2017.5.6，5.8~10总结

原创

foolisheddy 于 2017-05-17 00:07:29 发布 606 收藏

分类专栏： 每日F-Q 实习日记 渗透测试

本文链接：https://blog.csdn.net/qq_28921653/article/details/72355096


每日F-Q 同时被 3 个专栏收录

17 篇文章 0 订阅

订阅专栏


实习日记

6 篇文章 0 订阅

订阅专栏


渗透测试

5 篇文章 0 订阅

订阅专栏

2017-5-6

xss

定位、查看源码，

```
<input type="text" name="search" search_words="姜清新" value="&lt;script&gt;alert(/xss/)&lt;/script&gt;" class="search_inp" placeholder="姜清新" />
```

过滤了尖括号，也就是说用不了标签

使用双引号进行闭合，这种情况是在标签内，用事件进行弹框

```
<input type="text" name="search" search_words="姜清新" value="" onmouseover="javascript:alert('XSS') "" class="search_inp" placeholder="姜清新" />
```

ctf

1.信息泄露第一步

106.75.67.7:3080

git泄露，把文件拖下来打开就行

2.brain fuck语言

解密

static2.ichunqiu.com/icq/resources/fileupload/CTF/gz-chusai/Crypto04.txt?pass=db0115aed0c26586

```
+++++ +++++ [->++ +++++ +++<] >++.+ +++++ .<+++ [->-- -<]>- -.+++ +++.<
++++[ ->+++ +<]>+ +++.< ++++[ ->--- -<]>- ----- .<+++ +++[- >---- --<]>
----- ----- -.+.- ..--- ---.. <++++ +++[- >++++ +++<] >.<++ +++++ [->--
----- <]>-- -.+++ .++++ ++.<+ +++++ [->++ ++++< ]>+++ +++++ .<+++ +++[-
>---- --<]> ----- ----- .---- ---.+ +++++ +.<++ ++++[ ->+++ +++<] >++++
+++++ .<+++ +++[- >---- --<]> ----- ----- -.--. ---.< +++++ ++[-> +++++
++<]> ++++. <++++ +++[- >---- ---<] >.+++ +.+++ +.<++ +[->- --<]> ---.<
+++++ ++[-> +++++ ++<]> +++++ +.<++ ++++[ ->--- ---<] >---- ----- --.+.
-.-.. ---.+ +++.< +++++ +[->+ +++++ <]>++ +++++ ++++. <++++ +++[- >----
---<] >---. <++++ +++[- >++++ +++<] >++.< +++++ +[->- ----- <]>-- -----
----- .<+++ +++++ [->++ +++++ +<]>+ +++++ ++++. <
```

https://tio.run/nexus/brainfuck#code=LVstLS0tLS0tPis8XT4rKy4tLS0tLS0uWy0-KysrKys8XT4uPi0tWy0tPisrKysrPF0-Li0tLS0uK1stPisrKzxdPisrLj4tWy0tLT4rPF0-LS4tLVstLS0tLS0-KzxdPisuPi1bLS0tPis8XT4uLS1bLS0tPis8XT4tLS4tLS0tWy0-KysrPF0-LlstLS0-KzxdPi0tLS0uKysrKysrKy4rKysrKysrKysuLS0uLi0tLS0tLS0tLS0uLS0tLS4tLS0tLi0tLS0tLi0tWy0tLS0tPisrKzxdPi4tLS1bLT4rKzxdPi0uLS0tLS0tLS4rKysrKysrKysrLlstLS0-KzxdPisrKy4&input=

```
++<]> ++++. <++++ +++[- >---- ---<] >.+++ +.+++ +.<++ +[->- --<]> ---.<
+++++ ++[-> +++++ ++<]> +++++ +.<++ ++++[ ->--- ---<] >---- ----- --.+.
-.-.. ---.+ +++.< +++++ +[->+ +++++ <]>++ +++++ ++++. <++++ +++[- >----
---<] >---. <++++ +++[- >++++ +++<] >++.< +++++ +[->- ----- <]>-- -----
----- .<+++ +++++ [->++ +++++ +<]>+ +++++ ++++. <
```

▶ Footer

▶ Input

▶ Arguments

▼ Output

```
flag{e676600a-06b4-4a20-b159-d5654415d0c3}
```

▶ Debug

用搜索引擎，搜一下就知道答案了。

广东省红帽杯攻防大赛WriteUp

tips:

下次做ctf还是要硬着头皮上啊，然后要抓紧时间进行验证，否则题目链接很快就不在了，就失去了一次测试的机会。

2017-5-8

对广东移动的数据集市做复查，然后页面只能用ie打开，复查的时候需要抓包，但是我的ie浏览器在虚拟机里面，然后就用了fiddler，之前没用过，查了下资料，发现挺好上手的。

fiddler在抓ie的包的时候，不用设置断点，就会抓到所有的包，估计是因为所有的包要经过fiddler所以浏览起来挺慢的。

还有就是包很多，有时候一时间没办法找到自己想要的某个特定的包，但是这个时候，比如要抓一个post包的时候，多提交几次，就能在fiddler中比较容易找到啦。

使用Fiddler提高前端工作效率 (介绍篇)

使用Fiddler提高前端工作效率 (实例篇)

2017-5-9

对佛山移动的jiakuan页面进行测试。

开始前，先要确认测试范围，这个先要问清楚然后免得白做了很多工作。

感觉测试的时候效率不是很高啊，下次依据页面的功能依次进行测试好了 。

然后还有写报告的时候，好麻烦，下次要自己写个干净规范的原始模板才行。

还有就是每一步骤都要截图，避免重复测试啊，感觉效率挺低的。

绕过验证码验证：

一般在爆破的时候，由于登录界面存在验证码，往往就会在爆破的时候增添了不少难度。

这个时候就要尝试绕过验证码。

参考list:

无聊入门一下传说中的验证码识别技术，学习笔记

浅谈某些网页验证码以及绕过验证

2017-5-10

python脚本：

python

```
POST /xxxx/xxxxxx/mobile/getValidateCode HTTP/1.1
Host: xxxx:9080
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.12; rv:53.0) Gecko/20100101 Firefox/53.0
Accept: */*
Accept-Language: zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
Referer: http://xxxx
Content-Length: 18
Cookie: JSESSIONID=0000PFLIij1W3MgA9_fkHTEK-LA:1; uid=112
Connection: close

mobile=1363012xxxx
```

python脚本发包(未实现1小时发一次的功能)：

```
# -*- coding:utf-8 -*-
__author__ = 'jerry'

import urllib
import urllib2
import cookielib
import webbrowser
import re
import os
import sys
import time



# GET the server IP and return URL
svrIP = 'xxxx:9080'
loginURL = 'http://' + svrIP +'/fsyd/app/main/toLogin'
PostURL = 'http://' + svrIP + '/fsyd/foshan/mobile/getValidateCode'

# cookie and opener
Cookie = cookielib.CookieJar()  # 创建CookieJar对象，自动管理cookie
# 创建pageOpener对象，把cookiejar塞进去，每次打开页面都会自动使用cookie
```

```
pageOpener = urllib2.build_opener(urllib2.HTTPCookieProcessor(Cookie))

# open the login page
# 打开登录页面，获取cookie
loginPageRequest = urllib2.Request(loginURL)
loginPageHTML = pageOpener.open(loginPageRequest).read()

# print loginPageHTML
# print Cookie

# login 时需要post的数据
PostData = {
    'mobile': '1363012xxxx',
}

# headers
PostHeader = {
    'Host': 'xxxx',
    'User-Agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10.12; rv:53.0) Gecko/20100101 Firefox/53.0',
    'Accept': '*/*',
    'Accept-Language': 'zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3',
    'Accept-Encoding': 'gzip, deflate',
    'Content-Type': 'application/x-www-form-urlencoded; charset=UTF-8',
    'X-Requested-With': 'XMLHttpRequest',
    'Referer': 'http://xxxx/fsyd/foshan/mobile/index',
     'Content-Length': '18',
    'Connection': 'close'
}

# request object请求对象,需要Post header和cookie

PostPageRequest = urllib2.Request(PostURL, urllib.urlencode(PostData), PostHeader)

# LOGIN: HTTP POST send:cookie,header发送到 default3.aspx
loginResponse = pageOpener.open(PostPageRequest)
PostPageResponse = pageOpener.open(PostPageRequest)

try:
    PostPageHTML = PostPageResponse.read()


except urllib2.HTTPError, f:
    print "Error code:", f.code
    print "Return content:", f.read()


print PostPageHTML
print '############################################################'
print '成功将数据post上去'
```

一开始写这个脚本的时候思路很混乱，主要想利用之前写过的模拟登陆脚本来修改一下，但是出现了情况有点稍微不同，一下子思路转不过来，无法解决。

这次一开始是想利用旧的已有的cookie，直接将数据post到服务器端。
而之前用的是cookiejar，先是在登录页面获取了cookie然后放进cookiejar进行管理。

这个时候有2个方向：

- 将已有的cookie放进直接cookiejar中
- 不用cookiejar，使用别的方式直接发送cookie

1.要以cookiejar能接受的格式将cookie添加进去
2.查了下资料，好像可以用httplib发送cookie，还未用过

最后还是用了之前的方式，创建cookielib.CookieJar对象然后去方位页面获取cookie后，来自动管理Cookie，会稍微稍繁琐一些，但是一旦创建，可供urllib2创建opener，后续的所有cookie更新和过期删除都是自动处理的。

其实还是对cookie不是特别理解，才搞得写了个脚本花了一个早上的时间，**什么情况下服务器会发放cookie?**

Python使用Cookie字符串发起HTTP请求的几个方法(2)

Python使用Cookie字符串发起HTTP请求的几个方法(1)

---

cookie：

什么是cookie?

Cookie是一种在**客户端保持HTTP状态信息**的常用技术，基于Cookie的会话保持常常出现在很多AX的部署案例中，尤其是涉及电子交易的系统部署中。此类系统往往要求负载均衡设备按照服务器下发的Cookie值实现会话保持。

Cookie的主要构成：

其中name=value是必选项，其它都是可选项。如下：

- name:

一个唯一确定的cookie名称。通常来讲cookie的名称是不区分大小写的。

- value:

存储在cookie中的字符串值。最好为cookie的name和value进行url编码

- domain:cookie

对于哪个域是有效的。所有向该域发送的请求中都会包含这个cookie信息。这个值可以包含子域(如：yq.aliyun.com)，也可以不包含它(如：.aliyun.com，则对于aliyun.com的所有子域都有效).

- path:

表示这个cookie影响到的路径，浏览器跟会根据这项配置，像指定域中匹配的路径发送cookie。

- expires:

失效时间，表示cookie何时应该被删除的时间戳(也就是，何时应该停止向服务器发送这个cookie)。如果不设置这个时间戳，浏览器会在页面关闭时即将删除所有cookie；不过也可以自己设置删除时间。这个值是GMT时间格式，如果客户端和服务器端时间不一致，使用expires就会存在偏差。

- max-age:

与expires作用相同，用来告诉浏览器此cookie多久过期（单位是秒），而不是一个固定的时间点。正常情况下，max-age的优先级高于expires。

- HttpOnly:

告知浏览器不允许通过脚本document.cookie去更改这个值，同样这个值在document.cookie中也不可见。但在http请求张仍然会携带这个cookie。注意这个值虽然在脚本中不可获取，但仍然在浏览器安装目录中以文件形式存在。这项设置通常在服务器端设置。

- secure:

安全标志，指定后，只有在使用SSL链接时候才能发送到服务器，如果是http链接则不会传递该信息。就算设置了secure 属性也并不代表他人不能看到你机器本地保存的 cookie 信息，所以不要把重要信息放cookie就对了

大致的工作流程如下：

当客户进行第一次请求时，客户HTTP请求（不带cookie）发送到负载均衡设备， 负载均衡设备根据负载均衡算法策略选择后端一台服务器，并将请求发送至该服务器，后端服务器在HTTP回复的Header中执行**set-cookie的动作**，将包含服务器端cookie的回复数据包发回负载均衡设备；

负载均衡设备会根据服务器回复的Cookie值，在自身会话保持表中查询，如果表中没有相同的Cookie值记录，那么将此Cookie值作为会话保持的“Key”值，将此次会话添加到会话保持表中；并将带有服务器插入cookie值的HTTP回复返回到客户端；

当客户请求再次发生时，客户将带有上次服务器cookie的HTTP请求发送到负载均衡设备，之后负载均衡设备根据会话保持表中该cookie值的会话保持记录，将HTTP请求（带有与上面同样的cookie）发到会话保持中表记录的服务器；

后端服务器进行请求回复。

如此循环，只要在会话保持表中的**空闲保持时间（Age）**递减到0之前，客户端发送带有相同Cookie值的HTTP请求，负载均衡设备始终会将这些请求分配到相同的服务器上。

（1）Cookie由服务器端生成，发送给客户端。

（2）客户端将Cookie的key/value 保存到某个目录下的文本文件内。

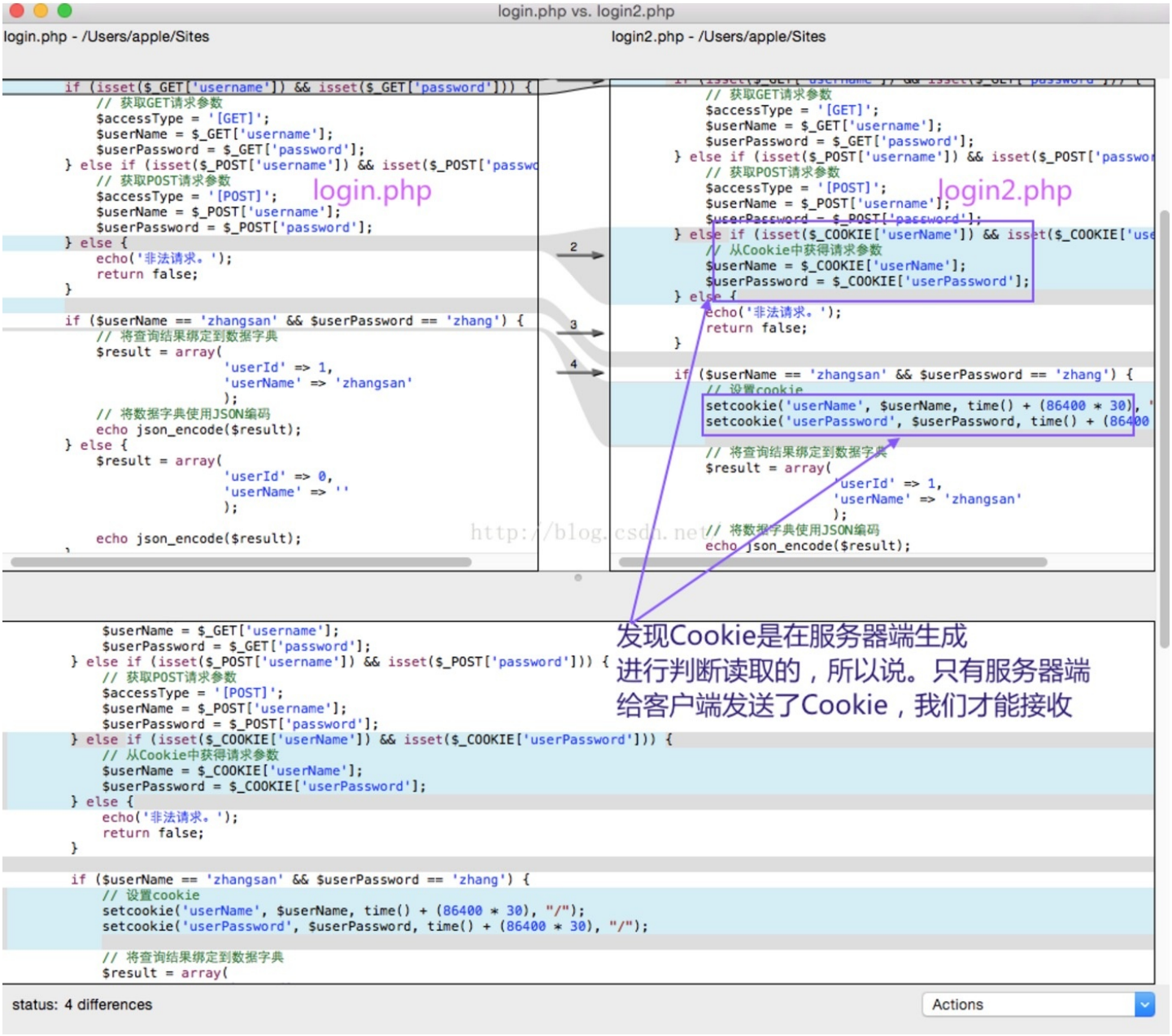
（3）如果客户端支持Cookie，下次请求同一网站时就可以Cookie直接发送给服务器。

（4）**Cookie名称和值由服务器端开发自己定义**。

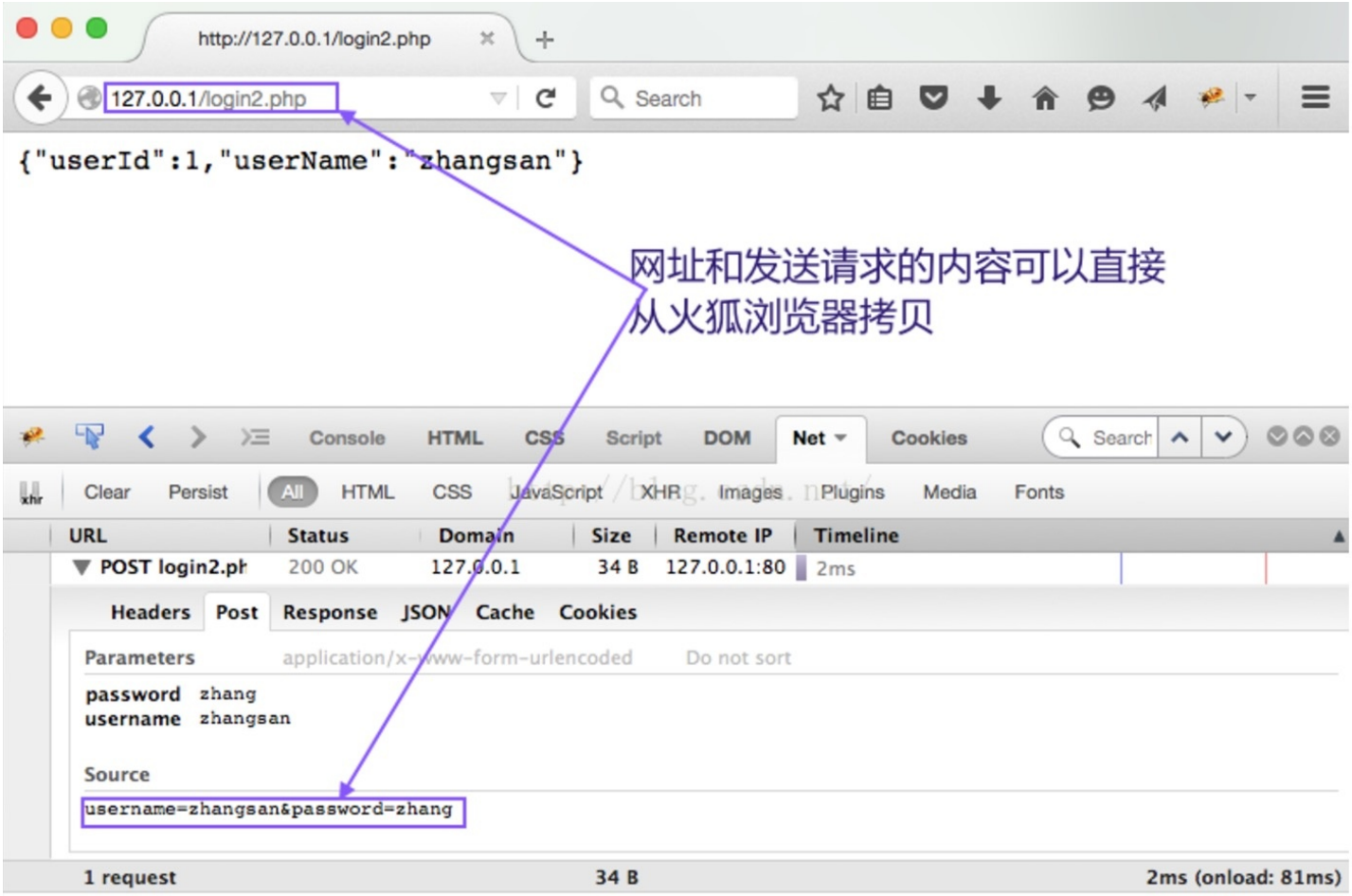
> 在渗透测试中，经常会遇到不同的cookie名称，通过cookie名称好像也可以判断服务器用的是什么组件。

---

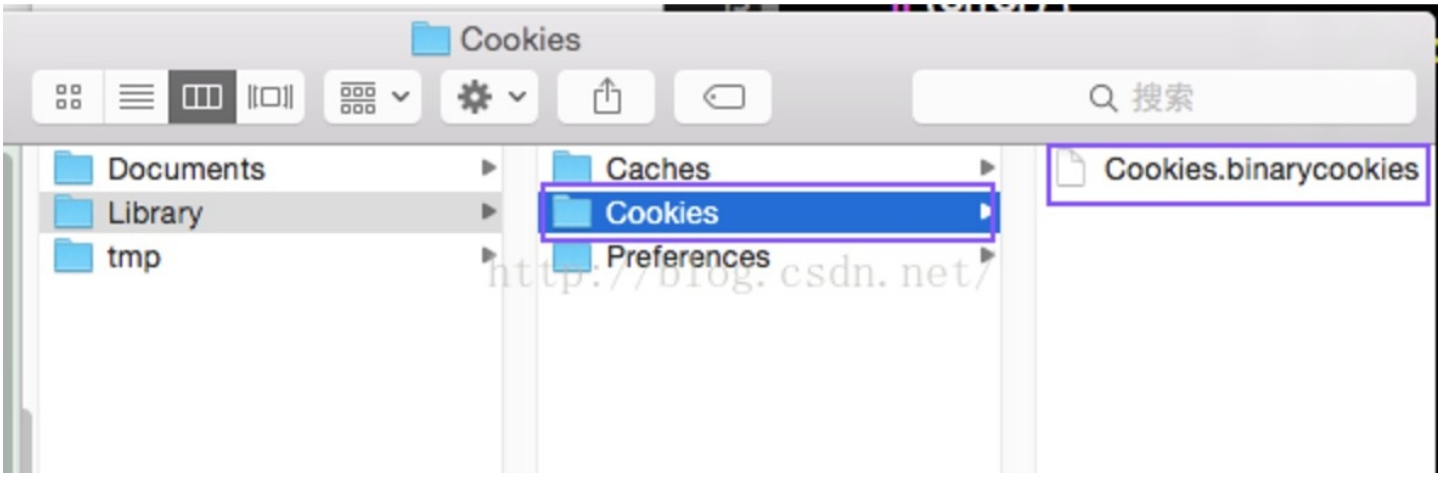
cookie流程的一个实例：


login.php vs. login2.php
login.php - /Users/apple/Sites
login2.php - /Users/apple/Sites
login.php
login2.php
2
3
4
发现Cookie是在服务器端生成
进行判断读取的，所以说。只有服务器端
给客户端发送了Cookie，我们才能接收
status: 4 differences
Actions

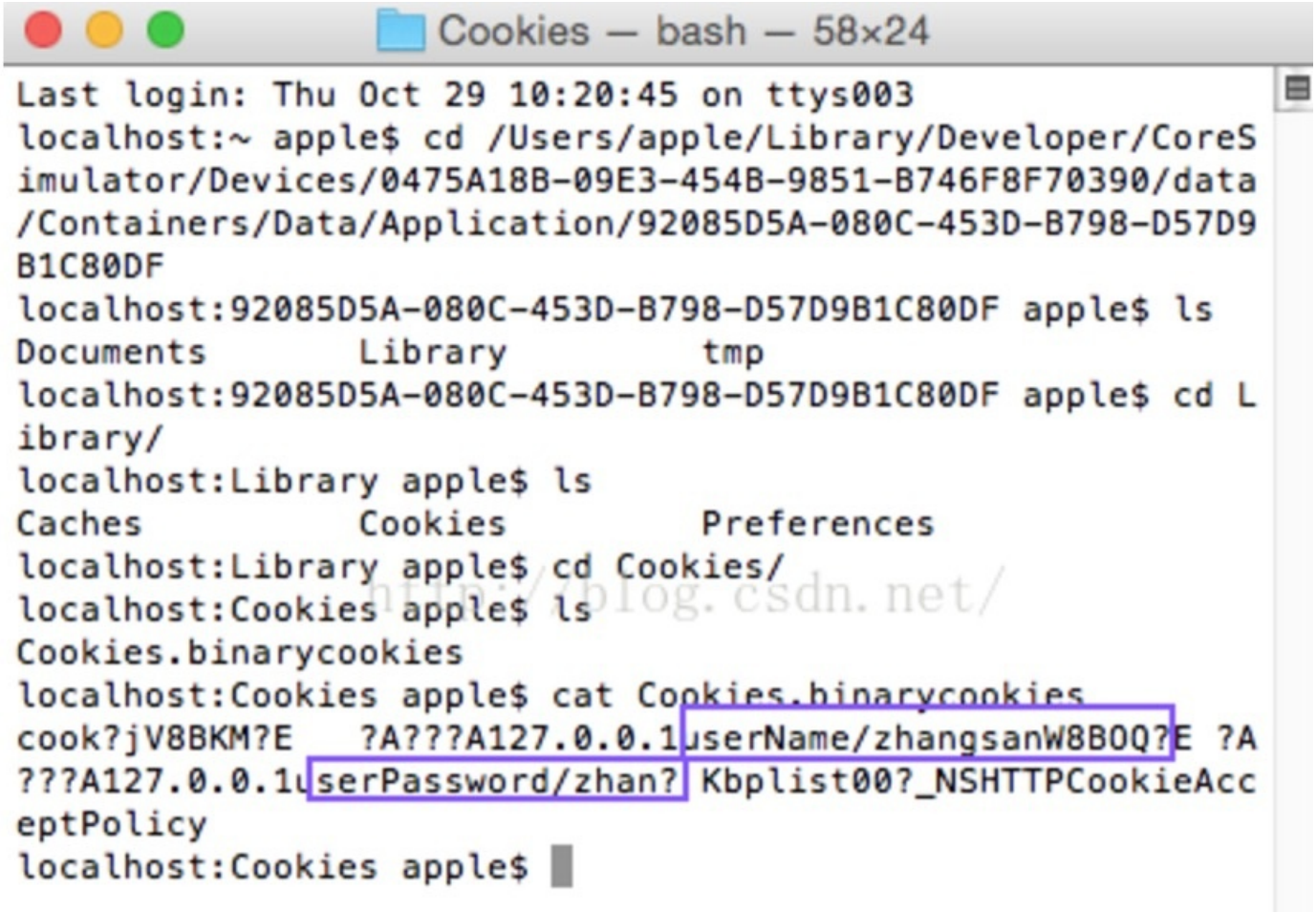
验证如下：



打开沙盒路径发现生成了Cookies文件夹，打开如下：



使用终端打开此Cookies文件夹内的文件如下：



Cookies如何传递

Cookies的信息是在Web服务器和浏览器之间传递的。保存在Http请求中。

（1）请求页面

在请求一个页面的Http头中，会将属于此页面的本地Cookies信息加在Http头中：

```
Cookie: My.Common.TestCookieInfo=Pkid=999&TestValue=aaabbbcccdddeee
```

（2）页面响应

如果页面要求写入Cookies信息，则返回的Http如下:

```
Set-Cookie: My.Common.TestCookieInfo=Pkid=999&TestValue=aaabbbcccdddeee; expires=Fri, 07-Aug-2009 03:40:59 GMT; path=/
```

Cookies如何查看

查看Cookies的txt文件

使用插件

Cookies高级知识:

1.Cookie 的限制

大多数浏览器支持最大为 4096 字节的 Cookie。
浏览器还限制站点可以在用户计算机上存储的 Cookie 的数量。

大多数浏览器只允许每个站点存储 20 个 Cookie；注意这里的20个是指主键值，也就是20条Cookies记录，但是每个Cookies记录还可以包含若干子键，下面会详细解释。如果试图存储更多 Cookie，则最旧的 Cookie 便会被丢弃。有些浏览器还会对它们将接受的来自所有站点的 Cookie 总数作出绝对限制，通常为 300 个。

2.Cookies的存储格式

Cookies可以包含一个主键， 主键再包含子键。比如asp.net中获取Cookies的格式是：
复制代码 代码如下:

```
Request.Cookies[key][subkey].ToString();
```

其中的key就是主键，subkey就是主键关联的子键。

（1）本地磁盘存储格式：

复制代码 代码如下:

```
My.Common.TestCookieInfo
Pkid=999&TestValue=aaabbbcccdddeee
localhost/
1536
3059603968
30021392
2348960464
30021191
*
```

其中的Pkid=999&TestValue=aaabbbcccdddeee 是Cookies的值，由于使用了subkey=subvalue的格式， 所以此Cookies是包含子键的。

（2）Javascript中的Cookie格式

在Javascript中给的Cookie是一个字符串，通过document.cookies获取。字符格式如下：
My.Common.SubKey=Pkid=999&TestValue=aaabbbcccdddeee; SingleKey=SingleKeyValue
上面的字符串包含了两个Cookies，一个是不包含子键的SingleKey， 一个是包含pkid和TextValue两个子键的My.Common.SubKey，两个Cookie通过“;”分割。

（3）Asp.Net中的Cookies格式

和所有的服务器端语言一样，Asp.Net中使用集合类保存Cookies集合：
复制代码 代码如下:

```
public sealed class HttpCookieCollection : NameObjectCollectionBase
{...}
```

通过HttpResquest和HttpResponse对象的Cookies属性，可以获取和写入当前页面的Cookies。

3.Cookies的内容编码格式

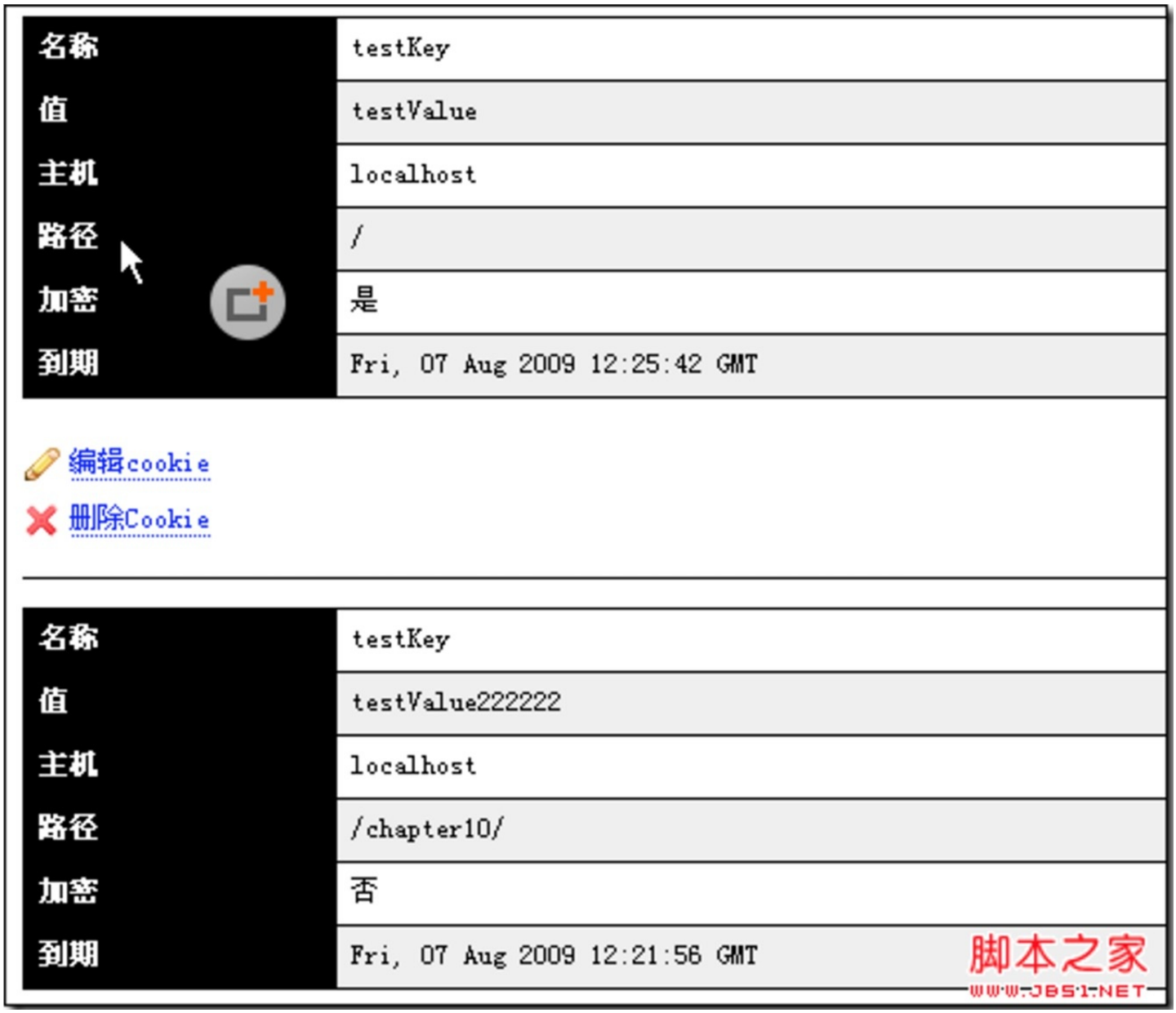
Cookies的值中可以保存除了“;”以外的标点符号。但是不能保存汉字。保存汉字会出现乱码。
所以对于Cookies中的内容要进行统一的编码和解码。

为了在浏览器端和服务器端都能够进行解码和编码， 所以要统一使用UTF编码格式。

主要是因为javascript中只能使用UTF编码格式。

4.Cookies的Path属性

Cookies的Path属性表示当前的Cookies可以作用在网站的那个路径下。
比如下面的两个同名的Cookies：



允许存在两个同名但是Path不同的Cookies。

无论是服务器端还是客户端，在获取时优先获取本页路径下面的Cookies。
也就是说如果在、/chapter10/路径下面的页面， 获取testKey这个Cookies的值，则只能获取到testValue222222这个值。

- path - domain属性的额外部分，判断特定路径的合法URL。如果域名和路径都符合，请求中会附上该cookie。如同domain属性一样，如果path属性也过于宽松，可能导致相同服务器的其他应用程序获取该cookie的漏洞。比如，如果path属性被设置为服务器根目录“/”，那么应用程序cookie可以在相同域名下的所有路径中发送。

5.Cookies的过期时间

如果保存Cookies时未设置过期时间， 则Cookies的过期时间为“当前浏览器进程有效”，即和Session一样关闭浏览器后则消失。
在asp.net中还可以通过设置HttpCookie对象的过期时间为DateTime.MinValue来指定此Cookies为跟随浏览器生效。（这句话来之不易啊，在脑袋等人的帮助下才查到的。）
如果设置了过期时间并且大于当前时间，则会保存Cookies值。
如果设置了过期时间但是小于等于当前时间，则清除Cookies值。

- expires - 这个属性用于设置持久性的cookie，cookie在设置日期前不会过期。持久化的cookie被用于浏览器会话和随后的会话，直到过期为止。一旦过期，浏览器会删除这个cookie。相对的，如果这个属性没有被设置，那么这个cookie仅仅在当前浏览器会话生命周期内才有效，当会话结束后将被删除。

6.Cookies与Session

有时我们会忽略Cookies与Session的关系。但是两者是密不可分的。
Session的唯一标示：SessionID是通常保存在Cookies中的（也可以保存在URL中）。对于Asp.Net而言，SessionID保存在键值为“ASP.NET_SessionId”的Cookies中，如图：

因为Cookies的存储数量是有限制的，所以我们的系统在保存Cookies的时候一定要注意防止冲掉这一个关键的Cookies。在下文介绍的最佳实践-以强对象方式保存Cookies的代码中特意对这个Cookies做了处理。

7.cookie加密：

取决于cookie的敏感属性，他们通常被**编码**或**加密**来包含其中的数据。 —《owasp-testing-guide-v4-zh》

在设置Cookies的属性时，有一个选项Secure用来控制Cookie的加密特性。
如果通过 SSL 连接 (HTTPS) 传输 Cookie，则为 true；否则为 false。默认为 false。

如果我们保存一个Cookies并设置加密，那么在非HTTPS的页面中，无论是使用javascript还是服务器端都无法获得此Cookies。但是在本地依然可以看到此Cookies的存在。

8.Cookies与Ajax

如果Ajax请求访问一个服务器页面，此服务器页面是可以向用户浏览器写入Cookies和Session的。

增加cookie安全性：

> secure属性
> 当设置为true时，表示创建的 Cookie 会被以安全的形式向服务器传输，也就是只能在 HTTPS 连接中被浏览器传递到服务器端进行会话验证，如果是 HTTP 连接则不会传递该信息，所以不会被窃取到Cookie 的具体内容。
>
> HttpOnly属性
> 如果在Cookie中设置了"HttpOnly"属性，那么通过程序(JS脚本、Applet等)将无法读取到Cookie信息，这样能有效的防止XSS攻击。

对于以上两个属性，
首先，secure属性是防止信息在传递的过程中被监听捕获后信息泄漏，HttpOnly属性的目的是防止程序获取cookie后进行攻击。
其次，GlassFish2.x支持的是servlet2.5，而servlet2.5不支持Session Cookie的"HttpOnly"属性。不过使用Filter做一定的处理可以简单的实现HttpOnly属性。GlashFish3.0(支持servlet3.0)默认开启Session Cookie的HttpOnly属性。
也就是说两个属性，并不能解决cookie在本机出现的信息泄漏的问题(FireFox的插件FireBug能直接看到cookie的相关信息)。

参考lists：

Cookies的各方面知识(基础/高级)深度了解

Cookie的原理解析——利用服务器发送来的Cookie进行判断并保存一些信息

会话令牌：

本地js加密&调试：

view-source:http://211.139.201.104:9080/fsyd/wego/js/des.js

还没试着怎么去本地调试

---

others参考list：

密码逻辑漏洞小总结

总结：

什么情况下服务器会发放cookie?
- 查了资料后，简单地说，看第一次访问页面时，看页面的Response有没set-cookie字段
- 再具体一些就看，cookie设置中的domain字段宽松与否，如果比较宽松，那么一开始获得cookie就可以一直用下去

- domain - 这个属性用于比较服务器请求的URL域名。如果该域名符合设置的domain或者是其子域，那么再检查path属性。

  注意只有特定域名的主机才能设置cookie的domain属性。domain属性也不能是顶级域名（如.gov或.com）来防止为其他域设置任意属性。如果domain属性没有设置，那么产生cookie的服务器域名被用于作为默认domain属性。

  举例说明，如果cookie在app.mydomain.com中设置，并且没有domian属性，那么cookie会在所有接下来的app.mydomain.com以及其子域（如hacker.app.mydomain.com）中提交，但不会在ptherapp.mydomain.com中出现。如果开发者希望宽松限制，那么他们应该将该domain属性设置为mydomain.com。在这个例子中，cookie能被发送到app.mydomain.com和其子域名，如hacker.app.mydomain.com，甚至是bank.mydomain.com。如果一个在子域上的漏洞服务器（如otherapp.mydomain.com），且domian属性过于宽松（如mydomain.com），那么漏洞服务器可以用来获取cookie（比如会话令牌）。

- path - domain属性的额外部分，判断特定路径的合法URL。如果域名和路径都符合，请求中会附上该cookie。如同domain属性一样，如果path属性也过于宽松，可能导致相同服务器的其他应用程序获取该cookie的漏洞。比如，如果path属性被设置为服务器根目录“/”，那么应用程序cookie可以在相同域名下的所有路径中发送。