# 基于静态分析和动态检测的XSS漏洞发现

潘古兵 周彦晖
(西南大学计算机与信息科学学院 重庆 400715) (重庆信安网络安全等级测评有限公司 重庆 400715)

**摘 要** Web应用程序数量多、应用广泛,然而它们却存在各种能被利用的安全漏洞,这当中跨站脚本(XSS)的比例是最大的。因此为了更好地检测Web应用中的XSS漏洞,提出了一种结合污染传播模型的代码静态分析及净化单元动态检测的方法,其中包括XSS漏洞所对应的源规则、净化规则和接收规则的定义及净化单元动态检测算法的描述。分析表明,该方法能有效地发现Web应用中的XSS漏洞。

**关键词** XSS漏洞,污染传播模型,净化单元,静态分析,动态检测
**中图法分类号** TP393 **文献标识码** A

## Finding XSS Vulnerabilities Based on Static Analysis and Dynamic Testing

PAN Gu-bing ZHOU Yan-hui
(College of Computer and Information Science, Southwest University, Chongqing 400715, China)
(Chongqing Xin An Classified Network Security Testing and Evaluation Co. Ltd., Chongqing 400715, China)

**Abstract** Web applications have a variety of security vulnerabilities which can be exploited when large number of Web applications are widely used. Among these security vulnerabilities, the ratio of cross-site scripting (XSS) is the best. Therefore, in order to detect XSS vulnerabilities in Web applications more effectively, this paper presented a method that combines the static code analysis based on Tainted mode model with the sanitizing unit dynamic testing which includes the definition of the source rules, the sanitizing rules and the receiving rules of XSS vulnerabilities and the description of the dynamic detection algorithm for sanitizing unit. Analysis shows that this method can effectively find XSS vulnerabilities in Web applications.

**Keywords** XSS vulnerability, Tainted mode model, Sanitizing unit, Static analysis, Dynamic testing

随着Web应用程序在实际中的应用变得更加广泛,Web应用安全也受到越来越多的关注。特别地,当Web应用用于处理一些要求高安全性、高保密性领域(比如医药、金融、人力等)的敏感数据时,如果Web应用服务器存在安全漏洞而导致这些数据被泄露,将给社会带来巨大的损失。跨站脚本(XSS)是一类Web应用安全漏洞,利用该漏洞攻击者可以让受害者执行他们定制的JavaScript脚本来达到攻击目的,近年来跨站脚本攻击(XSS)一直名列10大Web安全威胁之首,是所有漏洞中最常见的。因此我们必须要在Web应用发布之前对它们进行全面的测试分析,找到Web应用中的XSS漏洞,消除潜在的威胁。

根据OWASP测试手册[2],发现Web应用漏洞最有效的方法是人工代码审查,但是这种方法非常耗时,并且很多漏洞容易被忽视,最主要的是它需要相当的专业技能。因此安全领域的专家们开发了很多用于自动实现代码审查的工具,包括Fortify SCA、PMD、Findbugs等。这些工具都假设程序的源码是可获得的,大部分都使用了污染传播模型来检测XSS漏洞并且都假设输入净化单元是适用的。然而由于Web应用的复杂性,净化单元通常不能进行有效的过滤,因此它们会漏报或者误报许多漏洞。

针对以上问题,本文提出了一种结合污染传播模型的代码静态分析及净化单元动态检测的方法来检测Web应用程序中的XSS漏洞。通过该方法我们不仅可以发现由于没有使用净化单元而造成的XSS漏洞,也可以发现由于无效的净化单元而引起的XSS漏洞。

## 1 XSS漏洞描述

跨站脚本(XSS)是一种针对Web应用的攻击,它把脚本代码插入Web应用的输出当中,然后这些输出被发送到用户的浏览中被执行,从而被攻击者利用来窃取用户的敏感数据或者让用户执行他不知情的一些操作。

以一个拍卖应用程序为例,它允许买家提出与某件商品有关的问题,然后由卖家回答。如果一名用户提出一个包含嵌入式JavaScript的问题,而Web应用程序在保存该问题时并不过滤或净化这个JavaScript,那么任何查看该问题的用户(包括卖家和潜在的买家)就会在其浏览器中执行该脚本,从而使得不知情的用户去竞标一件他不想要的商品或者让一位卖家接受他提出的低价,结束竞标。

由此,我们知道,当Web应用程序使用未加净化的用户输入构造Web动态页面时会造成跨站脚本(XSS)。

## 2 基于污染传播模型的静态分析

污染传播模型检测方法假设用户输入的数据是污染数据,并使用数据流来判断攻击者能控制的值。因此它要求知

潘古兵(1986-),男,硕士生,主要研究方向为软件工程、信息安全 E-mail:pan7@swu.edu.cn;周彦晖(1972-),男,副教授,硕士生导师。

道数据在程序的什么地方进入，并且怎样在程序中传送，最后到达存在漏洞的操作。污染传播模型检测方法是用来识别许多输入验证缺陷和表示缺陷的关键[4]。这样如果一个 Web 应用包含一个可被利用的 XSS 漏洞，那么它几乎总是会包含一条从接收用户输入的函数到有漏洞操作的路径。

因此，我们可以使用该模型来检测 Web 应用中的跨站脚本(XSS)漏洞。其中根据文献[8]，基于污染传播模型的 Web 应用程序 XSS 漏洞检测方法包括以下几个部分：(1)构建程序的抽象模拟；(2)定义 XSS 漏洞对应的源规则、传递规则、净化规则及接收规则；(3)分析 Web 应用程序 XSS 漏洞。

### 2.1 构建程序的抽象模拟

由于污染传播模型检查方法是一种基于程序数据流的检查方法，因此我们必须在进行漏洞分析前建立程序的数据流模型。根据文献[5]中所述，一个 Web 应用程序可以表示为由 HTTP 请求和 Web 应用当前状态到回复、程序依赖图和新的 Web 应用状态的映射：

$$W:\text{Request}\times\text{State}\rightarrow\text{PDG}\times\text{Response}\times\text{State} \tag{1}$$

Request 是提交到 Web 应用的 HTTP 请求，State(左式)表示 Web 应用的当前状态，它包括当前应用的上下文环境(比如，数据库、文件系统、LDAP 等)，Response 表示 Web 应用的 HTTP 回复，PDG(V,E)程序依赖图表示 Web 应用用来处理给定 HTTP 请求的控制流和数据流，State(右式)表示 Web 应用请求完成后的状态。

### 2.2 定义 XSS 漏洞对应规则

污染传播模型包含一系列的源规则、传递规则、接受规则及净化规则。这些规则说明了程序中源函数是怎样产生污染数据的，以及接受函数是怎样被利用的，如果污染数据传递给它们之前没有经过净化单元净化；并且说明了污染数据是如何通过传递函数来传递的。

源规则$(SS)=\{S_i \mid i\in[1,N]\}$：定义了受污染数据进入到 Web 应用程序的位置。

传递规则$(DS)=\{D_i \mid i\in[1,N]\}$：定义了函数操纵污染数据的方式即污染数据是如何在程序中传递的。

净化规则$(JS)=\{J_i \mid i\in[1,N]\}$：定义了 Web 应用程序中用于净化用户输入以防止潜在的跨站脚本的函数或模块，是传递规则的一种形式。

接收规则$(RS)=\{R_i \mid i\in[1,N]\}$：定义了不该接受污染数据的 Web 应用程序敏感位置。

因此，我们可以利用该模型具体地描述出一个 Java Web 应用跨站脚本(XSS)漏洞规则：

源规则从 HTTP 请求中获取数据：

〈HttpServletRequest. getParameter()〉

接受规则用于把获取数据返回给客户端：

〈ServletOutputStream. print()〉

为了在返回的信息中包含服务器其他信息，需要连接 String 字符串的传递规则：

〈StringBuffer. append()〉,〈StringBuffer. toString()〉

使用开源类库 XSS-HTML-filter 来净化用户输入防止跨站脚本：

〈XSS-HTML-filter. filter()〉

这里只定义了一个针对 Java Web 应用程序 XSS 漏洞的规则，更多的规则在我们的实验中定义。

### 2.3 分析 XSS 漏洞

根据以上描述，我们可以形式化地定义出检测 Web 应用程序 XSS 漏洞的方法：

$F:V\rightarrow\{input, filter, derivation, critical, common\}$ 表示式(1)中程序依赖图(PDG)中每个节点到它们行为的映射，其中这些行为分别为：污染数据输入、污染数据传递、净化污染数据、接受污染数据的敏感操作及程序的其他部分。

因此，对于污染传播模型中的各规则与式(1)中 Web 应用程序的对应关系可以定义为，

$$G:V\rightarrow\rightarrow SS\cup DS\cup JS\cup RS=\begin{cases}S_i\in SS, & \text{if } F(V)=input\\ D_i\in DS, & \text{if } F(V)=derivation\\ J_i\in JS, & \text{if } F(V)=filter\\ R_i\in RS, & \text{if } F(V)=critical\\ \varnothing, & \text{其他}\end{cases} \tag{2}$$

这样使用污染传播模型来静态分析 Web 应用跨站脚本(XSS)漏洞就为，

$$\begin{aligned}&\exists\, path v_1\cdots v_i\cdots v_k\in PDG\\ &F(v_1)=input, F(v_k)=critical\\ &\underset{1\leqslant i<k}{\forall} i: derivation(v_i, v_{i+1})\\ &\underset{1\leqslant j<k}{\forall} j: F(v_j)\neq filter\end{aligned} \tag{3}$$

即若存在一条从污染数据入口($v_1$)到敏感操作($v_k$)的路径，其中任意 $v_{i+1}$ 节点都是通过 $v_i$ 调用 derivation 得到的，并且这个污染数据在使用之前没有被净化($\underset{1\leqslant j<k}{\forall} j: F(v_j)\neq filter$)，那么 Web 应用就存在跨站脚本(XSS)漏洞。

从中可以看到，污染传播模型是通过判断是否使用了净化单元来确定是否存在跨站脚本(XSS)漏洞，而此时如果净化单元本身是错误的，那么检测出来的 Web 应用 XSS 漏洞就失去了可靠性，从而造成了跨站脚本(XSS)漏洞的误报和漏报。因此，我们必须在检测 XSS 漏洞的同时分析净化单元的有效性。

然而，在 Web 应用中可以通过多种方法执行 JavaScript，由此造成编写一个正确的、合适的净化单元是很困难的一项工作。在文献[9]中对净化单元进行静态字符串分析检测，但是静态检测不能反映程序与用户之间的复杂交互情况。事实上，正是由于 Web 应用与用户之间的动态交互的复杂多样性，使 Web 应用产生了大量的漏洞和攻击。因此，为了充分检测净化单元的有效性，从而发现由于不正确的净化单元而导致的跨站脚本(XSS)漏洞，本文提出了针对净化单元的动态检测算法。

## 3 净化单元动态检测算法

缺少对用户输入数据的验证是造成 Web 应用跨站脚本(XSS)的直接原因。因此为了避免跨站脚本攻击，所有的不被信任的数据都必须经过合适的验证才能写入数据库或返回给客户端浏览器。而这些用于验证用户输入的函数或模块我们称为净化单元，而目前，净化单元的比较常用的方法是采用黑名单过滤，一般的做法是采用正则表达式进行合法性检验；另一种常用的做法是仅接受已知的有效的数据(白名单过滤)或者限制输入串的长度；第三种常用的方法是尝试把恶意数据转义成无害的。

因此为了充分检测净化单元是否有效，我们应该尽可能多地构造注入模式的变体来检测净化单元是否都做了正确的处理，从而发现由于净化单元不正确而造成的跨站脚本(XSS)漏洞。具体流程如图1所示。

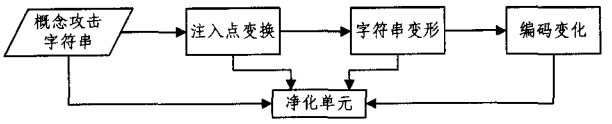


图1 动态测试算法流程

### 3.1 概念攻击字符串

概念攻击字符串即JavasScript脚本(如：〈script〉alert('xss')〈/script〉)通过各种变换输入到净化单元，同时对比相应的输出。

### 3.2 注入点变换

跨站脚本(XSS)使用的JavaScript可能被注入在响应页面中不同的位置，包括文本节点、注释节点、textarea元素节点、script或其它元素节点中。因此针对不同的注入位置，我们可以对概念攻击字符串进行不同的变换([Untrusted input]=概念攻击字符串)，如表1所列。

表1 常见注入点

| 注入点 | 说明 | 例子/模式 |
|---|---|---|
| 文本节点 | 最常见的情况，无须对字符串进行变形 | 〈a href="http://www.*.com"〉Click Here [Untrusted input]〈/a〉 |
| textarea节点 | 为使脚本能被执行而不是被显示在textarea域，我们必须关闭textarea标记 | 〈textarea〉[Untrusted input]〈/textarea〉 |
| JavaScript节点 | 我们必须终止字符串周围的单引号，用一个分号终止整个语句以保证脚本可被执行 | 〈script type="text/javascript"〉…[Untrusted input]…〈/script〉 |
| 属性值 | 为使脚本能被执行，我们必须用引号('、")关闭引号，再使用字符">"来关闭域标记。 | 〈text value=[Untrusted input]〉 |
| 注释节点 | 由于注释块中内容是不能执行的，我们必须在注入的代码前使用前缀"-->"来关闭注释块 | 〈! --[Untrusted input]--〉 |
| 其它元素节点 | 由于HTML语言的复杂性及不同浏览器平台和版本，还有其他可能的注入点 | 根据不同元素节点有不同模式 |

### 3.3 字符串变形

表2 常见字符串变形方法

| 变形方法 | 说明 | 例子/模式 |
|---|---|---|
| 添加空格符 | 许多浏览器接受结束括号前的空白符 | 〈script〉 |
| 改变字符大小写 | 由于一些过滤仅检查常用的小写恶意标签 | 〈ScRipt〉alert('xss')〈/sCriPt〉 |
| 使用多余的尖括号 | 由于浏览器接受多余的尖括号 | 〈〈script〉alert('xss')〈/script〉 |
| 插入URL编码空字节 | 遇到空字节，一些过滤会停止处理字符串 | %00[概念攻击字符串] |
| 插入冗余标签 | 检测净化单元是否进行递归删除 | 〈scr〈script〉ipt〉 |
| 采用脚本文件 | 检测净化单元是否过滤脚本文件 | 〈script src=xss.js〉〈/script〉 |
| 将脚本文件重命名成一个图片文件 | 检测净化单元是否确认图片文件内容 | 〈script src=xss.jpg〉〈/script〉 |

在实际中，Web应用攻击者会对攻击代码做各种变形以试图规避净化单元对用户输入进行的各种过滤和净化，从而使攻击字符串失去可执行能力。其中常见变形方法如表2所列。

### 3.4 编码变换

主要是对概念攻击字符串进行十六进制编码、UTF-8编码或UTF-16编码等来测试净化单元。

### 3.5 净化单元动态检测流程

根据以上描述，为了动态检测净化单元的可靠性，我们应对概念攻击字符串进行各种变化，模仿攻击者从不同的注入点以不同的方式攻击Web应用，从而动态检测出净化单元的有效性。具体流程如下：

1)选取概念攻击字符串，输入净化单元并对比输出；

2)对字符串进行注入点变换，模拟攻击者从不同的注入点进行XSS攻击，然后输入净化单元并对比输出；

3)对字符串进行变形，模拟攻击者对攻击代码做各种变形以试图规避净化单元的过滤，然后输入净化单元对比输出；

4)对字符串进行编码变化，模拟攻击者对攻击代码做编码变化，然后输入净化单元并对比输出。

### 3.6 动态检测结果分析

动态检测出净化单元的有效性之后，如果某净化单元存在错误，那么即使式(3)对应的污染数据入口($v_1$)到敏感操作($v_k$)的路径中，污染数据在使用之前被净化过(即 $\exists_{1\le j<k} j: F(v_j)=filter$)，该Web应用也仍然存在跨站脚本(XSS)漏洞。

## 4 实例分析

为了检验提出的XSS漏洞检测方法的有效性，我们实现了一个Web应用XSS漏洞检测系统——WXTS(Web XSS Test System)，并将其与Fortify SCA进行比较。

Fortify SCA是一个静态的、白盒的软件源代码安全测试工具，是目前全球使用最为广泛的软件源代码安全扫描工具，它提供对Web应用XSS漏洞的静态检测能力。我们使用Fortify SCA Demo4.0.0版本对WebGoat(由OWASP负责维护的一个漏洞百出的J2EE Web应用程序)进行静态分析，并且为了进行比较，只分析其中的跨站脚本(XSS)漏洞。之后，我们采用WXTS对同一Web应用(WebGoat)进行测试分析。

对于XSS漏洞的检测结果如表3所列。经过分析，我们的WXTS能成功地检测出Fortify SCA所检测出的真实的XSS漏洞。此外，我们的WXTS还检测出了更多的由于净化单元错误而造成的XSS漏洞，比Fortify SCA还多查出了5个XSS漏洞。

表3 WebGoat 5.0测试结果对比

| 检测工具 | 跨站脚本(XSS)数量 |
|---|---|
| Fortify SCA | 26 |
| WXTS | 31 |

因此可知，本文提出的方法能显著减少误报和漏报，从而能更有效地发现Web应用中的跨站脚本(XSS)漏洞。

**结束语** 本文通过对Web应用程序跨站脚本(XSS)漏洞进行分析，提出了结合污染传播模型和净化单元动态检测的Web应用跨站脚本(XSS)漏洞检测方法。在该方法中，通过定义XSS漏洞对应源规则、传递规则、净化规则和接收规

(下转第85页)

```
rep), receive_domain) &&Greaterthan(Node i, Node j, double T_
value, trust_domian));
//判断直接认证度是否达到信任阈并且获得的直接认证度与上次
  交易的认证度的差异在一个可以接受的范围之内
{
Feedback(Node I, Node j, SAT);//更新反馈信息
}
else
{
Recommend_value(friends_set, Node_ID, trust_vector);
//向 friends 节点询问被认证节点的认证度
Trust_I_T(Node i, trust_vector, double I_value);//计算间接认证度
Trust_T(Count_D_T(Node i, REP_CA(Node i, double rep), double
d_value), Trust_I_T(Node i, trust_vector, double T_value), double T
_value);
//计算综合认证度
if(Greaterthan(Node i, Node j, double T_value, trust_domian))
//判断认证度是否达到节点的认证阈值
{
Feedback(Node I, Node j, SAT);//更新反馈信息
}}}
```

## 5 可信信任度的演化测试模型

节点信任度测试用于验证基于中介机构的信任模型与传统的信任模型对于可信节点的认证度随认证次数的变化情况。模型随机选取一个可信节点为观察对象，分别记录这两个节点 3000 次交互之后在两种模型影响下的认证度。

实验交互过程如下：

- 在一段时间间隔中随机选择一个节点提出认证请求。
- 新节点的初始认证度为 0.5，表示可信度不确定。
- 认证结果分布在(0,1)之间。

结果图 3 所示。

从实验结果可以看出，基于中介机构第三方信任模型的可信节点其认证度变化较传统的信任模型上升较快，并且认证度也有较大的提高。可信节点经过较少的认证次数即可达到一个较高的平衡值，收敛速度较快。

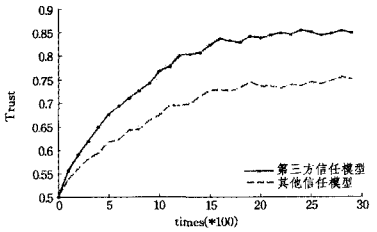


图 3 可信节点的信任度变化趋势

**结束语** 本文对基于节点信任度模型的算法部分进行了设计，根据模型的设计思想分析了影响认证度的几种因素，设计实现了模型的直接认证度算法、间接认证度算法和综合认证度算法。本文提出的认证度评估模型对可信节点的认证度较以往的信任模型有所提高。可信节点认证成功发生交易的可信性增大，有利于网络中资源的有效利用。

## 参 考 文 献


[1] 戚文静，张素，于承新，等. 几种身份认证技术的比较及其发展方向[J]. 山东建筑工程学院学报，2004，6：85-87

[2] 来学嘉. 基于挑战——响应的认证协议安全的必要条件[J]. 中国科学院研究生院学报，2002，19(3)：246-253

[3] 龙毅宏. 可信计算中的数字证书[Z]. Netinfo Security，2004

[4] 鲍宇，曾国荪，曾连荪. P2P 网络中防止欺骗行为的一种信任度计算方法[J]. 通信学报，2008，10：215-222

[5] 田祥宏，严浩，严筱永. P2P 环境下的一种混合式信任模型[J]. 计算机工程与应用，2009，45(31)：73-76

[6] 唐文，陈钟. 基于模糊集合理论的主观信任管理模型研究[J]. 软件学报，2003，14(9)：1401-1408

[7] Zhang Shi-bin, He D. Fuzzy model for trust evaluation[J]. Journal of Southwest jiaotong University, 2006, 14(1): 23-28

[8] Manchala D W. Trust metrics, models and protocols for electronic commerce transactions [C] // Inthe 18th International Conference on Distributed Computing Systems. 1998: 3-12

[9] 徐兰芳，张大圣，徐凤鸣. 基于灰色系统理论的主观信任模型[J]. 小型微型计算机系统，2007，28(5)：801-804


(上接第 53 页)

则来实现 XSS 漏洞的静态分析，并通过定义概念攻击字符串的常见变换来实现净化单元的动态检测，由此来实现 Web 应用跨站脚本(XSS)漏洞检测。分析表明该方法能有效发现 Web 应用程序中的跨站脚本(XSS)漏洞。

但是，本文目前针对净化单元的动态测试主要是手动测试，如何构建变换测试知识库，然后实现自动化动态测试净化单元是下一步的工作重点。

## 参 考 文 献


[1] 石华耀，等. 黑客攻防技术宝典[M]. 北京：人民邮电出版社，2009

[2] Open Web Application Security Project. Testing Guide 2008 V3.0

[3] Open Web Application Security Project. A guide to building secure Web applications

[4] 董启雄，韩平，程永敬，等. 安全编程：代码静态分析[M]. 北京：机械工业出版社，2008

[5] Petukhov A, Kozlov D. Detecting Security Vulnerabilities in Web Applications Using Dynamic Analysis with Penetration Testing [J]. OWASP Application Security Conference, 2008(3)

[6] 朱辉，沈明星，李善平. Web 应用中代码注入漏洞的测试方法[J]. 计算机工程，2010(10)

[7] Livshits B, Lam M S. Finding Security Vulnerabilities in Java Applications with Static Analysis[C]//Proceedings of the 14th conference on USENIX Security Symposium(SSYM'05). Volume 14

[8] Ragle D. Introduction to Perl's Taint Mode[EB/OL]. http://www.webreference.com/programming/perl/taint

[9] Wassermann G, Su Zhen-dong. Static Detection of Cross-Site Scripting Vulnerabilities [C] // Software Engineering, ACM/IEEE 30th International Conference on, ICSE '08. 2008

[10] Open Sourced HTML filtering utility for Java[EB/OL]. http://xss-html-filter.sourceforge.net/