**修订记录Revision record**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 日期  Date | 修订版本Revision version | 修改描述  change Description | 作者  Author | 审核 |
| 2016-04-25 | 0.1 | 初稿完成 | l00316555 |  |
| 2016-04-27 | 0.2 | 根据编码组内评审意见刷新 | l00316555 |  |
| 2016-04-28 | 1.0 | 补充工具部分 | 安全工具组 |  |
| 2016-04-28 | 1.1 | 修改css转码的小问题 | l00316555 |  |
|  |  |  |  |  |

# 背景

跨站脚本（XSS）攻击是一种注射类型的攻击，攻击者恶意将脚本注入受信任的网站。XSS攻击发生时，攻击者使用一个Web应用程序发送恶意代码，一般在浏览器端脚本的形式，为不同的终端用户。允许这些攻击成功的缺陷是相当普遍的，出现在任何一个网络应用程序对用户的输入或者输出不验证或者不编码。

攻击者可以利用XSS发送一个恶意脚本给不知情的用户。最终用户的浏览器无法知道该脚本不应该被信任，并且将执行该脚本。因为它认为脚本是来自可信来源，恶意脚本可以访问网站的Cookie，Session token，或其他敏感信息，当然这些脚本也可以重写HTML页面的内容。

XSS漏洞常见类型

## 存储型XSS

存储型XSS通常发生在用户输入的信息存储在目标服务器，如数据库中的信息，在论坛、日志、评论等领域。受害者能够通过Web应用程序直接访问到存储的没有进行安全保护的数据。随着HTML5的发展，我们可以预见的另一种存储型攻击是永久存储在受害者的浏览器，如HTML5本地数据库

## 反射性XSS

反射型XSS发生在返回用户直接输入后的信息，特别是以用户输入为展现结果。

## Dom XSS

由Amit克莱因定义的，他发表的关于这个问题的的第一篇文章中(<http://www.webappsec.org/projects/articles/071105.shtml>)，基于DOM的XSS是一种形式的XSS在整个污染数据流从源到端都发生在浏览器，即数据来源的DOM中，污染也在DOM中，整个数据流不离开浏览器。例如，源（如恶意数据读取）可以将网页的URL（例如，文件位置、href），也可以是HTML元素，或者是一个敏感的方法调用导致恶意数据的执行（例如，文档编写）。

经对近一年ICSL发现的电软web类问题统计，从统计数据看，跨站脚本问题单达到（210） ，主要发生存储型XSS。

发现XSS注入问题的产生原因主要包括：外部输入参数校验不当、安全输出标签没有使用、输出值没有编码。从问题产生原因来看，这些XSS注入问题绝大部分是很容易识别的。产品在开发过程中，增加开发人员的XSS注入防范意识；测试阶段，使用有效的代码静态扫描工具或者黑盒测试扫描工具对代码进行扫描，都可以有效消除或者减少该类问题。

本文档针XSS注入漏洞进行分析，并给出解决方案、排查指导，建议各产品线对现有产品做排查，并对问题整改。

# 漏洞案例

本节将对几个典型的XSS注入漏洞进行分析。

## 存储型的XSS报告

|  |  |  |  |
| --- | --- | --- | --- |
| BDI“预警消息设置—流程超时预警设置”页面存在存储型XSS | | | |
| **安全问题号** | DTS2016012110798 | **日期** | 2014-7-27 |
| **严重级别** |  | | |
| **产品简介** | | | |
|  | | | |
| **问题所在模块功能、使用场景说明** | | | |
|  | | | |
| **【问题现象】**  **在BDI“预警消息设置—流程超时预警设置”页面，注入如下短语：**http://dts.huawei.com/net/dts/fckeditor/download.ashx?Path=8AVOE1N2lm%2fs8tB2Ku5%2bgDePB9yER%2fGiX6WbyFvhxzJgXplwdxQyCqSQeko%2bTWnC  触发XSS注入  http://dts.huawei.com/net/dts/fckeditor/download.ashx?Path=8AVOE1N2lm%2fs8tB2Ku5%2bgDePB9yER%2fGiX6WbyFvhxzKTgzzpBB5%2btNplZh3hTlLZ | | | |
| **漏洞验证流程** | | | |
|  | | | |
| **问题详细技术分析** | | | |
| **输出到客户端的数据来自不可信的数据源没有做相应的编码或者转义** | | | |
| **对业务的影响** | | | |
|  | | | |
| **解决方案** | | | |
|  | | | |

## 反射型的XSS报告

|  |  |  |  |
| --- | --- | --- | --- |
| UMS dependedProduct的selectedProductIds参数未经校验，导致反射型XSS攻击 | | | |
| **安全问题号** | DTS2015071403560 | **日期** |  |
| **严重级别** |  | | |
| **产品简介** | | | |
|  | | | |
| **问题所在模块功能、使用场景说明** | | | |
| 步骤一：在Dependent Product Name中点击“选择”按钮，拦截报文；    拦截报文如下    通过手工修改报文，增加**selectedProductIds**参数，并且下发  GET /bme/business.action?BMEBusiness=dependedProduct& **selectedProductIds=11111%27;};alert%281111%29;%3C/script%3E/\***&BMEWebToken=985a6485-0b59-4322-9626-e304f02cac9b&BMEWebToken=985a6485-0b59-4322-9626-e304f02cac9b HTTP/1.1  页面显示XSS注入成功 | | | |
|  | | | |
| **问题详细技术分析** | | | |
| **直接EL表达式调用了参数，没有对参数进行转码** | | | |
| **对业务的影响** | | | |
|  | | | |
| **解决方案** | | | |
|  | | | |

# 根因分析

## 不可信任的数据直接输出

生成的页面对不可信任的数据并没有进行编码而直接输出 ，可导致原页面的原有元素，属性，脚本的改变，通常表现为添加或者修改原页面元素、属性、脚本。

## 不可信任的数据编码不完整，或者使用不当

很多产品的确对数据进行编码，但对于防范XSS的数据编码针对页面的属性、元素、脚本编码规则是不同的，产品往往只是实现了部分字符的在某种场景的转化。例如html 元素编码 和html属性编码的字符集合是不一致的， html, js, css的编码方式是不一致的。

## 对外部输入数据校验不彻底

前端参数校验的特征匹配能一定的有效的防护XSS，但是这种校验不一定是完全有效的，完全会可能出现多种遗漏的场景。

1. 场景一：特征匹配不全面，比如使用过滤javascript

但是攻击者使用空格或者tab，回车换行符号对javascript字符进行注入，导致特征匹配失效

1. 场景二：对于多参数拼接进行输入展现的，特征匹配往往容易被轻松绕过

A参数校验不允许输入script , B参数校验也不允许输入script

攻击者A输入sc 而B输入ript

A+B=script 最后攻击成功

## 框架使用不当

通常框架会提供防止XSS的编码方式，但产品在不了解编码原理的情况下，误用框架提供的XSS的编码方式，导致使用不当

# 推荐解决方案

对于XSS 的注入的防护措施通常分为输入参数校验和输出编码，通常这两种方法配套使用，而输出编码是解决方案根本。

## 参数校验

数据校验是除了参数化查询外的另一种功能强大的防护措施。对于不可信数据进行严格的校验，可有效降低XSS注入漏洞的产生。数据校验可采用白名单校验和黑名单校验。白名单校验是“接受已知好的数据”，黑名单校验是“拒绝已知坏的数据”。

1. 白名单校验

有些参数本身就具有独特的数据特性，比如数字，邮件，名字 等

**建议：当数据具有明确特性，能用白名单覆盖的，使用白名单校验**

2. 黑名单校验

考虑到html的语言标签复杂性，XSS的参数校验也可以使用黑名单校验。

**建议：黑名单校验由于设置疏漏，容易被绕过，黑名单需要不停的更新，注意预留更新黑名单的方式。**

XSS常见的黑名单校验的正则表达式：

* + - 1. script

|  |
| --- |
| (?i)(<script[^>]\*>[\\s\\S]\*?<\\/script[^>]\*>|<script[^>]\*>[\\s\\S]\*?<\\/script[[\\s\\S]]\*[\\s\\S]|<script[^>]\*>[\\s\\S]\*?<\\/script[\\s]\*[\\s]|<script[^>]\*>[\\s\\S]\*?<\\/script|<script[^>]\*>[\\s\\S]\*?) |

* + - 1. on event

|  |
| --- |
| (?i)([\\s\"'`;\\/0-9\\=]+on\\w+\\s\*=) |

* + - 1. 其他

|  |
| --- |
| (?i)((?:=|U\\s\*R\\s\*L\\s\*\\()\\s\*[^>]\*\\s\*S\\s\*C\\s\*R\\s\*I\\s\*P\\s\*T\\s\*:|&colon;|[\\s\\S]allowscriptaccess[\\s\\S]|[\\s\\S]src[\\s\\S]|[\\s\\S]data:text\\/html[\\s\\S]|[\\s\\S]xlink:href[\\s\\S]|[\\s\\S]base64[\\s\\S]|[\\s\\S]xmlns[\\s\\S]|[\\s\\S]xhtml[\\s\\S]|[\\s\\S]style[\\s\\S]|<style[^>]\*>[\\s\\S]\*?|[\\s\\S]@import[\\s\\S]|<applet[^>]\*>[\\s\\S]\*?|<meta[^>]\*>[\\s\\S]\*?|<object[^>]\*>[\\s\\S]\*?) |

**注：上面的正则表达式，并不能穷举所有XSS的攻击风险，参数校验并不是有效XSS的防范方法，只能是防范的辅助手段。**

## 输出编码

防止跨站脚本(XSS)相关的关键输出编码机制

|  |  |
| --- | --- |
| 编码类型 | 编码机制 |
| HTML实体编码 | * + - 1. 字符 , . - \_ 空格 不需要转码       2. 字母（a-z, A-Z）数字(0-9)不需要转码       3. 转换 & 为 &amp;       4. 转换 < 为 &lt;       5. 转换 > 为 &gt;       6. 转换 " 为 &quot;       7. 其余的转化为&#xFFFF 16进制 或者&#0000 十进制表表述 |
| HTML属性编码 | 在HTML实体编码规范中，对1中的空格需要转码 |
| URL 编码 | 1. 空格转为 + 2. 字母（a-z, A-Z）数字(0-9)不需要转码 3. -，-，.，\* 字符 不需要转码 4. 使用单字节对原编码字符进行转换   转化格式使用%FF FF是16进制表达  例如：“中” UTF-8的编码是E4 B8 AD  URL编码后 %E4%B8%AD |
| JavaScript编码 | 1. 字母 数字 ，. \_ 字符不需要转码 2. 小于256 编码转义成\xFF FF为十六进制，不足2位需要补0至2位 3. 大于256 编码转义成 \uFFFF FFFF为十六进制，如不足4位不足位需补0至4位 |
| CSS 十六进制编码 | 1. 大于等于32 小于等于126字符中，除"、'、<、&、/、\ 、> 字符其余都不需要转码 2. 其他的转为格式为 “\FFFF” FFFF为十六进制，无位数显示要求 3. 转义后可以增加空格来分割下一个转义 |

**注：上面除（URL编码）所列均为unicode 编码，上述规则需产品实现需要转码成unicode，然后在进行转码**

## 选择正确的编码方式

HTML编码

|  |
| --- |
| <span>encoderforhtml(UNTRUSTED DATA)</span> |

HTML 属性编码

|  |
| --- |
| <input type="text" name="fname" value=" encoderforhtmlattribute(UNTRUSTED DATA)"> |

URL 编码

|  |
| --- |
| <a href="/site/search?value= encoderforURL(UNTRUSTED DATA)">clickme</a> |

JS 编码

|  |
| --- |
| <script>var currentValue=' encoderforjavascript (UNTRUSTED DATA)';</script> |

CSS 编码

|  |
| --- |
| <div style="width: encoderforcss( UNTRUSTED DATA);">Selection</div> |

混合多种编码，要注意html的展现顺序

|  |
| --- |
| <script>  element.innerHTML="encoderforjavascript (encoderforhtml (UNTRUSTED DATA)) "  </script> |

浏览器在展现使用JS修改DOM时候，先调用decode对JS, 然后在对展现的数据进行html decode， 所有在编码的时候需要先对html 编码，然后在对JS进行编码

## XSS安全头

CSP 是浏览器实现的对XSS防护策略，主要依赖于服务器端返回的的安全头 Content Security Policy

|  |
| --- |
| * **Content-Security-Policy** : W3C Spec standard header. Supported by Firefox 23+, Chrome 25+ and Opera 19+ * **Content-Security-Policy-Report-Only** : W3C Spec standard header. Supported by Firefox 23+, Chrome 25+ and Opera 19+, whereby the policy is non-blocking ("fail open") and a report is sent to the URL designated by the **report-uri** directive. This is often used as a precursor to utilizing CSP in blocking mode ("fail closed") * **DO NOT** use X-Content-Security-Policy or X-WebKit-CSP. Their implementations are obsolete (since Firefox 23, Chrome 25), limited, inconsistent, and incredibly buggy. |

### Content Security Policy 1.0/2 版本的参数

|  |
| --- |
| CSP 1.0  * **connect-src** (d) - restricts which URLs the protected resource can load using script interfaces. (e.g. send() method of an XMLHttpRequest object) * **font-src** (d) - restricts from where the protected resource can load fonts * **img-src** (d) - restricts from where the protected resource can load images * **media-src** (d) - restricts from where the protected resource can load video, audio, and associated text tracks * **object-src** (d) - restricts from where the protected resource can load plugins * **script-src** (d) - restricts which scripts the protected resource can execute. Additional restrictions against, inline scripts, and eval. Additional directives in CSP2 for hash and nonce support * **style-src** (d) - restricts which styles the user may applies to the protected resource. Additional restrictions against inline and eval. * **default-src** - Covers any directive with *(d)* * **frame-src** - restricts from where the protected resource can embed frames. Note, deprecated in CSP2 * **report-uri** - specifies a URL to which the user agent sends reports about policy violation * **sandbox** - specifies an HTML sandbox policy that the user agent applies to the protected resource. Optional in 1.0  New in CSP2  * **form-action** - retricts which URLs can be used as the action of HTML form elements * **frame-ancestors** - indicates whether the user agent should allow embedding the resource using a frame, iframe, object, embed or applet element, or equivalent functionality in non-HTML resources * **plugin-types** - restricts the set of plugins that can be invoked by the protected resource by limiting the types of resources that can be embedded * **base-uri** - restricts the URLs that can be used to specify the document base URL * **child-src** (d) - governs the creation of nested browsing contexts as well as Worker execution contexts |

例子

Content-Security-Policy: default-src https:; connect-src https:; font-src https: data:; frame-src https:;

img-src https: data:; media-src https:; object-src https:; script-src 'unsafe-inline' 'unsafe-eval' https:;

style-src 'unsafe-inline' https:;

具体请参考 （<https://www.owasp.org/index.php/Content_Security_Policy_Cheat_Sheet>）

**建议：安全头是浏览器实现的XSS防范，本身有对浏览器的类型和版本的要求，是作为XSS防护的辅助手段。**

# 排查范围及工具

**排查范围：**所有涉及Web应用的自研产品

**排查模块：**Web应用系统中所有涉及输出的模块。

**排查内容：**排查所有涉及输出HTML的页面

## XSS问题工具排查

工具只能作为辅助手段，任何工具都无法完全排查XSS的问题。

### 白盒工具检查：

1. 使用CodeDEX工具进行扫描，重点排查是否存在下列规则告警：

Cross-Site Scripting: DOM

Cross-Site Scripting: External Links

Cross-Site Scripting: Inter-Component Communication

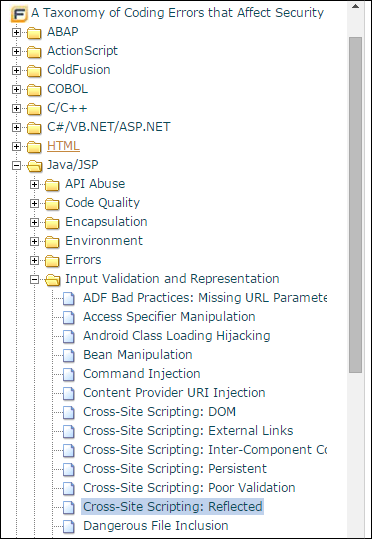
Cross-Site Scripting: Persistent

Cross-Site Scripting: Poor Validation

Cross-Site Scripting: Reflected

具体的告警说明和案例可以访问下面链接：

<http://www.hpenterprisesecurity.com/vulncat/zh_CN/vulncat/index.html>

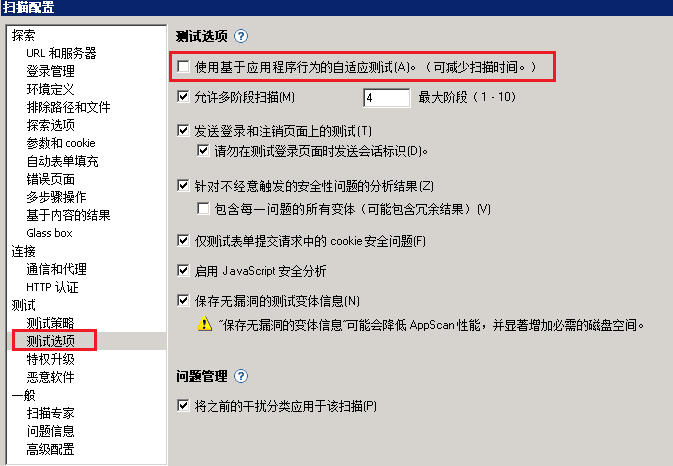


1. 使用BME平台的产品，必须要使用BME\_CI进行源码扫描，具体工具获取及使用方法详见：<http://3ms.huawei.com/hi/group/5255/blog_1744329.html?mapId=2503143&for_statistic_from=my_group_blog>

### 黑盒工具排查：

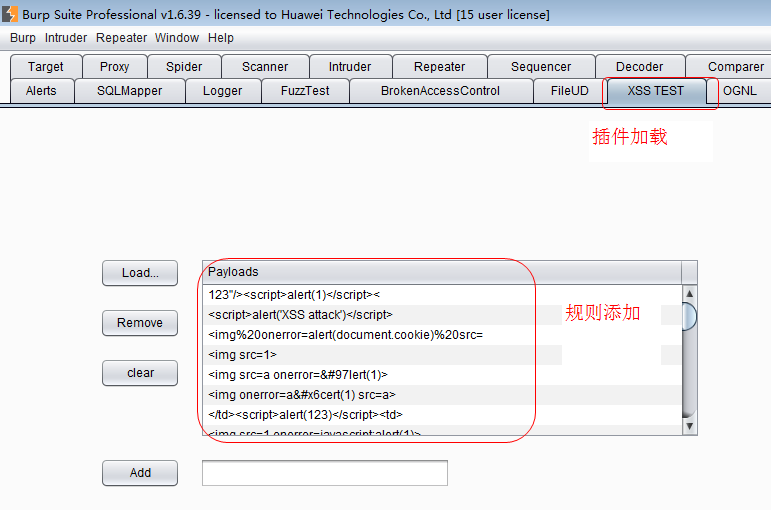
**APPscan使用注意点**

1. 使用AppScan时候建议补充人工页面爬取手段，保证测试页面的全面性；
2. 使用AppScan测试存储型XSS时，去勾选（默认勾选）“扫描配置”🡪“测试选项”🡪“使用基于应用程序行为的自适应测试”，可提高存储型XSS检出率。

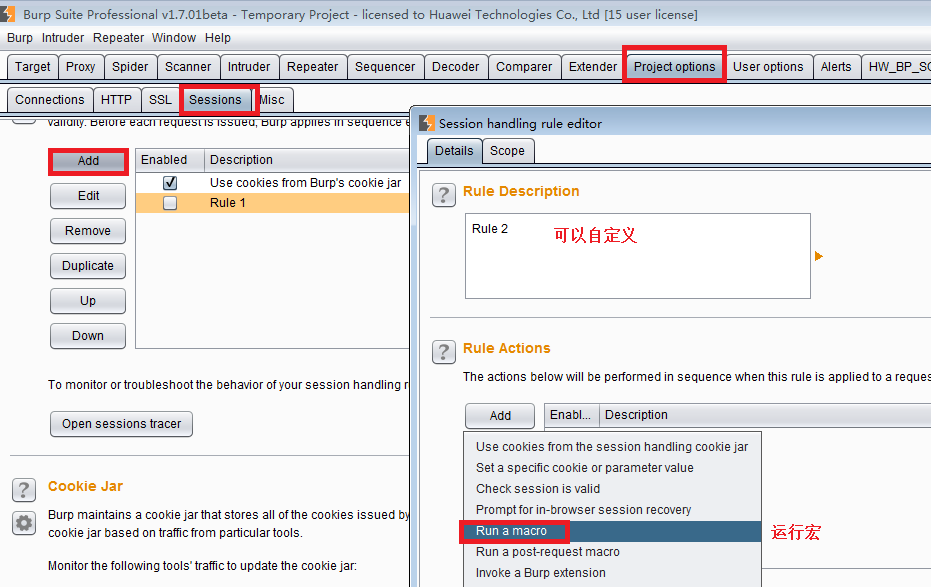


**Burp suite使用注意点**

1. 使用BurpSuite上的自研插件，实现对已知web安全问题的排查；



1. 使用时需要保证会话有效，建议对登录进行录制



具体录制方法请见：

http://3ms.huawei.com/hi/group/1007713/file\_9026071.html

# 排查指导

## 排查是否对输出进行编码

### JAVA 中使用BME框架

* 自定义标签对不可信数据输出时候是否对输出进行编码
* 无法使用标签，使用EL表达式动态输出不可信数据的时候是否采用BME的安全的EL编码表达式 ex. ${bme:htmlEncode(exceptionMessage)}
* 使用<%%>，或者HttpServletResponse动态输出的内容是否对不可信数据进行编码
* 输出编码方式上是否正确使用了不同的输出标签为html的元素，属性，脚本

Ex. BME提供了html，htmlattribute, css, js的编码方式，需要产品自检是否在正确的元素中使用了对应的编码方式

### JAVA 中使用传统的MVC框架

* 是否使用了框架本身自带的安全输出标签
* 自定义标签输出不可信数据时候是否进行编码
* EL表达式中，是否对不可信数据输出是否进行编码
* 输出编码方式是否按照不同的方式为html的元素，属性，脚本进行编码EX:提供了Html, Htmlattribute, Css , Js的编码

### C、C++ 的Web服务器（包括PHP）

* 是否对动态语言的输出，针对不可信数据使用了编码方式
* 输出编码方式是否按照不同的方式为html的元素，属性，脚本进行编码EX:提供了Html, Htmlattribute, Css , Js的编码
* PHP中使用htmlspecialchars 在DOM XSS注入是失效的

### Javascript

* 服务器端对ajax的调用的输出结果是否针对不同的html页面元素进行编码, 提供了Html, Htmlattribute, Css , Js的编码
* 客户端是否对ajax调用的不可信数据结果使用JS进行了不同类型的输出编码
* 常见的客户端框架JQuery, Angular等，需调用框架本身提供的输出安全编码函数，或者第三方提供的可信安全函数， 如果框架本身没有提供，需要排查是否对输出进行编码

## 排查是否使用成熟CBB编码

* 如果产品代码中使用了自研的输出编码的方式防止XSS注入，检查自研代码是否按照指导中的提供的编码方式进行编码。
* 采用CBB编码的

JAVA

1. 建议采用安全能力中心的提供的安全框架的安全API进行输出编码
2. 采用ESAPI安全API的

C

1. 安全能力中心提供的安全框架中的安全API

PHP

1. 使用HTMLPurifier

Javascript

1. 使用ESAPI4JS进行编码的

## 是否在输出CSP的安全头

* 产品在输出头中是否包含CSP的安全头

# 技术支持

漏洞排查过程中有技术问题及漏洞修复问题可咨询安全能力中心，联系人如下：

李科 l00316555