Année Universitaire 2023-2024 session 1 d'automne

Parcours : TCY Master CSI **UE :** 4TCY904U
Épreuve : Examen Sécurité Système
Date : 20 décembre **Heure :** 14h30 **Durée :** 3h
Modalités : Épreuve sans document, calculatrice autorisée
Épreuve de : Philippe Virouleau, Samuel Thibault 6 pages

université de BORDEAUX

Collège Sciences et technologie

Exercice 1. Compilation

Q1.1 Au cours des différents TPs vous avez compilé un certains nombre de fichiers sources, afin d'obtenir des exécutables permettant d'effectuer des obfuscations. Sans rentrer dans le détail de chaque étape, expliquer les différentes étapes (et leur ordre) qu'a du subir votre code source pour être transformé en programme exécutable.

Le programme ci-dessous vise à implémenter un simple programme calculant un cosinus :

```c
#include <math.h>

int main(int argc, char **argv) {
  (void)argv;
  printf("cos: %f", cos(argc));
  return 0;
}
```

Voici un extrait de la documentation de la fonction *cos* :

```
double cos(double x);
These functions return the cosine of x, where x is given in radians.
```

J'essaye de compiler ce programme, et voici ce que j'obtiens :

```
$ clang cos.c -o cos
cos.c:5:3: warning: call to undeclared library function 'printf' with type 'int (const char *, ...)';
ISO C99 and later do not support implicit function declarations [-Wimplicit-function-declaration]
  printf("cos: %f", cos(argc));
  ^
cos.c:5:3: note: include the header <stdio.h> or explicitly provide a declaration for 'printf'
1 warning generated.
/usr/bin/ld: /tmp/cos-880533.o: in function `main':
cos.c:(.text+0x1c): undefined reference to `cos'
clang-15: error: linker command failed with exit code 1 (use -v to see invocation)
```

Q1.2 Le compilateur a émit un avertissement et une erreur : détaillez à quelle(s) étape(s) du processus de compilation ils ont été détectés, et d'où viennent ces problèmes.

Q1.3 Quelles modifications devrais-je faire au programme et à la ligne de commande pour que le programme compile ?

Q1.4 Les infrastructures de compilation proposent généralement un support pour de multiples langages de programmation (C/C++, Fortran, ...) et de multiple architectures cibles (x86, arm, ...), c'est le cas notamment de *GNU Compiler Collection* (GCC) et de *LLVM*. Expliquer l'architecture générale de ce genre d'infrastructures, et ce qui leur permet de ne pas écrire un compilateur complet par langage et architecture cible (vous pouvez évidemment prendre l'exemple de LLVM).

Exercice 2. Questions de cours sur l'obfuscation

Pour les questions suivantes, une et une seule réponse est correcte. Une mauvaise réponse ne retire pas de point.

Q2.1 Quel langage de programmation avez-vous principalement utilisé pendant les TPs ?

1. C++.
2. Java.
3. COBOL.
4. Python.

Q2.2 Dans le contexte de ce cours, que signifie MBA ?

1. Montery Bay Aquarium.
2. Mixed Boolean-Arithmetic.
3. Multi-Bus Adapter.
4. MacBook Air.

Q2.3 Quel type de protections peut déranger un reverseur qui observerait le code à l'aide d'un décompilateur ?

1. Les protections mécaniques.
2. Les protections dynamiques.
3. Les protections numériques.
4. Les protections statiques.

Q2.4 En utilisant de l'obfuscation sur un logiciel vous pouvez :

1. Garantir que personne ne pourra retrouver les fonctionnalités de ce logiciel.
2. Offrir une protection à ce logiciel en complexifiant le travail d'un attaquant.
3. Faire tomber la pluie.
4. Analyser une partie de son code pour comprendre sa fonctionnalité.

Q2.5 Qu'est ce qu'un prédicat opaque ?

1. Un prédicat que l'on ne peut observer qu'aux rayons X.
2. Une opération permettant de remplacer une addition par une suite d'instructions équivalente.
3. Une expression qui retourne toujours vrai ou toujours faux.
4. Une manière de se protéger de gdb.

Exercice 3. Représentation intermédiaire LLVM

Le programme suivant est une implémentation naïve de `strlen`, qui calcule la longueur d'une chaine de caractères :

```
#include <stddef.h>
size_t strlen(const char *str) {
  const char *s;
  for (s = str; *s; ++s)
    ;
  return (s - str);
}
```

En utilisant `clang` pour compiler ce programme on peut obtenir la représentation intermédiaire suivante :

```
; Function Attrs: nofree norecurse nosync nounwind readonly uwtable
define dso_local i64 @strlen(ptr noundef %0) local_unnamed_addr #0 {
  br label %2

2:
  %3 = phi ptr [ %0, %1 ], [ %6, %2 ]
  %4 = load i8, ptr %3, align 1, !tbaa !5
  %5 = icmp eq i8 %4, 0
  %6 = getelementptr inbounds i8, ptr %3, i64 1
  br i1 %5, label %7, label %2, !llvm.loop !8

7:
  %8 = ptrtoint ptr %3 to i64
  %9 = ptrtoint ptr %0 to i64
  %10 = sub i64 %8, %9
  ret i64 %10
}
```

Q3.1 Combien y a-t-il de *Basic Blocks* dans le corps de la fonction `strlen` ?

Q3.2 Dans quel *basic block* se trouve le corps de la boucle ?

Q3.3 Combien de prédécesseur(s) le *basic block* `%2` a-t-il ?

Q3.4 Les instructions `ptrtoint ptr %arg to i64` permettent de convertir `%arg` vers le type `i64`. Quelle contrainte sur les opérandes de l'instruction `sub i64 %8, %9` cette conversion permet-elle de satisfaire ?

Exercice 4. Protection d'un programme

L'algorithme *Breadth-first search* permet de trouver le plus court chemin, dans un graphe (G), entre un nœud racine (*root*) et un nœud cible (*goal*). Une application possible peut être de trouver le plus court chemin dans un labyrinthe. Le pseudocode [1] ci dessous est une implémentation possible de cet algorithme d'exploration, et retourne le nœud cible, depuis lequel on peut revenir à la racine en suivant le lien *parent*.

```
procedure BFS(G, root, goal) is
  let Q be a queue
  label root as explored
  Q.enqueue(root)
  while Q is not empty do
    v := Q.dequeue()
    if v is equal to goal then
      return v
    for all edges from v to w in G.adjacentEdges(v) do
      if w is not labeled as explored then
        label w as explored
        w.parent := v
        Q.enqueue(w)
```

1. Cet algorithme est donné en pseudocode car d'une part l'implémentation réelle en C++ – avec tous les éléments pour que la fonction compile – est beaucoup plus longue ; et d'autre part car ni le langage, ni la compréhension de l'algorithme ne sont en fait nécessaires pour répondre à la question.

On souhaite protéger cet algorithme de manière à ce qu'un attaquant utilisant une analyse statique du binaire ne puisse pas facilement comprendre que cette fonction effectue un BFS.

Q4.1 Quels sont les éléments du code que vous jugeriez utile de protéger ?

Q4.2 Quelles protections / transformations pouvez-vous suggérer d'appliquer, et dans quel ordre ? Pensez à justifier vos choix, et à décrire votre raisonnement si les noms précis des transformations ne vous reviennent pas.

Exercice 5. Réflexion sur un choix de protection

Lors de la distribution d'un jeu vidéo, les éditeurs ont plusieurs choix en matière de protection. Ils peuvent :

- distribuer le jeu sans protection. C'est une solution généralement adoptée par les jeux indépendants, ou, pour donner un exemple de gros studio, par CD Projekt Red.
- distribuer le jeu avec différentes protections, que ce soit pour empêcher sa copie illégitime, ou pour empêcher la triche dans le contexte d'un jeu multijoueurs. Cette solution peut être très onéreuse, et elle est généralement adoptée par des gros studios comme Ubisoft ou EA.

Dans le cadre de la deuxième option, une solution populaire de DRM est Denuvo. La page de leur produit annonce que la solution "s'applique au niveau du binaire", et qu'elle permet "d'offrir un certain temps sans version piraté au moment du lancement du jeu, quand le plus d'exemplaires sont vendus" [2]. Une solution complète utilisant à la fois *Anti-Temper* et *Anti-Cheat* permet à la fois une protection de type license (pour diminuer les copies illégitimes), ainsi qu'une protection lors du jeu multijoueurs.

Q5.1 Comparativement à une solution comme celles vu en TP où la protection s'intègre dans le système de compilation du programme, décrivez quelques avantages et inconvénients de l'approche utilisée par Denuvo (protection d'un binaire).

Q5.2 Les protections mises en œuvre ici sont principalement dynamiques (elles sont pertinentes pendant l'exécution du logiciel), donnez quelques exemples d'attaques et leur protection associées qui peuvent être mises en œuvre dans le contexte d'un jeu vidéo. Vous pouvez séparer ces attaques/protections en fonction de la problématique : à la fois d'un point de vue légitimité du jeu (ce qui est pertinent pour les jeux en ligne comme les jeux "hors ligne"), et d'un point de vue de gestion de la triche lors de jeux en ligne.

Q5.3 Comme pour toute protection dynamique, la vérification de la légitimité du programme doit se faire lors de son exécution. Celle-ci peut être faite au démarrage du programme, ou plusieurs fois pendant l'exécution. Sachant que dans le contexte des jeux vidéos la performance est importante, et en supposant qu'on soit en mesure d'instrumenter le binaire, décrivez une technique qui permettrait de faire un choix de "bon endroit" où ajouter des vérifications de manière automatique.

Q5.4 Quelles difficultés spécifiques à l'instrumentation de binaires pouvez-vous anticiper pour la mise en œuvre de la protection ?

2. `https://aws.amazon.com/marketplace/pp/prodview-x443idlstvufi`

Important : pour les exercices à partir de celui-ci, composer sur une autre copie

Important (2) : Soyez **précis** dans vos réponses en donnant des exemples précis d'attaques possibles. Des réponses floues (la caricature étant « pour des raisons de sécurité ») n'apporteront aucun point...

Exercice 6. Questions de cours

Q6.1 Pourquoi les vérifications de permissions d'ouverture de fichiers sont effectuées dans `vfs_open` plutôt que dans `ext4_file_open` ?

Que peut-il se passer sinon ?

Q6.2 Quel genre un peu différent d'opération de sécurité est fait dans `vfs_read` plutôt que dans `ext4_file_read`, pourquoi ?

Que peut-il se passer sinon ?

Q6.3 Pourquoi dans le rootkit du TP2 a-t-on eu besoin de quelque chose de similaire pour détourner l'appel système `kill` ?

Q6.4 Expliquer la différence entre un modèle de contrôle d'accès discrétionnaire (Discretionary Access Control) et un modèle de contrôle d'accès obligatoire (Mandatory Access control).

Exercice 7. Systèmes de fichiers, FUSE, et BPF

Récemment il a été discuté [3] au sein de la communauté de développement de Linux du problème posé par les vulnérabilités existant au sein des implémentations des systèmes de fichiers. Lorsqu'une telle vulnérabilité existe, pour pouvoir l'exploiter il suffit d'exposer le noyau à un système de fichier malveillant.

Q7.1 Notamment, l'environnement de bureau Gnome demande automatiquement à Linux de *monter* le système de fichier contenu sur les clés USB branchées par l'utilisateur, pour montrer automatiquement leur contenu sur le bureau. Comment profiter de ce vecteur d'attaque ?

Q7.2 Certains implémentations de systèmes de fichiers tels que EXT4 sont très bien maintenues, et contiennent très peu de telles vulnérabilités. D'autres implémentations d'autres systèmes de fichiers (qui étaient utilisés sur des systèmes d'exploitation très anciens) sont par contre très mal maintenues, et contiennent potentiellement beaucoup de vulnérabilités. À quels autres types de situations cela vous fait penser, et comment peut-on déclencher l'exploitation dans ces autres cas ?

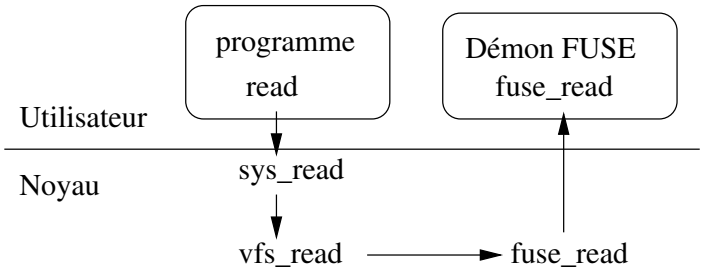
Q7.3 Pour éviter d'être exposé aux vulnérabilités des implémentations mal maintenues, `systemd` utilise par exemple un filtre BPF pour exclure automatiquement ceux-ci de l'auto-montage. Certains développeurs ont proposé de plutôt supprimer les implémentations des vieux systèmes de fichier. Pourquoi cette dernière proposition n'est pas une bonne idée, alors que pour les autres situations évoquées à la question précédente, c'est effectivement ce qui est généralement fait ?

Q7.4 Pour rappel, l'appel système `kernel_lockdown` peut être utilisé après le démarrage du système, pour indiquer que désormais même l'utilisateur `root` ne peut plus effectuer certaines opérations intrusives pour le noyau, telles que l'insertion de modules noyau, la lecture de mémoire physique depuis `/dev/mem`, etc., protégeant ainsi le noyau de l'utilisateur `root` lui-même. Pourquoi fait-on cela ?

Q7.5 Pourquoi peut-on dire qu'il vaudrait mieux, une fois `kernel_lockdown` appelé, empêcher `root` lui-même de monter des systèmes de fichiers ?

3. `https://lwn.net/Articles/939097/` et `https://lwn.net/Articles/951846/`

Il a été proposé, pour les systèmes de fichiers anciens et mal maintenus, de déplacer l'implémentation du noyau, pour la placer plutôt en espace utilisateur, à l'aide de l'interface FUSE (Filesystem in USErspace). Le principe est qu'un programme utilisateur, appelé *démon* FUSE, se met en attente des requêtes du noyau. Lorsque par exemple un programme effectue l'appel système `read()` et que donc la méthode `file_ops->read` du système de fichier est appelée, ici `fuse_read`, l'interface FUSE transmet la requête au démon, qui s'occupe de demander au disque dur de lire au bon endroit sur le disque dur, récupère les données, et les fournit à l'interface FUSE, et de proche en proche ces données sont finalement fournies à l'appel système `read()`.



Q7.6 Pourquoi gagne-t-on en sécurité ?

Q7.7 Est-ce que dans le cas où `kernel_lockdown` a été appelé, on préfère empêcher `root` de monter des systèmes de fichiers avec FUSE ?

Q7.8 Il a même été proposé d'utiliser `libguestfs` pour déplacer l'implémentation dans une machine virtuelle à part. Pourquoi gagne-t-on encore plus de sécurité par rapport au cas FUSE ?

Certains mainteneurs critiquent cependant qu'isoler ainsi l'implémentation en espace utilisateur (voire dans une machine virtuelle séparée) a un gros impact sur les performances.

Q7.9 L'interface FUSE définit entre autres les opérations `chown`, `close`, `getdents`, `open`, `read`, `rename`, `write`. Indiquez pour quelle(s) opération(s) l'impact de l'isolation est le plus grand sur les performances et expliquez pourquoi.

Il a été proposé [4] d'utiliser une implémentation hybride. Pour un système de fichier donné, on peut choisir (à l'aide d'un programme BPF) selon les opérations si l'on continue à utiliser l'implémentation noyau, ou si l'on fait effectuer l'opération en espace utilisateur par le démon FUSE.

Q7.10 Expliquez pourquoi on peut espérer ainsi garder à la fois une vitesse raisonnable et une sécurité raisonnable, en indiquant quels appels on fait effectuer par le démon FUSE et pourquoi.

L'utilisation de BPF pose cependant en lui-même des questions de sécurité [5].

Q7.11 Pourquoi l'utilisation de BPF est sensible aux attaques Spectre ?

Q7.12 Est-ce que dans le cas où `kernel_lockdown` a été appelé, on préfère empêcher l'espace utilisateur d'utiliser des programmes BPF ?

`iret`

4. `https://lwn.net/Articles/937433/`
5. `https://lwn.net/Articles/946389/`