PHÁT HIỆN LỖ HỔNG PHẦN MỀM DỰA TRÊN PHƯƠNG PHÁP XỬ LÝ NGÔN NGỮ TỰ NHIÊN

A SOLUTION FOR DETECTING SOFTWARE VULNERABILITIES BASED ON NATURAL LANGUAGE PROCESSING TECHNOLOGY

Bùi Văn Công

Khoa Công nghệ thông tin, Trường Đại học Kinh tế - Kỹ thuật Công nghiệp

Đến Tòa soạn ngày 22/02/2024, chấp nhận đăng ngày 30/5/2024

Tóm tắt: Đảm bảo chất lượng phần mềm là ưu tiên cao nhất trong toàn bộ quá trình phát triển phần mềm. Bài báo đề xuất một phương pháp kết hợp dựa trên 3 kỹ thuật xử lý chính: Word2vec kết hợp LSTM; Word2vec kết hợp Bidirectional LSTM; Phát hiện lỗ hổng phần mềm sử dụng BERT. Trong đó Word2vec kết hợp LSTM, BiLSTM thực hiện việc chuyển mã đã được xử lý thành các vectơ nhúng. Các vectơ nhúng được đưa vào mạng LSTM và BiLSTM với số lượng các nút ẩn khác nhau để tiến hành học và phân loại lỗ hổng. Mô hình BERT biểu diễn từ theo cơ chế hai chiều sử dụng kỹ thuật Transformer, các attention mechanism dùng để hiểu ngữ cảnh từ cả hai phía của các từ trong câu. Kết quả phân loại đưa qua hàm softmax để trả về một vectơ chứa dự đoán xác suất từng loại lỗ hổng. Để minh họa sự vượt trội của phương pháp, tiến hành thực nghiệm trên bộ dữ liệu SARD. Kết quả thực nghiệm cho độ chính xác lên đến 82,63% tương ứng tỷ lệ bỏ sót chỉ còn 17,37% cho thấy hiệu quả tốt.

Từ khóa: Lỗ hổng mã nguồn, phát hiện lỗ hổng mã nguồn, phân loại.

Abstract: The crucial work in process of software development is ensuring quality of software. This paper proposes a solution based on three main integration technologies for detecting software vulnerability: Word2vec combining to LSTM; Word2vec combining to Bidirectional LSTM and BERT. In two first combining technologies, source code is converted into embedded vectors before feeding into LSTM and BiLSTM networks with different numbers of hidden nodes to learn and classify vulnerabilities. The BERT model represents words bidirectionally using the Transformer mechanism which its attention mechanisms are used to interpret the context of words from both sides of a sentence. Several experiments are implemented with SARD dataset to prove the advantages of proposed solutions. The experimental results demonstrated its effectiveness with an accuracy of up to 82.63% as well as an error rate of only 17.37%.

Keywords: Source code vulnerability, source code vulnerability detection, classification

1. TỔNG QUAN NGHIÊN CỨU

1.1. Vấn đề phát hiện lỗ hổng hiện nay

Ngày nay, tỷ lệ gia tăng lỗ hổng trên phần mềm máy tính là rất lớn, dễ dàng trở thành mục tiêu cho các cuộc tấn công cố ý, trong khi các tin tặc lại có vô số các phương pháp khai thác các lỗ hổng này. Điều này thể hiện trong báo cáo [1] cho thấy mức độ thiệt hại và sự gia tăng đột biến về số lượng lỗ hổng phần mềm ngày càng nghiêm trọng. Khi hệ thống đang tồn tại lỗ hổng, nếu người quản lý không phát hiện ra và cài đặt bản vá trước khi tin tặc biết cách khai thác, chúng có thể bị sử dụng

để thực hiện các hành vi nguy hiểm như đánh cắp các dữ liệu quan trọng, gây từ chối dịch vụ,… và nguy hiểm nhất là nắm quyền kiểm soát hệ thống và phát tán virus. Để phát hiện lỗ hổng mã nguồn, nghiên cứu [2] chỉ ra có nhiều CVE được phát hiện và công bố. Đặc biệt mức độ nghiêm trọng của các CVE là rất lớn. Đồng nghĩa với việc khi các CVE này bị tấn công và khai thác sẽ dẫn đến hậu quả rất nguy hiểm cho các hệ thống của tổ chức. Có hai hình thức phát hiện lỗ hổng phần mềm đó là:

- Phát hiện lỗ hổng mã nguồn dựa trên phân tích thủ công: Đặc điểm của phương pháp này là các chuyên gia sẽ tự mình phân tích mã nguồn, sau đó dựa trên các kinh nghiệm, tập luật hoặc dấu hiệu để xác định đâu là mã nguồn chứa lỗ hổng và đâu là mã nguồn bình thường. Một số kỹ thuật thường được áp dụng theo hướng tiếp cận này là: dịch ngược, kiểm thử... Với cách tiếp cận này thì các lỗ hổng sẽ được phát hiện chính xác và mang lại hiệu quả cao. Tuy nhiên nhược điểm của phương pháp này là yêu cầu người phân tích phải có trình độ cao đồng thời tốn rất nhiều thời gian.

- Phát hiện lỗ hổng mã nguồn dựa trên phân tích tự động: Đang được nghiên cứu và áp dụng nhiều hiện nay do có nhiều công nghệ tính toán cũng như phương pháp hỗ trợ mạnh mẽ. Có thể nhận thấy các hướng tiếp cận gần đây thường tập trung vào phân tích mã nguồn theo dạng ngữ nghĩa và cú pháp rồi sử dụng các thuật toán học máy hoặc học sâu để tiếp tục phân tích và dự đoán lỗ hổng. Tuy nhiên đa phần chỉ phát hiện code smell hoặc bug trong cú pháp. Để khắc phục các vấn đề trên bài báo đề xuất mô hình sử dụng phân tích học sâu dựa trên luồng xử lý nhằm bổ sung thêm cho phân tích theo cú pháp còn được trích xuất qua đồ thị CFG, biểu diễn được logic cũng như cả mối quan hệ giữa các đỉnh với nhau thông qua sự kết hợp của 3 kỹ thuật xử lý chính: Word2vec kết hợp LSTM, Word2vec kết hợp BiLSTM, phát hiện lỗ hổng phần mềm sử dụng BERT.

1.2. Đề xuất hướng giải quyết vấn đề

Trong bài báo đề xuất phương pháp học sâu sử dụng kỹ thuật xử lý ngôn ngữ tự nhiên: trước tiên mã nguồn sẽ được phân tích và chuẩn hóa thông qua bước tiền xử lý bằng cách sử dụng thuật toán BERT. Đầu vào cho mô hình phát hiện lỗ hổng phần mềm sử dụng học sâu được lấy dưới dạng một chuỗi ký tự dài bao gồm toàn bộ tệp mã. Sau đó, chuỗi này được chia thành các từ và từ con bằng một trình mã hóa. Các ký tự cú pháp như dấu ngoặc, ngoặc, dấu chấm,… được coi là các từ riêng biệt. Sau đó các từ được mã hóa thành các vectơ biểu diễn bởi bộ mã hóa. Tiếp theo mô hình Word2vec + LSTM và Word2vec + BiLSTM được đề xuất nhằm tính toán và trích xuất các đặc trưng của mã nguồn. Trong mô hình này, các thành phần xử lý chính bao gồm tiền xử lý, sử dụng Word2vec để chuyển chuỗi mã đã được xử lý thành các vector nhúng. Sau đó các vector nhúng được đưa vào mạng LSTM và BiLSTM để tiến hành học và phân loại lỗ hổng. Kết quả phân loại sẽ được đi qua 1 hàm softmax để trả về 1 vector chứa dự đoán xác suất xảy ra của từng loại lỗ hổng đã cho.

1.3. Đóng góp của bài báo

Những đóng góp chính bài báo tóm tắt sau:

- Đề xuất sử dụng mô hình BERT với cơ chế hai chiều và attention cho kết quả phát hiện lỗ hổng tốt hơn hẳn. Tỷ lệ phát hiện lỗ hổng mã nguồn cho kết quả chính xác lên đến 82,63% tương ứng với tỷ lệ bỏ sót chỉ còn 17,37%. Các kết quả thực nghiệm trong mục 4 của bài báo đã chứng minh được sự hiệu quả vượt trội

đối với bài toán phát hiện lỗ hổng mã nguồn.

- Thực nghiệm so sánh và đánh giá sự hiệu quả trong việc phát hiện lỗ hổng của mã nguồn trong bảng 3 cho thấy số lượng layers tăng thì độ chính xác của mô hình cũng tăng theo. Tuy nhiên, việc tăng số layers làm tăng hiệu suất mô hình lên khoảng 2% nhưng đồng thời cũng tăng thời gian xử lý của mô hình lên. Trong thực tế, để cân bằng giữa hiệu suất và thời gian cần lựa chọn các tham số phù hợp để cho ra được kết quả như mong đợi. Các kết quả thử nghiệm để lựa chọn các tham số phù hợp cũng như mô hình được tác giả trình bày trong mục 4.4 của bài báo.

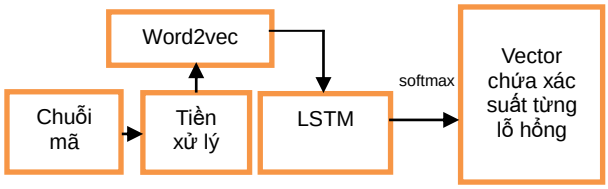
2. CÁC NGHIÊN CỨU LIÊN QUAN

Nghiên cứu [9], Junjun Guo cùng các cộng sự đề xuất mô hình kết hợp CNN và LSTM cho phát hiện lỗ hổng mã nguồn. Trong đó, mạng CNN dùng để trích xuất dữ liệu còn mạng LSTM để biểu diễn mã đặc trưng mã nguồn. Kết quả thực nghiệm cho thấy mô hình này đã phát hiện chính xác lỗ hổng mã nguồn là 80%. Tỷ lệ âm tính giả (FNR) và tỷ lệ dương tính giả (FPR) đều dưới 20%. Cùng hướng tiếp cận như vậy, nghiên cứu [7], Huan Mei cùng các cộng sự đã đề xuất mô hình kết hợp giữa 3 thành phần là CLSTM, CBiLSTM (sequential structure), and CNN-BiLSTM (parallel structure). Kết quả thực nghiệm cho thấy mô hình nhóm tác giả đề xuất mang lại hiệu quả cao hơn các hướng tiếp cận sử dụng CodeBERT, Word2Vec và FastText... Nghiên cứu [5] đề xuất phương pháp trích xuất đặc trưng mã nguồn dựa trên mười dự án mã nguồn mở. Kết quả thực nghiệm cho thấy đã cải thiện đáng kể hiệu quả việc phát hiện lỗi trên các phần mềm và các lỗi được chỉ ra trong các dự án. Nghiên cứu [8] sử dụng thuật toán học sâu LSTM để phân loại lỗ hổng mã nguồn sau khi mã nguồn được phân tích thông qua cây cú pháp trừu tượng. Jacob Devlin và các cộng sự [6] đề xuất 3 phương pháp phát hiện lỗ hổng phần mềm dựa trên dự đoán động. Đó kỹ thuật Tiêm lỗi (Fault injection), Fuzzing và Dynamic Taint. Với kỹ thuật Tiêm lỗi thì mỗi một lần thử thất bại là một lỗ hổng tiềm năng của hệ thống được đưa ra. Còn kỹ thuật Fuzzing, trong thử nghiệm đơn giản hơn kỹ thuật Tiêm lỗi vì luôn biết được thông qua khi kiểm tra hệ thống. Kỹ thuật Dynamic Taint tương tự như kỹ thuật phân tích dự đoán tĩnh. Nó cho phép phát hiện các vấn đề của dữ liệu đầu vào có thể làm phát sinh lỗ hổng. Poornima Haridas [3] và các cộng sự đề xuất mô hình SiCaGCN cho phát hiện lỗ hổng bảo mật dựa trên CFG và GCN. Kết quả thực nghiệm với việc điều chỉnh khoảng cách đồ thị đã mang lại hiệu quả tốt hơn cho các mạng đồ thị học sâu khác.

3. ĐỀ XUẤT MÔ HÌNH

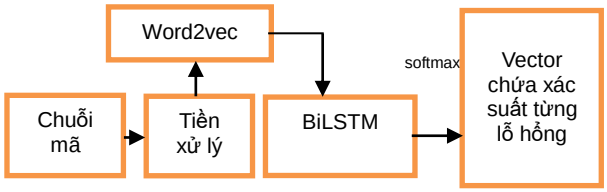
3.1. Kiến trúc mô hình đề xuất

Mô hình Word2vec kết hợp với LSTM



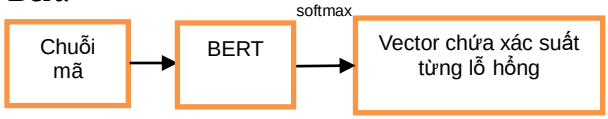
Hình 1. Mô hình Word2vec kết hợp LSTM

Mô hình Word2vec kết hợp với BiLSTM.



Hình 2. Mô hình Word2vec kết hợp BiLSTM

Mô hình phát hiện lỗ hổng phần mềm sử dụng Bert.



Hình 3. Các thành phần chính mô hình Bert

Chức năng và nhiệm vụ các khối con trong mô hình đề xuất thể hiện qua các hình như sau:

- **Chuỗi mã:** là dữ liệu phục vụ quá trình huấn luyện và phát hiện lỗ hổng. Mã nguồn sẽ bao gồm nhiều dạng ngôn ngữ và lỗ hổng khác nhau. Trong bài báo này sử dụng bộ dữ liệu tham chiếu về bảo hiểm phần mềm (SARD) do Viện Tiêu chuẩn và Công nghệ Quốc gia (NIST) tạo ra.

- **Khối tiền xử lý:** Với bộ dữ liệu tạo thành từ các tệp văn bản chứa mã thô gồm các tệp chứa lỗ hổng và các tệp không chứa lỗ hổng, tiến hành gắn nhãn để phân loại. Sau đó tiến hành loại bỏ các thành phần nhiễu khỏi dữ liệu bằng cách xóa những chú thích, đặt lại tên biến, tên hàm để tạo nên một cấu trúc mã nhất quán, không chứa những thông tin ảnh hưởng tới quá trình đào tạo mô hình.

- **Khối Word2vec:** là mô hình giúp tạo ra các biểu diễn embedding của từ trong một không gian có số chiều thấp hơn nhiều lần so với số từ trong từ điển. Ý tưởng đó là hai từ (từ con) xuất hiện trong những văn cảnh giống nhau thường có ý nghĩa gần với nhau. Hay có thể đoán được một từ (từ con) nếu biết các từ xung quanh nó trong câu. Như vậy có hai cách khác nhau để xây dựng mô hình Word2vec đó là dự đoán những từ ngữ cảnh nếu biết trước từ đích (Skip-gram) hoặc CBOW (Continuous Bag of Words) dựa vào những từ ngữ cảnh để dự đoán từ đích.

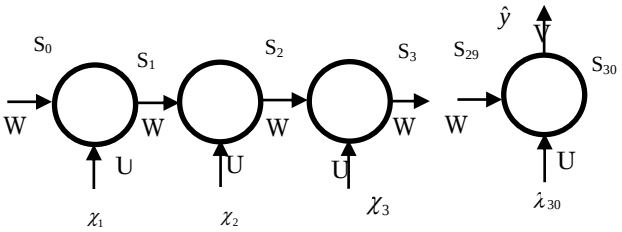
- **Khối BERT:** Khối BERT sử dụng Transformer là một mô hình attention học mối tương quan giữa các từ (hoặc 1 phần của từ) trong một văn bản [6].

- **Khối LSTM:** Khối này được đề xuất là một trong những giải pháp cho vấn đề phát hiện lỗ hổng phần mềm từ mã nguồn. Điểm chính trong kiến trúc LSTM chính là các ô nhớ với các cổng cho phép lưu trữ hoặc truy xuất thông tin. LSTM chỉ có thể dự đoán nhãn của từ hiện tại dựa trên thông tin có được từ các từ nằm trước đó.

- **Khối BiLSTM:** Như đã nêu, điểm hạn chế của LSTM chính là chỉ dự đoán nhãn của từ hiện tại dựa trên thông tin có được từ các từ nằm trước đó. Trong khi, việc nhận dạng chính xác cấu trúc và ngữ cảnh của một đoạn mã nguồn không chỉ cần thông tin ở phía trước của từ đang xét mà còn cả các thông tin phía sau. Điều này LSTM lại không thể nhận dạng được. Chính vì thế BiLSTM đã được tạo ra để khắc phục điểm yếu trên. Một kiến trúc BiLSTM thường chứa 2 mạng LSTM đơn được sử dụng đồng thời và độc lập để mô hình hóa chuỗi đầu vào theo 2 hướng: từ trái sang phải và từ phải sang trái.

Thuật toán:

Mạng LSTM là một dạng đặc biệt của mạng nơron hồi quy (RNN), được đưa ra để giải quyết vấn đề triệt tiêu gradient trong RNN.



Hình 4. Mô hình RNN

Đạo hàm của L với W ở state thứ i:

$$\frac{\partial L}{\partial \mathrm{w}} = \frac{\partial L}{\partial \hat{\mathrm{y}}} * \frac{\partial \hat{\mathrm{y}}}{\partial s_{so}} * \frac{\partial s_{so}}{\partial s_i} * \frac{\partial s_i^{'}}{\partial \mathrm{w}'} \quad (1)$$

Trong đó

$$\frac{\partial s_{so}}{\partial s_i} = \Pi_{j=1}^{29} \frac{\partial s_{j+1}}{\partial s_j}$$

Giả sử activation là hàm tanh,

$$\frac{\partial s_t}{\partial s_{t-1}} = \left(1 - s_t^2\right) * \mathrm{W}$$

$$\Rightarrow \frac{\partial s_{30}}{\partial s_i} = \mathrm{W}^{30-i} * \prod_{j=1}^{29} \left(1 - s_j^2\right)$$

Ta có: $s_j<1, W<1$ => Ở những state xa thì $\frac{\partial s_{30}}{\partial s_i} \approx 0$ hay: $\frac{\partial L}{\partial w} \approx 0$, hiện tượng **vanishing gradient.**

Cơ chế xác định các kết nối lân cận của mỗi nút thông qua phương trình sau đây:

$$\hat{h}_i^{l+1} = O_h^l \prod_{k=1}^{H} \left(\sum_{j \in N_i} w_{ij}^{k,l} V^{k,l} h_j^l \right)$$

Trong đó:

$$w_{ij}^{k,l} = \text{softmax}_j \left(\frac{Q^{k,l} h_i^l . K^{k,l} h_j^l}{\sqrt{d_k}} \right) \quad (2)$$

Vấn đề phát hiện: Để phân loại các vector của lỗ hổng và vector bình thường, tác giả sử dụng 2 lớp Fully Connected Layers và Softmax Layers. Fully Connected Layers giống như mạng MLP có nhiệm vụ học các thuộc tính được xử lý qua các lớp LSTM còn lớp Softmax Layers làm nhiệm vụ tính toán xác suất nhãn đầu ra. Hàm softmax [4] được mô tả như công thức (3):

$$a_i = \frac{e^{z_i}}{\sum_{j=1}^{C} e^{z_j}} \quad \forall i = 1, 2, ..., C \quad (3)$$

Trong đó:

C là số lượng lớp $z = [z_1, z_2, ..., z_C]$ là vector output của mạng LSTM tương ứng với đồ thị đầu vào cần được phân loại;

a_i là xác suất để input đó rơi vào lớp thứ i được tính toán bằng hàm softmax.

4. THỰC NGHIỆM VÀ ĐÁNH GIÁ

4.1. Dữ liệu thử nghiệm

Bộ dữ liệu được sử dụng để thử nghiệm chính là bộ dữ liệu tham chiếu về bảo hiểm phần mềm (SARD) do Viện Tiêu chuẩn và Công nghệ Quốc gia (NIST) tạo ra. SARD không chỉ chứa các lỗ hổng bảo mật mà còn chứa các bản sao đã không tồn tại lỗ hổng của chúng. Bộ dữ liệu được xử lý trước khi đào tạo bằng cách loại bỏ tất cả các phần không mong muốn có thể khiến các mô hình bị quá tải và mất cân bằng. Cuối cùng bộ dữ liệu còn lại hơn 100,000 tệp, với một nửa chứa một trong 118 lỗ hổng bảo mật và nửa còn lại chứa các lựa chọn thay thế không dễ bị tấn công. Bộ dữ liệu được tạo thành từ các tệp văn bản chứa mã thô. Mỗi tệp được gắn nhãn bằng một thẻ liệt kê điểm yếu chung (CWE) cụ thể tương ứng với lỗ hổng bảo mật mà mã chứa. Mặc dù có rất nhiều ngôn ngữ trong cơ sở dữ liệu SARD, tuy nhiên bài báo chỉ kiểm tra và đào tạo các mô hình trên tập hợp mã được viết bằng C/C ++. Đối với mỗi tệp chứa lỗ hổng được CWE gắn nhãn, có một tệp phù hợp chứa mã đã được sửa và không còn lỗ hổng. Từ đó, một nửa tập dữ liệu được tạo thành từ các tệp không có lỗ hổng bảo mật và nửa còn lại được tạo thành từ các tệp trùng khớp có lỗ hổng bảo mật. Tổng cộng, có 118 loại (nhãn) CWE trong tập dữ liệu. Sau khi thêm 118 nhãn tương ứng nữa để phân loại tệp là không dễ bị tấn công, tập dữ liệu có tổng cộng 236 nhãn. Việc thêm 118 nhãn tương ứng thay vì 1 nhãn đại diện cho dữ liệu không chứa lỗ lổng để tránh sự mất cân bằng số lượng mẫu trong mỗi nhãn khi đưa vào các mô hình học sâu.

4.2. Kịch bản thực nghiệm

4.2.1. Đối với dữ liệu thử nghiệm

Tiến hành chia bộ dữ liệu thành các thành phần khác nhau, sau đó tiến hành thực nghiệm và đánh giá độ chính xác của các mô hình đề xuất. Quá trình chia tập dữ liệu thực nghiệm cho các kịch bản được chọn một cách ngẫu nhiên trong đó 75% của tập dữ liệu được dùng trong quá trình huấn luyện, 25% còn lại dùng trong quá trình kiểm tra.

4.2.2. Kịch bản đánh giá

Để đánh giá được kết quả thử nghiệm tác giả

tiến hành 3 kịch bản thực nghiệm như sau:

Kịch bản 1: Phát hiện lỗ hổng phần mềm sử dụng Word2vec kết hợp với LSTM: Thử nghiệm trên mô hình LSTM với số lượng các nút ẩn khác nhau.

Kịch bản 2: Phát hiện lỗ hổng phần mềm sử dụng Word2vec kết hợp với BiLSTM: Với số lượng các nút ẩn khác nhau.

Kịch bản 3: Phát hiện lỗ hổng phần mềm sử dụng BERT: Với số lượng layers khác nhau.

4.3. Tiêu chí đánh giá

Accuracy: Tỷ lệ giữa số lượng mẫu được phân loại đúng và tổng số mẫu. Độ chính xác được tính theo công thức sau:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \times 100\%$$

Trong đó:

TP - True positive: Số lượng mẫu có lỗ hổng được phân loại chính xác.

FN - False negative: Số lượng mẫu có lỗ hổng được phân loại là không có lỗ hổng.

TN - True negative: Số lượng mẫu không có lỗ hổng được phân loại chính xác.

FP - False positive: Số lượng mẫu không có lỗ hổng được phân loại thành có lỗ hổng.

Precision: là tỷ lệ của true positive points trên tổng số điểm được phân loại là positive (TP + FP). Độ chính xác cao có nghĩa là độ chính xác cao của việc xác định points.

$$precision = \frac{TP}{TP + FP} \times 100\%$$

Recall: là tỷ lệ của các true positive points trên tổng số các real positive points (TP + FN). Recall cao có nghĩa là TP cao và tỷ lệ bỏ sót positive points là thấp.

$$recall = \frac{TP}{TP + FN} \times 100\%$$

F1-score: là trung bình hài hòa của độ chính xác và thu hồi. F1 càng cao, bộ phân loại càng tốt.

$$F1 = \frac{2 \times precision \times recall}{precision + recall}$$

4.4. Kết quả thực nghiệm

4.4.1. Kết quả thực nghiệm kịch bản 1

Phát hiện lỗ hổng phần mềm sử dụng kiến trúc mô hình Word2vec kết hợp với LSTM trên bộ dữ liệu SARD cho kết quả theo bảng 1, độ chính xác của quá trình phân loại tiến trình cho kết quả khác nhau khi thay đổi số lượng các nút ẩn. Với số lượng các nút ẩn là 400 thì thuật toán LSTM mang lại hiệu quả cao nhất trên tất cả các độ đo. Mặc dù số lượng các nút ẩn thay đổi từ 200 đến 1000 nhưng độ chính xác của thuật toán lại không có nhiều thay đổi. Cụ thể: accuracy tăng 1.64%, precision tăng 1.82%, recall tăng 1.94% và F1 tăng 1.89%. Như vậy, số lượng các nút ẩn tăng hay giảm cũng không làm cho độ chính xác trong quá trình phân loại thay đổi nhiều. Điều này cho thấy thuật toán LSTM cung cấp khả năng phân loại tốt và ổn định khi thay đổi số lượng các nút ẩn.

Bảng 1. Kết quả thực nghiệm cho kịch bản 1

Số các nút ẩn	precision	recall	f1	accuracy
1000	71,15	67,89	69,48	71,53
400	**72,97**	**69,83**	**71,37**	**73,17**
200	71,70	68,47	70,05	72,02

4.4.2. Kết quả thực nghiệm kịch bản 2

Trong kịch bản 2, sử dụng kiến trúc mô hình Word2vec kết hợp với BiLSTM trên bộ dữ liệu SARD cho kết quả từ bảng 1 và bảng 2 thấy được khi thay đổi số lượng các các nút ẩn

thì LSTM và BiLSTM cho ra các kết quả tương tự nhau. Dựa trên kết quả kịch bản 1 và kịch bản 2 đã chứng minh được mô hình NLP có ý nghĩa đối với bài toán phân loại lỗ hổng. Điều này được thể hiện trên độ chính xác tổng quát tối thiểu đều khoảng trên 71% và đối với giá trị recall tối thiểu là 67%. Nói chung, kết quả phân loại đối với kịch bản chỉ dừng ở mức không quá tệ và tạm chấp nhận được còn kịch bản 2 thì tương đối ổn. Kịch bản 2 sử dụng BiLSTM đã mang lại kết quả tốt hơn LSTM tới 6% cho thấy tầm quan trọng của việc lưu giữ thông tin theo ngữ cảnh trên từng phân đoạn. Bộ nhớ hai chiều trong BiLSTM có thể cải thiện đáng kể hiệu suất chính xác của việc phát hiện lỗ hổng phần mềm theo cách NLP vì thông tin theo ngữ cảnh được xem xét một cách toàn diện.

Bảng 2. Kết quả thực nghiệm cho kịch bản 2

Số các nút ẩn	precision	recall	f1	accuracy
1000	78,57	75,86	77,19	78,67
400	**80,13**	**77,56**	**78,83**	**79,83**
200	79,62	77,43	78,51	79,17

4.4.3. Kết quả thực nghiệm kịch bản 3

Từ bảng 3 cho thấy số lượng layers tăng thì độ chính xác của mô hình cũng tăng theo. Tuy nhiên, việc tăng số layers làm tăng hiệu suất mô hình lên khoảng 2% nhưng đồng thời cũng tăng thời gian xử lý của mô hình lên. Trong thực tế, để cân bằng giữa hiệu suất và thời gian thì cần phải cân nhắc lựa chọn được các tham số phù hợp để cho ra được kết quả như mong đợi. Đối với kịch bản 3 sử dụng mô hình BERT với cơ chế hai chiều và attention cho kết quả phát hiện lỗ hổng tốt hơn hẳn. Điều này được thể hiện trên kết quả recall cao nhất trong 3 kịch bản là 82% tương ứng với tỷ lệ bỏ sót là 18%. Về độ chính xác, mô hình BERT và các biến thể của nó đều hoạt động tốt hơn các mô hình học sâu NLP truyền thống đã được thử nghiệm. Điều này là do mô hình BERT có thể trích xuất thông tin nhận biết ngữ cảnh theo cách tốt hơn nhiều so với LSTM hoặc RNN. Kết quả này chứng minh được mô hình đề xuất có giá trị thực tiễn đối với bài toán phát hiện lỗ hổng mã nguồn.

Bảng 3. Kết quả thực nghiệm cho kịch bản 3

Number of layers	precision	recall	f1	accuracy
4	82,89	80,97	81,92	84,08
8	83,58	81,72	82,64	84,64
12	84,41	82,63	83,51	85,33

4.4.4. Kết quả thực nghiệm kịch bản 4

Như đã trình bày ở trên, tác giả thực nghiệm kịch bản này nhằm kiểm tra và đánh giá sự hiệu quả giữa các mô hình do chúng tôi đề xuất trên cùng bộ dữ liệu thực nghiệm SARD.

Bảng 4. Kết quả thực nghiệm cho kịch bản 4

Hướng tiếp cận	precision	recall	f1	accuracy
Word2vec+ LSTM	72,97	69,83	71,37	73,17
Word2vec+ BiLSTM	80,13	77,56	78,83	79,83
Mô hình BERT	84,41	82,63	83,51	85,33

So sánh kết quả thực nghiệm qua bảng 4, tác giả thấy giữa 3 mô hình thì mô hình phân loại lỗ hổng phần mềm sử dụng mô hình Bert cho kết quả vượt trội hơn so với 2 mô hình còn lại trên tất cả các độ đo. Cụ thể Mô hình Bert cho các kết quả thực nghiệm trên các độ đo Precision; Recall; F1 score; Accuracy lần lượt là 84,41%; 82,63%; 83,51%; 85,33% hiệu quả hơn so với mô hình Word2vec + LSTM theo thứ tự lần lượt là 11,44%; 12,8%; 12,14%;

12,16% và mô hình Word2vec + BiLSTM lần lượt là 4,28%; 5,07%; 4,68%; 5,5% trên tất cả các độ đo.

5. KẾT LUẬN

Trong bài báo này, tác giả đã thiết kế một vài mô hình học sâu để phát hiện và phân loại các lỗ hổng phần mềm trong mã nguồn dựa trên tiến bộ nghiên cứu về học sâu NLP gần đây. Thử nghiệm cho thấy mô hình học sâu Transformers NLP có thể đạt được kết quả ấn tượng trong việc phát hiện và phân loại các lỗ hổng phần mềm trong mã nguồn. Hơn nữa, tác giả nhận thấy rằng điều quan trọng nhất để có thể đạt được độ chính xác cao nhất là phải duy trì thông tin theo ngữ cảnh trên toàn bộ chuỗi đầu vào. Với sự hiệu quả vượt trội trên tất cả các độ đo quan trọng như Accuracy, Precision, Recal và F1 trong bài báo đã chứng minh được sự hiệu quả của mô hình đề xuất.

TÀI LIỆU THAM KHẢO

[1] 2021 “CVE,” http://cve.mitre.org

[2] 2021 “CWE TOP25,” https://cwe.mitre.org/top25/archive/2021/2021_cwe_top25.html

[3] P. Haridas et al., "Code Characterization With Graph Convolutions and Capsule Networks," in IEEE Access, vol. 8, pp. 136307-136315 (2020).

[4] E. Hoffer and N. Ailon, “Deep metric learning using triplet network,” in International Workshop on Similarity-Based Pattern Recognition. Springer, (2015).

[5] Wang, S.; Liu, T.; Tan, L. “Automatically learning semantic features for defect prediction”. In Proceedings of the 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE), Austin, TX, USA, 14–22 May (2016).

[6] Jacob Devlin et al., “Bert: Pre-training of deep bidirectional transformers for language understanding,” October 2018. DOI:10.18653/v1/N19-1423

[7] Huan Mei et al., “Detecting vulnerabilities in IoT software: New hybrid model and comprehensive data analysis”. Journal of Information Security and Applications Volume 74, May 2023, 103467. https://doi.org/10.1016/j.cose.2022.102948

[8] Lin, G et al., “Vulnerability discovery with function representation learning from unlabeled projects”. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, Dallas, TX, USA, 30 Oct–3 Nov (2017).

[9] Guo, J., Wang, Z., Li, H. et al., “Detecting vulnerability in source code using CNN and LSTM network”. Soft Computing Volume 27, July 2021, pages 1131–1141 (2023). https://doi.org/10.1007/s00500-021-05994-w

Thông tin liên hệ: **Bùi Văn Công**

Điện thoại: 0983 978 015 - Email: bvcong@uneti.edu.vn

Khoa: Công nghệ thông tin, Trường Đại học Kinh tế - Kỹ thuật Công nghiệp.

-
-