LOG3430 - Méthodes de test et de validation du logiciel

Laboratoire 3

Fuzzing

Département de génie informatique et de génie logiciel
École Polytechnique de Montréal

Automne 2023

1 Introduction

Dans ce laboratoire, vous apprendrez le concept de «fuzzing», créerez des entrées «fuzzy» pour un programme et évaluerez votre «fuzzer» avec des informations de couverture.

Fuzzing/Fuzz Testing : est un moyen de générer automatiquement des cas de test afin de découvrir un comportement inattendu du programme (par exemple, crash, des fuites de mémoire ou d'autres types de bogues). Dans de nombreux cas, nous essayons de générer automatiquement des entrées aléatoires pour tester un programme.

2 Objectifs

Les objectifs généraux de ce laboratoire sont :

1. Apprendre à utiliser un outil pour générer des tests de fuzzing aléatoires et des tests de fuzzing mutés.
2. Apprendre à générer des rapports basés sur des résultats de fuzzing.
3. Observez les différences entre les différentes méthodes de fuzzing et les effets qui en résultent sur la couverture du code de test.

3 Fuzzing

Selon que le testeur connaît ou non la structure du programme, il peut y avoir différents types de fuzzing :

Fuzzing de boîte noire : Traite le programme comme une boîte noire et ignore la structure interne du programme. Par exemple, un outil de test aléatoire qui génère des entrées au hasard est considéré comme un fuzzer de boîte noire.

Fuzzing de boîte blanche : S'appuie sur l'analyse du programme pour augmenter systématiquement la couverture du code ou pour atteindre certains emplacements critiques du programme. Par exemple, on peut exécuter le programme (en commençant par quelques entrées aléatoires), rassembler des contraintes sur les entrées à des instructions conditionnelles, utiliser un solveur de contraintes pour générer de nouvelles entrées de test.

Fuzzing de boîte grise : Teste le programme avec une connaissance partielle de son fonctionnement interne. Par exemple, un fuzzer de boîte grise pourrait tirer parti des retours de couverture d'autres instrumentations ou bibliothèques pour apprendre à approfondir le programme. Si une entrée générée augmente la couverture, elle sera apprise par le fuzzer pour améliorer davantage le fuzzing.

Mécanisme

Un testeur essaie de s'introduire dans le système, ou l'application, à l'aide de valeurs de données aléatoires (c'est-à-dire fuzz). Dans cette méthodologie, les erreurs de codage et les vulnérabilités de sécurité sont généralement explorées en fournissant des entrées invalides ou aléatoires au système ou à l'application logicielle. Cela peut être vu comme un processus

automatisé ou semi-automatisé, où des défauts importants, principalement des failles de sécurité et des crash, des fuites de mémoire potentielles, etc. sont révélés, afin de les corriger.

Les outils

De nombreux outils de fuzzing sont largement utilisés dans l'industrie et la recherche.

— **Chaos Monkey** - un outil développé par Netflix, responsable de l'arrêt aléatoire des instances en production pour s'assurer que les ingénieurs mettent en œuvre leurs services pour être résilients aux défaillances d'instance.

— **OSS Fuzz** - un framework de fuzzing open-source de Google. Il vise à rendre les logiciels open source communs plus sûrs et stables en combinant des techniques de fuzzing modernes avec une exécution évolutive et distribuée.

— **AFL** - un fuzzer orienté pour la sécurité qui utilise un nouveau type d'instrumentation au moment de la compilation et des algorithmes génétiques pour découvrir automatiquement les cas de test qui déclenchent de nouveaux états internes dans le binaire ciblé.

Outils Requis

Veuillez vous assurer que **Python**, **fuzzingbook** (`pip install fuzzingbook`), et **matplotlib** (`pip install matplotlib`) sont installés sur votre ordinateur.

Système d'exploitation : Linux/macOS/Windows
Python : version 3.7 (ou+)

Les tâches

Obtenez une copie locale du code source pour le Lab 3 sur Moodle. https://moodle.polymtl.ca/

Partie A - *Random fuzzer*

Contexte

La sortie d'un fuzzer aléatoire est une chaîne générée aléatoirement, qui peut être utilisée comme entrée pour d'autres programmes. En C, nous avons souvent besoin de spécifier la longueur des caractères que nous devons lire. Si la longueur réelle est supérieure à la longueur spécifiée, cela peut provoquer un débordement de la mémoire tampon. Ces types de bogues peuvent entraîner des problèmes de sécurité. En utilisant un fuzzer aléatoire, nous pouvons générer automatiquement des entrées pour tester ce type de situation.

Tâches

Utilisez le code sous le dossier "Partie-A" et terminez les tâches suivantes :

— Modifiez la «seed» aléatoire dans le code 'random_fuzzer.py' à votre numéro d'étudiant (par exemple, 40005678)

— Ajoutez d'autres instructions d'impression pour afficher la «seed» aléatoire. Voici un exemple de la sortie attendue :

```
trial: 0
input: =!=;:3"!/61?7(50/90 539,8*.7'3(0)864)5;7';
The random seed is:  40005678

Traceback (most recent call last):
  File "<ipython-input-25-449b7bccfeb6>", line 12, in <module>
    crash_if_too_long(inp)
  File "<ipython-input-6-82d8cf6c68ac>", line 3, in crash_if_too_lo
ng
    raise ValueError
ValueError
```

— Exécutez le script modifié et joignez la capture d'écran de la sortie à votre rapport (Question 1). Vous pouvez exécuter le script en exécutant la commande suivante : `python random_fuzzer.py`

— Quel est le type de ce fuzzer (Choisissez-en un parmi Blackbox, Whitebox et Greybox) (Question 2) ?

Partie B - *Mutation fuzzer*

Contexte

La génération aléatoire d'entrées peut ne pas sembler très détaillé. Pour différents programmes, nous pourrions souhaiter une manière plus personnalisée de générer automatiquement des entrées.

Par exemple, si nous voulons tester un programme d'analyseur d'URL, nous ne voudrons peut-être pas générer autant d'entrées textuelles aléatoires pures. Parce que les utilisateurs ne se contentent généralement pas de saisir des textes aléatoires en tant qu'URL. Parfois, ils sont plus susceptibles de taper des URL avec quelques erreurs mineures ou fautes de frappe. Par conséquent, les entrées textuelles aléatoires pures ne peuvent qu'augmenter le temps de génération du test sans couverture de code supplémentaire (plus à ce sujet plus tard).

```python
from urllib.parse import urlparse

def http_program(url):
    supported_schemes = ["http", "https"]
    result = urlparse(url)
    if result.scheme not in supported_schemes:
        raise ValueError("Scheme must be one of " + repr(supported_schemes))
    if result.netloc == '':
        raise ValueError("Host must be non-empty")

    # Do something with the URL
    return True

def is_valid_url(url):
    try:
        result = http_program(url)
        return True
    except ValueError:
        return False
```

Nous pouvons concevoir des opérateurs de mutation simples (obtenus à partir de `fuzzingbook.MutationFuzzer`) pour tester cet analyseur d'URL. `MutationFuzzer` fournit certaines fonctions pour supprimer, ajouter et retourner la chaîne d'entrée. Nous pouvons utiliser ce fuzzer de mutation pour tester notre analyseur d'URL.

```python
from fuzzingbook.MutationFuzzer import MutationFuzzer
import random
random.seed(2020)
seed = "http://www.google.com"
mutation_fuzzer = MutationFuzzer([seed])

valid_inputs = set()
trials = 20

for i in range(trials):
    inp = mutation_fuzzer.fuzz()
    print ("input " + inp)
    if is_valid_url(inp):
        valid_inputs.add(inp)

percentage_of_valid_url = (len(valid_inputs)/ trials)*100


print ("%s of the generated inputs are valid URLs" % percentage_of_valid_url)
```

```
input http://www.google.com
input ht|zp:/www.googOl e.co:m
input hLtp://www.google.com
input ttp://waGww.google.oM
input http://wws.goge.coEm,
input http//gww.gogl.com
input httVp:'www.#tgoowle.#{ Xm
input `ttt://lwwg+.g oge.com
input (tt:d// w5ww.googlae.cLom
input http://www.g8oo~'lke>com
input htktp://www.gogglSe.com)
input http:/3wZwOwn.oR;oogle.coi
input http//www.mgoJ1og!7le.bo
input http:/ogww*g/gl/e com
input httt://wuw.googde4.com
input http;o-ugwngogle.cFom
input http:&+/ww.gq"oooleggi
input http://wwDw.oogle.Com
input ehttp://wsw). goolu.com
input htp:E/www.google.com
20.0 of the generated inputs are valid URLs
```

Tâches

Utilisez le code sous le dossier “Partie-B” et terminez les tâches suivantes :

— Modifiez la «seed» aléatoire dans le code ’mutation_fuzzer.py’ à votre numéro d’étudiant (par exemple, 40005678)
— Modifiez la «seed» du fuzzer de mutation à “https ://www.polymtl.ca/”
— Modifier le nombre d’essais (trials) à 40
— Ajoutez d’autres déclarations `print` pour montrer la «seed» aléatoire (identique à l’exigence de la partie A).
— Exécutez le script modifié et joignez la capture d’écran de la sortie au rapport (Question 3).
— Quel est le type de fuzzing de cet exemple (Choisissez-en un parmi Blackbox, Whitebox et Greybox) (Question 4) ?

Partie C - *Fuzzing et Code Coverage*

Contexte

Nous pouvons facilement obtenir des informations de couverture en Python et utiliser les informations de couverture pour évaluer les fuzzers. Nous pouvons configurer une fonction de trace (plus d'informations peuvent être trouvées sur https://pymotw.com/3/sys/tracing.html), ou utiliser la classe Coverage de `fuzzingbook` (pour cette partie, nous l'utiliserons comme exemple). Nous pouvons donc concevoir une fonction et calculer le nombre de lignes couvertes par une entrée spécifique.

```
def calculate_cumulative_coverage(input_population, function):
    cumulative_coverage = []
    all_coverage = set()

    for inp in input_population:
        with Coverage() as cov:
            try:
                function(inp)
            except:
                # we ignore exceptions for the purpose of this code here
                pass
        # set union
        all_coverage |= cov.coverage()
        cumulative_coverage.append(len(all_coverage))
    return cumulative_coverage
```

Nous testons ensuite les fuzzers de mutation pour l'analyseur d'URL avec 10 essais.

```
from fuzzingbook.MutationFuzzer import MutationFuzzer
from fuzzingbook.Coverage import Coverage
import random
random.seed(2020)

trials = 10
fuzzer = MutationFuzzer(seed = ["http://www.google.com"])

input_set = []
for i in range(0, trials):
    input_set.append(fuzzer.fuzz())

cumulative_coverage = calculate_cumulative_coverage(input_set, is_valid_url)
cumulative_coverage
```

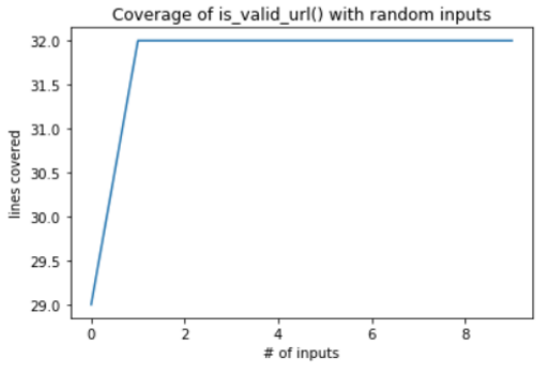
```
[29, 32, 32, 32, 32, 32, 32, 32, 32, 32]
```

D'après les résultats, nous avons pu voir que 29 à 32 lignes de code sont couvertes, avec différentes entrées générées par le fuzzer de mutation. Nous pouvons ensuite comparer les informations de couverture avec d'autres «seed» ou d'autres fuzzers. Nous pouvons également visualiser les informations de couverture en utilisant matplotlib :

```
import matplotlib.pyplot as plt


plt.plot(cumulative_coverage)
plt.title('Coverage')
plt.xlabel('# of inputs')
plt.ylabel('lines covered')
```

```
Text(0, 0.5, 'lines covered')
```



Tâches

Utilisez le code sous le dossier “Partie-C” et terminez les tâches suivantes :

— Exécutez `Random Fuzzer` de la partie A et `Mutation Fuzzer` de la partie B avec les informations de couverture de code générées respectivement, sur num2words (https://github.com/savoirfairelinux/num2words).
  — Plus précisément, vous devrez utiliser les résultats générés par les deux fuzzers comme entrées de num2words. Veuillez consulter le fichier readme de num2words pour savoir comment l’exécuter.
  (*indice* : num2words peut planter si les entrées ne sont pas des nombres. Ne vous inquiétez pas, c’est ce que nous attendons. Nous pouvons utiliser la fonction `calculate_cumulative_coverage(input_population, function)` fournie dans l’exemple pour obtenir le nombre de lignes de code couvertes jusqu’au crash point de num2words. Ainsi, nous pouvons comparer quel fuzzer peut générer des entrées qui pourraient couvrir plus de lignes de code)

— Vous allez peut être voir que le Random Fuzzer n’est pas assez efficace pour tester la fonction num2words. Votre tâche serait d’ajouter un nouveau Fuzzer, qui hérite de la classe Fuzzer de fuzzing book. Votre fuzzer doit générer des entrées plus pertinentes pour la fonction num2words i.e. la séquence de chiffres aléatoires. Voici un exemple de la définition de la classe :

```
from fuzzingbook.Fuzzer import Fuzzer
class MyFuzzer(Fuzzer):
    ...
```

—Exigences pour cette tâche—
1- Définissez la «seed» aléatoire comme votre numéro d’étudiant pour les trois fuzzers. N’hésitez pas d’expérimenter avec les différentes valeurs du seed.
2- Définissez la «seed» du fuzzer de mutation comme ”3452020” (cette «seed» est la chaîne initiale sur laquelle seront appliqués les opérateurs de mutation, et non une «seed» aléatoire).

3- Modifier le nombre d'essais (trials) à 500 pour les trois fuzzers
4- Tracez les informations de couverture de Random Fuzzer, de Mutation Muzzer et de votre Fuzzer dans un seul graphique.

- Joignez le graphique que vous générez à votre rapport (Question 5 - a)
- Décrivez le graphique que vous avez obtenu. Est-ce que la performance de votre Fuzzer est meilleur que Mutation Fuzzer en termes de la couverture de code ? (Question 5 - b)
- Enregistrez le code que vous avez implémenté en tant que q5.py (s'il s'agit d'un seul fichier) ou q5.zip (s'il y a plusieurs fichiers) avec le rapport (Question 5 - c).
- Quel est le type de fuzzer de mutation dans cette partie (choisissez-en un parmi Blackbox, Whitebox et Greybox) (Question 6) ?

Rapport - répondre au questions suivantes :

Question 1 : Collez la capture d'écran de la sortie de la partie A.

Question 2 : Quel est le type de fuzzer dans la partie A ? Expliquez brièvement votre raisonnement.

Question 3 : Collez la capture d'écran de la sortie de la partie B.

Question 4 : Quel est le type de fuzzer dans la partie B ? Expliquez brièvement votre raisonnement.

Question 5 - a : Collez la capture d'écran du graphique que vous avez généré pour comparer les informations de couverture de Random Fuzzer, de Mutation Fuzzer et votre Fuzzer.

Question 5 - b : Décrivez le graphique.

Question 5 - c : Incluez le code que vous avez implémenté en tant que q5.py (s'il s'agit d'un seul fichier) ou q5.zip (s'il y a plusieurs fichiers) avec votre rapport. Incluez la capture d'écran avec le code que vous avez ajouté, y compris l'implémentation du fuzzer, à votre rapport.

Question 6 : Quel est le type de fuzzer de mutation dans cette partie ? Expliquez brièvement votre raisonnement.

4 Livrables attendus

Les livrables suivants sont attendus :

- Un rapport pour le laboratoire. Le rapport doit contenir :
  - Vos réponses à la question 1 : (2 points).
  - Vos réponses à la question 2 : (2 points).
  - Vos réponses à la question 3 : (2 points).
  - Vos réponses à la question 4 : (2 points).
  - Vos réponses à la question 5 - a : (4 points).
  - Vos réponses à la question 5 - b : (2 points).
  - Vos réponses à la question 5 - c : (3 points).
  - Vos réponses à la question 6 : (2 points).
  - Qualité du rapport : (1 point).

— Le dossier COMPLET contenant les rapports (comme indiqué dans chaque question), ainsi que le code source nécessaire pour effectuer la question 5. **N'oubliez pas de bien commenter votre code**.

Le tout à remettre dans une seule archive **zip** avec le titre matricule1_matricule2_lab3.zip sur Moodle. Seulement une personne de l'équipe doit remettre le travail. Le rapport doit contenir le titre et numéro du laboratoire, les noms et matricules des coéquipiers ainsi que le numéro du groupe.

5 Information importante

1. Consultez le site Moodle du cours pour la date et l'heure limites de remise des fichiers.
2. Un retard de [0,24h] sera pénalisé de 10%, de [24h, 48h] de 20% et de plus de 48h de 50%.
3. Aucun plagiat n'est toléré. Vous devez soumettre uniquement le code et les rapports de couverture de code réalisé par les membres de votre équipe.