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

SLO2020

S3cur1t3 L0g1c13ll3

Laboratoire 3a

Analyze concolique

Département TIC - orientations TS/IL/IE

Professeur :

Jean-Marc Bost

Assistants :

Loic Haas

Yann Lederrey

05 mai 2020

SLO2020 – LAB3a – Fuzzing

Table des matières

1 Introduction

1.1 Objectifs

Les objectifs de ce laboratoire sont les suivants :

- Se familiariser avec l'analyse concolique
- Comparer avec le fuzzing

1.2 Rendu

Un court rapport répondant **UNIQUEMENT** aux questions posées doit être rendu le **mardi 26 mai 2020** sur Cyberlearn (par email aux professeur et assistants, en cas d'indisponibilité du service).

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

1.3 Evaluation

Ce rapport fera l'objet d'une évaluation basée sur :

- (90% de la note) exactitude, lisibilité, efficacité et exhaustivité des réponses aux questions
- (10% de la note) la qualité du document en termes de présentation, rédaction et illustrations

Les points rapportés par chaque question réussie sont indiqués ci-après, pour un total de 30 points maximum. Il ne sera pas nécessaire d'avoir 30 points pour obtenir la note de 6.

1.4 Matériel nécessaire

Tous les fichiers nécessaires sont disponibles sur Cyberlearn :

https://cyberlearn.hes-so.ch/course/view.php?id=14959

Ce laboratoire a été testé sur une machine virtuelle Ubuntu 18.04 LTS.

2 Principe du laboratoire

Le but de ce laboratoire est une introduction à l'outil d'analyse concolique **KLEE**.

Note : Il est préférable de travailler sous un système d'exploitation avec Docker fonctionnel. Il est aussi possible d'installer KLEE en natif mais ceci est déconseillé car sujet à beaucoup de difficultés d'installation.

Figure 1 – Head of a man - Paul Klee - 1922

3 Préparation

Nous allons installer un container Docker de KLEE et le lancer. Pour ceci, exécutez les commandes suivantes

```
$ docker  pull  klee/klee
$ docker  run  -ti  --name=klee_container  --ulimit='stack=-1:-1'  klee/klee
```

Vous trouverez une explication de ces commandes sur la page suivante : https://klee.github.io/docker/, ainsi que plus de détails concernant l'utilisation de Docker pour KLEE.

4 Analyser une Fonction Simple (7 pts)

MANIPULATION 4.1

Commencez par suivre le premier tutoriel de KLEE. Vous le trouverez à l'url https://klee.github.io/tutorials/testing-function/ .

Tous les fichiers nécessaires aux tutoriaux se trouvent dans le dossier /home/klee/klee_src/examples

MANIPULATION 4.2

Similairement, suivez le second tutoriel :
https://klee.github.io/tutorials/testing-regex/

QUESTION 4.1 (1PT)

Que fait l'option *–only-output-states-covering-new* ? Pourquoi l'activer ?

QUESTION 4.2 (2 PTS)

Que contiennent les fichiers en *.ptr.err*. Comment les ouvrez-vous ?

Que contiennent les fichiers en *.ktest*. Comment les lisez-vous ?

MANIPULATION 4.3

Maintenant que vous savez comment analyser une fonction simple, analysez à l'aide de KLEE la fonction qui se trouve dans le fichier *chgSign.c*.

QUESTION 4.3 (2 PTS)

Que fait la fonction fournie ?

QUESTION 4.4 (2 PTS)

Quel bug est trouvé par KLEE ? Donnez un input permettant d'obtenir ce bug et expliquez ce qui ne va pas dans la fonction.

5 Labyrinthe (5 pts)

Dans cet exercice, nous allons résoudre un petit jeu à l'aide de KLEE.

MANIPULATION 5.1

Compilez et exécutez le fichier *maze.c* et amusez-vous un peu avec le petit jeu. Réfléchissez aux solutions possibles permettant de gagner le jeu.

MANIPULATION 5.2

Nous allons maintenant utiliser KLEE pour trouver toutes les solutions à ce labyrinthe. Le programme maze.c prend une entrée de *STDIN*. Il va falloir transformer cette entrée en input symbolique. Vous pouvez faire cela, vous aurez besoin de la commande *klee_make_symbolic (program, ITERS, "program");* Il reste ensuite à faire crasher le programme lorsque l'on gagne.

Note : pour obtenir tous les chemins faisant crasher le programme, utilisez l'option *–emit-all-errors* lorsque vous lancez klee.

QUESTION 5.1 (5 PTS)

Listez tous les chemins gagnants trouvés par KLEE. Expliquez pourquoi chacun de ces chemins est gagnant en vous référant au code.

6 Keygen avec KLEE (5 pts)

Vous avez mis la main sur un programme mais vous ne connaissez pas le mot de passe (le sérial) permettant de le lancer. En utilisant Ghidra et un décompilateur, vous avez réussi à isoler le code C permettant de lancer le programme. Vous trouverez ce code dans le fichier *keygen.c*. Votre but est maintenant de trouver un input valide à ce programme afin que le logiciel se lance. Trouver cet input est tout à fait possible à la main, mais prend un peu de temps. Nous allons donc utiliser KLEE pour faire cela beaucoup plus rapidement.

MANIPULATION 6.1

Modifiez le fichier afin de pouvoir exécuter KLEE dessus. Faites-en sorte que le programme crashe lorsqu'une solution est trouvée. Lancez KLEE à l'aide de la commande suivante :

klee --optimize --libc=uclibc --posix-runtime keygen.bc --sym-arg 100

QUESTION 6.1 (1 PT)

A quoi servent les différentes options passées à KLEE ?

QUESTION 6.2 (4 PTS)

Quel est le mot de passe permettant de lancer le programme ?

7 Algorithmes de Tri (7 pts)

Finalement, nous allons utiliser KLEE sur l'exemple sort qui se trouve dans les sources de KLEE.

QUESTION 6.1 (2 PTS)

Que fait la fonction *test()* du fichier *sort.c* ?

MANIPULATION 6.1

Essayez de lancer KLEE sur cet exemple. Vous allez rencontrer un problème.

QUESTION 6.2 (1 PT)

Quel problème rencontrez-vous ?

Pour résoudre ce problème, entourez les lignes posant problème de la manière suivante :

```
#ifdef DISABLE_KLEE
//lignes problematiques
#endif
```

MANIPULATION 6.2

Lancez à nouveau KLEE sur ce programme modifié. Observez l'input qui pose problème.

Lancez ensuite le programme normal sur l'input posant problème. Pour vous

aider souvenez-vous de la dernière section du premier tutoriel http://klee.github.io/tutorials/testing-function/ et compilez à l'aide de l'option *-DDISABLE_KLEE* pour activer les lignes problématiques.

QUESTION 6.3 (4 PTS)

Quel est le bug dans le code ? Donnez un patch le corrigeant.

8 Comparaison avec le Fuzzer (6 pts)

MANIPULATION 7.1

Lancez maintenant KLEE sur le programme que nous avions utilisé pour tester afl dans le labo précédent.

QUESTION 7.1 (6 PTS)

Que pouvez-vous dire sur les performances de KLEE sur cet exemple par rapport à afl? Pourquoi avons-nous de telles différences ? Justifiez votre réponse.