Année Universitaire 2022-2023 session 1 d'automne

Parcours : TCY Master CSI **UE :** 4TCY904U
Épreuve : Examen Sécurité Système
Date : 15 décembre **Heure :** 14h **Durée :** 3h
Modalités : Épreuve sans document, calculatrice autorisée
Épreuve de : Philippe Virouleau, Samuel Thibault **7 pages**

université de BORDEAUX

Collège Sciences et technologie

Exercice 1. Compilation

Q1.1 Au cours du processus de compilation, un fichier source subit un certain nombre de transformations pour arriver à un fichier exécutable. Nommez quelques étapes importantes de ce processus.

Q1.2 Expliquez en quoi consiste le rôle de l'éditeur de lien.

Q1.3 Décrivez brièvement l'architecture générale de l'infrastructure de compilation LLVM.

Q1.4 Citez au moins un avantage et un inconvénient d'utiliser la représentation intermédiaire LLVM comme base pour faire de l'obfuscation.

Q1.5 Citez au moins un obfuscateur.

Exercice 2. Représentation intermédiaire LLVM

Étant donné le programme suivant :

```c
int main(int argc, char **argv) {
  size_t sum = 0;
  for (size_t i = 0; i < strlen(argv[0]); i++) {
    sum += argv[0][i];
  }
  printf("sum: %zu\n", sum);
  return 0;
}
```

Une compilation avec *clang* donne la représentation intermédiaire LLVM suivante :

```
define i32 @main(i32 %0, ptr %1) {
  %3 = load ptr, ptr %1, align 8
  %4 = tail call i64 @strlen(ptr %3)
  %5 = icmp eq i64 %4, 0
  br i1 %5, label %6, label %9
6:
  %7 = phi i64 [ 0, %2 ], [ %15, %9 ]
  %8 = tail call i32 (ptr, ...) @printf(ptr @.str, i64 %7)
  ret i32 0

9:
  %10 = phi i64 [ %16, %9 ], [ 0, %2 ]
  %11 = phi i64 [ %15, %9 ], [ 0, %2 ]
  %12 = getelementptr inbounds i8, ptr %3, i64 %10
  %13 = load i8, ptr %12, align 1
  %14 = sext i8 %13 to i64
  %15 = add i64 %11, %14
  %16 = add nuw i64 %10, 1
  %17 = icmp eq i64 %16, %4
  br i1 %17, label %6, label %9
}
```

Q2.1 Si *@main* est une fonction, que *%9* est un basic block, quel type d'élément est *%3* ou *%4* ?

Q2.2 L'instruction suivante est un noeud *phi* :

```
%10 = phi i64 [ %16, %9 ], [ 0, %2 ]
```

Dans quel cas *%10* peut-il prendre la valeur *0* ?

Q2.3 Rappelez l'utilité de ce type d'instructions dans le contexte de la représentation intermédiaire LLVM.

Q2.4 Compte tenu du nombre d'opérandes du noeud *phi*, combien de prédécesseurs possède le basic block *%9* ?

Exercice 3. 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.

Q3.1 Dans le second TP vous avez implémenté une protection statique visant à faire disparaître certaines constantes dans le binaire à l'aide d'un prédicat opaque. Laquelle de ces propriétés toujours vraie avez-vous utilisé ?

1. $x^2 > 0$
2. $\pi = 3.14159...$
3. $1 + 1 = 3$
4. $e = 2.71828...$

Q3.2 Quel est le but de la transformation *Mixed Boolean-Arithmetic* (MBA) ?

1. À transformer des entiers en booléens.
2. À manipuler la valeur du pointeur de pile pour protéger la valeur de l'opérande d'une instruction.
3. À substituer une instruction basique (comme une addition) par une suite d'instructions plus compliquée à comprendre.
4. À transformer des booléens en entiers.

Q3.3 Contre quel type d'attaque les protections dynamiques peuvent elles luter ?

1. Par canal auxiliaire (side channel).
2. Par instrumentation de code.
3. Par brute force.
4. Par analyse statique du binaire.

Q3.4 Laquelle de ces affirmations concernant l'obfuscation est vraie ?

1. Obfusquer un programme peut en changer la sémantique.
2. L'obfuscation offre une protection absolue contre un attaquant.
3. Obfusquer un programme peut permettre de corriger des bugs.
4. L'obfuscation est un compromis entre protection et performance du programme.

Q3.5 Quel outil de tests avez-vous utilisé dans les tps ?

1. GoogleTest.

2. Lit.
3. SpeedTest.
4. JUnit.

Exercice 4. Protection d'un programme

L'algorithme de Chudnovsky permet de calculer la valeur de π assez précisemment au bout d'un nombre d'itérations suffisantes. En supposant que les fonctions *pow* et *factorial* existent et sont implémentées correctement, une implémentation possible de cet algorithme est la suivante :

```
double pi(unsigned n) {
  double sum = 0.0;
  for (unsigned i = 0; i < n; i++) {
    double top = pow(-1, i) *
      factorial((6 * i)) * (13591409 + 545140134 * i);
    double bottom = factorial(3 * i) *
      pow(factorial(i), 3) * pow(640320, 3*i + 1.5);
    sum += top / bottom;
  }
  return 1 / (12 * sum);
}
```

Je 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 calcul de π.

Q4.1 Faites une analyse rapide du code ; quels sont les éléments du code que vous jugez utile de protéger ?

Q4.2 Quelles protections / transformations pourriez vous suggérer d'appliquer, et dans quel ordre (pensez à justifier vos choix) ?

Il y a plusieurs valeurs *signantes* [1] dans le code. Même si sa valeur n'apparait pas statiquement dans le binaire une fois les protections appliquées, elle pourrait toujours apparaître dans les registres lors de l'exécution.

Q4.3 Quel(s) type(s) d'attaque pourrait révéler cette valeur à l'attaquant ?

Q4.4 Quel(s) type(s) de protections pourriez vous suggérer pour la contrer ?

Q4.5 Quelle application peut on faire d'une fonction retournant une valeur connue à partir d'un paramètre arbitraire ?

Exercice 5. Réflexion sur un choix de protection

Vous êtes un éditeur de jeux vidéo qui souhaitez protéger votre dernier titre dans lequel vous avez mis tout votre investissement. Il est constitué d'un mode solo jouable hors ligne, ainsi que d'un mode multi joueurs agrémenté d'un classement en ligne. Vous souhaiteriez obtenir deux garanties :

— Qu'une personne ne puisse pas jouer sans une copie légitime du jeu
— Que lors du mode de jeu en ligne, une personne ne puisse pas tricher en changeant - par exemple - le comportement du jeu.

Q5.1 Faites une analyse rapide du contexte (le type de logiciel à protéger, le profil type de l'attaquant dont vous souhaitez potentiellement vous protéger, quelques autres considérations que vous pourriez avoir vis à vis de la protection de votre programme).

1. caractéristiques de l'algorithme

Q5.2 Compte tenu de votre analyse, proposez un ensemble de protections que vous pourriez appliquer sur le mode solo de votre jeu. Pensez à justifier vos choix, par exemple en illustrant quel type d'attaque vous pensez éviter.

Q5.3 Compte tenu de votre analyse, proposez un ensemble de protections que vous pourriez appliquer sur le mode multi joueurs de votre jeu. Pensez à justifier vos choix, par exemple en illustrant quel type d'attaque vous pensez éviter.

Q5.4 Donnez quelques raisons pour lesquelles il n'est pas très intéressant pour vous d'appliquer le même ensemble de protections aux deux modes de jeux.

Q5.5 Pouvez vous réellement garantir que vos protections seront efficaces sur le long terme ? Si oui comment ?

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

Exercice 6. Questions de cours

Q6.1 Pourquoi ajoute-t-on le plus systématiquement possible le qualificateur `const` sur les structures, et notamment celles contenant des pointeurs de fonctions ? (typiquement les structures `file_operations` par exemple)

Q6.2 Quelle autre stratégie de protection est mise en place pour augmenter la difficulté d'attaque de ces structures ?

Q6.3 Par quel genre de moyen peut-on parvenir à contourner cette protection ?

Q6.4 Classez les technologies suivantes selon les critères de modification du programme invité, de l'isolation, et de la vitesse :
- processus unix (par rapport à s'exécuter sans OS)
- chroot
- docker
- virtualisation totale

Q6.5 À quoi sert-il d'installer les "drivers virtualbox" dans l'invité, comment fonctionnent-ils ?

Exercice 7. Disciplines de ligne

Les disciplines de ligne sont des surcouches pour des canaux de communication existant. Par exemple, ajouter à un port série la discipline de ligne `N_MOUSE` permet d'interpréter les caractères reçus sur le port série avec le protocole souris. C'est donc notamment ainsi que l'on peut faire fonctionner une souris sur port série.

L'activation d'une discipline de ligne se fait simplement ainsi :

```
int disc = N_MOUSE;
ioctl(fd, TIOCSETD, &disc);
```

La macro `TIOCSETD` vaut simplement `0x5423`, et `fd` peut être n'importe quel descripteur de fichier d'un terminal texte, la sortie standard d'un programme, typiquement ; il n'y a pas besoin d'être root pour pouvoir faire cet appel. La variable `disc` contient le numéro de discipline de ligne voulue, ici `N_MOUSE` vaut `2`.

Lorsque cet appel `ioctl` est fait, le noyau charge alors automatiquement le module noyau `tty-ldisc-2` (2 comme le numéro de discipline), qui est un alias pour le module `serport`.

Q7.1 Pourquoi est-ce une mauvaise idée de charger ainsi un module de discipline de ligne automatiquement ?

Q7.2 Plus généralement, comment un utilisateur peut déclencher le chargement automatique d'un module ?

Q7.3 Le troisième paramètre de la fonction `ioctl` est un pointeur. Quel genre de problème cela peut-il poser ? Comment est-il géré par le noyau ?

Q7.4 Pourquoi `ioctl` pose-t-il de manière générale des problèmes de sécurité ?

Exercice 8. BPF

Le Berkeley Packet Filter (BPF) est à l'origine un outil permettant d'optimiser le fonctionnement de l'outil `tcpdump` : le processus utilisateur fournit au noyau le filtre de paquets à utiliser, sous la forme d'un mini-programme qui décide si un paquet doit être filtré ou non. L'utilisation de BPF a ensuite été étendu pour divers usages entre l'espace utilisateur et l'espace noyau.

Brièvement, l'ensemble des instructions possibles dans un mini-programme BPF est le suivant :

— ALU : effectuer un calcul entre deux registres `dst` et `src`, et stocker le résultat dans le registre `dst`.
— JMP : effectuer un saut, éventuellement selon le résultat d'une comparaison entre des registres `dst` et `src`.
— LD : charger une valeur depuis la mémoire.
— ST : stocker une valeur vers la mémoire.

Les registres sont numérotés de 0 à 15. Les instructions ayant besoin d'indiquer des registres contiennent des champs de 4 bit correspondant.

Q8.1 Pourquoi, si l'on n'utilise **que** des instructions ALU, aucun problème de sécurité particulier ne se pose ?

Q8.2 Lorsque l'on ajoute l'utilisation des instructions JMP, pourquoi souhaite-t-on en première approche imposer que l'adrese de destination du saut soit plus grande que l'adresse de l'instruction elle-même ?

Q8.3 Que pourrait-on faire simplement pour permettre d'autoriser que l'adresse de destination soit inférieure à l'adresse de l'instruction elle-même sans que cela pose un problème de sécurité ?

En pratique, le noyau utilise un *vérificateur* de programme BPF : quand l'espace utilisateur fournit un programme BPF au noyau, celui-ci analyse de manière approfondie le comportement du programme, pour déterminer s'il a un comportement potentiellement dangereux. C'est seulement si le noyau peut vérifier que le programme BPF ne peut pas avoir de tel comportement, que le programme est accepté.

Q8.4 Pourquoi a-t-on forcément des faux positifs dans la détection d'éventuel comportement dangereux ?

Q8.5 Pourquoi, de manière générale, préfère-t-on éviter que des portions mémoire du noyau aient à la fois les permissions `W` et `X` ?

Q8.6 Pourquoi les programmes BPF posent question par rapport à cela ?

Pour les instructions de chargement/stockage depuis/vers la mémoire, un champ de 32bits `imm` contient l'adresse mémoire au sein d'un tableau de taille limitée, ici à 4Ko.

Q8.7 Voici une implémentation de BPF, et notamment de l'instruction de stockage. Quel problème de sécurité pose-t-elle, comment l'exploiter ?

```c
static unsigned char memory[4096];
static int32_t registers[16];

void interpret_bpf(unsigned n, struct instruction prog[n]) {
  unsigned pc = 0; // Start at beginning of program

  while (1) {
    struct instruction instr = prog[pc];

    switch (instr->opcode) {
      // ...
      case ST:
        memory[instr->imm] = registers[instr->src];
        break;
      // ...
    }
  }
}
```

Q8.8 Proposez une correction de cette implémentation.

Q8.9 À quoi faut-il faire attention de manière similaire pour les instructions `JMP` ?

On souhaite intégrer aux instructions JMP des instructions `CALL` et `EXIT` pour permettre à un programme BPF d'avoir des fonctions ?

Q8.10 Qu'est-il nécessaire d'ajouter pour le fonctionnement de ces instructions, et à quoi faut-il faire attention ?

On souhaite désormais ajouter des instructions `ALLOC` et `FREE` au jeu d'instruction BPF.

Q8.11 Proposez des limitations et principes de fonctionnement de ces instructions pour éviter des problèmes de sécurité.

`iret`