客户端跨站脚本攻击的分层防御策略①

达斯孟 陆永忠 宁 峰 (华中科技大学 软件工程系 湖北 武汉 430074)

摘 要: 在今天 Web2.0 时代，越来越多的应用正在从桌面朝着网络化的方向发展。网络的内容从最初的静态的一些超链接逐渐的转变为一系列的多样化应用，包括电子商务，电子邮件，游戏娱乐，数字媒体等等都是可以装载到浏览器中的应用。随着浏览器平台的不断发展，带来了很多的安全隐患诸如网络钓鱼，Xss，Xsrf(cross-site request forgery),Dns 等等一系列的黑客手段日益成为威胁互联网用户的安全隐患。黑客可以利用恶意代码，或者钓鱼网站对用户个人信息进行随意的窃取，甚至造成很大的经济损失。针对上述威胁中的 Xss(跨站脚本攻击 cross site scripting)攻击进行分析，并提出一个新的架构来解决日益凸显的安全问题。

关键词: 信息安全；跨站脚本攻击；客户端；分层防御策略；独立线程分配模型

Layered Defense Strategy of the Cross-Site Scripting Attack on Client-Side

DA Si-Meng, LU Yong-Zhong, NING Feng

(School of Software Engineering HUST, Wuhan 430074, China)

Abstract: More and more applications are developing from desktop to networking in present WEB2.0 era. The content of the network gradually shifts from the original static hyperlink to a series of variegated applications including electronic commerce, electronic mailing, game recreation, digital media, which all can be loaded onto the browser. However, with the constant development of browser platform many hidden dangers concerning safety have arisen. For instance, a series of hacker methods such as Xss, xsrf(cross-site request forgery), DNS have become hidden threats to internet users. Hackers could steal users' personal information by utilizing malicious code or through phishing site, which may cause great economic loss. This paper aims to analyse Xss(cross site scripting) attack, and comes up with a new framework to solve this increasingly apparent safety problem.

Keywords: information security; cross-site scripting attacks; client side; layered defense; independent threading model

1 引言

随着越来越多的应用转向网络化，对于用户们的威胁方式也在发生变化。由于编写新的网络应用程序的语言的完善化，使得典型的缓存溢出这样的手段变的不可能[1]。但是新型的攻击手段例如 Xss 攻击这样的手段却日益严峻。并且和传统的威胁不一样的是像 Xss 这样的攻击手段是不能通过防火墙或者杀毒软件来侦测的。并且长久以来各大浏览器也并没有找到解决跨站脚本攻击的有效的防御策略[2]。

因此针对上述的情况，我们提出了一种新型的防御模型—基于独立分配线程和分层防御策略的安全模型。我们的模型与其它模型的主要区别在于它是在建立在客户端(浏览器)的，原因在于当浏览器渲染一个网页的时候如果不把其内容当作脚本编译的话，那么其内容就不会被执行。这样的特性使得浏览器成为了一个防御 Xss 攻击的非常理想的位置。

我们的模型主要由三大部分组成：

(1) 对每一个网页分配独立线程且分析资源消耗的“网页线程分析模块”；

(2) 包含分层防御策略四个规则的用户输入分析

① 收稿时间:2009-04-28

模块;

(3) 保存互联网上有关Xss恶意网站信息的Xss信息数据库。

2 Xss背景介绍

跨站脚本攻击的学术定义为:在远程的Web页面的HTML代码中插入的具有恶意目的的数据，用户认为该页面是可信赖的，但是当用户浏览器下载，编译该页面，嵌入其中的脚本将被解释执行从而造成危害[3]。

现在的网站为了提高用户体验而包含大量的动态内容，程序比过去复杂的多而这些动态内容也给 Xss 攻击提供了可能[4]。Xss 攻击具有危害性大，实施方式多，传播速度快等三大特点，因此使其越来越引起关注。

(1) 危害性：当黑客找到一种合适的方法来成功对网站进行了 Xss 攻击之后，他就可以对用户系统中的很多的私人信息进行窃取，其中包括：帐户劫持，击键记录，用户浏览网页产生的 Cookie， 浏览历史的盗窃。而正是这些个人信息可能就包含了用户的银行密码，或者别的一些敏感信息。Xss 攻击的代码可以通过 HTML、XHTML 等标记语言或者其他客户端脚本语言来写。其中最常用的就是JavaScript或者Jscript。典型的 Xss 攻击的步骤如图 1 所示：

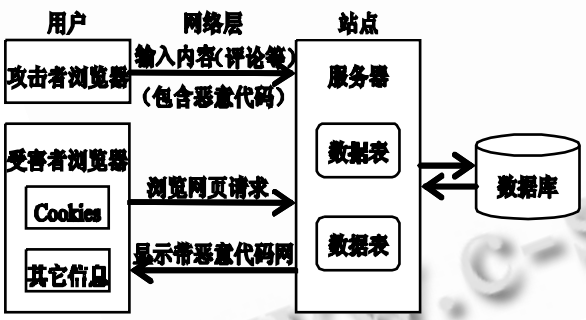

图 1 Xss 攻击的实施过程

(2) 多样性：下面我们列出在网页中嵌入脚本的一些方法[5]，以说明 Xss 攻击的复杂性。并且这些方法全部都可以导致严重的 Xss 攻击。

1) <html><head>

2) <script src="a.js"></script>

3) <script> ... </script>

4) <script for=foo event=onmouseover> ... </script>

5)<style>.bar{background-image:url("JavaScript:alert('JavaScript')");}</style>

6) </head>

7) <body onload="alert('JavaScript')">

8) <img id=foo src="image.jpg">

9)

10) ...

11) <XML ID=I><X><C><![CDATA[<IMG SRC="javas]]><![CDATA[cript:alert('XSS');">]]>

12) <meta http-equiv="refresh" content="0;url=data:text/html;base64,PHNjcmlwdD5hbGVydCgnWFNTJyk8L3NjcmlwdD4K">

13) <img src=j a v a script:alert('XSS')>

14) <img src=Java Script: alert (" 3")>

15) </body></html>

上述的方法充分的说明了 Xss 攻击的多元性和难以预知性。

(3) 传播速度快：使得 Xss 个攻击成为网络安全首要隐患的另外一个原因是，Xss 攻击的传播速度之快简直超出想象。图 2 为第一个走入人们视野的 Xss 攻击 Samy 在爆发后的首个 24 小时里所感染的计算机数目和其它的也曾在“病毒攻击史”上赫赫有名的几个病毒在首个 24 小时里感染计算机的数目的比较[6]。

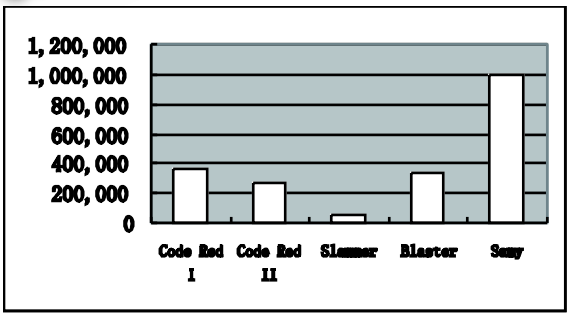

图 2 几个历史上著名的病毒在爆发后首个 24 小时的感染的计算机数量对比

3 基于独立分配线程和分层防御策略的安全模型

Xss 攻击中恶意代码最感兴趣的就是用户的私人

信息，例如用户的网上银行的账号，或者一些存储着敏感个人信息的网站的用户口令。这也是 Xss 攻击的一个特点，他们的目的在于得到一些经济上的“回报”。但是还有另外的一部分的 Xss 攻击针对却是用户的计算机系统本身。例如通过恶意代码的执行使得用户的计算机系统崩溃等等。

在我们的架构中我们不仅对于意图窃取用户敏感信息的 Xss 攻击进行，同时还对破坏计算机系统安全的 Xss 攻击也进行有效防御。

我们提出的一个潜在的解决方案如下：针对上述的以破坏用户的系统安全为目的恶意代码的运行我们提出对于每个进行访问的网页我们的架构要求操作系统分配给它一个独立的线程(这样的做法现在已经出现在 Google 的最新的浏览器 Chrome 中，这样的架构更加适应多核的时代)。这样做的好处为：为每个页面分配线程可以拥有自己的独立的内存空间从而不会产生恶意代码所要做的缓存溢出这样的危害。并且即使恶意代码在一个网页中执行也不会影响到其他的页面的正常运作，因为当网页线程分析模块发现异常的时候比如探查到某个进程所占用的内存或者 CPU 过多那么就会立即向用户发出警告信息，说明具体的哪个网页运行异常，然后我们就可以立即终止其运行，但是却不影响别的页面的运行，这样的就可以最大限度的减少应用程序崩溃而造成的数据的丢失。同时还保护了用户的计算机安全。但是仅仅是这样的防御显然是不够的，上述方案所针对的仅仅是对于危害计算机安全的 Xss 攻击的防御，而对于 Xss 攻击的主要目的，敏感信息的窃取却显得捉襟见肘。因此针对窃取敏感信息的 Xss 攻击我们在总结了别的众多学者的成果和实际的应用效果后提出更有效的防御手段。

我们的分层防御策略是建立在客户端并且带有一个针对 Xss 信息数据库的。我们的策略是：首先将来自服务器端的数据接收到浏览器后不对他们立即进行编译，而是发送到一个分析模块，对其中的网页内容，尤其是脚本内容(标签和属性)进行分析和比对。在此模块中分层防御策略会使用 4 个层次的谨慎筛选对网页的内容的安全性进行评定。如果任何一层的防御措施返回的结果为“true”则说明当前网页是安全的。但是如果模块返回“false”那么就继续进入到下一个规则的判断中，最后依照规则 4 进行最终的判定。

我们将服务器发送的数据中包含的脚本中的标签和属性和字符分为三大类：

(1) 信任列表：所包含的属性，标签和字符是可以安全的编译运行的；

(2) 拒绝列表：所包含的属性，标签和字符是被拒绝编译运行的；

(3) 不确定列表：所包含的属性，标签和字符是不能被简单的确定为允许和不允许。

定义：I{I1,I2,I3,...,In}表明接收网页内容的集合。

T{T1,T2,T3,...,Tn}表明信任列表。

D{D1,D2,D3,...,Dn}拒绝列表。

N{N1,N2,N3,...,Nn}不确定列表。

我们根据对输入的分析制定以下的规则，并且下列的规则按照其顺序执行，一旦某条规则返回“true”则停止分析。说明网页内容是安全的且可以编译运行。

分层防御规则：

(1) 如果当集合 I 是集合 T 的子集的时候，返回“true”。

(2) 如果集合 D 和集合 I 没有交集，且 I 和 N 产生交集的时候，如果用户对某网站发送的请求信息中包含任何集合 T 中没有的字符，那么我们记录这些标签，属性和字符且分析针对上述请求从服务器返回的内容中是否仍然包含相同的上述的这些标签，属性和字符。如果不是则返回“true”。

(3) 当上述的两步均返回“false”的时候，我们就要对网页内容中的所有的外部链接进行抽取和分析。我们将这些链接分别提取出来并发送到 Xss 信息数据库中，由于此数据库中记录了大量的恶意网站和存在风险的网站的信息，因此当接收到信息后便和数据库中的信息做比对。同时我们将一个网页中包含的所有外部链接进行一个特殊的标记，表明他们的相同来源。只有当这些相同来源的所有的外部链接都被表明是安全的时候，也就是从 Xss 信息数据库中返回的结果为“true”我们就认为这个网页是安全的。可是如果当 Xss 信息数据库中并没有关于上述网址的信息时，我们认为数据库应该返回“unknown”状态表明情况。此时我们使用规则二来判断，认定为危险的。至此对于一个网页的分层防御才算结束。

(4) 如果上述的三步均没有返回“true”则说明我们接收的输入是存在安全风险的，此时我们采取如下的两个措施：

① 记录该网站的 IP 地址，并且将这个 IP 地址发

送到上面所提到的 Xss 信息数据库中，且保存作为数据源(数据库的信息来源还包括 Google and StopBadware.org 中记录的信息)；

② 通知用户其所浏览的网页可能存在 Xss 安全漏洞。

4 基于独立分配线程和分层防御策略的安全模型实际应用

下面我们来用实际的 Xss 攻击的例子来演示分层防御策略的可用性。

用户经常在一些知名的且受信任的网站例如 good.com 上进行一些个人隐私数据的操作。这些网站和大多数的网站一样都利用 Cookie 来存储用户的信息来识别用户。由于 SOP(same orgin policy 同源策略)原则的限制，good.com 网站设置的 Cookie 只能由源自 good.com 的内容所接触。因为在 sop 框架下浏览器对于从不同的“源”所来的数据是完全不信任的。而对于从“同一个源”(很多情况下就是利用各种手段来伪装成“同一个源”)所得的数据则是完全的信任，且获得的权限非常的高。这时从“同源”得到的脚本代码一旦通过编译就可为所欲为，得到所有该网页所获得权限[7]。并且伪装成为同一个源也绝不是难事，只需要设定它的 document.domain 属性便可实现最简单的伪装。

例如我们的用户通过社会工程手段被诱使点击了下面的一个链接：(说明:此例中 www.good.com 是表示一个受大众信任的网站，而 www.bad.com 则表示一个恶意网站)

```
<a href= “http://www.good.com/
<script>
Document.location=
 ‘http://bad.com/steal-Cookie.php?’
+document.Cookie
</script>” >
Click here to collect price.
</a>
```

虽然上面的链接看其来非常的奇怪，但是黑客却可以通过各种社会工程手段对它进行完美的伪装，诱使用户点击。当用户被诱使点击该地址后，浏览器将发送一个 HTTP 请求到 www.good.com 来要求访问网页：

```
<script>
Document.location=
 ‘http://bad.com/steal-Cookie.php?’
+document.Cookie
</script>
```

然后当 www.good.com 接受到请求后变开始寻找上述的网页。当 www.good.com 确认没有被要求的这个网页的时候它就会返回“没有可显示的页面”这样消息。这时服务器还会返回到底是哪个文件没有被找到这样的信息给用户的浏览器。当这样的情况发生的时候，此时的文件名就是上述的 Script 代码。它将被返回给浏览器并且被视作是从 www.good.com 的“源”被发回来的。此时这样的 Script 代码就可以获得对 www.good.com 的 Cookie 的完全的控制权。从而用户的隐私就被泄露。

现在使用我们的分层防御策略来对上述的例子进行分析。

当接收到来自服务器的内容集合 I{I1,I2,I3,...,In}后我们对这些内容使用分层防御策略。假设集合 I 并不是集合 T 的子集(因为这样的情况不多见)。规则 1 返回“false”。我们进入规则 2，在规则 2 里我们发现当我们点击的内容和由服务器返回的内容中包含着相同的 <script>Document.location=‘http://bad.com/steal-Cookie.php? ‘+document. Cookie </script> 内容的字符。这时规则会返回“false”。则进入规则 3. 此时我们抽取上述 Script 代码中的外部链接。也就是 http://bad.com。将其发送到 Xss 信息数据库中发现这样的网站是恶意网站。则从规则 3 返回“false”.由此进入规则 4，向用户提出警告，说明存在 Xss 攻击。并且由于没有通过分层防御的分析，Script 代码并没有得到执行。从而起到了保护用户的目的。并且由于分析步骤分为三步，极大的降低了误报的可能性。

5 总结

由于近年来互联网发展的突飞猛进，使得对于像 Xss 攻击这样的威胁的防御的要求变得日益迫切。并且由于 Xss 攻击的普遍性和多元性使得其防御措施一直不尽如人意。本文在对跨站脚本攻击的原理和实施手段进行了详细的剖析和深入的分析后，提出了跨站脚本攻击的分层防御策略，而且它的实施不需要对现今的浏览器进行改动，只需要用插件的形式加入到浏览器中便可以使用。因此具有较高的实用性。

(下转第 204 页)

(上接第 32 页)

参考文献

1 Martin M, Lam MS. Automatic generation of XSS and SQL injection attacks with goal-directed model checking. Proc. of the 17th Conference on Security Symposium Jul. 2008.

2 Kirda E, Kruegel C, Vigna G, Jovanovic P. Noxes: a client-side solution for mitigating cross-site scripting attacks. Proc. of the 2006 ACM Symposium on Applied computing Apr. 2006.

3 古开元,周安民.跨站脚本攻击原理与防范.网络安全技术与应用, 2006,(12):19－21.

4 吴耀斌,王科,龙岳红.基于跨站脚本的网络漏洞攻击与防范.计算机系统应用, 2008,17(1):38－40.

5 Jim T, Swamy P, Hicks PM. Defeating script injection attacks with browser-enforced embedded policies. Proc. of the 16th International Conference on World Wide Web May. 2007.

6 Grossman J. Cross-site scripting worms and viruses the Impending Threat and the Best Defense. APRIL 2006.

7 Karlof C, Shankar U, Tygar JD, Wagner D. Dynamic pharming attacks and the locked same-origin policies for web browsers. Proc. of the 14th ACM Conference on Computer and Communications Security (CCS 2007), November 2007.