THÔNG TIN CHUNG CỦA NHÓM

- Link YouTube video của báo cáo (tối đa 5 phút): *https://youtu.be/jnNq7FAIprA*
- Link slides: https://github.com/kimngannguyenhoang1306/CS2205.NOV2024
- *Mỗi thành viên của nhóm điền thông tin vào một dòng theo mẫu bên dưới*
- *Sau đó điền vào Đề cương nghiên cứu (tối đa 5 trang), rồi chọn Turn in*
- *Lớp Cao học, mỗi nhóm một thành viên*

- Họ và Tên: Nguyễn Hoàng Kim Ngân
- MSSV: 240202010

- Lớp: CS2205.NOV2024
- Tự đánh giá (điểm tổng kết môn): 9/10
- Số buổi vắng: 0
- Số câu hỏi QT cá nhân: 6
- Link Github: https://github.com/kimngannguyenhoang1306/CS2205.NOV2024

ĐỀ CƯƠNG NGHIÊN CỨU

TÊN ĐỀ TÀI (IN HOA)

PHÂN TÍCH TĨNH THEO NGỮ CẢNH: PHÁT HIỆN VÀ GIẢM CẢNH BÁO SAI TRONG MÃ NGUỒN DỄ BỊ TẤN CÔNG

TÊN ĐỀ TÀI TIẾNG ANH (IN HOA)

CONTEXT-AWARE STATIC ANALYSIS: DETECTION AND REDUCTION OF FALSE POSITIVES IN VULNERABLE SOURCE CODE

TÓM TẮT *(Tối đa 400 từ)*

Mã nguồn dễ bị tấn công (vulnerable source code) là một trong những nguyên nhân hàng đầu gây ra lỗ hổng bảo mật trong các ứng dụng phần mềm. Phân tích tĩnh là một công cụ mạnh mẽ để phát hiện các đoạn mã nguy hiểm trước khi chúng được triển khai, nhưng thường tạo ra nhiều cảnh báo sai do thiếu thông tin ngữ cảnh. Nghiên cứu này giới thiệu phương pháp phân tích tĩnh theo ngữ cảnh, tập trung vào việc hiểu và xử lý các đặc điểm của mã nguồn dễ bị tấn công thông qua việc tích hợp thông tin về ngữ cảnh chương trình, luồng điều khiển, và phụ thuộc dữ liệu. Mục tiêu là giảm tỷ lệ cảnh báo sai, tăng cường độ chính xác, và phát triển một khung phân tích hiệu quả, có khả năng mở rộng. Các kỹ thuật phân tích tĩnh theo ngữ cảnh đã được nghiên cứu để giảm thiểu cảnh báo sai trong phát hiện lỗ hổng phần mềm, đặc biệt là trong bối cảnh lập trình động [1], phân tích dữ liệu phụ thuộc và luồng điều khiển [2], và thực thi biểu tượng trong phân tích bảo mật [3].

GIỚI THIỆU *(Tối đa 1 trang A4)*

Mã nguồn dễ bị tấn công chứa các đoạn mã có thể dẫn đến các lỗ hổng bảo mật như tràn bộ đệm, lỗi con trỏ, và các lỗ hổng quản lý bộ nhớ khác. Những lỗi này thường xuất hiện do thiếu kiểm tra đầu vào, sử dụng không đúng cách các hàm hệ thống, hoặc quản lý tài nguyên không an toàn. Các công cụ phân tích tĩnh hiện tại thường phát hiện lỗ hổng bằng cách kiểm tra các mẫu mã (code patterns) hoặc quy tắc (rules). Tuy

nhiên, việc thiếu thông tin về ngữ cảnh chương trình (e.g., cách dữ liệu được truyền giữa các hàm, luồng thực thi của chương trình) khiến các công cụ này tạo ra nhiều cảnh báo sai. Điều này gây khó khăn cho nhà phát triển trong việc phân biệt giữa lỗ hổng thực sự và cảnh báo không cần thiết.

Phạm vi nghiên cứu:

- Ngôn ngữ lập trình:
    - Tập trung vào C/C++ vì chúng thường chứa mã nguồn dễ bị tấn công liên quan đến bộ nhớ.
    - Lỗ hổng bảo mật: Tràn bộ đệm (buffer overflow), tràn ngăn xếp (stack overflow), sử dụng con trỏ null (null pointer dereference).
- Phạm vi phân tích:
    - Cấp độ hàm: Phân tích các lỗi liên quan đến logic nội bộ của hàm.
    - Cấp độ mô-đun: Phân tích cách các hàm tương tác với nhau để phát hiện các lỗi trong toàn bộ chương trình.

MỤC TIÊU *(Viết trong vòng 3 mục tiêu)*

- Phát hiện mã nguồn dễ bị tấn công: Xây dựng phương pháp phân tích để phát hiện chính xác các đoạn mã nguy hiểm.
- Giảm cảnh báo sai: Tích hợp thông tin ngữ cảnh, luồng điều khiển và phụ thuộc dữ liệu để giảm tỷ lệ cảnh báo sai.
- Tăng độ chính xác: Phát triển cơ chế phân tích nhạy ngữ cảnh nhằm xác định lỗ hổng bảo mật trong các tình huống phức tạp.

NỘI DUNG VÀ PHƯƠNG PHÁP

1. Phân tích ngữ cảnh mã nguồn (Context-Aware Analysis)

- Mục tiêu: Hiểu rõ cách chương trình hoạt động và các đoạn mã dễ bị tấn công dựa trên luồng thực thi và ngữ cảnh sử dụng.
- Phương pháp:

    - Xây dựng đồ thị luồng điều khiển (Control Flow Graph - CFG) để hiểu cách các khối mã được thực thi.
    - Phân tích đồ thị gọi hàm (Call Graph Analysis) để xác định các điểm giao tiếp giữa các hàm.
    - Sử dụng phân tích ngữ cảnh (Context-Sensitive Analysis) để xác định cách các đoạn mã nguy hiểm được gọi trong các điều kiện khác nhau.

2. Phân tích phụ thuộc dữ liệu (Data Dependency Analysis)

- Mục tiêu: Phát hiện các lỗ hổng liên quan đến cách dữ liệu được truyền và sử dụng trong chương trình.
- Phương pháp:
    - Triển khai phân tích lan truyền taint (Taint Propagation Analysis) để theo dõi dữ liệu từ nguồn không tin cậy.
    - Phân tích alias (Alias Analysis) để xác định các con trỏ hoặc tham chiếu đến cùng một vùng bộ nhớ.
    - Sử dụng thực thi biểu tượng (Symbolic Execution) để mô phỏng các giá trị dữ liệu tại thời điểm chạy giả định.

3. Giảm cảnh báo sai (False Positive Reduction)

- Mục tiêu: Loại bỏ các cảnh báo không cần thiết để tập trung vào các lỗ hổng thực sự.
- Phương pháp:
    - Áp dụng thuật toán đối sánh mẫu (Pattern Matching) để xác định các lỗ hổng đã biết.
    - Sử dụng giải ràng buộc (Constraint Solving) để kiểm tra tính khả thi của các điều kiện dẫn đến lỗi.
    - Triển khai cơ chế lọc nâng cao (Advanced Filtering Mechanisms) để loại bỏ các cảnh báo không phù hợp.

4. Tích hợp và tối ưu hóa

- Mục tiêu: Xây dựng một khung phân tích tĩnh có thể mở rộng và dễ tích hợp.
- Phương pháp:

- Thiết kế kiến trúc phân tích mô-đun (Modular Analysis Framework).
- Sử dụng xử lý song song (Parallel Processing) để tăng tốc phân tích.
- Tối ưu hóa lưu trữ với cơ chế lưu trữ đệm (Caching Mechanisms).

5. Kiểm tra và đánh giá

- Mục tiêu: Đánh giá hiệu quả của phương pháp trên các đoạn mã nguồn dễ bị tấn công thực tế.
- Phương pháp:
  - Xây dựng bộ dữ liệu kiểm thử gồm các mã nguồn dễ bị tấn công (e.g., mã từ CVE - Common Vulnerabilities and Exposures).
  - Đo lường tỷ lệ cảnh báo sai, độ chính xác, và hiệu suất xử lý của công cụ phân tích.

KẾT QUẢ MONG ĐỢI

- *Khung phân tích:*
  - *Giảm >50% tỷ lệ cảnh báo sai so với các công cụ hiện tại.*
  - *Tăng >30% tốc độ phân tích.*
- *Tìm hiểu kỹ thuật:*
  - *Hiểu rõ tác động của ngữ cảnh lên phát hiện lỗ hổng.*
  - *Phát triển các chỉ số về tính khả thi của đường dẫn và độ nhạy ngữ cảnh.*
- *Sản phẩm nghiên cứu:*
  - *Công cụ phân tích tĩnh.*
  - *Bộ dữ liệu kiểm thử.*
  - *Tài liệu hướng dẫn và báo cáo nghiên cứu.*

TÀI LIỆU THAM KHẢO *(Định dạng DBLP)*

[1] A. Johnson, "Reducing False Positives in Static Analysis with Context-Sensitive Techniques," IEEE Transactions on Software Engineering, vol. 47, no. 12, pp. 1234-1245, 2021.

[2] M. S. e. al., "Data Dependency and Control Flow in Vulnerability Detection," ACM SIGSOFT Software Engineering Notes, vol. 45, no. 4, pp. 67-75, 2020.
[3] K. Lee, "Symbolic Execution for Security Analysis," Proceedings of the IEEE, vol. 108, no. 9, pp. 1102-1115, 2019.