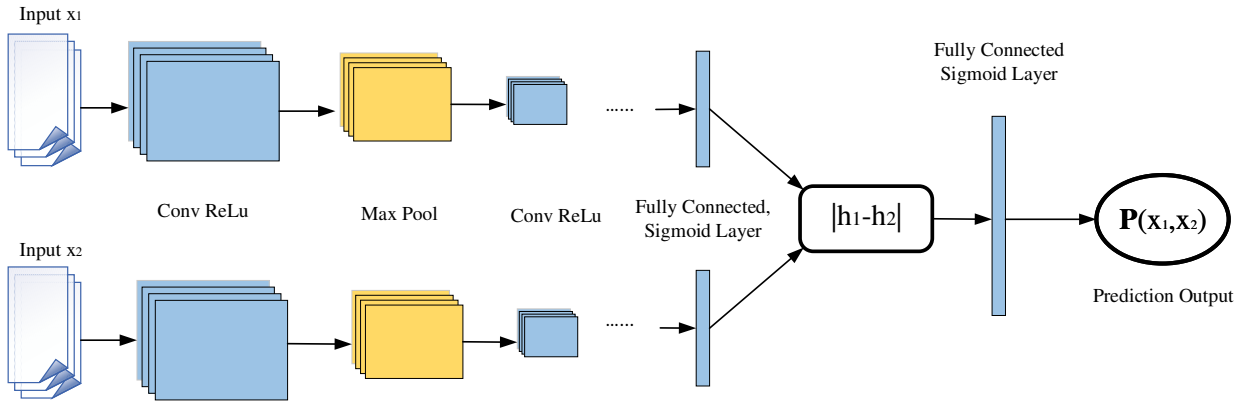

Figure 3. The architecture of Siamese Neural network-based binary code similarity detection.

classifier scenario takes the following form:

$$\mathcal{L}(x_1^{(i)}, x_2^{(i)}) = \mathbf{y}(x_1^{(i)}, x_2^{(i)})\ log\ \mathbf{P}(x_1^{(i)}, x_2^{(i)}) + (1 - \mathbf{y}(x_1^{(i)}, x_2^{(i)}))\ log\ (1 - \mathbf{P}(x_1^{(i)}, x_2^{(i)})) + \lambda^T |\mathbf{w}|^2, \tag{10}$$

in which $\lambda^T|\mathbf{w}|^2$ is the regularization term.

In terms of the update policy, we employ the standard backpropagation algorithm to optimize the parameter values. Let η_j denotes the learning rate and μ_j the momentum. The update rule at epoch T can be formularized as:

$$\mathbf{W}_{kj}^{(T)}(x_1^{(i)}, x_1^{(i)}) = \mathbf{W}_{kj}^{(T)} + \Delta\mathbf{W}_{kj}^{(T)}(x_1^{(i)}, x_1^{(i)}) + 2\lambda_j|W_{kj}|,$$
$$\Delta\mathbf{W}_{kj}^{(T)}(x_1^{(i)}, x_1^{(i)}) = -\eta_j \nabla w_{kj}^{(T)} + \mu_j \Delta\mathbf{w}_{kj}^{(T-1)}. \tag{11}$$

In which $\nabla w_{kj}^{(T)}$ denotes the partial derivative in terms of the weight between the j_{th} neuron in one layer and k_{th} neuron in the following layer.

In our following experiments, all the values of parameters within the convolutional neural network is assigned with a normal distribution of zero-mean and a standard deviation of 0.01. Bias is initialized with also a normal distribution of mean 0.5 and standard deviation 0.01. The initialization configuration is similar with those in [14] due to their similarity in terms of objective.

4. Experimental Results and Analysis

To validate the performance of our proposed framework, which using LSTM networks to embed the general features of binary files based on the local features of disassembled CFGs, and employing the Siamese neural network to perform the similarity detection, we experimented the framework on a dataset we collected by ourselves from real-world systems. The dataset includes 4000+ ELF malware executables. It contains 560 categories of binaries, and each category contains multiple binary files.

In the experiments, we implemented the framework based on TensorFlow v2.4.1 and Keras 2.4.3. The experiments are run on a Ubuntu 18.04 version. The framework is implemented using Python 3.6.9.

Local feature vector extraction. We first perform the disassembling operations on the binaries to obtain the corresponding CFGs. Then based on the graphs, we could extract the local block feature vectors, in our experiments, we build the local feature vectors as the following form:

[No. of string constants,
No. of numeric constants,
No. of arithmetic instructions,
No. of logic instructions,
No. of transfer instructions,
No. of calls,
No. of data transfer instructions,
No. of offspring,
the value of betweenness]^T

This is intuitive and easy to obtain from the disassembled CFGs, and also is widely used in existing methods[7, 9].

Based on the assigned rule, we could easily obtain the numerical forms of the local feature vectors for each block in a binary file. An example numerical features of a sequence of local blocks is shown as follows:

```
. . . . . .
4 0 0 0 1 1 1 2.0495163141498604e-05
1 10 1 0 0 1 2 2 3.415860523583101e-05
1 3 0 0 0 0 1 1 6.968355468109526e-05
0 1 0 0 0 1 0 1 0.00012570366726785811
1 6 0 0 0 0 2 2 6.0119145215062575e-05
```

```
0 2 0 0 0 0 0 1 4.9188391539596654e-05
0 1 0 0 0 1 0 1 0.00018172377985462097
1 8 0 0 0 0 3 2 6.968355468109526e-05
0 4 0 0 0 0 0 1 2.4594195769798327e-05
0 2 0 0 0 0 0 1 0.00011750560201125867
                ......
```

The local feature vectors extracted above represent both the block-level attributes information and inter-block level attributes information. As we can observe from local feature vectors, they are changing at each block. To obtain an indexable representation of the binary that contains multiple blocks, we feed this sequence of local feature vectors into the LSTM networks.

LSTM network-based embedding. To embed the obtained local feature vectors, we first need to truncate the blocks we have as we want to train each of the binary files on the same number of local features. However, the real-world binaries we collected are vary each other in terms of the length. We employ a strategy of choosing a fixed number of the blocks from each type of the binary file in a random way but without disrupting the order of the chosen sequence.

We form the embedding problem as a regression problem, in which the input is the feature vector at time t, while the output is the feature vector at time $t+1$. However, to make the prediction process could take more of the past time slots into consideration, we employ multiple previous time slots as the input to predict the following feature vector of the block. We found that if too many previous time slots are considered, then the prediction output will be less reflective regarding the variation and the number of training samples will decrease dramatically. There is a tradeoff balance between the flexibility and accuracy.

In the experiment, we employ 5 previous feature vectors as the input to train the model, and we set the dropout value as 0.1, and the training epoch is set as 100. At each prediction. The LSTM blocks or neurons we adopt is set as 16. In Fig. 4, we shown the relationship of the averaged Mean Squared Error (MSE) with the training epochs of three representative binaries, from which we can observe that when the training epochs approaching 70, the averaged MSEs for all three binaries are become stable. Thus, we could output the hidden cell state vectors as the embeddings for those binary files.

Siamese neural network-based similarity detection. After we obtained the embeddings of the binaries, we have the indexable representation of each binary file. Thus, the remaining work is to feed them into a Siamese neural network and perform the similarity detection.

In this group of experiments, we focused on classifying a test binary to the pre-trained binary categories. We chose 10 categories of the binaries that have the largest number of binaries from our collected dataset as the training categories. As some categories include only a very small number of binaries, thus we exclude them in our experiment such that we have enough binaries under each category that could be divided into a training set and a testing set. For the testing binaries, we'll include both the binaries from the training categories and from those categories that are not included in the training process except specially specified.

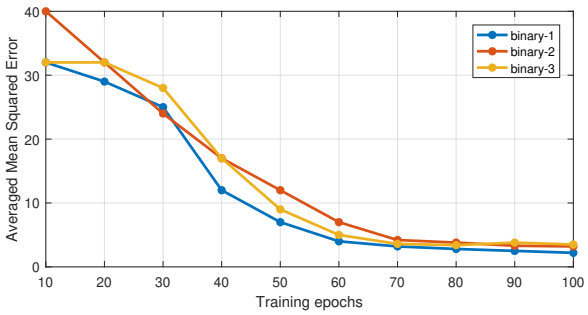

Figure 4. The relationship of the training epochs with the averaged MSE.

Since in a Siamese neural network, the weights from both sub-networks are supposed to be identical with each other, thus we use only one model and feed two inputs in succession. The optimization process or the backpropagation is conducted after we calculate the loss for two inputs. We build the Siamese neural network includes 3 fully connected convolutional layers with a structure of $16 \times 32 \times 16$, and the dropout value of 0.1.

We first show the experimental results of the training performance using only test data from the categories that we have trained on. We define a measure named as detection accuracy to describe the performance, which is defined mathematically as:

$$\text{Detection Accuracy} = \frac{\text{\# of test binaries rightly classified}}{\text{\# of total test binaries}} \tag{12}$$

We use half of the binaries within each category as the training data and the other half as the testing data, the relationship of the detection accuracy with the training epochs is shown in Fig.5. From the results we know that after training the Siamese neural network with at least 60 epochs, we can achieve the detection accuracy of around 90%, which validates the performance of the framework.

We also compared our proposed framework with existing methods as shown in [7, 10], which are representative methods to detect malware. [10] focuses on a graph embedding network to convert the graph into embeddings for binary functions, while [7] focuses on the maximum common subgraph isomorphism to

Table 1. The comparison of our proposed LSTM+Siamese neural network-based framework with existing works [7, 10].

	Our proposed method	Xu et al.[10]	Eschweiler et al.[7]
Detection Accuracy	0.85	0.85	0.83
Relative time cost for training	1.00	1.24	0.78
Relative time cost for testing	1.00	1.15	0.67

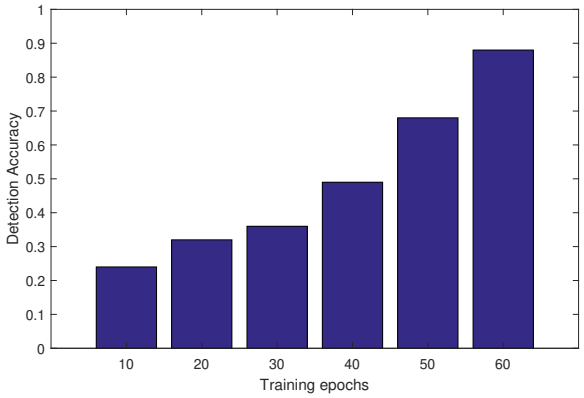

Figure 5. The relationship of detection accuracy with training epochs

measure the structural similarity between two different binaries.

We compared the performance and relative time cost of the three methods and the results are shown in the Table 1. Both the time cost for training and testing are compared with our proposed method in a relative way, which means our proposed method is set as a benchmark with the value of 1. From the comparison results we know that our method shows a higher training efficiency while still maintaining similar detection accuracy performance with method [10]. It is worth noting that here our detection accuracy is 85%, as the testing data also includes binaries from other categories of the dataset, which is differ from the previous 90% of the detection accuracy, in which the testing data is from the same categories of the training data.

Compared with [7], our method showed a slightly better performance though suffer from the training of two neural networks which cost a little bit more time. Compared with our proposed LSTM + Siamese neural network combination, the methods in [10] require to train a deep graph embedding neural network, while the methods in [7] require a graph similarity algorithm based on the maximum common subgraph isomorphism.

5. Conclusion

Binary code similarity detection plays an important role in evaluating the security of a software project that are closed-source. It also offers many other applications such as plagiarism and malware detection.

In this paper, we proposed an LSTM neural network-based method to obtain an indexable embedding from the dissembled control flow graphs of binary files. Also, we employed Siamese neural network to conduct the similarity comparison of two embeddings due to that Siamese neural network has the capability of learn the semantic information embedded in the inputs. Our experimental results show a promising performance compared with existing methods both in terms of detection accuracy and computation cost.

In our future work, we will focus on extracting more representative local features from the disassembled blocks of the binaries. Also, how to combine the LSTM network and Siamese Neural Network to make them work closely to reduce the training complexity will be one of another future work.

Acknowledgement. This material is based upon work supported by the U.S. Department of Energy, Office of Science under Award Number DE-SC0018476. Zhengping Luo was with University of South Florida when participating in this work.

References

[1] Liu, B., Huo, W., Zhang, C., Li, W., Li, F., Piao, A. and 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*: 667–678.

[2] Luo, L., Ming, J., Wu, D., Liu, P. and Zhu, S. (2017) Semantics-based obfuscation-resilient binary code similarity comparison with applications to software and algorithm plagiarism detection. *IEEE Transactions on Software Engineering* **43**(12): 1157–1177.

[3] Yu, Z., Cao, R., Tang, Q., Nie, S., Huang, J. and Wu, S. (2020) Order matters: semantic-aware neural networks for binary code similarity detection. In *Proceedings of the AAAI Conference on Artificial Intelligence*, **34**: 1145–1152.

[4] Jaramillo, L. (2018) Malware detection and mitigation techniques: lessons learned from mirai ddos attack. *Journal of Information Systems Engineering & Management* **3**(3): 19.

[5] Chen, L., Ye, Y. and Bourlai, T. (2017) Adversarial machine learning in malware detection: Arms race between evasion attack and defense. In *2017 European Intelligence and Security Informatics Conference (EISIC)* (IEEE): 99–106.

[6] Grieco, G., Grinblat, G.L., Uzal, L., Rawat, S., Feist, J. and Mounier, L. (2016) Toward large-scale vulnerability discovery using machine learning. In *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy*: 85–96.

[7] Eschweiler, S., Yakdan, K. and Gerhards-Padilla, E. (2016) discovre: Efficient cross-architecture identification of bugs in binary code. In *NDSS*, **52**: 58–79.

[8] Pewny, J., Garmany, B., Gawlik, R., Rossow, C. and Holz, T. (2015) Cross-architecture bug search in binary executables. In *2015 IEEE Symposium on Security and Privacy* (IEEE): 709–724.

[9] Feng, Q., Zhou, R., Xu, C., Cheng, Y., Testa, B. and Yin, H. (2016) Scalable graph-based bug search for firmware images. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*: 480–491.

[10] Xu, X., Liu, C., Feng, Q., Yin, H., Song, L. and Song, D. (2017) Neural network-based graph embedding for cross-platform binary code similarity detection. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*: 363–376.

[11] Shin, E.C.R., Song, D. and Moazzezi, R. (2015) Recognizing functions in binaries with neural networks. In *24th {USENIX} Security Symposium ({USENIX} Security 15)*: 611–626.

[12] Graves, A., Fernández, S., Gomez, F. and Schmidhuber, J. (2006) Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd international conference on Machine learning*: 369–376.

[13] Lafferty, J., McCallum, A. and Pereira, F.C. (2001) Conditional random fields: Probabilistic models for segmenting and labeling sequence data .

[14] Koch, G., Zemel, R. and Salakhutdinov, R. (2015) Siamese neural networks for one-shot image recognition. In *ICML deep learning workshop* (Lille), **2**.

[15] Staudemeyer, R.C. and Morris, E.R. (2019) Understanding lstm–a tutorial into long short-term memory recurrent neural networks. *arXiv preprint arXiv:1909.09586* .

[16] Graves, A. (2013) Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850* .

[17] Graves, A., Liwicki, M., Fernández, S., Bertolami, R., Bunke, H. and Schmidhuber, J. (2008) A novel connectionist system for unconstrained handwriting recognition. *IEEE transactions on pattern analysis and machine intelligence* **31**(5): 855–868.

[18] Shukla, S., Kolhe, G., PD, S.M. and Rafatirad, S. (2019) Stealthy malware detection using rnn-based automated localized feature extraction and classifier. In *2019 IEEE 31st international conference on tools with artificial intelligence (ICTAI)* (IEEE): 590–597.

[19] Bromley, J., Guyon, I., LeCun, Y., Säckinger, E. and Shah, R. (1994) Signature verification using a" siamese" time delay neural network. *Advances in neural information processing systems* : 737–737.

[20] Zhang, C., Liu, W., Ma, H. and Fu, H. (2016) Siamese neural network based gait recognition for human identification. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (IEEE): 2832–2836.

[21] Chicco, D. (2021) Siamese neural networks: An overview. *Artificial Neural Networks* : 73–94.

[22] Berlemont, S., Lefebvre, G., Duffner, S. and Garcia, C. (2018) Class-balanced siamese neural networks. *Neurocomputing* **273**: 47–56.

[23] https://angr.io/.