HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD
www.heig-vd.ch

SLB-2023

Sécurité logicielle bas niveau

Laboratoire 1

ransTroj

Département TIC - orientations ISCE/S

Professeur :
Jean-Marc Bost

Assistant :
Lucas Gianinetti

05 octobre 2023

SLB-2023 – labo01 – ransTroj

Table des matières

1 Introduction

1.1 Principe du laboratoire

Un ransomware a infecté un utilitaire très utilisé dans l'entreprise dans laquelle vous travaillez. Vous avez identifié le programme suspect parce que tous les autres documents contenus dans le même dossier sont systématiquement devenus illisibles sur toutes les machines qui l'hébergent. Votre entreprise perd beaucoup d'argent à chaque minute qui passe ; votre job consiste donc à restaurer la situation au plus vite pour pouvoir à nouveau lire les documents essentiels et exécuter le programme sans risque.

Pour votre analyse, vous avez à votre disposition le contenu du dossier */home* de l'une des machines infectées. Celui-ci contient :

- Le binaire suspect (nommé *calc*).

  Attention MALWARE: ne l'exécutez pas n'importe où !

- Des documents très importants devenus illisibles
- Un template de code C (nommé *modify.c*)

1.1 Objectifs

Les objectifs de ce laboratoire sont les suivants :

- maitriser ghidra et gdb…
- … dans un contexte d'analyse Reverse réaliste et légal

1.2 Rendu

Le rendu est optionnel. Il est composé de :

- un court rapport de 3 pages A4 max répondant aux **questions 4.1, 4.2, 4.7, 7.2 et 8.1 UNIQUEMENT**

- le **source** et le **binaire** du programme de déchiffrement que vous allez écrire pour restaurer les fichiers chiffrés par le ransomware
- le **binaire patché** par vos soins pour désactiver le malware

Aucune correction du laboratoire ne sera présentée en classe. L'équipe enseignante restent néanmoins à votre disposition (sur prise de rendez-vous) si vous avez des questions et/ou besoin d'aide.

1.3 Evaluation

Les rapports et programmes originaux, démontrant la réalité et la réussite des manipulations, et justifiant les réponses correctes aux questions donneront droit à un bonus sur le prochain laboratoire.

1.4 Matériel nécessaire

Les binaires doivent être générés sur la VM Ubuntu fournie en début de semestre avec gcc (voir la manipulation 3.2 pour les arguments).

Tous les fichiers nécessaires sont disponibles sur Cyberlearn :

https://cyberlearn.hes-so.ch/course/view.php?id=12570#section-2

NB : il y a un folder home par groupe => traitez UNIQUEMENT le vôtre !

2 1ère partie : rappels de C

Un ransomware lit et modifie des fichiers ce qui donne au code assembleur une structure bien particulière à laquelle vous souhaitez vous préparer. Vous allez donc coder en C un petit programme permettant de lire et modifier un fichier pour, ensuite, analyser le binaire correspondant avec votre outil favori : ghidra.

MANIPULATION 3.1

En utilisant uniquement les fonctions *fgetc()*, *fseek()*, *fputc()*, completez la fonction *encrypt* dans le fichier *modify.c* afin de faire un *XOR* sur chaque byte du fichier avec la constante *0xff*.

```
void encrypt(FILE* fp){
  int ch;
  while ((ch = /*TODO*/) != EOF){
    /*TODO*/
  }
```

QUESTION 3.1

Comment pouvez-vous déchiffrer ce fichier ?

MANIPULATION 3.2

Compilez votre programme à l'aide de la commande suivante :

gcc -O0 -fno-stack-protector -no-pie -m32 -fno-pic modify.c -o modify

et ouvrez le binaire avec ghidra. Que vaut la constante SEEK_CUR dans le code assembleur ?

3 2ème partie : structure du code

Vous allez, maintenant, essayer de comprendre la structure générale du code.

MANIPULATION 4.1

Ouvrez avec ghidra le fichier *calc*. Trouver tous les appels de fonctions appartenant au programme (hors librairie). Montrez le graphe des appels de ces fonctions depuis le *main*. Enfin, analysez le code des 2 fonctions appelées depuis le *main* à l'aide du code C décompilé.

QUESTION 4.1

Que fait le code au début de *main* avant l'invocation de la fonction *bruteforce* ? Quel est le but du malware, d'après vous ?

QUESTION 4.2

Que fait la fonction *bruteforce* invoquée au début du *main* ? Quel est le but du malware, d'après vous ?

QUESTION 4.3

Quelle est la valeur de l'argument passé à *bruteforce* ?

QUESTION 4.4

Que fait le code C selon les différentes valeurs de retour de la fonction *bruteforce* ?

QUESTION 4.5

Que fait la fonction *encrypt_dir* invoquée après l'appel à *bruteforce* dans *main* ?

QUESTION 4.6

Quelles sous-fonctions sont appelées par la fonction *encrypt_dir* et à quelle condition ?

QUESTION 4.7

Quelles exceptions prévoit le code de *encrypt_dir* ? Pourquoi ?

MANIPULATION 4.2

Commentez dans ghidra, le code C utilisé pour vos réponses aux questions 4.1 à 4.7 puis renommer et/ou « retyper » les variables locales selon nécessité.

QUESTION 4.8

Analysez le code assembleur correspondant à la fonctions *encrypt_dir* :

- Identifiez les parties de gestion de la pile comme telle (sans les expliquer) ;
- Expliquez les instructions qui implémentent la logique du chiffrement.

4 3ème partie : fonction *encrypt0*

Analysons le code de la fonction *encrypt0*.

MANIPULATION 5.1

Dans ghidra, afficher les codes C et assembleur de la fonction *encrypt0*.

QUESTION 5.1

Que fait le code C affiché par ghidra ?

MANIPULATION 5.2

Commentez le code C dans ghidra, puis renommer et/ou « retyper » les variables locales en conséquence de la réponse à la question 5.1.

QUESTION 5.2

Analysez le code assembleur correspondant à la partie chiffrement :

- Identifiez les parties de gestion de la pile comme telle (sans les expliquer)
- Expliquez les instructions qui implémentent la logique de la fonction *encrypt0*.

MANIPULATION 5.3

Ecrivez le code d'un programme capable de déchiffrer les fichier chiffré par la fonction *encrypt0*.

5 4ème partie : fonction *encrypt1*

Analysons le code de la fonction *encrypt1*.

MANIPULATION 6.1

Dans ghidra, afficher les codes C et assembleur de la fonction *encrypt1*.

QUESTION 6.1

Que fait le code C affiché par ghidra ?

MANIPULATION 6.2

Commentez le code C dans ghidra, puis renommer et/ou « retyper » les variables locales en conséquence de la réponse à la question 6.1.

QUESTION 6.2

Analysez le code assembleur correspondant à la partie chiffrement :

- Identifiez les parties de gestion de la pile comme telle (sans les expliquer)
- Expliquez les instructions qui implémentent la logique de la fonction *encrypt1*.

QUESTION 6.3

Quelle/s instruction/s n'a/ont pas été vue/s en cours ? Que fait/font elle/s ?

MANIPULATION 6.3

Ecrivez le code d'un programme capable de déchiffrer les fichier chiffré par la fonction *encrypt1*.

6 5ème partie : fonction *encrypt2*

Analysons le code de la fonction *encrypt2*.

MANIPULATION 7.1

Dans ghidra, afficher les codes C et assembleur de la fonction *encrypt2*.

QUESTION 7.1

Que fait le code C affiché par ghidra ?

MANIPULATION 7.2

Commentez le code C dans ghidra, puis renommer et/ou « retyper » les variables locales en conséquence de la réponse à la question 7.1.

QUESTION 7.2

Vous voulez restaurer les fichiers chiffrés avec *encrypt2*. Pourquoi n'est-il pas nécessaire de faire le reverse de la fonction appelée par *encrypt2*?

MANIPULATION 7.3

Ecrivez le code d'un programme capable de déchiffrer les fichiers chiffrés par la fonction *encrypt2*.

7 6ème partie : réparation (6 pts)

Maintenant que le malware n'a plus de secret pour nous, nous allons créer les binaires qui nous permettront de rétablir la situation au plus vite, avec :

- un utilitaire de déchiffrement pour restaurer les fichiers touchés
- l'utilitaire calc patché pour le rendre inoffensif

MANIPULATION 8.1

En utilisant les programmes que vous avez codés, déchiffrez tous les documents fournis dans le dossier */home*.

QUESTION 8.1

Expliquez l'algorithme de déchiffrement que vous avez implémenté. Indiquez les formules mathématiques appliquées par chaque méthode de déchiffrement ainsi que la logique du flux d'instruction qui permet d'appliquer l'une ou l'autre.

MANIPULATION 8.2

En utilisant ghidra, patchez le programme *calc* pour qu'il puisse à nouveau être utilisé sans risque.