RetroProjets (MEDIUM)

Auteur(s) :

`Hokanosekai`

Catégorie :

`Reverse`

Description :

Un programme informatique contenant tous les projets étudiants depuis 4 ans circule parmis les étudiants de l'université. Pour un projet vous souhaiteriez vous inspirer d'un des projets, mais le programme est protégé par une phrase secrète. Dépêchez vous de le récupérer avant que le rendu ne soit clos !

retro_projets

Hints

Hint 1

D'après les rumeurs, les étudiants qui ont conçu le programme n'ont pas bien suivi les cours de sécurité informatique, et ont utilisé un algorithme de chiffrement très faible.

Flag : `UHOCTF{fake_flag}`

Solution :

Nous avons donc en notre possession un éxécutable Linux, il suffit de l'exécuter pour voir qu'il nous demande un mot de passe.

```
hoka@hoka ~/c/U/U/R/RetroProjets (main)> ./retro_projets
Tu souhaite accéder aux projets ?
Entrez le mot de passe :
```

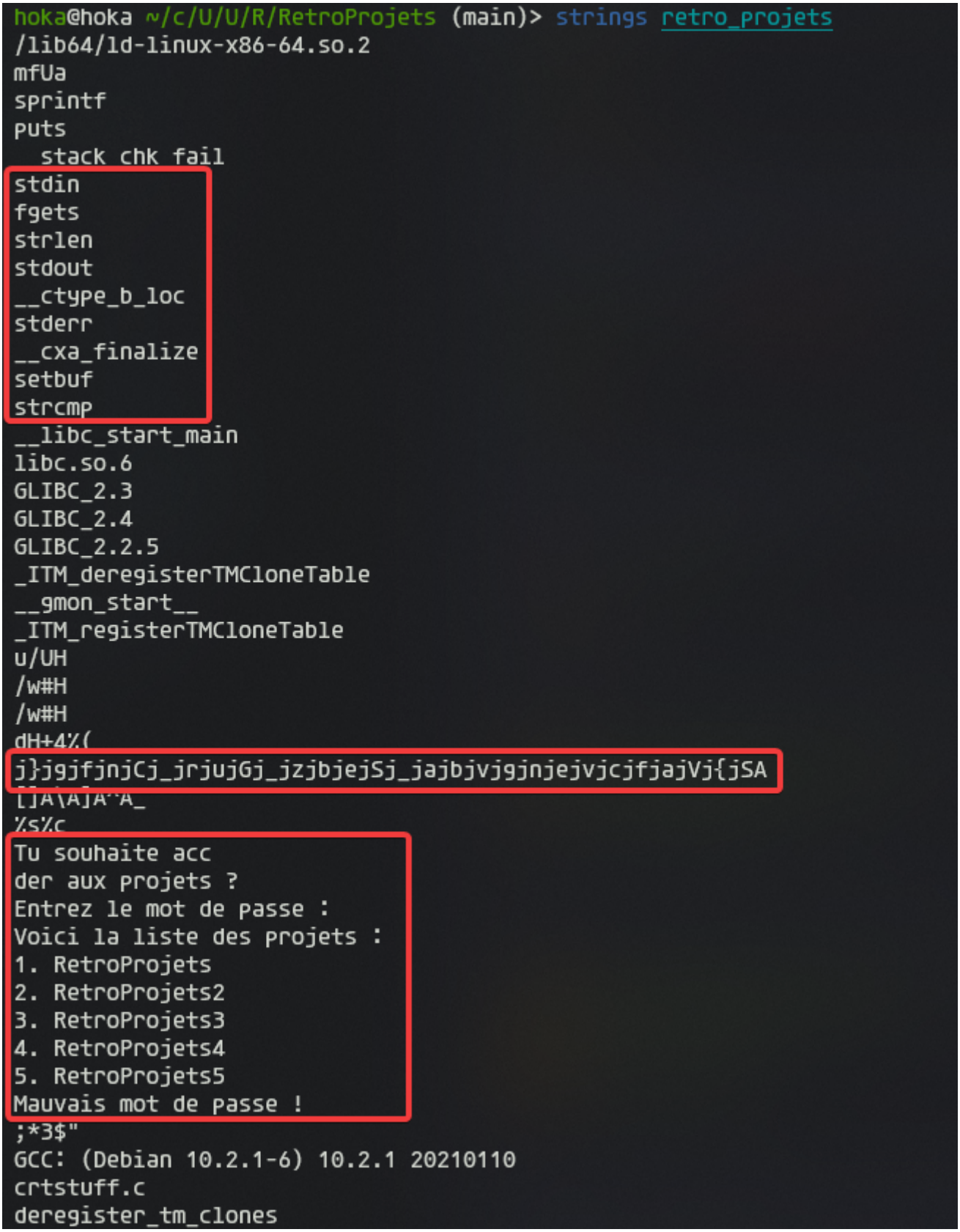
Si l'on rentre un mot de passe aléatoire, on obtient le message suivant :

```
hoka@hoka ~/c/U/U/R/RetroProjets (main)> ./retro_projets
Tu souhaite accéder aux projets ?
Entrez le mot de passe :
password
Mauvais mot de passe !
```

On peut donc en déduire que le mot de passe est vérifié en dur dans le programme.

Maintenant, il nous faut trouver cette phrase secrète, pour cela, on peut utiliser la commande `strings` qui permet d'afficher toutes les chaînes de caractères présentes dans un fichier.

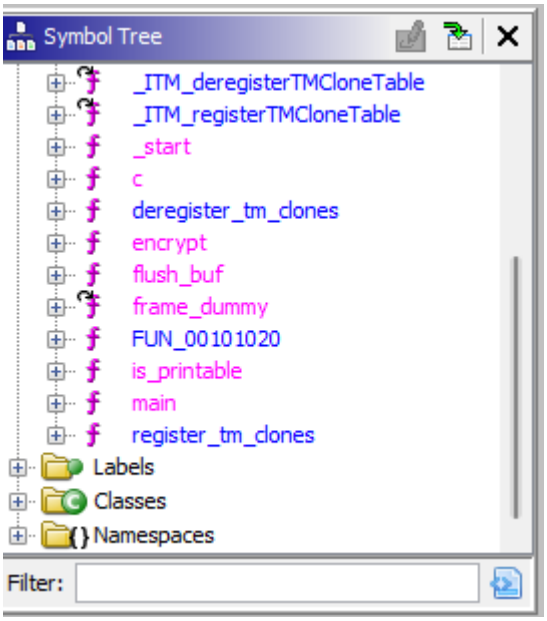
On peut donc constater le résultat suivant dans le début de la sortie :



On constate trois parties intéressantes, la première contient les différents imports du programme, la seconde ressemble à une chaîne de caractères encodée, enfin dans la troisième partie, on peut voir les différentes chaînes de caractères utilisées dans le programme.

On peut donc maintenant essayer de déassembler le programme avec le logiciel `Ghidra`.

Après avoir importer le programme dans `Ghidra`, et éxécuter un analyse de base, on peut se pencher sur la fenêtre `Symbol Tree` qui nous permet de voir les différentes fonctions du programme, les imports, les variables globales, etc.



On peut y constater une fonction `main` qui est la fonction principale du programme, et une fonction `encrypt` qui est appelée dans la fonction `main`.

```
undefined8 main(void)

{
  int iVar1;
  char *__s1;
  long in_FS_OFFSET;
  char local_68 [48];
  char local_38 [40];
  long local_10;

  local_10 = *(long *)(in_FS_OFFSET + 0x28);
  c(local_38,(void *)0x48);
  puts(&DAT_00102010);
  puts("Entrez le mot de passe :");
  iVar1 = 0x21;
  fgets(local_68,0x21,stdin);
  encrypt(local_68,iVar1);
  iVar1 = strcmp(__s1,local_38);
  if (iVar1 == 0) {
    puts("Voici la liste des projets :");
    puts("1. RetroProjets");
    puts("2. RetroProjets2");
```

```
    puts("3. RetroProjets3");
    puts("4. RetroProjets4");
    puts("5. RetroProjets5");
  }
  else {
    puts("Mauvais mot de passe !");
  }
  if (local_10 != *(long *)(in_FS_OFFSET + 0x28)) {
                    /* WARNING: Subroutine does not return */
    __stack_chk_fail();
  }
  return 0;
}
```

Dans la fonction `main` on peut retrouver les chaînes de caractères de la partie 3 de la commande `strings`, on peut donc en déduire que la chaîne de caractères encodée est décodée dans la fonction `encrypt`.

De plus on peut voir que la fonction `encrypt` prend en paramètre le mot de passe entré par l'utilisateur, et une valeur `0x21` qui est la taille du mot de passe.

Et enfin on peut voir l'appel de la fonction `strcmp` qui permet de comparer deux chaînes de caractères, on peut donc en déduire que la chaîne de caractères est comparée avec le mot de passe entré par l'utilisateur encodé.

Maintenant, il nous faut trouver comment est encodé le mot de passe, pour cela, on peut se pencher sur la fonction `encrypt`.

```
void encrypt(char *__block,int __edflag)

{
  long lVar1;
  ushort **ppuVar2;
  size_t sVar3;
  long in_FS_OFFSET;
  char local_25;
  int local_24;

  lVar1 = *(long *)(in_FS_OFFSET + 0x28);
  local_24 = 0;
  while( true ) {
    sVar3 = strlen(__block);
    if (sVar3 <= (ulong)(long)local_24) break;
    local_25 = __block[local_24];
    ppuVar2 = __ctype_b_loc();
    if (((*ppuVar2)[local_25] & 0x400) != 0) {
      ppuVar2 = __ctype_b_loc();
      if (((*ppuVar2)[local_25] & 0x100) == 0) {
        local_25 = (char)(local_25 + -0x54) + (char)((local_25 + -0x54) /
0x1a) * -0x1a + 'a';
      }
```

```
      else {
        local_25 = (char)(local_25 + -0x34) + (char)((local_25 + -0x34) /
0x1a) * -0x1a + 'A';
      }
    }
    __block[local_24] = local_25;
    local_24 = local_24 + 1;
  }
  if (lVar1 != *(long *)(in_FS_OFFSET + 0x28)) {
                    /* WARNING: Subroutine does not return */
    __stack_chk_fail();
  }
  return;
}
```

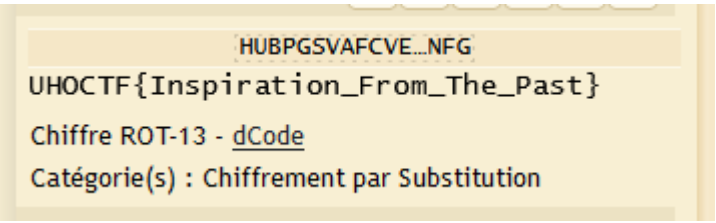
On peut voir que la fonction `encrypt` prend en paramètre le mot de passe entré par l'utilisateur, et une valeur `0x21` ou `33` en décimal qui doit donc être la taille du mot de passe.

On peut voir que la fonction `encrypt` parcourt chaque caractère du mot de passe, et effectue des opérations dessus.

Notamment, on peut voir que si le caractère est une lettre minuscule, on lui soustrait `0x54` ou `84` en décimal, puis on ajoute le résultat à `0x61` ou `97` en décimal, ce qui correspond au caractère `a` en ASCII.

De même, si le caractère est une lettre majuscule, on lui soustrait `0x34` ou `52` en décimal, puis on ajoute le résultat à `0x41` ou `65` en décimal, ce qui correspond au caractère `A` en ASCII.

On peut donc en déduire que le mot de passe est encodé en `ROT13`, on peut donc utiliser un site comme dcode pour le décoder.



Flag : `UHOCTF{Inspiration_From_The_Past}`

Commands

```
gcc -o retro_projets -fstack-protector-all ./src/source.c
```