A Fast Multicore-based Window Entropy Algorithm

Suha S.A. Shokr, Hazem M. Bahig
Information and Computer Science Department, College of Computer Science and Engineering
University of Ha'il, Ha'il 81481, Saudi Arabia

***Abstract*—Malware analysis is a major challenge in cybersecurity due to the regular appearance of new malware and its effect in cyberspace. The existing tools for malware analysis enable reverse engineering to understand the origin, purpose, attributes, and potential consequences of malicious software. An entropy method is one of the techniques used to analyze and detect malware, which is defined as a measure of information encoded in a series of values based upon the probability of those values appearing. The window entropy algorithm is one of the methods that can be applied to calculate entropy values in an effective manner. However, it requires a significant amount of time when the size of the file is large. In this paper, we solve this problem in two ways. The first way of improvement is determining the best window size that leads to minimizing the running time of the window entropy algorithm. The second way of improvement is by parallelizing the window entropy algorithm on a multicore system. The experimental studies using artificial data show that the improved sequential algorithm can reduce the window entropy method's running time by 79% on an average. Also, the proposed parallel algorithm outperforms the modified sequential algorithm by 77% and has super-linear speed up.**

***Keywords*—Entropy; window method; malware analysis; parallel algorithm; multicore**

I. Introduction

With increased internet use, social media, and data sharing, users must better secure themselves to protect vulnerable information. One of the dangerous software that faces the user is malware.

Malware is a set of instructions that run on a computer, specifically designed to harm the user or the target system by making the system do something that an attacker wants it to do, like steal personal information, delete files, commit fraud, or steal software serial numbers. Moreover, new malware is registered periodically, posing a challenge to cybersecurity efforts. Malware remains one of the most potent threats in cyberspace, despite significant advances in cyber security mechanisms and their ongoing evolution.

Different kinds of malware (e.g., viruses, bots, rootkit, backdoors) can be distributed via various channels, transmitted, and used in various techniques to perform malicious operations [1].

The process of analyzing the malware (with and without execution) is called malware analysis. The main objectives of analyzing malware are to study malicious software's origin, purpose, attributes, and potential impact.

A large number of malware analysis techniques have been proposed. These techniques can be classified into two main categories [2]: static and dynamic. The main difference between static and dynamic analysis is the examination of the malware with or without running it. The two categories contain many malware detection techniques such as signature-based, specification-based, behavioral-based, and heuristic-based [3]. One of the techniques used in malware analysis is the running window entropy (RWE) method, where the entropy is a measure of information encoded in a series of values based upon the probability of those values appearing [1].

The entropy method has been used in many applications such that malware detection [4], lung sound classification [5], gender violence classification [6], Arabic text classification [7], image steganography [8], and sediments quality evaluation [9].

For analyzing malware using entropy algorithm, there are many research papers have been proposed such as [10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28].

Lyda and Hamrock [15] analyzed malware utilizing structural entropy, which employs a skip value to ignore the entropy calculation at each index. McMillan and Garman [21] obtained the same result utilizing a statistical decision concept using malware entropy information.

Sorokin [26] proposed a method based on dividing the file into segments and then applying a discrete wavelet transform (DWT) to the structural entropy to compare the segments of the input file. Baysa et al. [11] used the same strategy to identify the malware changes. The method was implemented on binary files, and there was no need to perform any preprocessing to detect the malware.

Bat-Erdene et al. [12] used the entropy method to detect unknown packers. The method is based on representing the entropy value as a symbolic aggregate approximation (SAX) and measuring the symbol's similarity in the SAX sequence. The accuracy of this method on the tested data is 95%. Radkani et al. [25] suggested a new method to detect metamorphic malware based on entropy and dissimilarity measures. The drawback of this method is the measurement limitation, which is based on the opcode (operation code) frequency.

Jones and Wang [19] suggested a method to reduce the running time of the RWE algorithm. The method is based on removing a nested "for" loop from the old version of the algorithm. The same authors [20] used the entropy concept to extract the feature set used in machine learning algorithms.

Menéndez et al. [22] introduced a new concept called entropy time series to detect malware. The technique uses a time series to represent an entropy signature. Then, the

technique applies wavelet analysis to a file's entropy to obtain a simplified signature. The main advantage of this strategy is reducing the running time of malware analysis.

In light of the previous discussions about the use of entropy in malware analysis, the main question of the research is: how to reduce the running time of the entropy method when the size of the file is large?

This paper has focus on the RWE algorithm because the window method is one of the strategies used in different fields, such as [29, 30], to reduce the running time. This study aims to demonstrate the effect of changing the window size on the RWE algorithm in terms of running time. In addition, the paper explores if there is a common window size that reduces the running time of the RWE algorithm for all malware samples and look at ways to optimize the RWE algorithm's running time. Finally, how to parallelize the sequential algorithm to reduce the running time.

The remainder of this paper is organized as follows. Section II gives briefly the foundation concepts of the RWE algorithm. In Section III, the modified RWE algorithm in terms of execution time is given. The second improvement for RWE algorithm using a multicore system is given in Section IV. The experiments and their analysis for the proposed algorithms on artificial data are given in Section V. Finally, the conclusion and open question are given in Section VI.

II. Running Window Entropy Method

In this section, we briefly give the main concept of entropy in subsection A. In subsection B, we give the main idea and steps of the RWE method that are used to calculate the entropy values of the malware sample.

A. Entropy Concept

Shannon [1] introduced the concept of entropy in information theory to measure the degree of randomness in a data set. The general formula for Shannon entropy is given by

$$H(X)= -\sum_{x\in\Sigma} p(x) \log p(x) \quad (1)$$

where

- X is a discrete random variable and has values belonging to the alphabet Σ,
- Σ is the alphabetic domain of the malware sample, and
- p(x) is the probability distribution of the value x and has a value belonging to the range [0, 1].

In information theory, it is essential to recognize that more relevant information is present in the data when the entropy value is higher than when it is lower. This means that a higher entropy value denotes more randomness and useful, unpredictable information.

B. Running Window Entropy Method

Given a malware sample D of size n as $D=(d_1, d_2, d_3, \dots, d_n)$. The RWE algorithm, is a method based on dividing sample D as continuous overlap windows, each window of size w. A window of size w that starts at index i in the data D represents as follows.

$$W_i =(d_i, d_{i+1}, d_{i+2}, \dots, d_{i+w-1}), \quad (2)$$

where $1\le i\le n-w+1$.

Since the malware sample represents a sequence of bytes, each byte ranges from 0 to 255. Therefore, the entropy for W_i can be calculated as follows:

$$H_i=H(W_i)= -\sum_{0\le k\le 255} p(k) \log p(k) \quad (3)$$

where p(k) is the probability of byte k within W_i and is equal to the total number of the byte k (symbol) within W_i, t_k , to the size of the window; i.e. $p(k)=t_k/w$.

In the case of the byte, k, does not appear in the window W_i, then $t_k =0$ and the method ignores the log p(k) value because the logarithm of zero value is undefined.

Therefore, the malware sample after applying the RWE algorithm represents a sequence of values.

$$H =(H_1, H_2, H_3, \dots, H_{n-w+1}) \quad (4)$$

The algorithm uses two arrays:

- H is an array of size n-w+1 and the value of H_i represents the entropy value for the window number i, $1\le i\le n-w+1$.
- t is an array of size 256 and the value of t(j) represents the total number of the byte j in a certain window.

The RWE algorithm consists of n-w+1 iterations, $1\le i\le n-w+1$, as follows:

Step 1: Initialize the array t with zero.

Step 2: Compute t(j) for each symbol in the window W_i by incrementing t(j) with 1, where $0\le j\le w-1$.

Step 3: Calculate the entropy, *e*, of the window W_i using (3) if $t(j)\neq 0$ and $0<j<256$.

Step 4: Calculate $H_i=e/K$, where K=8.

The reason for taking the value of K equal to 8 is the values of (3) between 0 and 8.

III. Improved RWE Method

In this section, we introduce two comments on the RWE algorithm, then modify the RWE algorithm.

The first comment on the RWE algorithm is that the window size was selected experimentally from 256 to 2048 as in [19]. No experimental studies have addressed the effect of window size when w<256 or w>2048. What is the effect of these values of w on the RWE algorithm? Does the same behavior occur if we start from a window size equal to 4?

The second comment on the RWE algorithm is that when the size of window is less than 256, why do we take the summation of (3) from 0 to 255? Can we propose a fast method to calculate H_i?

For the second comment, we propose a modification on RWE algorithm as follows. If the value of w is less than 256, then we can compute (3) as follows.

$$H_i=H(W_i)= -\sum_{k\in Wi} p(k) \log p(k) \quad (5)$$

The difference between (3) and (5) is the boundaries of the summation that are taken over the term p(k) log p(k).

The following steps computes the value of H_i as follows.

- Compute t(j) for each symbol in the window W_i by increment t(j) with 1, where $1 \le j \le w$.
- For each symbol (byte) in the window W_i do the following: if t(j) ≠0 then update the value of the entropy, *e*, for the j-th symbol of window W_i and set t(j) equal to 0.

The reason for setting t(j)=0 after updating the entropy is that the symbol (byte) j may exist many times in the window W_i. Setting the value of t(j)=0, will remove the step of initializing the array t for each new iteration.

The steps of the modified window entropy method, MRWE, are given in Algorithm 1.

Algorithm 1: Modified Running Window Entropy (MRWE)

Input: Malware sample D of size n, window size w, K=8

```
1.    If w<256 then
2.        Initialize t with 0
3.        For i=1 to n-w+1 do
4.            For j=0 to w-1 do
5.                t(d_{i+j})= t(d_{i+j})+1
6.            End for
7.            e=0
8.            For j=0 to w-1 do
9.                If t(d_{i+j})≠0 then
10.                   e=e-( t(d_{i+j})/w*log(t(d_{i+j})/w))
11.                   t(d_{i+j})=0
12.               End if
13.           End for
14.           H(i)=e/K
15.       End for
16.   Else
17.       For i=1 to n-w+1 do
18.           For j=0 to w-1 do
19.               t(d_{i+j})= t(d_{i+j})+1
20.           End for
21.           e=0
22.           For j=0 to 256 do
23.               If t(j)≠0 then
24.                   e=e-( t(j)/w*log(t(j)/w))
25.               End if
26.           End for
27.           H(i)=e/K
28.       End for
29.   End if
```

Output: Entropy values $H=(H_1,H_2,\ldots,H_{n-w+1})$

Note that the first comment will be discussed in the experimental section.

IV. Parallelizing MRWE Algorithm

In this section, the parallelization of the MRWE algorithm (similarly RWE algorithm) on a parallel shared memory model is discussed. The parallel model consists of multi-processors/multi-threads that can be communicate via a global memory.

The idea of parallelizing MRWE, PMRWE, is based on dividing the input array D into t subarrays. Each subarray, $D_i=(d_{(i-1)\alpha+1}, d_{(i-1)\alpha+2}, d_{(i-1)\alpha+3},\ldots, d_{i\alpha})$, consists of α elements approximately, where $\alpha= (n-w+1)/t$ and $1 \le i \le n/t$. The value of numerator of α represents the number of windows in D. Then each thread, t_i, works on the subarray D_i to compute the entropy of this part. Each thread generates α entropy values, each value represents the entropy value for a window in D_i.

All the threads work on their subarray simultaneously. This means that no shared data between the threads. The complete pseudocode for PMRWE algorithm is given in Algorithm 2.

Algorithm 2: Parallel MRWE (PMRWE)

Input: Malware sample D of size n, window size w, K=8, and t threads.

```
1.    α= (n-w+1)/t
2.    If w<256 then
3.        Initialize t with 0
4.        For b=1 to t Do Parallel
5.          For i=1 to α do
6.            For j=0 to w-1 do
7.                t(d_{(b-1)α+i+j})= t(d_{(b-1)α+i+j})+1
8.            End for
9.            e=0
10.           For j=0 to w-1 do
11.               If t(d_{(b-1)α+i+j})≠0 then
12.                   e=e-( t(d_{(b-1)α+i+j})/w*log(t(d_{(b-1)α+i+j})/w))
13.                   t(d_{(b-1)α+i+j})=0
14.               End if
15.           End for
16.           H((b-1)α+i)=e/K
17.         End for
18.       End for parallel
19.   Else
20.       For b=1 to t Do Parallel
21.         For i=1 to n-w+1 do
22.           For j=0 to w-1 do
23.               t(d_{(b-1)α+i+j})= t(d_{(b-1)α+i+j})+1
24.           End for
25.           e=0
26.           For j=0 to 256 do
27.               If t(j)≠0 then
28.                   e=e-( t(j)/w*log(t(j)/w))
29.               End if
30.           End for
31.           H(i)=e/K
32.       End for
33.   End for parallel
```

Output: Entropy values $H=(H_1,H_2,\ldots,H_{n-w+1})$

V. Experimental Studies

The aim of this section is to study experimentally all proposed algorithms, sequential and parallel, and compare them with the original algorithm. The section consists of three

subsections. In the first subsection, we describe briefly the platform specifications and data set used in the implementation. The sequential and parallel comparisons are providing in the second and third subsections, respectively.

A. Platform Specifications and Data

All algorithms, RWE, MRWE, PRWE, and PMRWE, have been implemented using the programming language Python. The Python code for the RWE algorithm exist in [17]. All methods run on a machine of type Intel Core i7 with 8 logical processors speeds 3.1 GHz. The machine is able to run eight threads concurrently. The machine worked under the Windows operating system.

For sequential comparison, the running time of each sequential algorithm is measured by fixing n and w first. Then, generate a malware sample of size n and run the algorithms ten times on the malware sample. This process is repeated 20 times for different malware samples. Finally, the average time for all instances, i.e., the average of 200 runs, is calculated. The running time is measured in seconds.

The data sizes of malware samples are 1k, 2k, 4k, 8k, 16k, and 32k, whereas the window size is w=4, 8, 16, 32, …., and n/2.

For parallel comparison, the number of threads used in the comparison are 2, 4, and 8. The data sizes for malware samples are ½ MB, 1 MB, 2 MB, 4 MB, and 8 MB. The value of w is equal to w_{min}, where w_{min} is the size of window that minimize MRWE algorithm. The value of w_{min} is calculated experimentally in the second subsection.

B. Sequential Comparison

In this subsection, we study the two algorithms, RWE and MRWE, experimentally for two goals. The first goal is to answer the first comment mentioned in the previous section: what is the effect of changing the value of the window, w, on the RWE and MRWE? The second goal is: which value of w leads to minimal time for the method?

The results of running the two methods are shown in Fig. 1. From the analysis of the subfigures in Fig. 1, the behaviors of the two methods is divided into three ranges based on w as follows.

Case 1: w in the range [4,32]. In this case, the following observation were made.

- The running time for the MRWE is less time than the RWE in the case of w=4, 8, 16, and 32 for all values of n.
- The difference between the two running times for MRWE and RWE decreases with increasing the value of w.
- The improvement percentages of MRWE compared to RWE are shown in Table I. The average percentage of improvements are 79%, 66%, 45%, and 20% for w=4, 8, 16, 32, respectively.

Case 2: w in the range [64,256[. In this case, the following observation were made.

- The running time for the RWE is less than the MRWE algorithm in the case of w=64 and 128 for all values of n.
- The difference between the two methods when w=64 is very small and approximately equal to 5%.
- When w=128, the RWE algorithm improves the MRWE algorithm by 25% on average.

Case 3: w in the range [256, n]. In this case, the two methods are equal because the two methods use the same strategy.

From the observations of the three cases, we conclude that the MRWE algorithm can be achieve the minimal time when the window size is small.

TABLE I. PERCENTAGE OF IMPROVEMENTS FOR MRWE

n	**w**			
	4	***8***	***16***	***32***
1k	79.49%	66.14%	44.95%	20.05%
2k	79.19%	66.79%	45.83%	19.57%
4k	79.86%	66.45%	45.84%	20.29%
8k	79.29%	65.77%	45.77%	20.02%
16k	79.75%	66.06%	45.75%	20.57%
32k	79.06%	65.40%	45.83%	19.52%

C. Parallel Comparison

The results of running the two parallel algorithms, PRWE and PMRWE, on a multicore system using 2, 4, and 8 threads are shown in Fig. 2. The results show that the running time for PMRWE algorithm outperforms the PRWE algorithm when w_{min}=4. For example, the running times for PRWE and PMRWE algorithms are 39 and 17.6 seconds, respectively, when n=1 MB and t=2; see Fig. 2(a). From the experimental results, the percentage of improvements for the PMRWE algorithm over the PRWE algorithm using threads two, four and eight are 79.3%, 77.2%, and 77.8%, respectively, on average; see Table II.

Also, we can compute the speed up of the PMRWE algorithm by computing the ratio between the running time of MRWE algorithm and PMRWE algorithm. The results of speed up of PMRWE algorithm is given in Fig. 3. The value of speed up when t=2 is linear and this value increase to be super-linear speed up when t=4 and 8. The increasing in the speed up came from the size of file is divided into small segment when t>2.

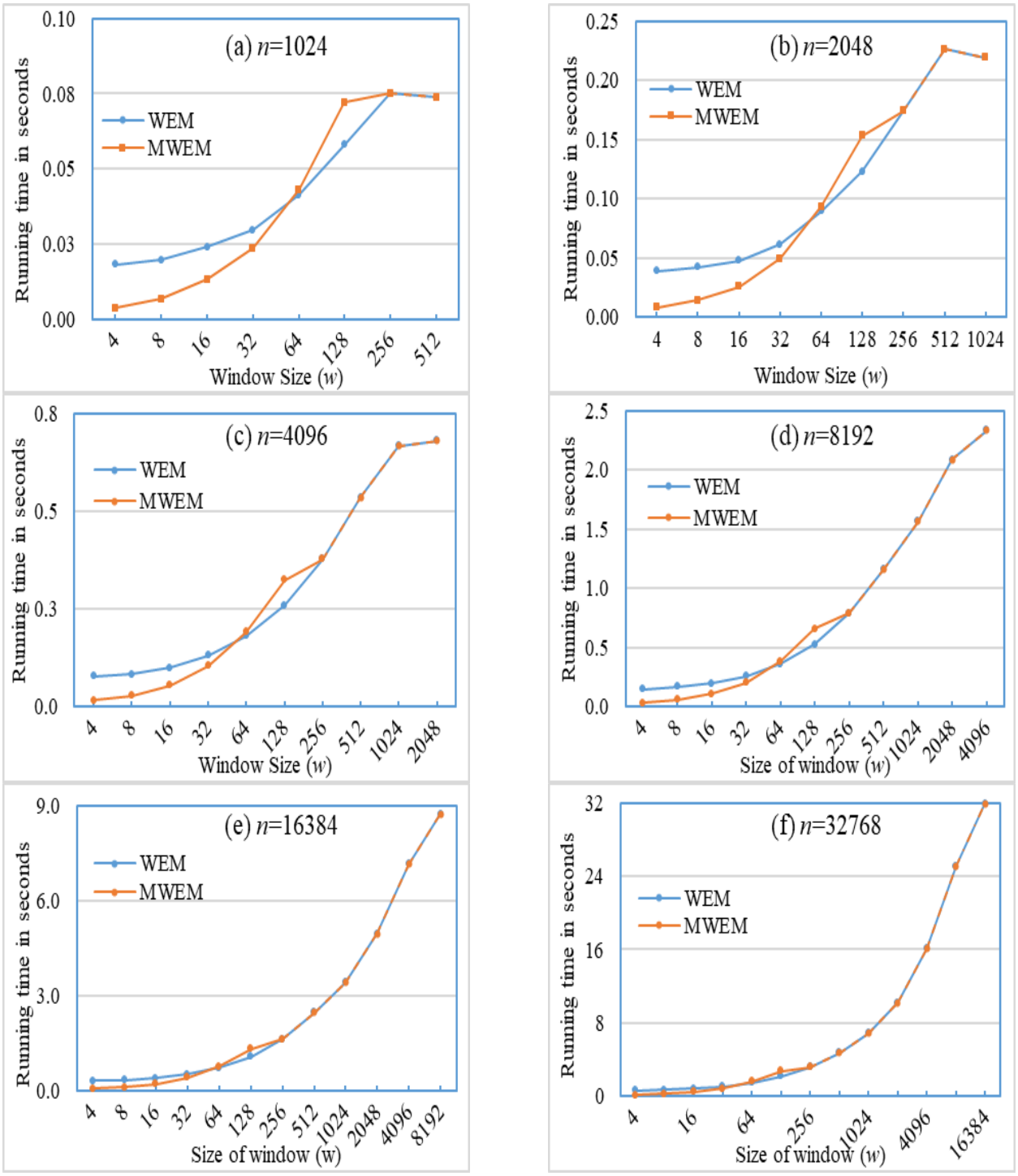

Fig. 1. Running Time Comparison between RWE and MRWE Algorithms.

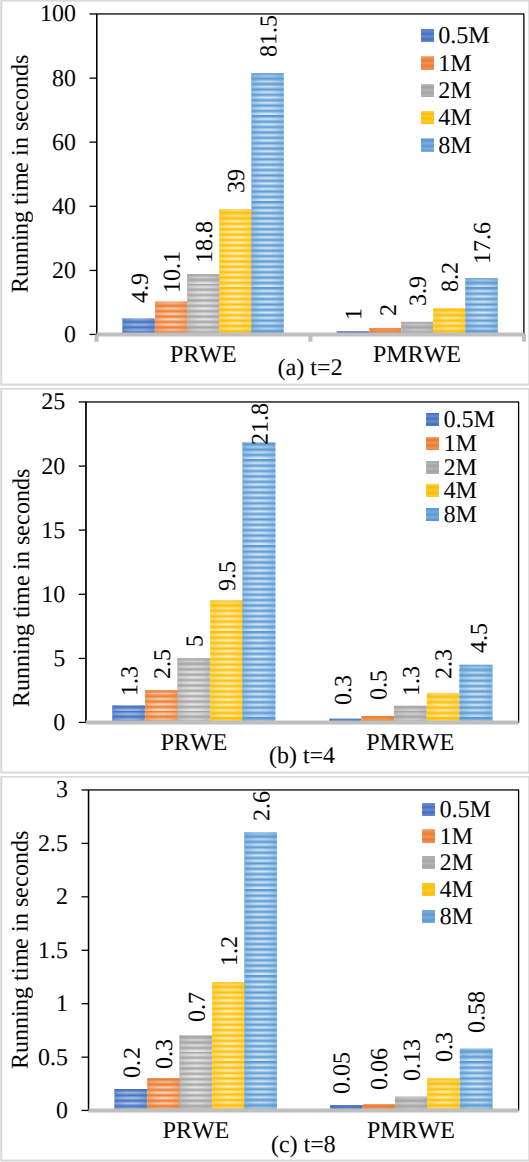

Fig. 2. Time Comparisons between PRWE and PMRWE Algorithms.

TABLE II. PERCENTAGE OF IMPROVEMENT FOR PMRWE ALGORITHM

n	Number of threads		
	2	*4*	*8*
0.5M	79.6%	76.9%	75.0%
1M	80.2%	80.0%	80.0%
2M	79.3%	74.0%	81.4%
4M	79.0%	75.8%	75.0%
8M	78.4%	79.4%	77.7%
Avg.	**79.3%**	**77.2%**	**77.8%**

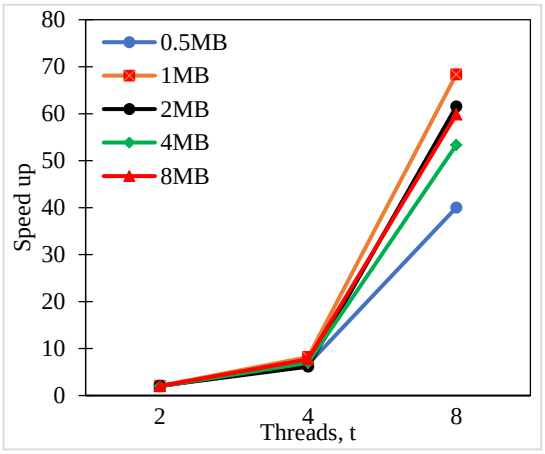

Fig. 3. Speed up of PMRWE Algorithm.

VI. CONCLUSION

Malware analysis is important to study malicious software's functionality, purpose, origin, and potential impact. The running window entropy method is one of the methods used to find the entropy values of the malware sample. The main drawback of this method is the amount of time when the size of file is large. In this paper, two improved algorithms are given. The first improved algorithm is based on determining the best window size to minimize the running time of the window entropy method. The second improved algorithm is the parallelization of the improved sequential algorithm.

The experimental studies for different malware sizes and best window size show that the improved sequential and parallel methods outperform the original and the improved sequential methods, with 79% and 77% faster time, respectively.

There are many open questions related to improve the entropy method for malware analysis as follows: (1) How to use other high-performance systems, such as distributed system, GPU (graphic processing unit) and cloud, to speed up the execution time? (2) How to extend this work to other method for window entropy?

ACKNOWLEDGMENT

This research has been funded by Scientific Research Deanship at University of Ha'il - Saudi Arabia through project number GR-22 031.

REFERENCES

[1] Panda Security, "2017 Cybersecurity Predictions," 2017.

[2] E. Gandotra, D. Bansal, S. Sofat, " Malware analysis and classification: a survey," J. of Inf. Security, 5, pp. 56-64, 2014.

[3] I. Kara, "A basic malware analysis method," Computer Fraud & Security, vol. 2019, Issue 6, pp. 11-19, 2019.

[4] Mafaz Mohsin Khalil Al-Anezi, "Generic Packing Detection Using Several Complexity Analysis for Accurate Malware Detection" International Journal of Advanced Computer Science and Applications, 5(1), 2014.

[5] Achmad Rizal, Risanuri Hidayat and Hanung Adi Nugroho, "Comparison of Multilevel Wavelet Packet Entropy using Various Entropy Measurement for Lung Sound Classification" International Journal of Advanced Computer Science and Applications(IJACSA), 10(2), 2019.

[6] Abdul Azim Ismail and Marina Yusoff, "An Efficient Hybrid LSTM-CNN and CNN-LSTM with GloVe for Text Multi-class Sentiment Classification in Gender Violence" International Journal of Advanced Computer Science and Applications, 13(9), 2022.

[7] Ibrahim S Alkhazi and William J. Teahan, "Classifying and Segmenting Classical and Modern Standard Arabic using Minimum Cross-Entropy" International Journal of Advanced Computer Science and Applications(IJACSA), 8(4), 2017.

[8] Ke-Huey Ng, Siau-Chuin Liew and Ferda Ernawan, "An Improved RDWT-based Image Steganography Scheme with QR Decomposition and Double Entropy" International Journal of Advanced Computer Science and Applications(IJACSA), 11(3), 2020.

[9] Alexi Delgado, Betsy Vilchez, Fabian Chipana, Gerson Trejo, Renato Acari, Rony Camarena, Víctor Galicia and Chiara Carbajal, "Applying Grey Clustering and Shannon's Entropy to Assess Sediment Quality from a Watershed" International Journal of Advanced Computer Science and Applications(IJACSA), 12(9), 2021.

[10] D. Baysa, "Structural entropy and metamorphic malware," J. Comput. Virol. Hacking Tech., vol. 9, no. 4, pp. 179–192, 2013.

[11] M. Bat-Erdene, H. Park, H. Li, H. Lee, M. Choi, "Entropy analysis to classify unknown packing algorithms for malware detection," Int. J. Inf. Secur. 16, pp. 227–248, 2017.

[12] M. Bat-Erdene, T. Kim, H. Li, and H. Lee, "Dynamic classification of packing algorithms for inspecting executables using entropy analysis," Proc. 8th Int. Conf. Malicious Unwanted Softw. "The Am. MALWARE, pp. 19–26, 2019.

[13] A. Chakrabarti, G. Cormode, and A. Mcgregor, "A near-optimal algorithm for estimating the entropy of a stream," ACM Trans. Algorithms, vol. 6, no. 3, pp. 1–21, 2020.

[14] K. Han, J. H. Lim, and E. G. Im, "Malware analysis method using visualization of binary files," Proceedings of the Research in Adaptive and Convergent Systems. ACM, Montreal, Quebec, Canada, 2019.

[15] R. Lyda and J. Hamrock, "Using entropy analysis to find encrypted and packed malware," IEEE Security & Privacy, vol. 5, no. 2. pp. 40–45, 2021.

[16] A. Lall, V. Sekar, M. Ogihara, J. Xu, and H. Zhang, "Data streaming algorithms for estimating entropy of network traffic," SIGMETRICS Perform. Eval. Rev., vol. 34, no. 1, pp. 145–156, 2006.

[17] K. Jones, "Algorithms #1 python source code," 2017. [Online]. Available: https://github.com/keithjjones/csc705-alg1.

[18] K. Jones, "malgazer," 2017. [Online]. Available: https://github.com/keithjjones/malgazer.

[19] K. Jones , Y. Wang, "An optimized running window entropy algorithm," National Cyber Summit (NCS), 5-7 June 2018, pp. 72-77, 2018.

[20] K. Jones , Y. Wang, "Malgazer: An automated malware classifier with running window entropy and machine learning," 2020 Sixth International Conference on Mobile And Secure Services (MobiSecServ), 22-23 Feb., pp 1-6, 2020.

[21] C. Mcmillan, J. Garman, "System and method for determining data entropy to identify malware," Application PCT/US2008/051383.

[22] H. Menéndez, S. Bhattacharya, D. Clark, E. Barr, "The arms race: adversarial search defeats entropy used to detect malware," Expert Systems With Applications, 118: pp 246–260, 2019.

[23] S. Naval, V. Laxmi, M. S. Gaur, and P. Vinod, "ESCAPE: entropy score analysis of packed executable," Proc. Fifth Int. Conf. Secur. Inf. Networks, pp. 197–200, 2017.

[24] M. Paavola, "An efficient entropy estimation approach," Universtity of Oulu, 2011.

[25] E. Radkani, S. Hashemi, A. Keshavarz-Haddad, M. Haeri, "An entropy-based distance measure for analysing and detecting metamorphic malware," Appl Intell, 48, pp. 536–1546, 2018.

[26] I. Sorokin, "Comparing files using structural entropy," J. Comput. Virol., vol. 7, no. 4, pp. 259–265, 2011.

[27] S. Treadwell and M. Zhou, "A heuristic approach for detection of obfuscated malware," in 2009 IEEE International Conference on Intelligence and Security Informatics, 2009, pp. 291–299.

[28] M. Weber, M. Schmid, M. Schatz, and D. Geyer, "A toolkit for detecting and analyzing malicious software," in Computer Security Applications Conference, 2002. Proceedings. 18th Annual, 2002, pp. 423–431.

[29] H. M. Bahig, K.A. Fathy. An efficient parallel strategy for high-cost prefix operation. J Supercomput 77, pp. 5267–5288, 2021.

[30] H. M. Bahig, H. M Bahig, K. A. Fathy, "Fast and scalable algorithm for product large data on multicore system," Concurrency and Computation: Practice and Experience, vol. 33, Issue 2, 2021 Wiley.