一般来说，密码有几种方式被泄露，第一种就是你的密码被泄露了，通过撞库和破解软件进行解密，第二种就是登陆破解进行爆破，第三种如通过社工进行业务逻辑获取，我通过破解的思路来分析密码的安全设计防御方案。

在前几年我用 python 仿了现在最主流的破解软件 hashcat，通过他的思路分析如何破解密码，通过他来先讲讲实时破解的这块。

首先他的代码分三大块，第一是你预想的密码长度，第二是你密码的破解格式，第三是加密的方式。

他代码的准备工作类似达芬奇密码的密码锁原理，假设你的密码长度是 8，那么你的长度设计为 8，然后配置你的密码破解格式?d?d?d?d?d?d?d?d，相当于你的 8 位的都是数字，然后选择你的加密方式，我设计成 md5.我的配置信息为：

format = "?d?d?d?d?d?d?d?d"

haxtype = "md5hax"

然后代码就会先判断你的 format，他发现 format 设计成 8 位，然后每个元素都是 d，则全部取 0-9 的数字进行遍历：

thenumlist = ["0","1","2","3","4","5","6","7","8","9"]

thesmllist = ["a","b","c","d","e","f","g","h","i","j","k","l","m","n","o","p","q","r","s","t","u","v","w","x","y","z"]

thebiglist =
["A","B","C","D","E","F","G","H","I","J","K","L","M","N","O","P","Q","R","S","T","U","V","W","X","Y","Z"]

从 thenumlist 里面取了 8 排进行 for 循环，从 00000000 遍历到 9999999.

当数字生成之后，按照 haxtype = md5hax，则调用 md5 进行加密，然后跟我们需要破解的密码进行匹配。

```
def md5hax(self,trgdata):

    m = hashlib.md5()

    m.update(trgdata.encode(encoding='UTF-8'))

    return m.hexdigest()
```

破出来，则答应出明文的密码，破不出来就退出程序。这个基本就是 hashcat 的执行逻辑。

从他的设计原理看出他有几个弱点，第一他没有分布式的功能，第二他的加密方式不能够自定义函数，第三他的破解格式过于固定，需要你先猜想密码的格式，这样你每跑一个密码的手工消耗太大了。

但是他的优势在于，第一他调用了 GPU，GPU 主要是用于处理图形，也就是显卡的功能，由于它的特性和存在的环境，在大部分时间里面是比 CPU 快，但是有个缺点，不少人用它来跑密码的话会减断显卡的寿命（在没限速情况下）；第二就是他利用汇编的语法配合 c 语言进行遍历，基本上是最快的语言了。我仿了他的源码之后，我就发现了它在使用过程中会出现的情况是：要么很快就破解密码，要么就破不出来的概率较大。所以定制了它的代码，做了像 cmd5 的工作，先把密码都破出来，存到数据库中，进行撞库。

接下来就是撞库这块的思路。

我们利用 hashcat 进行常用密码如数字，字母，大小写+字母的 6~11 位的组合进行常规的 md5，sha1 等密码进行加密，然后入库。如果我们想看我们产品的密码是否容易被破，则把密码丢进去库里查询，如果查得出来则需要告知我们的产品的密码复杂度和加密方式还是很弱。所以这时候撞库拼的就是时间+存储空间，你有足够的服务器存密码，你有足够时间跑密码，则能够覆盖大部分人的密码了。

到这里，我们先按照安全二级的需求进行防御：大小写字母+数字的复杂度，对密码进行 md5 的加密，登陆锁定限制。

这样在线暴力破解肯定是走不通了，剩下两条路，第一拖了生产环境的数据库，第二把其他相关联的旁站或者别的库进行用户名撞库。第二条我们就忽略，主要讲第一条。

如果我们的生产环境真的不幸数据库被拖了，按照大小写字母+数字的复杂度，对密码进行 md5 的加密，我们算算成本：

假设我们有 100w 条数据，范围是 8~16 位。如果我改造了 hashcat，他会分布式，也就是分成 8~16 的几个段进行加密，则有 9 个程序一直在跑，然后入库，条件为：

数据量 = 1000000 条

数据范围 =9 个

密码复杂度 = 数字 10 + 大写字母 26 +小写字母 26 = 62 个

以 8 位为例，每一位都是 62 个可能性，则密码量大概为 62 的 8 次方，这对于人类来说是很大的数字。然后有 9 组，每一组比前一组多 1 个次方。

那么这个数字的规模非常大，但是 hashcat 以每秒也能够生成 1.35 亿个哈希值，这样看数字就不大了。我准备 9 个电脑，对 100w 条数据进行爆破的话，8 位来说需要 2 小时爆破。这样看破 100w 条话的时间好像很长。但是大家忽略了实际情况。

第一，100w 条的数据肯定有不少重复的。第二，hashcat 本身也带智能批跑功能。我们算一个策略，首先先跑 8 位，9 位，10 位的组合，这时候假设跑完之后还剩下 10%没跑出来，再丢到 11~16 的机器慢慢跑。这样的话，除了有特殊字符的密码，基本上还是能够跑出来密码的，一般来说时间会在 3 个月到 6 个月时间内跑个 97%的进度了，毕竟超过 11 位密码的人不多。

这时候我们再加一个安全二级的策略，90 天强制修改密码。

这样子我们的密码跑只能够三个月内要跑完了，很显然这样跑又不存储肯定是不行的。要不我们学一下 cmd5 进行存储。假设我的空间够存所有密码，则我们把遍历的记录先存储进库里面。把常用的几个种加密方式都存进去。有一天，我们终于把密码都存进去，加上搜索引擎基本秒杀。这时候如果拖到密码库的话基本就全沦陷了。

为了防止这种情况发生，我们的密码存储再往上一个安全等级进行配置，第一，使用双层加密，第二使用 salt，第三，使用了向量配置。

简单来说，首先先用类似 md5 不可逆的加密方式把明文密码+随机数一起加密，随机数存在其他栏位中，再用一个类似可逆的配置向量的方式进行二次加密，设计如下：

finalPassowrd = des(md5(yourpassowrd + salt))>>v

你的向量可以随机可以不随机，但是我还是建议不随机，因为你随机了你还要存储库，按照设计的习惯，还不如跟 salt 一样，把向量存在同一表当中，这时候我还是鼓励你存储在代码配置中，因为蓝队攻击者能拿你的库，不一定拿到你的代码，除非是 getshell 了。

假设攻击者在这里跟你杠上了。。。。。

有一个情景攻击者还是拿到了你的源码，能否爆破呢？还是可以。这时候需要按照你的源码进行走读，把你的代码换成 c 的源码作为加密方式的一个库嵌入 hashcat，再进行爆破的话，还是在 3 个月内能够爆破出一定数量的明文密码来。

还有没有办法降低攻击者爆破的速度呢？有，在加密方式上。举个例子，你注册，修改密码，登陆用 1~2 秒的时间，其实作为客户的你还是能接受的，但是作为爆破方就是灾难了。所以可以学一些系统，自己设计多一层加密方式，这套加密方式基本就是不断在重复加密，例如把 3des 写成 1000des 那样，当你的团队有一个设计密码的高手可以这么玩。这样就算爆破再快也要在加密的速度那里进行了限速。

当然，作为一个普通的安全工程师，我一没资源二没时间，如果我 getshell 后怎么破密码呢？

我有点骚，一般不爆破。你有普通 shell 权限的话，改了对方的代码，在你的代码进来的那一刻，先把明文密码和对应的 id 写在本地的文件中。做好之后我就等了 1 个星期回去下这个文件，就拿到了并不少密码了。

这种手段千千万万，那是在不行，你如果不是点对点的攻击的话，我就在前端，你密码提交的那一刻，就用不可逆的加密进行了加密，再传到后台。可以是可以，不过我有办法改你的后端代码，也能改你的前端代码，基本上大部分人的源码还是放在同一个权限的文件夹当中的。

所以我们使用安全需求中最常用的一个需求，二次认证，加上短信验证码，或者说使用动态密码。这样就很安全了把，虽说还是有方法绕，但是成本以及非常高，这里已经做到非常安全了，作为服务器来说已经达标了。

最后总结一个存储型密码的防御设计安全建议：

1.前端生成密码后进行不可逆的加密方式进行加密。

2.到了后台进行格式加密密码校验。

3.进行二次加密，使用上 salt 和向量，并尽量加入自己设计的算法。

4.密码 90 天强制修改，并不能与历史密码相同。

5.验证的时候加短信验证码或者生成动态密码方式登陆。