

# Web登录很简单？开玩笑！

java思维导图 2 days ago

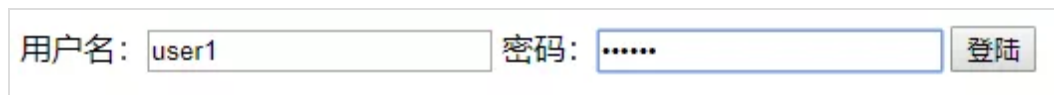
本文通过 Web 登录的例子探讨安全问题，登录不仅仅是简单地表达提交和记录写入，其安全问题才是重中之重。

## 1. 一个简单的HTML例子看看用户信息安全

标准的HTML语法中，支持在form表单中使用<input></input>标签来创建一个HTTP提交的属性，现代的WEB登录中，常见的是下面这样的表单：

```
<form action= "Application/login" method = "POST">
  用户名: <input id="username" name="username" type="text"/>
  密码: <input id="password" name="password" type="password" />
  <button type="submit">登陆</button>
</form>
```

form表单会在提交请求时,会获取form中input标签存在name的属性，作为HTTP请求的body中的参数传递给后台，进行登录校验。



例如我的账号是user1，密码是

123456，那么我在提交登录的时候会给后台发送的HTTP请求如下（Chrome或者FireFox开发者工具捕获，需开启Preserve log）：



```
▼ Form Data    view source    view URL encoded
username: user1
password: 123456
```

可以发现即便password字段是黑点，但是本机仍以明文的形式截获请求。

## 2. HTTP协议传输直接暴露用户密码字段

在网络传输过程中，被嗅探到的话会直接危及用户信息安全，以Fiddler或Wireshark为例，发现捕获的HTTP报文中包含敏感信息

Statistics		Inspectors		AutoResponder		Composer		
Headers	TextView	SyntaxView	WebForms	HexView	Auth	Cookies	Raw	JSON
XML								
QueryString								
Name				Value				
Body								
Name				Value				
username				user1				
password				123456				

### 3. 使用加密算法能保证密码安全吗？

WEB前端可以通过某种算法，对密码字段进行加密后，在将密码作为Http请求的内容进行提交，常见的包括对称和非对称加密。

对称加密:采用对称密码编码技术，它的特点是文件加密和解密使用相同的密钥加密。

非对称加密:需要两个密钥，公开密钥（publickey）和私有密钥（privatekey）。公开密钥与私有密钥是一对，如果用公开密钥对数据进行加密，只有用对应的私有密钥才能解密；如果用私有密钥对数据进行加密，那么只有用对应的公开密钥才能解密。

#### 3.1 使用对称加密

加密解密在前后台协商后，似乎是个不错的办法，比如，前台使用一个字符串位移+字符串反转的简单方法（举个例子，当然不能这么简单）。那么，如果原密码123456先移位：

123456-->456123

再进行反转：

456123-->321654

那么这样简单的方法似乎可以混淆原密码，并且轻松由后台进行相反操作复原。但是这有两个缺点：

- 1.前后端加密解密需要同时修改代码；
- 2.前端加密无非是写在JS里，但是JS有风险被直接破解从而识别加密方法。

#### 3.2 非对称加密HTTPS就一定是安全的吗？

非对称加密有着公钥私钥的存在，公钥可以随意获取，私钥是用来对公钥解密的本地存储，通过公私钥的机制似乎可以保证传输加密并且乃至现在还在使用的HTTPS就是基于这个原理。

但是HTTPS就一定安全吗？HTTP存在两种可能的风险：

- 1.HTTPS可以保证传输过程中的信息不被别人截获，但是细细思考下，HTTPS是应用层协议，下层采用SSL保证信息安全，但是在客户端和服务端，密文同样是可以被截获的；
- 2.HTTPS报文在传输过程中，如果客户端被恶意引导安装“中间人”的WEB信任证书，那么HTTPS中的“中间人攻击”一样会将明文密码泄露给别人。

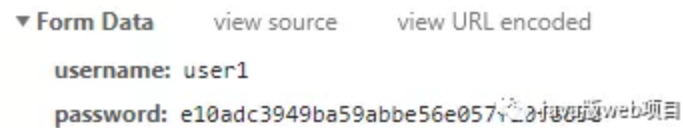
#### 4. 结论是，无论HTTP还是HTTPS，密码必须密文传输

想想HTTPS也不能一定保障用户密码信息，那么就应该考虑在应用层之上再继续对密码进行保护，也就是编写代码来进行控制，而不依赖特定协议，比较容易想到的就是利用不可逆加密散列函数MD5(string)，用户在注册输入密码的时候，就存储MD5(password)值，并且在WEB端先进行MD5(password)，然后将密码传输至后台，与数据库中的密文进行比较（PS：MD5函数在指定位数的情况下，对相同字符串运算值相同）。优点比较明显：

- 1.保证了用户数据库内部的密码信息安全；
- 2.传输过程中无论如何都不会使得用户的密文被破解出原密码；
- 3.简单高效，执行以及编码难度都不大，各种语言都提供MD5支持，开发快。

#### 5. 那太好了！这样可以省下HTTPS的钱了，真是这样吗？

回到开头的例子：用户输入的用户名是：user1，密码是：123456，那么不管在什么协议之下，可以看到实际发送的HTTP/HTTPS报文在MD5处理后是这样的：



▼ Form Data    view source    view URL encoded

username: user1

password: e10adc3949ba59abbe56e057f29f84e7

没错，加密登录成功了。但是，当我们庆祝密码安全的时候，发现账户的钱突然不翼而飞。这是为什么呢？黑客却笑的很开心：因为他们并不一定要获取到你的密码明文，如果直接截获你的密码密文，然后发送给服务器不是一样可以登录吗？因为数据库里的不也是MD5(password)的一样的密文吗？HTTP请求被伪造，一样可以登录成功，从而攫取其他的数据或者转走余额。

这怎么办？其实并不难，有很多种解决方法？其实原理都是类似的：那就是服务器缓存生成随机的验证字段，并发送给客户端，当客户端登录时，把这个一并字段传给服务器，用于校验。

##### 5.1 方案一：验证码

MVC场景。控制器将把数据的Model封装到View中，这种存在Session的连接方式，允许了在Session中存取信息。那么我们可以利用一些开源的验证码生成工具，例如JAVA中的Kaptcha，在服务端存放生成一个验证码值以及一个验证码的生成图片，将图片以Base64编码，并返回给View，在View中解码Base64并加载图片，并于用户下次登录时再进行比对。

## 5.2 方案二：token令牌

前后端分离场景。现在非常流行的前后端分离的开发模式大大提高了项目的开发效率。职责、分工明确，但是由于HTTP是无状态的（就是这一次请求并不知道上一次请求的内容），当用户登录时，根据用户的username作为key，生成随机令牌（例如UUID）作为value缓存在Redis中，并且将token返回给客户端，当客户端登录时，将完成校验，并且删除Redis中的那条缓存记录。

那么每次从服务器中获取认证的token，确实能保证HTTP请求是由前端传回来的了，因为token在每次登陆后都会删除并被重置，会导致黑客尝试重放账号密码数据信息来登陆的时候导致无法成功登陆。

总而言之，就是我拿到了账号以及密码的密文也登陆不了，因为，如果请求不包含后台认证的令牌token，是个非法请求。

## 6. 太不容易了！可是还别高兴的太早，当心数据被篡改

密码也加密了，黑客看不到明文了。加上Token了，登陆过程也没法再被截获重放了。可是想想这种情况，你在进行某宝上的网络支付，需要账号，密码，金额，token这四个字段进行操作，然后支付的时候你付了1块钱买了一袋包邮的小浣熊干脆面，某宝结算结束后，你发现你的账户余额被扣了1万元。这又是怎么回事呢？

因为即便黑客不登录，不操作，一样要搞破坏：当请求路由到黑客这边的时候，截获数据包，然后也不需要登录，反正账号密码都是对的，token也是对的，那么把数据包的字段改改，搞破坏就可以了，于是把money改成了1万，再传给服务器，作为受害者就莫名其妙踩了这个坑。可这该怎么解决呢？其实原理类似于HTTPS里的数字签名机制，首先科普下什么是数字摘要以及数字签名：

### 6.1 什么是“数字摘要”

我们在下载文件的时候经常会看到有的下载站点也提供下载文件的“数字摘要“，供下载者验证下载后的文件是否完整，或者说是否和服务器的文件”一模一样“。其实，数字摘要就是采用单项Hash函数将需要加密的明文“摘要”成一串固定长度（128位）的密文，这一串密文又称为数字指纹，它有固定的长度，而且不同的明文摘要成密文，其结果总是不同的，而同样的内容信息其摘要必定一致。

因此，“数字摘要“叫”数字指纹“可能会更贴切一些。“数字摘要“是HTTPS能确保数据完整性和防篡改的根本原因。

### 6.2 数字签名--水到渠成的技术

假如发送方想把一份报文发送给接收方，在发送报文前，发送方用一个哈希函数从报文文本中生成报文摘要，然后用自己的私人密钥对这个摘要进行加密，这个加密后的摘要将作为报文的“签名”和报文一起发送给接收方，接收方首先用与发送方一样的哈希函数从接收到的原始报文中计算出报文摘要，接着再用发送方的公用密钥来对报文附加的数字签名进行解密，如果这两个摘要相同、那么接收方就能确认报文是从发送方发送且没有被遗漏和修改过！这就是结合“非对称密钥加解密”和“数字摘要”技术所能做的事情，这也就是人们所说的“数字签名”技术。在这个过程中，对传送数据生成摘要并使用私钥进行加密地过程就是生成“数字签名”的过程，经过加密的数字摘要，就是“数字签名”。

因此，我们可以在WEB端对之前案例中提到的username+MD5(password)+token通过签名，得到一个字段checkCode，并将checkCode发送给服务器，服务器根据用户发送的checkCode以及自身对原始数据签名进行运算比对，从而确认数据是否中途被篡改，以保持数据的完整性。

## 7. 总结

看似非常简单的WEB登录，其实里面也存在着非常多的安全隐患。这些安全完善的过程是在一个实际WEB项目中遇到的，上面的分析演化是在应对项目安全的检查中所提出的解决方案，多少会有很多不足的地方，希望一起交流探讨，共同进步！

### 补充1：JS加密函数存在被破解

感谢园友mysgk指出完整性检验中关于JS加密函数存在被破解的问题：

问题描述：

如果黑客通过阅读前端js源码,发现加密算法,是否意味他可以构造可以被服务端解密的checkCode 来欺骗服务端呢？

我想了下，应该也是很多网站也在采取的策略：

摘要或加密JS算法不直接以静态文件的形式存在浏览器中，而是让WEB端去请求Server，服务器可以根据随机令牌token值决定返回一个相应随机的加密策略，以JS代码响应的方式返回，在异步请求响应中，加载JS摘要算法，这样客户端就可以动态加载数字摘要策略，保证无法仿造。

### 补充2：MD5存在隐患的问题

感谢园友EtherDream提出MD5已经过时，并且存在不安全的问题：

问题描述：

用MD5、SHA256 处理密码的过时了。。。现在 PBKDF、bcrypt 都在过时中。

- 1.本文重点侧重于方法思路的介绍，并不一定是要使用MD5函数，可以使用其他方式。
- 2.MD5存在隐患，之前确实没有考虑太多，不过非常感谢园友指出，确实是这样的，主要思想是：

对于MD5的破解，实际上都属于【碰撞】。比如原文A通过MD5可以生成摘要M，我们并不需要把M还原成A，只需要找到原文B，生成同样的摘要M即可。

设MD5的哈希函数是MD5()，那么：

$MD5(A) = M$

$MD5(B) = M$

任意一个B即为破解结果。

B有可能等于A，也可能不等于A。

大概意思也就是，截获了MD5加密后的密文，一样可以，找到一个不是原密码，但是加密后可以登陆成功的“伪原文”。

CSDN有一篇关于MD5风险的博客写的非常好，推荐一下：MD5算法如何被破解

从中可以看到一点，MD5函数确实能被反向“破解”，但是这个“破解”只是找到一个经过MD5运算后得到相同结果的原文，并非是用户的明文密码。但是这样会被破解登录的可能，确实是需要采用更完善的算法进行加密，再次感谢。

原文：[www.cnblogs.com/letcafe](http://www.cnblogs.com/letcafe)

作者 | letcafe；来源 | 博客园

People who liked this content also liked

【技术分享】SM2国密算法应用的高危漏洞——CVE-2021-3711

安全客

SSH 真的 yyds !

Java面试那些事儿

构建通用WebSocket推送网关的设计与实践

云时代架构