ĐẠI HỌC QUỐC GIA HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

NT521.P12.ANTT

CROSSFUZZ: CROSS-CONTRACT FUZZING FOR SMART CONTRACT VULNERABILITY DETECTION.

NHÓM 2

NỘI DUNG

01. Sơ lược đề tài

02. CrossFuzz

03. Thiết kế và đánh giá thí nghiệm

04. Demo

01.

SƠ LƯỢC ĐỀ TÀI

1. SƠ LƯỢC ĐỀ TÀI

Tổng quan

- Đối tượng: Hợp đồng thông minh Ethereum. (smart contract Ethereum).

- Do hợp đồng thông minh không thể chỉnh sửa sau triển khai **=> Việc phát hiện lỗ hổng trở nên cực kỳ quan trọng và cấp thiết.**

- Các phương pháp hiện nay: Static analysis, Symbolic execution, Machine learning. Chỉ tập trung vào contract riêng lẻ mà không tính đến các mối quan hệ và lời gọi hàm giữa các contract

=> Tỷ lệ dương tính giả và âm tính giả cao.

1. SƠ LƯỢC ĐỀ TÀI

Các vấn đề:

- Fuzzing có thể phát hiện các lỗ hổng giao dịch chéo thông qua việc tạo ra chuỗi giao dịch động. Hiện tại gặp 2 vấn đề chính:

+ Khởi tạo sai các thông số của constructor (L1)

+ Bỏ qua các hàm quan trọng khi tạo chuỗi giao dịch (L2)

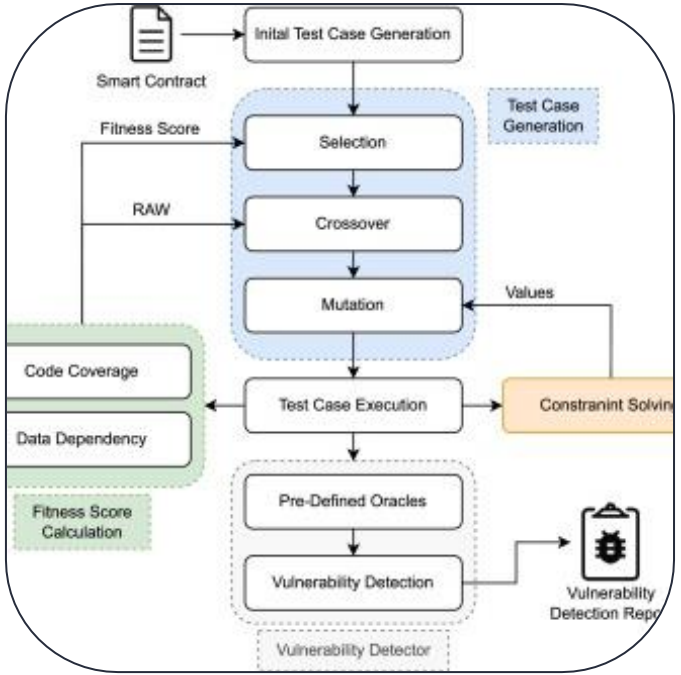
1. SƠ LƯỢC ĐỀ TÀI

Giải pháp:

- **CrossFuzz** (công cụ kiểm thử fuzz giúp phát hiện lỗ hổng giao dịch chéo hợp đồng) ra đời để giải quyết các vấn đề trên.

+ **Giải quyết vấn đề L1:** tạo tham số phù hợp cho constructor.

+ **Giải quyết vấn đề L2:** 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.

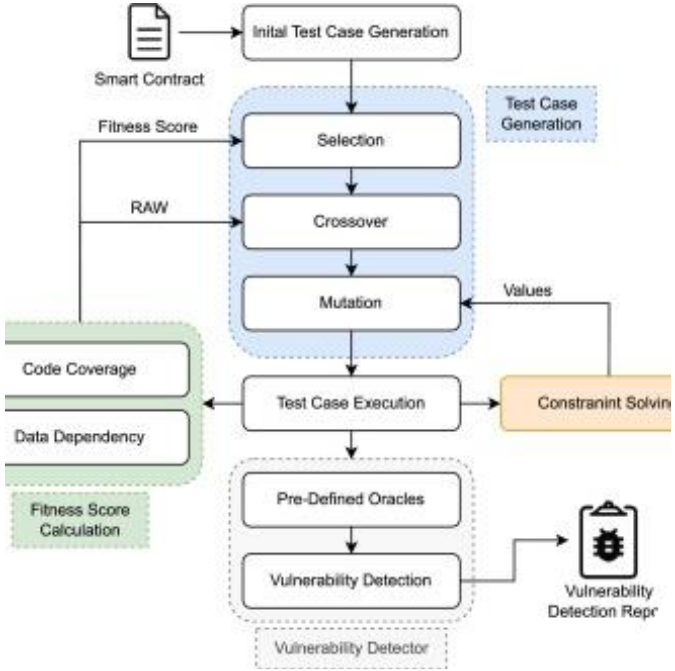


1. SƠ LƯỢC ĐỀ TÀI

Tổng quan

- **CrossFuzz** là công cụ kiểm thử fuzz testing 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.

- 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.



02.
CROSSFUZZ

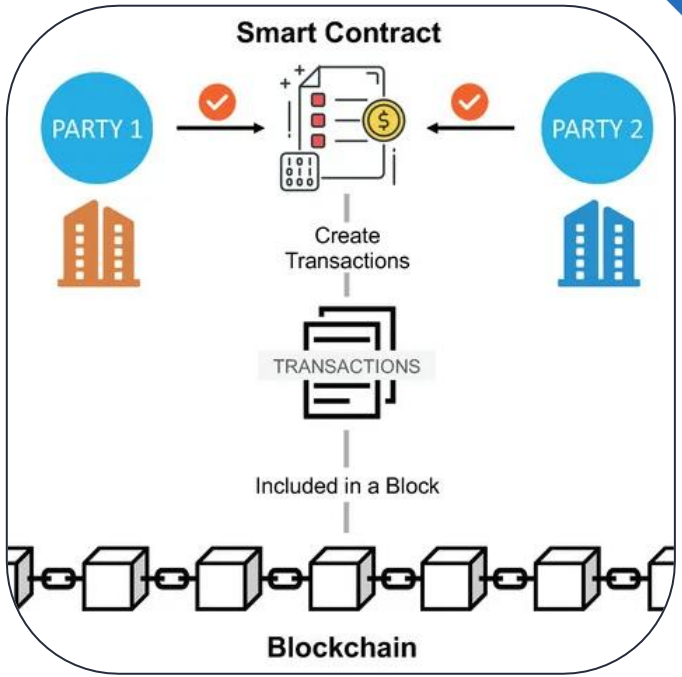
2. CROSSFUZZ

Ethereum

- Là một nền tảng blockchain phi tập trung

- Solidity là ngôn ngữ lập trình được sử dụng để viết các Ethereum smart contract, được biên dịch thành mã byte bởi các trình biên dịch như solc và được thực thi bởi Máy ảo Ethereum (EVM).

- Hợp đồng thông minh Solidity gồm nhiều hợp đồng, mỗi hợp đồng gồm: các biến trạng thái và chức năng. Các hàm được thực thi thông qua các giao dịch. Kí hiệu: **f** sẽ đại diện cho các giao dịch (function), **p** đại diện cho giá trị của các tham số (parameter) trong hàm.



2. CROSSFUZZ

ConFuzzius

- Kết hợp giải quyết ràng buộc với fuzz testing để khắc phục hạn chế "magic byte," giúp bao phủ các nhánh có điều kiện nghiêm ngặt khó tiếp cận.

- Cải tiến hàm fitness để tối ưu chọn seed, xử lý sự phụ thuộc Read-After-Write (RAW) giữa biến trạng thái. Chuỗi giao dịch sau đó được phân tách để tạo test case mới.

- Hạn chế:

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

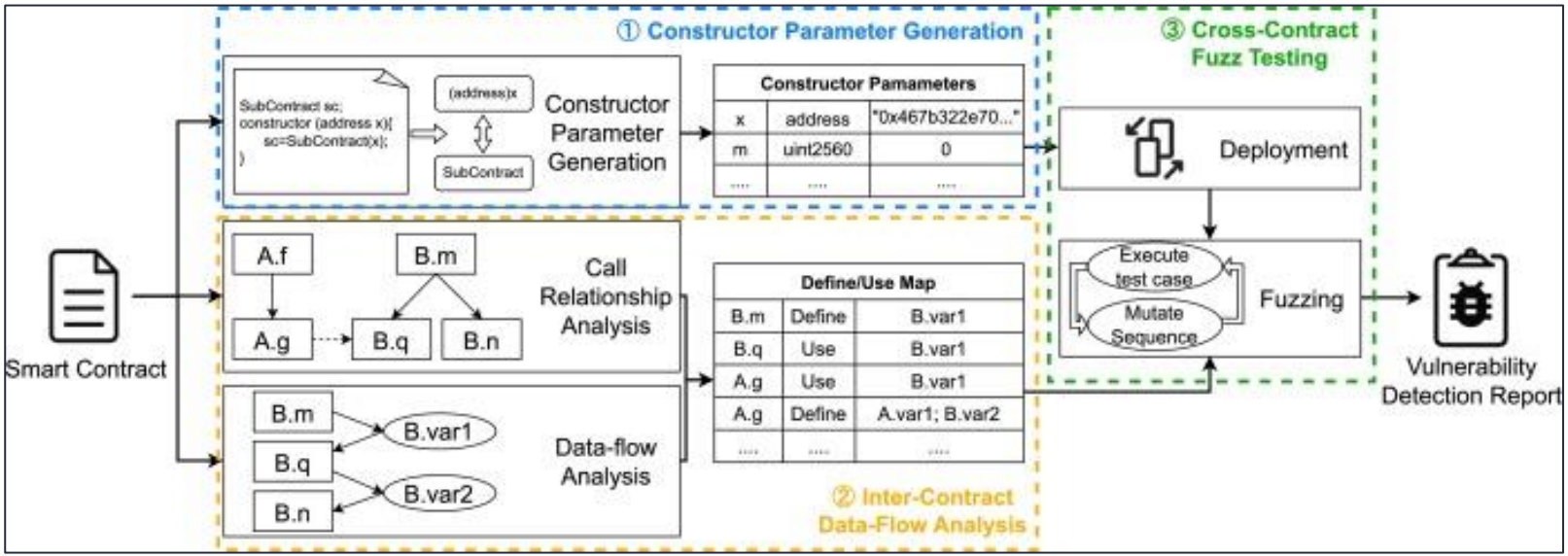
  + Sử dụng các giá trị mặc định cho tham số của hàm khởi tạo, bỏ qua ảnh hưởng của các cuộc gọi chéo giữa các hợp đồng đối với dòng dữ liệu

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

2. CROSSFUZZ

Phương pháp tiếp cận

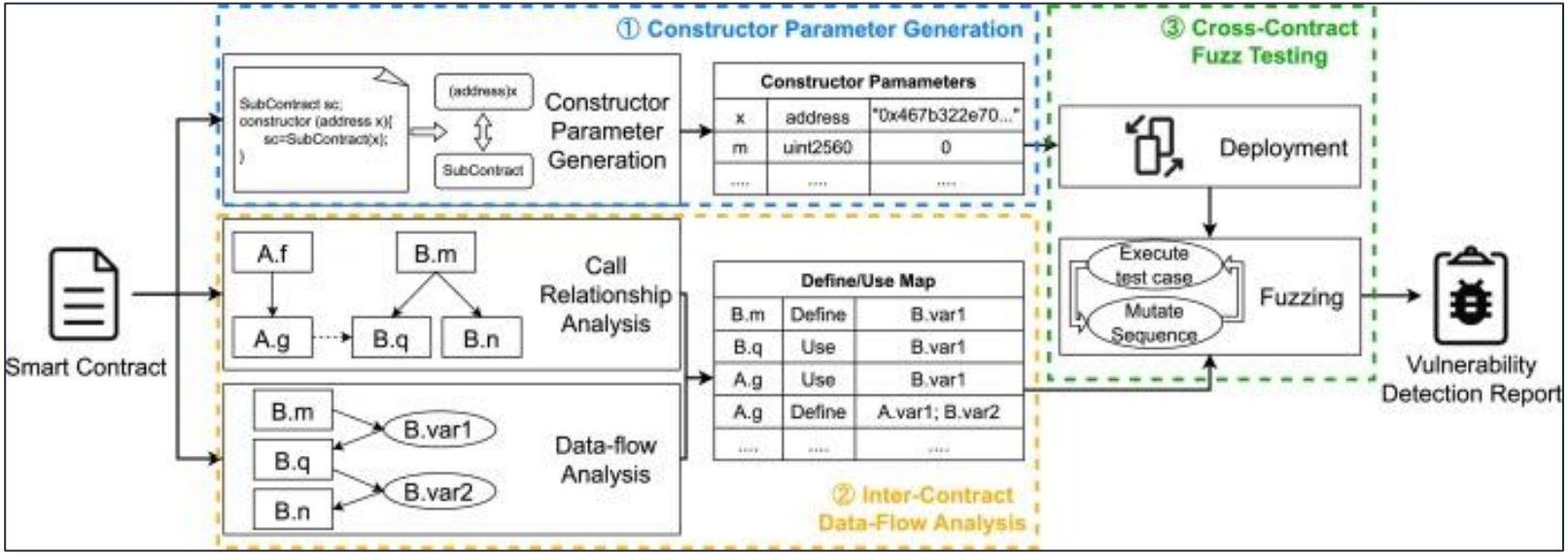
- CrossFuzz là một phương pháp kiểm thử fuzzing để phát hiện lỗ hổng bảo mật giữa các hợp đồng giao tiếp chéo:



2. CROSSFUZZ

Phương pháp tiếp cận

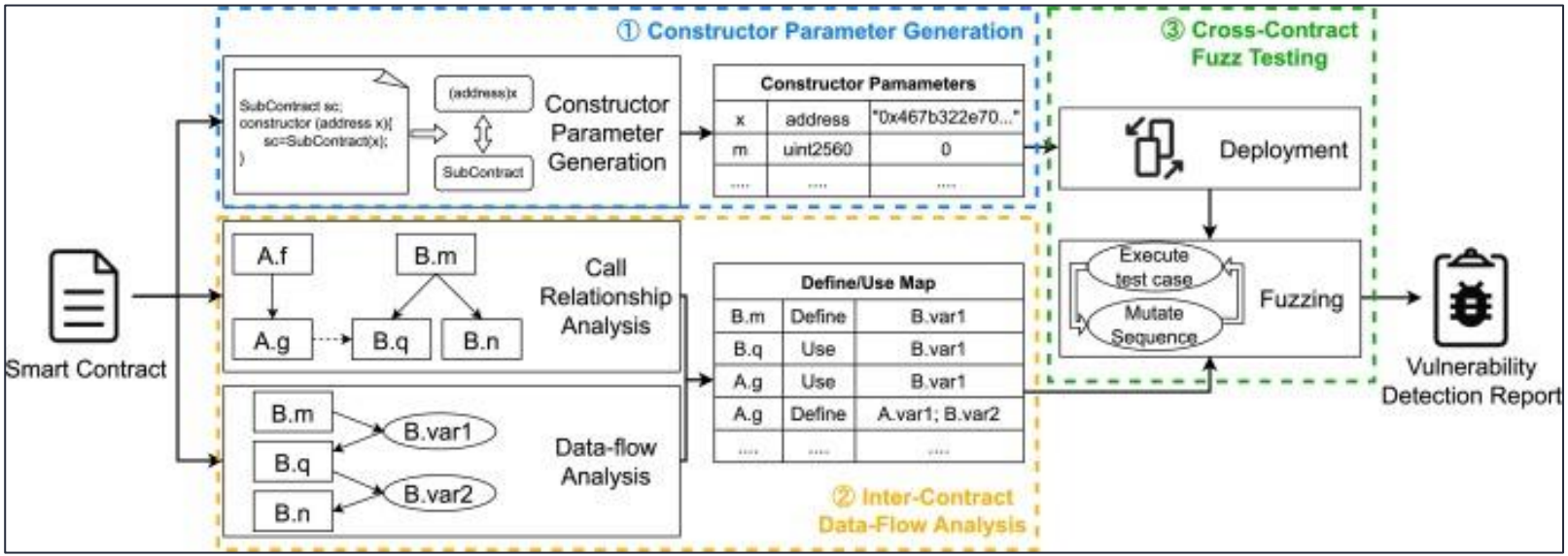
- **Đầu tiên**, CrossFuzz theo dõi các đường truyền dữ liệu của các tham số constructor, phân tích mối quan hệ giữa các tham số kiểu địa chỉ và các hợp đồng phụ thuộc, sau đó tạo các tham số cho constructor



2. CROSSFUZZ

Phương pháp tiếp cận

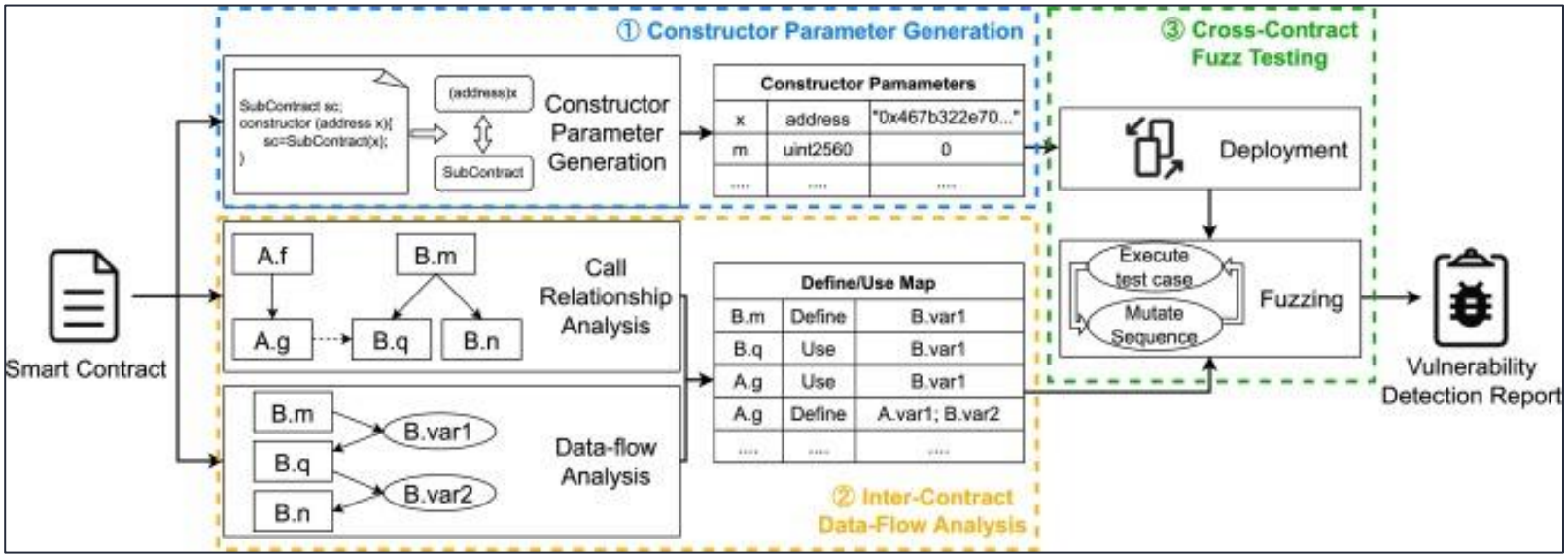
- **Tiếp theo**, CrossFuzz phân tích ICDF (Luồng dữ liệu giữa gác hợp đồng) để hướng dẫn việc biến đổi các chuỗi giao dịch.



2. CROSSFUZZ

Phương pháp tiếp cận

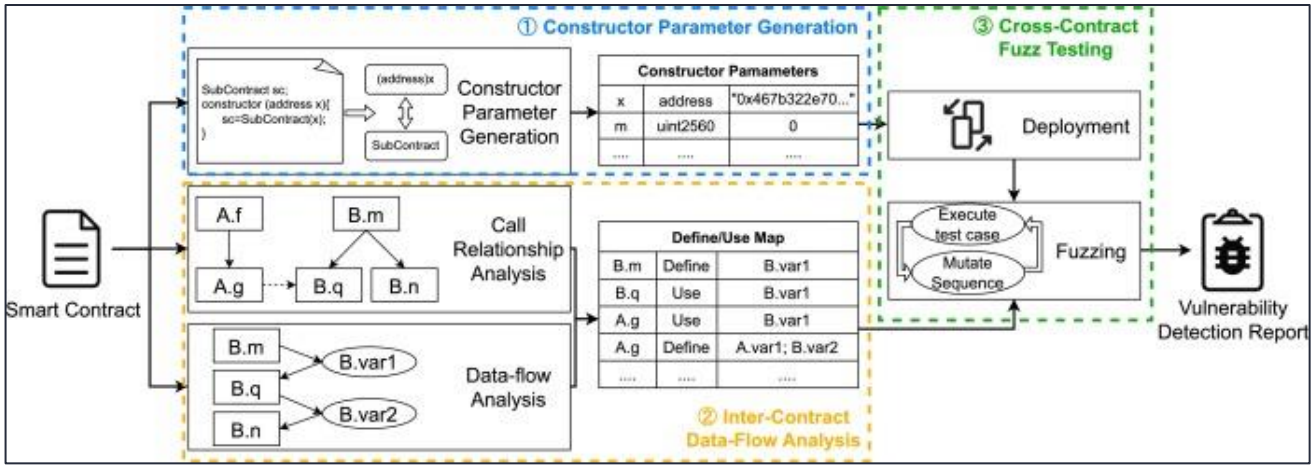
- **Cuối cùng**, CrossFuzz triển khai hợp đồng thông minh bằng các tham số constructor và thực hiện kiểm thử fuzzing



2. CROSSFUZZ

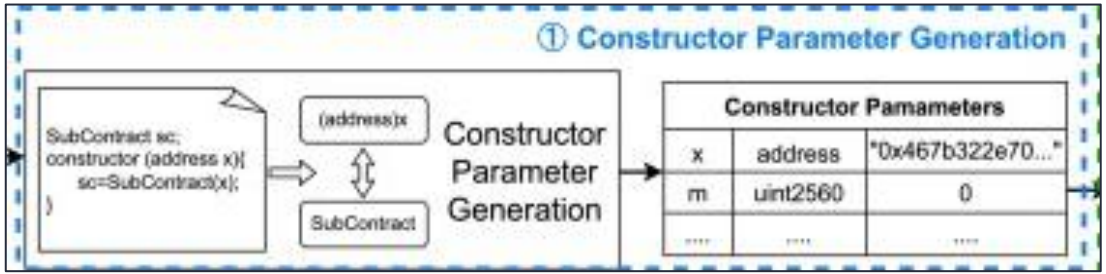
Phương pháp tiếp cận

- Trong quá trình 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.



2. CROSSFUZZ

Tạo tham số Constructor



- CrossFuzz theo dõi các đường truyền dữ liệu của các tham số kiểu địa chỉ trong constructor để xác định và ánh xạ chúng tới các hợp đồng phụ thuộc.

- Khi triển khai hợp đồng, nó tự động điền địa chỉ hợp đồng phụ thuộc (hoặc địa chỉ người tạo nếu không tìm thấy hợp đồng phụ thuộc tương ứng), còn các tham số khác được điền giá trị mặc định như trong fuzzing truyền thống.

2. CROSSFUZZ

Phân tích luồng dữ liệu liên hợp đồng (Inter-contract Data Flow - ICDF)

- Các công cụ fuzz testing hiện tại (điển hình là xFuzz và ILF giảm số lượng tổ hợp chuỗi giao dịch cần thực thi bằng mô hình học máy) thường bỏ qua việc ưu tiên các hàm liên quan đến biến trạng thái trong quá trình tạo chuỗi giao dịch. Điều này dẫn đến tập dữ liệu không cân bằng gây lãng phí tài nguyên và giảm hiệu quả kiểm tra của mô hình

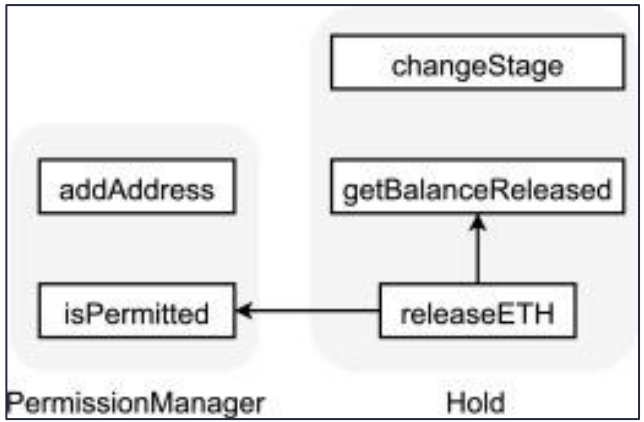
=> CrossFuzz sẽ tiếp cận phân tích ICDF theo hướng xem xét cách các biến trạng thái được định nghĩa, sử dụng trong hàm của các hợp đồng riêng lẻ rồi từ đó nó phân tích quan hệ giữa các hàm sử dụng biến trạng thái đó với các hàm khác trong các hợp đồng khác

2. CROSSFUZZ

Phân tích luồng dữ liệu liên hợp đồng (Inter-contract Data Flow - ICDF)

- Tiếp theo, CrossFuzz sẽ phân tích các mối quan hệ gọi hàm giữa các hợp đồng. Hành động này nhằm thu thập thông tin về cách các hợp đồng tương tác với nhau trong quá trình thực thi.

- 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



2. CROSSFUZZ

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

2. CROSSFUZZ

Cross-contract fuzz testing

- **Bước 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. Sau đó nó triển khai hợp đồng cần kiểm thử với các tham số khởi tạo được tạo ở phần 4.1

- **Bước hai:** dựa trên Application Binary Interface (ABI) của hợp đồng thì ABI cung cấp thông tin về tên hàm, kiểu tham số và kiểu giá trị trả về. Khi đó CrossFuzz sử dụng thông tin này để tạo ra các giao dịch thử nghiệm cơ bản (một lần gọi hàm cho mỗi hàm trong hợp đồng hoặc các hợp đồng phụ thuộc)

2. CROSSFUZZ

Cross-contract fuzz testing

- **Bước thứ ba:** 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

- Nếu có các điều kiện không đạt gây lỗi giao dịch thì lệnh REVERT của EVM sẽ được kích hoạt để hủy thay đổi và kết thúc giao dịch => các giao dịch bị kết thúc bởi lệnh REVERT được coi là các giao dịch ngoại lệ làm giảm khả năng bao phủ mã và cản trở việc thực thi các giao dịch tiếp theo

2. CROSSFUZZ

Cross-contract fuzz testing

- Thông qua define_map và use_map, ta đã có được các hàm có khả năng định nghĩa lại biến trạng thái được sử dụng trong giao dịch bị lỗi

- Sử dụng một công thức để tính điểm ưu tiên mà mỗi hàm trong define_map

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

trong đó: + Sdef là số biến trạng thái được hàm f định nghĩa mà chưa được định nghĩa trước giao dịch lỗi

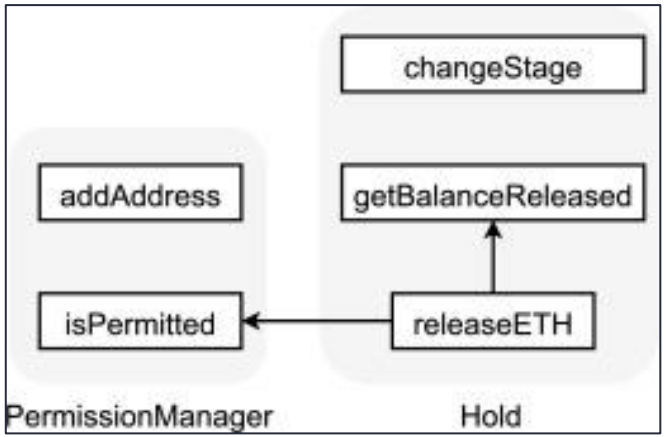
+ Sprovide là số biến trạng thái mà giao dịch lỗi cần sử dụng

+ Suse là số biến trạng thái mà hàm sử dụng nhưng không liên quan đến lỗi

2. CROSSFUZZ

Cross-contract fuzz testing

- **Bước cuối cùng:** Thực hiện biến đổi chuỗi giao dịch, tức là CrossFuzz thay đổi thứ tự giao dịch hoặc thay đổi các giá trị tham số cho đến khi tất cả các biến trạng thái cần thiết được định nghĩa, Chuỗi giao dịch đạt độ dài tối đa và các hàm tiềm năng đã được kiểm tra



03.
Thiết kế và đánh giá thí nghiệm

3. Thiết kế và đánh giá thí nghiệm

3.1. Thiết kế thí nghiệm

Tập dữ liệu:

- Thu thập ngẫu nhiên 500 mã nguồn hợp đồng thông minh từ EtherScan.

- Sử dụng 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: **dưới 100 dòng** (<100), từ **100 đến dưới 500 dòng** (<500) và từ **500 dòng trở lên** (>=500).

3. Thiết kế và đánh giá thí nghiệm

3.1. Thiết kế thí nghiệm

Chỉ số đánh giá:

- **Độ bao phủ mã bytecode** và **số lượng lỗ hổng được phát hiện.**

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.

3. Thiết kế và đánh giá thí nghiệm

3.1. Thiết kế thí nghiệm

Baselines:

CrossFuzz được so sánh với ba fuzzer mã nguồn mở khác: **sFuzz, ConFuzzius,** và **xFuzz.**

- sFuzz được phát triển dựa trên ContractFuzzer.

- ConFuzzius kết hợp kỹ thuật thực thi biểu tượng với fuzz testing.

- xFuzz được tối ưu hóa cho việc phát hiện lỗ hổng đa hợp đồng.

3. Thiết kế và đánh giá thí nghiệm

3.1. Thiết kế thí nghiệm

Thiết lập tham số:

- Tất cả các fuzzer đều sử dụng **tham số mặc định**.

+ Phiên bản EVM: 'byzantium'

+ConFuzzius được kích hoạt mô-đun phân tích phụ thuộc dữ liệu và mô-đun giải ràng buộc.

3. Thiết kế và đánh giá thí nghiệm

3.1. Thiết kế thí nghiệm

Thiết lập tham số:

+ Thời gian kiểm thử tối đa cho mỗi hợp đồng thông minh: 10 phút

+ 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.

3. Thiết kế và đánh giá thí nghiệm

3.2. Đánh giá

RQ1: Chiến lược đột biến trình tự giao dịch có cải thiện khả năng bao phủ mã của CrossFuzz không? Nếu vậy, xác suất tối ưu để thực hiện chiến lược này là bao nhiêu?

RQ2: CrossFuzz có thể đạt được phạm vi bao phủ mã và khả năng phát hiện lỗ hổng bảo mật cao hơn so với các phương pháp thử nghiệm fuzz hiện đại không?

RQ3: Các tham số hàm tạo được tạo có cải thiện phạm vi mã và khả năng phát hiện lỗ hổng không?

3. Thiết kế và đánh giá thí nghiệm

3.2. Đánh giá

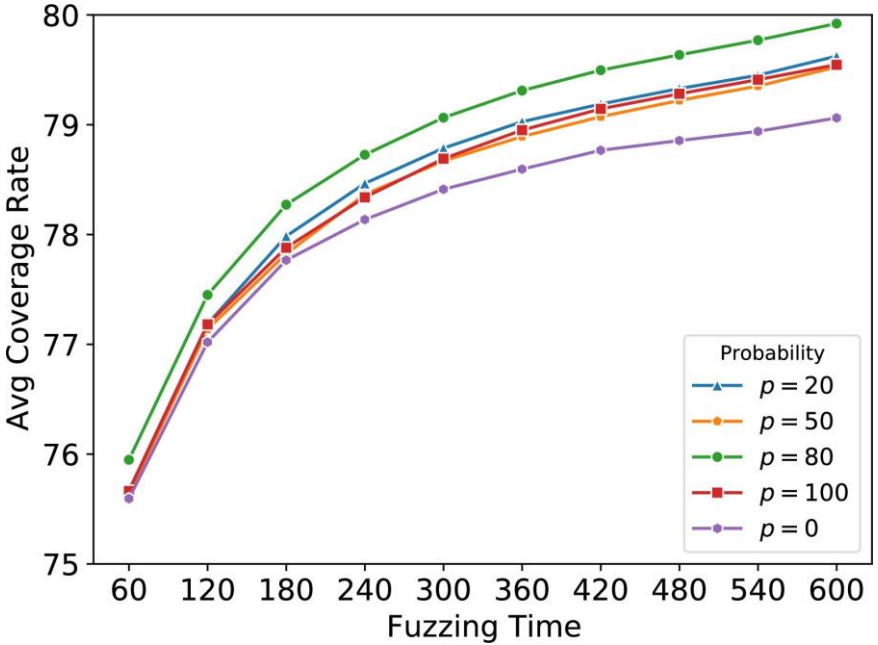
Kết quả cho RQ1:

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%.**

3. Thiết kế và đánh giá thí nghiệm

3.2. Đánh giá

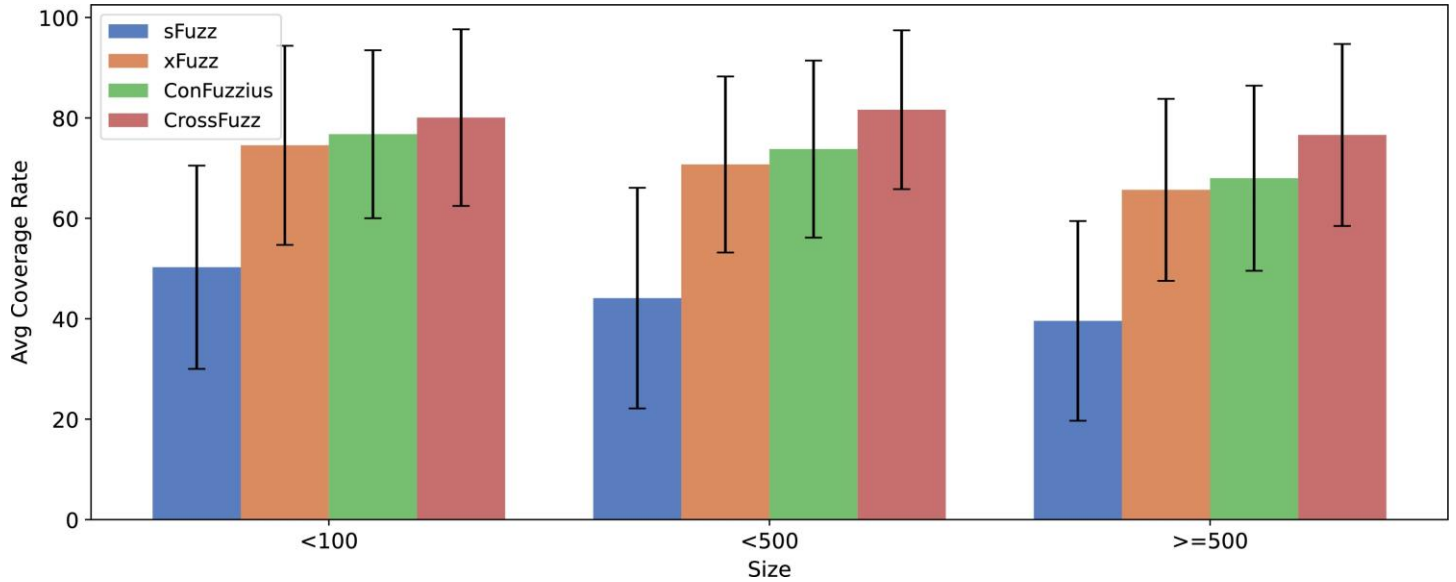
Kết quả cho RQ1:



3. Thiết kế và đánh giá thí nghiệm

3.2. Đánh giá

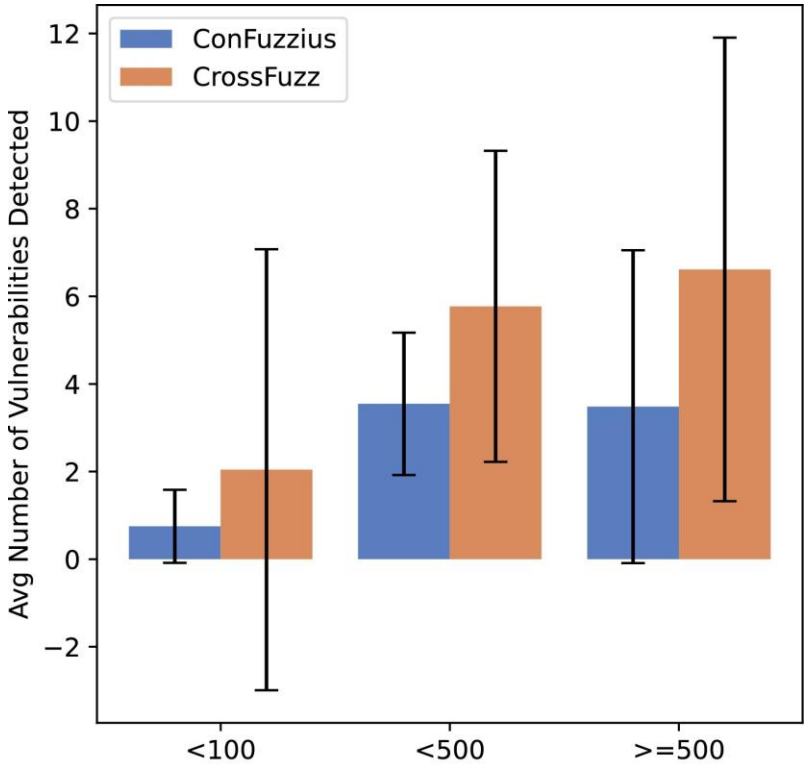
Kết quả cho RQ2:



3. Thiết kế và đánh giá thí nghiệm

3.2. Đánh giá

Kết quả cho RQ2:



3. Thiết kế và đánh giá thí nghiệm

3.2. Đánh giá

Kết quả cho RQ2:

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

3. Thiết kế và đánh giá thí nghiệm

3.2. Đánh giá

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%).

3. Thiết kế và đánh giá thí nghiệm

3.2. Đánh giá

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.

+ 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).

3. Thiết kế và đánh giá thí nghiệm

3.2. Đánh giá

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.

- 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.

3. Thiết kế và đánh giá thí nghiệm

3.2. Đánh giá

Kết quả nhóm triển khai

Số dòng	Độ bao phủ bytecode			Số lỗ hổng		
	CrossFuzz	*ConFuzzius*	*xFuzz*	*CrossFuzz*	*ConFuzzius*	*xFuzz*
Dưới 100 SL: 27	~91%	~86%	~83%	3	3	2
100 đến 500 SL: 10	~84%	~77%	~75%	5	3	2
Trên 500 SL: 3	~64%	~55%	~39%	5	2	X

3. Thiết kế và đánh giá thí nghiệm

3.2. Đánh giá

Kết quả cho RQ3:

CrossFuzz được thử nghiệm với ba chiến lược tạo tham số constructor: sử dụng $CrossFuzzrandom$ và $CrossFuzzzero$ và **theo dõi đường dẫn lan truyền dữ liệu**.

3. Thiết kế và đánh giá thí nghiệm

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)

3. Thiết kế và đánh giá thí nghiệm

3.2. Đánh giá

Kết quả cho RQ3:

- 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. Thiết kế và đánh giá thí nghiệm

5.3. Thảo luận

5.3.1. Các mối đe dọa đến tính hợp lệ

- **Tính hợp lệ nội bộ**: Đảm bảo rằng CrossFuzz được triển khai đúng cách để kết quả thử nghiệm phản ánh chính xác hiệu quả của công cụ.

→ 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 dựa trên **Slither**, mô-đun fuzz testing dựa trên **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, **sFuzz** giúp cải thiện hiệu quả của quá trình đột biến test case và thống nhất việc đánh giá độ bao phủ bytecode.

5. Thiết kế và đánh giá thí nghiệm

5.3. Thảo luận

5.3.1. Các mối đe dọa đến tính hợp lệ

Tính hợp lệ bên ngoài:Liệu kết quả thử nghiệm có thể được khái quát hóa hay không, dựa trên việc tập dữ liệu có đại diện hay không.

→ Các tập dữ liệu hiện có như của SmartDagger và Clairvoyance không được công khai, còn xFuzz thiếu chú thích lỗ hổng. Do đó, tác giả đã xây dựng tập dữ liệu bằng cách chọn ngẫu nhiên các hợp đồng từ EtherScan.

5. Thiết kế và đánh giá thí nghiệm

5.3. Thảo luận

5.3.1. Các mối đe dọa đến tính hợp lệ

Tính hợp lệ bên ngoài:Liệu kết quả thử nghiệm có thể được khái quát hóa hay không, dựa trên việc tập dữ liệu có đại diện hay không.

→ 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.

5. Thiết kế và đánh giá thí nghiệm

5.3. Thảo luận

5.3.1. Các mối đe dọa đến tính hợp lệ

Tính hợp lệ cấu trúc: Liệu các chỉ số đánh giá được sử dụng có thể đánh giá toàn diện hiệu quả của CrossFuzz hay không.

→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. Thiết kế và đánh giá thí nghiệm

5.3. Thảo luận

5.3.2. Các cải tiến trong tương lai

- Vài nhánh mã chỉ có thể được bao phủ bởi chuỗi giao dịch dài:

+ 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 thời gian của fuzz testing.

5. Thiết kế và đánh giá thí nghiệm

5.3. Thảo luận

5.3.2. Các cải tiến trong tương lai

- Một số nhánh mã chỉ được bao phủ nếu các biến trạng thái trong constructor có giá trị mặc định thỏa mãn điều kiện nhánh

+ CrossFuzz hiện không xử lý tình huống này và các giá trị tham số được tạo theo ICDF có thể làm giảm độ bao phủ mã

+ Khả năng cải thiện: Cung cấp giao diện cho phép người dùng nhập thủ công các giá trị tham số cụ thể cho constructor

5. Thiết kế và đánh giá thí nghiệm

5.3. Thảo luận

5.3.2. Các cải tiến trong tương lai

- 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.

+ Cho phép người dùng có thể thay đổi chiến lược này thông qua cài đặt tham số.

3. Thiết kế và đánh giá thí nghiệm

3.3. Thảo luận

3.3.2. Các cải tiến trong tương lai

- CrossFuzz chưa hỗ trợ kiểm thử các tình huống khi hợp đồng gọi các hợp đồng khác thông qua địa chỉ

+Trong tương lai, nhóm phát triển dự kiến sử dụng EtherScan để tìm kiếm mã nguồn của các hợp đồng bên ngoài, mở rộng khả năng áp dụng của CrossFuzz

04.
DEMO

THANKS FOR LISTENING !