



Sommaire

I. La sécurité des mots de passe sous Linux : la fonction crypt.....	1
a) Description.....	1
b) Types de cryptages.....	1
c) Prototype C.....	2
II. De la sécurité des mots de passe.....	2
III. Les mots de passes sous UNIX.....	2
IV. Utilisation de la fonction crypt.....	3
a) Encryptage de mot de passe à la Unix.....	3
b) Un cracker de mot de passe bruteforce	4
V. Méthode par dictionnaires.....	7
VI. Conclusion.....	7

I. La sécurité des mots de passe sous Linux : la fonction crypt

a) Description

La fonction `crypt` permet de crypter les mots de passe sous Linux.

La fonction `crypt` prend deux paramètres :

- `clé` : la donnée à crypter
- `salt` : un morceau de texte aléatoire servant à compliquer l'encryption. Ceci permet de ne « jamais » avoir le même résultat crypté pour un même mot de passe (juste en changeant ce paramètre).

Elle renvoie un pointeur vers la chaîne cryptée. Elle commence par le paramètre `salt` puis le texte crypté.

b) Types de cryptages

Il existe deux types de cryptage :

- DES (Data Encryption Standard, une ancienne méthode de hachage) : c'est la méthode traditionnelle . Pour utiliser cette méthode, il faut que `salt` ait soit une chaîne de deux caractères de la plage `[a-zA-Z0-9./]`. `Crypt` renvoie une chaîne commençant par `salt` suivie de 11 caractères représentant le mot de passe crypté.
- MD5 (méthode plus robuste) : cette méthode a été introduite par GNU. Pour utiliser cette méthode, il faut que `salt` ait le format suivant : « `1<chaîne de 8 caractères>$` ». `Crypt` sort alors une chaîne commençant par le `salt` suivie de 22

caractères (représentant le mot de passe crypté) de la plage [a-zA-Z0-9./].

c) Prototype C

Le prototype C de crypt est le suivant :

```
#include <unistd.h>
char *crypt (const char *clé, const char *salt);
```

Pour compiler, il faut utiliser l'option `-lcrypt`.

II. De la sécurité des mots de passe

Les mots de passes ne doivent pas comporter d'informations qui peuvent être devinées, comme des *noms propres ou communs*, des *dates* ou des *informations personnelles que l'on peut connaître*. Ce genre de mots de passe est très facilement trouvable par attaque par dictionnaires.

Il est donc préférable de choisir un mot de passe aléatoire, comme e34dv4s5g, certes moins facile à retenir mais beaucoup plus long à trouver. En effet, un bon mot de passe se caractérise par le fait que si un pirate arrive à trouver le mot de passe, il ne sera plus utilisé. Il ressort donc qu'**il faut changer régulièrement de mots de passe**, suffisamment pour ne pas laisser le temps à un pirate de trouver le mot de passe. De tels mots de passe ne permettent pas d'attaques par dictionnaires mais des attaques par *bruteforce* : on doit tester tous les mots de passes du domaine de caractères du mot de passe (au moins [a-zA-Z0-9]).

Il est aussi utile de tester la validité des mots de passe par `passwd` ou d'autres programmes capables de juger les mots de passes.

III. Les mots de passes sous UNIX

Au début, les mots de passes sous UNIX étaient stockés dans le *deuxième champ* de chaque ligne de `/etc/passwd`, crypté en *DES*. Ce fichier appartenant à *root* avait le droit de lecture pour tout le monde et le droit d'écriture seulement pour *root*. Ainsi, tout le monde pouvait voir les mots de passe cryptés.

Mais avec les problèmes de sécurité grandissants, il a fallu mettre les mots de passe cryptés ailleurs, dans le fichier `/etc/shadow`. Ce fichier est lisible uniquement par *root*. Pour modifier son mot de passe, l'utilisateur utilise `passwd` qui se `set-uid` à *root* pour pouvoir écrire dans ce fichier. Dans `/etc/passwd`, le deuxième champ est remplacé par `*` pour indiquer que le mot de passe est dans `/etc/shadow`.

Chaque ligne de ce fichier contient le nom de login, son mot de passe et des informations sur la péremption de ce mot de passe. Les mots de passe sont maintenant cryptés en MD5.

Note : il est évident que *root* a le droit de lire et d'écrire dans un fichier dont il n'a pas de droits réels. Par exemple, il a le droit d'écrire dans le fichier `/etc/shadow` bien que personne n'a le droit `w` sur ce fichier.

IV. Utilisation de la fonction crypt

a) Encryptage de mot de passe à la Unix

Le fonctionnement est le suivant :

- on génère un salt aléatoirement avec la fonction rand sous la forme : \$1\$[a-zA-Z0-9./]{8}\$
- on encrypte le mot de passe avec le salt avec la fonction crypt

encrypt.c :

```
#define _XOPEN_SOURCE_
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

char *crypt (const char *clef, const char *salt);

/*****
/* Cette fonction génère aléatoirement */
/* un salt pour hachage MD5          */
/* ===== */
/* buf : tampon où stocker le salt    */
/*****
void generate_md5_salt(char buf[13])
{
int i;

//on initialise le générateur de nombre aléatoire
srand(time(NULL));

//un salt pour MD5 a le format : $1$<chaine de 8 caractères>$
//les huit caractères sont à choisir parmi [a-zA-Z0-9./]
buf[0]='$';
buf[1]='1';
buf[2]='$';

for (i=3; i < 11;i++)
{
do{
buf[i]=rand() % 128;
}while(
(buf[i] < 'a' || buf[i] > 'z')
&& (buf[i] < 'A' || buf[i] > 'Z')
&& (buf[i] < '0' || buf[i] > '9')
&& (buf[i] != '.')
&& (buf[i] != '/')
);
};
```

```

}

buf[11]='$';
buf[12]='\0';
}

int main ()
{
char salt[13];
char buff[256];
generate_md5_salt(salt);

printf("Entrez le texte à crypter :");
scanf("%s",buff);
printf("Le salt est %s\nLe texte crypté est
%s\n",salt, crypt(buff,salt));

return 0;
}

```

```
[user]$ gcc -o encrypt -lcrypt encrypt.c
```

b) Un cracker de mot de passe bruteforce

Attention : je ne serais être tenu responsable de l'utilisation que vous en faites.

Le fonctionnement est le suivant :

- pour chaque taille de mot de passe possible (en partant de 1 et en augmentant)
 - pour chaque combinaison de lettre de la plage [a-zA-Z0-9] de la longueur en cours
 - on crypte le mot de passe avec le salt de celui à découvrir
 - on compare les deux hash
 - s'ils sont égaux, on a trouvé le mot de passe
 - sinon, on continue

C'est une méthode très lente pour les mots de passes de plus de 6 caractères de cette plage (il y a tout de même 64^6 combinaisons). Dans la réalité, il peut y avoir encore plus de caractères dans la plage de test ce qui a pour effet de faire augmenter le temps de calcul de manière exponentielle.

decrypt.c :

```

#define _XOPEN_SOURCE_
#include <unistd.h>
#include <stdio.h>
#include <string.h>
#include <malloc.h>

char *crypt (const char *clef, const char *salt);

//hash crypté du mot de passe
char pass[256]={0};
//salt du mot de passe crypté
char *salt;

```

```

/*****
/* Cette fonction teste tous les mots de passe      */
/* de taille taille et ayant les caractères        */
/* [a-zA-Z0-9]                                     */
/* ===== */
/* buff : tampon contenant le mot de passe         */
/* courant en train d'être construit et testé     */
/* taille : taille du mot de passe à rechercher   */
/* pos : position de la lettre en train de changer*/
/* du mot de passe courant à tester              */
*****/
int brute(char* buff,int taille,int pos)
{
    //quand taille arrive à 0 : on a construit tout le mot de
    passe à tester
    if (taille==0)
    {
        //on peut compare le hash du mot de passe construit avec
        celui que l'on cherche
        char *p=(char*)crypt(buff,salt);
        if (strcmp(pass,p)==0){
            //si on a trouvé le bon mot de passe
            return 1;
        }
        //sinon, on continue
        return 0;
    }
    else
    {
        int i;
        //on fait varier chaque lettre du mot de passe parmi [a-
        zA-Z0-9]

        for (i='a';i <= 'z';i++){
            buff[pos]=i;
            //on fait varier récursivement la lettre suivante
            jusqu'à la fin du mot
            if(brute(buff,taille-1,pos+1)==1) return 1;
        }
        for (i='A';i <= 'Z';i++){
            buff[pos]=i;
            //on fait varier récursivement la lettre suivante
            jusqu'à la fin du mot
            if(brute(buff,taille-1,pos+1)==1) return 1;
        }
        for (i='0';i <= '9';i++){
            buff[pos]=i;
            //on fait varier récursivement la lettre suivante
            jusqu'à la fin du mot
            if(brute(buff,taille-1,pos+1)==1) return 1;
        }
        return 0;
    }
}

```

```

    }
}

/*****
/* cette fonction extrait le salt du mot de passe crypté */
/* ===== */
/* hash : hash MD5 ou DES du mot de passe */
/* renvoie le salt dans un buffer alloué avec malloc */
*****/
char *extract_salt(char *hash)
{
    //buffer de retour
    char *ret;
    //taille du salt
    int len;

    //si mot de passe MD5
    if (strstr(hash,"$1$")==hash)
    {
        //"$1$chaine$"
        len=13;
    }
    //si mot de passe DES
    else
    {
        //deux caractères
        len=3;
    }
    //on alloue un buffer pour le salt
    ret=(char*)malloc(len*sizeof(char));
    //on copie le salt et on met un \0 à la fin
    strncpy(ret,hash,len-1)[len-1]='\0';

    return ret;
}

//ce programme tente de décrypter un mot de passe MD5 par
bruteforce
int main ()
{
    //tampon pour contenir les mots de passe testés
    char buff[10]={0};
    //longueur du mot de passe à rechercher
    int i;

    //on demande le hash du pass
    printf("Entrez le hash du mot de passe :\n");
    scanf("%s",pass);
    //on extrait le salt
    salt=extract_salt(pass);

    //on cherche un mot de passe de 1 à 10 caractères
    for (i=1;i<=10;i++)

```

```

    {
        printf("Passage à des mots de passe de taille %d\n",i);
        //si la fonction renvoie 1 alors on a trouvé le mot de
passe
        if (brute(buff,i,0)==1)
        {
            printf("%s\n",buff);
            free(salt);
            return 0;
        }
    }

    free(salt);

    return 0;
}

```

```
[user]$ gcc -o decrypt -lcrypt decrypt.c
```

V. Méthode par dictionnaires

Contrairement à la méthode par force brute, la méthode par dictionnaire consiste à partir d'une liste de mots à former des combinaisons de ces mots (ou les tester seuls) puis de les crypter et de comparer.

Cette méthode est beaucoup plus rapide que la brute force car le dictionnaire est toujours beaucoup plus petit que l'ensemble des combinaisons de la plage de caractères du mot de passe. Elle marche aussi souvent très bien avec les utilisateurs qui ne savent pas que « napoleon » n'est pas un bon mot de passe. Par contre, sur un mot de passe comme « a12g5ef5 », cette méthode est inefficace.

VI. Conclusion

Pour conclure, je dirais qu'il est toujours préférable de choisir un mot de passe comme « 4k6z1jio » que « toto ». De plus, pour pouvoir tenter une quelconque méthode de décryptage de mot de passe, il faut encore avoir le droit de lire le fichier dans lequel se trouve le mot de passe, par exemple `/etc/shadow`. Ceci n'est souvent possible qu'en tant que root ou au moins avoir un accès au système du fichier de la machine donc il faut un moyen de devenir root ou d'accès. Le quasi seul moyen est une faille logiciel.

On peut se protéger des autres utilisateurs par un mot de passe compliqué et des failles avec les mises à jour de sécurité.