J Comput Virol (2011) 7:259–265
DOI 10.1007/s11416-011-0153-9

ORIGINAL PAPER

Comparing files using structural entropy

Ivan Sorokin

Received: 1 March 2011 / Accepted: 26 May 2011 / Published online: 8 June 2011

Abstract One of the main trends in the modern anti-virus industry is the development of algorithms that help estimate the similarity of files. Since malware writers tend to use increasingly complex techniques to protect their code such as obfuscation and polymorphism, anti-virus software vendors face problems of the increasing difficulty of file scanning, the considerable growth of anti-virus databases, and file storages overgrowth. For solving such problems, a static analysis of files appears to be of some interest. Its use helps determine those file characteristics that are necessary for their comparison without executing malware samples within a protected environment. The solution provided in this article is based on the assumption that different samples of the same malicious program have a similar order of code and data areas. Each such file area may be characterized not only by its length, but also by its homogeneity. In other words, the file may be characterized by the complexity of its data order. Our approach consists of using wavelet analysis for the segmentation of files into segments of different entropy levels and using edit distance between sequence segments to determine the similarity of the files. The proposed solution has a number of advantages that help detect malicious programs efficiently on personal computers. First, this comparison does not take into account the functionality of analysed files and is based solely on determining the similarity in code and data area positions which makes the algorithm effective against many ways of protecting executable code. On the other hand, such a comparison may result in false alarms. Therefore, our solution is useful as a preliminary test that triggers the running of additional checks. Second, the method is relatively easy to implement and does not require code disassembly or emulation. And, third, the method makes the malicious file record compact which is significant when compiling anti-virus databases.

I. Sorokin (✉)
Doctor Web's Virus Lab, Ltd., 2 Malaya Monetnaya st.,
197101 Saint-Petersburg, Russia
e-mail: i.sorokin@drweb.com; igz@drweb.com

1 Introduction

One of the main trends in the modern anti-virus industry is the development of algorithms that help estimate the similarity of files. Since malware writers tend to use increasingly complex techniques to protect their code, e.g., obfuscation and polymorphism [14], anti-virus software vendors face several problems. First, there is the issue of the increasing difficulty of file scanning (e.g., due to additional emulation). Second, the use of outdated signature detection methods results in the considerable growth of anti-virus databases. Third, there is also a problem of filling file storages used by anti-virus software vendors with loads of sample files [15]. For solving such problems, a static analysis of files appears to be of some interest. Its use helps determine those file characteristics that are necessary for their comparison without executing malware samples within a protected environment.

Aside from the presence of similar byte sequences or headers in executable files undergoing comparison, a decision on their similarity may be drawn from more complex features such as file code patterns and data structures, e.g., a unique sequence of function calls or processor instructions. The solution provided in this article is based on the assumption that different samples (i.e., files) of the same malicious program have a similar order of code and data areas. Each such file area may be characterized not only by its length (i.e., the number of bytes), but also by its homogeneity (i.e., distinction of bytes). In other words, the file may be characterized by the complexity of its data order. To indicate this characteristic of a file, we use the concept of *structural*

entropy [17]. Our approach consists of two main parts. The first stage includes using wavelet analysis [2] for the segmentation of files into segments of different entropy levels. In the second stage, we use edit distance between sequence segments to determine the similarity of the files [16]. In summary, the main contribution of this article is the following:

A description of an algorithm for the segmentation of files into segments that are characterized by length and average entropy.
A review of the sequence alignment technique to compare files represented by sequences of segments.

2 Related work

As was mentioned above, our solution is based on two basic techniques: entropy analysis and sequence alignment. Both of these approaches have ever-widening application in information security.

Entropy analysis allows for the estimation of the package and encryption level of data. Such estimations may serve as a step in detecting packed data. Lyda and Hamrock [8] calculate the average and maximum entropy of a whole segmented file to identify packed and encrypted data. Perdisci et al. [10] calculate the entropy of individual segments of executable files. Together with other characteristics, this method allows for the effective use of pattern recognition techniques to classify files into "packed" and "unpacked" categories. Ebringer et al. [3] and then later Sun et al. [11], use Huffman codes to estimate entropy. By calculating a code for each byte, they use a sliding window method to build an entropy map of the whole file. This allows for a more detailed comparison of files and classifying them by packer type. Breitenbacher [1] uses his own algorithm for estimating the randomness of 16-byte blocks. Unfortunately his research is limited to reviewing an entropy map of the whole file.

In most cases, malicious code or activity can be represented as a sequence of elements or events. This allows for the use of comparison algorithms based on sequence alignment. For instance, the Smith–Waterman local alignment algorithm for sequences is convenient to use for the detection of malicious network traffic [9]. In their subsequent research of network traffic analysis, Kreibich and Crowcroft [6] propose an improved version of the Jacobson–Vo local alignment algorithm that is based on the identification of the longest increasing subsequence. Another approach [4] utilizes multiple sequence alignment (MSA) methods. The analysis of executable files is another field of use for sequence comparison algorithms. For example, in several articles [5,7,12,13], alignment algorithms are used for a behaviour comparison of malicious programs.

3 Methodology

The proposed solution lies in the static analysis of files. We do not take into account file types; that is, we ignore PE header attributes of Windows executables. The only thing of importance for us is file structure, that is, the order of its distinctive code and data areas. To determine such areas, we build an entropy map of the whole file first and then use wavelet analysis to segment it (see Sect. 3.1). When we have a representation of a file as a sequence of segments, we compare the file with other files using edit distance (see Sect. 3.2). Therefore, the algorithm as a whole includes the following stages: file segmentation and sequence comparison.

3.1 File segmentation

Any file can be characterized by properties of data it contains. For instance, one may consider how well-ordered the data are or how much space the data occupy. Among other things, if we take a look at executable files, we may notice that they contain data of various kinds: executable code, text, and packed data. All of these file areas differ not only in size, but also in the level of *informational entropy*. When an executable file may be considered as a system of such elements, then we can use the term *structural entropy* [17] for its characterization. Therefore, the main purpose of the suggested segmentation algorithm is splitting the file into segments that are characterised by size and entropy.

3.1.1 Entropy analysis

Initially the sliding window method is used to represent the source file as a time series $Y = \{y_i : i = 1, \ldots, N\}$, where N is the total number of windows.

To calculate entropy within each window, we use the Shannon's formula:

$$y_i = -\sum_{j=1}^{m} p(j) \log_2 p(j), \tag{1}$$

where $p(j)$ is the frequency of occurrence of the jth byte within the ith window, and m is a number of different bytes in the window. Please note that we consider the frequency of a byte's occurrence in an individual window and not within the whole file. This helps keep the window entropy level from depending on other bytes in the file. For instance, some researchers [3,11] calculate Huffman codes across the whole file. This results in different entropy diagrams for files of similar structure but differing length.

3.1.2 Wavelet analysis

The main task when segmenting a file is to determine those places within it where average entropy changes. We suggest using wavelet analysis [2] to extract this information from our resulting time series Y. The essence of the analysis follows. First, we choose a mother wavelet whose properties determine our ability to identify changes in analyzed data. Second, we calculate the wavelet transform of various scales. The obtained wavelet coefficients will contain information on the correlation between the used wavelet and the analyzed time series. As a result, we will be able to determine segments by analysing significant wavelet coefficients.

For the mother wavelet, we choose the Haar wavelet which has an asymmetrical form and whose zero moment equals zero:

$$\psi_{HAAR}(t) = \begin{cases} 1, & 0 \le t < 1/2, \\ -1, & 1/2 \le t < 1, \\ 0, & t < 0,\ t \ge 1. \end{cases} \tag{2}$$

Since continuous wavelet transform (CWT) is redundant due to the continuous change of scale coefficient and shift parameter, it is more cumbersome than discrete wavelet transform (DWT). Therefore, we use the following estimate to calculate DWT:

$$W(a,b) = \frac{1}{|a|^{1/2}} \sum_{i=1}^{N} y_i \psi_{HAAR}\left(\frac{t_i - b}{a}\right), \tag{3}$$

where a is a scale parameter, b is a mother wavelet shift parameter, y_i is an informational entropy level within the ith window, and N is the total number of windows in the file.

The main peculiarity of DWT is that the scale parameter a changes according to a power of 2. This means, first, that we can use multi-resolution analysis which allows us to use the values determined on the previous scale on each next scale of transformation. This results in a reduced number of reduced number of mathematical operations involving addition. Second, this placed a restriction on the source data. The number of counts of the time series should be divisible by a power of 2: $a_n = 2^n$ where n is the maximum scale. Therefore, we need to increase the time series on each side with averaged values.

From the received coefficients, we need to identify significant ones, i.e., the local extremums that have the maximum or minimum by the a and b variables. If all of the points of the local extremums in the time-scale plane are connected, then the resulting lines will build a *skeleton*. These lines represent the structure of the analysed data in full. Therefore, the segmentation algorithm's main task lies in building the skeleton of input data that is used afterwards to identify segments. The total number of segments is determined by significant wavelet coefficients on the maximum scale, while their limits are determined by significant wavelet coefficients on the minimal scale of transformation.

3.2 Sequence comparison

In most cases, similar malicious files are alike in terms of size; therefore, we will use global alignment to compare them. This method allows us to compare whole sequences while taking into account all of their elements. In turn, algorithms based on local alignment are applied mostly to sequences that differ in size and have just a few similar fragments.

For global alignment of sequences, we will use the Wagner–Fischer dynamic programming method based on the Levenshtein distance. Using this method, insertion, deletion and substitution operations will receive penalties depending on the characteristics of the compared elements (see Sect. 3.2.1).

The comparison will be carried out in two steps. First, we will align the sequences (see Sect. 3.2.2), i.e., we will look for the correspondence between similar elements. Then we will estimate the total degree of similarity between two sequences (see Sect. 3.2.3).

3.2.1 Edit cost function

Since each element of a sequence is identified by two characteristics (size and entropy), we need to select a general cost function that will determine a normalized penalty value for the mismatching of two elements depending on the difference in their sizes and averaged entropy values. We set the range for this function between zero and a certain constant which will indicate the absolute similarity and absolute difference of two sequences accordingly. So, by selecting such a function, we will be able to align two sequences.

Denoting the sizes of two elements by $size_1$ and $size_2$ while denoting their averaged entropy by ent_1 and ent_2, we may set the size penalty according to the following function:

$$\text{cost}_s = \frac{|size_1 - size_2|}{size_1 + size_2} \tag{4}$$

Here at least the size of one of the compared elements should be non-zero. For this formula, the maximum penalty for the difference in sizes equals 1. If the sizes are equal, then there is no penalty.

Now, let us set a penalty for the difference in entropy:

$$\text{cost}_e = \frac{1}{1 + \exp\left(-4 \cdot |ent_1 - ent_2| + 6.5\right)} - 0.001501 \tag{5}$$

In this formula, we use a sigmoid (see Fig. 1). Through its form we can regulate the normalized penalty value differently. In this case, two elements with a difference in entropy starting from 2 bits are considered different.

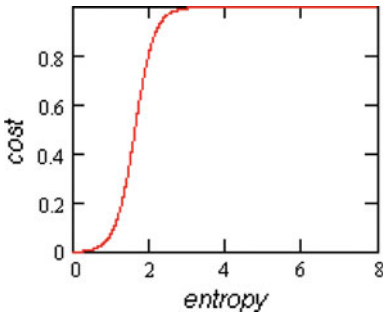

Fig. 1 Sigmoid for normalizing the difference in entropy between two segments

The total penalty for two segments is calculated as a sum of penalties for difference in size (4) and entropy (5):

$$\text{cost} = \text{cost}_s \cdot PART\_SIZE + \text{cost}_e \cdot PART\_ENT \quad (6)$$

The use of coefficients in formula (6) allows the fraction of penalty for size or entropy to be set differently when comparing two segments.

3.2.2 Sequence alignment

After we decide on the general cost function, we can determine an alignment algorithm for two sequences. Like any algorithm based on the Levenshtein distance, our algorithm utilizes dynamic programming. We set an edit matrix d, i.e., a two-dimensional array in which each element determines the comparison of corresponding subsequences. The essence of the algorithms lies in filling in this array and determining the last element that represents the resulting penalty for comparing two sequences.

Regardless of the fact that the general cost function takes into account differences in both size and the averaged entropy values of two elements, we will also additionally account for the logarithmic sizes of the corresponding elements when filling in array d. In addition, to allow for more flexible adjustment of total penalty value, we will use the constant TAX which represents the average share of the penalty for all elements.

So, when filling in the first column, which represents deletion of corresponding elements from the first sequence s_1, we will use the following formula:

$$d[i][0] = d[i-1][0] + TAX \cdot \log_{10}(s_1[i-1].size), \quad i = 1 \ldots length(s_1).$$

Likewise, to fill in the first row, which represents insertion of corresponding elements from the second sequence s_2, we use a similar formula:

$$d[0][j] = d[0][j-1] + TAX \cdot \log_{10}(s_2[j-1].size), \quad j = 1 \ldots length(s_2).$$

All other elements of array d are set according to the following formula:

$$d[i+1][j+1] = \min \begin{cases} d[i][j] + \text{cost}(s_1[i], s_2[j]) \\ \quad \cdot \log_{10}((s_1[i].size + s_2[j].size)/2) \\ d[i][j+1] + TAX \cdot \log_{10}(s_1[i].size) \\ d[i+1][j] + TAX \cdot \log_{10}(s_2[j].size) \end{cases} \quad (7)$$

In each step, we select one of the three minimal values. If the first summand is minimal, then it indicates that two elements are replaced. In this case, the edit operation receives a penalty not only from the cost function (6), but also from the average size of the two elements in logarithmic dependence. If the second summand is minimal, then it indicates that the element from the first sequence s_1 is deleted. In this case, the operation receives a penalty depending on the size of the area. Finally, if the third summand is minimal, then it indicates that the element from the second sequence s_2 is inserted. The penalty in this case also depends on the size of the area.

So, the resulting array d contains information on penalties received when comparing corresponding subsequences, and, therefore, its last element represents how large the penalty is when comparing the whole sequences s_1 and s_2. If we follow the array from its end while taking into account minimal values, then we obtain the full alignment of two sequences (Wagner & Fisher).

3.2.3 Estimating the degree of similarity

To estimate the degree of similarity between two sequences, we need to determine the maximum penalty that their comparison could have received. The maximum penalty means that all elements from the first sequence are deleted, and all elements from the second sequence are inserted with the corresponding penalties. The value is already calculated when filling in array d. Namely, it equals the sum of the last element in the first row and the last element in the first column. A good rule of thumb is to increase the estimate of the maximum penalty and thus bring together two sequences. For this, we need to recalculate the maximum penalty while taking into account the performed alignment:

$$\text{cost\_max+} = \begin{cases} 2 \cdot TAX \cdot (\log_{10}(s_1[i].size) \\ \quad + \log_{10}(s_2[j].size)), \\ \quad s_1[i]\ substitution\ s_2[j] \\ TAX \cdot \log_{10}(s_1[i].size),\ delete\ s_1[i] \\ TAX \cdot \log_{10}(s_2[j].size),\ insert\ s_2[j] \end{cases} \quad (8)$$

In other words, calculating value cost_max is similar to filling in the first column and the first row of array d, with the only difference being that we artificially increase the penalty by doubling it when replacing two elements.

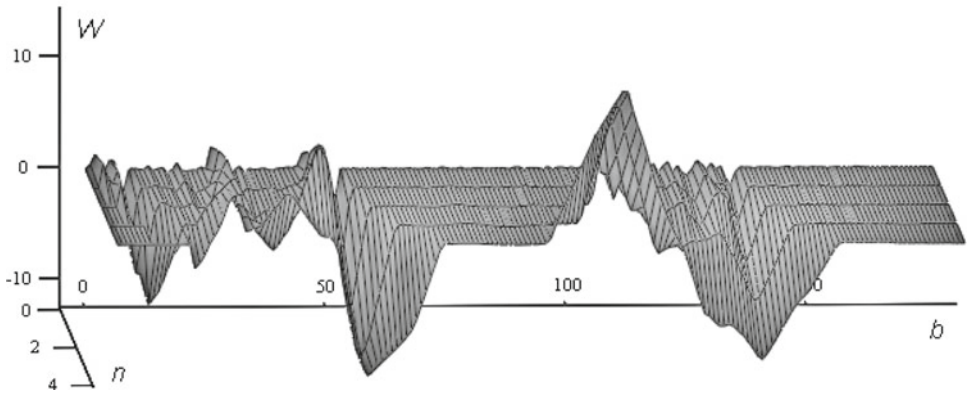

Fig. 2 Wavelet coefficients built using DWT on the first 160 counts of the first file

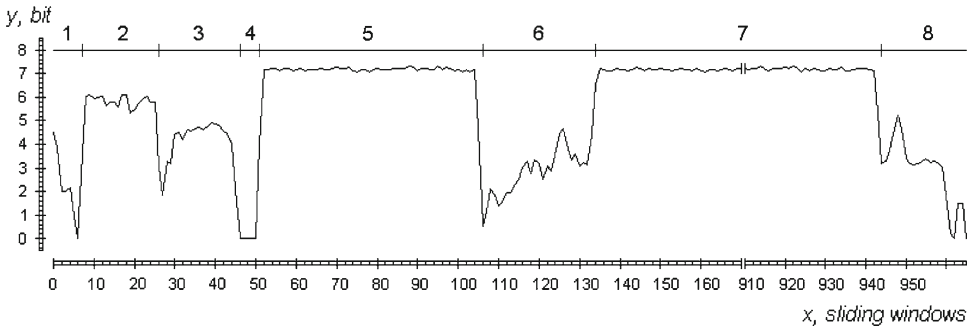

Fig. 3 Entropy diagram of the first file

Given that the last element of array d represent the true difference between two sequences while the maximal penalty cost_max represents how different the subsequences could have been in the worst case scenario, the resulting degree of similarity may be calculated as follows:

$$similarity = 100 - \frac{d[length(s_1)][length(s_2)]}{\text{cost\_max}} \times 100 \quad (9)$$

4 Experiment

To demonstrate the capabilities of the described method, let us examine the comparison of two files that differ in structure but belong to the same family of malicious programs. The structural differences are explained by the peculiarities of the polymorphic packer used to protect malicious functionality. To detect these malware, Dr. Web Anti-virus uses a special procedure that analyses the functionality of an executable file. If particular evidence is found, the anti-virus reports the detection of BackDoor.Tdss.based.7. Use of the suggested method allows for the similarity of such files to be detected on the basis of their structural entropy only.

Figures 3 and 4 display entropy diagrams of the first and second file accordingly. The diagrams are built using the sliding window method. For illustration purposes, intervals with packed data are shortened. As a window size, we selected 256 bytes. Therefore, the maximum entropy level in one window can reach up to 8 bit (1). For a window shift, we use 128 byte. Such a selection means that our algorithm is effective on files of 2 and more megabytes.

First, let us examine the segmentation algorithm in respect to the first file (see Fig. 3). To understand the capability of wavelet analysis used for segmentation, see Fig. 2 which displays wavelet coefficients surface $W(a, b)$ built using DWT. For this transform, we used the Haar wavelet as a mother wavelet. Each point on the surface represents the compliance of the source data with the selected mother wavelet. In this figure, you can see that at larger transformation scales, insignificant changes in source data are ignored and vice versa; on the lower scales, there is more detail. Therefore, the maximal transformation scale determines the resulting number of segments, while the minimal scale is responsible for accuracy within the limits of obtained segments.

To compare two files, we need to apply the following parameters to our segmentation algorithm. Let us set the maximum transformation scale to 16, which would mean that the number of wavelet translators is 4, i.e., formula (3) will be computed four times. The threshold limit for determining significant wavelet coefficients will be set to 0.5, which means that we will ignore peaks of wavelet coefficients less than 0.5 in height (as in Fig. 2) when segmenting the file. As a result, when using this algorithm, we will receive segments whose borders are displayed at the top of the diagrams in Figs. 3 and 4.

In these diagrams, you may see differences in the structure of the selected files. In the first file, the 5th segment is located to the right of the 3th segment of the second file.

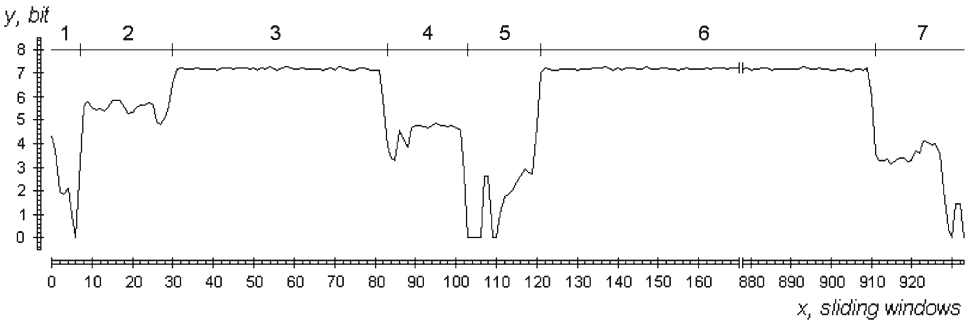

Fig. 4 Entropy diagram of the second file

Table 1 Cost array for comparing the segment sequences of two files

No.	0	1	2	3	4	5	6	7
0		0.254	0.662	1.179	1.570	1.946	2.816	3.224
1	0.254	0.000	0.409	0.926	1.317	1.693	2.562	2.971
2	0.637	0.384	0.083	0.600	0.991	1.367	2.237	2.645
3	1.027	0.774	0.473	0.991	0.605	0.982	1.851	2.260
4	1.237	0.984	0.683	1.200	0.815	1.192	2.061	2.470
5	1.759	1.506	1.205	0.704	1.094	1.470	2.340	2.748
6	2.193	1.940	1.639	1.138	1.528	1.501	2.370	2.423
7	3.066	2.813	2.512	2.010	2.401	2.374	1.523	1.931
8	3.469	3.216	2.915	2.413	2.803	2.706	1.926	1.541

This is explained by the fact that the polymorphic packer placed the compressed data in a different order: in the first file, these data are placed after the import section (3th segment), while in the second file, they are placed before the import section (4th segment). We should also note that in the first file, the segmentation algorithm singled out the area with zero entropy (4th segment). A similar area is also present in the second file (windows from 103 to 106), but it is shorter on one window and is placed to the left of the low entropy area. Therefore, the segmentation algorithm has not split the 5th segment of the second file. Other areas of the files have similar characteristics and equal positions.

After representing files as sequences of segments each of which is characterized by its size and averaged entropy, we apply our alignment algorithm. But first, we need to determine setting parameters. Let us set the coefficients from formula (6) as follows: $PART\_ENT = 0.6$, $PART\_SIZE = 1.4$. This will increase the effect of the difference in the size of the compared elements when calculating penalties. The parameter TAX will be set to 0.3. This means that the cost function for two mismatching elements will return 0.3 on average. After that, let us fill in array d (see Table 1). Note that column 0 in this table represents the cost of deleting all elements from the first sequence, while row 0 represents the cost of inserting all elements from the second sequence. In other words, in order to turn the first sequence into the second sequence, we need to delete all elements from the first sequence and then insert all elements from the second sequence.

Once we have filled in array d, let us examine the alignment procedure for two sequences (see Table 2). The procedure involves progressing through Table 1 from its lower right corner. Shifting to an element with the lowest cost, we determine one of the three edit operations. If the element with the lowest cost is located to the left of the current element, then it means that the current element from the second sequence is inserted. An example is the 4th segment of the second file. If we shift vertically, then it indicates that the element from the first sequence is deleted. Other examples are the 3- and 4th segments of the first file. If we shift diagonally, then it indicates the substitution of the corresponding elements.

As a result, to determine the degree of similarity between two sequences, we need to compare the real penalty to the maximum one. By real penalty, we understand this to be the resulting penalty received after filling in array d (Table 1). The maximum penalty represents how different the sequences could have been. The maximum penalty is calculated after alignment is completed in order to increase the penalty when comparing similar elements. For instance, if we look at the first segments of the files in our example, we will see in Table 1 that the deletion or insertion of each of them receives a penalty of 0.254. That means that the maximum penalty after an artificial increase will be 0.508. Such penalty will push away the compared sequences, though we can see that both segments are very alike. Therefore, we increase the maximal penalty and, taking into account formula (8), obtain the value of 1.014. So, when we determine

Table 2 Alignment of the segment sequences of two files

#	Number of windows	Entropy (bit)	#	Number of windows	Entropy (bit)	Real penalty	Max. penalty
1	7	2.2121	1	7	2.1520	0.000	1.014
2	19	5.7247	2	23	5.4067	0.083	2.598
3	20	4.1126			0.473	2.989	
4	5	0			0.683	3.198	
5	55	7.0767	3	53	7.1456	0.704	5.277
			4	20	4.3738	1.094	5.667
6	28	2.8067	5	18	1.6753	1.501	7.289
7	810	7.1768	6	790	7.1740	1.523	10.773
8	22	2.8163	7	23	2.8536	1.541	12.395

the share of the real penalty in the maximum one (9), we determine the degree of similarity between two sequences. In our example it equals 87.565%.

5 Conclusion

The proposed solution has a number of advantages that help detect malicious programs efficiently on personal computers. First, this comparison does not take into account the functionality of analysed files and is based solely on determining the similarity in code and data area positions. Therefore, the algorithm is effective against many ways of protecting executable code. On the other hand, such a comparison may result in false alarms. Therefore, our solution is useful as a preliminary test that triggers the running of additional checks. Second, the method is relatively easy to implement and does not require code disassembly or emulation. And, third, the malicious file record is compact which is significant when compiling anti-virus databases.

References

1. Breitenbacher, Z.: Entropy based detection of polymorphic malware. In: Proceedings of the 19th Annual EICAR Conference "ICT Security: Quo Vadis?", pp. 117–128. Presses Techniques de l'ESIEA, Paris (2010)
2. Daubechies, I.: Desjat' lektsij po vejvletam. [Ten lectures on wavelets]. Izhevsk: NIC Regular and Chaotic Dynamics (2001)
3. Ebringer, R., Sun, L., Boztas, S.: A fast randomness test that preserves local detail. In: Proceedings of the Virus Bulletin (VB) Conference, pp. 34–42. Virus Bulletin, Abingdon (2008)
4. Fabjanski K., Kruk T. (2008) Network traffic classification by common subsequence finding. In: Bubak M., van Albada G., Sloot P. (eds.) Computational Science—ICCS 2008, vol. 5101, pp. 499–508. Springer, Berlin
5. Gheorghescu, M.: An automated virus classification system. In: Proceedings of the Virus Bulletin (VB) Conference, pp. 294–300. Virus Bulletin, Abingdon (2005)
6. Kreibich, C., Crowcroft, J.: Efficient sequence alignment of network traffic. In: Proceedings of Internet Measurement Conference, pp. 307–312. IMC, Melbourne (2006)
7. Li, J., Xu, J., Xu, M., Zhao, H., Zheng, N.: Malware obfuscation measuring via evolutionary similarity. In: Proceedings of the International Conference on Future Information Networks, pp. 197–200. IEEE Computer Society, Los Alamitos (2009)
8. Lyda, R., Hamrock, J.: Using entropy analysis to find encrypted and packed malware. IEEE Security Priv. **5**(2), 40–45 (2007)
9. Newsome, J., Karp, B., Song, D.: Polygraph: Automatically generating signatures for polymorphic worms. In: Proceedings of the 2005 IEEE Symposium on Security and Privacy, pp. 226–241. IEEE Computer Society, Los Alamitos (2005)
10. Perdisci, R., Lanzi, A., Lee, W.: Classification of packed executables for accurate computer virus detection. Pattern Recognit. Lett. **29**(14), 1941–1946 (2008)
11. Sun, L., Versteeg, S., Boztas, S., Yann, T.: Pattern recognition techniques for the classification of malware packers. In: Proceedings of the 15th Australian Conference on Information Security and Privacy (pp. 370–390). Springer, Berlin (2010)
12. Sung, A.H., Xu, J., Chavez, P., Mukkamala, S.: Static analyzer of vicious executables (SAVE). In: Proceedings of the 20th Annual Computer Security Applications Conference, pp. 326–334. IEEE Computer Society, Washington (2004)
13. Wagener, G., State, R., Dulaunoy, A.: Malware behaviour analysis, extended version. J. Comput. Virol. **4**(4), 279–287 (2007)
14. Christodorescu, M., Jha, S.: Testing malware detectors. In: Proceedings of the ACM SIGSOFT International Symposium on Software Testing and Analysis, pp. 34–44. ACM, New York (2004)
15. Jacob, G., Neugschwandtner, M., Comparetti, P.M., Kruegel, C., Vigna, G.: A static, packer-agnostic filter to detect similar malware samples. Department of Computer Science University of California Santa Barbara Technical Report, 2010–26. Retrieved 29 November 2010 from http://www.cs.ucsb.edu/research/tech_reports/ (2010)
16. Wagner, R.A., Fischer, M.J.: The string-to-string correction problem. J. ACM **21**(1), 168–173 (1974)
17. Prangišvili, I.V.: Èntropijnye i drugie sistemnye zakonomernosti. Voprosy upravlenija složnymi sistemami (Entropy and other system laws. Issues of managing complex systems). p. 432. Nauka, Moscow (2003)