Faculté des Sciences & Techniques — Université de Limoges

Licence 3ème année

Initiation Sécurité

TP n°3

Rétro-conception ou « Reverse engineering »

Création de « *challenge* »

1 – a. Écrivez dans un **langage compilé** (C, C++, Rust, Go) un programme :
- ▷ demandant un **mot de passe** à l'utilisateur ;
- ▷ vérifie si ce mot de passe est le bon, en le **comparant** à un mot de passe interne ;
- ▷ retourne un message de **succès** ou **d'échec**.

Vous utiliserez l'option de déboguage pour compiler :

xterm
```
$ gcc -g -o mon_programme mon_programme.c
```

b. Faites **différentes variantes** de votre programme, en combinant les différents éléments :
- ◇ avec lecture au clavier du mot de passe ;
- ◇ avec utilisation d'un argument pour passer votre mot de passe ;
- ◇ avec l'appel d'une fonction pour vérifier le mot de passe ;
- ◇ sans utilisation de fonction.

Analyse des programmes

2 – Analysez vos propositions à l'aide des commandes suivantes :

strings	affiche les séquences de caractères d'au plus 4 lettres de caractères affichables
ltrace	affiche les appels aux bibliothèques réalisés par le programme
strace	affiche les appels systèmes réalisés par le programme
nm	affiche la liste des symboles du programme❶

❶ Que se passe-t-il si vous compiler le programme sans l'option déboguage ?
Vous pouvez utiliser la commande `strip` pour la supprimer, voir au dos de la page.

Documentation pour `ltrace` et `strace`

`-p pid` ⟹ Attaches to the specified pid. *Useful if the program is already running and you want to know its behavior.*
`-n 2` ⟹ Indent each nested call by 2 spaces.
`-f` ⟹ Follow fork

Comparez vos différentes variantes de programme avec les informations fournies par ces outils.

3 – Interceptez les appels de bibliothèques :
- ☐ la variable d'environnement `LD_PRELOAD` permet d'ajouter une bibliothèque à l'exécution d'un programme ;
- ☐ la variable d'environnement `LD_LIBRARY_PATH` permet d'indiquer au système où se trouve les bibliothèques dynamiques à charger.

Soit le programme suivant que l'on veut observer :

```
#include <stdio.h>
int main() {
  char str1[]="TGS";
  char str2[]="tgs";
  if(strcmp(str1,str2)) {
    printf("String are not matched\n");
  }
  else {
    printf("Strings are matched\n");
  }
}
```

Et sa compilation :

xterm
```
$ gcc -o mon_prog mon_prog.c
```

Il va afficher : `String are not matched`

On veut maintenant intercepter les appels à la bibliothèque :

```
#include <stdio.h>
int strcmp(const char *s1,
    const char *s2)
{
  // Always return 0.
  return 0;
}
```

```
xterm
$ gcc -o mabibliotheque.so -shared bibliotheque.c -ldl
$ export LD_LIBRARY_PATH=./:$LD_LIBRARY_PATH
$ LD_PRELOAD=mabibliotheque.so ./mon_prog ❷
```

❷ *on définit la variable* `LD_PRELOAD` *uniquement pour l'exécution ponctuel de* `mon_prog`

L'exécution du programme précédent devrait donner `Strings are matched` parce que le programme utilise **votre version** de la fonction de bibliothèque `strcmp` !

Essayez sur vos programmes, en l'adaptant à vos fonctions de bibliothèques si vous en utilisez.

4 – Essayez d'analyser vos programmes avec Ghidra qui est installé dans mon compte : `/home/bonnep02/Public` :

```
xterm
$ cd /home/bonnep02/Public
$ cd ghidra_10.2_PUBLIC
$ ./ghidraRun
```

Puis vous indiquerez le chemin du JDK installé dans mon compte :

```
bonnep02@fst-o-i-212-02:~/Public$ cd
ghidra_10.2_PUBLIC/ jdk-17.0.5/
bonnep02@fst-o-i-212-02:~/Public$ cd ghidra_10.2_PUBLIC/
bonnep02@fst-o-i-212-02:~/Public/ghidra_10.2_PUBLIC$ ./ghidraRun
************************************************************
JDK 17+ (64-bit) could not be found and must be manually chosen!
************************************************************
Enter path to JDK home directory (ENTER for dialog): /home/bonnep02/Public/jdk-17.0.5
Saved changes to /home/bonnep02/.ghidra/.ghidra_10.2_PUBLIC/java_home.save
bonnep02@fst-o-i-212-02:~/Public/ghidra_10.2_PUBLIC$
```

a. Comparez la « *rétro-compilation* » de vos programmes réalisée par Ghidra avec les sources de vos programmes.
   Que pouvez vous en apprendre ?

b. Quelles « *méthodes* » marchent le mieux pour dissimuler les actions de vos programmes ?

Relever les « *challenges* »

5 – Vous **échangerez entre vous** vos programmes pour essayer de les « *casser* », c-à-d de trouver le mot de passe utilisé en étudiant l'exécutable.

Essayez de voir si vous trouvez le mot de passe de vos camarades...

Pour supprimer la « *table des symboles* »

La « *table des symboles* » mémorise dans votre programme les liens avec votre source : nom des fonctions, corps des fonctions. Cette table n'est pas toujours dans les programmes car elle prend de la place. Si vous voulez la **supprimer** il faut utiliser la commande `strip` :

```
xterm
$ strip mon_prog
```

Pour installer Ghidra sur votre machine personnelle :
Vous téléchargerez Ghidra :

```
xterm
$ wget https://github.com/NationalSecurityAgency/ghidra/releases/download/Ghi
dra_10.2_build/ghidra_10.2_PUBLIC_20221101.zip
$ unzip ghidra_10.2_PUBLIC_20221101.zip
$ cd ghidra_10.2_PUBLIC
$ ./ghidraRun
```

Attention

Il vous faudra probablement installer un JDK de version correcte pour pouvoir l'utiliser :

```
xterm
$ wget https://download.oracle.com/java/17/latest/jdk-17_linux-x64_bin.tar.gz
$ tar xvfz jdk-17_linux-x64_bin.tar.gz
```