## 应用安全测试技术DAST、SAST、IAST对比分析

**一、全球面临软件安全危机**

2010年，大型社交网站rockyou.com被曝存在SQL注入漏洞，黑客利用此漏洞获取到3200万用户记录（包括E-mail、姓名及明文形式的密码）。

2015年，英国电话和宽带供应商TalkTalk被一名15岁的黑客利用SQL注入漏洞进行攻击，四百万TalkTalk客户的姓名、地址、出生日期、和信用卡/银行详细信息被黑客窃取。

2018年，台湾一男子利用花旗银行信用卡业务系统漏洞，刷卡消费达6300余万元（新台币约合人民币1345万元），花旗银行已通过司法途径，向该名客户求偿。

软件技术的发展与应用伴随着巨大的安全危机，解决软件漏洞问题是软件开发从业者和安全从业者们迫在眉睫的任务。

**二、什么是Web应用安全测试技术？**

为了发现软件的漏洞和缺陷，确保Web应用程序在交付之前和交付之后都是安全的，就需要利用Web应用安全测试技术识别Web应用程序中架构的薄弱点和漏洞，并且必须赶在网络黑客找到和利用它们之前。

Web应用安全测试技术经过多年的发展，目前业界常用的技术主要分为3大类别。

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。

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。

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。

本文主要分析这3种技术的实现原理、优劣势对比以及应用场景。

**三、DAST安全测试技术**

DAST是一种黑盒测试技术，是目前应用最广泛、使用最简单的一种Web应用安全测试方法，安全工程师常用的工具如AWVS、AppScan等就是基于DAST原理的产品。

**1. 实现原理**

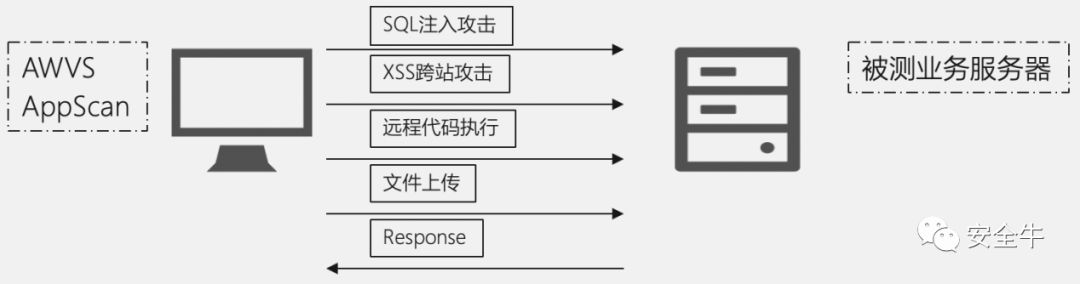


图 1：DAST原理

1）通过爬虫发现整个 Web 应用结构，爬虫会发现被测Web程序有多少个目录，多少个页面，页面中有哪些参数；

2）根据爬虫的分析结果，对发现的页面和参数发送修改的 HTTP Request 进行攻击尝试（扫描规则库）；

3）通过对于 Response 的分析验证是否存在安全漏洞。

**2. DAST优劣势分析**

DAST这种测试方法主要测试Web应用程序的功能点，测试人员无需具备编程能力，无需了解应用程序的内部逻辑结构，不区分测试对象的实现语言，采用攻击特征库来做漏洞发现与验证，能发现大部分的高风险问题，因此是业界Web安全测试使用非常普遍的一种安全测试方案。DAST除了可以扫描应用程序本身之外，还可以扫描发现第三方开源组件、第三方框架的漏洞。

从工作原理也可以分析出，DAST一方面需要爬虫尽可能的把应用程序的结构爬取完整，另一方面需要对被测应用程序发送漏洞攻击包。现在很多的应用程序含有AJAX页面、CSRF Token页面、验证码页面、API孤链、POST表单请求或者是设置了防重放攻击策略，这些页面无法被网络爬虫发现，因此DAST技术无法对这些页面进行安全测试。DAST技术对业务分支覆盖不全，即使爬到一个表单，要提交内容，服务端对内容做判断，是手机号码则进入业务1，不是手机号码进入业务2，爬虫不可能知道这里要填手机号码，所以业务分支1永远不会检测到。

另外DAST必须发送漏洞攻击包来进行安全测试，这就需要有安全专家不断更新漏洞扫描插件，而且这种测试方式会对业务测试造成一定的影响，安全测试的脏数据会污染业务测试的数据。

DAST的测试对象为HTTP/HTTPS的Web应用程序，对于IOS/Android上的APP也无能为力。

DAST发现漏洞后会定位漏洞的URL，无法定位漏洞的具体代码行数和产生漏洞的原因，需要比较长的时间来进行漏洞定位和原因分析，这使得DAST不太适合在DevOps的开发环境中使用。



图 2：DAST优势与劣势

**四、SAST安全测试技术**

超过50%的安全漏洞是由错误的编码产生的，开发人员一般安全开发意识和安全开发技能不足，更加关注业务功能的实现。想从源头上治理漏洞就需要制定代码检测机制，SAST是一种在开发阶段对源代码进行安全测试发现安全漏洞的测试方案。

**1. 实现原理**



图 3：SAST原理

1） 首先通过调用语言的编译器或者解释器把前端的语言代码（如JAVA，C/C++源代码）转换成一种中间代码，将其源代码之间的调用关系、执行环境、上下文等分析清楚。

2）语义分析：分析程序中不安全的函数，方法的使用的安全问题。

3）数据流分析：跟踪，记录并分析程序中的数据传递过程所产生的安全问题。

4）控制流分析：分析程序特定时间，状态下执行操作指令的安全问题。

5）配置分析：分析项目配置文件中的敏感信息和配置缺失的安全问题。

6）结构分析：分析程序上下文环境，结构中的安全问题。

7）结合2）-6）的结果，匹配所有规则库中的漏洞特征，一旦发现漏洞就抓取出来。

8）最后形成包含详细漏洞信息的漏洞检测报告，包括漏洞的具体代码行数以及漏洞修复的建议。

**2. SAST优劣势分析**

SAST需要从语义上理解程序的代码、依赖关系、配置文件。优势是代码具有高度可视性，能够检测更丰富的问题，包括漏洞及代码规范等问题。测试对象比DAST丰富，除Web应用程序之外还能够检测APP的漏洞，不需要用户界面，可通过IDE插件形式与集成开发环境（如Eclipse、IntelliJ IDEA）结合，实时检测代码漏洞问题，漏洞发现更及时，修复成本更低。

另一方面SAST不仅需要区分不同的开发语言（PHP、C＃、ASP、.NET、Java、Python等），还需要支持使用的Web程序框架，如果SAST工具不支持某个应用程序的开发语言和框架，那么测试时就会遇到障碍。DAST支持测试任何语言和框架开发的HTTP/HTTPS应用程序。

传统的SAST扫描时间很慢，如果是用SAST去扫描代码仓库，需要数小时甚至数天才能完成，这在日益自动化的持续集成和持续交付（CI/CD）环境中效果不佳。

还有一点是SAST的误报，业界商业级的SAST工具误报率普遍在30%以上，误报会降低工具的实用性，可能需要花费更多的时间来清除误报而不是修复漏洞。

SAST只对源代码进行检测，而不会分析整个应用程序，这迫使企业需要购买单独的软件组合分析工具（SCA），即使是SCA也只是识别公开的漏洞；开源、第三方API或框架中的未知漏洞超出了SAST和SCA的范围。

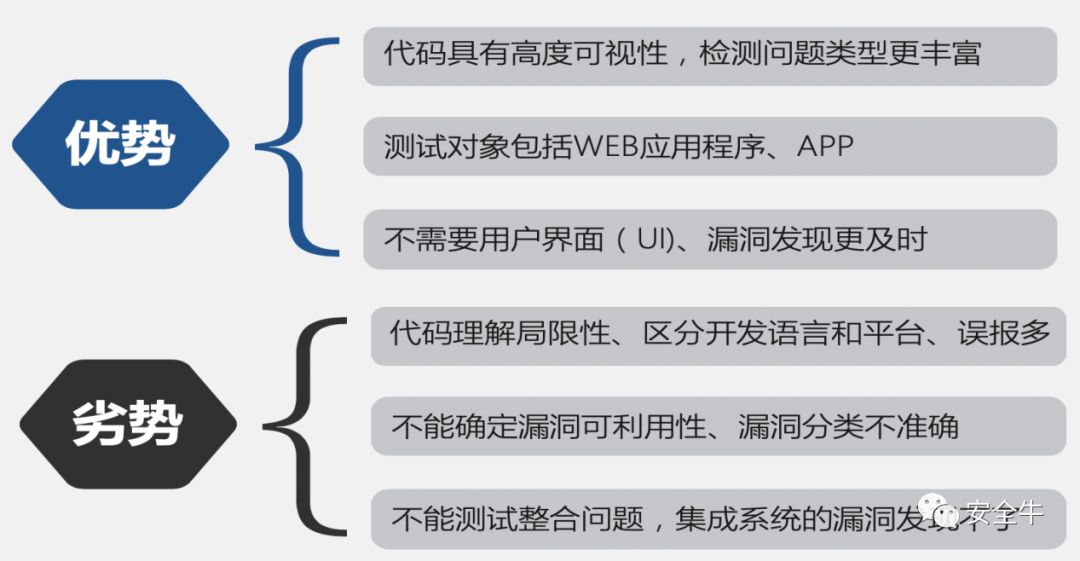


图 4：SAST优势与劣势

**五、IAST安全测试技术**

IAST交互式应用安全测试技术是最近几年比较火热的应用安全测试新技术，曾被Gartner咨询公司列为网络安全领域的Top 10技术之一。IAST融合了DAST和SAST的优势，漏洞检出率极高、误报率极低，同时可以定位到API接口和代码片段。

1. ****实现原理****

IAST的实现模式较多，常见的有代理模式、VPN、流量镜像、插桩模式，本文介绍最具代表性的2种模式，代理模式和插桩模式。

* 代理模式，在PC端浏览器或者移动端APP设置代理，通过代理拿到功能测试的流量，利用功能测试流量模拟多种漏洞检测方式对被测服务器进行安全测试。
* 插桩模式，插桩模式是在保证目标程序原有逻辑完整的情况下，在特定的位置插入探针，在应用程序运行时，通过探针获取请求、代码数据流、代码控制流等，基于请求、代码、数据流、控制流综合分析判断漏洞。插桩模式具体实现有2种模式，Active 插桩和Passive 插桩。

**1) 代理模式实现原理**

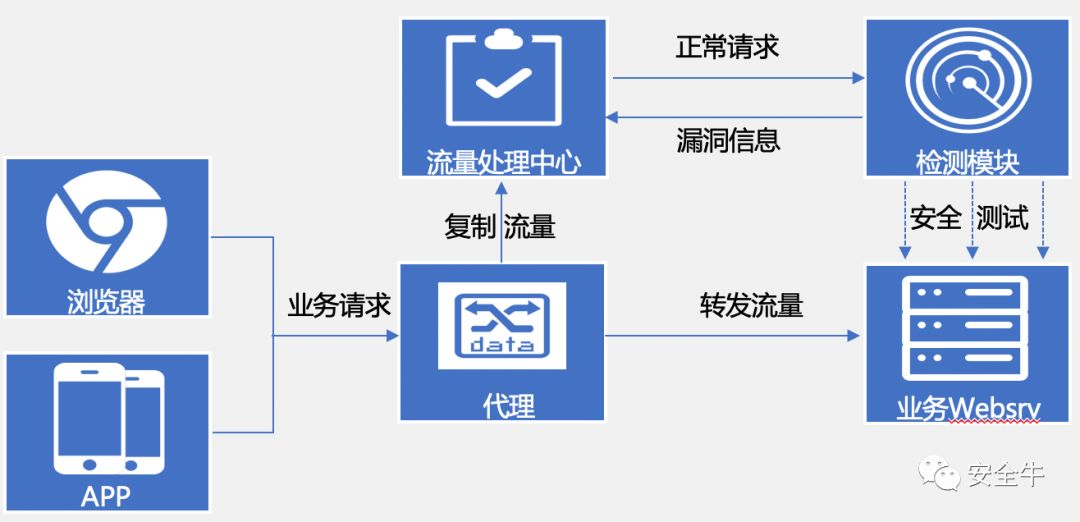


图 5：代理模式原理

a. 功能测试人员在浏览器或者APP中设置代理，将IAST设备地址填入；

b. 功能测试人员开始功能测试，测试流量经过IAST设备，IAST设备将流量复制一份，并且改造成安全测试的流量；

c. IAST设备利用改造后的流量对被测业务发起安全测试，根据返回的数据包判断漏洞信息。

插桩需要在服务器中部署Agent，不同的语言不同的容器要不同的Agent，这对有些用户来说是不可接受的。而代理模式不需要服务器中部署Agent，只是测试人员要配置代理，安全测试会产生一定的脏数据，漏洞的详情无法定位到代码片段，适合想用IAST技术又不接受在服务器中部署Agent的用户使用。

**2) Active插桩实现原理**

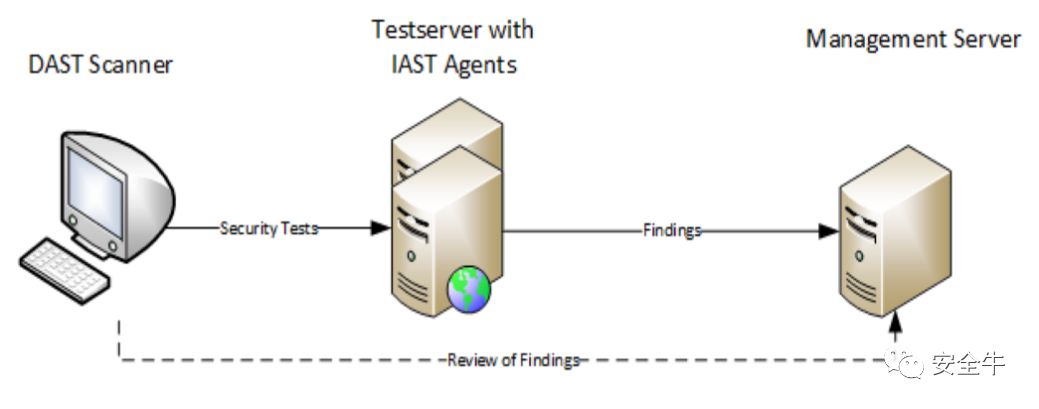


图 6：Active 插桩原理

a. 被测试服务器中安装IAST插桩 Agent；

b. DAST Scanner发起扫描测试；

c. IAST插桩 Agent追踪被测试应用程序在扫描期间的反应附加测试，覆盖率和上下文，将有关信息发送给Management Server，Management Server展示安全测试结果。

Active 插桩模式需要在被测试应用程序中部署插桩 Agent，使用时需要外部扫描器去触发这个Agent。一个组件产生恶意攻击流量，另一个组件在被测应用程序中监测应用程序的反应，由此来进行漏洞定位和降低误报。

Active 插桩模式更像是一种改进版的DAST技术，目前最新的AWVS、AppScan已经采用了Active 插桩模式。AWVS集成了“AcuSensor”模块，通过在源代码中部署传感器来增强定期动态扫描。AcuSensor能够在AWVS扫描期间检查Web应用程序执行时的源代码，在后端抓取应用程序，提供100％爬行覆盖率，查找并测试在黑盒扫描期间未发现的隐藏输入。AppScan则是集成了“Glass Box”服务模块，这使得AppScan支持 Web 2.0、JavaScript 和 AJAX 框架。

Active 插桩模式解决了传统DAST漏报和无法精确定位漏洞位置的问题，需要先做扫描，扫描触发漏洞需要一定的时间，而且扫描会对业务测试产生影响。在双向HTTPS加密、CSRF Token页面、防攻击重放等场景下Active 插桩模式依然无法进行安全测试。

1. **Passive插桩实现原理**

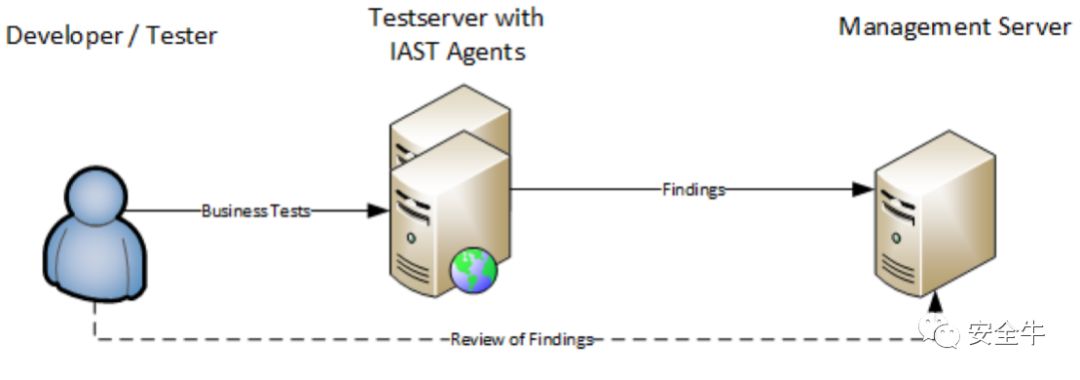


图 7：Passive 插桩原理

a. 被测试服务器中安装插桩 Agent；

b. 插桩 Agent在应用程序运行时获取请求和代码数据流、代码控制流；

c. 插桩Agent将获取的信息发送给Management Sever，Management Sever展示安全测试结果。

Passive 插桩在程序运行时监视应用并分析代码，它不会主动对Web应用程序执行攻击，而是纯粹被动地分析检测代码。这实际上是一个巨大的优势，因为它不会影响同时运行的其他测试活动，并且只需要业务测试（手动或自动）来触发安全测试，有测试流量过来就可以实时的进行漏洞检测。

插桩模式的关键是Agent，Agent需要根据不同语言进行开发，但是功能基本相同：

获取请求数据和返回数据；

代码执行中的参数传递；

数据库查询（如ODBC）；

目录查询（如LDAP），文件系统权限；

监听内存中特定的值，识别受污染的输入；

第三方库的使用;

对外部应用程序和服务的调用；

特定代码的执行等。

1. **IAST优劣势分析**

IAST融合了DAST和SAST的优势，漏洞检出率极高、误报率极低，同时可以定位到API接口和代码片段。

IAST插桩模式的技术基于请求、代码、数据流、控制流综合分析判断漏洞，漏洞测试准确性高，误报率极低。由于IAST插桩模式可获取更多的应用程序信息，因此发现的安全漏洞既可定位到代码行，还可以得到完整的请求和响应信息，完整的数据流和堆栈信息，便于定位、修复和验证安全漏洞。支持测试AJAX页面、CSRF Token页面、验证码页面、API孤链、POST表单请求等环境。

IAST插桩模式在完成应用程序功能测试的同时即可以实时完成安全测试，且不会受软件复杂度的影响，适用于各种复杂度的软件产品。不但可以检测应用程序本身的安全弱点，还可以检测应用程序中依赖的第三方软件的版本信息和包含的公开漏洞。整个过程无需安全专家介入，无需额外安全测试时间投入，不会对现有开发流程造成任何影响，符合敏捷开发和DevOps模式下软件产品快速迭代、快速交付的要求。

IAST插桩模式的核心技术在于探针，探针需要根据不同的语言进行开发，它只能在具有虚拟运行时环境的语言上执行，例如Java，C＃，Python和NodeJS。它不支持C，C ++和Golang等语言。其次，由于agent与真实webserver集成，稳定性非常重要，每次更新需要重启webserver，部署成本较大。业务逻辑漏洞也是IAST插桩模式无法解决的问题。

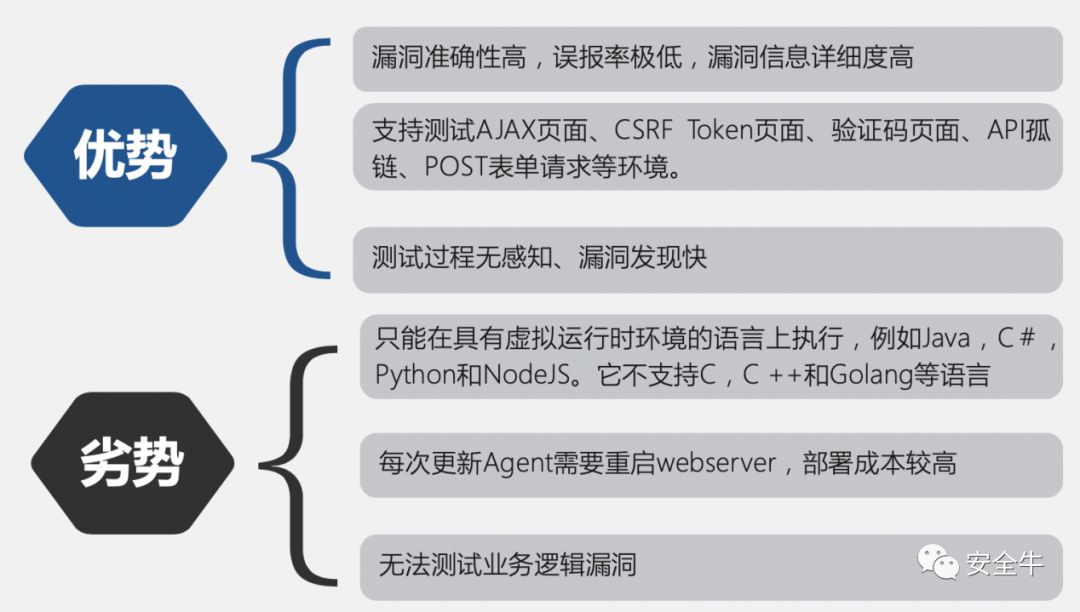


图 8：IAST插桩模式的优劣势

**六、三种技术的应用场景分析**

上文分析了DAST、SAST、IAST三种技术的具体实现原理和各自的优劣势，技术本身没有优劣之分，不同的技术能够解决不同场景下的问题，需要安全工程师能够因地制宜地选择对应的技术解决对应的问题。

|  |  |  |  |
| --- | --- | --- | --- |
| 对比项 | DAST | SAST | IAST |
| 研发流程集成 | 测试/线上运营阶段 | 研发阶段 | 测试阶段 |
| 误报率 | 低 | 高 | 极低（几乎为0） |
| 测试覆盖度 | 低 | 高 | 高 |
| 检查速度 | 随测试用例数量稳定增加 | 随代码量呈指数增长 | 实时检测 |
| 逻辑漏洞检测 | 支持部分 | 不支持 | 不支持 |
| 漏洞检出率 | 中 | 高 | 较高 |
| 影响漏洞检出率因素 | 与测试payload覆盖度相关  企业可优化和扩展 | 与检测策略相关  企业可在定制策略 | 与检测策略相关  企业可定制测量 |
| 第三方组件漏洞检测 | 支持 | 不支持 | 支持 |
| 使用成本 | 较低  基本无需人工验证 | 高  人工排除误报 | 低  基本没有误报 |
| 支持语言 | 不区分语言 | 区分语言 | 区分语言 |
| 支持框架 | 不区分框架 | 区分框架 | 区分框架 |
| 侵入性 | 较高，脏数据 | 低 | 低 |
| 风险程度 | 较高，扫挂/脏数据 | 低 | 低 |
| 漏洞详情 | 中，请求 | 较高，数据流+代码行数 | 高，请求+数据流+代码行数 |
| CI/CD集成 | 不支持 | 支持 | 支持 |
| 持续安全测试 | 不支持 | 支持 | 支持 |
| 工具集成 | 无 | 开发环境集成  构建工具、问题跟踪工具 | 构建工具、自动化测试、API |

图 9：DAST、SAST、IAST之间的对比

DAST技术比较适合用于线上运行环境的监控，研发阶段代码检测适合使用SAST技术，QA阶段适合使用IAST技术。

**七、实际应用**

1. **软件开发阶段:**

与程序员对话的源码安全审计主要基于SAST技术打造， SAST工具对用户的困扰主要来自于误报，通过数据流调用分析、变量关联分析、机器学习等多重手段极大地降低了误报率，减少工具对安全测试工作的困扰，改善用户体验，降低工具的使用成本。

1. **软件测试阶段**

基于IAST技术打造，支持代理、VPN、流量信使、流量镜像、爬虫、导入日志、Passive插桩共7种流量收集模式，真正结合了DAST、SAST、IAST三种技术的优势；漏洞检测率极高，包括水平越权、垂直越权等标准IAST技术无法检测的逻辑漏洞，误报率几乎为0；漏洞详情直接定位请求、数据流、代码片段，修复漏洞更容易；采用Passive 插桩技术，无需重放请求，不会形成脏数据，可覆盖加密、防重放、签名等任意场景；近实时检测漏洞，漏洞检测随着应用程序运行实时进行。

1. **应用上线运营阶段**

采用DAST技术打造资产风险监控系统，在大量企业客户中被用来对线上业务环境进行监控。从攻击者视角对企业进行资产探测，全面发现企业的资产暴露面和应用程序的漏洞，保障线上运营环境的安全；并且，部署模式紧跟业务的使用模式，支持在互联网环境、企业IDC、私有云、公有云、混合云等多种场景下部署使用。