



腾讯代码安全检查(Xcheck)产 品白皮书

2022 年 1 月

版权声明

本文档版权归深圳市腾讯计算机系统有限公司所有，并保留对本文档及本声明的最终解释权和修改权。本文档出现的任何文字叙述、文档格式、插图、照片、方法、过程等内容，除另有特别注明外，其著作权或其它相关权利均属于深圳市腾讯计算机系统有限公司，未经深圳市腾讯计算机系统有限公司书面同意，任何人不得以任何方式或形式对本文档内的任何部分进行复制、摘录、备份、修改、传播、翻译成其他语言、将其全部或部分用于商业用途。

免责条款

本文档仅用于为最终用户提供信息，其内容如有更改或撤回，恕不另行通知。

深圳市腾讯计算机系统有限公司已尽最大努力确保本文档内容准确可靠，但不提 供任何形式的担保。任何情况深圳市腾讯计算机系统有限公司均不对（包括但不限于）最终用户或任何第三方因使用本文档而造成的直接或间接的损失或损害负责。

修订记录

文档版本	产品版本	修改日期	修改人	更新内容
01	1.0	2022-01-05	v_vjjhli	输出整体内容

目录

一、 概述.....	1
1.1 应用安全测试技术概述.....	1
1.2 SAST 静态应用安全测试.....	1
1.3 DAST 动态应用安全测试.....	2
1.4 IAST 交互应用安全测试.....	4
1.5 SCA 组成分析.....	5
二、 腾讯 Xcheck 产品介绍.....	6
2.1 核心技术.....	6
2.2 产品功能.....	8
2.3 Xcheck 优势.....	12
2.4 Xcheck Java 引擎介绍.....	17
三、 管理体系.....	25
3.1 任务量分析.....	25
3.2 数据监控.....	25
3.3 风险状态分析.....	25
四、 应用案例.....	26
4.1 某保险企业应用案例.....	26
4.2 某金融客户应用案例.....	26
4.3 某教育平台应用案例.....	27
五、 FAQ.....	29

一、概述

1.1 应用安全测试技术概述

随着云原生和开源技术的普及，基于 B/S 的 Web 应用技术被广泛用于政企业务数字化转型发展中，随之而来的应用安全威胁也显著增加。根据 Gartner Group 的调研显示，75%的安全漏洞发生在应用程序层，而非我们以往认知的网络层；而 NIST（National Institute of Standards and Technology）也发布相似的调查数据，92%已知的安全漏洞存在于应用程序中，而非网络层中。

为了及时发现软件应用的漏洞和缺陷，确保 Web 应用程序在交付前后的安全性，就需要利用 Web 应用安全测试技术来主动识别 Web 应用程序中架构的薄弱点和漏洞，防止相关应用被黑客及非法人员利用，造成安全危害。

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四大类别。

1.2 SAST 静态应用安全测试

总体而言，软件编码人员通常更加重视和擅长业务功能的开发实现，在安全开发意识和技能储备上相对不足，进而导致我们不得不面临一个残酷的现实：超过 50%的应用安全漏洞都是由错误的编码产生的，其中每 1000 行代码至少产生一个业务逻辑漏洞。想从早期开始治理漏洞就需要制定软件代码检测机制，Static AST 静态应用安全测试是一种分析应用程序的源码，字节码或二进制代码中是否存在安全漏洞，通常在编码和/或测试阶段进行，是一种源码级漏洞检测技术。

SAST 作为开发阶段引入的白盒静态分析技术，需要从语义上理解程序的代码、依赖关系、配置文件。主要优势：

- 相比黑盒漏扫技术，白盒 SAST 技术介入开发阶段更早，可以较好辅助程序员做安全开发，可通过 IDE 插件形式与集成开发环境（如 Eclipse 等 IDE 环境）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；

- 代码具有一定可视性，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。
- 另一方面 SAST 技术也存在比较大的缺陷：
- 居高不下的误报率，业界商业级的 SAST 工具误报率普遍在 40%以上，误报会一定程度降低工具的实用性，可能需要花费更多的时间来清除误报而不是修复漏洞；
 - 不仅需要区分不同的开发语言（PHP、C++、Go、Node.js、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 黑盒技术支持测试任何语言和框架开发的 HTTP/HTTPS 应用程序；
 - 传统白盒 SAST 检测时间较长，如果拿 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；
 - SAST 只对源代码进行检测，而不会分析整个应用程序，这迫使政企组织需要购买单独的软件成分分析工具（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。

1.3 DAST 动态应用安全测试

Dynamic AST 动态应用安全测试是一种黑盒应用安全测试技术，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，应用程序处于运行状态，工具发起对应用程序的模拟攻击，分析应用程序反馈，确定是否有安全漏洞。传统安全工程师常用的工具如 AWVS、AppScan 等就是基于 DAST 原理的产品。



图 1.3 DAST 动态扫描技术原理

- 1) 通过爬虫抓取整个 Web 应用结构，爬虫会识别被测 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 Response 的分析验证是否存在安全漏洞。

DAST 作为运营阶段普遍引入的黑盒动态扫描技术，主要优势：

- 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；
- 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。

另一方面 DAST 技术也存在比较大的缺陷：

- 难以支持对 AJAX 页面、CSRF Token 页面、验证码页面、API 孤链、POST 表单请求等爬虫难以抓取的页面或者是设置了防重放攻击策略的应用的进一步安全测试；
- 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；
- 测试过程会对业务运行造成一定的影响，如脏数据会污染业务运营的数据；

- 测试对象仅限为 HTTP/HTTPS 的 Web 应用程序，对于 IOS/Android 上的 APP 也无能为力；

发现漏洞后只会定位漏洞的 URL，无法定位漏洞的具体代码行数和产生漏洞的原因。

1.4 IAST 交互应用安全测试

Interactive AST 交互应用安全测试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，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，从而发现安全漏洞。

灰盒 IAST 作为开发测试阶段引入的下一代交互式安全测试技术，主要优势：

- IAST 运行时插桩技术基于请求、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；
- IAST 终端流量代理技术基于中间人攻击技术，无需在业务服务器部署需适配特定开发语言的插桩探针，对业务场景侵入性低，并可检测屡屡频发的业务逻辑漏洞；
- IAST 运行时插桩技术可获取更多的应用程序信息，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；
- IAST 运行时插桩技术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；
- 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；
- 支持在完成应用程序功能测试的同时透明实时完成安全测试，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。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。

另一方面 IAST 技术也存在一定的缺陷：

- 该技术主要应用在 DevSecOps/S-SDLC 阶段的测试环节，从整体开发安全体系建设的角度来说，也需要从更加靠前的源头进行管控，如需求设计阶段的威胁建模和全流程漏洞管理等；
- 不管是哪种 IAST 模式，包括运行时插桩模式、终端流量代理/VPN 模式、主机流量嗅探模式及旁路流量镜像模式等，在实际应用中都需要借助实际业务的测试流量（手动或自动）来触发应用安全测试，因此它天然更加适合各类安全开发测试场景。

1.5 SCA 组成分析

Software composition analysis 软件组成分析是一种用于分析应用程序中使用的三方和开源组件的软件成份，发现已知的安全漏洞或其他信息（比如 license/敏感信息等）。

SCA 理论上来说是一种通用的分析方法，可以对任何开发语言对象进行分析，Java、C++、Go、Python、JavaScript 等等，它对关注的对象是从文件层面的文件内容，以及文件与文件之间的关联关系以及彼此组合成目标的过程细节。

首先对目标源代码或二进制文件进行解压，并从文件中提取特征，再对特征进行识别和分析，获得各个部分的关系，从而获得应用程序的组件名称与版本号，进而关联出存在的已知漏洞清单。

由于 SCA 分析过程中不需要把目标程序运行起来，因此具有分析过程对外部依赖少，分析全面，快捷、效率高的优点

二、腾讯 Xcheck 产品介绍

腾讯 Xcheck 代码安全检查工具（以下简称“Xcheck”）是完全自研的静态应用安全测试（SAST，Static application security testing）工具，致力于挖掘代码中隐藏的安全风险，提升代码安全质量。通过自研语义分析算法、精细化模型、推断识别等技术赋能 IT 从业人员，使传统安全小白（研发、测试、QA 等）完成应用编码或测试时即可实现深度业务安全测试，防止应用带病上线。Xcheck 支持 Top 语言及 OWASP 常见漏洞，具有低误报、低漏报、无需编译、污染链展示等优势。

2.1 核心技术



图 2.1 Xcheck 静态分析技术原理

- 1) 首先通过调用开发语言编译器或者解释器把前端的语言代码（如 Java, Python, 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及方法使用所存在的安全问题；
- 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及结构中的安全问题；
- 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。

2.1.1 纯自研语义分析算法

Xcheck 工具使用纯自研的语义识别算法，无需编译，准确“理解”代码，解决了因为不理解代码而造成的误报。并且识别用户自定义的安全防护措施，从而进一步降低误报的概率。精确识别常见的各种语言特性，能够以秒级的扫描速度快速分析代码。

2.1.2 精细化模型设计

基于动态污点追踪技术建立污染分析模型，通过在目标应用程序运行过程中实时跟踪监控并记录程序变量、寄存器和内存等的值，确定污点数据能否从污点源传播到污点汇聚点。动态污点追踪技术能够比较准确地获得应用程序执行过程中各变量和存储单元的状态，可以获得更多的程序运行实时的上下文信息，这极大的方便了检测时对各类数据分析的全面需求，可以更好的进行一些普通黑盒或者白盒不能完成的漏洞检测。

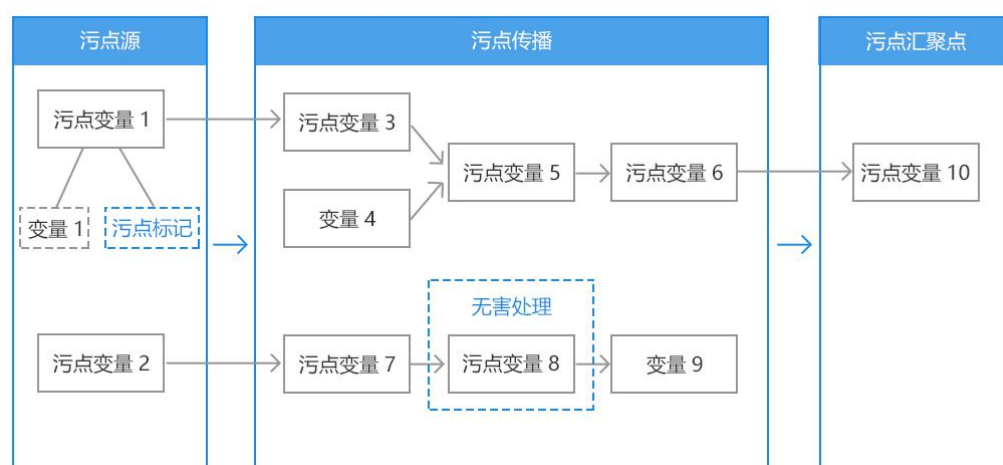


图 2.1.2 Xcheck 动态污点追踪流程示例

如上图所示，污点追踪的处理过程可以分为识别污点源和汇聚点、污点传播分析及无害处理 3 个关键阶段。

Xcheck 扫描后查看结果，可以查看到清晰的污染链条在代码中传播的情况，污点不会被放大或者消失。

2.1.3 灵活强大的扩展

Xcheck 针对未支持的框架和函数,还可以通过自定义规则补充对应的识别能力对于简单的污染点,风险函数和过滤函数,用户自己可以通过客户端轻松编辑生成对应的规则,解决检查器无法识别内部公共库函数的问题。对于复杂的框架适配,由 Xcheck 研发团队统一对外提供。

2.1.4 类型判断

Xcheck 通过词法分析、语法分析、语义分析、获取数据流、函数等优秀的算法进行代码扫描,从多个因素进行推断,识别逻辑上的防护写法,准确判断污点类型,进一步减少误报。

2.2 产品功能

Xcheck 现已支持 Go、Java、Node.js、PHP、Python、C++六种语言的安全检查,其他语言支持还在开发中(JavaScript、Android)。覆盖漏洞包括 SQL 注入、代码注入、命令注入、跨站脚本、反序列化漏洞、路径穿越等多种漏洞。

在框架支持上,Xcheck 内置覆盖了常见的 Web 框架,也可根据易编写易扩展的自定义规则模块自行编写规则对第三方框架进行覆盖支持。

2.2.1 支持 Top 语言及框架

表 2.2.1 Xcheck 支持语言与框架

语言	框架
Go	Gin,Beego,Iris,net/http,fastrouter,httprouter,go-restful,mux
Java	Spring,HttpServlet,WebService,jar-rs,struts2,EJBs,dubbo
Node.js	Koa,Express
PHP	Thinkphp,Laravel,Codelgniter,Yii,Yaf
Python	Django,Flask,Tornado,Webpy,Bottle,BaseHTTPServer
C++	腾讯内部框架,trpc-c++/spp/svrkit

2.2.2 OWASP 前十类风险支持情况

表 2.2.2 Xcheck 覆盖风险范围

支持	不支持
注入型 (SQL 注入、模板注入等)	逻辑类
敏感信息泄露	组件类
XML 外部实体	配置类
跨站脚本	
不安全的反序列化	

支持常见漏洞类型：



图 2.2.2 Xcheck 支持的漏洞类型

2.2.3 支持自定义规则

支持用户自定义扩展规则，可以使用 Python 编写复杂规则，支持本地断点调试。Xcheck 针对未支持的框架和函数，可以通过自定义规则补充对应的识别能力。对于简单的污染点，风险函数和过滤函数，用户自己可以通过客户端轻松编辑生成对应的规则；对于复杂的框架适配，由 Xcheck 研发团队统一对外提供。

Go 为例使用通用规则和专用规则：

通用规则

通用规则一般用于 mock 三方 package（例如 net/http）

通用规则的编写主要分为两部分，包结构定义和行为定义。其中包结构定义，主要是定义 `package` 的结构，包括 `package` 的名字，它的成员函数，结构体及结构体方法；行为定义，主要是定义成员函数及结构体方法具体的逻辑代码，检查器会执行这些代码，从而得到某些结果或触发某些检查行为等。

专用规则

专用规则适用于 `mock` 指定项目的 `package`。通过规则能让检查器“读懂”代码，从而发现代码问题。本质上专用规则和通用规则实现上基本差不多，只是触发条件不一样，适用不同的场景。专用规则需要使用 `@X.override` 接口注册函数。

如何生效

1. 在被扫代码根目录下，编写名为 `xcheck-Go.py` 的规则代码（第一优先级）；
2. 在 `Go/rules/match.yml` 文件配置（第二优先级）；
3. 在 `Go` 检查器内部，专用规则会优先通用规则生效。

匹配条件

检查器会在扫描前会尝试遍历所有专用规则，遇到第一个匹配成功的专用规则，文件会停止遍历，并且生效该规则文件。规则名称，后面是匹配条件，用于匹配特定项目的特征。匹配条件有：文件路径、特征字符串。

2.2.4 部署逻辑

私有化部署服务

Xcheck 提供私有化部署服务，产品使用的整个生命周期，源码都不会流出公司网络，杜绝源码泄露风险。企业可以定制个性使用方案，拓展性强，让产品更好的服务于企业，具有以下特点。

- 功能灵活，管理模式成熟，支持企业个性化定制；

- 安全性更高，系统部署在本地，数据更安全可控；
- 拓展性强，能二次开发；
- 可实现内外网隔离，局域网+外网办公发挥综合优势。

API 形式提供服务

Xcheck 支持以 WebAPI 的形式提供服务，或者可以通过浏览器来进行操作。



图 2.2.4 Xcheck 的 API 服务形式

2.2.5 交付使用情况

Xcheck 拥有一套优秀的代码分析算法，在保证精准分析的前提下，解决了传统静态分析工具效率低的问题。

表 2.2.5 Xcheck 的交付使用情况

	Xcheck	现有 SAST 工具
耗时	每秒千行级甚至万行级 一般项目秒速分析	几十分钟、数小时
误报	误报率低于 10%~15%	高误报
自定义规则	支持用户自定义规则 支持本地调试	基本规则
检出能力	语言特性全面支持 丰富的框架库工具知识 过程间分析	部分特性不支持 有限框架支持 过程内分析

2.2.6 应用场景

代码安全审计

Xcheck 可以代替人工进行代码安全审查, 自动化扫描源码中存在的安全漏洞, 具备以下特点:

- 支持命令注入、SQL 注入、XSS、XXE、URL 跳转、文件上传、文件读取、文件删除、SSRF、反序列化、模板注入等漏洞类型;
- 提高了漏洞挖掘的效率, 可以每天轻松完成扫描上万个项目;
- 沉淀了漏洞挖掘的技能, 使得漏洞挖掘的工作不再依赖掌握特定技能的人员。

持续集成

作为代码安全质量检测门禁接入流水线, 保证每次发布的代码没有已知类型的安全漏洞。Xcheck 支持以下两种创建扫描任务的方式:

- 在 Web 页面手工创建, 用于人工针对少量项目的操作;
- 调用 API 创建, 用于集成进 CI 流程, 批量创建任务操作。

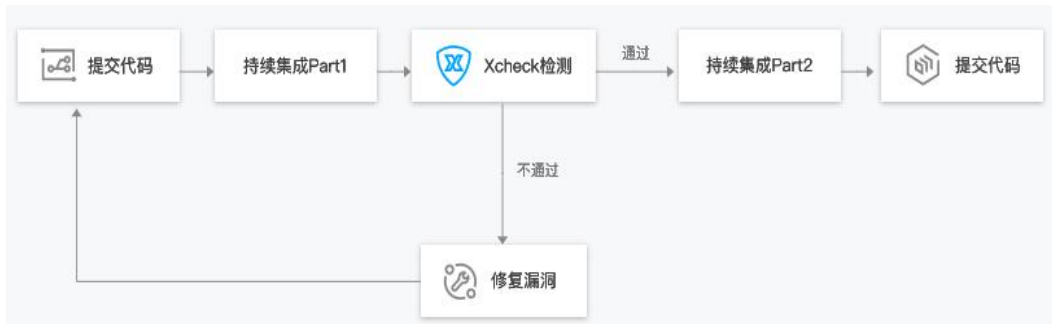


图 2.2.6 Xcheck 的持续集成

2.3 Xcheck 优势

Xcheck 在基于成熟的污点分析技术与对抽象语法树的精准剖解上, 通过巧妙优雅的实现来达到对污点的传递和跟踪的目的, 更精准地发现隐藏在代码中的安全风险。同时赋予了 Xcheck 以下优势:

2.3.1 低误报

经过团队对 Xcheck 投喂大量的项目进行误报优化，Xcheck 各语言的误报率低于 10%，以 Python 为例，Xcheck 对 GitHub 部分 Python 开源项目检测结果如下：



图 2.3.1 Xcheck 低误报

以 Go 为例，在 GitHub 上面选取了 10000 个 Go 项目进行安全检查，发现其中存在风险的项目 182 个，风险数 317 个，误报数 19 个，整体误报率 6% 左右。

表 2.3.1 Xcheck 中 Go 项目误报率

项目	数据
GitHub 项目数	10000
存在风险的项目数	182
问题总数	317
误报数	19
误报率	6%

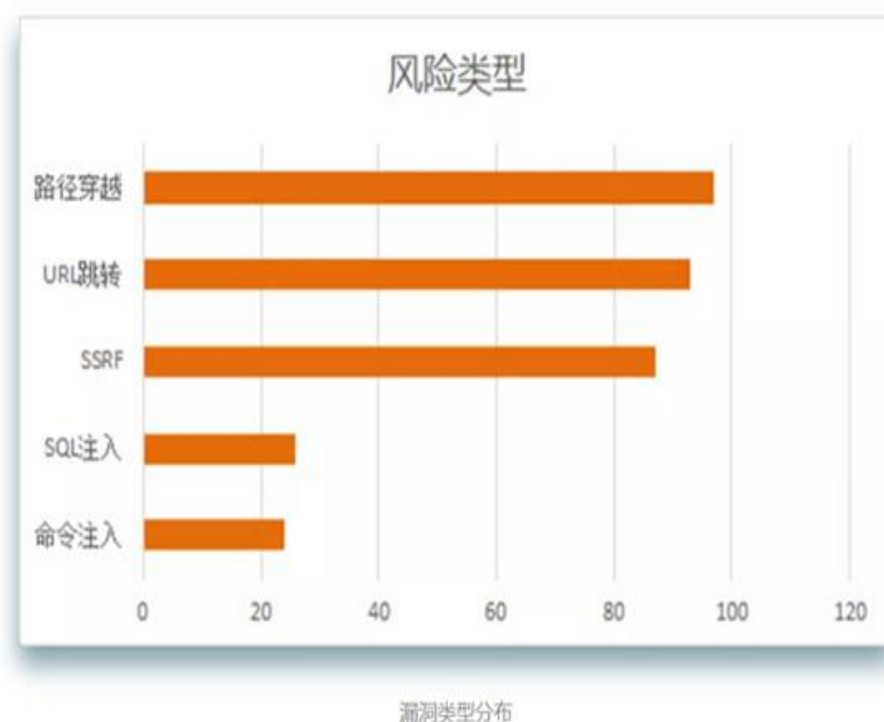


图 2.3.1 Xcheck 中 Go 项目风险类型

传统的 SAST 需要比较长的扫描分析时间，实时性比较差，一个稍大的项目扫描好几个小时，在版本迭代快速的 Web 应用开发中严重拖慢流水线，同时误报率也比较高，同一个漏洞报 n 次的情况更是常有发生，需要投入大量的安全团队资源来去除这些误报，因此无法敏捷地融入到 DevOps 中。但 Xcheck 基于这两大优势，可完美融入 DevSecOps，加速流水线又快又安全地建设。

2.3.2 低漏报

Xcheck 如有漏报我们还可以通过人工进行逐个审查、以下是 PHP 的例子在发现风险的项目里抽取进行人工审查情况：

GitHub 扫描项目总数：1528，发现有风险项目数：309

表 2.3.2 Xcheck 低漏报

类别	数目
SQLI	198
XSS	204
RCE	18
PATH	80

误报	19
总数	518
误报率	3.66%

2.3.3 速度快

Xcheck 拥有一套优秀的算法，解决了传统静态分析工具效率低的问题。使用 Xcheck 扫描内部 Go 项目 19w 次任务，其中耗时小于 1 分钟的项目占 18 万个，90%以上的项目都在 1 分钟内扫描完毕。在 4 核 16g 的 linux 云主机上，Xcheck 对项目的检查速 1w+/s，部分项目可以达到 2w+/s。以 28w 行的 wordpress 项目例，xcheck 检查时间为 18s。

2.3.4 无需编译

操作系统独立，代码扫描不依赖于特定操作系统，只需在企业范围内部署一台扫描服务器，就可以扫描其它操作系统开发环境下的代码。Xcheck 扫描代码时，代码不需要编译，可以直接扫描。

2.3.5 污染链展示

Xcheck 扫描后查看结果可以查看代码的污染链条，例如污点在 test1.py 文件的第 6 行传到 test2.py 的第 12 行传到 test3.py 的第 33 行再传到 test.py 的第 45 行这样就形成了污染链条。

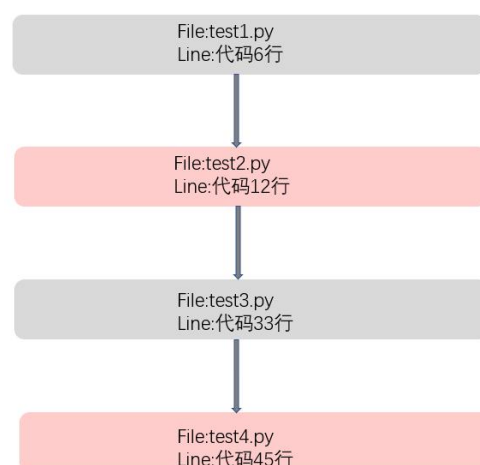


图 2.3.5 Xcheck 污染链展示

2.3.6 完美结合 DevOps

Xcheck 提供多种服务形式，完美结合 DevOps。支持完全本地分析，无需上传代码。



图 2.3.6-1 Xcheck 结合 DevOps

多种接入方式，支持 Web/API、代码检查插件、CI 插件、IDE 插件：

- Web 页面/API：通过 API 调用或 Web 调用接入；
- 代码检查平台插件：通过第三方插件 Codedog、Codecc 接入；
- CI 平台插件：融合 CODING-CI、蓝盾等 CI 插件接入；
- IDE 插件：支持 VS Code 中使用 Xcheck 插件。



图 2.3.6-2 Xcheck 多种服务形式

2.4 Xcheck Java 引擎介绍

2.4.1 技术原理

通过词法分析、语法分析、语义分析、获取数据流、函数等算法精准分析检查风险漏洞。

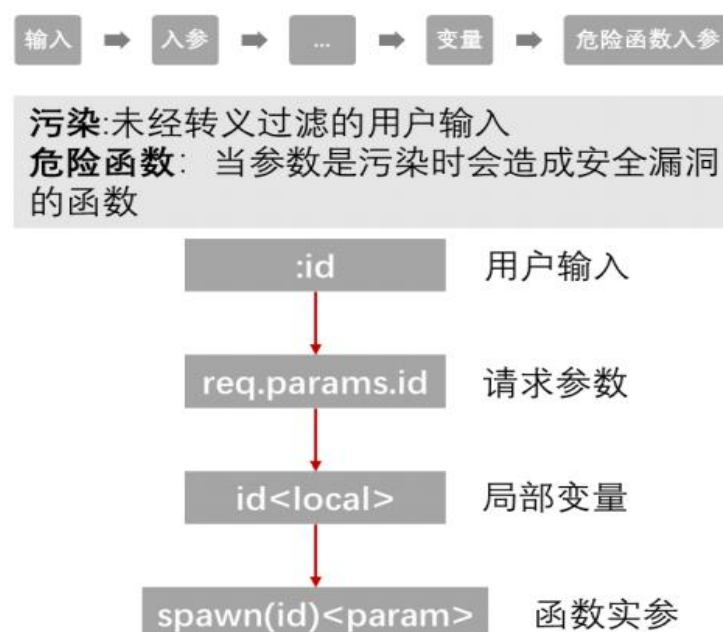


图 2.4.1-1 Javacheck 技术原理

```
const express = require('express');
const cp = require('child_process');
const server = express();

server.get('/abc/:id', (req, res) => {
  // 入参
  const cp = require('child_process');
  // 传递到res.send函数
  res.send(req.params.id);
  // 传递到局部变量id
  var id = req.params.id;
  // 传递到cp.spawn函数
  cp.spawn(id);
});

server.listen(3000);
```

图 2.4.1-2 Node.js 示例

2.4.2 能力介绍

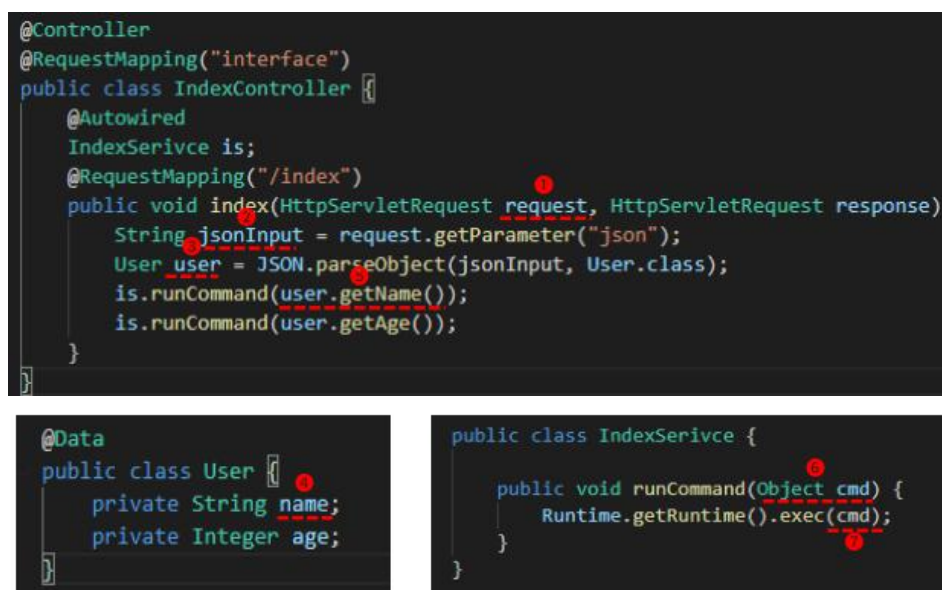
类变量识别

Xcheck 可以精确解析 model 类、map、JSONObject 等结构树对象，而不是把他们当做一个整体来处理。

```
"type": "RCE",
"confidence": 100,
"stmts": [
  {
    "file": "com\\controller\\IndexController.java",
    "line_no": 16,
    "info": "request    value: <*>"
  },
  {
    "file": "com\\controller\\IndexController.java",
    "line_no": 18,
    "info": "request -> jsonInput    value: <*>    confidence:100"
  },
  {
    "file": "com\\controller\\IndexController.java",
    "line_no": 7,
    "info": "jsonInput -> name    value: <*>    confidence:100"
  },
  {
    "file": "com\\controller\\IndexController.java",
    "line_no": 20,
    "info": "name -> runCommand(Object, )    value: <*>    confidence:100"
  },
  {
    "file": "com\\services\\IndexService.java",
    "line_no": 5,
    "info": "runCommand(Object, ) -> cmd    value: <*>    confidence:100"
  },
  {
    "file": "com\\services\\IndexService.java",
    "line_no": 6,
    "info": "cmd -> RCE    value: <*>    confidence:100"
  }
]
```

图 2.4.2-1 Javacheck 类变量识别

将客户端输入反序列化为 User 类对象时，会细化至 User 类的变量，所以污点传递至 user 对象的 name 变量，而 age 变量是 Integer 类型，不受影响。



```

@Controller
@RequestMapping("interface")
public class IndexController {
    @Autowired
    IndexService is;
    @RequestMapping("/index")
    public void index(HttpServletRequest request, HttpServletResponse response) {
        String jsonInput = request.getParameter("json");
        User user = JSON.parseObject(jsonInput, User.class);
        is.runCommand(user.getName());
        is.runCommand(user.getAge());
    }
}

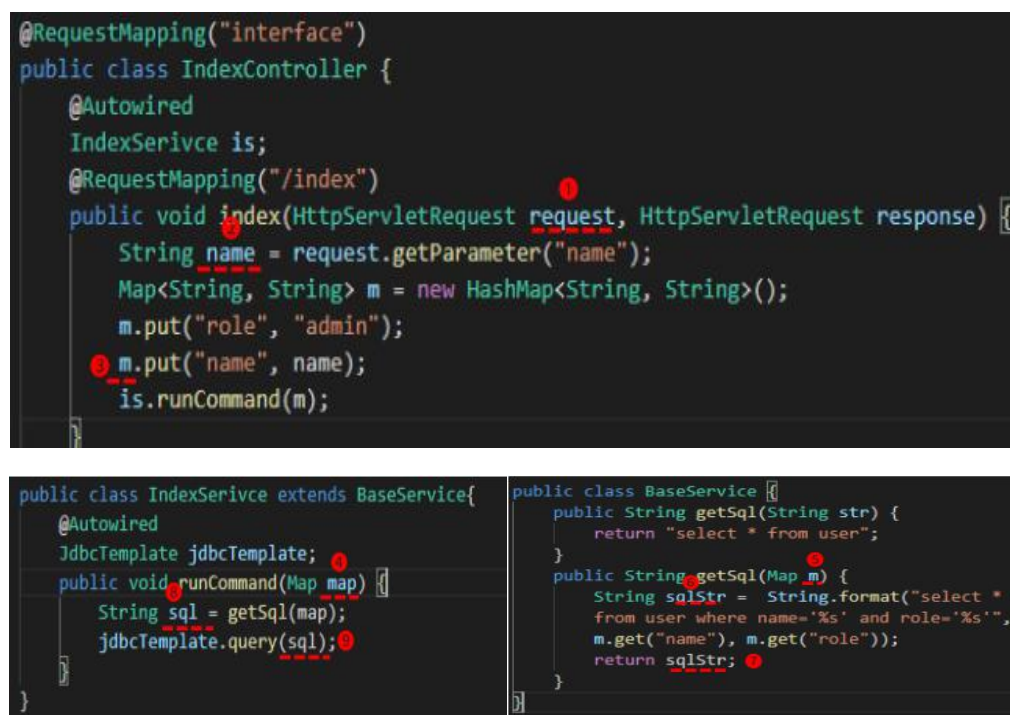
@Data
public class User {
    private String name;
    private Integer age;
}

public class IndexService {
    public void runCommand(Object cmd) {
        Runtime.getRuntime().exec(cmd);
    }
}
    
```

图 2.4.2-2 Javacheck 类变量识别

多态和重载

- 能够识别继承关系；
- 能够找到正确的重载函数；
- 对 Map 进行精确识别。



```

@RequestMapping("interface")
public class IndexController {
    @Autowired
    IndexService is;
    @RequestMapping("/index")
    public void index(HttpServletRequest request, HttpServletResponse response) {
        String name = request.getParameter("name");
        Map<String, String> m = new HashMap<String, String>();
        m.put("role", "admin");
        m.put("name", name);
        is.runCommand(m);
    }
}

public class IndexService extends BaseService {
    @Autowired
    JdbcTemplate jdbcTemplate;
    public void runCommand(Map map) {
        String sql = getSql(map);
        jdbcTemplate.query(sql);
    }
}

public class BaseService {
    public String getSql(String str) {
        return "select * from user";
    }
    public String getSql(Map m) {
        String sqlStr = String.format("select * from user where name='%s' and role='%s'", m.get("name"), m.get("role"));
        return sqlStr;
    }
}
    
```

图 2.4.2-3 Javacheck 识别多态和重载

反射支持

通过 Java 反射调用 IndexService.runCommand 方法，并将客户端传入的参数作为方法参数传递。检查器可以理解这种灵活的语义，污染可以正确传递。

```
"type": "RCE",
"confidence": 100,
"stmts": [
  {
    "file": "com\\controller\\IndexController.java",
    "line_no": 13,
    "info": "request  value: <*>"
  },
  {
    "file": "com\\controller\\IndexController.java",
    "line_no": 15,
    "info": "request -> cmd  value: <*>  confidence:100"
  },
  {
    "file": "com\\controller\\IndexController.java",
    "line_no": 18,
    "info": "cmd -> runCommand(String, )  value: <*>  confidence:100"
  },
  {
    "file": "com\\services\\IndexService.java",
    "line_no": 4,
    "info": "runCommand(String, ) -> cmd  value: <*>  confidence:100"
  },
  {
    "file": "com\\services\\IndexService.java",
    "line_no": 5,
    "info": "cmd -> RCE  value: <*>  confidence:100"
  }
]
```

图 2.4.2-4 Javacheck 反射支持

```
@Controller
@RequestMapping("interface")
public class IndexController {
    @Autowired
    IndexService is;
    @RequestMapping("/index")
    public void index(HttpServletRequest request, HttpServletResponse response) {
        String cmd = request.getParameter("cmd");
        Class cls = Class.forName("com.services.IndexService");
        Method m = cls.getDeclaredMethod("runCommand", new Class[]{String.class});
        m.invoke(cls.newInstance(), cmd);
    }
}
```

```
public class IndexService {
    public void runCommand(String cmd) {
        Runtime.getRuntime().exec(cmd);
    }
}
```

图 2.4.2-5 Javacheck 反射支持

扩展规则

支持过 Python 脚本新增扩展规则，建立内部公共规则，解决检查器无法识别内部公共库函数的问题。

```

3 import java.io.IOException;
4 import common.util.FileUtils;
5 import java.io.File;
6
7 public class FileTest {
8     @RequestMapping("fileDelete")
9     public String fileDelete(String fileName) throws IOException {
10         String path = "/tmp" + fileName;
11         File f = FileUtils.CreateFile(path);
12         f.delete();
13     }
14 }

```

```

@X.declare_object(''''
common.util.FileUtils(module):
    CreateFile(function): s
''')

@X.register('common.util.FileUtils.CreateFile')
def create_file(args: List[Object]) -> Object:
    param = args[0]
    cls_obj = X.ref_by_name("java.io.File")
    if X.is_taint(param):
        return X.taint("createFile", param, cls_obj)
    return cls_obj

```

```
@X.declare_object(''''
    common.util.FileUtils(module):
        CreateFile(function): s
        CreateSafeFile(function): s
    ''')

|

@X.register('common.util.FileUtils.CreateSafeFile')
def create_safe_file(args: List[Object]) -> Object:
    return X.ref_by_name("java.io.File")
```

```
{
  "cost": 0.7569847106933994,
  "result": [
    {
      "id": "9ed071f37f83cda5felf4f23ce7d48d1934ae85f",
      "type": "DEL",
      "confidence": 20,
      "stats": [
        {
          "file": "test.java",
          "line_no": 8,
          "info": "fileName value: <*>",
          "file": "test.java",
          "line_no": 10,
          "info": "fileName -> path value: /tmp<*>"
        },
        {
          "file": "test.java",
          "line_no": 11,
          "info": "path -> f value: /tmp<*>"
        },
        {
          "file": "test.java",
          "line_no": 12,
          "info": "f -> DEL value: /tmp<*>"
        }
      ]
    }
  ]
}
```

```
{
  "cost": 0.572478663861084,
  "result": [
    {
      "id": "9ed071f37f83cda5felf4f23ce7d48d1934ae85f",
      "type": "DEL",
      "confidence": 100,
      "stats": [
        {
          "file": "test.java",
          "line_no": 8,
          "info": "fileName value: <*>"
        },
        {
          "file": "test.java",
          "line_no": 10,
          "info": "fileName -> path value: /tmp<*>"
        },
        {
          "file": "test.java",
          "line_no": 11,
          "info": "path -> f value: /tmp<*>"
        },
        {
          "file": "test.java",
          "line_no": 12,
          "info": "f -> DEL value: /tmp<*>"
        }
      ]
    }
  ]
}
```

```
{
  "cost": 0.4395288421061543,
  "result": [
    {
      "id": "1002b664c024155f520136acd1ball4580b1656f",
      "type": "DEL",
      "confidence": 20,
      "stats": [
        {
          "file": "test.java",
          "line_no": 8,
          "info": "fileName value: <*>"
        },
        {
          "file": "test.java",
          "line_no": 10,
          "info": "fileName -> path value: /tmp<*>"
        },
        {
          "file": "test.java",
          "line_no": 11,
          "info": "path -> f value: /tmp<*>"
        },
        {
          "file": "test.java",
          "line_no": 12,
          "info": "f -> DEL value: /tmp<*>"
        }
      ]
    }
  ]
}
```

```
{
  "cost": 0.7134461402893066,
  "result": []
}
```

图 2.4.2-6 Javacheck 扩展规则

防护识别

Xcheck 可以对被修复过的代码进行防护识别。

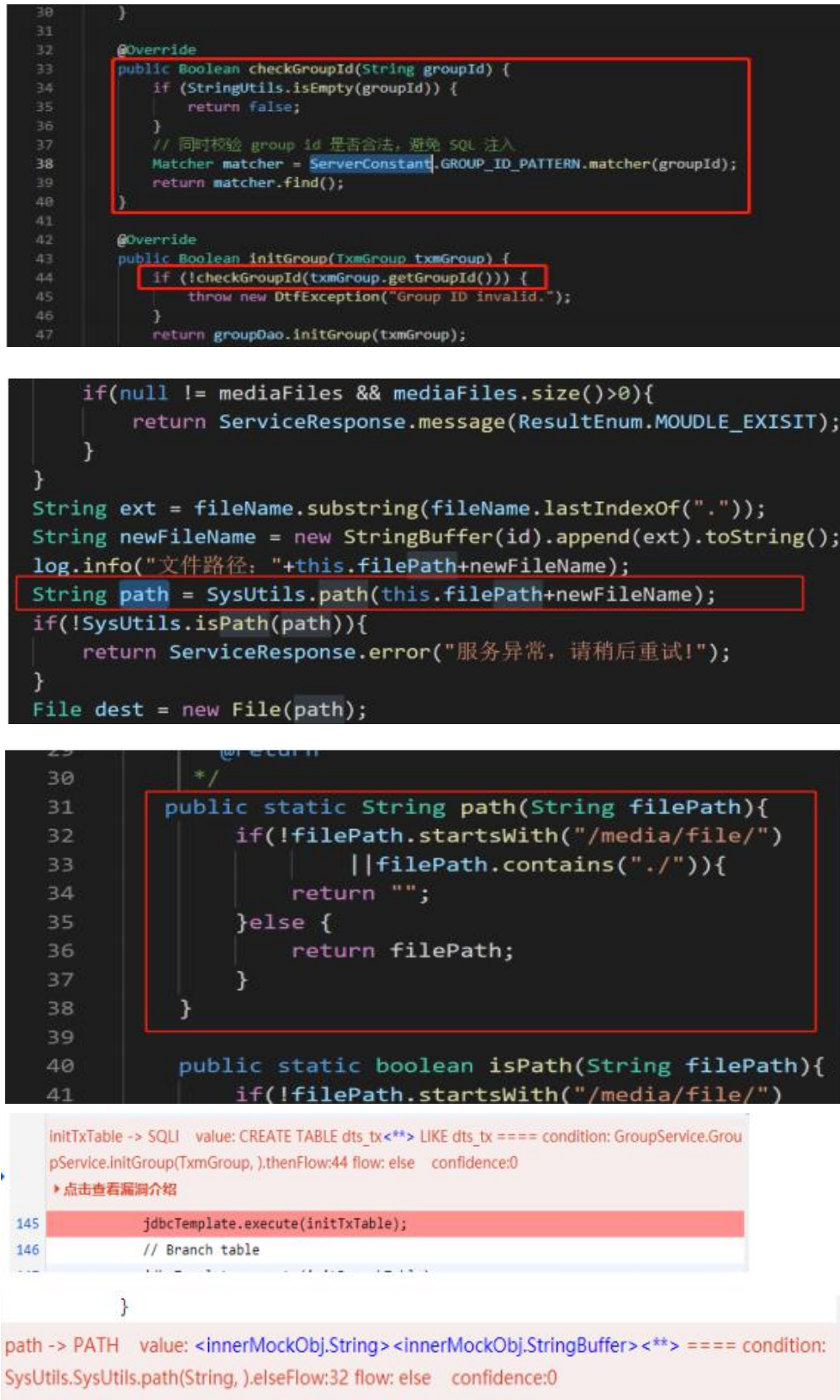


图 2.4.2-7 Javacheck 防护识别

2.4.3 测试集

外部测试：

使用 Javacheck 对 OWASP Benchmark 和 Juliet_Test_Suite_vl.3_for_Java 测试情况：

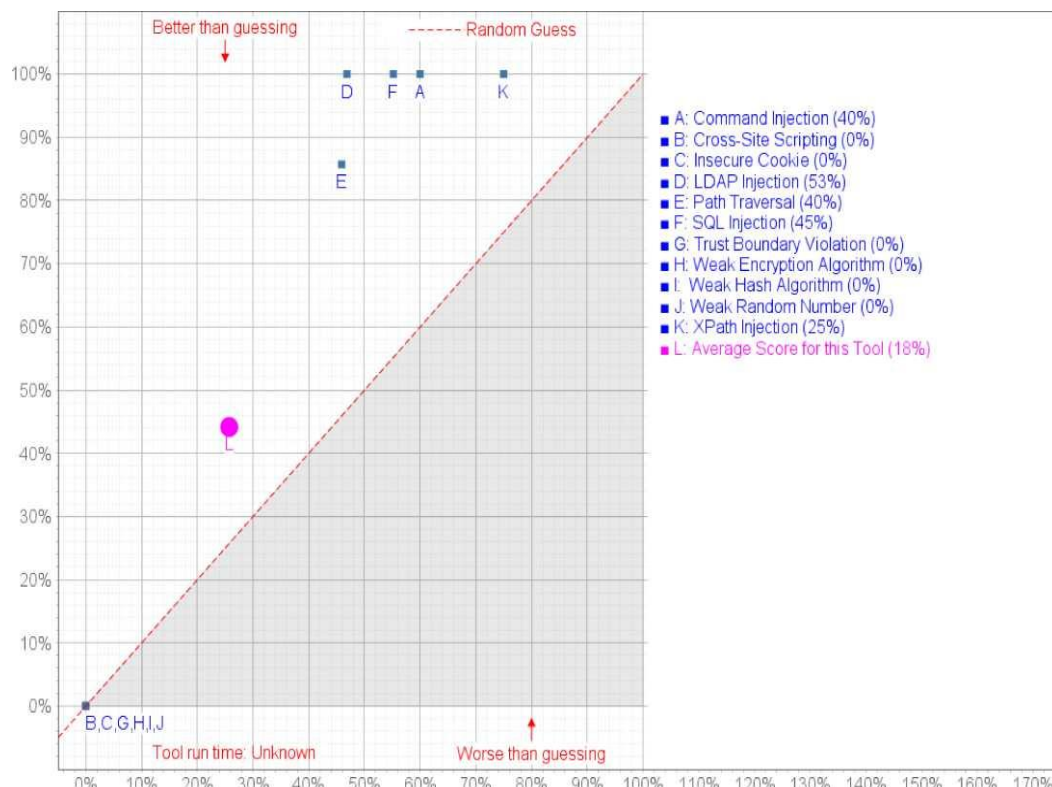


图 2.4.3-1 Javacheck 外部测试

自制测试套：

该测试套（[GitHub - hiahiah4/Java SAST TestCase Suite](https://github.com/hiahiah4/Java-SAST-TestCase-Suite)）为针对 SAST 能力评测的 Java 语言测试集。其中包含 55 个测试用例，涵盖到 Java 语言多态、重载、反射、Json 反序列化、Map、mybatis、lombok 注解等代码段，以及常见的安全防护用例，34 个用例通过一些防护措施之后，漏洞已经修复。

用例分布情况：

目录	用例个数	描述
defense	28	漏洞修复的代码
generics	2	多态
json	7	Json反序列化 (fastjson, jackson)
lombok	2	lombok注解
map	5	Map相关操作
mybatis	3	mybatis相关注解
overLoad	4	方法重载
reflect	4	Java反射调用

图 2.4.3-2 Javacheck 自制测试

三、管理体系

通过对数据的收集、统计、追踪和监控来搭建 Xcheck 管理体系，定制了一个能实时监控项目任务量、风险数、风险分布并可可视化的后台管理。

3.1 任务量分析

分析任务量和接入项目数量：



图 3.1 Xcheck 任务量分析

3.2 数据监控

监控告警数、修复率、误报数据：



图 3.2 Xcheck 数据监控

3.3 风险状态分析

任务语言、风险类别、任务状态、耗时等分布：

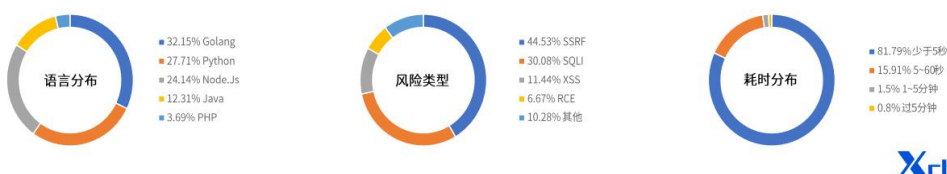


图 3.3 Xcheck 风险状态分析

四、应用案例

4.1 某保险企业应用案例

项目背景

用户直接面对互联网用户进行服务。在网络安全工作中遇到几个难题：

- 互联网式的敏捷迭代更新开发模式，新业务不断更新上线；
- 专业安全人员缺乏，长期处于“几个人的安全团队”的状态；
- 尝试引入安全属性到目前的 DevOps 平台，又担心影响开发节奏。

解决方案

部署腾讯 Xcheck，将安全能力嵌入 DevOps 平台，结合插件，实现功能验证的同时自动化完成业务应用上线前的安全审查，并将结果导入 DevOps 平台，协调项目管理。

用户收益

在安全资源严重不足的情况下，通过腾讯 Xcheck 实现内部业务安全众测，将安全能力嵌入到了 DevOps 平台，在不影响应用快速迭代模式下，实现 QA 阶段的透明安全测试，并将测试结果集成至 DevOps 平台，降低了人工成本，提高安全团队的生产力，帮助消控 90%以上业务中高危漏洞，有效防止了应用带病上线。

4.2 某金融客户应用案例

项目背景

某金融平台客户每天都完成海量业务，业务系统保存着用户的敏感信息，多个系统被评为较高的等保级别。截止目前，业务系统中水平/垂直越权、批量注册、业务接口乱序调用等业务逻辑漏洞和第三方开源组件漏洞频发。针对以上安全漏洞，客户安全管理部门已制定安全技术规范，但由于开发人员在开发过程中未能严格执行安全规范，导致安全漏洞反复出现，尤其是外包产品，大量引用来

源不明的第三方开源框架和组件。在目前开发人员安全基础较弱、传统白盒审计工具大量误报需要人力排查的背景下，如何将应用漏洞特别是业务逻辑漏洞和第三方开源组件漏洞的发现处置前置到功能测试环节，功能测试完成的同时高精度发现安全漏洞并快速让研发人员修复，是目前应用上线前安全审查的当务之急。

解决方案

部署腾讯 Xcheck，快速建立企业内部安全众测模式，通过新建扩展规则，建立内部研发测试的公共规则，在完成功能验证的同时自动化完成软件成分分析和业务逻辑缺陷审查。

用户收益

- 在内部测试环境部署腾讯 Xcheck 产品，协助客户实现建立内部安全众测模式，检测业务逻辑漏洞和第三方开源组件缺陷。开发阶段调试过程中、测试阶段功能测试过程中实现安全测试，并将测试结果集成至客户缺陷管理平台，降低了人工成本，提高安全团队的生产力。
- 在软件开发测试阶段引入 Xcheck 安全测试，第一时间发现并解决问题，防止应用“带病上线”，节约企业成本的同时对应用按期上线提供了安全保证，同时也加强了企业的开发安全意识。

4.3 某教育平台应用案例

项目背景

某国内线上教育平台，由于业务繁多，漏洞频发，数据泄露等安全问题，对该教育平台的声誉和形象造成了影响。急需在研发环节赋予安全能力，同时具备短平快特点，避免上线后爆发漏洞对企业造成不可估量的损失。

解决方案

部署腾讯 Xcheck，快速建立企业内部安全众测模式，通过配置接口完美结合 DevOps，将内部研发测试融合到腾讯 Xcheck，在完成功能验证的同时自动化完成和漏洞息息相关的业务逻辑缺陷审查。

用户收益

- 在教育平台部署腾讯 Xcheck 产品，协助客户实现建立内部安全众测模式，检测业务逻辑漏洞。开发阶段调试过程中、测试阶段功能测试过程中实现安全测试，并将测试结果集成至客户缺陷管理平台，降低了人工成本，提高安全团队的生产力。
- 在软件开发测试阶段引入安全测试，第一时间发现并解决问题，防止应用“带病上线”，节约企业成本的同时对应用按期上线提供了安全保证，同时也加强了企业的开发安全意识。
- 解决业务在日常运营过程中遇到的漏洞频发、黑客入侵、数据泄露等安全问题，为企业安全运营做出了不可估量的贡献。

五、FAQ

Q: Xcheck 如何与现有系统集成?

A:通过调用 Xcheck 提供的 API 方式集成。

1.创建扫描任务

方式一：上传代码文件

`/api/plugin/uploadZip`

说明：需要将被扫描代码压缩成 zip 格式，然后调用该接口上传

方式二：提交代码地址

`/api/plugin/create`

说明：直接提供被扫描项目的 git 地址，使用该方式前，需要配置好 git 的认证方式，并保证 Xcheck 与 git 服务器的连通性。

2. 查询扫描结果

`/api/plugin/result`

说明：如果任务创建成功，会返回任务 ID，提交任务 ID，接口查询该次任务的扫描结果。

Q: Xcheck 多长时间更新一个版本?

A: 根据实际情况不定期更新。例如：业界曝出重大的安全漏洞，新增框架新增 API,产品自身大版本更新等，都会给客户推送更新版本。

Q: Xcheck 支持哪些语言的安全扫描?

A: 目前支持六种语言：Python、Node.js、Go、Java、PHP、C++，即将支持 JavaScript。

Q: Xcheck 是否支持所有类型的代码扫描?

A: 只要是支持的语言就可以扫描。由于 Xcheck 会对代码进行精准解析，因此需要被检测的代码是完整项目，且语法正确。

涉及到多层嵌套递归调用的代码会有较长的处理时长，因此建议使用 Xcheck 扫描 Web 后台的上层应用代码，而不是某个库或框架。

Q: Xcheck 扫描一次需要多长时间？

A: 扫描时间受被测代码复杂度以及具体文件数量影响，通常一个小于 1M 大小的 zip 文件完成一次扫描仅需几秒。目前默认设置最长扫描时间不超过 5 分钟。超过 5 分钟自动结束扫描。

Q: Xcheck 是否支持 Android 应用漏洞扫描？

A: 即将支持 Android 应用扫描。