

# Evaluation of Malware clustering based on its dynamic behaviour

Ibai Gurrutxaga, Olatz Arbelaitz, Jesús M<sup>a</sup> Pérez, Javier Muguerza, José I. Martín, Iñigo Perona

Dept. of Computer Architecture and Technology  
University of the Basque Country  
M. Lardizabal, 1, 20018 Donostia, Spain

{i.gurrutxaga, olatz.arbelaitz, txus.perez, j.muguerza, j.martin, inigo.perona}@ehu.es

## Abstract

Malware detection is an important problem today. New malware appears every day and in order to be able to detect it, it is important to recognize families of existing malware. Data mining techniques will be very helpful in this context; concretely unsupervised learning methods will be adequate. This work presents a comparison of the behaviour of two representations for malware executables, a set of twelve distances for comparing them, and three variants of the hierarchical agglomerative clustering algorithm when used to capture the structure of different malware families and subfamilies. We propose a way the comparison can be done in an unsupervised learning environment. There are different conclusions we can draw from the whole work. Concerning to algorithms, the best option is average-linkage; this option seems to capture better the structure represented by the distance. The evaluation of the distances is more complex but some of them can be discarded because they behave clearly worse than the rest of the distances, and the group of distances behaving the best can be identified; the computational cost analysis can help when selecting the most convenient one.

*Keywords:* malware, hierarchical clustering, representation based on dynamic behaviour.

## 1 Introduction

Many of the most visible and serious problems facing the Internet today are related to malicious software and tools.

Malicious code, commonly called malware is software designed to infiltrate or damage a computer system without the owner's informed consent. It is considered malware based on the perceived intent of the creator rather than any particular features. Malware is generally the source of spam, phishing, denial of service attacks, botnets, and worms. New malware appears every day. In recent years, the first real viruses and worms for MacOS as well as Trojans for the J2ME mobile platform appeared. These last were designed to steal money from mobile user accounts.

The trends in malware evolution continue: Trojans are far more numerous than Worms, and the number of new malicious programs designed to inflict financial damage

have increased. As a consequence, the effect of malware has great economical cost.

Modern malware is very complex; many variants of the same virus with different abilities appear every day which makes the problem more difficult. Understanding this process and how attackers use the backdoors, key loggers, password stealers and other malware functions is becoming an increasingly difficult and important problem.

At this stage, the classification of new malware by human analysis, where memorization, looking up description libraries or searching sample collections is required, is not effective (Lee, and Mody 2006); it is too time consuming and subjective. As a consequence, automated and robust approaches to understanding malware are required in order to successfully face the problem. These processes will consist on representing and storing the knowledge in an adequate manner and learning from the stored information.

The representation used by antivirus focuses primarily on content-based signatures, which are inherently susceptible to inaccuracies due to polymorphic and metamorphic techniques. Their detection is based on static analysis, that is to say they represent the malware based on the structural information of a file. This kind of analysis fails to detect inter-component/system interaction information and would difficultly manage to detect malware created by collaboratively working packages (Bailey, Oberheide, Mao, Jahanian, and Nazario 2007). Code and data obfuscation also poses considerable challenges to static analysis.

The mentioned problems suggest that the use of data from runtime analysis, the collection of system calls (e.g., files written, processes created), is more invariant and directly useful than abstract code sequences. The dynamic behaviour can be directly used in assessing the potential damage malware can cause and it will enable detection and classification of new threats.

Clustering malware based on the dynamic analysis approach based on the execution of malware in controlled real environments can be very helpful for anti-malware professionals to address the limitations of existing automated classification and analysis tools and better analyze the new malware appearing every day. New malware could be assigned to existing clusters or malware families, so variations of known malware will be easily recognized. Clustering malware is not an evident task; many decisions have to be made:

1. Data needs to be collected, represented and stored so that the smallest amount of information is lost.

2. Distances for comparing the stored data need to be selected.
3. Finally, a clustering algorithm needs to be selected to be used with the chosen data representation and distance.

The idea of the use of the dynamic behaviour has been applied in other security areas. For example, papers applying this idea can be found in Gao, Reiter, and Son 2005, Yeung, and Yuxin 2003, Brugger 2004, Christodorescu, Jha, and Kruegel 2007; or in commercial systems such as Norman (Norman Solutions 2003), and CWSandbox (Willems, and Holz 2007). However, few works have been done based on the dynamic behaviour of malware. For example, Bailey, Oberheide, and Mao 2007 cluster malware and evaluate the obtained results based on 5 antivirus' output, and in Lee, and Mody 2006 malware is clustered based on an opaque representation, edit distance and k-medoid clustering algorithm, but not comparison of different strategies is presented.

In this sense, no analysis has been done on which are the best representation, distance functions and clustering algorithms to use.

The aim of this work is to make the analysis of the behaviour of two representations, a wide set of distances and some clustering algorithms when used to clustering malware based on its dynamic behaviour. This is not an easy task in an unsupervised context but we have designed an adequate experimentation and an evaluation methodology that can help doing this work properly.

As we have mentioned, lots of new malware are generated every day. In this context, the detection of malware needs to be efficient. This has two main consequences: on the one hand the used representations will be as simple as possible as far as they do not lose too much information. On the other hand, when evaluating the distances and algorithms, added to the achieved accuracy, the computational cost will also be an important feature to be taken into account.

The paper proceeds describing in Section 2 the different options for clustering malware we have evaluated, how data has been processed, the used representations, distances and algorithms. Section 3 is devoted to describe the details about the experimental methodology for evaluating distances on the one hand, and algorithms on the other one. In Section 4 we present an analysis of the experimental results where comparison of distances and algorithms are shown, together with an analysis of the computational cost. Finally Section 5 is devoted to show the conclusions and further work.

## 2 Malware analysis tools

As we mentioned in the introduction, the process of automatically clustering malware and later detecting new attacks is divided in three main steps: data needs to be collected, represented and stored; distances for comparing the stored data need to be selected, and finally, a clustering algorithm needs to be chosen to be used with the selected data representation and distance.

This section is devoted to describe the options that will be evaluated for each one of the processes in this work.

### 2.1 Data processing

In any data mining process, the starting point is the data collection, representation and storage process. This work is also part of the data mining process and, as a consequence, to be able to cluster malware based on dynamic analysis of code, it is compulsory to get some data.

The data used in this work has been provided by S21sec lab a leading security company in our environment. It has been generated in a real environment; no virtual machines have been used in the process. Malware code has been executed in monitorized machines; Pentium IV of 3.20 GHz with 1 GB RAM and Windows XP SP2 operating system. The information we have about the kind of malware we are working with is the output of Kaspersky antivirus. Some examples of the treated malware (based on the antivirus) are: Backdoor.Win32.Rbot, Trojan.Win32.Agent, Net-Worm.Win32.Mytob, SpamTool.Win32.Agent, Email-Worm.Win32.Mydoom ...

The code of the malware has been executed during one minute in a controlled environment and information about the system calls generated during this period has been collected. The security experts in S21sec company have decided to distribute the detected system calls in 45 groups (events), each one with its corresponding code; Table 1 shows an example of event codification.

Code	Event
1	File create
2	File open
...	...
44	Reg. Key Set Val.
45	Network Connect

**Table 1: Example of event codification**

The information stored for each of the executions is a sequence of events, that is to say, a list of codes, corresponding to the events detected during the minute the monitorization lasted. An example could be: <2, 44, 23, 23, 11, 16, 2, 16, 16, 16>.

This representation has been chosen because we find it adequate to represent the dynamic behaviour of the malware. It allows storing information related to the kind of the generated system calls, and the order the actions took place. This is a double edged sword; on the one hand, some meaningful information is stored, but, on the other hand, when using this data for clustering malware, sequence comparison methods will be required to work with it, which are in general very time consuming.

Simpler representations (vector representations) of the data can also be used; these methods will have some disadvantages, they will in general miss information, the information about the order of the events disappears, but on the contrary, the comparison strategies will be faster.

In this sense, in order to evaluate the accuracy/computational cost trade-off we have also worked with a projection vector of the stored data where the projection of the information stored as event sequences is done by representing the frequency of the different events (each one of the 45). This option has been called Count. As an example, the vector projection

obtained for the previous example (<2, 44, 23, 23, 11, 16, 2, 16, 16, 16>) is represented in Table 2.

Event	Frequency
1	0
2	2
...	...
11	1
...	...
16	4
...	...
23	2
...	...
44	1
45	0

**Table 2: Example of the vector projection called Count**

## 2.2 Tools for comparing the stored data

The aim of this work is the evaluation of representations, distances and clustering methods and this subsection is devoted to describe the distances used for comparing examples. The kind of distances that can be used depends on the used representation. To work with sequences, different distances have been evaluated, whereas in the case of vectorized data, Count, the Euclidean distance has been used as distance. Added to those options, some special cases have been studied in order to use them as baseline for the comparison.

### 2.2.1 Distance and similarity functions for comparing sequences

When the chosen representation has been event sequences, the comparison could not be done using the same distances used to compare vector representations. Thus, distances for sequence comparisons need to be selected. The experimentation has been done with four well known distances and some variants (normalized options). The used distances are described in the following paragraphs.

1. Edit Distance (ED): the edit distance between two strings of characters is the minimum number of edit operations on individual characters needed to transform one string to the other (Gusfield 1997). The permitted edit operations are the insertion, deletion and substitution (replacement) of a character.
2. Longest Common Substring (LCSg): the length of the longest common substring existing between two strings of characters. A substring common to two strings must consist of contiguous characters in both strings.
3. Longest Common Subsequence (LCSq): the length of the longest common subsequence existing between two strings of characters. A subsequence common to two strings needs not to consist of contiguous characters in any of the strings but the characters must be in the same order.

4. Normalized Compressed Distance (NCD): distance based on compression techniques to compare two character sequences (Li, Chen, Li, Ma, and Vitanyi 2004, Wehner 2005). NCD is a Kolmogorov complexity based metric. Since in practice Kolmogorov complexity cannot be computed directly, it is approximated with real compression algorithms.  $C(x)$  is defined as the length of the string obtained compressing  $x$ , and  $C(xy)$  as the length of the string obtained compressing the concatenation of  $x$  and  $y$ . The Normalized Compression distance is defined as follows:

$$NCD(x,y) = \{C(xy) - \min(C(x),C(y))\} / \max(C(x),C(y)).$$

### 2.2.2 Normalized option

Some of the options such as ED and Count will be very dependent on the sequence length. In fact we think that the use of these distances will be very similar to the comparison of sequences by their length. As a consequence, on the one hand we have added the normalized options: EDN, LCSgN and LCSqN, dividing the corresponding value with the length of the shortest sequence in the comparison, and EDX, LCSgX and LCSqX dividing the corresponding value with the length of the longest sequence in the comparison.

The vector projection has also been normalized in two different ways. In CountL option the frequency of each variable is normalized dividing it with the length of the sequence (relative frequency). In order to obtain CountM normalization, for each variable  $i$  the mean,  $m_i$ , and the standard deviation,  $s_i$ , in the whole database are computed. Each value  $x_{ij}$  is replaced by  $(x_{ij} - m_i) / s_i$ .

Notice that LCS functions are similarity functions, and, as a consequence, to be used in the clustering algorithms, they need to be converted to distances. The conversion has been done after the normalization process. The similarity value  $S$  has been converted into  $D=1-S$  distance value.

### 2.2.3 Special case

Added to the mentioned options, two simple representations have been used to confirm some suspicions, on the one hand, and as baseline, on the other one.

1. Length: each sequence is represented by its length and the used distance is the Euclidean one. This option has been used in order to compare results with the not normalized options of ED and Count and confirm that their behaviour is similar.
2. Random: a random distance value has been generated for each compared pair of sequences. This will be helpful to show whether the rest of the distances supply any information or not.

## 2.3 Selected algorithms

The aim of this work is to evaluate how different strategies work when clustering malware. We could say malware has a hierarchical structure (families, subfamilies, variants, types...). Taking into account this structure, we have selected hierarchical clustering algorithms for the experimentation. The results that can

be obtained from them go further than just some malware families; they can provide a hierarchy of the malware.

This work compares three hierarchical clustering algorithms (Jain, and Dubes 1988, Sneath, and Sokal 1973, Mirkin 2005), concretely three options of the agglomerative clustering algorithm differentiated by the method used for calculating distances between clusters.

In agglomerative hierarchical clustering, initially all instances are located in individual clusters and in each iteration the nearest clusters are merged. The process finishes when only one cluster containing all the instances is left. In order to compute the distance between two clusters we will use three of the most used definitions.

1. Single-linkage (or nearest neighbour): the distance between two clusters is computed as the distance between the two closest elements in the two clusters.
2. Complete-linkage (or furthest neighbour): the distance between two clusters is computed as the distance between the two furthest elements in the two clusters.
3. Average-linkage (or group average): the distance between two clusters is computed as the average distance between all the pairs of elements each one from a different cluster.

The output of these algorithms is a cluster hierarchy which is suitable to be represented in a graphical way. The most usual procedure is to draw them as a tree diagram called dendrogram. The root node of this tree contains the topmost partition (the cluster containing all the data points in the dataset) while the leaf nodes contain the partition at the lower level of the hierarchy where all the data points are usually clustered in singleton clusters.

### 3 Experimental methodology

The security company S21sec provided us with a database that contains 500 malware executions. As a consequence, the data we will use for this work are 500 event sequences, each one belonging to a different malware execution. Since the one minute execution is not always resulting in the same amount of relevant system calls, the sequences contain different number of events;. This length goes from 1 to 12,944 where the mean value is 1,008 and the standard deviation 2,173. The dataset has been randomly divided in 5 parts of 100 sequences and the same experimentation has been repeated with each one of them. This methodology allows proving that the obtained results are not biased to a particular sample.

One of the out comings of this work will be to propose an evaluation methodology that includes a meta-learning step for such an environment. When designing the experimentation, we have to take into account two aspects. On the one hand, we want to evaluate to what extent different distances are able to mark differences between malware examples belonging to different families and, on the other hand, we want to evaluate how the structure represented by these distances is captured by different clustering algorithms. The methodology has been designed having into account that the work has to be done in an unsupervised learning environment: that is to say, the relation existing between the 500 malware examples is unknown.

### 3.1 Methodology for evaluating distances

The first step has been to build a distance matrix for the malware examples to be clustered with each one of the 12 distances (Count, CountL, CountM, ED, EDN, EDX, LCSgN, LCSgX, LCSqN, LCSqX, Length, NCD and Random) to be compared.

In order to compare the 12 distance matrices, the correlation between them has been calculated. This will help to evaluate the strength and direction of the relationship existing between the different distances. The best known correlation coefficient is the Pearson product-moment correlation coefficient (Jain, and Dubes 1988), which is obtained by dividing the covariance of the two variables by the product of their standard deviations. This information has been used to try to evaluate differences and relationships between the evaluated distances.

### 3.2 Methodology for evaluating algorithms

The second objective is to evaluate how the three options for hierarchical clustering capture the structure represented by the distances.

The first step in this part has been to use the distance matrices to build a dendrogram with each one of the three algorithms we mean to evaluate, that is to say, they will be used to cluster malware based on different distances and algorithms. In order to evaluate to what extent the algorithms are able to capture the structure represented by each one of the distance matrices, the correlation between the distance matrix and the cophenetic matrix of the corresponding dendrogram has been calculated. The cophenetic matrix is the matrix of values  $[dc(xi,xj)]$  where  $dc(xi,xj)$  is the cophenetic proximity measure: the level in the dendrogram at which objects  $xi$  and  $xj$  are first included in the same cluster. The closer the cophenetic matrix and the given distance matrix are, the better the hierarchy fits the data (Jain, and Dubes 1988, Halkidi, Batistakis, and Vazirgiannis 2001).

Another important clue when evaluating distances is the computational cost. As we mentioned in the introduction, new malware appears every day and this malware will also need to be classed into a family. As a consequence a time consuming system would not be adequate.

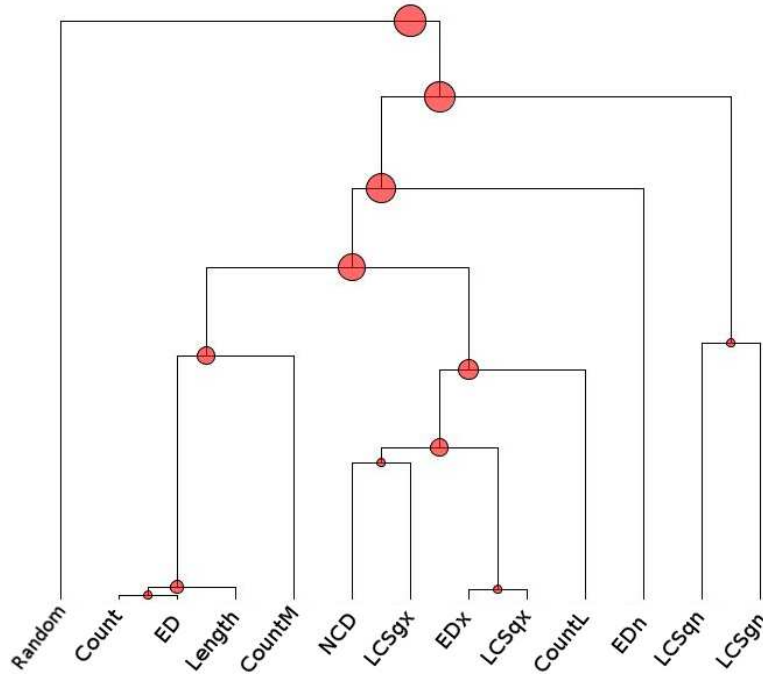
## 4 Results

We have executed many runs with different data of, 100, 200, 300, 400 and 500 sequences. The general conclusions have been similar in all of them and we will therefore show results for the first run executed with a sample of 100 sequences. In the section devoted to computational cost analysis execution times for incremental samples of 100, 200, 300, 400 and 500 are also shown.

### 4.1 Evaluation of distances

The values in Table 3 represent the correlation values between all the possible pairs of distance matrices (12\*12). Just the lower diagonal matrix is shown because the complete matrix would be symmetrical. A pair wise comparison has been done and the obtained values could be used to decide how similar or different the structure captured by different pairs of distances is.





**Figure 1: Dendrogram obtained from the correlation matrix converted into distance matrix, using the agglomerative clustering algorithm with average linkage**

The evaluation procedure has been divided in three steps for each of the evaluated algorithms:

1. Based on the corresponding distance matrix, a dendrogram has been built for each one of the 12 distances.
2. The cophenetic matrix of each dendrogram has been calculated.
3. The correlation between the distance matrix and the corresponding dendrogram (the cophenetic matrix obtained from the dendrogram) has been calculated.

Table 4 helps to identify how well the different options for agglomerative hierarchical clustering behave with each one of the distances.

	Average	Single	Complete
Count	0.99	0.97	0.99
CountL	0.88	0.62	0.76
CountM	0.98	0.97	0.95
ED	0.99	0.98	0.99
EDN	0.34	0.33	0.34
EDX	0.95	0.82	0.90
LCSgN	0.50	0.34	0.33
LCSgX	0.97	0.89	0.90
LCSqN	0.48	0.00	0.36
LCSqX	0.95	0.81	0.91
Length	0.97	0.95	0.98
NCD	0.92	0.83	0.81
Random	0.25	0.07	0.22
Mean	0.83	0.71	0.77

**Table 4: Correlation values for the distance matrices and the corresponding cophenetic matrices obtained based on three different clustering algorithms**

There is no doubt independently of the used distance: the best algorithm is average-linkage. It is the algorithm achieving the greatest correlation value, not only in average but for each one of the evaluated distances.

This study also shows that the Random distance is different to the others because none of the compared algorithms can obtain a representative dendrogram.

Apart from the evaluation of different algorithms, the information in Table 4 helps to make some discrimination between distances. The values in the table show that hierarchical agglomerative algorithms can hardly represent the structure of the distances normalized using the shortest length: EDN, LDSgN and LCSqN are by far the distances obtaining the smallest values for the three options being these values under 0.5. Besides, these distance functions have shown the greater instability in the 5 runs executed. We could say that for the rest of the distances the clustering algorithm is able to extract the structure of the distance matrix and besides, if the used option is average linkage, the correlation between the distance matrix and the cophenetic matrix is in average 0.96.

### 4.3 Computational cost

As we mentioned, having into account that malware detection needs to be done efficiently, and thousands of new malware appear every day, another important issue to be taken into account when designing a malware detection tool is the execution time.

Obviously, fastness will be important to discriminate distances when they present similar behaviour but it won't be the only desired feature of a distance. For instance, even if Length and Random are by far the fastest options, they would never been chosen. We have used them just as baseline.

When evaluating the obtained results, it has to be taken into account, that apart from the used distance, the only issue affecting to the time used to build the distance matrix is not the number of examples of the sample; the number of events of each of the examples will have in some cases a lot of influence. This influence will be greater in ED, LCSq and LCSg distances.

Next table (Table 5) shows the time required to build the distance matrices for the evaluated distances in milliseconds. The executions have been done in a Pentium IV, 3.2 GHz, 1G RAM. No times are shown for normalized distances because they will be similar to the original ones. We can observe in the table that there are huge differences between the times required to obtain the distance matrix for different algorithms.

Distance function	Time (ms)	speedup
ED	1,102,673	1
LCSq	63,648	17
LCSg	33,782	33
NCD	2,716	406
Count	347	3,178

**Table 5: Time required building the distance matrices in milliseconds. The speedup column shows the speedup of the rest of the distances in respect to Edit Distance (ED)**

As expected, the values in the table show that, in general, sequence comparisons take longer than vector projections. On the other hand, we should remember that sequence representation has semantically more meaning. Anyway, the differences between the distances applied to sequences are also very large. ED is the most expensive algorithm and the required time is significantly larger than the next couple: LCSq and LCSg, subsequence and substring comparison. The time spent by the last distance for sequences, NCD, is an order of magnitude smaller than LCSg. This time is reduced again an order of magnitude when the used distance is Count, that is to say, when the vector projection option is combined with the Euclidean distance.

The use of the 5 samples of 100 executables has given us the possibility to make an incremental analysis of the required execution time. Next table shows the time required to obtain distance matrices for the evaluated distances when the size of the sample varies from 100 to 500, 100 by 100. When analyzing these results, it has to be taken into account that even if the number of sequences increases gradually, due to the different length of the sequences, the number of events does not and this will also affect to the obtained times.

The second row in Table 6 shows the total number of events (calculated adding the number of events in each one of the sequences in the sample) of the samples used for the experimentation. No prior predictions can be done; it could happen that different options fit better to different distances, or the same option being always the one fitting the best. Table 4 shows the results for the evaluated distances and algorithms. The values for Random have not been included when calculating the mean (last row in the table).

Sequences	100	200	300	400	500
num. events	85,602	228,501	341,418	434,764	504,223
ED	1,102,673	8,298,168	19,195,187	31,030,384	42,197,967
LCSq	63,648	463,660	1,033,027	1,695,701	2,258,664
LCSg	33,782	246,621	563,657	912,167	1,231,900
NCD	2,716	11,728	26,071	45,657	64,564
Count	347	760	884	931	1,308

**Table 6: Time required to building the distance matrices for samples of 100, 200, 300, 400 and 500 examples in milliseconds**

We can observe that when computationally more expensive the calculation of a distance is, more affects the number of used sequences to how much time the process needs. This increment is more or less proportional to increase in the size of the sample for the NCD case, whereas it increments more than the proportion the sample increases for the rest of the sequence based distances.

## 5 Conclusions

Being aware of how important the automatic detection of new malware is, this work presents a methodology and a comparison of the behaviour of two representations, a set of twelve distances and three clustering algorithms when used to clustering malware based on its dynamic behaviour. All this work has been done in an unsupervised learning context and as a consequence the validation of the results is a difficult task. In this context, we propose a way this comparison can be done, adding a metalearning step to obtain more semantic information from the distance comparison part.

There are different conclusions we can draw from the whole work. Concerning to algorithms, there is not doubt average-linkage option behaves better than single linkage and complete linkage. This option works better independently of the used distance; it seems to capture better the structure represented by the distance.

The evaluation of the distances is more complex. There are huge variations from the two points of view: the structure they capture and the computational cost. Even if the choice of the best distance is not evident some of them, such as Length, ED, Count, CountM, EDN, LCSqN and LCSgN, can be discarded because they behave clearly worse than the rest of the distances. On the other hand, being the behaviour of EDX similar to the one of LCSqX, we should never use the first option because of the computational cost.

We concluded that the most interesting group is formed by CountL, NCD, LCSgX and LCSqX distances. Their distance matrices are highly correlated, and the clustering algorithms seem to be able to capture their structure. This could be somehow surprising. We would expect the first one, CountL, to behave the worst because it is based on the vector projection, whereas the rest are based on the sequence representation. Evidently from the computational cost point of view, CountL is by far the fastest option.

In order to evaluate the differences of the dendrograms built with CountL, NCD, LCSgX and LCSqX, a

preliminary analysis of the obtained trees has been done. We could observe that all the dendrograms were similar in their lower parts but differences arise at higher levels, when heterogeneous clusters are forced to merge. This suggests that the interesting part of the tree is practically the same for the four distances. As a consequence, after this analysis the best strategy to use would be the normalized vector projection CountL combined with average linkage option for the agglomerative clustering algorithm. However, we consider that a validation done by security experts or the use of labelled (totally or partially) data would help to clarify the differences among these four distances and choose the best one.

As a consequence, this work could be extended in different senses, on the one hand, due to the great amount of new malware appearing lately, it would be interesting to extend the work to new greater samples and more distances and algorithms could be evaluated using the same methodology. On the other hand, we used just system call sequences to represent malware, but other information such as file numbers or registry key names could be included in order to evaluating the trade-off between the obtained improvement and the efficiency. Besides, this new representation of malware will provide more detailed descriptions of the malware's dynamic behaviour that could be compared to the descriptions provided by known systems such as Norman (Norman Solutions 2003) and CWSandbox (Willems, and Holz 2007). As the reader will have realized, this work has been done using only malware code executions and it would be also interesting to get track of normal code executions and compare their behaviour. Finally, we should try to obtain feedback from security experts in order to confirm the obtained results.

## 6 Acknowledgements

The authors are grateful to the company S21sec labs for providing the dataset used in this paper and to their security experts for the help they offered to understand and identify the problem. This work has been (partially) supported by the GISD project (FIT-360001-2006-6) under the PROFIT-proyectos-tractores program from the Spanish Ministry of industry, tourism and trade.

This work was partly funded by the Diputación Foral de Gipuzkoa and the E.U.

## 7 References

- Lee T., Mody J.J. (2006): Behavioral classification. *Proc. of the EICAR*.
- Bailey M., Oberheide J., Mao Z.M., Jahanian F., Nazario J. (2007): Automated classification and analysis of internet malware. *Proc. of the 10th Symposium on Recent Advances in Intrusion Detection*, 178–197.
- Gao D., Reiter M. K., Song D. X. (2005): Behavioral distance for intrusion detection. *Proc. of the 10th Symposium on Recent Advances in Intrusion Detection*, 63-81.
- Yeung D., Yuxin D. (2003): Host-based intrusion detection using dynamic and static behavioral models. *Pattern Recognition* 36:229-243.
- Brugger T. (2004): Data mining methods for network intrusion detection. Technical Report. University of California at Davis.
- Christodorescu M., Jha S., Kruegel C. (2007): Mining specifications of malicious behavior. *ESEC/SIGSOFT FSE* 5-14.
- Norman Solutions (2003): Norman sandbox whitepaper. [http://download.norman.no/whitepapers/whitepaper\\_Norman\\_SandBox.pdf](http://download.norman.no/whitepapers/whitepaper_Norman_SandBox.pdf). Accessed 29 Sep 2008.
- Willems C., Holz T.: CWSandbox, <http://www.cwsandbox.org/>, Accessed 29 Sep 2008.
- Gusfield D. (1997): *Algorithms on strings, trees, and sequences*. Cambridge University Press.
- Li M., Chen X., Li X., Ma B., Vitanyi P.M.B. (2004): The similarity metric. *IEEE Transactions on Information Theory* 50:3250-3264.
- Wehner S. (2005): Analyzing worms and network traffic using compression. Technical Report. CWI, National research institute for mathematics and computer science in the Netherlands.
- Jain A.K., Dubes R.C. (1998): *Algorithms for clustering data*. Prentice-Hall.
- Sneath P.H.A., Sokal R.R. (1973): *Numerical taxonomy*. Freeman W.H. (eds). San Francisco.
- Mirkin, B. (2005): *Clustering for data mining: a data recovery approach*. Chapman & Hall/CRC.
- Halkidi M., Batistakis Y., Vazirgiannis M. (2001): On clustering validation techniques. *Journal of Intelligent Information Systems* 17:107-145.