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: Meta-Path Based Attentional Graph Learning Model for Vulnerability Detection

Mã nhóm: G07 Mã đề tài: 067

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	Đoàn Mạnh Đức	22520264	22520264@gm.uit.edu.vn
2	Võ Ngọc Bảo	22520127	22520127@gm.uit.edu.vn
3	Đỗ Hoàng Anh	22520041	22520041@gm.uit.edu.vn
4	Lê Quốc Anh	22520049	22520049@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:

Wen, X. C., Gao, C., Ye, J., Li, Y., Tian, Z., Jia, Y., & Wang, X. (2023). Meta-path based attentional graph learning model for vulnerability detection. *IEEE Transactions on Software Engineering*.

C. Dịch tên Tiếng Việt cho bài báo:

Mô hình Học Đồ Thị Chú Ý Dựa Trên Meta-Path để Phát Hiện Lỗ Hổng

[1] Ghi nội dung tương ứng theo mô tả

D. Tóm tắt nội dung chính:

Bài báo giới thiệu MAGNET, một mô hình học đồ thị chú ý dựa trên meta-path nhằm cải thiện hiệu quả phát hiện lỗ hổng mã. Trước thực trạng số lượng lỗ hổng phần mềm gia tăng mạnh mẽ, với mức tăng 185% các lỗ hổng rủi ro cao trong năm 2021 (theo báo cáo Bugcrowd), MAGNET được đề xuất để khắc phục các hạn chế của phương pháp hiện tại, vốn thường bỏ qua mối quan hệ không đồng nhất và khó nắm bắt phụ thuộc tầm xa trong đồ thị cấu trúc mã.

MAGNET kết hợp hai thành phần chính:

1. **Đồ thị meta-path đa cấp độ chi tiết:** Khai thác các mối quan hệ không đồng nhất trong đồ thị cấu trúc mã bằng cách xây dựng các meta-path, giúp biểu diễn thông tin chi tiết hơn.
2. **Mạng nơ-ron đồ thị chú ý phân cấp dựa trên meta-path (MHAGNN):** Sử dụng cơ chế chú ý meta-path để học các phụ thuộc cục bộ và cơ chế chú ý đa hạt để nắm bắt phụ thuộc tầm xa.

MAGNET được đánh giá trên ba tập dữ liệu phổ biến (FFMPeg+Qemu, Reveal và Fan et al.) và so sánh với sáu phương pháp tiên tiến, cho thấy vượt trội về điểm số F1 và recall. Đặc biệt, MAGNET đạt độ chính xác trên 70% đối với 25 loại CWE nguy hiểm nhất, cải thiện 27,78% so với phương pháp tốt nhất trước đó.

Tóm lại, MAGNET tận dụng hiệu quả các mối quan hệ không đồng nhất và phụ thuộc tầm xa, trở thành một phương pháp mới tiềm năng trong phát hiện lỗ hổng, góp phần nâng cao an ninh mạng.

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

1. Meta-Path: Bài báo giới thiệu khái niệm meta-path để biểu diễn các mối quan hệ phức tạp giữa các nút trong đồ thị cấu trúc mã. Meta-path là một chuỗi các nút và cạnh thể hiện một loại mối quan hệ cụ thể.

- **Mục đích:** Meta-path được sử dụng để biểu diễn các mối quan hệ phức tạp giữa các nút trong đồ thị cấu trúc mã. Thay vì chỉ xem xét mối quan hệ trực tiếp giữa hai nút, meta-path cho phép chúng ta xem xét các chuỗi nút và cạnh kết nối chúng, từ đó nắm bắt được ngữ cảnh và ý nghĩa của mối quan hệ.
- **Thuật toán: Xác định các loại nút và cạnh:** Đầu tiên, ta cần xác định các loại nút (A) và loại cạnh (R) có trong đồ thị cấu trúc mã. Ví dụ, loại nút có thể là "Statement", "Expression", "Symbol",... và loại cạnh có thể là "AST", "CFG", "DFG", "NCS" ....
- **Xây dựng meta-path:** Meta-path được định nghĩa là một chuỗi các loại nút và cạnh, ví dụ: (Statement, AST, Expression) hoặc (Identifier, DFG, Expression).
- **Lọc meta-path:** Bài báo đề xuất lọc bỏ các meta-path hiếm gặp để giảm độ phức tạp của mô hình.
- **Cách hoạt động:** Mỗi meta-path đại diện cho một loại quan hệ ngữ nghĩa cụ thể giữa các nút. Ví dụ, meta-path (Statement, AST, Expression) có thể biểu thị mối quan hệ "Câu lệnh chứa biểu thức". Bằng cách sử dụng meta-path, mô hình có thể học được các đặc trưng phức tạp hơn từ đồ thị cấu trúc mã.

2.Mạng nơ-ron đồ thị chú ý phân cấp dựa trên meta-path (Meta-path based Hierarchical Attentional Graph Neural Network - MHAGNN): Bài báo đề xuất một kiến trúc GNN mới được gọi là MHAGNN. MHAGNN sử dụng hai cơ chế chú ý chính:

Mục đích: MHAGNN là một kiến trúc GNN mới được đề xuất trong bài báo để học biểu diễn của đồ thị cấu trúc mã dựa trên meta-path. MHAGNN có khả năng nắm bắt thông tin cấu trúc của đồ thị, đồng thời kết hợp thông tin từ các meta-path và các cấp độ chi tiết của các loại nút.

- **Thuật toán:** MHAGNN sử dụng hai cơ chế chú ý chính:
- **Cơ chế chú ý meta-path:**

■ Tính toán điểm chú ý cho mỗi meta-path dựa trên loại nút và loại cạnh.
■ Điểm chú ý này thể hiện mức độ quan trọng của meta-path trong việc biểu diễn mối quan hệ giữa các nút.

- **Cơ chế chú ý đa cấp độ:**

■ Sử dụng cả average-pooling và max-pooling để học biểu diễn cho mỗi cấp độ chi tiết (Statement, Expression, Symbol).
■ Tính toán điểm chú ý cho mỗi cấp độ chi tiết, thể hiện mức độ quan trọng của cấp độ đó đối với biểu diễn của toàn bộ đồ thị.

- **Cách hoạt động:** MHAGNN học biểu diễn của đồ thị cấu trúc mã bằng cách kết hợp thông tin từ các meta-path và các cấp độ chi tiết của các loại nút. Quá trình này được thực hiện thông qua các tầng chú ý, cho phép mô hình tập trung vào những meta-path và cấp độ chi tiết quan trọng nhất.

3. Học sâu (Deep Learning):

- **Mục đích:** Mô hình MHAGNN được huấn luyện bằng cách sử dụng các kỹ thuật học sâu để học cách phân loại các đoạn mã nguồn là vulnerable hay non-vulnerable.
- **Thuật toán:**

■ Sử dụng hàm mất mát CrossEntropy để đo lường sự khác biệt giữa dự đoán của mô hình và nhãn thực tế.
■ Tối ưu hóa các tham số của mô hình bằng cách sử dụng thuật toán Adam.

- **Cách hoạt động:** Mô hình được huấn luyện trên tập dữ liệu lớn các đoạn mã nguồn có nhãn. Trong quá trình huấn luyện, mô hình tự động điều chỉnh các tham số của mình để dự đoán chính xác nhãn của các đoạn mã. Sau khi huấn luyện, mô hình có thể được sử dụng để phân loại các đoạn mã mới.

1. Môi trường thực nghiệm của bài báo:

- Cấu hình máy tính:
  - GPU: RTX 3090 với CUDA 11.4
  - OS: Ubuntu 20.04
- Các công cụ hỗ trợ sẵn có: Joern-tools (Trích xuất graph)
- Ngôn ngữ lập trình để hiện thực phương pháp: python
- Thư viện hỗ trợ: dgl. PyTorch
- Đối tượng nghiên cứu:
  - FFMpeg: 12460 vulnerables, 14858 non-vulnerables
  - Chrome_Debian: 2240 vulnerables, 20494 non-vulnerables
- Tiêu chí đánh giá tính hiệu quả của phương pháp: Accuracy, Recall, F1 score

2. Kết quả thực nghiệm của bài báo:

Metrics(%) \ Dataset	FFMPeg+Qemu [14]				Reveal [18]				Fan *et al.*. [37]			
Baseline	Accuracy	Precision	Recall	F1 score	Accuracy	Precision	Recall	F1 score	Accuracy	Precision	Recall	F1 score
VulDeePecker	49.61	46.05	32.55	38.14	76.37	21.13	13.10	16.17	81.19	**38.44**	12.75	19.15
Russell *et al.*	57.60	54.76	40.72	46.71	68.51	16.21	52.68	24.79	86.85	14.86	26.97	19.17
SySeVR	47.85	46.06	58.81	51.66	74.33	40.07	24.94	30.74	90.10	30.91	14.08	19.34
Devign	56.89	52.50	64.67	57.95	87.49	31.55	36.65	33.91	**92.78**	30.61	15.96	20.98
Reveal	61.07	55.50	70.70	62.19	81.77	31.55	61.14	41.62	87.14	17.22	34.04	22.87
IVDetect	57.26	52.37	57.55	54.84	-	-	-	-	-	-	-	-
MAGNET	**63.28**	**56.27**	**80.15**	**66.12**	**91.60**	**42.86**	**61.68**	**50.57**	91.38	22.71	**38.92**	**28.68**

Nhìn chung, MAGNET đạt kết quả tốt hơn và vượt trội hơn tất cả sáu phương pháp được so sánh trên cả ba bộ dữ liệu về chỉ số F1 và Recall.

Ba phương pháp dựa trên đồ thị (Devign, Reveal và IVDetect) cho kết quả tốt hơn so với ba phương pháp dựa trên token.

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

Dựa trên phân tích phương pháp và hệ thống từ bài báo tham khảo, nhóm đã thực hiện các công việc như sau:

Thu thập và chuẩn bị bộ dữ liệu mã nguồn từ các nguồn mở hoặc dự án thực tế, đồng thời tiền xử lý dữ liệu để chuẩn hóa và loại bỏ các thành phần không cần thiết.

Tiếp theo, cài đặt và sử dụng công cụ như Joern để xây dựng Code Property Graph (CPG), sau đó trích xuất các thông tin liên quan đến nút và cạnh (node, edge) từ CPG dựa trên giá trị và loại nút. Nhóm sẽ sử dụng phương pháp nhúng như Word2Vec để biểu diễn các nút dưới dạng vector và phân loại các loại nút chi tiết thành nhóm chính như Expression, Statement, và Function.

Từ đó, xây dựng mô hình học máy hoặc học sâu để xử lý các vector nút này, phục vụ các mục tiêu như phát hiện lỗi, phân tích bảo mật, hoặc tóm tắt mã. Sau khi huấn luyện mô hình, nhóm sẽ đánh giá hiệu suất dựa trên các tiêu chí cụ thể và tối ưu hóa mô hình cũng như pipeline xử lý.

Quá trình thực hiện bắt đầu bằng việc thu thập tập dữ liệu về mã nguồn hoặc các tạo phẩm lập trình (ở đây đã được cung cấp dataset sẵn là các source code C) sau đó được xử lý bằng công cụ như Joern để tạo ra Biểu đồ thuộc tính mã Code Property Praph (CPG) để nắm bắt cả cách biểu diễn cú pháp và ngữ nghĩa của mã. Các nút được trích xuất từ CPG cùng với các giá trị và loại liên quan của chúng, đồng thời chúng được đưa vào mô hình Word2Vec để tạo ra các phần nhúng vectơ nhằm nắm bắt các mối quan hệ theo ngữ cảnh. Sau đó, các phần nhúng nút được lọc và nhóm thành các danh mục cấp cao hơn, chẳng hạn như các nút Biểu thức, Câu lệnh và Hàm, giảm 69 loại nút ban đầu thành một tập hợp nhỏ hơn gồm ba loại

chính(Expression, Statement, Function). Cuối cùng, các biểu diễn vectơ rút gọn này được sử dụng làm đầu vào để huấn luyện mô hình học máy hoặc học sâu, cho phép thực hiện các tác vụ như phân loại, phân cụm hoặc phát hiện bất thường trong phân tích mã.

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

Trong quá trình thực hiện đề tài, nhóm gặp phải một số khó khăn:

Thứ nhất, việc thu thập và chuẩn bị dữ liệu mã nguồn có thể gặp thách thức do dữ liệu yêu cầu xử lý phức tạp để chuẩn hóa về dạng đúng.

Thứ hai, cấu hình và tối ưu hóa công cụ Joern hoặc các công cụ sinh Code Property Graph (CPG) cũng như khi triển khai các phương pháp nhúng (embedding) như Word2Vec hay các mô hình học sâu có thể tốn nhiều thời gian để tìm ra thiết lập phù hợp nhất, đồng thời đòi hỏi tài nguyên tính toán lớn. Khó khăn cũng xuất hiện trong quá trình giảm loại nút và phân loại chúng, đặc biệt khi đối mặt với các loại nút không rõ ràng.

Hơn nữa, việc xây dựng và huấn luyện mô hình học máy/học sâu để đạt được hiệu suất cao có thể phức tạp, do cần điều chỉnh nhiều tham số và tránh hiện tượng quá khớp (overfitting).

Cuối cùng, bộ dữ liệu sinh ra sau khi thực hiện các bước tiền xử lí sinh ra Control Flow Graph (CFG) để đưa vào mô hình có kích thước quá lớn, công thêm việc hạn chế về mặt tài nguyên tính toán nên kết quả của quá trình huấn luyện không đúng với mong đợi.

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

100%

4. NHẬT KÝ PHÂN CÔNG NHIỆM VỤ:

STT	Công việc	Phân công nhiệm vụ
1	Tìm hiểu phương pháp thực nghiệm của bài báo	Lê Quốc Anh Đỗ Hoàng Anh Võ Ngọc Bảo Đoàn Mạnh Đức
2	Tiền xử lí dữ liệu	Đỗ Hoàng Anh Võ Ngọc Bảo
3	Thực hiện train model	Võ Ngọc Bảo Đỗ Hoàng Anh Lê Quốc Anh Đoàn Mạnh Đức

4	Viết báo cáo đồ án	Lê Quốc Anh Đỗ Hoàng Anh Đoàn Mạnh Đức

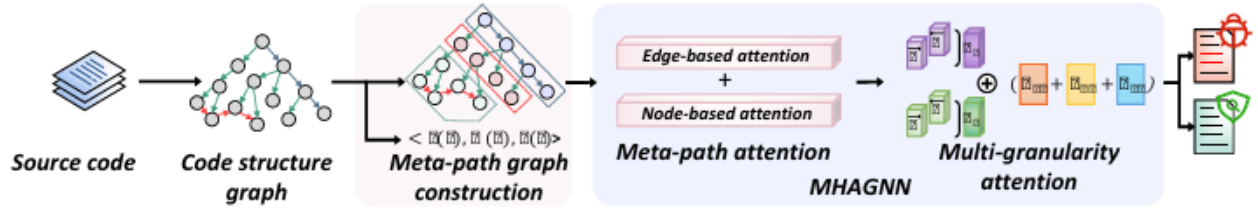
BÁO CÁO TỔNG KẾT CHI TIẾT

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)

1. Phương pháp thực hiện

Kiến trúc và thành phần của hệ thống MAGNET trong bài báo:



Bài báo đề xuất MAGNET, một mô hình học biểu diễn đồ dựa trên meta-path, nhằm phát hiện lỗ hổng bảo mật trong mã nguồn. Hệ thống MAGNET được cấu tạo bởi hai thành phần chính:

1. Đồ thị meta-path đa cấp độ chi tiết
2. Mạng nơ-ron đồ thị chú ý phân cấp dựa trên meta-path (MHAGNN)

Đối với phần 1. Đồ thị meta-path đa cấp độ chi tiết sẽ bao gồm :

- **Nhóm loại nút :**

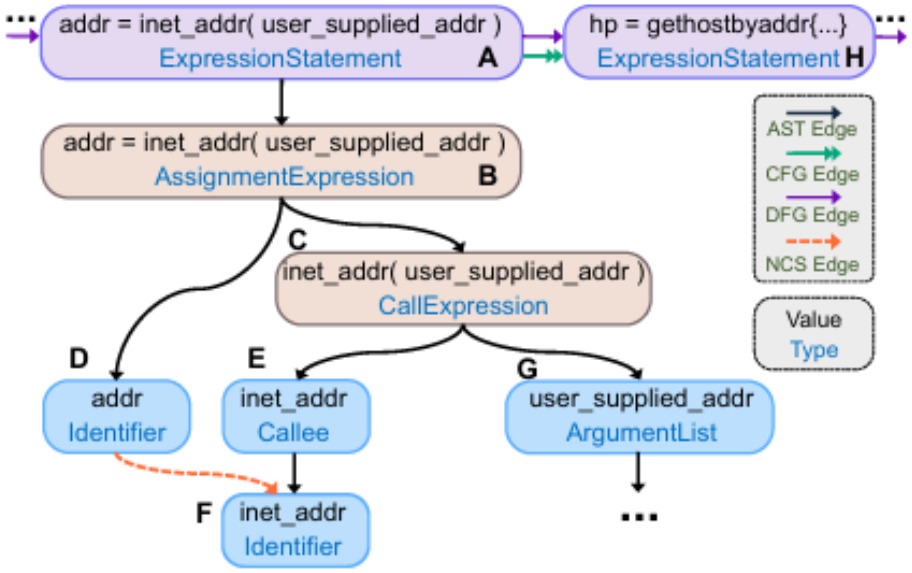
○ Nhằm giảm mức độ phức tạp của mô hình và tránh bị overfitting, bài báo đề xuất nhóm các loại nút trong đồ cấu trúc mã nguồn thành 3 cấp độ : "Statement", "Expression", và "Symbol".
○ Việc nhóm các loại nút trên nguyên tắc phân tích mã nguồn và phản ánh thông tin cấu trúc của giá trị nút, giúp quá trình học dựa trên DL hiệu quả hơn.

- **Xây dựng meta-path :**

○ Mỗi meta-path được định nghĩa là một bộ ba (τ(s), ψ(e), τ(t))
τ(s): Loại nút của nút nguồn *s*.
ψ(e): Loại cạnh của cạnh *e*
τ(t): Loại nút của nút đích *t*
- Ví dụ, meta-path **(St, 0, St)** biểu thị mối quan hệ giữa hai nút thuộc nhóm "Statement" được kết nối bởi cạnh "AST".

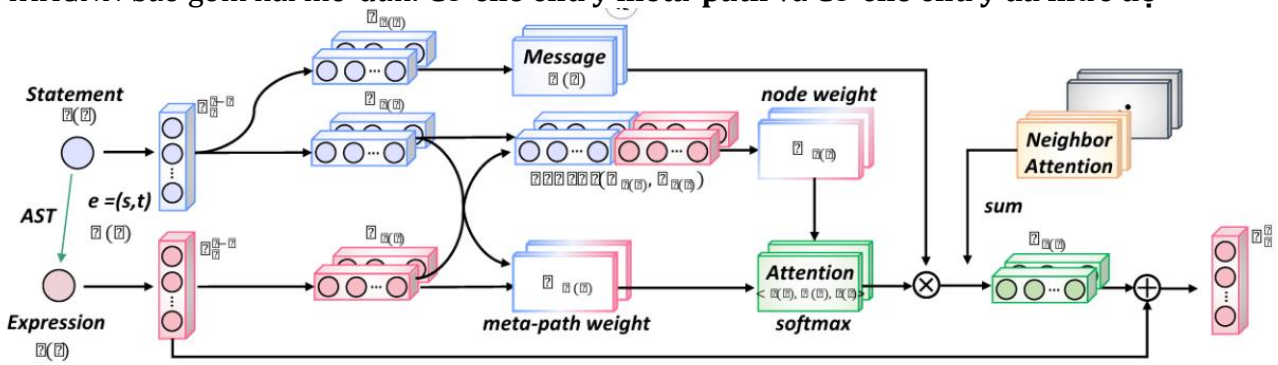
- **Cách meta-path ánh xạ với đồ thị cấu trúc mã nguồn:**

Giả sử ta muốn tìm meta-path giữa nút A ("ExpressionStatement") và nút H ("ExpressionStatement").
○ Ta thấy có một đường đi từ A đến H thông qua cạnh "DFG".
○ Cạnh "DFG" biểu thị luồng dữ liệu (Data Flow Graph) giữa hai câu lệnh.
○ Do đó, meta-path giữa A và H là **(St, DFG, St)**, thể hiện mối quan hệ luồng dữ liệu giữa hai nút thuộc nhóm "Statement".



○ Bài báo nhận thấy rằng việc sử dụng tất cả các meta-path có thể dẫn đến mô hình quá phức tạp và dễ bị overfitting. Do đó, họ đề xuất **lọc bỏ các meta-path hiếm gặp**, tức là những meta-path xuất hiện dưới 3 lần trong tập dữ liệu.
Ví dụ, meta-path **(Ex, 2, Ex)** có thể biểu thị mối quan hệ giữa hai nút thuộc nhóm "Expression" được kết nối bởi cạnh "CFG". Tuy nhiên, nếu meta-path này chỉ xuất hiện 1 hoặc 2 lần trong tập dữ liệu, nó sẽ bị loại bỏ.

2. Mạng nơ-ron đồ chú ý phân cấp dựa trên meta-path (MHAGNN)
MHAGNN bao gồm hai mô-đun: **Cơ chế chú ý meta-path** và **Cơ chế chú ý đa mức độ**



- **Cơ chế chú ý meta-path:** Nắm bắt biểu diễn của các mối quan hệ dị thể.

○ **Cơ chế chú ý nút**: Xác định mức độ quan trọng của các loại nút khác nhau trong việc biểu diễn cấu trúc đồ thị:
Đầu tiên, ta tính toán vectơ đại diện cho mối quan hệ giữa **nút nguồn s** và **nút đích t.** Vectơ này được tạo ra bằng cách **nối vectơ embedding** của nút s (Kl(s)) với vectơ embedding của nút t (Ql(t)).

- Sau đó, ta **nhân vectơ** này với **ma trận trọng số** W l τ(t) tương ứng với loại nút của nút đích t. Phép nhân này giúp đánh giá mức độ quan trọng của loại nút t đối với mối quan hệ giữa s và t.
- Cuối cùng, **kết quả được đưa qua hàm sigmoid** để thu được điểm attention nằm trong khoảng từ 0 đến 1.

$$Att^l_{node} = \sigma\left(W^l_{\tau(t)} \cdot \left(K^l(s)||Q^l(t)\right)\right) \quad (1)$$

$$K^l(s) = Linear_{\tau(s)}(h^{l-1}_s) \quad (2)$$

$$Q^l(t) = Linear_{\tau(t)}(h^{l-1}_t) \quad (3)$$

○ **Cơ chế chú ý cạnh:** Đánh giá tầm quan trọng của các loại cạnh khác nhau:

- Ta cũng tính toán vectơ đại diện cho mối quan hệ giữa nút nguồn s và nút đích t, sử dụng vectơ embedding của chúng (Kl(s) và Ql(t)).
- Vectơ này được nhân với ma trận trọng số Wψ(e) tương ứng với loại cạnh e.
- Kết quả được nhân thêm với tham số μ(ψ(e)) đại diện cho mức độ quan trọng của loại cạnh e.
- Cuối cùng, kết quả được chia cho √d/h để chuẩn hóa

$$Att^l_{edge} = \left(K^l(s)W_{\psi(e)}Q^l(t)^T\right) \cdot \frac{\mu(\psi(e))}{\sqrt{\frac{d}{h}}} \quad (4)$$

○ **Kết hợp chú ý nút và chú ý cạnh để tính toán điểm chú ý meta-path:** Kết hợp node-based attention và edge-based attention để biểu diễn quan hệ không đồng nhất giữa các nút

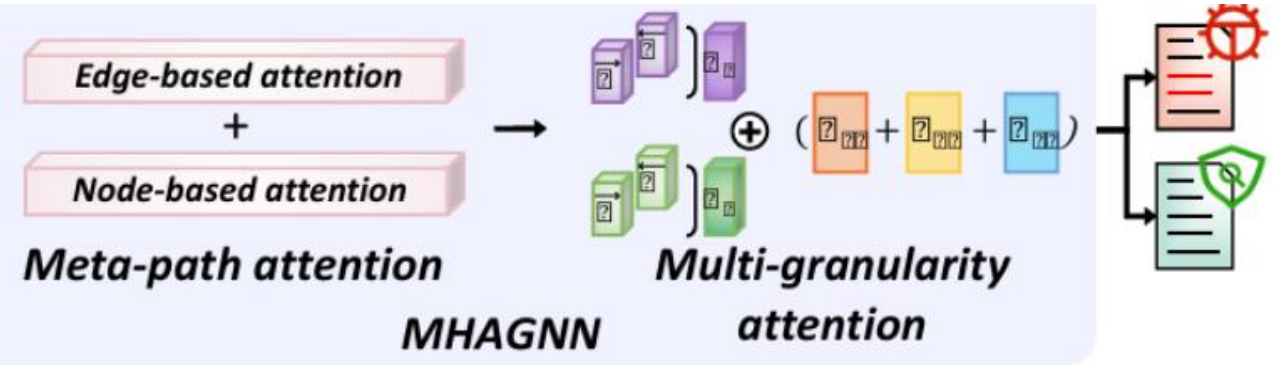
- Đầu tiên, ta cộng điểm node-based attention và điểm edge-based attention để thu được điểm attention tổng hợp cho mối quan hệ giữa nút s và t.
- Sau đó, ta nối kết quả và áp dụng hàm softmax để chuẩn hóa điểm attention, đảm bảo tổng các điểm attention bằng 1.

$$Att^l_{(\tau(s),\psi(e),\tau(t))} = softmax\left(\|^H_1\left(Att^l_{edge} + Att^l_{node}\right)\right) \quad (5)$$

● **Cơ chế chú ý đa cấp độ:** Module này được thiết kế để nắm bắt phụ thuộc tầm xa trong đồ thị meta-path.

- **Áp dụng average-pooling và max-pooling:** Trích xuất thông tin tổng hợp từ các nút ở các mức độ chi tiết (granularity) khác nhau.
- **Tính điểm chú ý cho mỗi mức độ chi tiết.**

-> **Kết hợp thông tin từ các mức độ chi tiết khác nhau:** Tạo biểu diễn đồ thị cuối cùng, MHAGNN dự đoán xem mã nguồn có chứa lỗi hay không.



Multi-granularity attention: Nắm bắt phụ thuộc phạm vi xa trong đồ thị meta-path bằng cách xem xét mức độ quan trọng của các quan hệ không đồng nhất ở các mức độ chi tiết (granularities) khác nhau.

- Đầu tiên, ta tính toán đặc trưng cho mỗi mức độ chi tiết i (Fi) bằng cách ghép nối (concat) các vectơ embedding của các nút thuộc loại i.
- Sau đó, ta áp dụng lớp average-pooling và max-pooling lên Fi để thu được hai vectơ đặc trưng đại diện cho mức độ chi tiết i.
- Hai vectơ này được nhân với trọng số ω1,i và ω2,i tương ứng, sau đó cộng lại với nhau.
- Cuối cùng, kết quả được đưa qua mạng perceptron đa lớp (MLP) và hàm sigmoid để thu được điểm multi-granularity attention cho mức độ chi tiết i

$$M = \sigma\left(MLP\left(\omega_{1,i} \cdot AvgPool(F_i) + \omega_{2,i} \cdot MaxPool(F_i)\right)\right) \quad (i = st, ex, sy) \tag{8}$$

Trong quá trình triển khai đề tài, nhóm đã xây dựng kiến trúc tổng quan của mô hình **Meta-Path Based Attentional Graph Learning Model for Vulnerability Detection**, bao gồm các thành phần chính sau:

1. **Tiền xử lý dữ liệu**:
   - Xử lý mã nguồn từ các kho lưu trữ mã mở, chuẩn hóa cú pháp và chuyển đổi thành biểu diễn đồ thị (AST - Abstract Syntax Tree).
   - Tích hợp thông tin ngữ cảnh từ mã nguồn và các tài liệu liên quan để tạo meta-paths.
2. **Xây dựng đồ thị và meta-path**:
   - Tạo đồ thị biểu diễn mối quan hệ giữa các thành phần mã nguồn như hàm, biến, và luồng điều khiển.
   - Khai thác meta-path để định nghĩa các quan hệ phức tạp hơn, như quan hệ giữa các hàm qua tham số hoặc biến dùng chung.
3. **Mô hình học sâu đồ thị (Graph Neural Network)**:
   - Xây dựng mô hình GNN với cơ chế chú ý (attention mechanism) để tập trung vào các meta-path quan trọng.

   - Triển khai các lớp học sâu (deep layers) để xử lý dữ liệu phức tạp và cải thiện độ chính xác trong việc phát hiện lỗ hổng.
4. **Huấn luyện và đánh giá**:
   - Sử dụng tập dữ liệu gán nhãn về các lỗ hổng đã biết để huấn luyện mô hình.
   - Đánh giá mô hình dựa trên các chỉ số như độ chính xác, độ nhạy (recall), và F1-score.

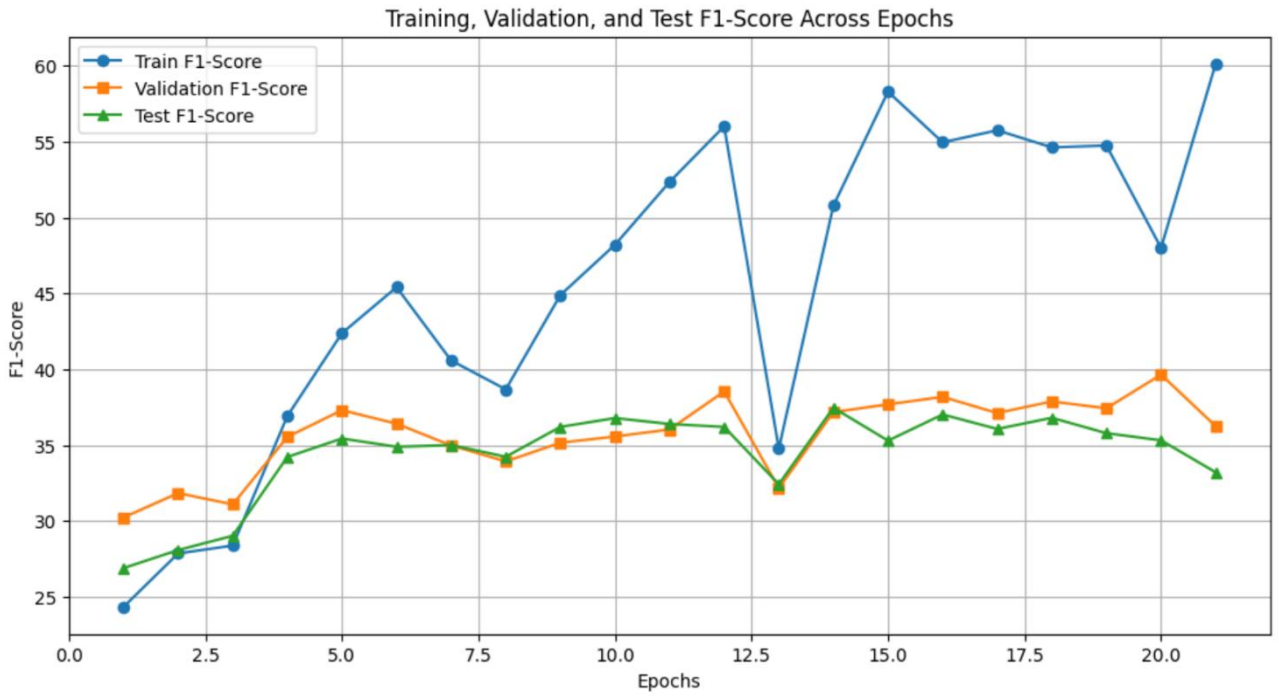
2. Chi tiết cài đặt, hiện thực

- Cài đặt cấu hình máy tính:
  - Chuẩn bị môi trường python: PyTorch, dgl
  - CUDA Computing Toolkit (Nếu có NVIDIA GPU)
  - Joern Tool
- Cấu hình máy tính sử dụng:
  1. Máy 1:
     - OS: Window 11
     - CPU: 6 cores, 12 processors
     - RAM: 32GB
     - GPU: RTX 3050Ti
     - Ngôn ngữ lập trình: Python 3.12
     - Thư viện hỗ trợ: dgl 2.5, PyTorch 2.5.1+cu124
     - CUDA 12.7
  2. Máy 2:
     - OS: Ubuntu Linux (64-bit)
     - CPU: 8 CPU(s)
     - RAM: 24GB
     - Ngôn ngữ lập trình: Python 3.12
     - Thư viện hỗ trợ: dgl 2.5, PyTorch 2.5.1+cu124
     - CUDA 12.7
- Dataset:
  - FFMpeg+Quemu
  - Chrome+Debian

3. Kết quả thực nghiệm

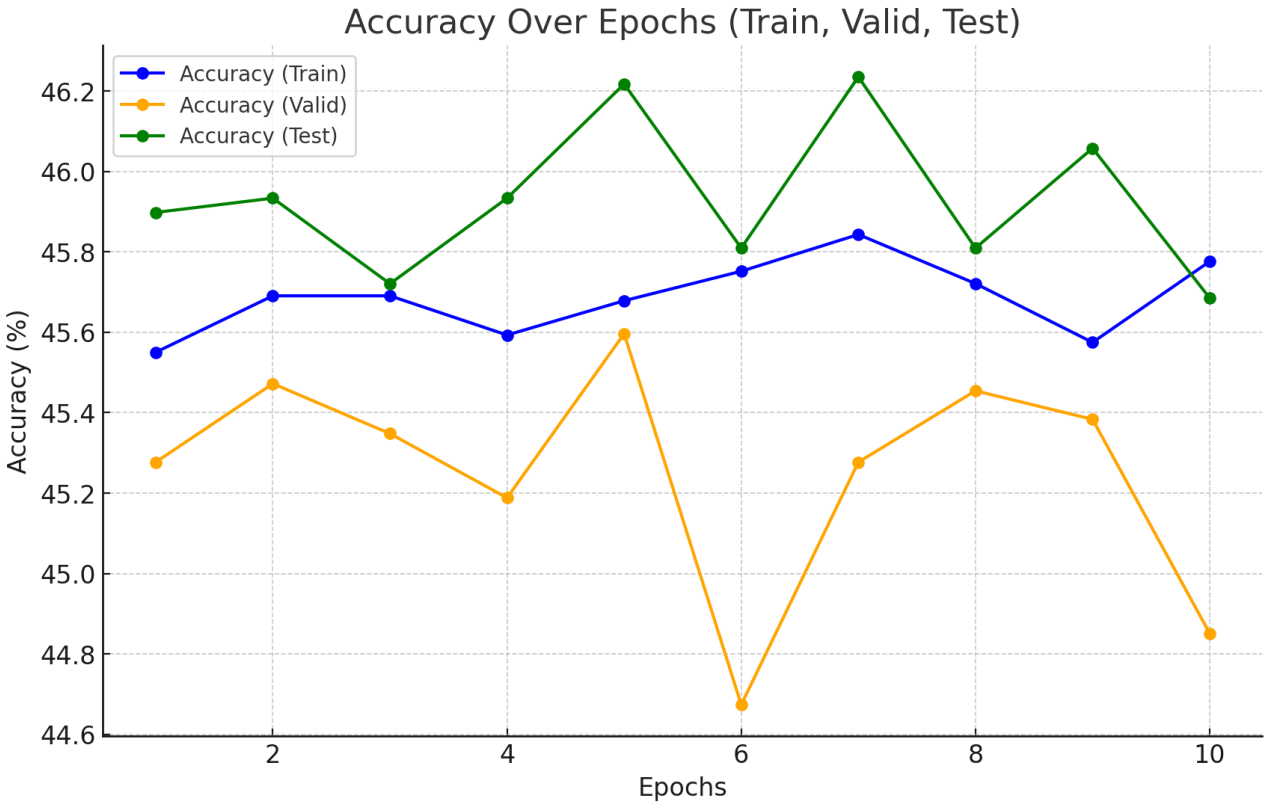
Kết quả thực nghiệm trên tập dataset Chrome+Debian

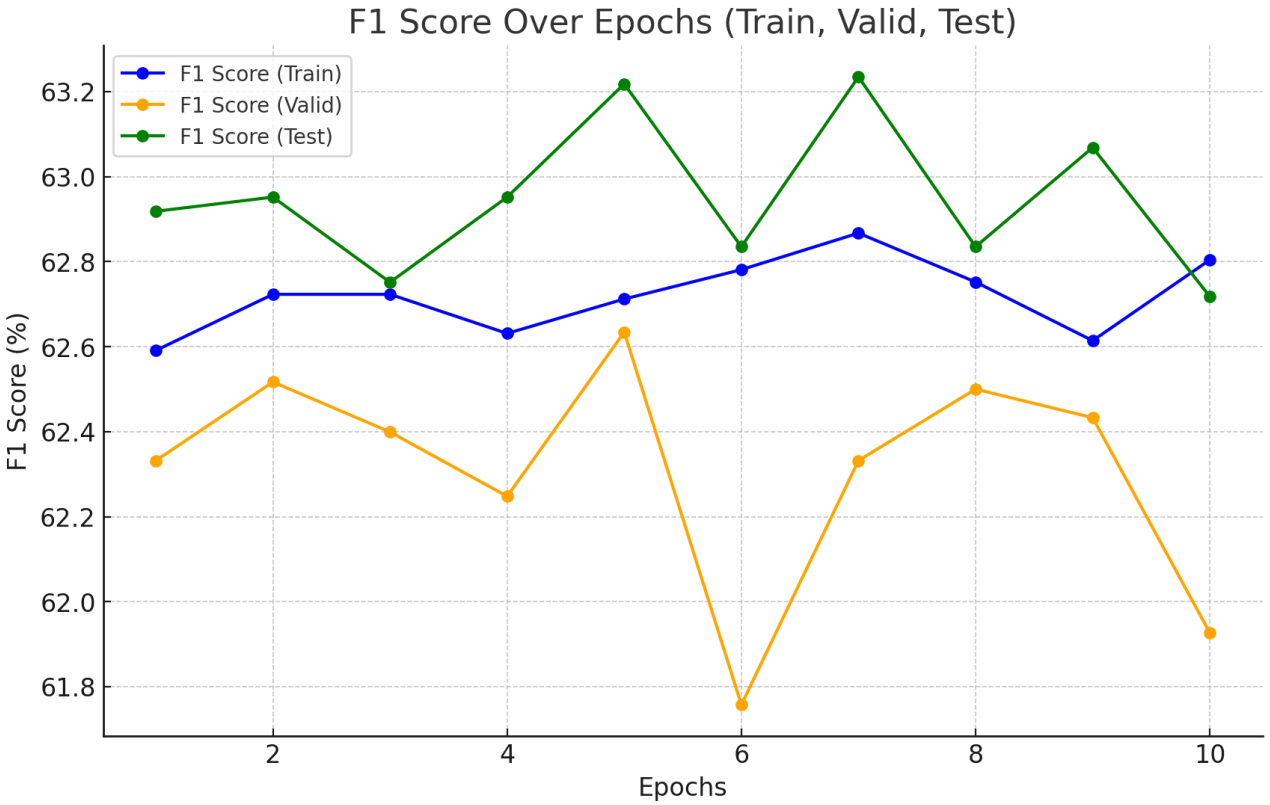
Kết quả cho thấy độ chính xác cao (>90%), nhưng F1-Score chỉ đạt 60%. Điều này cho thấy mô hình hoạt động tốt trong việc phân loại các mẫu thuộc lớp không dễ bị tấn công (non-vulnerable), nhưng gặp khó khăn trong việc phát hiện các mẫu thuộc lớp dễ bị tấn công (vulnerable), dẫn đến sự mất cân bằng giữa **precision** (độ chính xác khi dự đoán) và **recall** (khả năng phát hiện đúng).



Kết quả thực nghiệm trên tập dataset [FFMpeg+Quemu]

Độ chính xác thấp hơn đáng kể so với bài báo gốc (45,8% so với 63,28%). Tuy nhiên, F1-Score chỉ chênh lệch không đáng kể, cho thấy mô hình vẫn có khả năng phát hiện các mẫu dễ bị tấn công, nhưng chưa đạt được hiệu quả cao trong việc phân loại tổng thể.


Accuracy Over Epochs (Train, Valid, Test)
Accuracy (Train)
Accuracy (Valid)
Accuracy (Test)
Accuracy (%)
Epochs
46.2
46.0
45.8
45.6
45.4
45.2
45.0
44.8
44.6
2
4
6
8
10


F1 Score Over Epochs (Train, Valid, Test)
F1 Score (Train)
F1 Score (Valid)
F1 Score (Test)
F1 Score (%)
Epochs
63.2
63.0
62.8
62.6
62.4
62.2
62.0
61.8
2
4
6
8
10

4. Hướng phát triển

Đề tài **"Meta-Path Based Attentional Graph Learning Model for Vulnerability Detection"** có tiềm năng phát triển theo các hướng sau:

1. Mở Rộng Đối Tượng Áp Dụng

- **Đa ngôn ngữ lập trình**: Phát triển mô hình để hỗ trợ nhiều ngôn ngữ lập trình khác nhau, không chỉ giới hạn ở các ngôn ngữ như C hoặc C++ mà còn các ngôn ngữ phổ biến khác như Python, Java, ...
- **Ứng dụng cho các hệ thống phức tạp**: Nghiên cứu mô hình trên các hệ thống đa module hoặc hệ thống phân tán để phát hiện lỗ hổng trong môi trường thực tế.

2. Cải Tiến Kỹ Thuật

- **Tăng hiệu suất**: Áp dụng các cải tiến về tính toán phân tán hoặc tận dụng GPU/TPU để giảm thời gian xử lý, làm cho mô hình khả thi hơn khi áp dụng trên quy mô lớn.

Đề tài có tính ứng dụng cao, đặc biệt trong lĩnh vực bảo mật phần mềm. Với sự gia tăng của các cuộc tấn công mạng và số lượng mã nguồn mở được sử dụng, mô hình này mang lại các giá trị sau:

- **Tự động hóa phát hiện lỗ hổng**: Giảm đáng kể thời gian và nguồn lực cần thiết để kiểm tra mã nguồn thủ công.
- **Ứng dụng thực tế**: Có thể được sử dụng trong nhiều ngành công nghiệp như tài chính, y tế, và sản xuất, nơi bảo mật phần mềm là ưu tiên hàng đầu.
- **Hỗ trợ lập trình viên**: Giúp nhà phát triển nhanh chóng phát hiện và sửa lỗi, cải thiện chất lượng phần mềm.

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