BÁO CÁO TỔNG KẾT ĐỒ ÁN MÔN HỌC

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

**Tên chủ đề: BinDeep: A deep learning approach to binary code similarity**

**detection.**

*Mã nhóm: Nike. Mã đề tài: CK10.*

**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 Phúc Hải | 20521281 | 20521281@gm.uit.edu.vn |
| 2 | Nguyễn Xuân Quang | 20521808 | 20521808@gm.uit.edu.vn |
| 3 | Võ Thành Tín | 20522019 | [20522019@gm.uit.edu.vn](mailto:20522019@gm.uit.edu.vn) |
| 4 | Đỗ Minh Thọ | 20521972 | [20521972@gm.uit.edu.vn](mailto:20521972@gm.uit.edu.vn) |
| 5 | Trần Xuân Nam | 10521637 | 20521637@gm.uit.edu.vn |

1. **TÓM TẮT NỘI DUNG THỰC HIỆN:[[1]](#footnote-1)**
   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: ……………….

* 1. **Tên bài báo tham khảo chính:**

|  |
| --- |
| BinDeep: A deep learning approach to binary code similarity detection. |

* 1. **Dịch tên Tiếng Việt cho bài báo:**

|  |
| --- |
| BinDeep: Một cách tiếp cận deep learning để phát hiện sự tương đồng mã nhị phân. |

* 1. **Tóm tắt nội dung chính:**

|  |
| --- |
| **Bài báo tập trung vào những vấn đề sau:**   * Đề xuất một giải pháp dựa trên deep learning mới để phát hiện sự tương đồng mã nhị phân. Mô hình này sử dụng mạng hybrid siamese neural để đo độ tương đồng mã nhị phân. * Sử dụng mô hình instruction embedding để vectơ hóa các hướng dẫn được trích xuất. Áp dụng mô hình phân loại deep learning để xác định các loại chức năng cần so sánh. * Tiến hành các thử nghiệm rộng rãi để đánh giá cách tiếp cận được đề xuất. Kết quả thử nghiệm cho thấy BinDeep có thể đạt được độ chính xác trung bình, thu hồi và Điểm F1 lần lượt là 97,07%, 98,88% và 97,97% đối với BCSD. |

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

|  |
| --- |
| **Bài báo sử dụng 3 kỹ thuật cho 3 giai đoạn chính trong phương pháp đề xuất, bao gồm:**   * Kỹ thuật 01 khai thác IDA Pro để phân tích tĩnh mã nhị phân: Sau khi tháo rời các lệnh, chúng ta nhận được một chuỗi lệnh cho mỗi hàm trong mã nhị phân. Để thuận tiện cho việc sử dụng mô hình deep learning, chúng tôi sử dụng mô hình NLP để thực hiện nhúng hướng dẫn. Bằng cách đó, các chuỗi lệnh được chuyển đổi thành các vectơ. * Kỹ thuật 02 sử dụng mô hình deep learning để xác định kiến trúc CPU và mức độ tối ưu hóa của các hàm nhị phân mục tiêu: Theo các loại chức năng đã xác định, chúng tôi sẽ chọn một mô hình thích hợp để phát hiện sự tương tự mã nhị phân. Ưu điểm chính của giai đoạn này là chúng tôi có thể cải thiện khả năng phát hiện sự giống nhau một cách thích hợp. Nhờ giai đoạn này, các kịch bản so sánh khác nhau sẽ dẫn đến việc áp dụng các mô hình so sánh khác nhau trong giai đoạn tiếp theo. * Kỹ thuật 03 sử dụng mạng nơ-ron Siamese để phát hiện sự giống nhau về mã nhị phân: Có ba mô hình mạng Siamese và mỗi mô hình tương ứng với một kịch bản so sánh khác nhau. Cả ba mạng nơron Siamese này đều có cấu trúc giống nhau, nhưng các thông số mạng của chúng khác nhau. Khác với cấu trúc Siamese truyền thống, chúng tôi kết hợp mô hình CNN và LSTM để xây dựng mạng nơ-ron. Sau khi tất cả các mạng này được đào tạo tốt, chúng có thể chuyển đổi các hàm nhị phân tương tự (hoặc khác nhau) thành các vectơ tương tự (hoặc khác nhau). Bằng cách tính toán khoảng cách giữa hai hàm nhị phân, chúng ta có thể nhận được giá trị tương tự của chúng. Nếu giá trị nhỏ hơn ngưỡng được xác định trước, chúng tôi nghĩ rằng hai hàm nhị phân là tương tự. Nếu không, chúng khác nhau.     *Hình 1: Khung phát hiện sự giống nhau của BinDeep. Đầu vào cho khung là hai hàm nhị phân. Đầu ra là giá trị tương tự của hai hàm này.* |

* 1. **Môi trường thực nghiệm của bài báo:**

|  |
| --- |
| * **Cấu hình máy tính:** Thực hiện trên Máy chủ Dell T360 được trang bị hai CPU Intel Xeon E5-2603 V4, bộ nhớ 16GB, ổ cứng 2TB và một thẻ GPU NVIDIA Tesla P100 12GB. * **Các công cụ hỗ trợ sẵn có:** công cụ IDA Pro 7.0 dùng trong giai đoạn trích xuất chuỗi lệnh từ mỗi hàm nhị phân. * **Đối tượng nghiên cứu:** Các gói Linux phổ biến, bao gồm coreutils, findutils, diffutils, sg3utils và Operating-linux… Sau khi nhận được mã nguồn gói, chúng tôi sử dụng ba kiến trúc CPU (x86, x86-64 và ARM) và hai trình biên dịch (gcc và clang) với bốn mức tối ưu hóa (O0, O1, O2 và O3) để biên dịch mỗi chương trình. * **Tiêu chí đánh giá tính hiệu quả của phương pháp:** Tổng cộng, chúng tôi thu được 4729140 mẫu. Các mẫu này có thể được chia thành 5 loại: cross compiler, cross-optimization, cross-version, cross-architecture, và mixed function pairs... Để đánh giá hiệu quả của phương pháp của chúng tôi trên mã nhị phân không nhìn thấy, toàn bộ tập dữ liệu được chia thành ba tập con rời rạc để đào tạo, xác nhận và thử nghiệm. Chúng tôi đặt tỷ lệ của ba tập hợp con này là 4: 1: 1. Tất cả thông tin về biểu tượng gỡ lỗi đều bị loại bỏ trong các mẫu này. |

* 1. **Kết quả thực nghiệm của bài báo:**

|  |
| --- |
| 1. **Hiệu quả của feature processing:**      * Bảng 3 cho thấy phương pháp của chúng tôi tốt hơn so với các phương pháp chỉ sử dụng opcode làm tính năng trong các chỉ số đánh giá khác nhau, bao gồm độ chính xác, độ chính xác, thu hồi, điểm F1 và FPR. Lý do chính là toàn bộ hướng dẫn chứa nhiều thông tin hơn là opcode.  1. **Hiệu quả của embedding dimension:**        * Bảng 4 cho thấy các thước đo nhận dạng của mô hình LSTM. Khi kích thước nhúng được tăng lên, kết quả nhận dạng sẽ tốt hơn. Đối với mô hình CLSTM trong bảng 5, khi kích thước nhúng được thay đổi từ 100 thành 300, kết quả nhận dạng là tương đối ổn định. Các thử nghiệm này cho thấy mô hình CLSTM mạnh mẽ hơn mô hình LSTM về cài đặt embedding dimension.  1. **Hiệu quả của number of hidden unites:**      * Hình 6a và 6b cho thấy các kết quả về độ chính xác, độ thu hồi, điểm F1 và FPR khi sử dụng mô hình mạng nơ ron LSTM và CLSTM. Nói chung, khi số lượng hợp nhất ẩn tăng lên, độ chính xác và khả năng thu hồi của các mô hình này được tăng lên, FPR giảm. Trong mô hình LSTM, các chỉ số nhận dạng tương tự khi đặt số lượng hợp nhất ẩn thành 16, 18 và 20. Xem xét càng nhiều hợp nhất ẩn sẽ dẫn đến chi phí tính toán nhiều hơn, chúng tôi nghĩ rằng việc đặt số hợp nhất thành 16 là một thỏa hiệp tốt giữa thực dụng và hiệu quả. Trong mô hình CLSTM, khi số ẩn hợp nhất là 18, các số liệu đánh giá khác nhau là tối ưu.  1. **Hiệu quả của việc thêm mô hình phân loại:**      * Bảng 6 và bảng 7 cho thấy kết quả so sánh của mô hình mạng nơ ron LSTM và CLSTM khi mô hình phân loại được bật / tắt. Với sự trợ giúp của mô hình phân loại, độ chính xác nhận dạng, độ chính xác và điểm F1 đều được cải thiện, và FPR bị giảm trong các mô hình LSTM và CLSTM. Đặc biệt, độ chính xác, độ chính xác, thu hồi và điểm F1 của mô hình CLSTM được tăng 6,38%, 6,79%, 10,66% và 8,73%, FPR của mô hình CLSTM giảm 4,04%. Lý do chính cho hiệu quả của việc thêm mô hình phân loại là chúng ta có thể sử dụng nhiều mô hình mạng nơ-ron được nhắm mục tiêu hơn để phát hiện sự tương đồng.  1. **Hiệu quả của cấu trúc mạng neural:**      * Bảng 8 cho thấy kết quả đánh giá của các mô hình mạng nơ ron CNN, LSTM và CLSTM. Trong các mô hình này, thứ nguyên nhúng được đặt thành 300 và số lượng hợp nhất ẩn được đặt thành 18. Từ bảng này, chúng ta có thể thấy mô hình CLSTM rõ ràng có hiệu suất tốt hơn so với các mô hình CNN và LSTM.      * Bảng 9, bảng 10 và bảng 11 lần lượt cho thấy kết quả hiệu suất trên bộ dữ liệu đa kiến trúc, trình biên dịch chéo và bộ dữ liệu tối ưu hóa chéo. Tương tự, mô hình CLSTM có hiệu suất tốt hơn trong các thí nghiệm này. Đối với hiệu suất trên tập dữ liệu phiên bản chéo.      * Bảng 12 minh họa kết quả đánh giá. Trong đánh giá này, tập dữ liệu bao gồm 6 phiên bản của GNU Core Utilities, bao gồm cả phiên bản 8.31 mới nhất. Đánh giá này cũng chứng tỏ mô hình CLSTM vượt trội hơn so với mô hình CNN và LSTM. |

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

|  |
| --- |
| <liệt kê các công việc mà nhóm thực hiện cho đề tài dựa trên phân tích phương pháp/hệ thống được sử dụng trong bài báo đã tham khảo>  <liệt kê các công việc đã thực hiện+ tóm tắt kết quả của công việc này> |

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

|  |
| --- |
| * Nhóm gặp khó khăn khi thực hiện phần code demo của đồ án. Phần vì khó khăn khi tiếp cận với các kiến thức chuyên môn về deep learning và bindeep. Các kiến thức này khá mới và cần nhiều thời gian để tiếp cận làm quen cũng như hiểu rõ về bản chất. Phần vì khó khăn khi thực hiện nhiều đồ án của các môn khác nhau trong khoảng thời gian một học kỳ. |

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

|  |
| --- |
| 90% |

1. **NHẬT KÝ PHÂN CÔNG NHIỆM VỤ:**

|  |  |  |
| --- | --- | --- |
| **STT** | **Công việc** | **Phân công nhiệm vụ** |
| 1 | Làm báo cáo, slide, thuyết trình. Code phụ. | Nguyễn Phúc Hải |
| 2 | Code chính demo. | Nguyễn Xuân Quang |
| 3 | Code chính demo. | Võ Thành Tín |
| 4 | Giữ nhịp tiến độ, giúp đồ án thống nhất nội  dung giữa các thành viên trong nhóm. Tham  gia hỗ trợ các phần việc phát sinh. Code phụ. | Đỗ Minh Thọ |
| 5 | Tóm tắt bài báo. Tóm lược thông tin, chỉnh  sửa nội dung bài báo cáo. Code phụ. | Trần Xuân Nam |

BÁO CÁO TỔNG KẾT CHI TIẾT

* 1. Phương pháp thực hiện.

1. **Lời mở đầu:**

Binary code similarity detection (BCSD) đóng một vai trò quan trọng trong phân tích phần mềm độc hại và khám phá lỗ hổng. Các phương pháp hiện có chủ yếu dựa vào kiến thức về BCSD, có thể không đáng tin cậy trong một số trường hợp. Quan trọng hơn, độ chính xác phát hiện (Hoặc hiệu suất) của các phương pháp này không được thỏa mãn. Để giải quyết những vấn đề này, chúng tôi đề xuất BinDeep, một phương pháp deep learning để phát hiện sự giống nhau của mã nhị phân. Phương pháp này trước hết trích xuất chuỗi lệnh từ hàm nhị phân và sau đó sử dụng mô hình nhúng lệnh để vector hóa các tính năng của lệnh. Tiếp theo, BinDeep áp dụng mô hình deep learning Recurrent Neural Network (RNN) để xác định các loại cụ thể của hai chức năng để so sánh sau này. Theo từng loại thông tin, BinDeep chọn mô hình deep learning tương ứng để so sánh sự tương đồng. Cụ thể, BinDeep sử dụng các Siamese neural networks, kết hợp LSTM và CNN để đo lường sự giống nhau của hai chức năng mục tiêu. Khác với mô hình deep leaning truyền thống, mô hình kết hợp của chúng tôi tận dụng lợi thế của việc học cấu trúc không gian CNN và học trình tự LSTM. Đánh giá cho thấy cách tiếp cận của chúng tôi có thể đạt được BCSD tốt giữa cross-architecture, cross-compiler, cross-optimization và cross-version.

1. **Đặt vấn đề:**

Với sự phát triển nhanh chóng của công nghệ thông tin, một số lượng lớn phần mềm được sử dụng rộng rãi trong máy tính cá nhân và thiết bị IOT với các kiến trúc CPU khác nhau. Do đó, ngày càng có nhiều người chú ý đến bảo mật phần mềm. Các nhà phát triển độc hại có thể chèn mã độc hại vào phần mềm của họ. Chẳng hạn, một số nhà cung cấp khai thác kỹ thuật phần mềm độc hại để quảng cáo phần mềm thương mại. Ngay cả khi phần mềm được phát triển bởi một nhà phát triển đáng tin cậy, nó vẫn có thể chứa một hoặc nhiều lỗ hổng bảo mật do lỗi lập trình.

Để giải quyết những vấn đề này, một số phương pháp phân tích mã nhị phân đã được đề xuất trong những năm gần đây. Trong bài báo này, chúng tôi tập trung vào phương pháp binary code similarity detection (BCSD), rất hữu ích cho việc phân tích phần mềm độc hại và khám phá lỗ hổng. Bằng cách tiến hành so sánh sự giống nhau với các hàm nhị phân đã biết, chúng ta có thể xác định các lỗ hổng tương ứng (Hoặc các hàm độc hại) trong các mã nhị phân khác nhau.

Cùng một mã nguồn có thể được biên dịch với các trình biên dịch khác nhau và các mức tối ưu hóa khác nhau, nhắm vào các kiến trúc CPU khác nhau. Do đó, mã được biên dịch đích sẽ khác nhau về cross-compiler, cross-optimization, và cross-architecture binaries. Do sự khác biệt về cú pháp, rất khó để nhận ra các chức năng tương tự giữa các mã nhị phân khác nhau.

Các giải pháp trước đây để giải quyết vấn đề binary code similarity có thể được chia thành hai loại: Phân tích tĩnh và phân tích động. Nói chung, các giải pháp này gặp phải những hạn chế sau:

• Hầu hết các phương pháp hiện có chủ yếu dựa vào các đặc điểm cú pháp được thiết kế (Ví dụ: Biểu đồ luồng điều khiển) của mã nhị phân để so sánh độ tương tự.

• Một số phương pháp phát sinh chi phí hiệu suất đáng kể do để áp dụng các cơ chế tốn thời gian (Ví dụ: Khớp đồ thị và mô phỏng).

• Một số phương pháp không thể so sánh tốt độ tương tự của mã nhị phân giữa các kiến trúc CPU khác nhau và các phiên bản chương trình khác nhau.

Để giải quyết những vấn đề này, trong đồ án này, chúng tôi trình bày một binary similarity detection framework mới, BinDeep, bằng cách sử dụng các kỹ thuật deep learning. Khác với các giải pháp trước đây, phương pháp của chúng tôi không cần bất kỳ tính năng được chọn thủ công nào để tính toán độ tương tự.

Ý tưởng cơ bản của cách tiếp cận của chúng tôi là tận dụng siamese neural network để đo lường sự giống nhau của các hàm nhị phân. Để có được đầu vào cho neural network, trước tiên chúng tôi tách mã nhị phân và trích xuất chuỗi lệnh dưới dạng các tính năng. Tiếp theo, chúng tôi sử dụng mô hình classical natural language processing để chuyển đổi các chuỗi hướng dẫn thành các vectơ. Xem xét các kịch bản so sánh khác nhau, chúng tôi sử dụng siamese neural network khác nhau để đo lường độ tương tự. Với mục đích này, chúng tôi áp dụng mô hình deep learning để xác định các loại cụ thể của hai chức năng được so sánh. Không giống như cấu trúc mạng Siamese thông thường, chúng tôi sử dụng cấu trúc mạng lai, kết hợp mạng CNN và LSTM để đo độ tương tự của hàm nhị phân. Vì CNN có thể trích xuất các đặc điểm không gian cục bộ trong khi LSTM có khả năng tự động trích xuất các đặc điểm tuần tự, nên mô hình hybrid neural network có thể cải thiện khả năng phát hiện tương tự.

Chúng tôi đã triển khai phương pháp của mình dựa trên IDA Pro. Để đánh giá tính hiệu quả của phương pháp, chúng tôi chuẩn bị một tập dữ liệu tùy chỉnh, bao gồm hơn 47 triệu cặp hàm. Các thử nghiệm cho thấy rằng giải pháp của chúng tôi có thể xác định các cặp chức năng tương tự và không giống nhau một cách hiệu quả trên cross-architecture, cross-compiler, crossoptimization, và cross-version binaries.

**3. Mục tiêu:**

Mục tiêu của đồ án này trong thời gian học tập môn lập trình an toàn và khai thác lỗ hổng phần mềm có thể tìm hiểu và nghiên cứu được một giải pháp đang được phát triển trong những năm gần đây về deep learning để từ đó có thể hiểu rõ và áp dụng được nó trong việc phát hiện chính xác các sự tương đồng về mã nhị phân.

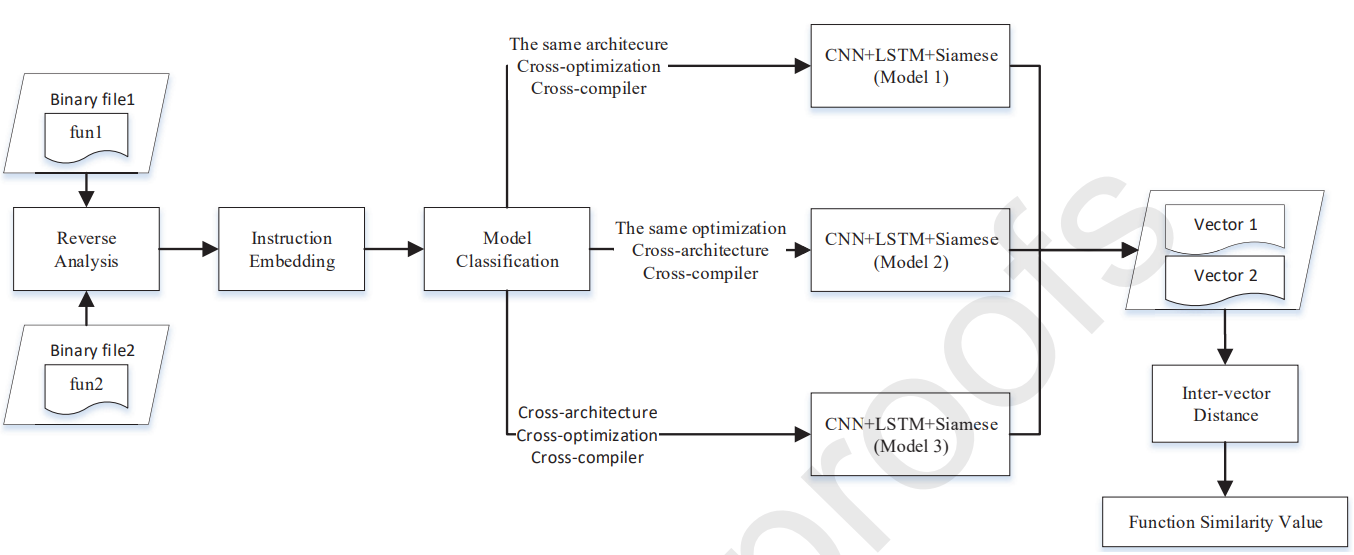
Hoàn thành tốt học phần, cũng cố thêm các kiên thức về bảo mật mạng và chuẩn bị cho các học kỳ sắp tới. Bài báo cáo tập trung vào những vấn đề sau:

• Đề xuất một giải pháp dựa trên deep learning mới để phát hiện sự tương đồng mã nhị phân. Mô hình này sử dụng mạng hybrid siamese neural để đo độ tương đồng mã nhị phân.

• Sử dụng mô hình instruction embedding để vectơ hóa các hướng dẫn được trích xuất. Áp dụng mô hình phân loại deep learning để xác định các loại chức năng cần so sánh.

• Tiến hành các thử nghiệm rộng rãi để đánh giá cách tiếp cận được đề xuất. Kết quả thử nghiệm cho thấy BinDeep có thể đạt được độ chính xác trung bình, thu hồi và Điểm F1 lần lượt là 97,07%, 98,88% và 97,97% đối với BCSD.

**4. Framework của BinDeep:**



*Hình 1: Framework Similarity Detection của BinDeep. Đầu vào của khung là hai hàm nhị phân. Đầu ra là giá trị tương tự của hai hàm này.*

Như mô tả trong Hình 1, Framework của BinDeep có thể được chia thành ba giai đoạn. Ứng với các kỹ thuật được sử dụng như sau:

**Kỹ thuật 01:** Khai thác IDA Pro để phân tích tĩnh mã nhị phân: Sau khi tháo rời các lệnh, chúng ta nhận được một chuỗi lệnh cho mỗi hàm trong mã nhị phân. Để thuận tiện cho việc sử dụng mô hình deep learning, chúng tôi sử dụng mô hình NLP để thực hiện nhúng hướng dẫn. Bằng cách đó, các chuỗi lệnh được chuyển đổi thành các vectơ.

**Kỹ thuật 02:** Sử dụng mô hình deep learning để xác định kiến trúc CPU và mức độ tối ưu hóa của các hàm nhị phân mục tiêu: Theo các loại chức năng đã xác định, chúng tôi sẽ chọn một mô hình thích hợp để phát hiện sự tương tự mã nhị phân. Ưu điểm chính của giai đoạn này là chúng tôi có thể cải thiện khả năng phát hiện sự giống nhau một cách thích hợp. Nhờ giai đoạn này, các kịch bản so sánh khác nhau sẽ dẫn đến việc áp dụng các mô hình so sánh khác nhau trong giai đoạn tiếp theo.

**Kỹ thuật 03:** Sử dụng mạng nơ-ron Siamese để phát hiện sự giống nhau về mã nhị phân: Có ba mô hình mạng Siamese và mỗi mô hình tương ứng với một kịch bản so sánh khác nhau. Cả ba mạng nơron Siamese này đều có cấu trúc giống nhau, nhưng các thông số mạng của chúng khác nhau. Khác với cấu trúc Siamese truyền thống, chúng tôi kết hợp mô hình CNN và LSTM để xây dựng mạng nơ-ron. Sau khi tất cả các mạng này được đào tạo tốt, chúng có thể chuyển đổi các hàm nhị phân tương tự (hoặc khác nhau) thành các vectơ tương tự (hoặc khác nhau). Bằng cách tính toán khoảng cách giữa hai hàm nhị phân, chúng ta có thể nhận được giá trị tương tự của chúng. Nếu giá trị nhỏ hơn ngưỡng được xác định trước, chúng tôi nghĩ rằng hai hàm nhị phân là tương tự. Nếu không, chúng khác nhau.

**5. Khai thác và sử lý tính năng:**

Trong các cách tiếp cận trước đây, hầu hết chúng đều dựa vào kiến thức có sẵn để trích xuất đặc trưng. Trong giải pháp của chúng tôi, chúng tôi chỉ sử dụng trình tự hướng dẫn làm tính năng. Đối với giao dịch này, chúng tôi khai thác IDA Pro để phân tách mã nhị phân và sau đó nhận chuỗi hướng dẫn cho từng chức năng. Để đơn giản, luồng kiểm soát nội bộ của một chức năng không được xem xét. Tương tự như các nghiên cứu gần đây, chúng tôi sử dụng mô hình NLP (Xử lý ngôn ngữ tự nhiên) để xây dựng cách nhúng hướng dẫn của mình. Nói chung, một lệnh có thể được chia thành hai phần: Một mã lệnh và một (hoặc nhiều) toán hạng. Số lượng loại opcode bị hạn chế trong khi biểu diễn của toán hạng thay đổi rất nhiều trong các bối cảnh điện toán khác nhau. Để giải quyết vấn đề này, một phương pháp đơn giản là chỉ sử dụng opcode lệnh làm mã thông báo để nhúng lệnh, bỏ qua các toán hạng lệnh. Tuy nhiên, làm như vậy sẽ dẫn đến mất thông tin. Trong thực tế, toán hạng hướng dẫn chứa thông tin ngữ nghĩa quan trọng để so sánh tương tự của các mã nhị phân.

Để giữ thông tin toán hạng, quá trình chuẩn hóa là cần thiết. Khác với phương pháp gần đây, chúng tôi đề xuất một cách đơn giản nhưng hiệu quả để chuẩn hóa toán hạng lệnh. Cụ thể, chúng tôi phân loại các toán hạng phổ biến thành 8 loại khác nhau:

Ảnh có chứa bàn

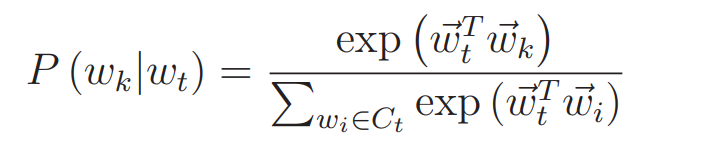
Mô tả được tạo tự động

*Bảng 1: Chuẩn hóa hướng dẫn.*

General Register, Direct Memory Reference, Memory Ref [Base Reg + Index Reg], Memory Reg [Base Reg + Index Reg + Displacement], Immediate Value, Immediate Far Address, Immediate Near Address và Other Type. Bảng 1 cho thấy các ví dụ về chuẩn hóa lệnh cho kiến trúc x86.

Sau khi các lệnh thô được chuẩn hóa, bước tiếp theo là thực hiện nhúng lệnh. Có hai phương pháp phổ biến để nhúng: One-hot encoding và word2vec. Mã hóa one-hot rất đơn giản để biểu diễn một lệnh, nhưng nó không thể nắm bắt được sự liên quan của hai lệnh giống nhau. Ngược lại, phương thức word2vec có khả năng chuyển đổi các hướng dẫn tương tự thành các vectơ tương tự. Ví dụ: Bằng cách sử dụng word2vec, các lệnh thêm và phụ sẽ được chuyển đổi thành các vectơ tương tự. Word2vec chứa hai mô hình khác nhau: CBOW (Túi từ liên tục) và Skip-gram. So với mô hình CBOW, mô hình Skip-gram có thể đạt được hiệu suất tốt hơn trên một tập dữ liệu lớn. Do đó, chúng tôi sử dụng mô hình Skip-gram để xây dựng nhúng hướng dẫn của chúng tôi.

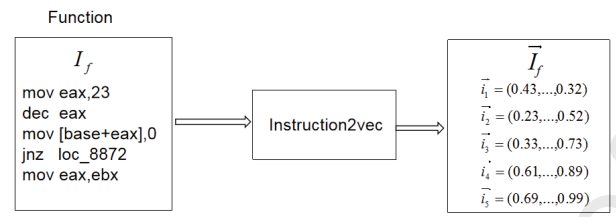
Ý tưởng cơ bản của mô hình Skip-gram là sử dụng thông tin ngữ cảnh để tìm hiểu cách nhúng từ trên luồng văn bản. Đối với mỗi từ, mô hình ban đầu sẽ thiết lập một vectơ mã hóa one-hot, sau đó nó sẽ được đào tạo khi đi qua từng sliding window. Điểm mấu chốt của mô hình là để tìm ra xác suất P của một từ tùy ý wk 1 trong sliding window Ct 2 được nhúng ~wt 3 của từ hiện tại wt. Với mục đích này, chức năng softmax được sử dụng như sau:



Trong đó ~wk và ~wi là phần nhúng của các từ wk và wi , ~w T t ~wi là phần giống nhau của hai từ wt và wi . Để đào tạo mô hình trên một chuỗi các từ T, chúng tôi sử dụng hệ số giảm dần độ dốc ngẫu nhiên để giảm thiểu hàm mục tiêu khả năng log J (w) được xác định như sau:

Ảnh có chứa văn bản, đồng hồ

Mô tả được tạo tự động



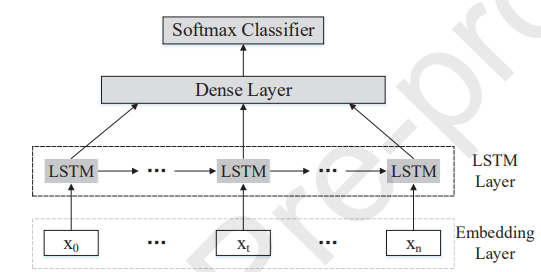
*Hình 2: Quá trình chuyển đổi hướng dẫn assembly thành các vectơ.*

Hình 2 minh họa các ví dụ về nhúng lệnh. Đầu vào của mô hình này là một lệnh, đầu ra là một vectơ 300 chiều. Nói cách khác, chuỗi lệnh sẽ được biểu diễn dưới dạng nhiều vectơ 300 chiều. Phần 3.4 cho thấy tác động của các kích thước nhúng khác nhau đối với các thử nghiệm.

**6. Phân loại mô hình:**

Sau khi chuỗi lệnh của các chức năng được vector hóa, bước tiếp theo là training mô hình so sánh dựa trên deep learning để có thể đo lường tốt sự giống nhau của hai chức năng. Các phương pháp trước đây chỉ sử dụng một mô hình deep learning duy nhất để phát hiện sự tương đồng giữa cross-architecture, cross-compiler, cross-optimization và cross-version binary code. Xem xét các mục tiêu so sánh không đồng nhất, việc sử dụng một mô hình deep learning có thể không đạt được độ chính xác phát hiện tốt trong một số trường hợp. Để giải quyết vấn đề này, chúng tôi áp dụng các mô hình deep learning khác nhau cho các kịch bản so sánh khác nhau. Nói cách khác, chúng tôi sử dụng một mô hình phù hợp cho từng tình huống so sánh. Cụ thể, chúng tôi phân loại các kịch bản so sánh thành ba loại: Các kiến trúc CPU giống nhau với cách biên dịch khác nhau, cùng mức độ tối ưu hóa với các kiến trúc CPU khác nhau và các kiến trúc CPU khác nhau với cách biên dịch và tối ưu hóa khác nhau. Theo đó, chúng tôi sử dụng ba mô hình mạng Siamese neural network khác nhau để so sánh sự giống nhau này. Như đã đề cập trước đây, các mô hình này có cấu trúc mạng giống nhau.

Để chọn một mô hình neural network phù hợp, chúng ta cần xác định các loại cụ thể của hai chức năng được so sánh. Với mục đích này, chúng tôi sử dụng bộ phân loại dựa trên RNN. Bộ phân loại bao gồm hai mô hình: Một mô hình để xác định kiến trúc CPU (x86/x64/arm) và mô hình còn lại để xác định mức độ tối ưu hóa (O0/O1/O2/O3). Trong mỗi lớp ẩn của mô hình RNN, đầu ra của thời điểm tiếp theo được xác định bởi đầu ra của thời điểm hiện tại và đầu vào của thời điểm tiếp theo. Nhờ thiết kế tái chế này, mô hình RNN có thể xử lý tốt các thông tin tuần tự như câu trong văn bản. Vì các chuỗi hướng dẫn tương tự như các câu nên mô hình RNN phù hợp với nhiệm vụ phân loại để xác định kiến trúc CPU và mức độ tối ưu hóa. Mặc dù mô hình RNN mạnh mẽ để xử lý dữ liệu tuần tự, nhưng rất khó để huấn luyện mô hình này do khuếch tán gradient lan truyền ngược hoặc bùng nổ gradient. Để giải quyết vấn đề này, chúng tôi sử dụng mô hình LSTM.



*Hình 3: Kiến trúc Neural network để phân loại mô hình. Đầu vào của mạng là một hàm nhị phân. Đầu ra là loại nhận dạng của chức năng này.*

Hình 3 minh họa kiến trúc của Neural network trên LSTM của chúng tôi để nhận dạng mức tối ưu hóa biên dịch. Neural network này bao gồm ba lớp. Lớp đầu tiên là lớp embedding. Bằng cách sử dụng lại các trọng số được đào tạo trước từ mô hình word2vec, lớp này có thể ánh xạ chuỗi lệnh thành các vectơ 300 chiều. Đầu vào của lớp này là các vectơ nhúng lệnh ~I = ~i1, ...~in,~ij ∈ R, 1 6 j 6 n, n = 1000 và đầu ra của lớp này là một lệnh nhúng khác vectơ ~l = ~l1, ..., ~lm , ~lj ∈ RL , 1 6 j 6 m, m = 1000, L = 300. Lớp thứ hai là lớp LSTM và kích thước đầu ra của nó là 50. Mỗi đơn vị LSTM chứa các phương trình sau:

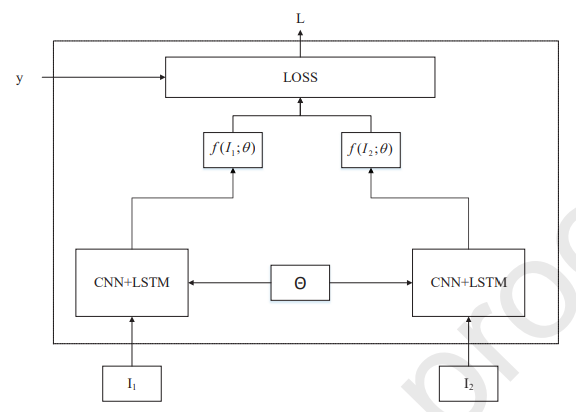
Ảnh có chứa văn bản

Mô tả được tạo tự động

Trong các phương trình trên, ct ,it , ft , ot biểu thị trạng thái bộ nhớ, cổng vào, quên, cổng ra tại thời điểm t ∈ {1, . . . Toán tử , T}, ⊙ là phép nhân theo điểm, W, U là ma trận trọng số và b là tham số sai lệch.

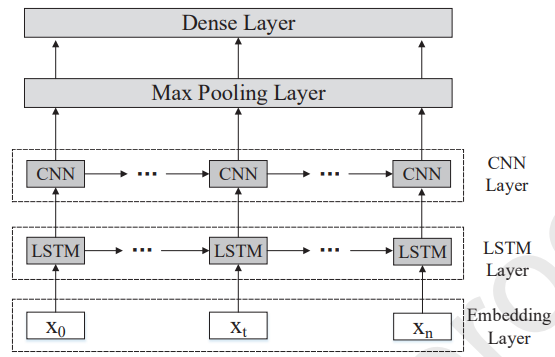
Lớp cuối cùng là lớp Desend output dày đặc với bốn đơn vị neural và đầu ra của nó là kết quả phân loại. Để tạo điều kiện cho nhiều phân loại, chúng tôi sử dụng softmax làm chức năng kích hoạt. Để training mạng neural này, chúng tôi sử dụng crossentropy phân loại làm hàm mất mát và áp dụng adam làm trình tối ưu hóa. Để ngăn mạng neural được đào tạo trở nên quá mức, chúng tôi sử dụng dropout và đặt tốc độ của nó thành 0,5.

**7. So sánh sự tương đồng.**



*Hình 4: Kiến trúc mạng Siamese để so sánh sự tương đồng. Đầu vào của mạng là hai hàm nhị phân. Đầu ra là kết quả so sánh tương tự.*

Sau khi xác định được kiểu của hai hàm nhị phân, chúng ta chọn mô hình mạng neural tương ứng để tính toán độ tương tự. Được thúc đẩy bởi phương pháp gần đây để đánh giá sự giống nhau về ngữ nghĩa giữa các câu, chúng tôi tận dụng mạng Siamese để đo lường sự giống nhau của hai hàm nhị phân. Ý tưởng cơ bản của mạng Siamese là sử dụng hai chức năng ánh xạ giống hệt nhau để chuyển đổi hai đầu vào thành hai vectơ chiều cố định. Bằng cách so sánh khoảng cách của hai vectơ này, chúng ta có thể tìm ra sự giống nhau của hai đầu vào ban đầu. Như được hiển thị trong Hình 4, kiến trúc mạng Siamese bao gồm hai mạng lưới neural nhúng giống hệt nhau. Khác với các phương pháp tiêu chuẩn, chúng tôi kết hợp các mô hình CNN và LSTM để xây dựng mạng lưới neural. Mô hình CNN rất tốt trong việc học cấu trúc không gian, trong khi mô hình LSTM vượt trội trong việc học trình tự. Mô hình kết hợp của chúng tôi tận dụng cả hai ưu điểm của hai mô hình này để có thể cải thiện khả năng phát hiện sự tương đồng dựa trên deep learning. Để dễ trình bày, chúng tôi sử dụng thuật ngữ mô hình lai và mô hình CLSTM (tức là CNN+LSTM) thay thế cho nhau.



*Hình 5: Kiến trúc mạng Hybrid neural.*

Hình 5 minh họa cấu trúc của mạng nơ-ron lai. Nó bao gồm 5 lớp. Lớp embedding đầu tiên được sử dụng để ánh xạ chuỗi lệnh thành một vectơ có chiều cố định. Lớp này có thể được đại diện bởi một hàm ánh xạ f : X → X~ , trong đó X là vectơ chỉ dẫn và X~ là vectơ ánh xạ. X = (x1, ..., x1000), xj ∈ R, 1 6 j 6 1000; X~ = (~x1, ..., ~x1000), ~xj ∈ RL , 1 6 j 6 1000, L = 300. Lớp này sử dụng lại các trọng số được đào tạo trước từ mô hình word2vec. Lớp thứ hai là lớp LSTM và kích thước đầu ra của nó là 20. Đầu ra của lớp này có thể được biểu thị như sau:

Ảnh có chứa văn bản

Mô tả được tạo tự động

Trong đó σ là hàm ReLU, Conv đại diện cho hoạt động tích chập, W2 và b2 là các tham số trọng số và độ lệch của các bộ lọc tích chập. A = (a1, ..., am), ai ∈ RL , 1 6 i 6 m, L = 996, m = 300. Lớp thứ tư là Max pooling layer. Nó được sử dụng để đơn giản hóa các tính năng trích xuất. Kích thước của các cửa sổ gộp tối đa được đặt thành 2 và sải chân cũng được đặt thành 2. Để đơn giản, thao tác gộp tối đa có thể được biểu diễn dưới dạng biểu thức: Ae = M ax(A). Ae = (ae1, ..., afm), aei ∈ RL , 1 6 i 6 m, L = 498, m = 300. Lớp cuối cùng là dense layer, có thể giúp kết nối các đặc trưng cục bộ. Đầu ra của nó có thể được biểu diễn như sau:

Ảnh có chứa văn bản, đồng hồ, thiết bị đo

Mô tả được tạo tự động

Trong đó · đại diện cho hoạt động sản phẩm chấm, W3 và b3 là các tham số trọng số và độ lệch của lớp này. O = (o1, ..., om), oi ∈ R, 1 6 i 6 m, m = 300.

Đầu vào của Siamese network là hai hàm nhị phân, đó là I1 và I2. Hai chức năng này có thể được biên dịch từ các kiến trúc CPU, trình biên dịch, mức độ tối ưu hóa và phiên bản chương trình khác nhau. Độ dài đầu vào được đặt thành 1000. Nếu hàm chứa ít hơn 1000 lệnh, chúng tôi sẽ sử dụng lệnh nop làm phần đệm. Ngược lại, nếu hàm chứa hơn 1000 lệnh, chúng ta sẽ cắt bớt các lệnh đuôi. Đầu ra của các lớp embedding của Siamese network là hai vectơ nhúng, cụ thể là f(I1, θ) và f(I2, θ), trong đó f 5 biểu thị cấu trúc mạng kết hợp và θ biểu thị các tham số của cấu trúc mạng này. Chúng tôi giả sử kích thước nhúng là m. Ngoài ra, có một đầu vào chỉ báo y cho Siamese network, cho biết liệu hai đầu vào có giống nhau hay không. Cụ thể, nếu y bằng 1 chứng tỏ hai hàm nhị phân đồng dạng, nếu y bằng 0 chứng tỏ hai hàm nhị phân không đồng dạng.

Để xác định hàm mất mát của mạng, chúng tôi tận dụng hàm mất mát tương phản. Ý tưởng cơ bản của hàm mất mát này là tối đa hóa khoảng cách giữa hai đầu vào khác nhau, nhưng để giảm thiểu khoảng cách giữa hai đầu vào tương tự. Với mục đích này, hàm mất mát được định nghĩa như sau:



Ảnh có chứa văn bản

Mô tả được tạo tự động

Trong đó D(I1, I2) biểu thị khoảng cách Manhattan giữa hai hàm nhị phân.Training Siamese network là tìm các tham số θ để cực tiểu hóa hàm mất mát. Để đạt được điều này, chúng tôi sử dụng Adam với thuật toán lan truyền ngược tiêu chuẩn.

Sau khi Siamese network được đào tạo tốt, chúng ta có thể suy ra hai tuyên bố. Khi hai hàm nhị phân tương tự nhau (Nghĩa là y=1), khoảng cách Manhattan của chúng phải gần bằng 0 để giá trị tổn thất sẽ nhỏ nhất. Khi hai hàm nhị phân không giống nhau (Nghĩa là y=0), khoảng cách Manhattan của chúng phải gần bằng one và giá trị hàm mất mát vẫn sẽ nhỏ nhất.

* 1. Chi tiết cài đặt, hiện thực.

**1. Thiết lập cấu hình:**

Các thử nghiệm của chúng tôi được thực hiện trên Máy chủ Dell T360 được trang bị hai CPU Intel Xeon E5-2603 V4, bộ nhớ 16GB, ổ cứng 2TB và một card GPU NVIDIA Tesla P100 12GB. Chúng tôi triển khai phần bổ trợ của công cụ IDA Pro 7.0 để trích xuất một chuỗi lệnh từ mỗi hàm nhị phân. Các mô hình mạng này được triển khai trong TensorFlow-1.8 và Keras-2.2. Tất cả các mạng này được đào tạo tốt trong vòng 50 kỷ nguyên.

**2. Thiết lập dữ liệu:**

Để chuẩn bị tập dữ liệu, chúng tôi chọn 6 gói Linux phổ biến, bao gồm coreutils, findutils, diffutils, sg3utils và util-linux,… Sau khi nhận được mã nguồn của gói, chúng tôi sử dụng ba kiến trúc CPU (x86, x86-64 và ARM) và hai trình biên dịch (gcc và clang) với bốn mức tối ưu hóa (O0, O1, O2 và O3) để biên dịch từng chương trình. Đối với kiến trúc x86 và x86-64, trình biên dịch được phép sử dụng tập lệnh mở rộng (Ví dụ: MMX và SSE). Nếu hai hàm nhị phân được biên dịch từ cùng một mã nguồn, chúng sẽ khớp với nhau. Nếu không, chúng không được khớp.

Để tạo thuận lợi cho supervised learning, tỷ lệ của các mẫu dương và âm (Nghĩa là các cặp hàm khớp và chưa khớp) là tương đối cân đối. Để có được các mẫu đào tạo tích cực, chúng tôi sử dụng thông tin chưa bị xóa trong mã nhị phân để xác định các hàm phù hợp trong các tệp được biên dịch khác nhau. Để lấy các mẫu đào tạo phủ định, chúng tôi chọn ngẫu nhiên các hàm trong các tệp nhị phân khác nhau với các tên hàm khác nhau. Nhãn cho các mẫu dương tính và mẫu âm tính là các số 1 và 0.

Ảnh có chứa bàn

Mô tả được tạo tự động

*Bảng 2: Mô tả các loại samples.*

Tổng cộng, chúng tôi thu được 4729140 mẫu. Như thể hiện trong Bảng 2, các mẫu này có thể được chia thành 5 loại: Cross-compiler, cross-optimization, cross-version, cross-architecture, và mixed function pairs. Để đánh giá tính hiệu quả của phương pháp của chúng tôi đối với mã nhị phân không nhìn thấy, toàn bộ tập dữ liệu được chia thành ba tập hợp con riêng biệt để đào tạo, xác thực và thử nghiệm. Chúng tôi đặt tỷ lệ của ba tập hợp con này thành 4:1:1. Tất cả thông tin về biểu tượng gỡ lỗi đều bị loại bỏ trong các mẫu này.

* 1. Kết quả thực nghiệm.

**1. Xây dựng công thức đánh giá:**

Để đánh giá hiệu suất của phương pháp, chúng tôi sử dụng các chỉ số tiêu chuẩn: độ chính xác, độ chính xác, thu hồi, F1 và TPR, được định nghĩa như sau:

Ảnh có chứa văn bản

Mô tả được tạo tự động

Trong các công thức trên, The True positive (TP) biểu thị số cặp hàm khớp được xác định chính xác. Kết quả Dương tính Sai (FP) đề cập đến số cặp chức năng được xác định sai khi mô hình deep learning xác định các cặp chức năng chưa khớp là khớp. True Negative (TN) đại diện cho số lượng các cặp chức năng chưa khớp được xác định chính xác. Phủ định sai (FN) đề cập đến số cặp chức năng chưa khớp được xác định sai. Độ chính xác đề cập đến tỷ lệ phần trăm của các cặp chức năng được xác định chính xác. Độ chính xác đo tỷ lệ phần trăm của các cặp chức năng phù hợp được dán nhãn chính xác. Recall thể hiện khả năng xác định chính xác các cặp chức năng phù hợp. FPR đo tỷ lệ phần trăm của các cặp chức năng chưa khớp được gắn nhãn không chính xác là các cặp chức năng đã khớp. Điểm F1 đề cập đến giá trị trung bình hài hòa của Độ chính xác và Độ thu hồi.

**2. Kết quả thử nghiệm:**

**a. Hiệu quả của việc xử lý tính năng.**

Ảnh có chứa văn bản, bàn

Mô tả được tạo tự động

*Bảng 3: Kết quả theo các tính năng hướng dẫn khác nhau.*

Nói chung, chúng tôi sử dụng trình tự hướng dẫn làm tính năng. Một lệnh bao gồm một opcode và một (Hoặc nhiều) toán hạng. Một số phương pháp sử dụng opcode làm tính năng, bỏ qua các toán hạng, trong khi phương pháp của chúng tôi sử dụng toàn bộ hướng dẫn làm tính năng. Để so sánh hiệu quả của việc xử lý tính năng khác nhau đối với việc xác định tính tương tự, chúng tôi tiến hành các thử nghiệm tương ứng. Như được hiển thị trong Bảng 3, phương pháp của chúng tôi tốt hơn các phương pháp chỉ sử dụng opcode làm tính năng trong các chỉ số đánh giá khác nhau, bao gồm độ chính xác, độ chính xác, khả năng thu hồi, điểm F1 và FPR. Lý do chính là toàn bộ hướng dẫn chứa nhiều thông tin hơn opcode.

**b. Hiệu quả của embedding dimension.**

Ảnh có chứa bàn

Mô tả được tạo tự động

*Bảng 4: Kết quả theo các embedding dimension khác nhau trong mô hình LSTM.*

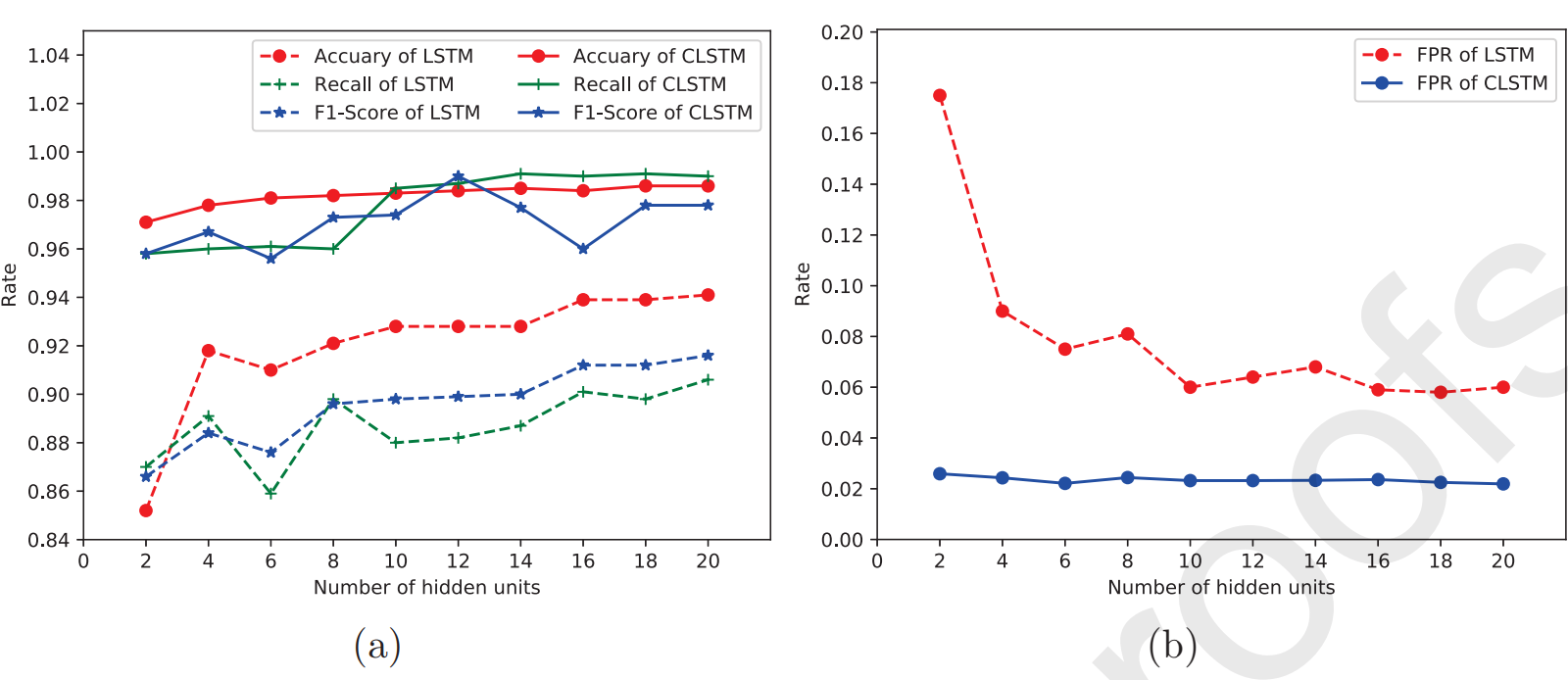
Ảnh có chứa bàn

Mô tả được tạo tự động

*Bảng 5: Kết quả theo các embedding dimension khác nhau trong mô hình CLSTM.*

Trong phần này, chúng tôi khám phá tác động của embedding dimension với kết quả nhận dạng. Với mục đích này, chúng tôi phân tích hiệu suất của các mô hình mạng neural LSTM và CLSTM cho các cặp chức năng hỗn hợp có kích thước nhúng khác nhau. Bảng 4 cho thấy các số liệu nhận dạng của mô hình LSTM. Khi embedding dimension tăng lên, kết quả nhận dạng sẽ tốt hơn. Đối với mô hình CLSTM được trình bày trong Bảng 5, khi embedding dimension được thay đổi từ 100 thành 300, kết quả nhận dạng tương đối ổn định. Các thử nghiệm này cho thấy mô hình CLSTM mạnh mẽ hơn mô hình LSTM trên cài đặt embedding dimension.

**c. Hiệu quả của number of hidden unites.**



*Hình 6: Kết quả dưới các đơn vị ẩn khác nhau trong mô hình LSTM và CLSTM.*

Để kiểm tra xem số lượng liên kết ẩn có ảnh hưởng đến kết quả nhận dạng hay không, chúng tôi thực hiện các thử nghiệm với số lượng liên kết ẩn khác nhau trong mạng neural từ 2 đến 20. Hình 6a và 6b hiển thị kết quả về độ chính xác, khả năng thu hồi, điểm F1 và FPR khi sử dụng mô hình mạng neural LSTM và CLSTM. Nói chung, khi số lượng liên kết ẩn tăng lên, độ chính xác và khả năng thu hồi của các mô hình này tăng lên, FPR giảm xuống. Trong mô hình LSTM, các số liệu nhận dạng tương tự khi đặt số liên kết ẩn thành 16, 18 và 20. Xem xét càng nhiều liên kết ẩn sẽ dẫn đến chi phí tính toán cao hơn, chúng tôi nghĩ rằng việc đặt số liên kết thành 16 là một sự thỏa hiệp tốt giữa thực dụng và hiệu quả. Trong mô hình CLSTM, khi số lượng liên kết ẩn là 18, các chỉ số đánh giá khác nhau là tối ưu.

**d. Tác dụng của việc thêm mô hình phân loại.**

Ảnh có chứa bàn

Mô tả được tạo tự động

*Bảng 6: Kết quả LSTM và mô hình phân loại + LSTM.*

Ảnh có chứa bàn

Mô tả được tạo tự động

*Bảng 7: Kết quả của CLSTM và mô hình phân loại + CLSTM.*

Các phương pháp trước đây chỉ sử dụng một mô hình neural network duy nhất để đo lường sự giống nhau của hai hàm. Khác với các phương pháp này, trước tiên chúng tôi sử dụng mô hình phân loại dựa trên LSTM để xác định các loại chức năng và sau đó chọn mô hình neural network phù hợp để đo lường độ tương tự. Để chứng minh tính hiệu quả của mô hình phân loại, chúng tôi tiến hành một bộ kiểm định. Bảng 6 và Bảng 7 cho thấy kết quả so sánh của mô hình neural network LSTM và CLSTM khi bật/tắt mô hình phân loại. Với sự trợ giúp của mô hình phân loại, độ chính xác nhận dạng, độ chính xác, thu hồi và điểm F1 đều được cải thiện và FPR giảm trong các mẫu LSTM và CLSTM. Trong đó, độ chính xác, độ chính xác, độ thu hồi và điểm F1 của mẫu CLSTM lần lượt tăng lên lần lượt là 6,38%, 6,79%, 10,66% và 8,73%, FPR của mẫu CLSTM giảm 4,04%. Lý do chính cho hiệu quả của việc thêm mô hình phân loại là chúng ta có thể sử dụng mô hình neural network được nhắm mục tiêu nhiều hơn để phát hiện sự tương đồng.

**e. Hiệu quả của cấu trúc mạng neural network.**

Lý do chính cho hiệu quả của việc thêm mô hình phân loại là chúng ta có thể sử dụng mô hình neural network được nhắm mục tiêu nhiều hơn để phát hiện sự tương đồng.Ảnh có chứa bàn

Mô tả được tạo tự động

*Bảng 8: Kết quả so sánh các mô hình LSTM, CNN và CLSTM trên bộ dữ liệu hỗn hợp.*

Để khám phá tác động của cấu trúc mạng đối với phép đo độ tương tự, chúng tôi thực hiện một bộ thử nghiệm trong các tình huống khác nhau. Như đã đề cập trước đây, tập dữ liệu có thể được chia thành 5 loại khác nhau, tương ứng với các kịch bản so sánh khác nhau. Về hiệu suất trên tập dữ liệu hỗn hợp, Bảng 8 cho thấy kết quả đánh giá của các mô hình neural network CNN, LSTM và CLSTM. Trong các mô hình này, embedding dimension được đặt thành 300 và số lượng liên kết ẩn được đặt thành 18. Từ bảng này, chúng ta có thể thấy mô hình CLSTM rõ ràng có hiệu suất tốt hơn so với các mô hình CNN và LSTM.

Ảnh có chứa văn bản

Mô tả được tạo tự động

*Bảng 9: Kết quả so sánh các mô hình LSTM, CNN và CLSTM trên bộ dữ liệu kiến trúc chéo.*

Ảnh có chứa văn bản

Mô tả được tạo tự động

*Bảng 10: Kết quả so sánh các mô hình LSTM, CNN và CLSTM trên bộ dữ liệu tối ưu hóa chéo.*

Ảnh có chứa văn bản, bàn

Mô tả được tạo tự động

*Bảng 11: Kết quả so sánh các mô hình LSTM, CNN và CLSTM trên bộ dữ liệu trình biên dịch chéo.*

Bảng 9, bảng 10 và bảng 11 hiển thị kết quả hoạt động trên bộ dữ liệu kiến trúc chéo, trình biên dịch chéo và bộ dữ liệu tối ưu hóa chéo tương ứng. Tương tự, mô hình CLSTM có hiệu suất tốt hơn trong các thử nghiệm này. Đối với hiệu suất trên bộ dữ liệu phiên bản chéo.

Ảnh có chứa bàn

Mô tả được tạo tự động

*Bảng 12: Kết quả so sánh LSTM, CNN và CLSTM trên bộ dữ liệu phiên bản chéo.*

Bảng 12 minh họa kết quả đánh giá. Trong đánh giá này, tập dữ liệu bao gồm 6 phiên bản của GNU Core Utilities, bao gồm cả phiên bản 8.31. Đánh giá này cũng cho thấy mô hình CLSTM vượt trội hơn so với mô hình CNN và LSTM.

D. Hạn chế và kết luận.

**1. Hạn chế:**

Tương tự như các nghiên cứu trước đây, phương pháp của chúng tôi bị giới hạn trong việc đối phó với mã nhị phân. Trước khi áp dụng phương pháp của chúng tôi, quy trình giải mã mã nguồn là cần thiết để trước tiên trích xuất logic bên trong từ mã mã nguồn. Để làm được điều này, chúng ta có thể tận dụng các kỹ thuật giải mã nguồn gần đây. Chúng tôi dự định khám phá sự kết hợp giữa phương pháp hiện tại và kỹ thuật giải mã nguồn như công việc trong tương lai của chúng tôi.

Để cải thiện hơn nữa độ chính xác phát hiện của phương pháp của chúng tôi, một giải pháp tiềm năng là sử dụng các cấu trúc mạng neural khác nhau cho các tình huống so sánh khác nhau. Ví dụ: Chúng tôi có thể áp dụng mô hình BiLSTM cho BCSD trên các kiến trúc khác nhau và sử dụng mô hình CLSTM cho BCSD trên 20 trình biên dịch khác nhau. Ngoài ra, chúng tôi có thể xem xét nhiều kịch bản so sánh hơn, sẽ tương ứng với nhiều mô hình mạng hơn. Để đo khoảng cách tương tự, chúng ta có thể khám phá một phương pháp khác. Ví dụ: Chúng tôi có thể sử dụng khoảng cách Hamming của các tính năng nhị phân tĩnh cho BCSD. Xem xét các cuộc tấn công sắp xếp dữ liệu gần đây vào các mô hình học máy, chúng tôi có thể tận dụng công việc hiện có để thực hiện biện pháp phòng thủ.

**2. Kết luận:**

Trong bài báo này, chúng tôi trình bày BinDeep, một giải pháp dựa trên deep learning mới để phát hiện sự giống nhau của mã nhị phân. Chúng tôi khai thác IDA Pro để trích xuất chuỗi lệnh dưới dạng các tính năng cho các chức năng đích. Để vector hóa các tính năng này, chúng tôi tận dụng mô hình NLP cổ điển. Tiếp theo, chúng tôi áp dụng mô hình dựa trên RNN để xác định các loại chức năng mục tiêu. Dựa trên các thông tin loại chức năng này, chúng tôi chọn mô hình Siamese neural network tương ứng để đo lường độ tương tự. So với công việc trước đây, chúng tôi kết hợp các mô hình CNN và LSTM để xây dựng Siamese neural network. Kết quả đánh giá cho thấy BinDeep có thể đạt được độ chính xác phát hiện trung bình là 98,43% đối với BCSD trên các kiến trúc, trình biên dịch và mức độ tối ưu hóa khác nhau.

1. **Lời cảm ơn:**

Trong thời gian học môn lập trình an toàn và khai thác lỗ hổng phần mềm, nhóm đã nhân được nhiều sự giúp đỡ, đóng góp ý kiến và chỉ bảo nhiệt tình của thầy cô, bạn bè. Với sự giúp đỡ này đã giúp nhóm rất nhiều trong việc cũng cố kiến thức và giải đáp những thắc mắc còn tồn đọng.

Nhóm xin gửi lời cám ơn chân thành đến cô Đỗ Thị Thu Hiền giảng viên khoa Mạng máy tính và truyền thông dữ liệu trường đại học Công Nghệ Thông Tin, người đã tận tình hướng dẫn, chỉ bảo nhóm trong suốt quá trình làm đồ án.

Với những kiến thức đã được học tại môn này nhóm báo cáo có thể tự tin hơn trên chặng đường học tập sắp tới và trong cuộc sống sau này. Nhóm xin được chúc thầy và các cán bộ giảng viên đang công tác tại trường thật nhiều sức khỏe và thành công trong cuộc sống.

1. **Trích dẫn tài liệu:**

Bài báo cáo dựa trên bài báo **BinDeep: A Deep Learning Approach to Binary Code Similarity Detection** của các tác giả Donghai Tian, Xiaoqi Jia, Rui Ma, Shuke Liu, Wenjing Liu, Changzhen Hu tại hội nghị ESWA 114348.

**Cùng với một số nguồn thông tin khác như sau:**

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D.G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., Zheng, X., 2016. Tensorflow: A system for largescale machine learning, in: Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation, USENIX Association, Berkeley, CA, USA. pp. 265–283.

Chandramohan, M., Xue, Y., Xu, Z., Liu, Y., Cho, C.Y., Tan, H.B.K., 2016. Bingo: Cross-architecture cross-os binary search, in: Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, ACM, New York, NY, USA. pp. 678–689.

David, Y., Yahav, E., 2014. Tracelet-based code search in executables, in: Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation, ACM. pp. 349–360.

Egele, M., Woo, M., Chapman, P., Brumley, D., 2014. Blanket execution: Dynamic similarity testing for program binaries and components, in: Proceedings of the 23rd USENIX Conference on Security Symposium, USENIX Association, Berkeley, CA, USA. pp. 303–317.

Eschweiler, S., Yakdan, K., Gerhards-Padilla, E., 2016. discovre: Efficient cross-architecture identification of bugs in binary code, in: Proceedings of the 2016 Network and Distributed Systems Security Symposium (NDSS).

Feng, Q., Zhou, R., Xu, C., Cheng, Y., Testa, B., Yin, H., 2016. Scalable graph-based bug search for firmware images, in: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, ACM, New York, NY, USA. pp. 480–491. 23

Gensim, 2018. Word2vec embeddings. http://radimrehurek.com/gensim/models/ word2vec.html.

HaddadPajouh, H., Dehghantanha, A., Khayami, R., Choo, K.K.R., 2018. A deep recurrent neural network based approach for internet of things malware threat hunting. Future Generation Computer Systems 85, 88 – 96.

Hadsell, R., Chopra, S., LeCun, Y., 2006. Dimensionality reduction by learning an invariant mapping, in: 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’06), pp. 1735–1742.

Hex-Rays, 2018. Ida pro disassembler and debugger. <https://www.hexrays.com/products/ida/index.shtml>.

Hu, Y., Zhang, Y., Li, J., Wang, H., Li, B., Gu, D., 2018. Binmatch: A semantics-based hybrid approach on binary code clone analysis, in: 2018 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp. 104–114.

Jhi, Y., Jia, X., Wang, X., Zhu, S., Liu, P., Wu, D., 2015. Program characterization using runtime values and its application to software plagiarism detection. IEEE Trans. Software Eng. 41, 925–943.

Liu, B., Huo, W., Zhang, C., Li, W., Li, F., Piao, A., Zou, W., 2018. αdiff: Cross-version binary code similarity detection with dnn, in: Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ACM. pp. 667–678.

Massarelli, L., Di Luna, G.A., Petroni, F., Baldoni, R., Querzoni, L., 2019. Safe: Self-attentive function embeddings for binary similarity, in: Perdisci, R., Maurice, C., Giacinto, G., Almgren, M. (Eds.), Detection of Intrusions and Malware, and Vulnerability Assessment, Springer International Publishing, Cham. pp. 309–329.

Mueller, J., Thyagarajan, A., 2016. Siamese recurrent architectures for learning sentence similarity, in: Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, AAAI Press. pp. 2786–2792.

Pewny, J., Garmany, B., Gawlik, R., Rossow, C., Holz, T., 2015. Crossarchitecture bug search in binary executables, in: 2015 IEEE Symposium on Security and Privacy, pp. 709–724.

Shalev, N., Partush, N., 2018. Binary similarity detection using machine learning, in: Proceedings of the 13th Workshop on Programming Languages and Analysis for Security, ACM, New York, NY, USA. pp. 42–47.

Taheri, R., Ghahramani, M., Javidan, R., Shojafar, M., Pooranian, Z., Conti, M., 2020a. Similarity-based android malware detection using hamming distance of static binary features. Future Generation Computer Systems 105, 230 – 247.

Taheri, R., Javidan, R., Shojafar, M., Vinod, P., Conti, M., 2020b. Can machine learning model with static features be fooled: an adversarial machine learning approach. Cluster Computing .

Wang, S., Wu, D., 2017. In-memory fuzzing for binary code similarity analysis, in: Proceedings of the 32Nd IEEE/ACM International Conference on Automated Software Engineering, pp. 319–330.

Wang, Y., Shen, J., Lin, J., Lou, R., 2019. Staged method of code similarity analysis for firmware vulnerability detection. IEEE Access 7, 14171–14185.

Wikipedia, 2018. One-hot. <https://en.wikipedia.org/wiki/One-hot>.

Xu, D., Ming, J., Fu, Y., Wu, D., 2018. Vmhunt: A verifiable approach to partially-virtualized binary code simplification, in: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, ACM. pp. 442–458. Xu, X., Liu, C., Feng, Q., Yin, H., Song, L., Song, D., 2017. Neural networkbased graph embedding for cross-platform binary code similarity detection, in: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, pp. 363–376.

Yadegari, B., Johannesmeyer, B., Whitely, B., Debray, S., 2015. A generic approach to automatic deobfuscation of executable code, in: 2015 IEEE Symposium on Security and Privacy, pp. 674–691.

Zhao, D., Lin, H., Ran, L., Han, M., Tian, J., Lu, L., Xiong, S., Xiang, J., 2019. Cvsksa: cross-architecture vulnerability search in firmware based on knn-svm and attributed control flow graph. Software Quality Journal .

Zuo, F., Li, X., Young, P., Luo, L., Zeng, Q., Zhang, Z., 2019. Neural machine translation inspired binary code similarity comparison beyond function pairs, in: Proceedings of the 2019 Network and Distributed Systems Security Symposium (NDSS).

**HẾT**

1. Ghi nội dung tương ứng theo mô tả [↑](#footnote-ref-1)