xss 简单渗透测试

Author: jianxin [80sec]
EMail: jianxin#80sec.com
Site: http://www.80sec.com
Date: 2009-07-25
From: http://www.80sec.com/release/xss-how-to-root.txt

[目录]

0x00 前言
0x01 xss 渗透测试基本思路
0x02 一次黑盒的 xss 渗透测试
0x03 一次白盒的 xss 渗透测试
0x04 总结

0x00 前言

在 web 蓬勃发展的今天，xss 毫无疑问已经变成最“流行”的漏洞，我曾经在安全公司的渗透测试报告里看到列为数十的高危 xss 漏洞，也看到越来越多的安全研究人员将目标投向 xss 攻击，发现 100 个甚至 1000 个之上的 xss。xss 变得如此流行的原因我猜测有几点，首先输入输出是一个应用程序最基本的交互，一个提供服务的应用程序可以不操作数据库，可以不与系统交互，但是肯定会将程序的处理结果返回给浏览器，加上程序员如果意识不到位，就必然发生 xss，对于一个互联网公司，这两方面的因素加起来就会导致这个漏洞数量就非常可观，所以可以经常见到互联网公司如腾讯，新浪，百度，搜狐等等的 xss 漏洞报告。

但是，在谈论 xss 的时候往往大家都会说这是一个 xss 漏洞，而不会提到这是个什么类型的 xss 漏洞，甚至这是个什么场景下的 xss 漏洞，包括有效的攻击方式上都不会去提，即使提到也是一些传说的危害譬如蠕虫，譬如挂马，譬如窃取用户信息，乃至钓鱼攻击。甚至由于种种原因，大家往往关注漏洞两个字甚于前面的 xss。作为安全研究人员，这是一种不负责任的行为，这种行为的后果就是网络安全淡化为 web 安全，web 安全淡化为 xss，而 xss 淡化为 alert()，总有一天程序员会不再信任我们，真正的威胁依然停留在系统里。这方面也有国际上的原因，太多的安全研究人员将重点放在 xss 的研究上，我不知道是否是国外的安全水平已经达到如此需要关注 xss 的水平，尽管 xss 是最普遍的漏洞，但是从黑客的利用程度上来看远远不如一个命令执行，一个文件上传漏洞来得实在。我相信 Google 已经达到这水平，但那只是 Google，不是我们。

我这里不想谈论如何防范 xss 漏洞，如果在你的系统里发现 1000 个之上的 xss，那你就应该停止寻找更多的 xss 而该去想想如何从根源上杜绝 xss，即使杜绝不了 xss 那也应该想想 xss 在这里是不是有什么可预见的风险，如果有的话那有什么方式可以将 xss 的危害弱化乃至一段时期内可以接受的程度。作为一个黑客实用主义者，我这里将描述一次 xss 渗透测试的简单思想以及实现，与

学院派不同，渗透测试不能只是说说而已，必须真实的获取自己想要的东西才可以是成功。

0x01 xss 渗透测试基本思路

在谈论具体的 xss 攻击之前，我们一定要清楚地知道我们的 xss 所处的位置在哪以及我们的 xss 的类型，也就是我上面说的 xss 攻击场景。反射型的 xss 比起持久型的 xss 来效果是不同的，访问量大的 xss 点与访问量小的 xss 点是不同的，发生在 http://www.google.com 和发生在 https://www.google.com 的 xss 点是不同的（你应该知道为什么不同），发生在前台的 xss 点与发生在后台的 xss 点同样不同。分析应用程序我们可以很快确认我们的 xss 点的场景，这在开放的程序里比较好确认，后面我们将讲述如何在一个黑盒的环境下分析出攻击的场景。

在确认好 xss 点的场景之后，我们可以根据我们的目的来决定后续的攻击方向和思路。如果发生的点是一个持久型的并且访问量比较大，你可以依据自己的喜好是否来引起一次混乱；如果你有幸发现发生的点是在管理员的后台或者你可以通过某些方式和管理员的后台进行交互，那么你可以考虑是否需要通过窃取管理员的 cookie 来尝试进入后台，到目前为止，窃取 cookie 依然是最有效的攻击方式，尽管某些 xss 攻击平台可以演示很炫的攻击效果（前提是你的目标会在一个页面停留 2 个小时，这种情况比较少发生）；如果发生的点是一款开源的或者所有数据请求你都可以分析的程序，你可以考虑利用 xss 做一些数据提交，但是前提是依赖于你的 xss 点发生的场景；或者大气点，有浏览器 0day 的直接上浏览器 0day 吧，成本有点高，看收获值不值得了，目前为止貌似这么说的比这么做的多：）

攻击方向和思路确定之后后面的就是一些常规的体力活了，编写攻击代码，获取最终想要的东西，但是这过程可能并不是一帆风顺，甚至需要反复的调试攻击代码以保证最终的效果跟预期的一致。

0x02 一次黑盒的 xss 渗透测试

因为某些原因我们很想测试下国内一个比较有名的评论网站，我们简单测试常规的安全漏洞之后我们没有发现什么有意思的如 SQL 注射之类的安全漏洞，甚至我们还没有办法找到它的后台，但是我们发现了他们的一个 xss 漏洞，比较悲观的是这是一个反射型的 xss 漏洞，所以我们不能直接把他页面黑了（如果是持久类型的 xss，攻击目的就是破坏或者测试的话就可以考虑这么做，当然，如果是为了 YY 也是可以的）。我们不要 YY，我们要 shell。通过对站点的分析我们发现系统有个投稿功能，通过审核之后就可以发表。既然会有审核那么就意味着应该有后台之类的东西，同时我们实际上获得了一个和管理员交互的平台，因为我们写的内容他们肯定会看。于是我们将编写好的 xss exploit url 写到文章里，并且声称在他们的站点内发现了一个黄色的文章。这个 xss exploit url 会请求我的某个 php 文件，这个文件将记录所有请求的 referer，cookie，ip，浏览器类型和当前的 location。等待几十分钟之后，我们顺利获得了这些基础

信息，但是我们发现这里的 location 和 referer 只能获得我们的 xss exploit url，这跟我们希望获得的管理员后台并不一致。分析管理员在后台的操作我们大概可以知道他是点击连接来访问这个 url 的，那么我们是可以获得 opener 窗口的一些信息的，包括 location 等等。在获得 location 之后我们还是很失望，这只是一个内容展现的窗口，并不是后台的真正管理的地址，后台应该是有一个 iframe 类的东西，于是我们再次通过 top.location 获得后台的地址，这次对了，在我们的面前出现了后台登陆的地址，后面的利用窃取的 cookie 进入后台很可行哦：）但是我们在尝试利用 cookie 进入后台时依然出了问题，我们的 cookie 看起来并不有效，这是为什么呢？后来几次测试之后才发现程序做了 cdn，我们访问到的登录地址并不是 cookie 所在的服务器，做了个 host 之后我们顺利登录进后台，后台界面出来的一瞬间让人感觉真是幸福：）后面的就简单，找后台的上传，传 webshell，涂首页：）

很明显，在这样一个场景下如果说到 xss 来做蠕虫肯定是不现实的，对于一个评论网站来说钓鱼也没有什么实际意义，对一个连用户机制都没有或者有用户机制但是用户交互比较低的应用程序来说偷取用户 cookie 同样也没有价值。而真正攻击的过程中也不是简单的说说那么容易，应用程序有很多机会可以防止这种攻击的发生，包括 cookie 和 ip 绑定，cookie 做 httponly，后台设置登录 ip 限制等等（不要跟我说那些看起来很神仙可以反弹的 xss 工具，这就跟物理学里面的理想环境里的实验一样不靠谱）。

在对未知的程序进行测试时，可能某些 xss 点发生在后台等我们未知的地方，而如果我们直接提交敏感的语句如<script>alert()</script>等过去的时候，我们是无法知道程序的返回结果的，而且一旦程序对输入做了处理，在后台出现乱码等字符时，很容易引起别人的警觉。这个时候我们引入类似于渗透测试中经常使用的扫描的手法，在 xss 渗透测试时我们可以利用<img 标签完成扫描工作。输入<img 标签有几个很明显的好处，首先它似乎没有那么有攻击性而且被广泛使用，第二利用它的 src 属性引用到我们自己的地址我们可以判断在哪个地方没有安全处理，可以获得后台的管理地址，可以获得管理员的浏览器类型（这个非常关键，浏览器的不同可以影响到我们的 xss 点的不同乃至利用的不同），第三，一般支持<img 的地方都可以 xss。利用这种探测方式我们攻击过很多站点的后台，一般站点都会有一些审核机制的，不是么？

0x03　一次白盒的 xss 渗透测试

因为某些原因我们想黑掉某个人的 blog，该 blog 系统的源码我们可以从网上获取到，在简单审核一些代码之后我们没有发现明显的 SQL 注射之类的漏洞，但是发现了几个非常有意思的 xss 漏洞，该漏洞同样是反射型的 xss，但是因为程序的原因可以使得 exploit url 变形得非常隐蔽。由于程序开源，我们通过本地搭建该环境可以轻松构造出可以加管理员，可以在后台写 shell 的小型 exploit，并且将 exploit 通过远程的方式隐藏在前面的 exploit url 里。通过分析该程序发现在评论回复时只有登录才可以回复，而目标经常性回复别人的评论，所以我们发表了一个评论并且将 exploit url 写在里面，通过一些手段诱使目标会访问该 url。在等待几个小时之后，我们看到该评论已经被管理员回复，

那么我们的 exploit 也应该是被顺利执行了。上后台用定义好的账户登录，很顺利，shell 也已经存在。OK，最后就是涂首页：）

对于这部分没有什么特别好说的，因为所有的数据和逻辑都是公开的，但是非常重要的一点依然是我们的场景。在某些应用程序里，因为前台的交互比较多，发生 xss 的点是前台，大部分用户的操作也都是前台发生的，但是这部分的权限非常没有意义，我们往往需要特定目标先访问后台，然后从后台访问我们的 xss 点才能获取相应的权限。这部分的攻击就变得比较困难了，而上面的攻击里，由于目标肯定会先访问后台然后访问该 xss 点，所以 xss 变得有趣多了。

0x04　总结

xss 的利用是一件非常有意思的事情，甚至可以独立于 xss 的查找成为一门学问，最关键的一点是所有的 xss 都不要脱离场景，脱离场景地谈论漏洞很不负责任。我给出的例子都是比较简单的，希望可以与大家更多地讨论更多的有意思的攻击。