LISV Laboratoire d'ingénierie des systèmes de Versailles
UVSQ université PARIS-SACLAY

# La SAÉ du Hacker

Halim Djerroud

révision 0.1

### Note

Ce projet est à réaliser dans le cadre de la SAÉ suivante :
— SAÉ.Cyber.03 - Administrer et Surveiller un système d'information sécurisé.

Compétences criblées
— Surveiller un système sécurisé.
— Développer un système sécurisé.

## 1 Introduction

Cette SAÉ, vise à obtenir des compétences permettant de comprendre le fonctionnement bas niveau d'un programme informatique. L'assembleur est généralement utilisé dans deux cas spécifiques :
— S'affranchir des compilateurs afin de produire des programmes rapides et moins gourmands en mémoire.
— Comprendre le fonctionnement des programmes, dont le code source n'est pas fourni (ingénierie inverse). C'est souvent la seule solution que nous avons pour comprendre et contrecarrer les programmes malveillants.

Dans cette SAÉ, les deux compétences sont traitées à travers deux minis projets distincts. Le primer consiste à récrire quelques fonctions de la `libc`. Le second consiste à résoudre des challenges de *CrackMeS*.

### 1.1 Formation des équipes

Des équipes de 3 ou 4 étudiants doivent être formés (les monômes, binômes, équipe de 5 ... ne sont pas acceptés). Il faut obligatoirement choisir un groupe et inscrire son nom et prénom sur le fichier suivant : https://lite.framacalc.org/groupes_sae_asm_uaesj6edcj-a2sb

### 1.2 Rendu attendu

— Former son équipe et fournir un planning prévisionnel du projet, à rendre la première semaine, pour le reste la date de rendu vous sera communiquée ultérieurement.
— Un lien vers l'archive demandée.
— Une démo fonctionnelle à montrer lors de la soutenance.
— Un document PDF d'environ 20 à 30 pages, de préférence rédigé en LaTeX(pas obligatoire), montrant votre démarche dans les deux parties.
— La démo (20 min max) sera présentée lors de la soutenance suivie d'une session de question/réponse (10 à 15 min). Pas de diapositives pour ce projet.

# Table des matières

# Première partie

# .::Super libc::.

## 1 Présentation

Dans ce projet nous souhaitions proposer une minie bibliothèque C qui implémente quelques fonctions de la *libc* en assembleur, sous le nom *s_libc* pour (super libc). Le but étant de proposer des fonctions très rapides et moins gourmandes en mémoire.

Les bibliothèques Linux sont proposées sous deux formats, en statique avec l'extension (`.a`) et en dynamique (ou partagée) avec l'extension (`.so`). Les bibliothèques sont composées d'un ensemble de fonctions qui peuvent être sur un seul, ou réparti sur plusieurs fichiers objet, regroupés dans une archive. Pour qu'une bibliothèque soit correctement utilisée par un développeur, il faut aussi fournir les fichiers d'en-tête *(headers)* correspondant, fichiers avec l'extension `(.h)` qui donne les prototypes des fonctions de la bibliothèque.

Une bibliothèque n'a pas de fonction main, car elle a pour objectif d'être (ré)utilisée par d'autres programmes. Le nom d'une bibliothèque doit obligatoirement commencer par `lib` suivi de son nom et de son extension. Par exemple dans notre cas, la bibliothèque doit s'appeler `libs_libc.so`. Les programmes utilisateurs qui souhaitent l'utiliser doivent ajouter lors de la compilation l'option `-l`, suivi du non de la bibliothèque (sans extension). Dans notre cas, l'option `gcc` correspondante est la suivante : `-ls_libc`. Il faut omettre le préfixe `lib` et l'extension *.so*. Ces options sont uniquement valables pour le compilateur `gcc`.

Dans ce projet, nous souhaitons implanter les fonctions données ci-après. Les fonctions sont regroupées dans des fichiers selon la même répartition que celles de la `libc`. Pour obtenir les informations sur le fonctionnement d'une fonction, il suffit de consulter sa page de manuel (section 3) avec la commande suivante `man 3` *`nom_de_la_fonction`*.

Seules les fonctions notées avec une seule étoile (fonctions faciles) sont obligatoires. Les restes des fonctions sont facultatives. Attention, l'implémentation des fonctions obligatoires uniquement ne conduit pas systématiquement à l'obtention de la note complète. Les fonctions sont réparties selon la convention suivante :

- `*` fonctions faciles (obligatoire)
- `**` fonctions moyennement faciles (souhaitables)
- `***` fonctions moyennement difficiles (bonus)

- `s_string.h`
  - `s_strcpy (*)`
  - `s_strncpy (*)`
  - `s_strcat (*)`
  - `s_strncat (*)`
  - `s_strlen (*)`
  - `s_strcmp (*)`
  - `s_strncmp (*)`
  - `s_strchr (*)`
- `s_math.h`
  - `s_abs (*)`
  - `s_div (*)`
  - `s_pow (*)`
  - `s_exp (**)`
- `s_stdio.h`
  - `s_puts (*)`
  - `s_fopen (***)`
  - `s_fread (***)`
  - `s_fwrite(***)`
  - `s_fclose(***)`
- `s_stdlib.h`

- `s_abs (*)`
- `s_atoi (**)`
- `s_exit (***)`

## 1.1 Rendu attendu

Vous devez rendre une archive `tar.gz` correctement nommée `s_libc_project_[num_groupe]_[num_equipe]`. Elle doit contenir les éléments suivantes :

- Un répertoire `src` qui contient les fichiers assembleur que vous avez créés par vos soins.
- Un répertoire `include` qui contenant les fichiers entête.
- Un répertoire `lib` qui contient la bibliothèque compilée.
- Un répertoire `test` qui contient un fichier `main.c`, ce fichier doit contenir le test de toutes les fonctions écrites dans votre bibliothèque.
- Un répertoire `build` qui doit être vide, de dossier doit servir à contenir les fichiers compilation temporaire générée lors de la compilation. Avant de rendre votre projet vous devez vider le contenu de ce dossier.
- Un répertoire `doc` qui doit contenir la documentation de vos fonctions, de préférence des pages `man`, ce qui vous permettra de vous inspirer des pages de manuel de `libc`. Vous pouvez aussi genrer votre documentation avec des outils automatique tel que doxygen, ou-bien proposer tout simplement une autre documentation fichiers textes, html, ...
- Un fichier `makefile` fonctionnel qui doit contenir les au minimum les entrées suivantes :
  - *all* : gérer la bibliothèque
  - *test* : gérer l'exécutable du fichier `main.c`
  - *clean* : supprimer les fichiers temporaires du répertoire `build`
  - *install* : copier la librairie générée dans le répertoire des librairies du système.
    Pour créer votre makefile suivez le tutoriel sur la page suivante https://gl.developpez.com/tutoriel/outil/makefile/
- Un fichier *README*, qui doit contenir une description de votre projet, comment l'utiliser, l'installer..., les auteurs (les étudiants de l'équipe), la licence et toute autres informations utiles.

L'arborescence de votre projet doit ressembler au schéma suivant :

```
|---s_libc_project_[num_groupe]_[num_equipe]/
    |
    |---src/
    |   |--- s_string.s
    |   |--- s_math.s
    |   |--- s_stdio.s
    |   |--- s_stdlib.s
    |
    |---include/
    |   |--- s_string.h
    |   |--- s_math.h
    |   |--- s_stdio.h
    |   |--- s_stdlib.h
    |
    |---lib/
    |   |--- libs_libc.so
    |
    |---test/
    |   |--- main.c
    |
    |---build/
    |   |---
    |
    |---doc/
    |   |--- ...
    |
    |---makefie
    |---README
```

# 2 Exemple

Le fichier *s_string.s*

```
  .text
  .global s_strlen

s_strlen:
    pushl   %ebp
    movl    %esp,           %ebp
    movl    8(%ebp),        %esi # @ de début de la chaine
    xorl    %eax,           %eax
sum_loop:
    movb    (%esi,%eax),    %dl
    cmpb    $0,             %dl
    jz      end_sum_loop
    incl    %eax
    jmp     sum_loop
end_sum_loop:
    pop     %ebp
    ret
```

Le fichier *s_string.h*

```
extern int s_strlen(char*);
```

Le fichier *main.c*

```
#include <stdio.h>
#include "s_string.h"

int main() {
  char *str = "Hello";
  int len = s_strlen(str);
  printf("Len de str = %d \n", len);
  return 0;
}
```

## 2.1 Tester

```
gcc -m32  main.c s_string.s -o my_prog
```

*Attention : il est possible que vous receviez le warning* `"absence de la section .note.GNU-stack"`*, vous pouvez tout simplement ignorer cette alerte.*

Pour tester votre programme :

```
./my_prog
Len de str = 5
```

## 2.2 Générer une bibliothèque partagée

Génération du fichier objet

```
gcc -m32  -c s_string.s -o s_string.o
```

Compilation de la bibliothèque

```
gcc -m32  -shared -o s_libc.so s_string.o
```

Compilation du programme

### 2.2 *Générer une bibliothèque partagée*

```
gcc -m32 -o my_prog  main.c -L. -ls_libc
```

Pour tester votre programme :

```
./my_prog
Len de str = 5
```

Deuxième partie

# . : : Désassemblage : : .

## 1 Introduction

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.

### 1.1 Travail demandé

Pour chaque challenge, vous devez donner.
- Le.s login.s et mot.s de passe qui fonctionne.ent.
- Expliquer votre démarche pour résoudre le challenge.
- Récrire le code original en C.

## 2 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 **
```

### 2.1 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*.

### 2.2 Challenge 3

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

## 3 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 :

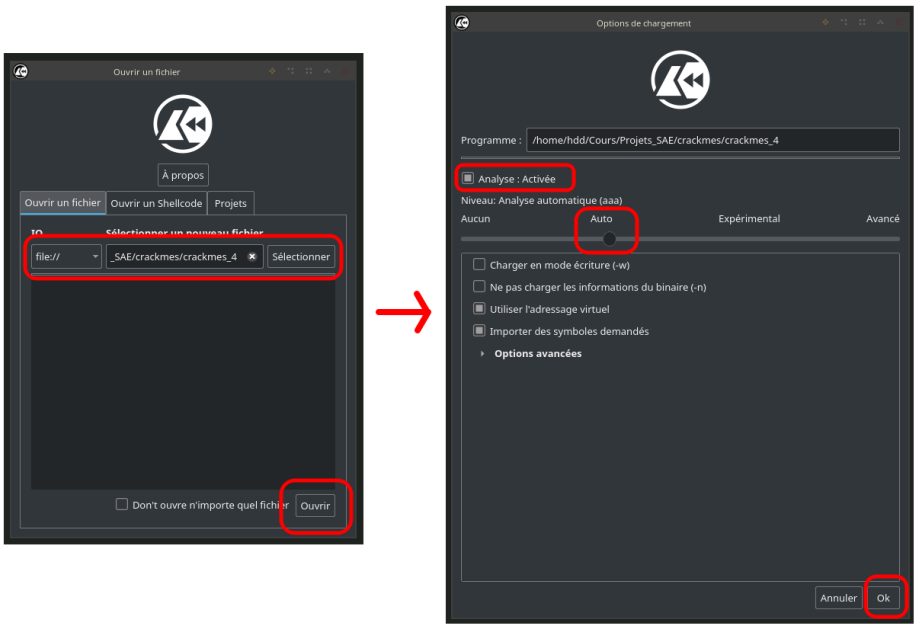
```
$ ./crackmes_4 test testpass
 ----------------------
 ----- CRACKMES 4 -----
 ----------------------

Incorrect login type
```

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

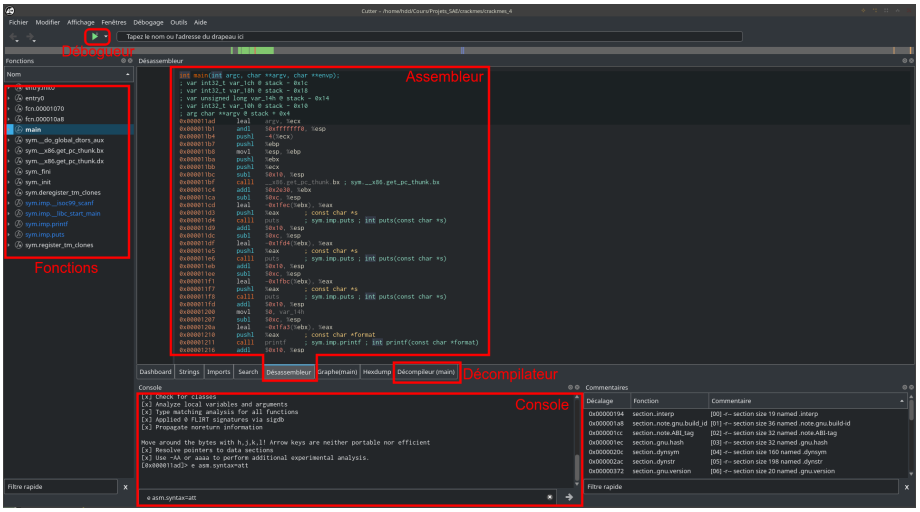
```
$ ./crackmes_4 123 456
 ----------------------
 ----- CRACKMES 4 -----
 ----------------------

** 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 :

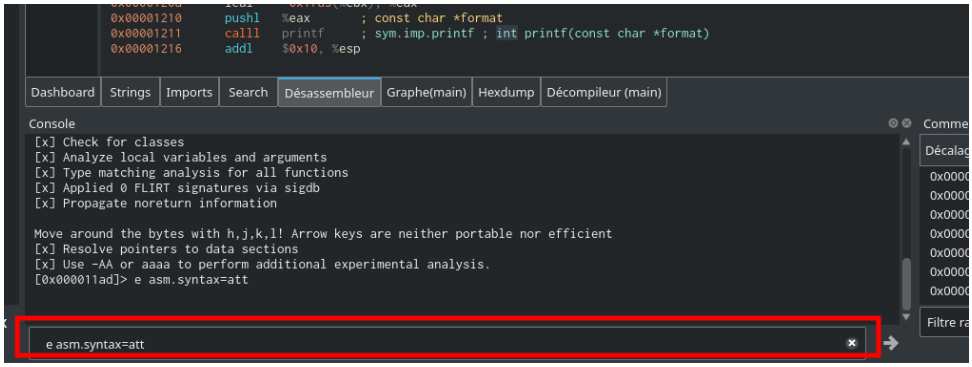



La fenêtre principale




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 :

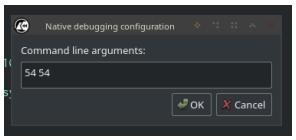
```
e asm.syntax=att
```




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.

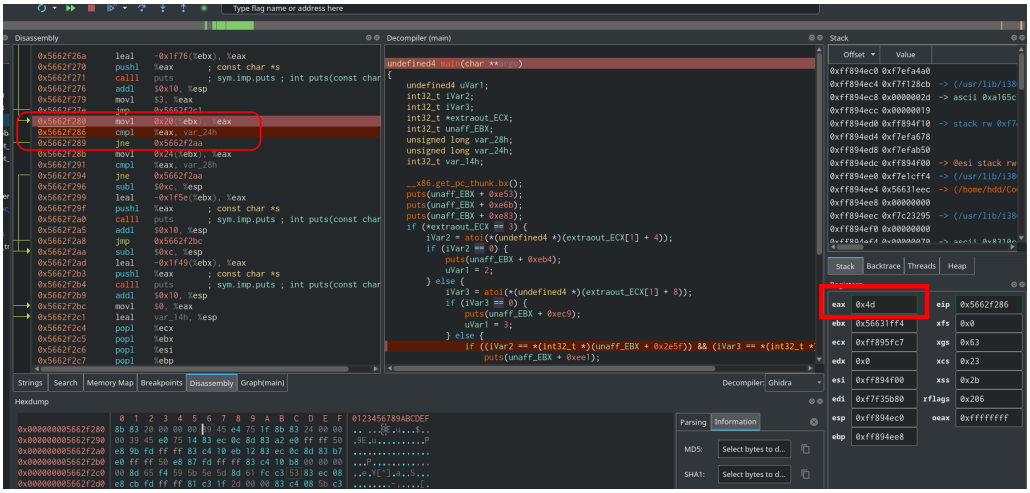
```
undefined4 main(char **argv)
{
    undefined4 uVar1;
    int32_t iVar2;
    int32_t iVar3;
    int32_t *extraout_ECX;
    int32_t unaff_EBX;
    unsigned long var_28h;
    unsigned long var_24h;
    int32_t var_14h;

    __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) :

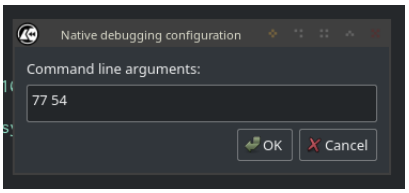



On remarque ici la valeur est du *login* est chargée dans `%eax` pour être comparé. Alors if suffi de consulter la valeur de ce registre : (`0x4d`) en hexadécimal ce qui fait 77 en décimale.

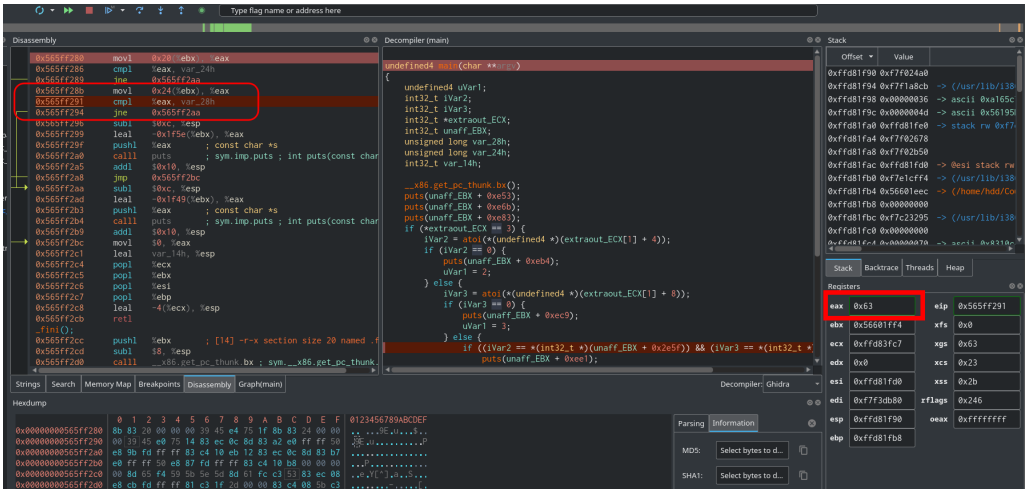
*3.1 Challenge 5*



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 **
```

## 3.1 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.*

# 4 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 ?

## 4.1 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.

## 4.2 Challenge 8

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

# 5 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"`.