BÁO CÁO TỔNG KẾT ĐỒ ÁN MÔN HỌC

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

**Tên chủ đề: Fuzz4All: Universal Fuzzing with Large Language Models**

*Mã nhóm: G15 Mã đề tài: CK28*

**Lớp**: **NT521.N11.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 | Đinh Trần Việt Khoa | 22520666 | 22520666@gm.uit.edu.vn |
| 2 | Đào Nhật Duy | 22520312 | 22520312@gm.uit.edu.vn |
| 3 | Trần Quang Kiệt | 21522265 | 21522265@gm.uit.edu.vn |
| 4 | Nguyễn Trần Duy Anh | 20520393 | 20520393@gm.uit.edu.vn |

1. **TÓM TẮT NỘI DUNG THỰC HIỆN:[[1]](#footnote-1)**
   1. **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: ……………….

* 1. **Tên bài báo tham khảo chính:**

|  |
| --- |
| Fuzz4All: Universal Fuzzing with Large Language Models |

* 1. **Dịch tên Tiếng Việt cho bài báo:**

|  |
| --- |
| Công cụ Fuzz4All: Fuzzing tổng quát với các mô hình ngôn ngữ lớn |

* 1. **Tóm tắt nội dung chính:**

|  |
| --- |
| Các chương trình phần mềm như: Compiler, Runtime engine, Constraint solver, Library,… là những phần mềm nhận dữ liệu đầu vào gồm các ngôn ngữ lập trình hoặc ngôn ngữ hình thức. Những chương trình này đặc biệt quan trọng và là nền tảng để xây dựng và phát triển các hệ thống phần mềm.  Chính vì vậy mà việc kiểm thử và đánh giá các phần mềm như thế này là một vấn đề cực kì quan trong. Một trong nhưng kĩ thuật kiểm thử đem lại nhiều hiệu quả và thành công trong việc tìm kiếm và phát hiện các lỗ hổng bảo mật là kĩ thuật kiểm thử Fuzzing.  Đối với từng chương trình Compiler, Runtime engine,… thì có rất nhiều ngôn ngữ lập trình và ngôn ngữ hình thức khác nhau cho nên có rất nhiều bộ Fuzzers áp dụng cho những chương trình như này nhưng chỉ hướng đến mục tiêu là một ngôn ngữ lập trình hoặc ngôn ngữ hình thức cụ thể. Do đó, không thể dễ dàng để áp dụng những Fuzzers này lên một ngôn ngữ khác hoặc thậm chí là các phiên bản của cùng một ngôn ngữ mà nó hỗ trợ.  Mục tiêu của bài báo là giới thiệu bộ công cụ Fuzz4All mang tính phổ quát có thể áp dụng trên nhiều ngôn ngữ khác nhau và đồng thời đánh giá mức độ hiệu quả trên một số chương trình phần mềm |

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

|  |
| --- |
| Bài báo sử dụng 2 kỹ thuật chính áp dụng trong công cụ Fuzz4All:   * Kỹ thuật 01 (Autoprompting): nhận dữ liệu đầu vào (code example, documentation, specification,…) từ người dùng và tự động phân tích, tổng hợp những user input này thành một prompt ngắn gọn và hiệu quả để sử dụng trong quá trình tạo fuzzing input      * Kỹ thuật 02 (Fuzzingloop): chịu trách nhiệm tạo ra các fuzzing input để đưa vào mô hình để kiểm thử, đồng thời liên tục cập nhật và chỉnh sửa prompt ban đầu để fuzzing input tạo ra đa dạng hơn |

* 1. **Môi trường thực nghiệm của bài báo:**

|  |
| --- |
| * Cấu hình máy tính: 64-core workstation with 256 GB RAM running Ubuntu 20.04.5 LTS with 4 NVIDIA RTX A6000 GPUs * Các công cụ hỗ trợ sẵn có:   + GPT-4-0613 với max\_token = 500 token sử dụng làm Distillation LLM cho kĩ thuật Autoprompting  + Bigcode/startcoderbase với max\_output=1024 token sử dụng làm Generation LLM cho cả 2 kĩ thuật Autoprompting và Fuzzingloop   * Ngôn ngữ lập trình để hiện thực phương pháp: Python * Tổng thời gian fuzzing: 24 giờ (bao gồm cả qua trình Autoprompting) * Số lượng fuzzing input tạo ra từ prompt: 30 input / prompt * Đối tượng nghiên cứu (chương trình phần mềm dùng để kiểm tra tính khả thi của phương pháp/tập dữ liệu – nếu có): đánh giá trên 6 ngôn ngữ khác nhau (C, C++, Go, SMT2, Java và Python) tương ứng với 6 chương trình (GCC, Clang, Z3, CVC5, OpenJDK, Qiskit * Tiêu chí đánh giá tính hiệu quả của phương pháp: tổng số lỗ hổng tìm kiếm được, coverage, hit rate |

* 1. **Kết quả thực nghiệm của bài báo:**

|  |
| --- |
| **Nhận xét:** công cụ Fuzz4All đã phát hiện được tổng cộng 98 bugs, trong đó có 64 bugs đã được xác nhận bởi lập trình viên. Tỉ lệ Coverage của công cụ Fuzz4All so với các công cụ khác như: GrayC, Csmith, Go-Fuzz, TypeFuzz, Yarpgen,… là cao nhất.  **Khả năng:** fuzzing inputs tạo ra từ công cụ Fuzz4All thực thi được nhiều phần mã nguồn của chương trình và có khả năng phát hiện các lỗi tìm ẩn cao hơn so với các fuzzer còn lại  **Ưu điểm:**   * Tạo input thông minh hơn, có ngữ cảnh và nhắm đến từng kịch bản cụ thể * Cải thiện coverage giúp input tạo ra đa dạng hơn từ đó nâng cao khả năng phát hiện các lỗi tìm ẩn * Khả năng mở rộng và áp dụng linh hoạt trên nhiều hệ thống phần mềm khác nhau   **Nhược điểm:**   * Chi phí tài nguyên phần cứng để vận hành các mô hình Large Langue Model (LLM) cao * Hiệu suất không ổn định do các mô hình ngôn ngữ hoạt động chủ yếu dựa vào ngữ cảnh và có thể tạo ra kết quả không nhất quán nếu yêu cầu không được cung cấp một cách rõ ràng * Hạn chế trong việc phát hiện các lỗi phức tạp * Phụ thuộc vào các tập dữ liệu để huấn luyện các mô hình ngôn ngữ |

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

|  |
| --- |
| Các công việc mà nhóm thực hiện cho đề tài:   * Nghiên cứu 2 kĩ thuật sử dụng trong công cụ Fuzz4All * Thực nghiệm lại đánh giá trong bài báo trên bộ công cụ GCC với score function mặc định dùng trong bài báo * Thực nghiệm lại đánh giá trong bài báo trên bộ công cụ GCC với score function do nhóm điều chỉnh   Các công việc đã hoàn thành:   * Nghiên cứu 2 kĩ thuật sử dụng trong công cụ Fuzz4All: nhóm đã tìm hiều, phân tích và trình bày lại hai kĩ thuật là Autoprompting và Fuzzingloop * Thực nghiệm đánh giá với score function mặc định: tạo ra hơn 600 fuzzing input để đánh giá và tổng hợp kết quả * Thực nghiệm đánh giá với score function điều chỉnh: tạo ra hơn 800 fuzzing input để đánh giá và tổng hợp kết quả |

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

|  |
| --- |
| Nhìn chung kết quá của nhóm không đạt được như kết quả mong đợi trong bài báo bởi một số khó khăn gặp phải:   * Thiết bị nhóm chạy thực nghiệm chỉ hỗ trợ CPU nên tốc độ chậm hơn nhiều so với việc sử dung GPU theo như trong bài báo. * Model bigcode/starcoderbase để làm Generation LLM trong bài báo quá nặng nên nhóm phải chuyển sang bigcode/starcoderbase-1b nhẹ hơn nhưng cho kết quả thiếu chính xác hơn. * Vì chỉ sử dụng CPU để chạy nên số lượng fuzzing input tạo ra của nhóm rất nhỏ so với số lượng trong bài báo và tổng thời gian fuzzing của giảm đi một nửa (từ 24 giờ -> 12 giờ) |

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

|  |
| --- |
| 90% |

1. **NHẬT KÝ PHÂN CÔNG NHIỆM VỤ:**

|  |  |  |
| --- | --- | --- |
| **STT** | **Công việc** | **Phân công nhiệm vụ** |
| 1 | Nghiên cứu, tổng hợp lí thuyết chung | Cả nhóm |
| 2 | Làm slide báo cáo | Đinh Trần Việt Khoa |
| 3 | Cài đặt và cấu hình môi trường thực nghiệm | Trần Quang Kiệt |
| 4 | Thực nghiệm và đánh giá | Trần Quang Kiệt |

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**

**1. Tổng quan công cụ fuzz4All**

**A diagram of a system

Description automatically generated**

*Hình A.1.1: Sơ đồ hoạt động Fuzz4All*

Bao gồm:

* Input từ người dùng: documentation (tài liệu mô tả về chương trình cần kiểm thử), example code (mẫu code để tạo fuzzing input),…
* Mô hình Distillation LLM: phân tích và tổng hợp để tạo ra prompt ngắn gọn và hiệu quả nhất
* Mô hình Generation LLM: tạo fuzzing input
* System Under Test: hệ thống và chương trình cần được kiểm thử

**2. Quy trình hoạt động tổng quan của công cụ Fuzz4All**

* Nhận dữ liệu đầu vào từ người dùng cung cấp mô tả các tính năng cụ thể hoặc các mẫu ví dụ cho trước.
* Input được đưa vào mô hình Distillation LLM để phân tích và tóm lược thành các prompt ngắn gọn.
* Những prompt vừa được tạo ra này sẽ được đánh giá thông qua một score function để chọn ra một prompt hiệu quả nhất
* Prompt này sao đó được đưa vào Generation LLM để tạo ra fuzzing input để đưa vào chương trình kiểm thử. Đồng thời prompt luôn được cập nhật liên tục để tạo ra các fuzzing input đa dạng hơn

**3. Kĩ thuật Autoprompting**

**Input:** dữ liệu mô tả chương trình của người dùng

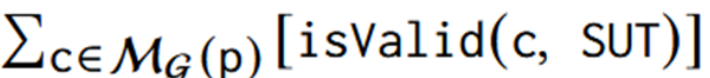
**Output:** prompt tốt nhất

Cơ chế hoạt động của thuật toán:

* Đưa user input vào mô hình Distillation LLM với ***APInstruction*** (là yêu cầu hướng dẫn LLM tóm tắt user input), ***temp*** = 0 (là một tham số đảm bảo kết quả nhất quán và ít có yếu tố ngẫu nhiên) để tạo ra ***Greedy Prompt*** (prompt đầu tiên trong danh sách các Candidate Prompt cần tạo)
* Ở mỗi vòng lặp thì tương tự cũng sẽ tạo ra một Prompt sử dụng Distillation LLM với ***APInstruction*** nhưng thay đổi ***temp*** = 1 (tham số đảm bảo kết quả sẽ có yếu tố ngẫu nhiên). Sau đó Prompt này cũng được thêm vào danh sách Candidate Prompt
* Vòng lặp được lặp lại nhiều lần cho đến khi đạt đủ số lượng danh sách Candidate Prompt yêu cầu
* Sau khi số lượng Prompt đã đạt yêu cầu thì sẽ dùng hàm Score Function để lựa chọn ra Prompt được đánh giá là tốt nhất làm kết quả trả về

Cách Score Function hoạt động:

* Từng Prompt trong danh sách Candidate Prompts sẽ được đưa vào mô hình Generation LLM để tạo ra các Fuzzing Input tạm thời.
* Nhưng Fuzzing Input này sẽ được đưa vào SUTs để đánh giá
* Score được định nghĩa như sau:



*Hình A.3.1: Công thức tính score function*

Trong đó:

C: Fuzzing input (code được tạo ra bởi Generation LLM)

P: Prompt

* Có thể hiểu là Score sẽ bằng tổng số lượng Fuzzing input được tạo ra bởi một prompt được tính là hợp lệ (biên dịch thành công,…) khi đưa vào SUT

Mã giả của thuật toán:

A white box with black text

Description automatically generated

*Hình A.3.2: Thuật toán Autoprompting*

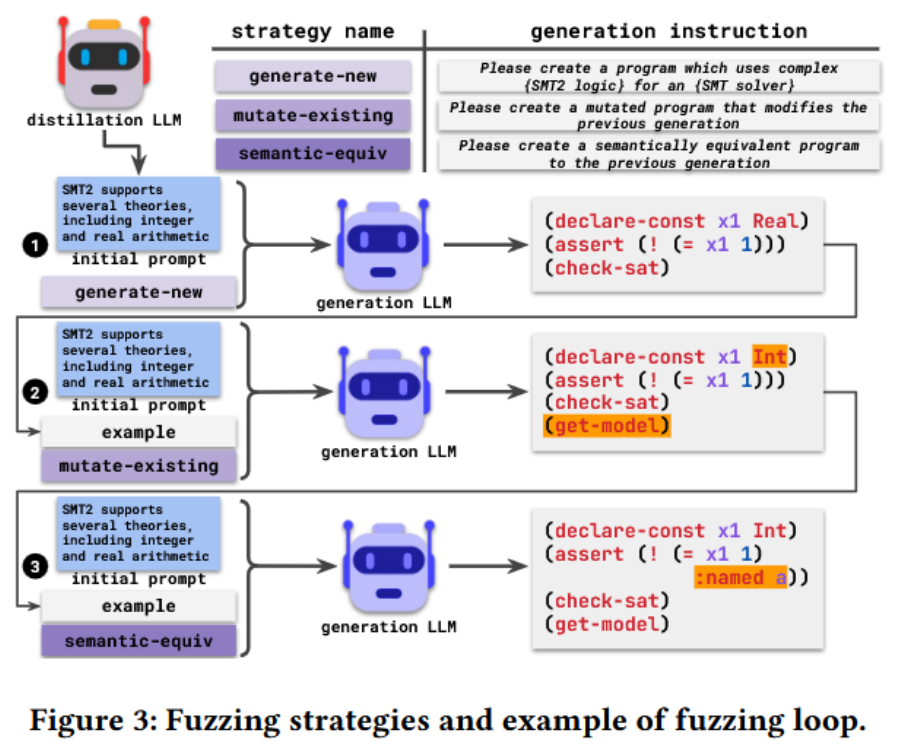
**4. Kĩ thuật Fuzzingloop**

**Input:** Prompt tốt nhất được chọn từ bước Autoprompting

**Output:** Lỗ hổng được phát hiện

Cơ chế hoạt động của Fuzzingloop:

* Để đảm bảo fuzzing input tạo ra không bị trùng lặp và mang tính đa dạng cao thì đầu tiên là cần thiết lập 3 chiến lược để làm biến đổi Input Prompt ban đầu. Ba chiến lược bao gồm: generation-new, mutation-existing và semantic-equiv. Mỗi một chiến lược sẽ có một api instruction riêng để hướng dẫn mô hình thực hiện



*Hình A.4.1: Ba chiến lược biến đổi Input Prompt*

* Mặc định lượt fuzzing input đầu tiên sẽ được tạo ra bằng cách sử dụng Input Prompt và chiến lược generation-new và sẽ được được vào SUT để kiểm thử
* Trong mỗi vòng lặp, ngẫu nhiên chọn ra một fuzzing input từ tập inputs đã tạo ra ở trước làm mẫu ví dụ. Sau đó chọn ngẫu nhiên 1 trong 3 chiến lược đề cập ở trên. Cuối cùng nhóm fuzzing inputs tiếp theo được tạo ra bằng cách sử dụng mẫu ví dụ kết hợp với Input Prompt và Chiến lược instruction
* Tương tự thì cứ mỗi lần một nhóm các Fuzizng Input được tạo ra thì nó sẽ được đừa vào SUT để kiểm thử. Vòng lặp sẽ liên tục lặp đi lặp lại cho đến khi đạt được thời gian yêu cầu
* Kết quả là danh sách lỗ hổng đã phát hiện được

Mã giả thuật toán Fuzzingloop:

A screenshot of a computer program

Description automatically generated

*Hình A.4.2: Thuật toán Fuzzing Loop*

**B. Chi tiết cài đặt, hiện thực**

Cách cài đặt công cụ Fuzz4All: <https://zenodo.org/records/10456883>

Cấu hình thực nghiệm của nhóm

|  |  |  |
| --- | --- | --- |
|  | **Nhóm** | **Paper** |
| **Environment** | VMware Unbuntu 20.04.5.LTS sử dụng Docker Container làm môi trường thực nghiệm | "64-core workstation with 256 GB RAM running Ubuntu 20.04.5 LTS with 4 NVIDIA  RTX A6000 GPUs (only one GPU is used per fuzzing run)." |
| **Distillation LLM** | "gpt-4-0613, max\_token 500" | "gpt-4-0613, max\_token 500" |
| **Generation LLM** | "bigcode/starcoderbase-1b, maximum output 1,024 token" | "bigcode/starcoderbase, maximum output 1,024 token" |
| **Device Run** | CPU | GPU |
| **Batch\_size fuzzing input** | 5 | 30 |
| **Total time fuzzing** | 12 hours | 1. hours |

*Hình B.1: Sơ đồ chi tiết cấu hình của nhóm với bài báo*

Kịch bản thực nghiệm:

* Thực hiện kiểm thủ Fuzzing sử dụng Fuzz4All trên compiler GCC 13.1.0.
* Nhóm thực hiện hai trường hợp gồm:
* Sử dụng hàm Score Function mặc định của bài báo.
* Sử dụng hàm Score Function được chỉnh sửa lại.
* Đánh giá kết quả dựa trên các tiêu chí: coverage, tỉ lệ fuzzing input tạo ra hợp lệ.
* Đối với hàm Score Function được chỉnh sửa lại như sau:
* Score sẽ được tính dựa trên số lượng fuzzing inputs tạo ra là hợp lệ và độ phức tạp của fuzzing inputs theo tỉ lệ:



*Hình B.2: Công thức score function chỉnh sửa*

* Độ phức tạp dựa trên công thức Cyclomtic Complexity:



*Hình B.3: Công thức tính độ phức tạp*

**N:** điểm quyết định hoặc khối lệnh trong luồng điều khiển

**P:** số đoạn mã đọc lập của chương trình

**E:** số đường dẫn kết nối giữa các điểm N

**C. Kết quả thực nghiệm**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Default** | **Modifed** | **Paper** |
| **Coverage** | 106,728 | 137,523 | 198,927 |
| **Programs** | 600 | 800 | 44,324 |
| **Valid** | 42,83% | 53,20% | 37.26% |

*Hình C.1: Kết quả thực nghiệm của nhóm*

**Nhận xét:** nhìn chung kết quá của nhóm không đạt được như kết quả mong đợi trong bài báo bởi một số khó khăn gặp phải

* Thiết bị nhóm chạy thực nghiệm chỉ hỗ trợ CPU nên tốc độ chậm hơn nhiều so với việc sử dung GPU theo như trong bài báo
* Model bigcode/starcoderbase để làm Generation LLM trong bài báo quá nặng nên nhóm phải chuyển sang bigcode/starcoderbase-1b nhẹ hơn nhưng cho kết quả thiếu chính xác hơn

**D. Kết quả thực nghiệm**

Fuzz4All trong tương lai có thể mở rộng hỗ trợ nhiều ngôn ngữ lập trình, tích hợp trí tuệ nhân tạo để tối ưu việc phát hiện lỗ hổng và tích hợp vào hệ thống CI/CD để kiểm thử tự động. Công cụ sẽ phát triển giao diện trực quan, hỗ trợ kiểm thử trên nền tảng đám mây và thúc đẩy hợp tác cộng đồng để nâng cao hiệu quả, tính phổ quát và khả năng ứng dụng trong nhiều lĩnh vực.

**Nhận xét:** Fuzz4All có tính ứng dụng cao trong kiểm thử và bảo mật phần mềm, với khả năng hỗ trợ nhiều ngôn ngữ lập trình, phát hiện lỗ hổng hiệu quả và tích hợp vào quy trình phát triển liên tục. Tính linh hoạt và khả năng mở rộng giúp công cụ phù hợp với nhiều hệ thống, từ IoT, blockchain đến AI, mang lại độ tin cậy cao và giảm thiểu rủi ro.

*Sinh viên báo cáo các nội dung mà nhóm đã thực hiện, có thể là 1 phần hoặc toàn bộ nội dung của bài báo. Nếu nội dung thực hiện có khác biệt với bài báo (như cấu hình, tập dữ liệu, kết quả,...), sinh viên cần chỉ rõ thêm khác biệt đó và nguyên nhân.*

---

***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**

1. Ghi nội dung tương ứng theo mô tả [↑](#footnote-ref-1)