

面向规则缺陷的浏览器 XSS 过滤器测试方法

桂智杰^{1,2}, 舒辉^{1,2}

(1. 中国人民解放军信息工程大学网络空间安全学院, 河南 郑州 450001;
2. 数学工程与先进计算国家重点实验室, 河南 郑州 450001)

摘要: 为了缓解跨站脚本 (XSS, cross-site scripting) 攻击, 现代浏览器使用 XSS 过滤器进行防御, 现有方法很难有效对浏览器 XSS 过滤器的安全性进行测试与评估。规则缺陷是浏览器 XSS 过滤器实现过程中的缺陷和安全隐患。面向浏览器 XSS 过滤器规则缺陷, 给出其形式化定义, 设计测试样例和场景生成算法。为了定量测试与评估不同浏览器 XSS 过滤器的过滤水平, 结合过滤成功率、误报率、输入损耗计算过滤能力。基于所提方法, 设计原型系统对几种主流浏览器 XSS 过滤器进行自动化测试, 得到了不同浏览器的 XSS 过滤能力。经过实际测试, 该系统具备发现未公开漏洞的能力。

关键词: 跨站脚本攻击; 浏览器 XSS 过滤器; 规则缺陷; 过滤能力

中图分类号: TP309

文献标识码: A

doi: 10.11959/j.issn.2096-109x.2018093

Rule-defect oriented browser XSS filter test method

GUI Zhijie^{1,2}, SHU Hui^{1,2}

1. School of Cyberspace Security, Information Support Engineering University of PLA, Zhengzhou 450001, China
2. State Key Laboratory of Mathematical Engineering and Advanced Computing, Zhengzhou 450001, China

Abstract: In order to alleviate XSS (cross-site scripting) attacks, modern browsers use XSS filters for defense. It is difficult to effectively test and evaluate the security of browser XSS filters. The rule-defect is the defect and security problem in the implementation process of browser XSS filter. The formal definition, design test sample and scene generation algorithm were presented for browser XSS filter rule-defects. In order to quantitatively test and evaluate the filtering level of different browser XSS filters, combined with filtering success rate, false positive rate, input loss calculation filtering ability. Based on the proposed method, the prototype system is designed to automate the testing of several mainstream browser XSS filters, and the XSS filtering capabilities of different browsers are obtained. Further, after actual testing, the system also has the ability to discover undisclosed vulnerabilities.

Key words: cross-site scripting attack, browser XSS filter, rule-defect, filtering capability

1 引言

XSS 攻击^[1]是一种利用服务器端防御不严导致针对 Web 应用前端的攻击方式。在不加防御的情况下, 攻击者利用 XSS 漏洞可以向用户发送恶意脚本, 最终被浏览器解释执行, 达到盗取 cookie

和登录账号等目的。到目前为止, 攻击者发送的恶意脚本已经不限于 JavaScript 片段, 而是包括 HTML、SVG 或者 Flash 等可能执行的任何其他类型的代码。

由于 XSS 攻击的影响, 如何缓解 XSS 显得尤为重要。目前有如下几种防御或者缓解方式:

收稿日期: 2018-10-10; 修回日期: 2018-11-05

通信作者: 桂智杰, 2448405909@qq.com

浏览器 XSS 过滤器、内容安全策略、HTML sanitizers 和 WAF (Web application firewall)。除基于客户端浏览器 XSS 过滤器外,其余几种都有较为通用的绕过思路和一般性安全问题,并且有系统的评估和测试方法。然而,目前测试评估浏览器 XSS 过滤器的方法还比较欠缺,无法有效地综合评估现有 XSS 过滤器的过滤能力。在此基础上,挖掘浏览器 XSS 过滤器模块漏洞主要采取模糊测试的方法,该方法为了提高代码覆盖率,导致产生大量误用测试样例,漏洞挖掘效率低。

Microsoft 从 Internet Explorer 8 开始实现了初代的 XSS 过滤器,称为 XSS Filter。浏览器内核 Webkit 自身也内嵌一个 XSS 过滤器,称为 XSS Auditor,故基于 Webkit 的浏览器(如 Chrome、Safari 等)使用的是 XSS Auditor,国内的一些浏览器(如搜狗和猎豹)也是内置 XSS Auditor。Firefox 浏览器由于多种原因并没有 XSS 过滤器,在本文测试过程中主要用于对照实验。本文中浏览器 XSS 过滤器泛指所有浏览器 XSS 的过滤模块,XSS Filter 特指 Internet Explorer (IE)浏览器的 XSS 过滤模块,XSS Auditor 特指基于 Webkit 内核浏览器的 XSS 过滤模块。

浏览器 XSS 过滤器研究领域已经有一些关于 XSS Filter 设计理念和实现原理的文章和综述。Ross 在文献[2]介绍了 IE 8 浏览器 XSS Filter 的体系结构和实现思路。IE 浏览器 XSS Filter 通过监控 Request URL,基于正则表达式进行匹配 XSS 攻击向量。若匹配成功,则建立一个签名(Signature),并在 HTTP Response 中使用中性字符替换。

在 IE XSS Filter 提出后,Nava 等在文献[3]给出了 IE 8 XSS Filter 滥用导致的问题。该方法利用 IE XSS Filter 正则替换存在的漏洞,注入虚假参数,导致 XSS 过滤器响应,对正常页面进行修改。

2010 年 Bates 等在文献[4]中提出了基于正则表达式的 XSS Filter 存在的错报误报问题和过滤不严被绕过问题,并在 Webkit 中构建了 XSS Auditor,在文章中说明了该模块的体系架构和技术原理。XSS Auditor 作用于网页解析时,通过对

Webkit 解析出的 Token 序列进行审计,检测如内联事件处理程序和资源引用等危险属性,并在 Response 中进行匹配。

综上所述,在浏览器 XSS 过滤器研究领域,目前尚无很好的测试过滤器的方法和定量的测试结果。在评估的基础上,现有的浏览器 XSS 过滤器漏洞挖掘效率低,无法有效找出其潜在的安全问题。

针对上述问题,本文有如下贡献。

1) 通过源码审计或者逆向分析,分析目前 2 种主流浏览器 XSS 过滤器原理以及实现上的问题,列举了 4 种典型的规则缺陷。

2) 形式化定义规则缺陷,并且设计面向规则缺陷生成测试样例和测试场景的算法。考虑过滤成功率、误报率、输入损耗,给出计算浏览器 XSS 过滤器过滤能力的公式。

3) 设计与实现原型测试系统,对目前几种主流的浏览器进行实际测试,得到了不同浏览器 XSS 过滤器的过滤能力。

4) 在分析规则缺陷和实际测试的基础上,对浏览器 XSS 过滤器潜在的安全漏洞进行挖掘,找到了多个浏览器最新版 XSS 过滤器的未公开漏洞,并通知厂商及时修补。

2 浏览器 XSS 过滤器规则缺陷分析

本文主要探讨 Chrome 和 Internet Explorer 2 种浏览器,其 XSS 过滤器有着不同的实现方式。Chrome 的 XSS Auditor 内嵌于 Webkit,作用在 HTML 解析过程中,对 HTML 解析器生成的词语流进行过滤。Internet Explorer 的 XSS Filter 实现在 mshtml.dll 动态链接库中,作用于服务器与客户端之间的数据请求与响应,对浏览器返回数据进行匹配过滤。

在对浏览器 XSS 过滤器模板进行分析和调试的基础上,下面介绍 2 种浏览器 XSS 过滤器的规则缺陷。

2.1 Chrome XSS Auditor 规则缺陷

XSS Auditor 作用于 HTML 语言的解释构建 DOM 过程,在词语生成后,Webkit 调用 XSS Auditor 验证词语流来缓解 XSS 攻击。XSS Auditor 模块架构如图 1 所示。

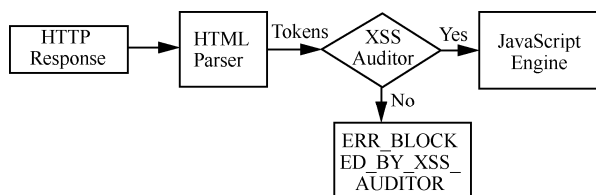


图 1 XSS Auditor 模块架构

HTML 是将 HTML 从字节流解释构建为 DOM 树的过程,由 HTML 解释器(HTML Parser)负责实施。在 Webkit 实现过程中,HTML Parser 模块中的 HTMLDocumentParser 类用于完成上述过程。在上述过程中,输入的资源首先从字节流转化为词语流(Tokens)。当词语生成后,XSS Auditor 模块对词语进行过滤。经过对 XSS Auditor 实现源码的阅读与实际测试,发现存在 2 种典型的规则缺陷,并且这里的规则缺陷适用于所有使用 Webkit 的浏览器。

1) 规避 HTML 解析

正如上述 XSS Auditor 模块架构所描述的,该过滤器基于 HTML 解析过程,如果返回代码没有经过 HTML 解析,就完全不会启动 XSS Auditor^[5]。

这种情况在实际 Web 开发中时常出现,如 Web 应用前端页面中使用 eval 函数执行从服务器端返回的数据,此时就不进行 HTML 解析,不启动 XSS Auditor,这样就从避免 HTML 解析的角度完全绕过 XSS Auditor 的过滤,这是一种 XSS Auditor 实现上的规则缺陷,利用该规则缺陷,在合适场景的情况下,将可以绕过该过滤器。

2) 请求代码与解析代码相异

在 XSS Auditor 作用中最为关键的步骤是比对解析后代码与请求是否相同,若相同,则判定存在反射性 XSS 攻击。从反射性 XSS 原理上来讲,就是通过攻击者发送恶意 Request,然后浏览器解析服务器端的 Response,输出到页面上,从而执行 JavaScript 代码。

想要绕过 XSS Auditor,这是与 XSS 攻击原理相矛盾的。这种情况下较为巧妙的绕过思路:通过构造 payload 令请求代码与解析代码相异,达到绕过的目的。具体来说,有 2 种情况可以使用该规则缺陷:加密传输场景和 HTML 规范化渲染。

2.2 Internet Explorer 规则缺陷

XSS Filter 并不是作用于 HTML 页面解析,

而是作用于客户端浏览器与服务器之间,拦截浏览器与服务器之间请求和返回的数据。XSS Filter 模块架构如图 2 所示。

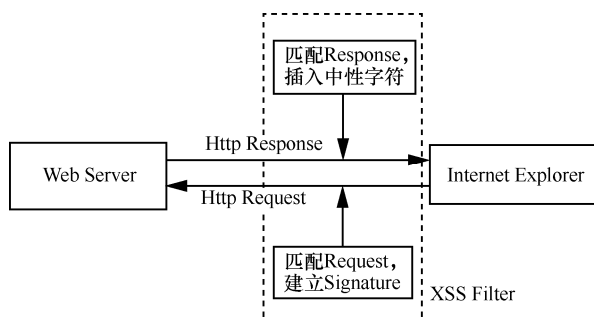


图 2 XSS Filter 模块架构

细化来看,XSS Filter 的工作逻辑为:浏览器监控 Request URL,通过正则表达式匹配 Http Request。若匹配成功,则认为识别出一组 XSS 攻击向量,就建立一个签名。生成的 Signature 用于扫描 HTTP Response,如果在响应中也扫描到 XSS 攻击向量,就返回修改。经过对 mshtml.dll 的逆向分析,该过滤器有如下 2 种规则缺陷。

1) 正则表达式非预期利用

在过滤器测试系统中,不仅要看其过滤的成功率,还要看误报率。Internet Explorer 的 XSS Filter 采用正则表达式匹配 Request,并且在 Response 中将匹配到的字符替换为中性字符,这是一种较为彻底的中和 XSS 攻击向量的方式。

正则表达式被期望用来寻找匹配攻击向量进行替换,通过构造可以导致正则表达式被非预期利用。若用户输入了匹配正则规则的 Request 参数,但这个参数并没有返回到页面,此时若页面已经有匹配项,XSS Filter 无法识别是用户输入与页面已经存在的,就将页面中已有的匹配项做替换。

这种正则表达式的非预期使用将导致 HTML 页面被破坏^[6],在场景合适的情况下,还会利用误报导致 XSS。

2) 正则匹配规则绕过

XSS Filter 的匹配是基于正则表达式的,该表达式在 mshtml.dll 动态链接库中。随着 Internet Explorer 的更新,这些表达式愈加复杂。图 3 显示了 mshtml.dll 动态链接库中部分正则

结果如下。

$$[html_{text}][<][tag_{name}][tag_{attrn}][tag_{attrv}][html_{textin}]$$

在测试系统中，场景是指后端服务器处理程序，一般使用 PHP 语言编写，用于处理前端的 Request。对场景结构特征进行分析，建立场景模板向量。

$$sceneTemp = \{scene_{text}, logic_{judge}, logic_{handle}, comm\}$$

多元组元素分别为场景文本、场景判断逻辑、场景处理逻辑和注释。以一个测试场景（测试场景一般用 PHP 编写）为例，形式化后的结构如图 4 所示。

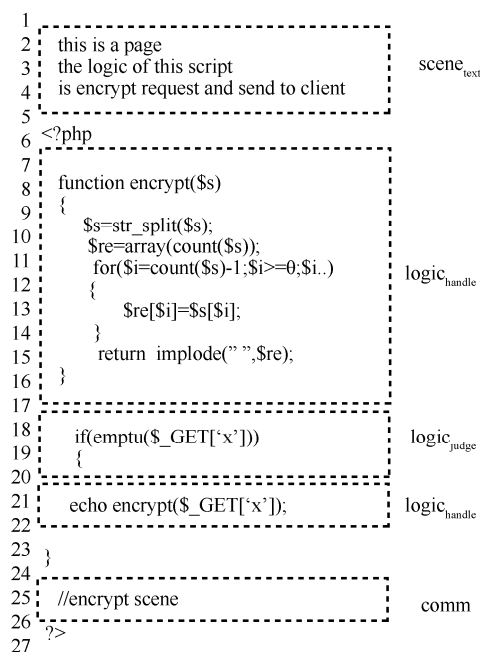


图 4 形式化后的测试场景

3.2 测试样例和场景生成算法

3.2.1 算法概述

在形式化定义规则缺陷向量、攻击模板向量和场景模板向量的基础上，可以面向规则缺陷生成测试样例和测试场景，用于对浏览器 XSS 过滤器进行自动化测试。为了将生成过程表述清楚，定义向量的二元运算符模板嵌入符号。

定义 1 模板嵌入符号。模板嵌入符号 $\Rightarrow$ ，该二元运算符表示将形式化后的规则缺陷向量 $ruleDef$ 嵌入 XSS 模板攻击向量 $sampleTemp$ 和场景模板向量 $sceneGen$ 的过程。其数学意义为在一

个集合中，根据已有规则集合，从集合中挑选子集。模板嵌入符号计算过程如下。

$$M(i) \Rightarrow N(j) = \{p \in N(M(j)) \mid 0 \leq n; 0 \leq j \leq m\}$$

在生成算法中，定义 2 个子算法，分别实现测试样例生成子算法 $sampleGen$ 和测试场景生成子算法 $sceneGen$ 。

3.2.2 测试样例生成

测试样例是一组 XSS 攻击向量，为了系统实现便捷，攻击向量的效果统一设置为弹窗，如果输入之后执行脚本产生弹窗，则认为页面产生了 XSS，XSS 过滤器没有成功防御；同时，会设置一些干扰样例，这些样例无法产生弹窗效果，但是存在一些敏感字符等^[9]，若过滤器报告页面存在 XSS 漏洞，则产生误报。如果对测试样例进行细分，可以分为基本测试样例集合 W_{bas} 、混淆集合 W_{mix} 以及面向规则缺陷生成的场景测试样例集合 W_{rul} 。其中， W_{bas} 和 W_{mix} 为固定集合，不需要再次生成和改变。

基本测试集合 W_{bas} 是常见的用于测试 XSS 漏洞是否存在的一些攻击向量，没有场景的约束。经过总结，基本测试集合如表 2 所示。

表 2 基本测试集合攻击向量

描述	攻击向量
script 标签直接注入 JavaScript	<script>alert("xxxx");</script>
引用文件属性加载 JavaScript	<imgsrc=http://xsss.js>
内联事件处理函数	<imgonerror=alert(1)>
CSS 表达式	<div style="xxx:expression(alert(1))">
.....	

面向规则缺陷产生场景测试样例集合 W_{rul} 是针对不同规则缺陷场景定义的符合场景需要的测试样例集合。测试样例生成算法 $sampleGen$ 基本逻辑为遍历 $sampleTemp$ ，取多元组的每个元素基于规则缺陷向量 $ruleDef$ 进行不同处理，最终可以得到面向规则缺陷生成的场景测试样例集合 W_{rul} ，该集合每个元素 w_i 为面向规则缺陷产生的 XSS 攻击向量，用多元组表示如下。

$$w_i = \{html_R_{text}, tag_R_{name}, tag_R_{attrn}, tag_R_{attrv}, html_R_{textin}, comm_R\}$$

该集合计算过程如下：

$$W_{rul} = \{w_i \mid 0 \leq i \leq n; w_i = \text{sampleGen}(\text{sampleTemp}, \text{ruleDef})\}$$

计算 sampleGen 的算法公式中, $\text{html_R}_{\text{text}}$ 、 $\text{html_R}_{\text{textin}}$ 和 comm_R 都是无关项, 计算过程就是在原来的式子上加上随机字符

$$\begin{aligned} \text{html_R}_{\text{text}} &= \text{html}_{\text{text}} \cup \text{random} \\ \text{html_R}_{\text{textin}} &= \text{html}_{\text{textin}} \cup \text{random} \\ \text{comms} &= \text{comm_R} \cup \text{random} \end{aligned}$$

结合模板嵌入符号可以计算其余 3 项内容。

$$\begin{aligned} \text{tag_R}_{\text{name}} &= \text{loc} \Rightarrow \text{tag}_{\text{name}} \\ \text{tag_R}_{\text{attrn}} &= (\text{loc} \Rightarrow \text{tag}_{\text{attrn}}) \cup (\text{way} \Rightarrow \text{tag}_{\text{attrn}}) \\ \text{tag_R}_{\text{attrv}} &= \text{way} \Rightarrow \text{tag}_{\text{attrv}} \end{aligned}$$

混淆集合 W_{mix} 是一些敏感字符、页面中已经存在的一些特殊字符等组成的测试集合。

3.2.3 测试场景生成

测试场景是一个存在 XSS 漏洞的后端代码, 为了测试方便, 所有场景基本逻辑都是将输入返回到前端页面, 程序本身不加以过滤或者转义处理。在自动化测试过程中, 为了进行对照实验, 同时设置基本测试场景和面向规则缺陷的测试场景进行测试。本系统测试场景一般使用 PHP 语言编写。

测试场景生成算法 sceneGen 基本逻辑为遍历 sceneTemp , 取场景向量的每个元素基于规则缺陷 ruleDef 进行填充, 最终可以得到多场景集合 S_{rul} 。其中, 该集合每个元素为 s_i , 该集合用多元组表示如下

$$s_i = \{\text{scene_R}_{\text{text}}, \text{logic_R}_{\text{judge}}, \text{logic_R}_{\text{handle}}, \text{comm_R}\}$$

多场景集合 S_{rul} 计算过程如下

$$S_{\text{rul}} = \{w_i \mid 0 \leq i \leq n; s_i = \text{sceneGen}(\text{sceneTemp}, \text{ruleDef})\}$$

与测试样例生成过程类似, $\text{scene_R}_{\text{text}}$ 和 comm_R 都是无关项, 计算方法为在原来的模板代码中加入随机字符。

$$\begin{aligned} \text{scene_R}_{\text{text}} &= \text{scene}_{\text{text}} \cup \text{random} \\ \text{comm_R} &= \text{comm} \cup \text{random} \end{aligned}$$

其余几项的计算方法如下

$$\begin{aligned} \text{logic_R}_{\text{judge}} &= \text{loc} \Rightarrow \text{logic}_{\text{judge}} \\ \text{logic_R}_{\text{handle}} &= \text{way} \Rightarrow \text{logic}_{\text{handle}} \end{aligned}$$

3.3 计算过滤能力算法

在自动化测试系统中, 为了直观地评价 XSS 过滤器的过滤能力 (filter capacity), 系统根据过滤成功率、误报率、输入损耗计算该值。下面分别介绍过滤能力及各个子参数的意义。

定义 2 过滤成功率。在测试过程中, 输入非混淆集合的情况下, 如果过滤器成功进行过滤或者阻止脚本执行弹窗, 则认为成功过滤一次。过滤成功率 p_{suc} 是某个测试场景 T_i 下, 过滤成功次数 c_{suc} 与所有测试次数 c_{sum} 之间的比例。

定义 3 误报率。在测试过程中, 输入混淆集合 W_{mix} 情况下, 如果过滤器产生组织页面渲染或产生警报, 则认为产生一次误报。误报率 p_{fai} 是某个测试场景 T_i 下, 误报次数 c_{fai} 与所有测试次数 c_{sum} 之间的比例。

定义 4 输入损耗。在某个过滤成功的测试中, 为了达到过滤目的, 对输入做修改或者删除。为了定量表示, 本文定义输入损耗值 w_{los} 计算算法, 如算法 1 所示。

算法 1 计算输入损耗值的算法

输入 测试样例集合 w_{pay} 和从服务器接收的返回值集合 w_{rep} , 加下标表示集合值

输出 每一次过滤成功测试情况下的输入损耗值 w_{los}

- 1) 初始化: $w_{\text{los}} = 0$
- 2) if 过滤成功{
- 3) for ($i=0$; $i < w_{\text{pay}}$; $i++$)
- 4) for($j=0$; $j < \text{len}(w_{\text{pay}_i})$; $j++$)
- 5) 如果返回值与请求的样例值不相同{
- 6) $w_{\text{los}}++$;
- 7) }
- 8) }

定义 5 过滤能力。为了定量和直观地评估一个 XSS 过滤器的过滤水平, 本文定义过滤能力值与过滤成功率 p_{suc} 和误报率 p_{fai} 有如下基本对应关系。

$$\text{FilterCapacity} \propto \frac{p_{\text{suc}}}{p_{\text{fai}}}$$

同时, 在过滤成功情况下, 需要考虑过滤的损失, 为了合理评估各个测试样例的过滤损失, 需要考虑其样例的长度。这里定义一个测试场景

下, 定量计算特定测试场景的过滤损失。

$$w_{scc}^{los} = \frac{\sum_i^{c_{suc}} \frac{w_{los_i}}{len_{pay_i}}}{c_{suc}}$$

综合上述定义, 该值计算公式为 (m 为测试场景个数)

$$FilterCapacity = \frac{\frac{\sum_i^m p_{sus_i} - w_{los}^{scc_i} + 1}{p_{fai_i}}}{m}$$

$$w_{los}^{scc_i} = \frac{\sum_j^{c_{suc}} \frac{w_{los_j}}{len_{pay_j}}}{c_{suc}}$$

4 系统设计及实验分析

为了验证本文方法的有效性, 本节构建了基于该方法的原型系统^[10-11], 对目前常用的 3 种浏览器 (Internet Explorer、Chrome、Firefox) 进行了测试, 得到了测试结果。在测试过程中, 基于该测试方法, 找到了多个浏览器 XSS 过滤器未公开漏洞。

4.1 系统设计与实现

测试系统架构如图 5 所示。

生成测试样例模块: 用于实现生成测试样例集合, 包括基本测试样例集合 W_{bas} 、混淆集合 W_{mix} 以及面向规则缺陷生成的场景测试样例集合 W_{rul} , 其中, W_{rul} 有多个, 分别针对不同规则缺陷场景。

生成场景测试模块: 分为基本测试场景和面向规则缺陷的测试场景, 使用 PHP 语言编写后端场景逻辑, 布置在 Web 服务器上^[12]。

捕获记录模块: 捕获页面弹窗, 并收集记录^[13], 通过过滤能力计算公式计算 $FilterCapacity$ 。

4.2 实验结果分析

本节利用原型系统, 选择不同的浏览器进行自动化测试, 验证面向规则缺陷的浏览器 XSS 过滤器测试方法对过滤器的评估和测试水平。进一步地, 将该方法用于对主流浏览器 XSS 过滤器的漏洞挖掘^[14]。本节测试环境如下:

测试机器配置: Windows 10 1803 64 bit;
python 2.7.13 64 bit;

测试服务器环境: Centos 7 32bit; php version 5.5.38;

测试浏览器版本: Chrome 68; Firefox 61; IE 11。

4.2.1 多场景测试结果

1) 基本测试场景

在基本测试场景中, 测试代码的基本逻辑是将浏览器通过 GET 方式传参内容回显到前端页面上, 所以该场景 Web 应用有最为基础的 XSS 漏洞。在基本测试样例中, 输入基本测试样例和混淆样例进行测试, 结果如表 3 所示。

上述实验结果表明: 对于基本的测试集合, 不加任何其他逻辑的 XSS 漏洞, IE 和 Chrome 的 XSS 过滤器能够进行过滤。但在此基础上, IE 浏览器的 XSS Filter 过滤存在较为严重的误报问题。

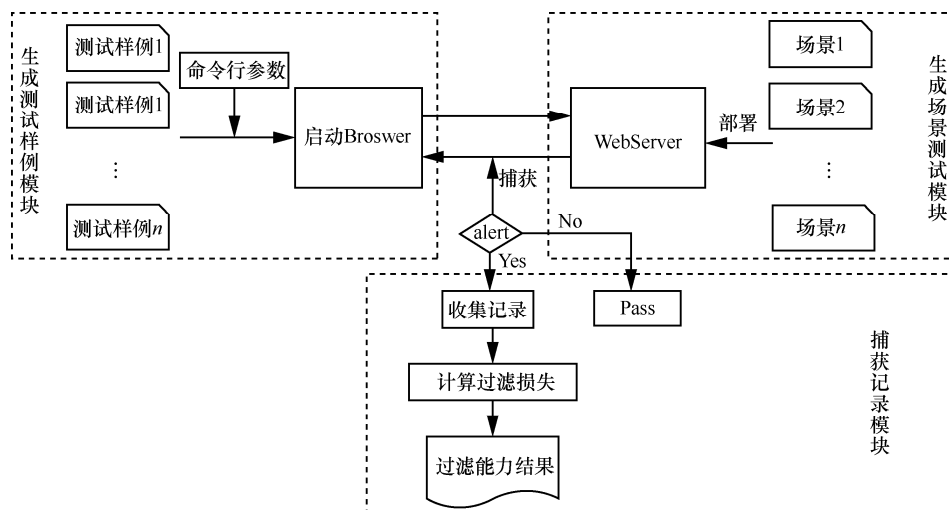


图 5 测试系统架构

表 3 基本测试场景测试结果

名称	版本号	测试样例集合	过滤成功数量	过滤失败数量	误报数量
IE	11.228.17134.0	基本测试样例	30	0	0
		混淆样例	0	0	12
Chrome	68.0.3440.106	基本测试样例	30	0	0
		混淆样例	0	0	4
Firefox	61.0.2	基本测试样例	0	30	0
		混淆样例	0	0	0

2) HTML 解析规避场景

规避 HTML 解析是本文提出的关于 Chrome XSS Auditor 实现上的一种规则缺陷,这里面向该规则缺陷编写 HTML 解析规避场景进行测试。为了实现该场景,前端使用 eval 等动态执行代码函数。在这种情况下,测试结果如表 4 所示。

上述实验结果表明:对于基于 HTML 解析规避的规则缺陷的场景,IE 和 Chrome 的 XSS 过滤器都不能很好地进行过滤,并且 Chrome 的 XSS Auditor 对该测试样例的过滤效果更差。

3) 加密传输场景

在加密传输的某些场景,前端请求值与服务端返回值不相同。浏览器 XSS 过滤器从实现原理上无法对这种存在 XSS 漏洞的情形进行处理,利用这种原理上的规则缺陷,生成一个前后端简单的加密场景,测试结果如表 5 所示。

上述实验结果表明:对于基于加密传输的规则缺陷的场景,IE 和 Chrome 的 XSS 过滤器完全无法进行防御。也就是说,如果 Web 应用涉及加

密传输,浏览器 XSS 过滤器无法对该应用提供保护^[15]。

4.2.2 浏览器 XSS 过滤器过滤能力分析

经过上述实验,根据浏览器 XSS 过滤器过滤能力的定义公式,可以计算得到上述 3 种浏览器 XSS 过滤器的过滤能力,结果如表 6 所示。

表 6 不同浏览器的过滤能力

名称	版本号	过滤能力
IE	11.228.17134.0	1.38
Chrome	68.0.3440.106	3.75
Firefox	61.0.2	0

4.3 未公开漏洞发现

在分析多款浏览器 XSS 过滤器规则缺陷的基础上,使用本文方法进行实际测试,发现多个浏览器 XSS 过滤器的未公开漏洞(CVE-2018-16415、CVE-2018-16734、CVE-2018-16735 等)。本文以国内最新搜狗浏览器(截至测试,最新版本号为 8.0.5.28300)为例,该浏览器 XSS 过滤器模块存

表 4 HTML 解析规避场景测试结果

名称	版本号	测试样例集合	过滤成功数量	过滤失败数量	误报数量
IE	11.228.17134.0	场景测试样例	20	10	0
Chrome	68.0.3440.106	场景测试样例	13	17	0
Firefox	61.0.2	场景测试样例	0	30	0

表 5 加密传输场景测试结果

名称	版本号	测试样例集合	过滤成功数量	过滤失败数量	误报数量
IE	11.228.17134.0	场景测试样例	0	30	0
Chrome	68.0.3440.106	场景测试样例	0	30	0
Firefox	61.0.2	场景测试样例	0	30	0

在处理零字节不严格问题，经过合理构造，可以做到无场景限制绕过其过滤模块（由于还未修补，这里并不直接给出攻击向量），已经通知厂商（搜狗安全应急响应中心）及时修补，该漏洞目前已经被确认。

5 结束语

本文针对浏览器 XSS 过滤器提供了一种面向规则缺陷的多场景测试方法，该方法在分析浏览器 XSS 过滤器规则缺陷的基础上，给出规则缺陷形式化定义及基于规则缺陷生成测试样例和测试场景的算法。为了更加直观地评估各个浏览器 XSS 过滤器的过滤能力，定量定义了浏览器 XSS 过滤器过滤能力，并给出了计算公式。本文实现了原型自动化测试系统，选择测试了部分主流浏览器 XSS 过滤器。实验结果表明，所提方法具有良好的测试与评估 XSS 过滤器的能力，且该方法生成的测试样例集合能够发现未知漏洞。

参考文献：

- [1] GUPTA BB, GUPTA S, GANGWAR S, et al. Meena PK (2015) cross-site scripting (XSS) abuse and defense: exploitation on several testing bed environments and its defense[J]. J Inf Privacy Secur 11(2): 118-136.
- [2] ROSS D. IE8 security part IV: the XSS filter[EB/OL]. <http://blogs.msdn.com/b/ie/archive/2008/07/02/ie8-security-part-iv-the-xss-filter.aspx>, July 2008.
- [3] NAVA E, LINDSAY D. Abusing Internet Explorer 8's XSS filters[C]// BlackHat Europe. 2010.
- [4] BATES D, BARTH A, JACKSON C. Regular expressions considered harmful in client-side XSS filters[C]//19th International World Wide Web Conference. 2010.
- [5] LEKIES S, STOCK B, JOHNS M. 2014. A tale of the weaknesses of current client-side filtering[C]//Black Hat Europe. 2014.
- [6] 刘雅楠. Web 前端攻击及安全防护技术研究与实现[D]. 北京: 北京邮电大学, 2017.
- LIU Y N. Research and implementation of Web front-end attack and protection technology[D]. Beijing: Beijing University of Posts and Telecommunications, 2017.
- [7] LIU J D. An improved XSS vulnerability detection method based on attack vector[C]//2018 International Conference on Modeling, Simulation and Analysis. 2018:6.
- [8] PAN J K, et al. Taint inference for cross-site scripting in context of URL rewriting and HTML Sanitization[J]. ETRI Journal, 2016(2): 376-386.
- [9] 黄娜娜, 万良. 一种基于序列最小优化算法的跨站脚本漏洞检测技术[J]. 信息安全, 2017(10):55-62.
- HUANG N N, WAN L. A cross site script vulnerability detection technology based on sequential minimum optimization algorithm[J]. Netinfo Securi, 2017(10):55-62.
- [10] SALUNKE S S. Selenium Web driver in Python: learn with examples[M]. CreateSpace Independent Publishing Platform, 2014.
- [11] BEKRAR S, BEKRAR C, GROZ R, et al. Finding software vulnerabilities by smart fuzzing[C]//IEEE Fourth International Conference on Software Testing Verification and Validation. 2011.
- [12] ANASTASIOS S, NTANTOGIAN C, XENAKIS C. Bypassing XSS auditor: taking advantage of badly written PHP code[C]//IEEE International Symposium on Signal Processing and Information Technology. 2015: 290-295.
- [13] Taint inference for cross-site scripting in context of URL rewriting and HTML sanitization[J]. ETRI Journal, 2016, (2): 376-386.
- [14] LIU B W. XSS vulnerability scanning algorithm based on anti-filtering rules[C]//International Conference on Computer, Electronics and Communication Engineering. 2017.
- [15] LEKIES S, KOTOWICZ K, GROB S, et al. Code-reuse attacks for the Web: breaking cross-site scripting mitigations via script gadgets[C]//ACM SIGSAC Conference on Computer and Communications Security. 2017: 1709-1723.

[作者简介]



桂智杰（1996-），男，安徽马鞍山人，中国人民解放军信息工程大学硕士生，主要研究方向为软件漏洞挖掘与利用、逆向分析、嵌入式设备。

舒辉（1974-），男，江苏盐城人，中国人民解放军信息工程大学教授，主要研究方向为逆向工程。