你保存用户密码的姿势正确吗？

作者：88250
原文链接：https://ld246.com/article/1456284835965
来源网站：链滴

摘要

这几年陆陆续续有很多大站被脱裤，最终导致了很多用户的密码明文泄露。本文不探讨脱裤技术，主研究的是如何正确（尽量安全）地保存用户密码。

哈希

“把明文加密后再保存数据库”应该是大家的共识。这个过程应该是不可逆的（不能通过加密后的串得到原文），所以对于这个“加密”过程更贴切的叫法应该是 - 哈希：

> Hash，一般翻译做“散列”，也有直接音译为“哈希”的。就是把任意长度的输入通过散列算法变成固定长度的输出，该输出就是散列值。
>
> 这种转换是一种压缩映射，也就是，散列值的空间通常远小于输入的空间，不同的输入可能会散列成同的输出，所以不可能从散列值来唯一的确定输入值。
>
> 简单的说就是一种将任意长度的消息压缩到某一固定长度的消息摘要的函数。

我们常用的 Hash 算法主要就是 MD5 和 SHA，并且很多时候我们就是使用其中一种来加密用户密码。

以上两类哈希算法主要是用于校验文件/数字签名，但并不适合用来保护用户密码，虽然对于我们来说感觉上_似乎起到了一定安全防护的效果，但其实效果很差，特别是 MD5，秒秒钟就可以找到碰撞值。

加盐

上面我们提到过，将用户明文直接哈希后保存并不是正确的姿势，那加盐后呢?

我们先看下加盐解决的问题：如果两个用户密码一样，那么哈希后的值也是一样，攻击者通过查表法容易就能破解原文并且一石多鸟。因为攻击者不知道用户的颜值和哈希算法，所以不大可能破解。

加盐的确是正确的思路，前提是：

- 盐长度不能太短，并且必须是随机生成的
- 用户更新密码的同时也需要更新盐值

并且加盐后使用的算法非常非常重要，而恰恰是这一点大家都没怎么弄对。

加盐哈希

我们直接说结论吧：

- **不要**使用自己设计的算法（比如 sha1(sha1(password+salt))、sha1(md5(salt) + md5(password)) 等类似的）
- **要**使用 crypt 函数并且使用安全算法的参数（$2y, $5, 6）。*nix 系统正是使用 crypt 保护用户口令的，我们应该使用这样久经考验的姿势

使用流程

以 golang 代码举例：

```
package main

import (
    "fmt"

    "github.com/kless/osutil/user/crypt/sha512_crypt"
)

func main() {
    // 1. 生成密码安全处理后 hash 串
    c := sha512_crypt.New()
    hash, _ := c.Generate([]byte("secret"), nil) // 第二个参数是 salt，设置为 nil 表示自动生成 salt

    fmt.Println(hash)

    // 2. 将该 hash 存库
    // ....

    // 3. 用户登录验证
    userInput := "secret"
    inputHash, _ := c.Generate([]byte(userInput), []byte(hash)) // 第一个参数是用户输入的密码，
二个参数是数据库中取出的 hash 串
    if inputHash == hash {
        fmt.Println("登录成功")
    } else {
        fmt.Println("登录失败")
    }
}
```

输出形如：

```
$6$P0pVrLOL89I7Y4.Y$IlUd2ABQnUcSecMr2nMgB8lay58RXrQGODwVOQHtfP7IYr.mpGE7dn5
nmxxb9RWkM8o/rNNJCxs3mLKsB5Xl.
登录成功
```

要理解第 3 步中的验证原理，需要先知道 salt 的格式。在上面的例子中，我们没有指定 salt，而是用自动生成的，最终生成的 hash 串中也是带有这个 salt 值的，也就是 6P0pVrLOL89I7Y4.Y$ 这部分。

我们指定一下 salt 再调用函数就清晰一些了：

```
hash, _ := c.Generate([]byte("secret"), []byte("$6$rounds=5000$saltstr"))
```

输出：

```
$6$rounds=5000$saltstr$SH73gRYn1O7I/XTiq3AjDklhcqGvJ9vp65/TuFq2vQOoJEaejlTvsXOfy
dBpHju9v0Vi.VOcFh.79yy/kksl1
```

- $6：指定了 crypt 算法为 SHA512
- $rounds=5000：指定了迭代 5000 次
- $saltstr：盐值

也就是说，salt 这个参数实际上是定义了算法、盐值两个部分。

回到登录验证的例子，第 3 步中把用户输入的密码和数据库中取出的 hash 作为 salt 加密就很好理解，其实数据库中取出的 hash 在该函数中只是使用了 salt 部分，所以如果用户输入的密码正确的话，算结果会和持久化的 hash 一致。

参考

- 文中提到的大部分内容在这篇 文章中有详述，建议阅读
- PHP: crypt - Manual
- A golang password hashing library