, # 黑盒扫描的漏洞类型
    "request_messages": [{ # 一组扫描对应的所有请求和响应信息
        "request": request,
        "response": response,
    }
    ],
    # 以下为dongtai对接相关信息。
    "dt_mark": [f'{get_uuid()}'], # dt-mark-header 的值 suiji
    "target": target, # 原始请求地址
    "dt_uuid_id": [f'{get_uuid()}'], # 可为空, 可为多个, 为插桩后Agent在响应头提供的uuid_id, 需要在dt-request-id处拆分出来suijiji
    "agent_id": [agent_id],
    "dongtai_vul_type": [dongtai_vul_type], # 安云的漏洞类型, 多个类型, 为空数组即对应所有调用链漏洞
    "dast_tag": dast_tag, # 所集成的黑盒扫描器标识
}
logger.log("INFOR", "[+]Report DongTai IAST DAST Data Success!")
```

```
headers = {
    "X-Dongtai-Dast-Vul-Api-Version": "v1",
    "X-Dongtai-Dast-Vul-Api-Authorization": config['DongTai_IAST'][2],
    "Content-Type": "application/json"
}
print(data)
r = requests.post(url=config['DongTai_IAST'][0] + config['DongTai_IAST'][1],
                   headers=headers,
                   data=json.dumps(data))
print(r.text)
return data
```

3. 测试与结果

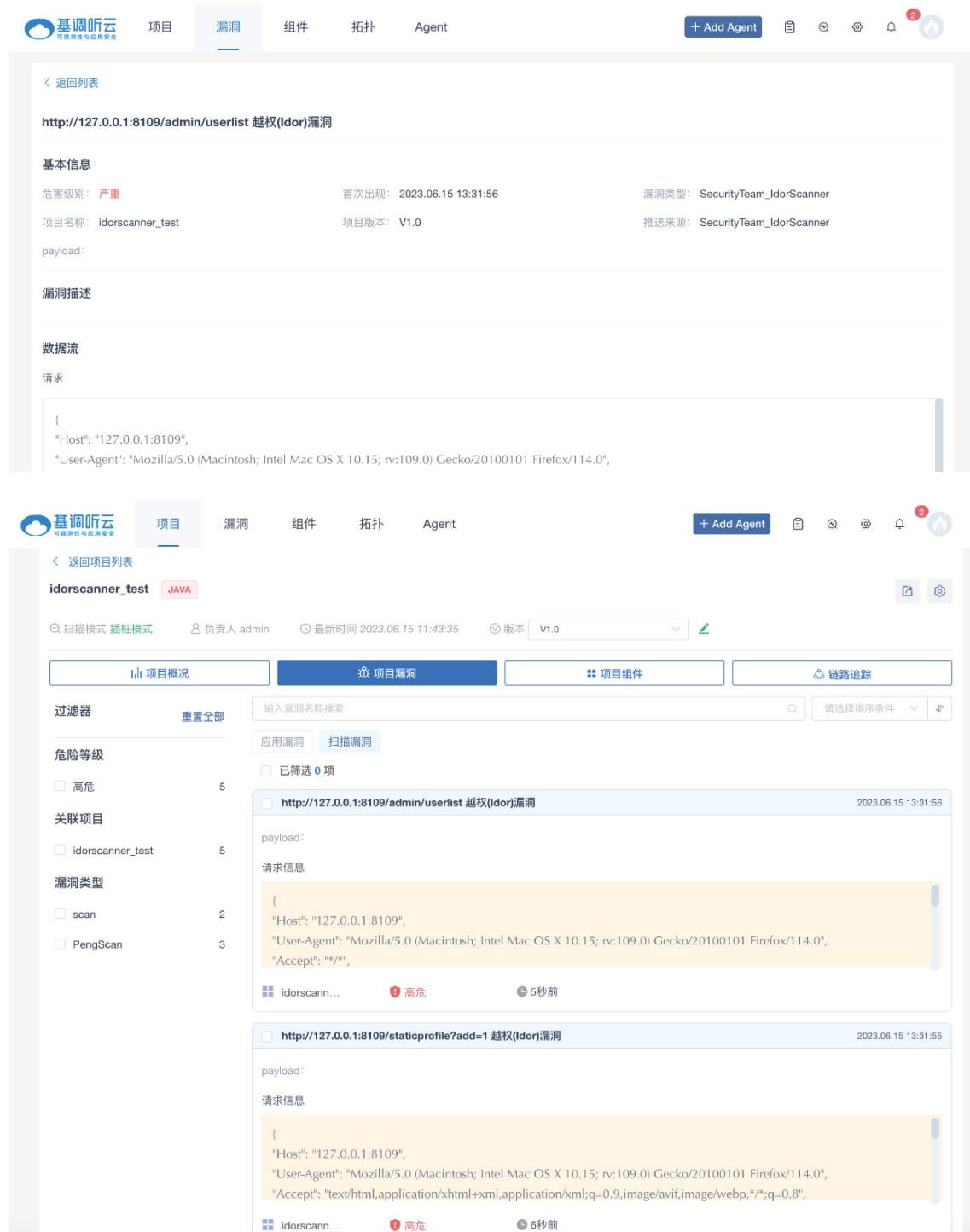
由于流程图二抛弃了安云 IAST 提供的代理，所以在此使用我司内部实现的代理进行漏洞扫描。

代理服务器

网址协议	代理协议	代理服务器	代理端口	
(默认)	SOCKS5	127.0.0.1	8081	▲ ▼ 锁
http://	(同默认)	127.0.0.1	8081	▲ ▼ 锁
https://	(同默认)	127.0.0.1	8081	▲ ▼ 锁
ftp://	(同默认)	127.0.0.1	8081	▲ ▼ 锁

– SwitchyOmega 配置 –

- 扫描队列信息 -



The screenshot displays two main sections of the Jiadolisten Cloud platform:

- Top Bar:** Includes the logo, navigation tabs (项目, 漏洞, 组件, 拓扑, Agent), a '+ Add Agent' button, and user profile icons.
- Left Column:** Contains sections for '基本信息' (Basic Information) and '漏洞描述' (Vulnerability Description). It also lists '数据流' (Data Flow) and '请求' (Request).
- Right Column:** Shows a detailed view of a specific vulnerability with its URL (`http://127.0.0.1:8109/admin/userlist 越权(Idor)漏洞`), first occurrence (2023.06.15 13:31:56), severity (严重 - High), and source (SecurityTeam_IdorScanner).
- Bottom Section:** A list of vulnerabilities for the project 'idorscanner_test' (JAVA). It includes filters for '扫描模式' (Scan Mode) and '插桩模式' (Instrumentation Mode), and search fields for '输入漏洞名称搜索' (Search Vulnerability Name) and '请选择排序条件' (Select Sort Condition).

4. 问题梳理

上述情况虽然可以将自研扫描器漏洞上传至 IAST，但是会存在以下几个问题：

1、目前没有经过安云的代理，所以 Agent 不会 Bypass 自研扫描器的请求，这可能会导致类似压测的情况出现。

2、目前漏洞没有进行关联，安云 IAST 实现关联主要通过 `dt_mark` 与 `dt_uuid_id`，在我们发起的请求中，是随机生成的，IAST 无法关联。

04 自研扫描器集成落地

(一) 流程梳理

上述总结了 2 个问题，所以我们思考一下流程：

有 dt-mark-header: <uuid> 且有 dt-dast: <scan-type> 的情况下不会采集；

仅有 dt-mark-header: <uuid> 的情况下会进行采集。

在第一次请求时需要给请求头中加入 dt_mark 也就是 dt-mark-header，随后将这个值保存，发送 payload 时带上 dt-mark-header 和 dt-dast 即可让 Agent bypass 请求。然后需要保存 dt_uuid_id 的值，最后上报漏洞的时候就将这个 dt-mark-header 和 dt_uuid_id 一起推送给接口即可。

所以得出流程如下，在经过自研代理时：

- 第一次获取请求头是需要单独加入 dt-mark-header，此时保存 dt-mark-header；
- 接下来构造 payload 发送请求，带上第一次保存的 dt-mark-header 和 dt-dast，这样就可以让 Agent bypass 这些 payload 请求；
- 如果请求存在漏洞，此时保存存在漏洞请求响应头中 Agent 返回的 dt_uuid_id (dt-request-id) 。

```
原始UUID: a9016cf4d407944da6a23498c97530a2b
赋值后的UUID: a9016cf4d407944da6a23498c97530a2b
第一次请求响应码: 200
第一次请求响应头: {"DongTai": "v1.11.0", "dt-request-id": "2.c0b6cdd5bd2d4081b019cc15b1a9fa7c", "X-Application-Context": "application:1234", "X-Content-Type-Options": "nosniff", "X-XSS-Protection": "1; mode=block", "payload": "1 and sleep(3)"}
扫描请求头: {"User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:109.0) Gecko/20100101 Firefox/114.0", "Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8"}
扫描URL: http://127.0.0.1:1234/sqli/mybatis/vuln01?username=admin and sleep(3)
扫描请求结果: {"DongTai": "v1.11.0", "dt-request-id": "2.3e5b258b94ee45fd8bad04880c09b0dd", "X-Application-Context": "application:1234", "X-Content-Type-Options": "nosniff", "X-XSS-Protection": "1; mode=block", "payload": "1 and sleep(3)"}
[19:52:48.625 [INFO] Scanner_Test:66 - [+]Report DongTai LAST DAST Data Success!
19:52:48.628 [INFO] Scanner_Test:72 - [+] Config Loader Success!
[2', c0b6cdd5bd2d4081b019cc15b1a9fa7c"]
2
c0b6cdd5bd2d4081b019cc15b1a9fa7c
{"vul_name": "http://127.0.0.1:1234/sqli/mybatis/vuln01?username=admin and sleep(3) SQL注入漏洞", "detail": "软件使用来自上游组件的外部影响的输入来构造全部或部分SQL命令，但不会中和或不正确地中和了特殊元素。这些特殊元素在将其发送到下游组件时可能被误认为是有效的SQL命令。", "status": "201", "msg": "\u064cd\u04f5c\u0621b\u0529f"}
```

○ 推送上报漏洞，需要满足的数据格式如下：

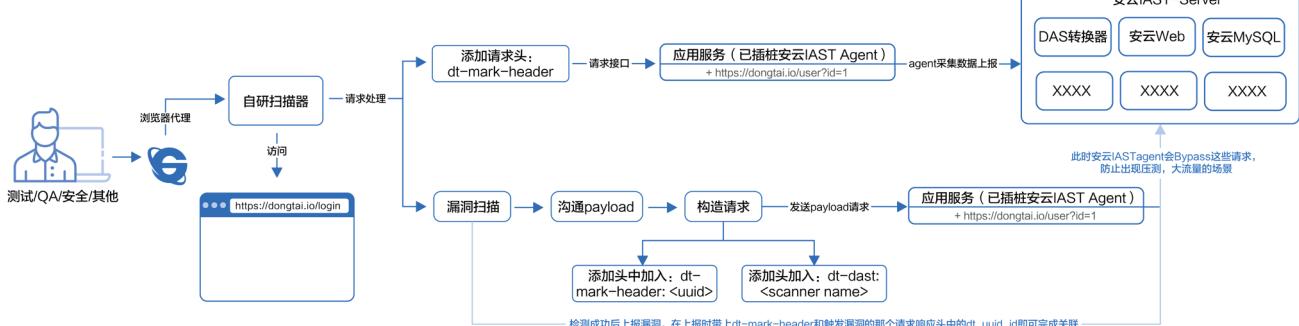
其中 dongtai_vul_type 类型可查看：https://github.com/HXSecurity/Dongtai_USB/blob/bddc914f37f73c139193dc5040f6a41579976c04/xray/model/vultype.go#L3

```
{
    "vul_name": "", #漏洞名 格式为 target+漏洞类型
    "detail": "", #漏洞详情
    "vul_level": "HIGH", #HIGH,MEDIUM,LOW,NOTE 漏洞等级，对应现在安云的4个等级
    "urls": [""], # 黑盒扫描发送的多个 url 地址
    "payload": "", # 黑盒扫描触发漏洞的 payload，可为空
    "create_time": 1679020853, # 时间戳(秒)
    "vul_type": "", #黑盒扫描的漏洞类型
    "request_messages": [{ # 一组扫描对应的所有请求和响应信息
        "request": "",
        "response": ""
    }]
},
#以下为dongtai对接相关信息。
# dt-mark-header 的值
"dt-mark": "",
"target": "", # 原始请求地址
"dt_uuid_id": [""], # 可为多个，为插桩后Agent在响应头提供的uuid_id，需要在dt-request-id处拆分出来
"agent_id": [""],
"dongtai_vul_type": ["sql-injection"], # 安云的漏洞类型，多个类型，为空数组即对应所有调用链漏洞
"dast_tag": "", # 所集成的黑盒扫描器标识
}
```

- 将第一次请求时保存的 dt-mark-header 填入 dt_mark;
- 将保存的 Agent 返回 dt_uuid_id (dt-request-id) 填入，并且使用 Agent 的序号 (id) ;
- 漏洞完成关联。

所以得出一个流程图：

流程三—自研Bypass漏洞关联



(二) 测试 Demo

为了方便快速测试，这里提供一个简单的 Demo。

```

import requests
import uuid
from utils.Timeutils import GetTimestampLimit
from config.log import logger
from config.config import config
import json

def DongTai_IAST_Dastscanner_Report(vul_name,
                                      detail,
                                      vul_level,
                                      urls,
                                      payload,
                                      create_time,
                                      vul_type,
                                      request,
                                      response,
                                      target,
                                      agent_id,
                                      dongtai_vul_type,
                                      dast_tag,
                                      dt_mark,
                                      dt_uuid_id):
    """
    :param vul_name: 漏洞名 格式为 target+漏洞类型 示例: http://127.0.0.1:8080/users?id=1231231存在垂直越权漏洞
    :param detail: 漏洞详情 示例: sql注入是xxxx
    :param vul_level: HIGH,MEDIUM,LOW,NOTE 漏洞等级, 对应现在安云的4个等级 示例: HIGH
    :param urls: 黑盒扫描发送的多个 url 地址 示例: http://127.0.0.1:8080/users?id=1231231
    :param payload: 黑盒扫描触发漏洞的 payload, 可为空 示例: http://127.0.0.1:8080/users?id=1231231
    :param create_time: 时间戳(秒)
    :param vul_type: 黑盒扫描的漏洞类型
    :param request: 一组扫描对应的所有请求信息
    :param response: 一组扫描对应的所有响应信息
    :param target: 原始请求地址
    :param agent_id: 从IAST Agent管理页面查看AGENT id
    :param dongtai_vul_type: 安云的漏洞类型, 多个类型, 为空数组即对应所有调用链漏洞
    :param dast_tag: 所集成的黑盒扫描器标识
    :return:
    """

```

```
splitData = dt_uuid_id.split(".")
get_dt_uuid_id = splitData[1]
get_agent_id = splitData[0]
print(splitData)
print(get_agent_id)
print(get_dt_uuid_id)
data = {
    "vul_name": vul_name, # 漏洞名 格式为 target+漏洞类型
    "detail": detail, # 漏洞详情
    "vul_level": vul_level, # HIGH,MEDIUM,LOW,NOTE 漏洞等级, 对应现在安云的4个等级
    "urls": [urls], # 黑盒扫描发送的多个 url 地址
    "payload": payload, # 黑盒扫描触发漏洞的 payload, 可为空
    "create_time": GetTimeStampLimit(), # 时间戳(秒)
    "vul_type": vul_type, # 黑盒扫描的漏洞类型
    "request_messages": [{ # 一组扫描对应的所有请求和响应信息
        "request": request,
        "response": response,
    }
],
# 以下为dongtai对接相关信息。
"dt_mark": [f'{dt_mark}'], # dt-mark-header 的值 suiiji
"target": target, # 原始请求地址
"dt_uuid_id": [f'{get_dt_uuid_id}'], # 可为空, 可为多个, 为插桩后Agent在响应头提供的
uuid_id, 需要在dt-request-id处拆分出来suiiji
"agent_id": [get_agent_id],
"dongtai_vul_type": [dongtai_vul_type], # 安云的漏洞类型, 多个类型, 为空数组即对应所有调
用链漏洞
    "dast_tag": dast_tag, # 所集成的黑盒扫描器标识
}
logger.log("INFOR", "[+]Report DongTai IAST DAST Data Success!")

# 从配置文件中获取DongTai的服务端上报配置
config = config().get_config()
if not config().check_config():
    exit(0)
logger.log("INFOR", "[+] config Loader success!")

headers = {
    "X-Dongtai-Dast-Vul-Api-Version": "v1",
    "X-Dongtai-Dast-Vul-Api-Authorization": config['DongTai_IAST'][2],
    "Content-Type": "application/json"
}
print(data)
r = requests.post(url=config['DongTai_IAST'][0] + config['DongTai_IAST'][1],
                  headers=headers,
                  data=json.dumps(data))
print(r.text)
return data
```

```

payloadList = [
    'and sleep(3)'
]
url = "http://127.0.0.1:1234/sqli/mybatis/vuln01?username=admin"

cookies = {"session":
"eyJ1c2VyaWQiojI3LCJ1c2Vycm9szwlkj05ox0.ZKOTVA.RgbQKAN7dwUMgfaMHkp2uqHDlCA",
    "JSESSIONID": "4520010E2178366B26487027A84EB15F",
    "remember-me": "true"
"YWRTaw46MTY4OTY4MDU5OTIxNTPkZTF1ZmRmNTcyNWY4YzA5NjMxNTBkMTY4MzUXN2YyNA",
    "XSRF-TOKEN": "bc5aff17-f1fb-4969-a944-1eaa901116d5"}
getuuid = uuid.uuid4().hex
print("原始UUID: ", getuuid)
retuuid = getuuid
print("赋值后的uuid: ", retuuid)

headers = {"User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:109.0) Gecko/20100101 Firefox/114.0",
    "Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8",
    "Accept-Language": "zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2",
    "Accept-Encoding": "gzip, deflate", "Connection": "close", "Referer": "http://127.0.0.1:1234/index",
    "Upgrade-Insecure-Requests": "1", "Sec-Fetch-Dest": "document", "Sec-Fetch-Mode": "navigate",
    "Sec-Fetch-Site": "same-origin", "Sec-Fetch-User": "?1",
    "dt-mark-header": retuuid
}
print("第一次请求: ", headers)
one_request = requests.get(url, headers=headers, cookies=cookies)
print("第一次请求响应码: ", one_request.status_code)
print("第一次请求响应头: ", one_request.headers)

for payload in payloadList:
    print("payload is : ", payload)
    url = "http://127.0.0.1:1234/sqli/mybatis/vuln01?username=admin " + payload
    headers = {"User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:109.0) Gecko/20100101 Firefox/114.0",
        "Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8",
        "Accept-Language": "zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2",
        "Accept-Encoding": "gzip, deflate", "Connection": "close", "Referer": "http://127.0.0.1:1234/index",
        "Upgrade-Insecure-Requests": "1", "Sec-Fetch-Dest": "document", "Sec-Fetch-Mode": "navigate",
        "Sec-Fetch-Site": "same-origin", "Sec-Fetch-User": "?1",
        "dt-mark-header": retuuid,
        "dt-dast": "SecurityTeam"
    }

```

```
print("扫描请求头: ", headers)
print("扫描URL: ", url)
two_request = requests.get(url, headers=headers, cookies=cookies)
print("扫描请求结果: ", two_request.headers)
DongTai_IAST_DastScanner_Report(vul_name=f"{url} SQL注入漏洞",
                                   detail="软件使用来自上游组件的外部影响的输入来构造全部或部分SQL命令, 但不会中和或不正确地中和了特殊元素, 这些特殊元素在将其发送到下游组件时可能会修改预期的SQL命令。",
                                   vul_level="HIGH",
                                   urls=url,
                                   payload=payload,
                                   create_time="",
                                   vul_type="sql-injection",
                                   request="source_request",
                                   response="source_resp",
                                   target=url,
                                   dongtai_vul_type="sql-injection",
                                   dast_tag="SecurityTeam",
                                   dt_mark=retuuid,
                                   dt_uuid_id=one_request.headers['dt-request-id'],
                                   agent_id=1)
```

语义描述 :上述代码先使用生成一个随机的 UUID.randomUUID().hex, 加入到请求头中, 随后不带上 dt-dast 头, 这样 IAST 的 Agent 就不会 bypass 这次请求, 并且 Agent 会记录此次的随机生成的 UUID, 随后开始 payload 扫描, 此时会带上第一次请求中随机生成的 uuid, 并且在请求头中带上 `dt-dast: securityTeamScanner` , 这样 Agent 会 bypass 这次的请求, 解决了压测、大流量导致性能消耗的情况。

最后通过上报漏洞时关联漏洞, 详情看 DongTai_IAST_DastScanner_Report 函数的实现, 需要将生成的 UUID 填入 dt_mark, 将存在漏洞那个请求的响应头中 dt-request-id 分割。 (2.3e5b258b94ee45fd8bad048 8d0c9abdd) 其中的 2 是 Agent 的序号, 后续是请求 request-id, 使用 split 分割后得到下面的结果:

```
['2', 'c0b6c6d5bd2d4081b019cc15b1a9fa7c']
```

此时只需要下标 0 给到 Agentid, 下标 1 给到 dt_uuid_id, 随后上报漏洞即可完成关联。

(三) 测试与效果展示

此时 IAST 能够提供到代码行的漏洞分析能力，并且自研扫描器能够提供一个 Payload。

The screenshot displays two main sections of the IAST platform interface:

- Top Section (Project Overview):**
 - Project: scanner_test_1 (JAVA)
 - Scanning Mode: 插桩模式 (Instrumentation Mode)
 - Responsible Person: admin
 - Last Scan Time: 2023.07.04 19:46:53
 - Version: V1.0
- Middle Section (Vulnerability Analysis):**
 - Filter: 重置全部 (Reset All)
 - Severity: 高危 (High Risk)
 - Associated Project: scanner_test_1
 - Vulnerability Type: SQL注入 (SQL Injection)
 - Status: 已确认 (Confirmed)
 - Description: /sqli/mybatis/vuln01 GET 出现 SQL注入位置:GET
javax.servlet.ServletRequest.getParameterValues
java.sql.Connection.prepareStatement
scanner_t... Apache Tomcat/8.5.85
关联漏洞1个
 - Table: 扫描漏洞 ID | 扫描漏洞类型 | payload | 操作
1 | sqli | and sleep(3) | 详情|关联
- Bottom Section (Payload Configuration):**
 - URL: http://127.0.0.1:1234/sqli/mybatis/vuln01?username=admin and sleep(3) SQL注入漏洞
 - Basic Information:

危害级别: 严重	首次出现: 2023.07.04 19:52:48	漏洞类型: sql-injection
项目名称: scanner_test_1	项目版本: V1.0	推送来源: SecurityTeam
payload: and sleep(3)		
 - Details:
 - 漏洞描述: 软件使用来自上游组件的外部影响的输入来构造全部或部分SQL命令，但不会中和或不正确地中和了特殊元素，这些特殊元素在将其发送到下游组件时可能会修改预期的SQL命令。
 - Data Flow:
 - Request: source_request
 - Response: source_resp

```

原始UUID: 0aabbe5f537644d3a53ef7c8139b40a8
赋值后的UUID: 0aabbe5f537644d3a53ef7c8139b40a8
第一次请求: {'User-Agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:109.0) Gecko/20100101 Firefox/114.0', 'Accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8'}
第一次请求响应码: 200
第一次请求响应头: {'DongTai': 'v1.11.0', 'dt-request-id': '2.11a53b7a4815423d857880c6469a231c', 'X-Application-Context': 'application:1234', 'X-Content-Type-Options': 'nosniff', 'X-XSS-Protection': '1; mode=block'}
payload is : and sleep(3)
扫描请求头: {'User-Agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:109.0) Gecko/20100101 Firefox/114.0', 'Accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8'}
扫描URL: http://127.0.0.1:1234/sqli/mybatis/vuln01?username=admin and sleep(3)
扫描请求结果: {'DongTai': 'v1.11.0', 'dt-request-id': '2.1b8e712dd8d4f9998c8ee44fa919f199', 'X-Application-Context': 'application:1234', 'X-Content-Type-Options': 'nosniff', 'X-XSS-Protection': '1; mode=block', 'vul_name': 'http://127.0.0.1:1234/sqli/vuln01?username=admin and sleep(3) SQL注入漏洞', 'detail': '软件使用来自上游组件的外部影响的输入来构造全部或部分SQL命令，但不会中和或不正确地中和了特殊元素。这些特殊元素在将其发送到下游时可能被忽略或以意外方式解释。', 'status': 201, "msg": "\u064cd\u04f5c\u06210\u0529f"}
19:46:47,844 [INFO] Scanner_test:66 - [*]Report DongTai IAST DAST Data Success!
19:46:47,847 [INFO] Scanner_test:72 - [*] Config Loader Success!
    
```

(四) Agent bypass

详细代码可以看到 io.dongtai.iast.core.handler.hookpoint.SpyDispatcherImpl#collectHttpRequest

<https://github.com/HXSecurity/DongTai-agent-java/blob/09a90e9a651eb89d84f6166d149f07c4853d1aed/dongtai-core/src/main/java/io/dongtai/iast/core/handler/hookpoint/controller/impl/HttpImpl.java>

```

no usages - lostsnow
@Override
public void collectHttpRequest(Object obj, Object req, Object resp, StringBuffer requestURL, String requestURI,
                               String queryString, String method, String protocol, String scheme,
                               String serverName, String contextPath, String remoteAddr,
                               boolean isSecure, int serverPort, Enumeration<?> headerNames) {
    try {
        ScopeManager.SCOPe_TRACKER.getPolicyScope().enterAgent();

        if (!EngineManager.isEngineRunning()) {
            return;
        }

        HttpImpl.createClassLoader(req);

        Map<String, String> headers = HttpImpl.parseRequestHeaders(req, headerNames);
        lostsnow
        Map<String, Object> requestMeta = new HashMap<String, Object>() {{
            put("requestURL", requestURL);
            put("requestURI", requestURI);
            put("queryString", queryString);
            put("method", method);
            put("protocol", protocol);
            put("scheme", scheme);
            put("serverName", serverName);
            put("contextPath", contextPath);
            put("remoteAddr", "0:0:0:0:0:0:1".equals(remoteAddr) ? "127.0.0.1" : remoteAddr);
            put("secure", isSecure);
            put("serverPort", serverPort);
            put("headers", headers);
            put("replay-request", !StringUtils.isEmpty(headers.get("dongtai-replay-id")));
        }};
        HttpImpl.solveHttpRequest(obj, req, resp, requestMeta);
    } catch (Throwable e) {
        DongTaiLog.warn(ErrorCode.get("SPY_COLLECT_HTTP_FAILED"), ...arguments: "request", e);
    } finally {
        ScopeManager.SCOPe_TRACKER.getPolicyScope().leaveAgent();
    }
}
    
```

io.dongtai.iast.core.handler.hookpoint.controller.impl.HttpImpl#parseRequestHeaders

```

1 usage  lostsnow
public static Map<String, String> parseRequestHeaders(Object req, Enumeration<?> headerNames) {
    Map<String, String> headers = new HashMap<String, String>( initialCapacity: 32);
    Method getHeaderMethod = ReflectUtils.getDeclaredMethodFromSuperClass(req.getClass(),
        methodName: "getHeader", new Class[]{String.class});
    if (getHeaderMethod == null) {
        return headers;
    }
    while (headerNames.hasMoreElements()) {
        try {
            String key = (String) headerNames.nextElement();
            String val = (String) getHeaderMethod.invoke(req, key);
            if ("content-type".equalsIgnoreCase(key)) {
                key = "Content-Type";
            } else if (HEADER_DAST_MARK.equalsIgnoreCase(key)) {
                key = HEADER_DAST_MARK;
            } else if (HEADER_DAST.equalsIgnoreCase(key)) {
                key = HEADER_DAST;
            }
            headers.put(key, val);
        } catch (Throwable ignore) {
        }
    }
    return headers;
}

```

```

public class HttpImpl {
    3 usages
    private static IastClassLoader iastClassLoader;
    6 usages
    public static File IAST_REQUEST_JAR_PACKAGE;
    3 usages
    public static final String HEADER_DAST_MARK = "dt-mark-header";
    3 usages
    public static final String HEADER_DAST = "dt-dast";
}

```

```

    }

    Boolean isReplay = (Boolean) requestMeta.get("replay-request");
    if (isReplay) {
        EngineManager.ENTER_REPLAY_ENTRYPOINT.enterEntry();
    }
    // todo Consider increasing the capture of html request responses
    if (ConfigMatcher.getInstance().disableExtension((String) requestMeta.get("requestURI"))) {
        return;
    }
    if (ConfigMatcher.getInstance().getBlackUrl(requestMeta)) {
        return;
    }

    try {
        boolean enableVersionHeader = ConfigBuilder.getInstance().get(ConfigKey.ENABLE_VERSION_HEADER);
        String dastHeader = ((Map<String, String>) requestMeta.get("headers")).get(HOLDER_DAST);
        String dastMarkHeader = ((Map<String, String>) requestMeta.get("headers")).get(HOLDER_DAST_MARK);
        if (enableVersionHeader || dastHeader != null || dastMarkHeader != null) {
            Method setHeaderMethod = ReflectUtils.getDeclaredMethodFromSuperClass(resp.getClass()
                .getDeclaredMethods(), "setHeader", new Class[]{String.class, String.class});
            if (setHeaderMethod != null) {
                if (enableVersionHeader) {
                    String versionHeaderKey = ConfigBuilder.getInstance().get(ConfigKey.VERSION_HEADER_KEY);
                    setHeaderMethod.invoke(resp, versionHeaderKey, AgentConstant.VERSION_VALUE);
                }
                if (dastMarkHeader != null) {
                    String reqId = String.valueOf(EngineManager.getAgentId()) + "."
                        + UUID.randomUUID().toString().replaceAll("\\-", "_");
                    setHeaderMethod.invoke(resp, "dt-request-id", reqId);
                }
            }
            if (dastHeader != null) {
                return;
            }
        }
    } catch (Throwable ignore) {
    }
}

```

© io.dongtai.iast.core.handler.hookpoint.controller.impl.HttpImpl
public static final String HEADER_DAST_MARK = "dt-mark-header"
© dongtai-core

04 扩展：IAST 与白盒的联动

在安全建设的任何一个环节中，没有任何一个工具能够保证全局所有业务的安全，也没有任何一个安全产品能够代替其他安全产品，不同的安全产品在不同的环节都有独有能力，将这些能力进行整合、互补，大大地提高安全建设的效率。

在这样的背景下，交互式应用安全测试（IAST）与白盒扫描的联动应用成为一种有力的策略。白盒扫描是一种静态代码分析方法，它深入分析源代码，检测潜在的安全漏洞。IAST 则在应用运行时检测漏洞，提供实际交互的情境，使其能够更准确地找到漏洞。两者的结合能够弥补彼此的不足，形成一种强大的安全防线。

这种联动方式的作用是多方面的：

1. **全面性与准确性提升：** 联动 IAST 和白盒扫描，可以从不同角度全面评估应用程序的安全性。白盒扫描检测源代码中的漏洞，而 IAST 在应用运行时发现潜在问题，两者结合能够提供更准确的漏洞信息。
2. **早期发现与修复：** 白盒扫描通常在开发早期进行，IAST 在运行时持续监测。通过在开发阶段和生产阶段同时运用这两种方法，可以更早地发现漏洞并及时修复，降低漏洞被攻击的风险。

3. 降低误报率：传统的静态代码分析可能会产生大量误报，而 IAST 在实际交互中能够排除部分误报。联动后，可以帮助开发团队更快速地确认和解决真正的漏洞。

联动 IAST 与白盒扫描的结果可能是更加强大的应用程序安全防护体系。开发团队可以利用两种方法的信息，有针对性地进行漏洞修复和代码改进。此外，这种联动还有助于提高团队对于应用程序的整体安全认知，促使开发、安全和运维团队更紧密地合作，共同保障应用的安全性和稳定性。

IAST 未来的百种可能： IAST 未来主流技术路线浅析

基调听云高级开发工程师 陈继成

01 IAST 技术发展的背景

从 Gartner 首次提出交互式应用安全测试（IAST）技术以来，IAST 作为保护应用程序安全的重要方法之一，一直处于不断演进的状态。IAST 是一种先进的安全测试方法，旨在检测应用程序中的漏洞和安全威胁，同时提供实时的反馈和准确的报告，有助于在应用上线前降低应用程序的安全风险。

随着信息技术的迅猛发展，经过多年的技术积累，IAST 在应用程序安全测试中取得了显著的进步，但客观上仍存在一些挑战和技术难点：

1. 性能消耗：IAST 需要通过插桩实时监视和分析应用程序运行时的行为。然而任何插桩行为都必然对应用的性能产生影响，这可能导致应用性能下降，特别是在高负载条件下。
2. 漏洞验证难：使用 IAST 本身的检测逻辑对漏洞进行验证时，难度较大且效果较差。在实际场景中难以做到 DAST 的准确率，同时其覆盖的漏洞场景也会比 DAST 更少。
3. 部署困难：集成 IAST 工具到现有的开发和部署流程可能会比较复杂，需要专业知识和资源。

02 性能、准度、适配？IAST 超进化

（一）代码性能优化

IAST 最被诟病的一点就是应用引入 IAST 之后对性能影响过大，这种性能主要分为两个方面：

1. 内存暴增：

因为 IAST 在运行的时候需要在内存中存储污点传播链的上下文，比如污点池以及污点相关的方法的参数、返回值等等，所以当并发较大或者被测试应用的单个请求比较重的时候，IAST 对应的上下文也会被迫随之增长。

虽然这种增长是无法避免的，但是是可以想办法尽量抑制增长速度的，比如提升代码质量，优化代码中使用到的数据结构，尽量避免 IAST 自身在内存中持有太多重复的数据；同时及时释放掉无用的污点对象等。

2.CPU 飙升：

IAST 的检测原理是对被测试应用原有的方法进行 Hook，在方法原有的字节码基础上增加一部分字节码指令，当被 Hook 的函数较多时 JVM 要执行的字节码指令就会随之增多。单个的方法调用可能只会产生不起眼的微小损耗，但是当被 Hook 的方法被很高频地调用时，这种数千万次的调用叠加起来就是一个很难被忽视的损耗了：CPU 利用率会升高，程序运行速度肉眼可见地变慢，针对这种问题其实没有太好的办法，这是 Agent 类产品的通病，不过即使如此仍有一些可以优化的点，比如：

1.Agent 代码本身尽可能避免使用到锁，避免因为锁竞争带来的损耗甚至死锁；

2.Agent 的代码里尽量不要再去调用会被 Transform 的类，比如 Java 中 StringBuilder 通常会作为传播节点被 Hook，但是因为这个类太常用了，Agent 内部很可能也会用到这个类来处理字符串，这就会带来一些不必要的性能损失；

3.Agent 的 Hook 调用的代码里尽量不要有重量级操作，任何一丁点的延迟被放大几千万倍都会是很严重的性能问题，对内存和 CPU 的使用要精打细算。

上面的问题要落实下来需要完善研发流程，代码提交到仓库落实 Review 机制，至少两个人审查过的代码才可以被 merge 到仓库，与其等出了问题花时间排查半天才发现是一些很愚蠢的错误后悔得拍大腿，倒不如把时间拿来控制代码质量，把问题尽可能扼杀在源头。

（二）污点传播能力提升

对于 IAST 的 Hook 的实现，大致有全量 Hook 和部分节点 Hook 的方式。因为全量 Hook 的性能损耗过大，此方案已经逐渐被弃用，安云采用的是 Hook 污点传播路径上的关键传播节点的方式，虽然这样能够带来极高的性能提升，但是相对而言也可能会导致污点的传播链路因为各种情况中断，污点从 Source 产生没办法传播到 Sink，未来在这一块还有很大的提升空间，提升的思路有很多，后文也将介绍一种基于 SAST 提升污点传播能力的思路。

(三) 主动验证 (API 安全)

目前主流的 IAST 产品基本都有 API 采集的功能，但是基本都是只做到从 Agent 采集了 API，上报到 Web 管理页面进行展示就结束了。这一块其实是可以深入做下去的，并且对 IAST 的漏洞检出能力有很大的提升。当我们把从 Agent 采集到的 API 信息标准化，比如对齐到 OpenAPI 的接口标准，那么当我们有一份符合 OpenAPI 规范的 API JSON 时我们就可以发挥想象力做非常多的事情：

1. 比如从 OpenAPI 的格式自动生成请求做 API 级别的主动发包触发污点传播，目前网络上也有一些根据 OpenAPI 的 JSON 生成请求的工具，人工测试总会有触发不到的分支，Fuzz 覆盖分支更全更可靠。

2. 再比如 IAST 检测到漏洞之后可以根据污点向前追溯到入口对应的 HTTP 接口，根据漏洞类型自动生成 Payload 做最小代价的精准验证漏洞，既不会产生不必要的流量，又能自证漏洞是否误报，甚至可以自动生成 Payload 的 curl 让安全团队自行从其它渠道验证，漏洞会更有说服力，这对于安全团队在企业内推广 IAST 也会有很大帮助。

当然鱼跟熊掌很难兼得，要做主动验证一个比较麻烦的点是如何绕过认证，目前的认证框架繁杂很难做适配，只适配一个 Shiro 是远远不够的，需要找到一种通用的认证适配方式。

(四) 漏洞类型的完善与迭代

随着技术的发展，不断有新的漏洞的类型被发现，如果死守着传统的漏洞去挖掘很快便会被时代所抛弃，IAST 的污点检测不仅需要适配传统的漏洞类型，也要与时俱进不断迭代适配新发现的漏洞类型，比如适配一些云原生的安全漏洞。

(五) 各种框架的适配与迭代 (主流框架、客制化框架、历史版本)

IAST 这种 Agent 类产品比较头疼的问题就是兼容性，兼容性是在 IAST 迭代过程中始终需要关注的问题。兼容性又大致分为两个方面：

一方面是对框架种类的支持，种类的支持又分为对主流框架的适配和对比较小众但是又必须支持的框架的适配。

1. 对于主流框架的适配很好理解，比如主流的 MQ，Kafka、RabbitMQ 之类的，比如对 Redis、MySQL 的驱动等等进行适配，并且随着技术的发展迭代会不断有新的主流框架产生，这些都需要及时做适配，这是一项没有终点的工作。

2. 对于小众框架的适配这个就比较头疼了，客户总是喜欢从主流框架 fork 一个分支搞二次开发，然后修改得面目全非让你没办法把已有的经验和方案直接迁移使用，或者客户干脆从零开始另起炉灶造火箭，打造客户自己内部使用的框架，这种框架可能都不对外开放，适配更是难上加难，但是客户的框架你如果不适配便没办法在他那里推广开来，你适配吧成本又非常高，这对整个 IAST 行业而言都是一个头疼的问题。

针对这种问题笔者抛出一个思路，虽然客户内部的框架千变万化，但变化的通常都是比较高层的给开发者直接使用的 API，底层都是要遵循一些规范的。比如客户自己开发的 Web 容器，它也要遵循 J2EE 的 Servlet 规范；比如客户自己开发的 MQ 框架，那它大概率也要遵循 JMS 规范。你可以造轮子，但不能乱造轮子，Java 中有相当多的 JSR (Java Specification Requests) 规范用来定制规范标准，造的轮子要遵循这些规范才能跟 Java 世界的其他框架适配得上从而融入整个 Java 生态系统，如此我们开发 IAST 便不必适配具体的框架，那会累死人，我们适配 JSR 规范，只要客制化框架遵循 JSR 规范便能够兼容。

第二个方面是对单个框架的历史版本的兼容适配。即使是同一个框架，随着它的版本迭代，可能也是不兼容自己的。比如某个框架，2.3 版本有个方法你觉得挺好把它作为污点传播节点 Hook 上了，但是 2.4 版本人家框架的开发者咔嚓一下把这个方法给去掉了，或者改名了，那你 IAST 的污点传播路径可就断了，这都算好的，甚至在 Transform 的时候会因为 ASM 织入字节码的兼容性问题程序整个崩掉，你说本来应用跑得好好的，加了你的 IAST 之后直接 crash 了，这就不好跟客户交代了。这种问题也很难说把每个版本都检查一遍，因为我们可能有数十个框架要适配，每个框架可能有数百个版本，这样一相乘，得到的数字几乎是人力不可为！针对这个问题笔者抛砖引玉提出两个思路：

1. 第一个思路是大力出奇迹，虽然要测试的版本足够多，但是只要我找到能够自动化测试的办法堆机器就是了，比如开发一个专门用于 Agent 测试的框架，针对要适配框架的 Source、Sink、传播节点等写测试用例，测试框架会枚举要适配框架的每一个版本号相乘，最终就能把有问题的版本号找出来并手动兼容修复。
2. 第二个思路就是反着来，比如我们来看第一种方案，一个框架可能有数百个版本，但是通常情况下其中仅仅有几个版本是不兼容发生了变更的，那么相当于大部分算力都被浪费掉了，那么我们就倒着来，通过分析要兼容的框架被 Hook 的类的历史变化情况，我们能够找到几个 Class 字节码发生变化的关键版本，那么我们只需要对这些关键版本做测试兼容，像玩消消乐一样消除掉重复工作量，如此便把不可能的事情变为了可能。

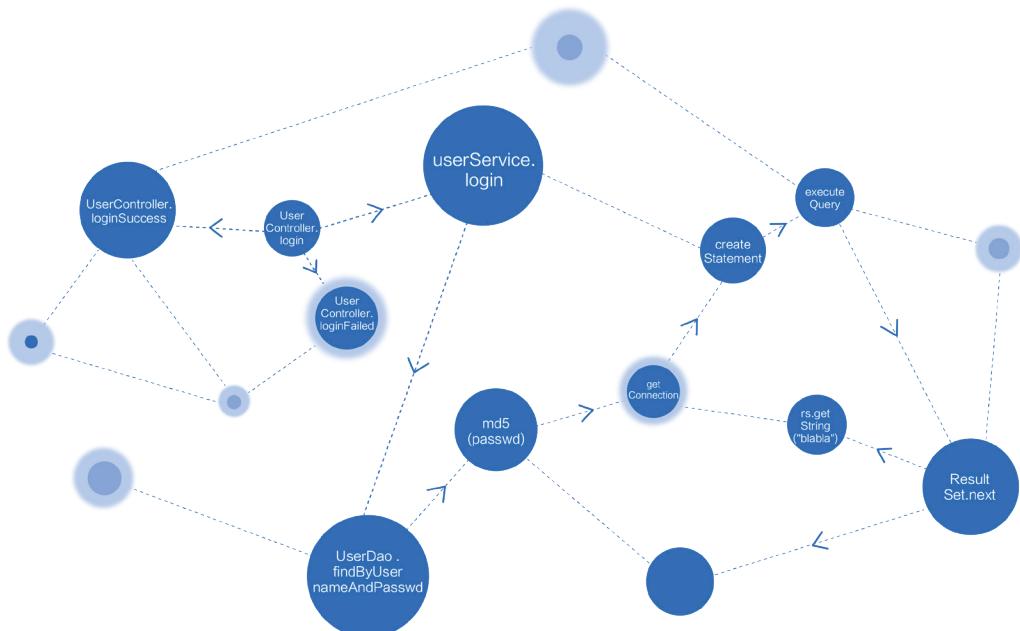
03 1+1>2 ? IAST 身上的无限可能

未来，除了上文阐述的 IAST 自身能力的逐步完善，通过与其他产品联动的方式，或许也是一个放大 IAST 能力的好办法。IAST+DAST 已经有安云的较为成熟的试验，并且目前也已有部分高质量文章进行了详细的介绍。那我们可否在这一思路上再做一些拓展，思考一下 IAST 与 SAST、RASP 等其他安全工具的可能性呢？

(一) IAST + SAST

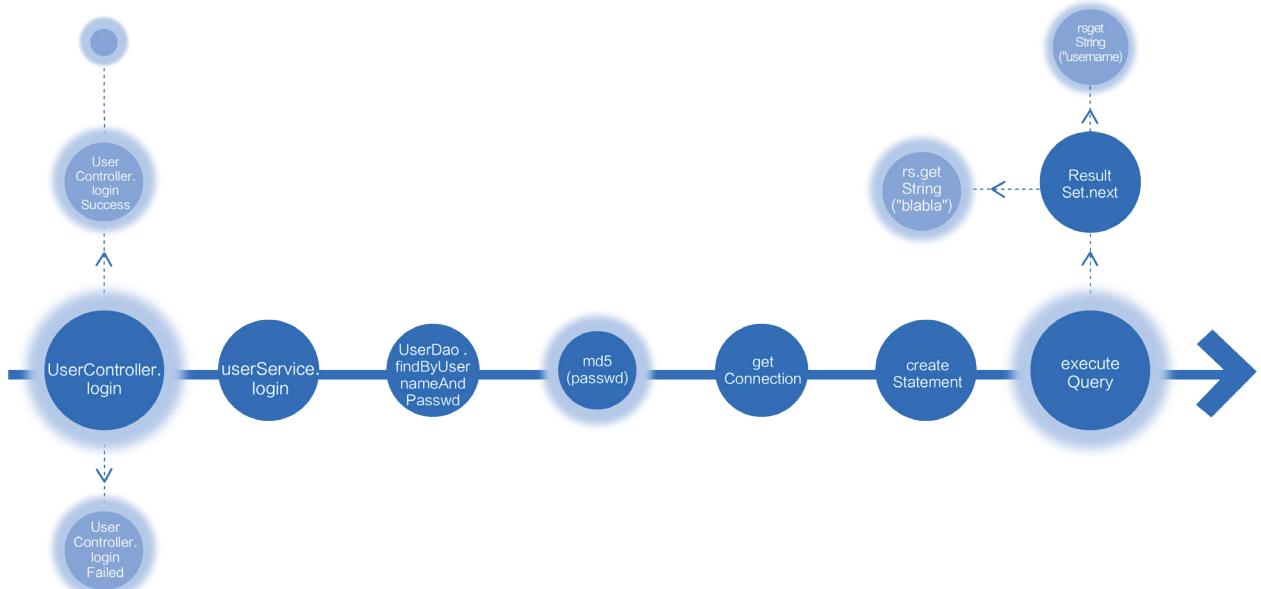
IAST 对漏洞检出的流程和 SAST 很像，都是基于污点跟踪技术，不同的是 IAST 能够在 SAST 的基础上追踪一些运行时产生的更为具体的数据，从而利用这些具体的数据尽可能降低追踪污点时会存在的误报率。但 SAST 也拥有 IAST 所不具备的优势，比如 IAST 出于性能考虑通常不会全量 Hook，这样就会导致污点追踪时传播链路常常会断掉（比如污点的值或者类型等发生了变化），这样追踪污点从 Source 到 Sink 的路上就经常跟丢，从而产生了很多漏报，而 SAST 则能够追踪污点传播时的完整的链路，针对这个问题笔者抛出一个结合 SAST 来优化 IAST 的污点传播链路的思路。

让我们来把问题的模型描述得更清楚一些，从 SAST 的角度来看应用程序的代码，把符号（比如方法、变量等）看做是图上的一个节点，把符号之间的引用关系（比如在方法中调用了另一个方法）看做是图上的一条有方向的线，这条线连接了图上的两个节点，则应用程序的代码可以整个展开为一张非常大的有方向的图（并不是有向无环图，可能存在环路，比如递归调用）。



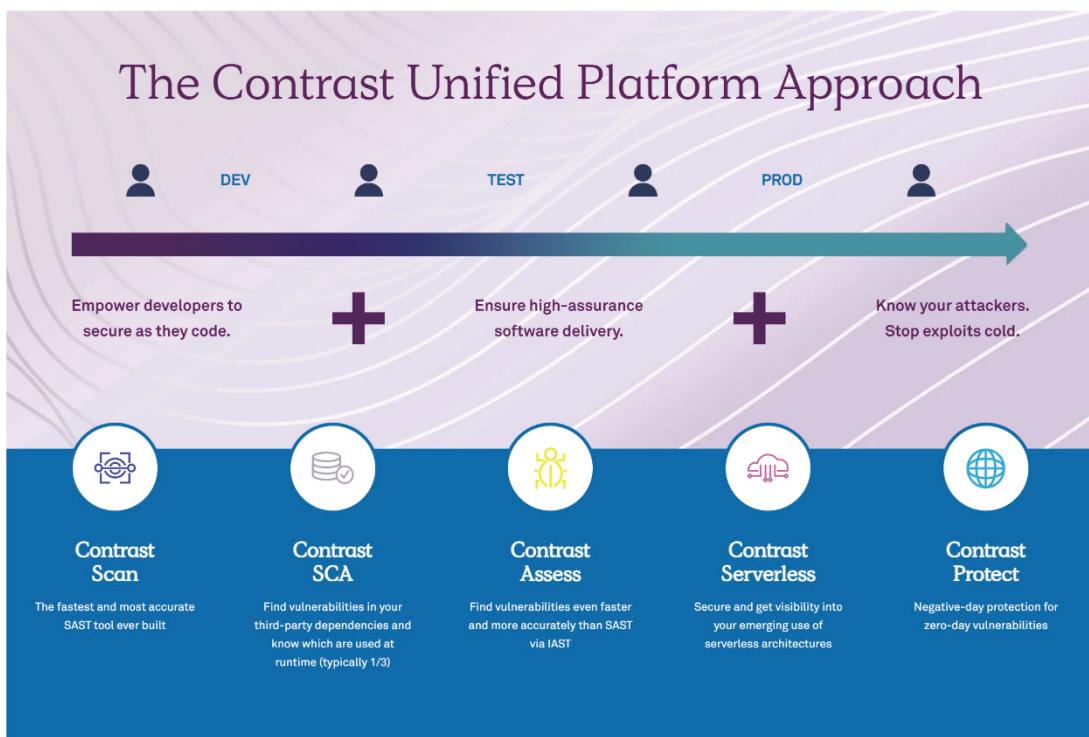
然后我们把 IAST 代入到这张 SAST 的有向图上，污点的 Source 可以对应到这张图上的一些顶点，污点的 Sink 也可以对应到一些顶点，而 Source 节点到 Sink 节点之间的传播链路就是有向图上这两个顶点之间的所有通路，如此我们便可以通过 SAST 的方式遍历图寻找所有 Source 到所有 Sink 之间的通路，这是第一步，这一步之后我们便得到了一些路径的链表集合（或者是子图，这里为了尽可能简单我们拆分为更细粒度的链表），这些路径就是 IAST 中的污点传播路径，路径的头节点是 Source 节点，尾节点是 Sink 节点。

然后第二步，我们依此对路径集合中的每条路径的每一步节点做检查，如果发现我们要追踪的污点在这一步发生了可能会导致传播链路断掉的操作，比如类型由字符串 String 变为了字节数组 byte[], 我们便将这一个节点标记为传播路径的关键节点，自动生成 IAST 的传播链路对应的传播节点的 Hook 规则，这样我们就能够得到一些 IAST 的 Hook 规则将其导入到我们的 IAST 系统中，如此便能实现借助 SAST 技术自动挖掘 IAST 污点传播节点。



(二) IAST + RASP

IAST 通常跑在测试环境，RASP 通常跑在生产环境，RASP 比 IAST 多了拦截的功能，RASP 对代码质量要求较高，因为 Agent 类产品很容易就会导致应用程序 crash，看起来它们是互斥的，其实 IAST 和 RASP 技术是可以结合起来的，这也是未来 IAST 发展的一个方向，IAST 领域的国际头部厂商 Contrast 推出的 Secure Code Platform，已经成功实现了将 SCA、IAST、SAST 以及 RASP 整合进一个整体的安全解决方案当中。



(二) IAST + Hadoop + HBase + Spark

IAST 对调用链的分析分为 Agent 端处理和 Server 端处理。对于 Server 端处理，通常是 Agent 端收集调用链上报到 Server 端，Server 收集到调用链之后再做其它处理比如检测漏洞，从这个角度来看，与其说 IAST 是安全类产品，不如说 IAST 更像是一款数据分析类产品。

基于收集的调用链能够做很多事情，目前大部分 IAST 类产品都没有发挥出调用链的潜力，IAST 能够比 APM 更好的观测应用程序的性能，追踪链路的调用，甚至分析代码的可达性，热点代码等等。2023 年 8 月，国际 APM 头部厂商 New Relic 也宣布基于他们在 APM 领域的技术积累，发布 IAST 产品，通过 APM 与 IAST 的能力结合，IAST 可以适应更多场景、提供更多能力。

(三) IAST + AI + ChatGPT

当前 AI 技术已经发展到一个新台阶，基于大模型技术的 ChatGPT 最擅长的就是问答，而目前 IAST 的一个比较大的问题就是陷入太多的人工服务。需要人工去跟踪客户的部署、需要人工去解答客户的各种问题、需要人工去验证检出的各种漏洞，而人工就意味着高昂的成本，笔者相信，在 IAST 安装部署与运营过程中引入基于私有数据训练的私有大模型，将极大地消除 IAST 推广的障碍，降低 IAST 安装部署的时间与经济成本。

EPILOGUE

结语

本期《IAST 金融行业专刊》从多个维度深入剖析了 IAST 在金融行业的技术发展、政策要求、实践案例以及未来发展趋势。作为应用安全新晋厂商，我们深知单体的力量与知识储备是有极限的，我们非常荣幸能够联合证通、火线安全等单位共同编写本专刊，为金融行业系统性落地 IAST 提供力所能及的贡献。

随着金融业务的数字化与网络化发展，金融行业面临着越来越多的网络安全风险。在这个背景下，IAST 作为一种新兴的应用安全技术，具有独特的优势和巨大的潜力。通过与真实甲方的对接，我们深入了解到了 IAST 在应对当前安全挑战方面的作用和价值。

在此，特别感谢证通黎总等行业资深专家的大力支持和合作，感谢他们与我们一同分享经验、探讨问题、推动行业的发展。同时，我们也要感谢所有参与本期专刊的作者、专家，正是他们的深度探索和积极贡献，使得本期专刊能够成为关于 IAST 在金融行业应用的重要参考资料。

我们将继续与各界合作伙伴携手前行，在 IAST 技术的研发、实践和创新上不断努力，为保障金融行业信息安全作出更大的贡献。

谢谢大家！

DISCLAIMER

法律声明

此专刊为基调听云出品，本专刊的内容仅供金融行业用户参考和信息交流之目的，并不构成任何法律、投资或其他专业建议。

专刊中的文字、图片、表格等版权均为基调听云所有，任何组织、个人未经基调听云及免费咨询授权，不得转载、更改或者以任何方式传送、复印、派发该专刊内容，违者将依法追究法律责任。

本专刊中可能包含有关法律、合规或知识产权的信息，但这些信息不能被解读为法律咨询或专业意见。读者在使用这些信息时，应该寻求适当的法律咨询或专业建议。

本专刊中可能包含未来展望性陈述，这些陈述基于我们的判断和预测，但是不能被视为对未来事实的保证。实际结果可能因众多因素而有所不同，我们不承担与此相关的责任。

如有任何疑问或需要进一步咨询，请联系我们。

电话：400-688-9582

邮箱：info@tingyun.com



北京基调网络股份有限公司

电话：400-688-9582

邮箱：info@tingyun.com