

BÁO CÁO TỔNG KẾT ĐỒ ÁN MÔN HỌC

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

Tên chủ đề: **HAEPG: An Automatic Multi-hop Exploitation Generation Framework**

Mã nhóm: 10 Mã đề tài: CK04

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	Nguyễn Hữu Minh Sang	20520921	20520921@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:

Zhao, Z., Wang, Y., Gong, X. (2020). HAEPG: An Automatic Multi-hop Exploitation Generation Framework. In: Maurice, C., Bilge, L., Stringhini, G., Neves, N. (eds) Detection of Intrusions and Malware, and Vulnerability Assessment. DIMVA 2020. Lecture Notes in Computer Science(), vol 12223. Springer, Cham.

C. Dịch tên Tiếng Việt cho bài báo:

HAEPG: Bộ khung phát triển và tự động hoá việc khai thác lỗ hổng Multi-hop

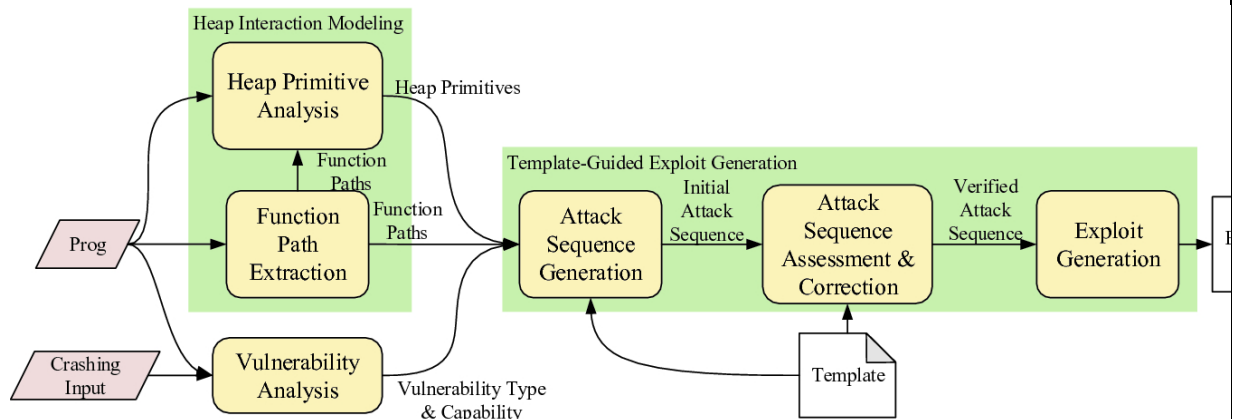
¹ Ghi nội dung tương ứng theo mô tả

D. Tóm tắt nội dung chính:

Ở bài báo này, nhóm nghiên cứu đề xuất một giải pháp có tên gọi là: "automatic exploit generation solution HAEPG" cho Heap vulnerabilities. Đây là một giải pháp công nghệ kết hợp hai yếu tố, giữa việc phân tích lỗi hỏng heap hay các vấn đề khai thác heap trong các ứng dụng cùng với khai thác các lỗi hỏng dạng multi-hop trong các ứng dụng. Ở đây, nhóm nghiên cứu đã đề xuất rằng: HAEPG đã có thể thực hiện một việc khai thác lỗi hỏng phức tạp trong lỗi hỏng của HEAP và khiến nó trở lên nguy hiểm hơn theo từng cấp độ một. Điều mà HAEPG làm được là hoàn toàn tự động, thay thế các công việc mà trước đây cần thực hiện thủ công rất nhiều công sức. Dù rằng bài báo chỉ thử nghiệm trên môi trường của các bài thi CTF nhưng kết quả cho ra rất đáng mong đợi. Và nhóm nghiên cứu cũng chỉ ra rằng: "we believe that HAEPG improves the state-of-the-art of automated exploit generation and provides useful building blocks for solving remaining challenges in the field." hay dịch ra là HAEPG sẽ là một công cụ hỗ trợ trong mảng khai thác lỗi hỏng một cách tự động và nhanh chóng

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

Các kỹ thuật trong HAEPG được chia nhỏ ra thành các phương pháp tung ứng



a) Heap Interaction Modeling (Mô hình tác động/ tương tác với HEAP)

- **Function Path Extraction:** Những phần code chức năng trong chương trình sẽ được phân tích. Sau đó các phần đó sẽ được xem là các block và chia ra làm 2 loại: **Atomicity** và **Reentrant**. Sau đó sẽ có các Flow-graph (sơ đồ cơ chế) để có thể tìm kiếm cách giải và đưa ra giải pháp

- **Heap Primitive Analysis:** Những phần bộ nhớ HEAP nguyên bản của chương trình sẽ được phân ra làm 3 loại: Allocation, Deallocation và Edit. Mỗi loại sẽ được HAEPG tìm và đưa vào các symbolic execution (bản mẫu thực hiện) để thực hiện, call API tương ứng hoặc là trích xuất giá trị

Allocation

size: the size of allocation
addr: the address returned by the heap allocator

Deallocation

addr: the address to be released

Edit

base: the base of edit address
offset: the offset of edit address
data: the data to be written

b) Vulnerability Analysis (Phân tích lỗ hổng)

- Những phần thực hiện bị thất bại hoặc các giá trị input (đầu vào) để giải các challenge CTF sẽ được lưu trữ lại, có một bảng các giá trị tương ứng, nếu có trường hợp sau này lập lại tương tự sẽ giảm thời gian thực hiện hoặc giải chương trình đó nhanh hơn

Vulnerability type	Trigger operation	Violation rule
Double free	Free a heap chunk	<code>mtag.status == free or ptag.status == free</code>
Use After Free	Store <i>n</i> bytes of data in memory address	<code>mtag.status == free or ptag.status == free</code>
Overflow	[<i>base</i> + <i>off</i>]	<code>n + off > ptag.size</code>
Poison Null Byte	(<i>base</i> and <i>off</i> are the base and offset of addressing)	<code>n + off == ptag.size + 1</code> and the last byte of data is null byte
Off by One		<code>n + off == ptag.size + 1</code>

c) Template-Guided Exploit Generation (Mẫu - hướng dẫn để hình thành các khai thác)

- Templates: khi mà các chương trình chạy sẽ luôn có các phần khác nhau, do đó sẽ thu thập những phần giống nhau, hình thành các phần cần thiết để dần về sau hình thành một cái hoàn chỉnh nhất và áp dụng được cho tất cả. Có 3 components (các phần) chung được phát triển: *Backbone Primitives Sequence*, *Layout Constraints* và *Requirements*. Các thành phần này sẽ giúp

xây dựng hệ thống khai thác multi-hop nhanh hơn, an toàn hơn và có thể tái sử dụng nhiều lần

- **Attack Sequence Generation:** Sẽ xây dựng và kiểm thử các templates đã phát triển trước đó, nếu không sử dụng được sẽ thay bằng một cái templates khác. Cơ chế sử dụng sẽ được hình thành và tạo ra 2 cách: **Heap Simulator** và **Symbolic Execution**
- **Attack Sequence Assessment and Correction:** Cũng sẽ là xây dựng và kiểm thử, nhưng giờ đây thay vì sử dụng một cái templates hoàn toàn mới, các templates này sẽ được sử dụng và sử lại cho phù hợp hơn, hoàn toàn tự động. Khi này sẽ có thêm phần S2E(?) để đảm bảo chính xác
- **Exploit Generation:** Khi này, các thành phần cần thiết đã đủ, HAEPG sẽ bắt đầu thực hiện việc Exploit Generation. Dựa vào Templates đã xây dựng trước đó. Giờ đây HAEPG sẽ tạo ra các input đầu vào cần thiết để thực hiện khai thác. Có ba kiểu đầu vào phân ra gồm: **Arbitrary Execution (AX)**, **Arbitrary Write (AW)** và **Arbitrary Allocation (AA)**

F. Môi trường thực nghiệm của bài báo:

- Cấu hình máy tính: Intel(R) Xeon(R) Gold 6154 CPU @ 3.00GHz*24 and 512GB RAM
- Ngôn ngữ lập trình để hiện thực phương pháp: Python, C, C++, Java, Assembly, Golang,
- Đối tượng nghiên cứu (chương trình phần mềm dùng để kiểm tra tính khả thi của phương pháp/tập dữ liệu – nếu có): 24 giải CTF được tổ chức, kiểm tra thông qua ctftime.org, pwnable.tw và github.com. Các chương trình được lựa chọn theo tiêu chí: chương trình phải có ít nhất một lỗi về heap và các dạng lỗi hỏng đa dạng. Ưu tiên các chương trình có nhiều điểm. Có các templates được viết sẵn cho các dạng khai thác lỗi nổi bật như: Fastbin attack, unsafe unlink, house of force và tcache poisoning.
- Tiêu chí đánh giá tính hiệu quả của phương pháp: Thời gian mà hệ thống dành ra để tương tác với lỗi hỏng và thực hiện khai thác thành công lỗi hỏng đó, đơn vị tính thời gian là giây (s)

G. Kết quả thực nghiệm của bài báo:

Dựa vào việc đưa HAEPG vào 24 giải CTF, có 21 giải đã xác định được lỗ hổng và cách khai thác đến chiếm 87.5%, tạo ra shell thực thi cho 16 giải chiếm 66.7%. Ngoài ra còn có thể vượt qua có chế bảo mật NX và Full RELRO

So với các kỹ thuật Revery và Mechanical Phish:

- Với Revery: các vấn đề với unsafe link sẽ không thực hiện, trong khi kỹ thuật HAEPG có thể, không thể khai thác được bởi vì khó fuzzing được vấn đề bộ nhớ state, điều mà các phần dữ liệu giả có thể gây khó khăn
- Với Mechanical Phish: Tạo ra các trường hợp lỗi với Driller, không thể khai thác lỗi vì không thể tạo ra các con trỏ bị lỗi tương ứng, bên cạnh đó cũng không điều khiển được con trỏ điều hướng hay là thực hiện shellcode hoặc rop-chains

Còn một số vấn đề bất cập:

- Trong một số bài CTF cụ thể, HAEPG không thể thực hiện khai thác các lỗi hổng dù rằng đó là điều nằm trong khả năng điều khiển của chương trình
- Chỉ có thể khai thác lỗi với các templates đã có sẵn, không thể thực hiện với các kỹ thuật mà chương trình không biết hoặc chương trình có sự thay đổi quá khác biệt so với thiết lập ban đầu
- Một số thư viện được sử dụng không phù hợp cho việc giải các bài tương ứng
- Phân tích lỗi sai dẫn đến đưa ra cách hướng giải quyết sai hoặc mất quá nhiều thời gian
- Các chương trình quá phức tạp hoặc bị nhiễu quá nhiều, chương trình HAEPG không thể thực hiện hoặc kiểm soát hết dẫn đến đưa ra kết quả sai hoặc không thể đưa ra được kết quả phân tích
- Chưa thể phân tích các chương trình thực tế, chỉ có thể với các chương trình đơn giản với các lỗi đã có thể xác định được thông qua các template có sẵn

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

Bởi vì tính chất của bài báo – nghiên cứu nằm ở phạm vi rất lớn, mục tiêu chính là đi đến hoàn toàn đến việc tự động hoá. Nên em đã không thể thực hiện hoá hết được. Thay vào đó em đã có một chương trình kiểm tra và đưa ra phân tích cho một lỗi nhất định và từng bài một

Kiểm tra và thực hiện lại nghiên cứu theo một phần, tìm hiểu hơn là tại sao có thể làm được và đưa ra hướng giải quyết cho sau này

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

Tìm ra demo cụ thể và rõ ràng nhất

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

90%

4. NHẬT KÝ PHÂN CÔNG NHIỆM VỤ:

STT	Công việc	Phân công nhiệm vụ
1	Làm đồ án	Nguyễn Hữu Minh Sang

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.

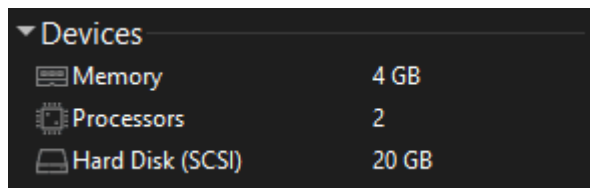
Qui định: Mô tả các bước thực hiện/ Phương pháp thực hiện/Nội dung tìm hiểu (Ảnh chụp màn hình, số liệu thống kê trong bảng biểu, có giải thích)

A. Phương pháp thực hiện

- Vì bài nghiên cứu thực tế làm ở môi trường máy tính hiện đại và có nhiều tai nguyên giúp tăng tốc nhanh quá trình kiểm tra và kiểm thử
- Hiện tại em chỉ có thể thực hiện được một phần nhỏ của bài nghiên cứu là mảng cho phân tích và đưa ra kết quả chạy chương trình
 - o Chương trình kiểm tra lỗi Heap Overflow có tồn tại trong chương trình
- Việc xây dựng và kiểm tra xem liệu có xác định đúng và cách thức khai thác được lỗi hay không

B. Chi tiết cài đặt, hiện thực

- Thực hiện trên máy có cấu hình: máy ảo chạy Ubuntu 18.04, 4gb ram, 2 luồng xử lý (i7-10750H)



▼ Devices	
Memory	4 GB
Processors	2
Hard Disk (SCSI)	20 GB

- Chạy môi trường ảo trên máy tính

```
sudo apt-get install virtualenv
virtualenv -p /usr/bin/python3 env
source env/bin/activate
```
- Lấy file source code tại github cho chương trình

```
git clone https://github.com/SoftwareSecurityLab/Heap-Overflow-Detection
cd Heap-Overflow-Detection
```
- Cài đặt các gói yêu cầu

```
pip install -r requirements.txt
```
- Bắt đầu thực hiện kiểm tra và xác định lỗi

```
(env) ubuntu@ubuntu:~$ ls
Desktop  Downloads  Heap-Overflow-Detection  Pictures  Templates
Documents  env        Music                  Public    Videos
(env) ubuntu@ubuntu:~$ cd he
bash: cd: he: No such file or directory
(env) ubuntu@ubuntu:~$ cd Heap-Overflow-Detection/
(env) ubuntu@ubuntu:~/Heap-Overflow-Detection$ pip install -r requirements.txt

Collecting ailment==9.0.9166
  Downloading ailment-9.0.9166-py3-none-any.whl (18 kB)
Collecting angr==9.0.9166
  Downloading angr-9.0.9166-py3-none-manylinux1_x86_64.whl (2.3 MB)
```

C. Kết quả thực nghiệm

- Chương trình kiểm tra lỗi là một chương trình nhỏ với lỗi ta đã xác định được là Heap Overflow, mục đích chính là kiểm soát được đến phần Heap có thể khai thác, cách tiếp cận và tiếp cận thế nào
- Xác định được lỗi trong chương trình, xác định và chỉ ra cách giải quyết và cách thực hiện khai thác

```
(env) ubuntu@ubuntu:~/Heap-Overflow-Detection/Project_Files$ ./run.py -b program
-----
\Oops, You Didn't Specify The Unit Prototype. Use -p Option to Set The Desired Unit Prototype.
-|Critical Units Are :
-----|authentication
-----|You Can Reach It Through These Chains :
-----|main → authentication
-----|signup
-----|You Can Reach It Through These Chains :
-----|main → signup
-----|check
-----|You Can Reach It Through These Chains :
-----|main → authentication → signin → check
(env) ubuntu@ubuntu:~/Heap-Overflow-Detection/Project_Files$
```


[illegible]

D. Hướng phát triển

- Trong tương lai mục đích là tạo ra chương trình có thể tự động hoà hoàn toàn và giải quyết được hết các vấn đề đang gặp hiện tại của các bài chưa giải được và các bài giải còn chưa tối ưu
- Mở rộng hướng giải quyết không chỉ trong các bài CTF mà trong ngành nghề thực tế hoặc các chương trình phức tạp thật sự

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