IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received November 19, 2019, accepted December 9, 2019, date of publication December 18, 2019,
date of current version December 27, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2960522

Marking Key Segment of Program Input via Attention Mechanism

XING ZHANG, CHAO FENG, RUNHAO LI, JING LEI, AND CHAOJING TANG
School of Electronic Science and Technology, National University of Defense Technology, Changsha 410073, China
Corresponding author: Chao Feng (chaofeng@nudt.edu.cn)

ABSTRACT Key segment of a program input is the specific part of the input that has significant affect on the execution of target function. Marking key segment plays an important role in software security analysis. Traditional dynamic analysis methods can not mark the key segments correctly because of control flow dependency problem. The root cause of such problem is that implicit flow analysis method cannot cover all the behavior of the code fragment in a branch, especially when the code snippet contains unexpected jump behavior. The neural network can learn to fit the behavior of the program with proper training data. In this paper, we introduce the attention based neural network to mark the key segments of program input accurately and efficiently. We propose an attention based two-parts network structure and map program inputs into the target code execution by such network. Then we propose a two-step training method to train our network to calculate the importance of each input component on the execution of target function. Finally, we mark the key segments by statistical analysis method. We implement such method and develop a key segment marking tool *AttentionMark*. Experiments on four real-world software show that *AttentionMark* outperforms *NeuralTaint* and traditional dynamic analysis tool in key segment marking.

INDEX TERMS Taint analysis, symbolic execution, software vulnerability, neural network, key segment marking, deep learning.

I. INTRODUCTION

Analyzing the relationship between the execution of program function and program input plays an important role in software analysis research. Many applications process complex, highly structured inputs. Small changes in some segments of the input can induce correspondingly large changes of target function execution times of the program. And in this paper, we call the specific components of a program input that have a significant effect on the execution of target function as **key segment**. Marking key segment of a user input is important in a number of software security research areas, like fuzzing technique, malware detection and program understanding.

Taint analysis and symbolic execution are two popular analysis method to mark the key segments of the program input [2]. These methods monitor code as it executes by preset rules and performs precise analysis with run-time information. Taint analysis method explores the relationship between user inputs and program execution information with taint propagation rules. And symbolic execution reasons about the behavior of a program on user input by building and solving logical formulas.

Marking key segment needs to deal with both information flow and control flow of the target program. Traditional analysis methods have made great progress in information flow analysis and applied widely in software analysis and security area. However, some programs with control flow dependency structure would cause the under taint problem, and both taint analysis and symbolic execution method may suffer from low accuracy while analyzing such program. Such problem is called **control flow dependency problem** [1]. In order to analyze the control dependency flow, implicit taint analysis method is proposed by many researchers [2]. Recent research applies the static analysis method to generate Control Flow Graph(CFG) to add extra taint to the variables belong to the tainted branch [3]–[5]. Despite that under taint problem is resolved to a certain extent, the extra taint may cause the over taint problem which also reduce the analysis accuracy. Although a lot of effort is being spent on improving under taint and over taint weakness, the efficient and effective method has yet to be developed.

There are two critical reasons that causing low accuracy of implicit analysis method. 1 is that the rules preset by human

The associate editor coordinating the review of this manuscript and approving it for publication was Fan Zhang.

experience can not distinguish all code fragments of conditional jumps accurately; 2 is that the instruction/basic block level analysis method can not describe the macro behavior of a function that has predominant influence on the control flow,e.g. function like *_exit*(0). Section III gives a motivate example to illustrate the problem.

Neural network has strong versatility, and theoretically it can fit all functions [7], [9]. With enough inputs and labels, neural network can learn the relationship between the input and output by appropriate training. The relationship is represented as the trained neural network itself, which is not understandable to human. Recently, neural network explainable technique gives insight the internal operation and behavior of neural networks [10]. *NeuralTaint* [35] applied gradient method to mark key segments. *NeuralTaint* maps program inputs into execution times of target function by neural network and then extract the gradient of target input from fitted network, finally mark the key segments with gradient information.

NeuralTaint can overcome the control flow dependency problem and has higher analysis accurate rate than traditional methods. However, *NeuralTaint* suffers over-fitting problem and gradient based explainable method has saturation problem [50]. Therefore, *NeuralTaint* has low analysis accurate and stability problem. Since program inputs are sequence data and attention mechanism [11] is an explainable method for neural networks that processing sequence data. Attention mechanism based network can directly assign importance score to the inputs for a given output. Consequently, we propose a key segment marking method based on attention mechanism network and develop a key segment marking tool *AttentionMark*. Instead of improving the fitting accuracy of network, we adopt attention based network to locate the specific part of program input which can affect the execution of target function.

In this paper, we generate large amount of program inputs with mutate algorithm. Next we propose a two-parts attention based neural network and a two-step training method to train the network for multiple rounds. The attention networks are extracted from the fitted networks and together with statistical analysis method we can achieve accurate key segment marking. By comparing with traditional dynamic analysis tools and *NeuralTaint* we verify the effectiveness of *AttentionMark*. Experiment results show that *AttentionMark* consistently outperforms traditional dynamic analysis tools and *NeuralTaint* in control flow taint analysis by a wide margin both in term of analysis accuracy rate.

In summary, the main contributions in this paper are listed as follows:

1) We utilize the attention mechanism in key segment marking and propose a two-parts attention based neural network structure. The two-parts network is divided into attention layer and regression layer which is different from traditional seq2seq [8] network, thus can fit in with the need of the program input and execution times output of the network.

2) We introduce two-step training method to train a well-fitted attention network. Traditional seq2seq network improves the fitting ability of network by attention mechanism and the explainable of the network is not always working. Instead of improve the fitting ability of network, we adopt the two-step training method to improve the explainable ability of our network.

3) We propose a statistical analysis method to achieve accurate key segment marking. Over fitting and fitting error are inevitable during training and thus the relevance calculated by attention layer may be inaccurate. We utilize multiple relevance score of rounds and epochs network to achieve accurate key segment marking.

4) We design and implement *AttentionMark* to mark key segments of program input. Experiments on several popular file formats processing software shows that *AttentionMark* can achieve accurate key segment marking and outperforms than traditional analysis tools and gradient based tool *NeuralTaint*.

The rest of the paper is organized as follows. In Section II, we describe previous research efforts related to this paper. We give a motivate example in section III and give our approach in Section IV, and Section V provides implementation details. We present the experiments in Section VI. Section VII concludes this paper.

II. RELATE WORK

Related research on program analysis and neural networks recently develops rapidly. Dawn Song [31] firstly realizes the recognition of function boundaries on machine-code level through neural networks. This method is the first to apply artificial intelligence techniques to the field of software vulnerability analysis. EKLAVYA [32] implements the identification of function parameters based on deep learning. NEUZZ [33] uses gradient-guide input generation method to increase the efficiency of fuzzing process based on surrogate neural network. NeuralTaint [35] marks the key segments by gradient-based explainable method of neural network. These works indicate that neural network can learn the relationship between network input and output without any program internal information.

A. DYNAMIC TAINT ANALYSIS

Currently, dynamic taint analysis tools are mainly divided into application-aware analysis tools and system-wide analysis tools. Application-aware taint methods analyze at a high speed, which is often used in vulnerability discovering. TaintScope [34], Dowser [36], BORG [37], VUzzer [38] and Angora [39] applied application-aware pure information flow analysis to improve fuzzing efficiency. System-wide taint analysis tools usually depend on the virtualized platform QEMU [40], which can effectively analyze the taint spreading process input into the system kernel. This method has higher accuracy, but suffers from lower speed and more complex implementation, thus not widely used. BitBlaze [41] and PANDA [42] are two main system-wide taint analysis tools to deal with explicit information flow.

B. DYNAMIC SYMBOLIC EXECUTION

Similar with taint analysis method, dynamic symbolic execution is mainly divided into application-aware methods and system-wide methods. System-wide symbolic execution tools are hardly used as a result of complicated structure and difficult development. *S2E* [30] is a system-wide platform for analyzing the properties and behavior of software systems based on QEMU, it uses selective symbolic execution and relaxed execution consistency models to scale to large software.

C. ATTENTION BASED NEURAL NETWORK

Attention mechanism has been widely adopted in neural based NLP networks. Attention mechanism can improve the fitting accuracy of the networks and give an explanation of the relationship between input vector and output vector. Attention layer equipped by the network can calculate the relevance scores of input tokens that denote the importance of component to the output vector. Bahdanau *et al.* [11] first introduces the attention mechanism into the networks of machine translation. And recently attention mechanism not only applied in NLP translation task. Wang *et al.* [48] formalized self-attention as a non-local operation to model the spatial-temporal dependencies in video sequences. Zhang *et al.* [46] applies the self-attention mechanism with GAN [47] to generate images that have fine details. However, no related work is applied in binary software analysis. We expand the usage of attention mechanism in control flow taint analysis.

Dynamic taint analysis and symbolic execution relies on heavy-weight program analysis techniques with nontrivial instrumentation overheads. On the contrary, *AttentionMark* marks the key segments of user input by attention mechanism without any expensive program analysis techniques. Thus *AttentionMark* provides a new way to overcome the difficulties in traditional dynamic binary analysis methods.

III. A MOTIVATE EXAMPLE

In order to give a vivid introduction of key segments marking, the weakness of traditional analysis method and the effectiveness of our method, we utilize a brief example to explain.

A. EXAMPLE CODE

Code 1 below is a brief code fragment example. The program gets first three components of program input as $x1, x2$ and $x3$. The program will exit if $x1$ is not 'a', y is a temporary variable and *target_function*() is executed only when $x3$ is 'c'. From the code, we can see that the content of $x1$ and $x3$ has significant influence on the execution of *target_function*(). In this paper, for a program input like "aXc", we call $x1$ and $x3$ as the **key segments** of "aXc" on the execution of *target_function*().

In order to analyze the relationship between user input and the execution of *target_function*() in Code 1, both taint analysis and symbolic execution set the flag registers in *line*12 as the sink point. Taint analysis checks the taint information of the flag registers to address the key segments and symbolic execution examines the logical formulas of the symbolic memory of the flag registers to locate the key segments.

Algorithm 1 Example Code

```
Input: x1 ← the 1st component of program input
       x2 ← the 2nd component of program input
       x3 ← the 3rd component of program input
 1: y = '0'
 2: if x1 != 'a' then
 3:     _exit(0)
 4: end if
 5: if x2 == 'b' then
 6:     y = '1'
 7:     other_function()
 8: end if
 9: if x3 == 'c' then
10:     y = 'd'
11: end if
12: if y == 'd' then
13:     target_function()
14: end if
```

Explicit analysis method deals with pure data information flow. Since code in *line*9 $\sim$ 13 contains control dependency structure and explicit analysis method occurs under taint problem [1]. The flag registers in *line*12 contains no taint or symbolic formula after analyzed by explicit analysis method. For traditional symbolic execution tools with SMT/Z3 solver. When the program reaches *target_function*(), symbolic execution method checks the symbolic memory of flag registers in *line*12, since temporary variable y is assigned with 'd' which is constant value in *line*10 and y contains no symbolic variable. That is, symbolic execution method collects nothing due to such control flow dependency structure. Such structure is first introduced in [1] which is a classic control flow dependency structure, that is, traditional symbolic execution tool can not deal with Code 1.

Implicit taint analysis method applies the static analysis method to generate Control Flow Graph(CFG) to add extra taint to the variables belong to the tainted branch [3]–[5]. In this case, *line*6 and *line*10 will add extra taint mark, thus the flag registers in line 12 contains taint from $x3$. Due to the relationship between *line*5 $\sim$ 8 and *line*9 $\sim$ 11 in CFG, some algorithms may add $x2$ as the taint source of the flag registers in *line*12 [2], [5], and others may not [3], [4]. That is, implicit analysis method may occur over taint problem [1].

From the code, $x1$ has great influence on the execution of *target_function*(), because *target_function*() will not be executed if $x1$ value is not 'a'. Such scenario often occurs when the program checks whether the header of user input is illegal. However, current implicit and explicit analysis method can not mark $x1$ as the key segment. The execution of one function may related to several branches through the whole control flow from the entry to the target function.

TABLE 1. Example Data-set.

data with label 1	data with label 0
"adc" — 1	"ddc" — 0
"aXc" — 1	"adW" — 0
"a\x00c" — 1	"\x08F," — 0
"abc" — 1	"X\xF8\xD9" — 0
"aWc" — 1	"aaa" — 0
"a\xFFc" — 1	".,A" — 0
...	...
"a,c" — 1	"\x01\xE1\xDD" — 0

Such feature makes it difficult for traditional dynamic analysis method to mark the key segments correctly and completely. The root cause of such problem is that neither static analysis nor extra propagation rules can not identify the specific behavior of every code fragment belongs to one branch correctly. Especially the code snippets which behaves like *exit* and *goto* instructions that can change the control flow. Therefore, under-taint and over-taint often occur in implicit analysis method [6].

B. SOLUTION

The overall idea of our work is that we map the program input into the execution times of target function with attention based neural network, then extract the attention network to calculate the relevance score of every input component and use statistical analysis method to get the key segments from the program input.

For the Code 1, suppose we get the input "adc", *target_function*() is the target function for analysis. We want to mark the key segments of "adc" that can affect the execution of *target_function*(). We utilize the attention based network to mark the key segments of "adc".

First, we construct the data-set. The network input is the program input, the label data is the execution times of *target_function*(). And the example data-set is shown in Table 1.

Second we construct a network with attention mechanism in Figure 1. Each input vector multiply a weight factor then calculate together and get a eigenvector. The eigenvector is then sent to neural network to calculate the execution times of *target_function*(). The network is a simplified version of a traditional attention based seq2seq back-end network. The input component contributes more to the output when such component's weight vector α is larger.

The weight factors α_1, α_2 and α_3 are initialized with average value 0.33. And the data-set in Table 1 is sent into the network to train the network. The first and third component of the input change from 'a' and 'c' to other bytes causes the output changes from 1 to 0, and the change of second component of the input has no affect on the output. So after training, the weight factors α_1, α_2 and α_3 may be $\alpha_1 = 0.48, \alpha_2 = 0.04$ and $\alpha_3 = 0.48$ when the fitted network calculates input "adc". Since α_1 and α_3 is obvious larger than α_2, we can mark the first and third component as the key segments of input "adc".

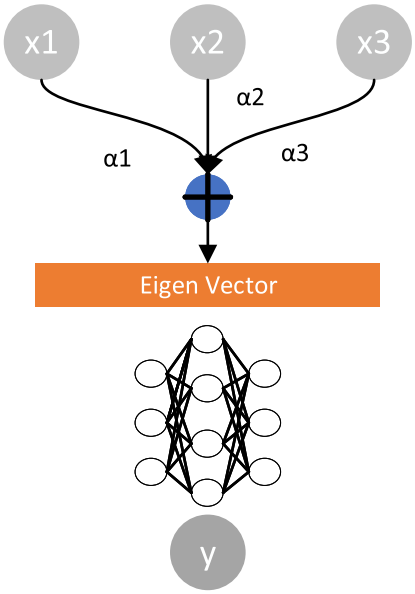


FIGURE 1. Example network.

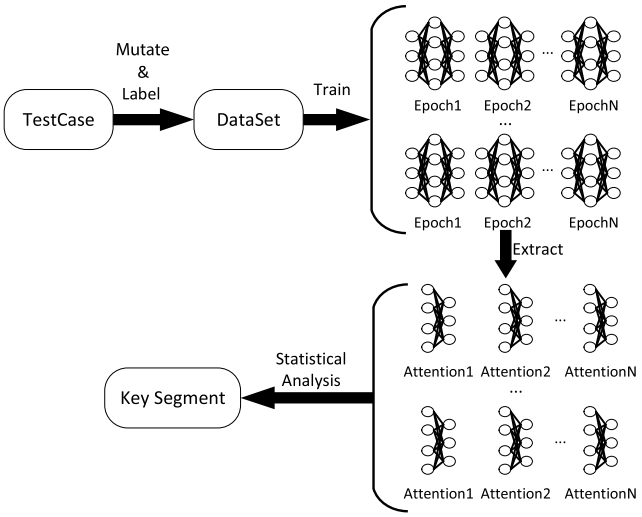


FIGURE 2. An overview of our approach.

Neural network directly analyzes the relationship between the execution of program function and program input which means that the network "simulates" the process of the execution of program through training. The attention mechanism can show the specific contribution of each input component to the output by numerical value, thus our approach can overcome control flow dependency problem and mark key segments accurately.

The rest of this paper introduces the detail solution of key segment marking method based on neural network with attention mechanism.

IV. OVERVIEW OF OUR APPROACH

Figure 2 presents a high level overview of our approach. Given a program input that can trigger the target function,

our approach can mark the key segment of such input.

A. DATA-SET CONSTRUCTION

Unlike traditional taint analysis, the key segment marking method based on neural network requires training before analyzing. Data-set construction is the first consideration before training neural network. There are three main problems in data-set construction, which are network input/output, data source and network input embedding.

The network input is program input and label is the execution times of target function. In order to collect the enough amount of training data, a mutate algorithm is proposed to generate program inputs. Neural network can only process the fixed length and numerical data. And in this paper, we utilize byte-level one-hot vector to embed the program input.

B. TWO-PARTS ATTENTION-BASED NEURAL NETWORK

Our network is a two-parts network, the front-end network is the attention network based on *GRU* [49] and the back-end network is the regression network based on *CNN* [18].

Although the structure of our network is different from traditional attention-based seq2seq network structure. We give a brief prove the transform equation of our network is basic the same with tradition attention-based network [11] in Section V.

C. TWO-STEP TRAINING METHOD

High fitting accuracy is the main goal of traditional attention based network and the explanation is not important. However, the network we need is the fitted attention network that can calculate the relevance score correctly. The network fitted by common training method may not always provide meaningful explanations [26]. In view of this situation, we propose two-step training method to train our network. Different part of two-parts network is trained with different order and data-set to guarantee the attention network can provide the correct explanation.

D. STATISTICAL ANALYSIS METHOD

Although two-step training method can help one network to calculate the relevance score correctly, the possibility of faulty key segment marking still exists for one single network. We train the networks for multiple rounds and epochs. And improve the key segment marking accuracy by statistical analysis method.

Figure 3 shows the overall process of key segment marking. The yellow blocks are the true key segments of the program input, the blue shadow blocks are the rough key segments calculated by attention network and the red shadow blocks are the final key segments analyzed by statistical analysis method.

V. IMPLEMENTATION

We describe the different components of our scheme in detail below.

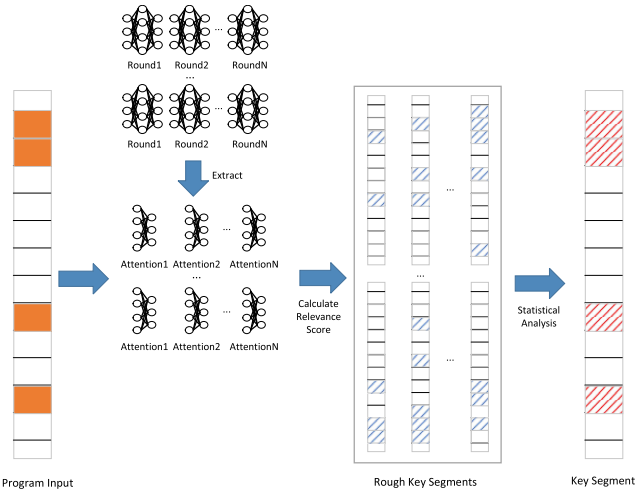


FIGURE 3. The overall process of key segment marking.

Algorithm 2 Mutating Algorithm

Input: *input* ← test case

1: **for** k ∈ [0, *len*(*input*)) **do**
2: *tmp_input* ← *input*
3: *tmp_input*[*k*] ← *randint*(0, 0*xFF*, 1)
4: *WriteToFile*(*tmp_input*)
5: *tmp_input* ← *input*
6: *randlen* ← *randint*(1, *len*(*input*) − *k*, 1)
7: *tmp_input*[*k* : *k* + *randlen*] ← *randint*(0, 0*xFF*, *randlen*)
8: *WriteToFile*(*tmp_input*)
9: **end for**

A. DATA-SET CONSTRUCTION

Besides the architecture of neural network, the training data-set also affects the fitting rate greatly. The training data-set includes input data with corresponding label data, which refers to program input and the number of target function execution times. We introduce the approaches for collecting network input data and constructing related label data.

1) DATA-SET GENERATION

Training network requires a huge amount of program inputs. The target test-case to be analyzed is single, so a method that can generate test-cases based on single file is urgent. We leverage a simple mutating algorithm to obtain program inputs, the specific algorithm is shown in Algorithm 2.

There are three reasons to utilize such algorithm to generate training data. (1) is that fixed length data is good for training the neural networks, and our algorithm will not change the length of program input; (2) is that our back-end network is a *CNN* based regression network, each *CNN* layer processes a block-like data and randomly change the block of original data can help *CNN* network identify the character of target program input; (3) is that the change of input may cause the change of label, which can rich the variety of data-set and improve the fitting accuracy.

Large amount of program inputs are collected by running the Algorithm2 for enough times. As for labels, we record the number of target function execution times through the binary instrumentation tool PinTool [25].

2) DATA-SET FILTERING

Many program inputs generated by Algorithm 2 cannot trigger the target function, which are regarded as noise data in this paper. The number of noise data should be selected appropriately since excessive noise data could result in low fitting rate, while insufficient noise data might cause over-fitting. Therefore, the noise data should be added appropriately in the data-set. In this paper, the training data-set is constructed according to a certain proportion of non-noise data and noise data.

3) PROGRAM INPUT EMBEDDING

User input embedding converses the program input into numeral matrix which is acceptable to the neural network. There are two general input embedding methods, real-valued vector embedding and one-hot vector embedding. The real-valued vector embedding maps the program input from the byte set $\{0x00, 0x01, \ldots, 0xFF\}^N$ to $\{\mathbb{R}\}^N$, converts a program input into an $N \times 1$ float vector. However, the real-valued element need to multiply by the relevance score calculated by attention network before the regression network, and may turn one element to another, thus causing the fitting error while training. One-hot embedding converts the program input from the integer set $\{0x00, 0x01, \ldots, 0xFF\}^N$ to a 256-dimension vector $\{[1, 0, \ldots, 0]^T, [0, 1, \ldots, 0]^T, \ldots, [0, 0, \ldots, 1]^T\}^N$ [24]. Although one-hot vector contains high dimension and introduces more parameters for the network, which resulting in time consumption and memory redundancy while training and predicting. Each element of one-hot vector is orthogonal, which means that no side effect when multiplied by relevance score. Thus we choose one-hot vector embedding to model program input.

The output of network is the number of target function execution times $y \in \mathbb{N}$, which is a positive integer number or zero. We use real-valued number to model y.

B. ARCHITECTURE OF ATTENTION-BASED TWO-PARTS NEURAL NETWORK

The overall of network structure is shown in Figure 4. The *GRU* layer and *Dense* layer calculate the relationship among every component of one-hot vector input, the *softmax* layer then generates the relevance score of each component, and origin one-hot vector input multiply the relevance score which is the $weighted-program-input$. The back-end *CNN* maps the $weighted-program-input$ into the execution times of target function.

1) ATTENTION-BASED FRONT-END NETWORK

The *GRU* is a natural generalization of feed forward neural networks to sequences [13], [14] and known to learn

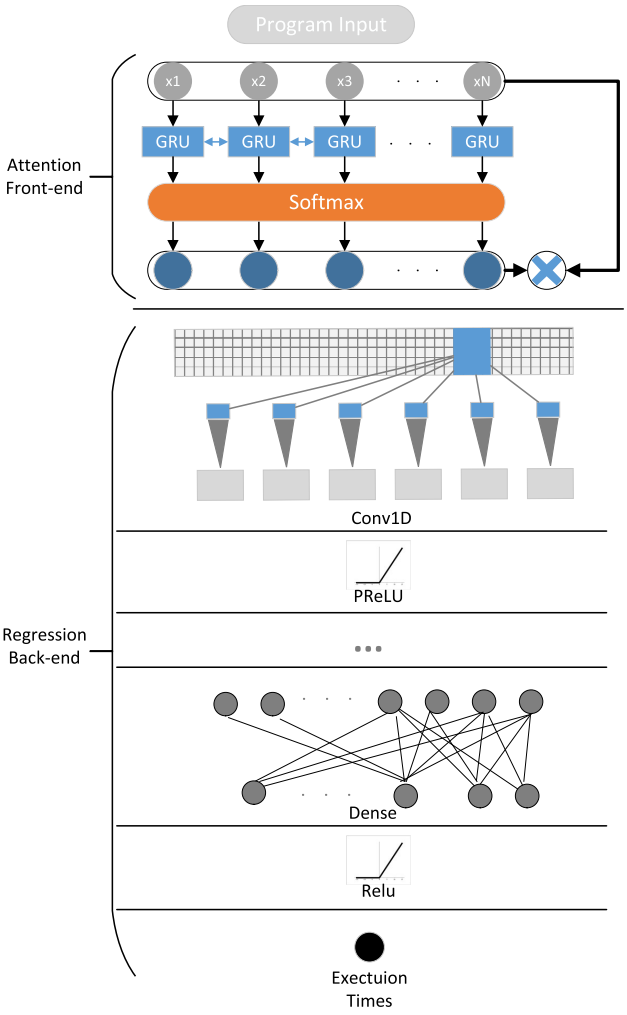


FIGURE 4. The attention-based network structure.

problems with long range temporal dependencies [12]. Given a program input model in one-hot vector $(x_1, x_2, \ldots, x_N)$ and $x_i \in \{0, 1\}^{256}$. Here, we define the *GRU* unit at each time step t to be a collection of vectors: an update gate z_t, a reset gate r_t, a candidate set lt and a hidden state h_t. And the *GRU* transition equations are the following Equation (1):

$$\begin{aligned} r_t &= sigmoid(W_r \cdot [h_{t-1}, x_t]), \\ z_t &= sigmoid(W_z \cdot [h_{t-1}, x_t]), \\ l_t &= tanh(W_l \cdot [r_t \times h_{t-1}, x_t]), \\ h_t &= (1 - z_t) \times h_{t-1} + z_t \times l_t, \\ o_t &= sigmoid(W_o \cdot h_t), \end{aligned} \tag{1}$$

where x_t is the $t - th$ component of program input and $\times$ denotes element-wise multiplication, [] denotes the cascade of two vectors [15]. Since the value of the gating variables vary for each vector element, the *GRU* layer can learn to represent information over multiple components scales.

The output of GRU can be simplify as Equation (2)

$$\begin{aligned} h_t &= f_{GRU}([h_{t-1}, x_t], h_{t-1}), \\ o_t &= sigmoid(W_o \cdot h_t) \end{aligned} \tag{2}$$

where f_{GRU} is the simplified *GRU* transition equation. The output o_t can represent the relationship between $\vec{x_t}$ and $(\vec{x_1}, \vec{x_2}, \ldots, \vec{x_{t-1}})$, and such form is similar to the attention mechanism in [26].

o_t can represent the importance of $\vec{x_t}$ in $(\vec{x_1}, \vec{x_2}, \ldots, \vec{x_{t-1}})$. However, the relevance score of $\vec{x_t}$ should be the importance of $\vec{x_t}$ in $(\vec{x_1}, \vec{x_2}, \ldots, \vec{x_N})$, and the score of every dimension among all $(\vec{x_1}, \vec{x_2}, \ldots, \vec{x_N})$ should also have a uniform standard. So we permute the output of *GRU* and pass it through *Dense* layer.

Dense layer is also named as fully connected(FC) layer and consists of linear transformation. The permuted output $(o_1, o_2, \ldots, o_N)^T = (\vec{D_1}, \vec{D_2}, \ldots, \vec{D_{256}})$ is then pass to two 256-channel *Dense* layers, that is Equation (3)

$$\begin{aligned} M_t &= Relu(W^{(1)}{}_t \cdot D_t + b^{(1)}{}_t), \\ m_t &= Relu(W^{(2)}{}_t \cdot M_t + b^{(2)}{}_t), \\ & 1 \le t \le 256, \quad t \in \mathbb{N} \end{aligned} \tag{3}$$

where *Relu* is an activation function. $(m_{1t}, m_{2i}, \ldots, m_{Ni})$ denotes the importance of $\vec{x_t}$ in $(\vec{x_1}, \vec{x_2}, \ldots, \vec{x_N})$.

The permuted output of *Dense* layer $(\vec{m_1}, \vec{m_2}, \ldots, \vec{m_{256}})^T$ is passed to the N-channel *softmax* layer to get a uniform standard of relevance score.

Softmax is a function that takes as input a vector of N real numbers, and normalizes it into a probability distribution consisting of N probabilities.

The over all transition equation of attention network is in Equation (4).

$$\begin{aligned} \vec{h_t} &= f_{GRU}([\vec{h_{t-1}}, \vec{x_t}], \vec{h_{t-1}}), \\ \vec{o_t} &= sigmoid(W_o \cdot \vec{h_t}), \\ \vec{M} &= Relu(W^{(1)} \cdot tanh(\vec{o})^T + b^{(1)}), \\ \vec{m} &= Relu(W^{(2)} \cdot \vec{M} + b^{(2)}), \\ w_{ij} &= \frac{e^{m_{ji}}}{\sum_{k=1}^{N} e^{m_{ki}}} \end{aligned} \tag{4}$$

The output of attention network is $\vec{w}$ and $0 < w_{ij} < 1$. All dimensions except X_t dimension of x_t are zero. And the input of back-end network is $\vec{x} \odot \vec{w}$. So $(w_{1X_1}, w_{2X_2}, \ldots, w_{NX_N})$ can be the relevance score of program input $(X_1, X_2, \ldots, X_N)$. And the higher the w_{tX_t} is, the more important that X_t is in $(X_1, X_2, \ldots, X_N)$.

2) REGRESSION-BASED BACK-END NETWORK

Generally, one attention layer is only effective to single output [26], and classification network is multiple output. Thus we choose regression network as back-end network. There are three network structures to build a regression network, which are feed forward neural network [17], *RNN* and *CNN*. With the expanding scale of software, the size of program input has gradually increased, and the program input of some commercial software has reached megabytes. The parameters to be trained of the forward neural network increase exponentially with the increase of network input, making it unable to deal with long sequence input. And gradient disappear problem often occurs during training the feed forward network. The *RNN* structure is uncommonly used in regression analysis. Typically *RNN* suffers long training time and low training efficiency compared with *CNN* [18]. *CNN*s have recently enjoyed a great success in classification network [10] and regression network [19], [20]. And *CNN*s can also deliver outstanding performance in the availability of large training set and making the training practical with powerful *GPU* implementations. As a result, we construct a regression network with *CNN*.

The output of attention network $\vec{w}$ multiplies the one-hot vector input $\vec{x}$ and pass it to the *CNN* based regression network. The output is Equation (5).

$$y = f_{CNN}([w_{1X_1} \cdot \vec{x_1}, w_{2X_2} \cdot \vec{x_2}, \ldots, w_{NX_N} \cdot \vec{x_N}]) \tag{5}$$

And the transition equation of the whole network is in Equation (6).

$$\begin{aligned} \vec{h_t} &= f_{GRU}([\vec{h_{t-1}}, \vec{x_t}], \vec{h_{t-1}}), \\ \vec{o_t} &= sigmoid(W_o \cdot \vec{h_t}), \\ \vec{M} &= Relu(W^{(1)} \cdot tanh(\vec{o})^T + b^{(1)}), \\ \vec{m} &= Relu(W^{(2)} \cdot \vec{M} + b^{(2)}), \\ w_{ij} &= \frac{e^{m_{ji}}}{\sum_{k=1}^{N} e^{m_{ki}}}, \\ y &= f_{CNN}([w_{1X_1} \cdot \vec{x_1}, w_{2X_2} \cdot \vec{x_2}, \ldots, w_{NX_N} \cdot \vec{x_N}]) \end{aligned} \tag{6}$$

Although the structure of our network is different from traditional attention-based network structure. But the total framework of Equation (6) is basic the same with traditional attention-based network [11].

C. TWO-STEP TRAINING METHOD

All traditional attention-based neural networks train the network as a whole. The attention layer may not always work well in explanation [26]. There are three reasons causing such failure, (1) is that the training data-set is not good enough to distinguish different labels and the attention layer focuses on the wrong part of the input to make prediction. It is extremely difficult to obtain perfect training data-set and such problem always appears when the network is over-fitting; (2) is that the attention layer introduces more training parameters for the whole network and it is easy to trigger over-fitting when network has too many parameters. And the value calculated by attention network will be abnormal when over-fitting occurs.(3) the layers behind the attention layer have no prior knowledge while training. And it is likely to turn the attention layer to a common regression layer like others and the regression layer is not explainable.

We propose a two-step training method to train the network and improve the probability of training a "good" attention layer. And the overall training method is shown in Figure 5. The back-end network is trained with averaged one-hot vector inputs in the first step. First step training just train the regression network and no attention layer is involved. Hence less over-fitting occurs. The whole network is trained with the

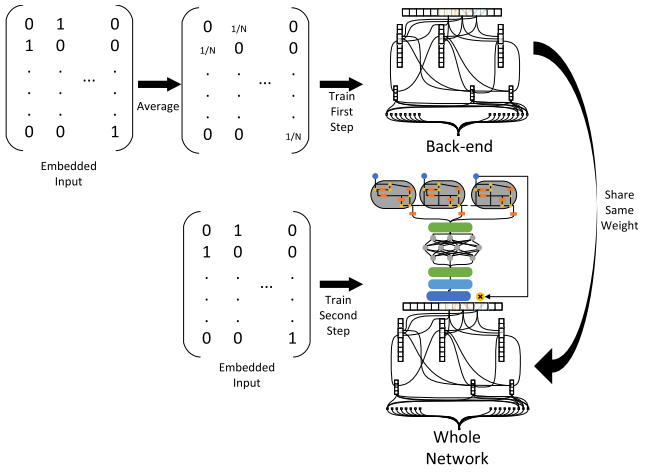


FIGURE 5. The two-step training method.

origin one-hot vector inputs in the second step. The change of key segment component will influence the output of fitted back-end greatly and the layers that process key segment component have a larger gradient value than other layers. That is, the attention network can get a clearly direction of where to focus as soon as training starts. Thus the attention network can focus on the important part of the input and improve the explainable ability.

The two-step training method can avoid some ill fittings of attention network, but cannot avoid them all. Over-fitting always occurs due to the random initialization and random gradient descent algorithm. Under these circumstances, the network is trained for several rounds and several epochs every round. We then extract the attention networks, compute the relevance scores of them and send them to statistical analysis algorithm to achieve accurate key segment marking.

D. STATISTICAL ANALYSIS ALGORITHM

Assuming that the network is trained for R rounds and each time for M epoches. $A^r{}_m$ denotes the $m-th$ attention network of the $r-th$ round training. The attention network $A^r{}_m$'s whole network fitting accuracy is $\alpha^r{}_m$. The test sample is $\vec{X} = (X_1, X_2, \ldots, X_N)$. The output of $A^r{}_m$ is $\vec{W}$. And the statistical analysis algorithm is shown in Algorithm 3.

For one round training, some epoches may suffer over-fitting problem that and some may suffer low fitting accuracy problem, it is very difficult to select a proper trained network to calculate the correct relevance score among all epochs. For every input component x_i, we multiply the relevance score with fitting accuracy $w_i \times \alpha$ and add the result of all epoches together(as in line 8). By adding all $\vec{w}$ together, the $\vec{w}$ of over-fitting network can be neutralized by other $\vec{w}$ of well-fitting network. And by multiplying with α, the $\vec{w}$ of low fitting accuracy network will take less importance among all epoches. So the new relevance score $\vec{w_{new}}$ can represent the relatively correct relevance score of one round fitted network. Then k rough key segments is selected by sorting $\vec{w_{new}}$ with descending order.

Algorithm 3 The Statistical Analysis Algorithm

Input: $\vec{X} \leftarrow$ program input
$\vec{x} \leftarrow$ one-hot vector program input
$A \leftarrow$ attention network
$\alpha \leftarrow$ fitting accuracy of the whole network
$\vec{W} \leftarrow$ the output of attention network
$\vec{w} \leftarrow$ relevance score
$k \leftarrow$ the top k number selected from the target list
Output: $KeySegment \leftarrow$ the key segment of $\vec{x}$

```
1:  keysegment_dict ← dict()
2:  for r ∈ R − rounds do
3:      score_w ← []
4:      for m ∈ M − epoches − of − round − r do
5:          W ← A^r_m.predict(X)
6:          w ← (W_{1x_1}, W_{2x_2}, ..., W_{Nx_N})
7:          for w_i ∈ w do
8:              score_w_i ← score_w_i + w_i × α^r_m
9:          end for
10:     end for
11:     for indice ∈ topk(k, score_w) do
12:         keysegment_dict[indice] + +
13:     end for
14: end for
15: for indice ∈ keysegment_dict.keys do
16:     if keysegment_dict[indice] > 0.4 × R then
17:         KeySegment.append(indice)
18:     end if
19: end for
20: return KeySegment
```

Every round training starts over with random initialization. Some rounds networks may suffer from over-fitting, and the attention layer may be inaccurate. Training loss is another problem that always occurs during network training and some marked key segments are inaccurate. Over-fitting and training loss are usually random, and will not dominant in all networks. In order to avoid the errors caused by such randomness as far as possible, we count the rough key segments in every round(as in *line*12) and select the segments whose count is larger than threshold as key segments(as in *line*16 $\sim$ 17). The threshold we set is a certain proportion of the total round number and the proportion is 0.4 in *line*16.

VI. EXPERIMENTS

We select 4 linux real world programs. The target file formats include jpg, elf, xml and pdf files. Each program is compiled under ubuntu 16.04.11 $\times$64 system with the default compile option and the gcc version is 5.4.0 build 20160609. For each program we choose one position that the test case would trigger. And we use *AttentionMark* to mark the key segments for each test case. And we select *S2E* and *NeuralTaint* as the comparison tools. The performance of *AttentionMark* is evaluated on three aspects:(1) the effectiveness of two-step training method; (2) the accuracy of key segments marking performance; (3) time consumption.

TABLE 2. Network configurations.

AttentionMark	AttentionMark	NeuralTaint
Attention Layer	Regression Layer	
input($inputlen \times 256$)		
GRU-256 tanh Permute Dense-inputlen PReLU Dense-inputlen Permute softmax	conv1D-128,3,2 PReLU conv1D-64,3,2 PReLU conv1D-32,3,2 PReLU Flatten Dense-256 PReLU Dense-1 ReLU	conv1D-300,3,2 PReLU conv1D-300,3,2 PReLU conv1D-300,2,1 PReLU conv1D-300,2,1 PReLU conv1D-300,2,1 PReLU conv1D-300,2,1 PReLU conv1D-300,2,1 PReLU conv1D-300,2,1 PReLU
output $inputlen \times 256$	output(1)	output(1)

* $inputlen$ is the length of the network input
* the number behind the layer denotes the units parameter

A. EXPERIMENT SETTINGS

All networks are trained in Ubuntu 16.04 x64 system and trained by keras 2.11.1 with tensorflow 1.11.0 as back-end. The graphics card is GTX 1080Ti, CPU is 5820k @3.4GHz, and the memory is 32G.

We leverage Algorithm 2 to collect program inputs based on the target test case. The label data of target function is recorded by a plugin of Pintool [25]. We executes the target test cases on multiple cores since our CPU contains 12 logic cores, thus save a lot of time in the execution of target test cases.

We regard the program input that result in zero output as noise data. First we filter the data-set to make sure that the numbers of non-noise data with different label contents are as close as possible. Then we mix the non-noise data and noise data in about 3: 1 ratio. After filtering and mixing, 15% of the data-set is selected as verification data and the rest is training data.

The overall network configuration is shown below in Table 2.

The network training procedure generally follows the common regression network [21]. Namely, the training is carried out by optimizing the mean square error(MSE [9]) objective using Adam [22](based on back-propagation [23]) with momentum. The batch size is set to 128, momentum to 0.8. The learning rate is initially set to 10^{-7}. The initialization of the network parameters is Xavier [27], [28].

For *AttentionMark*, we train network for 20 rounds and 10 epochs each round. For every round, we use two-step training method and common training method to train the network. The k in Algorithm 3 is 15% of the length of test case.

For *NeuralTaint*, we train network for 50 rounds.

For S2E, we check the registers of conditional jump related to target function. When registers are not tainted, we analyze the path constrains when target function is triggered and get the key segments.

```
ff d8 ff d8 00 10 4a 46 ff d8 00 01 01 d8 00 01 00 01 00 00 ff db 00
43 00 08 ff d8 07 06 05 08 07 07 07 09 0c 19 12 14 ff d8 1f 1e 1d
1c 20 24 2e 27 20 22 2c 23 1c 1c 28 37 29 2c 30 31 34 34 33 34 32
ff db 00 43 01 09 09 09 0c 0b 0c 18 0d 0d 18 32 21 1c ff d8 d8 d8
32 32 d8 d8 d8 32 32 32 ff c0 00 11 08 02 51 01 f4 03 01 22 00 02
11 01 03 11 01 ff c4 00 1f 00 00 01 05 01 01 01 01 01 01 00 00 00
00 00 00 00 00 01 02 03 04 05 06 07 08 09 0a 0b ff c4 00 b5 10 00
02 01 03 03 02 04 03 05 05 04 04 00 00 01 7d 01 02 03 00 04 11
05 12 21 31 41 06 13 51 61 07 22 71 14 32 a1 08 23 42 b1 c1 84
85 86 87 88 89 8a 92 93 94 95 96 97 98 99 9a a2 a3 03 01 01 01
01 01 01 01 01 00 02 01 02 04 04 03 04 07 05 04 04 00 01 02 77
00 01 02 03 11 04 05 21 71 13 22 32 81 08 14 42 91 a1 b1 c1 09
23 33 52 f0 15 62 72 d1 0a 16 24 34 e1 25 26 27 28 29 2a 35 36
37 38 39 3a 43 44 45 46 47 48 49 4a 53 54 55 56 57 58 96 97 98
99 9a a2 a3 a4 a5 a6 a7 a8 a9 aa b2 b3 b4 b5 b6 b7 b8 b9 ba c4 c7
c8 c9 ca d2 d3 d4 d5 f8 f9 fa ff da 00 0c 03 01 00 02 11 03 11 00 3f
4a 29 a2 9d c5 7e 94 7c 3d 85 1c 1a 78 6f 61 4d 15 2a a8 c7 35 2d
9a 41 83 04 7b d4 80 60 7f 8d 1b d9
```

FIGURE 6. The test case of jhead program. Yellow blocks are the key segments.

B. EXPERIMENT RESULTS

1) PROGRAM 1: JHEAD

Jhead is an image processing software in linux. The target file format is *jpg*. We choose the function *process_DHT* as the target function. Such function is used to evaluate the quality of picture. Triggering the function requires a $switch - case$ structure which can cause control flow dependency. The target file must have a file head with $FF\ D8$ and the *jpg* segment head must be $FF\ C4$ to trigger the *process_DHT* function and two bytes behind $FF\ C4$ checks weather the length of the segment is illegal.

The test case is shown in Figure 6. And the indices of key segments are [0, 1, 116, 117, 118, 119, 149, 150, 151, 152]. The key segments are $FF\ D8$(file head), $FF\ C4$(key word) and the data length followed them. The target test case can trigger *process_DHT* two times. The noise data generated by Algorithm 2 is 3300 and non-noise data is 9000.

After training and statistical analysis, the importance score of *AttentionMark*, *AttentionMark* without two-step training and *NeuralTaint* is shown in Figure 7.

In Figure 7, *AttentionMark* and *NeuralTaint* have higher relevance score on the key segments and *AttentionMark* with common training method does not have the higher relevance score on the key segments. However, *NeuralTaint* method focus on several wrong indices due to the several over-fitting rounds training.

Since the flag registers of control branch of *process_DHT* contains no symbolic memory due to the control flow dependency structure of jump table(switch-case structure). *S2E* checks the path constrains when triggering the *process_DHT*, and many other irrelevant indices are marked as key segments. The *S2E* marks every segment head of png file due to those heads affect the execution of other branches.

The analysis results of jhead program is shown in Table 3. The accuracy rate is the number of correctly marked indices divides the number of total correct indices. The false positive

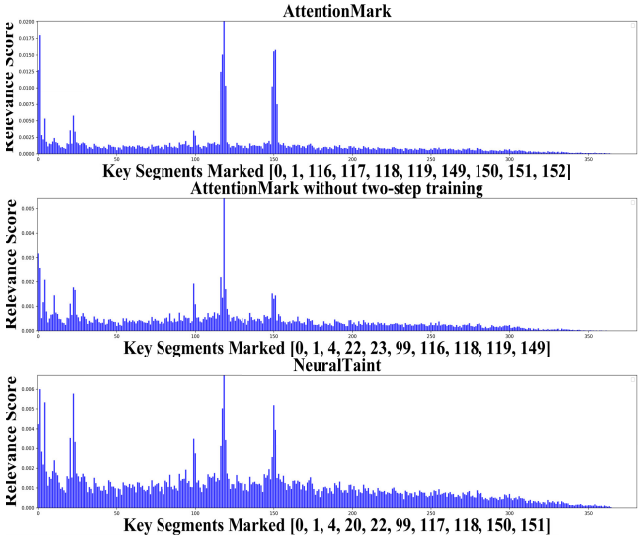


FIGURE 7. Relevance score of jhead for neural network based methods.

TABLE 3. Analysis results of jhead.

Tools	Result(indices)	AR	FP	T
AM	0,1,116,117,118,119,149,150,151,152	1.0	0.0	3120
AM no 2	0,1,4,22,23,99,116,118,119,149	0.6	0.4	2898
NT	0,1,4,22,99,116,118,150,151,152	0.7	0.3	2767
$S2E$	0,1,8,9,10,11,20,21,22,23,24,116,117, 118,119,149,150,151,152	1.0	0.47	90

*AM stands for $AttentionMark$, AM no 2 stands for $Attention-Mark$ without two-step training and NT stands for $NeuralTaint$
*AR stands for accuracy rate, FP stands for false positive rate and T stands for time consumption of whole process of key segment marking (second)

is the number of wrongly marked indices divides the number of total marked indices.

Form the Table 3, we can see that *AttentionMark* can mark the key segments accurately because of two-step training method and statistical analysis algorithm and can deal with the control flow dependency structure like jump table. And *NeuralTaint*'s gradient value becomes abnormal because of several rounds over-fitting training. For the time consumption, neural network based methods cost more time than traditional method because neural network needs training. Since *AttentionMark* has a more complex structure than *NeuralTaint*, *AttentionMark* costs more time than *NeuralTaint*.

2) PROGRAM 2: READELF

Readelf gets information from an elf file. The target file format is *elf*, an executable file format under linux platform. We choose the function *process_section_headers*, the number of execution times represents the amount of section heads. The function *process_section_headers* is executed in a cycle structure, thus the execution times is controlled by the cyclic control variable directly. The target file must have a file head with $7F$ 45 $4C$ 46, which is a standard *elf* file head, then the 48th offset of the target file is the cyclic control variable of the function *process_section_headers*.

```
7f 45 4c 46 1f af 73 54 9a 00 00 00 00 00 00 00 00 00 99 00 04 00
4d 00 00 00 27 aa c0 20 b5 b7 3b 00 00 00 0e 00 00 00 00 00 00 00
00 d5 00 60 00 06 00 9f 00 04 00 c9 00 6b 00 00 00 00 00 00 00 00
00 86 50 63 00 39 59 91 5d 8b f3 ed 04 00 00 00 1e 00 00 00 32
00 00 00 00 20 00 00 23 00 00 00 89 00 00 00 7c d4 0b 8a 02 a2
9b 2f c2 26 6d 7b 02 16 00 00 00 d9 00 00 00 b7 00 00 00 00 44
00 00 2b 73 00 00 00 d9 8c 00 00 00 de 6a 05 e9 6d 85 3f 57 c0 57
bd d9 b9 6c 79 6d a3 7e 3f 37 97 00 00 84 09 ff 9b 65 81 57 95 00
fa 08 3c 31 e8 00 41 24 55 29 a5 2a e1 dd 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 a2 00 00 00 0e 00 00 00
d2 00 00 00 d0 55 bc 3e 00 00 00 a3 00 00 00 00 00 00 00 00 00 00
00 00 17 00 00 00 00 00 00 ca 00 00 00 b9 00 00 00 54 00 00
00 55 3f dc b4 db a8 00 00 00 1c 00 00 00 00 00 00 00 00 00 00 00
3b 00 00 00 00 00 00 00 76 00 00 00 41 00 00 00 00 00 00 00 00 00
00 00 00 1b 00 00 00 b6 00 00 00 00 00 00 00 00 00 00 00 74 96
c0 00 00 00 00 00 00 fa
```

FIGURE 8. The test case of readelf program. Yellow blocks are the key segments.

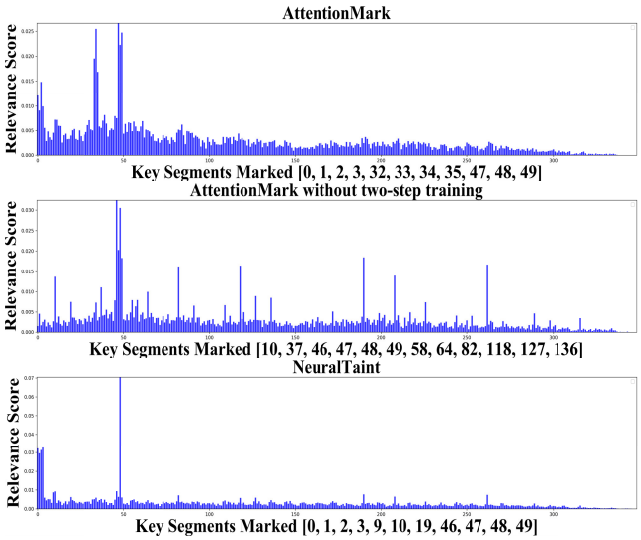


FIGURE 9. Relevance score of readelf For neural network based methods.

The test case is shown in Figure 8. And the indices of key segments are [0, 1, 2, 3, 33, 34, 35, 47, 48, 49]. The key segments are $7F$ 45 $4C$ 46(file head) and the 48th offset of input is the cyclic control variable of target function. The target test case can trigger *process_section_headers* six times. The noise data generated by Algorithm 2 is 7260 and non-noise data is 19800.

After training and statistical analysis, the importance score of *AttentionMark*, *AttentionMark* without two-step training and *NeuralTaint* is shown in Figure 9.

In Figure 9, *AttentionMark* and *NeuralTaint* have higher relevance score on the key segments and *AttentionMark* with common training method does not have the higher relevance score on all the key segments. However, *NeuralTaint* method focus on several wrong indices due to the several over-fitting rounds training.

For $S2E$, we directly checks the cyclic control variable of *process_section_headers* manually. Because every

TABLE 4. Analysis results of readelf.

Tools	Result(indices)	AR	FP	T
AM	0,1,2,3,32,33,34,35,47,48,49	1.0	0.09	3520
AM no 2	10,37,46,47,48,49,58,64,82,118,127, 136	0.3	0.75	3398
NT	0,1,2,3,9,10,19,46,47,48,49	0.7	0.36	3267
$S2E$	47,48,49	0.3	0.0	800

*AM stands for $AttentionMark$, AM no 2 stands for $Attention-Mark$ without two-step training and NT stands for $NeuralTaint$
*AR stands for accuracy rate, FP stands for false positive rate and T stands for time consumption of whole process of key segment marking (second)

component of *elf*'s head is processed by the program, and the path constrains of *process_section_headers* contain all indices from [0-49].

The analysis results of readelf program is shown in Table 4.

Form the Table 4, we can see that *AttentionMark* can mark the key segments accurately because of two-step training method and statistical analysis algorithm and can deal with the cycle structure. *NeuralTaint* has a very high gradient value at the cyclic control variable and other gradient value becomes inaccurate. For the program like readelf that every byte of the user input is processed by the program, it is quite difficult for traditional analysis tool like *S2E* to analysis the control information flow if the target's directly branch registers are not tainted. And under taint and over taint problem often arise here even with manually help.

3) PROGRAM 3: MUPDF

Mupdf is a *pdf* processing software under linux. The target file format is *pdf*. We choose the target function *pdf_load_object*. Target function process the *obj* object of target *pdf* file. Such function is called by a function pointer, which means the function is dynamically invoked. The execution times of function *pdf_load_object* depends on the number of object. The legal object structure consists a sequence number, a version number, key word "obj"(6F 62 6A), content split symbol and space(20) among them. This experiment target is a good case to test weather our method can deal with virtual pointer structure.

The test case is shown in Figure 10. And the indices of key segments are $[9-17, 43, 44, 51-60, 93, 94, 100-109, 138, 139, 146-155, 178, 179, 186-195, 218, 219, 225-233]$. And target test case can trigger *pdf_load_object* six times. The noise data generated by Algorithm 2 is 5280 and non-noise data is 14400.

After training and statistical analysis, the importance score of *AttentionMark*, *AttentionMark* without two-step training and *NeuralTaint* is shown in Figure 11.

In Figure 9, all the methods show six peaks, meaning that the target input should be divided into six parts which is corresponding to the fact. In *AttentionMark*'s relevance score figure, the heights of each peak are nearly the same. *AttentionMark* with common training method has one extremely high peak. *NeuralTaint*'s relevance score has more

25 50 44 46 2d 31 2e 30 0a 31 20 30 20 6f 62 6a 3c 3c 2f 54 79 70
65 2f 43 61 74 61 6c 6f 67 2f 50 61 67 65 73 20 32 20 30 20 52 3e
3e 65 6e 64 6f 62 6a 20 32 20 30 20 6f 62 6a 3c 3c 2f 54 79 70 65
2f 50 61 67 65 73 2f 4b 69 64 73 5b 33 20 30 20 52 5d 2f 43 6f 75
6e 74 20 31 3e 3e 65 6e 64 6f 62 6a 20 33 20 30 20 6f 62 6a 3c 3c
2f 54 79 70 65 2f 50 61 67 65 2f 4d 65 64 69 61 42 6f 78 5b 30 20
30 20 33 20 33 5d 3e 3e 65 6e 64 6f 62 6a 20 34 20 30 20 6f 62 6a
3c 3c 2f 54 79 70 65 2f 4f 75 74 6c 69 6e 65 73 2f 43 6f 75 6e 74
20 30 3e 3e 65 6e 64 6f 62 6a 20 35 20 30 20 6f 62 6a 3c 3c 2f 54
79 70 65 2f 4f 75 74 6c 69 6e 73 2f 43 6f 75 6e 74 20 30 3e 3e 78
78 78 78 78 78 20 36 20 30 20 6f 62 6a 20 3c 3c 20 2f 4c 65 6e 67
74 68 34 34 20 3e 3e 20 73 74 72 65 61 6d 20 42 54 20 2f 46 31
20 32 34 20 54 66 20 31 30 30 20 31 30 30 20 54 64 28 48 65 6c
6c 6f 20 57 6f 72 6c 64 29 20 54 6a 20 45 54 20 65 6e 64 73 74 72
65 61 6d 20 65 6e 64 6f 62 6a 20 74 72 61 69 6c 65 72 3c 3c 2f 53
69 7a 65 20 35 2f 52 6f 6f 74 20 31 20 30 20 52 3e 3e 0a

FIGURE 10. The test case of mupdf program. Yellow blocks are the key segments.

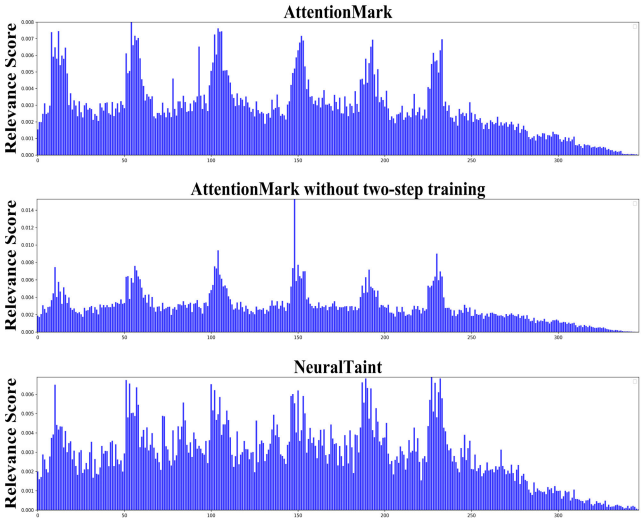


FIGURE 11. Relevance score of mupdf For neural network based methods.

"noise" than the other two methods, which may affect the accuracy and false positive of the analysis result.

For *S2E*, the control branch of *pdf_load_object* contains no symbolic memory due to the virtual pointer. And the process of decoding obj structure is in information flow. Therefore, the path constrains only contains the process of file head checking. However, the execution of *pdf_load_object* function has nothing to do with file head checking.

The analysis results of mupdf program is shown in Table 5.

Form the Table 5, we can see that *AttentionMark* can mark the key segments accurately because of two-step training method and statistical analysis algorithm and can deal with the virtual pointer structure and dynamic invoking. *NeuralTaint* still has more "noise" in its relevance score than *AttentionMark* which leads to inaccuracy. *S2E* cannot analysis the control flow dependency structure contains dynamic invoking and virtual pointer.

TABLE 5. Analysis results of mupdf.

Tools	Result(indices)	AR	FP	T
AM	9, 10, 11, 12, 13, 14, 15, 16, 17, 43, 44, 51, 52, 53, 54, 55, 56, 57, 58, 59, 93, 100, 101, 102, 103, 104, 105, 106, 107, 108, 128, 138, 139, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 178, 179, 186, 187, 188, 189, 190, 191, 192, 193, 194, 218, 225, 226, 227, 228, 229, 230, 231, 232, 233, 236, 247	0.94	0.05	3450
AM no 2	8, 9, 10, 11, 12, 13, 15, 16, 18, 48, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 64, 66, 92, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234	0.83	0.17	3212
NT	10, 11, 12, 13, 14, 51, 52, 53, 54, 55, 56, 57, 58, 61, 63, 72, 73, 78, 81, 83, 84, 85, 100, 101, 102, 103, 104, 105, 107, 108, 109, 110, 126, 135, 136, 138, 146, 147, 148, 150, 152, 153, 154, 157, 163, 165, 169, 180, 186, 187, 188, 189, 190, 191, 192, 193, 195, 196, 201, 217, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234	0.76	0.31	2967
$S2E$	0, 1, 2, 3, 4, 5, 6, 7	0	1	430

*AM stands for $AttentionMark$, AM no 2 stands for $Attention-Mark$ without two-step training and NT stands for $NeuralTaint$
*AR stands for accuracy rate, FP stands for false positive rate and T stands for time consumption of whole process of key segment marking (second)

```
3c 63 6f 64 65 42 6f 6f 6b 3e 20 20 0a 20 20 20 20 20 20 3c 69 6e
76 61 6c 72 6e 67 20 72 65 6c 61 74 69 6f 6e 3d 22 45 51 22 3e 0a
20 20 20 20 20 20 20 20 20 3c 69 74 65 6d 20 55 4e 49 54 53 3d
22 49 4e 54 22 20 56 41 4c 55 45 3d 22 39 22 2f 3e 0a 20 20 20 20
20 20 3c 2f 69 6e 76 61 6c 72 6e 67 3e 0a 20 20 20 3c 76 61 72 20
6e 61 6d 65 3d 22 43 41 53 45 49 44 22 20 49 44 3d 22 43 41 53
45 49 44 22 20 66 6f 72 6d 61 74 3d 22 22 20 64 63 6d 6c 3d 22
30 22 3e 0a 20 20 20 20 20 20 3c 6c 61 62 6c 20 6c 65 76 65 6c 3d
22 76 61 72 69 61 62 6c 65 22 3e 0a 20 20 20 20 20 20 20 20 20
42 65 72 6b 65 6c 65 79 20 43 61 73 65 20 49 44 20 4e 75 6d 62
65 72 0a 20 20 20 20 20 20 3c 2f 6c 61 62 6c 3e 0a 20 20 20 3c 3c
2f 76 61 72 3e 0a 3c 2f 63 6f 64 65 42 6f 6f 6b 3e 0a
```

FIGURE 12. The test case of xmlwf program. Yellow blocks are the key segments.

4) PROGRAM 4: XMLWF

Xmlwf is a *xml* processing software under linux. The target file format is *xml*. We choose the target function *docontent*. Target function process the *content tag head* object of target *xml* file. Such function is called under a switch-case structure and such structure is under a branch like the example code in Section II. The legal *content tag head* is consisted of $< .. >$ $\backslash n$ structure.

The test case is shown in Figure 12. And the indices of key segments are [0-9, 12, 19, 42, 43, 81, 82, 89, 99, 100, 104, 153, 154, 161, 183, 184, 224, 230, 231, 235, 241, 242, 243, 253, 254]. And target test case can trigger *doconten* eight times. The noise data generated by Algorithm 2 is 8580 and non-noise data is 23400.

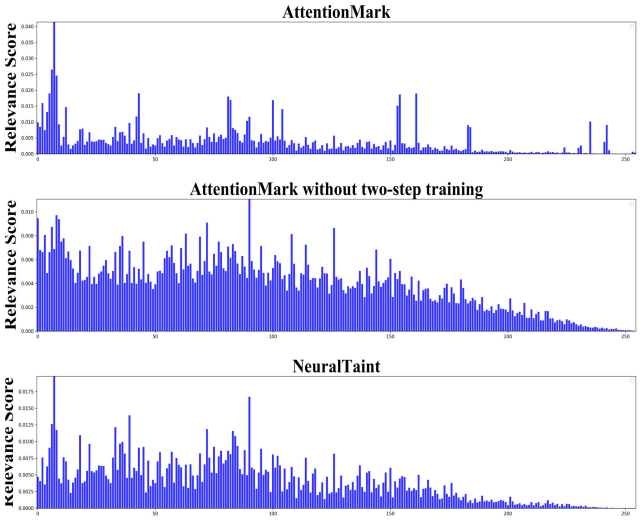


FIGURE 13. Relevance score of xmlwf for neural network based methods.

TABLE 6. Analysis results of xmlwf.

Tools	Result(indices)	AR	FP	T
AM	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 12, 18, 19, 22, 33, 35, 36, 39, 42, 43, 72, 75, 81, 82, 83, 84, 85, 89, 90, 100, 104, 153, 154, 161, 183, 184, 235, 242	0.76	0.32	4320
AM no 2	0, 1, 2, 3, 5, 6, 7, 8, 9, 10, 11, 13, 18, 22, 33, 35, 36, 39, 45, 55, 56, 57, 61, 63, 69, 72, 75, 77, 78, 81, 83, 84, 87, 90, 95, 101, 108, 114, 126, 144	0.29	0.75	4080
NT	2, 5, 6, 7, 8, 11, 12, 18, 22, 32, 33, 35, 36, 37, 39, 43, 45, 47, 51, 52, 57, 59, 69, 72, 73, 75, 77, 78, 79, 80, 81, 82, 83, 84, 85, 88, 90, 95, 100, 102, 114, 126	0.29	0.76	3890
$S2E$	0-254	1.0	0.89	1210

*AM stands for $AttentionMark$, AM no 2 stands for $Attention-Mark$ without two-step training and NT stands for $NeuralTaint$
*AR stands for accuracy rate, FP stands for false positive rate and T stands for time consumption of whole process of key segment marking (second)

After training and statistical analysis, the importance score of *AttentionMark*, *AttentionMark* without two-step training and *NeuralTaint* is shown in Figure 13.

In Figure 13, all the methods show six peaks, meaning that the target input should be divided into six parts which is corresponding to the fact. In *AttentionMark*'s relevance score figure, the heights of each peak are nearly the same. *AttentionMark* with common training method has one extremely high peak. *NeuralTaint*'s relevance score has more "noise" than the other two methods, which may affect the accuracy and false positive of the analysis result.

For $S2E$, the control branch of *doconten* contains no symbolic memory due to switch case and other control flow dependency structure. *xmlwf* processes the target file as a parser, which means that every byte in the user input is processed by the control branch of *xmlwf* . The path constrains in the *doconten* branch thus contains every component of target input.

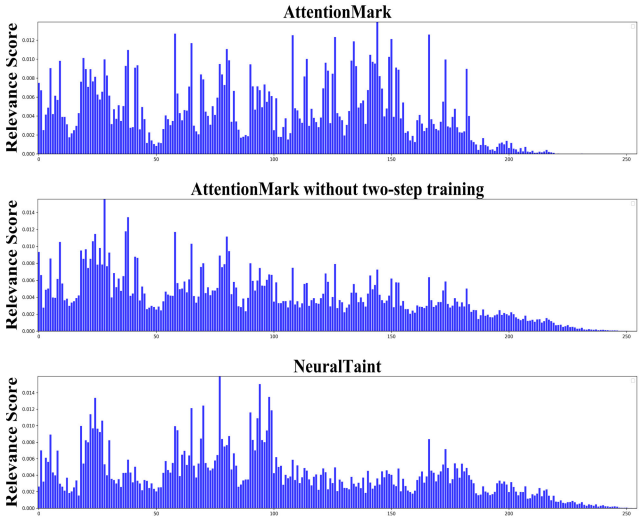


FIGURE 14. Relevance score of xmlwf for neural network based methods with visible character.

TABLE 7. Analysis results of xmlwf with visible character.

Tools	Result(indices)	AR	FP	T
AM	0, 1, 2, 4, 5, 6, 7, 8, 9, 12, 19, 42, 43, 81, 82, 83, 84, 85, 89, 100, 104, 153, 154, 161, 183, 184, 235, 242, 99, 224, 230, 231, 241, 243	0.91	0.08	4370
AM no 2	0, 1, 3, 5, 6, 7, 8, 9, 18, 22, 69, 72, 75, 77, 78, 81, 83, 126, 144, 4, 12, 19, 42, 43, 82, 89, 99, 100, 104, 153, 154, 161, 183, 184	0.71	0.29	3980
NT	2, 5, 6, 7, 8, 12, 18, 22, 32, 33, 35, 36, 37, 39, 43, 45, 47, 51, 81, 82, 100, 102, 114, 126, 1, 3, 4, 9, 19, 42, 89, 99, 104, 153, 154, 161, 183, 184	0.71	0.37	3670
$S2E$	0-254	1.0	0.89	1210

*AM stands for $AttentionMark$, AM no 2 stands for $Attention-Mark$ without two-step training and NT stands for $NeuralTaint$
*AR stands for accuracy rate, FP stands for false positive rate and T stands for time consumption of whole process of key segment marking (second)

The analysis result of xmlwf program is shown in Table 6.

Form the Table 6, we can see that *AttentionMark* has a higher accuracy rate and lower false positive rate than common training method and *NeuralTaint*. However, the analysis accuracy is not as high as other software. The reason is that *xmlwf*'s parser would stop parsing if current byte is not visible character. And the Algorithm 2 generates lot of test cases that contain invisible character, which reduces the fitting accuracy of the neural networks.

We then replace [0,0xFF] byte with the visible character in Algorithm 2, and run the *AttentionMark* and *NeuralTaint*. The relevance score with visible character is shown in Figure 14. And the analysis result of xmlwf program with visible character is shown in Table 7.

From Table 14, all three neural network methods show an improvement in analysis accuracy rate and false positive rate. *AttentionMark* can mark the key segments accurately because of two-step training method and statistical analysis algorithm and can deal with common control flow dependency structure. Traditional dynamic analysis tool *S2E* suffers over taint problem while analyzing common control flow dependency structure code.

Experiments on four general software show that attention mechanism based method shows a higher analysis rate and lower false positive rate than gradient based method and traditional dynamic analysis method. And two-step training method is more efficient than common training method. However, there are still inaccuracies and false positives in *AttentionMark*, especially some input contents need to be limited. Training method and statistical analysis algorithm still need to improve in future work. In general, attention mechanism shows great potential in key segments marking.

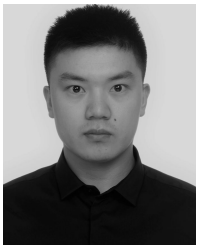
VII. CONCLUSION

In this paper, we present *AttentionMark*, a key segment marking tool based on attention mechanism and deep learning. We propose a two-parts network with attention mechanism to analyze the key segments of target program function execution and prove that the attention part of our network can work like traditional attention network both in theory and experiments. Further, we address the reasons for the inaccuracy of traditional attention network and propose the two-step training method to train our two-parts network. Experiment results show that two-step training method has higher analysis accuracy rate and lower false positive rate than common training method and gradient-based *NeuralTaint*. By comparing with traditional dynamic analysis tools, *AttentionMark* can correctly analyze control flow dependency structure like switch-case(jump table), cycle variable control and virtual pointer.

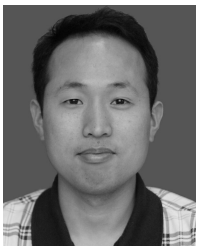
REFERENCES

[1] E. J. Schwartz, T. Avgerinos, and D. Brumley, "All you ever wanted to know about dynamic taint analysis and forward symbolic execution (but might have been afraid to ask)," in *Proc. IEEE Secur. Privacy*, May 2010, pp. 317–331.
[2] J. Clause, W. Li, and A. Orso, "Dytan: A generic dynamic taint analysis framework," in *Proc. Int. Symp. Softw. Test. Anal.*, 2007, pp. 196–206.
[3] T. Bao, Y. Zheng, Z. Lin, X. Zhang, and D. Xu, "Strict control dependence and its effect on dynamic information flow analyses," in *Proc. 19th Int. Symp. Softw. Test. Anal.*, 2010, pp. 13–24.
[4] M. G. Kang, S. McCamant, P. Poosankam, and D. Song, "DTA++: Dynamic taint analysis with targeted control-flow propagation," in *Proc. Netw. Distrib. Syst. Secur. Symp. (NDSS)*, San Diego, CA, USA, 2011, pp. 1–14.
[5] L. P. Cox, P. Gilbert, G. Lawler, V. Pistol, A. Razeen, B. Wu, and S. Cheemalapati, "SpanDex: Secure password tracking for Android," in *Proc. 23rd USENIX Secur. Symp. (USENIX Security 14)*, 2014, pp. 481–494.
[6] J. Ferrante, K. J. Ottenstein, and J. D. Warren, "The program dependence graph and its use in optimization," *ACM Trans. Program. Lang. Syst.*, vol. 9, no. 3, pp. 319–349, 1987.
[7] D. F. Specht, "The general regression neural network-rediscovered," *Neural Netw.*, vol. 6, no. 7, pp. 1033–1034, 1993.
[8] T. Liu, K. Wang, L. Sha, B. Chang, and Z. Sui, "Table-to-text generation by structure-aware seq2seq learning," in *Proc. 32nd AAAI Conf. Artif. Intell.*, 2017.

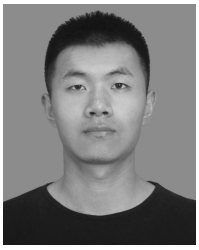
[9] Y. Ephraim and D. Malah, "Speech enhancement using a minimum mean-square error log-spectral amplitude estimator," *IEEE Trans. Acoust. Speech Signal Process.*, vol. 32, no. 6, pp. 1109–1121, Apr. 1984.
[10] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *Proc. Eur. Conf. Comput. Vis.*, 2013, pp. 818–833.
[11] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," 2014, *arXiv:1409.0473*. [Online]. Available: https://arxiv.org/abs/1409.0473
[12] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.
[13] P. J. Werbos, "Backpropagation through time: What it does and how to do it," *Proc. IEEE*, vol. 78, no. 10, pp. 1550–1560, Oct. 1990.
[14] D. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.
[15] K. Sheng, R. Socher, and C. D. Manning, "Improved semantic representations from tree-structured long short-term memory networks," 2015, *arXiv:1503.00075*. [Online]. Available: https://arxiv.org/abs/1503.00075
[16] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Trans. Neural Netw.*, vol. 5, no. 2, pp. 157–166, Mar. 1994.
[17] E. P. P. A. Derks, M. S. S. Pastor, and L. M. C. Buydens, "Robustness analysis of radial base function and multi-layered feed-forward neural network models," *Chemometrics Intell. Lab. Syst.*, vol. 28, no. 1, pp. 49–60, 1995.
[18] N. Kalchbrenner, E. Grefenstette, and P. Blunsom, "A convolutional neural network for modelling sentences," 2014, *arXiv:1404.2188*.[Online]. Available: https://arxiv.org/abs/1404.2188
[19] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, p. 354, 2017.
[20] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, p. 484, 2016.
[21] D. M. Bates and D. G. Watts, "Nonlinear regression analysis and its applications," *Technometrics*, vol. 32, no. 2, pp. 219–220, 1988.
[22] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*. [Online]. Available: https://arxiv.org/abs/1412.6980
[23] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Comput.*, vol. 1, no. 4, pp. 541–551, 1989.
[24] W.-C. Chou, P. A. Beerel, R. Ginosar, R. Kol, C. J. Myers, S. Rotem, K. Stevens, and K. Y. Yun, "Average-case optimized technology mapping of one-hot domino circuits," in *Proc. 4th Int. Symp. Adv. Res. Asynchronous Circuits Syst.*, Mar./Apr. 1998, pp. 80–91.
[25] [Online]. Available: https://software.intel.com/en-us/articles/pin-a-dynamic-binary-instrumentation-tool
[26] S. Jain and B. C. Wallace, "Attention is not explanation," 2019, *arXiv:1902.10186*. [Online]. Available: https://arxiv.org/abs/1902.10186
[27] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proc. Int. Conf. Artif. Intell. Statist.*, 2010, pp. 249–256.
[28] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," 2014, *arXiv:1408.5093*. [Online]. Available: https://arxiv.org/abs/1408.5093
[29] N. Stephens, J. Grosen, C. Salls, A. Dutcher, R. Wang, J. Corbetta, Y. Shoshitaishvili, C. Kruegel, and G. Vigna, "Driller: Augmenting fuzzing through selective symbolic execution," in *Proc. NDSS*, vol. 16, 2016, pp. 1–16.
[30] V. Chipounov, V. Kuznetsov, and G. Candea, "S2E: A platform for *in-vivo* multi-path analysis of software systems," in *Proc. 16th Int. Conf. Architect. Support Program. Lang. Oper. Syst.*, 2011, pp. 265–278.
[31] E. Shin, C. Richard, D. Song, and R. Moazzezi, "Recognizing functions in binaries with neural networks," in *Proc. 24th USENIX Secur. Symp. (USENIX Security 15)*, 2015, pp. 265–278.
[32] Z. L. Chua, S. Shen, P. Saxena, and Z. Liang, "Neural nets can learn function type signatures from binaries," in *Proc. 26th USENIX Secur. Symp. (USENIX Secur. 17)*, 2017, pp. 99–116.
[33] D. She, K. Pei, D. Epstein, J. Yang, B. Ray, and S. Jana, "NEUZZ: Efficient fuzzing with neural program smoothing," in *Proc. IEEE Symp. Secur. Privacy*, May 2019, pp. 803–817.
[34] T. Wang, T. Wei, G. Gu, and W. Zou, "TaintScope: A checksum-aware directed fuzzing tool for automatic software vulnerability detection," in *Proc. IEEE Symp. Secur. Privacy*, May 2010, pp. 497–512.
[35] X. Zhang, C. Feng, R. Li, J. Lei, and C. Tang, "NeuralTaint: A key segment marking tool based on neural network," *IEEE Access*, vol. 7, pp. 68786–68798, 2019.
[36] I. Haller, A. Slowinska, M. Neugschwandtner, and H. Bos, "Dowser: A guided fuzzer to find buffer overflow vulnerabilities," in *Proc. 22nd USENIX Secur. Symp.*, 2013, pp. 49–64.
[37] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D. Octeau, and P. McDaniel, "FlowDroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for Android apps," *ACM SIGPLAN Notices*, vol. 49, no. 6, pp. 259–269, 2014.
[38] S. Rawat, V. Jain, A. Kumar, L. Cojocar, C. Giuffrida, and H. Bos, "VUzzer: Application-aware evolutionary fuzzing," in *Proc. NDSS*, vol. 17, 2017, pp. 1–14.
[39] P. Chen and C. Hao, "Angora: Efficient fuzzing by principled search," in *Proc. IEEE Symp. Secur. Privacy*, May 2018, pp. 711–725.
[40] F. Bellard, "QEMU, a fast and portable dynamic translator," in *Proc. USENIX Annu. Tech. Conf. (FREENIX Track)*, vol. 41, 2005, p. 96.
[41] D. Song, D. Brumley, H. Yin, J. Caballero, I. Jager, M. G. Kang, Z. Liang, J. Newsome, P. Poosankam, and P. Saxena, "BitBlaze: A new approach to computer security via binary analysis," in *Proc. Int. Conf. Inf. Syst. Secur.* Berlin, Germany: Springer, 2008.
[42] B. Dolan-Gavitt, J. Hodosh, P. Hulin, T. Leek, and R. Whelan, "Repeatable reverse engineering with PANDA," in *Proc. 5th Program Protection Reverse Eng. Workshop*, Los Angeles, CA, USA, Dec. 2015, pp. 1–4.
[43] S. Lecomte, "Élaboration d'une représentation intermédiaire pour l'exécution concolique et le marquage de données sous windows," APA, Washington, DC, USA, Tech. Rep., 2014.
[44] B. S. Pak, "Hybrid fuzz testing: Discovering software bugs via fuzzing and symbolic execution," School Comput. Sci., Carnegie Mellon Univ., Pittsburgh, PA, USA, Tech. Rep. CMU-CS-12-116, 2012.
[45] I. Yun, S. Lee, M. Xu, Y. Jang, and T. Kim, "QSYM: A practical concolic execution engine tailored for hybrid fuzzing," in *Proc. 27th USENIX Secur. Symp. (USENIX Security 18)*, 2018, pp. 745–761.
[46] H. Zhang, I. Goodfellow, D. Metaxas, and A. Odena, "Self-attention generative adversarial networks," 2018, *arXiv:1805.08318*. [Online]. Available: https://arxiv.org/abs/1805.08318
[47] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. C. Courville, and Y. Bengio, "Generative adversarial nets," in *Proc. NIPS*, 2014, pp. 2672–2680.
[48] X. Wang, R. Girshick, A. Gupta, and K. He, "Non-local neural networks," in *Proc. CVPR*, Jun. 2018, pp. 7794–7803.
[49] J. Chung, C. Gulcehre, K. H. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," 2014, *arXiv:1412.3555*. [Online]. Available: https://arxiv.org/abs/1412.3555
[50] A. Shrikumar, P. Greenside, and A. Kundaje, "Learning important features through propagating activation differences," in *Proc. 34th Int. Conf. Mach. Learn.*, vol. 70, 2017, pp. 3145–3153.



XING ZHANG received the B.Sc. and M.Sc. degrees from the National University and Defense Technology, China, in 2014 and 2016, respectively. He is currently pursuing the Ph.D. degree. His main research interests include information security, deep learning, and software vulnerabilities.



CHAO FENG received the B.Sc., M.Sc., and Ph.D. degrees from the National University and Defense Technology (NUDT), China, in 2004, 2005, and 2011, respectively. He is currently an Associate Professor with the College of Electronic Science, NUDT. His main research interests include protocol analysis, network security, and software vulnerabilities.



RUNHAO LI received the B.Sc. degree from the National University and Defense Technology, China, in 2017. He is currently pursuing the M.Sc. degree. His main research interests include information security and software vulnerabilities.

JING LEI received the Ph.D. degree from the National University of Defense Technology (NUDT), China, in 2003. She is currently a Professor with the College of Electronic Science, NUDT. Her main research interests include information security and information coding.

CHAOJING TANG received the Ph.D. degree from the National University of Defense Technology (NUDT), China, in 2003. He is currently a Professor with the College of Electronic Science, NUDT. His main research interests include information security, software vulnerabilities, and electromagnetic countermeasure.