精简版 SDL 落地实践

Author:@好好学习英语的 abc

前言

一般安全都属于运维部下面，和上家公司的运维总监聊过几次一些日常安全工作能不能融入到 DevOps 中，没多久因为各种原因离职。18 年入职 5 月一家第三方支付公司，前半年在各种检查中度过，监管形势严峻加上大领导对安全的重视(主要还是监管)，所有部门 19 年的目标都和安全挂钩。由于支付公司需要面对各种监管机构的检查，部分安全做的比较完善，经过近一年对公司的熟悉发现应用安全方面比较薄弱。这部分业内比较好的解决方案就是 SDL，和各厂商交流过之后决定自己照葫芦画瓢在公司一点一点推广。

一、精简版 SDL

1. TRAINING	2. REQUIREMENTS	3. DESIGN	4. IMPLEMENTATION	5. VERIFICATION	6. RELEASE	7. RESPONSE
1. Core Security Training	2. Establish Security Requirements	5. Establish Design Requirements	8. Use Approved Tools	11. Perform Dynamic Analysis	14. Create an Incident Response Plan	Execute Incident Response Plan
	3. Create Quality Gates/Bug Bars	6. Perform Attack Surface Analysis/ Reduction	9. Deprecate Unsafe Functions	12. Perform Fuzz Testing	15. Conduct Final Security Review	
	4. Perform Security and Privacy Risk Assessments	7. Use Threat Modeling	10. Perform Static Analysis	13. Conduct Attack Surface Review	16. Certify Release and Archive	

上图为标准版的 SDL，由于运维采用 DevOps 体系，测试也使用自动化进行功能测试，版本迭代周期比较快，安全人手不足加上对 SDL 的威胁建模等方法也一头雾水、如果把安全在加入整个流程会严重影响交付时间。在这种情况调研了一些业内的一些做法，决定把 SDL 精简化 。精简版 SDL 如下：

二、精简版 SDL 落地实践

1 安全培训

SDL 核心之一就是安全培训，所以在安全培训上我们做了安全编码、

安全意识、安全知识库、安全 SDK

1.1 安全编码：

我们在网上找了一些 java 安全编码规范、产品安全设计及开发安全规范结合公司实际业务出了一版。


『JAVA安全编码规范』和『产品设计及开发安全规范』运行通知

各位好：

为规范编码、产品设计及开发安全，减少产品设计缺陷和编码层面产生的漏洞，降低被黑客恶意利用攻击的风险。安全部特编写《JAVA安全编码规范》、《产品设计及开发安全规范》供大家参考使用。详见附件或点击链接：《JAVA安全编码规范》、《产品设计及开发安全规范》

因为各种监管机构对培训都有要求，借此推了一下安全培训，定期对开发和新员工入职的培训。


答复 全部答复 转发

6/17 (周一)

需后续工作。 开始时间: 2019年6月17日星期一。 到期时间: 2019年6月17日星期一。

如果显示此邮件的方式有问题，请单击此处以在 Web 浏览器中查看该邮件。

单击此处可下载图片。为了帮助保护您的隐私，Outlook 禁止自动下载该邮件中的某些图片。

	系统性风险	安全部	代码安全	现场ppt	必修	公司内
			安全意识	现场ppt	选修	公司内

1.2 安全意识：

公司有企业微信公众号，大部分员工都关注了，在公众号推广了一波。

20[redacted]安全宣传周系列动漫①WiFi安全篇

2019-03-18 [redacted]安全部

为什么要做网络安全推广？

近年来，信息安全事件层出不穷、商业泄密触目惊心、个人信息唾手可得、

网络犯罪蒸蒸日上信息安全面临威胁无处不在。那么，这一切的一切究竟是什么原因造成的呢？所有的一切都是因为黑客吗？不！错了，黑客虽然可怕，可更多时候，内部人员威胁却更容易被忽略，但却更容易造成伤害，无数信息安全事件用血淋淋的教训告诉我们，员工信息安全意识薄弱是最根本原因。比如，将口令写在便签上，贴在电脑监视器旁；开着电脑离开，不锁屏，就像离开家却忘记关门那样；轻易相信来自陌生人的邮件，好奇打开邮箱附件；使用容易猜测到的弱口令，或者压根不设口令；丢失自己的笔记本电脑；所以，员工是信息安全的最后一道防线，为此，安全部向大家推广网络安全宣传周系列漫画，让大家充分认识到每个人都肩扛着信息安全的责任，深刻体会"信息安全，人人有责"！

接下来的一周里，我们会连续五天推出网络安全系列漫画，欢迎大家收看哦。

周一：WiFi安全篇
周二：邮件安全篇
周三：个人电脑安全篇
周四：办公区域安全篇
周五：日常交流篇

如今，随着移动终端的兴起和互联网技术的不断进步，那些古老的盗窃、诈骗、骚扰手段也换了新颜。不知道大家有没有意识到，大家每天使用的WiFi，其实并不安全。

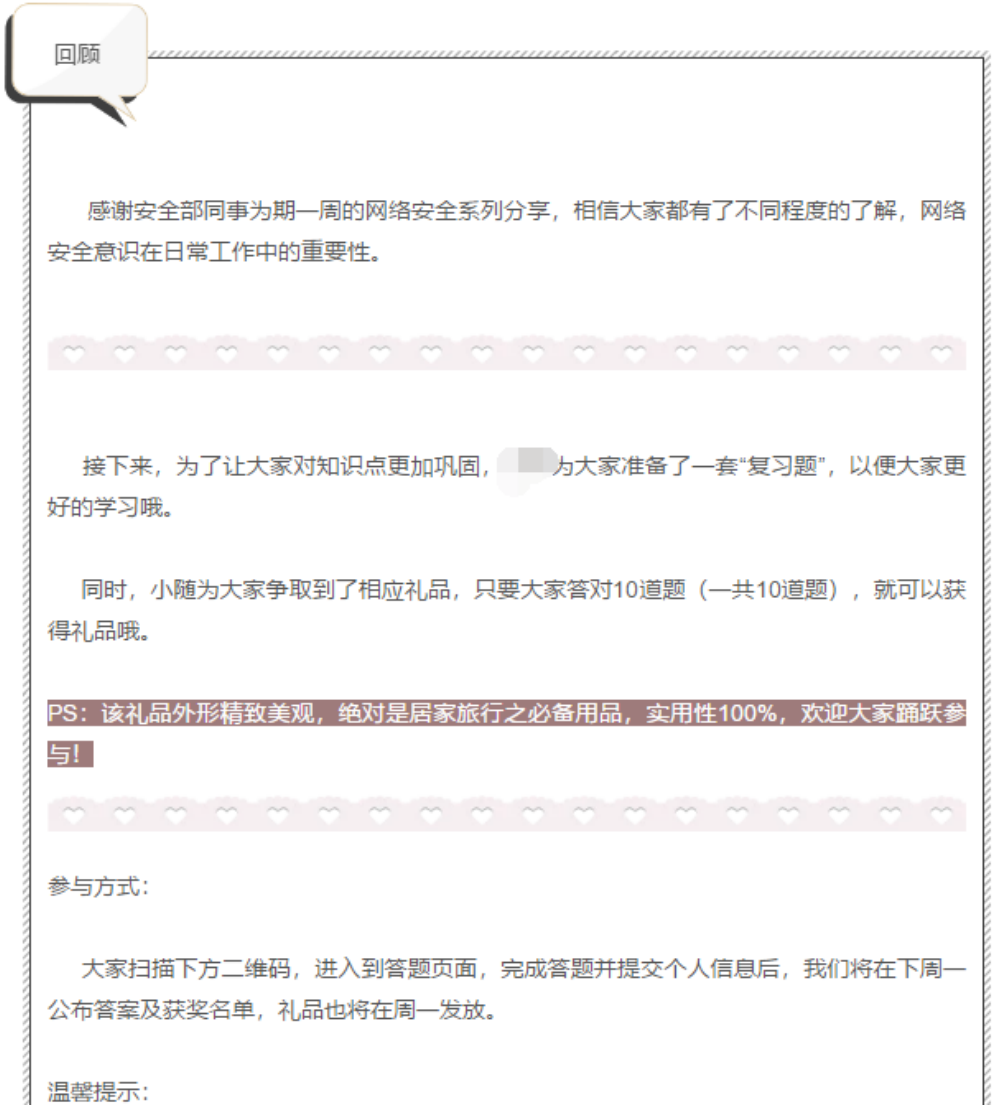
是的，你没看错，使用WiFi上网时，我们的个人信息安全或许面临着巨大的风险。

比如，黑客自己搭建一个"山寨WiFi"，取一个与附近WiFi相似的名字，不设登录密码诱导人们连接。使用时，传输的数据就会被黑客监控，个人隐私、账号名和密码等信息，也可以轻易被盗取。其实，不仅是蹭WiFi，不少人还喜欢随时开着手机无线网自动连接功能，这样风险就更大了。

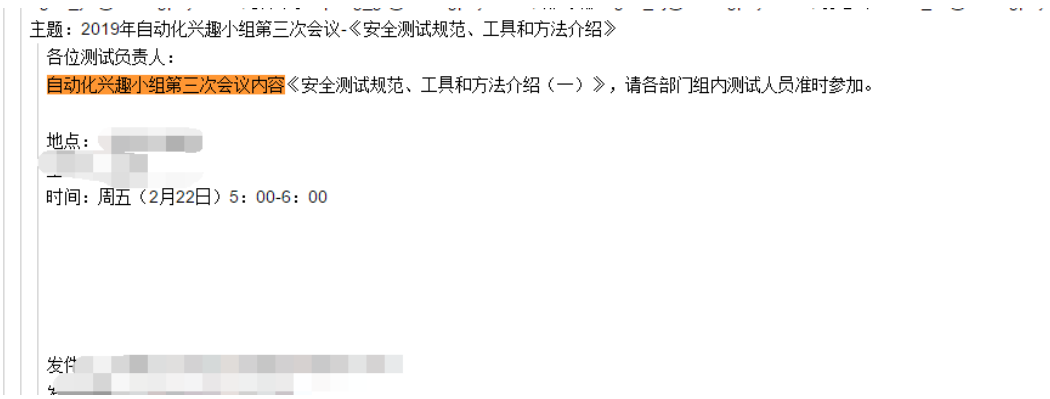
不信？看看下面这几幅漫画，这些危险场景，或许你都曾经置身其中。

免费WiFi接入

宣传完之后答题，答题满分送小礼品。



因为人手不足，而功能测试和安全测试本质上有很多相通的地方，测试部门也比较配合，针对测试人员做了一些安全测试相关的培训，但是效果并不是太理想。



1.3 安全知识库：

在漏洞修复过程中，开发很多不太了解漏洞原理、修复方案，所以我

们建立了安全知识库，开发先到安全知识库查相关解决方法。找不到的再和安全人员沟通，安全人员对知识库不断更新，形成一个闭环。

1.4 安全 SDK

由于公司有架构部门，开发框架基本是架构部门提供。我们将一些常见的漏洞和架构部门沟通之后，让架构将一些漏洞修复方式用 SDK 实现，开发只需要导入 JAR 包，在配置文件中配置即可。其中也挺多坑的，需要慢慢优化。

3. 导入依赖包

```
compile '                                    -starter-xss:xxx"
```

通过 h   取最新版本的jar。

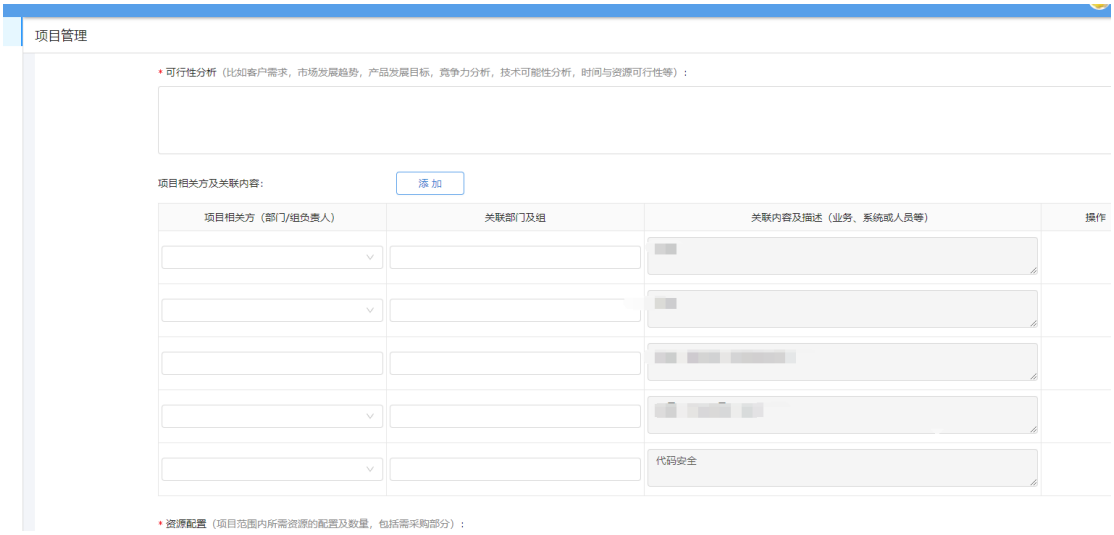
4. 使用方法

在application.yml中增加下面内容：

```
xss:
  enabled: true # 是否启用，默认值为true
  filter-name:          # 过滤器名称，默认值为
  order: 0 # 执行顺序，默认值为0
  url-patterns: # 匹配路径，默认值为[/*]
  - '/*'
  ignoreUrls: # 忽略路径
  - '/js/*'
  - '/css/*'
  - '/images/*'
  - '/static/*'
  - '/webjars/*'
```

2 安全需求设计

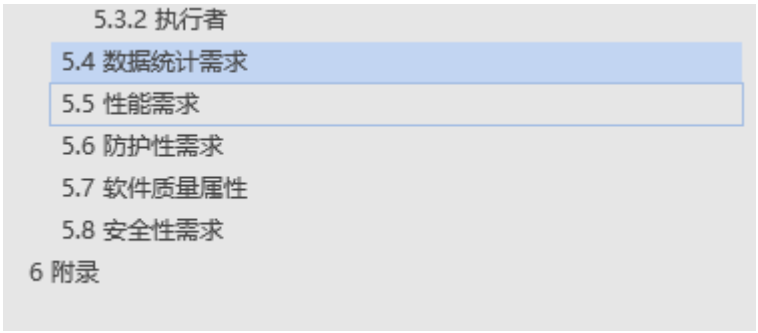
公司有项目立项系统，所有的项目立项都需要通过系统来进行立项，安全为必选项，评审会安全也必须要参与。



这个时候基本上项目经理会找安全人员进行沟通，copy了一份VIP的产品安全设计规范，根据需求文档和项目经理确定安全需求。

编号	类别	概述	细则	备注
L01	认证与鉴权	帐号锁定	除公司会员系统之外提供外网访问功能的系统，必须启用帐号登录失败锁定策略（如：3分钟20次登录失败，锁定30分钟）	
L02		错误提示	用户名或密码错误时，返回的提示信息必须一致（如："错误的用户名或密码"）	
L03		登录与注销	有登录功能的系统必须同时有注销功能	
L04		后台页面	后台页面必须对用户身份和访问权限进行检查	
L05	验证码	管理界面	管理后台的登录界面必须设置验证码	
L06		有效期	验证码必须设置有效期（有效时间和错误次数）	
L07		发送频率	使用短信/邮件验证时，必须限制同一ID或接收者的验证码发送频率	
L08	会话安全	会话超时	会话token/session必须设有超时机制	
L09		会话更新	用户登录成功后，必须更新会话ID；用户注销后，必须强制session/token过期	
L10	Cookie	HTTP Only	cookie参数中Session Id等认证相关的字段必须设置HTTP Only	
L11	上传下载	文件判断	对上传文件后缀进行白名单限制，严格判断文件内容与类型是否匹配	
L12		目录跳转	禁止客户端自定义文件下载路径（如：使用../../../../../进行跳转）	
L13		目录权限	存储上传文件的目录必须禁止脚本执行权限	
L14	传输安全	参数提交	禁止通过HTTP GET方式提交**不安全算法**[1]处理过的用户密码	
L15		明文传输	禁止在未加密的HTTP协议中明文传输用户登录密码、支付密码、银行卡卡号、有效期、持卡人姓名、身份证号码、CVV等交易敏感数据。会员系统、支付系统还应在此基础上进一步**增强安全措施**[2]。	
L16		支付安全	禁止在支付密码的传输过程中使用**不安全算法**[1]	
L17	存储安全	敏感数据存储	禁止数据库、日志文件中明文存储用户支付密码、银行卡卡号、有效期、持卡人姓名、身份证号码等交易敏感数据。禁止存储信用卡CVV信息。禁止使用**不安全算法**[1]存储用户身份校验凭据，如：密码。会员系统、支付系统还应在此基础上进一步**增强安全措施**[2]。	
L18	日志审计	审计内容	自建用户系统，必须记录：时间/用户ID/界面(Web或APP)/结果（成功或失败）/IP等信息	
L19		日志清除	除审计用户外，其他人员不应具备日志修改、删除或清空的权限。必须记录清空日志的行为	
L20		日志存储	禁止将日志直接保存在可被浏览器访问到的WEB目录中	
L21	其它	后门	禁止在代码中留置后门	
	备注[1]	不安全算法	明文、标准MD5算法、Base64编码、私有算法等。	
	备注[2]	增强安全措施	参考等级保护、PCI-DSS、ADSS等法规和标准并严格执行安全编码规范	

确认好安全需求之后将按需求加入到需求文档，并确认安全测试时间，此流程只针对新项目，已经上线的项目的需求并未按照此流程，后续在安全测试时候会讲到这部分的项目是怎么做的。



三、开发、安全测试

安全测试主要分为代码审计，漏洞扫描，手工安全测试。由此衍生出来的安全产品分为 3 类。DAST：动态应用程序安全测试（wvs,appscan）、SAST：静态应用程序安全测试(fortify,rips)、IAST：交互式应用程序安全测试（seeker，雳鉴），这三种产品的详细介绍可以参考 https://www.aqniu.com/learn/46910.html，下图为三种产品的测试结果对比。

对比项	DAST	SAST	IAST
研发流程集成	测试阶段/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
检测速度	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
漏洞检出率因素	与测试playload覆盖度相关，企业可优化和扩展	与检测策略相关，企业可在当前数据流基础上定制策略	与检测策略相关，企业可在当前数据流基础上定制策略
第三方组件漏洞检测	支持	不支持	支持
使用成本	较低，漏洞基本无需人工验证	高，需要人工排除误报	低，基本没有误报
支持语言	不区分语言	区分语言，不同工具支持的语言不同	区分语言，不同工具支持的语言不同
支持框架	不区分框架	区分框架，不同工具支持的框架不同	区分框架，不同工具支持的框架不同
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

这几类产品实现了自动化可以继承到 DevOps 中。接下来我们将这些工具融入到开发测试阶段。

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。一些调研过的产品如下图，具体测试结果就不公布了。

工具	支持语言	版权	主页地址
contrast	java,node	收费	https://contrastsecurity.com
seeker	java,node	收费	https://www.synopsys.com
CxIAST	java,node	收费	https://www.checkmarx.com/
vulhunter	java	收费	http://www.seczone.cn/
openrasp	java,php	开源	https://rasp.baidu.com/
雳鉴	不受语言限制	收费	https://www.moresec.cn/
GourdScar	不受语言限制	开源	https://github.com/ysrc/GourdScanV2
洞鉴(X-Ra	不受语言限制	开源	https://chaitin.github.io/xray/#/

2.1 开发阶段

在对几类产品调研的时候发现 IAST 的插桩模式可以直接放到开发环境，开发环境和测试环境的代码区别主要还是在于 application.yml 配置文件，所以可以提前将该模式放到开发阶段。

开发写完代码提交到 gitlab 部署到开发环境启动应用的时候，开发需要验证一下功能是否可用，这个时候就可以检测出是否存在漏洞。

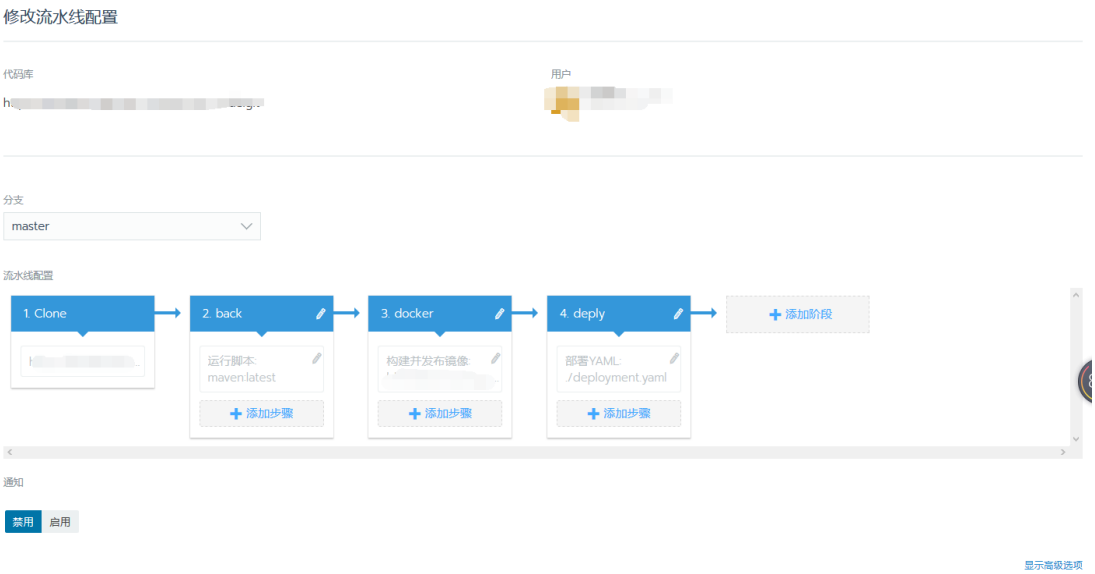
公司在测试环境使用 rancher，把 IAST 的 jar 包放入到项目的 gitlab,在部署的时候把代码拉到本地，通过修改 Dockerfile 文件把 jar 包添加到容器。

- ADD shell/xxx.jar /home/app/xx/lib

由于公司项目基本统一使用 spring-boot，所有的项目都通过一个 start.sh 脚本来启动应用，start.sh 和 Dockerfile 一样需要添加到项目的 gitlab，同时修改 start.sh 脚本文件即可。

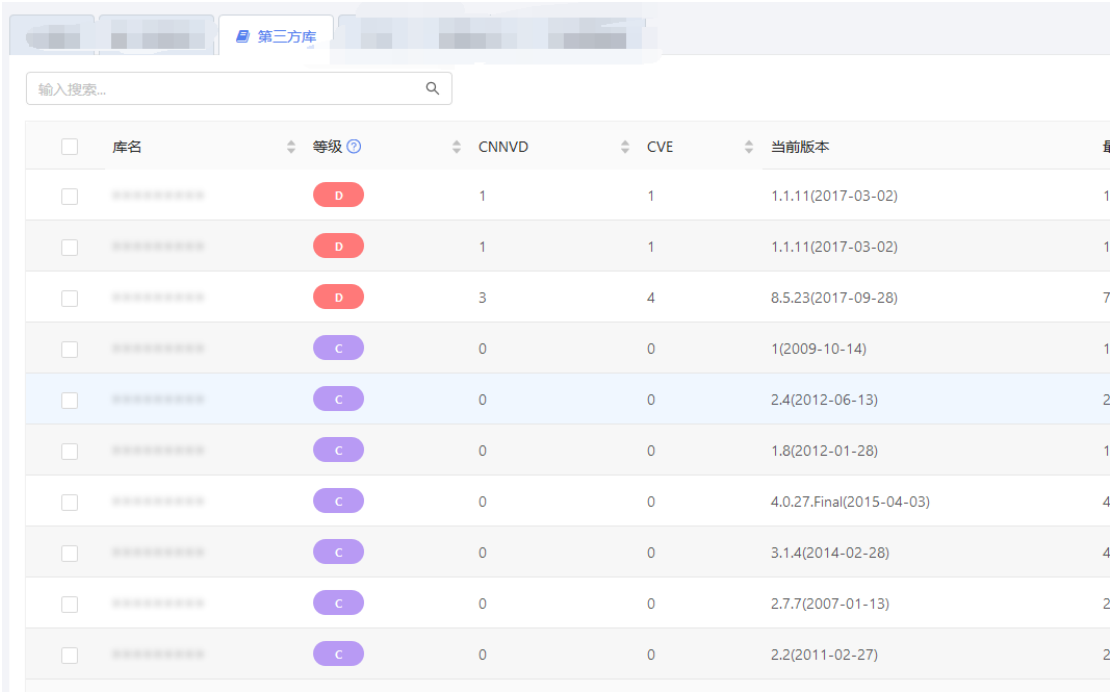
```
-javaagent:$APP_HOME/lib/xx.jar  -jar $APP_HOME/app/*.jar --
spring.profiles.active=dev >$APP_HOME/logs/startup.log 2>&1 &
```

测试项目如下，忽略错别字：



开发访问一下正常的功能即可在是否存在漏洞。

部分产品同时还会检测第三方组件包。



公司使用 harbor 来对镜像进行当仓库镜像，项目部署完成之后会打包成一个镜像上传到 harbor，harbor 自带镜像扫描功能。

2.2 测试阶段

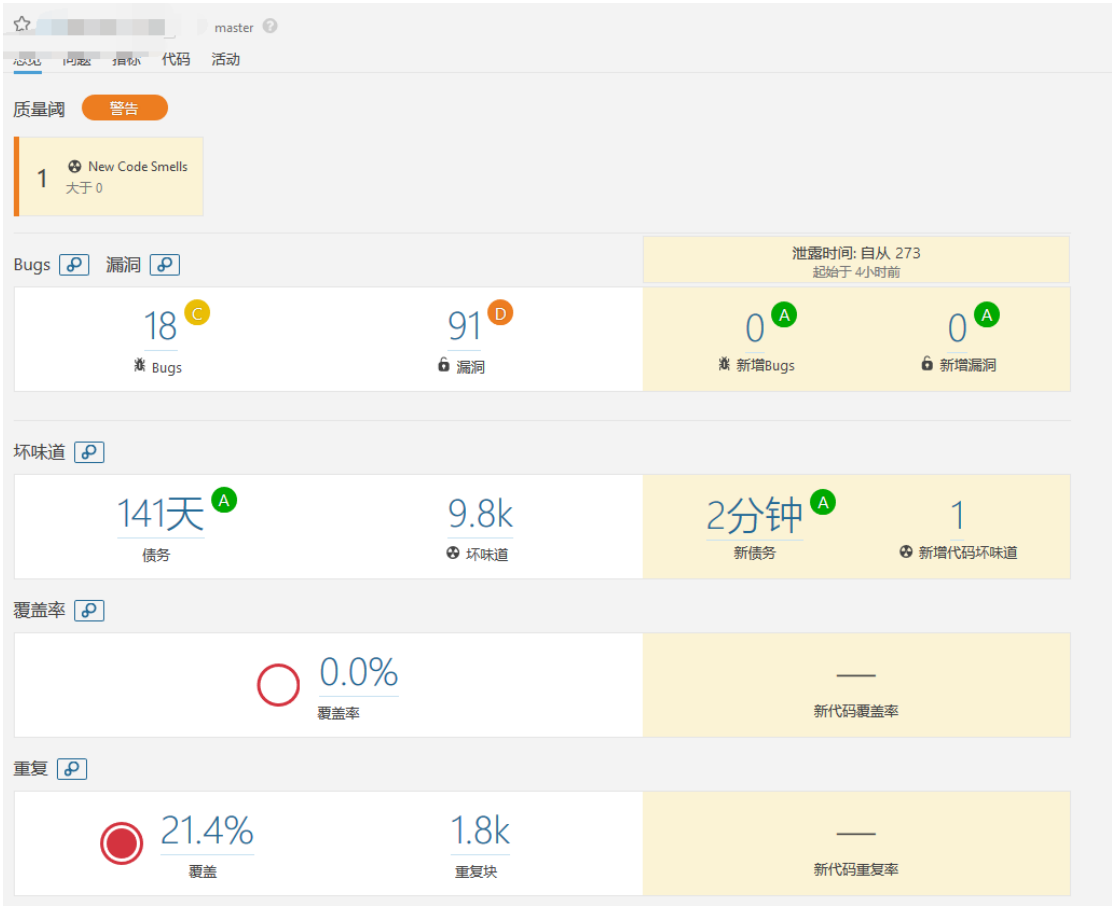
开发完成之后进入到测试阶段。这个阶段我们进行静态代码扫描，功能测试，安全测试。

2.2.1 静态代码扫描

利用静态代码扫描工具对代码在编译之前进行扫描，并在静态代码层面上发现各种问题，其中包括安全问题。部分工具列表：

工具	支持语言	版权	主页地址
Fortify	大部分语言	收费	http://www8.hp.com/us/en/software-solutions/static-code-analysis-sast/
Checkmarx	大部分语言	收费	https://www.checkmarx.com/
Flawfinder	C/C++	免费	http://www.dwheeler.com/flawfinder/
LAPSE	Java	免费	http://www.owasp.org/index.php/Category:OWASP_LAPSE_Project
Brakeman	Ruby on Rails	免费	https://github.com/presidentbeef/brakeman

由于预算有限，静态代码扫描采用 sonarQube 集成，我们使用的是 FindbugSecurity,精简规则，然后在持续构建过程中，进行静态代码 bug,安全扫描。

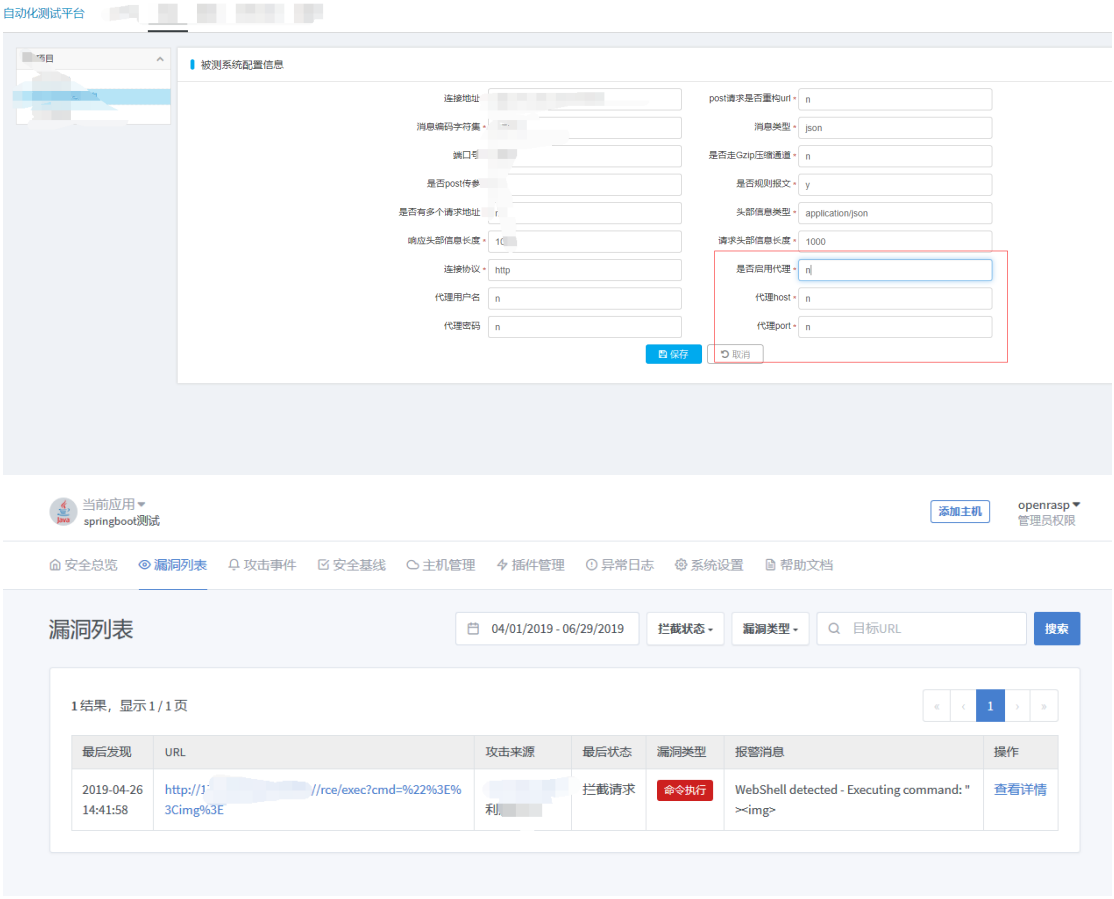


静态代码扫描的同时也可以扫描第三方依赖包，OWSAP 的 Dependency-Check 就可以集成到持续构建过程中，由于 IAST 类产品支持该功能，不多做介绍。

2.2.2 功能测试

功能测试方面，公司测试部门实现了自动化测试平台，前期我们并未使用 agent 的方式检测，一开始使用开源的 gourdscan 加上 openrasp，利用 openrasp 的默认开启不拦截模式和漏

洞记录功能来检测服务端无返回的漏洞。



后来测试反馈扫描的脏数据太多，效果也并不是很好，就放弃了此方案。改用开发阶段的 IAST 的插桩方式，同样在测试环境也和开发环境一样利用 agent 来检测问题。功能测试完成之后。由于测试人员对漏洞并不是太理解，所以定的流程为测试人员到平台查看报告和安全人员沟通哪些问题需要修复，然后将问题写入到测试报告。

各位好:

[redacted]经完成，附件[redacted]试报告》，以下为测试结论及测试概要。

一、 测试结论

测试已完成。

二、 测试概要

手工测试结果：

1、针对本次系统测试需求，执行用例数：2487个

2、共执行1次冒烟测试，以及2个轮次的测试；一次冒烟发现0个缺陷，一轮测试共发现缺陷35个，二轮测试共发现缺陷12个，二轮复测后无缺陷。

测试轮次	测试日期	手工用例数	UI自动化用例数	接口自动化用例数	回归接口自动化用例数	精减UI用例个数	预估节约时间（h）	测试结果分析						测试人员
								阻断	严重	一般	次要	无关紧要	合计	
冒烟测试	[redacted]					不涉及	0	0	0	0	0	0	0	[redacted]
第一轮	[redacted]		0	0	[redacted]	[redacted]	0	0	0	0	0	0	0	[redacted]
第[redacted]	[redacted]				0	不涉及	0	0	0	0	0	0	0	[redacted]

安全测试情况如下表所示：

安全用例分类	用例数量	测试人员	漏洞级别	漏洞数量	修复情况
跨站脚本	87	[redacted]	重要	1	已修复
Spring框架漏洞					
Cookie参数注入					
服务端请求伪造					
框架漏洞					
敏感数据存储					
Sql延时注入					
url重定向					
不安全的js代码库					
GET/POST混合使用					

三、 风险

由于安卓、ios机型较多，测试时无法全部覆盖市面上的所有机型，会有兼容适配问题

2.2.3 安全测试

在测试阶段已经将安全加入到整个流程里面，所有需求更改完成都需要通过功能测试，也就是所有的流程过一遍安全测试，这样安全人手也不是很足，决定采用内外服务区分的办法来确定是否需要安全人员介入。

2.2.4 漏洞管理

漏洞管理这一块制定了漏洞管理制度，根据影响程度对漏洞进行评级，严重漏洞必须改完之后才能上线，高中低危漏洞且影响较小需要排期。

四、监控

支付公司一般安全设备基本都有，这一块基本上将设备的 syslog 打到日志中心可视化，并定制对应的规则实现告警即可。

五、结束语

个人知识和经验不足对 sdl 的体系并不是很熟悉，没什么经验，所以只能做到目前的程度。后续还有许多地方可以优化，增加流程等。如果有什么好的建议欢迎交流。