A Deep Malware Detection Method Based on General-Purpose Register Features

Fang Li[1,2], Chao Yan[1,2], Ziyuan Zhu[1,2](✉), and Dan Meng[1]

[1] Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China

[2] School of Cyber Security, University of Chinese Academy of Sciences, Beijing, 100049, China

{lifang, yanchao, zhuziyuan, mengdan}@iie.ac.cn

Abstract. Based on low-level features at micro-architecture level, the existing detection methods usually need a long sample length to detect malicious behaviours and can hardly identify non-signature malware, which will inevitably affect the detection efficiency and effectiveness. To solve the above problems, we propose to use the General-Purpose Registers (GPRs) as our features and design a novel deep learning model for malware detection. Specifically, each register has specific functions and changes of its content contain the action information which can be used to detect illegal behaviours. Also, we design a deep detection model, which can jointly fuse spatial and temporal correlations of GPRs for malware detection only requiring a short sample length. The proposed deep detection model can well learn discriminative characteristics from GPRs between normal and abnormal processes, and thus can also identify non-signature malware. Comprehensive experimental results show that our proposed method performs better than the state-of-art methods for malicious behaviours detection relying on low-level features.

Keywords: Malware Detection · Malicious Attack · Registers Data · Neural Networks · GPRs Data

1 Introduction

The malicious executable file is a computer program with a destructive purpose. In recent years, malware increasingly becomes advanced and sophisticated. Meanwhile, a large number of malicious codes are opens-source and some obfuscation tools are also made freely available, which causes new variants of malware burst with an exponential growth. According to AV-Test [1], the total number of malware has reached more than 740 million. Moreover, a recent security industry analyzes three years' security data from 26 countries and reports that more than half of the attacks resulted in financial damages of more than US$500,000 (including but not limited to: lost revenue, customers, opportunities, and out-of-pocket costs). However, nowadays, most of ordinary users defend against attacks using signature-based scanning tools which fail to detect Zero-day attack and are not able to detect complex emerging malware. Thus, defending the evolutional malware from enormous legitimate processes in real time is imperative for an security computer application environment.

Malware detection is usually conducted based on the high-level events (i.e., system call sequence, frequency of system call and string information). Recently, some researchers began to use low-level events at micro-architecture level for malware detection, and these low-level features have been proved more effective than high-level features to identify malicious actions [27, 9]. One reason is that the detector employing low-level features can perceive tiny changes between normal and malicious processes, which is useful for improving the malware detection performance. The other is that using low-level features takes less processing time than using high-level features. For example, if we use high-level events as detection features, the detector cannot make a decision until the relevant low-level information is converted into the high-level one. [13, 12, 26] adopted performance counters as input features for malware detection, but these methods didn't achieve a high accuracy since they directly extracted low-level features from an isolate environment interfered with the normal processes. Actually, these micro-architecture features of normal processes are noise for malware detection [27]. Although [23, 20, 24] can identify malicious behaviours with a high accuracy, they all use temporal statistics as features without considering the interference with normal processes. In reality, interference of normal processes always exists. Also, the above methods all need a long sample length (25K or 10K instructions) and cannot detect the non-signature malicious software.

To address the above problems, we make full use of the spatial and temporal properties in low-level features for malicious behaviours detection. First, we collect the data in GPRs [3] as our low-level features. Each register has specific functions, and the changes of its content can contain action information. E.g., EAX is a scratch register, and most of the Win32 Application Programming Interface (API) functions return values in this register [17]. Thus, changes in GPRs can be used to distinguish legal behaviours from malicious ones. Also, in this paper, we design a deep learning model to extract the spatial and temporal correlations in GPRs for malware detection, which can effectively suppress the noise (normal processes) in input features. Finally, selecting GPRs as features are effective for identifying non-signature malware. Most of the non-signature malware employ obfuscation techniques to change their original malicious codes for evading defense tools. However, the obfuscated malware will retain the harmful functionality of its original code, which means that it cannot alter the final data in GPRs when acting as a malicious behaviour. Even though the malware employs register reassignment technique, it does not change the relationship between the eight GPRs. Experimental results show that our method can achieve better malware detection performance compared with other detection models using low-level features at micro-architecture level.

This paper involves the following *contributions*:

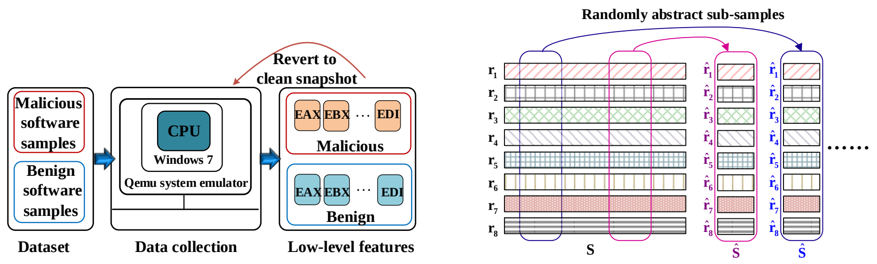
- We propose to use GPRs as features for malware detection. By jointly using the spatial and temporal properties in GPRs, our method can effectively detect malicious behaviours.
- We design a novel deep detection model (denoted as "FusionST"), which only needs a short length of GPRs (0.8K instructions, 12.5 times shorter compared than other methods) for accurate malware detection.
- By jointly fusing spatial and temporal correlations in GPRs, our model can suppress interference with normal processes, and can also learn discriminative characteristics between normal and abnormal behaviours for non-signature malware detection.

2 Related Works

There are many methods that used low-level features for malware detection. E.g., [11] pioneeringly showed that using opcode can detect malware. Using the relevance among opcodes, [25] then proposed a weighed opcode sequence frequency method to detect malicious behaviours, but it is hard to learn the true actions of a process relying on opcodes sequence. Hence, [14] further presented a control flow-based method to extract opcode behaviors. The above works based on low-level features are all static detection techniques in Operational System (OS) level, but some malicious files, such as obfuscation malware and Zero-day malware, can easily evade these techniques since they usually cannot be disassembled properly [19].

Afterwards, some researchers adopted information at micro-architecture level to dynamically identify malware. For example, [13] used performance counters to collect multi-dimensional performance counter statistics for classifying malware, but they should collect information every 25K instructions at least, which is time-consuming. Then, [26] applied a feature reduction technique to decrease the computational load. Both [13] and [26] achieve relatively low detection Accuracy, since their collected features contain a lot of noise resulting in bad detection performance with Machine Learning (ML) techniques. [17] can obtain above 95% detection accuracy, but it needs the whole action information of one sample (registers' values when important API is invoked before and after). Then, [24] and [23] proposed a hardware-supported malware detector using low-level features for malware detection. In their works, they find that neural networks can obtain good malware classification results based on the frequency of opcodes with the largest difference. However, they need to collect 10K successive instructions for only one test. Besides, they do not indicate which type of the neural networks performs the best for detecting malicious behaviours.

The methods [17, 13, 26, 23, 24] do not make fully use of the spatial and temporal properties in low-level features. Also, they do not analyze the detection performance of their methods on unknown malware. Different from the previous works, our deep learning model, with only a short sample length, extracts the spatial and temporal properties of GPRs for malware detection. Our low-level features are directly collected from the isolated environment based on Qemu



(a) Automatic data collection system (b) Illustration of sub-sample generation

Fig. 1. Sub-sample generation using automatic data collection system. (a) shows how to collect the low-level features, and (b) illustrates the generation of sub-samples.

developed in-house (shown in Section 3.1), which will inevitably introduce interference of normal processes to the collected data (close to the real environment). In other words, our method fetches malware features with a large amount of noise, which makes our detection task more difficult. Our designed deep learning model can effectively suppressing the noise by jointly fusing the spatial and temporal correlations in GPRs, and it can also identify non-signature malware.

3 Proposed Method

In this section, we introduce our deep malware detection model based on the spatial and temporal properties of low-level features (GPRs). Here, the spatial correlation means the relations of the eight GPRs. A malware will inevitably revises the data in GPRs, and the changes of the data may follow a particular way (i.e., correlations) which is different from that of the normal process. This difference is very helpful for malware detection.

We proceed in three parts: *low-level data collection*, *sub-sample generation* and *malware detection model*. The first part is used for collecting low-level features from CPU in an isolated environment. The *sub-sample generation* part introduces how to pre-process the GPRs features collected using the *low-level data collection* module, and the last part explains the architecture of the deep malware detection model.

3.1 Low-level data collection

To train the deep malware detection model (FusionST) we create an automatic data collection system to collect features of low-level events (see Fig. 1(a)). Though some systems have been proposed for collecting behaviour information in recent years, they have some restrictions. For example, the techniques in [22] and [21] need to be manually set up and operated. Besides, most of them major in extracting opcode information. Here, we designed an automatic data collection system that can extract GPRs' content effectively.

We first establish a 32-bit Windows 7 operation system using virtual machine. Then, we use a system-wide emulator Qemu [10] to emulate a standard computer environment on the operation system. Here, Qemu is an open-source emulator based on dynamic binary translation and we can get the information we need by modifying the corresponding source code of Qemu. To support malicious software operations, we disable the firework and Windows Security Services on this Qemu and connected it to the network. In this isolated environment, we collect low-level information once an instruction is executed. For each executable file, we keep executing it two minutes to collect abundant behaviour information for each sample [16]. Note that collecting data of two minutes is only a necessary process for training a classification model. When performing online malware detection, we just need to run the pre-trained detection model using sub-samples with a very short length, which means the time of data collection can be ignored.

3.2 Sub-sample Generation

Based on the automatic data collection system, we can simultaneously extract multiple low-level features, such as opcode features, branch features and register features. [23, 24] proved the opcode features (existence of opcodes and frequency of opcodes with largest difference) are effective for malicious behaviour detection, but they only use the temporal statistics as input features. In this paper, we use the spatial and temporal properties of GPRs for malware detection. In the experimental part, we give comparisons of the malware detection performance using different types of low-level features.

In the preliminary experiment, we found that only using one single register can not obtain a good detection result. Hence, we utilize eight GPRs as features to train our deep malware detection model. In this section, we introduce how to transform the eight GPRs into numeric samples which can be directly input into the deep malware model for classification (see Fig. 2).

First, we combine the eight GPRs of one sample in a fixed order: EAX, EBX, ECX, EDX, EBP, ESP, ESI, EDI. As shown in Fig. 1(b), the combined data can be written as:

$$\mathrm{S} = [\mathrm{r}_1, \mathrm{r}_2, \ldots, \mathrm{r}_8]\,, \tag{1}$$

where S is a two-dimensional matrix with height of 8, and it presents a combination of the above 8 GPRs information for one sample file. The notation r_i ($\mathrm{i} \in \{1, \ldots, 8\}$) is a one-dimensional vector, representing one of the 8 GPRs. Since S contains two minutes' information of a malicious or normal behaviour, it is not viable to use the whole S for real time detection. Then, we thus generate sub-samples from S to train the deep malware detection model. Specifically, a sub-sample is generated by randomly cutting the combined S with a fixed length, and *we call this fixed length as* ***sample length*** *in this paper*:

$$\hat{\mathrm{S}} = [\hat{\mathrm{r}}_1, \hat{\mathrm{r}}_2, \ldots, \hat{\mathrm{r}}_8]\,, \tag{2}$$

where $\hat{\mathrm{S}}$ is a sub-sample from S, and $\hat{\mathrm{r}}_\mathrm{i}$ represents a randomly intercepted piece of one register with a certain sub-sample length. Therefore,

$$\mathbf{L} = \hat{\mathrm{S}}_1, \cdots, \hat{\mathrm{S}}_\mathrm{N}, \mathrm{N} \in \mathrm{R} \tag{3}$$

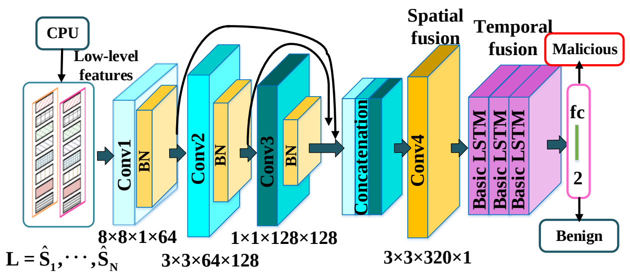


Fig. 2. Architecture of the detection model FusionST. The input of the model are the sub-samples collected from GPRs. The following CNNs and LSTMs fuse the spatial information and temporal information respectively for the final classification (output).

represent all the training samples to train the deep learning model, where N indicates the number of sub-samples (see Fig. 2). Through our experiments, we find that the cut point does not affect the malware detection performance, which implicity proves the robustness of our method.

3.3 Malware Detection Model

The generated sub-sample has both spatial and temporal correlations. Based on this property, we then propose a novel deep neural network by jointly using convolutional neural networks (CNNs) and Long Short-Term Memorys (LSTMs), to spot illegal behaviours with only a short sample length.

The architecture of the proposed FusionST model is shown in Fig. 2. The input of the model is a series of sub-samples with a particular sample length. The first four CNNs abstract the spatial correlations among GPRs by fusing information from different layers, and then the following LSTM blocks encode the sequential information (temporal correlations) for the final malware detection. Here, to well learn the correlations among the eight GPRs, Conv1 adopts 8×8 filter size with 64 filters. The filters setup for Conv2 and Conv3 are $3 \times 3 \times 64 \times 128$ and $1 \times 1 \times 128 \times 128$, respectively. As shown in Fig. 2, Conv4 is a spatial fusion layer with one-dimensional convolution ($3 \times 3 \times 320 \times 1$) to incorporate information from different receptive fields, which can improve the effectiveness of the detection model. To keep the feature map from each layer in the same scale, each convolutional layer is accompanied by bach normalization (momentum of 0.99, epsilon of 0.001) followed by a rectified linear unit (ReLU) activation.

The three LSTM blocks all have 32 hidden units to extract the time feature for detecting malicious action. LSTM module employs gate mechanism to capture long-term timing dependencies, which can learn feature from time series data effectively. However, LSTM will expend a large amount of computation and hardly get effective weight update if the length of the time dimension sequence excess 0.2K [28]. In this paper, we select 0.1K as time sequence length for FusionST model.

The categorical cross entropy, together with the weight decay, is used as our loss function

$$\ell = -\frac{1}{B}\sum_{j=1}^{B}\sum_{k=1}^{K} w_k \cdot p_{jk} \log \hat{p}_{jk} + \lambda \|\theta|_2^2, \tag{4}$$

where B is the number of batch size, K is the number of classes (2 classes for the detection task), $p_{\cdot k}$ is the true probability of class k and $\hat{p}_{\cdot k}$ is the predicted probability of class k. The weight w_k is used to balance class k in the training data set, and we set it to 1 for all the classes since our training data set is balanced. The weight decay term $\|\theta|_2^2$ smoonths the parameters θ in network *FusionST* to prevent overfitting, and λ is the corresponding regularization weight [18].

4 Experiments

4.1 Experimental Setup

We implement the FusionST model using the Tensorflow [7] framework. The standard stochastic gradient descent with momentum is employed for training, where the initial learning rate, momentum and weight decay are set to 10^{-4}, 0.99 and 10^{-3}. The network converges after approximately 40K iterations for training. All experiments are conducted in the environment: 64 Bit Ubuntu 14.04 on an Intel(R) Core (TM) i7-7800X Processor (3.50GHz) with 16GB of RAM. We run the networks on an Nvidia GeForce GTX 1080 Ti graphics card (GPU) and the Nvidia CUDA 8 software platform is used to accelerate the training process.

4.2 Dataset

VxHeaven [6] and VirusShare [4] are two popular datasets used for malware detection. In this paper, we randomly downloaded 1508 executables both from VxHeaven (60%) and VirusShare (40%) to make our malicious samples diverse. In contrast to [23, 24], we have a larger malware dataset for evaluation (1508 malwares vs 1087 malwares). Besides, note that the executables downloaded from the VirusShare are newly released in 2017, which makes malware detection more challenging. For benign samples, they include a various of legal processes, e.g. native utilities and application executables in an operation system. These benign applications are download from the SourceForge [2] that is a popular free software source. All these malicious and benign samples are invoked by our automatic data collection system to extract the low-level features.

The collected samples are randomly divided into training set and test set as shown in Table 1. Since each program is executed for two minutes in the automatic data collection environment, it will generate a large amount of data information. Hence, we generated sub-samples to train our malware detection model, which has been introduced in section: *Sub-sample Generation*.

Table 1. Malicious and benign dataset

Positive	**# Training**		**# Test**	
	Sample	Sub-Sample	Sample	Sub-Sample
Backdoor	376	50K	95	5K
Worm	187	50K	47	5K
Virus	209	50K	53	5K
Rootkit	278	50K	70	5K
Trojan	154	50K	39	5K
Total	1204	250K	304	25K
Negative	770	250K	131	25K

Table 2. Evaluation of sample lengths

Length	Accuracy	Precision	Recall	F1	FPr
0.1K	0.838	0.832	0.841	0.836	0.168
0.2K	0.888	0.883	0.890	0.886	0.117
0.4K	0.919	0.935	0.936	0.935	0.064
0.6K	0.949	0.956	0.957	0.956	0.043
0.8K	0.958	**0.962**	**0.964**	**0.963**	**0.038**
1.0K	**0.963**	0.955	0.958	0.956	0.045
1.2K	0.929	0.907	0.953	0.929	0.094
1.4K	0.922	0.961	0.964	0.962	0.064

4.3 Evaluation Metrics

In this paper, we set the labels of malicious sample and benign sample as positive and negative, respectively (see Table 1). The following are the four comprehensive metrics:

$$\begin{aligned}
&\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}},\\
&\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}},\ \text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \qquad (5)\\
&\text{F1} = \frac{2 * \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}},\ \text{FPr} = \frac{\text{FP}}{\text{FP} + \text{TN}},
\end{aligned}$$

where TP, TN, FP and FN represent True Positive, True Negative, False Positive and False Negative, respectively.

4.4 Experimental Results and Analysis

Evaluation of sample length Table 2 shows the effect of sample length on the detection performance of the FusionST model using GPRs as features. Here, we train 8 detection models using FusionST with the different sample lengths: 0.1K, 0.2K, 0.4K, 0.6K, 0.8K, 0.1K, 1.2K, 1.4K. A short sample length leads to a low Accuracy which will increase significantly with sample length increasing. Since illegal behaviors are more likely to be missed with larger sample length, FP_r value will grow when the sample length become too long, which is consistent with the conclusion in [23, 24]. Thus, we finally choose sample length 0.8K (namely 0.8K instructions) for malware detection to balance efficiency (sample length) and effectiveness (FP_r).

Evaluation of low-level features Table 3 shows the comparison results using three types of low-level features: *the existence of opcodes*, *the largest difference frequencies of opcodes* and *GPRs*. Here, we just compare our proposed features with opcode features proposed by [24, 23], since their papers have proved that the opcode features can achieve better results than those of other various of low-level features, such as memory address distance, direction categories and branch

Table 3. Comparisons with other low-level malware detection methods

Methods	[23]		[24]		Proposed work
	Opcodes 10K	Opcodes 0.8K	Frequency of opcodes 10K	Frequency of opcodes 0.8K	Registers 0.8K
Accuracy	0.909	0.762	0.813	0.649	**0.953**
Precision	0.877	0.735	0.828	0.601	**0.933**
Recall	0.956	0.824	0.798	0.877	**0.976**
F1	0.914	0.777	0.813	0.713	**0.955**
FPr	0.139	0.299	0.197	0.399	**0.068**
Kappa	0.818	0.524	0.626	0.300	**0.906**
G-mean	0.907	0.763	0.813	0.609	**0.953**
BAC	0.908	0.763	0.813	0.651	**0.953**

categories. In order to ensure the fairness of the comparison, we extract these three features concurrently in the isolation environment. Furthermore, each of their samples are collected at the same moment. According to the suggestion in [23, 24] (they obtain the best classification results when sample length is 10K), we train two models on opcode features using different sample lengths (10K and 0.8K), while our model are only trained on the registers' features of 0.8K instruction length. As shown in Table 3, though [23, 24] can effectively identify malicious behaviours using opcode features in their experimental environment, it fails to obtain the same excellent results in our experiments. E.g., the Accuracy is only 0.813 with the largest difference frequencies of 10K opcodes, while the Accuracy with the existence of opcodes is 0.909 with the 0.139 FPr when sample length is 10K. The reason may be that the opcode features contain noise when collected using our isolation environment. Our features with GPRs can achieve the Accuracy of 0.953 with the 0.068 FPr when the sample length is 0.8K. Since this experiment uses imbalanced data set, we also evaluate our method relying on Kappa, G-mean and BAC (Balanced Accuracy) Metrics [15]. Overall, our proposed GPRs perform better than opcodes features in terms of Accuracy, Precision, Recall, F1, FP_r, Kappa, G-mean and BAC.

Evaluation of different models To evaluate the effectiveness of our proposed model, we compare FusionST with other popular classification models using the Receiver-Operating Characteristics (ROC) curves. As shown in Fig. 3, the detection algorithms can be generally categorized into Machine Learning (ML) ones and Neural Network (NN) ones. Here, the ML algorithms contain the basic linear classification algorithm: Support Vector Machine (SVM) (linear kernel) and the non-linear classification algorithms: K-NearestNeighbor (KNN), Naive Bayes (NB) and Decision Tree (DT). From Fig. 3, the ML approaches can not differentiate malware from normal processes in the low-level space effectively. E.g., for ML algorithms, the highest detection Accuracy is 0.796 achieved by DB method while the linear SVM method hardly has the classification ability using GPRs as input features. The main reason may be that GPRs are micro-architecture level features, and they cannot directly reflect the action information before being converted into high dimensional spaces. Hence, the simple ML algorithms

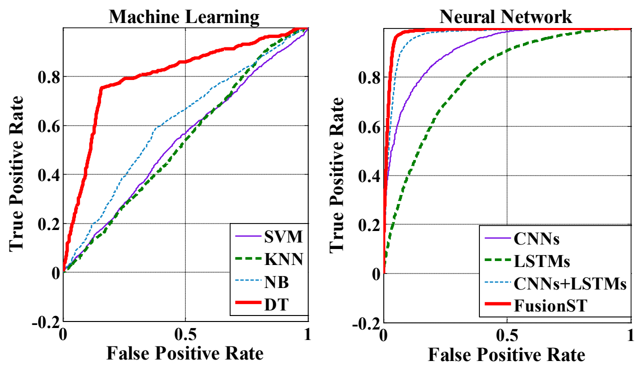


Fig. 3. Detection performance of different models.

are difficult to detect malicious behaviors in this paper. For NN algorithms, we compare FusionST with its three variants: CNNs, LSTMs, CNNs+LSTMs. The CNNs model only has the first three convolutional layers of FusionST, while L-STMs only keeps the last three LSTM blocks of FusionST. The CNNs+LSTMs is also a combination of CNNs and LSTMs but removes the fusion layer Conv4 in FusionST (Fig. 2). As seen from the results, FusionST achieves the best detection performance, and CNNs+LSTMs is a little inferior. The reason may be that the CNNs+LSTMs model is a simple combination of CNNs and LSTMs without the fusion layer (Conv4 in Fig. 2) which is crucial to fuse multi-scale features from different layers. Meanwhile, we also find that the CNNs model performs better than LSTMs, which implies that CNNs working as filters can suppress noise in the raw GPRs. These above results also show that the combination of CNNs (spatial fusion) and LSTMs (temporal fusion) is necessary, since GPRs have both spatial and temporal correlations.

Table 4. Non-signature test sets

Non-signature set	# Positive set		# Negative set
	Download set	Generate set	
Sample	150	150	150
Sub-sample	5K	5K	5K

Evaluation of detecting non-signature malware In this paper, a non-signature malware means that it does not belong to any malware families reported in Table 1. To test how the FusionST model performs when confronting with a completely new malware, we collected another two non-signature test sets for evaluation (see Table 4). The downloaded non-signature malware (positive samples) were newly released in 2018 by VirusShare. Also, we generated 150 unknown malware using Virus Maker Pack Ultimate Collection 2017 as another

Table 5. Evaluation of detecting non-signature malware

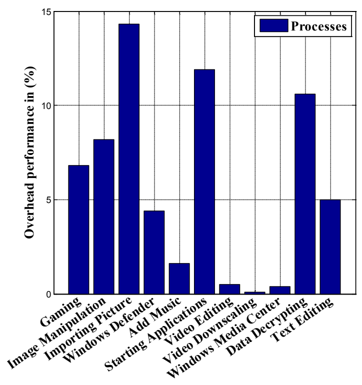
	Method	FusionST	AVware	ESET-NOD32	Kaspersky
Downloaded set	Accuracy	0.921	0.975	0.925	0.975
	Precision	0.881	1.00	1.00	1.00
	Recall	0.977	0.952	0.869	0.952
	F1	0.926	0.975	0.930	0.975
	FPr	0.132	0	0	0
Generated set	Accuracy	0.940	0.475	0.425	0.475
	Precision	0.917	0	0	0
	Recall	0.969	0	0	0
	F1	0.942	0	0	0
	FPr	0.087	0.513	0.541	0.513

non-signature positive test set. This tool set contains multiple malware makers, such as DarkHorse, Posion and Necro. Then, we download another 150 benign samples from the SourceForge malware as the negative test set. Using the two new test sets, we compare the FusionST model with other three popular anti-virus softwares (AVware, ESET-NOD32 and Kaspersky) (see Table 5). From Table 5, we can find that three anti-virus softwares achieve good malware detection performance for the downloaded test set which has been signatured for these three anti-virus softwares. However, three anti-virus softwares hardly identify the generated test samples, since the generated samples are Zero-day malware for these three anti-virus softwares. We also test these generated executable files on the VirusTotal [5] website. Each unknown malware is detected using 65 latest anti-virus engines. The report results indicate that none of anti-virus engines believe these new generated files are malicious. However, these softwares are all malicious. The FusionST model obtains satisfactory detection performance on the two non-signature test sets. E.g., the FusionST can achieve Accuracy of 0.940 with 0.087 FPr on the generated test set. In conclusion, the FusionST model has a good adaptive ability for identifying non-signature malware.

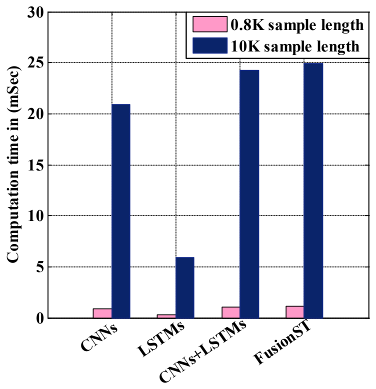
4.5 Overhead Evaluation

In this paper, we use PCMark 7 [8] to measure overhead performance of our malware detection model. PCMark 7 is a performance testing tool developed by Futuremark, a well-know global graphics and system test software development company.

To evaluate the overhead performance effectively, we first need to select the compared items from PCMark 7. We then iteratively test the performance of our computer five times with and without enabling our malware detection model. Finally, we compare the performance results of the two settings to compute the overhead performance of our detection model. Fig. 4(a) shows the overhead performance using our model to spot malware in real time. Overall, our model has a low overhead. Though Gaming overhead is 14.3% and Starting Application overhead is 11.9%, which takes up a little more resources than with other



(a) Overhead performance on PCMark 7



(b) Average times of different models

Fig. 4. Overhead performance and running times. (a) shows the overhead performance of FusionST model on PCMark 7, and (b) shows the times taken by our designed four detection models (CNNs, LSTMs, CNNs+LSTMs and FusionST) to detect one sample.

online malware detection methods, the rest of the test items has lower overhead performance, i.e. Video Downloading overhead is only 0.5%. Besides, as long as the detection model can be implemented in hardware, nearly zero consumption can be achieved.

Evaluation of detection times Fig. 4(b) shows the test times taken by our designed four detection models (CNNs, LSTMs, CNNs+LSTMs and FusionST) when sample length is 0.8K and 10K. As observed, we can find that sample length can significantly affect the detection efficiency. E.g., for FusionST model, using sample length 10K consumes about 23 times longer time than using sample length 0.8K. In terms of effectiveness and efficiency, we choose the detection window as 0.8K in this paper. Overall, all the four models take less than 1ms to complete one test when the sample length is 0.8K, which is important for our detection model to monitor malicious behaviours in real time. As we mentioned before, we choose the FusionST model as our basic classification model in this work. Though FusionST model needs slightly longer test time compared with other models, it performs better in terms of $\mathrm{FP_r}$ (see Fig. 3).

5 Conclusions and future work

With micro-architecture level features, the existing methods usually use performance counters, existence of opcodes or frequency of opcodes with largest difference for malware detection. Although these methods can achieve satisfactory malware detection performance, they need collecting a long length of low-level features (about 10K instructions). Meanwhile, these detection methods are prone to be affected by noise from the low-level features and can hardly detect non-signature malware. In this paper, we use GPRs as features for malware detection. Experimental results show that only using a short length of GPRs (0.8K) can achieve high Accuracy for malware detection. Using CNNs and LSTMs, we also propose a deep detection model by jointly fusing spatial and temporal correlations of GPRs to improve the effectiveness and robustness of our detection method. Our deep detection model can well learn the correlations among GPRs for normal system processes, and thus can also identify non-signature malware due to their different characteristics from benign.

Currently, our method mainly focuses on binary classification, i.e., malware detection, and it can not identify the specific types of malwares. We will extend our work to a multi-class problem by incorporating more low-level and high-level features. Meanwhile, we also tend to apply our method to other architectures, such as $X86\_64$, ARM and MIPS.

6 Acknowledgement

This work was supported by the National Science and Technology Major Project of China (2018ZX01028201).

References

1. AV−TEST — The Independent IT-Security Institute. `https://www.av-test.org/en/statistics/malware/`, accessed in March 2018
2. The Complete Open−Source and Business Software Platform. `https://SourceForge.net`, accessed in September 2018
3. Software Developer Manuals for Intel 64 and IA-32 Architectures. `https://www.intel.com/content/www/us/en/support/articles/000006715/processors.html?wapkw=developer/`, accessed in September 2018
4. VirusShare. `https://virusshare.com/`, accessed in September 2018
5. VirusTotal. `https://www.virustotal.com/en/`, accessed in September 2018
6. VX Heaven. `http://vxheaven.org/vl.php`, accessed in September 2018
7. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M.: Tensorflow: Large−scale machine learning on heterogeneous distributed systems (2015)
8. Ajay, K.M.A., Jaidhar, C.D.: Automated multi−level malware detection system based on reconstructed semantic view of executables using machine learning techniques at vmm. Future Generation Computer Systems (2017)
9. Banin, S., Dyrkolbotn, G.O.: Multinomial malware classification via low-level features. Digital Investigation **26**, S107–S117 (2018)
10. Bellard, F.: Qemu, a fast and portable dynamic translator. In: Conference on Usenix Technical Conference. pp. 41–41 (2005)
11. Bilar, D.: Opcodes as predictor for malware. International Journal of Electronic Security and Digital Forensics **1**(2), 156–168 (2007)
12. Cronin, P., Yang, C.: Lowering the barrier to online malware detection through low frequency sampling of hpcs. In: IEEE International Symposium on Hardware Oriented Security and Trust. pp. 177–180 (2018)
13. Demme, J., Maycock, M., Schmitz, J., Tang, A., Waksman, A., Sethumadhavan, S., Stolfo, S.: On the feasibility of online malware detection with performance counters. In: International Symposium on Computer Architecture. pp. 559–570 (2013)

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22744-9_17

14. Ding, Y., Dai, W., Yan, S., Zhang, Y.: Control flow−based opcode behavior analysis for malware detection. Computers and Security **44**(2), 65–74 (2014)
15. Fernández, A., García, S., Galar, M., Prati, R.C., Krawczyk, B., Herrera, F.: Performance Measures, pp. 47–61. Springer International Publishing (2018)
16. Fredrikson, M., Jha, S., Christodorescu, M., Sailer, R., Yan, X.: Synthesizing near−optimal malware specifications from suspicious behaviors. In: Security and Privacy. pp. 45–60 (2010)
17. Ghiasi, M., Sami, A., Salehi, Z.: Dynamic malware detection using registers values set analysis. In: International ISC Conference on Information Security and Cryptology. pp. 54–59 (2012)
18. Hoffer, E., Banner, R., Golan, I., Soudry, D.: Norm matters: efficient and accurate normalization schemes in deep networks (2018)
19. Islam, R., Tian, R., Versteeg, S., Versteeg, S.: Review: Classification of malware based on integrated static and dynamic features. Journal of Network and Computer Applications **36**(2), 646–656 (2013)
20. Khasawneh, K.N., Ozsoy, M., Donovick, C., Abu-Ghazaleh, N., Ponomarev, D.: Ensemble learning for low−level hardware−supported malware detection. In: International Symposium on Research in Attacks, Intrusions, and Defenses. pp. 3–25 (2015)
21. Okane, P., Sezer, S., Mclaughlin, K., Im, E.G.: Malware detection: program run length against detection rate. Iet Software **8**(8), 42–51 (2016)
22. O'Kane, P., Sezer, S., Mclaughlin, K., Im, E.G.: Svm training phase reduction using dataset feature filtering for malware detection. IEEE Transactions on Information Forensics and Security **8**(3), 500–509 (2013)
23. Ozsoy, M., Donovick, C., Gorelik, I., Abughazaleh, N., Ponomarev, D.: Malware−aware processors: A framework for efficient online malware detection. In: IEEE International Symposium on High PERFORMANCE Computer Architecture. pp. 651–661 (2015)
24. Ozsoy, M., Khasawneh, K.N., Donovick, C., Gorelik, I., Abughazaleh, N., Ponomarev, D.V.: Hardware−based malware detection using low level architectural features. IEEE Transactions on Computers **65**(11), 3332–3344 (2016)
25. Santos, I., Penya, Y.K., Devesa, J., Bringas, P.G.: N−grams−based file signatures for malware detection. In: International Conference on Enterpise Information Systems. pp. 317–320 (2009)
26. Sayadi, H., Makrani, H.M., Randive, O., D, S.M.P., Rafatirad, S., Homayoun, H.: Customized machine learning-based hardware-assisted malware detection in embedded devices. In: 2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE). pp. 1685–1688 (2018)
27. Tang, A., Sethumadhavan, S., Stolfo, S.J.: Unsupervised anomaly−based malware detection using hardware features. International Workshop on Recent Advances in Intrusion Detection **8688**, 109–129 (2014)
28. Yan, J., Qi, Y., Rao, Q.: Lstm−based hierarchical denoising network for android malware detection. Security and Communication Networks **2018**, 1–18 (2018)

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22744-9_17