**MỤC LỤC**

[1. Giới thiệu 2](#_Toc184472431)

[2. Động lực phát triển 4](#_Toc184472432)

[3. Background 4](#_Toc184472433)

[3.1. Hợp đồng thông minh Ethereum 4](#_Toc184472434)

[3.2. Kiểm thử Fuzzing và ConFuzzius 4](#_Toc184472435)

[4. Phương pháp tiếp cận của CrossFuzz 5](#_Toc184472436)

[4.1. Tạo tham số Constructor 6](#_Toc184472437)

[4.2. Phân tích dòng dữ liệu giữa các hợp đồng (ICDF) 7](#_Toc184472438)

[4.3. Cross-contract fuzz testing 9](#_Toc184472439)

[5. Thiết kế và đánh giá thí nghiệm 12](#_Toc184472440)

[5.1. Thiết kế thí nghiệm 12](#_Toc184472441)

[5.1.1. Tập dữ liệu: 12](#_Toc184472442)

[5.1.2. Chỉ số đánh giá: 12](#_Toc184472443)

[5.1.3. Hệ thống cơ sở (Baselines): 12](#_Toc184472444)

[5.1.4. Thiết lập tham số: 12](#_Toc184472445)

[5.2. Đánh giá 13](#_Toc184472446)

[5.2.1. Kết quả cho RQ1: 13](#_Toc184472447)

[5.2.2. Kết quả cho RQ2: 13](#_Toc184472448)

[5.2.3. Kết quả cho RQ3: 13](#_Toc184472449)

[5.3. Thảo luận 13](#_Toc184472450)

[5.3.1. Các mối đe dọa đến tính hợp lệ 13](#_Toc184472451)

[5.3.2. Các cải tiến trong tương lai 14](#_Toc184472452)

[6. Công việc liên quan 16](#_Toc184472453)

[7. Kết luận 17](#_Toc184472454)

# 1. Giới thiệu

Hợp đồng thông minh Ethereum được sử dụng rộng rãi trong tài chính, y tế, IoT, nhưng tiềm ẩn nhiều lỗ hổng bảo mật gây tổn thất lớn (ví dụ: vụ DAO 2016 làm thất thoát 60 triệu Ether). Tính đến 1/7/2023, đã có 1.109 sự cố bảo mật, thiệt hại 30 tỷ USD. Do hợp đồng thông minh không thể chỉnh sửa sau triển khai đã dẫn đến việc phát hiện lỗ hổng trở nên cực kỳ quan trọng và cấp thiết.

Nghiên cứu đã đạt được nhiều tiến bộ đáng kể trong việc phát hiện các lỗ hổng bảo mật hợp đồng thông minh dựa trên các kỹ thuật khác nhau, bao gồm phân tích tĩnh (Static analysis), thực thi biểu tượng (Symbolic execution), học máy (Machine learning) và kiểm thử fuzz. Tuy nhiên, hầu hết các phương pháp hiện tại chủ yếu tập trung phân tích các hợp đồng thông minh riêng lẻ mà không xem xét các mối quan hệ phụ thuộc và các lời gọi hàm giữa các hợp đồng, dẫn đến tỷ lệ dương tính giả và âm tính giả cao.

Khi các yêu cầu và chức năng phức tạp hơn, hợp đồng thông minh tiến hóa từ dạng riêng lẻ sang giao dịch chéo, làm tăng nguy cơ lỗ hổng. Lỗ hổng giao dịch chéo xuất hiện trong chuỗi giao dịch giữa hai hoặc nhiều hợp đồng. Theo Liao và cộng sự, số hợp đồng bị ảnh hưởng bởi lỗ hổng này tăng từ tháng 11/2020 đến tháng 12/2022, gây tổn thất 81,2 triệu USD. Các lỗ hổng này đặt ra những thách thức lớn hơn trong việc phân tích và phát hiện, vì không gian tìm kiếm lớn hơn rất nhiều so với các hợp đồng riêng lẻ.

Để phát hiện lỗ hổng giao dịch chéo, Xue và cộng sự đề xuất phương pháp Clairvoyance, xây dựng đồ thị luồng điều khiển (CFG) và luồng dữ liệu (DFG) để theo dõi đường dẫn lỗ hổng reentrancy, sử dụng thực thi biểu tượng để kiểm tra tính khả thi và giảm dương tính giả. Liao và cộng sự phát triển SmartDagger, giải mã bytecode hợp đồng và phục hồi thông tin ngữ nghĩa bằng học sâu, phát hiện lỗ hổng như reentrancy và phụ thuộc thời gian.

Mặc dù các phương pháp phân tích tĩnh cho thấy hiệu quả cao hơn trong việc phát hiện lỗ hổng bảo mật, nhưng chúng bỏ qua tác động của chuỗi giao dịch lên trạng thái của hợp đồng, dẫn đến kết quả dương tính giả và âm tính giả. Các lần thực thi giao dịch có thể thay đổi giá trị của các biến trạng thái, từ đó ảnh hưởng đến các giao dịch tiếp theo. Do đó, các phương pháp phân tích tĩnh có độ chính xác thấp hơn trong việc phát hiện lỗ hổng bảo mật kích hoạt bởi chuỗi giao dịch.

Các phương pháp kiểm thử fuzz phát hiện lỗ hổng giao dịch chéo hợp đồng tạo động các chuỗi giao dịch, cải thiện khả năng phát hiện lỗ hổng. Ví dụ, Xue và cộng sự đã đề xuất công cụ kiểm thử fuzz *xFuzz*, xây dựng trên *sFuzz*. *xFuzz* triển khai các hợp đồng liên kết và sử dụng mô hình học máy để tính toán điểm ưu tiên cho các hàm thực thi. Kết quả cho thấy xFuzz giảm 80% thời gian phát hiện so với sFuzz. Tuy nhiên, các phương pháp fuzz hiện tại gặp hai vấn đề chính:

**L1: Khởi tạo sai các thông số của constructor**: Các phương pháp này thường khởi tạo hợp đồng với giá trị mặc định, bỏ qua việc chỉ định chính xác địa chỉ hợp đồng phụ thuộc trong constructor. Điều này dẫn đến các lỗi khi gọi hàm giao dịch chéo, ngăn cản việc kiểm thử các đoạn mã sau đó.

**L2:** **Bỏ qua các hàm quan trọng khi tạo chuỗi giao dịch**: Các phương pháp như *xFuzz* và *ILF* giảm số lượng tổ hợp chuỗi giao dịch cần thực thi bằng mô hình học máy. Tuy nhiên, hiệu suất của các mô hình này có thể bị ảnh hưởng do tập dữ liệu không cân bằng, nhãn sai hoặc thiết lập tham số không phù hợp. Điều này khiến các hàm quan trọng có khả năng kích hoạt lỗ hổng bị bỏ qua.

Để khắc phục, nhóm tác giả đã giới thiệu CrossFuzz, công cụ kiểm thử fuzz cải tiến để phát hiện lỗ hổng giao dịch chéo hợp đồng, nâng cao độ bao phủ mã và hiệu quả phát hiện. CrossFuzz giải quyết vấn đề L1 bằng cách tạo tham số phù hợp cho constructor, phân tích Luồng Dữ Liệu Giữa Các Hợp Đồng (ICDF) để xác định việc sử dụng và mối quan hệ biến trạng thái, và tối ưu hóa chiến lược đột biến chuỗi giao dịch để bao phủ các nhánh chưa được khám phá (giải quyết L2). Điều này giúp cải thiện hiệu suất kiểm thử fuzz.

Nhóm tác giả đã triển khai CrossFuzz như một phần mở rộng của ConFuzzius và tiến hành thí nghiệm trên một tập dữ liệu gồm 396 hợp đồng thông minh. Kết quả thực nghiệm cho thấy, so với xFuzz, sFuzz và ConFuzzius, CrossFuzz cải thiện độ bao phủ bytecode từ 7,81% đến 36,89% và phát hiện được nhiều lỗ hổng bảo mật hơn.

Tóm lại, các đóng góp chính sau đây đã được cung cấp:

1. CrossFuzz là công cụ kiểm thử fuzz giúp phát hiện lỗ hổng giao dịch chéo hợp đồng, sử dụng phân tích luồng dữ liệu và tối ưu hóa chiến lược đột biến để nâng cao hiệu quả phát hiện lỗ hổng.

A screenshot of a computer code

Description automatically generated

Hình 1.  Một ví dụ về [Smart Contract](https://www.sciencedirect.com/topics/computer-science/smart-contract).

2. Thí nghiệm trên 396 hợp đồng thông minh từ EtherScan cho thấy CrossFuzz tăng độ bao phủ bytecode thêm 10,58% và phát hiện nhiều lỗ hổng bảo mật hơn 1,82 lần so với ConFuzzius.

3. CrossFuzz được phát hành dưới dạng mã nguồn mở tại [GitHub](https://github.com/yagol2020/CrossFuzz) để hỗ trợ nghiên cứu trong tương lai.

Phần còn lại của bài báo được tổ chức như sau: Mục 2 trình bày động lực nghiên cứu. Mục 3 giới thiệu nền tảng lý thuyết. Mục 4 giới thiệu ba bước quan trọng của CrossFuzz, bao gồm tạo tham số cho constructor, phân tích luồng dữ liệu giữa các hợp đồng, và kiểm thử fuzz giao dịch chéo hợp đồng. Mục 5 trình bày thiết kế thực nghiệm và phân tích kết quả. Mục 6 xem xét các nghiên cứu liên quan. Cuối cùng, Mục 7 đưa ra kết luận.

# 2. Động lực nghiên cứu

- Nói về vấn đề các phương pháp fuzz testing hiện tại gặp khó khăn trong việc phát hiện lỗ hổng giữa các hợp đồng thông minh giao tiếp chéo

- Đưa ra ví dụ rằng sFuzz và Confuzzius không bao phủ được một số hàm do các vấn đề trong việc xử lý địa chỉ mặc định và các hợp đồng phụ. Còn xFuzz sử dụng mô hình học máy để giảm số lượng chuỗi giao dịch cần thử, nhưng vẫn bỏ qua một số hàm

=> CrossFuzz giúp giải quyết các thách thức đã được đưa ra bằng cách tập trung vào các chiến lược tạo tham số của hàm tạo và đột biến chuỗi giao dịch để cải thiện khả năng bao phủ mã và phát hiện lỗ hổng

# 3. Nền tảng lý thuyết

## 3.1. Hợp đồng thông minh Ethereum

- Hợp đồng thông minh Ethereum là các chương trình máy tính chạy trên nền tảng chuỗi khối Ethereum. Solidity là ngôn ngữ lập trình được sử dụng để viết các hợp đồng thông minh Ethereum, được biên dịch thành mã byte bởi các trình biên dịch như solc và được thực thi bởi Máy ảo Ethereum (EVM). Cụ thể, hợp đồng thông minh Solidity bao gồm nhiều hợp đồng, mỗi hợp đồng bao gồm một số biến trạng thái và chức năng. Các hàm được thực thi thông qua các giao dịch, bao gồm người gửi, chữ ký hàm, giá trị tham số hàm và lượng ether mang theo

- Trong paper thì **f** sẽ đại diện cho **các giao dịch (function)**, **p** đại diện cho **giá trị của các tham số (parameter)** trong hàm

## 3.2. Kiểm thử Fuzzing và ConFuzzius

**-** Kiểm thử fuzzing tạo các test case với giá trị ngẫu nhiên và gửi chúng để thực thi chương trình cần kiểm tra rồi dựa trên các điều kiện đường dẫn thu thập được, nó xác định xem có lỗ hổng bảo mật hoặc hành vi bất thường nào bị kích hoạt không. Sau đó, các test case con được tạo ra theo thuật toán di truyền và tiếp tục được gửi để kiểm tra chương trình, cho đến khi đạt điều kiện dừng (ví dụ: thời gian kiểm thử đã định)

- Giới thiệu về ConFuzzius ( Theo ChatGPT dịch: ConFuzzius kết hợp sáng tạo các kỹ thuật giải quyết ràng buộc (constraint-solving) với phương pháp fuzz testing để giải quyết vấn đề "magic byte" trong fuzz testing, nơi các nhánh có điều kiện nghiêm ngặt khó được bao phủ bằng các đầu vào ngẫu nhiên, làm giảm khả năng bao phủ mã của quá trình fuzz testing. ConFuzzius cũng xem xét sự phụ thuộc Read-After-Write (RAW) giữa các biến trạng thái, cải thiện hàm fitness để tối ưu chiến lược chọn seed, và chỉ kết nối các chuỗi giao dịch thỏa mãn điều kiện RAW trong giai đoạn lai ghép chuỗi giao dịch. Sau đó, chuỗi giao dịch được chia thành hai chuỗi mới để tạo test case con )

- Nêu ra 2 hạn chế của ConFuzzius:

+ Không hỗ trợ phát hiện lỗ hổng trong các tình huống hợp đồng giao tiếp chéo.

+ Sử dụng các giá trị mặc định cho tham số của hàm khởi tạo khi triển khai hợp đồng, bỏ qua ảnh hưởng của các cuộc gọi chéo giữa các hợp đồng đối với dòng dữ liệu

=> CrossFuzz ra đời là phương pháp mở rộng của ConFuzzius

# 4. Phương pháp tiếp cận của CrossFuzz

- CrossFuzz là một phương pháp kiểm thử fuzzing để phát hiện lỗ hổng bảo mật giữa các hợp đồng thông minh, như thể hiện trong Hình 5.

- 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. Tiếp theo, CrossFuzz phân tích ICDF (Luồng dữ liệu giữa gác hợp đồng) để hướng dẫn việc biến đổi các chuỗi giao dịch. 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.

A diagram of a construction function

Description automatically generated

**Fig. 5. The framework of CrossFuzz.**

- Trong phần sau đây, sẽ trình bày chi tiết ba phần quan trọng của CrossFuzz, tức là tạo tham số Constructor, [phân tích luồng dữ liệu](https://www.sciencedirect.com/topics/computer-science/data-flow-analysis) giữa các hợp đồng (ICDF) và Cross-contract fuzz testing.

## 4.1. Tạo tham số Constructor

- Trong phần này, họ sẽ mô tả quá trình tạo các tham số constructor, 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.

A screenshot of a computer

Description automatically generated

Hình 2. 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.

- Ví dụ, các dòng từ 20-24 trong Hình 1 thiết lập địa chỉ multisig và các biến khác của hợp đồng Hold, trong khi dòng 25 khởi tạo hợp đồng phụ thuộc PM. Nếu các tham số constructor không được cung cấp trong khi triển khai hợp đồng, EVM sẽ điền chúng bằng các giá trị mặc định, như 0 cho các tham số kiểu uint và địa chỉ không hợp lệ cho các tham số kiểu địa chỉ. Tuy nhiên, nếu hợp đồng phụ thuộc được khởi tạo với địa chỉ không hợp lệ, các công cụ fuzzing sẽ không thể gọi các hàm trong hợp đồng đó, dẫn đến các bất thường trong việc thực thi giao dịch và làm giảm mức độ bao phủ mã nguồn của hợp đồng.

- Kỹ thuật Phân tích Dòng Thông Tin (Information-Flow Analysis - IFA) đảm bảo an ninh thông tin bằng cách phân tích tính hợp pháp của việc truyền dữ liệu. 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 1. 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 1 đến EVM (Ethereum Virtual Machine), 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.

## 4.2. 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 hợp đồng thông minh thay đổi các biến trạng thái của hợp đồng thông qua các cuộc gọi hàm (tức là các giao dịch) để thay đổi trạng thái của hợp đồng【26】【27】. Nếu giá trị của các biến trạng thái bị thay đổi bởi các giao dịch, việc thực thi các giao dịch tiếp theo sẽ bị ảnh hưởng. Hơn nữa, số lượng kết hợp chuỗi giao dịch có thể tăng theo cấp số mũ do các cuộc gọi hàm giữa các hợp đồng, vì số lượng các hàm sẽ tăng lên.

- 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【21】【26】【28】. 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.

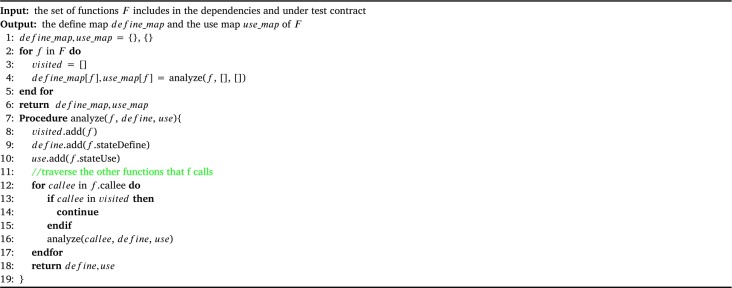
- Đầu tiên, CrossFuzz phân tích mối quan hệ gọi giữa các hợp đồng. Các câu lệnh trong một hàm được duyệt qua, và các câu lệnh gọi hàm được phân tích để lấy tên của các hợp đồng và hàm được gọi. Lưu ý rằng nếu một hợp đồng áp dụng bộ sửa đổi hàm fm cho hàm f, bộ sửa đổi sẽ được thực thi trước khi hàm 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 3. Call relationships between PM and hold*.

A diagram of a business process

Description automatically generated

Hình 3. Call relationships between PM and hold

**(Fig. 6.****Call relationships between PM and hold.)**



Hình 4. Analyze the definition and usage of state variables by functions

**Algorithm 2. Analyze the definition and usage of state variables by functions.**

- 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.

- CrossFuzz thu thập bản đồ định nghĩa và sử dụng của các hàm như được mô tả trong Thuật toán 2. 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 2 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.

## 4.3. 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.

- 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 4.1.

- 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 4.2 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. Biến cục bộ được định nghĩa hoặc sử dụng trong một giao dịch duy nhất và không thể ảnh hưởng trực tiếp đến việc thực thi các giao dịch tiếp theo. Vì vậy, khi đột biến chuỗi giao dịch, chúng tôi chỉ xem xét các biến trạng thái.

- 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.

- Mục tiêu là giải quyết giao dịch ngoại lệ. Khi các giao dịch này được thực thi lại, có thể các điều kiện nhánh mới sẽ được đáp ứng, giúp khám phá thêm mã chưa được kiểm thử. Để giải quyết vấn đề của các giao dịch ngoại lệ, CrossFuzz tối ưu hóa chiến lược đột biến chuỗi giao dịch bằng cách tính điểm ưu tiên dựa trên ICDF. 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, chúng tôi 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 gồm ba phần như thể hiện trong phương trình (1):



- SDef(f, exp) là số lượng biến trạng thái được định nghĩa bởi hàm f. Chúng tôi xem xét rằng việc thay đổi lặp đi lặp lại một biến trạng thái cụ thể có thể dẫn đến các chuỗi giao dịch quá dài, làm giảm hiệu quả của kiểm thử fuzz. Do đó, SDef(f, exp) chỉ xem xét số lượng biến trạng thái chỉ được định nghĩa bởi hàm f thay vì các hàm khác trong chuỗi giao dịch. Như thể hiện trong phương trình (2), SDef(f, exp) được tính bằng cách trừ tập hợp các biến trạng thái được định nghĩa bởi các giao dịch trước giao dịch ngoại lệ exp (ký hiệu là VPreDef(exp)) khỏi tập hợp các biến trạng thái được định nghĩa bởi hàm f (ký hiệu là VDef(f)):



- Tương tự như SDef(f, exp), SProvide(f, exp) dùng để đếm số lượng biến trạng thái mà hàm f có thể cung cấp cho exp. Sự khác biệt là, so với SDef(f, exp), SProvide(f, exp) quan tâm nhiều hơn đến các biến trạng thái bổ sung mà hàm f cung cấp cho chuỗi giao dịch. Như thể hiện trong phương trình (3), SProvide(f, exp) đầu tiên lấy giao của VDef(f) và các biến trạng thái được sử dụng bởi exp (ký hiệu là VUse(exp)), sau đó trừ VPreDef(exp) từ kết quả này:



- SUse(f, exp) được sử dụng để tính số lượng biến trạng thái chỉ được sử dụng bởi hàm f chứ không phải các biến trạng thái cũng được sử dụng bởi exp. Chúng tôi coi SUse(f, exp) là một tác động phụ, và giá trị này càng lớn, khả năng kích hoạt ngoại lệ khi thực thi hàm f càng cao. Như thể hiện trong phương trình (4), SUse(f, exp) được tính bằng cách trừ VUse(f) khỏi VUse(exp):



- Lưu ý rằng VDef(f) và VUse(f) có thể được lấy từ ICDF của hàm f, trong khi VUse(exp) và VPreDef(exp) có thể được lấy thông qua phân tích động của việc thực thi giao dịch. Cụ thể, CrossFuzz sử dụng giao diện do ConFuzzius cung cấp để phân tích các thay đổi trong ngăn xếp trong quá trình thực thi exp hoặc các giao dịch khác. Phân tích này cho phép CrossFuzz thu thập các thao tác dữ liệu (tức là định nghĩa hoặc sử dụng) của các biến trạng thái trong các giao dịch cụ thể, từ đó xác định VUse(exp) và VPreDef(exp).

- 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.

# 5. Thiết kế và đánh giá thí nghiệm

Phần này trình bày chi tiết về thiết kế và đánh giá hiệu quả của CrossFuzz, một phương pháp phát hiện lỗ hổng trong các hợp đồng thông minh (smart contract) trên Ethereum. CrossFuzz tập trung vào việc khai thác các lỗ hổng liên hợp đồng (cross-contract vulnerabilities) dựa trên fuzz testing.

## 5.1. 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:

### 5.1.1. 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).

### 5.1.2. 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.

### 5.1.3. 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.

### 5.1.4. 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.

## 5.2. Đánh giá

### 5.2.1. Kết quả cho RQ1:

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%.

### 5.2.2. Kết quả cho RQ2:

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.

### 5.2.3. Kết quả cho RQ3:

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.

## 5.3. Thảo luận

Phần này phân tích các mối đe dọa đến tính hợp lệ của nghiên cứu từ ba khía cạnh: tính hợp lệ nội bộ, tính hợp lệ bên ngoài và tính hợp lệ cấu trúc, đồng thời đề xuất các cải tiến trong tương lai cho CrossFuzz.

### 5.3.1. Các mối đe dọa đến tính hợp lệ

#### 1. Tính hợp lệ nội bộ:

- Mối đe dọa: Liệu việc triển khai CrossFuzz có chính xác hay không.

- Giải pháp: CrossFuzz được triển khai dựa trên các công cụ mã nguồn mở, đảm bảo tính chính xác và minh bạch. Mô-đun phân tích tĩnh được xây dựng dựa trên Slither, một công cụ phân tích tĩnh cho hợp đồng thông minh được cập nhật thường xuyên và có độ tin cậy cao. Mô-đun fuzz testing được phát triển dựa trên framework ConFuzzius, với các tính năng mở rộng như triển khai đa hợp đồng, cài đặt tham số constructor và chiến lược đột biến chuỗi giao dịch.

Việc tham khảo các công cụ fuzz testing mã nguồn mở khác như sFuzz giúp cải thiện hiệu quả của quá trình đột biến test case và thống nhất việc đánh giá độ bao phủ bytecode.

#### 2. Tính hợp lệ bên ngoài:

- Mối đe dọa: Liệu kết quả thử nghiệm có thể được khái quát hóa hay không, dựa trên việc tập dữ liệu có đại diện hay không.

- Giải pháp:Do các nghiên cứu trước đây về phát hiện lỗ hổng đa hợp đồng chưa công khai tập dữ liệu, nhóm nghiên cứu đã lựa chọn ngẫu nhiên 500 hợp đồng thông minh từ EtherScan, một nền tảng phân tích cho Ethereum, để xây dựng tập dữ liệu. Việc so sánh CrossFuzz với ba công cụ fuzz testing khác (sFuzz, xFuzz và ConFuzzius), bao gồm cả xFuzz là công cụ duy nhất tập trung vào lỗ hổng đa hợp đồng, giúp tăng cường tính khái quát của kết quả. Nhóm nghiên cứu đã công khai CrossFuzz như một công cụ mã nguồn mở, tạo điều kiện cho các nghiên cứu tiếp theo sử dụng và đánh giá.

#### 3. Tính hợp lệ cấu trúc:

- Mối đe dọa: Liệu các chỉ số đánh giá được sử dụng có thể đánh giá toàn diện hiệu quả của CrossFuzz hay không.

- Giải pháp: Nghiên cứu sử dụng hai chỉ số đánh giá phổ biến trong fuzz testing hợp đồng thông minh: độ bao phủ bytecode và số lượng lỗ hổng được phát hiện. Ngoài ra, nghiên cứu cũng đánh giá hiệu quả thời gian của CrossFuzz, bổ sung cho việc đánh giá hiệu quả.

### 5.3.2. Các cải tiến trong tương lai

Mặc dù cho thấy hiệu quả vượt trội, CrossFuzz vẫn có thể được cải thiện trong các trường hợp sau:

#### 1. Các nhánh mã chỉ có thể được bao phủ bởi chuỗi giao dịch dài hơn:

- Hiện tại, độ dài tối đa của chuỗi giao dịch được tạo bởi CrossFuzz là 5, được kế thừa từ ConFuzzius.

- Việc tăng giá trị tham số này có thể nâng cao độ bao phủ mã nhưng cũng làm tăng chi phí thời gian của fuzz testing.

#### 2. Các nhánh mã chỉ có thể được bao phủ bởi giá trị mặc định của biến trạng thái:

- CrossFuzz chưa xem xét trường hợp giá trị mặc định của biến trạng thái trong constructor đã thỏa mãn điều kiện của nhánh mã.

- Các giá trị tham số được tạo dựa trên ICDF có thể ảnh hưởng đến độ bao phủ mã.

- Tuy nhiên, CrossFuzz cung cấp giao diện cho người dùng nhập thủ công các giá trị tham số constructor cụ thể.

#### 3. Các nhánh mã chỉ có thể được bao phủ bằng cách thực thi một số hàm nhiều lần:

- Do CrossFuzz chỉ chọn các hàm chưa được thực thi để thêm vào chuỗi giao dịch, nên trường hợp này có thể bị bỏ sót.

- Tương tự như trường hợp (2), người dùng có thể thay đổi chiến lược này thông qua cài đặt tham số.

Ngoài ra, CrossFuzz chưa hỗ trợ kiểm tra các trường hợp hợp đồng gọi các hợp đồng khác thông qua địa chỉ. Trong tương lai, nhóm nghiên cứu dự định sử dụng EtherScan để tìm kiếm các hợp đồng bên ngoài và lấy mã nguồn của chúng, qua đó mở rộng khả năng ứng dụng của CrossFuzz.

# 6. Nghiên cứu liên quan

- Fuzz testing sử dụng dữ liệu ngẫu nhiên làm đầu vào cho chương trình và phân tích các lỗ hổng bảo mật bằng cách phát hiện các ngoại lệ của chương trình [30]. Trong kiểm thử bảo mật hợp đồng thông minh, phương pháp fuzz testing triển khai hợp đồng cần kiểm tra, tạo ra các test case chứa chuỗi giao dịch, và gửi chúng đến hợp đồng thông minh để thực thi. Trong quá trình thực thi, các thay đổi dữ liệu, như stack và môi trường block, sẽ được thu thập. Dựa trên thông tin thu thập được, fuzzer có thể phân tích xem một lỗ hổng bảo mật có bị kích hoạt hay không.

- Jiang et al. [20] đã đề xuất ContractFuzzer, phương pháp thu thập thông tin thực thi hợp đồng bằng cách cài đặt lại EVM, tạo ra các test case dựa trên ABI và phát hiện các lỗ hổng dựa trên các mẫu lỗ hổng đã được xác định trước. Nguyen et al. [19] đã cải tiến chiến lược đột biến seed dựa trên ContractFuzzer và giữ lại các seed gần với các đường đi điều kiện chặt chẽ để tạo ra các test case con. Kết quả thử nghiệm cho thấy sFuzz cải thiện độ bao phủ mã và phát hiện nhiều lỗ hổng bảo mật hơn. Choi et al. [21] đã đề xuất Smartian, phương pháp xem xét ảnh hưởng của các chuỗi giao dịch đối với độ bao phủ mã, tạo ra các chuỗi giao dịch dựa trên các mối quan hệ sử dụng đã được định nghĩa và giữ lại các test case có thể bao phủ nhiều dòng dữ liệu hơn làm seed. Kết quả thử nghiệm cho thấy Smartian có thể bao phủ nhiều mã hợp đồng hơn một cách nhanh chóng. Xue et al. [24] đã xây dựng xFuzz dựa trên sFuzz để giải quyết các tình huống hợp đồng chéo. xFuzz triển khai các hợp đồng thông minh phụ thuộc lẫn nhau trong môi trường thử nghiệm và sử dụng học máy cùng các chỉ số đánh giá độ phù hợp để chọn các hàm cần thực thi nhằm giảm số lượng kết hợp chuỗi giao dịch.

- Phương pháp dựa trên fuzz testing yêu cầu triển khai và thực thi hợp đồng thông minh. Nó có ưu điểm là độ chính xác cao và khả năng tạo test case để tái tạo các lỗ hổng bảo mật. Tuy nhiên, các phương pháp hiện tại chủ yếu tập trung vào một hợp đồng duy nhất. Mặc dù xFuzz đã tối ưu hóa cho các tình huống hợp đồng chéo, nhưng nó vẫn có những hạn chế về hiệu suất mô hình và việc tạo test case, chẳng hạn như bỏ qua một số chức năng quan trọng vì mô hình dự đoán rằng không có lỗ hổng trong các chức năng quan trọng đó. Hơn nữa, các phương pháp hiện tại không tính đến các tham số của constructor. Mặc dù các giá trị ngẫu nhiên có thể được tạo ra làm giá trị tham số, nhưng các hợp đồng có thể bị khởi tạo sai bởi những giá trị ngẫu nhiên này, điều này làm giảm độ bao phủ mã và khả năng phát hiện lỗ hổng bảo mật của các fuzzer.

- Để giải quyết các hạn chế này, nhóm tác giả đã đề xuất CrossFuzz, phương pháp này giới thiệu hai kỹ thuật mới. ***Đầu tiên***, CrossFuzz theo dõi các đường dẫn truyền dữ liệu để tạo ra các tham số của constructor, từ đó nâng cao khả năng bao phủ mã của nó. ***Thứ hai***, CrossFuzz tối ưu hóa chiến lược đột biến của các chuỗi giao dịch dựa trên điểm ưu tiên chức năng và bổ sung thông tin về các biến trạng thái cần thiết cho các giao dịch ngoại lệ. Kết quả thí nghiệm cho thấy phương pháp của chúng tôi cải thiện đáng kể hiệu quả của fuzz testing.

# 7. Kết luận

- 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 cho thấy CrossFuzz vượt trội hơn so với các công cụ kiểm thử fuzz hiện có như sFuzz, ConFuzzius và xFuzz2910. 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.

**- Hướng phát triển trong tương lai:**

+ **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.

- CrossFuzz là một bước tiến đáng kể trong lĩnh vực kiểm thử fuzz cho hợp đồng thông minh. Việc tập trung vào các khía cạnh đặc thù của kiểm thử liên hợp đồng giúp CrossFuzz đạt được hiệu quả vượt trội so với các công cụ hiện có. Tuy nhiên, CrossFuzz vẫn cần được cải tiến để khắc phục những hạn chế hiện có và nâng cao khả năng ứng dụng trong thực tế.