这篇文章将介绍一些技术，这些技术可以用来比较稳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。并且你可以看到cookie是怎样工作的，php 的session做了那些事情，以及怎样才能劫持session。

一、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请求到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 被定义为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。还有很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。

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。有些开发者借助请求中的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，如果用户这时断开代理，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，这种情况下，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。 还用另外一种情况就是，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，由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，因为他们是来自同一个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用来标识客户端，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决这一问题的。

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的一个扩展。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。当服务器返回给客户端一个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，并且在后续的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的话，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就会自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

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cookies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。 我们来看一个例子：

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, 在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开发者都称这样的数据会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数据就只能包含在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数据包含在请求的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是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，不存在这种情况。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

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较麻烦，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，我阐述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，因为这些数据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数据一般是保存在/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)到另一个页面。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

?>

让我们用几个例子来描述一下cookie和session机制之间的区别与联系。笔者曾经常去的一家咖啡店有喝5杯咖啡免费赠一杯咖啡的优惠，然而一次性消费5杯咖啡的机会微乎其微，这时就需要某种方式来纪录某位顾客的消费数量。想象一下其实也无外乎下面的几种方案：

1、该店的店员很厉害，能记住每位顾客的消费数量，只要顾客一走进咖啡店，店员就知道该怎么对待了。这种做法就是协议本身支持状态。

2、发给顾客一张卡片，上面记录着消费的数量，一般还有个有效期限。每次消费时，如果顾客出示这张卡片，则此次消费就会与以前或以后的消费相联系起来。这种做法就是在客户端保持状态。

3、发给顾客一张会员卡，除了卡号之外什么信息也不纪录，每次消费时，如果顾客出示该卡片，则店员在店里的纪录本上找到这个卡号对应的纪录添加一些消费信息。这种做法就是在服务器端保持状态。

由于HTTP协议是无状态的，而出于种种考虑也不希望使之成为有状态的，因此，后面两种方案就成为现实的选择。具体来说cookie机制采用的是在客户端保持状态的方案，而session机制采用的是在服务器端保持状态的方案。同时我们也看到，由于采用服务器端保持状态的方案在客户端也需要保存一个标识，所以session机制可能需要借助于cookie机制来达到保存标识的目的，但实际上它还有其他选择。

二、理解cookie机制

cookie机制的基本原理就如上面的例子一样简单，但是还有几个问题需要解决：“会员卡”如何分发；“会员卡”的内容；以及客户如何使用“会员卡”。

正统的cookie分发是通过扩展HTTP协议来实现的，服务器通过在HTTP的响应头中加上一行特殊的指示以提示浏览器按照指示生成相应的cookie。然而纯粹的客户端脚本如JavaScript或者VBScript也可以生成cookie。

而cookie的使用是由浏览器按照一定的原则在后台自动发送给服务器的。浏览器检查所有存储的cookie，如果某个cookie所声明的作用范围大于等于将要请求的资源所在的位置，则把该cookie附在请求资源的HTTP请求头上发送给服务器。意思是麦当劳的会员卡只能在麦当劳的店里出示，如果某家分店还发行了自己的会员卡，那么进这家店的时候除了要出示麦当劳的会员卡，还要出示这家店的会员卡。

cookie的内容主要包括：名字，值，过期时间，路径和域。

其中域可以指定某一个域比如.google.com，相当于总店招牌，比如宝洁公司，也可以指定一个域下的具体某台机器比如www.google.com或者froogle.google.com，可以用飘柔来做比。

路径就是跟在域名后面的URL路径，比如/或者/foo等等，可以用某飘柔专柜做比。路径与域合在一起就构成了cookie的作用范围。如果不设置过期时间，则表示这个cookie的生命期为浏览器会话期间，只要关闭浏览器窗口，cookie就消失了。这种生命期为浏览器会话期的cookie被称为会话cookie。会话cookie一般不存储在硬盘上而是保存在内存里，当然这种行为并不是规范规定的。如果设置了过期时间，浏览器就会把cookie保存到硬盘上，关闭后再次打开浏览器，这些cookie仍然有效直到超过设定的过期时间。

存储在硬盘上的cookie可以在不同的浏览器进程间共享，比如两个IE窗口。而对于保存在内存里的cookie，不同的浏览器有不同的处理方式。对于IE，在一个打开的窗口上按Ctrl-N（或者从文件菜单）打开的窗口可以与原窗口共享，而使用其他方式新开的IE进程则不能共享已经打开的窗口的内存cookie；对于Mozilla Firefox0.8，所有的进程和标签页都可以共享同样的cookie。一般来说是用javascript的window.open打开的窗口会与原窗口共享内存cookie。浏览器对于会话cookie的这种只认cookie不认人的处理方式经常给采用session机制的web应用程序开发者造成很大的困扰。

下面就是一个goolge设置cookie的响应头的例子 ：

HTTP/1.1 302 Found

Location: http://www.google.com/intl/zh-CN/

Set-Cookie: PREF=ID=0565f77e132de138:NW=1:TM=1098082649:LM=1098082649:S=KaeaCFPo49RiA\_d8;

expires=Sun, 17-Jan-2038 19:14:07 GMT; path=/; domain=.google.com

Content-Type: text/html

这是使用HTTPLook这个HTTP Sniffer软件来俘获的HTTP通讯纪录的一部分。

浏览器在再次访问goolge的资源时自动向外发送cookie。

使用Firefox可以很容易的观察现有的cookie的值，使用HTTPLook配合Firefox可以很容易的理解cookie的工作原理。

三、理解session机制

session机制是一种服务器端的机制，服务器使用一种类似于散列表的结构（也可能就是使用散列表）来保存信息。

当程序需要为某个客户端的请求创建一个session的时候，服务器首先检查这个客户端的请求里是否已包含了一个session标识 - 称为session id，如果已包含一个session id则说明以前已经为此客户端创建过session，服务器就按照session id把这个session检索出来使用（如果检索不到，可能会新建一个），如果客户端请求不包含session id，则为此客户端创建一个session并且生成一个与此session相关联的session id，session id的值应该是一个既不会重复，又不容易被找到规律以仿造的字符串，这个session id将被在本次响应中返回给客户端保存。保存这个session id的方式可以采用cookie，这样在交互过程中浏览器可以自动的按照规则把这个标识发送给服务器。一般这个cookie的名字都是类似于SEEESIONID，而。比如weblogic对于web应用程序生成的cookie，JSESSIONID=ByOK3vjFD75aPnrF7C2HmdnV6QZcEbzWoWiBYEnLerjQ99zWpBng!-145788764，它的名字就是JSESSIONID。

由于cookie可以被人为的禁止，必须有其他机制以便在cookie被禁止时仍然能够把session id传递回服务器。经常被使用的一种技术叫做URL重写，就是把session id直接附加在URL路径的后面，附加方式也有两种，一种是作为URL路径的附加信息，表现形式为：http://...../xxx;jsessionid=ByOK3vjFD75aPnrF7C2HmdnV6QZcEbzWoWiBYEnLerjQ99zWpBng!-145788764

另一种是作为查询字符串附加在URL后面，表现形式为：<http://...../xxx?jsessionid=ByOK3vjFD75aPnrF7C2HmdnV6QZcEbzWoWiBYEnLerjQ99zWpBng!-145788764>

Session和cookie的区别：

session: 储存用户访问的全局唯一变量,存储在服务器上的php指定的目录中的（session\_dir）的位置进行的存放；

cookie: 用来存储连续訪問一个頁面时所使用，是存储在客户端，对于Cookie来说是存储在用户WIN的Temp目录中的；

两者都可通过时间来设置时间长短；

cookie是保存在客户端机器的，对于未设置过期时间的cookie，cookie值会保存在机器的内存中，只要关闭浏览器则cookie自动消失。如果设置了cookie的过期时间，那么浏览器会把cookie以文本文件的形式保存到硬盘中，当再次打开浏览器时cookie值依然有效。

session是把用户需要存储的信息保存在服务器端。每个用户的session信息就像是键值对一样存储在服务器端，其中的键就是sessionid，而值就是用户需要存储信息。服务器就是通过sessionid来区分存储的session信息是哪个用户的。

两者最大的区别就是session存储在服务器端，而cookie是在客户端。session安全性更高，而cookie安全性弱。