防御手段

一、课前准备

一、课堂主题

二、课堂目标

- 密码强化
- 人机识别
- HTTPS
- 浏览器安全控制
- CSP(Content-Security-Policy)
- 密码学
    - 摘要 - md5 sha1 sha256 -hash
    - 对称
    - 非对称

##

四、知识要点

1. 密码安全（30min）

- 泄露渠道
    - 数据库被偷
    - 服务器被入侵
    - 通讯被窃听
    - 内部人员泄露
    - 其他网站（撞库）
- 防御
    - 严禁明文存储
    - 单向变换
    - 变换复杂度要求
    - 密码复杂度要求
    - 加盐（防拆解）
- 哈希算法
    - 明文 - 密文 - 一一对应
    - 雪崩效应 - 明文小幅变化 密文剧烈变化
    - 密文 -明文无法反推
    - 密文固定长度 md5 sha1 sha256
- 密码传输安全
    - https传输

    - 频次限制
    - 前端加密意义有限 - 传输层加密 不会泄露 但不代表不能登录
- 摘要加密的复杂度
    - md5反查

  https://www.cmd5.com/

A -> B

```
// /app/password.js
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 = '123432! @#! @#@! #'
console.log('md5', md5(psw))
console.log('sha1', sha1(psw))
module.exports = encryptPassword
```

两种强化方式

```
// index.js
  const encryptPassword = require('./password')
 if (res.length !== 0 && res[0].salt === null) {
        console.log('no salt ..')
        if (password === res[0].password) {
            sql = `
                update test.user
                set salt = ?,
                password = ?
                where username = ?
            `
            const salt = Math.random() * 99999 + new Date().getTime()
            res = await query(sql, [salt, encryptPassword(salt, password),
username])
            ctx.session.username = ctx.request.body.username
            ctx.redirect('/?from=china')
        }
    } else {
        console.log('has salt')
        if (encryptPassword(res[0].salt, password) === res[0].password) {
            ctx.session.username = ctx.request.body.username
            ctx.redirect('/?from=china')
        }
    }
```

讨论一下下列情况

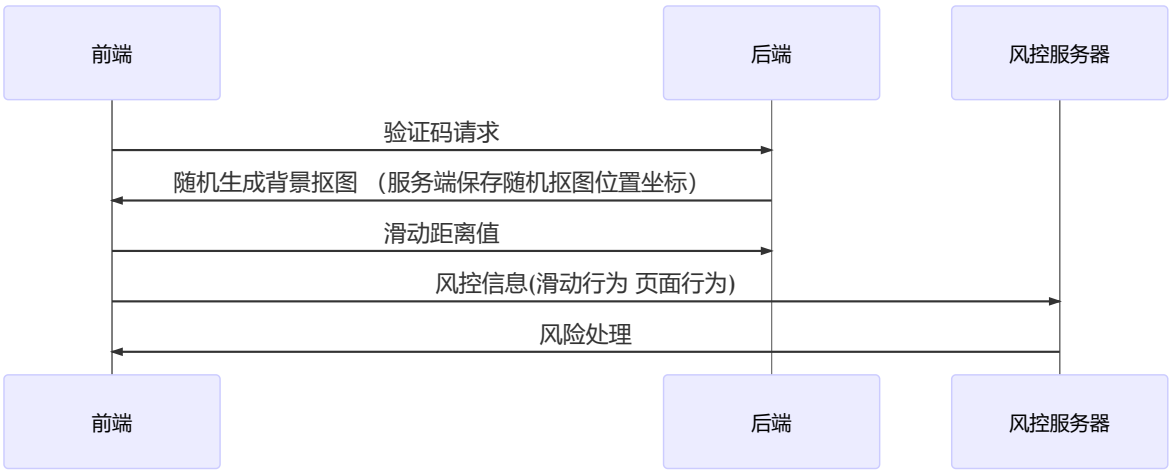
两次MD5是否可取 A OK B NO

- 只加盐好不好 A 好 B 不好
- 中间的字符串的作用
- 盐泄露是否会泄露密码
- 但密码很复杂还需要保证密码的复杂性吗 A 需要 B不需要

2. 人机验证 与 验证码

```
$('.verify-code font').text()
```

> http://www.lisa33xiaoq.net/1232.html



滑动验证码实现原理

1.服务端随机生成抠图和带有抠图阴影的背景图片，服务端保存随机抠图位置坐标；

2.前端实现滑动交互，将抠图拼在抠图阴影之上，获取到用户滑动距离值；

3.前端将用户滑动距离值传入服务端，服务端校验误差是否在容许范围内；

备注说明：单纯校验用户滑动距离是最基本的校验,处于更高安全考虑，可以考虑用户滑动整个轨迹、用户在当前页面上的行为等,可以将其细化复杂地步,可以根据实际情况设计。亦或借助用户行为数据分析模型,最终的目标都是增加非法的模拟和绕过的难度。

3. HTTPS 配置（30min）

> https 和密码学
> https://www.cnblogs.com/hai-blog/p/8311671.html
> 浏览器如何验证SSL证书
> http://wemedia.ifeng.com/70345206/wemedia.shtml

HTTP的弱点

```
#查看需要经过的节点
traceroute www.baidu.com
```

危害

- 窃听
  - 密码 敏感信息
- 篡改
  - 插入广告 重定向到其他网站(JS 和 Head头)

时代趋势

> - 目前全球互联网正在从HTTP向HTTPS的大迁移
> - Chrome和火狐浏览器将对不采用HTTPS 加密的网站提示不安全
> - 苹果要求所有APP通信都必须采用HTTPS加密
> - 小程序强制要求服务器端使用HTTPS请求

特点

- 保密性 (防泄密)
- 完整性（防篡改）
- 真实性（防假冒）

HTTP + SSL = HTTPS

什么是SSL证书

SSL证书由浏览器中“受信任的根证书颁发机构”在验证服务器身份后颁发,具有网站身份验证和加密传输双重功能

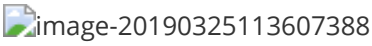
密码学

对称加密

> 对称加密的一大缺点是密钥的管理与分配，换句话说，如何把密钥发送到需要解密你的消息的人的手里是一个问题。在发送密钥的过程中，密钥有很大的风险会被黑客们拦截。现实中通常的做法是将对称加密的密钥进行非对称加密，然后传送给需要它的人。

> DES

不对称加密



- 产生一对秘钥
- 公钥负责加密
- 私钥负责解密
- 私钥无法解开说明公钥无效 - 抗抵赖
- 计算复杂对性能有影响(极端情况下 1000倍)

  常见算法 RSA（大质数）、Elgamal、背包算法、Rabin、D-H、ECC（椭圆曲线加密算法）。

> RSA原理
>
> http://www.ruanyifeng.com/blog/2013/06/rsa_algorithm_part_one.html
>
> 只能被1和本身整除的数叫质数,例如13,质数是无穷多的.得到两个巨大质数的乘积是简单的事,但想从该乘积反推出这两个巨大质数却没有任何有效的办法,这种不可逆的单向数学关系,是国际数学界公认的质因数分解难题.
> R、S、A三人巧妙利用这一假说,设计出RSA公匙加密算法的基本原理：
>
> 1、让计算机随机生成两个大质数p和q,得出乘积n；
>
> 2、利用p和q有条件的生成加密密钥e；
>
> 3、通过一系列计算,得到与n互为质数的解密密钥d,置于操作系统才知道的地方；
>
> 4、操作系统将n和e共同作为公匙对外发布,将私匙d秘密保存,把初始质数p和q秘密丢弃.
> 国际数学和密码学界已证明,企图利用公匙和密文推断出明文--或者企图利用公匙推断出私匙的难度等同于分解两个巨大质数的积.这就是Eve不可能对Alice的密文解密以及公匙可以在网上公布的原因.
> 至于"巨大质数"要多大才能保证安全的问题不用担心：利用当前可预测的计算能力,在十进制下,分解两个250位质数的积要用数十万年的时间；并且质数用尽或两台计算机偶然使用相同质数的概率小到可以被忽略.

SSH公钥登录原理

> https://www.cnblogs.com/scofi/p/6617394.html 原理介绍

- 密码口令登录

  通过密码进行登录，主要流程为：

  1、客户端连接上服务器之后，服务器把自己的公钥传给客户端

  2、客户端输入服务器密码通过公钥加密之后传给服务器

  3、服务器根据自己的私钥解密登录密码，如果正确那么就让客户端登录

- 公钥登录

公钥登录是为了解决每次登录服务器都要输入密码的问题，流行使用RSA加密方案，主要流程包含：

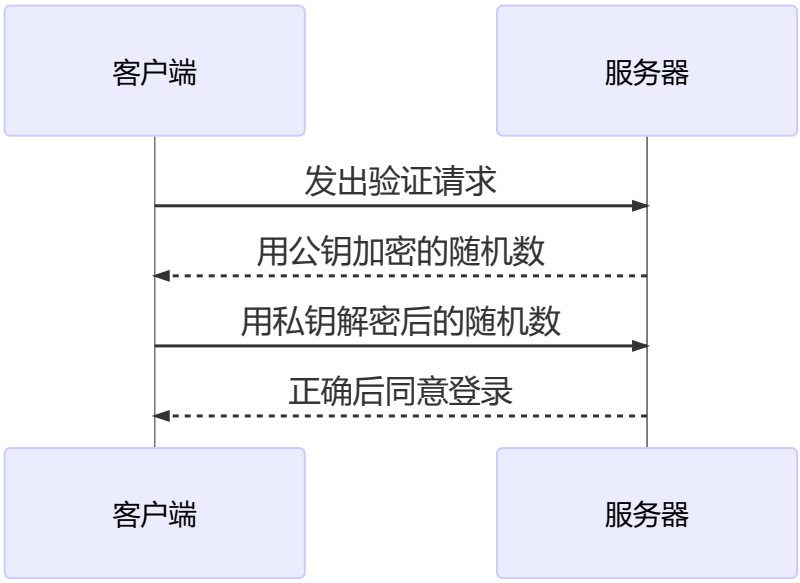
1、客户端生成RSA公钥和私钥

2、客户端将自己的公钥存放到服务器

3、客户端请求连接服务器，服务器将一个用公钥加密随机字符串发送给客户端

4、客户端根据自己的私钥加密这个随机字符串之后再发送给服务器

5、服务器接受到加密后的字符串之后用公钥解密，如果正确就让客户端登录，否则拒绝。



这样就不用使用密码了。

```
# 生成公钥
ssh-keygen -t rsa -P ''

xubin@xubindeMBP:~$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/Users/xubin/.ssh/id_rsa):
/Users/xubin/.ssh/id_rsa already exists.
Overwrite (y/n)? yes
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /Users/xubin/.ssh/id_rsa.
Your public key has been saved in /Users/xubin/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:IeFPfrcQ3hhP64SRTAFzGIHl2ROcopl5HotRi2XNOGk xubin@xubindeMBP
The key's randomart image is:
+---[RSA 2048]----+
|      .o*@=o     |
|     ..oEB=o     |
|      o@=+O .    |
|      B=+o @ .   |
|       =So* *    |
|      . o. = .   |
|           o     |
|                 |
|                 |
+----[SHA256]-----+
```

```
# 查看公钥
cat .ssh/id_rsa.pub

# 将公钥拷贝到服务器
scp ~/.ssh/id_rsa.pub root@47.98.252.XXX:/root

# 将公钥加入信任列表
cat id_dsa.pub >> ~/.ssh/authorized_keys
```

网站如何通过加密和用户安全通讯

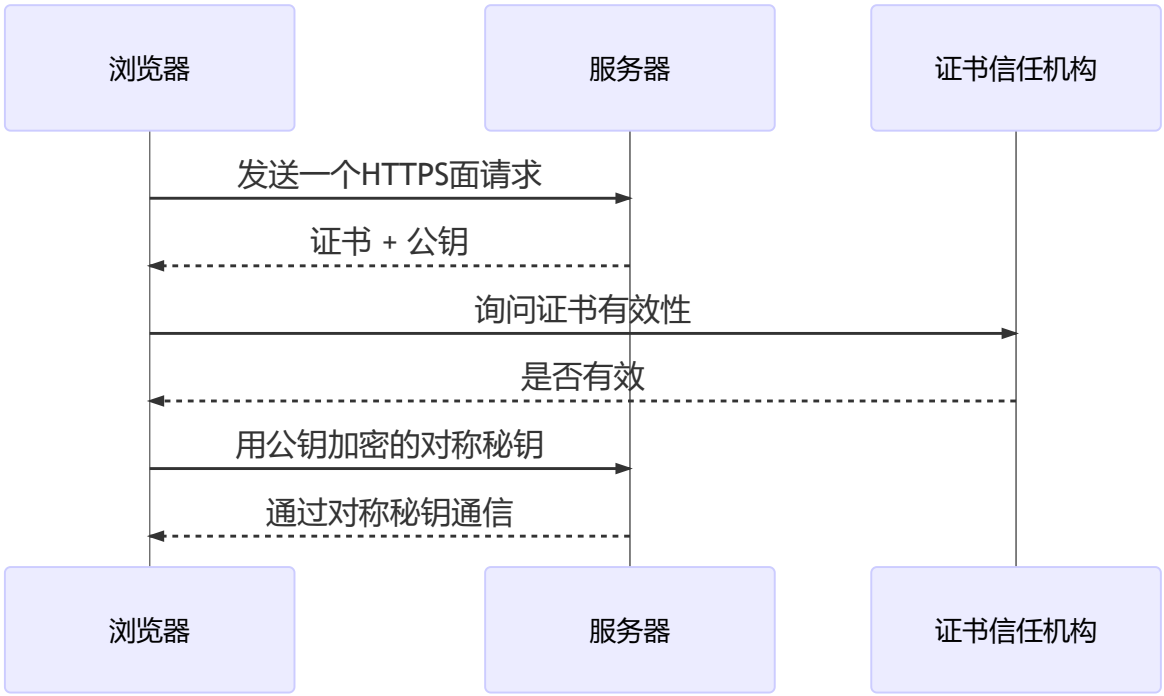


image-20190327161829513

https的主要实现过程说明：

（1）在通信之前，服务器端通过加密算法生成一对密钥，并把其公钥发给CA申请数字证书，CA审核后，结合服务端发来的相关信息生成数字证书，并把该数字证书发回给服务器端。

（2）客户端和服务器端经tcp三次握手，建立初步连接。

（3）客户端发送http报文请求并协商使用哪种加密算法。

（4）服务端响应报文并把自身的数字签名发给服务端。

（5）客服端下载CA的公钥，验证其数字证书的拥有者是否是服务器端（这个过程可以得到服务器端的公钥）。（一般是客户端验证服务端的身份，服务端不用验证客户端的身份。）

（6）如果验证通过，客户端生成一个随机对称密钥，用该密钥加密要发送的URL链接申请，再用服务器端的公钥加密该密钥，把加密的密钥和加密的URL链接一起发送到服务器。

（7）服务器端使用自身的私钥解密，获得一个对称密钥，再用该对称密钥解密经加密的URL链接，获得URL链接申请。

（8）服务器端根据获得的URL链接取得该链接的网页，并用客户端发来的对称密钥把该网页加密后发给客户端。

（9）客户端收到加密的网页，用自身的对称密钥解密，就能获得网页的内容了。

（10）TCP四次挥手，通信结束。

根证书在哪里

> windows

> 在Windows下按Windows+ R, 输入certmgr.msc，在“受信任的根证书颁发机构”-“证书中”找到“ROOTCA”，截止日期2025/08/23，单击右键，属性，可以查看其属性“禁用此证书的所有目的”
>  image-20190325151051619

> Mac
>
> 钥匙串
>
>  image-20190325153449439

> http://www.techug.com/post/https-ssl-tls.html HTTPS加密原理介绍

配置过程

- 修改开发机的host 前置

```
# 开发机的hosts文件 /etc/hosts
# 添加
127.0.0.1  www.josephxia.com
```

- 阿里云取得的真实证书 （域名 www.josephxia.com）

证书的格式说明

> PKCS 全称是 Public-Key Cryptography Standards ， 是由 RSA 实验室与其它安全系统开发商为促进公钥密码的发展而制订的一系列标准，PKCS 目前共发布过 15 个标准。 常用的有：
> PKCS#7 Cryptographic Message Syntax Standard
> PKCS#10 Certification Request Standard
> PKCS#12 Personal Information Exchange Syntax Standard
> X.509是常见通用的证书格式。所有的证书都符合为Public Key Infrastructure (PKI) 制定的 ITU-T X509 国际标准。
> PKCS#7 常用的后缀是： .P7B .P7C .SPC
> PKCS#12 常用的后缀有： .P12 .PFX
> X.509 DER 编码(ASCII)的后缀是： .DER .CER .CRT
> X.509 PAM 编码(Base64)的后缀是： .PEM .CER .CRT
> .cer/.crt是用于存放证书，它是2进制形式存放的，不含私钥。
> .pem跟crt/cer的区别是它以Ascii来表示。
> pfx/p12用于存放个人证书/私钥，他通常包含保护密码，2进制方式

> p10是证书请求
> p7r是CA对证书请求的回复，只用于导入

> 参考项目目录 nginx/conf.d/cert

- docker模拟nginx环境

```
# 安全课程根目录
version: '3.1'
services:
  nginx:
    restart: always
    image: nginx
    ports:
      - 80:80
      - 443:443
    volumes:
      - ./conf.d/:/etc/nginx/conf.d
      - ./html/:/var/www/html/
```

- 原始的80端口服务

```
# conf.d/www.josephxia.com.conf

server {
    listen       80;
    server_name  www.josephxia.com;

    location / {
        root   /var/www/html;
        index  index.html index.htm;
    }

}
# 增加的部分
server {
    listen 443;
    server_name localhost;
    ssl on;
    root html;
    index index.html index.htm;
    # 公钥 + 证书
    ssl_certificate   conf.d/cert/www.josephxia.com.pem;
    # 私钥
    ssl_certificate_key  conf.d/cert/www.josephxia.com.key;
    ssl_session_timeout 5m;
    ssl_ciphers ECDHE-RSA-AES128-GCM-
SHA256:ECDHE:ECDH:AES:HIGH:!NULL:!aNULL:!MD5:!ADH:!RC4;
    ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
    ssl_prefer_server_ciphers on;
    location / {
        root /var/www/html;
        index index.html index.htm;
    }
}
```

- 增加http -> https强制跳转

```
# conf.d/www.josephxia.com.conf
server {
    listen       80;
    server_name  www.josephxia.com;
    # location / {
    #     root   /var/www/html;
    #     index  index.html index.htm;
    # }

    location / {
        rewrite ^(.*) https://www.josephxia.com/$1 permanent;
    }
}
```

SSL证书分类

> https://blog.csdn.net/TrustAsia/article/details/73770588

入门级 DVSSL - 域名有效 无门槛

企业型 OVSSL - 企业资质、个人认证

增强型EVSSL - 浏览器给予绿色地址栏显示公司名字

3. helmet中间件（15min）

> 英[ˈhelmɪt] 头盔
>
> https://www.npmjs.com/package/koa-helmet

```
// npm i koa-helmet -s

const Koa = require("koa");
const helmet = require("koa-helmet");
const app = new Koa();

app.use(helmet());

app.use((ctx) => {
  ctx.body = "Hello World"
});

app.listen(4000);
```

- Strict-Transport-Security：强制使用安全连接（SSL/TLS之上的HTTPS）来连接到服务器。
- X-Frame-Options：提供对于“点击劫持”的保护。
- X-XSS-Protection：开启大多现代浏览器内建的对于跨站脚本攻击（XSS）的过滤功能。
- X-Content-Type-Options： 防止浏览器使用MIME-sniffing来确定响应的类型，转而使用明确的content-type来确定。
- Content-Security-Policy：防止受到跨站脚本攻击以及其他跨站注入攻击。

4. Session管理

对于cookie的安全使用，其重要性是不言而喻的。特别是对于动态的web应用，在如HTTP这样的无状态协议的之上，它们需要使用cookie来维持状态

- Cookie标示
  - secure - 这个属性告诉浏览器，仅在请求是通过HTTPS传输时，才传递cookie。
  - HttpOnly - 设置这个属性将禁止`javascript`脚本获取到这个cookie，这可以用来帮助防止跨站脚本攻击。
- Cookie域
  - domain - 这个属性用来比较请求URL中服务端的域名。如果域名匹配成功，或这是其子域名，则继续检查`path`属性。
  - path - 除了域名，cookie可用的URL路径也可以被指定。当域名和路径都匹配时，cookie才会随请求发送。
  - expires - 这个属性用来设置持久化的cookie，当设置了它之后，cookie在指定的时间到达之前都不会过期。

5. 浏览器安全控制

- X-XSS-Protection

  防止反射型XSS

- Strict-Transport-Security

  强制使用HTTPS通信

- CSP

> https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Headers/Content-Security-Policy__by_cnvoid#%E6%A6%82%E8%BF%B0
>
> https://juejin.im/post/5c6ad29ff265da2da00ea459

HTTP 响应头 **Content-Security-Policy** `允许站点管理者在指定的页面控制用户代理的资源。除了少数例外，这条政策将极大地指定服务源 以及脚本端点。这将帮助防止跨站脚本攻击（Cross-Site Script）` (XSS).

```
<meta http-equiv="Content-Security-Policy" content="default-src 'self'; img-src https://*; child-src 'none';">
```

安全防范的总结

https://www.tuicool.com/articles/7Ff2EbZ

无头浏览器技术 - JS 控制API 直接操纵Chrome