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: BofAEG: Automated Stack Buffer Overflow Vulnerability Detection and Exploit Generation Based on Symbolic Execution and Dynamic Analysis

*Mã nhóm: **4bn0rm4l**…… Mã đề tài: **CK07**……*

Lớp: NT521.O11.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 Huy Cường	21520667	21520667@gm.uit.edu.vn
2	Phan Gia Khánh	21522213	21522213@gm.uit.edu.vn
3	Trần Minh Duy	21522010	21522010@gm.uit.edu.vn
4	Nguyễn Đức Tài	21521395	21521395@gm.uit.edu.vn
5	Nguyễn Thái Bình	21521877	21521877@gm.uit.edu.vn

2. NỘI DUNG THỰC HIỆN:[1]

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:

L. Coppolino, S. Xu, and Y. Wang, "BofAEG: Automated Stack Buffer Overflow Vulnerability Detection and Exploit Generation Based on Symbolic Execution and Dynamic Analysis," Security and Communication Networks, vol. 2022, article ID 1251987, Jun. 2022. [Online]. Available: https://doi.org/10.1155/2022/1251987.

2.3. Dịch tên Tiếng Việt cho bài báo:

[1] Ghi nội dung tương ứng theo mô tả

BofAEG: Tự động phát hiện lỗ hổng Stack Buffer Overflow và tạo mã khai thác dựa trên Symbolic Execution và Dynamic Analysis

2.4. Tóm tắt nội dung chính:

Bài báo giới thiệu một phương pháp giải quyết vấn đề lỗ hổng **stack buffer overflow** trong phần mềm. Lỗ hổng này có thể **ghi đè địa chỉ trả về của hàm** và **chiếm quyền điều khiển chương trình**, gây ra những vấn đề nghiêm trọng cho hệ thống. Các phương pháp tự động tạo mã khai thác hiện tại không thể vượt qua các biện pháp ngăn chặn lỗ hổng PIE (Position-Independent Executable) và không thể xử lý tình huống khi hàm xuất ra tiêu chuẩn không được sử dụng trong chương trình.

Trong bài báo, tác giả đề xuất giải pháp **BofAEG**, dựa trên kỹ thuật **symbolic execution** và **dynamic analysis** để phát hiện tự động lỗ hổng stack buffer overflow và tạo mã khai thác. **BofAEG** được thử nghiệm trên các chương trình Capture the Flag (CTF) và các chương trình liên quan đến lỗ hổng phổ biến và đã được phổ biến (CVE). Kết quả thử nghiệm cho thấy **BofAEG** không chỉ có thể phát hiện và tạo mã khai thác một cách hiệu quả mà còn triển khai nhiều kỹ thuật khai thác hơn và nhanh hơn so với các phương pháp tự động tạo mã khai thác hiện có.

Bài báo cũng trình bày sáu kỹ thuật khai thác cho lỗ hổng stack buffer overflow và chứng minh rằng lỗ hổng này **gây ra hậu quả nghiêm trọng** và **có thể vượt qua** hầu hết các biện pháp ngăn chặn khai thác hiện có. Bên cạnh đó, bài báo trình bày các thí nghiệm so sánh giữa BofAEG và các giải pháp tồn tại. Thử nghiệm cho thấy BofAEG có thể phát hiện và tạo khai thác các lỗ hổng an toàn trong hầu hết trường hợp, bao gồm cả trường hợp chương trình được bảo vệ bởi PIE. Tuy nhiên, BofAEG vẫn còn hạn chế trong trường hợp lỗ hổng phức tạp hoặc đa loại. Nó cần phát triển hơn nữa để khắc phục các hạn chế này.

Đóng góp của bài báo bao gồm: (1) Nghiên cứu và tóm tắt đặc điểm và khó khăn của việc tạo mã khai thác tự động cho lỗ hổng **stack buffer overflow** và đề xuất giải pháp **BofAEG**. (2) Thực hiện thử nghiệm so sánh giữa **BofAEG** và các phương pháp tạo mã khai thác tự động hiện có. (3) Triển khai sáu kỹ thuật khai thác cho lỗ hổng stack buffer overflow và chứng minh tính nguy hiểm cao của lỗ hổng này và khả năng vượt qua hầu hết các biện pháp ngăn chặn khai thác hiện có.

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

Trong bài báo này, các tác giả sử dụng hai kỹ thuật chính để thực hiện nghiên cứu của họ: symbolic execution và dynamic analysis

1. **Symbolic Execution:** Kỹ thuật symbolic execution được sử dụng để phân tích chương trình và thu thập thông tin về các biểu thức biểu tượng trong chương trình. Thay vì sử dụng các giá trị cụ thể cho đầu vào, symbolic execution sử dụng các biểu tượng để đại diện cho các giá trị chưa biết. Kỹ thuật này cho phép tạo ra các ràng buộc (path constraints) dựa trên các biểu thức biểu tượng và từ đó phân tích các tuyến đường thực thi khác nhau của chương trình. Trong nghiên cứu này, symbolic Execution được sử dụng để xác định các điểm yếu trong chương trình liên quan đến lỗ hổng tràn bộ nhớ đệm ngăn xếp.

2. **Dynamic analysis**: Kỹ thuật Dynamic analysis được sử dụng để theo dõi và ghi lại các hoạt động thực thi của chương trình trong thời gian chạy. Kỹ thuật này cho phép kiểm tra và ghi lại các giá trị của các biến và các tuyến đường thực thi trong quá trình chạy chương trình. Trong nghiên cứu này, Dynamic analysis được sử dụng để giám sát hoạt động của chương trình và tạo ra các dữ liệu đầu vào đặc biệt để kích hoạt lỗ hổng tràn bộ nhớ đệm ngăn xếp và thu thập thông tin về việc chiếm quyền kiểm soát luồng chương trình.

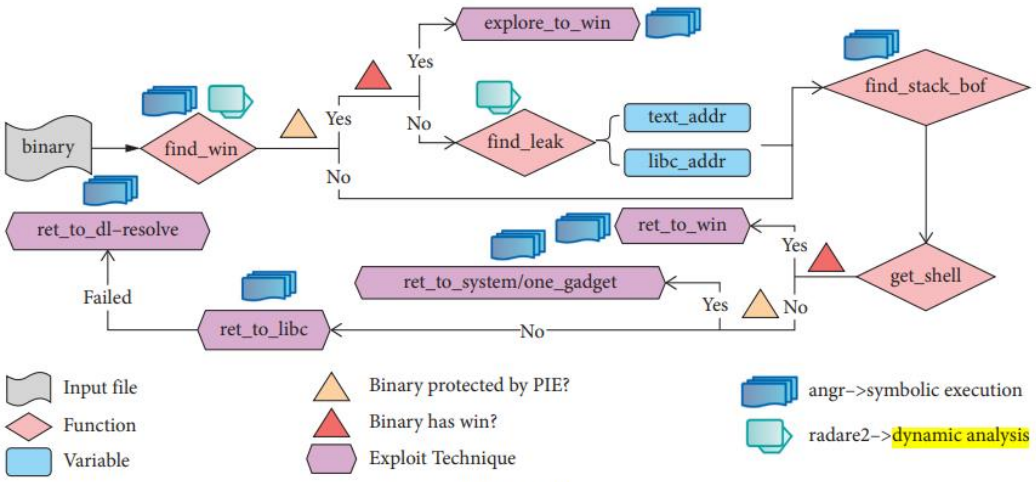

FIGURE 6: Method flow chart.

2.6. Môi trường thực nghiệm của bài báo:

Cấu hình máy tính: Ubuntu 18.04 64-bit với CPU Intel(R) Core(TM) i7-8700, RAM 16GB và phiên bản kernel 5.4.0.

Các công cụ hỗ trợ sẵn có: Công cụ thực thi symbolic được sử dụng là angr v9.1.11752, công cụ phân tích động được sử dụng là radare2 v5.6.4…

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): chương trình CTF (Capture the Flag) và CVE (Common Vulnerabilities and Exposures) có các lỗ hổng tràn bộ đệm stack để xem chương trình có được bảo vệ bởi PIE có chứa các hàm "win" hay không.

Tiêu chí đánh giá tính hiệu quả của phương pháp: so sánh với Zetatool dựa trên thời gian phát hiện lỗ hổng của các chương trình CTF, CVE; mức độ hỗ trợ hàm win (BofAEG hỗ trợ cả cuộc gọi hàm win cấp chức năng và cấp khối, trong khi Zeratool chỉ hỗ trợ cuộc gọi hàm win cấp chức năng.); kỹ thuật khai thác.

2.7. Kết quả thực nghiệm của bài báo:

Kết quả thực nghiệm cho thấy phương pháp BofAEG có hiệu suất cao và vượt trội so với các giải pháp hiện có. BofAEG không chỉ phát hiện được nhiều lỗ hổng tràn bộ nhớ

đệm ngăn xếp mà còn tạo ra khai thác một cách hiệu quả. Nó cũng thực hiện được nhiều kỹ thuật khai thác hơn và nhanh hơn.

Ưu điểm:

+ Khả năng tự động phát hiện và tạo ra exploit cho lỗ hổng stack buffer overflow một cách hiệu quả.

+ Xử lý được nhiều tình huống khác nhau và tạo ra exploit nhanh chóng.

+ Phương pháp kết hợp symbolic execution và dynamic analysis giúp đảm bảo độ chính xác và đáng tin cậy của việc phân tích chương trình.

Nhược điểm:

+ Giới hạn của symbolic execution: vấn đề path explosion và constraint solving. Khi đối mặt với các chương trình phức tạp, ví dụ như lexingtonctf2021_madlibs, các ràng buộc biểu tượng không thể được biểu diễn đúng cách, gây ra sự thất bại trong việc khám phá lỗ hổng tràn bộ đệm ngăn xếp trong cve-2004-1257_abc2mtex do vấn đề path explosion. Do đó, BofAEG gặp khó khăn cho việc phát hiện lỗ hổng tràn bộ nhớ đệm ngăn xếp đối với các chương trình CVE có độ phức tạp cao.

+ Giới hạn của một loại lỗ hổng duy nhất: Mặc dù lỗ hổng tràn bộ nhớ đệm ngăn xếp rất nguy hiểm, CANARY có thể hạn chế hiệu quả sự nguy hiểm của lỗ hổng này…

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

Kế hoạch của nhóm, các công việc mà nhóm dự định thực hiện:

1. Tìm hiểu, đọc bài báo và tài liệu liên quan để hiểu rõ nội dung, mục đích của bài báo; xác định cách triển khai mà bài báo đã làm; tài nguyên, môi trường thực hiện.
2. Tìm kiếm các chương trình CTF và CVE mà các tác giả sử dụng để triển khai.
3. Cài đặt Zeratool và BofAEG trên môi trường đã đề cập ở trên.
4. Sử dụng Zeratool và BofAEG để tìm kiếm các lỗ hổng CTF và CVE đã nêu trên, sau đó thống kê kết quả (có tìm kiếm được lỗ hổng hay không và nếu có thì mất bao nhiêu thời gian)
5. Soạn nội dung và làm slide để báo cáo cuối kỳ.
6. Viết báo cáo cuối kỳ.

Các công việc đã thực hiện và kết quả:

1. Tìm hiểu, đọc bài báo và tài liệu liên quan để hiểu rõ nội dung, mục đích của bài báo; xác định cách triển khai mà bài báo đã làm; tài nguyên, môi trường thực hiện.

   ➔ Kết quả: Tất cả các thành viên đã đọc và nắm rõ nội dung, mục đích, cách thực hiện, kết quả mà bài báo trình bày.

2. Tìm kiếm các chương trình CTF và CVE mà các tác giả sử dụng để triển khai.

   ➔ Kết quả: Đã tìm kiếm được các chương trình CTF và CVE. Kết quả tổng hợp: Tại đây

3. Cài đặt Zeratool và BofAEG trên môi trường đã đề cập ở trên

   ➔ Kết quả: Chưa cài đặt xong, còn gặp một số lỗi (đang trong quá trình sửa)

Các công việc cần thực hiện tiếp theo:

1. Sử dụng Zeratool và BofAEG để tìm kiếm các lỗ hỏng CTF và CVE đã nêu trên, sau đó thống kê kết quả (có tìm kiếm được lỗ hỏng hay không và nếu có thì mất bao nhiêu thời gian)
2. Soạn nội dung và làm slide để báo cáo cuối kỳ.
3. Viết báo cáo cuối kỳ.

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

1. Trong quá trình tìm kiếm các chương trình CTF và CVE, có một số chương trình chưa chắc chắn đã tìm đúng.
2. Chưa cài đặt được môi trường theo như mô tả của bài báo -> Cài đặt Ubuntu 18.04 64-bit trên cấu hình khác có sự tương đương (hoặc cao hơn)
3. Cài đặt Zeratool và BofAEG còn gặp một số lỗi.
4. Cần thời gian để hiểu rõ hơn về các kỹ thuật được sử dụng trong bài (symbolic execution & dynamic analysis) và cách áp dụng Zeratool và BofAEG vào các bài CTF & CVE.

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

Mức độ hoàn thành: 90%

5. NHẬT KÝ PHÂN CÔNG NHIỆM VỤ:

STT	Công việc	Phân công nhiệm vụ
1	Phân công công việc	Nguyễn Huy Cường
2	Đọc bài báo và viết báo cáo giữa kỳ	Cả nhóm
3	Tìm các chương trình CTF & CVE	Nguyễn Huy Cường
4	Cài đặt môi trường và thiết lập Zeratool và BofAEG	Cả nhóm
5	Tổng hợp nội dung lý thuyết	Phan Gia Khánh & Trần Minh Duy
6	Thực hiện các bài CTF bằng Zeratool và ghi nhận kết quả	Nguyễn Huy Cường & Nguyễn Thái Bình
7	Thực hiện các bài CTF bằng BofAEG và ghi nhận kết quả	Nguyễn Huy Cường & Nguyễn Thái Bình

8	Tổng hợp bài báo cáo, thiết kế slide và thuyết trình	Nguyễn Đức Tài

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.O11.ANTT]-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