Analyse de programmes

Rapport projet 3 : Fuzzer

Gaëtan Ramack

Université de Namur
Faculté d'informatique
Année académique 2023-2024

Table des matières

1 Cible choisie

Le code contient un fuzzer destiné à générer des markdowns spécifiques afin de vérifier les outputs de la librairie *markdown-to-json*. Cette librairie a pour fonction, selon sa spécification dans le markdown du repository GitHub, de transformer un string respectant le format markdown et contenant des headers, listes et du texte en une structure JSON pensée afin d'être utilisée dans des datasets à la base.

Cette librairie a été choisie car celle-ci a beaucoup de chance d'être utilisée par d'autres programmeurs afin de récupérer des inputs de source non fiables, ce qui peut présenter un risque pour les programmes l'utilisant selon la réaction de celle-ci face aux différents inputs rencontrés. De plus, quand bien même celle-ci recoit des inputs de sources fiables, si la spécification d'une librairie ne correspond pas au résultat obtenu en l'utilisant, il est important d'avoir des mécanismes allant jusqu'à au moins prévenir qu'un cas non pris en charge a été rencontré. Autrement, une source fiable peut tout aussi bien être nuisible au bon fonctionnement du programme qu'une source non-fiable.

2 Fuzzer

Le petit fuzzer créé afin d'observer le comportement de cette librairie utilise des éléments des sections *"Fuzzing with grammars"*, *"Fuzzing with generators"*, *"Greybox Fuzzing"* et *"Greybox fuzzing with grammars"* du Fuzzing Book. Le fuzzer est un fuzzer Blackbox ayant pour but de feed en entrée des inputs suivant la spécification et d'observer les comportements de la librairie comme si le fuzzer était à la place d'un utilisateur et n'incorpore donc pas la métrique de coverage.

2.1 Fonctionnement

Une grammaire spécifiée selon la section *"Fuzzing with grammars"* du Fuzzing Book a été créée afin de pouvoir générer des fichiers markdowns reprenant les aspects utilisés par la librairie que sont les headers, éléments de listes et sous-listes et du simple texte. Utiliser en entrée un string vide renvoie le seul message d'erreur auquel le fuzzer a fait face et a donc été négligé dans la grammaire.

Le fuzzer suit la structure suivante, tout d'abord, un string est généré à l'aide de la classe *GeneratorGrammarFuzzer* sur base de la grammaire. Ensuite, ce string est utilisé en tant que seed valide pour le générateur de fragment *FragmentGenerator* afin de pouvoir exploiter la grammaire pour générer des inputs mutés respectant toujours la grammaire dans le LangFuzzer. Cette technique de mutation est expliquée dans la sous-section *"Fuzzing with Input Fragments"* de la section *"Greybox fuzzing with grammars"* et mute les seeds en translatant, supprimant ou ajoutant des sections de l'arbre de dérivation de la seed.Enfin, L'input généré à partir du LangFuzzer est à la fois retranscrit dans un fichier afin de pouvoir observer manuellement le markdown généré et passé en input à la fonction de la librairie *markdown_to_json.jsonify()*, chargée de transformer le markdown en JSON.

2.2 Complications rencontrées

La spécification de la librairie mentionne le fait que ca fonction est simple et est de transformer des headers markdown en clés, les valeurs sous les headers en valeurs JSON et les listes markdown en arrays. Seulement, il s'avère que la transformation ne suive pas à la lettre la spécification. Certains résultats observés manuellement à l'aide d'un *GeneratorGrammarFuzzer* et d'une grammaire ont démontrés que certains n'étaient pas ceux escontés. Cela en soi peut arriver, seulement la librairie ne dispose pas de moyen de vérifier si ces résultats sont conformes.

Afin de pouvoir développer un oracle repérant les cas problématiques, l'initiative d'incorporer des fonctions dans la grammaire (comme introduit dans la section *Fuzzing with generators*) a été lancée afin de pouvoir récupérer les labels de headingss ce dans un but de les comparer avec les clés du JSON renvoyé par la librairie. Seulement, après avoir introduit la technique de la sous-section *"Fuzzing with Input Fragments"*, il s'est avéré que le l'introduction de fonctions dans la grammaire n'a pas été incorporée dans l'implémentation du *FragmentMutator*.

Etant donné l'échec de la tentative de récupération automatique des labels avec une grammaire augmentée à l'aide de opts(), un rapide oracle comparant les clés de l'input et de l'output de la fonction *markdown_to_json.jsonify()* a été implémentée. Cet oracle, bien que non-exhaustif faute de temps, n'a aucun mal à trouver des résultats ne respectant pas la spécification de la librairie.

3 Observations

En utilisant la grammaire *MD_GRAMMAR*, élaborée suivant les formats de headings de Markdown, il a été observé certains comportements non répertoriés dans la documentation.

Tout d'abord, il s'est avéré que les headings se servant des balises html comme style de heading n'étaient pas reconnues par la librairie comme tel, les transformant donc en valeurs simples.

De plus, lorsque 2 types de styles de headings différents sont utilisés, seul 1 des types sera reconnu comme heading et donc transformé en clé, les headings suivant l'autre type seront vus comme des valeurs.

Il a aussi été observé que lorsque les headings ne suivent pas une structure descendante avec au plus d'un seuil de différence, la structuration en sous-ensemble dans le JSON décrite dans la spécification n'est pas appliquée.(exemple de structure descendante idéale : *"# heading1 ## heading2"*, exemple de structure descendante problématique : *"# heading1 #### heading2"*)

Afin d'obtenir des inputs plus pertinents, la grammaire *REDUCED_GRAMMAR*, une grammaire sans les headings non pris en compte par la librairie *markdown-to-json* basée sur MD_GRAMMAR, est utilisée par le fuzzer.

La grammaire *REDUCED_EXPANDED_GRAMMAR* est la grammaire comprenant la tentative d'utilisation d'*opts()*(Introduite dans la section *Fuzzing with generators*)

4 Instructions d'utilisation

4.1 Prérequis

Le fuzzer a été implémenté en Python 3.11.7 et requiert l'installation de la librairie *fuzzingbook* et *markdown-to-json*.

```
pip install fuzzingbook markdown-to-json
```

L'import *fuzzingbook.GreyboxGrammarFuzzer* utilisé par le fuzzer requiert d'installer *pyan* et *graphviz*. Il est à noter que la librairie tentera de télécharger *pyan* automatiquement. Seulement, sur Windows, il faut installer *pyan3* à l'aide de pip :

```
pip install pyan3
```

Cela est dû au fait que le repo *pyan* sur lequel va chercher la librairie *fuzzingbook* ne s'installe pas correctement.

4.2 Usage

Pour lancer le fuzzer, il n'y a qu'à exécuter le script *fuzzer.py* avec le nombre d'itérations souhaitées passé en paramètre.

Lors de chaque itération, les listes d'éléments parsés par l'oracle et le json récupéré en output de la fonction *markdown_to_json.jsonify()* sont affichées dans la console.

Un fichier *test.md* est créé lors de l'exécution du script. Ce dernier est utilisé pour vérifier manuellement que le dernier input fuzzé respecte bien le format markdown et les résultats affichés.