Methodology for Behavioral-based Malware Analysis and Detection using Random Projections and K-Nearest Neighbors Classifiers

Jozsef Hegedus, Yoan Miche, Alexander Ilin and Amaury Lendasse
Department of Information and Computer Science,
Aalto University School of Science,
FI-00076 Aalto, Finland
Email: firstname.lastname@aalto.fi

***Abstract*—In this paper, a two-stage methodology to analyze and detect behavioral-based malware is presented. In the first stage, a random projection is decreasing the variable dimensionality of the problem and is simultaneously reducing the computational time of the classification task by several orders of magnitude. In the second stage, a modified K-Nearest Neighbors classifier is used with VirusTotal labeling of the file samples. This methodology is applied to a large number of file samples provided by F-Secure Corporation, for which a dynamic feature has been extracted during DeepGuard sandbox execution. As a result, the files classified as false negatives are used to detect possible malware that were not detected in the first place by VirusTotal. The reduced number of selected false negatives allows the manual inspection by a human expert.**

***Keywords*-malware detection; machine learning; random projections; k nearest neighbors;**

I. Introduction

Malware detection has been the subject of a large number of studies (see [1], [2], [3] and [4], [5], [6], [7], [8]), for example the work of Bailey [9] using signature-based malware detection approach has shown that recent malware types require additional information in order to obtain a good detection.

In this paper, an approach based on the extraction of dynamic features during sandbox execution is used, as suggested in [7]. In order to measure similarities between executable files, the Jaccard Index is used to measure the similarities between hash values (encoding the dynamic feature values obtained from the sandbox). The hash values are transformed into a large number of binary values which could be used to compute the Jaccard Index (see [10] for original work in French or [11] in English). Unfortunately, the dimensionality of such variable space does not allow the use of traditional classifiers in a reasonable computational time.

A two-stage methodology is proposed to circumvent this dimensionality problem. In the first stage, a random projection is decreasing the variable dimensionality of the problem and is simultaneously reducing the computational time by several orders of magnitude. In the second stage, a modified K-Nearest Neighbors classifier is used with VirusTotal [12] labeling of the file samples. This two-stage methodology is presented in section III.

The practical implementation of the methodology and the results are discussed in section IV. The different parameters (the random projection dimension and the number of nearest neighbors) are also analysed in this section.

As a global result, the methodology enables to identify the false negatives from the classification. Such samples can then be used to detect possible malware that were not detected in the first place by the VirusTotal labeling. Thanks to the methodology, the reduced number of identified false negatives allows for a manual inspection by a human expert.

Indeed, without this pruning of possibly malicious samples by the presented methodology, a manual inspection will not be possible since reliable experts are scarce and their availability is highly limited.

Using the proposed methodology and the know-how of one F-Secure Corporation expert, it has been possible to extract 24 malware candidates out of 2441 original candidates from which 25% are surely malicious and 50% which are probably malicious, have to be further investigated in order to obtain a decisive classification.

In section II, the data gathering and sample labeling are described. Section III presents the two-stage methodology while section IV shows the practical implementation, the results and the analysis of the results.

II. Behavioral Data Gathering and Sample labeling

The data set used in this paper is focused on behavior-based malware analysis and detection. The former approach of signature-based malware detection cannot be considered as sufficient anymore for reliable detection [9], [7]. Be it because of the development of polymorphic and metamorphic malware or the approach of flash worms — who only do some reconnaissance on the machines/network they scan for future deployment of targeted attacks —, the need for execution level identification is important.

A. *Sandboxing and Extracting Behavioral Features*

In this spirit, a currently popular approach [7], [6] is to sandbox the execution of the malware and analyze behavioral data extracted during the execution.

It has recently been demonstrated in [8] that the use of public sandbox submission systems might reveal network information regarding the sandbox machine identity. Through submission of a decoy sample by an attacker, it becomes possible to blacklist the hosts on which are sandboxed the samples and have the malware circumvent the sandbox execution and forth detection.

The Norman sandbox development kit [13] released in 2009 enables security companies to gather the behavioral data obtained during sandboxed execution and analyze that data with a custom engine. This avoids the pitfall of a publicly available sandbox machine mentioned.

The results in this paper were obtained on the data of 32683 samples collected by F-Secure Corporation. The samples data were produced by F-Secure by running the samples through their sandbox engine [14], [15], [16], which resulted in large numbers of feature-value pairs extracted for each sample. Individual features may have significant number of distinct values, and the values come in the form of hashes. The data cannot be considered complete, as the sandbox, for instance, may not be able to run some of the samples correctly or may miss relevant execution paths.

The samples were labeled using an online sample analysis tool explained in the next section.

B. *Obtaining the Sample labeling*

The VirusTotal [12] online analysis tool provides a simple interface for sample submission, returning a list of up to 43 (depending on the sample nature: executable, archive…) mainstream anti-virus software detection results. Among the most widely used and known are F-Prot, F-Secure, ClamAV, Antivir, AVG, BitDefender, eSafe, Avast, McAffee, NOD32, Norman, Panda, Symantec, TrendMicro, VirusBuster… See the VirusTotal web site for the full list of used engines [12].

The result of the submission of a sample file is the number of engines which detected the sample as malware. Figure 1 is a histogram of the detection levels for the set of 32683 samples used in this paper. As can be seen, a large proportion of the set is detected by at least one engine as malware. Less than 2500 samples are actually not detected by any engine.

In order to make the problem a binary classification one (i.e. identifying whether a sample should be considered "malware" or "clean"), an *a priori* and arbitrary threshold has been set on the amount of engines detecting a sample as malware. It is considered that for a sample i, if the amount m_i of engines identifying the sample as malware is such that $0 < m_i < 11$, then the sample is discarded. The disadvantage is that these samples are not considered in the whole methodology and therefore not classified. Nevertheless, they have also no influence on the rest of the data set and the final classification results.

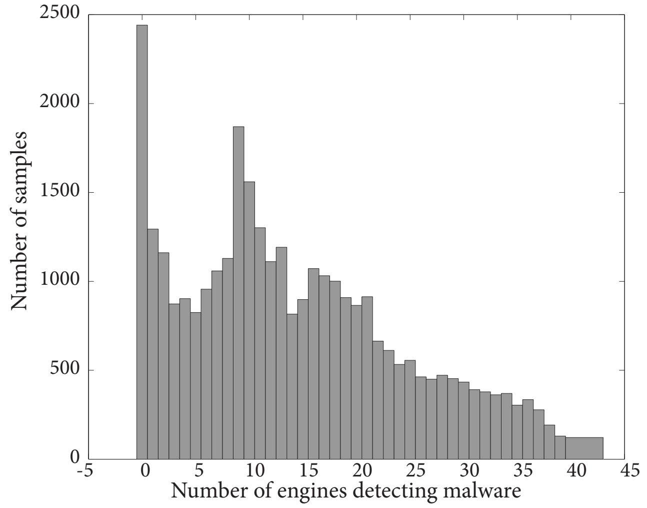

Figure 1. Histogram made using 32683 executable samples and querying from www.virustotal.com how many anti-virus engines raise a flag for each sample. Thus for each sample k a number m_k is obtained. For a given value x, on the x-axis, the y-axis shows for how many samples k it is true that $m_k = x$.

This is equivalent to setting a certainty threshold on the sample analysis, above which it can be considered as indeed malware (and no more a set of false positives from m_i different engines). Therefore, samples with a number m_i of detecting engines strictly above 10 are kept and considered as "malware" (with a relatively high probability), and samples with 0 detecting engines are kept and considered as "unpredictable" (and possibly "clean").

Figure 2 illustrates the pruned set of samples, with only samples for which $m_i = 0$ or $m_i > 10$ are kept, which amounts to 21053 (out of the original set of 32683): 18612 considered as "malware", and 2441 as "possibly clean".

It is clear that flagging the 2441 samples for which $m_i = 0$ as "possibly clean" is likely to hide a certain amount of false negatives (VirusTotal clearly states that $m_i = 0$ should in no way be considered as meaning "clean"). The "meta" goal of this paper is to actually identify such samples which are potential false negatives, using a methodology based on the Jaccard similarity [11], [10] measure and K-Nearest Neighbors classifiers.

III. Methodology

The overall process can be summarized by Figure 3, with the dynamic feature extraction described in the

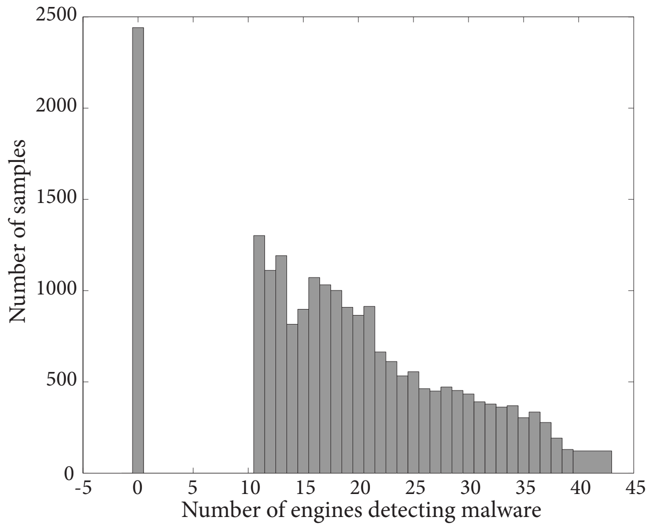

Figure 2. The m_k distribution for the samples used for this histogram is identical to Figure 1 with the important difference that samples such that $0 < m_i < 11$ are discarded. Here 2441 samples are depicted that can be considered as "clean" ($m_i = 0$), and 18612 samples that can be considered as malicious ($m_i > 10$).

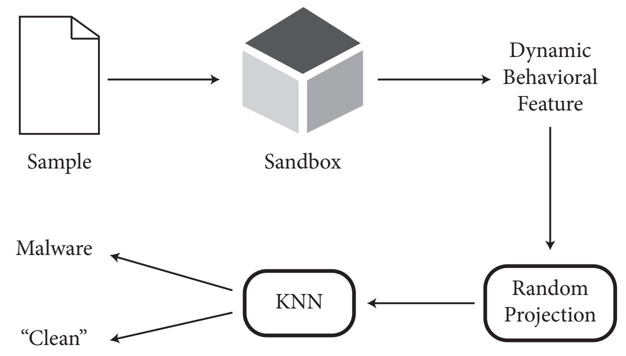

Figure 3. Global schematic of the methodology: a sample is run through the sandbox to obtain a set of dynamic features; the random projection approach then reduces the dimensionality of the problem while retaining most of the information conveyed by the original feature; finally, a K-Nearest Neighbors classifier in the random projection space gives prediction on the studied sample being malware or not.

previous section, followed by the actual methodology to identify potential false negatives, using a Random Projection approach and K-Nearest Neighbors classifiers (described in detail in sections III-C and III-B).

A. Measuring Similarity between Executables

In this section, an approach for measuring similarities between executables is detailed. Let A^i denote the set of hash values (produced by the sandbox) for file i.

Then, the $J_{Jaccard}$ Jaccard similarity between two executables i, i' is calculated as

$$J^{i,i'}_{Jaccard} = \frac{\left|A^i \cap A^{i'}\right|}{\left|A^i \cup A^{i'}\right|}. \tag{1}$$

Similarly, the J_{cosine} cosine similarity is given by

$$J^{i,i'}_{cosine} = \frac{\left|A^i \cap A^{i'}\right|}{\sqrt{\left|A^i\right| \left|A^{i'}\right|}}. \tag{2}$$

Note that the J_{cosine} cosine similarity is expressed as a scalar product.

Denote by

$$A = \bigcup_{i=1}^{N} A^i = \{a_1, a_2, ..., a_D\}, \tag{3}$$

where N is the total number of samples and D is the total number of unique hashes seen in all samples.

Then from an ordering of set A, N binary (0,1 valued) vectors $\underline{\mathbf{B}}^i$ can be constructed, each of K dimensions such that

$$\left|A^i \cap A^{i'}\right| = \langle \underline{\mathbf{B}}^i, \underline{\mathbf{B}}^{i'} \rangle, \tag{4}$$

and $\left|A^i\right| = \left\|\underline{\mathbf{B}}^i\right\|^2$. Here $\|\cdot\|$ denotes vector norm and $\langle \cdot, \cdot \rangle$ denotes scalar product. Since $\underline{\mathbf{B}}^i$ is a binary vector (with coordinates 0, 1 only), $\left\|\underline{\mathbf{B}}^i\right\|^2$ is the number of the coordinates in $\underline{\mathbf{B}}^i$ that are equal to 1.

So, the normalized scalar product of $\underline{\mathbf{B}}^i$ and $\underline{\mathbf{B}}^{i'}$ gives the cosine similarity:

$$J^{i,i'}_{cosine} = \frac{\langle \underline{\mathbf{B}}^i, \underline{\mathbf{B}}^{i'} \rangle}{\left\|\underline{\mathbf{B}}^i\right\| \cdot \left\|\underline{\mathbf{B}}^{i'}\right\|}. \tag{5}$$

Using the relationship between Euclidean distance

$$D_{euclidean} = \left\|\underline{\mathbf{B}}^i - \underline{\mathbf{B}}^{i'}\right\| \tag{6}$$

and cosine similarity in the case of $\left\|\underline{\mathbf{B}}^i\right\| = 1$ and $\left\|\underline{\mathbf{B}}^{i'}\right\| = 1$, it appears that

$$J_{cosine} = \frac{2 - D^2_{euclidean}}{2}. \tag{7}$$

From Equation 7 it appears that a classification or clustering based either on the cosine similarity or on the Euclidean distance will yield the same result if the norm of the feature vectors is unity.

B. K-Nearest Neighbor Classification

In this section, a standard method (K-NN, see for example [17], [18], [19], [20]) is described; it can be used to predict whether an unknown executable is malicious or benign. The essential assumption of the method is that malicious (resp. clean) executables are surrounded by malicious (resp. clean) executables in the D dimensional Euclidean space spanned by the normalized vectors

$$\frac{\underline{\mathbf{B}}^i}{\left\|\underline{\mathbf{B}}^i\right\|}, \tag{8}$$

with $\underline{\mathbf{B}}^i$ the binary vectors defined in the previous section. This means that the more hashes two samples have in common the closer they are in this space (assuming that the number of hashes in the two samples does not change).

Let us denote the set of k nearest neighbors of sample i by N_k^i. The classification is based on the data provided by VirusTotal, that is how many anti-virus engines have considered a given executable as malicious. Let us denote this number by m_i for sample i. In the results section is examined how well the m_i of the neighboring samples N_k^i can actually predict if the sample i in question is malicious or clean.

It is important to mention that to predict if a sample i is malicious or not, only neighboring samples are used and not the sample itself. This corresponds to a Leave-One-Out [21], [22], [23], [24] (LOO) classification rate when it comes to assessing the accuracy of the K-NN classifier in the Results section. In [21], [22], it is shown that the Leave-One-Out estimates well the generalization performances of a classifier if the number of samples is large enough, which is the case in the experiments.

As the dimensionality of $\underline{\mathbf{B}}^i$ is too large, random projections are used in order to reduce this dimensionality and therefore reduce the needed computational time and memory by several orders of magnitude. Random projections are explained in the following section.

C. Random Projections

As mentioned earlier the cosine similarity is calculated as

$$J_{cosine}^{i,i'} = \frac{\langle \underline{\mathbf{B}}^i, \underline{\mathbf{B}}^{i'} \rangle}{\left\|\underline{\mathbf{B}}^i\right\| \cdot \left\|\underline{\mathbf{B}}^{i'}\right\|}. \tag{9}$$

However, for practical purposes storing the vector $\underline{\mathbf{B}}^i$ is inconvenient as it requires too much memory (even if stored as a sparse vector). The reason for this is that D, the dimensionality of $\underline{\mathbf{B}}^i$ is in the range of a few millions. In order to alleviate this memory (and the related time) complexity, random projections are used. For the matter of projecting to a lower dimensional space, Johnson and Lindenstrauss [25] have shown that for a set of N points in d-dimensional space (using an Euclidean norm), there exists a linear transformation of the data toward a d_f-dimensional space, with $d_f \geq O(\varepsilon^{-2}\log(N))$ which preserves the distances (and hopefully the "topology" of the data) to a $1 \pm \varepsilon$ factor. Achlioptas [26] has recently extended this result and proposed a simpler projection matrix that preserves the distances to the same factor than the Johnson-Lindenstrauss theorem mentions, at the expense of a probability on the distance conservation. For theory and other applications of random projections in machine learning and classification, see for example [27], [28], [29], [30], [31].

To describe the random projection approach, let $m \in A^i$, and

$$\underline{\mathbf{X}}_m^i = [X_{m,1}^i, X_{m,2}^i, \ldots, X_{m,d}^i], \underline{\mathbf{X}}_m^i \sim \mathcal{N}(\underline{\mathbf{0}}, \underline{\underline{\mathbf{I}}}) \tag{10}$$

such that $\underline{\mathbf{X}}_m^i \perp\!\!\!\perp \underline{\mathbf{X}}_{m'}^{i'}$ if $m \neq m'$, however, if $m = m'$ then $\underline{\mathbf{X}}_m^i = \underline{\mathbf{X}}_{m'}^{i'}$. $\mathcal{N}(\underline{\mathbf{0}}, \underline{\underline{\mathbf{I}}})$ represents a d-dimensional standard normal distribution for which the covariance matrix is the identity matrix, $\underline{\underline{\mathbf{I}}}$.

Then, for each file i the corresponding random projection is the d-dimensional random vector $\underline{\mathbf{Y}}^i$ defined as

$$\underline{\mathbf{Y}}^i = \frac{1}{\sqrt{d}} \frac{1}{\sqrt{|A^i|}} \sum_{m \in A^i} \underline{\mathbf{X}}_m^i. \tag{11}$$

The scalar product of the random vectors gives the similarity J, which is a scalar valued random variable. Using

$$J^{i,i'} = \langle \underline{\mathbf{Y}}^i, \underline{\mathbf{Y}}^{i'} \rangle \tag{12}$$

and the definition of $\underline{\mathbf{Y}}^i$, one can see that $\Pr(J^{i,i} = 1) = 1$. Also if file i and i' do not have any hashes in common, i.e. $A^i \cap A^{i'} \neq \emptyset$, then $\mathbf{E}(J^{i,i'}) = 0$.

As an illustrative example, let us calculate the expected similarity, $\mathbf{E}(J^{i,i'})$ by assuming that $\left|A^i\right| = \left|A^{i'}\right| = l$ and $\left|A^i \cap A^{i'}\right| = k$. Note that the Jaccard distance between i and i' in this case is k/l. Also, due to independence

$$\mathbf{E}(\langle \underline{\mathbf{X}}_m^i, \underline{\mathbf{X}}_{m'}^{i'} \rangle) = 0 \Longleftrightarrow m \neq m'. \tag{13}$$

On the other hand, the following scalar product (in case of matching hashes, $m = m'$) has the chi-square distribution:

$$\langle \underline{\mathbf{X}}_m^i, \underline{\mathbf{X}}_m^{i'} \rangle \sim \chi^2(d) \tag{14}$$

where $\chi^2(d)$ denotes the chi-square distribution with d-degree of freedom, whose expectation value is d. Since only the $\langle \underline{\mathbf{X}}_m^i, \underline{\mathbf{X}}_m^{i'} \rangle$ terms contribute to $\mathbf{E}(J^{i,i'})$ it can be deduced that

$$\mathbf{E}(J^{i,i'}) = \frac{k}{l} \tag{15}$$

which agrees with the Jaccard and cosine similarity in this case. Note that in general if $\left|A^i\right| \neq \left|A^{i'}\right|$ then $\mathbf{E}(J^{i,i'}) \neq J_{Jaccard}$ but still $\mathbf{E}(J^{i,i'}) = J_{cosine}$. Therefore, the Jaccard index is approximated using the cosine similarity approach defined previously.

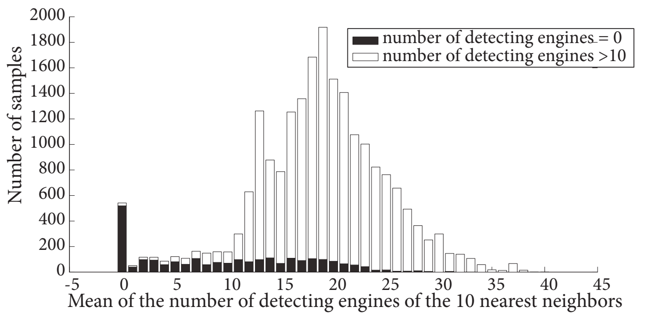

Figure 4. Illustration of the prediction accuracy of the K-NN method: Histogram of the number of detecting engines for $k = 10$ nearest neighbors.

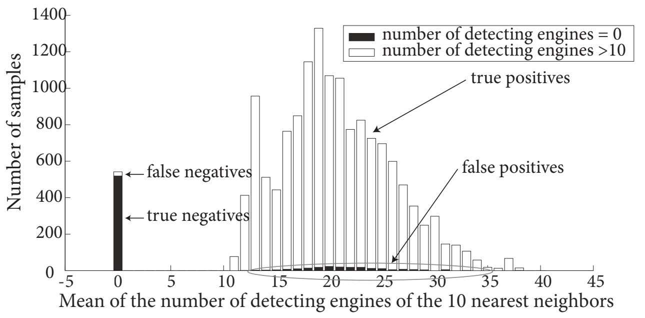

Figure 5. Prediction accuracy of the modified K-NN classifier.

IV. Results

In this section, Euclidean distance is used in the d-dimensional space spanned by the random projected representations $\underline{\mathbf{Y}}^i$ of the samples. As noted earlier, the use of Euclidean distance instead of cosine similarity does not change the results presented in this section as $\Pr(J^{i,i} = 1) = 1$. The $\underline{\mathbf{Y}}^i$ are normalized to unity.

A. *Accuracy of K-NN Classifier*

An illustration of the prediction accuracy of the K-NN method (see section III-B) is shown in Figure 4, and described in detail in the following.

Let N_{10}^i be the set of 10 nearest samples to sample i, then the prediction of the K-NN method for m_i is the mean $\hat{m}_i$ of values $\{m_{i'} : i' \in N_{10}^i\}$ expressed as

$$\hat{m}_i = \left|N_{10}^i\right|^{-1} \sum_{i' \in N_{10}^i} m_{i'}. \quad (16)$$

For a given value x on the x-axis, the height of the bar on y-axis shows for how many samples $\hat{m}_i = x$, i.e. $y(x) = |\{i : \hat{m}_i = x\}|$ is true.

The question is how well the number of detecting engines m_i given by VirusTotal compare with their predicted values, $\hat{m}_i$. In order to answer that question, the samples are divided into two categories: category 1 as 'supposedly clean' (i.e. $m_i = 0$) and category 2 as 'supposedly malicious' (i.e. $m_i > 10$). They are shown in Figures 4 and 5. Assuming that $\hat{m}_i = 0$ means that sample i is predicted to be clean and that $\hat{m}_i > 10$ that sample i is predicted to be malicious, there would be a considerable amount of false positives . The number of false positives can be reduced by introducing a third class into the K-NN classifier: 'unpredictable'. The next section details the results obtained using this additional third class and a modified K-NN.

B. *Accuracy of Modified K-NN Classifier*

Figure 5 shows the prediction accuracy of the modified K-NN classifier. Now, the K-NN classifier has 3 classes: 'predicted to be clean', 'predicted to be malicious', 'unpredictable'.

A sample i is classified as 'clean' if $\hat{m}_i = 0$. It is classified as 'malicious' if $m_{i'} > 10 : \forall i' \in N_{10}^i$, i.e. if all the 10 nearest neighbors N_{10}^i of i are 'supposedly malicious'. A neighboring sample is considered 'supposedly malicious' if $m_{i'} > 10$, i.e. if it has been flagged as malicious by more than 10 AV-engines. Furthermore, a sample i is considered to be 'unpredictable' if it does not fulfill the requirement to be classified as 'clean' or 'malicious'. In the production of the histogram depicted in Figure 5, samples that are 'unpredictable' are omitted. In Figure 5, the concepts of false negative, false positive, true positive and true negative are illustrated.

Introducing the 'unpredictable' class considerably improves the prediction accuracy for the two other classes. This improvement is due to the fact that the uncertainty on the neighbors is used to separate the 'predictable' and 'unpredictable' samples. An unpredictable sample is a sample i, such that not all of its neighbors are either 'supposedly' malicious (i.e. $m_{i'} > 10$) or 'supposedly' clean (i.e. $m_{i'} = 0$).

C. *Influence of the Number of Nearest Neighbors in the Modified K-NN Classifier on the Confusion Matrix*

In Figure 5 are illustrated the notions of false positive, false negative, true positive and true negative. A prediction for a sample i is considered to be a false positive if $m_{i'} > 10 : \forall i' \in N_k^i$ and $m_i = 0$ are true at the same time. This means that all the k-nearest-neighbors N_k^i of sample i are 'supposedly' malicious ($m_{i'} > 10 : \forall i' \in N_k^i$), however, sample i itself is considered to be 'supposedly' clean ($m_i = 0$). Similarly, true positive means that $m_{i'} > 10 : \forall i' \in N_k^i$ and $m_i > 10$ are true for sample i. Furthermore, false negatives are characterized by $m_{i'} = 0 : \forall i' \in N_k^i$ and $m_i > 10$, while a true negative is a sample i for which $m_{i'} = 0 : \forall i' \in N_k^i$ and $m_i = 0$ holds.

The entries of the confusion matrix (false positive, false negative, true positive and true negative) are plotted in Figure 6 as a function of the parameter k, the

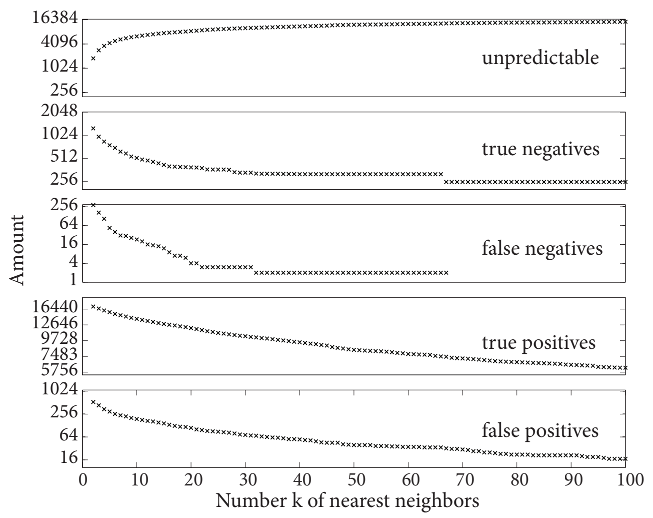

Figure 6. The entries of the confusion matrix (false positive, false negative, true positive and true negative) are plotted in this figure as a function of the parameter k, the number of nearest neighbors. In addition, the number of 'unpredictable' samples is represented.

number of nearest neighbors. Sample i is 'unpredictable' if neither $m_{i'} = 0 : \forall i' \in N_k^i$ nor $m_{i'} > 10 : \forall i' \in N_k^i$ is true. The number of 'unpredictable' samples increases monotonically with increasing k, this must be so as increasing k by one introduces an additional condition that has to be fulfilled in order for a sample to be classified as 'predictable' . In fact, if a sample is labeled as 'unpredictable' for k, it cannot become 'predictable' for $k + l$, $l > 0$.

In Figure 6, one can note that the number of false and true negatives stops decaying at $k = 40$. However, at $k = 40$ the number of true and false positives are still decaying at a rapid rate. The reason for this difference might be that there are much less 'supposedly clean' samples than 'supposedly malicious' ones. Also, the cluster size distribution might be different for these two categories, which could manifest itself in these different decay behaviors in Figure 6.

Figure 6 can be used to choose the parameter k that fits the needs of the user of the modified K-NN method. Furthermore, note the difference in the decay exponents for true and false positive rates. If k is increased from 2 to 100 the number of true positives decreases from 17150 to 6204, while the number of false positives decreases from 531 to 17. The decrease in the true positives is 64% while the decrease in false positives is 97%. So if one wants to increase the true positive/false positive ratio then its advisable to increase the number of neighbors, k. On the other hand one should not forget that by increasing k one also increases the number of 'unpredictable' samples. In order to limit this amount of 'unpredictable' samples, the number of nearest neighbors k to use has been chosen as 11 for the final detection of the false negatives.

D. Influence of the Random Projection Dimension on the Confusion Matrix

In the previous section, the dependency of the confusion matrix with respect to the number of neighbors is discussed and the dimension of the random projected vectors is fixed to be $d = 300$. In this section, the effects of varying d on the confusion matrix are investigated. In order to have a very small number of false negatives and to demonstrate the influence of d, the number of neighbors k is chosen to be 30 in this section. Figure 7 shows the dependency of the confusion matrix on the number of dimensions d of the projected vectors. Clearly, increasing d improves the results: the number of unpredictable samples decrease while the true positives increase and the false positives decrease.

The true and false negatives do not change much with increasing d. This might be related to the fact that at $k = 30$ the decay of true and false negatives in Figure 6 has almost completely stopped. So, even though the low value of $d = 300$ might mean that the distances in the $d = 300$ dimensional Euclidean random projected space are noisy compared to the $D > 10^6$ dimensional original space. The samples that are true and false negatives are insensitive to this noise.

Figure 7 indicates that convergence in all confusion matrix elements can be reached by using $d = 700$. By increasing d even more, no significant improvement is observed.

The necessity to use the random projection method is almost unavoidable: if one would like to use the original space (with dimensionality $D > 10^6$) the complexity of the problem (in terms of memory and computational time) can become an issue as D has been as high as 5.10^8 in other related experiments. In this situation, if one wishes to calculate distances between vectors in the original space then all the data needs to be located in the memory (since the original space is spanned by all the hashes produced by the sandbox). Furthermore, here a set of samples of cardinality of the order of 10^4 as been considered. However, future experiments will be on the scale of 10^6 samples, where using the original space might become prohibitive.

The total computational time needed to run the methodology on the 21053 samples is a few hours using Python implementation of the random projections and K-NN. In comparison, without the random projection approach, the computational time would be estimated to take few weeks, due to the dimensionality of $\underline{\mathbf{B}}^i$.

Finally, based on these results one might improve the previously presented random projection method by using different number of dimensions d for each pair of distance calculated. One could treat larger distances with less accuracy (lower d) while treating smaller distances with better accuracy (higher d). This is a possible direction for future research.

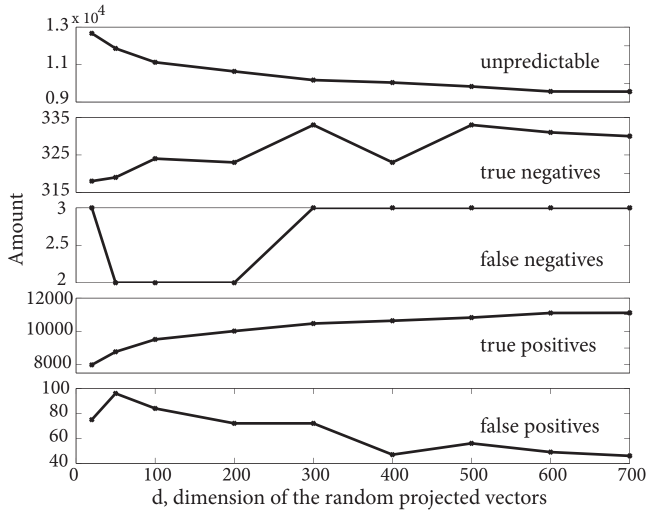

Figure 7. Dependency of the elements of the confusion matrix with respect to the number of dimensions d of the projected vectors.

E. Manual Analysis by a Human Expert and Further Work

Using $d = 100$ projection dimension and a modified K-NN with $k = 11$, 24 false negatives have been extracted out of the 2441 'possibly clean' files. This reduced number allows the manual analysis by a human expert. According to an F-Secure Corporation expert, 25% of these 24 files are surely malicious. 50% have a relatively high probability to be also malicious. The remaining 25% are considered as clean by the expert.

Even with such a reduced number of candidates, a human analysis is taking time and has high costs (especially if the 50% of "unsure" samples have to be further investigated). This shows the usefulness of the presented methodology since it would be impossible to find enough highly qualified experts to analyze the initial 2441 "possibly clean" files.

The same methodology will be applied in the future using different labeling than the one provided by VirusTotal. Also, different dynamic features will be investigated and eventually combined with some static features (code signatures, packer information…), and possibly other types of malware in the sample set.

V. Conclusion

In this paper, a robust two-stage methodology has been introduced in order to both perform classification of executable files and detect the files with the highest probability of being false negatives (malware that are labeled as possibly clean files). It has been shown that the methodology is not only accurate but is also reducing by several orders of magnitude the computational time. This makes the proposed methodology a valid candidate as a pre-processing tool to provide inputs to forensic experts in order to detect malwares that have not yet been detected by the AV engines used in VirusTotal. Furthermore, this methodology can also be applied to other labeling. Also, new and different dynamic features will be investigated and combined with static features (code signatures, packer information…) extracted from the samples before sandbox execution. This will be the natural continuation of the presented work.

Acknowledgments

The authors of this paper would like to acknowledge F-Secure Corporation for providing the data and software required to perform this research. Special thanks go to Pekka Orponen (Head of the ICS Department, Aalto University), Alexey Kirichenko (Research Collaboration Manager F-Secure) and Daavid Hentunen (Researcher F-Secure) for their valuable support and many useful comments. This work was supported by TEKES as part of the Future Internet Programme of TIVIT. Part of the work of Amaury Lendasse and Alexander Ilin is funded by the Adaptive Informatics Research Centre, Centre of Excellence of the Finnish Academy.

References

[1] Y. Liu, L. Zhang, J. Liang, S. Qu, and Z. Ni, "Detecting trojan horses based on system behavior using machine learning method," in *Machine Learning and Cybernetics (ICMLC), 2010 International Conference on*, vol. 2, July 2010, pp. 855 –860.

[2] I. Firdausi, C. Lim, A. Erwin, and A. Nugroho, "Analysis of machine learning techniques used in behavior-based malware detection," in *Advances in Computing, Control and Telecommunication Technologies (ACT), 2010 Second International Conference on*, December 2010, pp. 201 –203.

[3] E. Menahem, A. Shabtai, L. Rokach, and Y. Elovici, "Improving malware detection by applying multi-inducer ensemble," *Computational Statistics & Data Analysis*, vol. 53, no. 4, pp. 1483 – 1494, 2009.

[4] L. Sun, S. Versteeg, S. Boztaş, and T. Yann, "Pattern recognition techniques for the classification of malware packers," in *Information Security and Privacy*, ser. Lecture Notes in Computer Science, R. Steinfeld and P. Hawkes, Eds. Springer Berlin / Heidelberg, 2010, vol. 6168, pp. 370–390.

[5] J. Kinable and O. Kostakis, "Malware classification based on call graph clustering," *Journal in Computer Virology*, pp. 1–13, 2011.

[6] A. Srivastava and J. Giffin, “Automatic discovery of parasitic malware,” in *Recent Advances in Intrusion Detection (RAID’10)*, ser. Lecture Notes in Computer Science, S. Jha, R. Sommer, and C. Kreibich, Eds. Springer Berlin / Heidelberg, 2010, vol. 6307, pp. 97–117.

[7] C. Willems, T.Holz, and F. Freiling, “Toward automated dynamic malware analysis using cwsandbox,” *IEEE Security and Privacy*, vol. 5, pp. 32–39, March 2007.

[8] K. Yoshioka, Y. Hosobuchi, T. Orii, and T. Matsumoto, “Vulnerability in public malware sandbox analysis systems,” in *Proceedings of the 2010 10th IEEE/IPSJ International Symposium on Applications and the Internet*, ser. SAINT ’10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 265–268.

[9] M.Bailey, J. Andersen, Z. Morleymao, and F. Jahanian, “Automated classification and analysis of internet malware,” in *Recent Advances in Intrusion Detection (RAID’07)*, 2007.

[10] P. Jaccard, “Etude comparative de la distribution florale dans une portion des alpes et des jura,” *Bulletin de la Societe Vaudoise des Sciences Naturelles*, vol. 37, pp. 547–579, 1901.

[11] P. Tan, M. Steinbach, and V. Kumar, *Introduction to Data Mining.* Addison Wesley, 2005.

[12] Hispasec Systemas, “Virus total analysis tool,” 2011, http://www.virustotal.com.

[13] Norman ASA, “Norman launches sandbox sdk,” April 2009, http://www.norman.com/about_norman/press_center/news_archive/2009/67431/en.

[14] F-Secure Corporation, “F-secure deepguard – a proactive response to the evolving threat scenario,” November 2006, http://www.rp-net.com/online/filelink/340/20061106%20F-secure_deepguard_whitepaper.pdf.

[15] ——, “F-secure deepguard 2.0 - white paper,” September 2008, http://www.f-secure.com/system/fsgalleries/white-papers/f-secure_deepguard_2.0_whitepaper.pdf.

[16] ——, “Information about system control and deepguard,” January 2011, http://www.f-secure.com/kb/2034.

[17] D. Aha and D. Kibler, “Instance-based learning algorithms,” in *Machine Learning*, 1991, pp. 37–66.

[18] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft, “When is "nearest neighbor" meaningful?” in *Int. Conf. on Database Theory*, 1999, pp. 217–235.

[19] C. Bishop, *Neural Networks for Pattern Recognition*, 1st ed. Oxford University Press, USA, Jan. 1996.

[20] P. Devijver and J. Kittler, *Pattern recognition: A statistical approach.* Prentice Hall, 1982.

[21] B. Efron and R. Tibshirani, *An Introduction to the Bootstrap.* New York: Chapman & Hall, 1993.

[22] ——, “Improvemenets on cross-validation: The .632+ bootstrap method,” *Journal of the American Statistical Association*, vol. 92, no. 438, pp. 548–560, 1997.

[23] A. Lendasse, V. Wertz, and M. Verleysen, “Model selection with cross-validations and bootstraps - application to time series prediction with rbfn models,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 2714, pp. 573–580, 2003, cited By (since 1996) 8.

[24] Q. Yu, Y. Miche, A. Sorjamaa, A. Guillén, A. Lendasse, and E. Séverin, “OP-KNN: Method and applications,” *Advances in Artificial Neural Systems*, vol. 2010, no. 597373, February 2010, 6 pages.

[25] W. B. Johnson and J. Lindenstrauss, “Extensions of lipschitz mappings into a hilbert space,” in *Conference in Modern Analysis and Probability*, New Haven, USA, 1982, pp. 189–206.

[26] D. Achlioptas, “Database-friendly random projections: Johnson-lindenstrauss with binary coins,” *J. Comput. Syst. Sci.*, vol. 66, no. 4, pp. 671–687, 2003.

[27] S. Dasgupta, “Experiments with random projection,” in *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*, ser. UAI ’00. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2000, pp. 143–151.

[28] X. Fern and C. Brodley, “Random projection for high dimensional data clustering: A cluster ensemble approach,” in *International Conference on Machine Learning (ICML’03)*, 2003, pp. 186–193.

[29] D. Fradkin and D. Madigan, “Experiments with random projections for machine learning,” in *KDD ’03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining.* New York, NY, USA: ACM, 2003, pp. 517–522.

[30] Y. Miche, A. Sorjamaa, P. Bas, O. Simula, C. Jutten, and A. Lendasse, “OP-ELM: Optimally-pruned extreme learning machine,” *IEEE Transactions on Neural Networks*, vol. 21, no. 1, pp. 158–162, January 2010.

[31] S. Vempala, *The Random Projection Method*, ser. DIMACS Series in Discrete Mathematics and Theoretical Computer Science. American Mathematical Society, 2005, vol. 65.