DFRWS

CNN based Zero-day Malware Detection Using Small Binary Segments

Wen Qiaokun[a, *], K.P. Chow[a]

a. the University of Hong Kong, Hong Kong, China

Abstract

Malware detection is always an important task in digital forensics. With the advancement of technology, malware have become more and more polymorphic. In the process of digital investigation, forensics always cannot get the entire file of the malware. For example, when conducting corporate cybersecurity forensics, because the limit length of network packages, packets capture tools established by different companies often fail to get the entire file. Otherwise, deleting files may also cause residues of malware segments. Because we even do not know which part the segment we get is, so, we cannot use much domain knowledge to do the detection. Therefore, this paper proposes to detect malwares according to very small sequence binary fragments of PE files by using a CNN-based model. Datasets especially test set are often one of the most difficult problems in zero-day malware detection, because it means that the virus has never appeared before. In this paper, we collect the data by taking advantage of the differences in anti-virus tools at different time points. And Experiments are performed on malwares of different lengths, positions, and combinations. Through experiments, we found that only a short segment is needed to achieve a relatively good accuracy. In the end, for a random piece of continuous malicious code, we achieved an accuracy of up to 0.86 when the length of continuous fragments is 60,000 bytes. For non- contiguous and unordered random pieces of malicious code, we get an accuracy of up to 0.83 using only 1024 bytes(1KB) length fragments. And when using 60,000 bytes length fragment as the baseline, we can finally receive a 0.91 accuracy.

Keyword: Malware detection; Malware segments detection; CNN.

1. Introduction

In recent years, the development of computer and Internet technologies has led to an exponential increase in the number of malwares. The security situation of the network has also become more and more serious. Symantec's annual report shows that more than 200 million new malicious entities were discovered in 2018. In addition, the functions of malware are also gradually enriched. They can achieve malicious functions such as obtaining identity or other privacy information, extortion, currency stealing, and these even gradually forming an underground economy[1]. As technology advances and the demand for attacks diversifies, the technology used to make malware is becoming more complex as well. The requirement for malware detection is also increasingly appearing in digital forensics scenarios. Due to the special nature of digital forensics tasks, forensics often have to deal with fragmented files, which undoubtedly makes it difficult to do malware detection.

On the other hand, according to statistics, Windows still accounts for nearly 80% of the market share of many personal computer operating systems [2]. Among the malware intercepted by Symantec in 2018, malwares targeting Windows operating system accounted for 97.2% of the total, which was far higher than other operating systems[1]. Therefore, we have identified the malware studied in this article as portable executable files(PE files) for Windows systems.

Most traditional anti-virus tool uses signature as their detection method. This technique is primarily based on the idea of pattern matching. The staff manually or automatically creates a unique label for each malware, which is called signature. These signatures can include a variety of different attributes, such as file hashes, content strings or bytes. And then, the staffs create a malware signature database.

* Corresponding author. Tel.: +852-56176724 .
E-mail address: u3556283@connect.hku.hk .

When detecting, anti-virus tools will compare the signature of the unknown code with the contents of the database, if it matches, the code will be judged as malicious code. This method has the advantages of simple, convenient, fast detection speed and low false alarm rate, but the downside is that this method cannot do anything in the face of viruses that have not appeared before.

In order to detect new viruses, scientists have proposed a number of AI based methods. They hope to find out the underlying rules in all malicious code through some machine learning methods, and so that unknown viruses can be detected. However, some malwares began to adopt more sophisticated techniques, such as splitting a complete file into several parts and achieving its malicious purpose by calling each other between programs. There are also a lot of malware that launches a self-destruction program after the attack is executed[3], and it's nearly impossible to recover the whole file even though we use some recover techniques. In addition, companies generally use packet capture analysis to analyze the transmitted files which go through their systems, this sometimes results in only a partial segment of the file can be captured because of the limited length of the packets and the indefinite order of reception. This undoubtedly adds to the difficulty of digital forensics because it makes some static methods based on file header feature extraction or disassembly instruction analysis and all dynamic methods difficult to achieve its purpose. Therefore, this paper proposes a method for detecting small fragments of PE file, and it should also be able to handle fragments of various lengths.

The mainstream methods of malicious code detection generally require some domain knowledge to extract features. This part has been introduced in detail in Section 2. However, if there is only one fragment of the program, and we cannot know where the fragment is in the whole file, it is very difficult to use those kinds of methods. Deep learning is a new field in machine learning research. It mimics the mechanism of the human brain to interpret data, and is widely used in images, sounds and texts, also information security. To the lack of experience with small malware fragments features, deep learning has become a good way to solve this problem. It can combine low-level features to form more abstract high-level representation attribute categories or features to discover distributed feature representations of data. And so that we can give the model raw input directly, in malware detection area, the raw input should be binary.

2. Related works

At present, malware detection methods based on machine learning are mainly divided into two categories, static analysis and dynamic analysis. Static analysis is to analyze whether the files are malicious by analyzing the static characteristics of some files without executing the program itself. And dynamic detection is performed by actually running the virus in the sandbox and observing the characteristics of its running state.

2.1. Static Analysis

There are three basic ideas in all methods of static detection. The first one is feature based, those features are mainly based on the value of flag bits in PE- Header. The researchers analyze some of the flag bits in the header of the file, and then combine other simple features outside PE-Header but easy to find, such as entropy values, strings to form a feature set. These features are then screened by some methods, and finally trained by using some machine learning methods. In 2009, Fauzan Mirza and his group trained their model by using totally 189 features and gained over 99% accuracy on their dataset[4].

The second method is to use assembly codes. Researchers can obtain the assembly instructions of PE file through some disassembly tools[5-6]. By sorting the assembly instructions such as encoding the opcode to make the synonym distance shorter or extracting the n-gram features, the detection effect can be effectively improved. In 2018, Zeliang Kan and his group obtained over 95% accuracy on their dataset by using grouped instructions[6].

The third method is to directly throw the binary into the machine learning method for training. In past studies, scientists often used n-gram frequency statistics to get results. In recent years, with the wide use of deep learning methods based on neural networks in various fields, scientists devote to input binary directly into the network without any pre-processing. However, compared to other fields, PE files have a very long sequence, which is usually a few million or even more bytes, this become the bottleneck of this kind of method. In 2017, Edward Raff used only binary in PE-Header and gained over 90% accuracy in his dataset[7]. And in the same year, they also proposed a new network which used a very large kernel in Convolutional Neural Network (CNN) for detecting whole exe, finally they got over 98% accuracy[8].

The main advantage of the static analysis method is that malicious code does not need to be executed dynamically, it will not cause damage to the analysis system, so it is safer. In addition, this method is not subject to the specific process execution process, and

the code can be analyzed in detail. The problem is that it is difficult to deal with the increasingly complex malicious sample packing and confusing technology, and the expected effect cannot be obtained in the processing of obfuscated code. Therefore, with the continuous development of virtualization technology, researchers have begun to conduct malicious samples using dynamic analysis.

2.2. Static Analysis

Dynamic analysis Dynamic analysis uses virtual release and other mechanisms to deal with software packing and obfuscation techniques. It allows malicious samples to be fully released, and it uses the execution of core code to observe malicious behavior, which solves the problem of software packing and confusion to some extent. In 2009, C. Kolbitsch built fine-grained models that are designed to capture the behavior of malware based on system calls and reached 0.93 accuracy in total. [9].

However, with the confrontation between offensive and defensive techniques, this approach has gradually exposed its drawbacks. First, malware developers usually have a more thorough study of virtual environments and sandbox mechanisms. They always use a variety of detection and countermeasure technologies to make it difficult to perform malicious samples in virtualized and sandbox environments. Secondly, some of the malicious samples use the normal software digital signature and reuses the normal software code to disguise its behaviour, and they can even use the virtual machine technology to attack malware analysts. Finally, the dynamic analysis methods extract complex features, cost long detection time. They often consume a lot of resources, so the drawbacks of this approach are particularly prominent when applied to large-scale data. With the development of artificial intelligence and machine learning algorithms, the research environment of static analysis has gradually improved. Therefore, this article chose static methods for detecting. And since methods by extracting features and assembly instructions cannot satisfy the requirements of fragments of malware, so finally we used binary-based methods to do the training.

3. Model Architecture

In order to achieve higher accuracy, we hope that the model we design can better consider the characteristics of malware fragments: 1) Even the aim to detect is small fragments of malwares, it should have better universality and can detect fragments of any length.; 2) Since the binary of the malicious code fragment is a sparse timing sequence, the relationship of each byte should be effectively processed in our model. So, we designed a model with 12 layers (see Figure 2.).

3.1. Dataset Obtain

One of the most difficult parts of this project is the test set. This is because the malware dataset we need is zero-days, which means, they are new malwares that all the anti-virus tools cannot detect them successfully. To solve this problem, we first download some anti-virus tools in March, do snapshots for them, and never update them till now. At the same time, we collect data from VirusShare before March as the positive samples for training. After five months in August, we collect malwares between March and August from VirusShare, and put them into the snapshots of anti-virus tools which are made in March. If all of the anti-virus tool cannot detect the malware, it can be seen as a zero-day malware. And in these five months, some of the zero-day malwares are released by some hacker organization, those malwares are also collected to distribute the testing set.

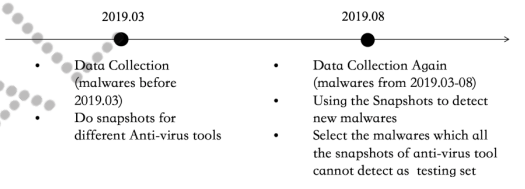

Figure 1. Method to obtain Zero-day malware

3.2. Model Selection

In general, RNN series methods is a better way to solve problems involving timing sequences. However, in ordinary time-series texts, often a word represents a meaning, and RNN can infer the meaning of the sentence through the association between different words, thereby performing works like classification and prediction. Different from natural languages, the numerical distribution of the malicious code binary is relatively sparse, often several bytes represent a word, and several words can form a complete assembly instruction. But RNN itself does not such suitable to deal with too sparse input, this is the first reason for us to give up RNN. And this can also explain the reason we added the embedding layer at the beginning of the model.

Moreover, due to the structure of the method of RNN series, the longer sequence that needs to be processed, the greater time complexity and space

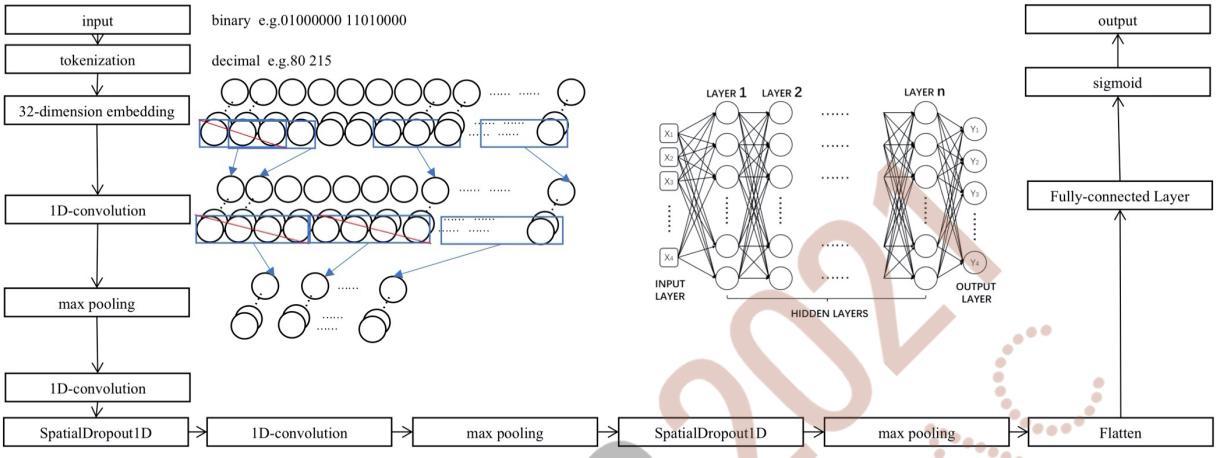

Figure 2. Architecture of the whole model

complexity are required for training. Malware binary sequences often have inputs of millions or tens of million bytes in length. Although the goal of this study is to detect small malware fragments, there are still cases where the acquired fragments with big amounts of bytes. In order to satisfy the versatility of the model and the characteristics of matching binary sparseness, we chose CNN-based method as the training model.

In addition, if the detection of longer segments is considered, because it is closer to the complete pe file, the overall characteristics of the PE files are needed to be considered. In a PE file, many parts may be able to change positions without affecting their enforceability, PE-Header stores pointers to all other contents. At the same time, the PE-Header can be located anywhere, its position is determined by the pointer at the end of the unique fixed constant MS-DOS Header. This spatial trait is not easily solved using current methods. However, the translation invariance of CNN is still a better way to solve this problem. Thus, Conv1D was chosen for training.

3.3. Parameters' Setting

According to Karen Simonyan and Andrew Zisserman's paper which raised the famous VGG structure in 2015, for a given receptive field, the use of stacked small convolution kernels is superior to the use of large convolution kernels, because multiple nonlinear layers can increase network depth[10]. This guarantee to learn more complex patterns at a lower cost (less parameters). Therefore, we used smaller kernels and a relatively larger number of layers than the previous method of training the malware binary using the deep learning method. We set the kernel size to 3 at the very beginning because it is the smallest size that captures the left, right, and center concepts. In the course of the experiment, we found that the model is more inclined to overfitting, especially when the fragment we extracted is very short. Therefore, we modified the kernel size of the last layer to 5 by doing continuous comparison experiments and added dropout. In fact, in most experiments, dropout was added to the fully connected layer. However, through experiments we found that such addition has little effect on the results. SpatialDropout is a dropout method proposed by Tompson et al. in the field of images in 2015[11]. Ordinary dropout will randomly set some elements to zero, and SpatialDropout will randomly set zero part of the area. This dropout method has proven to be effective in the field of image recognition. We tried to use this method instead of the normal dropout and added it behind the convolution layer, which effectively improved the average accuracy and stability of the model. We select the sigmoid function instead of Softmax in the last layer, because we will use the feature of the sigmoid function, its value range is between 0 and 1, to solve the detection problem of non-contiguous disordered malware fragments.

4. Model selection

4.1. Dataset

Training data with deep learning methods often requires a large number of positive and negative samples. There is not a standard dataset, this is also the main reason that different anti-virus company or institutes cannot achieve uniformity in malware detection area. In order to ensure the versatility of the experiment, this paper uses the famous malicious sample database VirusShare[12] as a positive sample. And benign samples are downloaded from Microsoft online shop, we tried to cover more kinds of benign software including games, music, social media, education, etc. In order to ensure the balance of positive and negative samples, we screened positive and negative samples to make their size divisions consistent. In the end, we got 5214 malwares as positive samples and 5211 benign files as negative samples, the average size of all the samples are 769.47KB. We then labelled all malwares as 1 and benign wares as 0.

4.2. Experiment Process

All experimental steps are shown in the following flow chart (See Figure 3).

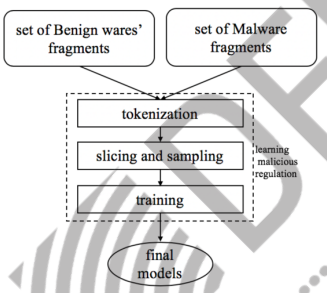

Figure 3. Flow gram of whole experiment process

Firstly, the binary of all samples should be obtained. Next, we sample it in different ways, two different types of sampling methods are implemented for our research. The first one is to continuously sample and explore the relationship between detecting result and length or position of the continuous fragments. The other is to randomly sample the PE files to explore the effect of non-continuous sample length and sampling order on the detecting results. By analyzing the experimental results, we try to obtain some models for different lengths. The result and their analysis can be found in Section 5.

4.3. Sampling Methods

For continuous sampling. The aim of this part is to simulate the situation that we can only obtain a continuous part of a PE file, for example, only a part of the of the file is broken. For training, we need to randomly extract different part of a PE file for each length we want to compare (see Figure 4). Suppose the white rectangle represents the entire PE file, the black part is the part we need to sample.

Since we want to have as many fragments as possible in our experiments, we hope that for some long files, we can sample for multiple times without putting them back. Also, in order to ensure the randomness of the sampling, when we need to sample multiple times in a file, we hope that the length of the unsampled portion of the file to be trained is much larger than our target length. Therefore, the number of times for sampling for each file should satisfy formula (1). Suppose $file\_len$ represents the length of the file, $frag\_len$ represents the length of each fragment that we want for the training, and $slice\_num$ be the final number of fragments we obtain from a file. Then,

$$slice\_num = [file\_num/frag\_len/k] \quad (1)$$

where k is a constant. To make the sample random enough for our experiments, we set k a large value, which equals to 50. After sampling, we shuffled all the fragments to form our final dataset.
In addition, we also try to explore the effect of the position of consecutive segments on the experimental results. So, we extract fragments from front and end, which is shown in Figure 5.

For un-continuous sampling. There is also situation that we cannot get such a long continuous fragment, but several small fragments from one PE file (see Figure 6). Assuming that the white rectangle represents the entire PE file, the black parts represent a small number of randomly extracted small locations in the entire PE file. We have designed two scenarios, the first one is ordered and the second one is unordered. An ordered scenario means that after sampling each PE file, the resulting sample order is consistent with the relative order of the fragments in the original file. In converse, the disordered scene will disturb the order of the samples. The reason for this is that we want to simulate the situation that staffs get the file fragment by capturing the packet, the order may be unsure by them.

Actually, we final chose the small fragment as 1024 bytes for experiments. And we de- duplicate the data used in this part and the data used in continuous

part to avoid the test results of this part be affected. The reason for doing this and using 1024 bytes as a baseline will be shown in Section 5.

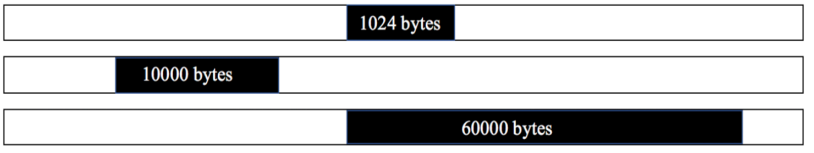

Figure 4. Schematic diagram of continuous sampling

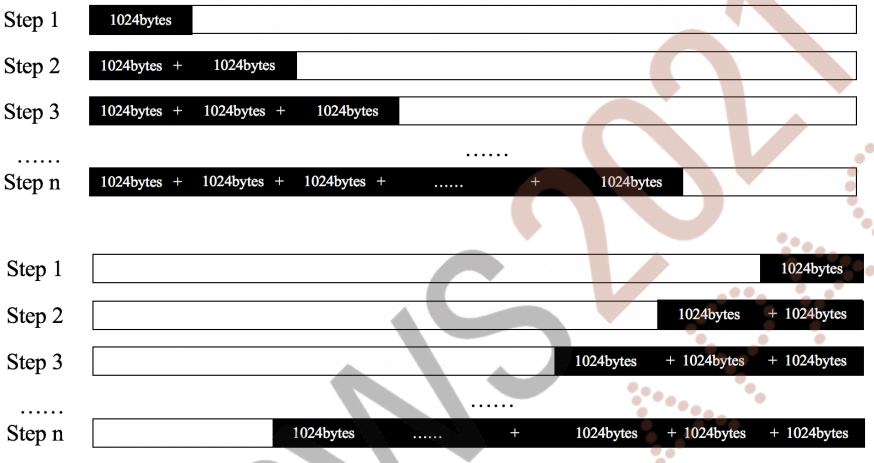

Figure 5. Continuous sampling from different positions

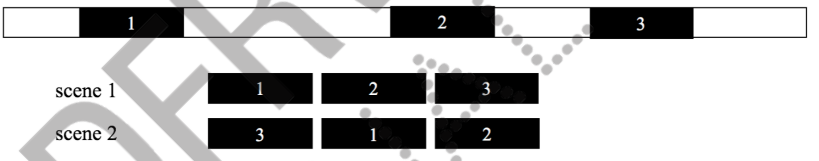

Figure 6. Schematic diagram of un-continuous sampling

5. Results and Evaluation

We first experimented with continuous fragments of different lengths in an attempt to find out the effect of fragment length on experimental results. Subsequently, we tested non-contiguous fragments. The tests were divided into two types, one with ordered fragments and the other with random sequences (there was a certain possibility that they are ordered). This experiment simulates the characteristics of the files we acquired during package capturing process that the packages obtained are not continuous and their order is unknown. Using the model above, the experimental results are as follows. The results shown in the table are averages of the results output by above model after testing for several times.

5.1. Continuous Sampling

In the experiment of continuous malicious code fragments, we compared the effect of the length of the extracted fragments on the experimental results. The results are shown in Table 1.

Table 1. Average results using continuous fragments of different lengths

Fragment length(bytes)	Accuracy (%)	Recall (%)	Precision (%)	f1-score (%)
32	64.39	63.93	67.79	65.81
128	67.85	66.28	67.13	66.70
512	70.65	72.73	65.80	69.10
1024	**76.75**	**76.63**	**79.81**	**77.81**
4096	78.59	79.67	78.24	79.21
10,000	**80.78**	**81.89**	**79.55**	**80.71**
30,000	80.13	81.10	80.62	80.86

60,000	**86.92**	**83.34**	**85.42**	**84.32**
100,000	82.79	83.38	82.62	83.00
200,000	79.47	78.19	71.63	79.87
>200,00	77.78	78.57	82.75	79.64

By observing the experimental results in the table, the following conclusions can be drawn:

1) Even when the sampling length is very short, only 32 bytes, the model is still available, the detection rate is close to 65%. This proves that there is indeed a certain pattern in a very short random segment that can distinguish malware, but it is very difficult to find by manually. Using deep learning can indeed summarize unexpected features. In addition, the reason for low detection rate when the fragments may because most of extracted fragment are both exists in many malwares and benign wares.
2) With the increase of sampling length, each data is called an upward trend. So, we can conclude when the length of the fragment is under 60000 bytes, the longer the theoretical segment, the better the effect.
3) The model receives breakthrough progress when the length of continuous fragment reaches 1024, 10000 and 60,000. The highest result 86% is reached at the 60,000-length point, followed by a slight decrease, and becomes even lower when the length is larger than 200,000 bytes. This explain that this model actually has the disadvantage that it is not suitable for too long input directly.

In order to explore the relationship between segment position and the result, the results are shown in Table 2.

Table 2. Average experimental results using continuous fragments of different lengths

Fragment length(bytes)	Extract from front (%)	Extract from End (%)	Extract randomly (%)
32	67.82	62.96	64.39
128	75.68	66.28	67.85
512	80.02	71.36	70.65
1024	82.84	72.09	76.75
4096	83.63	73.31	78.59
10,000	86.75	76.18	80.78
30,000	**87.99**	**80.71**	**80.13**
60,000	85.67	77.07	86.92
100,000	85.98	81.33	82.79
200,000	82.29	78.73	79.47
>200,00	80.07	80.52	77.78

It can be seen from Table 2 that the best detection is extracted from the front of a file. And extracting segments from the end has the worst detection effect. This indicates that the beginning of the file may contain more information useful for detection, and the PE file header is generally at the forefront of a PE file. Therefore, this result is in line with our understanding. This result reminds us that if we can get the full content of the file, we can also extract the beginning of it for fast detection.

5.2. Non-continuous Sampling

We next tested the effect of non-continuous fragments of different lengths. In detail, we randomly choose several fragments from one PE file and use these segments to do training and testing. Among them, we tested fragments with and without order.

Selection of baseline. In this experiment, in order to make our model universal, we hope that the baseline of fragment we selected is small enough but can achieve a relatively high detection accuracy. In this case, if the fragment of the same file is long, we can still divide the longer segment into several smaller base length segments, judge them separately, and finally integrate the results. This also allows us to detect malware fragments of any length without excessive padding. Another reason is that as mentioned in Section 1, the enterprises always perform packet capturing to gain malwares, the packet length is limited, and the order of packages is uncertain. In the process of packet transmission, the Maximum Transmission Unit of the packet is generally set to 1,500 bytes. We want the baseline length we choose to be smaller than this value, so that even if there is only one package, we can detect whether it is malware. Finally, we selected 1KB (1024 bytes) as the baseline.

Sig-vote method. In the beginning, two ideas for solving this problem were proposed. The first one is to directly train a given segment after splicing them to a long sequence, we call it method 1. The second one is to separately detect each segment of one file by using the trained model in Section 5.1 and integrate the results, we call it method 2. As mentioned in Section 4, we need to de- duplicate the data used in this part and in Section 5.1 to avoid the test data have been trained before. For all the ordered fragments, we found the results of both methods is similar. For un-ordered

fragments, we found the method 2 performs better. The method we proposed for integrating is similar to vote, but has a little bit difference, so we call it sig-vote. As mentioned in Section 3, the last layer in our model is sigmoid function. This allows us to calculate the sum of all fragments' output. By comparing this sum with our present threshold, if it is greater than the threshold, then the file is judged

Table 3. Average results using non-continuous fragments

Fragment length(bytes)	Order or not	Accuracy (%)	Recall (%)	Precision (%)	F1-socere(%)
2*1024	Y	81.17	82.16	81.51	81.84
2*1024	N	78.29	76.90	81.17	78.98
3*1024	Y	83.12	82.71	82.00	82.35
3*1024	N	79.48	78.47	81.25	79.84
4*1024	Y	82.71	82.58	82.05	82.09
4*1024	N	83.14	82.86	83.13	83.00
5*1024	**Y**	**82.68**	**83.88**	**84.23**	**84.03**
5*1024	N	81.66	81.39	81.57	81.48
6*1024	Y	85.27	84.81	86.24	85.43
6*1024	N	81.60	81.66	81.57	81.62
7*1024	Y	82.88	84.77	81.44	83.07
7*1024	N	83.59	83.11	84.01	83.56

to be malware. By doing lots of experiments, the threshold was set by using following equation (2) to get higher accuracy, the parameter in it is a result based on times of experiments. Suppose $frag\_num$ represents the number of un-continuous fragments, then

$$threshold = 0.42 \times frag\_num \quad (2)$$

However, this this is not necessarily the best threshold value for each case. In real life, the threshold can be adjusted according to the different requirements of recall and precision.
Table 3 shows the results for different length of un-continuous fragments. In 'fragment length' field, $n * 1024$ means we will randomly extract n 1024-byte fragments from the same file. In 'ordered or not' field, Y means that input is ordered, this part of results is gained by using method 1 above, and N means that the input order is random (there is also a certain probability that the fragment is ordered), this part of results is gained by using method 2 above.

1) If the input segment is ordered, the experimental results do not change significantly with the length of the fragments, and the results remain at around 82%.
2) If the input segment is unordered, the experimental result will increase slightly with the increase of the length, and then remain basically unchanged, maintaining at around 83%.
3) By comparing with the experimental results in Section 5.1, it can be found that, if the sum of the lengths of the extracted fragments is shorter, the discontinuous fragment effect is significantly better than the continuous fragments. As the length increases, this advantage gradually diminishes. This may be because when the sum of the lengths of the extracted segments is short, extracting two segments at different positions increases the generality of the extracted fragments to the overall file. Specifically, the extracted continuous segments hold bigger possibility that they are meaningless to determine whether it is malware, such as there are a lot of zeros in it, but randomly extracting two non-contiguous fragments can reduce this probability. Because in PE file, the end of each section is filled with a large number of zeros, and the probability that both

fragments are at the end of each section is reduced. When the extracted continuous segments become longer, the probability of this is reduced because the selected segments are long enough, they nearly never filled with zeros, so the two sampling methods behave similarly.

4) Although method 1 is better when the fragments is ordered, but it needs more cost because it always need to retrain the model when the fragments length change. Method 2 has higher universality.

5.3. Comparison Experiment

For continuous fragments. To validate the advantages of our proposed model, we compared it with other commonly used deep learning models including Normal CNN, Long-Short Term Memory(LSTM), Deep Neural Network(DNN) and Hidden Markov Model (HMM). For the other models, we have also done many experiments to tuned them perform better. Table 4 shows the results.

Table 4. Average results for continuous fragment of different kinds of model

Fragment length(bytes)	Our model (%)	LSTM (%)	Normal CNN (%)	DNN(%)	HMM(%)
32	64.39	60.47	57.89	59.28	60.32
128	67.85	68.16	64.01	66.51	66.76
512	70.65	69.87	68.29	68.09	65.01
1024	76.75	71.62	70.34	69.34	68.27
4096	78.59	73.59	73.01	70.26	69.56
10,000	80.78	79.48	77.95	74.29	72.84
30,000	80.13	75.61	75.37	72.54	71.92
60,000	**86.92**	76.69	75.20	70.43	69.34
100,000	82.79	72.77	73.94	72.21	73.69
200,000	79.47	70.56	69.42	76.23	74.07
>200,00	77.78	71.32	68.30	74.06	72.58

It can be seen that CNN's effect is significantly better than the other two methods, especially when the input sequence becomes longer.

For un-continuous fragments. Although the method of detecting separately for each segment and finally integrating the results is not very good in a given discontinuous ordered fragment, for a disordered fragment, sig-vote performs much better than directly connecting the fragments.

Table 5. Average results for un-continuous unorder fragments of different kinds of model

Unordered fragment length (bytes)	Sig-vote(%)	LSTM (%)	Text-CNN(%)
2*1024	78.29	78.69	78.84
3*1024	79.48	78.17	78.80
4*1024	**83.14**	80.61	82.71
5*1024	81.66	75.57	80.04
6*1024	81.60	74.47	74.68
7*1024	**83.59**	71.24	71.08

Table 5 proves that for non-contiguous segments, the results obtained by doing sig-vote for the chosen fragments are more ideal and stable. The more fragments are extracted, the more confusing the order is, and this will cause more serious impaction to results of the method which splicing the out-of-order segments together. In fact, the method sig-vote is very malleable as well. For example, in our experiments with continuous fragments, we found that the result of the continuous segment of 60,000 bytes is the best. When a given file fragment is very large, much larger than 60,000 bytes, we can use a continuous model of 60,000 bytes as the baseline for detection. In fact, we tested with a number of continuous and non-continuous samples containing several 60,000 bytes, we finally get up to 91.04% accuracy when there is 4 times length of 60,000. So, this method can also be applied to the detection of the entire file.

However, a 60,000 bytes (about equal to 59 KB) fragment is not small enough for many PE files, so it may not suitable for some of the scenes.

The training time is proportional to the length of the fragment. For our data set, the maximum training time when using GPU TITANx4 is no more than half an hour, which is very efficient.

6. Conclusion

In this paper, we have described the malware detection by using small fragments of them. CNN-based method is used to effectively avoid the risk of extracting fragile features. We experimented with continuous and non-contiguous, ordered and out-of-order malware fragments and came up with some conclusive experimental conclusions. We believe that we have contributed to this malware detection field, including:

1) We proposed a method based on the detection of small malicious code fragments, and gave the usage scenarios of the method in real life.
2) Through the observation of the data, we explained the reasons why the CNN model is selected when directly using binary to do the detection, the reasons include characters of binary like sparsity of the input, complexity of time and space, and the relationship between CNN's translation invariance and structure of the PE file.
3) We experimented with continuous and non-contiguous, ordered and unordered malicious code fragments, comparing their final effects and explaining the effects. Finally proved that it is feasible to detect malwares using small fragments only.

In the future, we will continually modify the model to improve the detection rate. We will also try to find out why deep learning can successfully detect malicious code using very short fragments and try to explore its interpretation in the next step.

Acknowledgements

These and the Reference headings are in bold but have no numbers. Text below continues as normal.

References

1. Symantec website, https://www.symantec.com/content/dam/symantec/docs/reports/istr-24- 2019-en.pdf, last accessed 2019/02/24

2. Statcounter GlobalStatus, https://gs.statcounter.com/, last accessed 2019/9/17

3. Song, W., Peng, G., Fu, J., Zhang, H., Chen, S.: Research on Malicious Code Evolution and Traceability Technology, Journal of Software 30(8), 2229−2267(2019)

4. Shafiq, M.Z., Tabish, S.M., Mirza, F., Farooq, M.: PE-Miner: Mining Structural Information to Detect Malicious Executables in Realtime. In: International Symposium On Recent Advances In Intrusion Detection (RAID), In: Lecture Notes in Computer Science, Springer, Berlin, (2009)

5. Zhang, D., Zhang, Z., Jiang, B., Tse, T.: The Impact of Lightweight Disassembler on Malware Detection: An Empirical Study. In: 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC), pp. 620–629, IEEE, Tokyo, Japan (2018)

6. Kan, Z., Wang, H., Xu, G., Guo, Y., Chen, X.: Towards Light-Weight Deep Learning Based Malware Detection. In: 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC), pp. 600–609, IEEE, Tokyo, Japan (2018)

7. Raff, E., Barker, J., Sylvester, J., Brandon, R., Catanzaro, B., Nicholas, C.: Malware Detection by Eating a Whole EXE. arXiv preprint arXiv:1710.09435 (2017)

8. Raff, E., Sylvester, J., Nicholas, C.: Learning the pe header, malware detection with minimal domain knowledge. arXiv preprint arXiv:1709.01471(2017)

9. Kolbitsch, C., Comparetti, P., Kruegel, C., Kirda, E., Zhou, X., Wang, X.: Effective and efficient malware detection at the end host. In: USENIX security symposium, pp. 351–366, Monteal, Canada(2009)

10.Simonyan, K., Zisserman, A.: Very deep convolutional networks forlarge-scale image recognition. arXiv:1409.1556 (2014)

11. Tompson, J., Goroshin, R., Jain, A., LeCun, Y., Bregler, C.: Efficient object localization using convolutional networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 648–656 (2015)

12. VirusShare, https://Virusshare.com, last accessed 2019/09/1