


# Cyber Vulnerability Intelligence for Internet of Things Binary

Shigang Liu , *Member, IEEE*, Mahdi Dibaei , Yonghang Tai, Chao Chen ,  
Jun Zhang , *Member, IEEE*, and Yang Xiang , *Senior Member, IEEE*

**Abstract**—Internet of Things (IoT) integrates a variety of software (e.g., autonomous vehicles and military systems) in order to enable the advanced and intelligent services. These software increase the potential of cyber-attacks because an adversary can launch an attack using system vulnerabilities. Existing software vulnerability analysis methods used to be relying on human experts crafted features, which usually miss many vulnerabilities. It is important to develop an automatic vulnerability analysis system to improve the countermeasures. However, source code is not always available (e.g., most IoT related industry software are closed source). Therefore, vulnerability detection on binary code is a demanding task. This article addresses the automatic binary-level software vulnerability detection problem by proposing a deep learning-based approach. The proposed approach consists of two phases: binary function extraction, and model building. First, we extract binary functions from the cleaned binary instructions obtained by using IDA Pro. Then, we employ the attention mechanism on top of a bidirectional long short-term memory for building the predictive model. To show the effectiveness of the proposed approach, we have collected datasets from several different sources. We have compared our proposed approach with a series of baselines including source code-based techniques and binary code-based techniques. We have also applied the proposed approach to real-world IoT related software such as VLC media player and LibTIFF project that used on Autonomous Vehicles. Experimental results show that our proposed approach betters the baselines and is able to detect more vulnerabilities.

**Index Terms**—Binary code, deep learning, machine learning, software vulnerability.

## I. INTRODUCTION

NOWADAYS, computer software deployed on devices, such as autonomous vehicles and military systems, are crucial parts of the industry Internet of Things (IoT) equipment [1], [2]. In a routine day, users depend on myriad software components running on different platforms and ranging from simple applications on hand-held devices to complicated industry software and critical embedded systems. Vulnerabilities in software cause serious damage and losses [3], [4]. For example, an adversary can attack the communication channels and sensors of autonomous vehicles, which may cause significant instability in the cooperative adaptive cruise control vehicle stream [5], [6]. Hence, automatic vulnerability discovery [7] is attracting a lot of attention in various IoT equipment especially autonomous vehicles and military systems.

Binary code analysis is a hot topic in the area of computer security [8] given that for many software projects the original high-level source code is either closed-source or inconvenient to use. First, the source code might be not available. For instance, military autonomous system and autonomous driving unlikely will release their source codes [9]. Second, compilation and optimisation may alter the structure of the compiled program (e.g., by reordering independent operations or eliminating dead code, or by taking advantage of undefined behavior in the source program) creating a mismatch between the source code and the compiled binary code [10]. Such optimisations can even introduce additional security problems. For example, to prevent a buffer which contained sensitive information being read by attackers, the buffer should be zeroed immediately after use. However, the apparently redundant code to zero the buffer may be optimised out by certain compilers [11]. Third, according to [12], testing results through binary-level code analysis can contribute to execution traces. Finally, the binary-level analysis is also a key building block in many specific areas such as binary-level software vulnerability detection, malware detection, function recognition, code clone detection, and security vulnerabilities of advanced driver assistance systems. Research interest has shown that autonomous vehicles are facing a heightened risk of cyber threat with more and more vehicles are being developed and sold [9]. Therefore, binary analysis is an attractive and widely applicable field not only because source code is not required but also because it can avoid the mismatch between source code and compiled binary code.

Manuscript received July 23, 2019; accepted August 24, 2019. Date of publication November 6, 2019; date of current version January 16, 2020. Paper no. TII-19-3281. (*Corresponding author: Yonghang Tai*).

S. Liu, C. Chen, J. Zhang, and Y. Xiang are with the School of Software and Electrical Engineering, Swinburne University of Technology, Hawthorn, VIC 3122, Australia (e-mail: shigangliu@swin.edu.au; chaochen@swin.edu.au; junzhang@swin.edu.au; yxiang@swin.edu.au).

M. Dibaei is with the Department of Computing, Macquarie University, Sydney, New South Wales 2109, Australia (e-mail: dibaeimahdi@yahoo.com).

Y. Tai is with the Yunnan Key Laboratory of Optoelectronic Information Technology, Yunnan Normal University, Kunming, Yunnan 650101, China (e-mail: taiyonghang@ynnu.edu.cn).

Color versions of one or more of the figures in this article are available online at <http://ieeexplore.ieee.org>.

Digital Object Identifier 10.1109/TII.2019.2942800

Static analysis [13]–[16] and dynamic analysis [17]–[21] and concolic execution (i.e., dynamic symbolic execution) [22] are popularly used techniques for vulnerability discovery. Concolic (and symbolic) approaches suffer from poor scalability due to path explosion, which can be an issue given that, for example, the operating system Debian has more than 30 000 programs (with 80 000 bug reports). Static analysis (the process of analyzing the code without executing the program) is also time-consuming and limited in the vulnerabilities that may be discovered. For example, it takes an average of 286.1 min to perform a binary search given 10 000 basic blocks when using the semantic similarity method [15].

To deal with these problems, machine learning especially deep learning has been employed to automatically detect software vulnerabilities [23]–[27]. However, previous works such as [23] and [24] are only designed for source code level vulnerability detection. As far as we know, other works only perform vulnerability detection based on assembly code or opcodes rather than binary code. Our experimental results also show that these techniques cause a very high false negative rate (more than 65%) when there are limited numbers of vulnerable functions in the training data. This motivated us to develop a novel and more efficient approach for vulnerability detection that works on binary code only.

In this article, we aim to employ the state-of-the-art, i.e., an attention model [28], to perform vulnerability detection at function level based on binary code only. From a higher level, given binary code, we first disassemble the code. Then, we identify a set of vulnerable and nonvulnerable functions as ground truth. The assumption of the ground truth is based on the publicly available vulnerability data repositories: Common Vulnerability and Exposures (CVE) and National Vulnerability Database (NVD), until 20th July 2017. Specifically, the function will be treated as vulnerable if the function contains at least one vulnerability, otherwise, it will be treated as nonvulnerable [23]. The ground truth will feed to attention neural networks for model building. The model is then used to predict whether a given binary function is vulnerable or not. To show the effectiveness of our proposed strategy, we set up a series of comparative experiments. We conduct experiments based on datasets collected from various resources such as the NVD<sup>1</sup> and CVE.<sup>2</sup> Experimental results show that our proposed strategy manages to predict vulnerabilities at function level with a true positive rate of more than 80% in most cases (except for LibTIFF dataset). In this article, a vulnerable function refers to a function that contains at least one vulnerability recorded in the NVD or CVE dataset. Without any recorded vulnerability in a function, the function will be treated as nonvulnerable. In summary, our main contributions are as follows.

- 1) We develop an automatically deep learning-based system for real-world IoT related software vulnerability detection based on binary code.
- 2) We propose to make use of binary instructions as features at the function level. Therefore, the granularity of

our proposed strategy can identify vulnerabilities at the function level.

- 3) We propose to employ the attention model for deep feature learning given binary code, since the vulnerability is related to certain binary instructions rather than the whole binary.
- 4) We compare our proposed approach with different baselines including both source code and binary code-based machine learning techniques. Our experiments show that our deep learning-based technique is able to learn rich representations based on binary instructions.

## II. RELATED WORK

Security issues in IoT have been a hot topic in the community [29]–[34]. For example, Xiao *et al.* [33] researched the machine learning techniques for IoT security solutions including the software attacks. Kumar and Chebrolu suggested to consider hashing techniques and authentication when updating the software [35]. Al-Turjman and Altrjman [34] studied the smart home and discussed the system requirements, architectures, practical challenges, and the related deployment aspects. In this section, we specifically recall recent machine learning-based techniques for binary level vulnerability detection because they are closely related to this article.

Recently, machine learning techniques have been applied to software vulnerability detection. The main idea of applying machine learning techniques for software vulnerability detection includes two stages: feature selection and model building. Feature selection is quite different from work to work. DiscovRE [26] makes use of the structure and the related control flow graph (CFG) for function similarity calculation in order to detect bugs. Grieco *et al.* [25] developed a tool named VDiscover that used lightweight static and dynamic features in terms of the C standard library for software vulnerability detection. Then, machine learning methods have been employed to evaluate the effectiveness of the extracted features. Li *et al.* [20] proposed a prototype named SemHunt for predicting vulnerable functions and their pairs (unpatched-function and patched-function) based on binary executable programs. Eschweiler *et al.* [26] emphasized the importance of security-critical vulnerability detection in binary code. They presented a new approach named discovRE, to efficiently search for similar functions in binary code. The main idea of their work is to compute the similarity between functions based on the structure of the corresponding CFGs.

Meanwhile, the rapid rise of deep learning is in part due to an ability to learn feature representations and complex nonlinear structure in datasets. Deep learning as a branch of machine learning has achieved particular successes in domains such as vision, speech, and natural language, which each exhibit hierarchies of patterns at fine to coarse scales. Application to cybersecurity may expect similar success owing to its complex, hierarchical, nonlinear detection tasks. Lee *et al.* [36] proposed to use deep learning for software vulnerability detection. From a high level, the proposed method requires opcodes as input, therefore, one needs to disassemble the binary code first.

<sup>1</sup>[Online]. Available: <https://nvd.nist.gov/>

<sup>2</sup>[Online]. Available: <https://cve.mitre.org/>

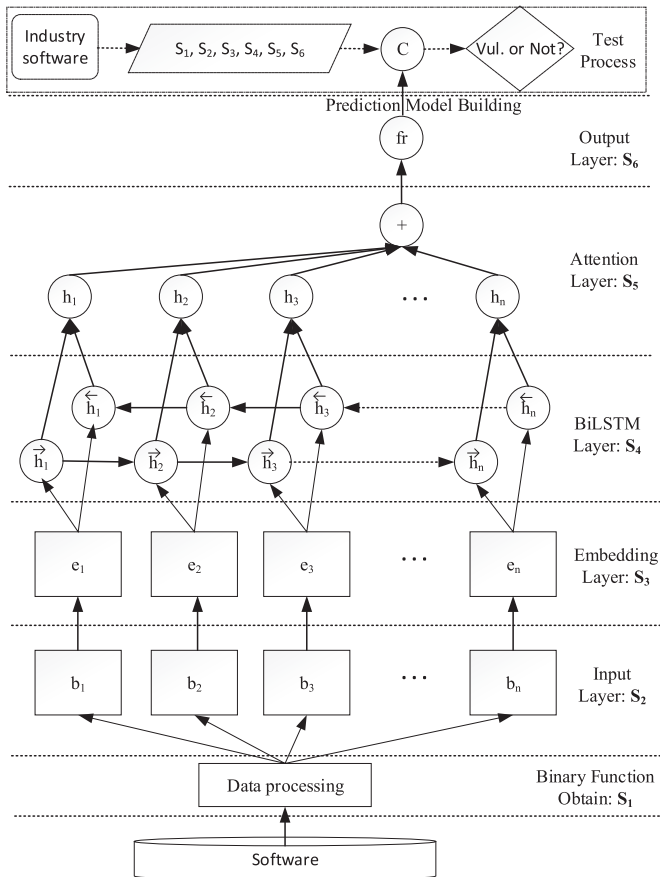


Fig. 1. Overview of the proposed framework.

Then, training data is produced by processing the opcodes by using the Instruction2vec method. Instruction2vec is a method that aims to process the instructions of the opcodes into a vector representation. Finally, Text-CNN is employed to find potential software vulnerabilities based on the training data.

### III. OVERVIEW OF THE PROPOSED APPROACH

This article aims to design a deep learning-based system that can automatically detect vulnerabilities given binary code only. We start from known vulnerabilities (vulnerable functions) and then build the prediction model using these binary vulnerabilities. We employ the attention neural network for building the prediction model. Fig. 1 presents a high-level overview of our proposed system. There are five layers in the proposed scheme: input layer, embedding layer, bidirectional long short-term memory (BiLSTM) layer, attention layer, and output layer.

The input layer requires binary instructions from each function as the original input. In this article, we employ (IDA Pro) to obtain binary instructions from a binary level software. After that, we make use of prior domain knowledge to identify whether a function is vulnerable or not. We extract functions from the binary instructions, labeling them as vulnerable or not vulnerable. The binary instructions of each function will be treated as binary features and fed to the embedding layer, then BiLSTM layer and attention layer will be employed for

high-representation feature learning. The output layer produces the learned high feature representations  $fr$ . We hypothesise that deep learning could automatically learn useful feature representations that contain much richer information than the shallow features driven by domain knowledge.  $fr$  will feed to a fully connected neural network for generating the prediction model.

When an industry software comes, it will first be processed by IDA Pro for binary instruction generation. Then, binary level functions will be extracted. Each function will be fed to the high-level feature representation learning model and then, prediction model, and the model will tell whether it is vulnerable or not.

### IV. METHODOLOGY

In this section, we present our proposed approach in detail by focusing on data preparation and model building.

#### A. Binary Function Obtain (Datasets)

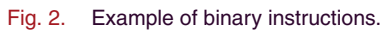
In order to study the binary level vulnerability detection problem, we need to collect various binaries from different sources. Moreover, in order to create ground truth, we need to know the type of vulnerabilities at the function level. This is possible if we collect the data or project from the NVD and CVE.

1) *Open Source Projects from NVD*: We collected two open source projects, which are VLC media project and LibTIFF.<sup>3</sup> VLC is a free and open source cross-platform multimedia player and framework that plays most multimedia files. It is one of the world's most popular free and open source communications frameworks which allows one to build communications applications. LibTIFF provides support for processing the tag image file format (TIFF), which is a widely used format for storing image data. These two projects are commonly deployed on IoT devices, such as autonomous vehicles and smart TVs for supporting multimedia functionality. We can identify the location of the vulnerabilities using CVE. For example, from the vulnerability categorization provided by NVD, we can see there is a buffer overflow vulnerability with CVE ID: CVE-2018-12900. The description shows that this is a heap-based buffer overflow in the `cpSeparateBufToContigBuf` function in `tiffcp.c`. Therefore, we treat the `cpSeparateBufToContigBuf` function as a buffer overflow vulnerability. We manually identified 26 vulnerable functions in the LibTIFF project and 36 vulnerable functions in the VLC project. We treat other functions as nonvulnerable functions. We use 64-bit versions of GCC to compile the LibTIFF and VLC projects under Windows.

2) *Open Source Project From CWE*: We also consider another dataset from CWE119, which are buffer errors. With attention to the limited memory of IoT devices, buffer errors are significant problems for IoT related software and this problem needs to be addressed urgently. We choose this dataset because it contains buffer errors from IoT related software. Moreover, this dataset contains vulnerable functions and the patched functions. It is created to demonstrate that the proposed approach can

<sup>3</sup>[Online]. Available: <https://www.videolan.org/>, <http://www.libtiff.org/>





Data	# Vul	# Non-vul	# Total Samples	Compile environment
CWE119	7916	7474	15390	Windows
LibTIFF	26	776	802	Windows
VLC	36	3895	3931	Windows

Our experiments show that deep learning can learn high-representations based on binary instructions. To achieve this, we first make use of IDA Prop to display the instructions of the binaries in hexadecimal form. IDA Prop can display various information given object files. Fig. 2 displays an example of *abstract-jb-c-jb-force-resynch-adaptive.o* file from the VLC project, which was compiled with MINGW64 on Windows. We can see there are two functions (i.e., function1 named *jb-force-resynch-adaptive*, function2 named: *main*) in total after processing by IDA Pro. For the LibTIFF and VLC projects, we compiled the whole projects. We apply IDA Pro to obtain binary instructions. Afterwards, vulnerable functions will be identified, while cross-function vulnerabilities are not included. For the CWE119 dataset, we first identify the vulnerable functions carefully and exclude the files that are not compilable. After this process, we can obtain two kinds of object files which are vulnerable and not vulnerable. Then, we apply IDA Pro to obtain binary instructions. Finally, binary functions can be obtained from the binary instructions. It is worth noting that some redundant features such as *ret*, *leave*, *pop*, will be removed in the data processing step because these features have less connection with vulnerabilities.

where  $\mathbf{h}_i \in \mathbb{R}^{64}$  and  $\mathbf{c}_i \in \mathbb{R}^{64}$  are the hidden states and cells of the LSTM at position  $i$ , respectively. The BiLSTM reads a token sequence with an LSTM in the forward direction and an LSTM in the backward direction. We concatenate the hidden states generated in the two directions  $\overrightarrow{\mathbf{h}}_i$  and  $\overleftarrow{\mathbf{h}}_i$  as the output  $\hat{\mathbf{h}}_i$  at position  $i$ .

Then, the feature representation  $fr$  learned from binary (i.e.,  $x_1, x_2, \dots, x_n$ ) by using the attention mechanism is

$$fr = H\alpha^T \quad (3)$$

where  $\alpha = \text{softmax}(\omega^n M)$  and  $M = \tan h(H)$ ,  $H \in R^{128 \times n}$ ,  $\omega$  is a trained parameter vector.

The high-level feature representation learned for  $x_1, x_2, \dots, x_n$  from attention mechanism is

$$h^* = \tan h(fr). \quad (4)$$

Finally, we employ a softmax classifier  $C$  to predict whether a binary level function ( $bf$ ) is vulnerable or not by taking  $h^*$  as input

$$\hat{y} = C(h^*|bf) = \text{softmax} \left( W^{(bf)} h^* + b^{(bf)} \right). \quad (5)$$

In other words

$$\hat{y} = C(h^*|bf) = \underset{y}{\text{argmax}} \hat{p}(y|bf) \quad (6)$$

where  $y$  is the probability produced for each class by  $C$ . It is worth noting that the threshold on the probability to flag the function as potentially vulnerable is 0.5. For example, if the probability of a function is larger than 0.5, it will be treated as a vulnerable function, otherwise, it is a nonvulnerable function.

### C. Model Explanation

In this section, motivated by previous work [38] we provide an explanation of the model behavior in order to show our proposed approach can capture the variances of the binary code and establish ‘trust.’

First of all, we hope that our proposed approach can capture the well-known heuristics in binary-level vulnerability detection. A reliable prediction model should be able to learn some useful knowledge from these differences. This motivates us to represent the attention words of the input test cases and remove the attention words in order to check the importance of these words. In order to provide an interpretation of our proposed approach, we start from a function level software (right-hand side of Fig. 3). This function will be compiled into a binary; then we employ IDA Pro to obtain binary instructions, and binary functions are extracted based on the obtained binary instructions; finally, the binary functions are fed to the proposed approach, where attention words have been printed out for model behavior understanding.

By analyzing the classification outputs and the attention words, we observe that our proposed approach can effectively capture the important information and can learn the difference. Fig. 3 shows seven important features to help recognize vulnerable functions. In particular, the features of 83 7 d 10 00, 78 32, and 8b 45 10 make great contribution to the final prediction. These features are highlighted in red to yellow according to their different importance. We compared the attention words with source code. The analysis shows that the binary code 83 7 d 10 00, 78 32, and 8b 45 10 are related to the source code of  $data \geq (10)$ , which is very important to discover vulnerabilities. After comparing the attention words with source code, we can see that the binary code 83 7 d 10 00, 78 32, and 8b 45 10 are

related to  $data \geq (10)$ . All these information are very important to discover the vulnerability.

All in all, the attention words selected from the feature space by our proposed approach illustrate that our proposed model can capture the important information from the binary code. This also confirms that our proposed approach can be employed to automatically detect possible vulnerabilities.

### D. How to Use the Proposed Approach

Our proposed approach is easy to use. First, software purchaser can feed the binary software into the proposed system. The output will be the locations of potential vulnerabilities at the function level. Then, the results can be passed to professional software developers to double check, and patch the vulnerabilities where necessary. Professional software developers can make use of the proposed system to analyze their developed binary software, and check the results. They can convert the binary code to disassemble by using a tool such as IDA Pro, and then check the locations of vulnerabilities in the source code, and fix them finally. Professional software developers can, therefore, make use of the proposed system to analyze their developed binary software, and check the results in order to make sure whether there exists vulnerability or not. Once there are vulnerabilities, they can convert the binary code to disassemble by using tools such as IDA Pro or objdump, and then check the location of vulnerability in the source code, and fix them finally.

## V. EXPERIMENTAL SETUP

In the experiments, we aim to answer the following three research questions:

- 1) *Research Question 1 (RQ1)*: Can source code-based software vulnerability detection methods be applied to binary code to effectively detect vulnerabilities compared with our proposed approach? This research question is important because one may argue that previously developed source code-based approaches can solve the binary level vulnerability detection problem. In order to answer this question, we apply two recently developed binary code-based approaches over two datasets.
- 2) *Research Question 2 (RQ2)*: How effective is the proposed approach when compared with other machine learning-based vulnerability detection for IoT binaries? In order to show that the proposed approach outperforms the other machine learning-based approaches, we choose CWE119 dataset to conduct the comparative experiments.
- 3) *Research Question 3 (RQ3)*: Is the proposed approach practical with significant impact on real security problems? This research question is very important because any security-related system should have impacts on real-world security problems. To answer this question, we explore the improvements of the proposed approach on real software: VLC and LibTIFF. We report the top-k precision for this research question so that it will help researchers to understand the proportion of vulnerabilities detected based on the top-k retrieved functions.

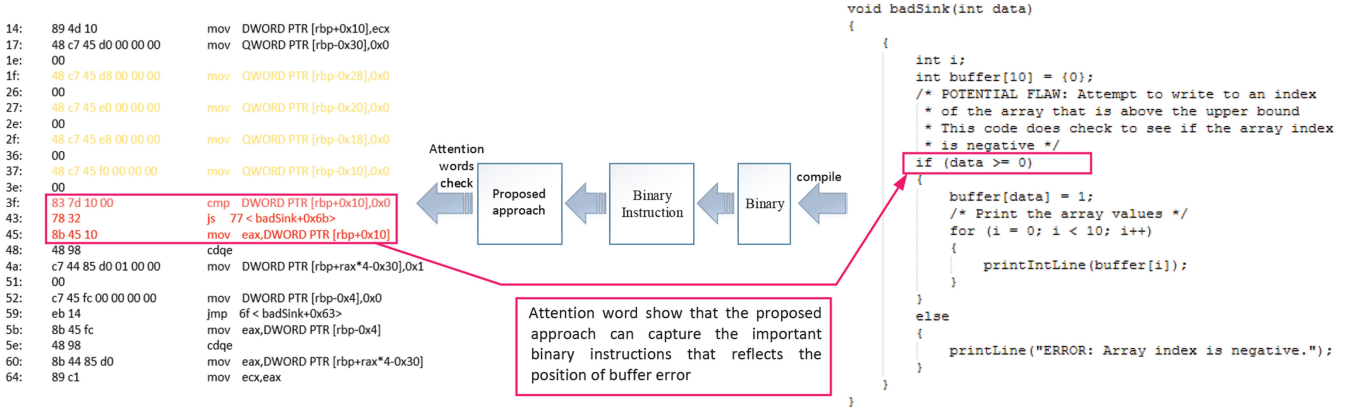


Fig. 3. Model explanation. The importance of attention words decrease from red to yellow.

TABLE II  
EXPERIMENTAL RESULTS REGARDING COMPARISON WITH  
SOURCE CODE-BASED APPROACHES

Technique	Accuracy	Precision	TPR	FPR	F1-measure
VulDeePecker	69.71%	73.73%	64.41%	21.96%	67.01%
BiLSTM + RF	71.35%	74.66%	64.80%	22.07%	69.38%
Proposed-app	83.93%	79.7%	90.41%	22.37%	<b>84.72%</b>

The bold value shows the the best performance.

### A. Baselines

We recognise that there are works addressing software vulnerability detection at source code level: BiLSTM+RF (Random Forest) [23] and VulDeePecker [24]. BiLSTM+RF leverages a customized LSTM network for learning function-level representations based on function ASTs. Then, the learned representations are treated as features for training a random forest classifier. VulDeePecker system directly applies a BiLSTM model to construct high-level features from code gadgets and feeds them to a feed-forward network. One may consider applying these two models directly on our selected binary features. Therefore, we apply BiLSTM+RF and VulDeePecker to our collected data. In this article, to save space, our proposed approach is short for ‘Proposed-app’ in the tables and figures.

As far as we know, there is not much work using deep learning for vulnerability detection based on binary code. We identified the closest works to ours as: Text-CNN-based approach [36], VDiscover [25], and DiscovRE [26]. However, considering the fact that DiscovRE is developed for cross-architecture bug discovery, we only compare our article with the Text-CNN-based approach [36] and VDiscover [25]. Note that, we only make use of static features with the VDiscover approach given that there is not much improvement from combining both static and dynamic features.

### B. Evaluation Metrics

We consider some widely used evaluation metrics in our experiments: Accuracy, Precision, TPR (i.e., Recall), F1-measure (F1), and false positive rate (FPR). Since false negative rate

TABLE III  
EXPERIMENTAL RESULTS REGARDING COMPARISON WITH BINARY  
CODE-BASED APPROACHES ON CWE119

Technique	Accuracy	Precision	TPR	FPR	F1-measure
Ins2vec -TCNN	66.69%	70.4%	58.4%	24.9%	63.8%
VDiscover	76.32%	74.4%	72.8%	24.6%	73.6%
Proposed-app	83.93%	79.7%	90.41%	22.37%	<b>84.72%</b>

The bold value shows the the best performance.

(FNR) = 1 - Recall, we only report TPR (recall). Compared to other techniques, we expect our proposed approach can achieve higher TPR values, lower FPR values, and higher FM values as well because FM is a balanced performance measure to show the overall performance of the classifier.

In order to show the effectiveness of our proposed approach, top-k precision (denoted as  $P@K$ ) is also considered in this article. This metric is popularly used in the area of information retrieval systems to show the requested relevant documents in terms of the top-k retrieved documents [23]. In this article, we report top-k to show the proportion of vulnerabilities in the top-k retrieved functions.

### C. Implementation Details

We implemented the deep feature model in Keras (2.0.8) with TensorFlow (1.3.0) as the backend. The token embedding was pretrained using the Gensim package (3.0.1) with the default settings. We selected 70% of the data as training data, 15% as evaluation data, and the remaining 15% as the test cases. Considering there is a critical class imbalance problem [39] with the LibTIFF and VLC datasets, by following a previous work [25], we employed a well-tested method called random oversampling (ROS, with an oversampling ratio as 200%) to alleviate the problem.

### D. Results and Discussion

We structure our evaluation by stepping through each of our three research questions (RQ1-RQ3).

**RQ1:** In order to show that the proposed approach outperforms the other sourced code-based machine learning ap-

**TABLE IV**  
EXPERIMENTAL RESULTS REGARDING REAL-WORLD SECURITY PROBLEM

Data	Technique	Accuracy	Precision	TPR	FPR	F1-measure
LibTIFF	Ins2vec-TCNN	83.33%	38.46%	38.46%	9.64%	38.46%
	VDiscover	82.52%	55.56%	50.00%	9.64%	52.63%
	Proposed-app	97.33%	57.14%	72.73%	1.84%	<b>64.00%</b>
VLC	Ins2vec-TCNN	99.39%	66.67%	42.86%	0.10%	40.00%
	VDiscover	99.29%	50.00%	14.29%	0.10%	22.22%
	Proposed-app	99.29%	50.00%	85.71%	0.61%	<b>63.16%</b>

The bold value shows the the best performance.

proaches, we choose CWE119 to conduct the comparative experiments.

**Table II** presents the experimental results of the proposed approach, VulDeePecker and BiLSTM+RF on CWE119. From **Table II**, we can see that the proposed approach outperforms the other two deep models with a wide margin based on binary instruction features. The proposed approach has a much higher F1-score (i.e., 84.72% versus 67.01% for VulDeePecker and 69.38% for BiLSTM+RF) on the CWE119 dataset because it has much higher Precision (i.e., 79.7% versus 73.73% for VulDeePecker and 74.66% for BiLSTM+RF) and TPR (i.e., much lower FNR), while noting that accuracy is 83.93% (versus 69.71% for VulDeePecker and 71.35% for BiLSTM+RF).

It is worth noting that the comparison is **indicative** since VulDeePecker and BiLSTM+RF are developed for source code level vulnerability detection. In this article, we apply these two models on the binary code to show that source code-based software vulnerability detection approaches cannot solve the binary level vulnerability detection problem. We think there are two reasons behind this. 1) VulDeePecker and BiLSTM+RF were developed based on different inputs. For example, VulDeePecker uses code gadget as input while BiLSTM+RF uses ASTs (abstract syntax trees). However, in our article, we are not able to extract these features based on binary instructions. The use of binary instruction as input may affect the performance of these two models. 2) The proposed approach employs an attention model rather than BiLSTM – we believe that the attention model can capture the important part of the binary code and pay more attention to the specific binary features. As a result, the proposed approach can discover more vulnerable functions comparatively.

Therefore, we confirm that there is much room to improve when applying source code-based vulnerability detection approaches. Experimental results show that our proposed approach achieves outstanding performance than source code based-software vulnerability detection approaches.

**RQ2:** In order to show that the proposed approach outperforms the other machine learning-based approaches for binary level vulnerability detection, we choose CWE119 to conduct the comparative experiments.

**Table III** presents the experimental results of the proposed approach, Ins2vec-TCNN (Instruction2vec + TextCNN) and VDiscover when used with CWE119, LibTIFF, and VLC. From **Table III**, we can see that our proposed approach substantially improves over the baselines. The proposed method has a much higher F1-score (i.e., 84.72% versus 63.8% for Ins2vec-TCNN and 73.6% for VDiscover) on the CWE119 dataset because it

has much higher Precision and TPR (i.e., much lower FNR), while noting that accuracy is 83.93% (versus 66.69% for Ins2vec-TCNN and 76.32% for VDiscover).

We hypothesise that the reasons that the proposed approach outperforms the other two approaches are three-fold. 1) Ins2vec-TCNN uses opcodes such as mov as features, which we think may not be enough at the function level. For example, in one of the CWE119 samples, the main function only contains two mov opcodes (i.e., mov ebp, esp and mov eax, 0x0). Text-CNN might not be able to learn high-quality representations on these features. 2) We suspect deep learning in general performs better than word2vec [23], which we believe is the reason why the proposed approach outperforms VDiscover. 3) The vulnerabilities are related to certain parts of the code rather than the whole function, and in the decode process of the attention mechanism, different parts can be paid different attention. This leads us to speculate that the proposed approach can pay more attention to the vulnerability causing instructions (more information can be seen in Section IV-C: model explanation).

**RQ3:** The effectiveness of the proposed approach against real security problems? To answer this question, we use real-world software: LibTIFF and VLC. We focus on exploring the proposed approach in this section rather than comparing with other techniques in order to show that the proposed approach can impact on real-world security problems.

**Table IV** presents the experimental results of the proposed approach, VDiscover, and Ins2vec-TCNN-based on LibTIFF and VLC. For LibTIFF, the test set contains 337 functions among which there are 11 vulnerable ones; for VLC, the test set contains 983 functions including 7 vulnerable ones. For LibTIFF and VLC, we report the top-k precision as well in order to show the effectiveness of our proposed algorithm.

From **Table IV** we can see that the proposed approach outperforms the other algorithms with a large margin. Take VLC project as an example, the proposed approach achieves a higher F1-measure with 63.16% (compared with Ins2vec-TCNN with 40% and VDiscover with only 22.22%) because of the higher TPR with 85.71% (i.e., versus Ins2vec-TCNN with 42.86% and VDiscover with only 14.29%) Note that, the accuracy of three techniques are similar, this is because the only about 7 vulnerable functions in the test cases, even though it has false negatives, there is less impact on the overall accuracy. In addition, one can observe that the proposed approach results in lowest TPR on VLC project which means it cause many high false negatives. While Ins2vec-TCNN results in higher precision at the cost of TPR, which means it also has many false negatives. Therefore, the proposed approach has higher TPR



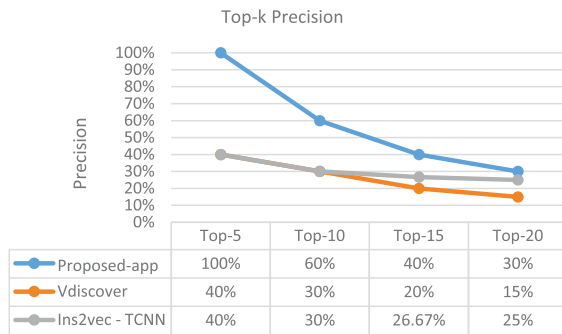


Fig. 4. Experimental results of top-k precision on VLC.

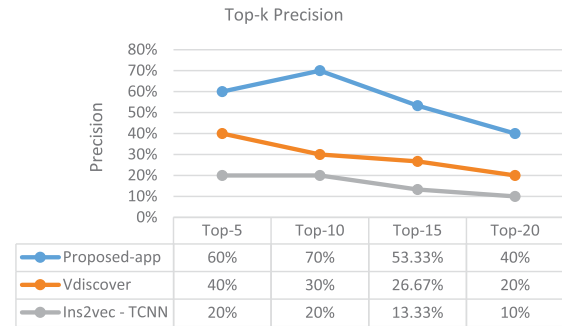


Fig. 5. Experimental results of top-k precision on LibTIFF.

and comparative Precision compared with the other algorithms on both LibTIFF and VLC projects.

From Table IV we can observe that all three models show worse performance on LibTIFF and VLC compared with CWE119. We suspect the reason is that the ratio of nonvulnerable functions and vulnerable functions varies. Table I shows that the ratio of the CWE119 dataset is nearly 1:1, while the ratios for the LibTIFF and VLC datasets are about 30:1 and 108:1, respectively. This may cause the classifier to be biased toward the nonvulnerable function class and cause false negatives. Even though the Precision of the proposed approach is the best among all the techniques, we acknowledge that the best Precision obtained by our proposed method is still less than 65%. We plan to study this problem and employ some effective techniques to mitigate this problem in the future.

From Fig. 4 we can see that top-5 precision in terms of VLC is 100%, which means the top five functions identified by the proposed approach are all vulnerabilities. When searching for the 15 and 20 most probable vulnerable functions, our proposed system can identify six vulnerable functions. Therefore, the proposed approach can successfully detect six out of seven vulnerable functions by only checking the top ten functions. This emphasized the practicality of our proposed system.

Fig. 5 indicates that the top-k precision of the proposed approach on LibTIFF ranges from 70% to 40%, which means when retrieving the ten most probable vulnerable functions, seven out of 11 vulnerable functions can be detected. Moreover, within the top 15 functions, eight out of 11 vulnerable functions can be detected. This means that one can identify 73% (8/11) vulnerable functions by examining the top 15 or 20 functions.

It is worth noting that the proposed approach still causes false negatives because it missed one vulnerability on VLC project and three vulnerabilities on LibTIFF project. We hypothesise that the reasons that the proposed approach causes false positives are twofold. 1) The size of the dataset is not big enough compared with CWE119. Since the proposed approach is a data-driven system, we believe the prediction performance can be improved with the increasing number of training data samples. 2) There is severe class imbalance problem in LibTIFF and VLC projects, which may cause the prediction model biased toward the nonvulnerable class. We plan to improve this problem by developing new data-redistribution algorithms in the future.

## VI. CONCLUSION

In this article, we proposed a deep learning-based approach for IoT related software (e.g., autonomous vehicles and military systems) vulnerability detection based on binary code rather than the source code. To the best of our knowledge, we are among the first of developing a system that applies deep learning to detect vulnerabilities in binary machine code.

To achieve this, we used IDA Pro to prepare the binary instructions and attention model for high-level feature representation learning. Afterwards, the prediction model is trained based on the high-level representations. Experiments were conducted based on real-world IoT related projects. Experimental results showed that the proposed approach achieves superior performance compared with the baselines, which emphasized that our proposed approach is practical and applicable to real-world IoT security problems.

In future, we will study the case of the function inlining scenario. Function inlining could change the structure of the binary code, which cannot be addressed by our proposed approach in this article. Moreover, the Precision on each dataset reported in this article is less than 80%, so there is much space to improve. We will explore other artificial intelligence-based techniques in the topic of vulnerability detection in binary code.

## REFERENCES

- [1] S. Wen, M. S. Haghighi, C. Chen, Y. Xiang, W. Zhou, and W. Jia, "A sword with two edges: Propagation studies on both positive and negative information in online social networks," *IEEE Trans. Comput.*, vol. 64, no. 3, pp. 640–653, Mar. 2015.
- [2] N. Sun, J. Zhang, P. Rimba, S. Gao, Y. Xiang, and L. Y. Zhang, "Data-driven cybersecurity incident prediction: A survey," *IEEE Commun. Surv. Tut.*, vol. 21, no. 2, pp. 1744–1772, 2Q 2019.
- [3] S. M. Ghaffarian and H. R. Shahriari, "Software vulnerability analysis and discovery using machine-learning and data-mining techniques: A survey," *ACM Comput. Surv.*, vol. 50, no. 4, 2017, Art. no. 56.
- [4] J. Jiang, S. Wen, S. Yu, Y. Xiang, and W. Zhou, "Identifying propagation sources in networks: State-of-the-art and comparative studies," *IEEE Commun. Surv. Tut.*, vol. 19, no. 1, pp. 465–481, 1Q 2017.
- [5] M. Amoozadeh *et al.*, "Security vulnerabilities of connected vehicle streams and their impact on cooperative driving," *IEEE Commun. Mag.*, vol. 53, no. 6, pp. 126–132, Jun. 2015.
- [6] T. Wu, S. Wen, Y. Xiang, and W. Zhou, "Twitter spam detection: Survey of new approaches and comparative study," *Comput. Secur.*, vol. 76, pp. 265–284, 2018.
- [7] X. Ban, S. Liu, C. Chen, and C. Chua, "A performance evaluation of deep-learned features for software vulnerability detection," *Concurrency Comput.: Pract. Experience*, 2019, Art. no. e5103.



- [8] J. Lee, T. Avgerinos, and D. Brumley, "TIE: Principled reverse engineering of types in binary programs," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, San Diego, California, USA, Feb. 2011.
- [9] S. Parkinson, P. Ward, K. Wilson, and J. Miller, "Cyber threats facing autonomous and connected vehicles: Future challenges," *IEEE Trans. Intell. Transp. Syst.*, vol. 18, no. 11, pp. 2898–2915, Nov. 2017.
- [10] G. Balakrishnan and T. Reps, "WYSINWYX: What you see is not what you execute," *ACM Trans. Program. Lang. Syst.*, vol. 32, no. 6, 2010, Art. no. 23.
- [11] V. D'Silva, M. Payer, and D. Song, "The correctness-security gap in compiler optimization," in *Proc. Secur. Privacy Workshops*, 2015, pp. 73–87.
- [12] B. S. Gulavani, T. A. Henzinger, Y. Kannan, A. V. Nori, and S. K. Rajamani, "Synergy: A new algorithm for property checking," in *Proc. 14th ACM Int. Symp. Found. Softw. Eng.*, 2006, pp. 117–127.
- [13] T. Wang, T. Wei, Z. Lin, and W. Zou, "IntScope: Automatically detecting integer overflow vulnerability in X86 binary using symbolic execution," in *Proc. 16th Annu. Netw. Distrib. Syst. Secur. Symp.*, 2009, pp. 497–512.
- [14] M. Zhang, R. Qiao, N. Hasabnis, and R. Sekar, "A platform for secure static binary instrumentation," *ACM SIGPLAN Notices*, vol. 49, no. 7, pp. 129–140, 2014.
- [15] J. Pewny, B. Garmany, R. Gawlik, C. Rossow, and T. Holz, "Cross-architecture bug search in binary executables," in *Proc. IEEE Symp. Secur. Privacy*, May 2015, pp. 709–724.
- [16] P. Nunes, I. Medeiros, J. C. Fonseca, N. Neves, M. Correia, and M. Vieira, "Benchmarking static analysis tools for web security," *IEEE Trans. Rel.*, vol. 67, no. 3, pp. 1159–1175, Sep. 2018.
- [17] D. Miyani, Z. Huang, and D. Lie, "Binpro: A tool for binary source code provenance," 2017, *arXiv:1711.00830*.
- [18] A. Henderson, L. K. Yan, X. Hu, A. Prakash, H. Yin, and S. McCamant, "DECAF: A platform-neutral whole-system dynamic binary analysis platform," *IEEE Trans. Softw. Eng.*, vol. 43, no. 2, pp. 164–184, Feb. 2017.
- [19] M. Luo, O. Starov, N. Honarmand, and N. Nikiforakis, "Hindsight: Understanding the evolution of ui vulnerabilities in mobile browsers," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2017, pp. 149–162.
- [20] Y. Li, W. Xu, Y. Tang, X. Mi, and B. Wang, "Semhunt: Identifying vulnerability type with double validation in binary code," in *Proc. 29th Int. Conf. Softw. Eng. Knowl. Eng.*, Wyndham Pittsburgh University Center, Pittsburgh, PA, USA, Jul. 2017, pp. 491–494.
- [21] W. Li *et al.*, "Memory access integrity: Detecting fine-grained memory access errors in binary code," *Cybersecurity*, vol. 2, no. 1, 2019, Art. no. 17.
- [22] N. Stephens *et al.*, "Driller: Augmenting fuzzing through selective symbolic execution," in *Proc. Annu. Netw. Distrib. Syst. Secur. Symp.*, 2016, vol. 16, pp. 1–16.
- [23] G. Lin *et al.*, "Cross-project transfer representation learning for vulnerable function discovery," *IEEE Trans. Ind. Informat.*, vol. 14, no. 7, pp. 3289–3297, Jul. 2018.
- [24] Z. Li *et al.*, "VulDeePecker: A deep learning-based system for vulnerability detection," in *Proc. 25th Annu. Netw. Distrib. Syst. Secur. Symp.*, San Diego, California, USA, Feb. 2018.
- [25] G. Grieco, G. L. Grinblat, L. Uzal, S. Rawat, J. Feist, and L. Mounier, "Toward large-scale vulnerability discovery using machine learning," in *Proc. 6th ACM Conf. Data Appl. Secur. Privacy*, 2016, pp. 85–96.
- [26] S. Eschweiler, K. Yakdan, and E. Gerhards-Padilla, "discovRE: Efficient cross-architecture identification of bugs in binary code," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2016.
- [27] L. Liu, O. De Vel, Q.-L. Han, J. Zhang, and Y. Xiang, "Detecting and preventing cyber insider threats: A survey," *IEEE Commun. Surv. Tut.*, vol. 20, no. 2, pp. 1397–1417, 2Q 2018.
- [28] M.-T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," 2015, *arXiv:1508.04025*.
- [29] S. A. Kumar, T. Vealey, and H. Srivastava, "Security in Internet of Things: Challenges, solutions and future directions," in *Proc. 49th Hawaii Int. Conf. Syst. Sci.*, 2016, pp. 5772–5781.
- [30] E. Bertino and N. Islam, "Botnets and Internet of Things security," *Computer*, vol. 50, no. 2, pp. 76–79, Feb. 2017.
- [31] I. Andrea, C. Chrysostomou, and G. Hadjichristofi, "Internet of Things: Security vulnerabilities and challenges," in *Proc. IEEE Symp. Comput. Commun.*, Jul. 2015, pp. 180–187.
- [32] M. Abomhara and G. M. K  ien, "Cyber security and the internet of things: Vulnerabilities, threats, intruders and attacks," *J. Cyber Secur. Mobility*, vol. 4, no. 1, pp. 65–88, 2015.
- [33] L. Xiao, X. Wan, X. Lu, Y. Zhang, and D. Wu, "IoT security techniques based on machine learning: How do IoT devices use AI to enhance security?" *IEEE Signal Process. Mag.*, vol. 35, no. 5, pp. 41–49, Sep. 2018.
- [34] F. Al-Turjman and C. Altrjman, "IoT smart homes and security issues: An overview," in *Security IoT-Enabled Spaces*. Boca Raton, FL, USA: CRC Press, 2019, pp. 111–137.
- [35] A. D. Kumar, K. N. R. Chebrolu, V. R. , and S. K. P. , "A brief survey on autonomous vehicle possible attacks, exploits and vulnerabilities," 2018, *arXiv:1810.04144*.
- [36] Y. J. Lee, S.-H. Choi, C. Kim, S.-H. Lim, and K.-W. Park, "Learning binary code with deep learning to detect software weakness," in *Proc. KSII 9th Int. Conf. Internet Symp.*, 2017, pp. 245–249.
- [37] A. Vaswani *et al.*, "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 5998–6008.
- [38] W. Guo, D. Mu, J. Xu, P. Su, G. Wang, and X. Xing, "Lemna: Explaining deep learning based security applications," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2018, pp. 364–379.
- [39] S. Liu, J. Zhang, Y. Xiang, and W. Zhou, "Fuzzy-based information decomposition for incomplete and imbalanced data learning," *IEEE Trans. Fuzzy Syst.*, vol. 25, no. 6, pp. 1476–1490, Dec. 2017.



**Shigang Liu** received the Ph.D. in computer science from Deakin University, Victoria, Australia, in 2017.

Currently, he is a second-year Research Fellow with the School of Software and Electrical Engineering, Swinburne University of Technology, Hawthorn, Australia. He has published more than 20 research papers in many international journals and conferences. His research mainly focuses on security, applied machine learning, and fuzzy information processing.

Dr. Liu is a member of the Editorial Board of the Journal of Mathematics and Informatics. He has been the Program Committee member, Publication Chair and Local Arrangement Chair for several conferences including the 2nd International Conference of Machine Learning for Cyber Security (ML4CS2019), the 9th International Symposium on Cyberspace Safety and Security (CSS2017), the 6th Asia-Pacific Conference on Computer Science and Data Engineering (CSDE2019), and so on. He is currently leading a research group on applying deep learning to detect software vulnerabilities. This research topic won the first place of the World Change Maker Prize in Swinburne Research Conference in 2019.

**Mahdi Dibaei**, photograph and biography not available at the time of publication.

**Yonghang Tai**, photograph and biography not available at the time of publication.



**Chao Chen** received the Ph.D. degree in computer science from Deakin University, Victoria, Australia, in 2017.

From 2016 to 2018, he worked as a Data Scientist with Telstra to create customer value from huge and heterogeneous data sources using advanced analytics and big data techniques. He is currently a Research Scientist with the School of Software and Electrical Engineering, Swinburne University of Technology, Hawthorn, Australia, and also a core member of Swinburne Cybersecurity lab. He is currently conducting research on applying advanced analytics to solve emerging cyber security issues, such as insider threat detection and information leakage. He has published more than 20 research papers in refereed international journals and conferences. His major research interests include cyber data analytics, network traffic classification, and Twitter spam detection.

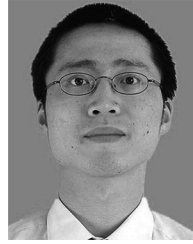


**Jun Zhang** (M'12) received the Ph.D. degree in computer science from the University of Wollongong, Wollongong, Australia, in 2011.

He is the Co-Founder and Deputy Director of the Cybersecurity Lab, Swinburne University of Technology, Hawthorn, Australia. He is the Chief Investigator of several projects in cybersecurity, funded by Australian Research Council (ARC). He has published more than 100 research papers in many international journals and conferences. His research interests include

cybersecurity and applied machine learning.

Dr. Zhang's research has been widely cited in the area of cybersecurity. He has been internationally recognised as an active Researcher in cybersecurity, evidenced by his chairing of ten international conferences, and presenting of invited keynote addresses in four conferences and an invited lecture in IEEE SMC Victorian Chapter. Two of his papers were selected as the featured articles in the July/August 2014 issue of *IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING* and the March/April 2016 issue of *IEEE IT Professional*.



**Yang Xiang** received the Ph.D. degree in computer science from Deakin University, Geelong, VIC, Australia.

He is currently the Dean of Digital Research and Innovation Capability Platform, Swinburne University of Technology, Hawthorn, VIC, Australia. He is the Chief Investigator of several projects in network and system security, funded by the Australian Research Council (ARC). He is leading his team developing active defense systems against large-scale distributed network

attacks. He has authored or coauthored more than 200 research papers in many international journals and conferences. His research interests include network and system security, distributed systems, and networking.

Dr. Yang was the Program/General Chair and PC member for more than 60 international conferences in distributed systems, networking, and security. He is the Coordinator, Asia, for IEEE Computer Society Technical Committee on Distributed Processing.