An Idea of Intermediate Language Memory Conflict Detection Based on Bi-LSTM

Zeyao Xu [1 *], Letian Sha [2], Yuye Wang [3] and Haotian Zhang [4]

Nanjing University of Posts and Telecommunications, South, 210023, China

Email: 623025516@qq.com

Keywords: Deep learning; Vulnerability detection; Bi-LSTM; Cross-platform; Intermediate language

Abstract: The mining of binary program memory vulnerabilities has always been one of the main directions of software security research. The ideas provided in the article include extracting vulnerability characteristics based on intermediate languages, performing cross-platform binary analysis through angr's simulated state management, serializing with Word2Vec, and then using Bi-LSTM deep learning algorithms to build a memory conflict model of the binary program, testing the binary The program analyzes and finds memory conflicts, and finally finds the overflow point for verification through the dynamic symbol execution of angr, so as to find the existence of the vulnerability. Experiments have been performed with the three types of collected vulnerabilities, and the feasibility of the integration method has been verified.

1. Introduction

Vulnerability detection technology is an important method to improve software quality security and reduce software security vulnerabilities. In order to automate vulnerability discovery, machine learning (ML) -based vulnerability detection technology has attracted widespread attention.

Symbol execution, stain tracking, fuzzing, etc. are all commonly used methods in vulnerability mining. Traditional symbolic execution techniques only target source code and do not run programs. Dynamic symbol execution makes up for some of its shortcomings, but as the program scale grows, it often faces the problem of path explosion. Dynamic stain tracking technology can be separated from the program source code, but requires a lot of running memory. Fuzzing can be implemented quickly and the false alarm rate is low, but it is difficult to traverse different state spaces due to the data integrity check mechanism.

Compared with the traditional vulnerability mining methods, the new vulnerability mining technology using machine learning has significant advantages in efficiency, accuracy and application scope. Among them, the combination of artificial neural tree (ANN) and vulnerability mining [1] proved to have significant effects. Deep learning is pre-trained in a multi-layer self-encoding neural network based on the traditional ANN, which improves efficiency and data capacity. Since 2012, small-scale recurrent neural networks based on deep learning [2] (reduced-size RNNs) and bidirectional LSTM (Bi-LSTM) [3] networks have been used for efficient vulnerability analysis experiments, but the latter is only for source code.

Currently, machine learning-based vulnerability mining technologies are mainly focused on component or file level detection, which still requires a large amount of manual participation. The use of source code detection also limits the general performance of the model across platforms. Because binary programs lack semantics such as functions and variable types Information, source-oriented techniques cannot be used with binary programs. On the other hand, the false positive rate of vulnerability analysis in static analysis is high, and the false negative rate in dynamic analysis is high.

Intermediate Representation (IR) [4] is a language with both source code and assembly language characteristics. It can be used for both static and dynamic analysis with the support of specific tools (such as angr). This paper proposes a memory conflict detection method based on the Bi-LSTM

  DOI: 10.25236/icme.2019.065

algorithm using an intermediate language.

2 Method Workflow Analysis

2.1 Generate training samples

2.1.1 Translate the original sample into vex IR

We use a large number of published vulnerabilities and their patches (source code), compile them into binary executable files, and then translate them into vex IR. VEX IR is an arch-agnostic intermediate language generated by the memory leak detection tool Valgrind [5]. It consists of many IR SuperBlocks (IrSB). Each block contains 1 to 50 vex instructions. The vex instruction includes information such as variable types, status values, and jump addresses, and can describe operations such as memory reads and writes and arithmetic operations. Take x86 machine instructions addl% eax% ebx as an example, the corresponding IrSB is shown in Table 1:

Table 1 VEX translation result of an x86 instruction

--- IMark(0x24F275,4,0) ---	#(address,rows,offset)
t3 = GET:I32(0)	# get %eax, a 32-bit integer
t2 = GET:I32(12)	# get %ebx, a 32-bit integer
t1 = Add32(t3,t2)	# addl
PUT(0) = t1	# put %eax

2.1.2 Serialization into input vector

angr is a platform-agnostic python framework for analyzing binary files [6]. Simulation Managers is one of its main functions. Importing Valgrind's libvex library into angr can traverse the state space of binary files on any platform and generate VEX IR. Angr can also use pyvex to encapsulate the vex statement internally and express the semantics concisely, thereby providing great convenience for filtering irrelevant instructions and refining vex IR. Referring to the Valgrind document, we finally selected 1100 related operation instructions for the process description, allowing their IR blocks to be combined in different orders to form a text sequence that uniquely identifies various machine instructions, as shown in Table 2.

Table 2 VEX IR keyword extraction example

IrSB	Keyword
--- IMark(0x24F275, 7, 0) ---	IMark
t3 = GET:I32(0)	WrTmp, GET:I32
t2 = GET:I32(12)	WrTmp, GET:I32
t1 = Add32(t3,t2)	WrTmp, Add32
PUT(0) = t1	PUT, WrTmp
Output : [IMark,WrTmp,GET:I32,WrTmp,GET:I32,PUT,…]	

We selected three types of vulnerabilities (stack overflow danger function, format string, UAF) from a large amount of data according to vex semantic classification. Based on the text sequence, it is also necessary to construct a digital vector that can be used as the input of the ML algorithm. We initially uniquely mapped these instruction elements to integers from 1 to 100 to identify each text element, and mapped operations with similar logical relationships to related Adjacent integers are

used to express semantics (see Figure 1). But on this basis, we consider that in reality, the different execution order (ie, timing relationship) of the same operation in the execution flow is also the key to affecting whether it is a vulnerability.

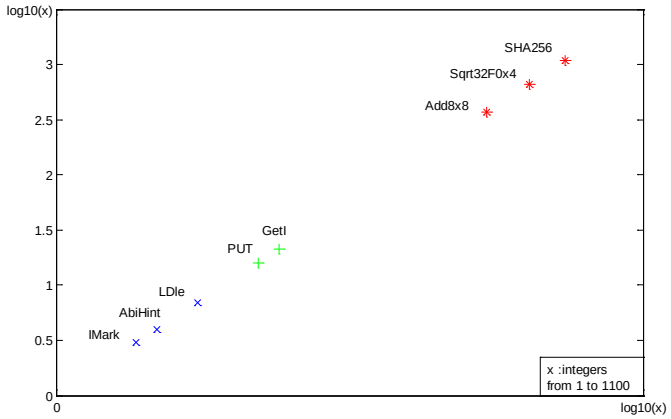


Figure 1 Mapping relationship between VEX keywords and integers

Word2vec is a neural network model used to generate word vectors. We use the VEX syntax definition and a large number of VEX original texts as Word2vec's corpus, and have implemented a two-dimensional array representing both time series and logic to represent vex instructions. The visual representation is shown in Figure 2. Statements with similar execution times and statements with similar logical relationships are adjacent.

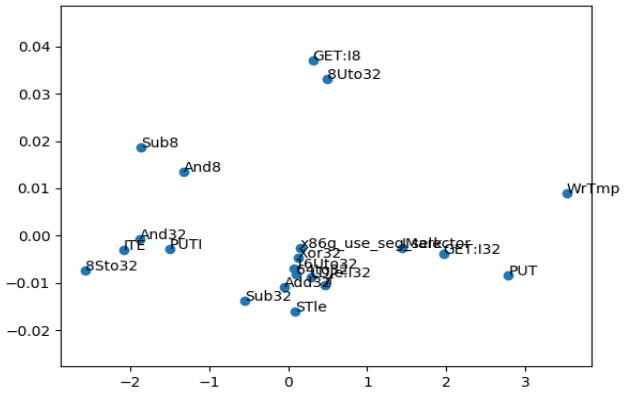


Figure 2 Vector projection illustration

2.2 The Bi-LSTM neural tree model

Since the digitized input has two dimensions, we must adopt an algorithm that is suitable for expressing the function of time series relations. When the RNN processes the sequence in time series, it simply adds future information to each point in the sequence by delaying one-way. The bidirectional long-short-term memory recurrent neural network (Bi-LSTM [7]) can capture the complete past and future information of each point in the input sequence, and can capture the long-term program logic and the temporal relationship formed by the semantic structure, thereby more Effectively capture vulnerable programming patterns.

Train Bi-LSTM model under Keras framework. Take the vector (csv file) obtained in 2.1.3 as input, the data set size is 1633, of which 1249 are used as training set (76%), and 384 are used as validation set (24%). Indicates that the number of flags is set to 294, batch_size = 64, epochs = 55; binary_crossentropy is used for the loss function, rmsprop and verbose = 2 are optimized for model training, and one record is output for each epoch (Figure 3).

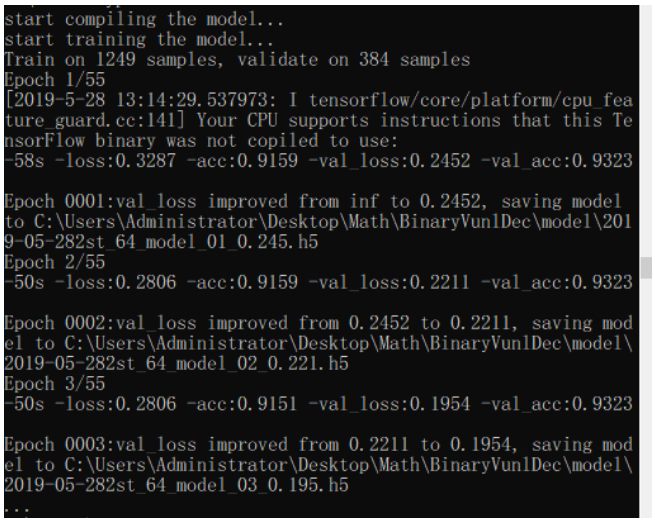


Figure 3 Screenshot of the running process

The details of the output Bi-LSTM model are shown in Figure 4. Our model has a total of 5 layers, the first two layers output 3D tensors, and the last 3 layers output 2D tensors. Activations represents the data dimension after each layer. Param in the third column represents the training amount.

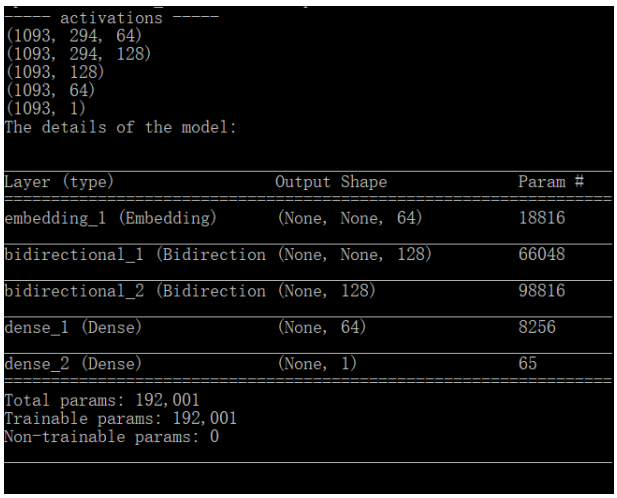


Figure 4 Model details display

2.3 Model prediction performance test

We finally selected Sqlite (old version) with a certain number of vulnerabilities and a code size of about 200,000 lines to simulate memory conflict analysis.

Sqlite is converted into a keyword sequence according to the same process of generating Bi-LSTM training samples. At this time, a function name is added to the instruction header and backed up, as shown in Table 3. It is then serialized and predicted using the Bi-LSTM model.

Table 3 Sqlite function VEX IR keyword serialization (partial display)

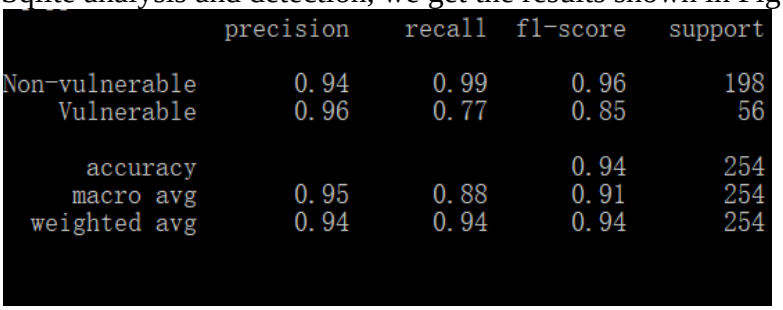
	A	B	C	D	E	F	G	H
1	process_sqliterc	IMark	WrTmp	GET:I32	WrTmp	WrTmp	WrTmp	Add32
2	main	IMark	WrTmp	Add32	PUT	LDle:I32	PUT	WrTmp
3	UNKNOWN_FUNCTI	IMark	WrTmp	Add32	PUT	LDle:I32	PUT	WrTmp
4	NAME7R6F	IMark	WrTmp	GET:I32	PUT	WrTmp	WrTmp	LDle:I32
5	printBold	IMark	WrTmp	Add32	STle	WrTmp	PUT	WrTmp
6	find_home_dir	IMark	WrTmp	Add32	WrTmp	LDle:I32	WrTmp	WrTmp
7	process_input	IMark	WrTmp	WrTmp	PUT	LDle:I32	PUT	WrTmp
8	one_input_line	IMark	WrTmp	GET:I32	STle	WrTmp	PUT	WrTmp

Another main function of the angr framework is the vulnerability detection combined with static

and dynamic symbol execution [6], and angr can remove unwanted paths through parameter settings to avoid path explosion. After predicting the model, the data block where the memory conflict is found is obtained. Based on the function name backed up before, the analysis function of angr is used to find the specific overflow point of the vulnerability.

3. The Experimental Results and Analysis

For the accuracy of Sqlite analysis and detection, we get the results shown in Figure 5:



	precision	recall	f1-score	support
Non-vulnerable	0.94	0.99	0.96	198
Vulnerable	0.96	0.77	0.85	56
accuracy			0.94	254
macro avg	0.95	0.88	0.91	254
weighted avg	0.94	0.94	0.94	254

Figure 5 Memory leak detection results for Sqlite

We made predictions on 254 data blocks, including 198 Non-vulnerable and 56 vulnerable data. According to Figure 10, 43 vulnerable data blocks were accurately predicted, 13 were missed, 2 were false positives, precision was 0.96, recall was 0.77, and F1-score was 0.85. The data proves that the test results of this method are more accurate under certain error tolerance. The prediction false alarm rate for memory conflict is low, but the false negative rate is relatively high. Considering the small amount of vulnerable data in this experiment, under the larger vulnerability data, the model established by this method should obtain more ideal results.

Conclusion

It has been verified that the deep learning-based cross-platform memory conflict detection method proposed for software binary vulnerabilities proposed in this article is relatively common based on file or function level methods and can perform deep learning-based detection with finer granularity. While traversing the execution flow, more accurately and efficiently locate the location of memory conflicts, greatly reducing the scope of vulnerability detection, thereby greatly reducing the target scale of dynamic symbol execution. Has good application prospects and development space.

References

[1] B. Ingre and A. Yadav. Performance analysis of NSL-KDD dataset using ANN. In Proc. Int. Conf. Signal Process. Commun. Eng. Syst., Jan. 2015, pp. 92-96.

[2] M. Sheikhan, Z. Jadidi, and A. Farrokhi. Intrusion detection using reduced-size RNN based on feature grouping. Neural Comput. Appl., vol.21, no.6, pp. 1185-1190, sep.2012.

[3] Guanjun Lin, Jun Zhang, Wei Luo, and Lei Pan. Vulnerability discovery with function representation learning from unlabeled projects. Presented at the CCS'17, Oct 30-Nov3, 2017, Dallas, TX, USA., pp. 19-49.

[4] Kim S, Faerevaag M, Jung M, et al. Testing intermediate representations for binary analysis[C]//Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering. IEEE Press, 2017: 353-364.

[5] Nethercote N, Seward J. Valgrind:a framework for heavyweight dynamic binary instrumentation[J]. Acm Sigplan Notices, 2007, 42(6):89-100

[6] Analysis an doptimization of Angr in dynamic software test application[J]. Computer Engineering and Science; 2018 Issue z1. pp.163-168

[7] Huang Z, Xu W, Yu K. Bidirectional LSTM-CRF models for sequence tagging[J]. arXiv preprint arXiv:1508.01991, 2015