跨站脚本攻击（XSS 攻击）

原理：

跨站脚本攻击(Cross Site Script 为了区别于 CSS 简称为 XSS)指的是恶意攻击者往 Web 页面里插入恶意 html 代码，当用户浏览该页之时，嵌入其中 Web 里面的 html 代码会被执行，从而达到恶意用户的特殊目的。

例子：

1、用户提交的数据未经处理，直接住注入到动态页面中

一个简单的留言板

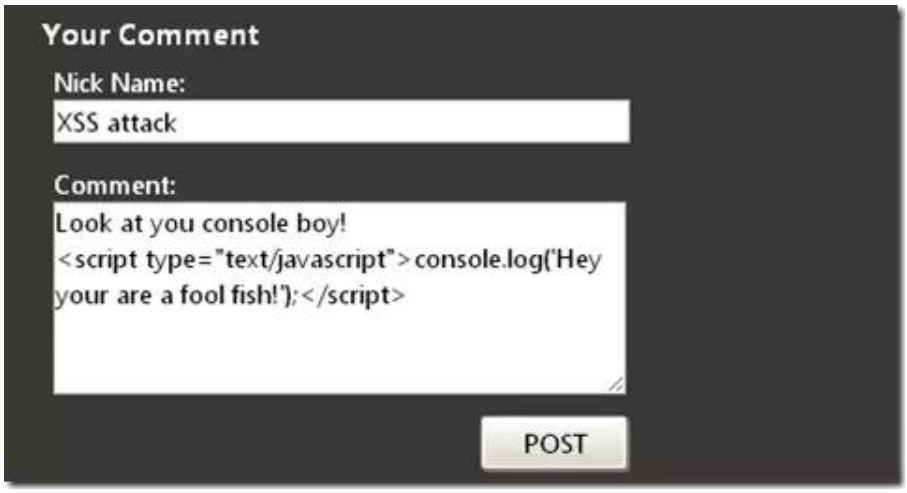
我们有个页面用于允许用户发表留言，然后在页面底部显示留言列表

```
<pre style="margin: 0px; white-space: pre-wrap; overflow-wrap: break-word; padding: 0p
<html>
<head>
    <?php include('/components/headerinclude.php');?></head>
    <style type="text/css"> .comment-title{ font-size:14px; margin: 6px 0px 2px 4px;
        } .comment-body{ font-size: 14px; color:#ccc; font-style: italic; border-botto
        }
    </style>
    <script type="text/javascript" src="/js/cookies.js"></script>
<body>
    <form method="post" action="list.php">
        <div style="margin:20px;">
            <div style="font-size:16px;font-weight:bold;">Your Comment</div>
            <div style="padding:6px;"> Nick Name: <br/>
                <input name="name" type="text" style="width:300px;"/>
            </div>
            <div style="padding:6px;"> Comment: <br/>
                <textarea name="comment" style="height:100px; width:300px;"></textarea
            </div>
            <div style="padding-left:230px;">
                <input type="submit" value="POST" style="padding:4px 0px; width:80px;"
            </div>
            <div style="border-bottom:solid 1px #fff;margin-top:10px;">
                <div style="font-size:16px;font-weight:bold;">Comments</div>
            </div>
            <?php
                require('/components/comments.php');
                if(!empty($_POST['name'])){
                    addElement($_POST['name'],$_POST['comment']);
                }
                renderComments(); ?>
        </div>
    </form>
</body>
</html></pre>
```

addElement()方法用于添加新的留言，而 renderComments()方法用于展留言列表，网页看起来是这样的

XSS

因为我们完全信任了用户输入，但有些别有用心的用户会像这样的输入



这样无论是谁访问这个页面的时候控制台都会输出“Hey you are a fool fish!”，如果这只是个恶意的小玩笑，有些人做的事情就不可爱了，有些用户会利用这个漏洞窃取用户信息、诱骗人打开恶意网站或者下载恶意程序等

危害：

1、盗取各类用户帐号，如机器登录帐号、用户网银帐号、各类管理员帐号
2、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
3、盗窃企业重要的具有商业价值的资料
4、非法转账
5、强制发送电子邮件
6、网站挂马
7、控制受害者机器向其它网站发起攻击

防范：
XSS 攻击其核心都是利用了脚本注入，因此我们解决办法其实很简单，不信赖用户输入，对

特殊字符如"<",">"转义，就可以从根本上防止这一问题，当然很多解决方案都对 XSS 做了特定限制，如上面这中做法在 ASP.NET 中不幸不同，微软 validateRequest 对表单提交自动做了 XSS 验证。

跨站请求伪造(CSRF 攻击)

原理：

CSRF(Cross Site Request Forgery)，即跨站请求伪造，是一种常见的 Web 攻击。CSRF 攻击过程的受害者用户登录网站 A，输入个人信息，在本地保存服务器生成的 cookie。然后在 A 网站点击由攻击者构建一条恶意链接跳转到 B 网站，然后 B 网站携带着的用户 cookie 信息去访问 B 网站。让 A 网站造成是用户自己访问的假相，从而来进行一些列的操作，常见的就是转账。

例子：

1、一个网站用户 Bob 可能正在浏览聊天论坛，而同时另一个用户 Alice 也在此论坛中，并且后者刚刚发布了一个具有 Bob 银行链接的图片消息。设想一下，Alice 编写了一个在 Bob 的银行站点上进行取款的 form 提交的链接，并将此链接作为图片 src。如果 Bob 的银行在 cookie 中保存他的授权信息，并且此 cookie 没有过期，那么当 Bob 的浏览器尝试装载图片时将提交这个取款 form 和他的 cookie，这样在没经 Bob 同意的情况下便授权了这次事务。

危害：

通过基于受信任的输入 form 和对特定行为无需授权的已认证的用户来执行某些行为的 web 应用。已经通过被保存在用户浏览器中的 cookie 进行认证的用户将在完全无知的情况下发送 HTTP 请求到那个信任他的站点，进而进行用户不愿做的行为。

防范：

1、验证码。

应用程序和用户进行交互过程中，特别是账户交易这种核心步骤，强制用户输入验证码，才能完成最终请求。在通常情况下，验证码够很好地遏制

CSRF 攻击。但增加验证码降低了用户的体验，网站不能给所有的操作都加上验证码。所以只能将验证码作为一种辅助手段，在关键业务点设置验证码。

2、Anti CSRF Token。

目前比较完善的解决方案是加入 Anti-CSRF-Token，即发送请求时在 HTTP 请
求中以参数的形式加入一个随机产生的 token，并在服务器建立一个拦截器来验证这个 token。服务器读取浏览器当前域 cookie 中这个 token 值，会进行校验该请求当中的 token 和 cookie 当中的 token 值是否都存在且相等，才认为这是合法的请求。

SQL 注入攻击

原理：

SQL 注入(SQL Injection)，应用程序在向后台数据库传递 SQL(Structured Query Language，结构化查询语言)时，攻击者将 SQL 命令插入到 Web 表单提交或输入域名或页面请求的查询字符串，最终达到欺骗服务器执行恶意的 SQL 命令。

例子：

某个网站的登录验证的 SQL 查询代码为：

strSQL = ``"SELECT * FROM users WHERE (name = '" + userName + ``"') and (pw = '"``+

``passWord +``");"
恶意填入
userName = ``"1' OR '1'='1"``;
与
passWord = ``"1' OR '1'='1"``;
时，将导致原本的 SQL 字符串被填为
strSQL = ``"SELECT * FROM users WHERE (name = '1' OR '1'='1') and (pw = '1' OR '1'='1');"
也就是实际上运行的 SQL 命令会变成下面这样的
strSQL = ``"SELECT * FROM users;"
因此达到无账号密码，亦可登录网站。所以 SQL 注入攻击被俗称为黑客的填空游戏。
危害：
得到管理员权限
防范：
1、增加黑名单或者白名单验证
白名单验证一般指，检查用户输入是否是符合预期的类型、长度、数值范围或者其他格式标准。黑名单验证是指，若在用户输入中，包含明显的恶意内容则拒绝该条用户请求。在使用白名单验证时，一般会配合黑名单验证。
2、安全检测
在项目完成的时候，始终坚持安全检测。
3、防止系统敏感信息泄露
对数据表的访问权限进行严格控制，尽量限制用户不必要的访问权限

文件上传漏洞
原理：
由于文件上传功能实现代码没有严格限制用户上传的文件后缀以及文件类型，导致允许攻击者向某个可通过 Web 访问的目录上传任意后台文件，并能够将这些文件传递给解释器，就可以在远程服务器上执行任意后台脚本。
防范：
1、检查服务器是否判断了上传文件类型及后缀。
2、定义上传文件类型白名单，即只允许白名单里面类型的文件上传。
3、文件上传目录禁止执行脚本解析，避免攻击者进行二次攻击。