

基于IAST技术

灰盒安全测试工具产品分析

目录 CONTENT

- 01 | 灰盒安全测试工具技术原理简介
- 02 | 灰盒安全测试工具常见功能
- 03 | 灰盒安全测试工具优势及不足
- 04 | 在企业内落地实施建议



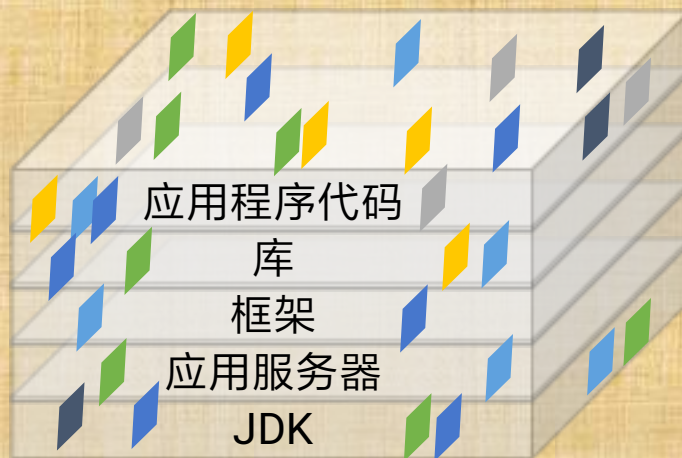
01

灰盒安全测试工具技术原理简介

一个常见的术语

- 灰盒工具通过在字节码中插桩检测探针，在应用程序运行过程中，通过探针采集各种运行时的上下文信息

深度
插桩



不同语言的实现

JAVA

1. 通过Java Agent方式实现 (Agent利用JVM TI暴露的一些接口)
2. 使用字节码修改框架, 例如ASM
3. 使用Agent OnLoad或OnAttach机制

PHP

1. 通过PHP扩展库来实现
2. 通过扩展库获取请求和响应信息
3. 通过扩展库对字符串变量进行标记 (xmark, php7扩展库)

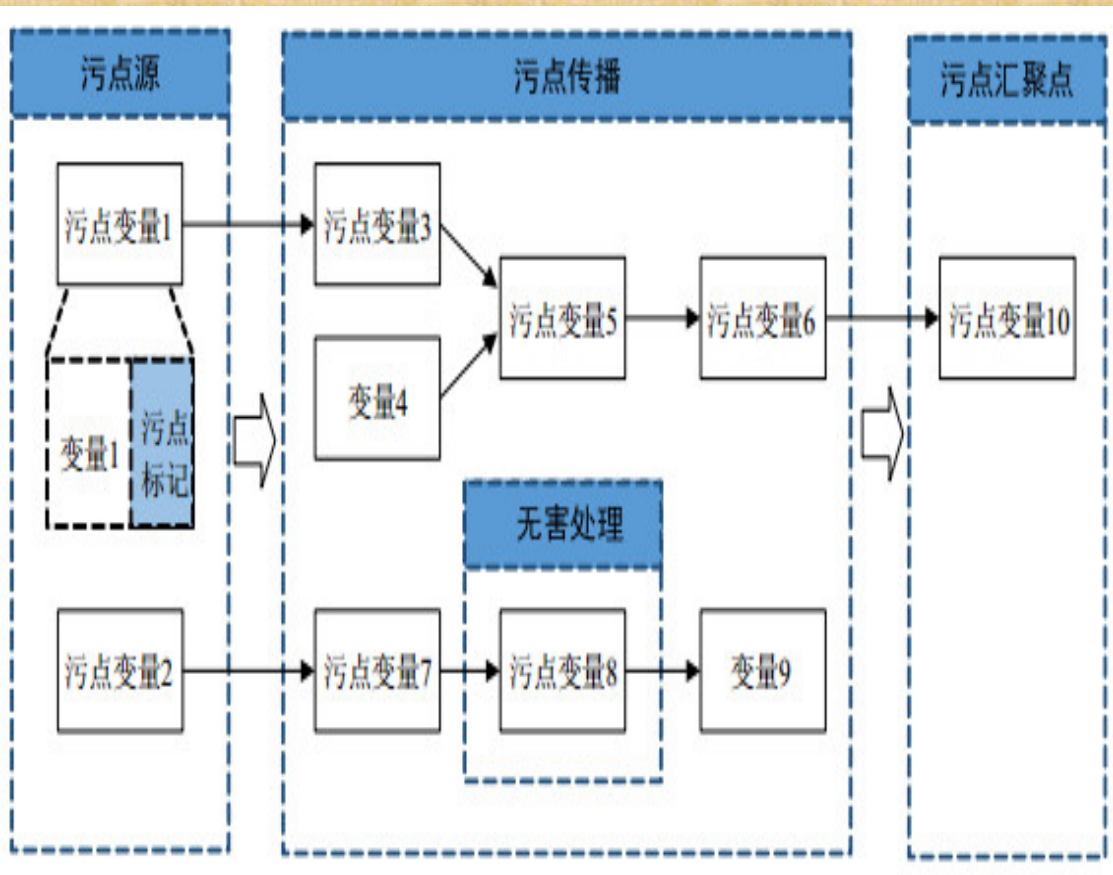
.NET

1. 通过IHostingStartup (承载启动) 实现
2. 继承IHostingStartup可以注册插件并在项目启动时自动加载
3. 更多信息请查询microsoft官方文档-ASP.NET Core 中使用承载启动程序集

基础

对编程语言的危险底层函数进行HOOK

污点跟踪技术



```
输入
javax.servlet.ServletRequestWrapper.getParameterValues(userid)
> getParameterValues()@ServletWebRequest.java line: 159
[admin]

传播阶段
java.lang.StringBuilder.append(admin)
>
SELECT * FROM user_data WHERE userid = admin

java.lang.StringBuilder.toString()
>
SELECT * FROM user_data WHERE userid = admin

输出
org.hsqldb.jdbc.JDBCStatement.executeQuery(SELECT * FROM user_data WHERE userid = admin)
> injectableQuery()@SqlInjectionLesson5b.java line: 69
(SELECT * FROM user_data WHERE userid = admin)
```

提示

针对组件做专门的研究，制订针对性的跟踪策略，工具才能很好发现使用该组件后的弱点（例：SpringCloud）

三种检测模式

基于插桩技术

使用污点跟踪技术，在程序运行之前或运行时，往程序中写入检测代码，从而跟踪程序数据流程，据此发现可能存在的漏洞

未知模式

More.....



以流量为检测对象

通过设置代理、流量镜像等方式，以应用系统的请求/响应流量为检测对象，据此发现可能存在的漏洞

基于插桩技术和爬虫技术

在被动污点跟踪模式的基础上，增加爬虫功能，并能发payload攻击



02

灰盒安全测试工具常见功能

应用识别功能

— 发现被测应用的基本信息

- 识别应用的名称
- 识别中间件类型及版本
- 某些工具能自动识别，可减少配置步骤
- 某些工具只能通过先行配置才能识别，安装过程会稍麻烦
- 如果灰盒工具未明确支持某中间件，则会导致识别不准确的问题或无法安装Agent

数据流跟踪功能

— 用于漏洞发现的基础功能

```
javax.servlet.ServletRequestWrapper.getParameterValues(userid)
> getParameterValues()@ServletWebRequest.java line: 159
[admin]
```

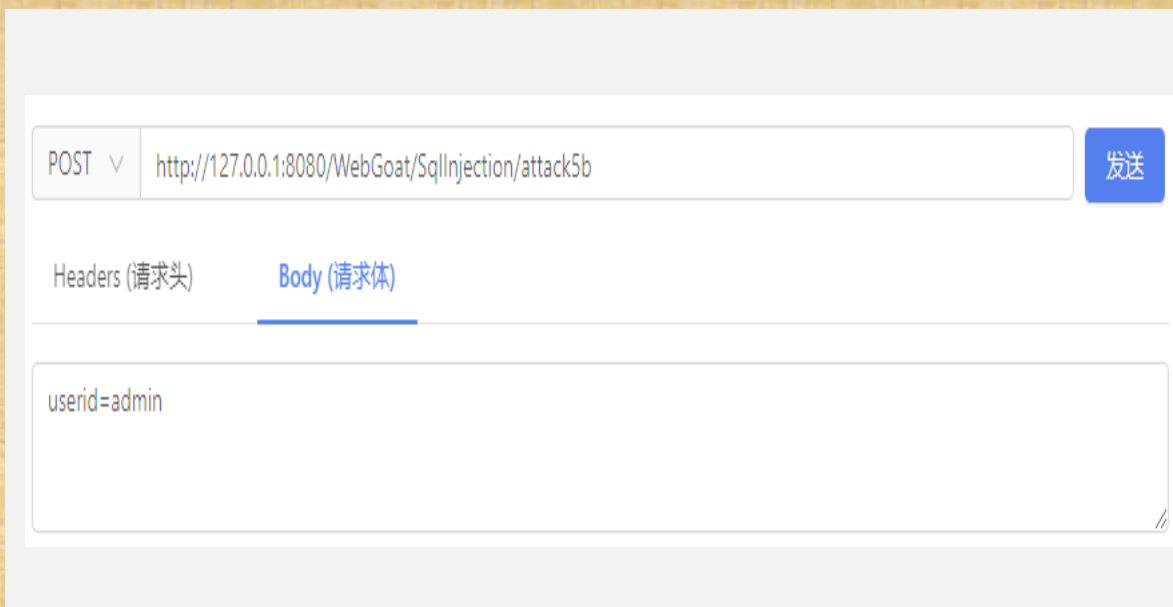
```
java.lang.StringBuilder.append(admin)
> SELECT * FROM user_data WHERE userid = admin
```

```
org.hsqldb.jdbc.JDBCStatement.executeQuery(SELECT * FROM user_data WHERE userid = admin)
> injectableQuery()@SqlInjectionLesson5b.java line: 69
(SELECT * FROM user_data WHERE userid = admin)
```

- 基于污点跟踪技术的数据流跟踪是发现弱点、识别弱点的基础
- 数据流颗粒度越细，漏洞定位越准确，同时带来的性能损耗也越大
- 工具应该提供不同的跟踪深度供选择
- 数据跟踪流是确认漏洞和修复漏洞的重要依据，工具应该能跟踪到用户编写的代码，而不能仅限于框架或组件的代码

漏洞验证功能

— 帮助降低工作量



The screenshot shows a web proxy tool interface. At the top, there is a dropdown menu set to 'POST' and a text input field containing the URL 'http://127.0.0.1:8080/WebGoat/SqlInjection/attack5b'. To the right of the URL field is a blue button labeled '发送' (Send). Below the URL field, there are two tabs: 'Headers (请求头)' and 'Body (请求体)'. The 'Body (请求体)' tab is selected and underlined. The body content area contains the text 'userid=admin'. At the bottom right of the body area, there is a double-slash icon '//'.

- 目前暂没有工具能做到100%准确度，在得到大量的测试结果后，能进行漏洞验证是必要的功能
- 漏洞验证包含两个层次：检测结果验证（自动和手动），修复验证
- 可以例举出可用于验证的payload
- 漏洞验证存在一些限制，比如session过期、应用系统本身重放限制

消除误报功能

— 提供基本的规则配置能力

- 提供清晰的规则设置功能，帮助消除常见漏洞的误报
- 规则配置要简洁、易懂
- 提供灵活的配置文件，通过配置文件设置复杂的检测规则
- 漏洞类型可开关，用于SDL实施过程中开展漏洞专项排查

防止脏数据产生

— 避免污染测试数据

节点配置

节点总开关:

检测模式: 交互式缺陷定位 动态污点追踪

应用运行监控:

脏数据开关:

- 使用被动的污点跟踪检测模式并不会产生脏数据
- 使用主动污点跟踪模式检测, 由于有爬虫和攻击流量, 会产生脏数据
- 提工具应该在脏数据进入数据库之前拦截, 使用插桩技术可实现

发现敏感信息泄露

一 检测敏感信息传播

- 使用插桩技术检测敏感信息具备天然优势
- 可从输入、输出、数据传播方面发现敏感信息
- 可用于发现应用不同模块之间的敏感数据流转，避免仅用于本模块的数据意外进入其他模块
- 可结合数据保护相关合规制度、行业标准做检测，方便合规要求落地

检测第三方组件

— 发现第三方组件及组件漏洞

库名	等级	CNNVD	CVE
classmate-1.3.4.jar	E	3	3
xstream-1.4.7.jar	E	3	3
tomcat-annotations-api-8.5.29.jar	E	20	20

- 组件引入即可被发现
- 发现组件的版本信息及引入代码点
- 关联组件CVE、CNNVD漏洞
- 插桩组件自身类，发现潜在漏洞
- 组件数据、CVE数据、CNNVD数据是该功能的基础，需要有机制去保障数据能及时更新

弱点描述和修复建议

— 帮助修复应用弱点

当需要在代码中拼接sql语句时，也要使用预编译sql语句（PreparedStatement）和参数化查询，

```
String firstname = req.getParameter("firstname");
String lastname = req.getParameter("lastname");
String query = "SELECT id, firstname, lastname FROM authors WHERE forename
PreparedStatement pstmt = connection.prepareStatement( query );
pstmt.setString( 1, firstname );
pstmt.setString( 2, lastname );
try
{
ResultSet results = pstmt.execute( );
```

- 弱点的描述和修复建议应该是本地化
- 弱点的描述和修复建议应该与具体的弱点强相关
- 应该包含对应的代码示例
- 弱点的描述和修复建议应该是清晰易懂的
- 良好的弱点描述和修复建议能帮助减轻安全人员工作负担
- 大部分产品不太重视



03

灰盒安全测试工具优势及不足

不独立占用时间

- 在开发和测试阶段均可以无缝集成，“零成本”实现代码审计、安全测试和第三方软件检测



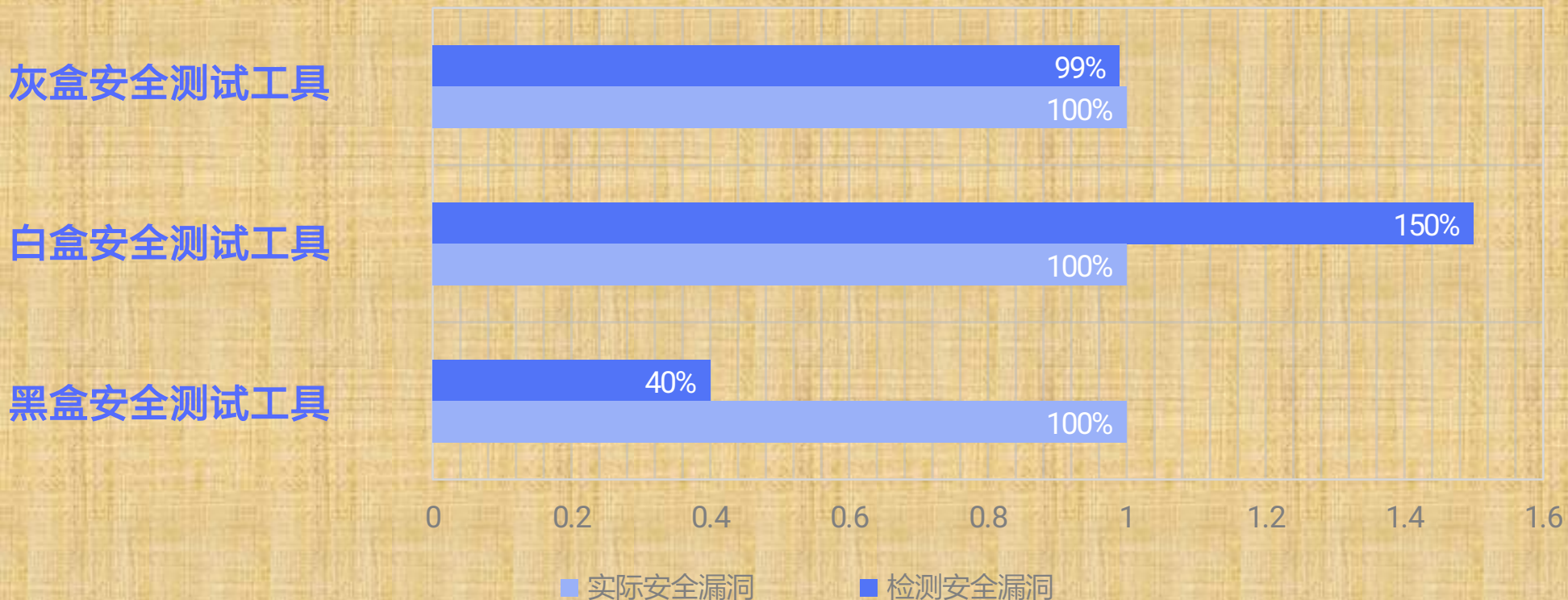
弱点信息丰富

- 在弱点信息越全，越有助于确认和修复弱点

	SAST	DAST	IAST
http 请求		√	√
http 响应		√	√
数据流	√ (静态)		√
第三方软件和框架			√
配置信息			√
DB连接信息			√

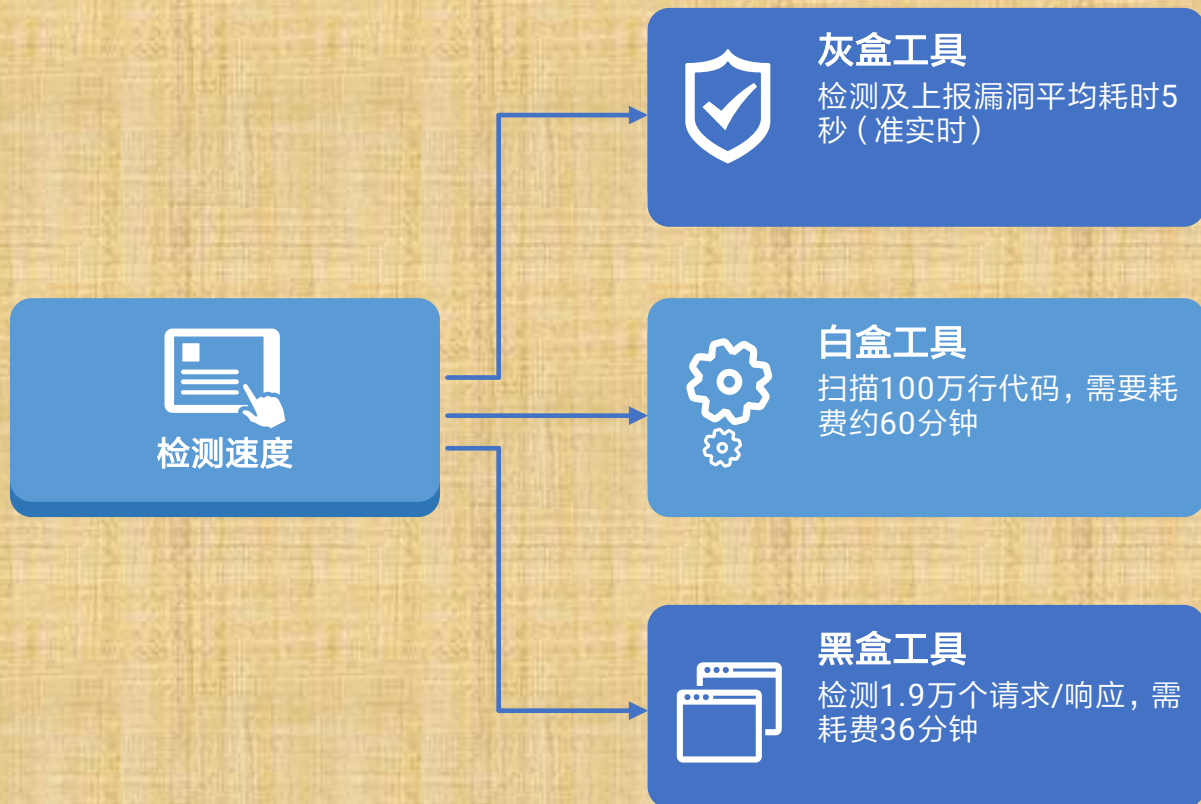
误报率较低

- 基于丰富的弱点信息，相对于白盒、黑盒测试工具，误报率较低



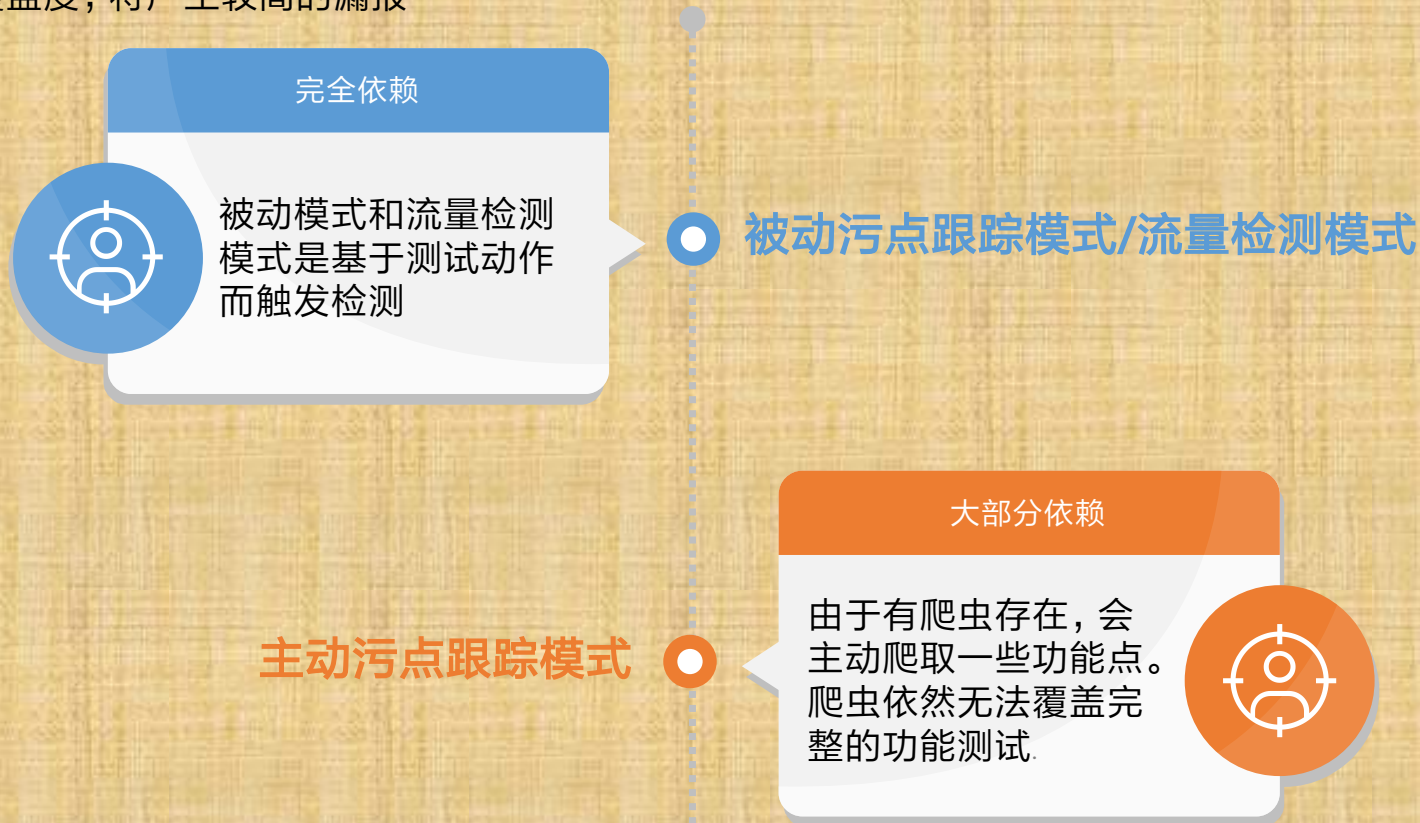
检测速度较快

- 得益于插桩技术，使得安全测试可以与普通功能测试同步完成



依赖测试覆盖度

- 较低的测试覆盖度, 将产生较高的漏报



Agent性能问题

- 实际的Agent性能与理论值存在些许差异（可以使用压测工具、黑盒工具来验证性能）



对被测应用的性能影响在3%-5%之间

理论



性能与一些因素相关
1、API交互链条长短
2、应用的大小量级
3、框架的复杂程度

实际

对环境要求较高

- 使用灰盒测试需要满足较多的先天条件

01



编码语言有要求 (Java、C#、NodeJS)

03



不支持的框架无测试效果 (可能出现冲突、无结果或结果很差)
(例如: 微服务框架、OSGi框架)

02



必须运行到特定的中间件中
(Tomcat、jetty、Jboss...)

04



不支持的组件无测试效果 (可能出现冲突、无结果或结果很差)
(例如: XStream)

对应用强侵入

- Agent需部署到应用中间件中或附加到JVM进程中



人员配合难



与白盒相比，想支持新的漏洞，必须依赖供应商

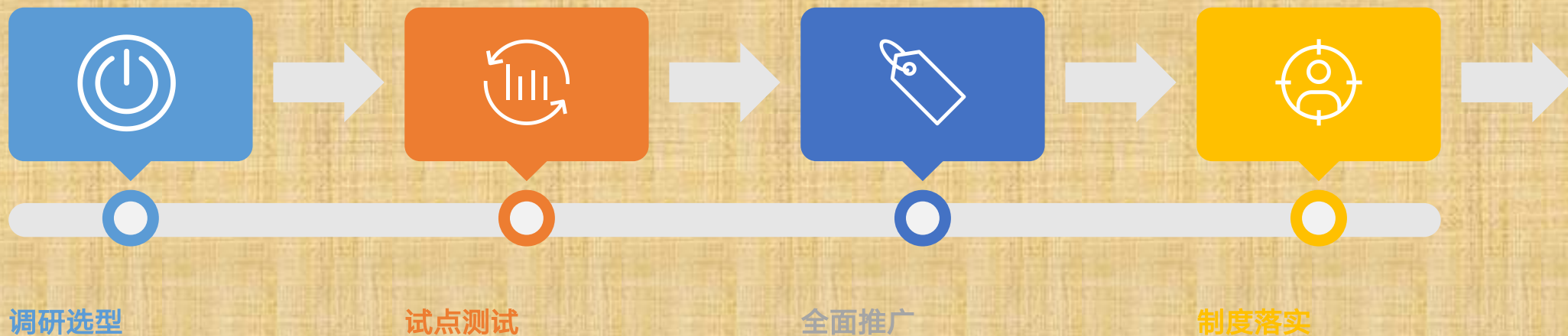


04

在企业内落地实施建议

工具引入过程

- 常规新技术、新工具在企业内的引进过程



试点测试阶段

- 难免要“找关系”



全面推广阶段

- 向自动化靠拢

01

部署自动化

将Agent部署任务集成到自动化平台（devops平台），使用自动化脚本自动下载Agent，找到JVM进程，附着进去



03

测试自动化

采用自动化测试脚本测试，可以有效覆盖应用功能，弥补灰盒工具“漏报”的不足



02

部署自动化

不具备自动化平台，可采用“抱大腿”方式。比如监控平台，通过监控平台下发脚本，执行Agent下载和JVM进程附着的操作



04

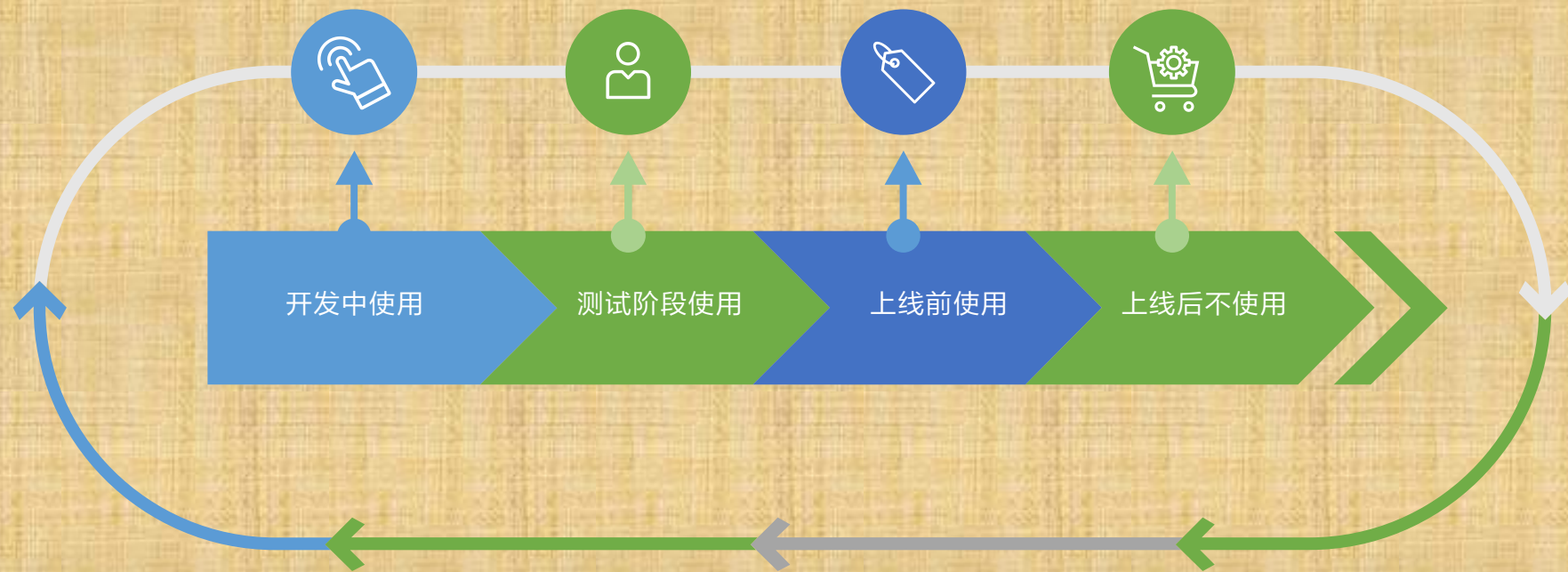
测试用例全面

如果不具备自动化测试条件，则要把测试用例覆盖全面，减少“漏报”



制度落实阶段

- 把灰盒工具的使用写入内控制度中



谢谢

2021年3月