Trường Đại học Công nghệ Thông tin – ĐHQG Tp. HCM
Khoa Mạng máy tính và truyền thông

BÁO CÁO TIẾN ĐỘ ĐỒ ÁN MÔN HỌC

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

Tên chủ đề: Binary-level Directed Fuzzing for Use-After-Free Vulnerabilities

Mã nhóm: Nhóm 03 Mã đề tài: CK14

Lớp: NT521.O12.ATCL

1. THÔNG TIN THÀNH VIÊN NHÓM:

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

STT	Họ và tên	MSSV	Email
56	Nguyễn Ngọc Trà My	21520353	21520353@gm.uit.edu.vn
13	Lê Hoàng Oanh	21521253	21521253@gm.uit.edu.vn
47	Huỳnh Minh Tân Tiến	21521520	21521520@gm.uit.edu.vn
28	Bùi Hoàng Trúc Anh	21521817	21521817@gm.uit.edu.vn

2. NỘI DUNG THỰC HIỆN:

2.1. 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: ……..

2.2. Tên bài báo tham khảo chính:

M.-D. Nguyen, S. Bardin, R. Bonichon, R. Groz, and M. Lemerre, “Binary-level Directed Fuzzing for Use-After-Free Vulnerabilities.” Accessed: Oct. 29, 2023. [Online]. Available: https://www.usenix.org/system/files/raid20-nguyen.pdf

2.3. Dịch tên Tiếng Việt cho bài báo:

Kiểm thử Fuzzing có định hướng ở mức độ nhị phân cho các lỗ hổng Use-After-Free

2.4. Tóm tắt nội dung chính:

UAFuzz là một DGF được tùy chỉnh để tập trung tìm kiếm lỗi ở mức độ nhị phân liên quan đến UAF (use after free), lỗ hổng liên quan đến việc các phần tử heap được sử dụng sau khi giải phóng, gây ra nguy cơ đối với dữ liệu.

UAFuzz phát hiện ra các UAF bằng cách tạo và chọn lựa các đầu vào có các sự kiện cấp phát, sử dụng và giải phóng, để làm cho chương trình lỗi và ghi nhận lại các bug traces.

Phương pháp mà UAFuzz tiếp cận: đề xuất ra các số liệu để đánh giá độ phát hiện ra UAF; tạo ra các thuật toán để đánh giá và lựa chọn input: seed selection, power schedule, cut-ede coverage metric, energy assignment,… và sử dụng các số liệu có sẵn để xử lý và phân loại lỗi.

Một số thuật toán mà UAF sử dụng: thuật toán Seed Selection, thuật toán tính toán seed distance, thuật toán tính toán độ bao phủ của seed, thuật toán xác định cạnh cắt và cạnh không cắt nhằm mục tiêu cải thiện hiệu suất và kết quả fuzzing.

Triển khai UAFuzz: Đầu vào của hệ thống tổng thể bao gồm một tập các "seed" ban đầu, mã nguồn mục tiêu tại dạng nhị phân và các vị trí mục tiêu được trích xuất từ dấu vết lỗi. Đầu ra là một tập các đầu vào gây ra lỗi duy nhất. Bản nguyên mẫu được xây dựng trên cơ sở của AFL 2.52b và QEMU 2.10.0 để thực hiện fuzzing, và nền tảng phân tích nhị phân BINSEC để thực hiện phân tích tĩnh (nhẹ).

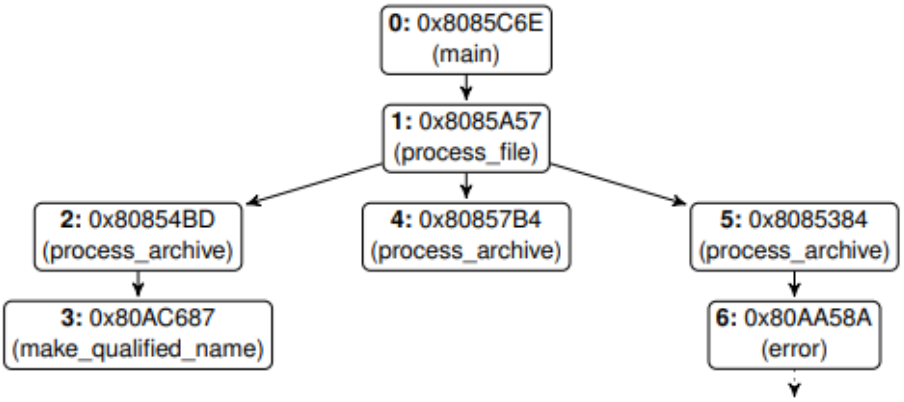
Đánh giá UAFuzz: đánh giá dựa trên 4 câu hỏi: khả năng tái tạo lỗi, chi phí, phân loại UAF, và những đóng góp của công cụ để thấy được sự vượt trội của UAFuzz so với các fuzzer khác như Hawkeye, AFLGo,...

Kết luận: UAFuzz là một DGB có hiệu năng vượt trội về việc tìm kiếm các lỗ hổng UAF so với các fuzzer đối thủ

2.5. Các kỹ thuật chính được mô tả sử dụng trong bài báo:

Bài báo sử dụng kỹ thuật cho 3 giai đoạn chính trong phương pháp đề xuất, bao gồm:

- Kỹ thuật 01: Bug Trace Flattening



chuyển đổi các thông tin từ bug trace thành một định dạng dễ dàng tiếp cận và phân tích hơn.

- Kỹ thuật 02: Seed Selection based on Target Similarity được sử dụng trong việc chọn lọc seeds(inputs) trong quá trình fuzzing. Chiến lược này tập trung vào việc ưu tiên seeds mà tương tự với các mục tiêu mà chúng tạo ra.

Algorithm 2: IS_FAVORED

Input : A seed s
Output : *true* if s is favored, otherwise *false*

```
1 global t_max = 0;          ▷ maximum target similar metric score
2 if t(s) ≥ t_max then t_max = t(s); return true;        ▷ update t_max
3 else if new_cov(s) then return true;          ▷ increase coverage
4 else return false;
```

Thuật toán trong việc chọn seeds ưu tiên

- Kỹ thuật 03: Target Similarity Metrics được sử dụng để đánh giá mức độ tương tự giữa một đầu vào (input) và mục tiêu cụ thể (target) trong quá trình kiểm thử hướng dẫn (directed fuzzing).

Dưới đây là 4 loại đo lường tương tự và cách chúng hoạt động:

Target Prefix (tP(s,T)): Đo lường này xác định các vị trí trong dấu vết mục tiêu T mà hạt giống 's' bao phủ tuần tự cho đến khi có sự phân kỳ đầu tiên.

UAF Prefix (t3T P(s,T)): Đo lường này chỉ xem xét các sự kiện UAF của dấu vết mục tiêu T và xác định liệu hạt giống 's' có bao phủ chúng tuần tự cho đến khi có sự phân kỳ đầu tiên hay không.

Target Bag (tB(s,T)): Đo lường này xác định các vị trí trong dấu vết mục tiêu T mà hạt giống 's' bao phủ bằng cách thực thi, mà không cần theo thứ tự nào cả.

UAF Bag (t3T B(s,T)): Đo lường này chỉ xem xét các sự kiện UAF của dấu vết mục tiêu T và xác định liệu hạt giống 's' có bao phủ chúng hay không, mà không cần theo thứ tự nào cả.

- Kỹ thuật 04: Combining Target Similarity Metrics: sử dụng một phương pháp hoặc thuật toán để kết hợp các độ đo tương tự khác nhau

Ví dụ, P-3TP-B metric được định nghĩa như sau:

$$t_{P-3TP-B}(s,T) \triangleq \langle t_P(s,T), t_{3TP}(s,T), t_B(s,T) \rangle$$

- Kỹ thuật 05: UAF-based Seed Distance: là một phương pháp để đo đạc sự tương tự giữa việc thực thi của một hạt giống (seed) và dấu vết lỗi mục tiêu UAF (Use-After-Free).
- Kỹ thuật 06: là một thuật toán quyết định cách phân bổ tài nguyên (ví dụ: số lượng hạt giống được tạo ra từ một hạt giống hiện tại) để tăng cường khả năng tìm kiếm các lỗi UAF.
- Kỹ thuật 07: Cut-edge Coverage Metric là một phương pháp đo lường sự bao phủ của các nhánh cần đi qua để đạt được mục tiêu trong việc tìm kiếm lỗi sử dụng sau khi giải phóng (UAF). Đây là một yếu tố trong lịch trình sức mạnh của công cụ dò tìm lỗi UAF. Cut-edge Coverage Metric đóng vai trò quan trọng trong việc quyết định số lượng năng lượng được gán cho mỗi hạt giống, nhằm tăng cường khả năng tìm kiếm lỗi UAF.

Algorithm 3: Accumulating cut edges

Input : Program P; dynamic calling tree T of a bug trace
Output : Set of cut edges E_{cut}

```
1 E_cut ← ∅;
2 nodes ← flatten(T);
3 for n ∈ nodes ∧ pn the node before n in T do
4     if n.func == pn.func then
5         ce ← calculate_cut_edges(n.func, pn.bb, n.bb);
6     else if pn is a call to n.func then
7         ce ← calculate_cut_edges(n.func, n.func.entry_bb, n.bb);
8     E_cut ← E_cut ∪ ce;
9 return E_cut;
```

Algorithm 4: calculate_cut_edges inside a function

Input : A function f; Two basic blocks bb_{source} and bb_{sink} in f
Output : Set of cut edges ce

```
1 ce ← ∅;
2 cfg ← get_CFG(f);
3 decision_nodes ← {dn : ∃ a path bb_source →* dn →* bb_sink in cfg}
4 for dn ∈ decision_nodes do
5     outgoing_edges ← get_outgoing_edges(cfg, dn);
6     for edge ∈ outgoing_edges do
7         if reachable(cfg, edge, bb_sink) then
8             ce ← ce ∪ {edge};
9 return ce;
```

- Kỹ thuật 08: Energy Assignment là quá trình xác định mức độ "năng lượng" (hoặc tài nguyên) được gán cho các hạt giống (input) được chọn để tạo ra các đầu vào mới trong quá trình kiểm tra và tìm lỗi trong phần mềm.

2.6. Môi trường thực nghiệm của bài báo:

- Cấu hình máy tính: Intel Xeon CPU E3-1505M v6 @ 3.00GHz CPU with 32GB RAM and Ubuntu 16.04 64-bit.
- Các công cụ hỗ trợ sẵn có:

IDA Pro v6.9 and v7.6 (32-bit).

Graph-Easy v0.7.6 for converting IDA's call graph into dot format.

The profiling tool Valgrind.

The binary analysis framework BINSEC.

Coverage-guided greybox fuzzer AFL v2.52b in QEMU mode.

- Ngôn ngữ lập trình để hiện thực phương pháp: python v2.7
- 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 11 chương trình C trong thế giới thực có kích thước thay đổi từ 26 Kb đến 3,8 Mb. Các chương trình được chọn bao gồm từ xử lý hình ảnh đến lưu trữ dữ liệu, xử lý video và phát triển web.
- Tiêu chí đánh giá tính hiệu quả của phương pháp:

Số lượng lỗ hổng được phát hiện: 13

Table 3: Overview of our evaluation benchmark

Bug ID	Program		Bug		#Targets in trace
	Project	Size	Type	Crash	
giflib-bug-74	GIFLIB	59 Kb	DF	✗	7
CVE-2018-11496	lrzip	581 Kb	UAF	✗	12
yasm-issue-91	yasm	1.4 Mb	UAF	✗	19
CVE-2016-4487	Binutils	3.8 Mb	UAF	✓	7
CVE-2018-11416	jpegoptim	62 Kb	DF	✗	5
mjs-issue-78	mjs	255 Kb	UAF	✗	19
mjs-issue-73	mjs	254 Kb	UAF	✗	28
CVE-2018-10685	lrzip	576 Kb	UAF	✗	7
CVE-2019-6455	Recutils	604 Kb	DF	✗	15
CVE-2017-10686	NASM	1.8 Mb	UAF	✓	10
gifsicle-issue-122	Gifsicle	374 Kb	DF	✗	11
CVE-2016-3189	bzip2	26 Kb	UAF	✓	5
CVE-2016-20623	Binutils	1.0 Mb	UAF	✗	7

2.7. Kết quả thực nghiệm của bài báo:

UAFuzz cũng được chứng minh là có hiệu quả trong thử nghiệm bản vá, dẫn đến việc phát hiện 30 lỗi chưa xác định (11 UAFS, 7 CVES) trong các dự án như Perl, GPAC, MuPDF và GNU Patch (bao gồm 4 bản vá lỗi). Cho đến nay đã có 17 lỗi đã được sửa chữa.

Ưu điểm của UAFuzz: do đã được tùy chỉnh cho phù hợp với việc theo dấu các uaf bug nên khả năng tìm kiếm, phát hiện, tái tạo lỗi trong nhiều ngữ cảnh bảo mật điều vượt trội, cùng với đó là chi phí tính toán thấp, thời gian tìm lỗi ngắn, phát hiện được nhiều lỗi liên quan đến UAF ở mức nhị phân.

Nhược điểm: do được tùy chỉnh cụ thể nó sẽ giảm khả năng tùy biến và đa dụng trong việc phát hiện các lổ hổng khác ngoài uaf, cùng với đó là việc cấu hình, hiện thực công cụ cũng sẽ trở nên phức tạp hơn.

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

Kế hoạch:

1. Tìm hiểu lý thuyết về ngữ cảnh
2. Tìm hiểu về mô hình, cách thực nghiệm của đề tài
3. Cài đặt theo đường link github trong bài báo để xây dựng công cụ demo

Đã thực hiện được:

1. Xây dựng môi trường để chạy UAFuzz
2. Tìm hiểu về mô hình, cách thực nghiệm của đề tài

3. Cài đặt được môi trường để xây dựng công cụ demo

4. Cài đặt được các phần mềm yêu cầu để xây dựng công cụ demo

```
ubuntu@ubuntu-virtual-machine:~$ python --version
Python 2.7.18
```

```
ubuntu@ubuntu-virtual-machine:~$ valgrind --version
valgrind-3.18.1
```

```
ubuntu@ubuntu-virtual-machine:~$ graph-easy --version
Graph::Easy v0.76  (c) by Tels 2004-2008.  Released under the GPL 2.0 or later.

Running under Perl v5.034000.
```

```
penguin@penguin-ubuntu:~/Desktop$ sudo apt install ocaml ocaml-native-compilers camlp4-extra opam emacs llvm-15-dev pkg-config protobuf-compiler libgmp-dev libzmq3-dev cmake valgrin
d
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Note, selecting 'ocaml' instead of 'ocaml-native-compilers'
Note, selecting 'camlp4' instead of 'camlp4-extra'
ocaml is already the newest version (4.13.1-3ubuntu1).
ocaml set to manually installed.
opam is already the newest version (2.1.2-1).
llvm-15-dev is already the newest version (1:15.0.7-0ubuntu0.22.04.3).
The following additional packages will be installed:
  cmake-data comerr-dev dh-elpa-helper emacs-bin-common emacs-common emacs-el emacs-gtk krb5-multidev libbsd-dev libc6-i386 libcamlp4-ocaml-dev libfindlib-ocaml
  libfindlib-ocaml-dev libgmpxx4ldbl libgssrpc4 libjsoncpp25 libkadm5clnt-mit12 libkadm5srv-mit12 libkdb5-10 libkrb5-dev libm17n-0 libmd-dev libnorm-dev libotf1 libpgm-dev
  libprotobuf-dev libprotobuf-lite23 libprotoc23 librhash0 libsodium-dev m17n-db ocaml-findlib zlib1g-dev
Suggested packages:
  cmake-doc ninja-build cmake-format doc-base mailutils emacs-common-non-dfsg ncurses-term krb5-doc gmp-doc libgmp10-doc libmpfr-dev krb5-user m17n-docs libnorm-doc gawk
  protobuf-mode-el valgrind-dbg valgrind-mpi kcachegrind alleyoop valkyrie
The following NEW packages will be installed:
  camlp4 cmake cmake-data comerr-dev dh-elpa-helper emacs emacs-bin-common emacs-common emacs-el emacs-gtk krb5-multidev libbsd-dev libc6-i386 libcamlp4-ocaml-dev libfindlib-ocaml
  libfindlib-ocaml-dev libgmp-dev libgmpxx4ldbl libgssrpc4 libjsoncpp25 libkadm5clnt-mit12 libkadm5srv-mit12 libkdb5-10 libkrb5-dev libm17n-0 libmd-dev libnorm-dev libotf1
  libpgm-dev libprotobuf-dev libprotobuf-lite23 libprotoc23 librhash0 libsodium-dev libzmq3-dev m17n-db ocaml-findlib pkg-config protobuf-compiler valgrind zlib1g-dev
0 upgraded, 41 newly installed, 0 to remove and 76 not upgraded.
Need to get 107 MB of archives.
After this operation, 459 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://vn.archive.ubuntu.com/ubuntu jammy/universe amd64 libcamlp4-ocaml-dev amd64 4.13+1-1build1 [28,7 MB]
Get:2 http://vn.archive.ubuntu.com/ubuntu jammy/universe amd64 camlp4 amd64 4.13+1-1build1 [12,5 MB]
Show Applications archive.ubuntu.com/ubuntu jammy/main amd64 libjsoncpp25 amd64 1.9.5-3 [80,0 kB]
Get:4 http://vn.archive.ubuntu.com/ubuntu jammy/main amd64 librhash0 amd64 1.4.2-1ubuntu1 [125 kB]
```

Còn lại:

Xây dựng hoàn thiện công cụ demo

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

- Một số phiên bản của các phần mềm được dùng trong demo của bài báo đã cũ không thể sử dụng.

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

50%

4. NHẬT KÝ PHÂN CÔNG NHIỆM VỤ:

STT	Công việc	Phân công nhiệm vụ
1	Tìm hiểu lý thuyết ngữ cảnh demo	Bùi Hoàng Trúc Anh
2	Tìm hiểu các kỹ thuật demo	Nguyễn Ngọc Trà My
3	Tìm hiểu và cài đặt chương trình demo	Lê Hoàng Oanh, Huỳnh Minh Tân Tiến

YÊU CẦU CHUNG

Báo cáo:

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

  Ví dụ: [NT521.O12.ATCL]-MidTerm_Nhom02_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