COLLECTIVE CLASSIFICATION FOR UNKNOWN MALWARE DETECTION

Igor Santos, Carlos Laorden and Pablo G. Bringas
S^{3}Lab, DeustoTech - Computing Deusto Institute of Technology, University of Deusto
Avenida de las Universidades 24, 48007, Bilbao, Spain

Keywords: Security, Computer viruses, Data-mining, Malware detection, Machine learning.

Abstract: Malware is any type of computer software harmful to computers and networks. The amount of malware is increasing every year and poses as a serious global security threat. Signature-based detection is the most broadly used commercial antivirus method, however, it fails to detect new and previously unseen malware. Supervised machine-learning models have been proposed in order to solve this issue, but the usefulness of supervised learning is far to be perfect because it requires a significant amount of malicious code and benign software to be identified and labelled in beforehand. In this paper, we propose a new method that adopts a collective learning approach to detect unknown malware. Collective classification is a type of semi-supervised learning that presents an interesting method for optimising the classification of partially-labelled data. In this way, we propose here, for the first time, collective classification algorithms to build different machine-learning classifiers using a set of labelled (as malware and legitimate software) and unlabelled instances. We perform an empirical validation demonstrating that the labelling efforts are lower than when supervised learning is used, while maintaining high accuracy rates.

1 INTRODUCTION

Commercial anti-malware solutions are based on signature databases for the detection of known malicious executables (Morley, 2001). However, there are several problems that make the signature-based methods less than completely reliable: first they cannot cope with code obfuscations and second cannot detect previously unseen malware. In this paper, we are going to focus on the second problem: detecting previously unseen malware.

Machine-learning-based approaches train classification algorithms that detect new malware (this type of malware is usually called 'in the wild' by the anti-malware industry and research community), relying on datasets composed of several characteristic features of both malicious and benign software. Schultz et al. (Schultz et al., 2001) were the first to introduce the concept of applying machine-learning algorithms for the detection of malware based on their respective binary codes. They applied different classifiers to three types of feature sets: (i) program headers, (ii) strings and (iii) byte sequences. Later, Kolter et al. (Kolter and Maloof, 2004) improved Schulz's results by applying n-grams (i.e., overlapping byte sequences) instead of non-overlapping sequences. This approach employed several algorithms, achieving the best results with a boosted[1] decision tree. Substantial research has focused on n-gram distributions of byte sequences and machine-learning algorithms (Moskovitch et al., 2008; Zhou and Inge, 2008; Santos et al., 2009).

Nevertheless, machine-learning classifiers require a high number of labelled executables for each of the classes (i.e., malware and benign). It is quite difficult to obtain this amount of labelled data for a real-world problem such as the malicious code analysis. To generate these data, a time-consuming process of analysis is mandatory and, in the process, some malicious executables can avoid detection. Within the full scope of machine-learning, several approaches have been proposed to deal with this issue. Semi-supervised learning is a type of machine-learning technique that is specially useful when a limited amount of labelled data exists for each class (Chapelle et al., 2006). In particular, collective classification (Neville and Jensen, 2003) is an approach that uses the relational

[1]Boosting is a machine-learning technique that builds a strong classifier composed of a high number of weak classifiers (Schapire, 2003).

Santos I., Laorden C. and G. Bringas P..
COLLECTIVE CLASSIFICATION FOR UNKNOWN MALWARE DETECTION.
DOI: 10.5220/0003452802510256
In *Proceedings of the International Conference on Security and Cryptography* (SECRYPT-2011), pages 251-256
ISBN: 978-989-8425-71-3
Copyright © 2011 SCITEPRESS (Science and Technology Publications, Lda.)

structure of the combined labelled and unlabelled data-set to enhance classification accuracy. With relational approaches, the predicted label of an example will often be influenced by the labels of related examples. The idea behind collective classification is that the predicted labels of a test-example should also be influenced by the predictions made for the related test-examples. In some cases, it is interesting to determine the topic of not just a single evidence, but to infer it for a collection of unlabelled evidences. Collective classification tries to collectively optimise the problem taking into account the connections present among the executables' machine code. This is a semi-supervised technique, i.e., uses both labelled and unlabelled data – typically a small amount of labelled data and a large amount of unlabelled data – that reduces the labelling work.

Given this background, in a previous work (Santos et al., 2011), we proposed the adoption of LLGC, a semi-supervised approach, for malware detection. Here, we propose the first approach that employs collective classification algorithms for the detection of unknown malware that improves our previous results. These methods are able to learn from both labelled and unlabelled data. For the representation of executables, we chose the byte n-gram distribution, a well-known technique that has achieved significant results with supervised machine learning (Kolter and Maloof, 2004; Moskovitch et al., 2008; Santos et al., 2009; Devesa et al., 2010). However, the presented collective methodology is scalable to any representation susceptible to be represented as a feature vector.

Summarising, our main findings in this paper are: (i) we describe how to adopt collective classification for unknown malware detection, (ii) we determine the optimal number of labelled instances and evaluate how this parameter affects the final accuracy of the models and (iii) we show that labelling efforts can be reduced in the industry, while still maintaining a high rate of accuracy.

2 BYTE N-GRAM REPRESENTATION

Byte n-grams frequencies distribution is a well-known approach for training machine-learning classifiers to detect unknown malicious code (Schultz et al., 2001; Kolter and Maloof, 2004; Moskovitch et al., 2008; Shafiq et al., 2008; Zhou and Inge, 2008; Santos et al., 2009). To obtain a representation of the executables by the use of byte n-grams, we need to extract every possible sequence of bytes and their appearance frequency.

Specifically, a binary program $\mathcal{P}$ can be represented as a sequence of ℓ bytes b as $\mathcal{P} = \{b_1, b_2, b_3, ..., b_{\ell-1}, b_\ell\}$. A byte n-gram sequence g is defined as a subset of consecutive bytes within an executable file where $g \subseteq \mathcal{P}$ and it is made up of bytes b, such as $g = (b_1, b_2, b_3, ..., b_{n-1}, b_n)$ where n is the length of the byte n-gram g. Therefore, a program $\mathcal{P}$ is composed of byte n-grams such as $\mathcal{P} = (g_1, g_2, ..., g_{\ell-1}, g_\ell)$ where ℓ is the total number of possible n-grams of a fixed length n.

We use 'term frequency - inverse document frequency' $(tf - idf)$ (McGill and Salton, 1983) to obtain the weight of each byte n-gram, whereas the weight of the i^{th} n-gram in the j^{th} executable, denoted by $weight(i, j)$, is defined by: $weight(i, j) = tf_{i,j} \cdot idf_i$ where the term frequency $tf_{i,j}$ (McGill and Salton, 1983) is defined as: $tf_{i,j} = \frac{m_{i,j}}{\sum_k m_{k,j}}$ where $m_{i,j}$ is the number of times the n-gram $t_{i,j}$ appears in an executable e, and $\sum_k m_{k,j}$ is the total number of n-grams in the executable e.

On the other hand, the inverse document frequency idf_i is defined as: $idf_i = \frac{|\mathcal{E}|}{|\mathcal{E} : t_i \in e|}$ where $|\mathcal{E}|$ is the total number of executables and $|\mathcal{E} : t_i \in e|$ is the number of executables containing the n-gram t_i.

Finally, we can obtain a vector $\vec{v}$ composed of byte n-grams frequencies, $\vec{v} = ((g_1, weight_1), ..., (g_{m-1}, weight_{m-1}), (g_m, weight_m))$, where g_i is the byte n-gram and $weight_i$ is the value of $tf - idf$ for that particular n-gram.

3 COLLECTIVE CLASSIFICATION

Collective classification is a combinatorial optimization problem, in which we are given a set of documents, or nodes, $\mathcal{D} = \{d_1, ..., d_n\}$ and a neighbourhood function N, where $N_i \subseteq \mathcal{D} \setminus \{\mathcal{D}_i\}$, which describes the underlying network structure (Namata et al., 2009). Being $\mathcal{D}$ a random collection of documents, it is divided into two sets $\mathcal{X}$ and $\mathcal{Y}$ where $\mathcal{X}$ corresponds to the documents for which we know the correct values and $\mathcal{Y}$ are the documents whose values need to be determined. Therefore, the task is to label the nodes $\mathcal{Y}_i \in \mathcal{Y}$ with one of a small number of labels, $\mathcal{L} = \{l_1, ..., l_q\}$. We use the *Waikato Environment for Knowledge Analysis* (WEKA) (Garner, 1995) and its Semi-Supervised Learning and Collective Classification plugin[2] which provides the following collective classifiers:

[2] Available at: http://www.scms.waikato.ac.nz/~fracpete/projects/collectiveclassification

- **CollectiveIBk:** It uses internally WEKA's classic IBk algorithm, implementation of the *K-Nearest Neighbour* (KNN), to determine the best k on the training set and builds then, for all instances from the test set, a neighbourhood consisting of k instances from the pool of train and test set (either a naïve search over the complete set of instances or a k-dimensional tree is used to determine neighbours), which are sorted according to their distance to the test instance they belong to. For every unlabelled test instance with the highest rank, the class label is determined by majority vote. This is performed until no further unlabelled test instances remain. The classification terminates by returning the class label of the instance that is about to be classified.
- **CollectiveForest:** It uses WEKA's implementation of RandomTree as base classifier to divide the test set into folds containing the same number of elements. The first iteration trains using the original training set and generates the distribution for all the instances in the test set. The best instances are then added to the original training set (being the number of instances chosen the same as in a fold). The next iterations train with the new training set and generate then the distributions for the remaining instances in the test set.
- **CollectiveWoods & CollectiveTree**: CollectiveWoods works like CollectiveForest using CollectiveTree instead of RandomTree. Collective tree splits the attribute at that position that divides the current subset of instances (training and test instances) into two halves. The process finishes if one of the following conditions is met: (i) only training instances would be covered (the labels for these instances are already known), (ii) only test instances in the leaf, case in which distribution from the parent node is taken and (iii) only training instances of one class, case in which all test instances are considered to have this class. To calculate the class distribution of a complete set or a subset, the weights are summed up according to the weights in the training set, and then normalised. The nominal attribute distribution corresponds to the normalised sum of weights for each distinct value and, for the numeric attribute, distribution of the binary split based on median is calculated and then the weights are summed up for the two bins and finally normalised.
- **RandomWoods:** It works like WEKA's classic RandomForest but using CollectiveBagging (classic Bagging, a machine learning ensemble meta-algorithm to improve stability and classification accuracy, extended to make it available to collective classifiers) in combination with CollectiveTree in contrast to RandomForest, which uses Bagging and RandomTree.

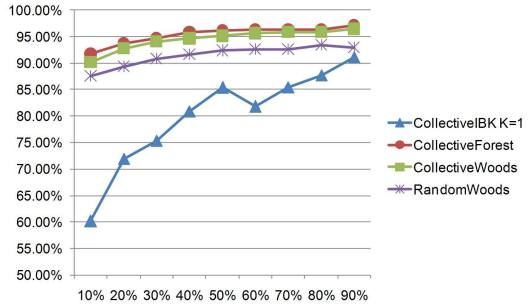
4 EXPERIMENTAL RESULTS

The research question we seek to answer through this empirical validation is the following one: *What is the minimum number of labelled instances required to assure a suitable performance using collective classification?* To this end, we collected a dataset comprising 1,000 malicious executables and 1,000 benign ones. For the malware, we gathered random samples from the website VxHeavens[3], which has assembled a malware collection of more than 17,000 malicious programs, including 585 malware families that represent different types of current malware such as Trojan horses, viruses and worms. Although they had already been labelled according to their family and variant names, we analysed them using Eset Antivirus[4] to confirm this labelling. For the benign dataset, we collected legitimate executables from our own computers. We also performed an analysis of the benign files using Eset Antivirus to confirm their legitimacy.

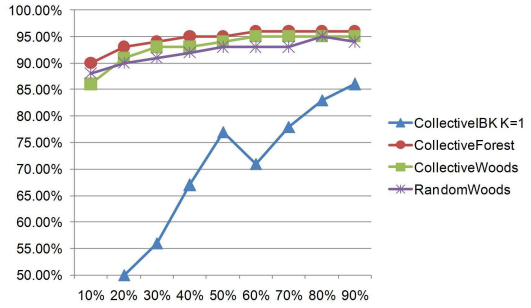
By means of this dataset, we conducted the following methodology to answer the research question and thus, evaluate the proposed method:

1. **Byte N-gram Extraction.** We extracted the byte n-gram representation for each file in the dataset for $n = 2$. This specific length was chosen because it is the number of bytes that an operation represented by an operational code needs in machine code, a widely-used n-gram length in the literature (Schultz et al., 2001; Kolter and Maloof, 2004).
2. **Feature Extraction.** Because the total number of features we obtained was high, we applied a feature selection step based on a Document Frequency (DF) measure, which counts the number of documents in which a specific n-gram appears, selecting the 5,000 top ranked byte n-grams. This concrete number of features was chosen because it provides a balance between efficiency and accuracy and it has been proven to be effective (Moskovitch et al., 2008).
3. **Training and Test Generation.** We constructed an *ARFF* file (Holmes et al., 1994) (i.e., Attribute Relation File Format) with the resultant vector representations of the executables to build the aforementioned WEKA's classifiers. Next,

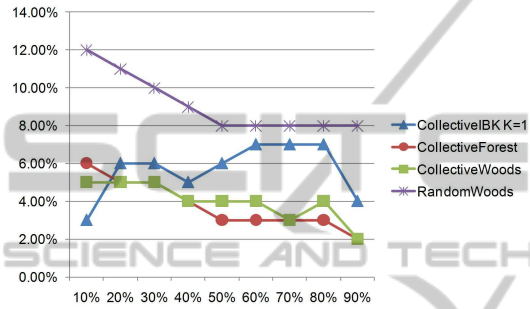
[3]Available at: http://vx.netlux.org/
[4]Available at: http://www.eset.com/



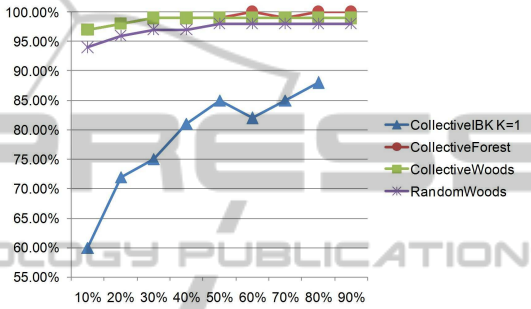
(a) Accuracy results. The X axis represents the amount of labelled data (training dataset) whereas the Y axis represents the accuracy of the different collective classifiers.



(b) TPR results. The X axis represents the amount of labelled data (training dataset) whereas the Y axis represents the TPR of the different collective classifiers.



(c) FPR results. The X axis represents the amount of labelled data (training dataset) whereas the Y axis represents the FPR of the different collective classifiers.



(d) AUC results. The X axis represents the amount of labelled data (training dataset) whereas the Y axis represents the AUC of the different collective classifiers.

Figure 1: The results of our collective-classification-based malware detection method. Collective Forest was the classifier with the highest accuracy, TPR and AUC, and the lowest value of FPR. Our results outline that, obviously, the higher the number of labelled executables are in the dataset the better results are achieved.

SCITEPRESS
SCIENCE AND TECHNOLOGY PUBLICATIONS

we split the dataset into different percentages of training and tested instances. In other words, we changed the number of labelled instances from 10% to 90% to measure the effect of the number of labelled instances on the final performance of collective classification in detecting unknown malware. In particular, we used the collective classification implementations provided by the *Semi-Supervised Learning and Collective Classification* package for the well-known machine-learning tool WEKA (Garner, 1995). All the classifiers were tested with their default parameters.

4. **Testing the Models.** To test the approach, we measured the *True Positive Ratio* (TPR), i.e., the number of malware instances correctly detected divided by the total number of malware files: $TPR = (TP)/(TP+FN)$ where TP is the number of malware cases correctly classified (true positives) and FN is the number of malware cases misclassified as legitimate software (false negatives). We also measured the *False Positive Ratio* (FPR), i.e., the number of benign executables misclassified as malware divided by the total number of benign files: $FPR = (FP)/(FP+TN)$ where FP is the number of benign software cases incorrectly detected as malware and TN is the number of legitimate executables correctly classified. Furthermore, we measured *accuracy*, i.e., the total number of hits of the classifiers divided by the number of instances in the whole dataset: $Accuracy = (TP+TN)/(TP+FP+TP+TN)$ Finally, we measured the *Area Under the ROC Curve* (AUC), which establishes the relation between false negatives and false positives (Singh et al., 2009). The ROC curve is obtained by plotting the TPR against the FPR. All the these measures refer to the test instances.

Figure 1 shows the obtained results in terms of accuracy, AUC, TPR and FPR. Figure 1(a) shows the accuracy results of our proposed method. As one may think in beforehand, the higher the number of training instances the higher the accuracy of the different models. In particular, the best overall results were obtained by Collective Forest, achieving results higher than 90% for every possible configuration and higher than 95% when half of the instances were used for

training. On the other hand, Collective IBK obtained the lowest accuracy results, achieving an accuracy higher than 80% only when more than the 50% of the instances were employed. Figure 1(b) shows the obtained results in terms of correctly classified malware executables. Collective Forest was also the best, detecting more than the 90% of the malware executables for every tested configuration. Figure 1(c) shows the FPR results. Every classifier obtained results lower than the 10%. In particular, the lowest FPR was of 2%, achieved by CollectiveForest using the 90% of the instances for training. However, in order to guarantee results of FPR lower than 5%, Collective Forest only needs to be trained with a minimum of 20% of the dataset. Finally, regarding AUC (shown in Figure 1(d)), Collective Forest was the best with results higher than 97% for every configuration.

5 DISCUSSION

The obtained results validate our initial hypothesis that building an unknown malware detector based on collective classification is feasible. The classifiers achieved high performance in classifying unknown malware, improving our previous results using LLGC (Santos et al., 2011), which achieved a 86% of accuracy in its best configuration. Therefore, we believe that our results will have a strong impact in the area of unknown malware detection, which usually relies on supervised machine learning (Schultz et al., 2001; Kolter and Maloof, 2004). Training the model through supervised machine-learning algorithms can be a problem itself because supervised learning requires each instance in the dataset to be properly labelled. This demands a large amount of time and resources. We have dealt with this problem by using a collective approach that only needs a certain amount of data to be labelled. In this way, we tried to find among our results the number of labelled malware that is needed to assure a certain performance in unknown malware detection. In particular, we found out that if we label the 10% of the the total corpus, our method can achieve an accuracy and a F-measure greater than 90%. Nevertheless, there are several considerations regarding the viability of this method.

First, because of the static nature of the proposed method, it cannot counter *packed* malware. Packed malware is the result of cyphering the payload of the executable and deciphering it when the executable is finally loaded into memory. One solution to solve this obvious limitation of our malware detection method is the use of a generic dynamic unpacking schema such as PolyUnpack (Royal et al., 2006), Renovo (Kang et al., 2007), OmniUnpack (Martignoni et al., 2007) or Eureka (Sharif et al., 2008). These methods execute the sample in a contained environment and extract the actual payload, allowing further static or dynamic analysis of the executable.

Second, our method can be considered as a statistical representation of executables. Therefore, an attacker can surpass this method of detection by adding several 'good' byte sequences. For example, in the field of spam filtering (spam is defined as unsolicited bulk mail), *Good Word Attack* is a method that modifies the term statistics by appending a set of words that are characteristic of legitimate e-mails, thereby bypass spam filters. Nevertheless, we can adapt to this malware detector some of the methods that have been proposed in order to improve spam filtering , such as *Multiple Instance Learning* (MIL) (Zhou et al., 2007). MIL will divide an executable or a vector in the traditional methods into several sub-instances and will classify the original vector or classifier based on the sub-instances.

Finally, in our experiments, we used a training dataset that is very small in comparison to commercial antivirus databases. Whenever the dataset size grows, the issue of scalability becomes a concern. This problem produces excessive storage requirements, increases time complexity and impairs the general accuracy of the models (Cano et al., 2006). To reduce disproportionate storage and time costs, it is necessary to reduce the size of the original training set (Czarnowski and Jedrzejowicz, 2006). To solve this issue, data reduction is normally considered an appropriate preprocessing optimisation technique (Pyle, 1999; Tsang et al., 2003). Data reduction can be implemented in two ways. Instance selection (IS) seeks to reduce the number of evidences (i.e., number of rows) in the training set by selecting the most relevant instances or by re-sampling new ones (Liu and Motoda, 2001). Feature selection (FS) decreases the number of attributes or features (i.e., columns) in the training set (Liu and Motoda, 2008).

Future work will be oriented on three main directions. First, we will extend our study of collective learning by applying more algorithms to this issue. Second, we will use different features for training these kind of models. Third, we will face packed executables with a hybrid dynamic-static approach.

REFERENCES

Cano, J., Herrera, F., and Lozano, M. (2006). On the combination of evolutionary algorithms and stratified strategies for training set selection in data mining. *Applied Soft Computing Journal*, 6(3):323–332.

Chapelle, O., Schölkopf, B., and Zien, A. (2006). *Semi-supervised learning*. MIT Press.

Czarnowski, I. and Jedrzejowicz, P. (2006). Instance reduction approach to machine learning and multi-database mining. In *Proceedings of the Scientific Session organized during XXI Fall Meeting of the Polish Information Processing Society, Informatica, ANNALES Universitatis Mariae Curie-Skłodowska, Lublin*, pages 60–71.

Devesa, J., Santos, I., Cantero, X., Penya, Y. K., and Bringas, P. G. (2010). Automatic Behaviour-based Analysis and Classification System for Malware Detection. In *Proceedings of the 12th International Conference on Enterprise Information Systems (ICEIS)*, pages 395–399.

Garner, S. (1995). Weka: The Waikato environment for knowledge analysis. In *Proceedings of the New Zealand Computer Science Research Students Conference*, pages 57–64.

Holmes, G., Donkin, A., and Witten, I. H. (1994). Weka: a machine learning workbench. pages 357–361.

Kang, M., Poosankam, P., and Yin, H. (2007). Renovo: A hidden code extractor for packed executables. In *Proceedings of the 2007 ACM workshop on Recurring malcode*, pages 46–53.

Kolter, J. and Maloof, M. (2004). Learning to detect malicious executables in the wild. In *Proceedings of the 10th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 470–478. ACM New York, NY, USA.

Liu, H. and Motoda, H. (2001). *Instance selection and construction for data mining*. Kluwer Academic Pub.

Liu, H. and Motoda, H. (2008). *Computational methods of feature selection*. Chapman & Hall/CRC.

Martignoni, L., Christodorescu, M., and Jha, S. (2007). Omniunpack: Fast, generic, and safe unpacking of malware. In *Proceedings of the 23rd Annual Computer Security Applications Conference (ACSAC)*, pages 431–441.

McGill, M. and Salton, G. (1983). *Introduction to modern information retrieval*. McGraw-Hill.

Morley, P. (2001). Processing virus collections. In *Proceedings of the 2001 Virus Bulletin Conference (VB2001)*, pages 129–134. Virus Bulletin.

Moskovitch, R., Stopel, D., Feher, C., Nissim, N., and Elovici, Y. (2008). Unknown malcode detection via text categorization and the imbalance problem. In *Proceedings of the 6th IEEE International Conference on Intelligence and Security Informatics (ISI)*, pages 156–161.

Namata, G., Sen, P., Bilgic, M., and Getoor, L. (2009). Collective classification for text classification. *Text Mining*, pages 51–69.

Neville, J. and Jensen, D. (2003). Collective classification with relational dependency networks. In *Proceedings of the Workshop on Multi-Relational Data Mining (MRDM)*.

Pyle, D. (1999). *Data preparation for data mining*. Morgan Kaufmann.

Royal, P., Halpin, M., Dagon, D., Edmonds, R., and Lee, W. (2006). Polyunpack: Automating the hidden-code extraction of unpack-executing malware. In *Proceedings of the 22nd Annual Computer Security Applications Conference (ACSAC)*, pages 289–300.

Santos, I., Nieves, J., and Bringas, P. (2011). Semi-supervised learning for unknown malware detection. In Abraham, A., Corchado, J., Gonzlez, S., and De Paz Santana, J., editors, *International Symposium on Distributed Computing and Artificial Intelligence*, volume 91 of *Advances in Intelligent and Soft Computing*, pages 415–422. Springer Berlin / Heidelberg.

Santos, I., Penya, Y., Devesa, J., and Bringas, P. (2009). N-Grams-based file signatures for malware detection. In *Proceedings of the 11th International Conference on Enterprise Information Systems (ICEIS), Volume AIDSS*, pages 317–320.

Schapire, R. (2003). The boosting approach to machine learning: An overview. *Lecture Notes in Statistics*, pages 149–172.

Schultz, M., Eskin, E., Zadok, F., and Stolfo, S. (2001). Data mining methods for detection of new malicious executables. In *Proceedings of the 22nd IEEE Symposium on Security and Privacy.*, pages 38–49.

Shafiq, M., Khayam, S., and Farooq, M. (2008). Embedded Malware Detection Using Markov n-Grams. *Lecture Notes in Computer Science*, 5137:88–107.

Sharif, M., Yegneswaran, V., Saidi, H., Porras, P., and Lee, W. (2008). Eureka: A Framework for Enabling Static Malware Analysis. In *Proceedings of the European Symposium on Research in Computer Security (ESORICS)*, pages 481–500.

Singh, Y., Kaur, A., and Malhotra, R. (2009). Comparative analysis of regression and machine learning methods for predicting fault proneness models. *International Journal of Computer Applications in Technology*, 35(2):183–193.

Tsang, E., Yeung, D., and Wang, X. (2003). OFFSS: optimal fuzzy-valued feature subset selection. *IEEE transactions on fuzzy systems*, 11(2):202–213.

Zhou, Y. and Inge, W. (2008). Malware detection using adaptive data compression. In *Proceedings of the 1st ACM workshop on Workshop on AISec*, pages 53–60. ACM New York, NY, USA.

Zhou, Y., Jorgensen, Z., and Inge, M. (2007). Combating Good Word Attacks on Statistical Spam Filters with Multiple Instance Learning. In *Proceedings of the 19th IEEE International Conference on Tools with Artificial Intelligence-Volume 02*, pages 298–305.