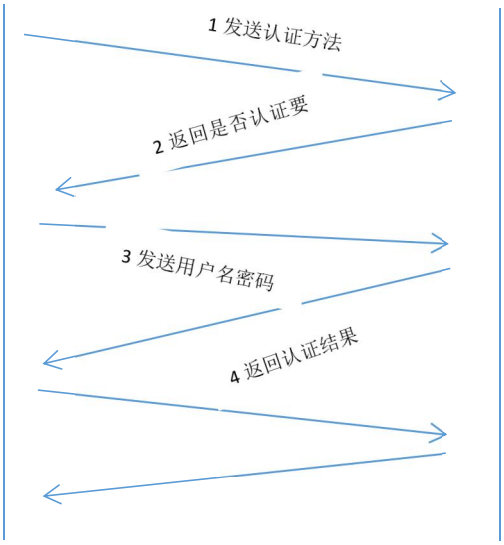
前言：

Socks5 作为一种代理协议，在连接的初期需要经过几次握手以交换必要的信息，在握手协议中有要求认证和无需认证方式，要求认证时，客户端需要上送用户名字密码，服务端验证通过之后才能进入后续的通信，一般用户名密码都是预配置在客户端和服务端，并且客户端是以明文的方式上送，因此存在泄漏风险，通过简单的线路抓包即可获取用户名密码，本文讨论的是如何在最小修改标准协议的基础上实现密码的保护。

Socks5 的握手协议（要求认证）



Socks5 的需要用户密码的认证过程如上所示（需要认证时三来三回，如果不需要认证只需二来二回），一般上述的握手过程的报文为(hex 格式)：

1. [05 02 00 02]
2. [05 02]
3. [01 07 7A 68 61 6E 67 7A 68 03 31 32 33]
4. [01 00]
5. [05 01 00 01 DC B5 26 94 01 BB]
6. [05 00 00 01 7F 00 00 01 04 38]

用户名密码包含在第三个报文报文里面，该报文格式为：

[01] 认证子协议版本

[07] 用户名长度，后面跟用户名

[7A 68 61 6E 67 7A 68] 用户名

[03] 密码长度，后面跟密码

[31 32 33] 密码

从中可以看出，用户名密码都是明文传输，没有任何保护，有人说可以对密码进行加密，那问题来了，加密用什么密钥，那这个密钥又通过什么方式分发呢？

所谓密码泄露的本质就是外界在抓取相关信息之后可以使用这些信息进行重放以伪装成合

法的用户，因此为了避免重放，要求对密码做转换，并且每次转换的结果都要求不一样，不一样的因子由服务端决定，基于这些原则，我们修改协议如下：

服务端在下发报文 2 的时候，同时下发一串随机数，客户端使用该随机数和密码做运算得出结果 passwd_cli，使用该运算结果作为报文 3 的密码域上送，服务端使用已下发的随机数和预置的密码做运算得出结果 passwd_srv，然后比对 passwd_cli 和 passwd_srv，如果两者一样，则验证通过，反之则拒绝服务。

由于每次下发的随机数都是不一样的，所以每次上送的 passwd_cli 都是不一样的，从而解决了报文被重放的问题，而且密码域是随机数和真实密码的运算结果，因此也解决了密码泄露的问题。

该方法的本质其实就是，在认证的时候不是直接交换密码，而是由认证方提供一段【信息】被认证方使用双方预置的密码作为密钥对信息进行加密，由于只有持有密码的人才能计算正确的结果，所以只要被认证方提供的结果和认证方计算的结果是匹配的，则可以认为被认证方是合法的，通过这种方式可以防止密码在线路上传输，而且只要认证方每次提供的【信息】都是不一样的，那么就可以防止报文截取后重放。

修改之后的握手过程如下：

7. [05 02 00 02]
8. [05 02 随机数长度 + 随机数]
9. [01 07 7A 68 61 6E 67 7A 68 03 + 用户密码和随机数的运算结果]
10. [01 00]
11. [05 01 00 01 DC B5 26 94 01 BB]
12. [05 00 00 01 7F 00 00 01 04 38]

根据设计我们的代码实现如下：

1. 在 socks.c 中，下发第二个报文的时候增加如下处理，生成一段随机数 append 到原有报文的后面，随机数保存在 salt 中

```
//Append salt string to the tail of packet
memset(param->salt, 0, sizeof(param->salt));
genRandomStr(param->salt, 8);
D_LOG(HEX_FORMAT, param->salt, 8);
buf[2] = 8; //len field
memcpy(buf + 3, param->salt, 8);
if(socksend(param->clisock, buf, 2 + 9 , conf.timeouts[STRING_S])!=(2 + 9)){RETURN(401);}
//if(socksend(param->clisock, buf, 2, conf.timeouts[STRING_S])!=(2)){RETURN(401);}
D_LOG(HEX_FORMAT, buf, 2 + 9);
```

2. TCPTunnel.h 中，在下发的第二个报文的解析中增加如下处理，获取上个步骤中下发的随机数，和密码运算后的结果作为密码域在第三个报文中上送，运算方法由函数 PasswdEncrypt 决定，只要客户端和服务端对该函数的定义相同即可。

```
//注意：由于标准的sock5协议返回的数据只有两个字节，因此如果是此非标客户端
//不然会有内存越界问题，导致客户端崩溃重启
int  iSaltLen;
char szSalt[8 + 1];
memset(szSalt, 0, sizeof(szSalt));
iSaltLen = ((char*)pr)[2];
memcpy(szSalt, (char*)pr + 3, iSaltLen);

//对已有的用户密码进行异或加密
char szTemp[50];
memset(szTemp, 0, sizeof(szTemp));
strcpy(szTemp, pd->userPassword.c_str());
PasswdEncrypt(szTemp, strlen(szTemp), szSalt, iSaltLen);
string sPasswdEncrypt = szTemp;
```

3. 3proxy auth.c 中，获取到用户上送的用户密码之后，同样在服务端使用下发的随机数 salt 和预置的密码进行同样的运算，并将运算结果和客户端上送的结果进行比较。

```
memset(password_encrypt, 0, sizeof(password_encrypt));
strcpy(password_encrypt, (char *)pwl->password);
PasswdEncrypt(password_encrypt, strlen(password_encrypt), (char *)param->salt, strlen((char *)param->salt));
D_LOG("password_encrypt calc by system with uername[%s]:", (char *)pwl->user);
D_LOG(HEX_FORMAT, password_encrypt, strlen(password_encrypt));

if(!strcmp((char *)pwl->user, (char *)param->username)) switch(pwl->pwtype) {
    case CL:
        if(!pwl->password || !*pwl->password){
            break;
        }
        else if (!param->pwtype && param->password && !strcmp((char *)param->password, (char *)password_encrypt)){
            D_LOG("password verify correct!");
            break;
        }
```