北京大学学报(自然科学版) 第 54 卷 第 2 期 2018 年 3 月
Acta Scientiarum Naturalium Universitatis Pekinensis, Vol. 54, No. 2 (Mar. 2018)
doi: 10.13209/j.0479-8023.2017.172

基于分隔符的跨站脚本攻击防御方法

张慧琳[1,2] 李冠成[1] 丁羽[1] 段镭[1] 韩心慧[1,†] 肖建国[1]

1. 北京大学计算机科学技术研究所, 北京 100871; 2. 国家计算机网络应急技术处理协调中心, 北京 100029;
† 通信作者, E-mail: hanxinhui@pku.edu.cn

摘要 通过分析跨站脚本攻击的特性, 提出一种基于分隔符的跨站脚本攻击防御方法, 该方法适用于 UTF-8 编码的 Web 应用程序。首先, 仅对可信数据中的分隔符进行积极污点标记; 然后, 利用字符 UTF-8 编码值的转换轻量级完成污点标记, 该污点信息可随着字符串操作直接传播到结果页面; 最后, 根据结果页面中分隔符的污点信息及页面上下文分析, 检查脚本执行节点的合法性和脚本内容的可靠性, 精确地检测并防御跨站脚本攻击。针对 PHP 平台实现了原型系统 XSSCleaner。实验证明, XSSCleaner 可轻量级地完成污点分析, 并且能够对跨站脚本攻击进行精确防御, 页面生成的时间开销平均为12.9%。

关键词 跨站脚本攻击; 分隔符; 动态污点分析; 积极污点标记; 影子字节; 页面上下文
中图分类号 TP393

Cross Site Script Prevention Based on Delimiters

ZHANG Huilin[1,2], LI Guancheng[1], DING Yu[1], DUAN Lei[1], HAN Xinhui[1,†], XIAO Jianguo[1]

1. Institute of Computer Science and Technology, Peking University, Beijing 100871; 2. National Computer Network Emergency Response Technical Team/Coordination Center of China, Beijing 100029; † Corresponding author, E-mail: hanxinhui@pku.edu.cn

Abstract The authors propose a practical and accurate cross site script prevention method based on delimiters for UTF-8 encoded web applications. Only trusted delimiters are tainted into corresponding UTF-8 shadow bytes, and these tainted shadow bytes are automatically propagated in web applications and can be directly delivered into output pages. Cross site script is prevented by analyzing the tainted delimiters and HTML context of output pages. A prototype called XSSCleaner is implemented on PHP. The evaluation shows that XSSCleaner can accurately protect web applications from real world cross site script attacks with an average overhead 12.9%.

Key words cross site script; delimiter; dynamic taint analysis; positive taint; shadow bytes; context

随着 Web 的发展, Web 应用程序越来越趋于富交互模式: 用户通过表单、URL 参数等向网站提交数据; 服务器端应用程序动态地生成结果页面(即 HTML 格式的响应页面), 结果页面中包含用户提交内容。典型的应用如博客、论坛、微博等。这种交互模式在方便人们生活的同时, 也带来安全问题。开放式 Web 应用程序安全项目组织(Open Web Application Security Project, OWASP)发布的 Web 安全评估报告显示, 跨站脚本攻击(cross site script, XSS)占据 Web 安全威胁前列[1]。

跨站脚本攻击是由于网站服务器端没有对用户输入和 URL 参数进行合理检查或过滤, 使得结果页面中含有攻击者精心注入的非授权脚本(通常为客户端 JavaScript 脚本代码)[2]。一个典型的场景: 攻击者在论坛网站的评论栏中输入一段 JavaScript 脚本并提交, 服务器端未经充分验证就将该内容放入结果页面中。任意用户访问该结果页面时, 非授权脚本均会被客户端浏览器解析并执行。非授权脚本能窃取页面隐私数据(如网站 cookie、用户输入的口令)、操纵页面组件、劫持会话等[3], 严重危害 Web 安全和用户隐私。

在服务器端的防御方案中, 工业界采用的输入

国家发展和改革委员会2011年信息安全专项《信息系统个人信息保护标准体系建设及关键标准研究验证》资助
收稿日期: 2017–02–08; 修回日期: 2017–03–01; 网络出版日期: 2017–12–05

验证(input validation)或输出净化(output sanitization)的漏报和误报均较多。研究人员基于动态污点分析提出各种检测方案[4–5]，但由于污点分析的性能损失较大，这些方案难以实际应用，且存在一定的漏报或误报。

本文通过分析跨站脚本攻击的特征，提出一种基于分隔符的跨站脚本攻击防御方法，可在服务器端轻量级部署、精确防御跨站脚本攻击。本文的贡献如下。

1) 分析跨站脚本攻击的特性，围绕分隔符进行跨站脚本攻击防御。只对分隔符进行针对性的污点标记和污点传播，根据结果页面中分隔符的污点信息及页面上下文分析，检查脚本执行节点的合法性和脚本内容的可靠性，精确地检测并防御跨站脚本攻击。

2) 利用对字符编码值的转换，进行轻量级的污点标记。污点信息存储在字符的字节值中，不额外占用空间。在字符串拼接等操作中自动完成污点传播，时间开销较小。该方法广泛适用于UTF-8编码的Web应用程序。

3) 基于PHP平台实现原型系统XSSCleaner，能自动检测并防御跨站脚本攻击。

4) 测试了XSSCleaner的防御效果和性能开销，构建数据集来对比多个相关研究的漏报和误报。结果表明，XSSCleaner能准确地防御跨站脚本攻击，污点分析带来的时间开销平均为12.9%。

1　相关工作

按照部署位置不同，将相关工作分为服务器端防御、浏览器端防御、服务器和浏览器双端协同防御。

1.1　浏览器端跨站脚本攻击防御

IE中的XSS filter[6]通过将结果页面与URL中的参数值进行正则表达式匹配来发现跨站脚本攻击。被Chrome采用的XSSauditor[7]将匹配过程放在浏览器解析页面之后，将script和object等敏感标签以及onclick和onload等敏感属性的内容与URL请求中的参数值进行匹配。这种方案只能处理反射型跨站脚本攻击，无法防御存储型跨站脚本攻击。

1.2　双端协同的跨站脚本攻击防御

学术界相继提出Noncespace[8]、DSI[9]和Beep[10]等双端协同防御方案，这些方案的实现较为复杂，且均需修改客户端浏览器，难以部署和推广。

内容安全策略(content security policy, CSP)[11]是业界支持的跨站脚本防御方案，服务器端在页面响应头中指明可信的第三方域，浏览器仅允许加载这些白名单中的资源。这种基于域名白名单的防御方法，仅能防御带有src的外链脚本(external JavaScript)，无法防御本地脚本(inline JavaScript)(除非将本地脚本全部禁止执行，但会大大影响程序功能)。研究表明，网站制定的各种CSP策略常常不能有效地防御跨站脚本攻击[12–14]。研究人员还提出一些方法，自动化地使服务器端支持CSP[15–17]，提出一些基于脚本内容签名匹配的方案[18]，但短时间内难以实际应用。

1.3　服务器端跨站脚本攻击防御

工业界通常在服务器端通过输入验证或输出净化来防御跨站脚本攻击[19]，即分别在接收不可信输入时或将页面片段输出到结果页面时，进行字符串的检查和过滤。这种方式的防御效果有限。一方面，跨站脚本的攻击向量有众多变形及混淆[20]，难以通过简单的字符串检查完全过滤，并且攻击者也可以刻意地对抗字符串检查(如构造ononclickclick来对抗单次过滤onclick关键词)；另一方面，检查的对象仅仅是页面一个片段，缺少数据在结果页面中的上下文信息，容易造成漏报和误报[21–25]。跨站脚本攻击在现实中仍广泛存在[26]。

为了实现更精准的检测和防御，研究人员基于动态污点分析，检测跨站脚本攻击[4–5]。首先，进行污点标记：在Web应用程序执行引擎中，对可信数据或不可信数据标记积极污点信息或消极污点信息；然后，进行污点传播：在字符串操作中将污点信息传播到结果字符串中。最后，进行污点检查：根据一定的安全策略，在输出页面片段时或生成结果页面后进行安全检查。

现有的动态污点分析方案在易用性和检测效果方面存在不足。一方面，在污点标记和传播阶段，现有方法需要修改Web应用程序执行引擎的数据结构或者分配额外内存来存储、跟踪污点信息，难以部署，且性能损失较大；另一方面，在污点检查阶段，现有的安全策略对分隔符及字母、数字等普通字符不加以区分，检测粒度较粗，容易产生误报或漏报。

本文方法属于服务器端防御，设计的污点分析方案为轻量级，易部署，污点标记和传播的性能损

失较小。在污点检查阶段, 围绕分隔符检查脚本执行节点的合法性和脚本内容的可靠性, 能精确地防御跨站脚本攻击。

2 跨站脚本攻击特性分析

通常情况下, Web 应用程序执行引擎解释以及执行服务器端代码(PHP 和 JSP 等)时, 将可信数据(如页面模板和常量字符串)和不可信数据(如用户输入)混杂在一起操作, 通过页面输出操作(如 print 和 echo 等 API)多次输出页面片段, 最终生成一个结果页面返回客户端。

在实际的跨站脚本攻击中, 攻击者需要根据注入点的位置, 精心地构造不同的攻击向量。本文按照注入位置及攻击向量的不同, 将跨站脚本攻击分为 5 种不同类型, 如表 1 所示。表 1 第 1 列展示 5 种不同的注入位置, [uname]代表用户输入。攻击者分析注入位置, 刻意构造第 2 列所示的畸形输入(即跨站脚本攻击向量, 用下划线表示), 该输入未经过 Web 应用程序的充分验证就被放入结果页面中, 导致注入非授权脚本(第 2 列中阴影部分)。客户端浏览器加载该结果页面时, 会执行 alert(123)恶意脚本语句。上述过程对应以下 5 种注入类型。

1) Node splitting。Web 应用程序将用户输入置入 等文本标签中。攻击者用开始标签<script>和闭合标签</script>封装恶意脚本, 提交给服务器端。客户端浏览器在解析该 HTML 片断时, 将其作为脚本标签解析, 随后执行其中的恶意脚本。

2) Attribute splitting。Web 应用程序将用户输入置入属性值中。攻击者在攻击向量中闭合该属性值, 并引入 onclick 事件属性。客户端浏览器将 onclick 作为合法的标签属性解析, 用户的点击操作会触发该事件, 从而执行恶意脚本。类似的事件属性有 onmouseover 和 onmouseout 等。

3) JS statement splitting。Web 应用程序将用户输入置入一段已有的脚本中。攻击者在攻击向量中闭合当前的脚本语句, 引入恶意脚本语句, 脚本的语法结构被破坏。

4) Dynamic HTML update。Web 应用程序将用户输入置入一段已有的脚本中, 并作为 document.write 函数的参数。浏览器为 JavaScript 脚本提供一系列的 API, 可以与页面交互, 增、删、改、查页面元素。document.write 函数将参数字符串动态写入页面中, 浏览器会对参数值进行二次解析、渲染。攻击者在参数值中注入<script>脚本片段(闭合标签<\/script>带反斜线), 浏览器将其当作 document.write 的参数, 脚本引擎执行该函数后, 向页面动态注入恶意脚本(第 1 种类型则是直接影响页面结构, 静态注入<script>脚本标签)。类似的 JavaScript 接口有 innerHTML 和 document.writeln。

5) Dynamic eval。Web 应用程序将用户输入置入一段已有的脚本中, 并且作为 eval 函数的参数。浏览器中的脚本引擎执行 eval 函数时, 将参数作为 JavaScript 代码, 对其进行解析并执行, 攻击者可在该注入点任意注入恶意脚本语句。类似的函数有 setTimeout 和 setInterval。

分析表 1 中的跨站脚本攻击可以发现, 5 类跨站脚本攻击均需利用 HTML 和 JavaScript 中的语法分隔符来注入非授权脚本。

类型 XSS1 和类型 XSS2 利用< >"[空格]/=等分隔符, 在页面上下文中引入非期望的脚本执行节点(如 XSS1 中的 script 标签节点和 XSS2 中的 onclick 事件属性节点), 从而注入恶意脚本。类型 XSS3、XSS4 和 XSS5 则在页面上下文已有的脚本执行环境中注入";< >等分隔符, 进而破坏脚本语法结构(类型 XSS3)或影响脚本动态执行的行为, 导致动态注入脚本节点(类型 XSS4)或动态执行注入的脚本语句(类型 XSS5)。

本文根据上述特性, 围绕分隔符进行跨站脚本

表 1 跨站脚本注入类型

Table 1 Type of cross site script

注入位置	跨站脚本攻击向量及结果页面	注入方式	注入类型
hello [uname]	hello alice<script>alert(123)</script>	Node splitting	XSS1
<input type="text" value="[uname]" />	<input type="text" value="alice" onclick="alert(123)"/>	Attribute splitting	XSS2
<script>name="[uname]";</script>	<script>name="alice";alert(123);""; </script>	JS statement splitting	XSS3
<script>document.write("hello [uname]"); </script>	<script>document.write("hello alice<script>alert(123)<\/script>"); </script>	Dynamic HTML update	XSS4
<script>eval("n='hello [uname]';"); </script>	<script>eval("n='hello alice';alert(123)//';");</script>	Dynamic eval	XSS5

攻击的检测与防御。

3　防御方法设计

基于跨站脚本攻击特性分析，本文的防御方法在服务器端做如下约束：1) 用户输入的分隔符不会引入新的脚本执行节点，保证脚本执行节点的合法性；2) 已有的脚本执行节点中不含有用户输入的分隔符，保证脚本内容的可靠性。

上述思路的关键在于：1) 区分结果页面中分隔符的来源(来自可信数据或不可信数据)；2) 根据结果页面上下文以及分隔符来源，检查脚本执行节点的合法性和脚本内容的可靠性。

Web 应用程序执行引擎将页面模板、常量字符串、数据库数据和用户输入混杂在一起操作，生成结果页面。本文在服务器端进行跨站脚本攻击防御，如图 1 所示。首先，为了区分字符来源，对 Web 应用程序内部的可信数据(页面模板、常量字符串)中的分隔符进行污点标记，利用编码值转换，将污点信息直接存储在字符的字节值中；然后，随着程序的执行，将污点信息传播到结果页面；最后，对带有污点信息的结果页面进行上下文分析，检查脚本执行节点的合法性和脚本内容的可靠性。

3.1　分隔符污点标记

定义1　分隔符(delimeter)是一些能影响页面结构和脚本执行逻辑的字符。OWASP[19−20,27]将这些字符总结如下：&[空格]%*+,-/;<=>^|"`'[tab]。考虑到攻击者构造畸形输入时会使用实体编码、JavaScript 编码等混淆手段来躲避检测，本文补充了脚本混淆变形时常用的关键字符#\(.)[]$:{}等，最后形成分隔符集合 D，共31个字符：

D={&空格%回车*+,换行-/;<=>^|"`'制表#\(.)[]$:{}}。

为了区分分隔符来源，可以对可信数据中的分隔符标记一个积极污点信息或对不可信数据中的分隔符标记一个消极污点信息。由于 Web 应用程序接收的不可信数据来源广泛，易产生漏标记从而影响检测准确性；而程序内部的可信数据相对固定，本文对可信数据中的分隔符进行积极污点标记。污点标记的方式为定义 2 中的编码值转换。

定义 2　字符编码值转换。UTF-8 是Unicode 字符集的一种变长编码方式，UTF-8 单字节字符编码与ASCII编码相同(形如 0xxx xxxx)，集合D中的分隔符在UTF-8 中均为单字节编码。多字节字符的首字节由n位连续的 1 加 1 位 0 开始(n为总字节数)，其后的每个字节开头固定位设置为“10”(例如，3 字节字符的UTF-8 编码形如“1110xxxx 10xxxxxx 10xxxxxx”)。本文利用单字节字符编码空洞 10xx xxxx，为集合D中的 31 个单字节分隔符一一分配影子字节值，通过将原字节值转换为影子字节值，对集合D中的分隔符进行污点标记。影子字节值自带污点信息，并能通过映射关系一一映射回原字符。

定义 2 中的编码值转换不会引起二义性，因为“10”开头的字节不会存在于常规 UTF-8 编码的首字节，因此遇到首字节为“10xxxxxx”形式时，则该字符一定为污点形态的分隔符。该方案适用于 UTF-8 编码的 Web 应用程序。

如图 1 所示，污点标记模块采用编码值转换的方式，对 Web 应用程序中页面模板、常量字符串中的分隔符进行污点标记。污点标记之后，可信数据中的分隔符以影子字节值存在，而不可信数据中

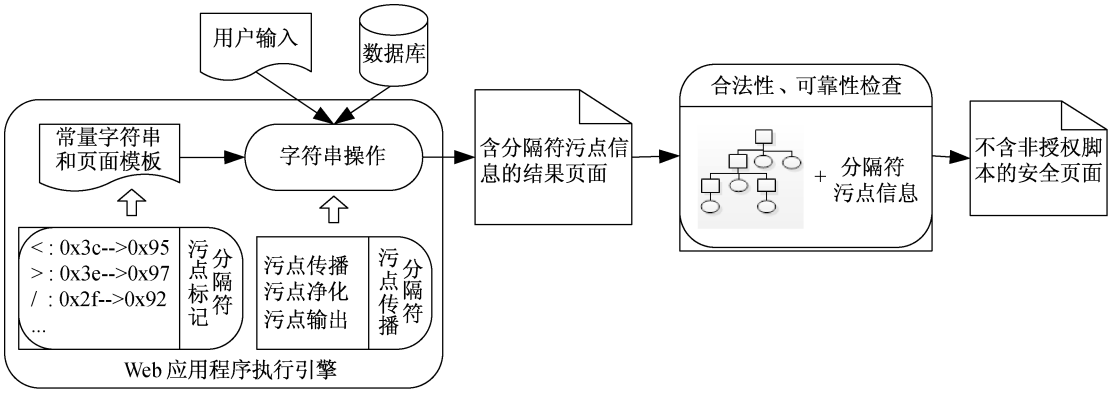

图 1　跨站脚本攻击防御方法设计

Fig. 1　Design of the cross site script prevention method

的分隔符则以标准字节值存在, 因此通过字节值可区分分隔符的来源。

3.2 分隔符污点传播及净化

Web 应用程序使用各种字符串函数将可信数据和不可信数据混杂在一起操作, 因此需要在字符串函数中加入污点传播算法, 在结果字符串中存储和更新相应的污点信息。需要在一些敏感操作处进行污点净化, 保证程序逻辑正常执行。污点信息随着页面片段被输出至页面, 生成带有污点信息的结果页面。

3.2.1 污点传播

一方面, 字节值转换的污点标记方式将污点信息直接存在于影子字节值中, 污点信息会随着字节的移动和复制直接传播到结果中的相应位置, 因此, 类似 join, substr 和 concat 等字符串拼接和分割类函数, 无须加入额外的污点传播方法, 即可将污点信息正确地传播到结果中。

污点标记造成字节值的改变, 使同一个分隔符有两种字节值表示。因此, 在一些涉及字符比较的字符串函数中需要做特殊处理, 如在replace等函数中加入影子字节值和标准字节值的等价比较逻辑, 既能保证函数执行的正确性, 也能保证污点的正确传播。

3.2.2 污点净化

字节值转换的污点标记方式改变了一些参数的字节流, 为了不影响 Web 应用程序正常的业务逻辑, 需要在一些敏感操作执行处(如 include 和 mysql_query)对带有影子字节值的参数进行污点净化(转换回标准字节值), 从而完成正常操作。

在一些数值操作类函数(如md5)中, 也需将参数转换回标准字节值, 从而得到正确结果。

3.2.3 污点输出

在污点标记和污点传播之后, 污点信息以影子字节值的形式直接包含在待输出的页面片段中, 通过输出操作, 可以将其直接输出到结果页面中, 生成一个带有污点字节值的结果页面 OutputPage。页面中影子字节值对应的是可信的分隔符, 不可信的分隔符则以标准字节值的形式存在。

3.3 脚本执行节点合法性和可靠性检查

基于页面上下文分析, 结合结果页面中的分隔符污点信息, 检查脚本执行节点的合法性和脚本内容的可靠性, 即检查不可信分隔符是否引入了新的脚本执行节点, 以及不可信分隔符是否存在于已有的脚本执行节点中。

脚本执行节点 JSNode 指可作为脚本代码执行载体的标签结点和属性节点(包括<script>标签和 onclick 属性等)[22]。脚本执行节点可通过 DOM 上下文解析得到。

脚本执行节点 JSNode 分为代码部分和载体部分, 其中代码部分 JSNode.code 为脚本代码(下面示例中的加框部分), 其余部分为载体部分 JSNode.land。

示例 1 onclick=”alert(123)”。

示例 2 <script>alert(123)</script>。

脚本节点合法性策略 P1: 要求脚本执行节点的载体部分 JSNode.land 不含不可信的分隔符。

脚本内容可靠性策略 P2: 要求脚本执行节点的代码部分 JSNode.code 不含不可信的分隔符。

综合策略 P1和策略 P2, 要求每个脚本执行节点的代码部分和载体部分均不含不可信的分隔符, 否则将其删除。具体见算法 1。

算法 1 脚本执行节点合法性和可靠性检查。

输入含有分隔符污点信息的结果页面 Output-Page
输出删除非授权脚本后的安全页面

```
1  UntrustedIndex:=Ø
2  for i← 0 to |OutputPage|–1
3    if tainted(OutputPage [i]) then
4        clearTaint(OutputPage [i])
5    else
6        UntrustedIndex.add (i)
7  if UntrustedIndex== Ø
8      return OutputPage
9  DomTree←parseDOM(OutputPage)
10 JSExecutors ←DomTree.getJSExecutor( )
11 foreach j in UntrustedIndex:
12   foreach JSNode in JSExecutors:
13       if   JSNode.start⩽j⩽JSNode.end
14            delete JSNode from OutputPage
15 return OutputPage
```

算法 1 的输入为含有分隔符污点信息的结果页面, 其中, 可信的分隔符以影子字节值表示, 不可信的分隔符以 UTF-8 原字节值表示。

步骤 1~6 首先遍历结果页面 OutputPage, 记录不可信分隔符的下标, 并将可信的分隔符恢复为 UTF-8 标准编码值。

若页面不含不可信分隔符, 则在步骤 7~8 返回结果页面。若页面含不可信分隔符, 则步骤 9~10 通过页面上下文分析得到所有的 JSNode 脚本执行节点, 每个 JSNode 对应一个在原文档中的起始下标

JSNode.start 和在原文档中的结束下标 JSNode.end。

步骤 11~14 将不可信分隔符的下角标与脚本执行节点的起止下角标比对，判断脚本执行节点 JSNode 中是否存在不可信分隔符，若存在，则该脚本执行节点不满足可靠性或合法性，将该脚本执行节点删除。

算法 1 在步骤 15 返回不含有非授权脚本的安全页面。

4　实现

本文基于 PHP 解释执行引擎 Zend (版本5.4.21)，实现 XSSCleaner 原型系统。XSSCleaner 包括一个 PHP 扩展和一个代理模块。PHP 扩展完成污点标记及污点传播，代理模块完成安全检查。

4.1　污点标记及传播的实现

在一个典型的 Web 服务器中，PHP 程序被送入 PHP Zend 虚拟机中进行解释，生成中间代码，最后执行中间代码。Zend 引擎提供完整的插件开发 API，使开发者可以在整个 PHP 的解释和执行过程中插装代码。XSSCleaner 通过对 PHP 引擎的一系列中间代码和函数的 hook，实现污点标记和传播。XSSCleaner 将该部分实现为插件，较易部署。同时，PHP 引擎是 C 语言编写，插件也是 C 语言编写，执行效率高。

1) 污点标记。Zend 引擎将页面模板字符串和常量字符串转化为属性为 CONST 的内部字符串结构，并将它们传递给中间码操作。XSSCleaner 在 Zend 引擎生成中间代码后，遍历所有的中间代码，检查每一个中间代码的两个参数。如果参数类型为字符串，并且具有属性 IS_CONST，那么将其中的分隔符的字节值转换为影子字节值，进行污点标记。

2) 污点传播。PHP 中的内置函数在 PHP 引擎中对应着 zend_function 结构体，结构体中指明函数 handler 来对参数进行具体操作。XSSCleaner 对 PHP 内置字符串函数的 handler 进行 hook，按照参数的污点信息以及函数功能，加入污点传播算法。

3) 污点净化。为了防止影子字节值影响程序的正常逻辑和函数功能，XSSCleaner 通过 hook 中间码的处理函数和重写内置函数(如 INCLUDE_OR_EVAL 中间码和 mysql_query 函数、md5函数等)，对参数进行污点净化。

4) 污点输出。PHP 引擎通过 ECHO 和 PRINT 等中间码的执行，向结果页面中输出页面片段，随着中间码的执行，污点字节值自动输出到结果页面中。

4.2　安全检查模块实现

XSSCleaner 将安全检查实现为一个代理：接收 Zend 引擎生成的带有污点信息的结果页面；遍历页面，记录不可信分隔符的下标；用 Beautiful-Soup[28]进行页面上下文分析并提取脚本执行节点，检查脚本执行节点是否含有不可信分隔符；将不安全的脚本执行节点删除后，将安全的页面返回客户端。

5　实验及分析

本文通过实验，对 XSSCleaner 的跨站脚本攻击防御效果以及性能损失进行分析和比较。

5.1　防御测试集中的跨站脚本攻击

选取文献[29]构建的跨站脚本攻击测试集 T1，安装相应版本的应用程序，构造攻击向量，以验证 XSSCleaner 的有效性。如表 2 所示，XSSCleaner 能有效地检测并防御测试集 T1 中的跨站脚本攻击。

5.2　防御真实跨站脚本攻击

为了验证 XSSCleaner 在真实的跨站脚本攻击中的防御效果，我们选取真实的跨站脚本攻击案例，形成测试集 T2，如表 3 所示。一方面，选取最新披露的针对常见 PHP Web 应用程序的 9 个跨站脚本攻击，安装相应版本的开源 PHP 应用程序，构造攻击向量；另一方面，自己编写 PHP 页面，模拟腾讯娱乐和社区动力两个网站中存在的 XSS 漏洞，构造攻击向量，重放攻击。表 3 中的 11 个攻击涵盖跨站脚本攻击的 5 种类型。实验证明，XSSCleaner 均能准确地检测并防御这 5 类跨站脚本攻击。

表 3 中的案例 5 如图 2 所示，利用 ed2k 标准链接中的文件大小域，构造'+alert(123)+'畸形输入，利用'+ 和 +' 破坏脚本结构，恶意增加语句 alert(123)。XSSCleaner 准确地检测出注入在脚本内容中的加号、单引号、括号等不可信分隔符，判断其违背了脚本内容可靠性，从而防御跨站脚本攻击。

表 2　XSSCleaner 对测试集 T1 的防御效果

Table 2　Effectiveness of XSSCleaner on benchmark T1

应用程序	攻击数量	防御数量
Schoolmate1.5.4	12	12
Webchess0.9.0	10	10

表 3 XSSCleaner 对测试集 T2 的防御效果

Table 3 Effectiveness of XSSCleaner on benchmark T2

应用及漏洞页面	攻击向量	攻击类型	XSSCleaner
1. Wordpress<=4.4.1 (post.php)	content="[<audio width="300" height="32" src="wp"onerror="alert(/XSS/)"></audio>"&lots of params	XSS1	防御
2. Wordpress<=3.9.2 (wp-comments-post.php)	comment="[[" <!-- onmouseover=alert(/XSS/)//><!-- -->XSS<a>]"&lots of params	XSS1	防御
3. Wordpress<=4.2 (wp-comments-post.php)	comment=" "&lots of params	XSS1	防御
4. WordPress<=4.1.1 (wp-comments-post.php)	<blockquotecite='x onmouseover=alert(/XSS/)后面接一个 utf8mb4 字符"&lots of paras	XSS1	防御
5. Discuz! X1 ~ X3.1(forum.php)	message=" ed2k://|file|test|'+alert(123)+'|test/"&lots of params	XSS3	防御
6. MyBB <=1.6.12 (search.php)	keywords="qor'("\2a<script>alert(/xss)</script>"	XSS1	防御
7. MyBB <=1.8.2(Usercp.php)	Usertitle="<img src=x onerror=alert(/XSS/);>"	XSS1	防御
8. Advanced Electron Forum v1.0.9 (AEF, index.php)	fredirect="<script>alert(\XSS\)</script>	XSS1	防御
9. Advanced Electron Forum v1.0.9 (AEF, index.php)	Fredirect="" onmouseover="alert(123)"	XSS2	防御
10. 模拟-腾讯娱乐(cgi-bin/search)	<script>document.getElementById("titleshow").innerHTML="按职业检索: \x3Cimg\u0020src=1\u0020onerror=alert(1234)\x3e"; </script>	XSS4	防御
11. 模拟-社区动力(connect.php)	<script>setTimeout("window.location.href='http://www.discuz.net/./a\u0027.replace(/.\u002b/,'javascript:alert(123)/.source);//';", 2);</script>	XSS5	防御

说明: 案例 10 和 11 的第 2 列为结果页面片段, 下划线为用户输入。

ed2k **标准链接**:

ed2k://|file|<**文件名称**>|<**文件大小**>|<**文件哈希值**>|/

攻击向量:

ed2k://|file|test|'+alert(123)+'|test/

结果页面(下划线为恶意注入的片段):

```
<script language="javascript">
$('ed2k_YZ6').innerHTML=htmlspecialchars(unescape
(decodeURIComponent('test')))+'('+alert(123)+' Bytes)';
</script>
```

图 2 Discuz中的跨站脚本攻击实例

Fig. 2 Cross site script in discuz

表 4 XSSCleaner 在测试集 T2 上的时间开销

Table 4 Overhead of XSSCleaner on benchmark T2

案例	应用	Zend 引擎执行时间/ms		性能损失/%
		原始	使用 XSSCleaner	
1	Wordpress	293.0	320.53	9.40
2	Wordpress	254.0	301.46	18.70
3	Wordpress	250.7	299.89	19.60
4	Wordpress	231.9	295.01	27.20
5	Discuz	60.0	64.79	8.00
6	MyBB	56.6	59.23	4.64
7	MyBB	57.1	66.08	15.70
8	AEF	25.3	27.11	7.20
9	AEF	27.1	28.66	5.80

5.3 时间开销

我们在测试集 T2中的9个真实案例上测试 XSSCleaner 给 Zend 引擎带来的时间开销。

表 4 列出 Zend 引擎在原始情况下以及安装了 XSSCleaner 插件后结果页面的生成时间(即 Zend 引擎的执行时间, 不包括 XSSCleaner 进行页面上下文分析的时间)。为了忽略客户端浏览器和网络环境等因素的影响, 我们在服务器本机用 python 脚本发送访问请求, 然后统计响应时间。从表 4 可以得到, XSSCleaner 的污点分析给 Zend 引擎带来平均 12.9%的额外时间开销。

5.4 性能对比及分析

5.4.1 额外空间占用对比

与传统的污点标记及传播方案相比, XSSCleaner 具有较小的额外空间占用。如表 5 所示, XSSCleaner

表 5　污点信息占用额外空间对比

Table 5　Comparison of taint meta-data space allocation

方案	应用类型	污点信息的额外空间占用/%
GRASP[30]	开源 CMS 系统(未指明名称)	30
XSSCleaner	Wordpress 等开源 PHP 应用	≈0

将污点信息直接存储在字节值中，污点信息的额外空间占用为 0%。GRASP[30]修改 PHP 引擎的数据结构，在 ZVAL 结构体中增加一个污点属性域，来标识字符串中各个字符的污点信息，GRASP 的污点标记方式带来 30%的额外空间占用。

5.4.2　时间开销对比

与传统的污点标记及传播方案相比，XSSCleaner 具有较小的额外时间开销，原因在于：一方面，实际的 web 应用程序中，Concat 类和 Add 类字符串拼接操作占大部分；另一方面，XSSCleaner 在 Concat 类和 Add 类操作时能自动实现污点传播，无需额外的时间损耗。

表 6 统计了 5 个开源的 PHP 网站首页中 Concat 类和 Add 类字符串操作以及其余字符串操作的调用次数。可以看出，Concat 类和 Add 类操作占很大比例。

表 7 对比 XSSCleaner 和 WASP[31]在拼接类字符串操作中的时间开销。XSSCleaner 在 Concat 类和 Add 类字符串操作时，能随着拼接操作直接将污点字节值传播到结果字符串中，无需额外操作，时间性能损失约为 0。WASP[31]的污点标记及传播方案中，需要在字符串函数中检查参数中每个字符的污点信息，从而对结果字符串标记相应的污点信息，耗用额外时间，在 concat 和 append 操作上的性能损失高达170%和7100%。

XSSCleaner 的时间损耗来自一些污点净化处理以及在 strpos 等字符串查找类函数和 str_replace 等字符串替换类函数中增加的字符等价判断逻辑。如表 7 所示，XSSCleaner 在 strpos 函数中的性能损失为 16.7%，在 str_replace 函数中的性能损失为 28.97%。

表 8 对比 XSSCleaner 和 ASPIS[32]在生成 wordpress 页面时的时间损耗，证明了 XSSCleaner 是一种轻量级的污点分析方案。

表 6　5 个网站首页的字符串操作统计

Table 6　Statistics of string function usage in 5 applications

操作类型	操作调用次数				
	aef	discuz	expcms	mybb	wordpress
Concat 类(.和.=)	626	302	575	589	3557
Add 类(+)	1	0	1	0	0
其他字符串函数	101	524	324	396	4322

表 7　污点传播的时间开销对比

Table 7　Overhead comparison on string manupulation

方案	字符串函数	性能损失/%
WASP[31]	String.Concat(String)	170.83
	StinrgBuilder.Append(StringBuilder)	7100.00
	StinrgBuilder.Append(String)	2500.00
XSSCleaner	Stringcat	≈0
	concat(.)	≈0
	strpos	16.70
	str_replace	28.97

表 8　页面应答的时间损耗对比

Table 8　Overhead comparison on popular web application

方案	应用类型	时间损耗/%
ASPIS[32]	Wordpress	220
XSSCleaner	Wordpress	18.7

5.5　漏报和误报对比及分析

为了便于对比 XSSCleaner 与已有研究的漏报、误报情况，本文构建了一个跨站脚本攻击测试集 T3。

测试集 T3 在 5 种不同的注入位置分别构造两个攻击向量和两个良性输入，共 10 个攻击向量(表 9)和 10 个良性输入(表 10)。攻击向量 6，8，9 和 10 使用属性编码、关键词拆分和 JS 编码等常见的跨站脚本攻击向量混淆及变形手段。表 10 中良性输入 1~5 为不含分隔符的字符串，良性输入 6~10 为含有分隔符的字符串，良性输入 6 代表用户在富文本框中的良性输入。

本节对比分析 XSSCleaner 等方案在测试集 T3 上的漏报、误报情况，如表 11 所示。

Filter1 为输入过滤方案，过滤关键词 script，onclick 和 eval。此方案由于缺乏页面上下文分析，导致对良性输入造成大量误报。同时，该方案依赖关键词匹配，防御的准确性依赖关键词黑名单的完备性，且不能防御带有混淆或变形的攻击向量，造成较多的漏报。

表 9 测试集 T3 中的 10 个攻击向量

Table 9 10 attacker vectors in benchmark T3

置入位置	攻击向量	攻击向量(混淆或变形)
hello [uname]	1. <script>alert(123)</script>	6. test
<input type="text" value="[uname]" />	2. alice" onclick="alert(123)	7. alice" onmouseover="alert(123)
<script>name="[uname]";</script>	3. alice";eval('alert(123)');"	8. alice";window["ev"+"al"]('alert(123)');"
<script>document.write("hello [uname]");</script>	4. alice<script>alert(123)<\/script>	9. alice\u003c\u0073cript\u003ealert(123)\u003c\u002f\u0073cript\u003e
<script>eval("n='hello [uname]';"); </script>	5. alice';alert(123)//	10. alice\u0027\u003balert(123)//

表 10 测试集 T3 中的 10 个良性输入

Table 10 10 benign user inputs in benchmark T3

置入位置	良性输入(不含分隔符)	良性输入(含有分隔符)
hello [uname]	1. alicescript	6. alice script pku
<input type="text" value="[uname]" />	2. alicescript	7. onclick=alert(123)
<script>name="[uname]";</script>	3. alicescript	8. alice<script1alice@pku.edu.cn
<script>document.write("hello [uname]");</script>	4. alicescript	9. alice<script1alice@pku.edu.cn
<script>eval("n='hello [uname]';"); </script>	5. alicescript	10. alice<script1alice@pku.edu.cn

表 11 测试集 T3 上的漏报和误报情况

Table 11 Comparison of false positives and false negatives on benchmark T3

防御方案	攻击向量										良性输入									
	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	10
Filter1	+	+	+	+	–	–	–	–	–	–	×	×	×	×	×	×	×	×	×	×
Filter2	+	+	+	+	+	+	+	+	+	+	✓	✓	✓	✓	✓	×	×	×	×	×
Enhancer[5]	+	–	–	+	–	–	–	–	–	–	✓	✓	✓	✓	✓	✓	✓	×	×	×
PHPHard[33]	+	+	+	+	+	+	+	+	+	+	✓	✓	×	×	×	✓	✓	×	×	×
XSSCleaner	+	+	+	+	+	+	+	+	+	+	✓	✓	✓	✓	✓	✓	✓	×	×	×

说明: +表示攻击被防御, –表示攻击未被防御; ✓表示允许输入, ×表示不允许输入。

Filter2为输入过滤方案, 过滤集合 D 中所有分隔符。该方案杜绝了跨站脚本的产生, 但用户体验较差, 不允许带有任何分隔符的用户输入。

Enhancer[5]为污点分析结合关键词匹配方案, 在输出页面片段时, 过滤来自用户输入的<script。该方案依赖特定的字符串匹配, 无法防御多样的XSS 注入类型, 且无法防御带有混淆或变形的攻击向量, 漏报较多, 如果不结合页面上下文分析, 容易产生误报。

PHPHard[33]为污点分析结合页面上下文分析的方案, 要求脚本执行节点不能含用户输入的任意字符。该方案不依赖关键词匹配, 能很好地防御跨站脚本攻击, 但该方案的检测策略过于严格, 忽略了字母、数字等普通字符不会影响脚本执行。如表 11 所示, 当 Web 应用程序将用户输入置入一段已有的脚本中会产生误报。与 PHPHard[33]相比, XSS-Cleaner 考虑到普通字符不会引入跨站脚本攻击, 而从分隔符作为跨站脚本攻击检测和判定的切入点, 减少了误报。

如表 11 所示, XSSCleaner 为污点分析结合上下文分析方案, 不依赖关键词匹配, 可防御多种类型的跨站脚本攻击向量, 并能防御混淆、变形后的跨站脚本攻击向量。XSSCleaner 结合上下文分析, 支持富文本框中 等内容的良性输入。XSSCleaner 的误报发生在用户输入的置入位置在一段已有的脚本中, 并且用户输入中含分隔符。只有当两个条件同时发生时, XSSCleaner 才通过判定脚本的可靠性被破坏, 判定为跨站脚本攻击, 引入误报。测试

集 T3 是对不同使用场景的列举，而在实际系统中，条件 1 中场景发生的概率仅为 2.6%(与其他置入位置[23]相比)，因此，即使用户的输入含有分隔符，XSSCleaner 发生误报的情况也比较少见。本文在对 XSSCleaner 的实际使用中，未发生误报。

6　结论

本文提出基于分隔符的跨站脚本攻击防御方法。实验对比表明，利用分隔符字节值进行污点标记及传播是一种轻量级的、有效的污点分析方法；利用结果页面中的分隔符污点信息和页面上下文分析，检测脚本执行节点的合法性和脚本内容的可靠性，能较完备、精确地防御跨站脚本攻击。

虽然本文实验都是针对已知的跨站脚本攻击进行验证，但根据对跨站脚本攻击的机理分析，本文方法同样可以对利用未知漏洞的跨站脚本攻击进行防御。

参考文献

[1] The Open Web Application Security Project. OWASP TOP10 critical web application security risks [EB/OL]. (2016–12–17)[2017–01–01]. https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

[2] The Open Web Application Security Project. Cross-site scripting (XSS) [EB/OL]. (2016–06–04)[2017–01–01]. https://www.owasp.org/index.php/XSS

[3] Budianto E, Jia Y, Dong X, et al. You can't be me: enabling trusted paths & user sub-origins in web browsers // 17th International Symposium on Research in Attacks, Intrusions and Defenses. Gothenburg, 2014: 150–171

[4] Nguyen-Tuong A, Guarnieri S, Greene D, et al. Automatically hardening web applications using precise tainting // Proc of 20th IFIP International Information Security Conference. Chiba, 2005: 295–307

[5] Xu W, Bhatkar S, Sekar R. Taint-enhanced policy enforcement: a practical approach to defeat a wide range of attacks // Proc of the 15th conference on USENIX Security Symposium. Vancouver, 2006: 121–136

[6] Internet Explorer Team. IE8 Security Part IV The XSS Filter-IEBlog-Site Home-MSDN Blogs [EB/OL]. (2008–07–02)[2016–03–25]. http://blogs.msdn.com/b/ie/archive/2008/07/02/ie8-security-part-iv-the-xss-filter.aspx

[7] Bates D, Barth A, Jackson C. Regular expressions considered harmful in client-side XSS filters // Proceedings of the 19th International Conference on World Wide Web. Raleigh, 2010: 91–99

[8] Gundy V M, Chen H. Noncespaces: using randomization to enforce information flow tracking and thwart cross-site scripting attacks // 16th Annual Network & Distributed System Security Symposium (NDSS). San Diego, 2009: 77–84

[9] Nadji Y, Saxena P, Song D. Document structure integrity: a robust basis for cross-site scripting defense // 16th Annual Network & Distributed System Security Symposium (NDSS). San Diego, 2009: 1–20

[10] Jim T, Swamy N, Hicks M. Defeating script injection attacks with browser-enforced embedded policies // Proceedings of the 16th international conference on World Wide Web. Banff, 2007: 601–610

[11] Stamm S, Sterne B, Markham G. Reining in the web with content security policy // Proceedings of the 19th International Conference on World Wide Web. San Diego, 2010: 921–930

[12] Weissbacher M, Lauinger T, Robertson W. Why is CSP failing? trends and challenges in CSP adoption // 17th International Symposium on Research in Attacks, Intrusions and Defenses. Gothenburg, 2014: 212–233

[13] Weichselbaum L, Spagnuolo M, Janc A, et al. CSP is dead, long live CSP ! On the insecurity of whitelists and the future of content security policy // Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. Vienna, 2016: 1376–1387

[14] Calzavara S, Bugliesi M. Content security problems? evaluating the effectiveness of content security policy in the wild // Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. Vienna, 2016: 1365–1375

[15] Fazzini M, Saxena P, Orso A. AutoCSP: automatically retrofitting csp to web applications // International Conference on Software Engineering (ICSE). Firenze, 2015: 336–346

[16] Doupé A, Peinado M, Kruegel C, et al. deDacota: toward preventing server-side xss via automatic code and data separation categories and subject descriptors // Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security. Berlin, 2013:

1205–1216

[17] Pan X, Cao Y, Liu S, et al. CSPAutoGen: Black-box enforcement of content security policy upon real-world websites // Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. Denver, 2016: 653–665

[18] Soni P, Budianto E, Saxena P. The SICILIAN defense: signature-based whitelisting of web JavaScript // Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. Denver, 2015: 1542–1557

[19] The Open Web Application Security Project. XSS (Cross Site Scripting) Prevention Cheat Sheet-OWASP [EB/OL]. (2016–03–27)[2017–01–01]. https://www.owasp.org/index.php/XSS_%28Cross_Site_Scripting%29_Prevention_Cheat_Sheet

[20] The Open Web Application Security Project. XSS Filter Evasion Cheat Sheet-OWASP [EB/OL]. (2016–10–04)[2017–01–01]. https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet

[21] Balzarotti D, Cova M, Felmetsger V, et al. Saner: composing static and dynamic analysis to validate sanitization in web applications // Proc of the 2008 IEEE Symposium on Security and Privacy. Oakland, 2008: 387–401

[22] Bisht P, Venkatakrishnan V N. XSS-GUARD: precise dynamic prevention of cross-site scripting attacks // Proceedings of the 5th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA). Paris, 2008: 23–43

[23] Weinberger J, Saxena P, Akhawe D, et al. A Systematic analysis of XSS sanitization in web application frameworks // Proceedings of the 16th European Conference on Research in Computer Security (ESORICS). Leuven, 2011: 150–171

[24] Saxena P, Molnar D, Livshits B. ScriptGard: automatic context-sensitive sanitization for large-scale legacy web applications // Proceedings of the 18th ACM conference on Computer and Communications Security. Chicago, 2011: 601–614

[25] Samuel M, Saxena P, Song D. Context-sensitive auto-sanitization in web templating languages using type qualifiers // Proceedings of the 18th ACM Conference on Computer and Communications Security. Chicago, 2011: 587–600

[26] The XSSed project. XSS Archive_XSSed [EB/OL]. (2015–03–13)[2016–03–25]. http://www.xssed.com/

[27] Abraham A. The ultimate XSS protection cheat sheet for developers [EB/OL]. [2016–03–25]. https://xenotix.in/The%20Ultimate%20XSS%20Protection%20Cheat%20Sheet%20for%20Developers.pdf

[28] Richardson L. Beautiful Soup Documentation — Beautiful Soup 4 [EB/OL]. [2016–03–25]. https://www.crummy.com/software/BeautifulSoup/bs4/doc/

[29] Kieyzun A, Guo P J, Jayaraman K, et al. Automatic creation of SQL Injection and cross-site scripting attacks // Proc of the 31st International Conference on Software Engineering. Vancouver, 2009: 199–209

[30] Futoransky A, Gutesman E, Waissbein A. A dynamic technique for enhancing the security and privacy of web applications // Conference of Black Hat Usa Briefings. Las Vegas, 2007: 1–16

[31] Halfond W, Orso A, Manolios P. WASP: protecting web applications using positive tainting and syntax-aware evaluation. IEEE Transactions on Software Engineering, 2008, 34(1): 65–81

[32] Papagiannis I, Migliavacca M, Pietzuch P. PHP Aspis: using partial taint tracking to protect against injection attacks // Usenix Conference on Web Application Development. Portland, OR, 2011: 2

[33] 王溢, 李舟军, 郭涛. 防御代码注入式攻击的字面值污染方法. 计算机研究与发展, 2012, 49(11): 2414–2423