UFR Math-Info
Master 2 SIRIS

Sécurité Avancée
CI 2

Strace

I) Introduction

strace permet de comprendre ce que fait un processus en analysant les appels système qu'il effectue. L'objectif principal est d'apprendre à filtrer de grandes quantités d'informations pour distinguer un comportement normal d'un comportement anormal.

II) Matériel

Télécharger la VM : https://seafile.unistra.fr/f/d21450923631420ca2e2/?dl=1

III) Principes

Une des compétences majeure dans le domaine de la cyber-sécurité est la capacité d'analyser un exécutable malicieux (malware). Cette compétence fait partie d'une pratique plus générale appelée rétro-ingénierie.

Cet exercice se focalise sur l'analyse dynamique d'un exécutable, c'est-à-dire l'analyse d'un programme durant son exécution. En fait, un programme, pour remplir sa fonction, s'appuie fortement sur le système d'exploitation sous-jacent. Les appels systèmes que le programme effectue peuvent révéler beaucoup de chose sur ce qu'il est en train de faire (lire un fichier, envoyer un paquet réseau etc.). *trace* est l'outil le plus connu sur Linux qui permet d'analyser les appels système.

La première chose à faire pour prendre connaissance de strace est de regarder quelles sont ses options (il suffit de lire la page de man).

Il y a 3 approches différentes pour analyser un exécutable : analyse en boite blanche, en boite grise, en boite noire (resp. *white-box testing*, *grey-box testing*, *black-box testing*) L'analyse en boite blanche s'appuie sur un accès au code source, alors que l'analyse en boite noire ne dispose d'aucun code source ni aucune information sur le programme. L'analyse en boite grise offre un accès partiel aux structures de données, à la documentation et à l'architecture de l'application mais pas d'accès à la totalité du code source.

Au début, vous allez tester des programmes en boite blanche, avec un accès au code source. Puis, vous allez analyser des programmes en boite noire en lisant des fichiers de trace. En lisant les traces, vous aurez besoin de savoir ce que font les appels systèmes et notamment comment les paramètres sont utilisé. Aidez-vous des pages de manuel (section 2 en général ; exemple : *man 2 write*)

IV) Objectifs pédagogiques

- Savoir comment analyser un séquence d'appel système et reconnaitre et interpréter des motifs.
- Déterminer si un programme se comporte de manière attendue.
- Identifier quand un processus fait un fork d'un autre processus.
- Identifier quand un processus ouvre un fichier ou un socket.

- Identifier quand un processus efface un fichier.
- Identifier quels appels système introduisent une menace et comment cela arrive.

V) Instructions

Si vous manquez de pratique pour la ligne de commande, c'est le bon moment de se munir de cheat-sheet sur les commandes *grep, tail, head, sed, awk*, sur les redirections et les pipelines et sur les Regexp. Ces éléments seront nécessaires pour filtrer la sortie de *strace*.

a) Un programme vide

Votre répertoire contient différents fichiers qui seront utilisés dans ce TP. Un de fichier, *vide.c* , contient ceci :

```
int main(void) {}
```

Compiler ce programme ce la manière suivante :

```
gcc -o vide vide.c
```

À présent, lancer strace pour exécuter ce programme :

```
strace ./vide
```

Que pensez-vous que la sortie de strace montre (étant donné que le programme est censé ne rien faire). Combien d'appels système différents voyez-vous ?

b) Un programme simple

Le fichier bonjour.c contient ce simple programme :

```
#include <stdio.h>

int main(void) {
  printf("Hello world!\n");
}
```

Compiler *bonjour.c* en *bonjour* and executez-le avec *strace* :

```
gcc -o bonjour bonjour.c
strace ./bonjour
```

Comparer la sortie de strace pour *vide* et pour *bonjour*. Quelle partie de la sortie de strace est toujours la même ? Quelle partie est spécifique au programme *bonjour* ?

c) Distinguer le bon grain de l'ivraie AKA voir au dela du *boilerplate*

L'option -o de strace écrit le résultat dans un fichier. Lancer les commandes suivantes :

```
strace -o trace-vide ./vide
strace -o trace-bonjour ./bonjour
diff trace-vide trace-bonjour
```

Expliquer les differences entre les traces affichées avec la commande *diff* (*vimdiff* et *tkdiff* sont des outils plus riches pour observer les différences).

d) Première analyse en boite blanche

Analyser le programme *copie.c*

```
#include <stdio.h>
#include <stdlib.h>

int main (int argc, char** argv) {
  char c;
  FILE* input;
  FILE* output;
  char outputname[256];
  if (argc != 3) {
    printf("usage: ./copy <infile> <outfile>\n");
    exit(1);
  }
  snprintf(outputname, sizeof(outputname), "%s/%s", getenv("HOME"), argv[2]);
  input = fopen(argv[1], "r");
  output = fopen(outputname, "w");
  printf("Copie de %s vers %s\n", argv[1], outputname);
  while ((c = fgetc(input)) != EOF) {
       fprintf(output, "%c", c);
  }
  fclose(input);
  fclose(output);
}
```

Compilez ce programme en l'appelant *copie* et utilisez strace pour l'exécuter :

```
gcc -o copie copie.c
strace ./copie albatros.txt monalbatros.txt
```

Expliquer les parties spécifiques en associant chaque élément de la trace aux lignes correspondantes dans *copie.c* Pour quelles lignes du programme vous attendiez-vous à voir des éléments correspondants dans la trace mais qui n'y sont pas ?

e) Analyse en boite noite

Le fichier *identifier-trace* a été créé en appelant strace sur une commande. La première ligne de la trace a été effacée pour rendre l'identification de la commande plus difficile. Déterminer la commande sur laquelle strace a été lancée pour produire cette trace.

f) Avec ou sans filtre

Parfois, strace produit une quantité énorme de lignes. Une des façons de filtrer le résultat est d'enregistrer la sortie de strace dans un fichier et d'y chercher des lignes avec la commande *grep*. Mais la commande *strace* inclus certaines options qui peuvent résumer et filtrer. Pour les tester, lancer les commandes suivantes et expliquer les résultats :

```
find /etc/default
strace find /etc/default
strace -c find /etc/default
strace -e trace=file find /etc/default
strace -e trace=open,close,read,write find /etc/default
```

g) Le processus fait des petits

Le fichier *script.sh* contient le shellscript suivant :

```
#!/bin/bash
echo "a" > foo.txt
echo "bc" >> foo.txt
echo id -urn >> foo.txt
chmod 750 foo.txt
cat foo.txt | wc
chmod 644 foo.txt
```

Comparez la sortie des deux commandes suivantes. Chaque commande appelle strace qui va exécuter le shellscript, l'une avec l'option *-f* et l'autre sans. Expliquez ce que vous voyez dans les traces.

```
strace ./script.sh
strace -f ./script.sh
```

h) Qu'est-ce ?

Le fichier *ckoi* contient un exécutable dont le code source n'est pas disponible. Utilisez strace pour expliquer ce que fait ce programme dans le contexte des exemples suivants :

```
./ckoi foo abc
./ckoi foo def
./ckoi baz ghi
```

i) Big Brother is watching you

Créer un fichier d'une ligne que vous appellerez *secret.txt* comme dans l'exemple suivant (vous devez évidemment choisir quelque chose de différent comme secret) :

```
echo "Mon téléphone est le 12 34 56 78 90" > secret.txt
```

Avec la commande *chmod*, protégez le fichier avec les permissions appropriées (seul le propriétaire du fichier peut le lire).

Affichez le secret avec la commande *cat* :

```
cat secret.txt
Mon téléphone est le 12 34 56 78 90
```

- Est-ce que votre secret est réellement secret ?
- Quelle confiance accordez-vous à la commande *cat* ?

Essayez de lancer *strace cat secret.txt* pour déterminer ce que ça fait réellement. En vous basant sur ces expérimentations, répondez aux questions suivantes :

- La commande cat fait-elle plus qu'afficher le contenu d'un fichier ? Que fait-elle en plus exactement ?
- Comment pouvez-vous afficher le contenu d'un fichier sans les actions supplémentaires décrites ci-dessus.
- Est-ce que quelqu'un d'autre peut lire votre secret ?
- Pouvez-vous lire les secrets de quelqu'un d'autre ?
- Comment le programme cat contenant un cheval de troie a-t-il été écrit ? Comment pensez-vous qu'il a été installé ? Justifiez vos explications.

VI) Synthèse

- Quels sont les principaux types d'appels système ?
- Quels sont ceux que vous rechercheriez en priorité en faisant une analyse en boite noire ?
- Comment dissimuleriez-vous un rootkit qui copie un fichier dans un répertoire caché ?
- Comment dissimuleriez-vous un rootkit qui permet l'accès à un tier pour exécuter des commandes (un *reverse shell*)