DePaul University

Final Report

Automated Vulnerability Detection in Binary Codes

Amy Aumpansub

Professor Zhen Huang

March 22nd , 2021

Introduction

During the past few years, Deep Learning-based techniques have been utilized for automated vulnerability detection and served as an alternative solution of static vulnerability detection system which is primary based on source code analysis and requires an extensive domain knowledge in pattern matching and feature extractions [1]. Several recent studies of DL-based techniques have demonstrated promising results in achieving a high accuracy at detecting vulnerabilities [2]. However, those studies are based on the analysis of source codes which is applicable to only open-source software. When there is no access to the source code, the binary code analysis is preferable as it can be used to assess the code security and detect vulnerability without visibility into the source code. Binary codes provide great benefits in conducting an in-depth analysis of understanding and analyzing the development of code base while requiring the fewer domain knowledge. The recent studies on automated static binary code analysis relies on Control Flow Graph to covert the assembly instructions to the vectors which can quickly predict the vulnerability function; however, the available tools do not have promising results with low accuracy rate [3]. To provide an effective solution, our study purposed the method which combines the binary code analysis and effective machine learning and deep learning techniques to automatically assess and predict software vulnerability. Our approach is based on the extraction of machine instruction and operand features from CFG basic blocks and the use of Machine Learning Algorithms such as Decision Tree and Random Forest to predict vulnerability by learning decision rules inferred from the features and identify the important features contributing to vulnerability detection. In advanced model development phase, the Support Vector Machine algorithm and Neural Networks including Bidirectional and traditional Recurrent Neural Networks (LSTM and GRU) were applied to build the models with hyper-parameter tuning to optimize the model in automated vulnerability detection from those machine instruction and operand features. Our purposed method is found to be more effective than the traditional binary code analysis as it achieves an accuracy up to 83%.

Dataset

The binary programs used in this study was retrieved by complying the original 13,443 programs written in C/C++ languages which was collected from Software Assurance Reference Dataset (SARD). The programs were compiled with GNU on Linux x86-64. The binary programs comprise 44,494 binary functions which include:

- 31,009 non-vulnerable functions (69.69 % of total functions)
- 13,485 vulnerable functions (30.31 % of total functions)

Dependent Variable:
A binary variable (0 is vulnerable function and 1 is non-vulnerable function)

Independent Variables: The numeric vectors of 105 features extracted from basic blocks in all functions using the CounterVectorizer Model. The sample observations of dataset are shown in Fig. 1 in which each observation contains 105 vectors. Each vector represents the total number of each feature appeared in basic blocks of a binary function. The details of how the dataset was generated and feature extraction were discussed in the data preprocessing section.

105 Independent Variables

1 Dependent Variable

	add	and	byte	call	cdq	cdqe	cmp	dword	endbr64	idiv	...	qword	rep	ret	shl	shr	sub	test	xor	target
0	1	0	1	1	0	0	2	7	1	0	...	5	0	1	0	0	1	0	2	0
1	0	0	1	3	0	0	0	1	2	0	...	5	0	1	0	0	1	1	2	0
2	2	0	2	1	0	2	2	11	1	0	...	5	0	1	0	0	1	0	2	0
3	1	0	1	1	0	1	2	13	1	0	...	5	0	1	0	0	1	0	2	0
4	0	0	1	3	0	0	0	1	1	0	...	5	0	1	0	0	1	0	2	0

FIG.1 SAMPLE OBSERVATIONS

Methodology

The binary programs were retrieved by complying original C/C++ programs with GNU on Linux x86-64. The CFG basic blocks were extracted from each binary function using Angr 8, a python framework for analyzing binaries [4]. Then, the machine instructions and operands were extracted from all basic blocks and transformed to a vector using the CounterVectorizer for feature extraction. Several Machine Learning algorithms were utilized to derive the classification rules and identify the important features. In advanced model development phase, the models were built using Support Vector Machine and Deep Learning algorithms to predict target (binary) attribute, whether the binary function is vulnerable or not. The Neural Networks were trained using traditional Long Short-Term Memory (LSTM), Bidirectional LSTM (Bi-LSTM), and Bidirectional GRU (Bi-GRU). Several Python packages include Pandas, Matplotlib, NumPy, Scikit-learn, Keras, and Tensorflow were utilized in this study.

Data Preprocessing

A. Instructions and Operands Extraction

The process to extract the set of instructions and operands from the binary programs was shown in Fig. 2. The first step is to extract vulnerable functions and non-vulnerable functions from binary programs. Then, the CFG was generated to extract the basic blocks associated with each function. The third step involves the machine instructions and operands extraction. Finally, the unique features were extracted from the set of machine instructions and operands of all functions using the CounterVectorizer Model.

Step 1: The vulnerable functions and non-vulnerable functions were extracted from a binary program (object files). and the address of each function was parsed as an input for the next step.

Step 2: Each binary function was parsed to Angr 8 with an address of function to construct Control Flow Graph which provides all basic blocks belonging to the function.

Step 3: For each function, the machine instructions and operands were extracted from the basic blocks as a set of tokens with an assigned test number (Fig. 3). The output of this step is the set of instructions and operands appeared in each function. The machine instructions such as mov, call, jg, etc. were extracted without grouping them. For operands, they can be divided into 4 types and labeled by number 1 and 2 which are a position appeared after each machine instruction as follows:

a) Data types including qword, dword, byte, etc.
b) Registers including 32-bit and 64-bit register
c) Address including the address instruction
d) Constant including any constants

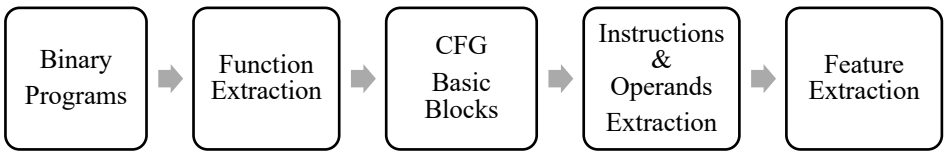


FIG.2 PROCESS OF EXTRACTING FEATURES FROM BINARY PROGRAMS

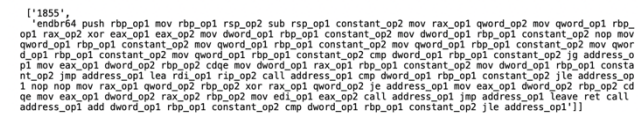


FIG.3 SET OF INSTRUCTIONS AND OPERANDS EXTRACTED FROM BASIC BLOCKS OF 1 FUNCTION

TABLE 1: THE 105 FEATURES EXTRACTED FROM ALL FUNCTIONS

Instructions	**Operands (position 1, position 2)**			
	Data Type	***Register***	***Address***	***Constant***
mov, add, call, …	byte_1, byte_2, dword_1, dword_2, …	32-bit 64-bit eax_1, eax_2, …	address_1 address_2	constant_ 1 constant_ 2
54	7	40	2	2

B. Feature and Vector Extraction

There are a total of 105 unique features extracted from the sets of instructions and operands for all functions, which includes 54 machine instructions and 51 operands comprising 7 data types, 40 registers, 2 groups of addresses, 2 groups of constants (Table 1). The process of transforming a set of instructions and operands to a set of vectors is shown in Fig 4. The CounterVectorizer Model from Scikit-learn package was utilized to transform the set of instructions and operands of each function into a vector on the basis of the frequency (count) of each feature that occurs in the entire set [5]. The output of the whole process is the dataset shown in Fig 1. Each unique feature represents 1 instruction or operand which is represented by a column of matrix, while the value of each cell is the count of the feature in that set of instructions and operands from each function. The row represents the vectors of one function with a target label 0 for non-vulnerability function and 1 for vulnerability function. In short, each vector represents the total number of each feature appeared in basic blocks of a binary function.

C. Z-score Normalization

Support Vector Machine algorithm classifies the data by finding the hyperplane that maximizes the margin (distance) between the two classes, thus can be sensitive to feature scaling [6]. To achieve an accurate model, the set of vectors needs to be normalized before fitting the SVM model. Additionally, the normalized data were later used to build neural networks. The Z-score normalization transforms vectors of each feature by subtracting the mean and then scaling to unit variance. The normalized data have a normal distribution with a mean of 0 and a standard deviation of 1.

D. Dataset Splitting

The 44,494 observations were into 3 sets: Training, Validation, and Test sets using stratify sampling to ensure the proper ensure that each class within 3 sets has a proper representation of the entire population. There are a 69.63% of class 0 and 30.30% of class 1 in each set which represent the entire population of class 0 and class 1 in the original dataset (Fig. 5). The total number of observations in each set were shown in Fig 5 in which 10% of dataset was randomly selected as a test set for an unbiased evaluation for only final model. The remaining data were split into a training set (70%) for fitting model and a validation set (30%) for model evaluation while tuning hyper-parameters.

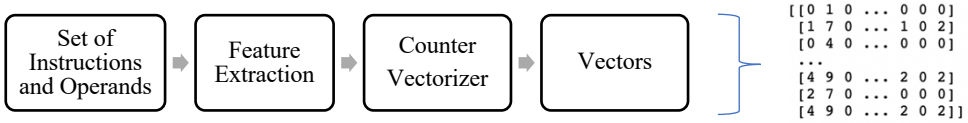


FIG.4 PROCESS OF TRANSFORMING SET OF INSTRUCTIONS AND OPERANDS TO VECTORS

Train_Count	Val_Count	Test_Count	Train_Percent	Val_Percent	Test_Percent
19535	8373	3101	0.696932	0.696937	0.696854
8495	3641	1349	0.303068	0.303063	0.303146

FIG.5 TOTAL NUMBER OF OBSERVATIONS AND CLASS PROPORTION IN EACH SET

Model Development I

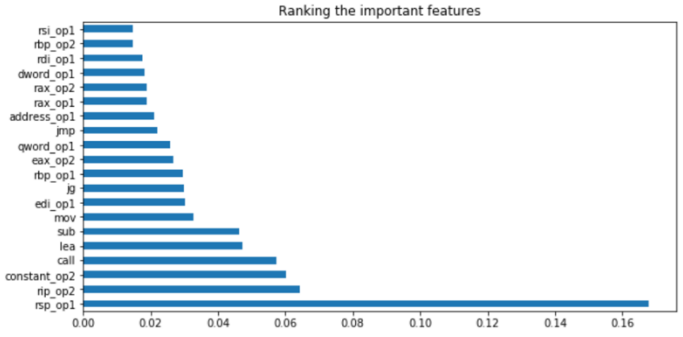
Machine Learning

To justify our method to use operands in addition to machine instructions as features, we need to identify the important features in classification to ensure the operand features provide useful information in classifying whether a binary function is vulnerable or not. The set of important features can be derived from the Decision Tree and Random Forest models which compute the important scores of features. The "Gini" Impurity criterion was selected. It is the probability of incorrect classification of a new randomly chosen element in the dataset if it were randomly classified according to the class distribution in the dataset, computed as $G = \sum_{i=1}^{c} f_i\,(1 - f_i)$ where f_i is the frequency of label i at node and C is the number of unique labels [7]. The higher value of Gini index is preferable, and the best split in training the models is selected by maximizing Gini index. For the feature importance, it is computed as the decrease in node impurity weighted by the probability of reaching that node or the number of observations reaching the node divided by the total number of observations. Thus, the higher scores are the more important of features in classification.

1. Decision Tree

The model was trained by using DecisionTreeClassifier function from sklearn.tree module with an optimized version of CART algorithm. The GridSearchCV was performed for tree optimization in which the set of different parameters including maximum depth of tree, minimum splits, and class weights was parsed with the "Gini" Impurity criterion to build various trees. The optimal tree contains a maximum depth of 40, a minimum split of 30 samples, and the class weight of 0.71 (class 0) and 1.64 (class 1).

- The dependent variable is a binary variable (Is binary function vulnerable?).
- Fig 6 shows the top 20 important features ranked by feature important scores.
- The most 3 important attributes are rsp in the first operand (rsp_op1), rip in the second operand (rip_op2), and constant in the second operand (constant_op2).
- From top 20 important features, there are 14 operand features such as register rsp, rip, etc. and 6 instruction features such as call, lea, sub, etc.
- This implies that the operand features also play an important role in classification.
- The classification matrix (Fig. 7) shows that the final decision tree model has a good performance in prediction with validation accuracy of 81% and training accuracy of 83%, but the model is slightly overfitting. The F1 score is 74% which is comparable to the specificity of 76%, so the model has a moderately good performance in predicting class 0 and class 1.




```
Important Scores:
rsp_op1         0.167610
rip_op2         0.064422
constant_op2    0.060118
call            0.057314
lea             0.047177
sub             0.046356
mov             0.032621
edi_op1         0.030351
jg              0.029847
rbp_op1         0.029446
eax_op2         0.026600
qword_op1       0.025661
jmp             0.022064
address_op1     0.021038
rax_op1         0.018963
rax_op2         0.018793
dword_op1       0.018274
rdi_op1         0.017571
rbp_op2         0.014896
rsi_op1         0.014858
```

FIG.6 TOP 20 IMPORTANT FEATURS RANKED BY IMPORTANT SCORES (DECISION TREE)

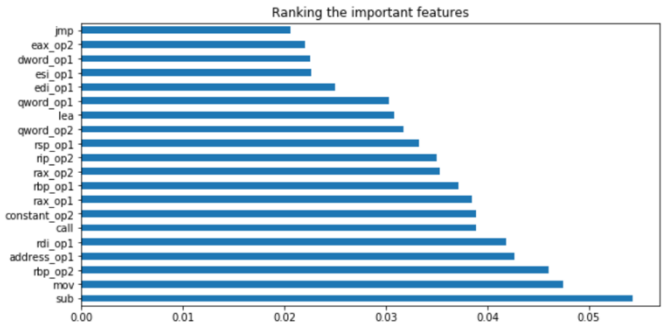
	precision	recall	f1-score	support
0	0.95	0.76	0.85	8373
1	0.63	0.91	0.74	3641
accuracy			0.81	12014
macro avg	0.79	0.84	0.80	12014
weighted avg	0.85	0.81	0.82	12014

FIG.7 CLASSIFICATION METRIC OF DECISION TREE

2. Random Forest

The model was trained by using RandomForestClassifier function from sklearn.ensemble module. The random forest algorithm is an ensemble learning method which fits a number of decision trees (estimators) with the subset of training set and uses bagging in feature randomness to fit each tree [8]. This algorithm provides uncorrelated forest of trees with an average prediction that control overfitting. The GridSearchCV was performed for model optimization in which the set of different parameters including maximum depth of tree, minimum splits, number of estimators and class weights was parsed with the "Gini" impurity criterion to build various forests. The optimal forest contains a maximum depth of 20, a minimum split of 30 samples, the number of estimators of 300, and the class weight of 0.71 (class 0) and 1.64 (class 1).

- The dependent variable is a binary variable (Is binary function vulnerable?).
- Fig 8 shows the ranking of 20 important features with feature important scores.
- The most 3 important attributes are sub, mov, and register rbp in the second position (rbp_op2).
- From top 20 important features, there are 15 operand features such as register rbp, rdi, etc. and 5 instruction features such as sub, mov, call, etc.
- This implies that the operand features also play a main role in classification.
- The classification matrix (Fig. 7) shows that final model has a slight improvement from Decision Tree. It has a good performance with 82% validation accuracy and 83% training accuracy, so the model is less overfitting than Decision Tree. The F1 score is 75% which is comparable to the specificity of 77%, so the model has a moderately good performance in predicting class 0 and class 1.




```
Important Scores:
sub             0.054257
mov             0.047494
rbp_op2         0.045984
address_op1     0.042671
rdi_op1         0.041853
call            0.038903
constant_op2    0.038899
rax_op1         0.038429
rbp_op1         0.037096
rax_op2         0.035269
rip_op2         0.034965
rsp_op1         0.033312
qword_op2       0.031694
lea             0.030770
qword_op1       0.030286
edi_op1         0.024951
esi_op1         0.022635
dword_op1       0.022503
eax_op2         0.022028
jmp             0.020641
```

FIG.8 TOP 20 IMPORTANT FEATURS RANKED BY IMPORTANT SCORES (RANDOM FOREST)

	precision	recall	f1-score	support
0	0.96	0.77	0.85	8373
1	0.63	0.93	0.75	3641
accuracy			0.82	12014
macro avg	0.80	0.85	0.80	12014
weighted avg	0.86	0.82	0.82	12014

FIG.9 CLASSIFICATION METRIC OF RANDOM FOREST

3. Support Vector Machine

Support vector machine is a supervised learning algorithm and can be used in classification problem as support vector classifier (SVC). The model is based on the findings of an optimal hyperplane with the largest minimum margins between two classes, which can clearly sperate two classes. The model was trained by using SVC function from sklearn.svm module. The Radial Basis Function (RBF) kernel was applied to this study as it can work with our non-linear dataset. The RBF kernel can be represented as $K(X_1, X_2) = \exp(-\gamma \|X_1 - X_2\|^2)$ where $\|X_1 - X_2\|$ is the Euclidean distance between 2 points (L2 norm) and γ is the parameter that sets the spread of kernel [9]. The gamma value has an inverse relationship with sigma, in which larger γ is less curvature. Another hypermeter associated with non-linear SVC with RBF kernel is the C value which controls the trade-off between training errors and the errors of unseen data where a larger C allows larger misclassification, which makes the margin wider.

To find the optimal hyperplane, hyperparameter tuning was performed. The C and Gamma values used in hyperparameter tuning ranges from 0.0001 to 1000. The SVC models were fit with different C values. The optimal C value is 100 which achieves the highest validation accuracy (Fig.10). This C value was carried over for tuning Gamma values. For the gamma values, the optimal gamma value is 0.01 which has the highest validation accuracy.

The final model fitted with the optimal C and Gamma value of 100 and 0.01 has a good performance in prediction with 81% accuracy of validation set and 82.4% of training set and is less overfitting than Decision Tree and Random Forest (Fig. 11). The F1 score is 74% which is comparable to the specificity of 76%, so the model has a moderately good performance in predicting class 0 and class 1. However, among three machine learning algorithms, the Random Forest model performs the best which has the highest accuracy rate of 82%, F1 score of 75%, and specificity of 77% (Table 2).

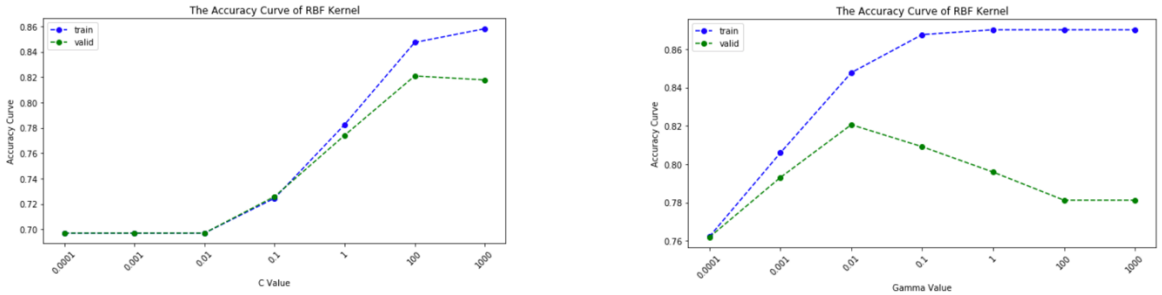


FIG.10 ACCURACY CURVE ACROSS C AND GAMMA VALUES

	precision	recall	f1-score	support
0	0.95	0.76	0.85	8373
1	0.62	0.90	0.74	3641
accuracy			0.81	12014
macro avg	0.79	0.83	0.79	12014
weighted avg	0.85	0.81	0.81	12014

FIG.11 CLASSIFICATION METRIC OF SVC WITH RBF KERNEL

TABLE 2 MODEL COMPARISION

Model	**Accuracy**		**F1-Scores**	**Specificity**
	Training	*Validation*	*Validation*	*Validation*
Decision Tree	83.14%	80.85%	74.00%	76.00%
Random Forest	83.21%	81.50%	75.00%	77.00%
SVM	82.42%	80.64%	74.00%	76.00%

Model Development II

Neural Networks

Each vector array discussed in the previous section represents the number of feature counts in one binary function. The vector arrays were reshaped from two- to three-dimensional array of vectors to feed the deep learning model. The dependent variable is a binary variable (Is a binary function vulnerable?). The architecture of Neural Network was shown in Fig. 12.

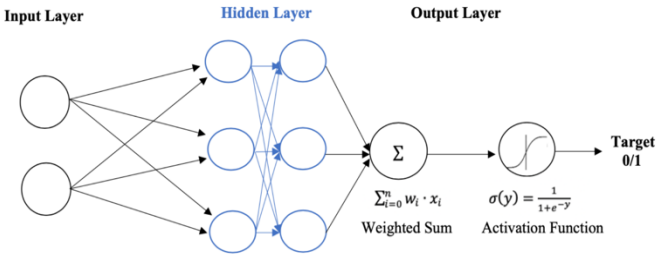


FIG.12 ARCHITECTURE OF NEURAL NETWORK

Loss Function

Loss Function: The binary cross-entropy loss (BCE) is applied in this study as it fit our dataset which has binary target class 0 or 1. Each output unit from neural network model is a separate random binary variable, so total loss for the entire outputs in each step is computed as product of the loss of single binary variables.
Cross-entropy loss is computed as follows [10]:

$$C = -\frac{1}{n}\sum_{x}[y \ln a + (1-y)\ln(1-a)]$$

Comparing to Mean Squared Error, Cross-entropy loss can speed up the learning process of neuron network as process will converge faster than Mean Squared Error which is based on a quadratic function. Cross entropy with sigmoid activation is recommended to use in a classification problem as it is found to speed up the learning process and its curve decreases faster in the earlier epochs [11].

Optimizer

ADAM optimizer is a combination of RMSprop and SGD with momentum, which is an adaptive learning rate method and computationally efficient as it computes individual learning rates for different parameters. ADAM method is appropriate for problems with large dataset and/or parameters, with non-stationary objectives, and for problems with very noisy and/or sparse gradients [12]. Given the nature of software vulnerabilities which contain various causes, the ADAM method is an appropriate method to further examine and apply towards the model development in this study. ADAM optimizer would improve our neural network when a network has weak signals which is not sufficient to tune its weights effectively.

Activation Function (Output Layer)

The Sigmoid function was chosen in the study which was applied to the output layer to produce the activation outputs from the neural network (Fig. 12). Sigmoid activation is continuous and differentiable based on logistic function. Outputs range between 0 to 1, implying probability of each binary class. Thus, it is proper for our binary classification problem with a binary target class (0 non-vulnerability and 1 vulnerability). Sigmoid is commonly used as it does not destroy the activation, but it may vanish gradient. The "Sigmoid" function was implemented in the last activation layer to compute the outputs as [13].

$$o = \sigma(\textstyle\sum_{i=0}^{n} w_i \cdot x_i) \quad \text{where } \sigma(y) = \frac{1}{1+e^{-y}}$$

The weight update formula: $\Delta w_i = \Delta w_i + n(t_d - o_d)o_d\,(1-o_d)\,x_i$ where n is learning rate

Activation Function (Hidden Layers)

The activation function of hidden layers was tuned in this step. The models were built with the same architecture and parameters (Fig. 13) except the activation function chosen for hidden layers in which the "ReLu"function was applied to Base Model 1 and the "Tanh" function was applied to Base Model 2

a) The first activation function applied to hidden layers is "ReLu" activation which can compute outputs faster than sigmoid function and have benefits of sparsity and a reduced likelihood of vanishing gradient [14].
b) The second activation function is "Tanh" which is utilized to compute output as the weighted sum of input and biases for the hidden layers [15]:

$$o = tanh(\sum_{i=0}^{n} w_i \cdot x_i) \text{ where } tanh(y) = \frac{e^y - e^{-y}}{e^y + e^{-y}}$$

The weight update formula: $\Delta w_i = \Delta w_i + n(t_d - o_d)\,(1 - {o_d}^2)\; x_i$ where n is learning rate.

Base Model 1 and 2 were built using a sequential model from Keras package. The Long short-term memory (LSTM), an artificial recurrent neural network was chosen. The neural network contains 256 neuron units and 2 hidden layers fitted with a batch size of 32 for 100 epochs. The "Sigmoid" function was applied to the output layer. The binary cross-entropy loss is selected as loss function as it can speed up the learning process and convergence. The model was compiled with "Adam" optimizer and the learning rate is 0.001.

For Base model 1 fitted with "ReLu" activation function, the model starts to be overfitted after Epoch 40 as the validation loss starts to increase while the training loss keeps decreasing (Fig.13). The loss curve fluctuates dramatically. The accuracy curve shows the similar pattern in which the training accuracy keeps increasing while the validation accuracy has no improvement beyond epoch 40. The highest validation accuracy is 81.10% with the loss rate of 0.34 (Epoch 40).

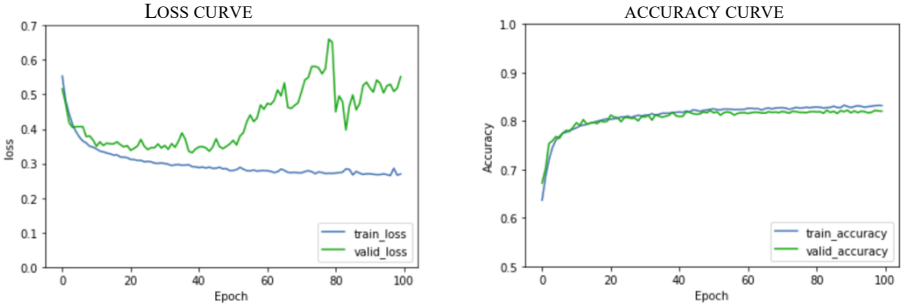


FIG.13 BASE MODEL I PERFORMANCE (RELU)

For Base Model 2 fit with "Tanh" activation function, the loss curve is smoother than the model 1 which is preferable as the model continues to converge as the training process goes. The model starts to be overfitted after Epoch 31 as the validation loss reaches the minimum and is constant while the training loss keeps decrease (Fig.14). The accuracy curve shows the similar pattern in which the training accuracy keeps increasing while the validation accuracy has no improvement beyond epoch 31. The highest validation accuracy is 81.41% with the loss rate of 0.32 (Epoch 31). Comparing to Base Model 1 ("ReLu"), the Base Model 2 ("Tanh") performs slightly better as it has higher accuracy rate and lower loss rate. Its loss curve is also smoother and less overfit than model 1.
Thus, the "tanh" activation was carried over to the next model development.

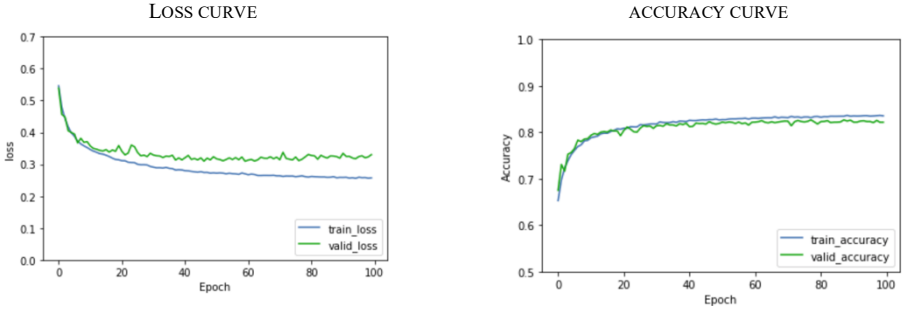


FIG.14 BASE MODEL II PERFORMANCE (TANH)

Bidirectional Recurrent Neural Networks:

Bidirectional Recurrent Neural Networks promote the better understanding of context as it provides the original input sequence to the first layer and a reversed copy of the input sequence to the second layer, so there are two layers side-by-side. These Bidirectional networks have access to the past as well as the future information; therefore, the output is generated from both the past and future context and leads to a better prediction and classifying sequential patterns. Additionally, Bidirectional RNNs are found to be more effective than regular RNNs. It has been widely used as it can overcome the limitations of a regular RNN [16]. The regular RNN model preserves only past information.

The Bi-LSTM model was fitted with the same architecture as the Base Model 2 except the Bidirectional LSTM was applied instead of regular LSTM. The performance metrics of LSTM and BLSTM models shown in Table 3 also indicates that the bidirectional unit outperforms the regular LSTM holding other hyperparameters constant.

The Bi-LSTM model slightly performs better than the LSTM Model. It has a lower loss rate of 0.31 compared to a loss rate of 0.32 from LSTM. The Bi-LSTM also have an accuracy rate of 82.2% higher than that of LSTM model of 81.41%. The Bi-LSTM starts to overfit after Epoch 33 similar to the LSTM model which starts to overfit after Epoch 31. The learning rate was adjusted in the next model to reduce the overfitting effect beyond epoch 30.

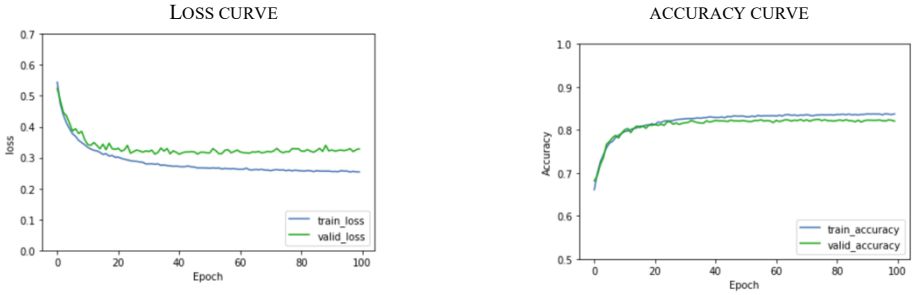


FIG.15 BI-LSTM MODEL PERFORMANCE

Gated Recurrent Unit

In this phase, the neural network was continually built the model with the use of Bidirectional Recurrent Neural Networks with GRU. GRU has no explicit memory unit and no forget gate and update gate, hence it trains the model faster than LSTM, but may lead to a lower accuracy rate. Comparing to LSTM, the GRU has a simpler architecture which reduces the number of hyperparameters. The Bi-GRU model performs slightly worse than the LSTM model. Its model highest accuracy rate (validation) is 81.06% which is 1.5% lower than the Bi-LSTM model.

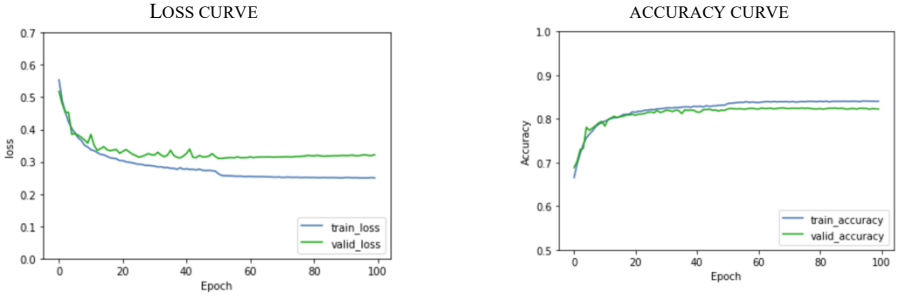


FIG.16 BI-GRU MODEL PERFORMANCE

TABLE 3 LSTM, BI-LSTM . BI-GRU MODEL COMPARISON

Model	**Loss**		**Accuracy**	
	Training	*Validation*	*Training*	*Validation*
LSTM	0.29	0.32	81.77%	81.41%
Bi-LSTM	0.28	0.31	82.62%	82.20%
Bi-GRU	0.30	0.32	81.61%	81.06%

Threshold Values for Binary Classification

The Sigmoid activation function was applied to all previous models to produce the Neural Network outputs in the last layer. The outputs are the decimal number ranges from 0 to 1, indicating the probability of the outputs to be classified as class 0 or 1 based on the chosen threshold. The previous models use the threshold of 0.5 which was applied to outputs for predicting the target class. For example, if the outputs are less than or equal to 0.5, they are classified as class 0 (Non-vulnerability functions). According to the performance of the previous models, the models suffer from some Type I and Type II errors, thus the threshold value needs to be tuned to minimize the cost of Type 1 Error and Type 2 Error. Fig 17 shows the accuracy rates and F1 scores of validation sets across different threshold values. The optimal threshold is 0.45 which has the highest accuracy rate and F1 score.

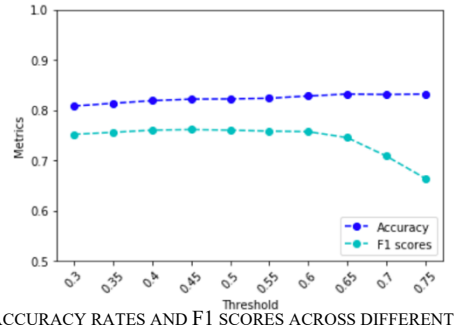


FIG.17 ACCURACY RATES AND F1 SCORES ACROSS DIFFERENT THRESHOLD

Final Model Evaluation (Bi-LSTM model)

Several neural networks were built during the model development process. The best model is Bi-LSTM model. The model contains 256 neuron units and 2 hidden layers fitted with a batch size of 32 for 50 epochs. The "Tanh" function was applied to the hidden layers and the "Sigmoid" function was applied to the output layer. The model was compiled with "Adam" optimizer and the initial learning rate of 0.001 with a balanced class weight tuning. The optimal threshold value of 0.45 was chosen for class prediction of the test set. The final model of Neural Network was used to predict the test set which indicates how well the model performs on unseen data. The model has a good performance with an accuracy rate of 81.3% , F1 score of 75% , specificity of 75%, and recall of 95% (Fig.20). The final deep learning model performs in predicting the unseen data as well as the final Random Forest model as shown on the classification metrics (Fig. 19 and 20). The Random Forest model is preferable in this classification studies as it requires the lower computation costs and can be trained faster. The model also provides the useful information of feature importance in classification.

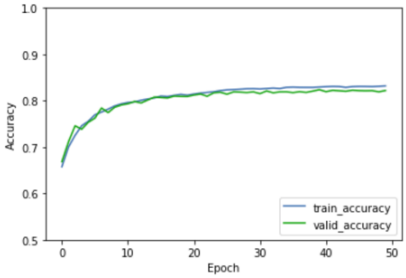


FIG.18 ACCURACY CURVE OF FINAL MODEL

Accuracy Scores (Test): 0.8085393258426966

	precision	recall	f1-score	support
0	0.97	0.75	0.85	3101
1	0.62	0.94	0.75	1349
accuracy			0.81	4450
macro avg	0.79	0.85	0.80	4450
weighted avg	0.86	0.81	0.82	4450

FIG.19 CLASSIFICATION METRICS OF BEST RANDOM FOREST (TEST SET)

Accuracy Scores (Test): 0.812808988764045

	precision	recall	f1-score	support
0	0.97	0.75	0.85	3101
1	0.63	0.95	0.75	1349
accuracy			0.81	4450
macro avg	0.80	0.85	0.80	4450
weighted avg	0.87	0.81	0.82	4450

FIG.20 CLASSIFICATION METRICS OF FINAL MODEL (TEST SET)

Conclusion

The binary programs were preprocessed to extract CFG basic blocks of each function and generate set of instructions and operands which were transformed to a vector using the CounterVectorizer model for feature extraction. Several Machine Learning algorithms including Decision Tree and Random Forest were utilized to build classifiers and identify the important features. Based on those models, the operand features pay more important role in classifying whether the binary function is vulnerable or not. In advanced model development phase, the Support Vector Machine and Deep learning algorithms were utilized to build models to predict target class (binary). The Neural Networks were trained using both traditional Long Short-Term Memory (LSTM) and Bidirectional LSTM and GRU (Bi-LSTM and Bi-GRU). Additionally, the hyperparameters such as activation function, Gate Unit, and learning rates were tuned in model development phase. The Bi-LSTM model is the best neural network which has the test accuracy rate of 81.28% (Fig. 20). For Machine Learning, the random forest is the best model with the accuracy rate of 80.85% (Fig. 19). Our study shows that Machine Learning models perform as well as Neural Networks in this classification problem but requires lower computation cost and can be trained faster.

References

[1] Zhen Li, Deqing Zou, Shouhuai Xu, Xinyu Ou, Hai Jin, Sujuan Wang, Zhijun Deng, Yuyi Zhong: "VulDeePecker: A Deep Learning-Based System for Vulnerability Detection", *NDSS 2018*

[2] Zhidong Shen, Si Chen, "A Survey of Automatic Software Vulnerability Detection, Program Repair, and Defect Prediction Techniques", Security and Communication Networks, vol. 2020, Article ID 8858010, 16 pages, 2020. https://doi.org/10.1155/2020/8858010

[3] J. Gao *et al*., "Semantic Learning and Emulation Based Cross-platform Binary Vulnerability Seeker," in *IEEE Transactions on Software Engineering*, doi: 10.1109/TSE.2019.2956932.

[4] Angr 8. https://angr.io

[5] Scikit-learn 0.24 package. https://scikit-learn.org/stable/

[6] Minaxi Arora, Lekha Bhambhu , "Role of Scaling in Data Classification Using SVM", *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 4. Issue 10 , 2014.

[7] "Gini Index. In: The Concise Encyclopedia of Statistics" Springer, New York, NY. 2008 https://doi.org/10.1007/978-0-387-32833-1_169

[8] Liu, P., Wang, X., Yin, L. *et al.* Flat random forest: a new ensemble learning method towards better training efficiency and adaptive model size to deep forest. *Int. J. Mach. Learn. & Cyber.* 11, 2501–2513 (2020). https://doi.org/10.1007/s13042-020-01136-0

[9] Matthias Ring, Bjoern M. Eskofier, "An approximation of the Gaussian RBF kernel for efficient classification with SVMs", *Pattern Recognition Letters*, Volume 84, 2016, Pages 107-113, ISSN 0167-8655, https://doi.org/10.1016/j.patrec.2016.08.013.

[10] Nielsen, M . *Neural Networks and Deep Learning* (Chapter 3), 2019 Retrieved from http://neuralnetworksanddeeplearning.com/chap3.html

[11] Dufourq, E., & Bassett, B. (2017). Automated Problem Identification: Regression vs Classification via Evolutionary Deep Networks. *Proceedings of the South African Institute of Computer Scientists and Information Technologists,* 2017. Article No.12, 1–9. 2017. https://doi.org/10.1145/3129416.3129429

[12] Kingma, D., Ba J. (2015). Adam: A Method for Stochastic Optimization. *The 3rd International Conference for Learning Representations, May 2015*

[13] Mitchell, T. (1997). *Machine Learning (pp. 96-98).* McGraw-Hill Science.

[14] Castaneda, G., Morris, P. & Khoshgoftaar, T.M. Evaluation of maxout activations in deep learning across several big data domains. *J Big Data* 6, 72 (2019). https://doi.org/10.1186/s40537-019-0233-0

[15] Calins, O. (2020). *Deep Learning Architectures A Mathematical Approach (pp. 28-30).* Springer.

[16] Schuster, M., & Paliwal, K. (1997). Bidirectional Recurrent Neural Networks. *IEEE Transactions on Signal Processing, November 1997* Retrieved from: https://doi.org/10.1109/78.650093.