你了解 XSS 攻击么

网络千万条，安全第一条。网安不规范，网站都完蛋！

作为前端工程师接触最多的漏洞我想就是 XSS 漏洞了，然鹅并不是所有的同学对其都有一个清晰的认识。这篇文章将带领大家认清 XSS 攻击，以及对于 XSS 攻击该如何防范。

什么是 XSS 攻击

XSS 攻击指的是：通过利用网页开发时留下的漏洞，恶意攻击者往 Web 页面里插入恶意 Script 代码，当用户浏览时，嵌入其中 Web 里面的 Script 代码会被执行，从而达到恶意攻击用户的目的。

XSS 全称是：跨站脚本攻击（cross site script）。按照国际惯例，命名应该以 CSS 命名，但是 CSS 与大家熟知的 层叠样式表（Cascading Style Sheets）重名了，因此取名为 XSS。

而实际上，就连“跨站脚本攻击”这个名字本身也另有来历，在这种行为最初出现之时，所有的演示案例全是跨域行为，所以叫做 ”跨站脚本” 。时至今日，随着 Web 端功能的复杂化，应用化，是否跨站已经不重要了，但 XSS 这个名字却一直保留下来。

由于现代浏览器的“同源策略”已经让运行在浏览器中的 javascript 代码很难对外站进行访问了，所以， 这个漏洞的名称可能存在一定的误导性，让很多初学者看了很多次都不能理解这个漏洞的原理。

随着 Web 发展迅速发展，JavaScript 通吃前后端，甚至还可以开发 APP，所以在产生的应用场景越来越多，越来越复杂的情况下， XSS 愈来愈难统一针对，现在业内达成的共识就是，针对不同的场景而产生的不同 XSS ，需要区分对待。

可即便如此，复杂应用仍然是 XSS 滋生的温床，尤其是很多企业实行迅捷开发，一周一版本，两周一大版本的情况下，忽略了安全这一重要属性，一旦遭到攻击，后果将不堪设想。

XSS 攻击带来的危害

据近些年 OWASP(OWASP 是世界上最知名的 Web 安全与数据库安全研究组织)统计 XSS 占所有 web 攻击的 22%，高居所有 web 威胁榜首。

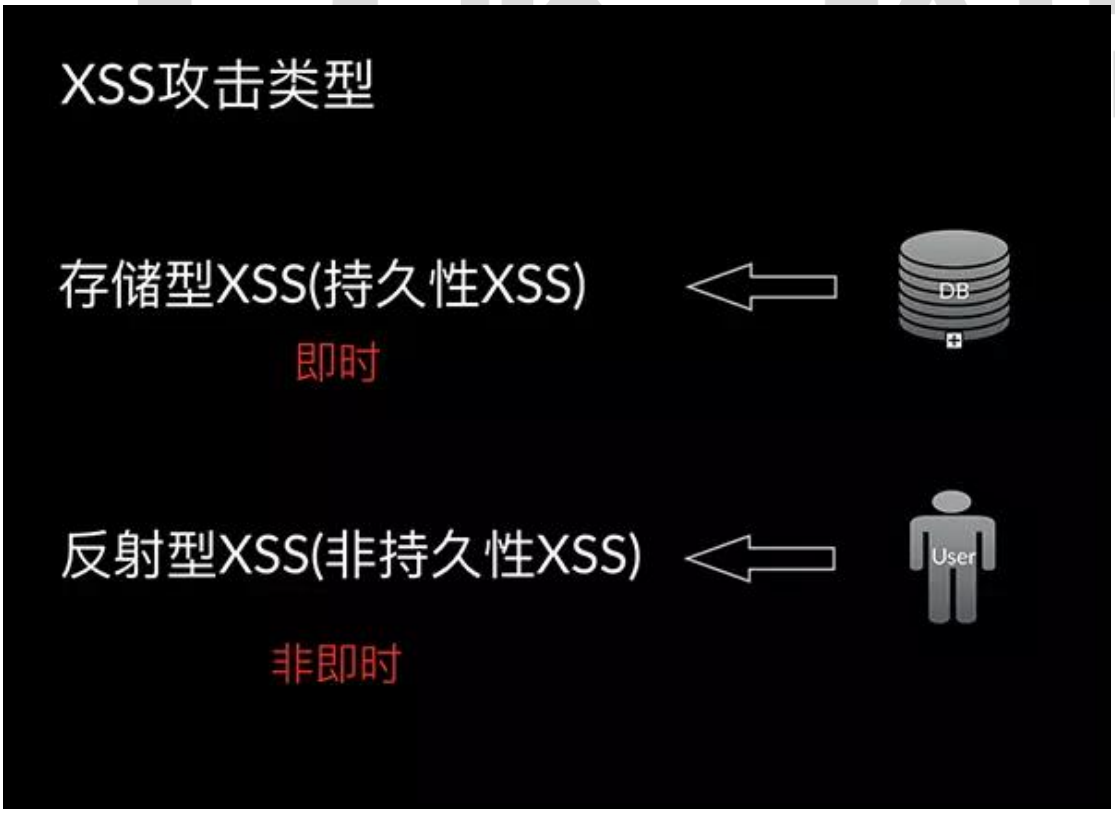
主要危害有：

- 通过 document.cookie 盗取 cookie 中的信息
- 使用 js 或 css 破坏页面正常的结构与样式
- 流量劫持（通过访问某段具有 window.location.href 定位到其他页面）
- dos 攻击：利用合理的客户端请求来占用过多的服务器资源，从而使合法用户无法得到服务器响应。并且通过携带过程的 cookie 信息可以使服务端返回 400 开头的状态码，从而拒绝合理的请求服务。
- 利用 iframe、frame、XMLHttpRequest 或上述 Flash 等方式，以（被攻击）用户的身份执行一些管理动作，或执行一些一般的如发微博、加好友、发私信等操作，并且攻击者还可以利用 iframe，frame 进一步的进行 CSRF 攻击。
- 控制企业数据，包括读取、篡改、添加、删除企业敏感数据的能力。

上面总结了这么多的危害，那么对于这么多的危害来说该怎么解决？这个是大家关注的问题。对于该怎么解决这些危害的前提，我们需要了解 XSS 攻击的类型，好对症下药，针对不同的情况有效的解决以上问题。

XSS 攻击分类

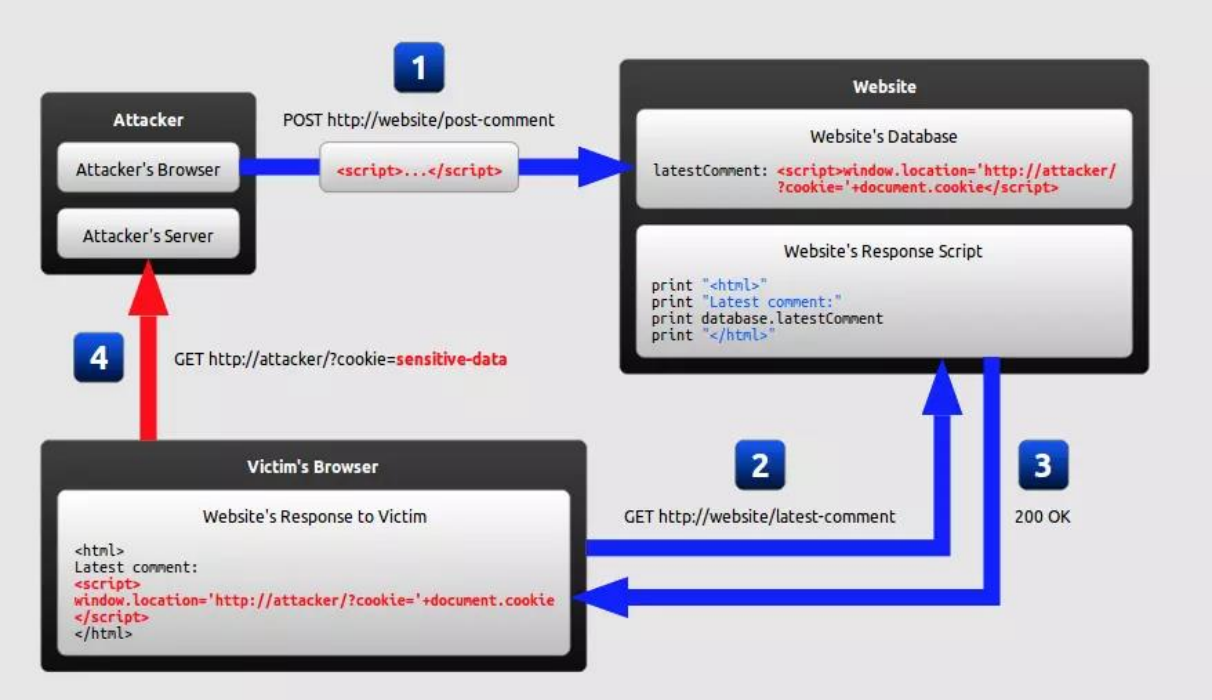
XSS 攻击大体分为两种：**反射型 XSS 攻击，与存储型 XSS 攻击。**



1. 存储型 XSS 攻击

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。这就意味着只要访问了这个页面的访客，都有可能会执行这段恶意脚本，因此储存型 XSS 的危害会更大。

存储型 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，**存储型 XSS 攻击更多时候用于攻击用户，而且在工作中的防范更多是防范存储型 XSS 攻击**。



接下来我们具体来看一下，存储型 XSS 攻击的示例

注型攻击常见的地方就是留言评论或者是含有表单提交的地方。 例如下面我们就以要给留言评论为例子来说明注入型攻击：

1. 首先，攻击者向一个 textarea 输入以下内容：

```
<script>getData(document.cookie)</script>
```

2. 然后，前端调用 ajax 向后端传值

```
$('.send').click(function(){
        $.post('message.htm',{'msg':$('textarea').val()},function(){});
});
```

3.接着，后端接收值写入数据库，同时又返回给前端展示。

```
app.post('message.html',function(req,res,next){

        //写入数据库
         //...
        //响应前端
    res.json({
        test: req.body.msg
    })

});
```

4.最后新的用户访问的时候，会读取数据库，并返回注入恶意代码的网站，用于获取用户信息，将用户信息返回给攻击者。

以上是一个存储型 XSS 攻击示例，接下来讨论一下反射性 XSS 攻击。

2.反射型 XSS 攻击

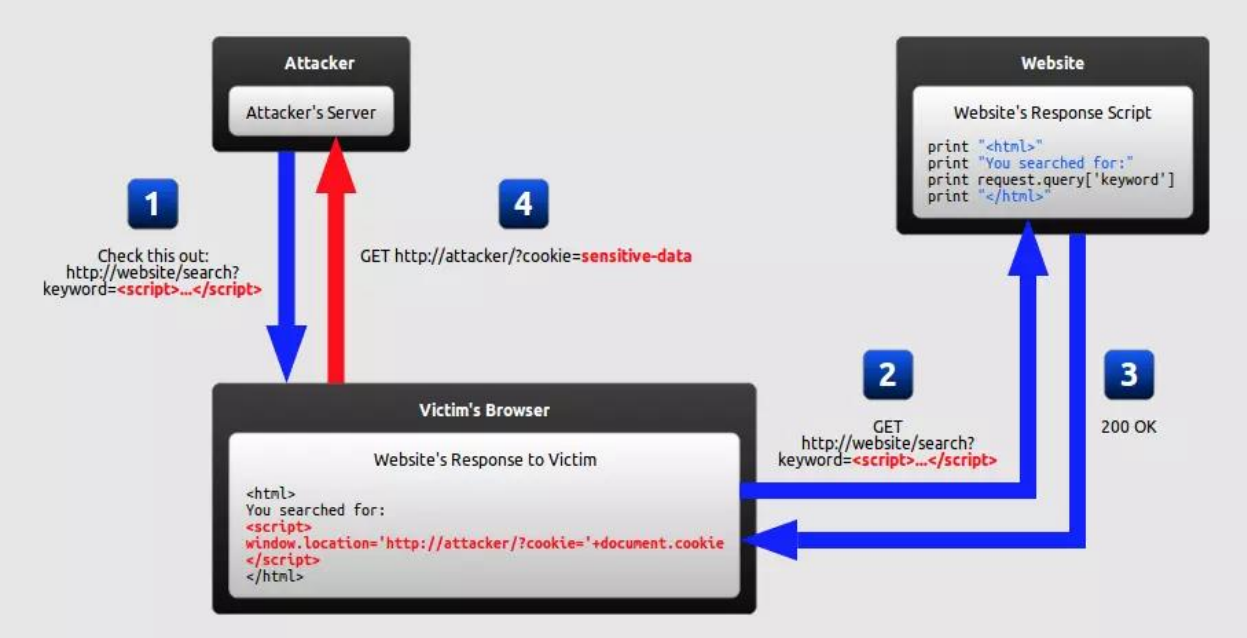
反射型 XSS 一般是攻击者通过特定手法（如电子邮件），诱使用户去访问一个包含恶意代码的 URL，当受害者点击这些专门设计的链接的时候，恶意代码会直接在受害者主机上的浏览器执行。

对于访问者而言是一次性的，具体表现在我们**把我们的恶意脚本通过 URL 的方式传递给了服务器，而服务器则只是不加处理的把脚本“反射”回访问者的浏览器而使访问者的浏览器执行相应的脚本。**

反射型 XSS 的触发有后端的参与，**要避免反射性 XSS，必须需要后端的协调，后端解析前端的数据时首先做相关的字串检测和转义处理。**

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.

整个攻击过程大约如下：



对于反射型 XSS 攻击，还存在一种单纯发生在客户端 XSS 攻击，也称为 DOM-XSS，当用户在当前的页面自己输入具有问题的代码，导致页面被恶意注入，但是不会影响到服务器。

比如下面的这个页面：

在页面中输入用名并显示：

当我们在输入框中输入 带有恶意代买的 script 标签 （死循环之类的）当我们点击确定，这段脚本就会作为标签插入当前页面。破坏当前页面执行。这种属于 DOM-XSS，尽管不会对服务器产生影响，但是也要防范。

接下来我们具体来看一下，反射型 XSS 攻击的示例：

前提： 目标网站存在 XSS 攻击漏洞。

1.攻击者诱导用户点击具有 XSS 攻击代码的目标网站链接(http://www.test.com?kw=<script>document.cookie</script>， 假设 test.com 是一个搜索网站)。

2.目标网站的服务器收到相应的链接，不作安全处理（问题就出在这里），处理完正常的业务逻辑，将搜索内容返回。

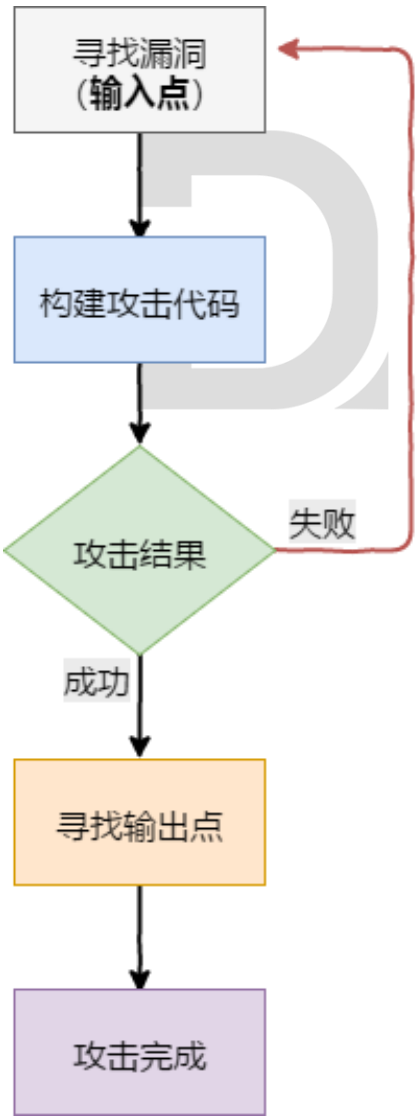
3.当搜索内容返回后，在用户端进行展示。并执行 XSS 攻击代码，(获取用户 cookie 等操作)。

4.攻击者通过向目标网站注入代码，代码客户端执行，并返回客户端的信息，完成了一次 XSS 攻击。

从上面的过程来看，反射型 XSS 攻击是不走数据库的，所以是非持久的攻击，而且只对当前客户端（用户）有效。

不论是反射型攻击还是存储型，攻击者总需要找到两个要点，即“输入点”与"输出点"，也只有这两者都满足，XSS 攻击才会生效。“输入点”用于向 web 页面注入所需的攻击代码，而“输出点”就是攻击代码被执行的地方。

大致上，攻击者进行 XSS 攻击要经过以下几个步骤：



经过上面的总结，我们也知道 XSS 如何造成攻击，接下来说一说如何防御这些攻击。

XSS 防御

1. XSS 防御之 HTML 编码

应用范围：将不可信数据放入到 HTML 标签内（例如 div、span 等）的时候进行 HTML 编码。

编码规则：将 & < > " ' / 转义为实体字符（或者十进制、十六进制）。

示例代码：

```
function encodeForHTML(str, kwargs){
    return ('' + str)
      .replace(/&/g, '&amp;')
      .replace(/</g, '&lt;')      // DEC=> &#60; HEX=> &#x3c; Entity=> &lt;
      .replace(/>/g, '&gt;')
      .replace(/"/g, '&quot;')
      .replace(/'/g, '&#x27;')    // &apos; 不推荐，因为它不在 HTML 规范中
      .replace(/\//g, '&#x2F;');
  };
```


HTML 有三种编码表现方式：十进制、十六进制、命名实体。例如小于号（<）可以编码为"十进制> <", "十六进制=> <", "命名实体=> <" 三种方式。对于单引号（'）由于实体字符编码方式不在 HTML 规范中，所以此处使用了十六进制编码。

2. XSS 防御之 HTML Attribute 编码

应用范围：将不可信数据放入 HTML 属性时（不含 src、href、style 和事件处理属性），进行 HTML Attribute 编码

编码规则：除了字母数字字符以外，使用 &#xHH;(或者可用的命名实体)格式来转义 ASCII 值小于 256 所有的字符

示例代码：

```
function encodeForHTMLAttibute(str, kwargs){
```

```
    let encoded = '';
    for(let i = 0; i < str.length; i++) {
      let ch = hex = str[i];
      if (!/[A-Za-z0-9]/.test(str[i]) && str.charCodeAt(i) < 256) {
        hex = '&#x' + ch.charCodeAt(0).toString(16) + ';';
      }
      encoded += hex;
    }
    return encoded;
  };
```

3. XSS 防御之 JavaScript 编码

作用范围：将不可信数据放入事件处理属性、JavaScirpt 值时进行 JavaScript 编码

编码规则：除字母数字字符外，请使用\xHH 格式转义 ASCII 码小于 256 的所有字符

示例代码：

```
function encodeForJavascript(str, kwargs) {
    let encoded = '';
    for(let i = 0; i < str.length; i++) {
      let cc = hex = str[i];
      if (!/[A-Za-z0-9]/.test(str[i]) && str.charCodeAt(i) < 256) {
        hex = '\\x' + cc.charCodeAt().toString(16);
      }
      encoded += hex;
    }
    return encoded;
  };
```

4. XSS 防御之 URL 编码

作用范围：将不可信数据作为 URL 参数值时需要对参数进行 URL 编码

编码规则：将参数值进行 encodeURIComponent 编码

示例代码：

```
 function encodeForURL(str, kwargs){
    return encodeURIComponent(str);
```

```
};
```

5. XSS 防御之 CSS 编码

作用范围：将不可信数据作为 CSS 时进行 CSS 编码

编码规则：除了字母数字字符以外，使用\XXXXXX 格式来转义 ASCII 值小于 256 的所有字符

示例代码：

```
function encodeForCSS (attr, str, kwargs){
  let encoded = '';
  for (let i = 0; i < str.length; i++) {
    let ch = str.charAt(i);
    if (!ch.match(/[a-zA-Z0-9]/) {
      let hex = str.charCodeAt(i).toString(16);
      let pad = '000000'.substr((hex.length));
      encoded += '\\' + pad + hex;
    } else {
      encoded += ch;
    }
  }
  return encoded;
};
```


任何时候用户的输入都是不可信的。对于 HTTP 参数，理论上都要进行验证，例如某个字段是枚举类型，其就不应该出现枚举以为的值；对于不可信数据的输出要进行相应的编码。

XSS 漏洞有时比较难发现，所幸当下 React、Vue 等框架都从框架层面引入了 XSS 防御机制，一定程度上解放了我们的双手。 但是作为开发人员依然要了解 XSS 基本知识、于细节处避免制造 XSS 漏洞。框架是辅助，我们仍需以人为本，规范开发习惯，提高 Web 前端安全意识。