Trường Đại học Công nghệ Thông tin – ĐHQG Tp. HCM
Khoa Mạng máy tính và truyền thông

BÁO CÁO TỔNG KẾT ĐỒ ÁN MÔN HỌC

Môn học: Lập trình an toàn và khai thác lỗ hổng phần mềm

Tên chủ đề: CROSSFUZZ - CROSS-CONTRACT FUZZING FOR SMART CONTRACT VULNERABILITY DETECTION.

Mã nhóm: G02 Mã đề tài: CK18

Lớp: **NT521.P12.ANTT**

1. THÔNG TIN THÀNH VIÊN NHÓM:

STT	Họ và tên	MSSV	Email
1	Lại Quan Thiên	22521385	22521385@gm.uit.edu.vn
2	Mai Nguyễn Nam Phương	22521164	22521164@gm.uit.edu.vn
3	Đặng Đức Tài	22521270	22521270@gm.uit.edu.vn
4	Hồ Diệp Huy	22520541	22520541@gm.uit.edu.vn

2. TÓM TẮT NỘI DUNG THỰC HIỆN:[1]

A. Chủ đề nghiên cứu trong lĩnh vực An toàn phần mềm:

☑ Phát hiện lỗ hổng bảo mật phần mềm

☐ Khai thác lỗ hổng bảo mật phần mềm

☐ Sửa lỗi bảo mật phần mềm tự động

☐ Lập trình an toàn

☐ Khác: …….

B. Tên bài báo tham khảo chính:

H. Yang, X. Gu, X. Chen, L. Zheng, and Z. Cui, "CrossFuzz: Cross-contract fuzzing for smart contract vulnerability detection," *School of Computer Science, Beijing Information Science and Technology University, Beijing, China*, vol. 4, pp. 1-11, Aug. 2023, Revised 22 Dec. 2023, Accepted 27 Dec. 2023, Available online 4 Jan. 2024, Version of Record 9 Jan. 2024.

[1] Ghi nội dung tương ứng theo mô tả

C. Dịch tên Tiếng Việt cho bài báo:

CrossFuzz: Kiểm thử fuzzing giữa các hợp đồng để phát hiện lỗ hổng bảo mật trong hợp đồng thông minh.

D. Tóm tắt nội dung chính:

Nội dung chính của bài báo:

1. **Giới thiệu vấn đề**:

- Hợp đồng thông minh Ethereum ngày càng phổ biến nhưng dễ gặp các lỗ hổng bảo mật.

- Một loại lỗi phức tạp là **lỗ hổng giao dịch chéo** (cross-transaction vulnerabilities), thường khó phát hiện do liên quan đến nhiều giao dịch hoặc nhiều hợp đồng.

2. **Giải pháp đề xuất**:

- **CrossFuzz**, một công cụ kiểm thử tự động, được thiết kế để phát hiện các lỗ hổng bảo mật giao dịch chéo.

- Kết hợp phân tích dữ liệu liên hợp đồng và chiến lược fuzzing tối ưu để tạo ra các kịch bản kiểm thử hiệu quả.

3. **Phương pháp kỹ thuật**:

- Phân tích luồng dữ liệu liên hợp đồng (Inter-Contract Data Flow Analysis).

- Sử dụng chiến lược đột biến (mutation strategies) và điều chỉnh kiểm thử dựa trên dữ liệu.

- Áp dụng fuzzing để tìm các trạng thái nguy hiểm và khai thác lỗi.

4. **Thực nghiệm và kết quả**:

- CrossFuzz được đánh giá trên bộ hợp đồng thông minh thực tế.

- So sánh với các công cụ hiện có (Confuzzius, sFuzz, xFuzz), CrossFuzz phát hiện được nhiều lỗ hổng hơn và đạt độ bao phủ mã cao hơn.

5. **Đóng góp**:

- Giải pháp mới để phát hiện lỗi bảo mật phức tạp.

- Góp phần tăng cường an ninh cho hệ sinh thái hợp đồng thông minh Ethereum.

E. Tóm tắt các kỹ thuật chính được mô tả sử dụng trong bài báo:

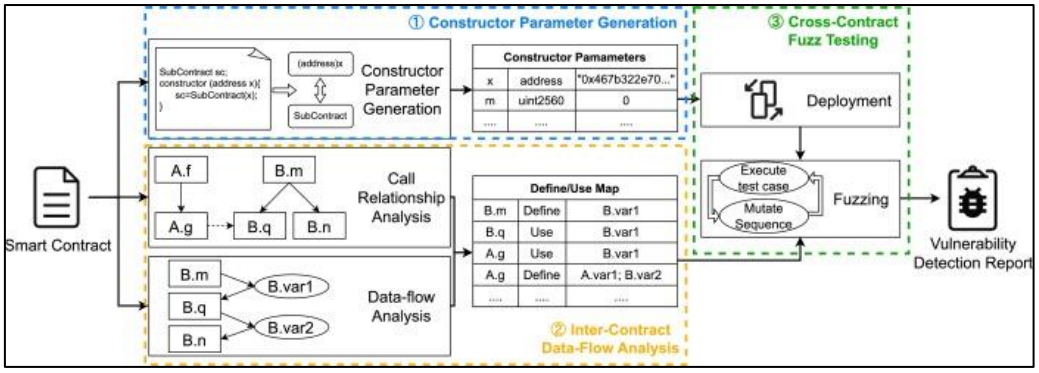
CrossFuzz là một phương pháp kiểm thử fuzzing để phát hiện lỗ hổng bảo mật giữa các hợp đồng thông minh.

1. Đầu tiên, CrossFuzz theo dõi các đường truyền dữ liệu của các tham số constructor, phân tích mối quan hệ giữa các tham số kiểu địa chỉ và các hợp đồng phụ thuộc, sau đó tạo các tham số cho constructor.

2. Tiếp theo, CrossFuzz phân tích ICDF để hướng dẫn việc biến đổi các chuỗi giao dịch.

3. Cuối cùng, CrossFuzz triển khai hợp đồng thông minh bằng các tham số constructor và thực hiện kiểm thử fuzzing.

Trong quá trình kiểm thử fuzzing, CrossFuzz thực thi các test case và giám sát việc thực thi các giao dịch, sau đó biến đổi chuỗi giao dịch dựa trên thông tin dòng dữ liệu. Khi thời gian kiểm thử đạt đến giới hạn đã định, CrossFuzz kết thúc và xuất báo cáo về các chuỗi giao dịch kích hoạt lỗ hổng. Các báo cáo này có thể giúp các nhà phát triển phân tích và xác định vị trí các lỗ hổng trong mã nguồn.



The framework of CrossFuzz

F. Môi trường thực nghiệm của bài báo:

- Cấu hình máy tính: Intel Core i7-12700, 32 GB RAM.
- Hệ điều hành Ubuntu 20.04 LTS.
- *Các công cụ hỗ trợ sẵn có*:

+ **ConFuzzius**: Làm nền tảng chính để xây dựng CrossFuzz, sử dụng trong toàn bộ giai đoạn thử nghiệm.

+ **Slither**: Dùng trong giai đoạn phân tích tĩnh để loại bỏ các hợp đồng không có lời gọi hàm ngoại vi.

+ **xFuzz**: Tham khảo tiêu chuẩn lựa chọn và tiền xử lý dữ liệu.

+ **Solidity Compiler (solc)**: Hỗ trợ trong việc triển khai và kiểm tra các hợp đồng thông minh.

- Ngôn ngữ lập trình để hiện thực phương pháp: Python

- *Đối tượng nghiên cứu*:

+ Tập dữ liệu bao gồm **396 hợp đồng thông minh** được thu thập từ **EtherScan**, đã qua xử lý để loại bỏ các hợp đồng không có lời gọi hàm ngoại vi.

+ Các hợp đồng được chia theo độ lớn (số dòng mã): <100 dòng, từ 100 đến <500 dòng, và ≥500 dòng.

- *Tiêu chí đánh giá tính hiệu quả của phương pháp*:

+ 2 tiêu chí: **Bytecode coverage** và **Số lượng lỗ hổng bảo mật được phát hiện.**

+ Kết quả được chạy 5 lần để lấy trung bình nhằm tránh ảnh hưởng của yếu tố ngẫu nhiên.

+ So sánh với các công cụ khác: Đánh giá với sFuzz, ConFuzzius, và xFuzz về bytecode coverage và số lượng lỗ hổng phát hiện được.

G. Kết quả thực nghiệm của bài báo:

Tóm lại, CrossFuzz là một công cụ kiểm thử fuzz được thiết kế đặc biệt để phát hiện lỗ hổng bảo mật liên hợp đồng trong các hợp đồng thông minh.

CrossFuzz giải quyết hai thách thức chính: *tạo tham số constructor và đột biến chuỗi giao dịch, nhằm cải thiện độ bao phủ mã và hiệu quả phát hiện lỗ hổng*. CrossFuzz hoạt động bằng cách:

- Theo dõi đường dẫn lan truyền dữ liệu của các tham số kiểu địa chỉ trong tham số constructor để tạo ra các tham số phù hợp.

- Phân tích mối quan hệ gọi hàm giữa các hợp đồng để xác định định nghĩa và cách sử dụng biến trạng thái của mỗi hàm.

- Thực hiện kiểm thử fuzz liên hợp đồng bằng cách tạo và thực thi các chuỗi giao dịch.

- Tối ưu hóa chiến lược đột biến chuỗi giao dịch dựa trên ICDF (Inverse Cumulative Distribution Function) để tăng khả năng khám phá các đường dẫn thực thi khác nhau.

Các thí nghiệm trong bài báo cho thấy CrossFuzz vượt trội hơn so với các công cụ kiểm thử fuzz hiện có như sFuzz, ConFuzzius và xFuzz. Cụ thể, CrossFuzz:

- Cải thiện độ bao phủ mã bytecode thêm 10,58% so với xFuzz, một công cụ fuzzing tập trung vào lỗ hổng liên hợp đồng.

- Phát hiện nhiều lỗ hổng bảo mật hơn 1,82 lần so với ConFuzzius, một công cụ kiểm thử fuzz kết hợp kỹ thuật thực thi biểu tượng.

H. Công việc/tính năng/kỹ thuật mà nhóm thực hiện lập trình và triển khai cho demo:

1. Tìm hiểu lý thuyết và phân tích bài báo

- Công việc:

+ Đọc và nghiên cứu chi tiết bài báo gốc về CrossFuzz.

+ Hiểu các khái niệm cơ bản liên quan đến fuzzing, đặc biệt là cross-contract fuzzing.

+ Phân tích cách thức hoạt động và các chiến lược đột biến được áp dụng trong CrossFuzz.

- Kết quả:

+ Nắm vững cấu trúc hệ thống của CrossFuzz.

+ Định nghĩa rõ các câu hỏi nghiên cứu (RQ1, RQ2, RQ3) để kiểm tra hiệu quả của phương pháp.

2. Xây dựng môi trường thực nghiệm

- Công việc:

+ Tạo môi trường thực nghiệm dựa trên công cụ CrossFuzz từ bài báo.

+ Chuẩn bị dữ liệu đầu vào gồm 40 hợp đồng thông minh, được tạo và tinh chỉnh với sự hỗ trợ của AI.

+ Thiết lập các phương pháp so sánh như ConFuzzius và xFuzz.

- Kết quả: Môi trường thực nghiệm sẵn sàng với đầy đủ dữ liệu và công cụ cần thiết.

3. Thực hiện thí nghiệm

- Công việc:

+ Thử nghiệm các chiến lược đột biến của CrossFuzz trên tập hợp các hợp đồng thông minh.

+ So sánh hiệu suất của CrossFuzz với ConFuzzius và xFuzz theo các tiêu chí:

- Khả năng bao phủ mã (Bytecode Coverage).
- Khả năng phát hiện lỗ hổng (Vulnerabilities Detection).

- Kết quả:

+ CrossFuzz đạt kết quả vượt trội:

- Bao phủ mã bytecode trung bình: 79.67% (cao hơn ConFuzzius và xFuzz).

- Phát hiện lỗ hổng: Trung bình 4.33 lỗ hổng/hợp đồng (cao hơn 1.625 lần so với ConFuzzius).

4. Phân tích kết quả thực nghiệm

- Công việc:

+ Phân tích chi tiết từng câu hỏi nghiên cứu (RQ1, RQ2).

+ Tổng hợp số liệu thống kê các loại lỗ hổng được phát hiện.

- Kết quả:

+ RQ1:

- Hiệu quả của chiến lược đột biến chuỗi giao dịch không rõ ràng trên các hợp đồng nhỏ (dưới 500 dòng).
- Độ bao phủ mã chỉ tăng khoảng 1% khi kích hoạt chiến lược này với các hợp đồng trên 500 dòng.

+ RQ2: CrossFuzz đạt kết quả vượt trội cả về khả năng bao phủ mã và phát hiện lỗ hổng so với các công cụ khác.

+ RQ3: Nhóm chưa thực nghiệm được.

+ Phân tích lỗ hổng:

- Thường xuyên phát hiện được:
  - Unchecked Return Value (34 lần).
  - Integer Overflow (26 lần).
  - Reentrancy (11 lần).
- Ít phát hiện hoặc không phát hiện được các lỗ hổng như Transaction Order Dependency (4 lần), Leaking Ether, Arbitrary Memory Access.

5. Đánh giá và nhận định

- Công việc:

+ Đánh giá chất lượng tệp dữ liệu và phương pháp thực nghiệm.

+ Đưa ra nhận định về hạn chế trong quá trình thực hiện.

- Kết quả:

+ Hạn chế về tính đa dạng của tập dữ liệu do được hỗ trợ bởi AI, chưa đủ đại diện cho các hợp đồng thông minh thực tế.

+ Một số lỗ hổng hiếm gặp chưa được phát hiện hoặc xuất hiện ít lần.

I. Các khó khăn, thách thức hiện tại khi thực hiện:

- **Lần đầu tiếp xúc với blockchain và hợp đồng thông minh**: Các thành viên trong nhóm chưa có nhiều kinh nghiệm với blockchain và hợp đồng thông minh, điều này gây khó khăn trong việc hiểu và làm việc với các công nghệ mới mẻ và phức tạp.
- **Cần tự học và nghiên cứu thêm nhiều kiến thức**: Blockchain, hợp đồng thông minh và các kỹ thuật fuzzing đòi hỏi kiến thức chuyên sâu về lập trình, bảo mật, và cách thức hoạt động của các hệ thống phân tán. Nhóm sẽ phải dành nhiều thời gian để tự nghiên cứu, học hỏi các công cụ và phương pháp mới.
- **Chưa thể build được công cụ CrossFuzz cho hệ điều hành Windows**, các thực nghiệm trong đồ án được thực hiện trên Ubuntu.

3. TỰ ĐÁNH GIÁ MỨC ĐỘ HOÀN THÀNH SO VỚI KẾ HOẠCH THỰC HIỆN:

90%

4. NHẬT KÝ PHÂN CÔNG NHIỆM VỤ:

STT	Công việc	Phân công nhiệm vụ
1	Lên ý tưởng, đọc và nghiên cứu bài báo	Cả nhóm
2	Quay video và thực hiện Demo CrossFuzz với 4 hợp đồng do nhóm biên soạn	Lại Quan Thiên Mai Nguyễn Nam Phương
3	Thực hiện so sánh giữa CrossFuzz với xFuzz và ConFuzzius	Đặng Đức Tài Hồ Diệp Huy
4	Tinh chỉnh lại mã nguồn cho dễ hiểu, thêm chú thích các hàm, biến, lớp…	Cả nhóm
5	Thiết kế Slide báo cáo	Cả nhóm
6	Viết báo cáo	Cả nhóm

BÁO CÁO TỔNG KẾT CHI TIẾT

Phần bên dưới của báo cáo này là tài liệu báo cáo tổng kết - chi tiết của nhóm thực hiện cho đề tài này.

Qui định: Mô tả các bước thực hiện/ Phương pháp thực hiện/Nội dung tìm hiểu (Ảnh chụp màn hình, số liệu thống kê trong bảng biểu, có giải thích)

A. Phương pháp thực hiện

Ở phần này nhóm sử dụng kiến trúc và thành phần của hệ thống CrossFuzz giống như bài báo đã đề cập. Cụ thể phương pháp tiếp cận của CrossFuzz sẽ đặc biệt ở 3 kỹ thuật chính bao gồm:

1. Tạo tham số Constructor

- Quá trình tạo các tham số constructor của CrossFuzz bao gồm việc theo dõi đường truyền dữ liệu của các tham số kiểu địa chỉ trong constructor. Các tham số constructor được tạo ra giúp constructor khởi tạo chính xác các hợp đồng phụ thuộc, điều này giúp vượt qua các hạn chế của các phương pháp fuzzing hiện tại

```
Input: the contract under test C; the set of dependency contracts Deps
Output: the constructor parameters args of the contract C
 1: args = {}
 2: adr = getAddressTypeArgs(C.constructor.args)
 3: //iterate over the constructor statements
 4: for stmt in C.constructor.statements do
 5:    //statements for converting the type of variables
 6:    if stmt is TypeConvert then
 7:       if stmt.param in adr and stmt.type in Deps then
 8:          //set the parameter value
 9:          args[stmt.param] = stmt.type
10:       end if
11:    end if
12: end for
13: for arg in C.constructor.args do
14:    //the parameter value has not been set
15:    if args[arg] is Empty then
16:       if arg.type is address then
17:          //the address of the contract's creator
18:          args[arg] = contract_creator
19:       else
20:          args[arg] = default_value
21:       end if
22:    end if
23: end for
24: return arg
```

Hình. Constructor parameter generation

- Các hợp đồng thông minh tự động thực thi constructor của chúng trong quá trình triển khai, cho phép các nhà phát triển gán giá trị cho các biến trạng thái và khởi tạo trạng thái hợp đồng. Vì chỉ có người tạo hợp đồng mới có thể gọi constructor, họ thường thiết lập các thông tin quan trọng như chủ sở hữu hợp đồng, số tiền ban đầu, thời gian dịch vụ kết thúc, và địa chỉ của các hợp đồng phụ thuộc

- Phân tích Dòng Thông Tin (Information-Flow Analysis - IFA) là một kỹ thuật quan trọng để đảm bảo an ninh thông tin. Kỹ thuật này phân tích tính hợp pháp của các luồng dữ

liệu trong hệ thống. Trong IFA, quy trình "taint analysis" coi dữ liệu từ bên ngoài là nguồn "taint". Kỹ thuật này theo dõi luồng dữ liệu từ nguồn đó để đánh giá xem hệ thống có đảm bảo an toàn hay không. Dựa trên IFA, quy trình phân tích "taint" coi dữ liệu đầu vào từ bên ngoài là nguồn "taint" và theo dõi các đường truyền dữ liệu của chúng để phân tích xem hệ thống phần mềm có an toàn hay không. Giống như các kỹ thuật phân tích "taint", CrossFuzz theo dõi các đường truyền dữ liệu của các tham số kiểu địa chỉ trong constructor, xác định xem tham số đó có được sử dụng để khởi tạo hợp đồng phụ thuộc hay không, và thiết lập một ánh xạ giữa các tham số kiểu địa chỉ và các hợp đồng phụ thuộc. Khi triển khai hợp đồng đang thử nghiệm, CrossFuzz điền các địa chỉ của các hợp đồng phụ thuộc vào các tham số constructor để khởi tạo chính xác các hợp đồng phụ thuộc. Đối với các tham số kiểu địa chỉ không tìm thấy hợp đồng phụ thuộc tương ứng, CrossFuzz điền địa chỉ của người tạo hợp đồng vào. Các tham số constructor khác sẽ được điền các giá trị mặc định, giống như các phương pháp fuzzing truyền thống

- CrossFuzz tạo các tham số constructor như được mô tả trong Thuật toán ở hình Constructor parameter generation. Thuật toán nhận vào hợp đồng đang thử nghiệm C và tập hợp các hợp đồng phụ thuộc Deps, và sau đó xuất ra các tham số constructor đã được tạo cho hợp đồng C. Các bước trong thuật toán bao gồm:

+ Dòng 1-2 khởi tạo một danh sách tham số trống và lấy các tham số kiểu địa chỉ trong constructor.

+ Dòng 3-12 lặp qua các câu lệnh trong constructor, nếu câu lệnh là một câu lệnh chuyển đổi kiểu (chuyển đổi tham số kiểu địa chỉ thành hợp đồng phụ thuộc), tham số sẽ được ánh xạ đến hợp đồng phụ thuộc và giá trị tham số được thiết lập theo hợp đồng đó.

+ Dòng 13-23 lặp qua các tham số constructor chưa tìm thấy giá trị phù hợp và điền các tham số kiểu địa chỉ bằng địa chỉ của người tạo hợp đồng. Các tham số khác được điền với giá trị mặc định.

- Sau đó, CrossFuzz gửi các tham số constructor được tạo ra từ Thuật toán ở hình Constructor parameter generation đến EVM, giúp CrossFuzz thực thi chính xác các hàm trong các hợp đồng phụ thuộc. Cụ thể, sau khi triển khai các hợp đồng phụ thuộc, CrossFuzz thiết lập các tham số constructor của hợp đồng đang thử nghiệm và gửi bytecode cùng với tham số đến EVM. Sau đó, EVM triển khai hợp đồng đang thử nghiệm và tự động thực thi constructor của hợp đồng.

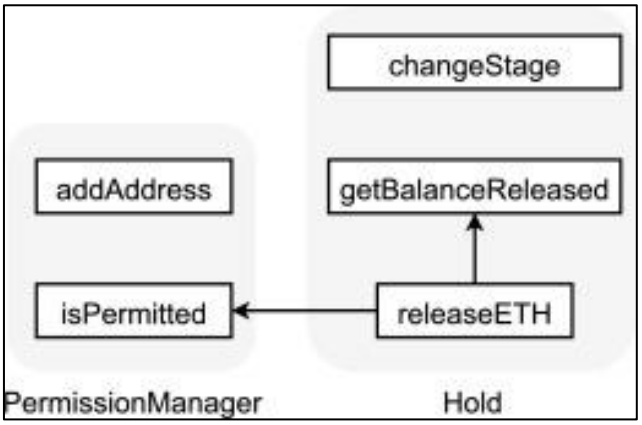
2. Phân tích dòng dữ liệu giữa các hợp đồng (ICDF)

- Để tối ưu hóa chiến lược biến đổi chuỗi giao dịch của CrossFuzz, nhóm tác giả thực hiện phân tích dòng dữ liệu giữa các hợp đồng (Inter-contract Data Flow - ICDF) trước khi tiến hành kiểm thử fuzzing. Quá trình này không chỉ xem xét các thao tác định nghĩa và sử dụng các biến trạng thái trong các hợp đồng riêng lẻ, mà còn xem xét các mối quan hệ gọi hàm giữa các hợp đồng khác nhau

- Các phương pháp kiểm thử fuzzing cho hợp đồng thông minh thường tạo ra các chuỗi giao dịch dựa trên phân tích dòng dữ liệu, có nghĩa là các giao dịch chứa các thao tác thay đổi biến trạng thái sẽ được thực thi trước, sau đó các giao dịch chứa thao tác đọc biến trạng thái

sẽ được thực thi sau. Dựa trên ConFuzzius, CrossFuzz phân tích ICDF để thu được thông tin về định nghĩa và cách sử dụng các biến trạng thái bởi các hàm. Thông tin này giúp tối ưu hóa chiến lược biến đổi chuỗi giao dịch, nâng cao hiệu quả phát hiện lỗ hổng

- Đầu tiên, CrossFuzz phân tích mối quan hệ gọi giữa các hợp đồng. Các câu lệnh trong một hàm được duyệt qua, và các câu lệnh gọi hàm được phân tích để lấy tên của các hợp đồng và hàm được gọi. Lưu ý rằng nếu một hợp đồng áp dụng bộ sửa đổi hàm fm cho hàm f, bộ sửa đổi sẽ được thực thi trước khi hàm f thực thi. Vì vậy, CrossFuzz phân tích các hàm nào được gọi bởi f và fm để xác định mối quan hệ gọi của f. Ví dụ, mối quan hệ gọi giữa các hợp đồng Hold và PM được thể hiện trong *Hình. Call relationships between PM and hold*



Hình. Call relationships between PM and hold

- Tiếp theo, các biến trạng thái mà các hàm định nghĩa và sử dụng được phân tích và ghi lại trong các define_map và use_map. Cụ thể, define_map ghi lại các biến trạng thái được định nghĩa bởi các hàm, trong khi use_map ghi lại các biến trạng thái được sử dụng bởi các hàm

```
Input: the set of functions F includes in the dependencies and under test contract
Output: the define map define_map and the use map use_map of F
 1: define_map,use_map = {}, {}
 2: for f in F do
 3:     visited = []
 4:     define_map[f],use_map[f] = analyze(f, [], [])
 5: end for
 6: return define_map,use_map
 7: Procedure analyze(f, define, use){
 8:     visited.add(f)
 9:     define.add(f.stateDefine)
10:     use.add(f.stateUse)
11:     //traverse the other functions that f calls
12:     for callee in f.callee do
13:         if callee in visited then
14:             continue
15:         endif
16:         analyze(callee, define, use)
17:     endfor
18:     return define,use
19: }
```

Hình. Analyze the definition and usage of state variables by functions

- CrossFuzz thông qua thuật toán ở hình Hình. Analyze the definition and usage of state variables by functions, thu thập được define_map và use_map đã được đề cập ở trên. Thuật toán này nhận vào tất cả các hàm F trong hợp đồng đang thử nghiệm và các hợp đồng phụ thuộc, sau đó xuất ra define_map và use_map của F. Các dòng trong thuật toán như sau:

Dòng 1-6 là phần chính của thuật toán, trong khi dòng 7-19 là quy trình đệ quy, được sử dụng để thu thập define_map và use_map của hàm f. Cụ thể:

+ Dòng 1 của thuật toán khởi tạo define_map và use_map.

+ Dòng 2-5 lần lượt phân tích mỗi hàm, trong đó Dòng 3 khởi tạo danh sách visited để ghi nhận các hàm đã được duyệt qua, và Dòng 4 gọi quy trình analyze để đệ quy kiểm tra các biến trạng thái được định nghĩa và sử dụng bởi hàm f đã cho.

+ Dòng 6 trả về define_map và use_map.

+ Trong quy trình đệ quy analyze, Dòng 8 thêm hàm f vào danh sách visited. Các Dòng 9-10 xác định các biến trạng thái được định nghĩa và sử dụng bởi f. Dòng 11-17, dựa trên mối quan hệ gọi hàm giữa các hợp đồng, tiếp tục đệ quy kiểm tra các hàm khác mà f gọi.

+ Dòng 18 trả về kết quả của define_map và use_map.

- Quy trình này giúp CrossFuzz thu thập được thông tin đầy đủ về dòng dữ liệu giữa các hợp đồng và tối ưu hóa chiến lược biến đổi chuỗi giao dịch, từ đó nâng cao khả năng phát hiện các lỗ hổng bảo mật.

3. Kiểm tra fuzzing hợp đồng chéo (Cross-contract fuzz testing)

- CrossFuzz thực hiện kiểm thử fuzz đa hợp đồng bằng cách tạo giao dịch gọi hàm từ hợp đồng cần kiểm thử và các hợp đồng phụ thuộc, rồi gửi chúng đến Máy ảo Ethereum (EVM) để phát hiện lỗ hổng bảo mật. Quá trình gồm bốn bước: triển khai hợp đồng, tạo trường hợp thử nghiệm, thực thi thử nghiệm, và đột biến thử nghiệm. Cụ thể như sau

- Đầu tiên, các hợp đồng phụ thuộc được triển khai trên Máy ảo Ethereum (EVM), và CrossFuzz lưu lại các địa chỉ của chúng để sử dụng trong quá trình thử nghiệm.. Tiếp theo, nó triển khai hợp đồng cần kiểm thử với các tham số khởi tạo được tạo ở phần 1.

- Tiếp theo, CrossFuzz sử dụng ABI (Application Binary Interface - Giao diện nhị phân ứng dụng) của các hợp đồng thông minh để lấy thông tin về các hàm, kiểu tham số và kiểu trả về. Từ thông tin này, CrossFuzz tạo ra các trường hợp thử nghiệm ban đầu với các chuỗi giao dịch chỉ chứa một lần gọi hàm cho mỗi hàm trong hợp đồng hoặc các hợp đồng phụ thuộc. Phương pháp này đảm bảo mỗi hàm được gọi ít nhất một lần, giúp kiểm thử toàn diện hơn so với các phương pháp khác như ConFuzzius.

- Sau đó, CrossFuzz mã hóa các chuỗi giao dịch thành dữ liệu hệ thập lục phân rồi gửi đến EVM để thực thi. Trong quá trình thực thi, nó giám sát các thay đổi trong ngăn xếp, lượng gas tiêu thụ, môi trường khối, và các thông tin khác để phát hiện lỗ hổng bảo mật.

+ CrossFuzz sử dụng phương pháp phản hồi phạm vi mã nhằm tăng cường phạm vi kiểm thử cho các hợp đồng thông minh. Những nhánh mã chưa được khám phá sẽ được ưu tiên đột biến để sinh ra các trường hợp thử nghiệm mới. Dựa theo độ chi tiết, đột biến trường hợp thử nghiệm có thể được phân thành hai loại: cấp độ chuỗi và cấp độ giao dịch. Cấp độ chuỗi thay đổi thứ tự hoặc số lượng giao dịch, trong khi cấp độ giao dịch thay đổi tham số trong các giao dịch.

+ Phần 2 phân tích ICDF đã giải thích cách các hợp đồng thông minh thay đổi biến trạng thái thông qua các giao dịch để thay đổi trạng thái của hợp đồng. Hợp đồng sử

dụng các câu lệnh như "require" hoặc "assert" để đảm bảo trạng thái hợp đồng đáp ứng điều kiện cần thiết khi thực thi. Nếu các điều kiện không đạt, lệnh REVERT của EVM sẽ được kích hoạt để hủy thay đổi và kết thúc giao dịch. Lệnh EXTCODESIZE được EVM sử dụng để kiểm tra xem hợp đồng được gọi có tồn tại hay không. Nếu không tồn tại, lệnh REVERT cũng sẽ được kích hoạt để ngăn việc thực thi tiếp

+ Trong các tình huống được mô tả ở trên, EVM đưa ra phán đoán dựa trên các biến trạng thái hoặc biến cục bộ của hợp đồng để xác định xem có gọi hợp đồng trống hoặc kích hoạt REVERT trong quá trình thực thi hợp đồng hay không. Với các giao dịch bị kết thúc bởi lệnh REVERT được coi là các giao dịch ngoại lệ. Các giao dịch ngoại lệ này có thể làm giảm khả năng bao phủ mã và cản trở việc thực thi các giao dịch tiếp theo, từ đó làm giảm hiệu quả của kiểm thử

+ Thông qua define_map và use_map, CrossFuzz sẽ tính điểm ưu tiên dựa trên các hàm bên trong define_map. Các hàm có điểm ưu tiên cao nhất được thêm vào chuỗi giao dịch để bổ sung thông tin biến trạng thái cần thiết cho các giao dịch ngoại lệ ban đầu

+ Lưu ý rằng CrossFuzz không phân biệt nguyên nhân của các ngoại lệ trong giao dịch. Do đó, CrossFuzz thực hiện đột biến chuỗi giao dịch cho bất kỳ giao dịch nào thực thi lệnh REVERT. Để tính điểm ưu tiên, nhóm tác gán điểm tích cực một cách heuristic nếu biến trạng thái được định nghĩa bởi hàm và điểm tiêu cực nếu biến trạng thái được sử dụng bởi hàm. Với một chuỗi giao dịch T và một giao dịch ngoại lệ exp, điểm ưu tiên của một hàm f sẽ được tính như sau:

$$S(f, exp) = S_{Def}(f, exp) + S_{Provide}(f, exp) - S_{Use}(f, exp) \quad (1)$$

+ Trong đó $S_{Def}(f, exp)$ là số biến trạng thái được hàm f định nghĩa ma chưa được định nghĩa trước giao dịch lỗi, $S_{Provide}(f, exp)$ là số biến trạng thái mà giao dịch lỗi cần sử dụng còn $S_{Use}(f, exp)$ là số biến trạng thái mà hàm sử dụng nhưng không liên quân đến lỗi

- Cuối cùng CrossFuzz thực thi các chuỗi giao dịch đã được đột biến trên EVM, thu thập thông tin để kiểm tra xem có phát hiện lỗ hổng bảo mật nào không. Các chuỗi tiếp tục được đột biến và thử nghiệm cho đến khi kết thúc thời gian kiểm thử, cuối cùng xuất ra báo cáo về các lỗ hổng được phát hiện

B. Chi tiết cài đặt, hiện thực

1. Cách cài đặt trên máy tính

- Sử dụng hệ điều hành: Ubuntu (phiên bản thích hợp nhất là Ubuntu 20.04 LTS)

- Sử dụng phiên bản Python 3.8

- Cài đặt môi trường và các dependencies lần lượt như sau:

+ sudo apt install python3.8 python3-distutils python3-dev gcc python3-pip python3-venv

+ git clone https://github.com/WanThinnn/CrossFuzz

+ cd CrossFuzz/

+ python3 -m venv myenv

+ source myenv/bin/activate

+ pip3 install wheel

+ pip3 install -r requirements.txt

+ solc-select install 0.4.26

+ solc-select use 0.4.26

+ mkdir fuzzer/result/

- Ngoài ra ta còn phải setting lại phiên bản solc mà ta đã tải trước đó để sử dụng (lấy đường dẫn solc bằng câu lệnh which solc) trong đường dẫn CrossFuzz/config.py, thay thế đường dẫn của ta vào biến **“SOLC_BIN_PATH = "/usr/local/bin/solc" # set to your solc path”**

2. Cấu hình máy tính sử dụng để lấy dữ liệu

- Laptop Lenovo Gaming LOQ 15IAX9

+ CPU: Intel Core i5 Alder Lake - 12450HX – 8 Cores – 12 Threads

+ Ram: 24GB DDR5

+ Card: RTX 3050 6GB

+ Ổ cứng: 512GB SSD

3. Dữ liệu thử nghiệm

- Vì tệp dữ liệu của nhóm tác giả không được công khai, nên nhóm đã thực hiện tìm kiếm và nhờ vào một số công cụ AI hỗ trợ, thực hiện lấy được 1 tệp dữ liệu thử nghiệm gồm 40 bản đa hợp đồng, bao gồm 3 tệp dữ liệu cụ thể là dưới 100 dòng, từ 100 đến 500 dòng và trên 500 dòng, tất cả được lưu trữ trên thư mục examples trong github chung của nhóm ở đây: Github. Cụ thể tập dữ liệu sẽ như sau:

+ 27 hợp đồng dưới 100 dòng

+ 10 hợp đồng từ 100 đến 500 dòng

+ 3 hợp đồng trên 500 dòng

4. Cách chạy công cụ

- Với 1 hợp đồng cơ bản như sau:

```
pragma solidity 0.4.26;


contract Sub {
    mapping(address => uint) public balances;

    function addBalances(address _addr, uint _amount) public {
        balances[_addr] += _amount;
    }

    function checkBalance(address _addr) public view returns (uint) {
        return balances[_addr];
    }
}

contract Child {
    uint inner;
}

contract E is Child {
    Sub sub;
    uint count;
    bool flag;

    function E(Sub _sub) public {
        sub = Sub(_sub);
    }

    function setSub(Sub _sub) public {
        sub = Sub(_sub);
    }

    function setCount(uint _count) public {
```

```
        count = _count;
    }

    modifier minBalance {
        require(sub.checkBalance(msg.sender) >= 1 ether);
        _;
    }

    function addBalance(address _addr, uint _amount) public {
        sub.addBalances(_addr, _amount);
        count += 1;
    }

    function getFlag() public returns (bool) {
        return flag;
    }

    function setFlag(bool _flag) public {
        flag = _flag;
    }

    function getInner() public returns (uint) {
        return inner;
    }

    function withdraw(address _addr, uint _amount) public minBalance {
        if (count > 50) {
            revert();
        } else {
            count += 2;
        }
        _addr.transfer(_amount);
    }

}
```

- Ta sẽ cần các cài đặt đơn giản như sau để kiểm thử 1 hợp đồng bên trong file .sol của ta, tất cả cài đặt sẽ nằm ở file CrossFuzz.py ở thư mục chính

```
def test_run():
    # absolute path
    _file_path = "./examples/T.sol"
    _main_contract = "E"
    solc_version = "0.4.26"
    evm_version = "byzantium"
    timeout = 10
    solc_path = config.SOLC_BIN_PATH
    _depend_contracts, _sl = analysis_depend_contract(file_path=_file_path, _contract_name=_main_contract,
                                                      _solc_version=solc_version, _solc_path=solc_path)
    max_individual_length = 10
    _constructor_args = analysis_main_contract_constructor(file_path=_file_path, _contract_name=_main_contract, sl=_sl)
    run(_file_path, _main_contract, solc_version, evm_version, timeout, _depend_contracts, max_individual_length,
        _constructor_args, _solc_path=config.SOLC_BIN_PATH)
```

+ _file_path: là đường dẫn đến file .sol của ta cần kiểm thử

+ _main_contract: tên hợp đồng chính mà ta thực hiện kiểm thử

+ solc_version: phiên bản solc mà ta đã tải và sử dụng

+ timeout: thời gian thực hiện kiểm thử

- Sau khi đã cài dặt các tham số đúng với nhu cầu của mình, thực hiện chạy CrossFuzz bằng script như sau: python3 CrossFuzz.py demo_test, đây là script nhóm tự sửa đổi lại để thuận tiện hơn cho việc kiểm tra các tham số đầu vào

```
def cli():
    if len(sys.argv) < 2:
        print("Usage: python3 CrossFuzz.py <option>")
        print("Please enter: \n 'python3 CrossFuzz.py demo_test' for demo \n 'python3 CrossFuzz.py --help' for using CLI ap
        sys.exit(1)
    option = sys.argv[1]

    if option.lower() == "demo_test":
        test_run()
        sys.exit(0)  # Kết thúc chương trình sau khi thực hiện test_run()

    # Hiển thị hướng dẫn sử dụng khi không có đủ tham số hoặc gọi `-help`
    if len(sys.argv) < 10 or sys.argv[1] in ("-help", "--help"):
        print(
            f"Usage: python {sys.argv[0]} <file_path> <contract_name> <solc_version> <max_trans_length> "
            "<fuzz_time> <res_saved_path> <solc_path> <constructor_params_path> <trans_duplication>\n\n"
            "Arguments:\n"
            "  file_path             Path to the Solidity file to be fuzzed\n"
            "  contract_name         Name of the contract to be fuzzed\n"
            "  solc_version          Supported Solidity compiler version (0.4.24, 0.4.26, 0.6.12, 0.8.4)\n"
            "  max_trans_length      Maximum transaction length (e.g., 10)\n"
            "  fuzz_time             Fuzzing duration in seconds (e.g., 60)\n"
            "  res_saved_path        Path to save the result JSON (e.g., ./result.json)\n"
            "  solc_path             Path to the Solidity compiler (solc)\n"
            "  constructor_params_path Path to constructor parameters (e.g., 'auto' or './examples/p.json')\n"
            "  trans_duplication     Duplicate transactions: 0 (no), 1 (yes)\n"
        )
        sys.exit(1)  # Kết thúc chương trình khi hiển thị hướng dẫn
    # Gán các tham số
    p = sys.argv[1]  # sol file path, which is the file path to be fuzzed
```

5. Thực hiện kiểm thử dữ liệu của nhóm

- Ở phần này nhóm sẽ chỉ lấy một số hình ảnh đại diện cho mỗi tệp dữ liệu nhất định, kết quả thực nghiệm sẽ được nêu rõ ở phần sau

+ Các dữ liệu dưới 100 dòng

```
INFO:Detector:          000000000001
INFO:Detector:-----------------------------------------------------
INFO:Analysis:Generation number 4       Code coverage: 96.46% (682/707)       Branch coverage: 91.30% (42/46)       Transactions: 443 (76 unique, 20 from cros
s)      Time: 2.0214688777923584
INFO:Analysis:Generation number 5       Code coverage: 96.46% (682/707)       Branch coverage: 91.30% (42/46)       Transactions: 615 (91 unique, 25 from cros
s)      Time: 2.759169578552246
INFO:Analysis:Generation number 6       Code coverage: 96.46% (682/707)       Branch coverage: 91.30% (42/46)       Transactions: 828 (106 unique, 30 from cro
ss)     Time: 3.6533358097076416
INFO:Analysis:Generation number 7       Code coverage: 96.46% (682/707)       Branch coverage: 91.30% (42/46)       Transactions: 1064 (121 unique, 35 from cr
oss)    Time: 4.653296709060669
INFO:Analysis:Generation number 8       Code coverage: 96.46% (682/707)       Branch coverage: 91.30% (42/46)       Transactions: 1326 (136 unique, 40 from cr
oss)    Time: 5.660953760147095
INFO:Analysis:Generation number 9       Code coverage: 96.46% (682/707)       Branch coverage: 91.30% (42/46)       Transactions: 1605 (151 unique, 45 from cr
oss)    Time: 6.817273855209351
INFO:Analysis:Generation number 10      Code coverage: 96.46% (682/707)       Branch coverage: 91.30% (42/46)       Transactions: 1902 (166 unique, 50 from cr
oss)    Time: 8.035222053527832
INFO:Analysis:Generation number 11      Code coverage: 96.46% (682/707)       Branch coverage: 91.30% (42/46)       Transactions: 2222 (181 unique, 55 from cr
oss)    Time: 9.255090951919556
INFO:Analysis:Generation number 12      Code coverage: 96.46% (682/707)       Branch coverage: 91.30% (42/46)       Transactions: 2560 (196 unique, 60 from cr
oss)    Time: 10.526272535324097
INFO:Analysis:-----------------------------------------------------
INFO:Analysis:Number of generations:    12
INFO:Analysis:Number of transactions:   2560 (196 unique)
INFO:Analysis:Transactions per second:  242
INFO:Analysis:Total code coverage:      96.46% (682/707)
INFO:Analysis:Total branch coverage:    91.30% (42/46)
INFO:Analysis:Total execution time:     10.57 seconds
INFO:Analysis:Total memory consumption:         95.22 MB
(myenv) ddücktai@ubuntu:~/Project-LTAT/CrossFuzz$
```

```
INFO:Analysis:----------------------------------------------------
INFO:Analysis:Number of generations:      13
INFO:Analysis:Number of transactions:     2676 (211 unique)
INFO:Analysis:Transactions per second:    266
INFO:Analysis:Total code coverage:        97.45% (689/707)
INFO:Analysis:Total branch coverage:      93.48% (43/46)
INFO:Analysis:Total execution time:       10.04 seconds
INFO:Analysis:Total memory consumption:           95.97 MB
(myenv) wanthinnn@ThinLinux:~/Documents/NT521/project/CrossFuzz$
```

```
INFO:Detector:          000000000001
INFO:Detector:----------------------------------------------------
INFO:Analysis:Generation number 4       Code coverage: 96.46% (682/707)       Branch coverage: 91.30% (42/46)       Transactions: 443 (76 unique, 20 from cros
s)      Time: 2.0214688777923584
INFO:Analysis:Generation number 5       Code coverage: 96.46% (682/707)       Branch coverage: 91.30% (42/46)       Transactions: 615 (91 unique, 25 from cros
s)      Time: 2.759169578552246
INFO:Analysis:Generation number 6       Code coverage: 96.46% (682/707)       Branch coverage: 91.30% (42/46)       Transactions: 828 (106 unique, 30 from cro
ss)     Time: 3.6533358097076416
INFO:Analysis:Generation number 7       Code coverage: 96.46% (682/707)       Branch coverage: 91.30% (42/46)       Transactions: 1064 (121 unique, 35 from cr
oss)    Time: 4.653296709060669
INFO:Analysis:Generation number 8       Code coverage: 96.46% (682/707)       Branch coverage: 91.30% (42/46)       Transactions: 1326 (136 unique, 40 from cr
oss)    Time: 5.660953760147095
INFO:Analysis:Generation number 9       Code coverage: 96.46% (682/707)       Branch coverage: 91.30% (42/46)       Transactions: 1605 (151 unique, 45 from cr
oss)    Time: 6.817273855209351
INFO:Analysis:Generation number 10      Code coverage: 96.46% (682/707)       Branch coverage: 91.30% (42/46)       Transactions: 1902 (166 unique, 50 from cr
oss)    Time: 8.035222053527832
INFO:Analysis:Generation number 11      Code coverage: 96.46% (682/707)       Branch coverage: 91.30% (42/46)       Transactions: 2222 (181 unique, 55 from cr
oss)    Time: 9.255090951919556
INFO:Analysis:Generation number 12      Code coverage: 96.46% (682/707)       Branch coverage: 91.30% (42/46)       Transactions: 2560 (196 unique, 60 from cr
oss)    Time: 10.526272535324097
INFO:Analysis:----------------------------------------------------
INFO:Analysis:Number of generations:    12
INFO:Analysis:Number of transactions:   2560 (196 unique)
INFO:Analysis:Transactions per second:  242
INFO:Analysis:Total code coverage:      96.46% (682/707)
INFO:Analysis:Total branch coverage:    91.30% (42/46)
INFO:Analysis:Total execution time:     10.57 seconds
INFO:Analysis:Total memory consumption:         95.22 MB
(myenv) ddücktai@ubuntu:~/Project-LTAT/CrossFuzz$
```

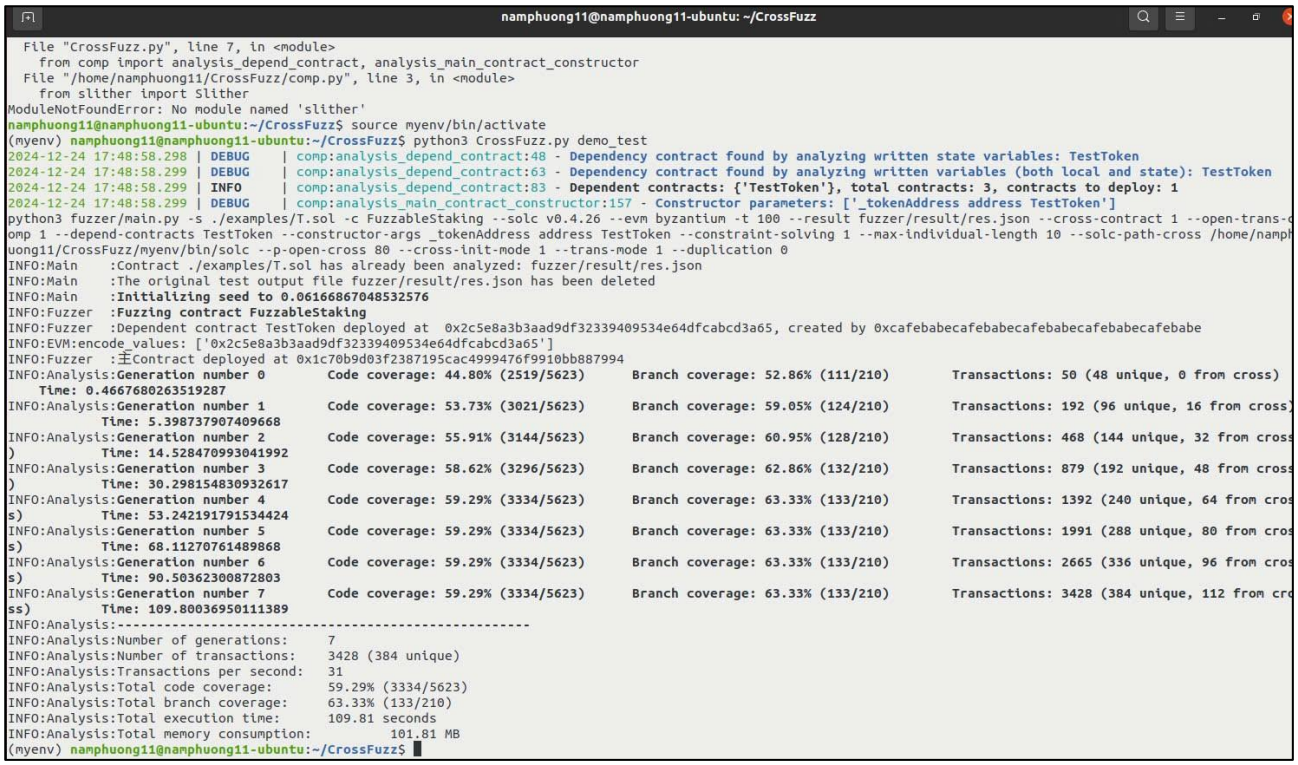
+ Các dữ liệu từ 100 đến 500 dòng

```
Time: 10.798091650009155
INFO:Analysis:----------------------------------------------------
INFO:Analysis:Number of generations:      24
INFO:Analysis:Number of transactions:     2454 (154 unique)
INFO:Analysis:Transactions per second:    227
INFO:Analysis:Total code coverage:        85.94% (489/569)
INFO:Analysis:Total branch coverage:      88.46% (23/26)
INFO:Analysis:Total execution time:       10.81 seconds
INFO:Analysis:Total memory consumption:           96.81 MB
(myenv) wanthinnn@ThinLinux:~/Documents/NT521/project/CrossFuzz$
```

```
INFO:Analysis:Generation number 16       Code coverage: 81.55% (411/504)        Branch coverage: 79.17% (19/24)        Transac
tions: 1796 (160 unique, 48 from cross)          Time: 12.425839900970459
INFO:Analysis:Generation number 17       Code coverage: 81.55% (411/504)        Branch coverage: 79.17% (19/24)        Transac
tions: 1932 (169 unique, 51 from cross)          Time: 13.506703853607178
INFO:Analysis:Generation number 18       Code coverage: 81.55% (411/504)        Branch coverage: 79.17% (19/24)        Transac
tions: 2092 (178 unique, 54 from cross)          Time: 14.873717784881592
INFO:Analysis:Generation number 19       Code coverage: 81.55% (411/504)        Branch coverage: 79.17% (19/24)        Transac
tions: 2264 (187 unique, 57 from cross)          Time: 16.198938846588135
INFO:Analysis:Generation number 20       Code coverage: 81.55% (411/504)        Branch coverage: 79.17% (19/24)        Transac
tions: 2423 (196 unique, 60 from cross)          Time: 17.400791883468628
INFO:Analysis:Generation number 21       Code coverage: 81.55% (411/504)        Branch coverage: 79.17% (19/24)        Transac
tions: 2560 (205 unique, 63 from cross)          Time: 18.374456644058228
INFO:Analysis:Generation number 22       Code coverage: 81.55% (411/504)        Branch coverage: 79.17% (19/24)        Transac
tions: 2760 (214 unique, 66 from cross)          Time: 20.00449848175049
INFO:Analysis:------------------------------------------------------
INFO:Analysis:Number of generations:     22
INFO:Analysis:Number of transactions:    2760 (214 unique)
INFO:Analysis:Transactions per second:   138
INFO:Analysis:Total code coverage:       81.55% (411/504)
INFO:Analysis:Total branch coverage:     79.17% (19/24)
INFO:Analysis:Total execution time:      20.02 seconds
INFO:Analysis:Total memory consumption:          96.37 MB
(myenv) dducktai@ubuntu:~/Project-LTAT/CrossFuzz$
```

+ Các dữ liệu trên 500 dòng

```
INFO:Analysis:Generation number 10       Code coverage: 53.99% (298/552)        Branch coverage: 56.82% (25/44)        Transactions: 783
(88 unique, 20 from cross)     Time: 4.522428750991821
INFO:Analysis:Generation number 11       Code coverage: 53.99% (298/552)        Branch coverage: 56.82% (25/44)        Transactions: 922
(96 unique, 22 from cross)     Time: 5.246276140213013
INFO:Analysis:Generation number 12       Code coverage: 53.99% (298/552)        Branch coverage: 56.82% (25/44)        Transactions: 1070
 (104 unique, 24 from cross)           Time: 6.0942699909210205
INFO:Analysis:Generation number 13       Code coverage: 53.99% (298/552)        Branch coverage: 56.82% (25/44)        Transactions: 1227
 (112 unique, 26 from cross)           Time: 7.068890810012817
INFO:Analysis:Generation number 14       Code coverage: 53.99% (298/552)        Branch coverage: 56.82% (25/44)        Transactions: 1393
 (120 unique, 28 from cross)           Time: 7.998387098312378
INFO:Analysis:Generation number 15       Code coverage: 53.99% (298/552)        Branch coverage: 56.82% (25/44)        Transactions: 1569
 (128 unique, 30 from cross)           Time: 8.977399826049805
INFO:Analysis:Generation number 16       Code coverage: 53.99% (298/552)        Branch coverage: 56.82% (25/44)        Transactions: 1595
 (142 unique, 32 from cross)           Time: 9.155169486999512
INFO:Analysis:Generation number 17       Code coverage: 53.99% (298/552)        Branch coverage: 56.82% (25/44)        Transactions: 1628
 (150 unique, 34 from cross)           Time: 9.376222848892212
INFO:Analysis:Generation number 18       Code coverage: 53.99% (298/552)        Branch coverage: 56.82% (25/44)        Transactions: 1696
 (158 unique, 36 from cross)           Time: 9.940489530563354
INFO:Analysis:Generation number 19       Code coverage: 53.99% (298/552)        Branch coverage: 56.82% (25/44)        Transactions: 1756
 (166 unique, 38 from cross)           Time: 10.398024559020996
INFO:Analysis:------------------------------------------------------
INFO:Analysis:Number of generations:     19
INFO:Analysis:Number of transactions:    1756 (166 unique)
INFO:Analysis:Transactions per second:   169
INFO:Analysis:Total code coverage:       53.99% (298/552)
INFO:Analysis:Total branch coverage:     56.82% (25/44)
INFO:Analysis:Total execution time:      10.42 seconds
INFO:Analysis:Total memory consumption:          94.95 MB
(myenv) dducktai@ubuntu:~/Project-LTAT/CrossFuzz$
```


namphuong11@namphuong11-ubuntu: ~/CrossFuzz
File "CrossFuzz.py", line 7, in <module>
from comp import analysis_depend_contract, analysis_main_contract_constructor
File "/home/namphuong11/CrossFuzz/comp.py", line 3, in <module>
from slither import Slither
ModuleNotFoundError: No module named 'slither'
namphuong11@namphuong11-ubuntu:~/CrossFuzz$ source myenv/bin/activate
(myenv) namphuong11@namphuong11-ubuntu:~/CrossFuzz$ python3 CrossFuzz.py demo_test
2024-12-24 17:48:58.298 | DEBUG | comp:analysis_depend_contract:48 - Dependency contract found by analyzing written state variables: TestToken
2024-12-24 17:48:58.299 | DEBUG | comp:analysis_depend_contract:63 - Dependency contract found by analyzing written variables (both local and state): TestToken
2024-12-24 17:48:58.299 | INFO | comp:analysis_depend_contract:83 - Dependent contracts: {'TestToken'}, total contracts: 3, contracts to deploy: 1
2024-12-24 17:48:58.299 | DEBUG | comp:analysis_main_contract_constructor:157 - Constructor parameters: ['_tokenAddress address TestToken']
python3 fuzzer/main.py -s ./examples/T.sol -c FuzzableStaking --solc v0.4.26 --evm byzantium -t 100 --result fuzzer/result/res.json --cross-contract 1 --open-trans-c
omp 1 --depend-contracts TestToken --constructor-args _tokenAddress address TestToken --constraint-solving 1 --max-individual-length 10 --solc-path-cross /home/namph
uong11/CrossFuzz/myenv/bin/solc --p-open-cross 80 --cross-init-mode 1 --trans-mode 1 --duplication 0
INFO:Main :Contract ./examples/T.sol has already been analyzed: fuzzer/result/res.json
INFO:Main :The original test output file fuzzer/result/res.json has been deleted
INFO:Main :Initializing seed to 0.06166867048532576
INFO:Fuzzer :Fuzzing contract FuzzableStaking
INFO:Fuzzer :Dependent contract TestToken deployed at 0x2c5e8a3b3aad9df32339409534e64dfcabcd3a65, created by 0xcafebabecafebabecafebabecafebabecafebabe
INFO:EVM:encode_values: ['0x2c5e8a3b3aad9df32339409534e64dfcabcd3a65']
INFO:Fuzzer :主Contract deployed at 0x1c70b9d03f2387195cac4999476f9910bb887994
INFO:Analysis:Generation number 0 Code coverage: 44.80% (2519/5623) Branch coverage: 52.86% (111/210) Transactions: 50 (48 unique, 0 from cross)
Time: 0.4667680263519287
INFO:Analysis:Generation number 1 Code coverage: 53.73% (3021/5623) Branch coverage: 59.05% (124/210) Transactions: 192 (96 unique, 16 from cross)
Time: 5.398737907409668
INFO:Analysis:Generation number 2 Code coverage: 55.91% (3144/5623) Branch coverage: 60.95% (128/210) Transactions: 468 (144 unique, 32 from cross
) Time: 14.528470993041992
INFO:Analysis:Generation number 3 Code coverage: 58.62% (3296/5623) Branch coverage: 62.86% (132/210) Transactions: 879 (192 unique, 48 from cross
) Time: 30.298154830932617
INFO:Analysis:Generation number 4 Code coverage: 59.29% (3334/5623) Branch coverage: 63.33% (133/210) Transactions: 1392 (240 unique, 64 from cros
s) Time: 53.242191791534424
INFO:Analysis:Generation number 5 Code coverage: 59.29% (3334/5623) Branch coverage: 63.33% (133/210) Transactions: 1991 (288 unique, 80 from cros
s) Time: 68.11270761489868
INFO:Analysis:Generation number 6 Code coverage: 59.29% (3334/5623) Branch coverage: 63.33% (133/210) Transactions: 2665 (336 unique, 96 from cros
s) Time: 90.50362300872803
INFO:Analysis:Generation number 7 Code coverage: 59.29% (3334/5623) Branch coverage: 63.33% (133/210) Transactions: 3428 (384 unique, 112 from cro
ss) Time: 109.80036950111389
INFO:Analysis:----------
INFO:Analysis:Number of generations: 7
INFO:Analysis:Number of transactions: 3428 (384 unique)
INFO:Analysis:Transactions per second: 31
INFO:Analysis:Total code coverage: 59.29% (3334/5623)
INFO:Analysis:Total branch coverage: 63.33% (133/210)
INFO:Analysis:Total execution time: 109.81 seconds
INFO:Analysis:Total memory consumption: 101.81 MB
(myenv) namphuong11@namphuong11-ubuntu:~/CrossFuzz$

C. Kết quả thực nghiệm

1. Thiết kế thí nghiệm

Để đánh giá hiệu quả của CrossFuzz, các nhà nghiên cứu đã xây dựng một quy trình thực nghiệm chi tiết, bao gồm các khía cạnh sau:

- **Tập dữ liệu**: Do các nghiên cứu trước đó về phát hiện lỗ hổng đa hợp đồng như Clairvoyance và SmartDagger chưa công khai tập dữ liệu của họ, và tập dữ liệu của xFuzz lại thiếu nhãn lỗ hổng, nhóm nghiên cứu đã tự xây dựng tập dữ liệu của riêng mình. Họ thu thập ngẫu nhiên 500 mã nguồn hợp đồng thông minh từ EtherScan, một nền tảng phân tích cho Ethereum. Sau đó, họ sử dụng công cụ phân tích tĩnh Slither để loại bỏ các hợp đồng không có lời gọi hàm ngoài (external function call), giữ lại 396 hợp đồng để tiến hành thử nghiệm. Các hợp đồng được phân loại theo ba kích thước dựa trên số dòng mã: dưới 100 dòng (<100), từ 100 đến dưới 500 dòng (<500) và từ 500 dòng trở lên (>=500).

- **Chỉ số đánh giá:** Hai chỉ số đánh giá chính được sử dụng là: Độ bao phủ mã bytecode và số lượng lỗ hổng được phát hiện. Độ bao phủ mã bytecode tính toán dựa trên công thức: độ_bao_phủ_bytecode = số_lệnh_được_thực_thi / tổng_số_lệnh. Số lượng lỗ hổng được phát hiện chỉ được so sánh giữa CrossFuzz và ConFuzzius, hai công cụ có cùng quy tắc phát hiện. Cả hai công cụ đều có khả năng phát hiện 11 loại lỗ hổng bảo mật. Ngoài ra, độ lệch chuẩn cũng được tính toán và hiển thị trên biểu đồ để minh họa ý nghĩa thống kê.

- **Hệ thống cơ sở (Baselines):** CrossFuzz được so sánh với ba fuzzer mã nguồn mở khác là: sFuzz, ConFuzzius, và xFuzz. sFuzz được phát triển dựa trên ContractFuzzer và cho kết quả tốt hơn về phát hiện lỗ hổng. ConFuzzius kết hợp kỹ thuật thực thi biểu tượng với fuzz testing, có khả năng bao phủ mã tốt. xFuzz được tối ưu hóa cho việc phát hiện lỗ hổng đa hợp đồng, cũng dựa trên sFuzz và ContractFuzzer.

- **Thiết lập tham số:** Tất cả các fuzzer đều sử dụng tham số mặc định. Phiên bản EVM được đặt là 'byzantium', ConFuzzius được kích hoạt mô-đun phân tích phụ thuộc dữ liệu và mô-đun giải ràng buộc. Thời gian kiểm thử tối đa cho mỗi hợp đồng thông minh được đặt là 10 phút, và mỗi hợp đồng được kiểm tra 5 lần, với kết quả cuối cùng là trung bình của 5 lần kiểm tra. Các thử nghiệm được thực hiện trên máy Ubuntu 20.04 LTS với bộ xử lý Intel Core i7-12700 và 32 GB RAM.

Tóm lại, để đánh giá khả năng bao phủ mã và phát hiện lỗ hổng của CrossFuzz thì ta sẽ tập trung vào 3 Research Questions:

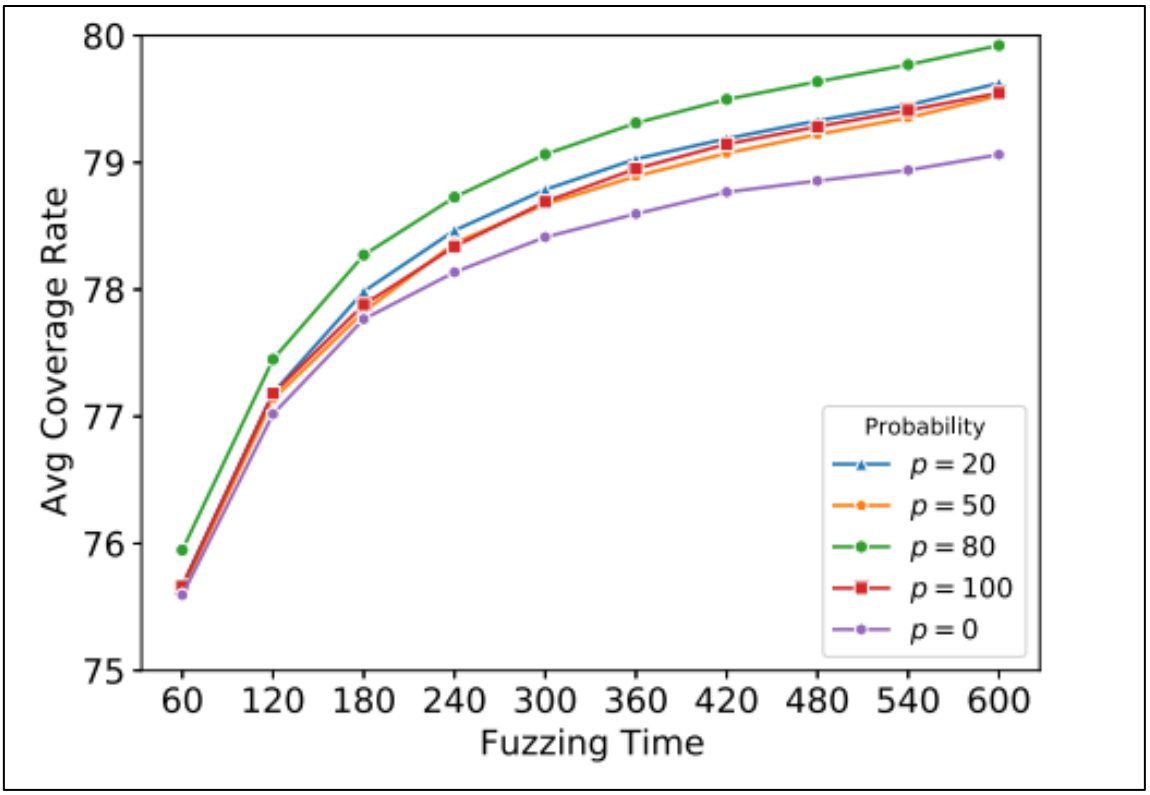
- RQ1: Liệu chiến lược đột biến chuỗi giao dịch có cải thiện khả năng bao phủ mã không? Nếu có, xác suất tối ưu để kích hoạt chiến lược này là bao nhiêu?

- RQ2: Liệu CrossFuzz có thể đạt được khả năng bao phủ mã cao hơn và phát hiện lỗ hổng tốt hơn so với các phương pháp fuzz testing hiện đại không?

- RQ3: Các tham số được tạo cho hàm khởi tạo có cải thiện khả năng bao phủ mã và phát hiện lỗ hổng không?

2. Đánh giá

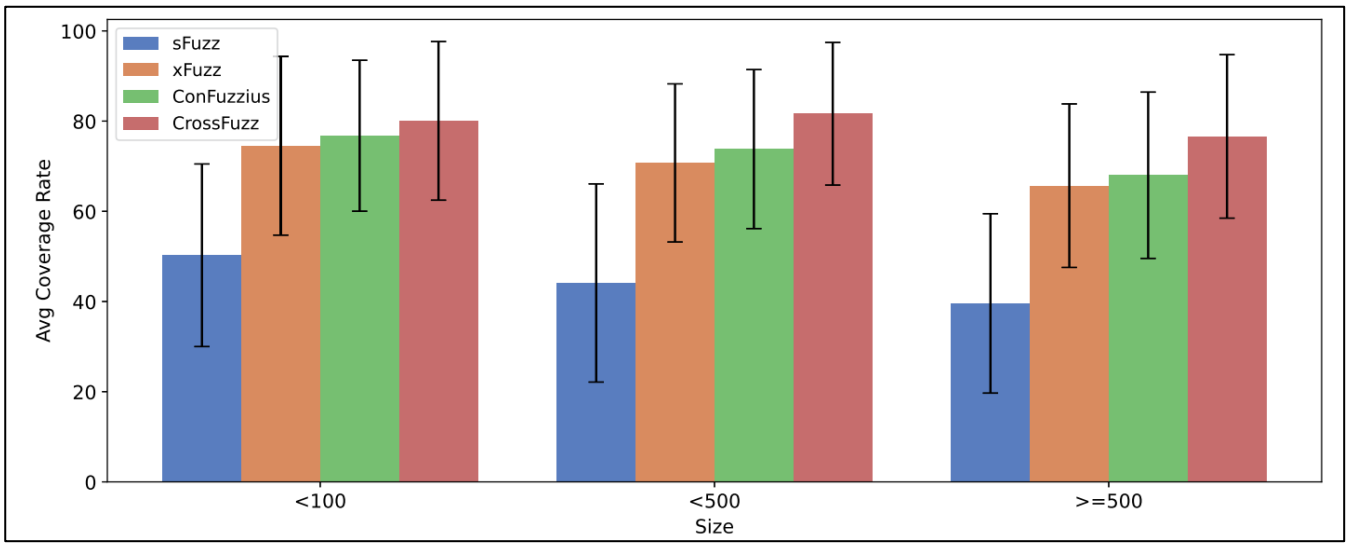
- Kết quả cho RQ1:



Comparison of enabling the transaction sequence mutation strategy with different activation probabilities.

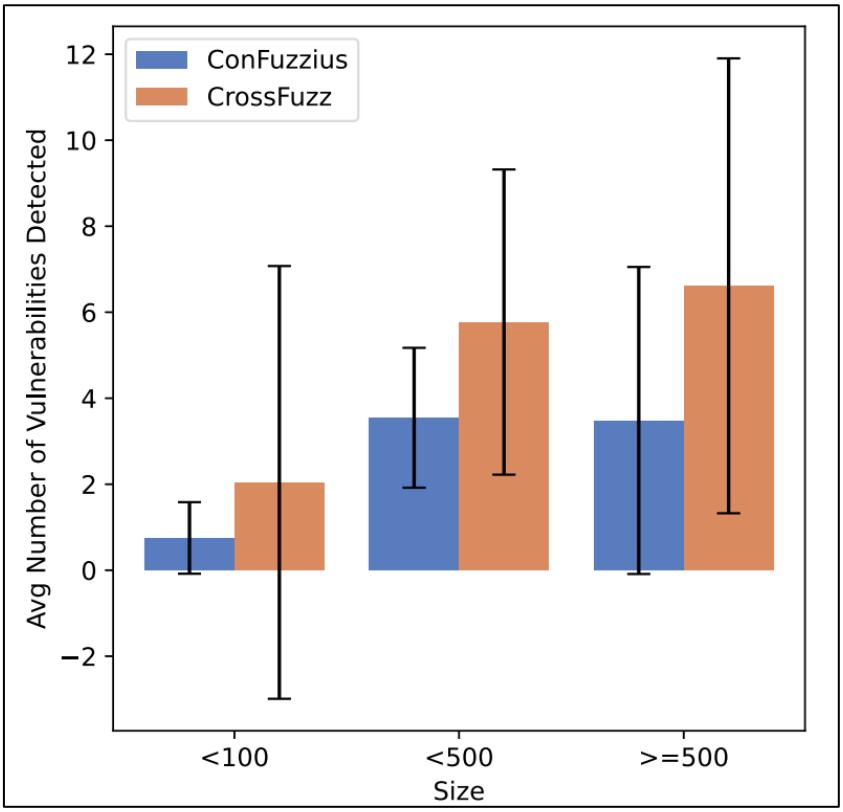
- Của nhóm tác giả: Để đánh giá ảnh hưởng của chiến lược đột biến chuỗi giao dịch, các nhà nghiên cứu đã thử nghiệm CrossFuzz với các xác suất kích hoạt chiến lược này khác nhau (0%, 20%, 50%, 80% và 100%). Kết quả cho thấy việc kích hoạt chiến lược đột biến chuỗi giao dịch dựa trên ICDF giúp cải thiện độ bao phủ mã của CrossFuzz, với xác suất tối ưu là 80%.

- Kết quả cho RQ2:



Comparison of different fuzzing tools in terms of coverage

CrossFuzz cho thấy hiệu quả vượt trội so với các fuzzer khác về độ bao phủ mã và khả năng phát hiện lỗ hổng. Độ bao phủ mã bytecode trung bình của CrossFuzz (79.92%) cao hơn ConFuzzius (72.11%), xFuzz (69.34%) và sFuzz (43.03%). Về số lượng lỗ hổng được phát hiện, CrossFuzz phát hiện được trung bình 6.11 (±5.08) lỗ hổng cho mỗi hợp đồng, cao hơn 1.82 lần so với ConFuzzius (3.36 ±3.52). CrossFuzz đạt hiệu quả cao hơn ở các hợp đồng có kích thước lớn do khả năng bao phủ nhiều mã hơn, đặc biệt là các hàm cốt lõi phức tạp thường dễ chứa lỗ hổng.



Comparison of different fuzzing tools in terms of number of vulnerabilities detected.

Vulnerability Type	Number of vulnerabilities detected	
	ConFuzzius	CrossFuzz
Arbitrary Memory Access	20	38
Assertion Failure	1712	2189
Integer Overflow	771	955
Reentrancy	15	672
Transaction Order Dependency	229	944
Block Dependency	3477	4537
Unhandled Exceptions	423	2756
Unsafe Delegate Call	0	0
Leaking Ether	15	1
Locking Ether	0	0
Unprotected Self-Destruct	0	0
Sum	6662	12092

Comparison of the number of vulnerabilities detected by ConFuzzius and CrossFuzz

- Kết quả cho RQ3:

Size	Statements in Constructor		Methods	Avg Coverage	Avg Number of Vulnerabilities Detected
	# Total	# Contract Conversions			
<100	13	0	CrossFuzz$_{zero}$	79.62 (±21.69)	1.88 (±1.62)
			CrossFuzz$_{random}$	79.69 (±21.75)	1.92 (±1.60)
			CrossFuzz	80.06 (±21.88)	2.04 (±1.62)
<500	726	68	CrossFuzz$_{zero}$	77.49 (±18.31)	4.88 (±4.57)
			CrossFuzz$_{random}$	80.09 (±17.99)	5.42 (±4.80)
			CrossFuzz	81.63 (±18.07)	5.77 (±4.97)
>=500	332	62	CrossFuzz$_{zero}$	71.11 (±18.89)	5.29 (±4.83)
			CrossFuzz$_{random}$	73.56 (±18.22)	6.19 (±5.14)
			CrossFuzz	77.01 (±76.61)	6.61 (±5.19)
Sum	1071	130	CrossFuzz$_{zero}$	75.58 (±18.98)	4.83 (±4.60)
			CrossFuzz$_{random}$	77.97 (±18.56)	5.46 (±4.88)
			CrossFuzz	79.92 (±18.34)	5.82 (±5.08)

Comparison of CrossFuzz with different contractor parameter generation strategies

CrossFuzz được thử nghiệm với ba chiến lược tạo tham số constructor: sử dụng ***CrossFuzzrandom*** và ***CrossFuzzzero***(địa chỉ ngẫu nhiên, địa chỉ zero) và theo dõi đường dẫn lan truyền dữ liệu. Kết quả cho thấy việc theo dõi đường dẫn lan truyền dữ liệu để tạo tham số constructor giúp CrossFuzz đạt được độ bao phủ mã và số lượng lỗ hổng phát hiện cao hơn so với hai chiến lược còn lại. Đặc biệt, CrossFuzz cho thấy hiệu quả rõ rệt ở các hợp đồng có kích thước lớn (>= 500 dòng) và có nhiều câu lệnh chuyển đổi hợp đồng trong constructor.

3. Đánh giá kết quả thực nghiệm của nhóm

- Về cơ bản, kết quả thực nghiệm của nhóm cho ra kết quả trả lời cơ bản tương tự trong bài báo mà nhóm tác giả đã đưa, chỉ có một số vấn đề còn tồn đọng như sau:

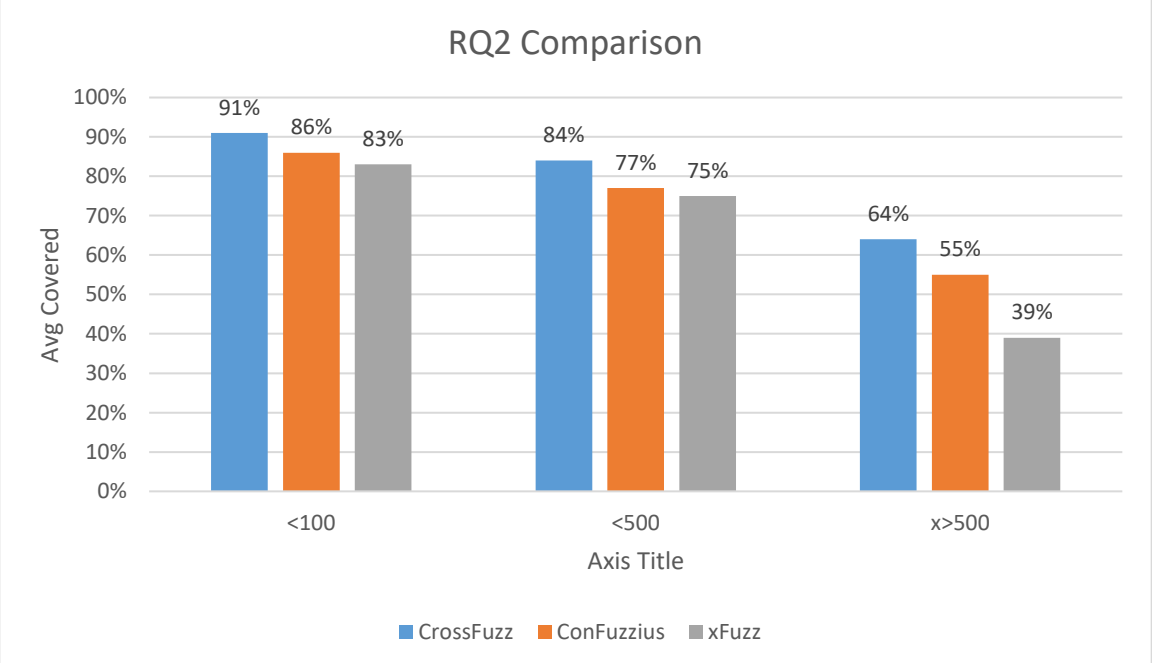
+ RQ1: Liệu chiến lược đột biến chuỗi giao dịch có cải thiện khả năng bao phủ mã không? Nếu có, xác suất tối ưu để kích hoạt chiến lược này là bao nhiêu?

⇨ Đối với mục RQ1 này, tệp dữ liệu của nhóm chỉ thật sự thay đổi kết quả đối với nhóm dữ liệu trên 500 dòng nhưng chỉ giao động trong ~1%, có thể vì tệp dữ liệu được tham khảo từ AI nên nó không đủ thực tế để cải thiện khả năng bao phủ mã khi thay đổi chiến lược đột biến chuỗi giao dịch

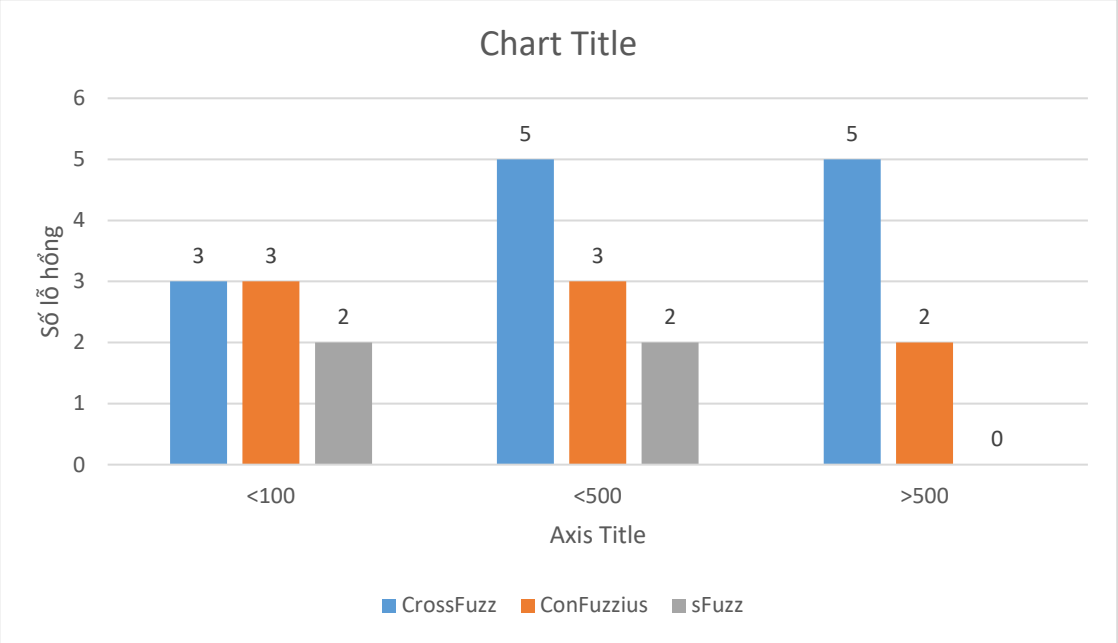
+ RQ2: Liệu CrossFuzz có thể đạt được khả năng bao phủ mã cao hơn và phát hiện lỗ hổng tốt hơn so với các phương pháp fuzz testing hiện đại không?

⇨ Đây là mục RQ mà tệp dữ liệu của nhóm trả lại kết quả tương đồng nhất với nhóm tác giả. Cụ thể sẽ như sau:

- Khả năng bao phủ mã: Độ bao phủ mã bytecode trung bình của CrossFuzz (79.67%) cao hơn ConFuzzius (72.67%), xFuzz (65.67%)



- Khả năng phát hiện lỗ hổng: số lượng lỗ hổng được phát hiện, CrossFuzz phát hiện được trung bình 4.33 lỗ hổng cho mỗi hợp đồng, cao hơn 1.625 lần so với ConFuzzius



+ Các loại lỗ hổng CrossFuzz thường phát hiện được

⇨ Với tệp 40 bản đa hợp đồng của nhóm thì có 3 lỗ hổng mà nhóm thường xuyên phát hiện được nhất chính là **Uncheck Return Value, Reentrency và Interger Overflow.** Về kết quả này bản thân nhóm nghĩ là chưa đủ khái quát bởi vì tập dữ liệu này được nhóm phát triển nhờ AI can thiệp khá nhiều, nó sẽ không đầy đủ các yếu tố như các hợp đồng thật sự được triển khai và chủ yếu các là từ cùng một số hợp đồng cơ bản phát triển lên nên chưa gặp được những lỗi khác hoặc chỉ gặp 1 vài lần rất ít như lỗi transaction_order_dependency

Tên lỗi	Số lần detect được
arbitrary_memory_access	X
assertion_failure	X
block_dependency	X
integer_overflow	26
leaking_ether	X
locking_ether	X
transaction_order_dependency	4
unchecked_return_value	34

unprotected_selfdestruct	X
unsafe_delegatecall	X
reentrancy	11

D. Hướng phát triển

Nhóm tác giả đã đưa ra một số cải tiến trong tương lai như sau

- Mở rộng khả năng áp dụng bằng cách hỗ trợ kiểm thử các trường hợp phức tạp hơn như hợp đồng gọi hợp đồng khác thông qua địa chỉ.

- Nâng cao hiệu suất bằng cách tối ưu hóa thuật toán tạo chuỗi giao dịch và xử lý các trường hợp đặc biệt.

- Cải thiện khả năng phát hiện lỗ hổng bằng cách tinh chỉnh cơ chế phân tích và xác định lỗ hổng.

Tính tới thời điểm hiện tại (12/2024), những cải tiến trên vẫn chưa được thực hiện, và phiên bản CrossFuzz được công khai trên Github vẫn là phiên bản mới nhất. Nhóm sẽ cố gắng dành thời gian để nghiên cứu thêm và triển khai các ý tưởng trên (nếu được).

---------HẾT---------