Semi-supervised Learning for Unknown Malware Detection

Igor Santos, Javier Nieves and Pablo G. Bringas

Abstract Malware is any kind of computer software potentially harmful to both computers and networks. The amount of malware is increasing every year and poses a serious global security threat. Signature-based detection is the most widely used commercial antivirus method, however, it consistently fails to detect new malware. Supervised machine-learning models have been used to solve this issue, but the usefulness of supervised learning is far to be perfect because it requires that a significant amount of malicious code and benign software to be identified and labelled beforehand. In this paper, we propose a new method of malware protection that adopts a semi-supervised learning approach to detect unknown malware. This method is designed to build a machine-learning classifier using a set of labelled (malware and legitimate software) and unlabelled instances. We performed an empirical validation demonstrating that the labelling efforts are lower than when supervised learning is used, while maintaining high accuracy rates.

1 Introduction

Malware is any computer software intentionally designed to damage computers. Although 'fame and glory' were the main goals of malware writers in the past, more recently, their reasons have evolved into economic considerations [9].

Commercial anti-malware solutions generally base their main detection systems on signature databases [7]. A signature is a unique sequence of bytes that is always present within malicious executables and in the files already infected by that malware. The main problem of such an approach is that specialists have to wait until new malware has damaged several computers to generate a signature file and they

Igor Santos, Javier Nieves and Pablo G. Bringas
Laboratory for Smartness, Semantics and Security (S^3Lab), DeustoTech - Computing, University of Deusto, Avenida de las Universidades 24, 48007 Bilbao, Spain, e-mail: {isantos, jnieves, pablo.garcia.bringas}@deusto.es

can provide a suitable solution. Suspect files that are later subjected to analysis are compared with this list of signatures. When the signatures match, the file being tested is flagged as malware. Although this approach has been demonstrated to be effective when threats are known in advance, signature methods cannot cope with code obfuscation, with previously unseen malware, or with large amounts of new malware [11, 12].

Machine-learning-based approaches train classification algorithms that detect new malware (i.e., 'in the wild'), relying on datasets composed of several characteristic features of both malicious and benign software. Schultz et al. [14] were the first to introduce the concept of applying machine-learning models to the detection of malware based on their respective binary codes. Specifically, they applied several classifiers to three different feature sets: (i) program headers, (ii) strings and (iii) byte sequence. Later, Kolter et al. [4] improved Schulz's results [14], 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. In a similar vein, substantial research has focused on n-gram distributions of byte sequences and data-mining [8, 19, 12].

Machine-learning classifiers require a high number of labelled executables for each of the classes (i.e., malware and benign datasets). Nevertheless, it is quite difficult to obtain this amount of labelled data for a real-world problem such as 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 techniques that is specially useful when a limited amount of labelled data exists for each class. These techniques create a supervised classifier based on labelled data and predict the label for all unlabelled instances. The instances whose classes have been predicted with a certain threshold of confidence are added to the labelled dataset. The process is repeated until certain conditions are satisfied (a commonly used criterion is the maximum likelihood found by the expectation-maximisation technique). These approaches improve the accuracy of fully unsupervised methods (i.e., no labels within the dataset) [1].

In light of this background, we propose here the first approach that employs a semi-supervised learning technique for the detection of unknown malware. In particular, we utilise the method *Learning with Local and Global Consistency* (LLGC) [18] able to learn from both labelled and unlabelled data and capable of providing a *smooth* solution with respect to the intrinsic structure displayed by both labelled and unlabelled instances. For the representation of executables, we choose the byte n-gram distribution, a well-known technique that have achieved significant results with supervised machine learning (e.g., [4, 8, 12]). However, the presented semi-supervised methodology is scalable to any representation susceptible to be represented as a feature vector. Summarising, our main findings in this paper are: (i) we

[1] Boosting is a machine-learning technique that builds a strong classifier composed by weak classifiers [13].

describe how to adopt LLGC for unknown malware detection, (ii) we determine the optimal number of labelled instances and we evaluated 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.

The remainder of this paper is organised as follows. Section 2 provides the background regarding the representation of executables based on byte n-gram frequencies. Section 3 describes the LLGC method and how it can be adopted for unknown malware detection. Section 4 describes the experiments and presents results. Finally, Section 5 concludes the paper and outlines avenues for future work.

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 [14, 4, 8, 15, 19, 12]. 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 $\mathscr{P}$ can be represented as a sequence of ℓ bytes b as $\mathscr{P} = \{b_1, b_2, b_3, ..., b_{\ell-1}, b_\ell\}$. A byte n-gram sequence g is defined as a subset of consecutive bytes within an executable file where $g \subseteq \mathscr{P}$ 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 $\mathscr{P}$ is composed of byte n-grams such as $\mathscr{P} = (g_1, g_2, ..., g_{\ell-1}, g_\ell)$ where ℓ is the total number of possible n-grams of a fixed length n.

```
4D 5A 90 00
03 00 00 00
04 00 00 00
FF FF 00 00
```

Fig. 1 Machine code example.

Consider an example based on the machine code snippet shown in Fig. 1; the following byte bi-grams can be generated: $g_1 = (\texttt{4D}, \texttt{5A})$, $g_2 = (\texttt{5A}, \texttt{90})$, $g_3 = (\texttt{90}, \texttt{00})$, $g_4 = (\texttt{00}, \texttt{03})$, $g_5 = (\texttt{03}, \texttt{00})$, $g_6 = (\texttt{00}, \texttt{00})$, $g_7 = (\texttt{00}, \texttt{00})$, $g_8 = (\texttt{00}, \texttt{04})$, $g_9 = (\texttt{04}, \texttt{00})$, $g_{10} = (\texttt{00}, \texttt{00})$, $g_{11} = (\texttt{00}, \texttt{00})$, $g_{12} = (\texttt{00}, \texttt{FF})$, $g_{13} = (\texttt{FF}, \texttt{FF})$, $g_{14} = (\texttt{FF}, \texttt{00})$, and $g_{15} = (\texttt{00}, \texttt{00})$.

We use 'term frequency - inverse document frequency' $(tf - idf)$ [6] to obtain the weight of each byte n-grams, 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 $tf_{i,j}$ [6] 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: $idf_i = \log(|\mathscr{E}|/(|\mathscr{E} : t_i \in e|)$ where $|\mathscr{E}|$ is the total number of executables and $|\mathscr{E} : t_i \in e|$ is the number of documents containing the n-gram t_i.

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

3 Overview of LLGC

Learning with Local and Global Consistency (LLGC) [18] is a semi-supervised algorithm that provides *smooth* classification with respect to the intrinsic structure revealed by known labelled and unlabelled points. The method is a simple iteration algorithm that constructs a smooth function coherent to the next assumptions: (i) nearby points are likely to have the same label and (ii) points on the same structure are likely to have the same label [18].

Formally, the algorithm is stated as follows. Let $\mathscr{X} = \{x_1, x_2, ..., x_{\ell-1}, x_\ell\} \subset \mathbb{R}^m$ be the set composed of the data instances and $\mathscr{L} = \{1, ..., c\}$ the set of labels (in our case, this set comprises two classes: malware and legitimate software) and $x_u(\ell + 1 \leq u \leq n)$ the unlabelled instances. The goal of LLGC (and every semi-supervised algorithm) is to predict the class of the unlabelled instances. $\mathscr{F}$ is the set of $n \times c$ matrices with non-negative entries, composed of matrices $F = [F_1^T, ..., F_n^T]^T$ that match to the classification on the dataset $\mathscr{X}$ of each instance x_i. with the label assigned by $y_i = \text{argmax}_{j \leq c} F_{i,j}$. F can be defined as a vectorial function such as $F : \mathscr{X} \rightarrow \mathbb{R}^c$ to assign a vector F_i to the instances x_i. Y is an $n \times c$ matrix such as $Y \in F$ with $Y_{i,j} = 1$ when x_i is labelled as $y_i = j$ and $Y_{i,j} = 0$ otherwise Considering this, the LLGC algorithm performs as follows:

if $i \neq j$ *and* $W_{i,i} = 0$ **then**
  Form the affinity matrix W defined by $W_{i,j} = \exp\left(\frac{-||x_i - x_j||^2}{2 \cdot \sigma^2}\right)$;
Generate the matrix $S = D^{-1/2} \cdot W \cdot D^{-1/2}$ where D is the diagonal matrix with its (i,i) element equal to the sum of the i-th row of W;
while $\neg$ *Convergence* **do**
  $F(t+1) = \alpha \cdot S \cdot F(t) + (1 - \alpha) \cdot Y$ where α is in the range $(0,1)$;
F^* is the limit of the sequence $\{F(t)\}$;
Label each point x_i as $\text{argmax}_{j \leq c} F^*_{i,j}$;

Fig. 2 LLGC algorithm.

The algorithm first defines a pairwise relationship W on the dataset $\mathscr{X}$ setting the diagonal elements to zero. Suppose that a graph $G = (V, E)$ is defined within $\mathscr{X}$, where the vertex set V is equal to $\mathscr{X}$ and the edge set $\mathscr{E}$ is weighted by the values in W. Next, the algorithm normalises symmetrically the matrix W of G. This

step is mandatory to assure the convergence of the iteration. During each iteration each instance receives the information from its nearby instances while it keeps its initial information. The parameter α denotes the relative amount of the information from the nearest instances and the initial class information of each instance. The information is spread symmetrically because S is a symmetric matrix. Finally, the algorithm sets the class of each unlabelled specimen to the class of which it has received most information during the iteration process.

4 Empirical Validation

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 LLGC?* 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[2]. Although they had already been labelled according to their family and variant names, we analysed them using Eset Antivirus[3] 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.

Hereafter, 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 a operation represented by an operational code needs in machine code and it is a widely-used n-gram length in the literature (e.g., [14, 4]) 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 1,000 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 [8].

Next, 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 LLGC in detecting unknown malware. We did not use cross-validation because in the validation we do not want to test the performance of the classifier when a fixed size of training instances is used iteratively. Otherwise, we employ a variable number of training instances and try to predict the class of the remaining ones using LLGC in order to determine which is the best training set size. In this case, the training instances are the labelled ones whereas the unlabelled ones are the ones in the test dataset. In particular, we used the LLGC implementation provided by the *Semi-Supervised Learning and Collective Classification* package[4] for the

[2] http://vx.netlux.org/

[3] http://www.eset.com/

[4] Available at:http://www.scms.waikato.ac.nz/∼fracpete/projects/collective-classification/downloads.html

well-known machine-learning tool WEKA [2]. Specifically, we configured it with a transductive stochastic matrix W [18] and we employed the Euclidean distance.

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 the hits of the classifiers divided by the number of instances in the whole dataset: $Accuracy(\%) = (TP+TN)/(TP+FP+TP+TN)$ Besides, we measured the *Area Under the ROC Curve* (AUC) that establishes the relation between false negatives and false positives [17]. The ROC curve is obtained by plotting the TPR against the FPR. All the these measures refer to the test instances.

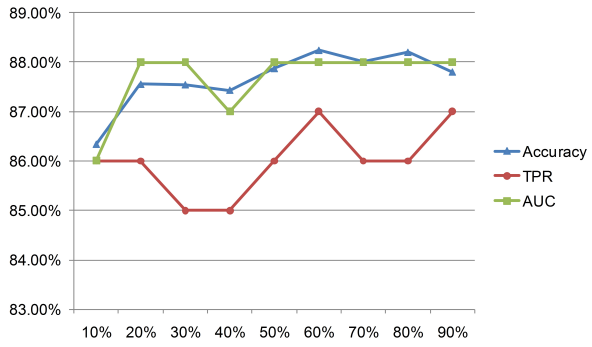

Fig. 3 Accuracy, TPR and AUC results. The X axis represents the percentage of labelled instances. The best AUC results with a 50% size for the labelled dataset.

Fig. 3 and Fig. 4 show the obtained results. The best results in terms of AUC were obtained with a training set containing 50% of labelled instances. These results indicate that we can reduce the efforts of labelling software in a 50% while maintaining a AUC higher than 88%. In terms of accuracy, the best results were achieved with a training size of 65%.

Previous supervised learning obtains better results (above 90% of accuracy [14, 4, 8]) than this semi-supervised approach. However, the main contribution of this paper is the reduction in the number of required labelled instances while maintaining a relative high precision. We consider that these results are significant for the anti-malware industry. The reduction of the efforts required for unknown malware can help to deal with the increasing amount of new malware.

However, because of the static nature of the features we used with LLGC, it cannot counter *packed* malware. Packed malware is produced by cyphering the payload

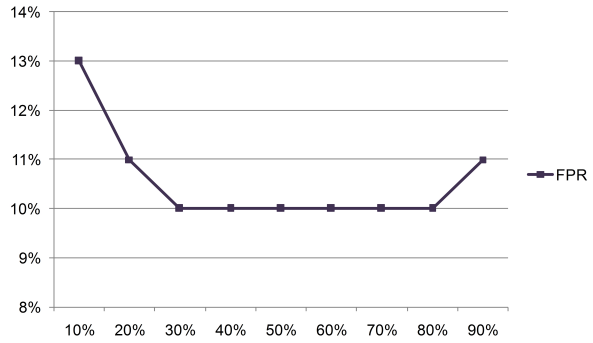

Fig. 4 FPR results. The X axis represents the percentage of labelled instances. In particular, the best results were obtained with a size greater than 30%.

of the executable and having it deciphered when finally loaded into memory. Indeed, broadly-used static detection methods can deal with packed malware only by using the signatures of the packers. Accordingly, dynamic analysis seems to be a more promising solution to this problem [3]. One solution for this obvious limitation of our malware detection method is the use of a generic dynamic unpacking schema such as PolyUnpack [10], Renovo [3], OmniUnpack [5] and Eureka [16].

5 Concluding Remarks

Unknown malware detection has become an important topic of research and concern owing to the growth of malicious code in recent years. Moreover, it is well known that the classic signature methods employed by antivirus vendors are no longer completely effective in facing the large volumes of new malware. Therefore, signature methods must be complemented with more complex approaches that provide the detection of unknown malware families. While machine-learning methods are a suitable approach for unknown malware, they require a high number of labelled executables for each classes (i.e., malware and benign datasets). Since it is difficult to obtain such amounts of labelled data in a real-word environment, a time-consuming process of analysis is mandatory.

In this paper, we propose for the fist time the use of a semi-supervised learning approach for unknown malware detection. This learning technique does not need a large amount of labelled data; it only needs several instances to be labelled. Therefore, this methodology can reduce efforts in unknown malware detection. By labelling 50% of the software, we can achieve results with more than 86% of accuracy.

Future work will be focused on three main directions. First, we plan to extend our study of semi-supervised learning approaches by applying more algorithms to this issue. Second, we will use different features for training these kinds of models.

Finally, we will focus on facing packed executables with a hybrid dynamic-static approach.

References

1. Chapelle, O., Schölkopf, B., Zien, A.: Semi-supervised learning. MIT Press (2006)
2. Garner, S.: Weka: The Waikato environment for knowledge analysis. In: Proceedings of the New Zealand Computer Science Research Students Conference, pp. 57–64 (1995)
3. Kang, M., Poosankam, P., Yin, H.: Renovo: A hidden code extractor for packed executables. In: Proceedings of the 2007 ACM workshop on Recurring malcode, pp. 46–53 (2007)
4. Kolter, J., Maloof, M.: Learning to detect malicious executables in the wild. In: Proceedings of the 10^{th} ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 470–478. ACM New York, NY, USA (2004)
5. Martignoni, L., Christodorescu, M., Jha, S.: Omniunpack: Fast, generic, and safe unpacking of malware. In: Proceedings of the 23^{rd} Annual Computer Security Applications Conference (ACSAC), pp. 431–441 (2007)
6. McGill, M., Salton, G.: Introduction to modern information retrieval. McGraw-Hill (1983)
7. Morley, P.: Processing virus collections. In: Proceedings of the 2001 Virus Bulletin Conference (VB2001), pp. 129–134. Virus Bulletin (2001)
8. Moskovitch, R., Stopel, D., Feher, C., Nissim, N., Elovici, Y.: Unknown malcode detection via text categorization and the imbalance problem. In: Proceedings of the 6^{th} IEEE International Conference on Intelligence and Security Informatics (ISI), pp. 156–161 (2008)
9. Ollmann, G.: The evolution of commercial malware development kits and colour-by-numbers custom malware. Computer Fraud & Security **2008**(9), 4–7 (2008)
10. Royal, P., Halpin, M., Dagon, D., Edmonds, R., Lee, W.: Polyunpack: Automating the hidden-code extraction of unpack-executing malware. In: Proceedings of the 22^{nd} Annual Computer Security Applications Conference (ACSAC), pp. 289–300 (2006)
11. Santos, I., Brezo, F., Nieves, J., Penya, Y., Sanz, B., Laorden, C., Bringas, P.: Idea: Opcode-sequence-based malware detection. In: Engineering Secure Software and Systems, *LNCS*, vol. 5965, pp. 35–43 (2010). URL http://dx.doi.org/10.1007/978-3-642-11747-3_3. 10.1007/978-3-642-11747-3_3
12. Santos, I., Penya, Y., Devesa, J., Bringas, P.: N-Grams-based file signatures for malware detection. In: Proceedings of the 11^{th} International Conference on Enterprise Information Systems (ICEIS), Volume AIDSS, pp. 317–320 (2009)
13. Schapire, R.: The boosting approach to machine learning: An overview. Lecture Notes in Statistics pp. 149–172 (2003)
14. Schultz, M., Eskin, E., Zadok, F., Stolfo, S.: Data mining methods for detection of new malicious executables. In: Proceedings of the $22^n d$ IEEE Symposium on Security and Privacy., pp. 38–49 (2001)
15. Shafiq, M., Khayam, S., Farooq, M.: Embedded Malware Detection Using Markov n-Grams. Lecture Notes in Computer Science **5137**, 88–107 (2008)
16. Sharif, M., Yegneswaran, V., Saidi, H., Porras, P., Lee, W.: Eureka: A Framework for Enabling Static Malware Analysis. In: Proceedings of the European Symposium on Research in Computer Security (ESORICS), pp. 481–500 (2008)
17. Singh, Y., Kaur, A., Malhotra, R.: Comparative analysis of regression and machine learning methods for predicting fault proneness models. International Journal of Computer Applications in Technology **35**(2), 183–193 (2009)
18. Zhou, D., Bousquet, O., Lal, T., Weston, J., Schölkopf, B.: Learning with local and global consistency. In: Advances in Neural Information Processing Systems 16: Proceedings of the 2003 Conference, pp. 595–602 (2004)
19. Zhou, Y., Inge, W.: Malware detection using adaptive data compression. In: Proceedings of the 1st ACM workshop on Workshop on AISec, pp. 53–60. ACM New York, NY, USA (2008)