Zhang et al. Cybersecurity (2018) 1:10
https://doi.org/10.1186/s42400-018-0010-y

Cybersecurity

RESEARCH Open Access

Sensitive system calls based packed malware variants detection using principal component initialized MultiLayers neural networks

Jixin Zhang[1,2], Kehuan Zhang[1*], Zheng Qin[2], Hui Yin[2] and Qixin Wu[2]

Abstract

Malware detection has become mission sensitive as its threats spread from computer systems to Internet of things systems. Modern malware variants are generally equipped with sophisticated packers, which allow them bypass modern machine learning based detection systems. To detect packed malware variants, unpacking techniques and dynamic malware analysis are the two choices. However, unpacking techniques cannot always be useful since there exist some packers such as private packers which are hard to unpack. Although dynamic malware analysis can obtain the running behaviours of executables, the unpacking behaviours of packers add noisy information to the real behaviours of executables, which has a bad affect on accuracy. To overcome these challenges, in this paper, we propose a new method which first extracts a series of system calls which is sensitive to malicious behaviours, then use principal component analysis to extract features of these sensitive system calls, and finally adopt multi-layers neural networks to classify the features of malware variants and legitimate ones. Theoretical analysis and real-life experimental results show that our packed malware variants detection technique is comparable with the the state-of-art methods in terms of accuracy. Our approach can achieve more than 95.6\% of detection accuracy and 0.048 s of classification time cost.

Keywords: Malware variants, Multi-layers neural networks, Principal component analysis, Sensitive system calls, Sophisticated packers

Introduction

Malware is one of the major Internet security threats today, anti-detection mechanisms such as code-morphism make the malware evolved into many variants which make signatured based detection schemes perform poorly. Detecting malware variants improves signature based detection methods. In recent years, researchers focus on detecting malware variants by using machine learning methods, which transform the malware variants detection problem to a program similarity searching problem. When a new program is sufficiently similar to any signatured malicious program in a training data set, the program is checked as a malicious program.

Since malware analysis includes two kinds of ways: static analysis and dynamic analysis. Some researches, such as (Santos et al. 2011; Cesare et al. 2014; Nataraj et al. 2011; Zhang et al. 2016a; Zhang et al. 2016b; Yang et al. 2015; Raman et al. 2012), propose to use static analysis which extracts features from binaries without actually executing programs, such as operation codes, control flow graph, etc. to detect malware variants. However, when the malware variants had already packed, it prevents further analysis from disassembly tools, synthesis tools and other static analysis tools.

Modern malware variants are always equipped with sophisticated packers such as ASPack (2017), ASProtect (2017), UPX (2017), VMProtect (2017), ZProtect (2017), etc., which allow the malware variants bypass traditional and modern detection systems. These packers include two kinds of packers: encryption packers and compression packers, which work by taking an existing application,

* Correspondence: khzhang@ie.cuhk.edu.hk
[1]Department of Information Engineering, Chinese University of Hong Kong, Hong Kong, China
Full list of author information is available at the end of the article

packing it, and then wrapping an unpacking utility around it, the unpacking utility works to unpack the inner executable in memory and transfers execution to it. The problem lies in the fact that there is nothing inherently malicious about a packer or unpacking code (Treadwell et al. 2009). When ignoring the packers, it is hard to detect if an executable is malicious due to the encryption or compression of the executable, which prevents detection systems from getting original features, especially for static analysis.

Such situation forces researches to adopt unpacking techniques or dynamic malware analysis to detect packed malware variants. However, there still exists some challenges. On one hand, some researches prefer to unpack packed programs and then detect the unpacked ones. But unpacking techniques cannot always be useful since crackers can write their private packers which are hard to be unpacked. On the other hand, another researches, such as (Zhang et al. 2016c; Huang et al. 2014; Xu et al. 2016; Kumar et al. 2012; Konrad et al. 2011; Bai et al. 2014; Santos et al. 2013), prefer to use dynamic analysis which monitors running interactions between operating system and programs in sandboxes or virtual machines to collect the features such as system calls, traffics, etc.. Although dynamic analysis can obtain running behaviours of a packed executable, the running behaviours not only include original behaviours but also include behaviours of packers of the executable which obfuscate the original behaviours. The existing methods do not take the obfuscation caused by behaviours of packers into considerations.

To overcome these challenges, in this paper, we aim to propose a novel approach which can detect packed malware variants without unpacking process. Since dynamic analysis can get running behaviours, we obtain a sequence of running system calls by monitoring system interactions in a sandbox.

Recently, there exist several related works on system call based analysis. Some of them prefer to use n-gram to represent the temporal sequential relationships of system calls and adopt classifiers to classify malicious executables and legitimate ones, such as (Konrad et al. 2011; Canzanese et al. 2015), etc.

However, to detect packed malware variants with these system calls, we have to address several challenging problems. One challenge is that the system calls of packers obfuscate the original distribution and hide the real malicious intention. In addition, as a high level representative of executables, system call is coarse-gained and sparse, which leads a bad generalization of features. What's more, this sharpens the obfuscation problem caused by packers.

Since the system calls of malware variants which are in the same families share similar distributions, and there exist a significant difference of the distributions between malware and benign (Jang et al. 2015), some system calls are used more often in malware variants. We propose to extract a series of sensitive system calls, embed their frequencies into a vector and adopt deep learning method to solve these problems. Some recent researches also used deep learning for vulnerability or malware detection, which achieve better accuracy, such as (Li et al. 2018; Kolosnjaji et al. 2016), etc. We first extract a series of system calls which is more sensitive to malicious behaviours based information entropy theory. We call these system calls as sensitive system calls which reduce a degree of obfuscation. Then we embed the system calls to a vector by using occurrence frequency. The sensitive system calls will later be sent to a neural network for training or classification. Next we prefer to use multi-layers neural networks to train a model. Finally we use the model to detect and classify malware variants.

However, since such multi-layers neural networks exist some problems such as gradient disappearance and distributed representation, it is necessary to improve the convergence ability of the neural networks to achieve better performance. We propose a principal component initialized multi-layers neural networks method to accelerate the convergence rate and to improve the accuracy rate. The principal component initialization transforms the sensitive system calls to a few new column vectors which are linear combinations of the system calls, the new column vectors are linearly independent, which can reduce the computation complexity and accelerate convergence rate.

Contributions

The main contributions of this paper are summarized as follows.

1. To reduce the obfuscation caused by packers, we extract a series system calls from unpacked instances which are more sensitive to malicious behaviours by learning with information gain, which do skip the unpacking knowledge.
2. To detect with sparse representation of sensitive system calls, we propose our principal component initialized multi-layers neural networks as an efficient and effective classifier to classify the packed malicious variants and packed legitimate ones.
3. The experimental results demonstrate that our approach 95.6% of detection accuracy and 0.048 s of classification time cost. What's more, the evaluation results show that our approach achieves very low false positive rate which means it seldom make mistake in packed benign instances detection.

Paper organizations

The remainders of this paper is organized as follows. Section "Methodology" presents our packed malware

variants detection technique. Section "Experiments" shows the experimental results and Section "Related Works" introduces the related works. Section "Limitations" and Section "Conclusions" show the limitation and conclusion.

Methodology

In this paper, we transform the packed malware variants detection problem to a system calls classification problem. To reduce the obfuscation which is caused by packers, we first extract sensitive system calls and abandon obfuscated system calls. Then we organize these sensitive system calls as a vector which will be sent to our neural net- works later. As system call is a coarse-gained and sparse representation of executables, it causes bad training approximation and feature generalization. So we next propose our principal component initialized multi-layers neural networks to efficiently and effectively train and detect malicious instance with these sparse vectors.

Our approach contains the following two phases, a training phase and a detection phase. The work flow of our approach is shown in Fig. 1, in training phase, we monitor the system interactions of executables in Cuckoo sandbox (Malwr 2018) to obtain the system calls. Each profile of executables we got from Cuckoo sandbox contains several fields: time-stamp, system call, base address, file name, executing times, etc. We only consider system calls since it can give us enough information to describe characteristics of behaviours of malware while reducing the noise and redundant. Then, based on information gain (Peng et al. 2005), a selector is used for sensitive system calls extraction which select a series of high frequency system calls in malicious executables and abandon the other system calls which are common used anywhere. The selector output a vector organized by these sensitive system calls. Finally, our principal component initialized multi-layers neural networks train these sensitive system calls and obtain parameters which will be used for classification in the detection phase. In detection phase, our neural networks are equipped with these parameters to classify packed malware variants and packed benigns.

Information gain based sensitive system calls extraction

We obtain system calls of executables by monitoring their running behaviours in Cuckoo sandbox. As modern malicious executables always are equipped with sophisticated packers, the system calls we got contain not only the system calls of originals but also the system calls of packers which obfuscate the distribution of original system calls. It limits the detection accuracy. To retain detection accuracy, in this paper, we first reduce the obfuscation from packers by extracting sensitive system calls. At the beginning, we give a definition of our sensitive system calls.

Definition 1

The sensitive system calls is a part of system calls which highly frequently act in unpacked malicious executables while not in unpacked legitimate ones.

This insight is based on an important observation that the average distribution of sensitive system calls of

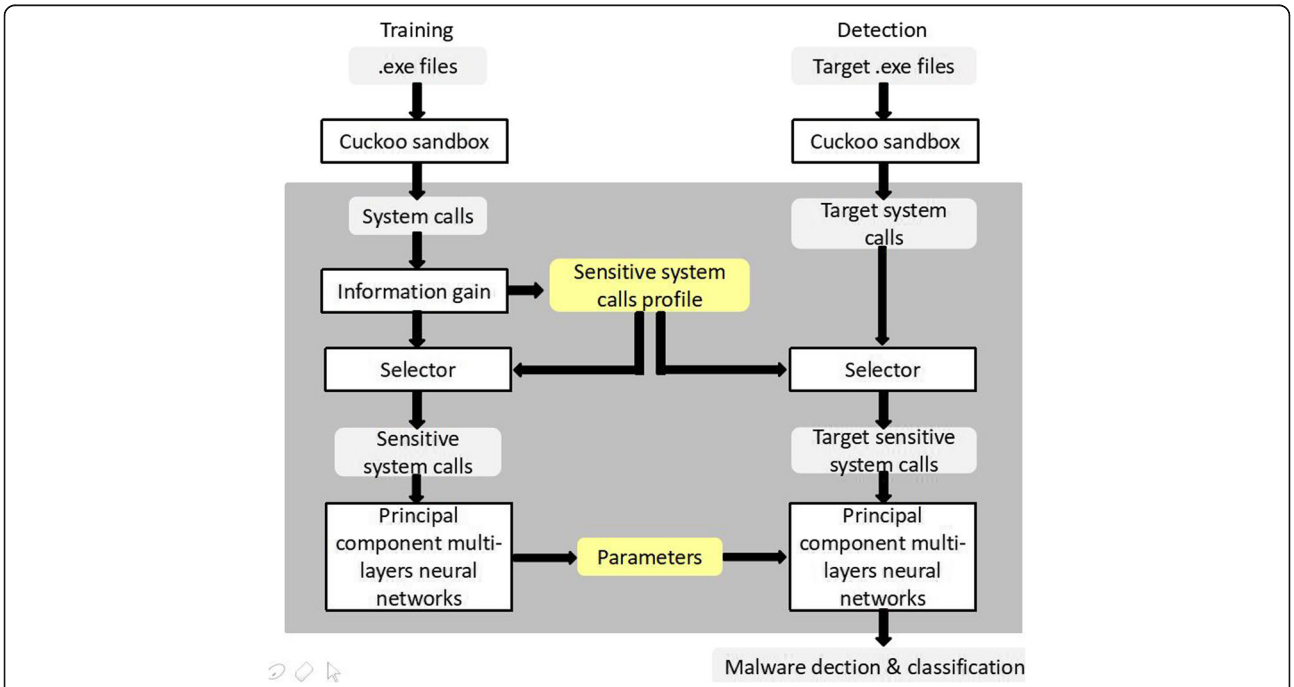

Fig. 1 The training and detection work flow

unpacked malicious executables is nearly the same as packed ones, which means that our sensitive system calls also low frequently act in packers, as a deduction of our approach. So based on this deduction, we use the sensitive system calls as representation of malicious executables.

In this paper, we use information gain which has been widely used for feature selection. Let Y be the training data sets, where y_1 is the malware data set and y_2 is the benign data set. Let S be the set of total system calls, where s_i is the i^{th} system call in S. Let X be the set of sensitive system calls extracted from S, where x_j is the jthsensitive system call in X. To extract the sensitive system calls, we use information gain gain (s_i) as the weight for each system call siaccording to Eq. (1), where p (s_i) is the probability for each s_i, p (y_1) is the probability of malware variants, p ($s_i|y_1$) is the probability for each siin y_1, and t is a constant value. The gain (s_i) is larger when the siis more relevant to malicious executables.

$$\mathrm{gain}(s_i) = p(s_i|y_1) \cdot \frac{\log(p(s_i|y_1)}{p(s_i) \cdot p(y_1)} \tag{1}$$

Let f_k be the k^{th} executable in Y, we calculate the probability p ($x_j|f_k$) for each x_j in f_k, where N (f_k) is the total count of all sensitive system calls in f_k and N ($x_j|f_k$) is the total count of x_j in f_k, according to Eq. (2).

$$p(x_j|f_k) = \frac{N(x_j|f_k)}{N(f_k)} \tag{2}$$

The p ($x_j|f_k$) as inputs will be sent to our principal component initialized multi-layers neural networks to detect malicious executables.

Principal component initialized multi-layers neural networks for malware detection

Once we have extracted the sensitive system calls, in this section, we now discuss how to detect packed malware variants by using our principal component initialized multi-layers neural networks.

As an efficient classifier, neural networks are widely used for classification in many fields such as image recognition, natural language processing, etc. In this paper, we use neural networks to classify malicious and legitimate executables. Multi-layers neural networks (Fernándezcaballero et al. 2003; Esmaily et al. 2015; Salai Selvam et al. 2011; Salcedo Parra et al. 2014) as one of deep learning methods achieve faster convergence rate and higher accuracy rate by comparing with single hidden layer neural networks, but also bring some drawbacks, such as gradient disappearance, over-fitting, etc. To overcome these drawbacks and further improve convergence rate and accuracy rate, we propose our principal component initialized multi-layers neural networks.

The architecture of our neural networks is presented in Fig. 2, which has several layers, one input layer (a sensitive system call probability vector), principal component initialized feature layer, four hidden unit layers (consider the trade-of between accuracy and time cost, we choose four hidden unit layers to improve accuracy rate while retaining training and detection time consumption) and one output layer. During the forward pass, the neural networks first uses an orthogonal transformation to convert a set of inputs into a set of features of linearly uncorrelated variables called principal components (PCA 2017). These principal components allow to quickly converge our neural networks. Each principal component initialized feature fully connect the units in the next hidden layer and each unit in the hidden layer fully connect the next layer. The output is a vector consisted by 1 and 0 which separately represents the label of malware or benign. During back propagation, the neural networks use the gradient descent method (Gradient descent 2017) to propagate the variance from the output layer to the principal component initialized feature layers and update weight matrixes of connections between two layers.

We first assign the weight matrixes a set of random values and calculate the average probability of sensitive system call S_{avg} in training datasets. For each executable, push its p ($x_j|f_k$) as inputs into the neural networks. In the principal component initialized feature layer, the networks first calculate variance vector S_{var} (f_k) according to Eq. (3). Then, the networks calculate the covariance matrix and eigenvectors. Let cvmat be the covariance matrix, according to Eq. (4), where $n_{trainingis}$ the count number of training samples.

$$S_{var}(f_k)_j = \frac{p(x_j|f_k) - S_{avg_j}}{n_{training}} \tag{3}$$

$$\mathrm{cvmat} = \frac{\sum S_{var}(f_k)^T \cdot S_{var}(f_k)}{n_{traning}} \tag{4}$$

Let eigenV be the column eigenvectors according to cvmat, where eigenV_i is the i^{th} eigenvector in eigenV order by eigenvalue a_i from maximum to minimum, according to Eq. (5).

$$|\mathrm{cvmat} - a \cdot E| = 0 \tag{5}$$

We organized top t eigenvectors (column vectors, t is 50 as our principal component initialized feature dimension number) to generate a new matrix eigenM and calculate the principal component initialized features pc_j according to Eq. (6). These features are the inputs for next hidden layers which enlarge the contrast of the average distribution between packed malicious executables and packed legitimate executables.

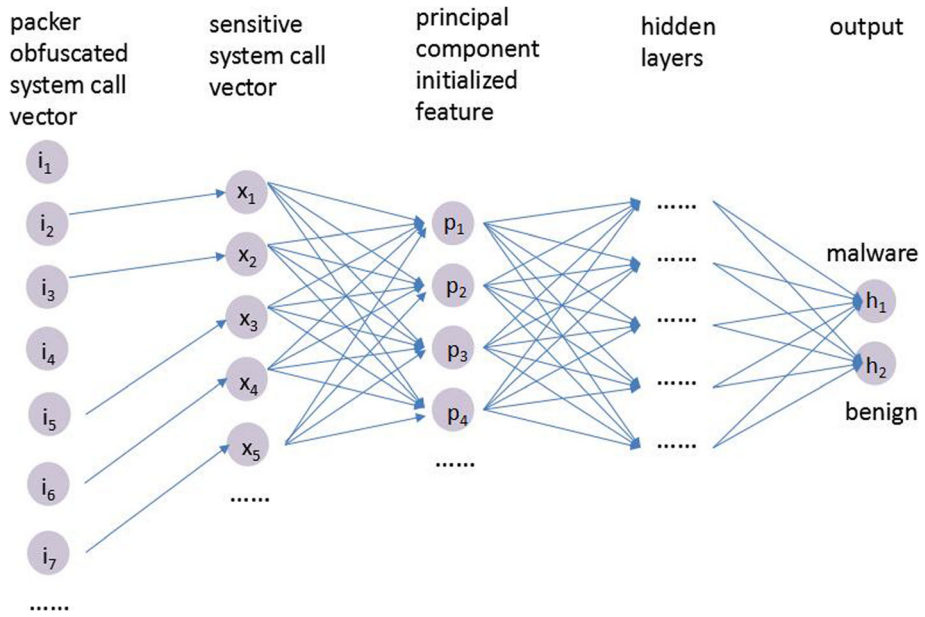

Fig. 2 The architecture of our principal component initialized multi-layers neural networks

$$pc_j = S_{va}r(f_k)_j \cdot eigenM \tag{6}$$

Let $u_i^{(1)}$ be the i^{th} unit in the first hidden layer, we calculate the $u_i^{(1)}$ according to Eq. (7), where $w_{j,i}^{(1)}$ is the weight matrix between the j^{th} principal component initialized layer and the i^{th} unit in the next hidden layer.

$$u_i^{(1)} = \frac{1}{1 + e^{-\sum pc_j \cdot w_{j,i}^{(1)}}} \tag{7}$$

Let $u_i^{(m+1)}$ be the i^{th} unit in the $(m+1)^{th}$ hidden layer, we calculate the $u_i^{(m+1)}$ according to Eq. (8), where $w_{j,i}^{(m+1)}$ is the weight matrix between the j^{th} unit in the m^{th} hidden layer and the i^{th} unit in the $(m+1)^{th}$ hidden layer.

$$u_i^{(m+1)} = \frac{1}{1 + e^{-\sum u_j^{(m)} \cdot w_{j,i}^{(m+1)}}} \tag{8}$$

Let h_i be the i^{th} unit in the output layer, we calculate the h_i according to Eq. (9), where $w_{j,i}^{(n)}$ is the weight matrix between the j^{th} unit in the nth (last) hidden layer and the i^{th} unit in the output layer.

$$h_i = \frac{1}{1 + e^{-\sum u_j^{(n)} \cdot w_{j,i}^{(n)}}} \tag{9}$$

To approach the target, the neural networks trains the inputs and corrects weight matrixes through back propagation by using gradient descent method (Gradient descent 2017). The loss function we use in our method is square loss function E (x) according to Eq. (10), where H (x) which includes a set of h_i is the output of the neural networks and V is the real value.

$$E(x) = \sum (V\text{-}H(x))^2 \tag{10}$$

We update the weight matrix $w_{j,i}^{(n+1)}$ between the last hidden layer $u_i^{(n)}$ and the output layer h_i according to the Eq.(11), where v_i is the real label value of an executable in the training set and a is a const value.

$$w_{j,i}^{(n+1)} = w_{j,i}^{(n+1)} + \alpha \cdot u_j^{(n)} \cdot h_i(1-h_i)(v_i-h_i) \tag{11}$$

Let $w_{j,i}^{(n)}$ be the weight matrix between the $(n-1)^{th}$ hidden layer and the $u_i^{(n)}$ hidden layer according to Eqs. (12 and 13), where $var.^{(n)}$ is the variance between the $(n-1)^{th}$ hidden layer and the n^{th} hidden layer.

$$w_{j,i}^{(n)} = w_{j,i}^{(n)} + \alpha \cdot u_j^{(n-1)} \cdot u_i^{(n)} \cdot \left(1-v_i^{(n)}\right) \cdot var^{(n)} \tag{12}$$

$$var^{(n)} = \sum (v_i - h_i) \cdot w_{j,i}^{(n+1)} \tag{13}$$

Let $w_{j,i}^{(m+1)}$ be the weight matrix between the m^{th} hidden layer and the $(m+1)^{th}$ hidden layer according to Eqs. (14 and 15), where $var.^{(m+1)}$ is the variance between the m^{th} hidden layer and the $(m+1)^{th}$ hidden layer.

$$w_{j,i}^{(m+1)} = w_{j,i}^{(m+1)} + \alpha \cdot u_j^{(m)} \cdot u_i^{(m+1)} \cdot \left(1-u_i^{(m+1)}\right) \cdot var^{(m+1)} \tag{14}$$

$$var^{(m+1)} = var^{(m+2)} \cdot w_{j,i}^{(m+2)} \tag{15}$$

Let $w_{j,i}^{(1)}$ be the weight matrix between the principal component initialized feature layer and the first hidden layer according to Eqs. (16 and 17), where var.(1) is the

variance between the principal component initialized layer and the first hidden layer.

$$w_{j,i}^{(1)} = w_{j,i}^{(1)} + \alpha \cdot pc_j \cdot u_i^{(1)} \cdot \left(1 - u_i^{(1)}\right) \cdot \mathrm{var}^{(1)} \quad (16)$$

$$\mathrm{var}^{(1)} = \mathrm{var}^{(2)} \cdot w_{j,i}^{(2)} \quad (17)$$

After accomplishing training phase, we can obtain a set parameters, with which the neural networks equip to classify packed malware variants. The output is a vector consisted by two confidence values, each value separately represents the probability of malware or benign. When the confidence value of malware is big enough, we deem this detection is sufficiently believable and consider the target instance as a malware variants for the next retraining. We use a copy of our neural networks to retrain and generate new parameters with which will be equipped the current neural networks. To avoid poisonous data attack from crackers, we first prepare a set of already known testing cases and then use these cases to test the retrained neural networks. We next equip the current neural networks with these retrained parameters only if the testing accuracy do not suddenly drop.

Experiments

In this section, we present several real-life experiments to show the performance of our approach. In the following, we first present the experiment setup, the data set and the cross validation of our approach. Then, we present several the state-of-art methods for comparison. In the last, we give differential analysis, convergence process analysis, accuracy evaluation and time cost evaluation of our approach.

Experiment setup, data set and validation

We implement our approach on one computer. The version of the CPU is i5–6500 @ 3.20GHz, the RAM is 16.0GB, the operation system is Windows 7. Our approach is developed by Java programming language with Jre 1.6.

To validate our approach, we use two different data sets to test our malware variant detection methodology: one is for training, the other is for detection. The training data set includes 3167 unpacked malware executables and 2894 unpacked benign executables. The detection data set includes 2083 packed malware variants and 1986 packed benign instances. We use several packers to pack the malware variants and benign instances, such as ASPack, ASProtect, UPX, VMProtect, Armadillo, ZProtect, etc.

Our malware instances are downloaded from the VxHeavens (2017) website, which includes 5 large malware families such as Backdoor, Worm, Trojan Dropper, Trojan Banker and Virus as shown in Table 1. These malware had already been labelled with their families and variant names by the website. For the benign instances, we gathered them from our personal computers as shown in Table 2.

Table 1 The malware data set

Malware family	Number
Backdoor	1045
Worm	795
Trojan Dropper	908
Trojan Banker	1522
Virus	980
total	5250

To evaluate the performance of our approach, we use k-fold cross validation in our experiments. In this way, for each group of experiments, the training data set was split into 10 groups. For each group, we randomly select 2000 unpacked malware executables and 2000 benign executables for training, and the remained 2083 packed malware variants and 1986 packed benign instances are used for classification and detection. The benchmarks we used for evaluation include classification accuracy, true positive rate (TPR), false positive rate (FPR), precision, recall, training iterations, detection time cost.

Differential analysis of distributions of system calls between malware and benign

In this subsection, we analyze the difference of system calls between malware variants and benigns to demonstrate the effectiveness of our sensitive system calls. As shown in Fig. 3, the distribution comparison of original system calls between unpacked malware and unpacked benign shows that a part of system calls high frequently act in unpacked malicious executables while not in unpacked legitimate ones, which demonstrates the intuitive of sensitive system calls.

For our extracted sensitive system calls, we use Kullback-Leibler Divergence (KLD) to metric the difference of distributions of the sensitive calls between packed malware/benign and unpacked malware/benign, according to (Kullback-Leibler divergence 2018). The KLD between packed malware variants and unpacked ones is 0.4277 and the KLD between packed benigns and unpacked ones is 0.3032. It is so small to mean that our sensitive system calls reduce the noise of original system call caused by packers. To intuitively show the

Table 2 The benign data set

Benign Source	Number
Windows System	3080
Personal Application	1800
total	4880

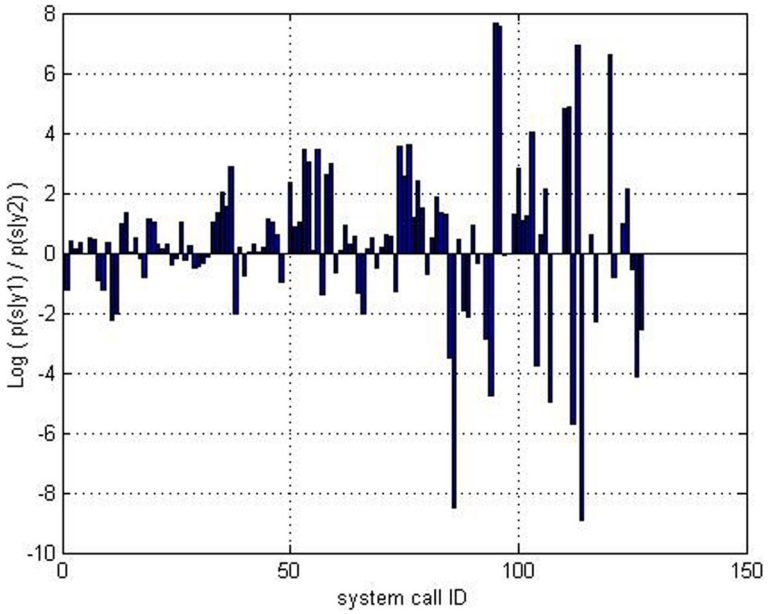

Fig. 3 The distribution comparison of system call between unpacked malware and unpacked benign

difference of distributions, we present histograms of the average distributions of sensitive system calls between packed executables and unpacked ones, as shown in Fig. 4. In this paper, we select several system calls as our sensitive system calls, such as LdrGetProcedureAddress, RegOpenKeyExW, RegQueryValueExW, RegCloseKey, FindFirstFileExW, NtDelayExecution, RegOpenKeyExA, RegQueryValueExA, RegEnumValueW, GetCursorPos, etc.

We further improve the feature by extract principal component initialized features. These features not only enlarge the average distribution contrast between unpacked malicious executables and unpacked legitimate executables, but also enlarge the average distribution contrast between packed malicious executables and packed legitimate executables, as shown in Fig. 5, by comparing with Fig. 4. The contrast is large enough to distinguish malware and benign, which demonstrates that our principal component initialized feature is an effective and optimal features for classification.

Convergence process analysis of different layers of our neural networks

In this subsection, we analyze the convergence process of our principal component initialized multi-layers neural networks and show the efficiency of the neural networks.

Since multi-layers neural networks always bring drawbacks such as gradient disappearance, over-fitting, etc., which have bad effect on convergence rate, we overcome these drawbacks and further improve convergence rate and accuracy rate, To demonstrate that our approach is efficient, we compare our approach with the other methods during convergence process, as shown in Fig. 6 and Fig. 7. In Fig. 6, it shows the convergence process of our neural networks with 1 hidden layer comparing with normal neural networks with 1 hidden layer and Fig. 7 shows the convergence process of our neural networks with 4 hidden layers comparing with normal neural networks with 4 hidden layers. From the experimental results, we can easily find out that no matter with 1 hidden layer or 4 hidden layers, our approach can significantly reduce the gap between training accuracy and detection accuracy, which means that our approach can depress over-fitting. In addition, our neural networks can rapidly converge.

For neural networks, as multi-layers can further improve convergence rate by comparing with single hidden layer, we also compare with different layers and demonstrate that multi-layers can further improve the convergence rate of our neural networks. The experimental results, as shown in Fig. 8, demonstrate that 4 hidden layers of our networks can significantly improve the convergence speed by comparing with 1 hidden layers.

When we append more hidden layers (more than 4 layers) to our networks, the detection accuracy increases slower while the detection time cost grows rapidly. So considering the trade-of between accuracy and time consumption, we choose 4 hidden layers as the best layers of our approach to improve detection accuracy rate while retaining training and detection time consumption.

State-of-art approaches for comparison

To demonstrate that our approach is efficient and effective, we compare our approach with the other state-of-art

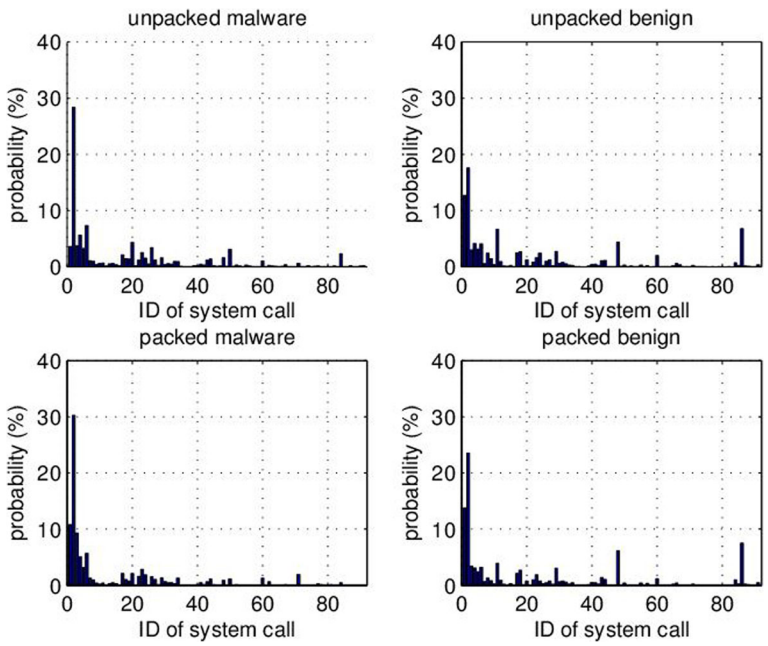

Fig. 4 The Average distribution comparisons of sensitive system calls between packed executables and unpacked ones

methods, such as (Konrad et al. 2011; Canzanese et al. 2015). Konrad Rieck et al. (2011) proposed to embedding system calls to a vector and adopt clustering and classification methods to detect malware. Raymond Canzanese et al. (2015) proposed to use a vector of system call n-gram frequencies and several classifiers such as Support Vector Machine, Logistic regression, etc. to detect malware.

Accuracy evaluation

We demonstrate that our approach is highly accurate by comparing with the other state-of-art methods in this subsection. The accuracy results are presented in Fig. 9. It shows that our neural networks with 4 hidden layers can achieve higher accuracy than the other methods. The Receiver Operating Characteristic (ROC) curves (2018) in Fig. 10 shows that our approach performs

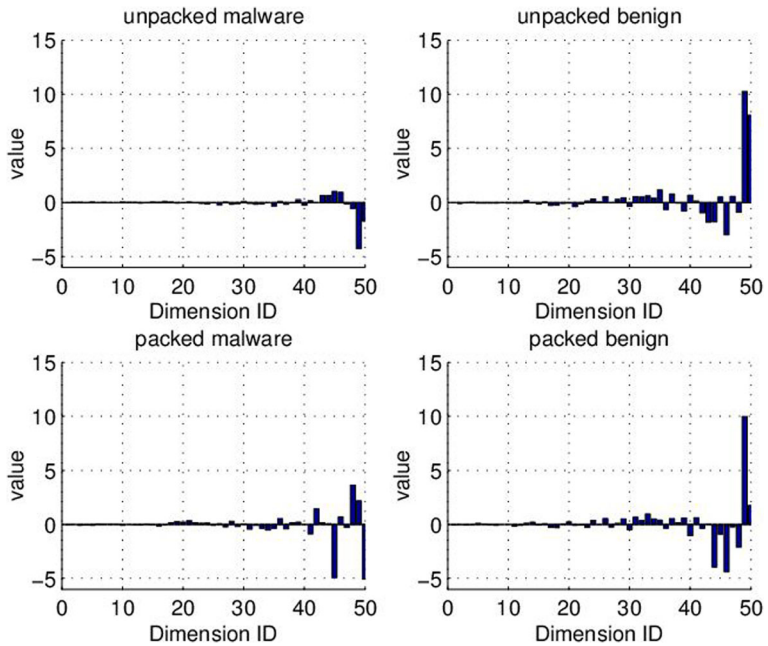

Fig. 5 The Average distribution comparisons of principal component initialized features between packed executables and unpacked ones

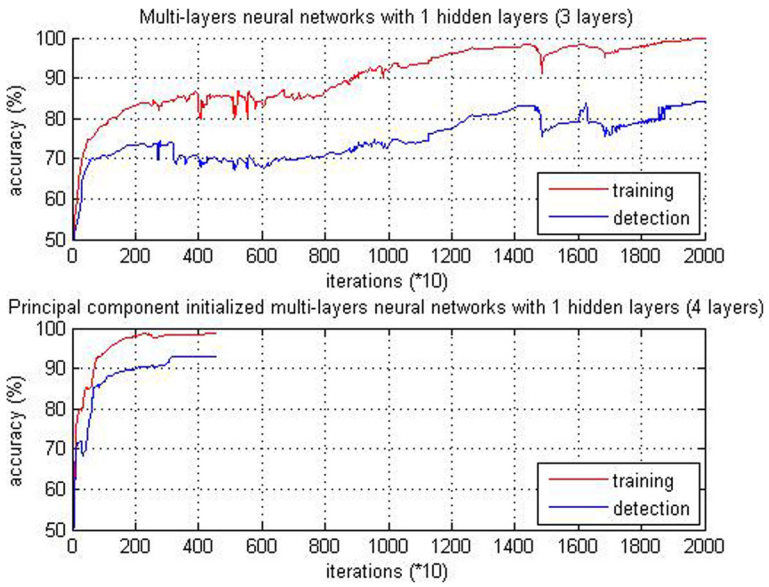

Fig. 6 The Convergence process comparison between our neural networks and normal neural networks with 1 hidden layer

outstanding in both of true positive rate and false positive rate. As a useful fact, since client users more concern on false positive rate, the false positive rate of our approach is nearly 0% while the true positive rate is more than 90%, which means that it wouldn't make mistake when detecting packed benigns.

After detecting packed malware variants, we also further classify their families by using our neural networks with the sensitive system calls. We classify them to their families in the next. The neural networks which we used for malware family classification is similar to the neural networks we used for malware detection. We only change the output layer in the architecture of our neural networks for malware detection. The output layer is a one-hot vector consisted by 1 and 0 which separately represents the label of each malware family. According to pre-labeled family-labels of malicious samples, we train a model by our networks and use it to classify malicious executables to their families after detecting of malware. As shown in Fig. 11, the results show that our approach achieves an average classification accuracy rate of 85.68% and the accuracy rates of most of families are more than 84%, which means

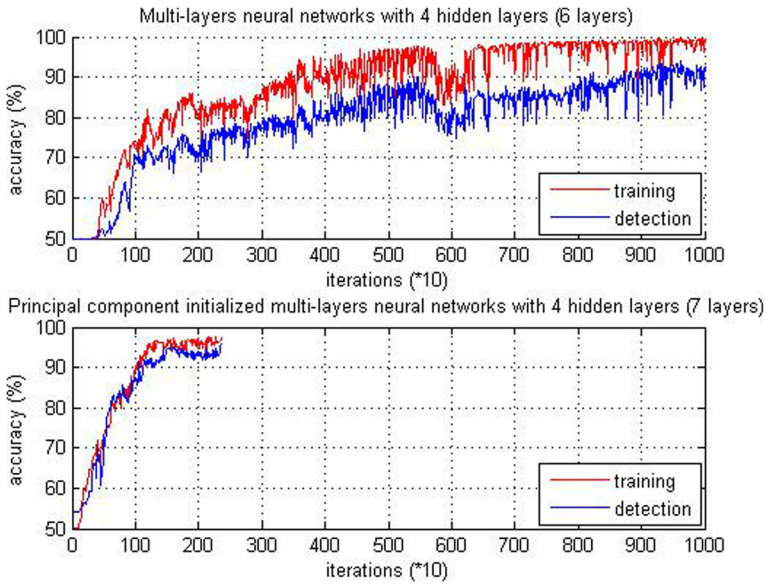

Fig. 7 The Convergence process comparison between our neural networks and normal neural networks with 4 hidden layers

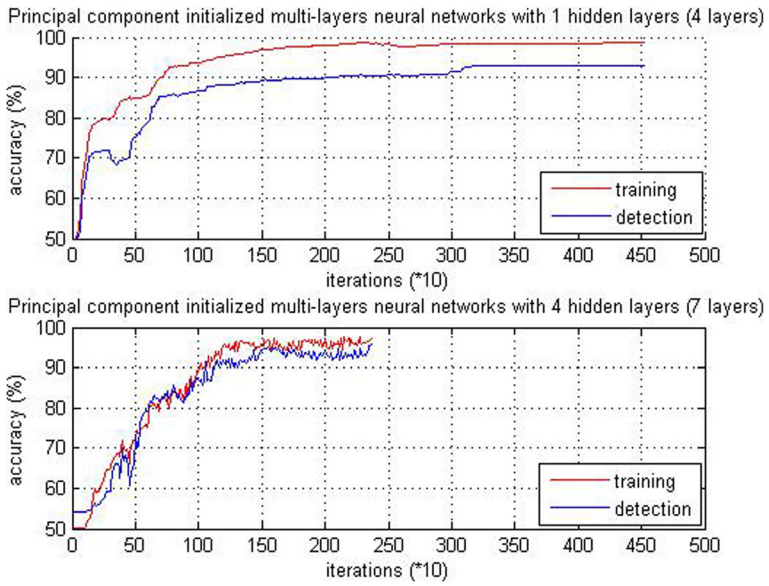

Fig. 8 The Convergence process comparison between our neural networks with 1 hidden layers and 4 hidden layers

our approach can not only precisely detect packed malicious executables, but can also classify most of them to their families.

Time cost analysis

As shown in Table 3, from the comparisons of training iterations, our neural networks with 4 hidden layers achieve less iterations because of higher convergence speed than the other layers. Although Konrad Rieck et al.' method does not need training phase, their methods need more detection time cost which is more sensitive to client users. By comparing with Raymond Canzanese et al.' approach, our approach is competitive in terms of training iterations and detection time cost because of the higher convergence speed and less feature dimensions of our approach.

From the comparisons of detection time cost, we find that our approach can significantly improve the detection speed by comparing with the other state-of-art methods. Because in detection phase, our neural

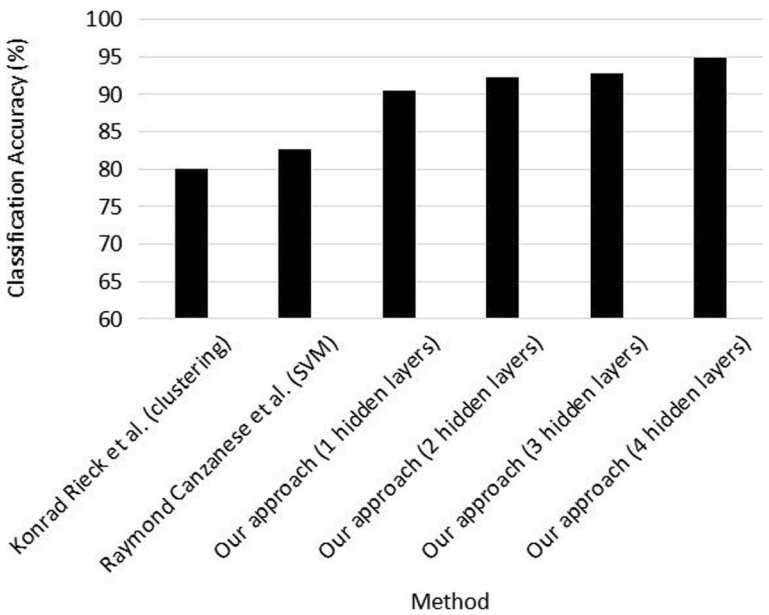

Fig. 9 The comparisons of accuracy

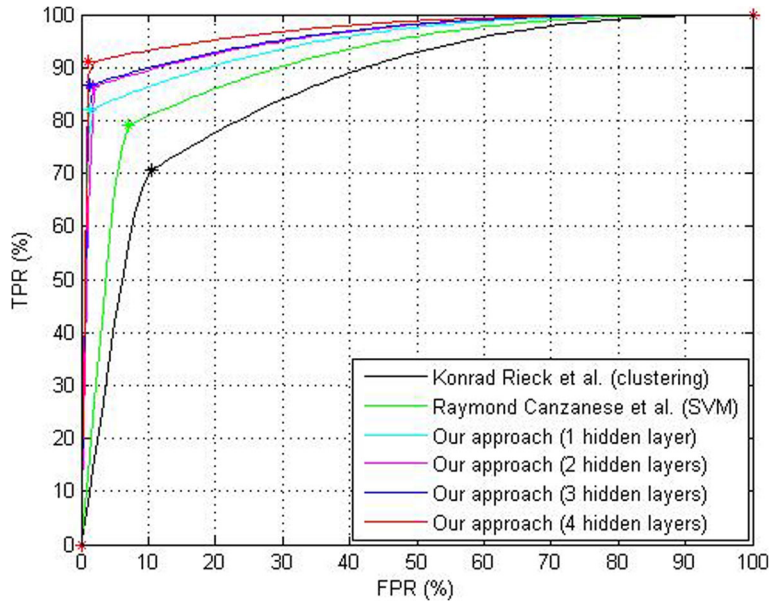

Fig. 10 The ROC curve of several methods

networks only need to forward pass the inputs with the already trained parameters which cost only a little time, while the other methods need to search similarities in a serial manner which cost a longer delay.

Related works

Malware is a pervasive problem in distributed computer and network systems today. Many approaches were proposed to detect malware by using machine learning. Some of them prefer to use static analysis, the rest prefer to use dynamic analysis. However, when facing varied packers, these methods cannot always perform well. In this section, we review some of them below.

Static analysis

I. Santos et al. (2011) proposed a data mining technique to mine the relevance of each op-code and assess the frequency of each opcode sequence, and then used Euclidean space (Euclidean Space, (2017) to measure the distance between software instances. S. Cesare et al. (2014) proposed a technique that performs similarity searching of sets of control flow graphs. L. Nataraj et al.

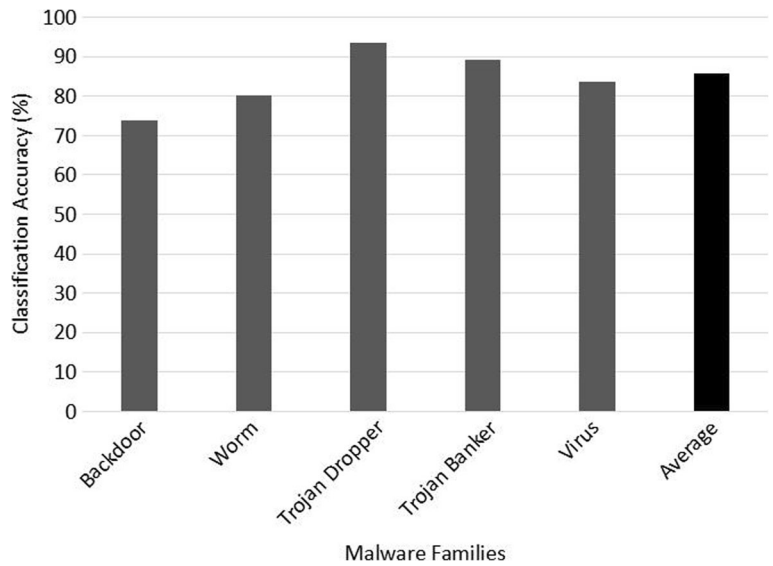

Fig. 11 The accuracy evaluation of malware family classification

Table 3 The comparisons of training iterations and detection time cost

Methods	Training iterations	Detection time cost
Konrad Rieck et al. (clustering)	0	14.793 s
Raymond Canzanese et al. (SVM)	10,000	0.094 s
Our approach (1 hidden layers)	4510	0.025 s
Our approach (2 hidden layers)	4170	0.029 s
Our approach (3 hidden layers)	3680	0.034 s
Our approach (4 hidden layers)	2360	0.048 s

(2011) proposed a method for visualizing and classifying malware binaries as gray-scale images. J. Zhang et al. (2016a; Zhang et al. 2016b) proposed to convert opcodes into 2-D matrix and used image processing method to recognize the malware executables. W. Yang et al. (2015) proposed an approach of static analysis that extracts the contexts of security-sensitive behaviors to assist app analysis in differentiating between malicious and benign behaviors. However, a number of malware authors use packing techniques to compress and encrypt the malicious codes, which make these approaches cannot work if the packers cannot be identified or unpacked.

Dynamic analysis

R. Konrad et al. (2011) proposed to automatically identifying novel classes of malware with similar sequential system calls and assigning unknown malware to these discovered classes. H. Bai et al. (2014) proposed to identify malware variants by using support vector machine with malicious behaviours which are triggered with their resulting outcomes. C. Kumar et al.'s (2012) check whether a target's system call dependency follows the same dependency of signatured malware. H. Zhang et al. (2016) proposed discoverd the underlying triggering relations of a amount of network events which detected malware activities on a host. J. Huang et al. (2014) analyzed the user interface component associated with the top level function and find the mismatch of the two to detect stealthy behaviour. L. Xu et al. (2016) implemented graph-based representation for system calls, then used the graph kernels to compute pair-wise similarities and feed these similarity measures into a support vector machine for classification. I. Santos et al. (2013) proposed a hybrid malware variant detector called OPEM, which utilizes a set of features obtained from both static and dynamic analysis of malicious code. However, these mentioned approaches do not consider packers' behaviours which obfuscate original behaviours of executables.

Packed analysis

G. Suarez-Tangil et al.'s work (2016) proposes to analyze the behavioral differences be- tween the original app and some automatically repackaged versions of it, however, when a new variant was packed by an unknown tool, their approach can no longer work because it has not analyzed the differences yet. Z. Shehu et al.'s work (2016) proposes to compute a execution fingerprint of an obfuscated app, and compare it to an available database of fingerprints of known malwares to discover possible matches, however, this matches can be easily confused by varied packers and benigns. J. Calvet et al.'s work (2012) proposes a method for identifying cryptographic functions, K. Coogan et al.'s work (2009) proposes to identify the transition points in the code where execution transitions from unpacker code to the unpacked code, P. Royal et al.'s work (2006) proposes a tool named PolyUnpacker which observes the sequences of packed or hidden code in a malware can be made self-identifying when its runtime execution is checked against its static code model. However, these unpacking techniques cannot always be helpful since not all packers can be unpacked.

Limitations

As our approach is based on deep learning method which might be attacked by adversaries, this causes another security problem. Although we design a retraining and testing process to avoid poisonous data attack from crackers and retain the detection performance (in Section "Methodology"), a persistent attack could disable a further updation of our neural networks brought retraining with new detected samples.

Conclusions

In this paper, we propose a novel approach which can detect packed malware variants without unpacking. To achieve our approach, we propose a sensitive system call based principal component initialized multi-layers neural networks, which can highly perform well in terms of classification accuracy and speed. Theoretical analysis and real-life experimental results show that our packed malware variants detection technique is comparable with the the state-of-art methods.

As a future work, besides of system calls, we will take more running behaviours such as connections, user operations, etc. into consideration to strength our detection. In addition, we will focus on protecting our malware detection system from poisonous data attack.

Acknowledgments

This work is partially supported by the National Science foundation of China under Grant No. 61772191, No. 61472131.

Funding

National Science foundation of China under Grant No. 61772191, No. 61472131.

Availability of data and materials
Our malware instances are downloaded from the VxHeavens (2017) website. I would like to declare on behalf of my co-authors that the work described was original research that none of the data and material in the paper has been published or is under consideration for publication elsewhere. All the authors listed have approved the manuscript that is enclosed.

Authors' contributions
In this work, Jixin Zhang and Kehuan Zhang conceived the paper, verified and conducted the analysis and the results. Jixin Zhang and Hui Yin designed and developed the prototype. Jixin Zhang and Qixin Wu wrote the text presented here. Hui Yin and Qixin Wu collected the data and prepared the data. Kehuan Zhang and Zheng Qin supervised the whole process. All authors provided input and approved the manuscript.

Competing interests
No conflict of interest exits in the submission of this manuscript, and manuscript is approved by all authors for publication.

Publisher's Note

Author details
[1]Department of Information Engineering, Chinese University of Hong Kong, Hong Kong, China. [2]College of Computer Science and Electronic Engineering, Hunan University, Hunan, China.

Received: 26 April 2018 Accepted: 13 August 2018
Published online: 10 September 2018

References

ASPack, http://www.aspack.com (2017)
Bai H et al (2014) Approach for malware identification using dynamic behaviour and outcome triggering. IET Inf Secur 8(2):140–151
Calvet J et al (2012) Aligot: Cryptographic Function Identification in Obfuscated Binary Programs. In: Proc. of ACM Conference on Computer and Communications Security, pp 169–182
Canzanese R. et al. (2015) System call-based detection of malicious processes. In proc. of 2015 IEEE international conference on software quality, Reliability and Security, 119–24
Cesare S et al (2014) Control flow-based malware variant detection. IEEE Trans Dependable and Secure Comput 11(4):307–317
Coogan K et al (2009) Automatic Static Unpacking of Malware Binaries. In: Proc. of Working Conference on Reverse Engineering, pp 167–176
Esmaily J et al (2015) Intrusion detection system based on Multi-Layer Perceptron Neural Networks and Decision Tree. In: Proc. of IEEE Conference on Information and Knowledge Technology, pp 1–5
Euclidean Space, https://en.wikipedia.org/wiki/Euclidean_space (2017)
Fernández-caballero A et al (2003) On motion detection through a multi-layer neural network architecture. Neural Netw 16(2):205–222
Gradient descent, https://en.wikipedia.org/wiki/Gradient_descent (2017)
Huang J et al (2014) AsDroid detecting stealthy behaviors in Android applications by user interface and program behavior contradiction. In: Proc. of ACM/IEEE International Conference on Software Engineering, pp 1036–1046
Jang J et al (2015) Mal-Netminer: Malware Classification Approach Based on Social Network Analysis of System Call Graph. In: Proc. of the 23rd international conference on World wide web companion pp 731–34.
Kolosnjaji B et al (2016) Deep Learning for Classication of Malware System Call Sequences. In: Proc. of Australasian Joint Conference on Artificial Intelligence pp 137–149
Konrad R et al (2011) Automatic analysis of malware behavior using machine learning. J Comput Secur 19:639–668
Kullback-Leibler divergence, https://en.wikipedia.org/wiki/Kullback-Leibler_divergence (2018)
Kumar C et al (2012) Obfuscated Malware Detection Using API Call Dependency. In: Proc. Of ACM International Conference on Security of Internet of Things, pp 289–300
Li Z. et al.: VulDeePecker: A Deep Learning-Based System for Vulnerability Detection. In Proc. of arXiv:1801.01681v1 [cs.CR] (2018)
Malwr, https://malwr.com/ (2018)
Nataraj L et al (2011) A Comparative Assessment of Malware Classification using Binary Texture Analysis and Dynamic Analysis. In: Proc. of ACM Workshop on Security & Artificial Intelligence, pp 21–30
PCA, https://en.wikipedia.org/wiki/Principal_component_analysis (2017)
Peng H et al (2005) Feature selection based on mutual information: criteria of max-dependency, max-relevance, and min-redundancy. IEEE Trans Pattern Anal Mach Intell 27(8):1226–1238
Raman K et al (2012) Selecting features to classify malware. In: InfoSec Southwest
Receiver Operating Characteristic, https://en.wikipedia.org/wiki/Receiver_operating_characteristic (2018)
Royal P et al (2006) PolyUnpack: Automating the Hidden-Code Extraction of Unpac Executing Malware. In: Proc. of 22nd Annual Computer Security Applications Conference, pp 289–300
Salai Selvam V et al (2011) Brain tumor detection using scalp eeg with modified Wavelet-ICA and multi layer feed forward neural network. In: Proc. of Annual International Conference of the IEEE Engineering in Medicine and Biology Society, pp 6104–6109
Salcedo Parra O et al (2014) Traffic forecasting using a multi layer perceptron model. In: Proc. of ACM symposium on QoS and security for wireless and mobile networks, pp 133–136
Santos I et al (2011) Opcode sequences as representation of executables for data mining based malware variant detection. Inf Sci 231(9):64–82
Santos I et al (2013) OPEM: A Static-Dynamic Approach for Machine Learning Base Malware Detection. In: Proc. of International Conference CISIS'12, pp 271–280
Shehu Z et al (2016) Towards the Usage of Invariant-Based App Behavioral Fingerprinting for the Detection of Obfuscated Versions of Known Malware. In: Proc. of IEEE International Conference on Next Generation Mobile Applications, Security and Technologies, pp 289–300
Suarez-Tangil G et al (2016) ALTERDROID: differential fault analysis of obfuscated smart-phone malware. IEEE Trans Mob Comput 15(4):789–802
Treadwell S et al (2009) A Heuristic Approach for Detection of Obfuscated Malware. In: Proc. of IEEE International Conference on Intelligence & Security Informatics, pp 291–299
UPX, https://upx.github.io (2017)
VMProtect, https://vmpsoft.com/products/vmprotect/ (2017)
VX Heaven, https://hypestat.com/info/vxheaven.org (2017)
Xu L et al (2016) Dynamic Android Malware Classification Using Graph-Based Representations. In: Proc. of IEEE International Conference on Cyber Security and Cloud Computing, pp 220–231
W. Yang et al. (2015) AppContext: differentiating malicious and benign mobile app behaviors using context. In: Proc. of IEEE/ACM International Conference on Software Engineering (2015), Firenze, Italy, pp 303–313
Zhang J et al (2016a) Malware Variant Detection Using Opcode Image Recognition with Small Training Sets. In: Proc. of IEEE International Conference on Computer Communication and Networks, pp 1–9
Zhang J et al (2016b) IRMD: Malware Variant Detection Using Opcode Image Recognition. In: Proc. of IEEE International Conference on Parallel and Distributed Systems, pp 1175–1180
Zhang H et al (2016c) Detection of stealthy malware activities with traffic causality and scalable triggering relation discovery. ACM Transactions on Privacy and Security 19(2):article 4
ZProtect, https://tuts4you.com/download.php?view.3017 (2017)