

# ThoughtWorks®

*Build Security In*

---

# 内建安全的软件开发

---

演讲者：刘庆华

# 内容概要

---

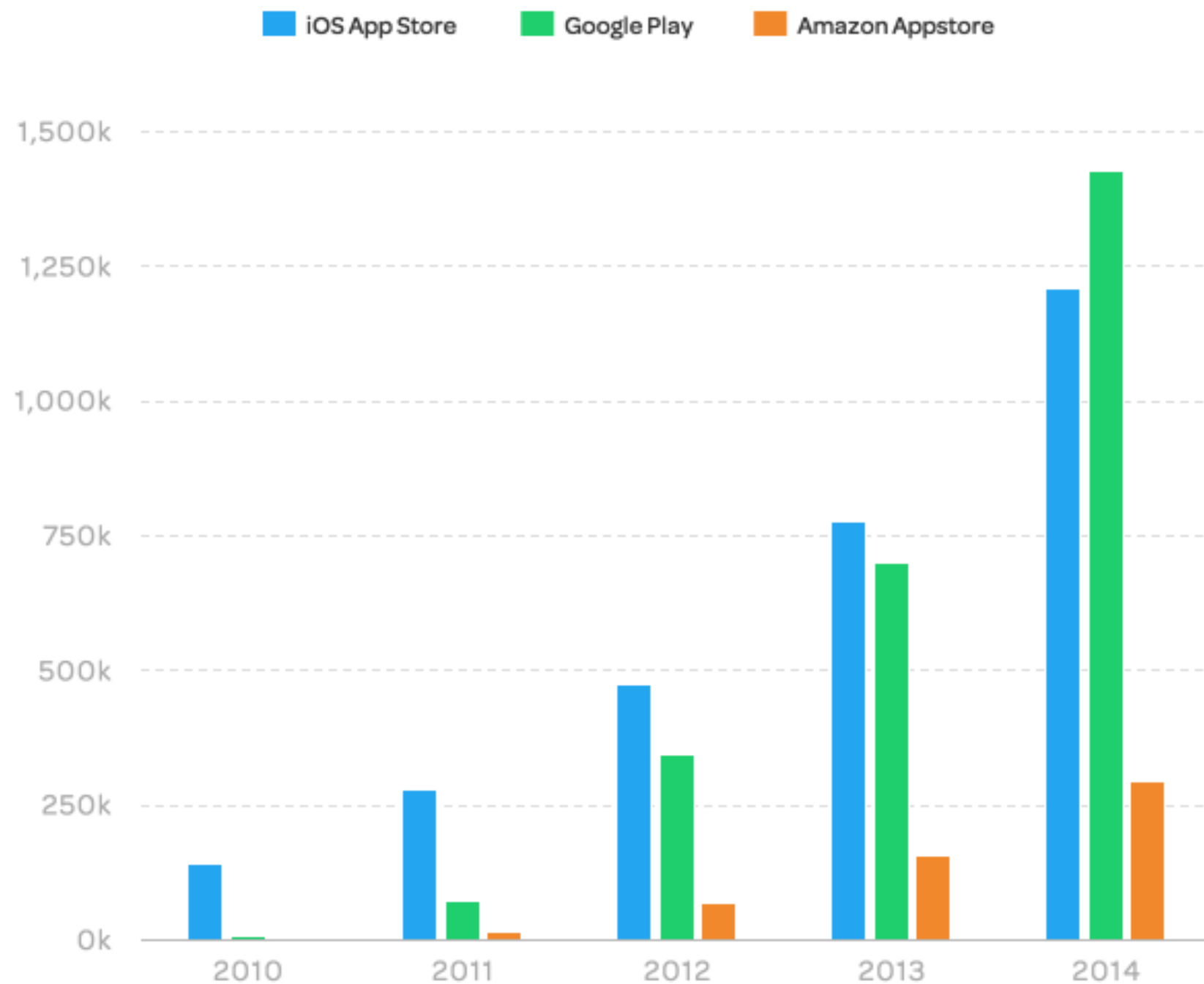
- 传统安全实践面临着严峻的挑战
- 为何安全漏洞如此难以消除？
- 如何更彻底地解决安全问题？
- 实践分享
- 总结
- 问答

# ThoughtWorks®

## 严峻的安全挑战

# 应用数量增长迅速

互联网应用，尤其是移动应用增长趋势相当惊人



# 安全风险与日剧增

---

3,930

次已知数据泄漏事件

安全问题给企业造成的影响

736,000,000

□ 法律风险 条记录遭遇泄漏

□ 财务损失

□ 名誉损失

□ 竞争不利

# 应用层的安全性特别值得关注

---

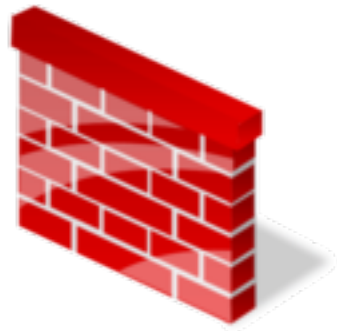


~ 80%的安全漏洞发生在应用层

*World Quality Report 2015/16*

# 应对安全问题的措施

---



WAF



安全监控



渗透测试

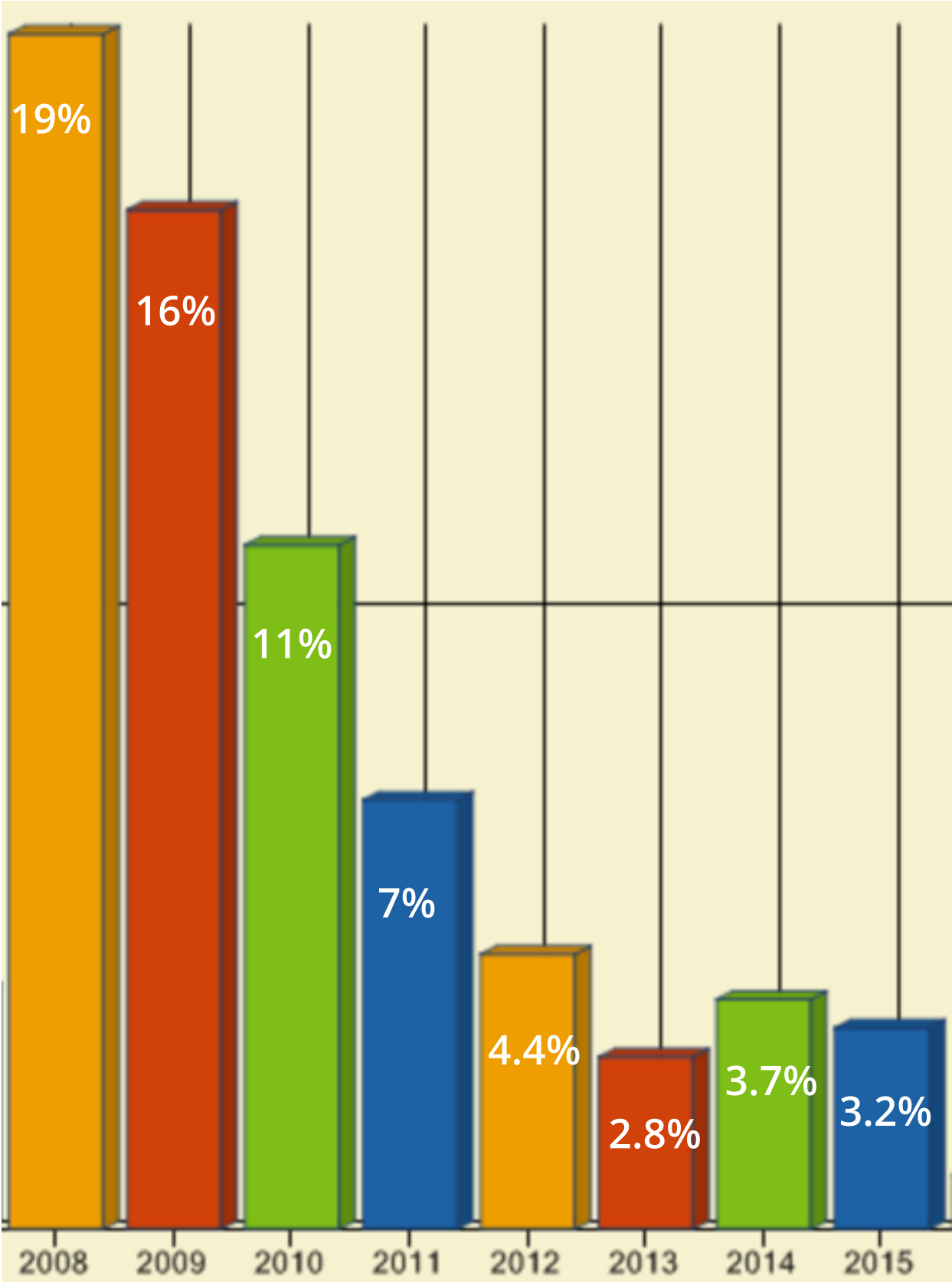


规范文档



安全培训

# 某些安全漏洞数量得到了显著的控制，例如SQL注入

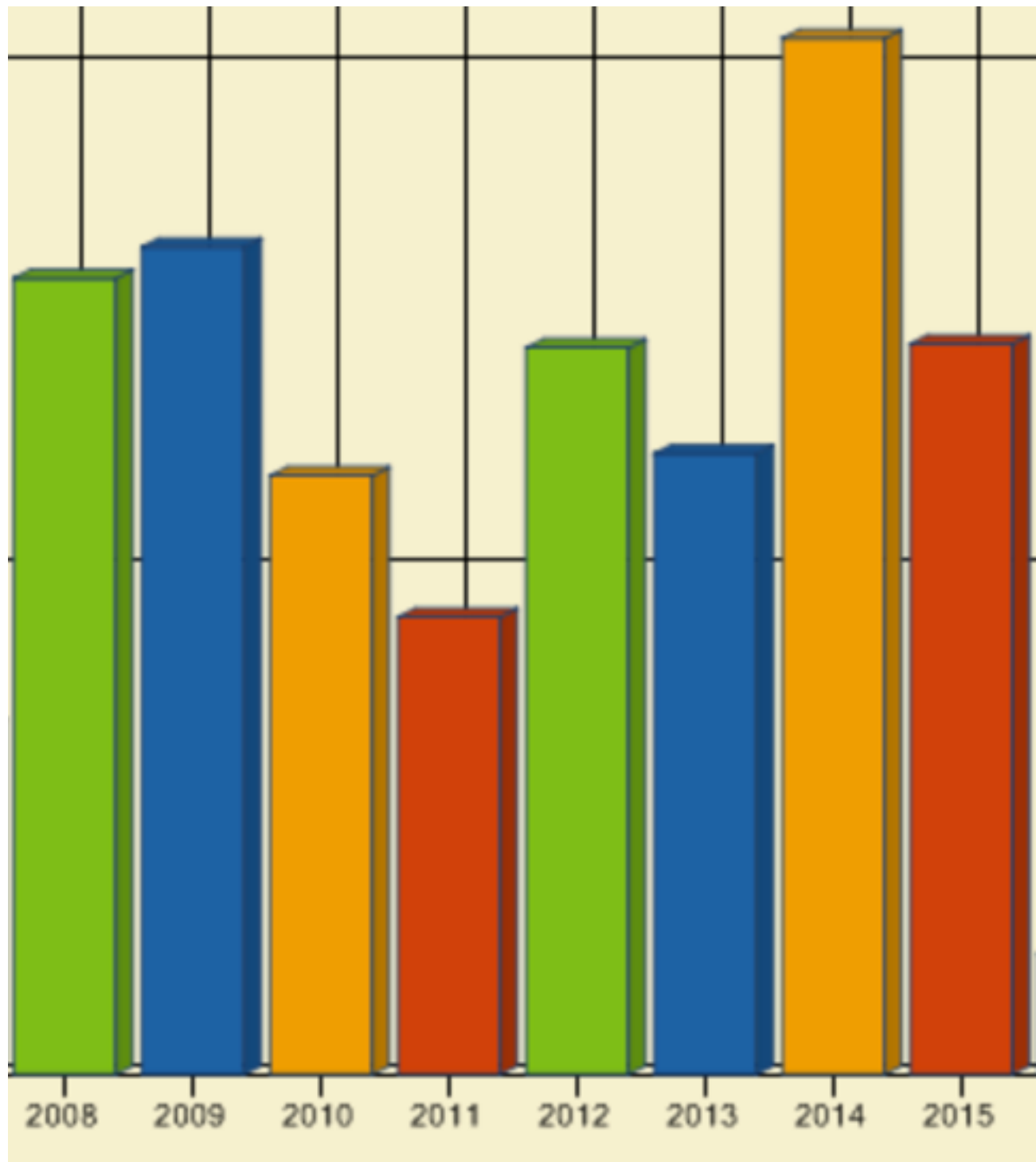


时间	数量	占比
2008	1092	19.39%
2009	948	16.54%
2010	515	11.10%
2011	289	6.96%
2012	236	4.46%
2013	145	2.80%
2014	296	3.73%
2015	212	3.27%

统计数据来自NVD



# 然而有些漏洞却没有多少改观， 例如XSS

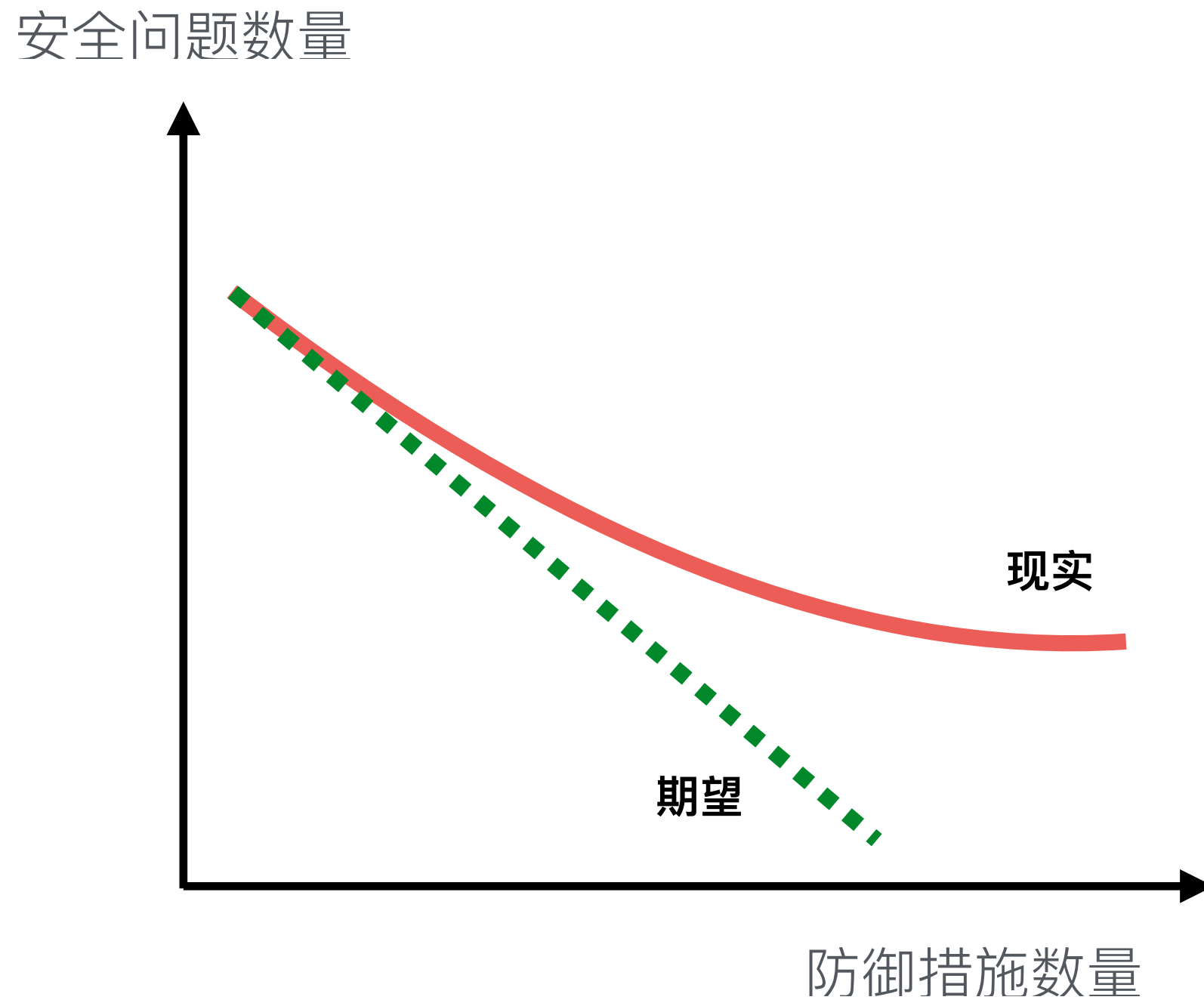


XSS漏洞数量

时间	数量	占比
2008	790	14.03%
2009	821	14.32%
2010	594	12.80%
2011	454	10.94%
2012	721	13.63%
2013	616	11.88%
2014	1028	12.95%
2015	725	11.17%

统计数据来自NVD

# 对于消除安全漏洞，理想和现实有差距



## 更多的困境

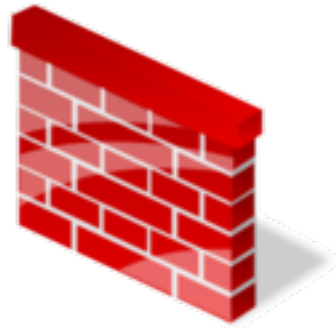
---

- 安全漏洞很晚才被发现，修复成本高昂
- 安全漏洞检查不出来，导致风险增加
- 安全措施容易被误解为是团队的负担
- 人员安全技能缺失

为何安全漏洞如此难以消除？

# 严重依赖于WAF和渗透测试

---

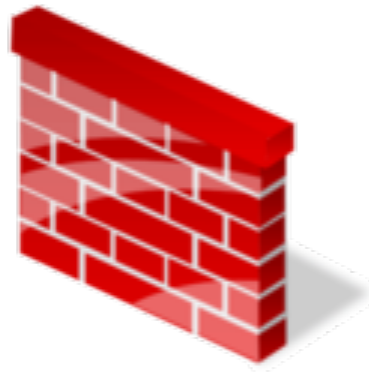


WAF



渗透测试 or  
安全审计

渗透测试 / WAF 虽然有效果，但也存在着不足



WAF

- ❑ 问题代码没被修复，漏洞就一直存在
- ❑ 误报、漏报

# 渗透测试

---



太晚才能得到安全反馈



修复成本高



不可持续

# 难以预测到所有的安全问题



规范文档



安全培训

CWE Version 2.9  
Table of Contents

CWE Version 2.9  
Table of Contents

CWE Version 2.9  
Table of Contents

CWE-237	CWE-35	CWE-480: Use of Incorrect Operator	812
CWE-238	CWE-36	CWE-481: Assigning instead of Comparing	814
CWE-239	CWE-36	CWE-482: Comparing instead of Assigning	817
CWE-240	CWE-36	CWE-483: Incorrect Block Delimitation	818
CWE-241	CWE-36	CWE-484: Omitted Break Statement in Switch	820
CWE-242	CWE-36	CWE-485: Insufficient Encapsulation	822
CWE-243	CWE-36	CWE-486: Comparison of Classes by Name	823
CWE-244	CWE-36	CWE-487: Reliance on Package-level Scope	825
CWE-245	CWE-36	CWE-488: Exposure of Data Element to Wrong Session	826
CWE-246	CWE-36	CWE-489: Leftover Debug Code	827
CWE-247	CWE-36	CWE-490: Mobile Code Issues	829
CWE-248	CWE-36	CWE-491: Public disable() Method Without Final ('Object Hijack')	829
CWE-249	CWE-36	CWE-492: Use of Inner Class Containing Sensitive Data	830
CWE-250	CWE-36	CWE-493: Critical Public Variable Without Final Modifier	836
CWE-251	CWE-36	CWE-494: Download of Code Without Integrity Check	838
CWE-252	CWE-36	CWE-495: Private Array-Typed Field Returned From A Public Method	842
CWE-253	CWE-36	CWE-496: Public Data Assigned to Private Array-Typed Field	843
CWE-254	CWE-36	CWE-497: Exposure of System Data to an Unauthorized Control Sphere	844
CWE-255	CWE-36	CWE-498: Concealable Class Containing Sensitive Information	845
CWE-256	CWE-36	CWE-499: Serializable Class Containing Sensitive Data	847
CWE-257	CWE-36	CWE-500: Public Static Field Not Marked Final	848
CWE-258	CWE-36	CWE-501: Trust Boundary Violation	850
CWE-259	CWE-36	CWE-502: Deserialization of Untrusted Data	850
CWE-260	CWE-36	CWE-503: Byte/Object Code	853
CWE-261	CWE-36	CWE-504: Motivation/Intent	853
CWE-262	CWE-36	CWE-505: Intentionally Introduced Weakness	854
CWE-263	CWE-36	CWE-506: Embedded Malicious Code	854
CWE-264	CWE-36	CWE-507: Trojan Horse	856
CWE-265	CWE-36	CWE-508: Non-Replicating Malicious Code	857
CWE-266	CWE-36	CWE-509: Replicating Malicious Code (Virus or Worm)	857
CWE-267	CWE-36	CWE-510: Trapdoor	858
CWE-268	CWE-36	CWE-511: Logic/Time Bomb	859
CWE-269	CWE-36	CWE-512: Spyware	860
CWE-270	CWE-36	CWE-513: Intentionally Introduced Nonmalicious Weakness	861
CWE-271	CWE-36	CWE-514: Covert Channel	861
CWE-272	CWE-36	CWE-515: Covert Storage Channel	862
CWE-273	CWE-36	CWE-516: DEPRECATED (Duplicate): Covert Timing Channel	863
CWE-274	CWE-36	CWE-517: Other Intentional, Nonmalicious Weakness	863
CWE-275	CWE-36	CWE-518: Inadvertently Introduced Weakness	864
CWE-276	CWE-36	CWE-519: .NET Environment Issues	864
CWE-277	CWE-36	CWE-520: .NET Misconfiguration: Use of Impersonation	864
CWE-278	CWE-36	CWE-521: Weak Password Requirements	865
CWE-279	CWE-36	CWE-522: Insufficiently Protected Credentials	866
CWE-280	CWE-36	CWE-523: Unprotected Transport of Credentials	869
CWE-281	CWE-36	CWE-524: Information Exposure Through Caching	870
CWE-282	CWE-36	CWE-525: Information Exposure Through Browser Caching	870
CWE-283	CWE-36	CWE-526: Information Exposure Through Environmental Variables	871
CWE-284	CWE-36	CWE-527: Exposure of CVS Repository to an Unauthorized Control Sphere	872
CWE-285	CWE-36	CWE-528: Exposure of Core Dump File to an Unauthorized Control Sphere	873
CWE-286	CWE-36	CWE-529: Exposure of Access Control List Files to an Unauthorized Control Sphere	873
CWE-287	CWE-36	CWE-530: Exposure of Backup File to an Unauthorized Control Sphere	874
CWE-288	CWE-36	CWE-531: Information Exposure Through Test Code	875
CWE-289	CWE-36	CWE-532: Information Exposure Through Log Files	875
CWE-290	CWE-36	CWE-533: Information Exposure Through Server Log Files	877
CWE-291	CWE-36	CWE-534: Information Exposure Through Debug Log Files	878
CWE-292	CWE-36	CWE-535: Information Exposure Through Shell Error Message	878
CWE-293	CWE-36	CWE-536: Information Exposure Through Servlet Runtime Error Message	879
CWE-294	CWE-36	CWE-537: Information Exposure Through Java Runtime Error Message	879
CWE-295	CWE-36	CWE-538: File and Directory Information Exposure	881
CWE-296	CWE-36	CWE-539: Information Exposure Through Persistent Cookies	882
CWE-297	CWE-36	CWE-540: Information Exposure Through Source Code	883

1000多种类型的安全漏洞



## 现有措施存在的种种不足

---

- 过于依赖WAF等被动防御
- 安全问题的反馈周期过长
- 基于预测的控制方式难以预料到所有的变化

# ThoughtWorks®

## 如何更彻底的解决安全问题?

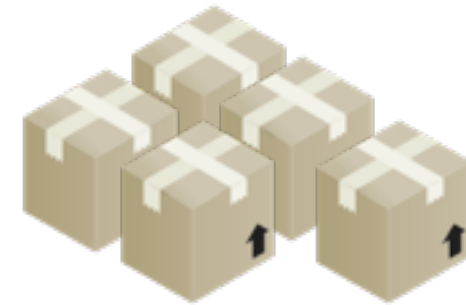
# 极其相似的困境

---



软件测试

- 很晚才测试
- 测试执行速度慢
- 集中式的、一次性测试
- 专人负责测试



系统集成

- 很晚才集成
- 集成过程缓慢
- 一次性集成
- 专人负责集成

# 软件测试是如何应对这些问题的?

---

**尽早测试**

TDD/BDD/DDDD

**自动化测试**

测试工具

**持续构建**

**所有人**

对软件质量负责

- 尽早: 越早发现软件缺陷越有利于修复
- 更快: 加速测试执行速度, 更快的获得软件质量的反馈
- 持续: 软件测试活动贯穿于整个开发周期里
- 共同承担职责: 团队所有成员均对软件质量负责

# 对于解决安全问题，也应如此

---

尽早

更快

持续

共担职责

- **尽早:** 越早发现越早修复
- **更快:** 加快安全反馈获取速度
- **持续:** 在整个开发流程中持续关注安全
- **共同承担职责:** 每位团队成员均对安全负责

# 尽早获取安全反馈



# 通过自动化加速获取安全反馈的速度

---



Automation enables development team  
do more with less

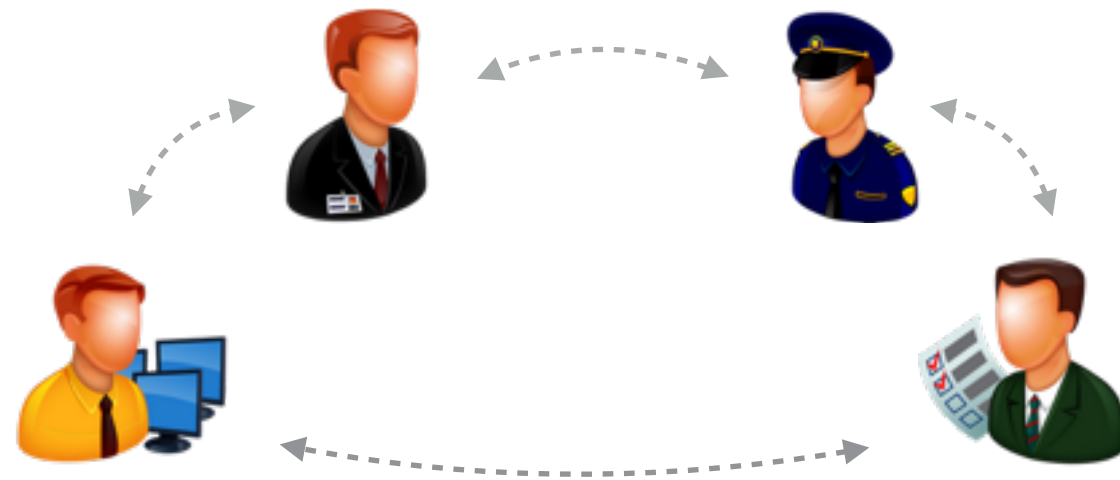
例如:

- ❑ 检测第三方组件是否存在已知安全漏洞
- ❑ 自动化的安全测试工具使得测试人员能更加快速的对产品进行安全测试
- ❑ 安全性的回归测试
- ❑ 易于被集成到构建流水线，以便提供持续性的安全反馈





# 共同承担安全职责



开发团队 & 安全团队



开发团队

安全团队

## Build Security In

**尽早、迅速获取安全反馈** 胜于 等待后期漏洞扫描

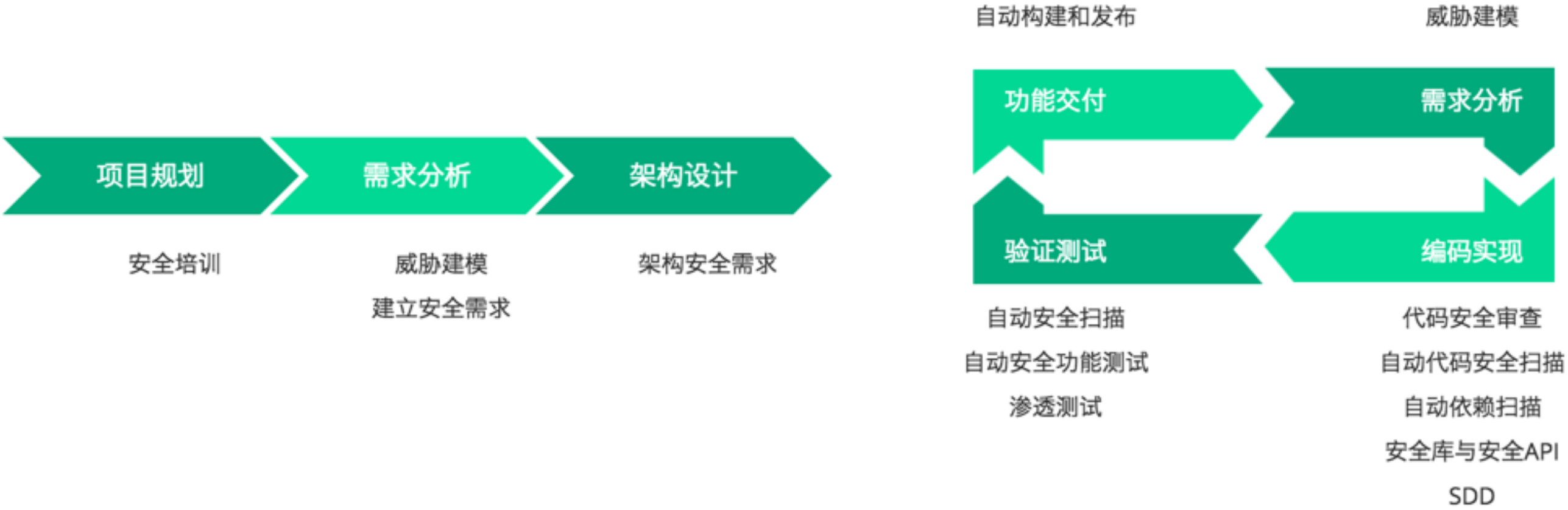
**持续关注安全** 胜于 单次安全审查

**共同承担安全职责** 胜于 过度依赖安全团队

# ThoughtWorks®

## 实践分享

# 在开发过程中引入安全实践





## □ 分析威胁，转换为安全需求

- 思考可能会出现的安全问题
- 寻找应对威胁的解决办法
- 尽早预知潜在问题，尽早应对

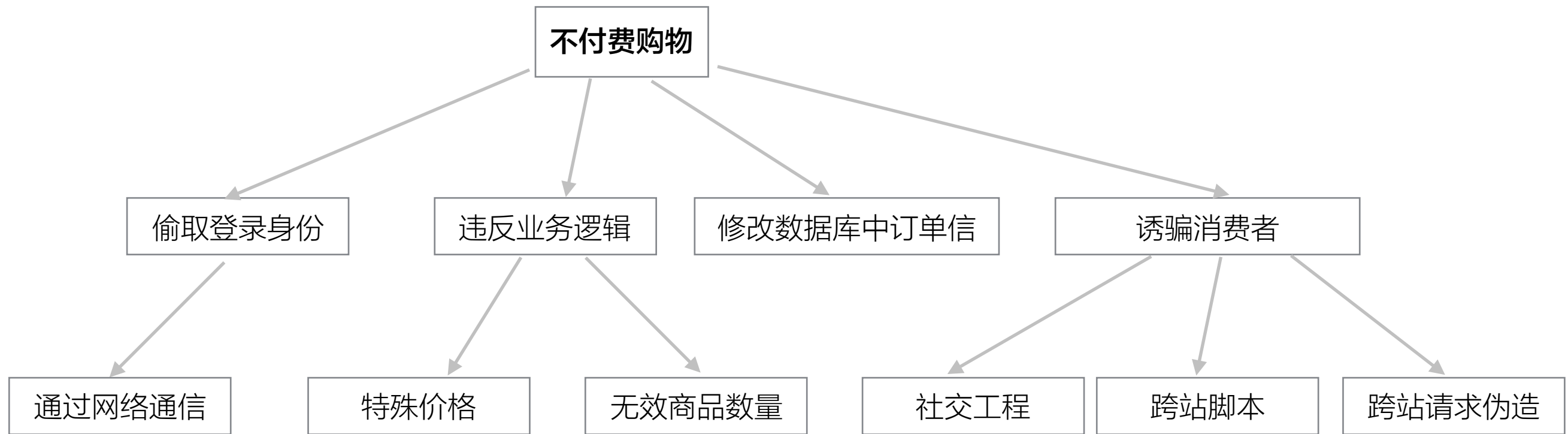
## □ 什么时候做

- 越早越好，推荐在业务需求分析阶段进行

## □ 谁来做

- 业务分析师 / 开发 / 测试

# 安全实践：威胁建模



电商订单支付攻击树模型

## 产出物：威胁清单及应对办法

### □ 数据传输过程存在泄漏风险

- 应对办法：SSL / 敏感数据加密后传输

### □ 产品有多种用户角色，可能出现鉴权漏洞

- 应对办法：严格的权限校验 / 默认只给最小权限 / 异常访问报警 / 专门针对角色和权限的自动化测试 ...

### □ 可能有XSS漏洞

- 应对办法：前端输出编码 / 使用AngularJS的时候避免使用原始数据输出

### □ .....

# 安全实践：和CI集成的自动化安全测试

---

**Given** an anonymous visitor

**When** I try to access report page without authentication

**Then** I was been redirected to login page

**Given** a user without report access permission

**When** I try to access report page with authentication

**Then** I was been redirected to error page

**Given** a system manager

**When** I try to access report page with authentication

**Then** I can access report page successfully



# 安全实践：和CI集成的自动化安全测试

---

```
public void anonymousVisitorCanNotAccessReportPage() {  
    Page currentPage = accessReportPage();  
    assertThat(currentPage, is(LOGIN_PAGE));  
}
```

```
public void userWithoutProperPermissionCanNotAccessReportPage() {  
    loginAsMember();  
    Page currentPage = accessReportPage();  
    assertThat(currentPage, is(PERMISSION_REQUIRED_ERROR_PAGE));  
}
```

```
public void managerCanAccessReportPage() {  
    loginAsManager();  
    Page currentPage = accessReportPage();  
    assertThat(currentPage, is(REPORT_PAGE));  
}
```

# 安全实践：和CI集成的自动化安全测试



Build

Functional

Security

Deploy

## Dependency-Check Report

Dependency-Check is an open source tool performing a best effort analysis of 3rd party dependencies. It is at the user's risk. In no event shall the copyright holder or OWASP be held liable for any damages.

### Project: Maven Quick Start Archetype

Scan Information ([show all](#)):

- dependency-check version: 1.2.9
- Report Generated On: Mar 31, 2015 at 16:51:38 CST
- Dependencies Scanned: 11
- Vulnerable Dependencies: 2
- Vulnerabilities Found: 17
- Vulnerabilities Suppressed: 0
- ...

Display: [Showing Vulnerable Dependencies \(click to show all\)](#)

Dependency	CPE
<a href="#">commons-fileupload-1.2.2.jar</a>	<a href="#">cpe:/a:apache:commons_fileupload:1.2.2</a>
<a href="#">struts2-core-2.3.14.jar</a>	<a href="#">cpe:/a:apache:struts:2.3.14</a>

### Dependencies

commons-fileupload-1.2.2.jar

**Description:** The FileUpload component provides a simple y

**License:**

<http://www.apache.org/licenses/LICENSE-2.0>

**File Path:** /Users/wma/.m2/repository/commons-fileupload/co

**MD5:** A0AD9550A7062DDB6528D8725C8230DD

**SHA1:** 1E48256A2341047E7D729217ADEEC8217F6E3A1A

**Referenced In Project:** Maven Quick Start Archetype

## ZAP Alerts

### 1. General Information

Target website:	http://10.17.6.21:8080
Report generated at:	Fri Apr 03 12:58:24 CST 2015

### 2. Security Alerts Summary

Number of alerts in total: 541

Alerts by severity	Amount
High	2
Medium	5
Low	360
Informational	174

### 3. Security Alerts By Classification

Classification	Amount
Cross Site Scripting (Reflected)	1
SQL Injection	1
Session ID in URL rewrite	4

Use of the tool and the reporting provided constitutes acceptance for use in an AS IS condition, and there is no warranty, expressed or implied, for the resulting report.

Alert Severity	CVE Count	CPE Confidence	Evidence Count
High	2	HIGHEST	16
Medium	15	HIGHEST	13

Availability to servlets and web applications.

# 安全实践：STORY级别的安全测试

---

## □ 主动对应用的安全性进行验证

- 是否满足安全验收标准
- 寻找隐藏在应用中的安全漏洞
- 借助自动化工具的力量



OWASP ZAP



Burp Suite



SQLMap

# 总结

---

## ❑ 面临着严峻的安全挑战

- 过于依赖应用防火墙等被动防御手段
- 太晚才能获取到安全反馈
- 基于预测的控制方式难以预料到所有的变化

## ❑ 解决之道：内建安全于软件开发 (Build Security In)



尽早获取安全反馈



加快安全反馈的速度



持续关注安全



共同承担安全职责

# ThoughtWorks®

# 谢谢

# THANK YOU



# ThoughtWorks®

## Q & A