# 你必须了解的Session的本质

Panni_007 2013-06-09 共580726人围观，发现 28 个不明物体 WEB安全

有一点我们必须承认，大多数web应用程序都离不开session的使用。这篇文章将会结合php以及http协议来分析
何建立一个安全的会话管理机制。我们先简单的了解一些http的知识，从而理解该协议的无状态特性。然后，学
一些关于cookie的基本操作。最后，我会一步步阐述如何使用一些简单，高效的方法来提高你的php应用程序的
全性以及稳定行。

我想大多数的php初级程序员一定会认为php默认的session机制的安全性似乎是有一定保障的，事实恰好相反
php团队只是提供了一套便捷的session的解决方案提供给程序员使用，至于安全性的话，应该由程序员来加强
这是应用程序开发团队的责任。因为，这里面的方法很多，可以这么说吧，没有最好，只有更好。攻击的方式在
断变化，防守方也需要不断变招，所以，我个人认为php团队的做法还是比较明智的。

## 无状态性

Http是一种无状态性的协议。这是因为此种协议不要求浏览器在每次请求中标明它自己的身份，并且浏览器以及
务器之间并没有保持一个持久性的连接用于多个页面之间的访问。当一个用户访问一个站点的时候，用户的浏览
发送一个http请求到服务器，服务器返回给浏览器一个http响应。其实很简单的一个概念，客户端一个请求，服
器端一个回复，这就是整个基于http协议的通讯过程。

因为web应用程序是基于http协议进行通讯的，而我们已经讲过了http是无状态的，这就增加了维护web应用程
状态的难度，对于开发者来说，是一个不小的挑战。Cookies是作为http的一个扩展诞生的，其主要用途是弥补h
的无状态特性，提供了一种保持客户端与服务器端之间状态的途径，但是由于出于安全性的考虑，有的用户在浏
器中是禁止掉cookie的。这种情况下，状态信息只能通过url中的参数来传递到服务器端，不过这种方式的安全
很差。事实上，按照通常的想法，应该有客户端来表明自己的身份，从而和服务器之间维持一种状态，但是出于
全性方面的考虑，我们都应该明白一点 — 来自客户端的信息都是不能完全信任的。

尽管这样，针对维持web应用程序状态的问题，相对来说，还是有比较优雅的解决方案的。不过，应该说是没有
美的解决方案的，再好的解决方案也不可能适用所有的情况。这篇文章将介绍一些技术。这些技术可以用来比较
定地维持应用程序的状态以及抵御一些针对session的攻击，比如会话劫持。并且你可以学习到cookie是怎样工
的，php 的session做了那些事情，以及怎样才能劫持session。

## HTTP 概览

如何才能保持web应用程序的状态以及选择最合适的解决方案呢？在回答这个问题之前，必须得先了解web的底
协议 — Hypertext Transfer Protocol (HTTP)。

当用户访问http://example.com这个域名的时候，浏览器就会自动和服务器建立tcp/ip连接，然后发送http请求example.com的服务器的80端口。该个请求的语法如下所示：

```
GET / HTTP/1.1
Host: example.org
```

以上第一行叫做请求行，第二个参数（一个反斜线在这个例子中）表示所请求资源的路径。反斜线代表了根目录服务器会转换这个根目录为服务器文件系统中的一个具体目录。

Apache的用户常用DocumentRoot这个命令来设置这个文档根路径。如果请求的url是http://example.org/path/to/script.php,那么请求的路径就是/path/to/script.php。假如document root 被定为usr/lcoal/apache/htdocs的话,整个请求的资源路径就是/usr/local/apache/htdocs/path/to/script.php。

第二行描述的是http头部的语法。在这个例子中的头部是Host, 它标识了浏览器希望获取资源的域名主机。还有多其它的请求头部可以包含在http请求中，比如user-Agent头部，在php可以通过$_SERVER['HTTP_USER_AGENT']获取请求中所携带的这个头部信息。

但是遗憾的是，在这个请求例子中，没有任何信息可以唯一标识当前这个发出请求的客户端。有些开发者借助请中的ip头部来唯一标识发出此次请求的客户端，但是这种方式存在很多问题。因为，有些用户是通过代理来访问的，比如用户A通过代理B连接网站www.example.com, 服务器端获取的ip信息是代理B分配给A的ip地址，如果户这时断开代理，然后再次连接代理的话，它的代理ip地址又再次改变，也就说一个用户对应了多个ip地址，这情况下，服务器端根据ip地址来标识用户的话，会认为请求是来自不同的用户，事实上是同一个用户。 还用另一种情况就是，比如很多用户是在同一个局域网里通过路由连接互联网，然后都访问www.example.com的话，于这些用户共享同一个外网ip地址，这会导致服务器认为这些用户是同一个用户发出的请求，因为他们是来自同个ip地址的访问。

保持应用程序状态的第一步就是要知道如何来唯一地标识每个客户端。因为只有在http中请求中携带的信息才能来标识客户端，所以在请求中必须包含某种可以用来标识客户端唯一身份的信息。Cookie设计出来就是用来解这一问题的。

## Cookies

如果你把Cookies看成为http协议的一个扩展的话，理解起来就容易的多了，其实本质上cookies就是http的一个展。有两个http头部是专门负责设置以及发送cookie的,它们分别是Set-Cookie以及Cookie。当服务器返回给客端一个http响应信息时，其中如果包含Set-Cookie这个头部时，意思就是指示客户端建立一个cookie，并且在续的http请求中自动发送这个cookie到服务器端，直到这个cookie过期。如果cookie的生存时间是整个会话期间话，那么浏览器会将cookie保存在内存中，浏览器关闭时就会自动清除这个cookie。另外一种情况就是保存在客户端的硬盘中，浏览器关闭的话，该cookie也不会被清除，下次打开浏览器访问对应网站时，这个cookie就会动再次发送到服务器端。一个cookie的设置以及发送过程分为以下四步：

客户端发送一个http请求到服务器端
服务器端发送一个http响应到客户端，其中包含Set-Cookie头部
客户端发送一个http请求到服务器端，其中包含Cookie头部
服务器端发送一个http响应到客户端

这个通讯过程也可以用以下下示意图来描述：

在客户端的第二次请求中包含的Cookie头部中，提供给了服务器端可以用来唯一标识客户端身份的信息。这时服务器端也就可以判断客户端是否启用了cookies。尽管，用户可能在和应用程序交互的过程中突然禁用cookie的使用，但是，这个情况基本是不太可能发生的，所以可以不加以考虑，这在实践中也被证明是对的。

### GET and POST Data

除了cookies,客户端还可以将发送给服务器的数据包含在请求的url中，比如请求的参数或者请求的路径中。 我来看一个例子：

```
GET /index.php?foo=bar HTTP/1.1
Host: example.org
```

以上就是一个常规的http get 请求，该get请求发送到example.org域名对应的web 服务器下的index.php脚本，index.php脚本中，可以通过$_GET['foo']来获取对应的url中foo参数的值，也就是'bar'。大多数php开发者都和样的数据会GET数据，也有少数称它为查询数据或者url变量。但是大家需要注意一点，不是说GET数据就只能含在HTTP GET类型的请求中，在HTTP POST类型的请求中同样可以包含GET数据，只要将相关GET数据包含请求的url中即可，也就是说GET数据的传递不依赖与具体请求的类型。
另外一种客户端传递数据到服务器端的方式是将数据包含在http请求的内容区域内。 这种方式需要请求的类型POST的，看下面一个例子：

```
POST /index.php HTTP/1.1
Host: example.org
Content-Type: application/x-www-form-urlencoded
Content-Length: 7

foo=bar
```

在这种情况下，在脚本index.php可以通过调用$_POST['foo']来获取对应的值bar。开发者称这个数据为POST数据,也就是大家熟知的form以post方式提交请求的方式。

在一个请求中，可以同时包含这两种形式的数据：

```
POST /index.php?myget=foo HTTP/1.1
Host: example.orgContent-Type: application/x-www-form-urlencoded
Content-Length: 11

mypost=bar
```

这两种传递数据的方式，比起用cookies来传递数据更稳定，因为cookie可能被禁用，但是以GET以及POST方式传递数据时，在这种情况。我们可以将PHPSESSID包含在http请求的url中，就像下面的例子一样：

```
GET /index.php?PHPSESSID=12345 HTTP/1.1
Host: example.org
```

以这种方式传递session id的话，可以跟用cookie头部传递s
要开发者认为地将session id附加在url中或者作为隐藏字段

户端创建cookie成功以后，客户端在后续的请求中，会自动第将对应的没有过期的cookie传递给服务器端。当然，php在开启session.use_trans_sid后，也可以自动地将session id 附加在url中以及表单的隐藏字段中，但是个选项不建议开启，因为存在安全问题。这样的话，容易泄露session id, 比如有的用户会bookmark一个url或者享一个url，那么session id也就暴露了，加入这个session id还没有过期，那是有一定的安全问题存在的，除非务器端，除了session id外，还附加了其它方式进行验证用户的合法性！

尽管以POST的方式来传递session id的话，相对GET的方式来说，会安全的多。但是，这种方式的缺点就是比麻烦，因为这样的话，在你的应用程序中比较将所有的请求都转换成post的请求，这显然是不太合适的。

### Session的管理

直到现在，我只讨论了如何维护应用程序的状态，只是简单地涉及到了如果保持请求之间的关系。接下来，我下在实际中用到比较多的技术 — Session的管理。涉及到session的管理，就不是单单地维持各个请求之间的状态，还需要维持会话期间针对每个特定用户使用到的数据。我们常常把这种数据叫做session数据，因为这些数是跟某个特定用户与服务器之间的会话相关联的。如果你使用php内置的session的管理机制，那么session数据般是保存在/tmp这个服务器端的文件夹中，并且其中的session数据会被自动地保存到超级数组$_SESSION中一个最简单的使用session的例子，就是将相关的session数据从一个页面传递(注意：实际传递的是session id)另一个页面。下面用示例代码1, start.php, 对这个例子加以演示：

```
<?php
session_start();
$_SESSION['foo'] = 'bar';
?>
<a href="continue.php">continue.php</a>
```

假如用户点击start.php中的链接访问continue.php,那么在continue.php中就可以通过$_SESSION['foo']获取在start.php中的定义的值'bar'。看下面的示例代码2:

**示例代码2 — continue.php**

```
<?php
session_start();
echo $_SESSION['foo']; /* bar */
?>
```

是不是非常简单，但是我要指出的话，如果你真的这样来写代码的话，说明你对php底层的对于session的实现制还不是非常了解透彻。在不了解php内部给你自动做了多少事情的情况下，你会发现如果程序出错的话，这样代码将变的很难调试，事实上，这样的代码也完全没有安全性可言。

### Session的安全性问题

一直以来很多开发者都认为php内置的session管理机制是具有一定的安全性，可以对一般的session攻击起到防御。事实上，这是一种误解，php团队只实现了一种方便有效的机制。具体的安全措施，应该有应用程序的开发队来实施。 就像开篇谈到的，没有最好的解决方案，只有最合适你的方案。

现在，我们来看下一个比较常规的针对session的攻击：

> 用户访问http://www.example.org，并且登录。
> example.org的服务器设置指示客户端设置相关cookie — PHPSESSID=12345
> 攻击者这时访问http://www.example.org/,并且在请求中携带了对应的cookie — PHPSESSID=12345
> 这样情况下，因为example.orge的服务器通过PHPSESSID来辨认对应的用户的，所以服务器错把攻击者成了合法的用户。

整个过程的描述，请看下面的示例图：

当然这种攻击的方式，前提条件是攻击者必须通过某种手段固定，劫持或者猜测出某个合法用户的PHPSESSID虽然这看起来难度很高，但是也不是不可能的事情。

### 安全性的加强

有很多技术可以用来加强Session的安全性，主要思想就是要使验证的过程对于合法用户来说，越简单越好，然对于攻击者来说，步骤要越复杂越好。当然，这似乎是比较难于平衡的，要根据你应用程序的具体设计来做决策

最简单的居于HTTP/1.1请求包括请求行以及一些Host的头部：

```
GET / HTTP/1.1
Host: example.org
```

如果客户端通过PHPSESSID传递相关的session标识符，可以将PHPSESSID放在cookie头部中进行传递：

```
GET / HTTP/1.1
Host: example.org
Cookie: PHPSESSID=12345
```

同样地，客户端也可以将session标识符放在请求的url中进行传递。

```
GET /?PHPSESSID=12345
HTTP/1.1Host: example.org
```

当然，session标识符也可以包含在POST数据中，但是这对用户体验有影响，所以这种方式很少采用。

因为来自TCP/IP信息也不一定可以完全信任的，所以，对于web开发者来说，利用TCP/IP中的信息来加强安全也是不太合适的。 不过，攻击者也必须提供一个合法用户的唯一的标识符，才能假扮成合法用户进入系统。因此，看起来唯一能够有效的保护系统的措施，就是尽量地隐藏session标识符或者使之难于猜测出来。最好就是者都能实施。

PHP会自动生成一个随机的session ID，基本来说是不可能被猜测出来的，所以这方面的安全还是有一定保障的但是，要防止攻击者获取一个合法的session ID是相当困难的，这基本上不是开发者所能控制的。

事实上，许多情况下都有可能导致session ID的泄露。 比如说，如果通过GET数据来传递session ID的话，就有能暴露这个敏感的身份信息。因为，有的用户可能会将带有session ID的链接缓存，收藏或者发送在邮件内容中Cookies是一种像相对来说安全一点的机制，但是用户是可以在客户端中禁止掉cookies的！在一些IE的版本中有比较严重的安全漏洞，比较有名的就是会泄露cookies给一些有安全隐患的邪恶站点。

因此，作为一个开发者，可以肯定session ID是不能被猜测出来的，但是还是有可能被攻击者使用某些方法获取到。所以，必须采取一些额外的安全措施来防止此类情况在你的应用程序中发生。

实际上，一个标准的HTTP请求中除了Host等必须包含的头部，还包含了一些可选的头部.举一个例子，看下面的个请求：

```
GET / HTTP/1.1
Host: example.org
Cookie: PHPSESSID=12345
User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X; en-US; rv:1.8.1.1) Gecko/
0061204 Firefox/2.0.0.1
Accept: text/html;q=0.9, */*;q=0.1
Accept-Charset: ISO-8859-1, utf-8;q=0.66, *;q=0.66
Accept-Language: en
```

我们可以看到，在以上的一个请求例子中包含了四个额外的头部，分别是User-Agent, Accept, Accept-Charset以及Accept-Language。因为这些头部不是必须的，所以完全依赖他们在你的应用程序中发挥作用是不太明智的。但是，如果一个用户的浏览器确实发送了这些头部到服务器，那么可以肯定的是在接下来的同一个用户通过同一个浏览器发送的请求中，必然也会携带这些头部。当然，这其中也会有极少数的特殊情况发生。假如以上例子是由一个当前的跟服务器建立了会话的用户发出的请求，考虑下面的一个请求：

```
GET / HTTP/1.1
Host: example.org
Cookie: PHPSESSID=12345
User-Agent: Mozilla/5.0
```

因为有相同的session id包含在请求的Cookie头部中，所以相同的php session将会被访问到。但是，请求里的User-Agent头部跟先前的请求中的信息是不同的，系统是否可以假定这两个请求是同一个用户发出的?

像这种情况下，发现浏览器的头部改变了，但是不能肯定这是否是一次来自攻击者的请求的话，比较好的措施就弹出一个要求输入密码的输入框让用户输入，这样的话，对用户体验的影响不会很大，又能很有效地防止攻击。

当然，你可以在系统中加入核查User-Agent头部的代码，类似示例3中的代码：

**示例代码3**

```
<?php

session_start();
```

```
if (md5($_SERVER['HTTP_USER_AGENT']) != $_SESSION['HTTP_USER_AGENT'])
{  /* 弹出密码输入框 */  exit;
}

?>
```

当然，你先必须在第一次请求时，初始化session的时候，用MD5算法加密user agent信息并且保存在session中
类似下面示例4中的代码：

**示例代码4**

```
<?php

session_start();

$_SESSION['HTTP_USER_AGENT'] = md5($_SERVER['HTTP_USER_AGENT']);

?>
```

虽然不一定需要用MD5来加密这个User-Agent信息，但使用这种方式以后就不需要再过滤这个$_SERVER['HTTP_USER_AGENT']数据了。不然的话，在使用这个数据以前必须要进行数据过滤，因为任何来客户端的数据都是不可信任的，必须要注意这一点。

在你检查这个User-Agent客户端头部信息以后，做为一个攻击者必须要完成两步才能劫持一个session:

获取一个合法的session id
包含一个相同的User-Agent头部在伪造的请求中

你可能会说，居然攻击者能获得有效的session id,那么以他的水平，伪造一个相同的User-Agent不是件难事。错，但是我们可以说这至少给他添加了一些麻烦，在一定程度上也增加了session机制的安全性。

你应该也能想到了，既然我们可以检查User-Agent这个头部来加强安全性，那么不妨再利用其它的一些头部信息，把他们组合起来生成一个加密的token，并且让客户端在后续的请求中携带这个token！这样的话，攻击者本上不可能猜测出这样一个token是怎么生成出来的。这好比你用信用卡在超市付款，一个你必须有信用卡（好session id），另外你也必须输入一个支付密码（好比token）,这有这两者都符合的情况下，你才能成功进入账付款。 看下面一段代码：

```
<?php
session_start();
$token = 'SHIFLETT' . $_SERVER['HTTP_USER_AGENT'];
$_SESSION['token'] = md5($token . session_id());
?>
```

注意：Accept这个头部不应该被用来生成token,因为有些浏览器会自动改变这个头部，当用户刷新浏览器的时候。

在你的验证机制中加入了这个非常难于猜测出来的token以后session id一样的方式来进行传递，这种情况下，一个攻击者

获取一个合法的session ID
在请求中加入相同的User-Agent头部,用与生成token
在请求中携带被攻击者的token

这里面有个问题。如果session id以及token都是通过GET数据来传递的话，那么对于能获取session ID的攻击者同样就能够获取到这个token。所以，比较安全靠谱的方式应该是利用两种不同的数据传递方式来分别传递session id以及token。例如，通过cookie来传递session id,然后通过GET数据来传递token。因此，假如攻击者通过某种手段获得了这个唯一的用户身份标识，也是不太可能同时轻松地获取到这个token，它相对来说依然是安全的。
还有很多的技术手段可以用来加强你的session机制的安全性。希望你在大致了解session的内部本质以后，可以设计出适合你的应用系统的验证机制，从而大大的提高系统的安全性。毕竟，你是最熟悉当下你开发的系统的开发之一，可以根据实际情况来实施一些特有的，额外的安全措施。

**总结**

以上只是大概地描述了session的工作机制，以及简单地阐述了一些安全措施。但要记住，以上的方法都是能够加强安全性，不是说能够完全保护你的系统，希望读者自己再去调研相关内容。在这个调研过程中，相信你会学到有实际使用价值的方案。

```
Author : 360weboy
Site:http://www.360weboy.com/
MicroBlog:http://weibo.com/360weboy
```

上一篇：XSS解决方案系列之三: 例解过后，再回首您正在维护的产品
下一篇：XSS解决方案系列之四：关于编码

## 这些评论亮了



jiayzhan （5级） 硅谷某知名IT企业中国区高级应用安全工程师 回复

1. 关于Session固定的问题，通常不是PHP的机制，而是Web Server的机制问题，解决问题的通用做法是在PHP代码当中：renew sessionID after login web应用系统
2. PHP的安全问题我觉得类似变量初努化，register_global选项。。。挺严重
3. 如果是获取了用户的SessionID，那就是会话劫持，与Session固定不是一个问题

Session固定的过程：

1. send 一个请求like: http://a.b.c/a.php;PHPSESSIONID=5s4d6fs46dfas65d4, 无需带上一个存在的SessionID，只需要一个格式合法的SessionID
2. 诱导用户点击，用户发送以上的URL请求后，服务器发现客户端送过来一个SessionID，格式合法，就直接使用它建立会话
3. 此时如果此用户登录系统，这里攻击者因为早已拥有此SessionID而可以直接以被攻击者身份冒充被攻击者使用
4. 业界通用的解决方法上面说了，登录后:renew SessionID，就可以解决问题，有空我专门写文阐述一下。
5. 这是单一验证机制的弊端，我们不应该只使用会话ID来作为用户的身份标识符，应该辅以自有的使用通用算法生成的身份标识符来标识已经登录用户身份

亮了(24



郭添森 回复

最讨厌不标明出处的：http://shiflett.org/articles/the-truth-about-sessions

亮了(20



9527 回复

楼主目测是公务猿------讲了半天都是废话，没有任何有用的东西