Computer Science and Application 计算机科学与应用, 2021, 11(1), 195-206
Published Online January 2021 in Hans. http://www.hanspub.org/journal/csa
https://doi.org/10.12677/csa.2021.111020

Hans汉斯

跨站脚本(XSS)攻击与防护方法研究

王亚东

新疆大学信息科学与工程学院，新疆 乌鲁木齐
Email: yadong2002@163.com

收稿日期：2020年12月27日；录用日期：2021年1月21日；发布日期：2021年1月28日

摘　要

跨站脚本(XSS)攻击多次位列OWASP (开放Web应用安全项目) Top10漏洞列表，是Web客户端面临的最严重的安全危害之一，为有效防御XSS攻击，本文提出了客户端和服务端相结合的XSS攻击防护方法。在客户端，首先判断输入是否为编码后的文本，如果是编码后的文本内容则解码后进行黑名单过滤，如果是未编码的，则直接过滤；在服务端采取强制输出格式的方法对输出到页面的内容进行格式限制，以此防御未知攻击载荷的XSS攻击。此外，通过搭建本地测试环境WAMP (Windows + Apache + Mysql + PHP)，对XSS攻击过程进行模拟研究，并对本文提出的防护方法进行验证。实验的结果表明，本文提出的XSS防护方法能够有效合理地防御XSS攻击。

关键词

跨站脚本，Web漏洞，防护方法

Research on Cross-Site Scripting Attack and Prevention Methods

Yadong Wang

College of Information Science and Engineering, Xinjiang University, Urumqi Xinjiang
Email: yadong2002@163.com

Received: Dec. 27th, 2020; accepted: Jan. 21st, 2021; published: Jan. 28th, 2021

Abstract

Cross-site scripting (XSS) attack has been listed as one of the Top10 vulnerabilities in OWASP (open Web application security project) many times, and is one of the most serious security hazards faced by Web clients. In order to effectively defend against XSS attacks, this paper proposes

文章引用: 王亚东. 跨站脚本(XSS)攻击与防护方法研究[J]. 计算机科学与应用, 2021, 11(1): 195-206.
DOI: 10.12677/csa.2021.111020

an XSS attack protection method combining client and server. On the client side, the input is judged to be encoded text at first. If it is encoded text, the blacklist is filtered after decoding; if it is unencoded, it is directly filtered. At the server side, the output format is forced to restrict the content output to the page, so as to prevent the XSS attack of unknown attack payloads. In addition, the local test environment WAMP (Windows + Apache + Mysql + PHP) was set up to simulate the XSS attack process, and the protection method proposed in this paper was verified. The experimental results show that the XSS protection method proposed in this paper can effectively and reasonably defend against XSS attacks.

Keywords

Cross-Site Scripting, Web Application Vulnerability, Prevention Methods

1. 引言

Web 技术的快速发展，几乎使得人们生活领域的方方面面都应用到了 Web 应用程序，Web 应用程序使用了大量的脚本语言，发展出了社交网络、在线购物和新闻发布浏览等复合型功能的 Web 程序。但随着 Web 程序功能复杂性的提高，也出现了很多问题，比如在很多应用程序的应用场景中，会要求用户提供个人的身份认证等隐私数据，这也给恶意攻击者提供了可乘之机，恶意攻击者会借助各种非法手段盗取用户的隐私信息，进而实施其他更严重的违法行为，给用户的利益造成伤害。

跨站脚本攻击(Cross-site scripting, XSS)是 Web 应用安全问题的主要威胁之一[1]。造成跨站脚本攻击的主要原因是 Web 应用程序未对用户输入的内容进行有效的过滤，导致恶意脚本代码被用户的浏览器所执行，导致了用户信息的泄露、钓鱼欺骗等一系列恶意事件。近年来，跨站脚本攻击事件屡见不鲜，网络安全公司 WebCohort 的报告显示，超过 80%的应用程序存在 XSS 漏洞[2]，Twitter、微博和百度贴吧多次发生跨站脚本蠕虫攻击[3]。OWASP (Open Web Application Security Project)作为 Web 应用安全领域的权威，其每四年发布的 OWASP Top10 漏洞列表中，跨站脚本攻击始终位居前列。针对频发的跨站脚本攻击事件，国内外研究学者提出了多种解决方案。Shanly C 等提出了一种 Skip List 算法，来进行 XSS 漏洞的检测，借助 XSS 攻击向量表，在识别准确率和算法的性能上行有了较大的提高[4]；Shashank Gupta 提出了 XSS 防御框架，该框架是基于内容敏感的，目的是检测出隐藏在 HTML5 Web 应用程序中的 XSS 恶意代码，缺点是该框架必须部署在云平台，对服务器的性能要求较高，对中小企业并不友好[5]。

本文主要研究了 XSS 攻击的原理和存在的根本性原因，并对 XSS 的防护方法进行了深入研究，通过对 XSS 攻击相关的 Web 安全策略的研究和对这些策略存在缺陷的探讨阐述，进一步搭建“留言板系统”仿真实验环境，总结 XSS 攻击过程和原理，并在最后提出了针对 XSS 攻击的防护方法，将本文的方法部署在仿真环境中，通过攻击测试实验，证明了 XSS 防护方法的有效性。

2. XSS 漏洞存在的原因分析

2.1. 同源策略及缺陷

同源策略(SOP)亦称为同域策略，指的是互相访问的站点之间具备相同的协议、域名和端口[6]。即限定动态的内容(比如：JavaScript、ActionScript)仅可以读取或修改和自身同源的 HTTP 请求及 Cookie，不

同源的请求或者内容不被允许进行读取和修改等操作，但是不同源的网站之间可以提出请求。同源策略没有明确规范的协议，保证了浏览器的正常跨域通信功能，市面上的浏览器基本上都支持同源策略。

假定有域 xj.com，这个域包含多个子域，以 m.xj.com 和 n.xj.com 为例，进一步设定该域名下的所有页面均具有 JavaScript 脚本，表 1 描述的是不同页面之间能否互相访问，即发送通信请求并读取响应来判断是否遵循同源策略。

Table 1. Same-origin policy instance
表 1. 同源策略实例

站点 1	站点 2	是否同源	原因
http://www.xj.com/a.html	http://www.xj.com/b.html	是	协议、域名、端口均相同
http://www.xj.com/a.html	http://www.xj.com:81/a.html	否	http 协议默认端口 80，站点 2 端口为 81
http://www.xj.com/a.html	http://m.xj.com/a.html	否	域与子域之间认定是不同域
http://m.xj.com/a.html	http://n.xj.com/a.html	否	不同子域之间被看作不同域
http://www.xj.com/a.html	http://www.xj.com/a/c.html	是	页面所在目录不同，但端口、协议、域名均相同
http://www.xj.com/a.html	https://www.xj.com/a.html	否	遵从的协议不相同
http://www.xj.com/a.html	http: //xj.com/a.html	否	主机名不同。前者为 www.xj.com，后者为 xj.com

HTML 中所使用的<script>、<link>、<iframe>等标签可以跨域加载需要的资源而不被同源策略限制，此外，XMLHttpRequest 通过目标域返回 HTTP 授权时也是可以跨域访问的，因为 JavaScript 不能控制 HTTP 的头部。从表 1 中可以看出，同属于一个域的子域之间也会被同源策略所限制，使得在开发 Web 应用时产生很多不便。为了解决这个问题，同源策略允许同一个域的不同子域的页面之间，通过对 document.domain 变量进行设置，有限的违反同源策略，使得子域的页面之间可以相互访问。这种有限度的违反同源策略也给 Web 应用带来了安全隐患。假设域中的某个页面存在 XSS 漏洞，那么恶意攻击者就可以向这个页面注入 JavaScript 代码，因为 document.domain 变量的存在，则意味着同时可以向该域中的任意页面注入 JavaScript 代码。

2.2. Cookie 安全策略

RFC2109 规定：Web 服务器被允许把用户信息及 HTTP 会话状态等数据写入客户端，并且以文件的形式存储在用户的计算机中[7]。当用户发起 HTTP 会话时，HTTP 会话能使用 HTTP 头部把用户的 Cookie 发送到指定的 Web 服务器。近乎所有的 Web 应用均是利用 Cookie 来标记用户和保持会话状态。根据 Cookie 生存周期的不同，一般将其分为对话 Cookie (亦称临时 Cookie)和本地 Cookie (亦称第三方 Cookie)。会话 Cookie，在用户浏览网页时，浏览器与服务器建立了 Session，Cookie 存放在浏览器的进程内存中，一旦浏览器关闭，则 Cookie 失效，只要浏览器不关闭，用户打开多少个别的标签页，Cookie 依然是有效的；本地 Cookie 的值存放在客户端浏览器之中，本地 Cookie 在被创建时一同设置了生存周期，超过生存周期的 Cookie 便失效。

Cookie 包含 Domain、Path、Secure、Expires 和 HttpOnly 等安全属性。

Secure 属性是非必选的，该属性被设定时，只有用户发起 HTTPS 会话时，才会发送 Cookie。该属性允许 HTTP 和 HTTPS 响应对设置进行修改，所以 HTTP 应答也能改变 HTTPS 应答所设置的 Cookie 的 Secure 属性，进一步导致 HTTP 请求能够获取 Cookie 值，容易造成 CSRF 攻击和 XSS 攻击；Expires 属性能够限制 Cookie 的有效时间，一般情况下，未设置该属性，则浏览器关闭时，Cookie 就被删除，设置了该属性，则 Cookie 在设定的时间之前都是有效的；HttpOnly 属性的设置目的为保证用户 Cookie 的安

全性，一旦该属性被设置，则浏览器无法通过 JavaScript 的 document.cookie 进行访问。

因为浏览器的安全策略不尽相同，当浏览器中某个域的页面想要加载其他域中的资源，有的浏览器会禁止发送第三方 Cookie。通过对不同浏览器的调研得到表 2。浏览器支持第三方 Cookie 的发送将会使得 XSS 攻击盗取用户身份信息变得简单，不必保证用户登录的网页还处于活动状态，只需要用户在第三方 Cookie 生存周期内登陆过目标网站即可。

Table 2. Third-party Cookie support in the browser
表 2. 浏览器的第三方 Cookie 支持情况

浏览器名称	是否默认支持第三方 Cookie	是否支持 P3P
IE6-IE9	否	是
火狐	是	否
Chrome	是	部分支持
Safari	否	否
Opera	是	否

3. XSS 攻击原理与攻击模拟分析

3.1. XSS 攻击原理

跨站脚本攻击(Cross Site Scripting)是一种经常出现在 Web 应用程序中的计算机安全漏洞，是由于 Web 应用程序对用户的输入信息过滤不足而产生的[8]。攻击者利用网站漏洞把恶意的脚本代码(一般包括 JavaScript 和 HTML 代码)注入网页，当其他用户浏览被注入脚本代码的网页时，便会自动执行嵌入的恶意代码，可能导致用户的身份信息盗用、钓鱼欺骗和 XSS 蠕虫病毒等。

因为这种漏洞最早展示的时候是跨域执行的，故称为跨站脚本攻击，跨域指的是绕过同源策略[9]。又因为其与层叠样式表(Cascading Style Sheets, CSS)的缩写冲突，为了防止混淆，故将其简写为“XSS”。通常情况下，既可以将跨站脚本理解成一种 Web 安全漏洞，也可以看作是一种攻击手段。

XSS 漏洞存在许多种不同的表现形式，普遍将其分为三类：反射型、存储型和 DOM 型，划分的依据是 XSS 漏洞的触发方式[10]。但因为 DOM 类型的本质依然是反射型 XSS，故也有学者将 XSS 分为反射型和存储型两类。本文默认跨站脚本攻击分为三类。

3.2. XSS 漏洞的分类

3.2.1. 反射型 XSS

也称为非持久型 XSS，是最常见也是使用最广的一种跨站脚本攻击[11]。攻击者通过电子邮件等特定手法，诱使潜在受害者去访问包含恶意代码的 URL，一旦受害者点击了这些专门设计的链接时，恶意脚本代码便会直接在受害者的主机浏览器上执行，如果 Web 应用程序对用户的输入过滤不全，就会很容易触发反射型 XSS。因为反射型 XSS 只有当用户单击时才能触发，而且只执行一次，所以也被称为非持久性 XSS，图 1 为反射型 XSS 攻击流程示意图。

图 1 为反射型 XSS 攻击流程示意图，可以看出反射型 XSS 攻击流程如下：

1) 用户在 Web 应用程序中登录网站，该程序保存用户的 Cookie 信息；
2) 攻击者将经过设计的含有恶意脚本代码的 URL 发送给目标用户，诱使用户点击 URL；
3) 用户访问该 URL 后，URL 中包含的恶意脚本代码便被发送到服务器；

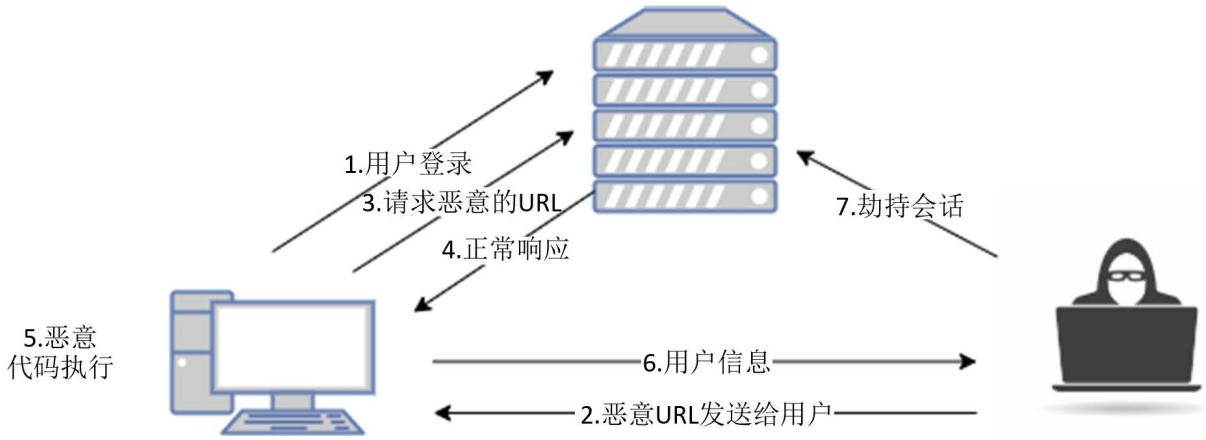

Figure 1. Process of reflect cross-site scripting attack
图 1. 反射型 XSS 攻击过程

4) 服务器解析用户请求，并将恶意脚本嵌入含有漏洞的 HTML 页面中反馈给用户；
5) 用户的 Web 应用程序接收到反馈后，执行页面中包含的脚本代码；
6) 恶意脚本在用户浏览器中执行后，用户的 Cookie 信息等被发送到攻击者的服务器；
7) 攻击者获取用户信息，完成劫持会话等操作。

3.2.2. 存储型 XSS

也称为持久型 XSS。因为存储型 XSS 须要 Web 应用程序向服务器发大于两次的 HTTP 请求，故也称作二阶跨站脚本攻击[12]。其中攻击者把恶意脚本代码上传并存储到服务器上是第一次 HTTP 请求；目标用户链接到包含恶意代码的网页并触发 HTTP 请求是第二次 HTTP 请求。允许用户存储数据的 Web 应用程序都可能会出现存储型 XSS，一般情况下，出现这种漏洞的原因都是网页或者 Web 应用程序未对用户的输入信息进行过滤，导致脚本代码被直接存储到数据库。

图 2 为存储型 XSS 攻击流程示意图。与反射型 XSS 流程图相比可以看出，存储型 XSS 攻击的第一步是将恶意脚本的代码存储到 Web 服务器中，等待目标浏览存储恶意代码的页面即可；而反射型 XSS 则要求将构造好的包含恶意脚本代码的 URL 直接发送给目标，并诱使目标点击访问该 URL。由于存储型 XSS 的这些特点，因此存储型 XSS 更容易造成大规模的安全事件，还有可能对 Web 服务器的安全性产生不利影响。

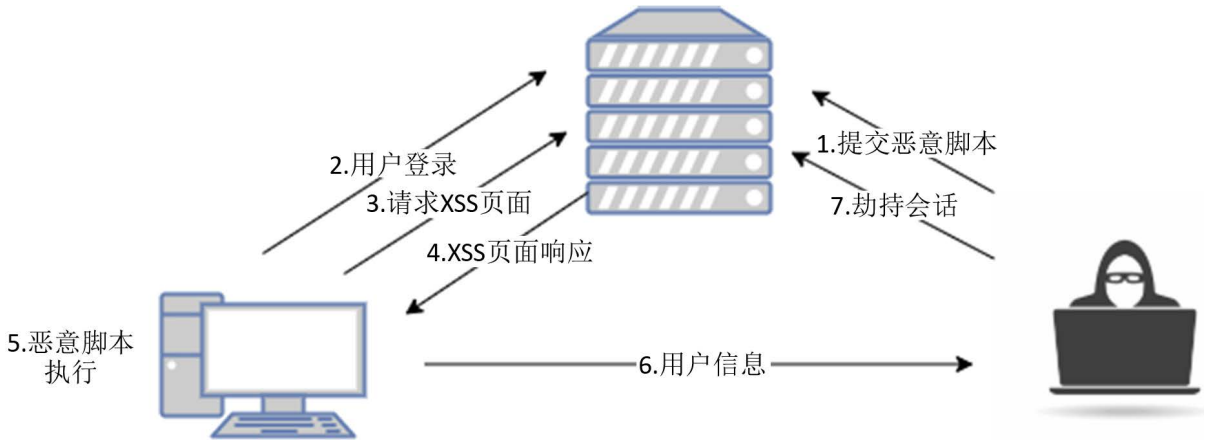

Figure 2. Process of store cross-site scripting attack
图 2. 存储型 XSS 攻击过程

图 2 为存储型 XSS 攻击流程示意图，可以看出存储型 XSS 攻击流程如下：

1) 攻击者将经过包含恶意脚本代码提交至服务器，服务器存储恶意代码；
2) 用户在 Web 应用程序中登录网站，该程序保存用户的 Cookie 信息；
3) 用户访问包含恶意代码的网页；
4) 服务器解析用户请求，并包含恶意脚本代码的页面中反馈给用户；
5) 用户的 Web 应用程序接收到反馈后，执行页面中包含的脚本代码；
6) 恶意脚本在用户浏览器中执行后，用户的 Cookie 信息等被发送到攻击者的服务器；
7) 攻击者获取用户信息，完成劫持会话等操作。

3.2.3. DOM 型 XSS

反射型 XSS 和存储型 XSS 均需要将构造的 URL 发送到服务器，URL 中包含的恶意脚本代码由服务器端执行并嵌入到反馈给受害者的页面中，DOM 型 XSS 不需要将 URL 发送给服务器处理，而是在客户端的 Web 应用程序中执行[13]。DOM 型 XSS 实现的原理是脚本代码能够借用 document.URL 变量访问 URL，可以将脚本嵌入到脚本程序中，进一步使用该脚本调用 document.URL 变量，获取其中的数据信息，通过数据信息动态更新页面。由于页面显示的内容是通过这些数据完成更新，所以有可能造成 DOM 型 XSS 攻击的发生。

反射型 XSS 和存储型 XSS 攻击造成的结果可以在服务器返回的页面中查看到跨站脚本的攻击载荷，但是在 DOM 型 XSS 的攻击中，HTML 源码与遭到攻击后返回的页面源码完全一致，无法在返回的页面中查找到跨站脚本攻击的攻击载荷，只能通过动态运行或者查看网页的 DOM 树时才能观测到。图 3 是 DOM 型 XSS 攻击流程的示意图。

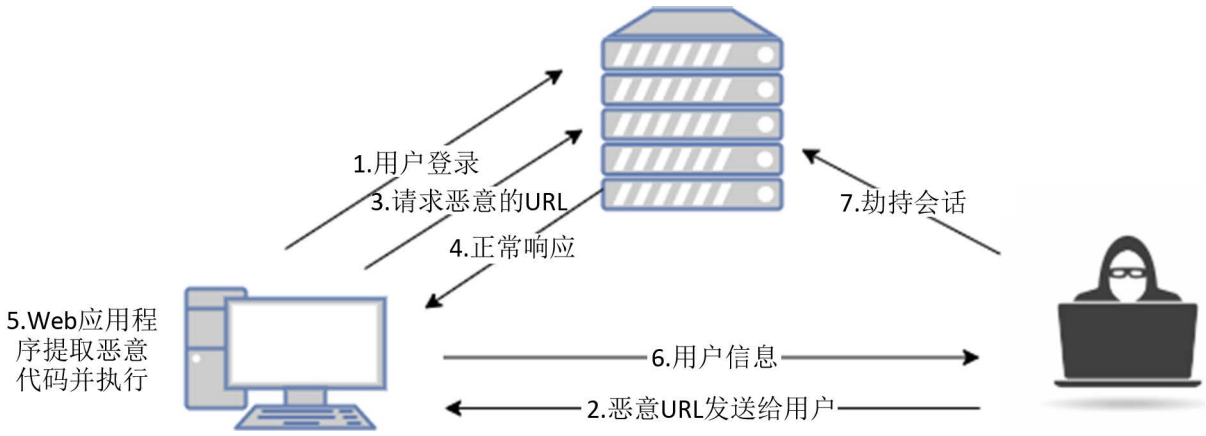

Figure 3. Process of DOM base cross-site scripting attack
图 3. DOM 型 XSS 攻击过程

从图 3 中可以看出，DOM 型 XSS 与反射型 XSS 的攻击过程基本相似，均为攻击者诱使用户访问恶意 URL 才能发动攻击。然而，存在差异的点为 DOM 型 XSS 攻击中，恶意脚本代码并没有经过服务器中转，服务器仅仅是对正常页面的响应，当浏览器处理服务器返回的响应时，恶意脚本代码才被嵌入页面并执行。

3.3. XSS 攻击模拟分析

在仔细研究 XSS 攻击原理的基础上，通过搭建仿真测试环境“留言板系统”为例，对存储型 XSS 攻击进行模拟测试和研究。留言板系统的开发语言为 PHP，使用 Apache 服务器，数据库为 MySQL5.5。

该系统可以实现查看留言、留言和后台的留言管理功能，留言需要审核后才能被查看。系统界面如图 4 所示。

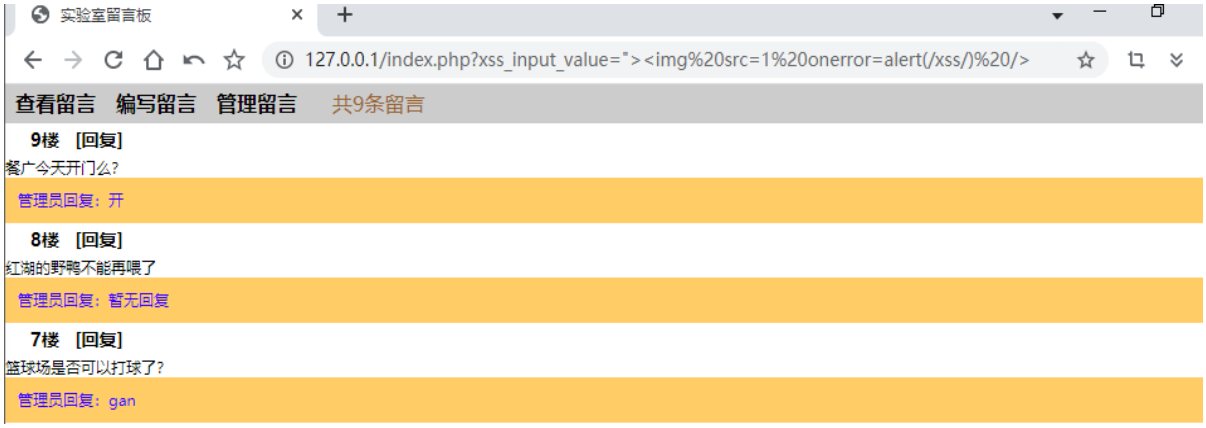

Figure 4. Message board system interface
图 4. 留言板系统界面

在“编写留言”界面输入“<script>alert("存在存储型 xss")</script>”，点击提交留言后，留言内容被提交到数据库。管理员登录后台，对留言内容进行审查，此时便出现了弹窗，如图 5 所示。

Figure 5. XSS popover attack
图 5. XSS 弹窗攻击

出现弹窗的原因是，管理员登录后台进行留言管理时，数据库存放的留言内容(如图 6)被直接发送到管理员，而页面并未对输出到页面端的内容进行过滤。

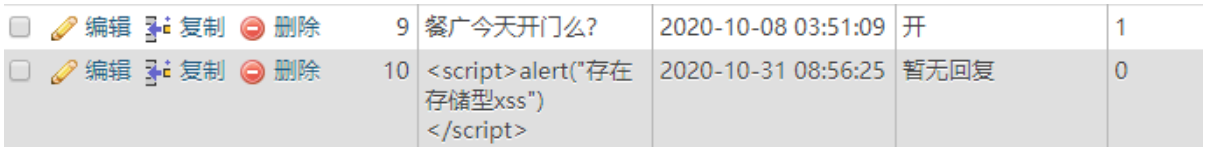

Figure 6. Database message content
图 6. 数据库留言内容

接下来用管理员身份对留言进行审核，审核通过后，再次查看留言板主页面，即“查看留言”界面，此时，页面弹窗提示，如图 7 所示。

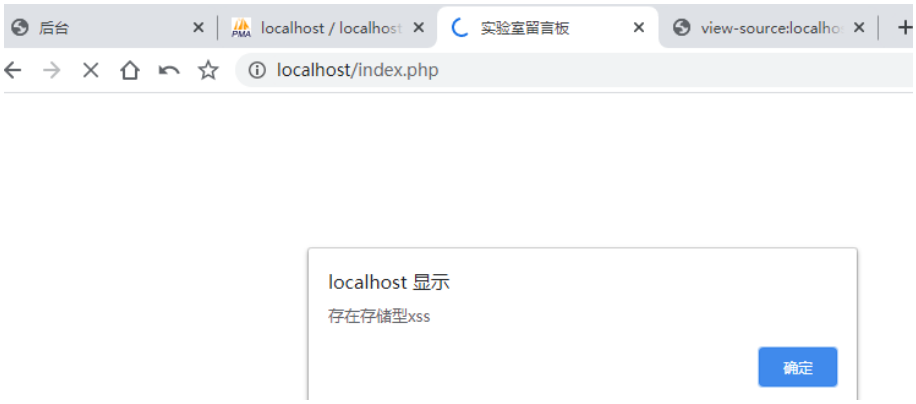

Figure 7. Message board front popup window
图 7. 留言板前端弹窗

对主页的源码进行分析发现(如图 8)，“查看留言”页面并未对数据库返回的内容进行过滤，使得数据库存储的语句被前端的 JavaScript 引擎直接运行 alert 命令，导致弹窗现象的发生，说明了该留言板系统在“查看留言”“编写留言”“留言管理”三个页面中均存在存储型 XSS 漏洞。

```
<div class="contentC"><script>alert("存在存储型xss")</script></div>
<div class="contentA"> <span style="color:#30F">管理员回复：暂无回复</span></div>
```

Figure 8. Front end popover code
图 8. 前端弹窗代码

4. XSS 攻击的防护方法

XSS 漏洞攻击的防御一般有两种方式：输入过滤或输出转义(输出编码)。

输入过滤是指对 Web 用户提交到服务器的数据进行验证，判断其合规性，比如查验输入的内容是否仅含有合法的字符、输入的数字是否在有效范围内以及特殊格式是否符合格式要求等，对于不满足条件的输入内容服务器或者浏览器进行忽略操作，要求用户重新输入符合要求的内容，此外还会过滤<script>、<iframe>等关键字以及“onchange”等 JavaScript 事件标签等。

输出转义是指对会话请求的报文中输入信息的特殊符号执行转义编码操作，导致 XSS payload 丧失原有的攻击能力，此时再将请求报文的信息写入响应报文，反馈给客户端的 Web 应用程序，Web 应用程序接收报文后解析响应不会触发已经失效的 XSS payload，常见编码符号如“<”和“&”的实体编号分别为“&#60”和“&#38”。

4.1. XSS 攻击防护框架

本文提出一种跨站脚本攻击防护方法，框架如图 9 所示。包含“输入管理”和“输出管理”两个核心模块。

输入管理模块：这个模块实现两个功能，一是判定用户输入的内容是否是编码过的；二是根据过滤的规则对输入内容进行过滤。如果用户输入的内容为编码后的，则解码后进行过滤，如果输入内容未编码，则直接过滤。

输出管理模块：根据网页不同版块的内容强制输出内容的格式，防止脚本代码等非信任信息的输出。

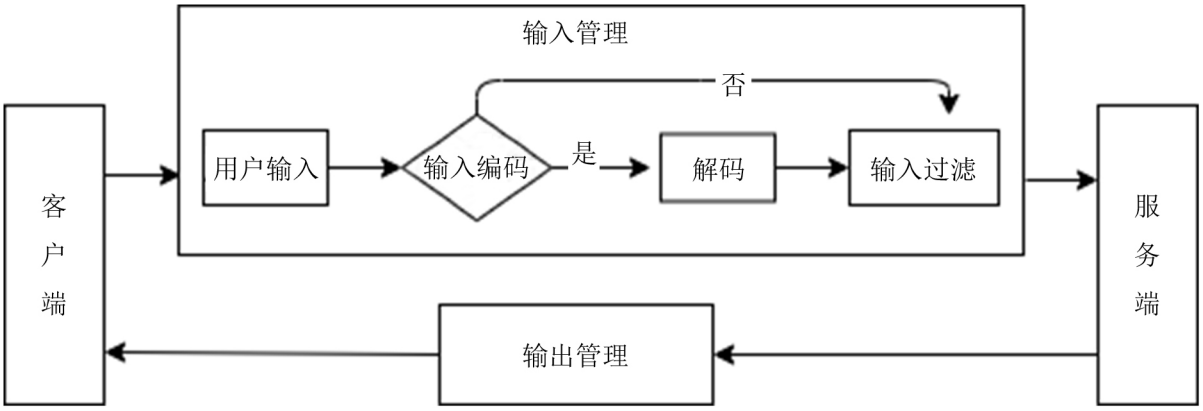

Figure 9. XSS attack protection framework
图 9. XSS 攻击防护框架

4.2. 输入管理

4.2.1. 输入解码

用户正常的输入内容不会引起 XSS 攻击，但是攻击者会利用用户的输入框输入攻击内容进行攻击，由于 Web 支持多种编码方式(如：URL 编码、Hex 编码等)，攻击者可以通过编码等方式来绕过服务端或者管理员的审核，导致 XSS 攻击载荷的成功注入，因此需要对用具的输入进行判断，如果输入的内容经过编码，则需要对编码后的信息进行解码，以此进行下一步的过滤操作。

本文将出现频率较高的 URL 编码、Hex 编码、HTML 编码和 Unicode 编码方式纳入解码的范围，不同的编码会导致字符有不同的结果，以“<iframe>”为例，四种不同编码方式编码的结果如表 3 所示。

Table 3. <iframe> encoding method and result
表 3. <iframe>编码方式与结果

编码对象	编码方式	编码结果
<iframe>	URL 编码	%3ciframe%3e
<iframe>	Hex 编码	%3C%69%66%72%61%6D%65%3E
<iframe>	HTML 编码	<iframe>
<iframe>	Unicode 编码	\u0026\u006c\u0074\u003b\u0069\u0066\u0072\u0061\u006d\u0065\u0026\u0067\u0074\u003b

从编码的结果可以看出特征字符的不同，通过匹配特征字符区分编码的方式，然后分别进行解码，以 Unicode 解码为例，核心代码如图 10 所示。

```
function unicodeDecode($content){
    $json = '{"str":"'.$content.'"}';
    $arr = json_decode($json,true);
    if(empty($arr)) return '';
    return $arr['str'];
```

Figure 10. Unicode decode
图 10. Unicode 解码

4.2.2. 输入过滤

Web 应用程序认为输入的内容是完全正常无害的，所以对用户输入的内容不进行过滤便读取执行，

这就是造成 XSS 攻击泛滥的原因之一。在 XSS 供给端的防护方案中，输入过滤模块是防御 XSS 攻击的主要模块，该模块主要对用户输入的内容进行审核，判断输入的内容是否含有 XSS 攻击。

输入过滤模块首先将输入流中格式进行统一，例如将所有大写字符变为小写等；其次对输入流中的特殊字符进行处理，包括过滤非打印字符，进行回车、换行和连续空格字符的替换等；最后根据黑名单筛选输入流。黑名单所涵盖的对象如表 4 所示，非法字符的过滤和黑名单(部分)的设置代码如图 11 所示。

Table 4. Blacklist object settings
表 4. 黑名单对象设置

对象	内容
HTML 标签内容	script、object、style、iframe 等
JavaScript 事件处理函数	ondrop、onhelp、onerror、onmouseup 等
其他	JavaScript、VBScript、applet、xml 等

```
    $str = preg_replace('/([\x00-\x08,\x0b-\x0c,\x0e-\x19])/', '', $str);
    $str = str_replace(' ','',$str);
    $str = str_replace(array("\r\n", "\r", "\n","\t"), "<br />", $str);
    $ral = array('javascript', 'vbscript', 'expression', 'applet', 'meta', 'xml', 'blink',
'link', 'style', 'script', 'embed', 'object', 'iframe', 'frame', 'frameset', 'ilayer',
'layer', 'bgsound', 'title', 'base');
```

Figure 11. Blacklist filtering
图 11. 黑名单过滤

4.2.3. 输出管理

输入过滤模块仅能针对已知的 XSS 攻击命令来防止 XSS 攻击，对未知的 XSS 攻击防范较差，所以本文对服务端返回客户端的信息也进行了处理，来防范未知的 XSS 攻击形式。

常规的输出端管理采取的是输出编码的方式，即对特定字符进行转义后输出，例如将“&”、“<”、“>”转义为“&”、“<”、“>”后输出，避免了脚本标签的执行。由于网页输出内容的相对固定，比如本文的存储型 XSS 攻击模拟实验的留言板管理系统，输出到网页端的内容完全为文本格式，因此本文通过在输出端强制内容输出的格式，避免输出内容被当做 HTML 解析，来防止 XSS 攻击的发生。以留言板管理系统为例，可以在“content-type”中指定读取文件的类型为“text”或者使用“<xmp>”来规定内容显示形式来规避 XSS 攻击。

4.3. XSS 防护方法实验评估

实验评估使用本文开发的留言板系统进行，测试数据集为 XSS Cheat Sheet 中的 300 个攻击载荷样本和正常输入的 300 个样本，对比对象为 Chrome 上比较流行的第三方 XSS 脚本过滤扩展程序 Html Purify 和 AntiXssUF，测试结果如表 5 所示，图 12 为强制输出格式实验测试效果。

Table 5. Comparison of the effect of three detection filters
表 5. 3 种检测过滤器的效果对比

测试插件	准确率	误报率	漏洞率
Html Purify	90.3%	5.3%	8.3%
AntiXssUF	88.9%	5.1%	7.1%
XSS Protect (本文方法)	92.6%	4.9%	5.5%

Figure 12. Server side protection effect verification
图 12. 服务端防护效果验证

由三款过滤器的实验对比结果可以看出，本文提出的 XSS 防护方案中的前端过滤模块可以较好的过滤已知的攻击载荷，并且在识别准确率、误报率和漏报率的表现均优于 Html Purify 插件及 AntiXssUF 插件；并且在输出管理模块中，原本造成弹窗的“<script>alert (“存在存储型 XSS”)</script>”的攻击载荷并未引起弹窗(图 12 所示)，而是被完整的以文本的形式输出。综上可以证明本文提出的 XSS 防护方案的有效性。

5. 结束语

本文通过解析与 XSS 漏洞相关的 Web 安全策略概念和缺陷，进一步对产生 XSS 攻击的原理和过程进行了研究，通过搭建本地仿真测试环境“留言板系统”，对影响最大的存储型 XSS 攻击进行了实例分析，并且提出了针对 XSS 漏洞攻击的防护方法，辅以攻击脚本的测试实验，证明了本文方法的有效性。由于在客户端过滤模块依然采取的是人工设置黑名单的方法，因此具备一定的滞后性，今后应该结合深度学习提出自动化更新黑名单的方法。

参考文献

[1] 2019CWE/SANS Top 25 Most Dangerous Software Errors https://cwe.mitre.org/top25/archive/2019/2019_cwe_top25.html

[2] Shanmugam, J. and Ponnavaikko, M. (2007) Xss Application Worms: New Internet Infestation and Optimized Protective Measures. *Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing* (*SNPD* 2007), **3**, 1164-1169. https://doi.org/10.1109/SNPD.2007.514

[3] Wit, E. and McClure, J. (2004) Statistics for Microarrays: Design, Analysis, and Inference. 5th Edition, John Wiley & Sons Ltd., Chichester, 5-18.

[4] Chun, S., Jing, C., Changzhen, H., *et al.* (2016) A XSS Attack Detection Method based on Skip List. *International Journal of Security and its Applications*, **10**, 95-106. https://doi.org/10.14257/ijsia.2016.10.5.09

[5] Gupta, S. and Gupta, B.B. (2016) CSSXC: Context-Sensitive Sanitization Framework for Web Applications against XSS Vulnerabilities in Cloud Environments. *Procedia Computer Science*, **85**, 198-205. https://doi.org/10.1016/j.procs.2016.05.211

[6] Schwenk, J., Niemietz, M. and Mainka, C. (2017) Same-Origin Policy: Evaluation in Modern Browsers. 26*th* {*USENIX*} *Security Symposium* {*USENIX*} *Security*, **17**, 713-727.

[7] Barth, A. (2011) Rfc 6265-http State Management Mechanism. *Internet Engineering Task Force* (*IETF*), April 2011, 2070-1721. https://doi.org/10.17487/rfc6265

[8] Di Lucca, G.A., Fasolino, A.R., Mastoianni, M. and Tramontana, P. (2004) Identifying Cross Site Scripting Vulnerabilities in Web Applications. Proceedings. *Sixth IEEE International Workshop on Web Site Evolution*, Chicago, IL, 71-80. https://doi.org/10.1109/WSE.2004.10013

[9] Fogie, S., Grossman, J., Hansen, R., *et al.* (2007) XSS Attacks: Cross Site Scripting Exploits and Defense. Syngress Publishing, Burlington. https://doi.org/10.1016/B978-159749154-9/50005-6

[10] Wassermann, G. and Su, Z. (2008) Static Detection of Cross-Site Scripting Vulnerabilities. 30*th International Conference on Software Engineering* (*ICSE* 2008), Leipzig, 10-18 May 2008, 171-180.

[11] 王岩, 程绍银, 蒋凡. 自动化检测 Android 应用反射型跨站脚本漏洞的方法[J]. 计算机系统应用, 2015, 24(7):

195-199.

[12] 窦永富, 崔为红. 应用程序安全设计探析[J]. 计算机系统应用, 2006, 15(9): 83-86.

[13] Lekies, S., Stock, B. and Johns, M. (2013) 25 Million Flows Later—Large-Scale Detection of DOM-Based XSS. *Proceedings of the* 2013 *ACM SIGSAC Conference on Computer & Communications Security*, Berlin, 4-8 November 2013, 1193-1204. https://doi.org/10.1145/2508859.2516703