Trường Đại học Công nghệ Thông tin (UIT)

BÁO CÁO ĐỒ ÁN CUỐI KỲ

Môn học: BẢO MẬT WEB VÀ ỨNG DỤNG

ĐỀ TÀI :

Web Application Firewall using Character-level Convolutional Neural Network

Nhóm : BronzeGod

1. THÔNG TIN CHUNG:

Lớp: NT213.N21.ATCL

STT	Họ và tên	MSSV	Phân công
1	Đỗ Quang Thắng	20521893	Tìm hiểu-xây dựng mô hình CLCNN
2	Nguyễn Đoàn Thiên Cung	20521146	Tìm hiểu các kĩ thuật học máy liên quan, Kiến trúc A
3	Vũ Trọng Nghĩa	20520651	Tìm hiểu về các dataset, kiến trúc B
4	Trần Minh Đạt	20521178	Thực hiện hóa đánh giá mô hình sau khi xây dựng

2. NỘI DUNG THỰC HIỆN:[1]

[1] Ghi nội dung công việc, các kịch bản trong bài Thực hành

MỤC LỤC

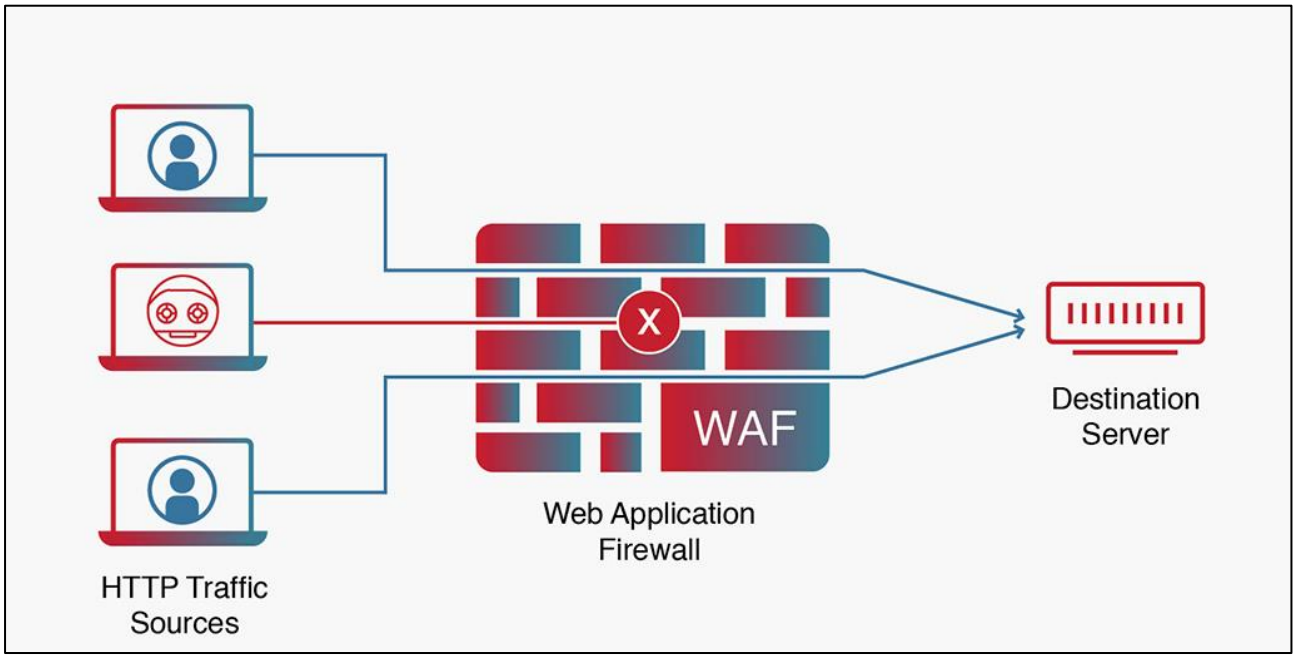
BÁO CÁO CHI TIẾT

I. Bối cảnh -Tổng quan

1. Bối cảnh

Tường lửa được sử dụng để chặn các gói độc hại xâm nhập vào mạng. Tuy nhiên, tường lửa thông thường chỉ chặn các gói tin lên đến tầng truyền tải trong mô hình TCP/IP.

Tường lửa ứng dụng web (WAF) được thiết kế để chặn các cuộc tấn công vào lớp ứng dụng, nằm ngoài phạm vi của tường lửa thông thường.



WAF sử dụng các chiến lược đối sánh mẫu để xác định và chặn các gói tin độc hại hoặc đáng ngờ. Các chiến lược này có thể được phân loại thành hai loại: phương pháp dựa trên danh sách đen và dựa trên danh sách trắng.

Các phương pháp dựa trên Blacklist có chi phí tương đối thấp vì chúng chỉ yêu cầu định nghĩa black signature dựa trên các cuộc tấn công đã biết. Tuy nhiên, chúng không thể xử lý các cuộc tấn công không xác định hoặc phân loài.

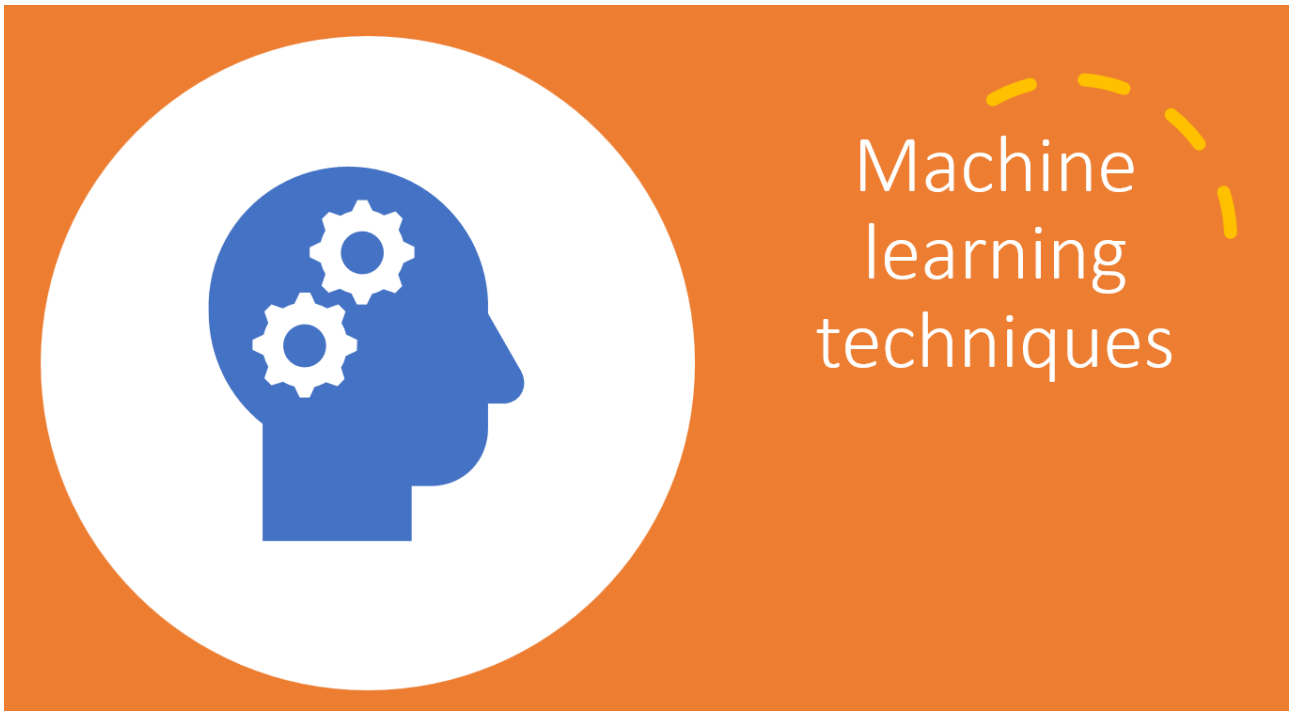
Các phương thức dựa trên Whitelist có nhiều khả năng chặn các cuộc tấn công không xác định, nhưng chúng yêu cầu định nghĩa chi tiết về white signature dựa trên các giao dịch thông thường trên ứng dụng. Điều này làm cho chúng tốn kém hơn.Các phương

thức dựa trên Whitelist cũng có khả năng áp dụng hạn chế vì chúng chỉ cho phép các giao dịch có trong danh sách, có thể bị hạn chế.

Chi phí và hiệu quả của WAF phụ thuộc vào loại chiến lược đối sánh mẫu được sử dụng và mức độ chi tiết trong các định nghĩa chữ ký.

⇨ Do đó các phương pháp học máy được đề xuất để giảm chi phí và cải thiện độ chính xác phát hiện gói tin độc hại.

2. Tổng quan đề xuất



Hai nghiên cứu được đề cập trong đó Wakiou et al. đã sử dụng kĩ thuật học máy naive Bayes và hệ thống kết hợp mẫu lai để phát hiện các cuộc tấn công SQL injection với độ chính xác 97,6%, trong khi Zhang et al. phát hiện các cuộc tấn công trong lưu lượng HTTP bằng Bayes ngây thơ với độ chính xác 82,3%. Tuy nhiên, vẫn còn chỗ để cải thiện về mặt thực tiễn.

Các kỹ thuật học sâu, đặc biệt là mạng nơ-ron tích chập-Convolutional Neural Network (CNN), đã được chú ý trong lĩnh vực học máy. CNN chủ yếu được sử dụng cho các vấn đề nhận dạng hình ảnh và đã vượt qua những khó khăn trong việc thiết kế và triển khai các tính năng làm bằng tay hiệu quả để phân loại và chống lại sự phù hợp quá mức.

Trong các nhiệm vụ xử lý ngôn ngữ tự nhiên- natural language processing (NLP), mạng nơ-ron tái phát- recurrent neural network (RNN) và Long Short Term Memory(LSTM)

thường được sử dụng và đã chứng minh qua các thực nghiệm. LSTM được sử dụng rộng rãi vì nó có thể xử lý dữ liệu chuỗi thời gian trong khi xem xét thông tin trong quá khứ dài hơn RNN. Tuy nhiên, chi phí đào tạo của LSTM nói chung khá tốn kém.

Character-level CNN (CLCNN) là một CNN chuyên xử lý văn bản, biến đổi các chuỗi ký tự theo hướng một chiều. CLCNN thường được sử dụng trong các nhiệm vụ NLP và đã cho thấy kết quả đầy hứa hẹn. Ưu điểm của CLCNN là nó có thời gian đào tạo ngắn hơn nhiều so với LSTM và có tương thích cao với nhiều chiến lược hiệu quả và nổi tiếng có nguồn gốc từ CNN.

Các chiến lược này bao gồm tăng cường dữ liệu trên kích thước đầu vào, loại bỏ để chuẩn hóa, phân tích phương pháp của các bản đồ đặc trưng để giải thích mô hình và dùng để phân tích thêm trên các đặc trưng khác.

3. Tổng quan các nghiên cứu về kỹ thuật học máy

Cụ thể các nghiên cứu trước đây đã sử dụng các kỹ thuật học sâu cho các nhiệm vụ liên quan đến bảo mật như sau :

Melicher và cộng sự đã sử dụng LSTM để đo lường mức độ dễ đoán mật khẩu và đạt độ chính xác lên đến 70%, tốt hơn so với các phương pháp khác như mô hình Markov và ngữ pháp không có ngữ cảnh xác suất (PCFG).

Raff và cộng sự. đã sử dụng CLCNN để phân tích toàn bộ tệp thực thi động (PE) của Windows dưới dạng chuỗi ký tự để nhận dạng phần mềm độc hại và đạt được độ chính xác lên đến 94%. Các tác giả gợi ý rằng việc điều tra toàn bộ chuỗi chuỗi là cần thiết và tổng hợp tối đa toàn cục có hiệu quả trong trường hợp các phần độc hại được phân phối không đồng đều trong một phần ngắn giữa các mục tiêu cần phân tích.

Saxe et al. đã tạo ra một công cụ phân biệt URL độc hại bằng cách sử dụng CLCNN với các kích thước khác nhau của độ dài đoạn ký tự từ 2 đến 5. Họ đã báo cáo AUC là 0,993, cao hơn phương pháp mạng nơ-ron sâu N-gam+trên cùng một dữ liệu.

Nghiên cứu trên cho thấy CLCNN có hiệu quả trong việc phân tích dữ liệu trong lĩnh vực bảo mật.

Tuy nhiên, khi xử lý dữ liệu quy mô lớn, có những lo ngại về các kích thước tích chập khác nhau của mô hình song song, điều này khiến cho không thể khai thác triệt để tính song song trong một GPU duy nhất và có thể gây ra sự chậm trễ. Ngoài ra, có mức sử dụng bộ nhớ GPU lớn do ba lớp được kết nối đầy đủ (FC) tương đối lớn.

Trong nghiên cứu này, các tác giả đã xây dựng một hệ thống nhận dạng cho các yêu cầu HTTP độc hại bằng cách sử dụng kiến trúc CLCNN đơn giản nhận toàn văn của yêu cầu HTTP làm đầu vào và thực hiện tổng hợp tối đa toàn cầu rất lớn. Cách tiếp cận này làm giảm đáng kể kích thước đầu vào ngay cả trong kiến trúc mạng nông, đạt được độ chính xác cao trong khi triệt tiêu các tài nguyên cần thiết.

II. Phương thức

1. Dataset

Tập dữ liệu được sử dụng trong nghiên cứu là tập dữ liệu HTTP DATASET CSIC 2010 cung cấp bởi tổ chức Spanish Research National Council (CSIC).

Tập dữ liệu bao gồm cả lưu lượng truy cập bình thường và độc hại trên ứng dụng web thương mại điện tử Tây Ban Nha, với 36.000 lưu lượng truy cập bình thường và hơn 25.000 lưu lượng độc hại.

Lưu lượng độc hại bao gồm các loại yêu cầu HTTP tấn công khác nhau như SQL injection, tấn công tràn bộ đệm và thu thập thông tin.

```
GET http://localhost:8080/tienda1/publico/anadir.jsp?id=2&nombre=Jam%F3
n+lb%E9rico&precio=85&cantidad=%27%3B+DROP+TABLE+usuarios%3B+
SELECT+*+FROM+datos+WHERE+nombre+LIKE+%27%25&B1=A%F1adir+al+carrito HTTP/1.1
User-Agent: Mozilla/5.0 (compatible; Konqueror/3.5; Linux) KHTML/3.5.8 (like Gecko)
Pragma: no-cache
Cache-control: no-cache
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Encoding: x-gzip, x-deflate, gzip, deflate
Accept-Charset: utf-8, utf-8;q=0.5, *;q=0.5
Accept-Language: en
Host: localhost:8080
Cookie: JSESSIONID=B92A8B48B9008CD29F622A994E0F650D
Connection: close
```

Fig. 1. Example of one SQL injection request data in the HTTP DATASET CSIC 2010

Ví dụ về SQL injection được hiển thị trong Hình 1 làm nổi bật một chuỗi ký tự liên quan đến cuộc tấn công, với độ dài tối thiểu, trung bình và tối đa lần lượt là 473, 583 và 983 ký tự.

2. Xử lý dataset

Tập dữ liệu chứa các yêu cầu HTTP thô với mã hóa URL đã được thực hiện cho tất cả các ký tự trong yêu cầu.

Để trích xuất các tính năng hữu ích của truy vấn, giải mã URL được thực hiện cho tất cả các truy vấn và mã hóa lại bằng Unicode, trong đó mỗi ký tự có trong chuỗi ký tự của yêu cầu HTTP được biểu diễn bằng một chuỗi số 8 bit.

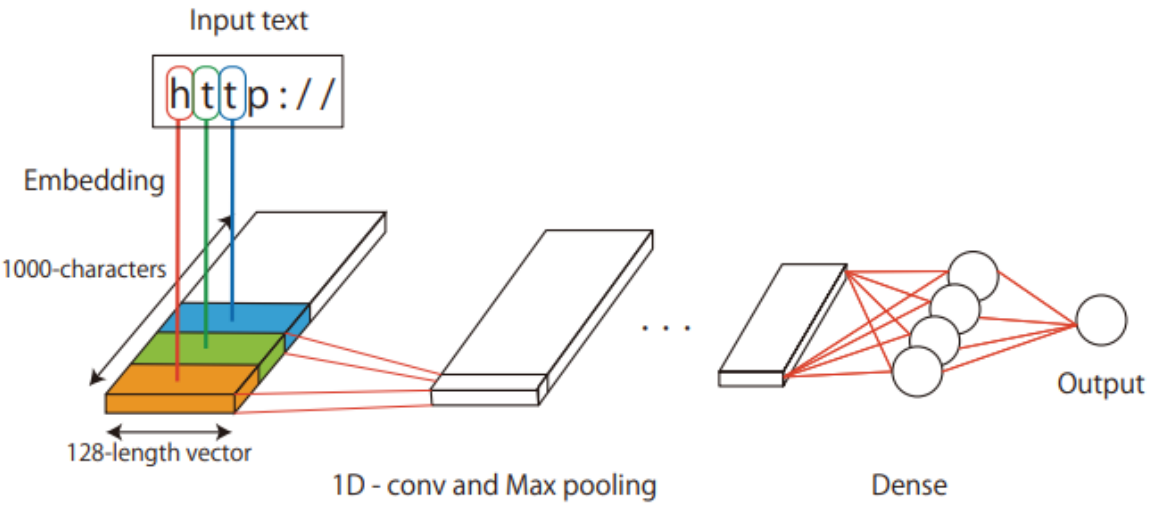


Fig. 2. Outline of our malicious query detection system using CLCNN

Độ dài yêu cầu đầu vào của CLCNN được xác định là 1000 ký tự dựa trên sự phân bố độ dài của mỗi yêu cầu và các phát hiện được đề xuất bởi Raff et al.

Đối với phần không đủ độ dài, tập dữ liệu được đệm bằng '0' như là 1 kí tự null trong Unicode.Tập dữ liệu đã được xử lý trước theo cách này để chuẩn bị sử dụng trong mô hình CLCNN để xác định các yêu cầu bình thường và độc hại.

III. Mô hình triển khai CLCNN

1. Kiến trúc A

Hệ thống phát hiện truy vấn độc hại này bao gồm một tầng nhúng ở đầu tiên để thực hiện tiền xử lý và nhúng dữ liệu theo mô tả trong phần trước. Mỗi ký tự của câu truy

vấn HTTP đầu vào được chuyển đổi thành một biểu diễn vector 128 chiều trong tầng nhúng và được truyền đến các tầng kế tiếp. Kiến trúc cơ bản của hệ thống được định nghĩa và kiến trúc A được xác định dựa trên các thử nghiệm sơ bộ. Cấu hình của kiến trúc A được hiển thị trong Hình 3.

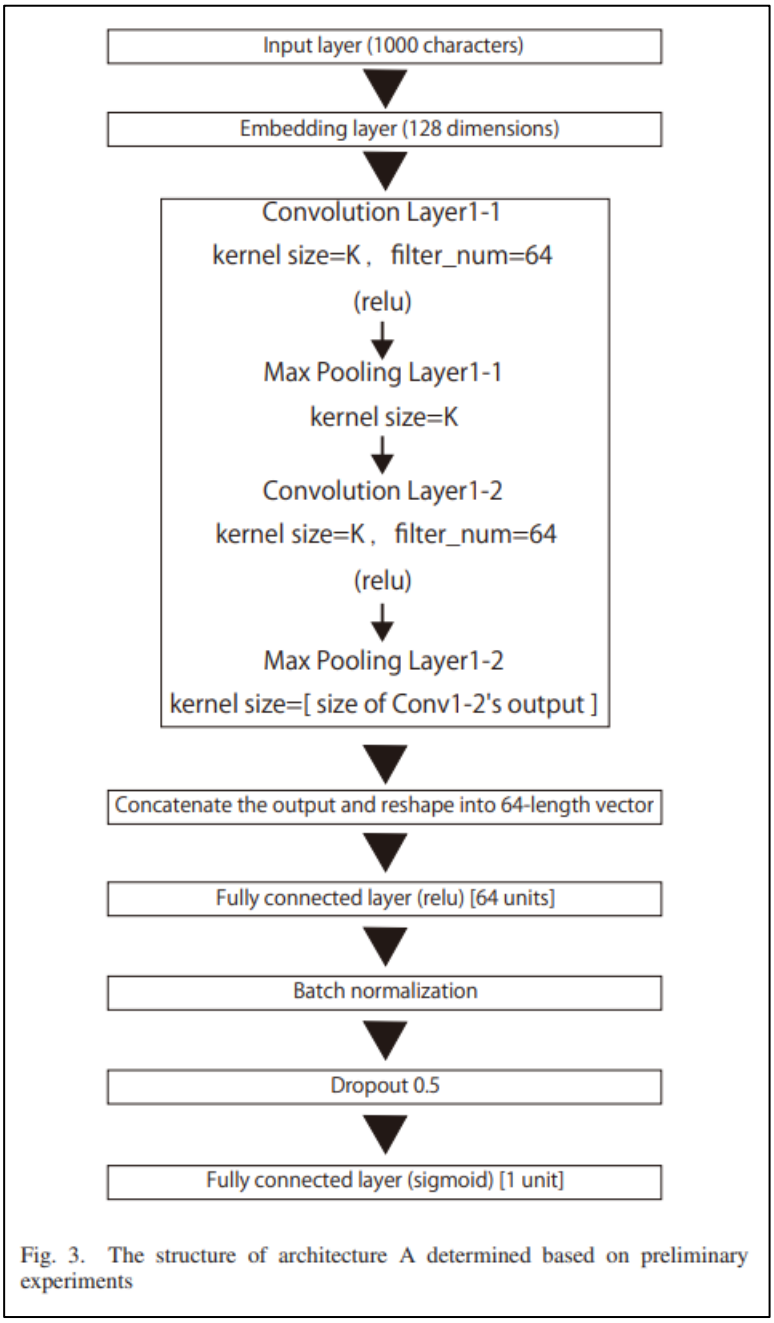


Fig. 3. The structure of architecture A determined based on preliminary experiments

Kiến trúc A thực hiện tích chập và lấy giá trị lớn nhất hai lần và được kết nối với đầu ra một chiều thông qua tầng kết nối đầy đủ. Trong lần lấy giá trị lớn nhất thứ hai, kích thước lấy mẫu được thiết lập giống như kích thước đầu ra của tích chập, tức là mỗi bản đồ tích chập cho ra đầu ra 1x1 với quá trình này và tương ứng với số lượng bản đồ đặc trưng ngay lập tức, chúng tạo ra vector có kích thước phù hợp với số lượng bản đồ đặc trưng. Tham số siêu của kiến trúc A chỉ có một; kích thước kernel K. Trong tất cả các tầng tích chập, kernel 1xK được áp dụng và tham số này cũng được sử dụng trong lần lấy mẫu max-pooling toàn cục đầu tiên. Các hàm kích hoạt đều sử dụng hàm đơn vị

tuyến tính hiệu chỉnh (ReLU). Kiến trúc được xử lý đặc trưng bởi sự giảm kích thước chiều vô cùng lớn trong lần lấy mẫu max-pooling thứ hai (ví dụ: 498:1 tại K = 2), do đó số lượng phần tử FC trong tầng kế tiếp được giảm đáng kể. Điều này giúp giảm kích thước bộ nhớ GPU cần thiết.

2. Kiến trúc B

Để so sánh, xây dựng kiến trúc B như được hiển thị trong **Hình 4**, kiến trúc này ghép nối bốn kiến trúc song song như trong Saxe et al. [5] và cuối cùng làm phức tạp chúng. Các tham số siêu của kiến trúc B là kích thước kernel tích chập giống như kiến trúc A, nhưng có bốn; K1, K2, K3, K4. Chúng cũng được áp dụng trong kích thước lấy mẫu max-pooling đầu tiên. Kích thước bước của tích chập và lấy mẫu max-pooling đều là 1. Số lượng bản đồ đặc trưng trong mỗi tầng tích chập của kiến trúc B bằng 1/4 của kiến trúc A, tức là tổng số bản đồ đặc trưng của chúng trong kiến trúc B tương đương với kiến trúc A.

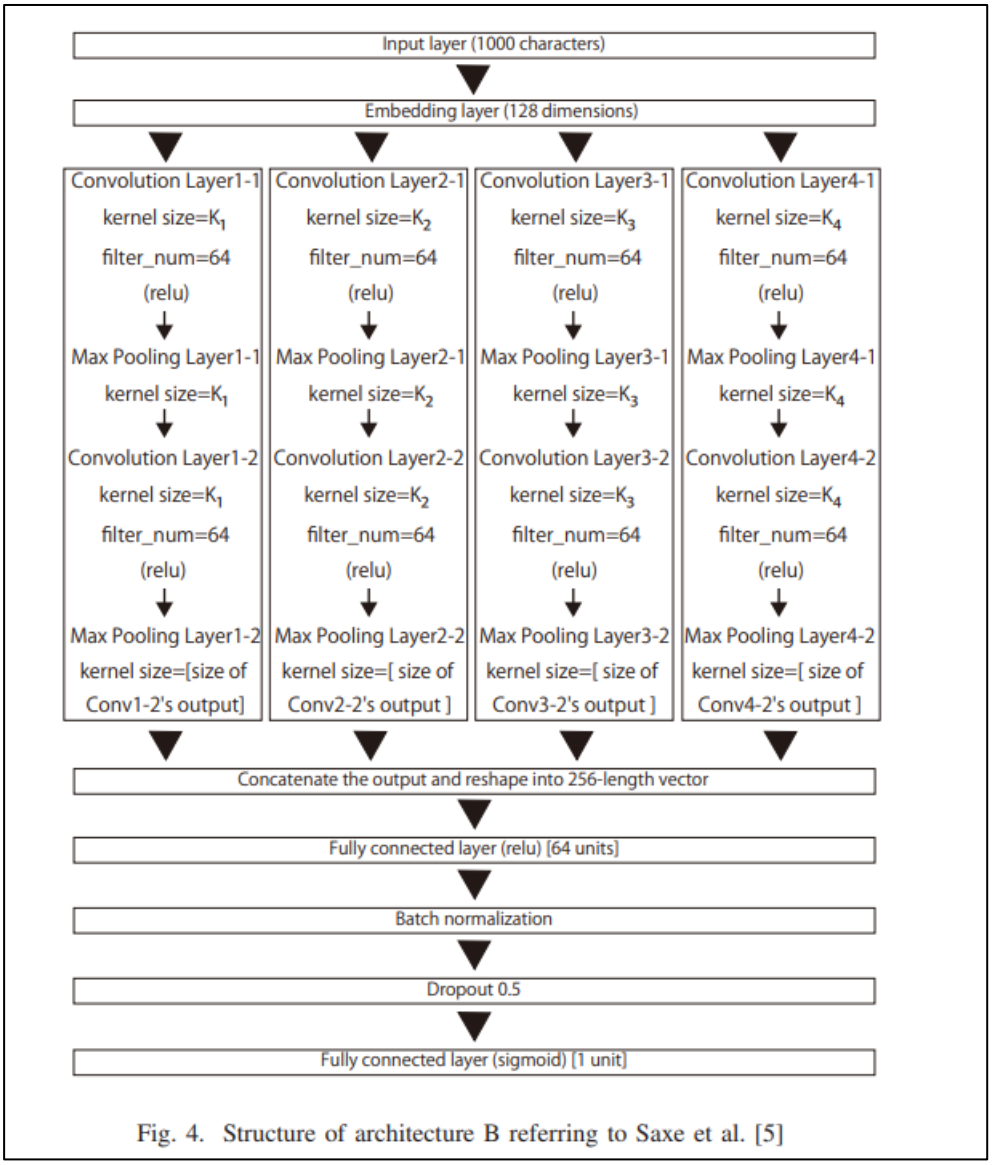


Fig. 4. Structure of architecture B referring to Saxe et al. [5]

IV. Triển khai thực nghiệm

Resource:https://drive.google.com/drive/folders/10lMU1oigdKXdu_d9yyDY_7sfZRIroFtA?usp=drive_link

Dataset: được chia với tỉ lệ 6-2-2 (train-evaluate-test)

Cấu hình triển khai: mô hình được xây dựng, huấn luyện trên *colab*.

Công cụ sử dụng: kera, tenserflow, python (được cài đặt mặc định trên *colab*)

Kiến trúc A:

Các giá trị thông số được sử dụng trong mô hình:

```
OPTIMIZER = "adam"
LOSS = "binary_crossentropy"
K = 7
INPUT_SIZE = 1000
```

Input sẽ nhập đầu vào là 1000 ký tự. Ta thực hiện xây dựng input với shape là (1000,). Tiếp đến sẽ cho qua lớp embedding mục đích làm tăng mối quan hệ giữa các từ, làm cho các từ có liên hệ gần nghĩa với nhau.

```
# Define input layer
inputs = Input(shape=(INPUT_SIZE,), name="Input_layer")

# Add embedding layer
embeddings = Embedding(input_dim=INPUT_SIZE, output_dim=128, name="Embedding_layer")(inputs)
```

Lớp embedding được đưa qua các lớp convolution để chiết xuất các thuộc tính quan trọng. Các kernel size sẽ được chọn theo bài báo là 7 nhằm tối ưu về độ chính xác, hàm kích hoạt là relu và max pooling để chọn ra thuộc tính có trọng số lớn nhất.

```
# Add convolution layers
conv1_1 = Conv1D(filters=64, kernel_size=K, activation='relu', name="Convolution_Layer1-1")(embeddings)
maxpool1_1 = MaxPooling1D(pool_size=K, name="Max_Pooling_Layer1-1")(conv1_1)
conv1_2 = Conv1D(filters=64, kernel_size=K, activation='relu', name="Convolution_Layer1-2")(maxpool1_1)
maxpool1_2 = MaxPooling1D(pool_size=conv1_2.shape[1], name="Max_Pooling_Layer1-2")(conv1_2)
```

Tiếp đến là gom các output của các lớp convolution và đưa qua fully connected layers nhằm huấn luyện đồng thời được sử dụng kết hợp batch normalize và dropout để tăng tính ổn định, giảm thời gian huấn luyện và tránh overfitting. Cuối cùng là output của mô hình là 1 unit với hàm kích hoạt là sigmoid để phân loại request là độc hay lành tính.

```
# Concatenate pooling outputs and reshape into a 64-length vector
concat = Concatenate(axis=1)([maxpool1_1, maxpool1_2])
reshape = Flatten()(concat)

# Add 2 fully connected layers
fc1 = Dense(64, activation='relu', name="Fully_connected_layer")(reshape)
norm = BatchNormalization(name="Batch_normalization")(fc1)

dropout = Dropout(rate=0.5, name="dropout")(norm)
fc2 = Dense(1, activation='sigmoid', name="Output_layer")(dropout)
```

Kiến trúc B:

Kiến trúc B được xây dựng gần tương tự với kiến trúc A. Hai kiến trúc khác nhau ở bước triển khai các lớp convolution. Các thông số được sử dụng ở kiến trúc B:

```
OPTIMIZER = "adam"
LOSS = "binary_crossentropy"
K1, K2, K3, K4 = 4, 5, 6, 7
```

Tương tự như kiến trúc A, kiến trúc B nhận đầu vào là 1000 ký tự nên shape (1000,) và được đưa qua lớp embedding.

```
# Define input layer
inputs = Input(shape=(INPUT_SIZE,), name="Input_layer")

# Add embedding layer
embeddings = Embedding(input_dim=INPUT_SIZE, output_dim=128, name="Embedding_layer")(inputs)
```

Khác với kiến trúc A, kiến trúc B sử dụng 4 lớp convolution với các kernel size lần lượt là k1, k2, k3, k4 đã định nghĩa ở trên. Các lớp này sử dụng hàm kích hoạt là relu và đều sự dụng max-pooling để chiết xuất thông tin.

```
# Add convolution_1 layers
conv1_1 = Conv1D(filters=64, kernel_size=K1, activation='relu', name="Convolution_Layer1-1")(embeddings)
maxpool1_1 = MaxPooling1D(pool_size=K1, name="Max_Pooling_Layer1-1")(conv1_1)
conv1_2 = Conv1D(filters=64, kernel_size=K1, activation='relu', name="Convolution_Layer1-2")(maxpool1_1)
maxpool1_2 = MaxPooling1D(pool_size=conv1_2.shape[1], name="Max_Pooling_Layer1-2")(conv1_2)

# Add convolution_2 layers
conv2_1 = Conv1D(filters=64, kernel_size=K2, activation='relu', name="Convolution_Layer2-1")(embeddings)
maxpool2_1 = MaxPooling1D(pool_size=K2, name="Max_Pooling_Layer2-1")(conv2_1)
conv2_2 = Conv1D(filters=64, kernel_size=K2, activation='relu', name="Convolution_Layer2-2")(maxpool2_1)
maxpool2_2 = MaxPooling1D(pool_size=conv2_2.shape[1], name="Max_Pooling_Layer2-2")(conv2_2)

# Add convolution_3 layers
conv3_1 = Conv1D(filters=64, kernel_size=K3, activation='relu', name="Convolution_Layer3-1")(embeddings)
maxpool3_1 = MaxPooling1D(pool_size=K3, name="Max_Pooling_Layer3-1")(conv3_1)
conv3_2 = Conv1D(filters=64, kernel_size=K3, activation='relu', name="Convolution_Layer3-2")(maxpool3_1)
maxpool3_2 = MaxPooling1D(pool_size=conv3_2.shape[1], name="Max_Pooling_Layer3-2")(conv3_2)

# Add convolution_4 layers
conv4_1 = Conv1D(filters=64, kernel_size=K4, activation='relu', name="Convolution_Layer4-1")(embeddings)
maxpool4_1 = MaxPooling1D(pool_size=K4, name="Max_Pooling_Layer4-1")(conv4_1)
conv4_2 = Conv1D(filters=64, kernel_size=K4, activation='relu', name="Convolution_Layer4-2")(maxpool4_1)
maxpool4_2 = MaxPooling1D(pool_size=conv4_2.shape[1], name="Max_Pooling_Layer4-2")(conv4_2)
```

Tương tự kiến trúc A, tại kiến trúc B các lớp maxpooling sẽ là đầu vào cho lớp Fully connected để huấn luyện kết hợp batch normalization và dropout để lấy output là 1 unit nhằm phân loại request là độc hay lành tính.

```
# Concatenate pooling outputs and reshape into a 64-length vector
concat = Concatenate(axis=1)([maxpool1_1, maxpool1_2, maxpool2_1, maxpool2_2, maxpool3_1, maxpool3_2, maxpool4_1, maxpool4_2])
# outg_reshaped = Reshape((256, ))(concat)
reshape = Flatten()(concat)

# Add 2 fully connected layers
fc1 = Dense(256, activation='relu', name="Fully_connected_layer")(reshape)
norm = BatchNormalization(name="Batch_normalization")(fc1)

dropout = Dropout(rate=0.5, name="dropout")(norm)
fc2 = Dense(1, activation='sigmoid', name="Output_layer")(dropout)
```

V. Kết quả và đánh giá-đề xuất

Kết quả sau khi train và đánh giá các thông số:

	precision	recall	accuracy	F1 score
Kiến trúc A	0.9990	1	0.9875	0.9995
Kiến trúc B	0.9982	1	0.9833	0.9998

So với kết quả thu được từ bài báo:

Kiến trúc A có accuracy 98.8% và kiến trúc B là 98.2%. Kết quả triển khai của nhóm gần với kết quả triển khai từ bài báo.

Dựa vào kết quả thu được có thể đưa ra một số nhận định:

- Kiến trúc A thu được với độ chính xác cao hơn về precision và accuracy.
- Với việc sử dụng 1 lớp convolution ở kiến trúc A so với 4 lớp ở kiến trúc B, giúp tối ưu về thời gian huấn luyện hơn, giảm yêu cầu về cấu hình phần cứng hơn đồng thời giúp giảm thời gian dự đoán của mô hình khi triển khai.
- Việc sử dụng 4 lớp tích chập làm giảm khả năng tính toán song song của GPU làm giảm hiệu năng và không tối ưu được phần cứng

Đề xuất cải tiến:

- Dựa theo gợi ý của tác giả, mô hình có thể được triển khai với kích thước đầu vào lớn hơn 1000 ký tự. Từ đó có thể tăng thêm kích thước của lớp tích chập. Đồng thời không cần tăng thêm lớp fully connected do các thông tin quan trọng đều được chiết xuất thông qua mạng tích chập như vậy có thể tối ưu về thời gian huấn luyện hơn.
- Việc tăng thêm lớp tích chập cần thay đổi về kernel size, có thể thực hiện thử nghiệm để xác định được thông số cần sử dụng khi thay đổi kích thước input và số lớp của mạng tích chập.
- Đồng thời có thể kiểm trả các thông số và cách bố trí của 2 lớp dropout và batch normalization để hạn chế mâu thuẩn khi huấn luyện nhằm tối ưu về thời gian.

TÀI LIỆU THAM KHẢO

1. Ito, M., & Iyatomi, H. (2018, March). Web application firewall using character-level convolutional neural network. In *2018 IEEE 14th International Colloquium on Signal Processing & Its Applications (CSPA)* (pp. 103-106). IEEE.
2. Zhang, M., Xu, B., Bai, S., Lu, S., & Lin, Z. (2017). A deep learning method to detect web attacks using a specially designed CNN. In *Neural Information Processing: 24th International Conference, ICONIP 2017, Guangzhou, China, November 14–18, 2017, Proceedings, Part V 24* (pp. 828-836). Springer International Publishing.