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: MuFuzz: Sequence-Aware Mutation and Seed Mask Guidance for Blockchain Smart Contract Fuzzing

Mã nhóm: G08 Mã đề tài: CK23

Lớp: NT521.P11.ANTT

1. THÔNG TIN THÀNH VIÊN NHÓM:

(Sinh viên liệt kê tất cả các thành viên trong nhóm)

STT	Họ và tên	MSSV	Email
1	Hồ Vỉ Khánh	22520633	22520633@gm.uit.edu.vn
2	Lê Quốc Ngô	22520951	22520951@gm.uit.edu.vn
3	Võ Văn Phúc	22521147	22521147@gm.uit.edu.vn
4	Nguyễn Hồ Nhật Khoa	22520677	22520677@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:

[1] Y. Zhang, X. Wang, Z. Li, and J. Liu, "MuFuzz: Sequence-Aware Mutation and Seed Mask Guidance for Blockchain Smart Contract Fuzzing," 2022 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 2022, pp. 1234-1250. DOI: 10.1109/SP46214.2022.00000.

[1] Ghi nội dung tương ứng theo mô tả

C. Dịch tên Tiếng Việt cho bài báo:

MuFuzz: Hướng dẫn Đột biến Nhận thức Chuỗi và Mặt Nạ Dữ liệu Khởi tạo cho Fuzzing Hợp đồng Thông minh trên Blockchain

D. Tóm tắt nội dung chính:

<mô tả nội dung tóm tắt của bài báo/chủ đề trong vòng 350 từ>

MuFuzz là một công cụ fuzzing dành cho smart contract trên blockchain. Sử dụng chiến lược hướng dẫn đột biến trên trình tự (sequence-aware mutation) và mặt nạ seed (seed mask guidance) để khám phá các trạng thái sâu của các hợp đồng thông minh.

Phương pháp tiếp cận của MuFuzz là phân tích luồng dữ liệu để xác định thứ tự giao dịch, kết hợp với chiến lược đột biến trình tự để thâm nhập vào các trạng thái sâu của hợp đồng thông minh. Đưa ra chiến lược đột biến hướng dẫn bởi mặt nạ để tạo các đầu vào giao dịch đạt được các nhánh mục tiêu, đồng thời sử dụng cơ chế cân bằng năng lượng để tối ưu hóa việc phân bổ tài nguyên trong quá trình fuzzing.

MuFuzz được triển khai và đánh giá trên ba bộ dữ liệu: hơn 21.000 hợp đồng Ethereum, 155 hợp đồng chứa các lỗ hổng đã biết, và 500 hợp đồng thực tế quy mô lớn. Kết quả cho thấy MuFuzz vượt trội so với các công cụ hiện có, đạt độ bao phủ nhánh cao hơn đến 25% và phát hiện nhiều lỗi hơn 30%.

MuFuzz cải thiện hiệu quả và độ chính xác trong fuzzing hợp đồng thông minh, phù hợp sử dụng trên các hợp đồng thực tế và có quy mô lớn. MuFuzz có tiềm năng lớn trong việc kiểm thử và bảo mật hợp đồng thông minh, đồng thời có mã nguồn mở giúp ích cho việc nghiên cứu và phát triển sau này.

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

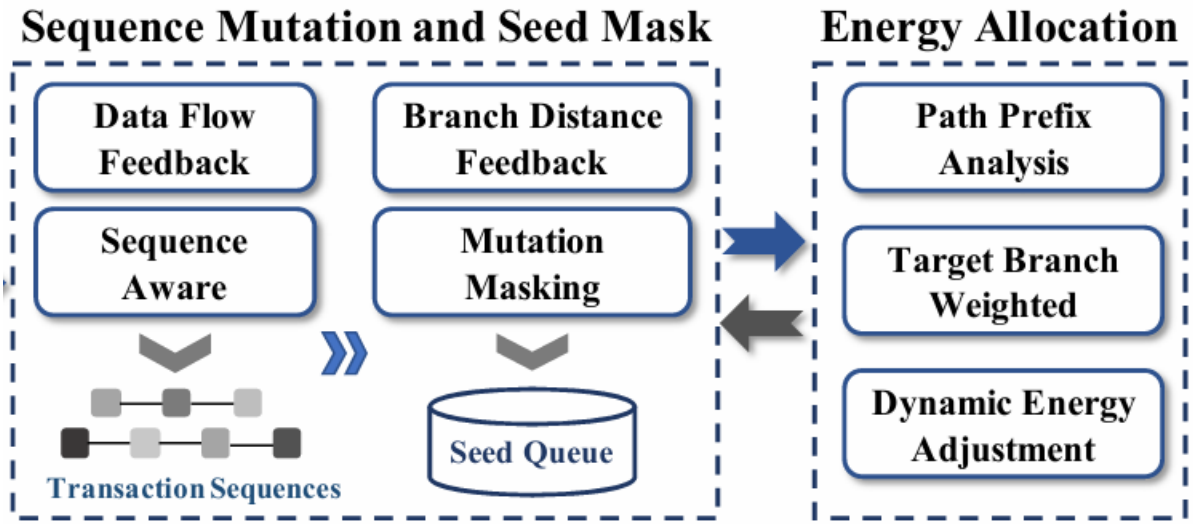
<mô tả các kỹ thuật chính được dùng trong nghiên cứu của bài báo>

(liệt kê tóm tắt vai trò của các kỹ thuật đó – kèm hình ảnh minh họa về hệ thống/kỹ thuật/phương pháp)

Bài báo sử dụng 3 kỹ thuật cho 3 giai đoạn chính trong phương pháp fuzzing hợp đồng thông minh, bao gồm:

- **Kỹ thuật 01**: Đột biến trình tự có ý thức (Sequence-aware Mutation), với vai trò và nhiệm vụ là phân tích luồng dữ liệu trong hợp đồng thông minh để xác định các mối quan hệ phụ thuộc giữa các biến trạng thái và các hàm. Xây dựng các chuỗi giao dịch có ý nghĩa, tối ưu hóa việc khám phá các trạng thái sâu. VD: Hệ thống tạo chuỗi giao dịch như ***[invest → refund → invest → withdraw]*** để kích hoạt điều kiện nhánh sâu, đảm bảo bao phủ toàn bộ logic.

- **Kỹ thuật 02**: Đột biến hạt giống hướng dẫn bằng mặt nạ (Mask-guided Seed Mutation), với vai trò và nhiệm vụ là tập trung vào các phần quan trọng trong đầu vào giao dịch bằng cách sử dụng "mặt nạ" để bảo vệ các giá trị không được đột biến ngẫu nhiên, loại bỏ các đột biến dư thừa, không hiệu quả, tăng khả năng đi được tới các nhánh mục tiêu có thể chứa lỗi. VD: Một mặt nạ được áp dụng để bảo vệ giá trị quan trọng như ***msg.value = 88 finney***, giúp khám phá nhánh có điều kiện chặt chẽ.
- **Kỹ thuật 03**: Điều chỉnh năng lượng động (Dynamic Energy Adjustment), với vai trò và nhiệm vụ là cân bằng tài nguyên đột biến giữa các nhánh thường xuyên, nhánh phức tạp hoặc có nguy cơ chứa lỗ hổng. Bằng cách gán trọng số cao hơn cho các nhánh phức tạp và có thể chứa lỗi, tang khả năng phát hiện lỗ hổng bằng cách tập trung vào các nhánh quan trọng.



F. Môi trường thực nghiệm của bài báo:

<mô tả môi trường thực nghiệm của nhóm tác giả, bao gồm cấu hình máy tính (phần cứng, phần mềm), các chương trình hỗ trợ để hiện thực phương pháp, ngôn ngữ lập trình được sử dụng, các chương trình phần mềm dùng để kiểm tra khả năng/tính năng của phương pháp>

- Cấu hình máy tính: Ubuntu 18.04.6 LTS server, with two Inter(R) Core(TM) E5-2630 v3 CPUs at 2.40GHz (32 cores) and 256GB of memory.
  CMake: version >=3.5.1
  Python: version 3.6
  Go: version 1.15
  solc: version 0.4.26
  leveldb
  Geth & Tools
  numpy

- Các công cụ hỗ trợ sẵn có: solc (dùng biên dịch mã Solidity thành bytecode trong giao đoạn tiền xử lí), sfuzz (cơ chế phản hồi nhánh trong giai đoạn fuzzing chính).
- Ngôn ngữ lập trình để hiện thực phương pháp: Python.
- Đối tượng nghiên cứu (chương trình phần mềm dùng để kiểm tra tính khả thi của phương pháp/tập dữ liệu – nếu có): Tác giả sử dụng 3 tập dữ liệu:
  D1 từ nguồn ConFuzzius sử dụng để đánh giá độ bao phủ nhánh (branch coverage). Số lượng gồm 17,803 hợp đồng nhỏ (≤ 3,632 instructions) và 3,344 hợp đồng lớn (> 3,632 instructions).
  D2 từ nguồn VeriSmart, TMP, SmartBugs, SWC registry sử dụng để đánh giá khả năng phát hiện lỗ hổng. Số lượng 155 hợp đồng chứa 217 lỗ hổng.
  D3 từ nguồn Smartian sử dụng đánh giá trên hợp đồng thực tế lớn gồm 500 hợp đồng thông minh thực tế và phức tạp, mỗi hợp đồng chứa hơn 30.000 giao dịch.
- Tiêu chí đánh giá tính hiệu quả của phương pháp: Số lượng lỗ hổng được phát hiện, thời gian chạy, coverage so với các công cụ khác.

G. Kết quả thực nghiệm của bài báo:

<mô tả ngắn gọn kết quả thực nghiệm của bài báo, tự nhận xét về khả năng, ưu và nhược điểm của phương pháp được đề cập trong bài báo>

Kết quả: MuFuzz đạt độ bao phủ nhánh cao hơn đến 25% so với các công cụ fuzzing hiện đại như IR-Fuzz, sFuzz, ConFuzzius. MuFuzz phát hiện nhiều lỗi hơn 30% so với các công cụ phát hiện lỗi tốt nhất hiện tại. Cụ thể, trên bộ dữ liệu 155 hợp đồng, MuFuzz phát hiện 195 lỗ hổng với tỷ lệ chính xác (true positives) là 94%. MuFuzz được thử nghiệm trên 500 hợp đồng thông minh thực tế lớn, mỗi hợp đồng có hơn 30.000 giao dịch. Nó đạt được độ bao phủ trung bình 80.71%, chứng minh tính hiệu quả trên quy mô lớn.

Nhận xét khả năng: MuFuzz vượt trội trong việc khám phá các nhánh phức tạp và trạng thái sâu nhờ kỹ thuật đột biến trình tự và mặt nạ seed. Công cụ xử lý tốt các hợp đồng quy mô lớn và có khả năng phát hiện các lỗi đa dạng, từ lỗi tràn số nguyên đến lỗi gọi lại (reentrancy).

Ưu điểm: Hiệu quả về độ bao phủ và khả năng phát hiện lỗi cao hơn các công cụ hiện có. Tính mở rộng cao khi hoạt động tốt trên các hợp đồng thông minh lớn và phức tạp. Sáng tạo với phương pháp đột biến trình tự và điều chỉnh năng lượng được cải tiến đáng kể.

Nhược điểm: Phân tích luồng dữ liệu và điều chỉnh năng lượng động có thể gây ra chi phí tính toán lớn, đặc biệt với hợp đồng phức tạp. Một số phụ thuộc dữ liệu

chồng chéo hoặc liên quan đến các biến trạng thái ngoài phạm vi phân tích có thể bị bỏ sót. Với các hợp đồng lớn, thời gian fuzzing có thể kéo dài.

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

<liệt kê các công việc mà nhóm thực hiện cho đề tài dựa trên phân tích phương pháp/hệ thống được sử dụng trong bài báo đã tham khảo>

- Cài đặt môi trường: OS ubuntu 18.04.6, các packet cần thiết: python, go, solc, numpy, ...

- Cài đặt fuzzer: MuFuzz, ConFuzzius.

- Config công cụ để có thể chạy trên môi trường nhóm triển khai.

- Thực hiện fuzzing bằng MuFuzz kiểm thử tập dữ liệu tác giả chuẩn bị.

- Đánh giá MuFuzz và so sánh với ConFuzzius.

<liệt kê các công việc đã thực hiện+ tóm tắt kết quả của công việc này>

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

- Tài nguyên máy có giới hạn nên gặp khó khăn trong việc fuzzing các smart contract lớn.

- Thời gian thực thi các smart contract lớn tốn quá nhiều thời gian.

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

99.99%

4. NHẬT KÝ PHÂN CÔNG NHIỆM VỤ:

STT	Công việc	Phân công nhiệm vụ
1	Nghiên cứu và mục tiêu, nguyên lý hoạt động của MuFuzz từ bài báo và code nguồn trên GitHub.	Lê Quốc Ngô Võ Văn Phúc
2	Tóm tắt chức năng chính, cách cấu hình, và cách sử dụng công cụ, viết tài liệu hướng dẫn cho nhóm.	Lê Quốc Ngô Võ Văn Phúc Nguyễn Hồ Nhật Khoa
3	Phân tích chi tiết từng module trong source code của MuFuzz (sFuzz, …)	Lê Quốc Ngô Hồ Vỉ Khánh
4	Tải và phân tích dataset từ nguồn của bài báo.	Võ Văn Phúc Nguyễn Hồ Nhật Khoa
5	Thực hiện fuzzing trên dataset D1.	Võ Văn Phúc Nguyễn Hồ Nhật Khoa
6	Thực hiện fuzzing trên dataset D2.	Hồ Vỉ Khánh
7	Thực hiện fuzzing trên dataset D3.	Lê Quốc Ngô
8	Tìm và chạy thử công cụ ConFuzzius.	Võ Văn Phúc
9	Thu thập, phân tích, so sánh hiệu suất và kết quả chạy được từ 2 công cụ MuFuzz và ConFuzzius.	Hồ Vỉ Khánh
10	Chuẩn bị slide thuyết trình.	Cả nhóm
11	Thuyết trình báo cáo đồ án với thầy và cả lớp.	Cả nhóm
12	Tổng hợp thông tin để viết báo cáo hoàn chỉnh cho đề tài.	Cả nhóm

BÁO CÁO TỔNG KẾT CHI TIẾT

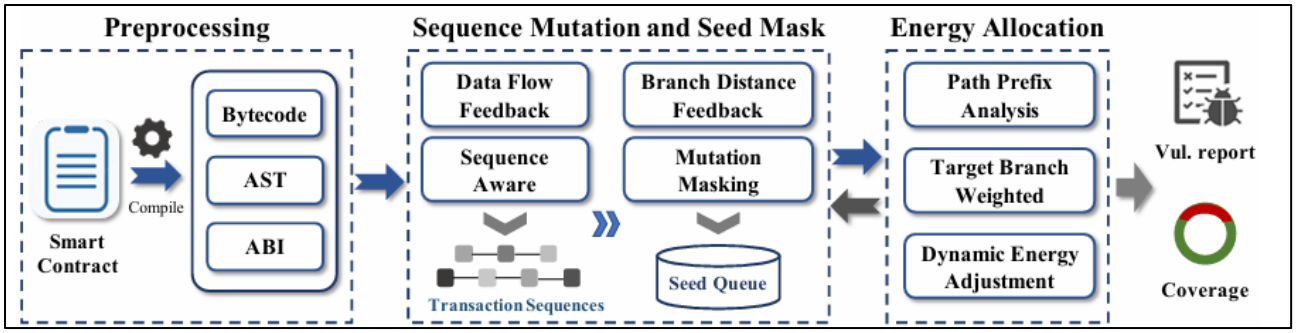
Phần bên dưới của báo cáo này là tài liệu báo cáo tổng kết - chi tiết của nhóm thực hiện cho đề tài này.

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

<Trình bày kiến trúc, thành phần của hệ thống trong bài báo>

Kiến trúc hệ thống



MuFuzz được chia thành ba giai đoạn chính:

(1) Preprocessing (Tiền xử lý):

Giai đoạn này chuẩn bị dữ liệu cho quá trình fuzzing. Nó nhận mã nguồn từ smart contract và thực hiện biên dịch mã nguồn thành ba dạng biểu diễn: bytecode, ABI (Application Binary Interface) và AST (Abstract Syntax Tree).

Mục đích: giúp MuFuzz "hiểu" được cấu trúc và logic của hợp đồn, tạo nền tảng cho việc tạo chuỗi giao dịch và đột biến input hiệu quả.

(2) Sequence Mutation and Seed Mask

Đây là giai đoạn xử lý chính của MuFuzz, được chia thành 2 phần nhỏ:

(2.1) Sequence-Aware Mutation (Đột biến chuỗi)

Module này tập trung vào việc tạo ra chuỗi có nhận thức. Với việc phân tích dữ liệu đã được chuẩn bị ở giai đoạn trước, MuFuzz có thể dễ dàng xác định mỗi quan hệ giữa các hàm, biến trạng thái sẽ ảnh hưởng tới điều kiện của hợp đồng. Từ đó tạo ra các chuỗi giao dịch phản ánh đúng logic và quy trình làm việc của hợp đồng, giúp kiểm thử sâu hơn hợp đồng.

VD: Hệ thống tạo chuỗi giao dịch như [invest → refund → invest → withdraw] để kích hoạt điều kiện nhánh sâu, đảm bảo bao phủ toàn bộ logic.

```
contract Crowdsale {
  // State Variables
  uint256 phase = 0;  // 0: Active, 1: Success
  uint256 goal;
  uint256 invested;
  address owner;
  mapping(address => uint256) invests;

  constructor() public {
    goal = 100 ether;
    invested = 0;
    owner = msg.sender;
  }
  function invest(uint256 donations) public
      payable {
    if (invested < goal) {
      invests[msg.sender] += donations;
      invested += donations;
      phase = 0;
    } else {
      phase = 1;
    }
  }
  function refund() public {
    if(phase == 0) {
      msg.sender.transfer(invests[msg.sender]);
      invests[msg.sender] = 0;
    }
  }
  function withdraw() public {
    if (phase == 1) {
      bug();  // There exists a bug
      owner.transfer(invested);
    }
  }
}
```

(2.2) Mask-Guided Seed Mutation (Đột Biến Seed Được Hướng Dẫn Bởi Mặt Nạ)

Module này có thể gọi là phần lựa chọn seed và đột biến seed dựa trên seedmask. Các bước chính của module này như sau:

- Lựa chọn Seed tiềm năng: dựa trên thuật toán Branch Distance Feedback để chọn ra những seed nào có khả năng dẫn đến việc khám phá các nhánh mã mới hoặc các nhánh sâu hơn. Những seed này được coi là "tiềm năng" vì chúng "gần" với việc thỏa mãn điều kiện của các nhánh chưa được bao phủ.

```
1  SEEDQUEUE ← ∅;
2  Seeds ← INITSEED();
3  while ¬TERMINATED() do
4      foreach seed ∈ Seeds do
5          if seed covers a new branch then
6              SEEDQUEUE.ADD(seed);
       // Branch-Distance-Feedback Seed Selection
7      Let BR_uncover be uncovered branches;
8      foreach b_r ∈ BR_uncover do
9          Let min be +∞;
10         foreach seed ∈ Seeds do
11             if DISTANCE(seed, b_r) < min then
12                 Let min be DISTANCE(seed, b_r);
13         SEEDQUEUE.ADD(seed);
```

Thuật toán lựa chọn seed

- Tạo Seed Mask: đối với mỗi seed được chọn, MuFuzz sẽ tạo một mask (mặt nạ) để bảo vệ những phần quan trọng của seed không cần biến đổi. Mục đích của việc tạo mặt nạ này làm cho quá trình đột biến seed chỉ biến đổi những phần quan trọng, có khả năng gây lỗi hoặc đi đến nhánh mới giúp tiết kiệm được tài nguyên và thời gian.

Algorithm 2: COMPUTEMASK(*seed*, *branch*, $BR_{uncover}$)

```
1  mask ← INITEMPTYMASK(|seed|);
2  n ← RANDOM(1, |seed|);
3  x ∈ {O, I, R, D};
4  foreach 0 ≤ i ≤ |seed| do
5      m ← (x, n);
6      seed_O ← MUTATE(seed, m, i);
7      if branch ∈ NESTEDHIT(seed_O) or DISTANCE(seed_O,
         BR_uncover) decrease then
8          mask[i] ← mask[i] ∪ {O};
9      seed_I ← MUTATE(seed, m, i);
10     if branch ∈ NESTEDHIT(seed_I) or DISTANCE(seed_I,
         BR_uncover) decrease then
11         mask[i] ← mask[i] ∪ {I};
12     seed_R ← MUTATE(seed, m, i);
13     if branch ∈ NESTEDHIT(seed_R) or DISTANCE(seed_R,
         BR_uncover) decrease then
14         mask[i] ← mask[i] ∪ {R};
15     seed_D ← MUTATE(seed, m, i);
16     if branch ∈ NESTEDHIT(seed_D) or DISTANCE(seed_D,
         BR_uncover) decrease then
17         mask[i] ← mask[i] ∪ {D};
18 return mask;
```

Thuật toán tạo seedmask dựa trên seed

- Đột biến Seed: MuFuzz áp dụng các kỹ thuật đột biến lên seed đã chọn, nhưng chỉ tác động lên những phần của seed mà mặt nạ cho phép. Các kỹ thuật đột biến được sử dụng có thể bao gồm bit flipping, byte flipping, arithmetic mutation, interest value mutation, v.v

```
14  energy ← ASSIGNMUTATIONENERGY();
15  while energy > 0 do
16      foreach seed ∈ SEEDQUEUE do
17          if (seed does not cover a nested branch) or
              (DISTANCE(seed, b_r) does not decrease) then
18              continue;
19          nestedBranch ← NESTEDHIT(seed);
20          mask ←
              COMPUTEMASK(seed, nestedBranch, BR_uncover);
21          foreach 0 ≤ i ≤ |seed| do
22              foreach mutation ∈ mutationTypes do
23                  if ¬OKTOMUTATE(mask, mutation, i)
                      then
24                      continue;
25                  newseed ← MUTATE(seed, mutation, i);
26                  result ← FUZZRUN(newseed);
27                  if NEWCOVERAGE(result) then
28                      SEEDQUEUE.ADD(newseed);
29                  energy ← UPDATEENERGY(result,
                      energy);
30  return SEEDQUEUE;
```

Thuật toán đột biến seed

(3) Energy Allocation (Phân bổ năng lượng)

Mục đích của giai đoạn này để tối ưu hóa việc sử dụng tài nguyên fuzzing, tập trung năng lượng vào các nhánh mã "tiềm năng" hơn, tức là những nhánh có khả năng cao chứa lỗi. MuFuzz sẽ chạy 1 bước gọi là pre-fuzzing. Trong bước này, MuFuzz chạy một seed input để ghi lại đường dẫn thực thi (execution path). Sau đó phân tích đường dẫn dựa trên thuật toán Branch Weighted để đánh giá 2 yếu tố: độ lồng nhau và khả năng chứa lỗi của hợp đồng. Thuật toán Branch Weighted sẽ gán trọng số cho từng nhánh, nhánh có độ lồng nhau cao hơn và/hoặc khả năng chứa lỗi cao hơn thì được gán trọng số cao hơn. Dựa trên các trọng số đó, MuFuzz khi fuzzing sẽ phân bổ năng lượng cho các nhánh có trọng số cao hơn.

```
1  π ← PREFUZZRUN(seed); // Pre-Fuzz for Collecting Path
2  π_resource ← INITRESOURCE(π); // Init Fuzzing Resource
3  (B_nested, W_1, N_s) ← (∅, ∅, ∅);
4  (B_vulnerable, W_2) ← (∅, ∅);
5  nested_score ← 0;
6  foreach i < |π| do
7      if ISBRANCHINSTRUCTION(i, π) then
8          nested_score ← nested_score + 1;
9          w_1 ← WEIGHTASSIGN(nested_score);
10         (B_nested, W_1, N_s).ADD(π[i], w_1, nested_score);
11         π_pre ← π[0...i + 1];
12         φ, loc ← PREFIXINFERENCE(π_pre); // Prefix Analysis
13         if ISVULNERABLEINSTRUCTREACHED(φ, loc, instLoc)
             then
14             w_2 ← WEIGHTASSIGN();
15             (B_vulnerable, W_2).ADD(π[i], w_2);
16     i ← i + 1;
17 return (B_nested, W_1, N_s), (B_vulnerable, W_2);
```

Thuật toán Branch Weighted dùng để đánh giá trọng số của nhánh

B. Chi tiết cài đặt, hiện thực

<cách cài đặt, lập trình trên máy tính, cấu hình máy tính sử dụng, chuẩn bị dữ liệu, v.v>

B.1 Chi tiết cài đặt

Cài đặt các packet cần thiết: cmake g++ git python3 python3-pip golang-go libgoogle-perftools-dev libleveldb-dev libssl-dev

```
ubuntu@ubuntu:~/Nhom08/MuFuzz$ sudo apt install cmake g++ git python3 python3-pip golang-go libgoogle-perftools-dev libleveldb-dev libssl-dev
[sudo] password for ubuntu:
Reading package lists... Done
Building dependency tree
Reading state information... Done
golang-go is already the newest version (2:1.10~4ubuntu1).
libgoogle-perftools-dev is already the newest version (2.5-2.2ubuntu3).
libleveldb-dev is already the newest version (1.20-2).
cmake is already the newest version (3.10.2-1ubuntu2.18.04.2).
g++ is already the newest version (4:7.4.0-1ubuntu2.3).
git is already the newest version (1:2.17.1-1ubuntu0.18).
libssl-dev is already the newest version (1.1.1-1ubuntu2.1~18.04.23).
python3 is already the newest version (3.6.7-1~18.04).
python3-pip is already the newest version (9.0.1-2.3~ubuntu1.18.04.8).
The following packages were automatically installed and are no longer required:
  fonts-liberation2 fonts-opensymbol gir1.2-goa-1.0 gir1.2-gst-plugins-base-1.0 gir1.2-gstreamer-1.0 gir1.2-gudev-1.0 gir1.2-snapd-1 gir1.2-ud
  grilo-plugins-0.3-base gstreamer1.0-gtk3 libboost-date-time1.65.1 libboost-filesystem1.65.1 libboost-iostreams1.65.1 libboost-locale1.65.1 l
  libclucene-contribs1v5 libclucene-core1v5 libcmis-0.5-5v5 libcolamd2 libdazzle-1.0-0 libe-book-0.1-1 libedataserverui-1.2-2 libeot0 libepubg
  libetonyek-0.1-1 libevent-2.1-6 libexiv2-14 libfreerdp-client2-2 libfreerdp2-2 libgc1c2 libgee-0.8-2 libgexiv2-2 libgom-1.0-0 libgpgmepp6 li
  libgpod4 liblangtag-common liblangtag1 liblirc-client0 liblua5.3-0 libmediaart-2.0-0 libmspub-0.1-1 libodfgen-0.1-1 libqqwing2v5 libraw16 li
  libsgutils2-2 libssh-4 libsuitesparseconfig5 libvncclient1 libwinpr2-2 libxapian30 libxmlsec1-nss lp-solve media-player-info python3-mako py
  syslinux syslinux-common syslinux-legacy usb-creator-common
```

Cài đặt geth

```
ubuntu@ubuntu:~/Nhom08/MuFuzz$ sudo apt install ethereum
Reading package lists... Done
Building dependency tree
Reading state information... Done
ethereum is already the newest version (1.14.12+build30503+bionic).
The following packages were automatically installed and are no longer required:
  fonts-liberation2 fonts-opensymbol gir1.2-goa-1.0 gir1.2-gst-plugins-base-1.0 gir1.
  grilo-plugins-0.3-base gstreamer1.0-gtk3 libboost-date-time1.65.1 libboost-filesyst
  libclucene-contribs1v5 libclucene-core1v5 libcmis-0.5-5v5 libcolamd2 libdazzle-1.0-
  libetonyek-0.1-1 libevent-2.1-6 libexiv2-14 libfreerdp-client2-2 libfreerdp2-2 libg
  libgpod4 liblangtag-common liblangtag1 liblirc-client0 liblua5.3-0 libmediaart-2.0-
  libsgutils2-2 libssh-4 libsuitesparseconfig5 libvncclient1 libwinpr2-2 libxapian30
  syslinux syslinux-common syslinux-legacy usb-creator-common
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
```

Cài đặt solc (phiên bản 0.4.26)

```
ubuntu@ubuntu:~/Nhom08/MuFuzz$ pip3 install solc-select
Collecting solc-select
  Using cached https://files.pythonhosted.org/packages/27/a6/e2b2529f77431bd610de0a09d633768e9538
Collecting packaging (from solc-select)
  Using cached https://files.pythonhosted.org/packages/05/8e/8de486cbd03baba4deef4142bd643a3e7bbe
Collecting pycryptodome>=3.4.6 (from solc-select)
Collecting pyparsing!=3.0.5,>=2.0.2 (from packaging->solc-select)
  Using cached https://files.pythonhosted.org/packages/e5/0c/0e3c05b1c87bb6a1c76d281b0f35e78d2d80
Installing collected packages: pyparsing, packaging, pycryptodome, solc-select
Successfully installed packaging-21.3 pycryptodome-3.21.0 pyparsing-3.1.4 solc-select-1.0.4
ubuntu@ubuntu:~/Nhom08/MuFuzz$ solc-select install 0.4.26
Installing solc '0.4.26'...
Version '0.4.26' installed.
```

Cài đặt công cụ MuFuzz

```
ubuntu@ubuntu:~/Nhom08$ git clone https://github.com/Messi-Q/MuFuzz.git
Cloning into 'MuFuzz'...
remote: Enumerating objects: 1036, done.
remote: Counting objects: 100% (1036/1036), done.
remote: Compressing objects: 100% (783/783), done.
remote: Total 1036 (delta 226), reused 1013 (delta 213), pack-reused 0 (from 0)
Receiving objects: 100% (1036/1036), 4.36 MiB | 2.27 MiB/s, done.
Resolving deltas: 100% (226/226), done.
```

Sửa file initial.sh

Tạo ra 3 folder: source_code: chứa code của hợp đồng cần fuzzing, clean_source_code: chứa clean_source_code (loại bỏ comment và đổi tên contracts), contracts: chứa các hợp đồng đã được phân tích.

```
# !/bin/bash
mkdir source_code clean_source_code contracts logs branch_msg sFuzz/build

cd sFuzz/build/
cmake ..
cd fuzzer/
make -j 32
cp fuzzer ../../../fuzz

cd ../../../bran/
go build -v -o ../analyse_prefix
```

Chạy file khởi tạo: initial.sh

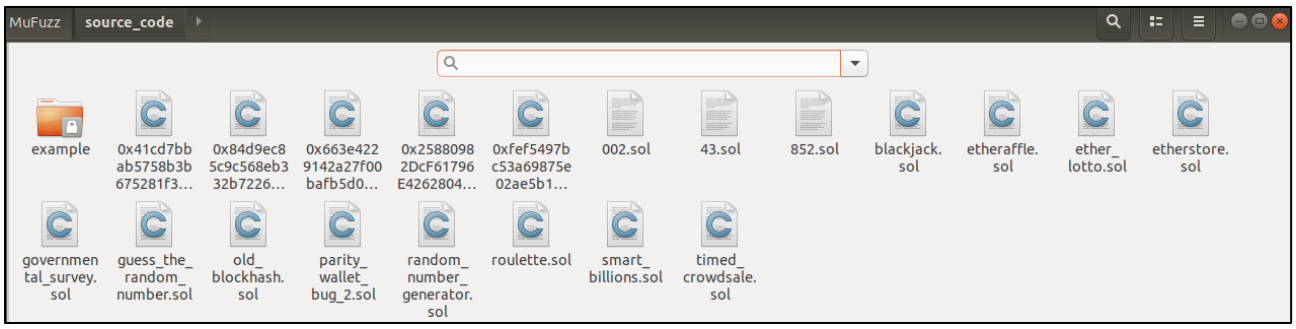
```
ubuntu@ubuntu:~/Nhom08/MuFuzz$ ./initial.sh
mkdir: cannot create directory ‘source_code’: File exists
-- [cable ] Cable 0.2.12 initialized
-- The C compiler identification is GNU 7.5.0
-- The CXX compiler identification is GNU 7.5.0
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- [cable ] Build type: RelWithDebInfo
-- Found Git: /usr/bin/git (found version "2.17.1")
-- Performing Test fstack-protector-strong
-- Performing Test fstack-protector-strong - Success
-- Performing Test Wimplicit-fallthrough
-- Performing Test Wimplicit-fallthrough - Success
-- [hunter] Calculating Toolchain-SHA1
-- [hunter] Calculating Config-SHA1
-- [hunter] HUNTER ROOT: /home/ubuntu/.hunter
```

```
-- Configuring aleth
-----------------------------------------------------------------------
-- CMake 3.10.2 (/usr/bin/cmake)
-- CMAKE_BUILD_TYPE Build type                                  RelWithDebInfo
-- TARGET_PLATFORM  Target platform                             Linux
-- BUILD_SHARED_LIBS                                           OFF
------------------------------------------------------------------- features
-- VMTRACE          VM execution tracing                       OFF
-- EVM_OPTIMIZE     Enable VM optimizations                    ON
-- FATDB            Full database exploring                    ON
-- DB               Database implementation                    LEVELDB
-- PARANOID         -                                          OFF
-- MINIUPNPC        -                                          OFF
------------------------------------------------------------------ components
-- TESTS            Build tests                                ON
-- TOOLS            Build tools                                ON
---------------------------------------------------------------------- tests
-- FASTCTEST        Run only test suites in ctest              OFF
-- TESTETH_ARGS     Testeth arguments in ctest:

-----------------------------------------------------------------------

-- [cable ] Cable 0.2.12 (version newer than 0.2.11) initialized in the `aleth` parent project
-- Found leveldb: /usr/lib/x86_64-linux-gnu/libleveldb.so
-- Configuring done
-- Generating done
-- Build files have been written to: /home/ubuntu/Nhom08/MuFuzz/sFuzz/build
```

```
Scanning dependencies of target secp256k1
Scanning dependencies of target aleth-buildinfo-git
Scanning dependencies of target instructions
Scanning dependencies of target loader
[  1%] Creating directories for 'mpir'
[  1%] Creating directories for 'secp256k1'
[  2%] Building C object evmc/lib/instructions/CMakeFiles/instructions.dir/instruction_metrics.c.o
[  3%] Building C object evmc/lib/loader/CMakeFiles/loader.dir/loader.c.o
[  3%] Building C object evmc/lib/instructions/CMakeFiles/instructions.dir/instruction_names.c.o
[  3%] Built target aleth-buildinfo-git
[  3%] Updating aleth-buildinfo:
[  3%] Performing download step (download, verify and extract) for 'mpir'
       Project Version:  1.5.0-alpha.3+commit.4451e539.dirty (prerelease)
       System Name:      linux
       System Processor: x86_64
       Compiler ID:      gnu
       Compiler Version: 7.5.0
       Build Type:       relwithdebinfo
       Git Info:           4451e539318e0dfc71a5d298793a275517f84b73 (dirty)
       Timestamp:        2025-01-03T01:23:49
[  4%] Performing download step (download, verify and extract) for 'secp256k1'
-- Downloading...
   dst='/home/ubuntu/Nhom08/MuFuzz/sFuzz/build/deps/src/mpir-cmake.tar.gz'
   timeout='none'
-- Using src='https://github.com/chfast/mpir/archive/cmake.tar.gz'
-- Downloading...
   dst='/home/ubuntu/Nhom08/MuFuzz/sFuzz/build/deps/src/secp256k1-ac8ccf29.tar.gz'
   timeout='none'
-- Using src='https://github.com/chfast/secp256k1/archive/ac8ccf29b8c6b2b793bc734661ce43d1f952977a.tar.gz'
Scanning dependencies of target aleth-buildinfo
[  6%] Building C object CMakeFiles/aleth-buildinfo.dir/aleth/buildinfo.c.o
[  6%] Linking C static library aleth/libaleth-buildinfo.a
[  7%] Linking C static library libevmc-instructions.a
[  7%] Linking C static library libevmc-loader.a
[  7%] Built target loader
[  7%] Built target aleth-buildinfo
[  7%] Built target instructions
Scanning dependencies of target devcore
```

```
[ 98%] Building C object CMakeFiles/mpir.dir/scanf/scanf.c.o
[ 99%] Building C object CMakeFiles/mpir.dir/scanf/sscanf.c.o
[ 99%] Building C object CMakeFiles/mpir.dir/scanf/fscanffuns.c.o
[ 99%] Building C object CMakeFiles/mpir.dir/scanf/sscanffuns.c.o
[ 99%] Building C object CMakeFiles/mpir.dir/scanf/vfscanf.c.o
[ 99%] Building C object CMakeFiles/mpir.dir/scanf/vscanf.c.o
[ 99%] Building C object CMakeFiles/mpir.dir/scanf/vsscanf.c.o
[100%] Linking C static library libmpir.a
[100%] Built target mpir
[ 29%] Performing install step for 'mpir'
[100%] Built target mpir
Install the project...
-- Install configuration: "Release"
-- Installing: /home/ubuntu/Nhom08/MuFuzz/sFuzz/build/deps/include/mpir.h
-- Installing: /home/ubuntu/Nhom08/MuFuzz/sFuzz/build/deps/include/gmp.h
-- Installing: /home/ubuntu/Nhom08/MuFuzz/sFuzz/build/deps/lib/libmpir.a
[ 30%] Completed 'mpir'
```

Đưa mã nguồn vào folder source_code



Chạy file rename_src.sh để chuẩn bị dữ liệu cho quá trình fuzzing.

```
ubuntu@ubuntu:~/MuFuzz$ ./rename_src.sh
guess_the_random_number.sol
etheraffle.sol
852.sol
example
Traceback (most recent call last):
  File "remove_comment.py", line 48, in <module>
    remove_comment(original_dir + i, output_dir + i)
  File "remove_comment.py", line 7, in remove_comment
    fdr = open(inputFile, 'r', encoding="utf-8")
IsADirectoryError: [Errno 21] Is a directory: '../source_code/example'
```

Chạy file run.sh để bắt đầu fuzzing

```
ubuntu@ubuntu:~/MuFuzz$ ./run.sh
ERC20.sol may be abstract, not implementing an abstract parent's methods completely
or not invoking an inherited contract's constructor correctly. ERROR: at line 5,6,11,12,13
ERC20Basic.sol may be abstract, not implementing an abstract parent's methods completely
or not invoking an inherited contract's constructor correctly. ERROR: at line 5,6,11,12,13
> Created "fuzzMe"
> To fuzz contracts:
  chmod +x fuzzMe
  ./fuzzMe
contracts/guess_the_random_number/GuessTheRandomNumberChallenge.sol:8:5: Warning: Defining constructors
ated. Use "constructor(...) { ... }" instead.
```

B.2 Hiện thực

Tiến hành chạy thực nghiệm công cụ MuFuzz với một tệp mã nguồn Solidity nằm trong Dataset tác giả dùng để thực nghiệm.

Cấu trúc source code của công cụ MuFuzz.

Tệp mã nguồn Solidity dùng để thực hiện fuzzing. Tệp này sẽ chứa nhiều smart contract, hợp đồng chính (MiniMeToken) sẽ được sử dụng độc lập trong khi các hợp đồng khác sẽ cung cấp logic và hỗ trợ cho hợp đồng chính.

contract@0x310e33962f5f64db7884c6b33103f442698d4d6a.sol U

source_code > contract@0x310e33962f5f64db7884c6b33103f442698d4d6a.sol

```
pragma solidity ^0.4.15;

/*
    Copyright 2016, Jordi Baylina

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/

/// @title MiniMeToken Contract
/// @author Jordi Baylina
/// @dev This token contract's goal is to make it easy for anyone to clone this
///  token using the token distribution at a given block, this will allow DAO's
///  and DApps to upgrade their features in a decentralized manner without
///  affecting the original token
/// @dev It is ERC20 compliant, but still needs to under go further testing.


/// @dev The token controller contract must implement these functions
contract TokenController {
    /// @notice Called when `_owner` sends ether to the MiniMe Token contract
    /// @param _owner The address that sent the ether to create tokens
    /// @return True if the ether is accepted, false if it throws
    function proxyPayment(address _owner) payable returns(bool);

    /// @notice Notifies the controller about a token transfer allowing the
    ///  controller to react if desired
```

Contract TokenController.

```
/// @dev The token controller contract must implement these functions
contract TokenController {
    /// @notice Called when `_owner` sends ether to the MiniMe Token contract
    /// @param _owner The address that sent the ether to create tokens
    /// @return True if the ether is accepted, false if it throws
    function proxyPayment(address _owner) payable returns(bool);

    /// @notice Notifies the controller about a token transfer allowing the
    ///  controller to react if desired
    /// @param _from The origin of the transfer
    /// @param _to The destination of the transfer
    /// @param _amount The amount of the transfer
    /// @return False if the controller does not authorize the transfer
    function onTransfer(address _from, address _to, uint _amount) returns(bool);

    /// @notice Notifies the controller about an approval allowing the
    ///  controller to react if desired
    /// @param _owner The address that calls `approve()`
    /// @param _spender The spender in the `approve()` call
    /// @param _amount The amount in the `approve()` call
    /// @return False if the controller does not authorize the approval
    function onApprove(address _owner, address _spender, uint _amount)
        returns(bool);
}
```

Contract Controlled.

```
54  contract Controlled {
55      /// @notice The address of the controller is the only address that can call
56      ///  a function with this modifier
57      modifier onlyController { require(msg.sender == controller); _; }
58
59      address public controller;
60
61      function Controlled() { controller = msg.sender;}
62
63      /// @notice Changes the controller of the contract
64      /// @param _newController The new controller of the contract
65      function changeController(address _newController) onlyController {
66          controller = _newController;
67      }
68  }
```

Contract ApproveAndCallFallBack.

```
70  contract ApproveAndCallFallBack {
71      function receiveApproval(address from, uint256 _amount, address _token, bytes _data);
72  }
73
```

Contract MiniMeToken.

```
74  /// @dev The actual token contract, the default controller is the msg.sender
75  ///  that deploys the contract, so usually this token will be deployed by a
76  ///  token controller contract, which Giveth will call a "Campaign"
77  contract MiniMeToken is Controlled {
78
79      string public name;                //The Token's name: e.g. DigixDAO Tokens
80      uint8 public decimals;             //Number of decimals of the smallest unit
81      string public symbol;              //An identifier: e.g. REP
82      string public version = 'MMT_0.1'; //An arbitrary versioning scheme
83
84
85      /// @dev `Checkpoint` is the structure that attaches a block number to a
86      ///  given value, the block number attached is the one that last changed the
87      ///  value
88      struct  Checkpoint {
89
90          // `fromBlock` is the block number that the value was generated from
91          uint128 fromBlock;
92
93          // `value` is the amount of tokens at a specific block number
94          uint128 value;
95      }
96
97      // `parentToken` is the Token address that was cloned to produce this token;
98      //  it will be 0x0 for a token that was not cloned
99      MiniMeToken public parentToken;
100
101     // `parentSnapShotBlock` is the block number from the Parent Token that was
102     //  used to determine the initial distribution of the Clone Token
103     uint public parentSnapShotBlock;
104
```

Contract MiniMeTokenFactory.

```
566  /// @dev This contract is used to generate clone contracts from a contract.
567  ///  In solidity this is the way to create a contract from a contract of the
568  ///  same class
569  contract MiniMeTokenFactory {
570
571      /// @notice Update the DApp by creating a new token with new functionalities
572      ///  the msg.sender becomes the controller of this clone token
573      /// @param _parentToken Address of the token being cloned
574      /// @param _snapshotBlock Block of the parent token that will
575      ///  determine the initial distribution of the clone token
576      /// @param _tokenName Name of the new token
577      /// @param _decimalUnits Number of decimals of the new token
578      /// @param _tokenSymbol Token Symbol for the new token
579      /// @param _transfersEnabled If true, tokens will be able to be transferred
580      /// @return The address of the new token contract
581      function createCloneToken(
582          address _parentToken,
583          uint _snapshotBlock,
584          string _tokenName,
585          uint8 _decimalUnits,
586          string _tokenSymbol,
587          bool _transfersEnabled
588      ) returns (MiniMeToken) {
589          MiniMeToken newToken = new MiniMeToken(
590              this,
591              _parentToken,
592              _snapshotBlock,
593              _tokenName,
594              _decimalUnits,
595              _tokenSymbol,
596              _transfersEnabled
597              );
598
```

Contract WCT2.

```
606  /**
607   * @title WePower Contribution Token
608   *
609   * @dev Simple ERC20 Token, with pre-sale logic
610   * @dev IMPORTANT NOTE: do not use or deploy this contract as-is. It needs some changes to be
611   * production ready.
612   */
613  contract WCT2 is MiniMeToken {
614    /**
615      * @dev Constructor
616     */
617    function WCT2(address _tokenFactory)
618      MiniMeToken(
619        _tokenFactory,
620        0x0,                    // no parent token
621        0,                      // no snapshot block number from parent
622        "WePower Contribution Token 2", // Token name
623        18,                     // Decimals
624        "WCT2",                  // Symbol
625        true                    // Enable transfers
626      ) {}
627  }
628
```

Tiến hành chạy công cụ MuFuzz để thực hiện fuzzing tệp Solidity ở trên.

- Đầu tiên sẽ chạy file rename_src.sh để làm sạch code và thay đổi tên contract.
- Tiếp theo chạy file run.sh để thực hiện fuzzing.

```
p2v@LAPTOP-7UPGKHFU:/mnt/d/MuFuzz$ ./rename_src.sh
contract@0x310e33962f5f64db7884c6b33103f442698d4d6a.sol
p2v@LAPTOP-7UPGKHFU:/mnt/d/MuFuzz$ ./run.sh
ApproveAndCallFallBack.sol may be absract, not implement an abstract parent's methods completely
or not invoke an inherited contract's constructor correctly. ERROR: at line 19,27,35,56
TokenController.sol may be absract, not implement an abstract parent's methods completely
or not invoke an inherited contract's constructor correctly. ERROR: at line 19,27,35,56
============= Pre Analysis Success! =============
=============== Prefuzzing Start! ===============
> Created "fuzzMe"
> To fuzz contracts:
  chmod +x fuzzMe
  ./fuzzMe
```

Đây sẽ là các report về từng contract trong giai đoạn Pre Fuzzing.

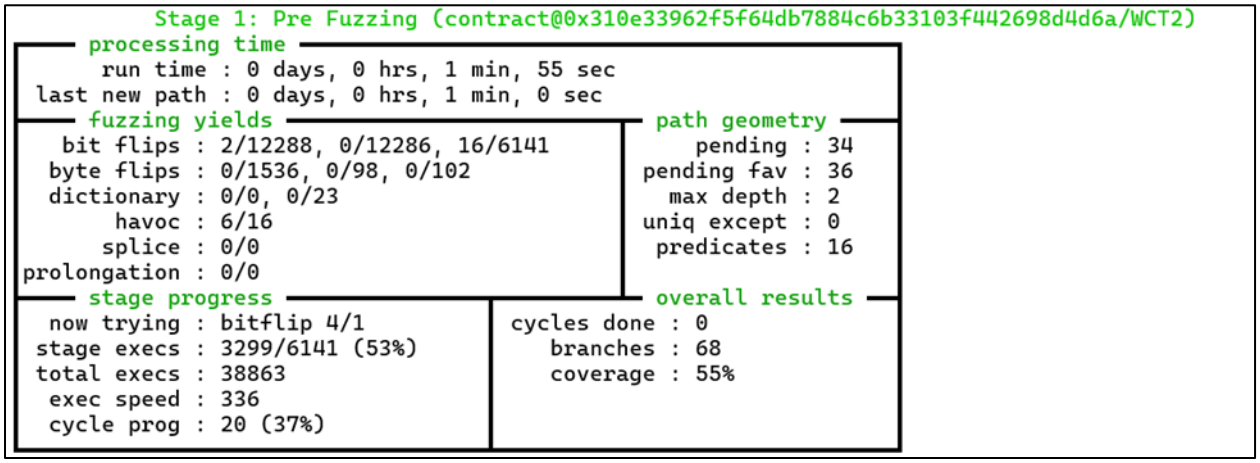
Kết quả giai đoạn 1 với contract Controlled.

```
        Stage 1: Pre Fuzzing (contract@0x310e33962f5f64db7884c6b33103f442698d4d6a/Controlled)
┌─── processing time ──────────────────────────────────────────────────┐
│      run time : 0 days, 0 hrs, 0 min, 0 sec                          │
│ last new path : 0 days, 0 hrs, 0 min, 0 sec                          │
├─── fuzzing yields ──────────────────┬─── path geometry ──────────────┤
│   bit flips : 0/0, 0/0, 0/0        │    pending : 1                 │
│  byte flips : 0/0, 0/0, 0/0        │ pending fav : 2                │
│  dictionary : 0/0, 0/0             │   max depth : 1                │
│       havoc : 0/0                  │ uniq except : 0                │
│      splice : 0/0                  │  predicates : 0                │
│prolongation : 0/0                  │                                │
├─── stage progress ──────────────────┼─── overall results ────────────┤
│  now trying : init                 │cycles done : 0                 │
│ stage execs : 0/0 (100%)           │   branches : 2                 │
│ total execs : 1                    │   coverage : 100%              │
│  exec speed : 29                   │                                │
│  cycle prog : 1 (50%)              │                                │
└────────────────────────────────────┴────────────────────────────────┘
```

Kết quả giai đoạn 1 với contract MiniMeToken.

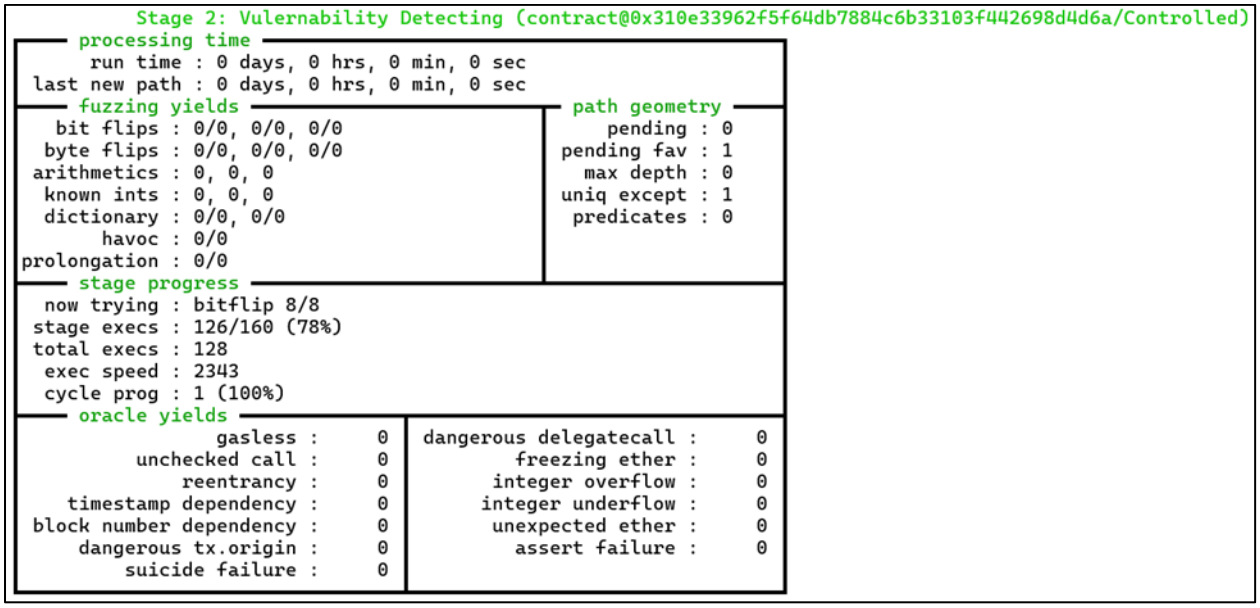
```
        Stage 1: Pre Fuzzing (contract@0x310e33962f5f64db7884c6b33103f442698d4d6a/MiniMeToken)
┌─── processing time ──────────────────────────────────────────────────┐
│      run time : 0 days, 0 hrs, 1 min, 8 sec                          │
│ last new path : 0 days, 0 hrs, 1 min, 0 sec                          │
├─── fuzzing yields ──────────────────┬─── path geometry ──────────────┤
│   bit flips : 8/7936, 0/0, 0/0     │    pending : 23                │
│  byte flips : 0/992, 0/0, 0/0      │ pending fav : 24               │
│  dictionary : 0/0, 0/0             │   max depth : 2                │
│       havoc : 0/0                  │ uniq except : 0                │
│      splice : 0/0                  │  predicates : 16               │
│prolongation : 0/0                  │                                │
├─── stage progress ──────────────────┼─── overall results ────────────┤
│  now trying : bitflip 2/1          │cycles done : 0                 │
│ stage execs : 5225/7935 (65%)      │   branches : 68                │
│ total execs : 16139                │   coverage : 41%               │
│  exec speed : 234                  │                                │
│  cycle prog : 21 (47%)             │                                │
└────────────────────────────────────┴────────────────────────────────┘
```

Kết quả giai đoạn 1 với contract WCT2.

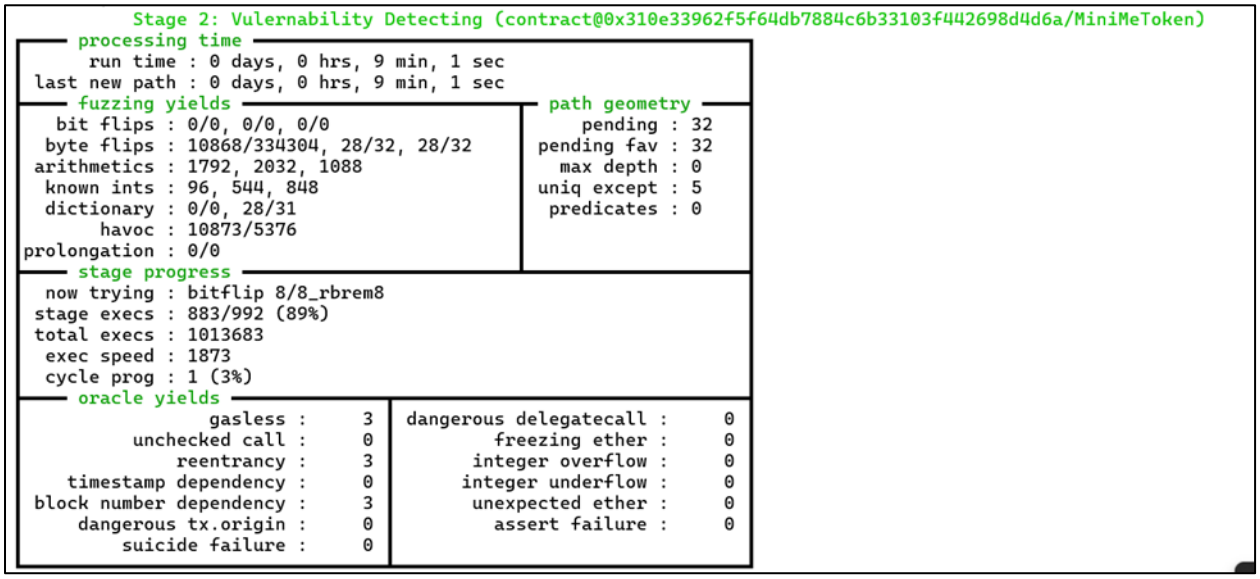


Kết quả report của từng contract trong giai đoạn Main Fuzzing. Đây là giai đoạn cho biết kết quả khi fuzzing các contract, cho biết lỗi và số liệu thông kê khi fuzzing.

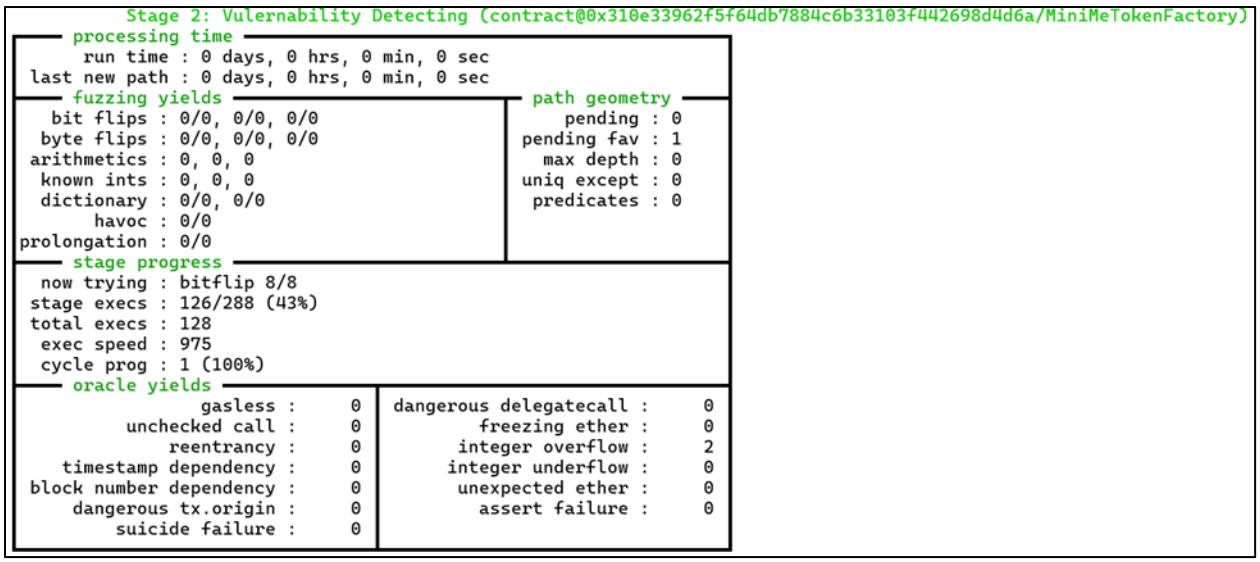
Kết quả giai đoạn 2 với contract Controlled.



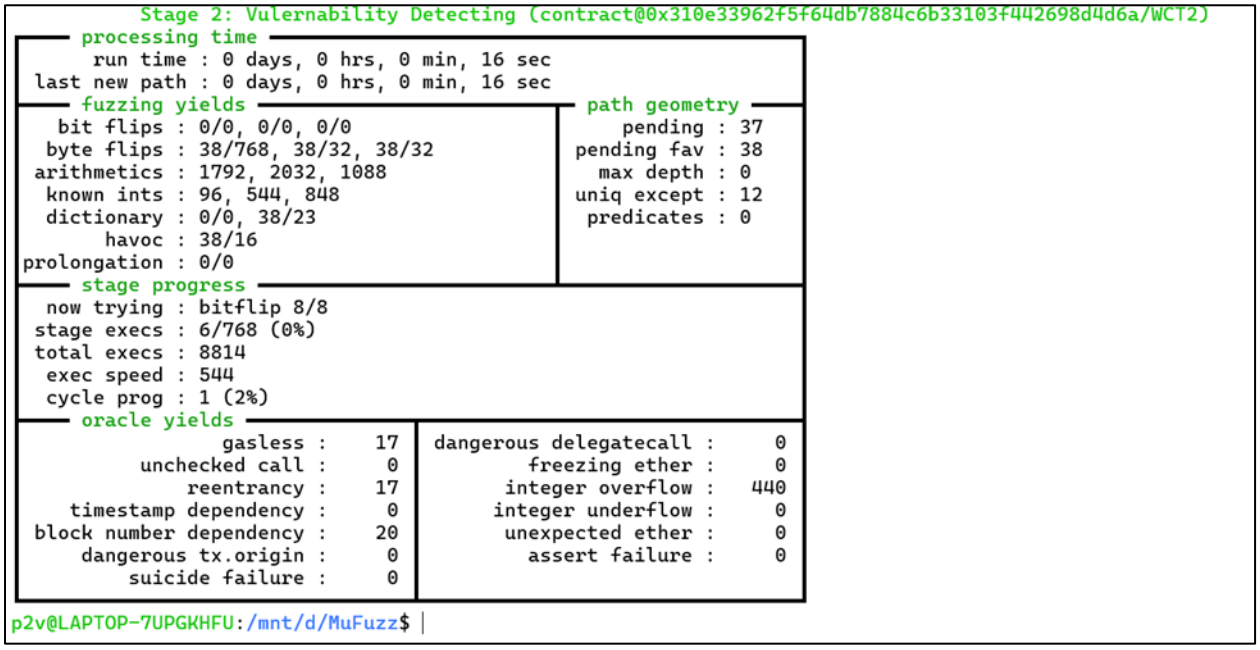
Kết quả giai đoạn 2 với contract MiniMeToken.



Kết quả giai đoạn 2 với contract MiniMeTokenFactory.



Kết quả giai đoạn 2 với contract WCT2.



Giải thích các số liệu trong kết quả report Stage 1 của hợp đồng WCT2:

1. Processing Time:

- Runtime (1m28s):
  - Đây là thời gian tổng thể mà công cụ fuzzing đã hoạt động trong giai đoạn "Pre-Fuzzing". Giai đoạn này chủ yếu kiểm tra các kỹ thuật biến đổi dữ liệu (mutation) để tìm ra các nhánh chương trình mới.
- Last new path (31s):
  - Công cụ đã tìm thấy một nhánh (path) chương trình mới gần nhất vào thời điểm 31 giây trước. Chỉ số này phản ánh tốc độ phát hiện các phần mã chưa được kiểm tra (unexplored code paths), giúp đánh giá khả năng bao phủ mã (coverage) của công cụ.

2. Fuzzing Yields:

- Bit flips (2/12288, 0/12286, 16/6141):
  - Đây là phương pháp thay đổi từng bit trong dữ liệu đầu vào. Trong tổng số 12,288 input (test cases) được thử nghiệm, 2 biến thể input đã tìm được những nhánh mới. Tuy nhiên, trong các nhóm input thử nghiệm khác (0/12286 và 16/6141) không tìm thấy các nhánh mới.
  - Điều này cho thấy lật bit có thể không phải là kỹ thuật tối ưu trong giai đoạn này.
- Byte flips (0/1536, 0/98, 0/102):

- Thay đổi giá trị byte trong dữ liệu đầu vào. Có 1,536 biến thể input được thử nghiệm, nhưng không phát hiện bất kỳ trường hợp nào Tương tự, các nhóm khác (0/98, 0/102) cũng không mang lại kết quả.

- Dictionary (0/0, 0/23):
  - Kỹ thuật này sử dụng các "key word" để chèn vào input đầu vào. Không có trường hợp nào phát hiện được các nhánh mới.
- Havoc (6/16):
  - Đây là một chiến lược biến đổi ngẫu nhiên trên dữ liệu đầu vào. Trong tổng số 16 biến thể, có 6 biến thể tạo ra hiệu quả. Điều này cho thấy Havoc là một kỹ thuật hiệu quả hơn trong giai đoạn này.
- Splice (0/0):
  - Phương pháp kết hợp các phần của hai input khác nhau để tạo ra input mới. Chưa được áp dụng trong giai đoạn này.
- Prolongation (0/0):
  - Kỹ thuật kéo dài hoặc thay đổi độ dài của input. Chưa được áp dụng trong giai đoạn này.

3. Stage Progress:

- Now trying (bitflip 4/1):
  - Công cụ hiện tại đang thử nghiệm kỹ thuật lật bit, với trạng thái cụ thể là bitflip 4/1.
- Stage execs (1866/6141 – 30%):
  - Tổng số 1,866 thử nghiệm đã hoàn thành trong giai đoạn này, chiếm 30% tổng số dự kiến (6,141). Chỉ số này thể hiện tiến độ của kỹ thuật đang áp dụng.
- Total execs (37430):
  - Đây là tổng số thử nghiệm đầu vào (test cases) đã được thực hiện trong toàn bộ giai đoạn Pre-Fuzzing. Chỉ số này càng cao, độ tin cậy của công cụ càng lớn vì nó đã kiểm tra nhiều khả năng.
- Exec speed (316 execs/sec):
  - Tốc độ thực thi trung bình của công cụ là 316 trường hợp/giây. Chỉ số này phụ thuộc vào hiệu suất của phần cứng và độ phức tạp của chương trình được fuzzing.
- Cycles prog (20 – 37%):

- Chu kỳ (cycle) là một vòng lặp hoàn chỉnh của tất cả các test cases đang chờ xử lý. Công cụ đã thực hiện 20 chu kỳ, chiếm 37% tiến độ của chương trình.

4. Path Geometry:

- Pending (34):
  - Có 34 nhánh chương trình chưa được kiểm tra. Đây là các phần mã mà công cụ dự kiến sẽ xử lý trong các giai đoạn tiếp theo.
- Pending fav (36):
  - Trong số các nhánh chưa được kiểm tra, có 36 nhánh được đánh dấu là ưu tiên xử lý. Những nhánh này có khả năng mang lại thông tin mới hoặc quan trọng hơn.
- Max depth (2):
  - Công cụ đã đi được sâu nhất là 2 mức độ trong cây nhánh chương trình. Chỉ số này phản ánh khả năng khám phá các đường dẫn sâu hơn trong mã nguồn.
- Uniq expect (0):
  - Không có trường hợp đầu vào duy nhất nào được dự đoán trong giai đoạn này.
- Predicates (16):
  - Có 16 ràng buộc logic được phát hiện. Các ràng buộc này thường là các điều kiện "if-else" trong mã nguồn.

5. Overall Result:

- Cycles done (0):
  - Chưa hoàn thành hoàn thành chu kỳ nào trong giai đoạn Pre-Fuzzing.
- Branches discovered (68):
  - Đã phát hiện được 68 nhánh mới trong mã nguồn. Điều này chứng tỏ công cụ có hiệu quả trong việc khám phá mã nguồn.
- Coverage (55%):
  - Công cụ đã bao phủ được 55% mã nguồn của chương trình.

Giải thích các số liệu trong kết quả report Stage 2 của hợp đồng WCT2:

1. Processing Time

- Run time (16s):

- Tổng thời gian chạy của giai đoạn Fuzzing chính 16 giây. Mặc dù thời gian ngắn, nhưng đây là giai đoạn tập trung kiểm tra sâu các lỗ hổng cụ thể trong mã nguồn, sử dụng các chiến lược đã tối ưu từ giai đoạn trước.

- Last new path (16s):
  - Công cụ tìm thấy nhánh chương trình mới vào cuối cùng của thời gian chạy. Điều này phản ánh khả năng tiếp tục khám phá các vùng mã chưa được kiểm tra.

2. Fuzzing Yields

- Bit flips (0/0, 0/0, 0/0):
  - Không có kết quả từ kỹ thuật lật bit trong giai đoạn này, điều này là do các biến thể đã được kiểm tra đầy đủ trong giai đoạn trước hoặc không còn mang lại nhánh mới.
- Byte flips (38/768, 38/32, 38/32):
  - Kỹ thuật thay đổi giá trị byte trong input đầu vào đã cho thấy hiệu quả. Cụ thể, trong 768 thử nghiệm, có 38 kết quả thành công. Kỹ thuật này có khả năng tạo ra các trạng thái chương trình mới hữu ích.
- Arithmetics (1792, 2032, 1088):
  - Đây là cách thực hiện các phép toán số học (cộng/trừ) để tạo ra các biến thể input. Phương pháp này đã thử nghiệm tổng cộng 4,912 biến thể input và giúp kiểm tra các lỗ hổng như tràn số (integer overflow).
- Known inits (96, 544, 848):
  - Sử dụng các giá trị khởi tạo đã biết (known initial values) từ các thử nghiệm trước, với tổng cộng 1,488 thử nghiệm. Kỹ thuật này có thể giúp tái sử dụng thông tin từ giai đoạn trước để tìm lỗ hổng nhanh hơn.
- Dictionary (0/0, 38/23):
  - Kỹ thuật này sử dụng các "key word" để chèn vào input đầu vào. Công cụ fuzzing đã tạo ra 38 biến thể, trong đó 23 biến thể mang lại kết quả đáng chú ý.
- Havoc (38/16):
  - Kỹ thuật biến đổi ngẫu nhiên (havoc) tiếp tục là một chiến lược hiệu quả, với 38 kết quả thành công từ 16 thử nghiệm.
- Prolongation (0/0):
  - Kỹ thuật kéo dài input đầu vào chưa được áp dụng trong giai đoạn này.

3. Stage Progress

- Now trying (bitflip 8/8):
  - Công cụ đang thử nghiệm bước thứ 8 của kỹ thuật lật bit, mặc dù kỹ thuật này chưa mang lại kết quả trong giai đoạn này.
- Stage execs (6/768 – 0%):
  - Công cụ fuzzing hoàn thành 6/768 thử nghiệm trong giai đoạn hiện tại, tương đương 0% tiến độ.
- Total execs (8,814):
  - Tổng số lần thử nghiệm (test cases) từ đầu giai đoạn là 8,814.
- Exec speed (549 execs/sec):
  - Tốc độ thực thi trung bình đã tăng đáng kể lên 549 trường hợp/giây.
- Cycles prog (1 – 2%):
  - Công cụ đã hoàn thành 1 chu kỳ, tương đương 2% tiến độ.

4. Path Geometry

- Pending (37):
  - Có 37 nhánh chương trình chưa được kiểm tra. Đây là các phần mã mà công cụ cần xử lý tiếp trong các giai đoạn sau.
- Pending fav (38):
  - Trong số 37 nhánh đang chờ, có 38 nhánh được đánh dấu là ưu tiên. Nhánh ưu tiên thường có khả năng mang lại thông tin quan trọng hơn.
- Max depth (0):
  - Độ sâu tối đa của nhánh chương trình trong giai đoạn này là 0.
- Uniq expect (12):
  - Công cụ Fuzzing dự đoán có 12 nhánh có giá trị đặc biệt. Đây là những trường hợp có thể giúp khám phá thêm nhánh mới hoặc lỗ hổng.'

5. Oracle Yields

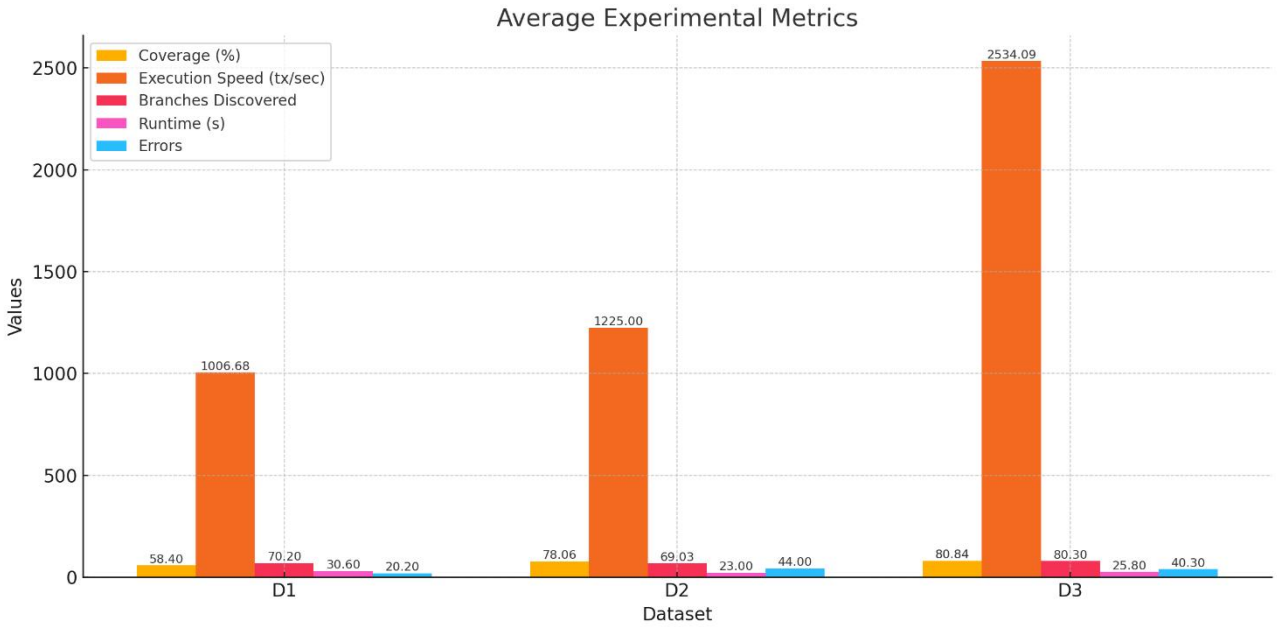
- Reentrancy (17):
  - Phát hiện 17 lỗ hổng reentrancy. Đây là lỗ hổng phổ biến trong các hợp đồng thông minh (smart contracts), cho phép kẻ tấn công gọi lại (re-enter) hàm không mong muốn và khai thác logic chương trình.

- Integer Overflow (440):
    - Số lượng lỗ hổng tràn số (integer overflow) được phát hiện là 440 lỗ hổng. Đây thường là lỗi liên quan đến các phép toán vượt quá giới hạn lưu trữ của kiểu dữ liệu.
- Gasless (17):
    - Có 17 trường hợp lỗ hổng gasless được phát hiện. Đây là lỗi liên quan đến việc tính toán gas không chính xác trong các giao dịch hợp đồng thông minh.
- Block Number Dependency (20):
    - Phát hiện 20 lỗ hổng phụ thuộc vào số block (block number). Đây là lỗi liên quan đến việc sử dụng số block hiện tại trong các tính toán quan trọng, dễ dẫn đến kết quả không mong muốn khi block thay đổi.

C. Kết quả thực nghiệm

Bảng số liệu thống kê các thông số trung bình khi chạy các hợp đồng trong dataset được bài báo đề cập:

	Numbers of Contracts	**Coverage (%)**	**Execution Speed (tx/sec)**	**Branches Discovered**	**Runtime (s)**	**Số lỗi**
D1	7,000 small contracts	58.4	1006.68	70.2	30.6	20.2
D2	100 contracts	78.06	1225.0	69.03	23.0	44.0
D3	100 contracts	80.84	2534.09	80.3	25.8	40.3



Sơ đồ thống kê trung bình các kết quả chạy thực nghiệm

D. Hướng phát triển

<Nêu hướng phát triển tiềm năng của đề tài này trong tương lai. Nhận xét về tính ứng dụng của đề tài>.

Hướng phát triển của MuFuzz tập trung vào nâng cao hiệu quả fuzzing và tối ưu hóa tài nguyên. MuFuzz cần cải tiến phân tích phụ thuộc dữ liệu và loại bỏ luồng giả để tập trung vào các nhánh mã quan trọng. Bên cạnh đó, phân tích chuỗi sẽ được tối ưu nhằm tạo các biến thể đầu vào hiệu quả, giảm thiểu số lần thực thi lặp lại trong quá trình fuzzing. Chiến lược tạo chuỗi giao dịch hợp lý, cho phép chuyển trực tiếp đến các trạng thái trung gian mà không cần thực hiện lại toàn bộ chuỗi, giúp tiết kiệm thời gian và mở rộng phạm vi khám phá trạng thái tiềm năng. Đồng thời, việc tích hợp các kỹ thuật học máy và học tăng cường sẽ tự động hóa quy trình fuzzing, nâng cao hiệu quả và tối ưu hóa tài nguyên sử dụng. Những cải tiến này sẽ giúp MuFuzz đáp ứng tốt hơn các yêu cầu phân tích và bảo mật hợp đồng thông minh.

Sinh viên đọc kỹ yêu cầu trình bày bên dưới trang này

YÊU CẦU CHUNG

- Sinh viên tìm hiểu và thực hiện bài tập theo yêu cầu, hướng dẫn.
- Nộp báo cáo kết quả chi tiết những việc (**Report**) bạn đã thực hiện, quan sát thấy và kèm ảnh chụp màn hình kết quả (nếu có); giải thích cho quan sát (nếu có).
- Sinh viên báo cáo kết quả thực hiện và nộp bài.

Báo cáo:

- File .PDF. Tập trung vào nội dung, không mô tả lý thuyết.
- Đặt tên theo định dạng: [Mã lớp]-Project_Final_NhomX_Madetai. (trong đó X và Madetai là mã số thứ tự nhóm và Mã đề tài trong danh sách đăng ký nhóm đồ án).

  Ví dụ: [NT521.N11.ANTT]-Project_Final_Nhom03_CK01.
- Nếu báo cáo có nhiều file, nén tất cả file vào file .ZIP với cùng tên file báo cáo.
- Nộp file báo cáo trên theo thời gian đã thống nhất tại courses.uit.edu.vn.

Đánh giá:

- Hoàn thành tốt yêu cầu được giao.
- Có nội dung mở rộng, ứng dụng.

Bài sao chép, trễ, ... sẽ được xử lý tùy mức độ vi phạm.

HẾT