安全小课堂第134期【浅谈URL跳转漏洞的挖掘与防御】

京东安全应急响应中心 3月25日

随着业务发展，越来越多的Web应用需要与公司内部其他应用甚至外部第三方应用交互，将用户引导至不同功能页面。在不同功能页面跳转的过程中难免出现一种常见且利用门槛较低的漏洞——URL跳转漏洞，该漏洞像XSS一样有多种绕过方式，防御过程中也常出现意想不到的情况。

JSRC **安全小课堂第134期**，邀请到**ChangeS**师傅就**URL跳转漏洞的挖掘与防御**为大家进行分享。同时感谢白帽子们的精彩讨论。

URL跳转漏洞的危害和利用方式是什么?

京安小妹

ChangeS:

URL跳转漏洞本质上是利用Web应用中带有重定向功能的业务，将用户从一个网站重定向到另一个网站。其最简单的利用方式为诱导用户访问http://www.aaa.com?returnUrl=http://www.evil.com，借助www.aaa.com让用户访问www.evil.com，这种漏洞被利用了对用户和公司都是一种损失。对于安全意识较低的用户，当他们从聊天工具、论坛回帖等地方遇到上述链接时，根据链接的开头部分用户会认为打开之后是aaa公司的业务，尤其当www.aaa.com是耳熟能详的大型互联网公司时，用户的防备心理会更小，打开之后无论怎么跳转，用户都可能认为是值得信任的，当www.evil.com是钓鱼页面时，用户也可能毫不犹豫地输入用户名和密码登录网站。跳转链接再配合上“点击领取满100减99优惠券”、“点击领取免费会员”等诱惑性文字，恶意散播在论坛、聊天群中，难免会有用户上当受骗。若再配合编码、短网址等转换，即使有防范意识的用户也可能分不清链接中www.aaa.com之后的一大串参数是什么意思，总之认准aaa公司就对了。在一些即时聊天工具中，对于用户发送的网站链接会有安全性检测功能，对其附上“安全可访问”或者“危险请勿点击”的标记。借助可信域名www.aaa.com和其网站中的URL跳转漏洞，无论是恶意网站evil1还是evil2，都能以http://www.aaa.com?returnUrl=http://www.xxx.com的形式躲过聊天工具的检测算法，给用户造成安全可信的错觉，一旦用户访问之后发生财产损失，对于aaa公司和聊天软件公司都会造成名誉损害，降低公司产品在用户心中的可信度。若在跳转的过程中，通过代码向跳转目标网站拼接了请求参数和值，一旦跳转目标被修改成恶意网站，这些参数会被泄漏到恶意网站，进一步造成损失。比如正常跳转链接为http://www.aaa.com?returnUrl=http://order.aaa.com，后端代码会给其拼接上?token=LJK321JL321J3L,这样实现了跳转结束之后通过实时生成的token可查询订单,若该业务存在URL跳转漏洞，链接最终就被修改为http://www.aaa.com?returnUrl=http://www.evil.com?token=LJK321JL321J3L,实时随机生成的token信息就被泄漏到evil的服务器，若被进一步恶意利用，将会对公司业务造成更多损失。URL跳转漏洞在各家SRC中通常被定级为低危漏洞，但其利用方式及其简单，技术门槛低，常被黑产利用进行钓鱼、诈骗等目的，也可作为渗透测试的起点，通过跳转收集数据之后再进一步挖掘更深层的漏洞。

讲师

URL跳转漏洞常出现在哪些业务点?

京安小妹

ChangeS:

寻找URL跳转漏洞，只需要思考哪些功能需要进行跳转。登录功能一直是URL跳转漏洞的重灾区，用户访问网站某个业务，当涉及到账号角色权限的时候一定需要跳转到登陆界面，为了确保用户体验认证结束之后需要自动返回用户之前浏览的页面。这一去一回之间就产生了URL跳转漏洞的隐患。举个例子，比如用户正在访问http://www.aaa.com/detail?sku=123456的商品，添加购物车时触发登录操作，跳转到统一登陆认证页面进行登录，这时的访问链接为http://login.aaa.com?returnUrl=http://www.aaa.com/detail?sku=123456，认证成功之后浏览器继续返回商品详情页面方便用户进行购买操作。若login.aaa.com对returnUrl参数检查不严格甚至未检查，通过该链接可跳转至任意网站。

登录之后302跳转到百度首页，此处returnUrl未进行任何检查，可任意跳转到第三方页面。与登录功能同理，网站的退出功能同样存在URL跳转漏洞的风险。其他类似的跳转功能还有短信验证码认证之后跳转、分享或者收藏之后跳转、给第三方授权之后跳转，他们的共同特点都是从一个页面为了某种操作进入另一个页面，操作之后返回原页面继续浏览。有的多步操作业务中，点击下一步按钮时会传递fromUrl参数，该参数会成为返回上一步的超链接，如下所示：

甚至有的业务中callback参数也存在URL跳转漏洞。常见的参数值有return、redirect、url、jump、goto、target、link等，平时挖漏洞的过程中不妨关注下请求中是否含有比较完整的URL地址，对该类参数进行下测试。

讲师

URL跳转漏洞都有哪些挖掘姿势?

京安小妹

Changes:

先给大家看下最简单情况的开发代码，最简单的情况是没有任何检查，传入什么就跳转到什么，类似java代码如下：

```
String url = request.getParameter("returnUrl");
response.sendRedirect(url); ChangeS(28709166)15:57:54
```

不管传进来是什么，统统重定向，这也是挖掘难度最低的
再进一步会在代码中判断是否为目标域名，但开发小哥哥们喜欢用字符串包含来判断，类似java代码如下：

```
String url = request.getParameter("returnUrl");
```

```
if (url.indexOf("www.aaa.com") !=-1){
    response.sendRedirect(url);
}
```

对于上述的字符串检测操作，均可以用欺骗手法绕过判断。简单可用的payload如下所示：
http://www.aaa.com?returnUrl=http://www.aaa.com.evil.com
http://www.aaa.com?returnUrl=http://www.evil.com/www.aaa.com
http://www.aaa.com?returnUrl=http://www.xxxaaa.com
若再配合URL的各种特性符号，绕过姿势可是多种多样。比如
利用问号?：
http://www.aaa.com?returnUrl=http://www.evil.com?www.aaa.com
利用反斜线\：
http://www.aaa.com?returnUrl=http://www.evil.com\www.aaa.com
http://www.aaa.com?returnUrl=http://www.evil.com\\www.aaa.com
利用@符号：
http://www.aaa.com?returnUrl=http://www.aaa.com@www.evil.com
利用井号#：
http://www.aaa.com?returnUrl=http://www.evil.com#www.aaa.com
http://www.aaa.com?returnUrl=http://www.evil.com#www.aaa.com?www.aaa.com
缺失协议：
http://www.aaa.com?returnUrl=/www.evil.com
http://www.aaa.com?returnUrl=//www.evil.com
多次跳转,即aaa公司信任ccc公司，ccc公司同样存在漏洞或者提供跳转服务:
http://www.aaa.com?returnUrl=http://www.ccc.com?jumpto=http://www.evil.com
实际挖掘过程中还可以将上述方法混合使用，甚至使用URL编码、ip地址替代域名等手段。上述介绍的挖掘姿势以GET请求为例，但其在POST请求中也同样适用。

讲师

这类漏洞如何防御呢?

京安小妹

ChangeS:

防御该漏洞最有效的方法之一就是严格控制将要跳转的域名。如果某个业务事先已经确定将要跳转的网站，最稳妥的方式是将其直接编码在源代码中，通过URL中传入的参数来映射跳转网址。比如传入jumpto=1则跳转到order.aaa.com,传入jumpto=2则跳转到detail.aaa.com,传入预期之外的参数直接报错即可。但该方式可扩展性很差，随着业务不断发展，将会给开发工作添加额外的麻烦。通用性较好的做法是严格限制子域名，只能在公司内的业务之间跳转，在java中可使用下列方案：

```
String url =request.getParameter("returnUrl");
String host = "";
try {
   host = new URL(url).getHost();
} catch (MalformedURLException e) {
   e.printStackTrace();
}
if host.endsWith(".aaa.com"){
   //跳转
}else{
   //不跳转，报错
}
```

上述代码中主要校验了客户端传来的returnUrl参数值，使用java.net.URL包中的getHost()方法获取了将要跳转URL的host，判断host是否为目标域，上述代码中限制了必须跳转到xxx.aaa.com的域名，即只能是aaa公司的子域名，如果业务可以确定跳转范围，可以将判断条件限制的更加严格，写成host.endsWith(".order.aaa.com"),从而排除了跳转到不可信域名的可能。不同的编程语言中都会有类似的工具包，借助这些工具包来获取将要跳转的域名再进一步判断限制即可。也有表哥建议限制referer、添加token，这样可以避免恶意用户构造跳转链接到处散播，但我认为修复该漏洞最根本的方法还是上述的严格检查跳转域名。

讲师

URL跳转漏洞防御过程中都有哪些坑?

京安小妹

ChangeS:

刚才我们提到可以借助编程语言中的工具包来获取returnUrl的域名，对该域名进行严格限制，在java中可以使用java.net.URL包中的getHost()方法。但在实际使用过程中，这些工具包可能无法做到准确无误地取出域名，还需对代码进行反复测试验证。getHost()方法可以被反斜线绕过，即returnUrl=http://www.evil.com\www.aaa.com会被代码认为是将要跳转到.aaa.com，而实际在浏览器中反斜线被纠正为正斜线，跳转到www.evil.com/www.aaa.com，最终还是跳到www.evil.com的服务器。使用下列代码进行测试：

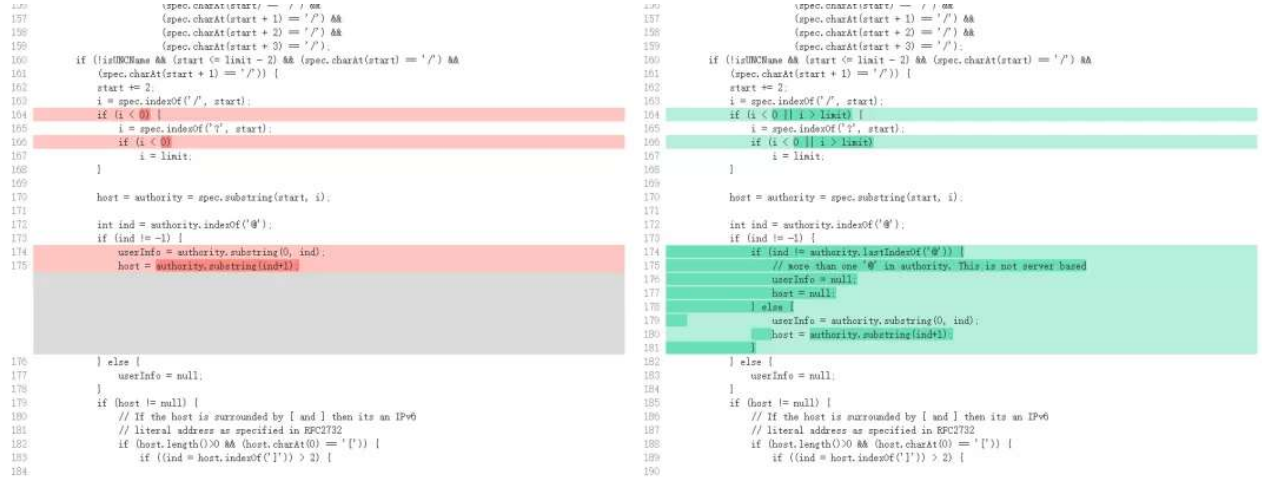
```
public class Main {

    public static void main(String[] args) {
        String url = "https://www.evil.com\\www.aaa.com?x=123";
        String host = "";
        try {
            host = new URL(url).getHost();
        } catch (MalformedURLException e) {
            e.printStackTrace();
        }
        System.out.println("host---"+host);
        System.out.println("url---"+url);
     }
}
```

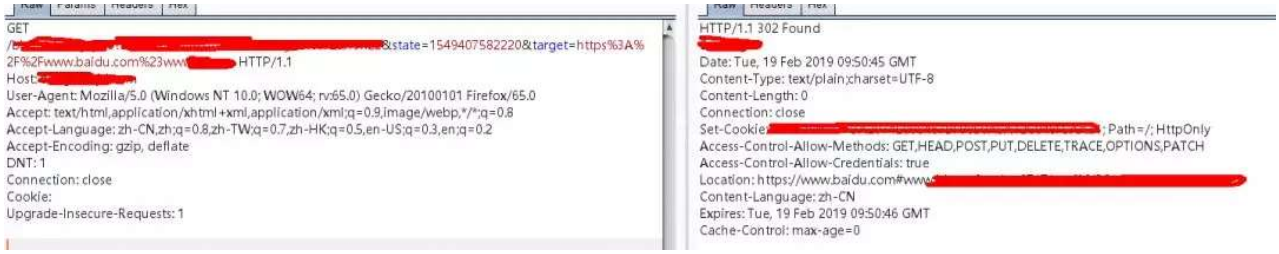
url参数的域名getHost()之后是www.aaa.com还是www.evil.com呢？打印结果如下：

```
"C:\Program Files\Java\jdk1.8.0_181\bin\java.exe"
host---www.evil.com\www.aaa.com
url---https://www.evil.com\www.aaa.com?x=123
```

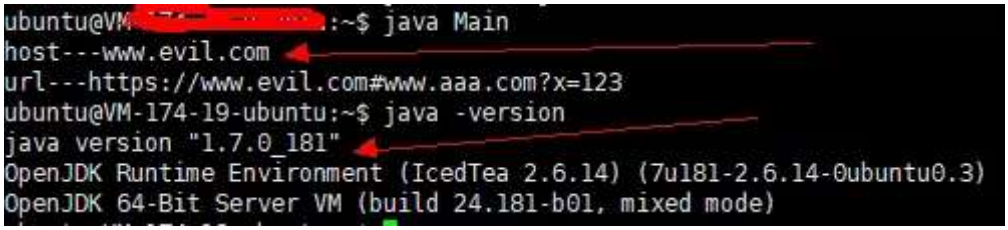
该结果会被endsWith(".aaa.com")方法判断为真，从而成功执行跳转，但实际在浏览器中www.evil.com\www.aaa.com被纠正为www.evil.com/www.aaa.com，成功跳转到了www.evil.com网站，该问题在目前所有JDK中都存在。这是个挖掘姿势哦，getHost方法还有另一个坑。getHost()方法的结果在不同JDK版本中对井号#的处理结果不同，通常井号被用作页面锚点，对于https://www.evil.com#www.aaa.com?x=123这个url，较高版本的JDK中，取出结果为www.evil.com，低版本中为www.evil.com#www.aaa.com，从而低版本又可绕过endsWith(".aaa.com")方法，成功跳转。这里所说的高版本指的是java version 1.8.0_181或者java version1.7.0_161中的181和161，与JDK7还是8无关。可能java在某个时间集中修复了JDK6/7/8中的URL库。测试过程中发现1.6.0_45，1.7.0_71，1.8.0_25均可被#绕过，即不同的JDK中低版本均存在问题。通过对比rt.jar---java---net--URLStreamHandler.java代码（低版本为左边，高版本为右边）找到问题所在如下图所示，代码中的start为url中冒号位置，limit为url中井号位置：



从代码中可以发现，低版本中未考虑到一个完整url中斜线/或者问号？之前会出现井号#的情况，如果url中有斜线/或者问号？，取host就以斜线或者问号为终止，即使中间包含井号也不处理；而高版本中进行了井号位置的判断，排除了使用井号绕过的可能。使用井号配合斜线或者问号来绕过域名检测，即将target设置为URL编码后的https://www.baidu.com#www.aaa.com?x=123，该站即可302跳转到百度。



再送一个真实例子，同一段代码在不同JDK版本中打印出的host值不同，在低版本中包含了井号及其后边的部分。



大家可以看出来，host的打印结果是不一样的。

综合上述两个坑，若想使用getHost()来修复任意URL跳转漏洞，需要考虑到反斜线和井号绕过，可使用如下代码：

```
String url = request.getParameter("returnUrl");
String host = "";
try {
    url = url.replaceAll("[\\\\#],"/"); //替换掉反斜线和井号
    host = new URL(url).getHost();
} catch (MalformedURLException e) {
    e.printStackTrace();
}
if host.endsWith(".aaa.com"){
    //跳转
}else{
    //不跳转，报错
}
```

可见，编程语言自带的原生包也并不一定可以放心使用，其他语言或许也存在类似

的问题，需要经过各方面测试验证才能找到较好的防御方案，攻与防是相辅相成的，只有在不断挖掘漏洞和防御漏洞的过程中踩坑才能有所收获。

讲师

本期 JSRC 安全小课堂到此结束。更多内容请期待下期安全小课堂。如果还有你希望出现在安全小课堂内容暂时未出现，也欢迎留言告诉我们。
安全小课堂的往期内容开通了自助查询，点击菜单栏进入“安全小课堂”即可浏览。

简历请发送：anquan@jd.com

微信公众号：jsrc_team

新浪官方微博：京东安全应急响应中心