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: The Relevance of Classic Fuzz Testing: Have We Solved This One?

Mã nhóm: G08. Mã đề tài: CK03

Lớp: NT521.N11.ANTN

1. THÔNG TIN THÀNH VIÊN NHÓM:

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

STT	Họ và tên	MSSV	Email
1	Đỗ Minh Khôi	20520592	20520592@gm.uit.edu.vn
2	Phạm Trần Thanh Quang	20521810	20521810@gm.uit.edu.vn
3	Nguyễn Bảo Phương	20520704	20520704@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:

B. P. Miller, M. Zhang and E. R. Heymann, "The Relevance of Classic Fuzz Testing: Have We Solved This One?," in IEEE Transactions on Software Engineering, vol. 48, no. 6, pp. 2028-2039, 1 June 2022, doi: 10.1109/TSE.2020.3047766.

[1] Ghi nội dung tương ứng theo mô tả

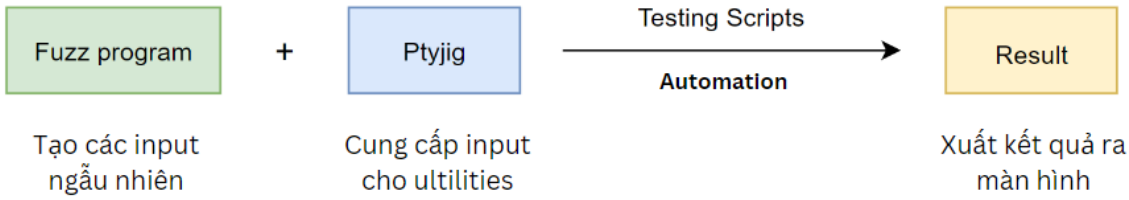
C. Dịch tên Tiếng Việt cho bài báo:

Tầm quan trọng của kiểm thử Fuzz cổ điển: Chúng ta đã giải quyết được chưa?

D. Tóm tắt nội dung chính:

- Câu hỏi được đặt ra rằng liệu công nghệ fuzz cổ điển có còn giá trị trong thời đại hiện nay hay không. Để trả lời câu hỏi này, tác giả đã cập nhật các công cụ fuzz cơ bản cũng như các đoạn script và thử chúng với các tiện ích họ Unix - có sẵn trên Linux, FreeBSD và Mac OS. Tiêu chí thất bại của nhóm tác giả là chương trình có bị lỗi (crash) hay bị treo hay không.
- Nhóm tác giả nhận thấy rằng 9 tiện ích bị lỗi hoặc treo trong số 74 tiện ích trên Linux, 15 trong số 78 tiện ích trên FreeBSD và 12 trong số 76 tiện ích trên MacOS. Có tổng cộng 24 tiện ích khác nhau bị lỗi trên ba nền tảng. Tỷ lệ lỗi này cao hơn một chút so với các nghiên cứu trước đó của nhóm tác giả trong các năm 1995, 2000 và 2006 về độ tin cậy của các tiện ích dòng lệnh.
- Nhóm tác giả đã tiến hành debug từng tiện ích bị lỗi và phân tích nguyên nhân gây ra lỗi. Các loại lỗi cổ điển, chẳng hạn như lỗi con trỏ, lỗi mảng hoặc không kiểm tra return code vẫn xuất hiện khá nhiều trong các chương trình hiện nay. Ngoài ra, nhóm tác giả còn tìm thấy một số loại lỗi mới xuất hiện.
- Nhóm tác giả cũng đã thử nghiệm với một số các tiện ích có sẵn trong một ngôn ngữ lập trình hiện đại (Rust) và nhận thấy độ tin cậy của chúng cũng không tốt hơn so với các tiện ích cổ điển.

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



- Đối tượng được fuzz: các tiện ích có trên 3 hệ điều hành là ubuntu, macOS và FreeBSD
- Kỹ thuật chính được mô tả trong bài viết là black-box fuzzing. Đưa các input vào để kiểm tra các tiện ích, dựa vào kết quả **return code** của quá trình fuzzing để xác định tiện ích có bị treo/ crash hay không.
- Fuzzer bao gồm các thành phần:
  - **./fuzz**: đây là tool dùng để sinh ra chuỗi kí tự input.
  - **./ptyjig**: tool đẩy những input được sinh ra từ *./fuzz* làm đầu vào cho các tiện ích.
  - **Automation scripts**: tự động sinh ra input với các mẫu có kích thước khác nhau và tự động thực hiện fuzzing trên nhiều tiện ích cùng lúc.

F. Môi trường thực nghiệm của bài báo:

- **Cấu hình máy tính:**
  - **[1] Linux**: Ubuntu 18.04.3

    Spec: Intel 3.20GHz Core i5 using GCC version 7.5.0
  - **[2] MacOS:** Mojave 10.14.5

    Spec: Intel 2.4GHz Core i5 using AppleLLVM version 10.0.0.
  - **[3] FreeBSD:** cài đặt trên VirtualBox 6.0.14 sử dụng image FreeBSD-11.3-RELEASE-amd64-disc1.iso

    Spec: 4GB RAM và 20GB dynamic allocated virtual box disk

- **Các công cụ hỗ trợ sẵn có:** Fuzz (sinh chuỗi ngẫu nhiên), Ptyjig (cung cấp input cho các tiện ích đọc input từ terminal)
- **Ngôn ngữ lập trình để hiện thực phương pháp:** C, Rust
- **Đối tượng nghiên cứu** (chương trình phần mềm dùng để kiểm tra tính khả thi của phương pháp/tập dữ liệu – nếu có): Kiểm thử 84 tiện ích trên 3 nền tảng (Linux, MacOS, FreeBSD), trong đó có 68 tiện ích có thể hoạt động trên cả 3 nền tảng.
- **Tiêu chí đánh giá tính hiệu quả của phương pháp:** Số liệu crash/hang, lí do crash/hang.

G. Kết quả thực nghiệm của bài báo:

Có 9 lỗi trong số 74 tiện ích kiểm tra trên Linux (12%), 12 trên 76 trên MacOS (16%) và 15 trên 78 trên FreeBSD (19%). Nhóm tác giả đã tạo ra lỗi trong tổng số 24 tiện ích khác nhau, 10 trong số đó gây ra lỗi trên nhiều nền tảng và ba trong số đó lỗi trên cả ba nền tảng (gdb và trff). 3 trong số 14 tiện ích lõi (coreutils) được viết bằng Rust mà khi thử nghiệm đã bị treo (comm, fold và ptx), so với chỉ một coreutil bị treo trên Linux, MacOS hoặc FreeBSD (ptx).

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

- **Tìm hiểu về phương pháp kiểm thử fuzz testing**:

- Nền tảng lý thuyết của kỹ thuật được nhóm tác giả sử dụng.
- Nguyên lý hoạt động của fuzzer. Tham khảo mả nguồn của fuzzer.
- Các giai đoạn tổng quan của quá trình fuzzing.

- **Tải và cài đặt fuzzer được sử dụng trong bài báo**:

- Thực hiện compile các tool của fuzzer sử dụng Makefile theo hướng dẫn của file README có trong mã nguồn.

- Dùng lệnh **man** (trên Ubuntub) và đọc các thư mục README đi kèm với mã nguồn để chuẩn bị đầy đủ và hiểu rõ cách hoạt động của các tool cần thiết.

- **Triển khai fuzzing**:

- Thử các chức năng của tool ***./fuzz*** và ***./ptyjig*** đã lấy về bằng cách chạy trực tiếp fuzzing như trong báo cáo giữa kỳ. Quan sát các kết quả.
- Thay thế việc chạy trực tiếp fuzzing bằng cách dùng các scripts tự động hóa.
- Lựa chọn một số tiện ích tiêu biểu, chạy quá trình fuzzing và so sánh với một vài kết quả đã có trong bài báo.

- **Tìm hiểu lí do phát sinh crash/ treo tiện ích**:

- Tác giả có cung cấp một số nguyên nhân gây ra crash/ treo ở một số tiện ích bằng cách phân tích mã nguồn của tiện ích đó. Dựa vào các lí do được tác giả đưa ra, nhóm thực hiện tìm mã nguồn và phân tích lại 1 trong những lỗi đó.

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

- Trong quá trình thực hiện, nhóm gặp phải một số khó khăn, thách thức sau:
  - Lựa chọn tiện ích nào để có thể đưa ra được nhận xét và so sánh với báo cáo của tác giả.
  - Đưa ra số lượng test cases, kích thước mỗi test cases, thời gian thực hiện fuzzing để có báo cáo đầy đủ nhất.
  - Thống kê kết quả trực tiếp mà không có sẵn công cụ thống kê kết quả tự động, điều này khiến việc fuzzing mất thời gian.
- Với những khó khăn đối với phần cứng, nhóm đã được giảng viên hướng dẫn nên đã có thể giải quyết các vấn đề về phần cứng cũng như dung lượng.

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

95%

4. NHẬT KÝ PHÂN CÔNG NHIỆM VỤ:

STT	Công việc	Phân công nhiệm vụ
1	Tổng kết và lên kế hoạch thực hiện công việc.	Nguyễn Bảo Phương
2	Tóm tắt ý tưởng, môi trường và kết quả tổng quan của thử nghiệm	Đỗ Minh Khôi, Phạm Trần Thanh Quang
3	Đóng góp và triển khai demo.	Phạm Trần Thanh Quang, Nguyễn Bảo Phương
4	Làm slide	Đỗ Minh Khôi, Nguyễn Bảo Phương
5	Thuyết trình nội dung lý thuyết	Đỗ Minh Khôi
6	Thuyết trình demo	Phạm Trần Thanh Quang
7	Viết báo cáo tổng kết	Phạm Trần Thanh Quang, Đỗ Minh Khôi

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

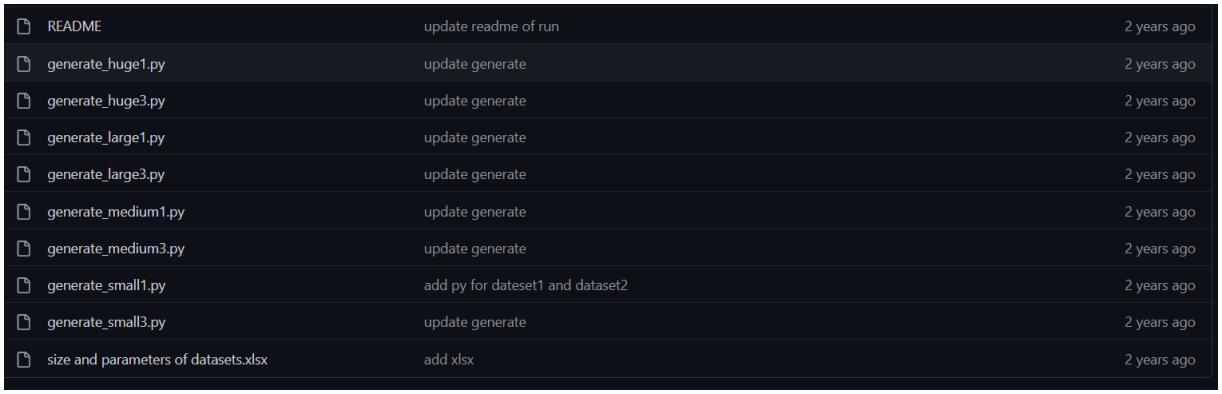
a) Kiến trúc và thành phần của fuzzer trong bài báo

1. Tài nguyên và thành phần

- Nhóm tác giả cung cấp đường dẫn đến github - nơi lưu mã nguồn của fuzzer và các đoạn script giúp tự động hóa quá trình fuzzing:

https://github.com/dyninst/fuzz

- Trong đường dẫn trên, ta có các thư mục như sau:
  - **./src**: gồm mã nguồn của fuzzer được nhóm tác giả sử dụng trong bài, bao gồm file **fuzz.c** và **ptyjig.c**.
  - **./doc**: đây là tài liệu hướng dẫn sử dụng fuzzer có trong bài báo, bao gồm cách sử dụng, lệnh và các options được dùng chung tương ứng với từng thành phần có trong fuzzer.
  - **./generate_test**: gồm các file code Python giúp tự động tạo ra các file input cho quá trình fuzzing, các file input này sẽ chứa các bytes kí tự ngẫu nhiên. Kích thước của mỗi file input được nhóm tác giả cung cấp là:
    - Small: 1000 kí tự hoặc 10 dòng, mỗi dòng 100 kí tự
    - Medium: 100.000 kí tự hoặc 1000 dòng, mỗi dòng 100 kí tự
    - Large: 10.000.000 kí tự hoặc 100.000 dòng, mỗi dòng 100 kí tự
    - Huge: 100.000.000 kí tự hoặc 1.000.000 dòng, mỗi dòng 100 kí tự



Hình 1. *Các file python input generator cho các kích thước khác nhau*

- **./run_test**: gồm file **run.py** dùng để thực thi việc tự động hóa quá trình fuzzing, ngoài ra còn có một thư mục khác có tên là **end**, đây là thư mục dùng để thêm kí tự kết thúc tiện ích, dành cho một số tiện ích đặc biệt. Một số thư mục như **test_Linux**, **test_FreeBSD**, … sẽ bao gồm các đoạn script được quy ước sẵn để tự động hóa việc fuzzing trên các nền tảng hệ điều hành tương ứng như Linux, FreeBSD, ...

Hình 2. *Nội dung của thư mục **run_test***

- Nếu muốn việc download fuzzer diễn ra nhanh chóng, thay vì dùng đường dẫn github bên trên, nhóm tác giả có cung cấp thêm một đường dẫn để download fuzzer bên trên sử dụng ftp như sau:

ftp://ftp.cs.wisc.edu/paradyn/fuzz/fuzz-2020/

2. Kiến trúc và hướng thực hiện của tác giả

- Nhóm tác giả thực hiện fuzzing các tiện ích cơ bản có trên 3 hệ điều hành là Ubuntu, FreeBSD và MacOS - đây đều là 3 phiên bản hệ điều hành của họ UNIX.

- Quá trình triển khai: sử dụng công cụ fuzzer và đoạn script tự động có sẵn để thực hiện fuzzing, sau đó kiểm tra kết quả bằng cách thống kê trực tiếp (không thống kê kết quả tự động). Đối tượng được hực hiện fuzzing là các tiện ích thông dụng có trong họ UNIX, ví dụ: ls (lệnh list file và directory), …
- Việc fuzzing trong bài báo không hướng tới việc các tiện ích này có chạy đúng các chức năng hay không mà nhằm mục đích kiểm tra xem các tiện ích này có thể bị crash/treo (hang) với một số lượng ký tự input cho trước.
- Khi thực hiện fuzzing, để biết tiện ích hoạt động tốt hoặc có bị crash hay không thì dựa vào giá trị **return code**, còn để biết chương trình có bị treo hay không thì cần dựa vào kết quả quan sát trực tiếp.

b) Kiến trúc và thành phần của fuzzer mà nhóm đã thực hiện trên thực tế

1. Tài nguyên và thành phần

- Nhóm thực hiện fuzzing sử dụng tài nguyên được tác giả bài báo cung cấp.
- Trong các file python input generator có trong thư mục **generate_test**, nhóm lựa chọn sẽ dùng **3 loại kích thước** cho file input cho quá trình fuzzing, đó là: Small, Large, Huge.

2. Kiến trúc và hướng thực hiện

- Nhóm chỉ thực hiện fuzzing trên hệ điều hành Ubuntu mà không thực hiện trên các hệ điều hành khác.
- Triển khai fuzzing tương tự như ý tưởng và hướng dẫn của nhóm tác giả, tuy nhiên có rút ngắn bớt một số file dữ liệu cũng như số lượng tiện ích được kiểm thử.

B. Chi tiết cài đặt, hiện thực

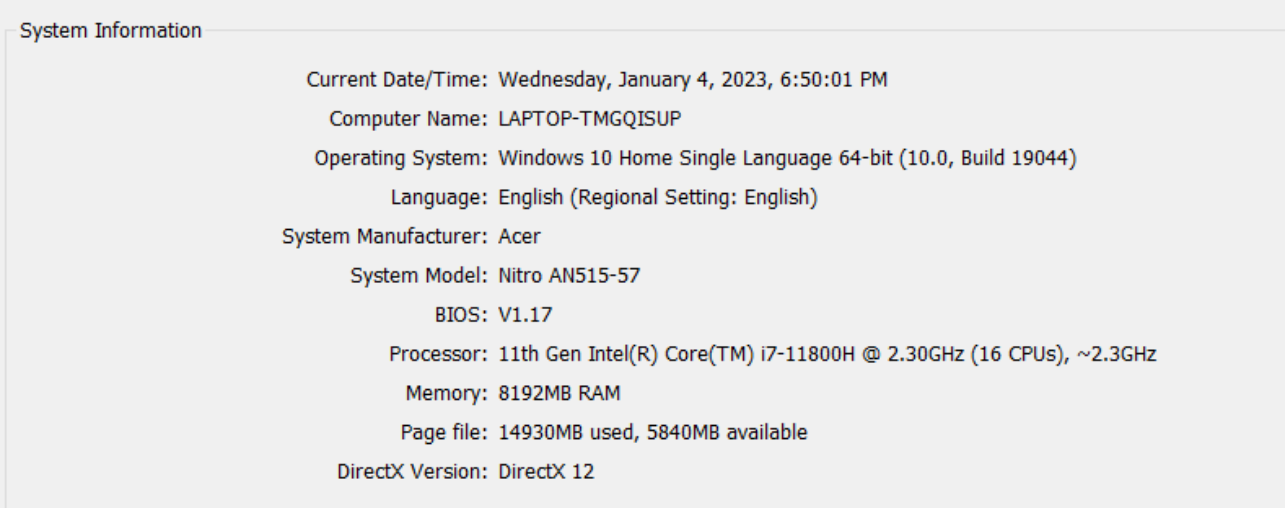
1. Cấu hình máy tính sử dụng

- Về tác giả: thực hiện trên hệ điều hành Ubuntu phiên bản **18.04.3**, trên **CPU Intel 3.20GHz Core i5**, phiên bản GCC trên máy là **7.5.0**.
- Về nhóm: thực hiện fuzzing trên một máy ảo có hệ điều hành Ubuntu phiên bản **18.04.5**:

```
ubuntu@ubuntu1804:~/DoAn_LTAT/fuzz2020/test/Small1$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 18.04.5 LTS
Release:        18.04
Codename:       bionic
ubuntu@ubuntu1804:~/DoAn_LTAT/fuzz2020/test/Small1$
```

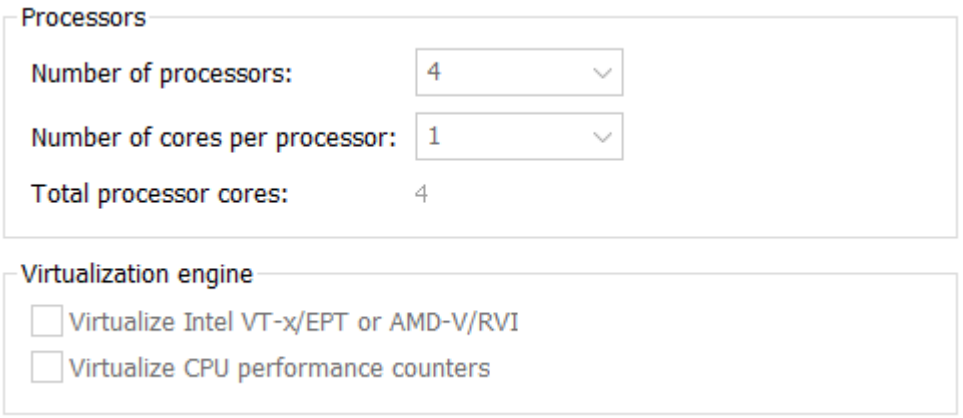
Hình 3. *Phiên bản máy thực nghiệm là 18.04.5*

- Máy thật chạy Windows 10 có **CPU Intel 11th 2.30GHz Core i7**:



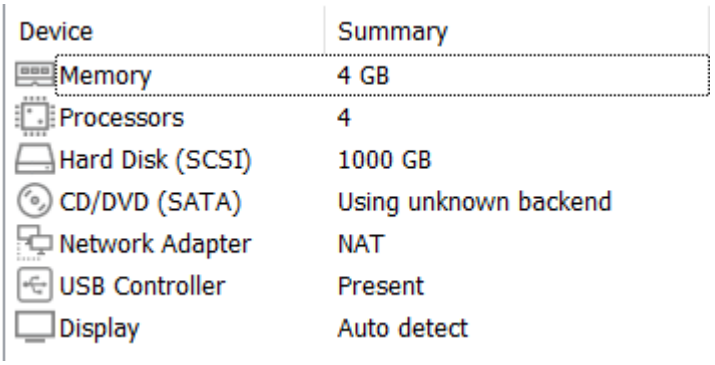
Hình 4. *Cấu hình máy thật: chạy Win10 và thông tin CPU*

- Máy ảo được cấp **4 processors** ảo dựa trên CPU máy thật:



Hình 5. *Cấu hình CPU virtual processors cấp cho máy ảo*

- Máy có 4GB RAM, ảo hóa dựa vào phần mềm VMware Workstation 17 Pro (17.0.0 build-20800274). Việc lựa chọn 4GB RAM là vì khi chạy fuzzing dùng Huge input file với 2GB RAM cho máy ảo, máy của nhóm đã bị crash nhiều lần, ảnh hưởng đến các tiện ích hệ thống khác.

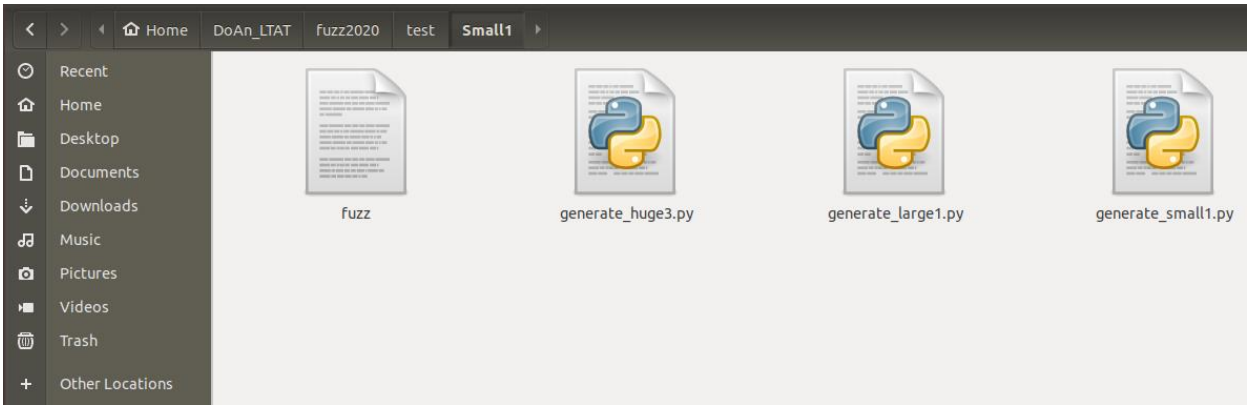


Hình 6. *Tổng quan về cấu trúc của máy Ubuntu được thực hiện*

- Nhóm không thể thỏa mãn các yêu câu về CPU bởi vì các thành viên chưa thể kiếm được cấu hình CPU giống hệt như trong bài báo. Về việc lựa chọn phiên bản Ubuntu, nhóm kiểm tra version của các tiện ích có trong phiên bản Ubuntu **18.04.5** hoàn toàn giống với Ubuntu **18.04.3**. Ngoài ra, việc lựa chọn

2. Chuẩn bị dữ liệu

- Như đã nói ở trên, ta sẽ tạo ra các file input cho fuzzing bằng cách sử dụng các file python có trong thư mục **generate_test**:



Hình 7. *Các file python generate input được nhóm lựa chọn cho demo*

- Với small input file: sử dụng file **generate_small1.py**
- Với large input file: sử dụng file **generate_large1.py**

- Với large input file: sử dụng file **generate_huge3.py**

- Quan sát đoạn code của file generate_small1.py:

Open ▾ generate_small1.py
~/DoAn_LTAT/fuzz2020/test/Small1

```
#  Copyright (c) 2020 Emma He, Mengxiao Zhang, Barton Miller
#
#  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.

# this script is used to generate small test cases.

import os, sys, re
import subprocess
import random
import time

fnull = open(os.devnull, 'w')
path = "./Small1"

if not os.path.exists(path):
  os.mkdir(path)

inc = 200
start = 0

# -0
for i in range(start, start+inc):
  print(i)
  if os.path.isfile(os.path.join(path, "t%d" % i)):
    continue
  # sleep more than 1 second to prevent fuzz from using the same random seed as the previous one.
  time.sleep(1.1)
  n = random.randint(0, 1e3)
  subprocess.call(["./fuzz", "%d" % n, "-0", "-o", os.path.join(path, "t%d" % i)], stdout=fnull, stderr=subprocess.STDOUT)
start = start + inc
```

Hình 8. *Các file python generate input được nhóm lựa chọn cho demo*

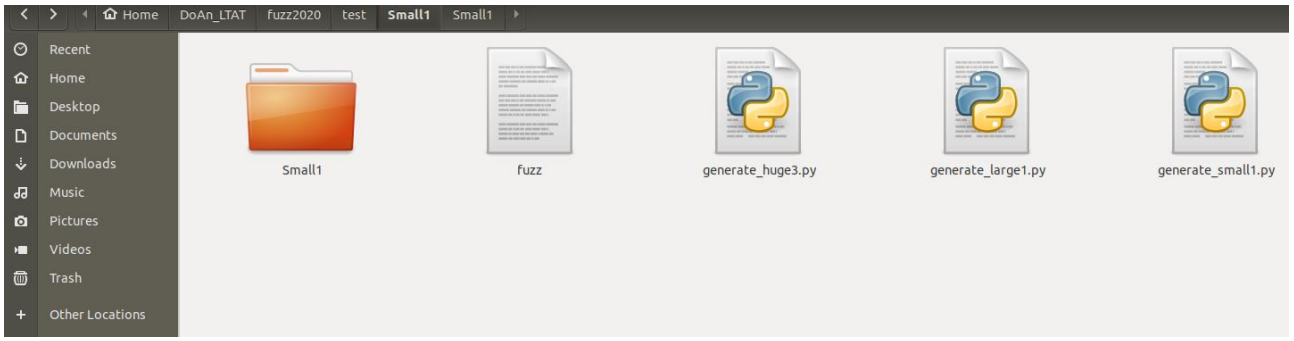
- Thực hiện chạy python code để tạo ra cac file input:

```
ubuntu@ubuntu1804:~/DoAn_LTAT/fuzz2020/test/Small1$ python3 generate_small1.py
0
1
2
3
4
5
6
7
8
9
```

Hình 9. *Tạo ra các file input với kích thước là Small*

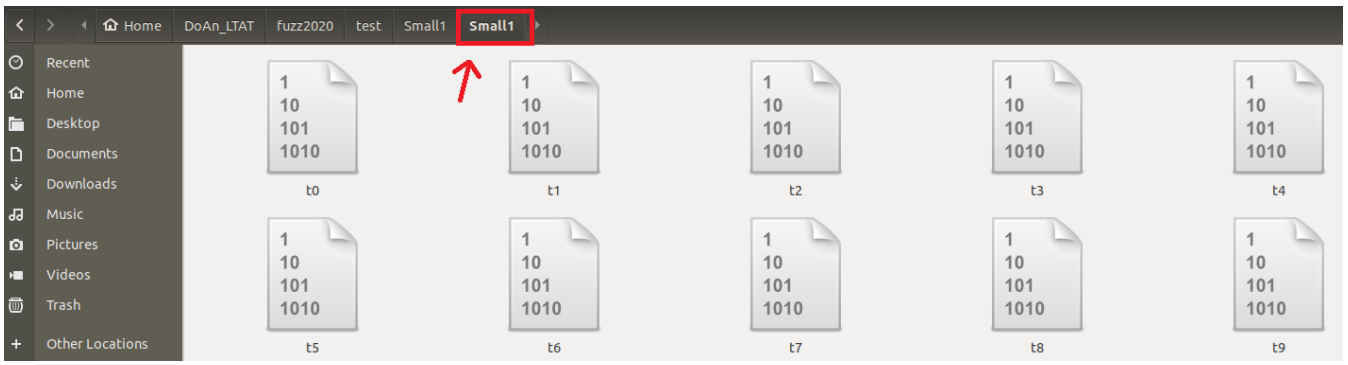
Quan sát cách chạy của chương trình và source code của file **generate_small1.py**, ta thấy chương trình tạo ra các file input bằng cách gọi chương trình **fuzz** để tạo ra có chuỗi ký tự

có độ dài random từ 0 đến 1e3. Các con số trên terminal chính là số thứ tự của các file input được tạo ra và được đặt trong một thư mục có tên là **Small1**:



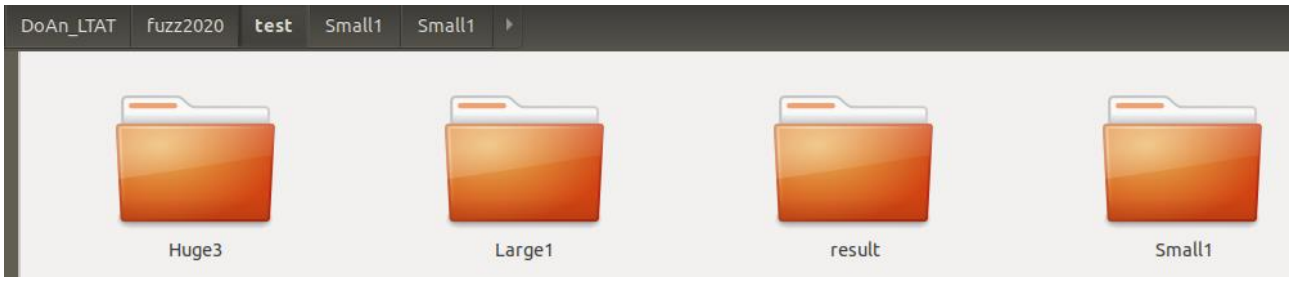
Hình 10. *Thư mục **Small1** được tạo ra*

Các file input sẽ có tên là "**t**" kết hợp với số thứ tự được tạo ra trên terminal:



Hình 11. *Các file input được sử dụng*

- Tương tự như vậy, ta cũng chạy các file **generate_large1.py**, **generate_huge3.py** để lần lượt tạo ra 2 folder **Large1** và **Huge3**:



Hình 12. *Đã tạo ra các folder với kích thước test khác nhau*

- Các file này sẽ là input để nhóm thực hiện fuzzing.

3. Triển khai fuzzing

- Để thực hiện tự động hóa việc fuzzing, ta sử dụng cú pháp sau:

python3 run.py **config_file** -i **input_file**

trong đó:

  - run.py: đây là file được dùng
  - config_file: là file nằm trong thư mục **./test_Linux** đã đề cập ở trên. config_file có 3 file, bao gồm: **regular**, **pty** và **coreutil_rust** (file này dùng cho một số tiện ích có mã nguồn là rust)**.** config_file chứa các đoạn script được tác giả quy ước sẵn và khi file run.py được chạy, nó sẽ đọc hiểu các đoạn script này để tự động hóa quá trình fuzzing.
  - -i [input_file]: là thư mục input đầu vào cho fuzzing, đây chính là các thư mục **Small1**, **Large1** và **Huge3** đã được tạo ra ở trên.

- Các tiện ích được nhóm tập trung quan sát là ***as, speel*** và ***troff*** (hay **groff - GNU roff**). Lí do nhóm sử dụng 3 tiện ích này để thực hiện fuzzing mà không quan sát tất cả các tiện ích là vì số lượng tiện ích quá nhiều, ngoài ra, tác giả sử dụng phương pháp thống kê kết quả fuzzing trực tiếp chứ không tự động hóa việc thống kê, do đó để tránh mất thời gian nhưng vẫn đảm bảo khai thác được các góc độ trong bài báo, nhóm đã thống nhất chỉ lấy 3 tiện ích trên để có được kết quả báo cáo thực nghiệm.
- Khi tìm kiếm config_file cho 3 tiệc ích này, nhóm thấy có config_file là **regular** có đoạn script để fuzzing cho cả 3 tiện ích này:

Open

regular
~/DoAn_LTAT/Fuzz2020/test

```
stdin dc
stdin dd
two_files diff [--normal -q -s -e -n -y --left-column --suppress-common-lines -p -F -t -T --suppress-blank-empty]
stdin ed [-G -l -r -s -v]
file eqn [-C -N -r -R]
file expand [-i]
stdin flex [-Ca -Ce -Cf -Cm -Cr -f -F -Cem -d -b -p -s -T -w -v --hex -7 -8 -B -i -l]
file fmt [-c -s -t -u]
file fold [-b -s]
cp t.c gcc [-c -S -E -g]
stdin gdb [-write -nh -nx -quiet -batch -fullname]
cp t.f gfortran [-static-libgfortran -cpp -C -P]
stdin grep a [-P -i -v -w -x -L -l -o -q -s -H -h -n -u -Z -a -I -r -R -U -Z]
file groff [-e -k -l -N -p -s -S -t -U -z -Z -a -b -c -C -E]
stdin head [-q -v -z]
two_files join [-i --check-order --nocheck-order --header -z]
file look a [-b -d -f]
file m4 [-E -P -Q -s -g]
stdin mail -s "subject" avengerzmx@outlook.com <
file make [-B -d -e -i -k -L -n -p -q -r -R -s -S -t -w] -f
file md5sum [-b --tag -z]
stdin neqn
file nm [-a -B --no-demangle -D -g -l -n -v -u --synthetic]
stdin pdftex
file pic [-C -S -U -n -t -c -z -D]
file pr [-c -d -F -J -m -r -t -T -v]
stdin ptx [-A -G -R -f -r -t]
file rev
two_files sdiff [-i -E -Z -b -W -B --strip-trailing-cr -d -l -s -t -d -H]
cp tmp sed 's/a/b/' [-n --follow-symlinks --posix -s -u -z]
file soelim [-C -r -t]
stdin sort [-b -g -r]
file spell [-l -n -s -v -x]
file split [-d -x -e -u]
file strings [-a -d -f -w]
cp tmp strip [-s -g --strip-dwo --strip-unneeded -M -p -D -U -w -x -X --keep-file-symbols --only-keep-debug]
stdin sum [-r -s]
file tail [-q --retry -z]
stdin tbl [-C]
stdin tee [-a -p]
stdin tex [-file-line-error -no-file-line-error -file-line-error-style -ini -ipc -mltex -recorder -shell-escape -no-shell-escape -src-specials]
stdin tr a b
file troff [-a -b -c -C -E -U -z]
```

Hình 13. *Tiện ích được test trong file config regular*

- Đọc nội dung của file run.py để hiểu rõ cách hoạt động của file này:

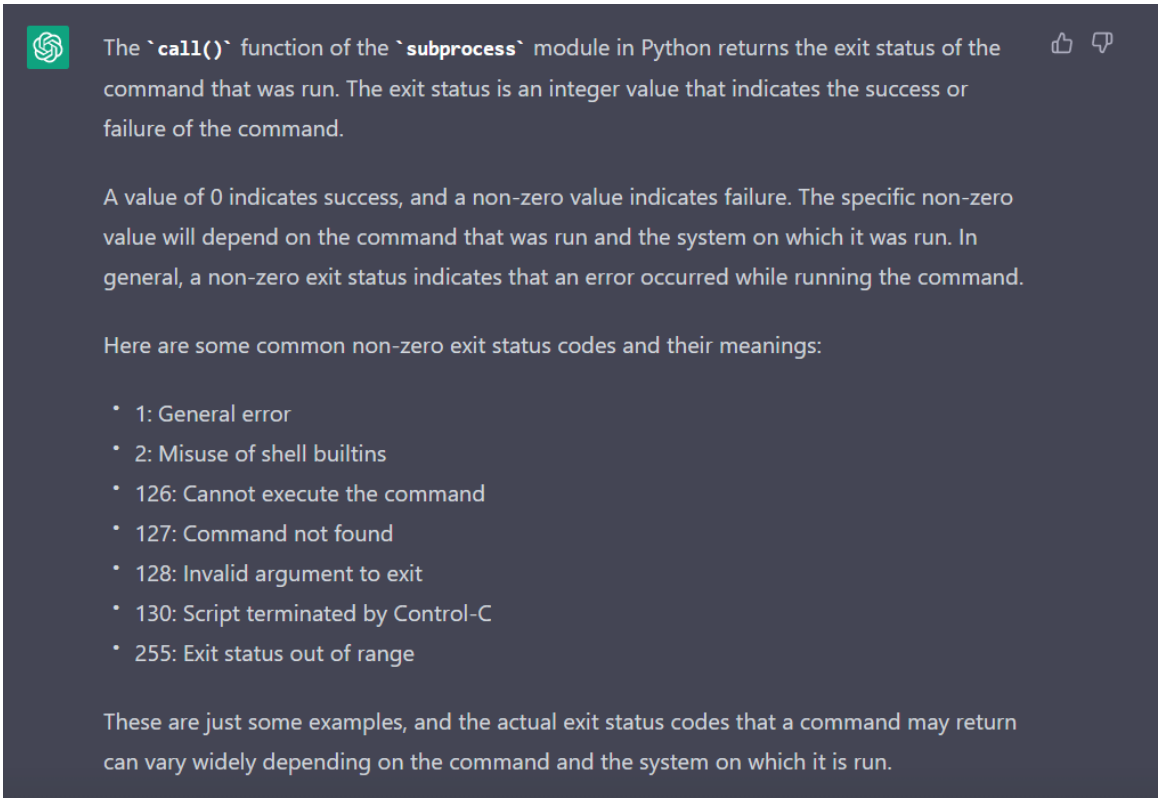
```
try:
  retcode = subprocess.call(final_cmd, shell=True, stdout=fnull, stderr=subprocess.STDOUT, timeout=timeout)

except(subprocess.TimeoutExpired):
  log_writer.write("%s hung\n" % final_cmd)

else:
  print("retcode is %d" % retcode)
  if(utility_name not in ("sh", "csh", "zsh") and retcode == 127):
    log_writer.write("%s not found\n" % utility_name)
    break
  # check return value, record exit code with special meaning
  if retcode >= return_value or retcode < 0:
    log_writer.write("%s failed, error: %d\n" % (final_cmd, retcode))
```

Hình 14. *Cách hoạt động của run.py*

Biến **retcode** là 1 biến quan trọng trong chương trình **run.py**. Quan sát thấy giá trị của biến retcode chính là giá trị trả về của hàm **subprocess.call()** trong python, xem thử giá trị trả về của hàm này ở hình bên dưới:



Hình 15. *Giá trị trả về của hàm **subprocess.call()** trong python*

Kết hợp giữa đoạn code của **run.py** và hình trên, ta kết luận rằng khi **retcode** có giá trị lớn hơn **131** hoặc nhỏ hơn **0** thì chương trình chính là bị crash.

- Đối với các tiện ích bị treo: trong file **run.py**, tác giả cung cấp thời gian để xác định một chương trình có trị treo hay không dựa vào thời gian **timeout** với mặc định là **300 giây**, tức là, sau **300 giây** mà tiện ích vẫn chưa chạy xong input thì tiện ích đó bị treo:

```
# if the cmd does not finish in timeout(300 by default) seconds, the test result will be considered as a hang
timeout = 300
```

Hình 16. *Giá trị timeout để xem tiện ích có bị treo không*

- Nếu chương trình thực sự bị treo, tác giả đã có quy định trong file **run.py**:

```
try:
  retcode = subprocess.call(final_cmd, shell=True, stdout=fnull, stderr=subprocess.STDOUT, timeout=timeout)

except(subprocess.TimeoutExpired):
  log_writer.write("%s hung\n" % final_cmd)

else:
  print("retcode is %d" % retcode)
  if(retcode == 127):
    log_writer.write("%s not found\n" % utility_name)
    break
  # check return value, record exit code with special meaning
  if retcode >= return_value or retcode < 0:
    log_writer.write("%s failed, error: %d\n" % (final_cmd, retcode))
```

Hình 17. *Cách xử lý khi tiện ích bị treo*

Quan sát đoạn code trên, ta thấy khi timeout thì chương trình **run.py** sẽ không hiển thị ra màn hình bất kỳ thông tin gì. Do đó, nếu tiện ích hoạt động đúng hoặc ngay cả khi tiện ích bị crash, không tìm thấy, lỗi input, … thì luôn có retcode trả về, ***còn khi fuzzing tiện ích mà không trả về retcode thì tiện ích đó chắc chắn đã bị treo***.

4. Cài đặt trên máy tính

- Nhóm thực hiện kiểm tra kết quả chủ yếu trên 3 tiện ích chính là **as**, **spell** và **troff**, do đó nhóm trình bày version của 3 tiện ích này trên máy ảo mà nhóm sử dụng:

```
ubuntu@ubuntu1804:~/DoAn_LTAT/fuzz2020/test$ as --version
GNU assembler (GNU Binutils for Ubuntu) 2.30
Copyright (C) 2018 Free Software Foundation, Inc.
This program is free software; you may redistribute it under the terms of
the GNU General Public License version 3 or later.
This program has absolutely no warranty.
This assembler was configured for a target of `x86_64-linux-gnu'.
ubuntu@ubuntu1804:~/DoAn_LTAT/fuzz2020/test$ spell --version
spell: version 1.0
ubuntu@ubuntu1804:~/DoAn_LTAT/fuzz2020/test$ troff --version
GNU troff (groff) version 1.22.3
ubuntu@ubuntu1804:~/DoAn_LTAT/fuzz2020/test$
```

Hình 18. *Kiểm tra version của tiện ích có trên máy nhóm sử dụng*

Kết quả cho thấy version hoàn toàn trùng khớp với nhóm tác giả:

Utility	Linux	
	version	fail
as	2.30	o

spell	1.0
split	8.28
strings	2.30
strip	2.30
sum	8.28
tail	8.28
tbl	1.22.3
tcsh	—
tee	1.22.3
telnet	1.14
tex	6.2.3
top	3.3.12
tr	8.28
troff	1.22.3

Hình 19. *Version của các tiện ích được nhóm demo*

Như vậy, nhóm không cần cài đặt thêm các tiện ích hoặc các version cho phù hợp vì các tiện ích được nhóm quan tâm đều đã có cấu hình tương ứng với bài báo.

C. Kết quả thực nghiệm

- Dựa trên báo cáo chi tiết của tác giả:

Utility	Linux		MacOS		FreeBSD	
	version	fail	version	fail	version	fail
as	2.30	○	11.0.0		2.17.50	○
awk	4.1.4		20070501		20121220	
bash	4.4.20		3.2.57	●	5.0.16	
bc	1.07.1		1.07.1		1.1	
bison	3.0.4	○	3.3	○	3.4.2	
calendar	*		1.19	●	8.3	
cat	8.28		1.32		8.2	
checknr	–		1.9		8.1	●
clang	8.0.0		11.0.0		8.0.0	
cmp	3.6		2.8.1		8.3	
col	*		1.19		8.5	●
colcrt	*		1.18		8.1	
colrm	*		1.12		8.2	
comm	8.28		1.21		8.4	
compress	–		1.23		8.2	
csh	20110502-5		–		–	
ctags	25.2		5.8_1	●	8.4	●
cut	8.28		1.30		8.3	
dash	0.5.10.2-6		*		0.5.10.2	
dc	1.4.1	○	1.3	● ○	1.3	
dd	8.28		1.36		8.5	
diff	3.6		2.8.1		2.8.7	
ed	1.10		*		1.5	
eqn	1.22.3		1.19.2		1.19.2	
ex/vim	8.0		8.1		8.1	
expand	8.28		1.15		8.1	
flex	2.6.4		2.5.35		2.5.37	●
fmt	8.28		1.22		8.1	
fold	8.28		1.13		8.1	
ftp	0.17-34.1		–		8.6	●
gcc	7.4.0		–		9.2.0	
gdb	8.1.0	●	8.3.1	●	6.1.1	●
gfortran	7.4.0		–		–	
grep	3.1		2.5.1		2.5.1	
grn	–		1.19.2		1.19.2	
groff	1.22.3		1.19.2	○	1.19.2	○
head	8.28		1.20		8.2	
htop	2.1.0		2.2.0		2.2.0	
indent	–		5.17	●	5.17	●
join	8.28		1.2		8.6	
less	551	●	487	●	530	
lldb	–		9.0.1	●	8.0.0	●

Utility	Linux		MacOS		FreeBSD	
	version	fail	version	fail	version	fail
look	*		1.18.10		8.2	○
m4	1.4.18		1.4.6		1.4.18_1	
mail	*		8.2		8.2	
make	4.1		3.8.1		8.3	●
md5/md5su	8.28		1.34		*	
mig	–		116		–	
more	2.31.1		–		–	
neqn	1.22.3		1.19.2		1.19.2	
nm	2.30		11.0.0		3504	
pdftex	6.2.3		6.2.3	●	6.2.1	
pic	1.22.3		1.19.2		1.19.2	
pr	8.28		1.18		8.2	
ptx	8.28	○	–		–	
refer	–		1.19.2		1.19.2	
rev	2.31.1		1.12		8.3	
sdiff	3.6		2.8.1		1.36	
sed	4.4		1.39		8.2	
sh	–		–		8.6	●
soelim	1.22.3		1.19.2		*	
sort	8.28		2.3		2.3	
spell	1.0	○	–		–	
split	8.28		1.17		8.2	
strings	2.30		*		r3614M	
strip	2.30		*		r3614M	
sum	8.28		1.17		*	
tail	8.28		101.40.1		8.1	
tbl	1.22.3		1.19.2		1.19.2	
tcsh	–		6.21.00		6.20.00	
tee	1.22.3		1.6		8.1	
telnet	1.14		1.16		8.4	
tex	6.2.3	●	6.2.3		6.2.1	
top	3.3.12		125		3.5beta12	
tr	8.28		1.24		8.2	
troff	1.22.3	●	1.19.2	●	1.19.2	●
tsort	8.28		1.13		8.3	
ul	*		101.40.1		8.1	
uniq	8.28		101.40.1		8.3	
units	–		*		*	
wc	8.28		1.21		8.1	
xargs	4.7.0		1.57		8.1	
zic	2.27		8.22		8.22	
zsh	5.4.2		5.7.1		5.7.1	●

87 utilities were tested on Unix, MacOS, and freeBSD, 67 of which were tested on all three systems.
● = crashed, ○ = hung, – = unavailable on that system, * = version information unavailable.

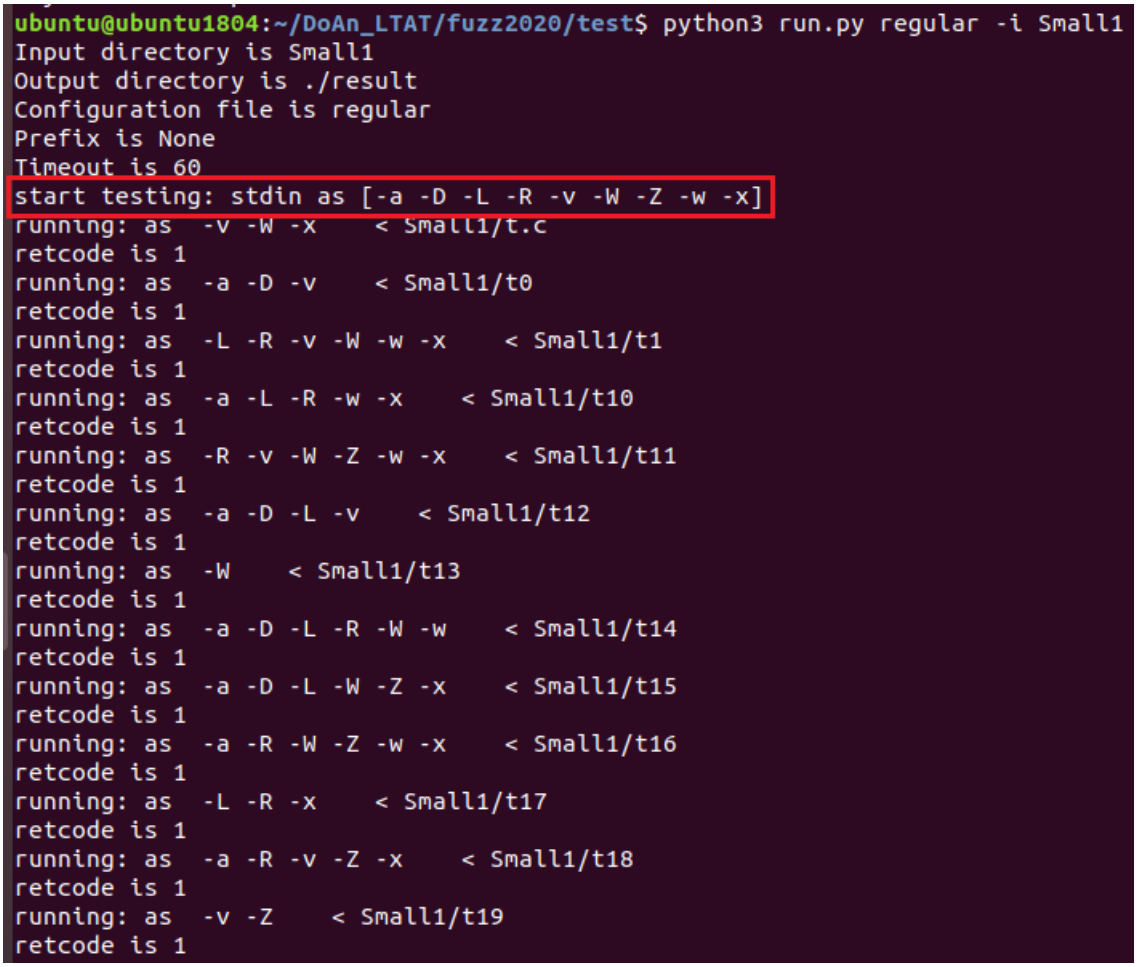
Hình 20. *Kết quả thử nghiệm của tác giả*

Nhóm rút ra được:

- Tiện ích **as** bị treo.
- Tiện ích **spell** bị treo.
- Tiện ich **troff** bị crash.

1. **Fuzzing với file input có kích thước Small (khoảng 1000 kí tự)**

- Trước hết, fuzzing với thư mục input là **Small1** *(số lượng file input: 49 files)*:



Hình 21. *Chạy script test cho file input với kích thước **Small***

Kết quả thực tế cho thấy:

- Đối với tiện ích **as**: **retcode = 1**, tiện ích mắc một số lỗi (general error) chứ **không hề bị treo.**
- Đối với tiện ích **spell**: **retcode = 0**, tiện ích hoạt động bình thường, **không hề bị treo.**

```
start testing: file spell [-l -n -s -v -x]
running: spell  -n -v    Small1/t.c
retcode is 0
running: spell  -l -n    Small1/t0
retcode is 0
running: spell  -l -x    Small1/t1
retcode is 0
running: spell  -l -s -v -x    Small1/t10
retcode is 0
running: spell  -s -v -x    Small1/t11
retcode is 0
running: spell  -l -n -s -x    Small1/t12
retcode is 0
running: spell  -l -s -x    Small1/t13
retcode is 0
running: spell  -l -n -s -v    Small1/t14
retcode is 0
running: spell  -l -x    Small1/t15
retcode is 0
running: spell  -s -v -x    Small1/t16
retcode is 0
running: spell  -n -s    Small1/t17
retcode is 0
running: spell  -l -s    Small1/t18
retcode is 0
running: spell  -n -s -v -x    Small1/t19
retcode is 0
running: spell  -n -v    Small1/t2
retcode is 0
running: spell  -s -v -x    Small1/t20
retcode is 0
running: spell  -l    Small1/t21
retcode is 0
running: spell  -n -s    Small1/t22
```

Hình 22. *Tiện ích **spell** với **Small** input*

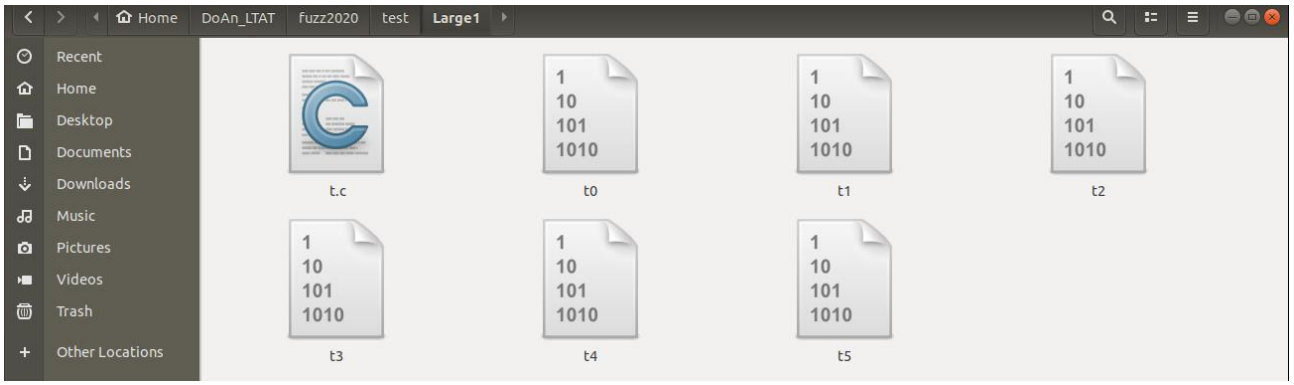
- Đối với tiện ích **troff**: **retcode = 0**, tiện ích hoạt động bình thường, **không hề bị crash.**

```
start testing: file troff [-a -b -c -C -E -U -z]
running: troff  -c -C -U -z    Small1/t.c
retcode is 0
running: troff  -a -b -c -C -E -z    Small1/t0
retcode is 0
running: troff  -a -E    Small1/t1
retcode is 0
running: troff  -c -C -E    Small1/t10
retcode is 0
running: troff  -a -b -c -C -E    Small1/t11
retcode is 0
running: troff  -C    Small1/t12
retcode is 0
running: troff  -a -E    Small1/t13
retcode is 0
running: troff  -a -b -c -E -z    Small1/t14
retcode is 0
running: troff  -b -z    Small1/t15
retcode is 0
running: troff  -a -b -C -E -U    Small1/t16
retcode is 0
running: troff  -b -C -E -U -z    Small1/t17
retcode is 0
running: troff  -a -b -c -C -E -U    Small1/t18
retcode is 0
running: troff  -z    Small1/t19
retcode is 0
running: troff  -a -c    Small1/t2
retcode is 0
running: troff  -C    Small1/t20
retcode is 0
running: troff  -b -z    Small1/t21
retcode is 0
running: troff     Small1/t22
retcode is 0
running: troff  -a -c -C -E -z    Small1/t23
retcode is 0
```

Hình 23. *Tiện ích **troff** với **Small** input*

2. Fuzzing với file input có kích thước Large (khoảng 10.000.000 kí tự)

- Fuzzing với **Large1** input file (số lượng input file: 7 files):



Hình 24. *7 files input với kích thước **large***

Kết quả thực tế cho thấy:

- Đối với tiện ích **as**: **retcode = 1**, tiện ích mắc một số lỗi (general error) chứ **không hề bị treo.**

```
ubuntu@ubuntu1804:~/DoAn_LTAT/fuzz2020/test$ python3 run.py regular -i Large1
Input directory is Large1
Output directory is ./result
Configuration file is regular
Prefix is None
start testing: stdin as [-a -D -L -R -v -W -Z -w -x]
running: as  -a -L -W -Z -w    < Large1/t.c
retcode is 1
running: as  -v -Z -w -x    < Large1/t0
retcode is 1
running: as  -L -v -W -x    < Large1/t1
retcode is 1
running: as  -L -x    < Large1/t2
retcode is 1
running: as  -D -L -v -W -Z    < Large1/t3
retcode is 1
running: as  -a -L    < Large1/t4
retcode is 1
running: as  -D -v -w -x    < Large1/t5
retcode is 1
```

Hình 25. *Test với tiện ích **as***

- Đối với tiện ích **spell**: tiện ích khi xử lý file **t0** đến file **t5** đều bị treo do không thấy trả về **retcode**. Vậy tiện ích này thực sự đã bị treo khi xử lý **Large input file.**

```
start testing: file spell [-l -n -s -v -x]
running: spell  -n -s -x    Large1/t.c
retcode is 0
running: spell  -n -s -v    Large1/t0
running: spell  -l -n    Large1/t1
running: spell  -n -s -v -x    Large1/t2
running: spell  -l -n -s -x    Large1/t3
running: spell  -v    Large1/t4
running: spell  -l -n -x    Large1/t5
finished: file spell [-l -n -s -v -x]
```

Hình 26. *Test với tiện ích **spell***

- Đối với tiện ích **troff**:

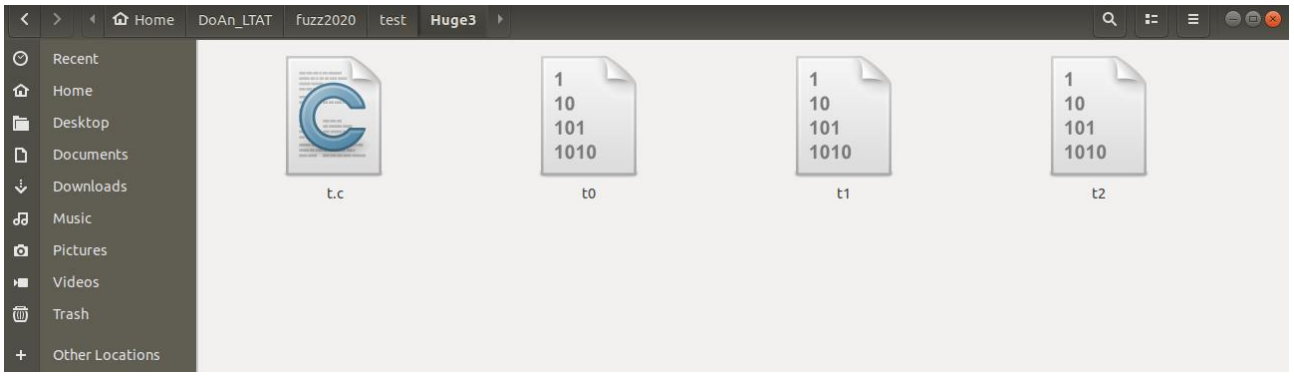
```
start testing: file troff [-a -b -c -C -E -U -z]
running: troff  -a -b -U    Large1/t.c
retcode is 0
running: troff  -a -b -c -E -z    Large1/t0
retcode is 139
running: troff  -b -C -U    Large1/t1
retcode is 0
running: troff  -a -b -c -C -E    Large1/t2
retcode is 139
running: troff  -c -C -E -U -z    Large1/t3
retcode is 139
running: troff  -a -b -c -E -U -z    Large1/t4
retcode is 139
running: troff  -a -b -c    Large1/t5
retcode is 0
finished: file troff [-a -b -c -C -E -U -z]
```

Hình 26. *Test với tiện ích **troff***

- 3/7 test cases: retcode = 0. Tiện ích hoạt động bình thường.
- 4/7 test cases: retcode = 139 (lớn hơn 131). Tiện ích bị crash.

3. Fuzzing với file input có kích thước Huge (khoảng 100.000.000 kí tự)

- Fuzzing với **Huge3** input file (số lượng input file: 4 files):



Hình 25. *4 files input với kích thước **Huge***

- Đối với tiện ích **as**: **retcode = 1**, tiện ích mắc một số lỗi (general error) chứ **không hề bị treo.**

```
ubuntu@ubuntu1804:~/DoAn_LTAT/fuzz2020/test$ python3 run.py regular -i Huge3
Input directory is Huge3
Output directory is ./result
Configuration file is regular
Prefix is None
Timeout is 300
start testing: stdin as [-a -D -L -R -v -W -Z -w -x]
running: as  -a -R -W -w -x    < Huge3/t.c
retcode is 1
running: as  -a -L -v -W -Z -x    < Huge3/t0
retcode is 1
running: as  -L -Z -w -x    < Huge3/t1
retcode is 1
running: as  -a -v -W -w -x    < Huge3/t2
retcode is 1
finished: stdin as [-a -D -L -R -v -W -Z -w -x]
```

Hình 26. *Test tiện ích **as** với file input có kích thước **Huge***

- Đối với tiện ích **spell**: tiện ích không trả về retcode, tiện ích bị **treo**.

```
start testing: file spell [-l -n -s -v -x]
running: spell  -n -s -v -x    Huge3/t.c
retcode is 0
running: spell  -s -v    Huge3/t0
running: spell  -n -v    Huge3/t1
running: spell  -n    Huge3/t2
finished: file spell [-l -n -s -v -x]
```

Hình 27. *Test tiện ích **spell** với file input có kích thước **Huge***

- Đối với tiện ích **troff**: tiện ích trả về retcode = 129 (lớn hơn 131), tiện ích bị **crash**.

```
start testing: file troff [-a -b -c -C -E -U -z]
running: troff  -a -b -c -z    Huge3/t.c
retcode is 0
running: troff  -a -c -E -U -z    Huge3/t0
retcode is 139
running: troff  -a -b -C -E -U    Huge3/t1
retcode is 139
running: troff  -a -C -E -U    Huge3/t2
retcode is 139
finished: file troff [-a -b -c -C -E -U -z]
```

Hình 28. *Test tiện ích **troff** với file input có kích thước **Huge***

- Chương trình troff được hệ điều hành báo bị crash khi fuzzing đang diễn ra:

Hình 29. *Tiện ích bị hệ điều hành báo lỗi*

4. Kết luận, giải thích

- Thông qua các kết quả trên, tổng kết lại thành bảng sau:

	Tiện ích **as**	Tiện ích **spell**	Tiện ích **troff**
Tác giả	Bị treo	Bị treo	Bị crash
Nhóm thực hiện	Gặp một số lỗi chung, không bị treo	Bị treo	Bị crash

- Kết luận:
    - Tiện ích **as** có sự khác biệt khi kiểm tra giữa tác giả và nhóm. Khác biệt này có thể đến từ nhiều nguyên nhân, tuy nhiên có 2 nguyên nhân chủ yếu:
        - **Sự khác biệt về phần cứng**: mặc dù nhóm tác giả sử dụng CPU core i5, tuy nhiên, xung nhịp lại lên đến 3.2GHz, ngoài ra tác giả cũng không hề đề cập đến đời CPU, dung lượng RAM, thời hạn đã sử dụng và loại ổ cứng được sử dụng.
        - **Độ phủ (Coverage) của kỹ thuật fuzzing này chưa cao**: có thể đánh giá phương pháp fuzzing của nhóm tác giả có độ phủ rất thấp. Khi nhìn vào đoạn source code của chương trình **fuzz**, ta thấy việc tạo ra chuỗi ký tự được sinh ra phụ thuộc quá nhiều vào hàm **rand()** của ngôn ngữ C. Ngoài ra, đây kỹ thuật của tác giả không phải là smart fuzzing nên input được sinh ra lúc sau sẽ được tính toán lại từ kết quả của những input trước.

```c
/*
 * Decide th effective range of the random characters
 */
void fuzz()
{
     int     m, h;

     /*
      * Every random character is of the form c = rand() % m + h
      */
     h = 1;
     m = 255;                    /* Defaults, 1-255 */
     if (flag0) {
          h = 0;
          m = 256;               /* All ASCII, including 0, 0-255 */
     }
     if (!flaga) {
          h = 32;
          m = 95 + (flag0!=0);   /* Printables, 32-126 */
     }
     if (flagl)
          fuzzstr(m, h);
     else
          fuzzchar(m, h);
}
```

```c
/*
 * Make a random character
 */
void fuzzchar(int m, int h)
{
     int     i,c;

     for (i = 0; i < length; i++) {
          c = (int) (rand() % m) + h;
          if (flag0 && !flaga && c == 127)
               c = 0;
          putch(c);
     }
}


/*
 * make random strings
 */
void fuzzstr(int m, int h)
{
     int     i, j, l, c;

     for (i = 0; i < length; i++) {
          l = rand() % flagl;    /* Line length  */
          for (j = 0; j < l; j++) {
               c = (int) (rand() % m) + h;
               if (flag0 && !flaga && c == 127)
                    c = 0;
               putch(c);
          }
          putch('\n');
     }
}
```

Hình 30. *Đoạn mã nguồn của tool **./fuzz** để tạo ra chuỗi ký tự random*

- Đối với 2 tiện ích ***spell*** và ***troff***: kỹ thuật fuzzing của tác giả tỏ ra hiệu quả khi việc thực nghiệm lại nhiều lần vẫn cho ra cùng một kết quả. Trong bài báo, tác giả có nêu nguyên nhân dẫn đến treo/ crash của 2 tiện ích này. Ví dụ, đối với tiện ích **troff**, lí do khiến tiện ích bị crash là do gặp phải bug **Failure to Check Return Value**. Xem xét đoạn mã nguồn của troff (hay GNU roff):

https://opensource.apple.com/source/groff/groff-39/groff/src/roff/troff/node.cpp

Quan sát mã nguồn, chú ý dòng code sau:

```
special_node::special_node(const macro &m, int n)
: mac(m), no_init_string(n)
{
  font_size fs = curenv->get_font_size();
  int char_height = curenv->get_char_height();
  int char_slant = curenv->get_char_slant();
  int fontno = env_definite_font(curenv);
  tf = font_table[fontno]->get_tfont(fs, char_height, char_slant, fontno);
  if (curenv->is_composite())
    tf = tf->get_plain();
  gcol = curenv->get_glyph_color();
  fcol = curenv->get_fill_color();
  is_special = 1;
}
```

Hình 31. *Một đoạn mã nguồn của tiện ích **troff** gây ra lỗi*

Hàm **env_definite_font()** có thể return về **-1** (error). Khi đó, **fontno = -1** và nếu mang giá trị này truy cập đến **font_table[fontno]** thì chương trình đang truy cập vào vùng nhớ không cho phép, dẫn đến tiện ích **troff** bị crash.

D. Hướng phát triển

1. Hướng phát triển đề tài

- Cải thiện về kỹ thuật:

  - Trong tương lai, nhóm có thể thay đổi kỹ thuật từ fuzzing một cách thụ động trở nên chủ động hơn dựa vào kỹ thuật smart fuzzing hay dynamic analysis technique. Những phương pháp này sẽ cho input ngày càng có giá trị hơn chứ không phải lặp đi lặp lại 1 chuỗi input vô nghĩa, từ đó giúp gia tăng độ phủ (coverage) của test case, thu hẹp phạm vi lỗi và thời gian fuzzing.
  - Có thể nâng cấp mã nguồn của ***./fuzz*** bằng cách tạo ra nhiều thuật toán random khác nhau mà không phụ thuộc quá nhiều vào hàm có sẵn của ngôn ngữ C.

- Cải thiện về đối tượng fuzzing: gia tăng số lượng các tiện ích được thử nghiệm và thực hiện nghiệm thử nghiệm trên các phiên bản mới nhất của mỗi hệ điều hành.

2. Tính ứng dụng của đề tài

- Nhóm tác giả đã thực hiện và nâng cấp kỹ thuật này từ nhiều bài báo trong quá khứ. Việc này có thể giúp người đọc có thể cải thiện được tư duy fuzzing.

- Các kiến thức trong bài báo cung cấp thật sự thích hợp cho bất kỳ ai đang làm quen với fuzzing và tự động hóa trong fuzzing. Việc triển khai cũng được tác giả nêu chi tiết, việc thực hiện lại thử nghiệm là khá đơn giản.

- Người đọc có thể rút được kinh nghiệm từ những thiếu sót trong kỹ thuật của tác giả.

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:

D. Hoàn thành tốt yêu cầu được giao.

E. Có nội dung mở rộng, ứng dụng.

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

HẾT