Expert Systems with Applications xxx (2014) xxx–xxx

Contents lists available at ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

Novel active learning methods for enhanced PC malware detection in windows OS

Q1 Nir Nissim [*], Robert Moskovitch, Lior Rokach, Yuval Elovici

Telekom Innovation Laboratories at Ben-Gurion University, Department of Information Systems Engineering, Ben-Gurion University of the Negev, P.O.B 653, Be'erSheva 84105, Israel

A R T I C L E I N F O

Keywords:
Malware
Malicious code
Machine Learning
Active learning
SVM

A B S T R A C T

The formation of new malwares every day poses a significant challenge to anti-virus vendors since anti-virus tools, using manually crafted signatures, are only capable of identifying known malware instances and their relatively similar variants. To identify new and unknown malwares for updating their anti-virus signature repository, anti-virus vendors must daily collect new, suspicious files that need to be analyzed manually by information security experts who then label them as malware or benign. Analyzing suspected files is a time-consuming task and it is impossible to manually analyze all of them. Consequently, anti-virus vendors use machine learning algorithms and heuristics in order to reduce the number of suspect files that must be inspected manually. These techniques, however, lack an essential element – they cannot be daily updated. In this work we introduce a solution for this updatability gap. We present an active learning (AL) framework and introduce two new AL methods that will assist anti-virus vendors to focus their analytical efforts by acquiring those files that are most probably malicious. Those new AL methods are designed and oriented towards new malware acquisition. To test the capability of our methods for acquiring new malwares from a stream of unknown files, we conducted a series of experiments over a ten-day period. A comparison of our methods to existing high performance AL methods and to random selection, which is the naïve method, indicates that the AL methods outperformed random selection for all performance measures. Our AL methods outperformed existing AL method in two respects, both related to the number of new malwares acquired daily, the core measure in this study. First, our best performing AL method, termed "Exploitation", acquired on the 9th day of the experiment about 2.6 times more malwares than the existing AL method and 7.8 more times than the random selection. Secondly, while the existing AL method showed a decrease in the number of new malwares acquired over 10 days, our AL methods showed an increase and a daily improvement in the number of new malwares acquired. Both results point towards increased efficiency that can possibly assist anti-virus vendors.

1. Introduction

The number of new malicious files is constantly increasing Q3 (Schultz, Eskin, Zadok, & Stolfo, 2001). The term "malicious software" (malware) commonly refers to pieces of code, not necessarily executable files, that contain malicious functionality. Malwares are classified into four main categories mainly based on their transport mechanisms: worms, viruses, Trojans, and backdoors (Shabtai, Moskovitch, Elovici, & Glezer, 2009).

Unlike in the past, creating a malware today is relatively easy because malicious code libraries are shared between attackers. For example, on the website http://vx.netlux.org/, malware developers share tools for facilitating a generation of new malwares. Furthermore, attackers have become much more sophisticated, creating malicious code files that seem to act like benign files but are harder to detect, such as the case with Trojan horses (CERT, 1999). Additionally, attackers either detect new vulnerabilities or follow announcements about the latest vulnerabilities before creating malicious codes to exploit these vulnerabilities. Attackers also know that anti-virus vendors distribute patches to their anti-virus packages relatively slowly, providing the virus with a window of opportunity to attack and spread.

Current anti-virus packages rely mainly on signature-based detection of malwares that have already been seen. In addition, sets of heuristics and rules are defined to look for generic and distinguishing characteristics of malwares in unknown files. These

Q2 * Corresponding author. Tel.: +972 0578121406.
E-mail addresses: nirni@bgu.ac.il (N. Nissim), robertmo@bgu.ac.il (R. Moskovitch), liorrk@bgu.ac.il (L. Rokach), elovici@bgu.ac.il (Y. Elovici).

http://dx.doi.org/10.1016/j.eswa.2014.02.053
0957-4174/© 2014 Elsevier Ltd. All rights reserved.

various methods depend on the presence of a previously detected malware. Consequently, a new unknown malware (with new characteristics) will not be detected since its signature does not bear any similarity (including rules and heuristics) to signatures in the repository. Until an update is distributed damages can be extensive. "Slammer" was the fastest computer worm recorded in history. Within 10 min (Moore et al., 2003), it infected about 75,000 vulnerable hosts. Another harmful and famous worm, "Code Red", infected 359,000 hosts within 14 h (David Moore, Shannon, & Claffy, 2002).

In order to accurately and quickly detect the newest malicious files, anti-virus companies devote considerable effort to preserving the updatability of their signature repository of malicious code files. These efforts include monitoring new and unknown malicious code files sent over the Internet or using various types of honeypots to catch the malicious files (Mokube, 2007; Provos & Holz, 2008). This mission is complicated and time-consuming, especially when done manually (Iyatiti Mokube & Adams, 2007).

A trivial solution to the problem of finding new malicious files would be to randomly select files from the Internet in order to determine whether these files were malicious or not. As a result, the signature repository would be updated and the anti-virus application enhanced. However, such a strategy is obviously inefficient and the chances of finding a new unknown malicious file by random selection is low given that the percentage of malicious files on the Internet amounts to about 10% (Moskovitch, Stopel, Feher, Nissim, & Elovici, 2008; Moskovitch, Stopel, et al., 2009).

In order to effectively analyze every day tens of thousands of new, potentially malicious code files, anti-virus vendors have integrated into the core of their signature repository update activities, a component of a detection model based on machine learning methods (Kiem, Thuy, & Quang, 2004). This component, which is responsible for determining what files are most likely to be malicious, is intended to reduce the number of files sent to the human expert for labeling. This approach, however, suffers from several shortcomings. First, it has to be constantly updated with new informative files to effectively maintain a high level of classification. Since there are dozens of new unlabeled files every day, the problem is determining which files should be acquired and analyzed by a human expert, and labeled as either malicious or benign. Additionally, this approach focuses on finding new malicious files in order to keep the signatures repository as updated as possible. However, this can only be done by updating the detection model with new, informative benign files that are also crucial for accurately discriminating between malicious and benign files.

In this paper we present a framework and active learning (AL) methods for frequently updating anti-virus software by focusing expert efforts on labeling those files that are most likely to be malware or on benign files that can possibly improve the detection model. Note that, both the anti-virus and the detection model must be updated with new and labeled files. Using an updated detection model we can detect new malwares that are used to sustain the anti-virus signature repository.

The framework that we present maintains a detection model based on a classifier that is trained on a representative set of files (malicious and benign) using static analysis. The advantages of the detection model is that it has generalization capabilities that enable it to detect new unknown malwares at a high probability level even before the files have infected the host computer (due to static analysis).

While machine learning has been successfully used for inducing malware detection models (Abou-Assaleh, Cercone, Keselj, & Sweidan, 2004; Henchiri & Japkowicz, 2006; Jang, Brumley, & Venkataraman, 2011; Kolter & Maloof, 2004; Kolter & Maloof, 2006; Moskovitch Stopel, et al., 2008; Moskovitch, Stopel, et al., 2009; Schultz et al., 2001; Tahan, Rokach, & Shahar, 2012), most studies focus on passive learning. We, on the other hand, focus on active learning and present novel AL methods that have been especially designed to enrich the signature repository of the anti-virus with new malwares in the course of several days thus ensuring that the detection model is as up-to-date as possible. In a series of experiments that simulate reality, we show that AL can efficiently scan an ongoing stream of executable files and select the most informative ones for manual analysis by human experts. These files are then used to update the detection model, with the malwares being used to update the signatures repository. Accordingly, both the detection model and ant-virus are being updated daily and detection capabilities improved.

We are aware of the limitations of static analysis in malware detection but circumvent these difficulties by focusing on an AL approach rather than specific analysis, which can be either static or dynamic. The use of AL concepts actually leverages the knowledge of the detection model, therefore our approach is effective in both analysis instances (Moskovitch, Nissim, & Elovici, 2009; Nissim, Moskovitch, Rokach, & Elovici, 2012).

Our contributions in this paper are twofold:

– First, we present a framework for efficiently updating PC anti-viruses tools on a daily basis.
– Secondly, we present two AL methods for acquiring new malwares. The two methods, termed Exploitation and Combination, acquired a larger number of malwares daily compared to the existing AL method SVM-Margin. Our methods can be used for any domain for which an acquisition of specific class is needed.

The rest of the paper is organized as follows. Section 2 surveys related work while Section 3 presents the dataset. Section 4 describes the methods we applied and introduces the proposed framework for improving detection capabilities. Section 5 discusses the measures used for measuring and evaluating the proposed framework followed by a presentation of the experiment's design. Section 6 presents the results from the proposed approach, while Section 7 discusses how the framework copes with potential attacks. Finally, Section 8 provides conclusions, discusses the advantages of the described framework, and suggests future research directions.

2. Background

Over the past 15 years, a number of studies have investigated the possibility of detecting an unknown malcode based on its binary code. Schultz et al. (2001) were the first to introduce the use of classification methods on static representation using various sets of features including program headers, string features, and byte sequence features. They used standard classifiers that outperform a signature-based method (anti-virus).The use of *n*-grams to represent binary files was further performed by Abou-Assaleh et al. (2004), Kolter and Maloof (2004) and Schultz et al. (2001) using various combinations of classifiers and feature selection methods. Later Kolter and Maloof (2006) extended their work into classifying malwares into families (classes) based on the functions of their respective payloads. This approach compared to previous experiments was not successful. This lack of success indicated the importance of maintaining the training set by acquiring new executables, an approach that we propose in this paper. Due to the large number of features extracted by the *n*-grams, the feature selection issue was specifically investigated by several researchers including Henchiri and Japkowicz (2006) who presented a hierarchical feature selection approach and Schultz et al. (2001). Working on a very large test collection of more than 30,000 executable files, Moskovitcdh et al. investigated the problem of imbalance in unknown malicious code detection using *n*-grams (Moskovitch, Stopel, et al., 2008; Moskovitch, Nissim, Englert, & Elovici, 2008) and op codes (Moskovitch, Feher, et al., 2008). They subsequently extended their work (Moskovitch, Stopel, et al., 2009) and provided additional insights regarding their results such as that their methods able to classify executables based on the function of their payload. Menahem, Shabtai, Rokach, and Elovici (2009) presented a framework for improving malware detection by applying a multi-inducer ensemble that actually leverages the knowledge of several different classifiers by utilizing the classification decisions of every classifier for calculating the final classification decision. In 2011, Jang et al. (2011) presented Bitshred, a system that is mainly designed for malware similarity analysis and used for sorting and clustering on a large-scale. They used feature hashing to reduce the feature space and thus made the triage of the malware faster and more efficient. More recent studies have shown the advantage of these automatic methods both in time and in accuracy (Nataraj et al., 2011a; Nataraj et al., 2011b; Tahan et al., 2012).

Dynamic analysis, also known as behavioral analysis, is aimed at tracing the behavior of the file and its implications for the environment in which it is being executed. This type of analysis, which has been thoroughly explored over the past several years, presents versatile methods for detecting an unknown malware based on its behavior. These methods are aimed at detecting malicious activity that cannot be discovered using static analysis (discussed above). However, since we are focusing on static analysis we will just mention several notable studies based on dynamic analysis: CWSandbox (Willems, Holz, & Freiling, 2007) Rotaluḿe (Sharif, Lanzi, Giffin, & Lee, 2009), Polyunpack (Rieck, Holz, Willems, Düssel, & Laskov, 2008; Rieck, Trinius, Willems, & Holz, 2011; Royal, Halpin, Dagon, Edmonds, & Lee, 2006), BitBlaze (Song et al., 2008), McBoost framework (Moser, Kruegel, & Kirda, 2007a; Perdisci, Lanzi, & Lee, 2008), K-Tracer (Bayer, Comparetti, Hlauschek, Kruegel, & Kirda, 2009; Jacob, Debar, & Filiol, 2009; Kolbitsch et al., 2009; Lanzi, Sharif, & Lee, 2009).

Static analysis methods have several advantages over dynamic analysis. First, they are virtually undetectable – the PC file cannot detect that it is being analyzed since it is not being executed. While it is possible to create static analysis "traps" to deter analysis, these traps can actually be used as a contributing feature for detecting malware. In addition, since static analysis is relatively efficient and fast, it can be performed in an acceptable timeframe and consequently will not cause bottlenecks. Static analysis is also simple to implement, monitor and measure. Moreover, it scrutinizes the file's "genes" and not its current behavior which can be changed or delayed to an unexpected time in order to evade the dynamic analysis. An additional aspect is that static analysis can be used for a scalable pre-check of malwares.

Labeling examples, which is crucial for the learning process, is often an expensive task since it involves human experts. Active learning (AL) was designed to reduce the labeling efforts, by actively selecting the examples with the highest potential contribution to the learning process of the detection model. AL is roughly divided into two major approaches: membership queries (Angluin, 1988), in which examples are artificially generated from the problem space and selective-sampling (Lewis & Gale, 1994), in which examples are selected from a pool, which is used in this study.

Studies in several domains have successfully applied active learning in order to reduce the time and money required for labeling examples. Unlike random learning, in which a classifier randomly selects examples from which to learn, the classifier actively indicates the specific examples that should be labeled and which are commonly the most informative examples for the training task. SVM-Simple-Margin (Tong & Koller, 2000–2001) is an existing AL method considered in our experiments. Moskovitch, Nissim, et al. (2008) and Nissim et al. (2012) successfully used AL methods to detect unknown computer worms, enhancing the methods proposed in Moskovitch, Elovici, and Rokach (2008), Moskovitch et al. (2007) and Stopel, Boger, Moskovitch, Shahar, and Elovici (2006). Using AL in such cases was very useful in removing noisy examples and in selecting the most informative examples. Other studies, using AL for unknown malware detection (Moskovitch, Nissim, et al., 2008; Moskovitch, Nissim, et al., 2009), demonstrate a somewhat limited approach, since an attempt is made to replace an antivirus with AL, which is unrealistic. Additionally, in their experimental work, these researchers do not refer to the real and crucial need of repeatingly and frequently updating the detection components along time.

In this paper we provide an answer to these shortcomings. We used the detection model in combination with the AL to assist and improve the performance of the detection model and the updatability of the anti-virus tool. This makes our framework solution much more practical and secure. Consequently our framework is feasible for widespread use since it can efficiently enrich the signature repository and quickly update the anti-virus tool on a daily basis. To look at it in a slightly different way, the framework acts as a consultant, suggesting which of the suspected files should be sent to a human expert for labeling. Additionally, we present in our paper a comprehensive set of experiments that focus on the daily process of updating the detection model and the signature repository by intelligently acquiring malicious files. Presenting the framework on a daily basis with only new files that do not appear in the training set or the signature repository results in a highly accurate experiment that closely reflects reality.

In light of the advantages and disadvantages of dynamic and static analysis briefly presented above, we decided to focus on the static approach in our work. Our aim was to provide an applicable active learning framework that would be empirically evaluated over a large set of PC files in a reasonable execution time.

3. Dataset creation

3.1. Dataset collection

We created a dataset of malicious and benign executables for the Windows operating system, the most commonly attacked system. We acquired 7688 malicious files from the VX Heaven website[1] that contains several types of malicious files such as viruses, worms, Trojans and malware (probes). To identify the files, we used the Kaspersky[2] anti-virus software and the Windows version of the Unix 'file' command for file type identification. We also included the obfuscated executables that VX Heaven provides. Among these executables, some were obfuscated using compression or encryption while others were obfuscated using both techniques. As was the case with Kolter and Maloof (2006), we were not informed which of the files were obfuscated and which were not. The files in the benign set, including executable and dynamic-link library (DLL) files, were gathered from computers running the Windows operating system. The benign set contained 22,735 files that the Kaspersky anti-virus program reported as being completely virus-free. To the best of our knowledge none of the benign files were obfuscated.

3.2. Dataset representation using text categorization

In order to detect and acquire unknown malicious codes, we implemented well-studied concepts from the field of information

1 http://vx.netlux.org.
2 http://www.kaspersky.com.

retrieval (IR) and, more specifically, the text categorization domain. In the course of implementing our task, binary files (executables) are parsed and *n*-gram terms are extracted. Each *n*-gram term (the sequence of bytes in the binary code) is analogous to words in the textual domain. Here we present the IR concepts used in this study. Salton, Wong, and Yang (1975), presented the vector space model to represent a textual file as a bag-of-words. For clarity, a word in our case is a binary sequence of bytes. For instance, a 4-gram word will be in the form of 0101, 1110 among a total of 16 possibilities. After parsing the text and extracting the words, a vocabulary of the entire collection of words is constructed. Each of these words may appear multiple times in a document or not at all. A vector of terms is created such that each element in the vector represents the term frequency (TF) in the document. Eq. (1) shows the definition of a normalized TF in which the TF is divided by the frequency of the maximally appearing term in the document with values in the range of [0–1]. Another common representation is the TF inverse document frequency (TFIDF) that combines the term frequency in the document with its frequency in the document's collection, as shown in Eq. (2), in which the term's (normalized) TF value is multiplied by the IDF = log (*N*/*n*), where *N* is the number of documents in the entire file collection and *n* is the number of documents in which the term appears.

$$\mathrm{TF} = \frac{\text{term frequency}}{\max(\text{term frequency in document})} \tag{1}$$

$$\mathrm{TFIDF} = \mathrm{TF}.\log(\mathrm{DF}), \tag{2}$$

where

$$\mathrm{DF} = \frac{N}{n}.$$

3.3. Data preparation and feature selection

We parsed the binary code of the executable files using several *n*-gram length sliding windows. The parsing was performed on the raw file without any decryption or decompression. Vocabularies of 16,777,216, 1,084,793,035, 1,575,804,954 and 1,936,342,220 for 3-gram, 4-gram, 5-gram and 6-gram, respectively, were extracted. Later, the TF and TFIDF representations were calculated for each *n*-gram in each file.

In machine learning applications, the large number of features in many domains, many of which do not contribute to the accuracy of the detection model and may even decrease it, present a huge problem. Moreover, in our task, a reduction in the number of features is crucial for practical reasons but must be performed while simultaneously maintaining a high level of accuracy. This is due to the fact that, as shown earlier, the vocabulary size may exceed billions of features, far more than can be processed by any feature selection tool within a reasonable period of time. Additionally, it is important to identify those features that appear in most of the files in order to avoid zeroed representation vectors. Thus, the features with the highest document frequency (DF) value (Eq. (2)) were initially extracted. Based on the DF measure, two sets were selected; the top 5500 terms and the top 1000–6500 terms. The use of a set consisting of the top 1000–6500 sets of features was motivated by the removal of stop-words, as is often done in information retrieval for common words. Later, feature selection methods were applied to each of these two sets. Since feature selection preprocessing is not the focus of this paper, we will describe it very briefly. We used a filter approach (Bi, Bennett, Embrechts, Breneman, & Song, 2003) to compare the performances of the different classification algorithms, where the measure was independent of any classification algorithm. In a filter approach, a measure is used to quantify the correlation of each feature to the class (malicious or benign) and to estimate its expected contribution to the classification task. Three feature selection measures were used. As a baseline we used the document frequency measure DF (Eq. (2)); Gain ratio (GR) Mitchell, 1997; and Fisher score (Golub et al., 1999). Eventually the following top features 50, 100, 200 300, 1000, 1500 and 2000 were selected using each feature selection method.

4. Machine learning methods and the suggested framework

4.1. Support vector machines (SVM)

We employed the SVM classification algorithm using the radial basis function (RBF) kernel in a supervised learning approach. There are several reasons for using SVM as the classification algorithm. First and foremost, SVM has been successfully used in detecting worms, as three previous works indicated (Masud, Khan, & Thuraisingham, 2007; Wang, Yu, Champion, Fu, & Xuan, 2007; Yu, Xin-cai, & Hai-bin, 2008). Moreover, in the first work (Wang et al., 2007) the authors noted that "SVM learns a black-box classifier that is hard for worm writers to interpret". As Moskovitch, Nissim, et al. (2009) have presented, SMV has proven to be very efficient when combined with AL methods. Lastly, SVM was also successfully used by Chen et al. (2012) in their "Malware Evaluator", a system that classifies malwares into species and detects zero day attacks based on information and features provided by anti-virus vendors about every known malware and its breed.

We now briefly introduce the SVM classification algorithm and the active learning principles and their implementation as used in this study. SVM is a binary classifier that finds a linear hyperplane that separates given examples into two specific classes, yet is also capable of handling multiclass classification (Vapnik, 1982). As Joachims (1999) demonstrated, SVM is widely known for its capability for handling large amounts of features, such as texts.

We used the Lib-SVM implementation of Chang and Lin (2011), which also supports multiclass classification. Given a training set in which an example is a vector $x_i = \langle f_1, f_2, \ldots, f_m \rangle$, where f_i' is a feature, and labeled by $y_i = \{-1, +1\}$, the SVM attempts to specify a linear hyperplane with the maximal margin defined by the maximal (perpendicular) distance between the examples of the two classes. Fig. 1 illustrates a two-dimensional space where the examples are located according to their features. The hyperplane splits them according to their label.

The examples lying closest to the hyperplane are the "supporting vectors". *W*, the Normal of the hyperplane, is a linear combination of the most important examples (supporting vectors) multiplied by LaGrange multipliers (alphas), as can be seen in Eq. (5). Since the dataset in the original space cannot often be linearly separated, a kernel function *K* is used. SVM actually projects the examples into a higher dimensional space in order to create a linear separation of the examples. Note that when the kernel function

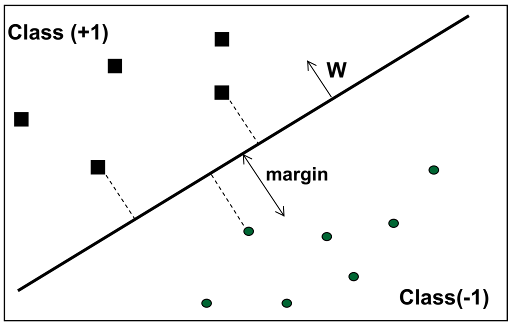

Fig. 1. An SVM that separates the training set into two classes, with a maximal margin, in a two-dimensional space.

satisfies Mercer's condition, as Burges (1998) explained, K can be presented using Eq. (3), where Φ is a function that maps the example from the original feature space into a higher dimensional space while K relies on the "inner product" between the mappings of examples x_1, x_2. For the general case, the SVM classifier will be in the form shown in Eq. (4), where n is the number of examples in the training set; K is the kernel function; alpha is the LaGrange multiplier that defines the linear combination of the Normal W; and y_i is the class label of support vector X_i.

$$K(x_1, x_2) = \Phi(x_1)\Phi(x_2) \tag{3}$$

$$f(x) = sign(w \cdot \Phi(x)) = sign\left(\sum_{1}^{n} \alpha_i y_i K(x_i x)\right) \tag{4}$$

$$w = \sum_{1}^{n} \alpha_i y_i \Phi(x_i) \tag{5}$$

Two commonly used kernel functions were used. First, the polynomial kernel, as shown in Eq. (6), creates polynomial values of degree p, where the output depends on the direction of the two vectors, examples x_1, x_2, in the original problem space. Note that a private case of a polynomial kernel, where $p = 1$, is actually the linear kernel. The second is the radial basis function (RBF), as shown in Eq. (7), in which a Gaussian function is used as the RBF and the output of the kernel depends on the Euclidean distance of examples x_1, x_2.

$$K(x_1, x_2) = (x_1 \cdot x_2 + 1)^P \tag{6}$$

$$K(x_1, x_2) = e^{\left(-\frac{\|x_1 - x_2\|^2}{2\sigma^2}\right)} \tag{7}$$

We now provide a formal and rigorous explanation about the algorithm that builds a SVM classifier from a given training set D that includes N examples x_i with class label y_i. Each instance is in a vector form with n features f_i:

Training set: $D = \{x_i y_{i,}\}_{i=1}^{N}$
Instance in training set in vector form: $x_i = \{f_i^1, \ldots, f_i^n\}^T \in \mathbb{R}^2$
Class labels: $y_i \in \{+1, -1\}$

According to Vapnik (1998) original definition and formulation, the SVM classifier that will be induced from the training set D, should satisfy the following conditions in Eq. (8) where W is the weight vector, b is the bias and ϕ is a function that maps the examples from the original problem space (called weight space as well) into a higher dimensional space referred to as the feature space. Eq. (8) can be simply summarized as Eq. (9):

$$\begin{cases} \text{If} \quad y_i = +1 \quad \text{then} \quad w^T\phi(x_i) + b \geqslant +1 \\ \text{If} \quad y_i = -1 \quad \text{then} \quad w^T\phi(x_i) + b \leqslant -1 \end{cases} \tag{8}$$

$$y_i[w^T\phi(x_i) + b] \geqslant 1, \quad \text{where} \quad i = 1, \ldots, N \tag{9}$$

Eq. (9) actually defines the construction of two parallel surfaces that lie at similar distances from both sides of the separating hyperplane as depicted in Fig. 1. The separating hyperplane can be seen in Eq. (10), where the decision of the classifier (positive or negative class) on a given example x is provided by the sign calculated from Eq. (11):

$$w^T\phi(x_i) + b = 0 \tag{10}$$

$$sign(w^T\phi(x_i) + b) \tag{11}$$

As explained above, since in reality most classification problems cannot be linearly separated, a slack variable (ξ_i) is used in order to permit misclassifications in the training phase and to compensate accordingly. Thus the primal problem can be defined by optimization as in Eq. (12), where C is a parameter for tuning between the importance of classification errors and the width of the margin.

$$Min_{(w,b,\xi)} \frac{1}{2} w^T w + C\sum_{i=1}^{N} \xi_i \tag{12}$$

subject to

$$\begin{cases} y_i[w^T\phi(x_i) + b] \geqslant 1 - \xi_i, & \text{where} \quad i = 1, \ldots, N \\ \xi_i \geqslant 0, & \text{where} \quad i = 1, \ldots, N \end{cases}$$

The solution of the primal problem in Eq. (12) requires using a Lagrange multiplier α_i for every training instance, where the conditions of the primal problem result in a quadratic programming (QP) problem with Lagrange multiplier α_i. Those training instances x_i for which α_i is not zero, are called the support vectors and are therefore the only instances that play a role in identifying the separating hyperplane of the SVM. Since the primal problem in Eq. (12) is hard to solve, it can be converted into dual problem that will be easier to solve due to the fact that the decision variables are actually the support vectors. The dual problem can be seen in Eq. (13) where Q is a $N \times N$ semi-definite matrix, as defined in Eq. (14) that uses the kernel trick from Eq. (3), and e is a vector of all "ones".

$$Max_{(\alpha)} \frac{1}{2} \alpha^T Q \alpha - e^T \alpha \tag{13}$$

subject to

$$\begin{cases} 0 \leqslant \alpha_i \leqslant C, \quad \text{where} \quad i = 1, \ldots, N \\ y^T\alpha = 0 \end{cases}$$

$$Q_{ij} = y_i y_j K(x_i, x_j) \tag{14}$$

After solving the optimization problem, the SVM classifier is in the form presented in Eq. (4) as shown above.

4.2. Selective sampling and active learning methods

Since our framework aims to provide solutions to real problems it must be based on a sophisticated, fast, and selective high-performance sampling method. We compared our proposed AL method to several strategies described below. Q4

4.2.1. Random selection

Random selection is obviously not an active learning method yet it is actually the "lower bound" of the selection methods that will be discussed. As far as we know, no anti-virus tool uses an active learning method for preserving and improving its updatability. Consequently, we expect that all the AL methods will perform better than a selection process that is based on the random acquisition of files.

4.2.2. The SVM-Simple-Margin active learning method (SVM-Margin)

The SVM-Simple-Margin method (Tong & Koller, 2000–2001) (referred to as SVM-Margin in the discussion below) is directly related to the SVM classifier. As is well-known, using a kernel function, the SVM implicitly projects the training examples into a different (usually a higher dimensional) feature space denoted by F. In this space there is a set of hypotheses that are consistent with the training set. This means that these hypotheses create a linear separation of the training set. This set of consistent hypotheses is referred to as the version-space (VS). From among the consistent hypotheses, the SVM then identifies the best hypothesis with the maximal margin. To achieve a situation where the VS contains the most accurate and consistent hypothesis, the SVM-Margin AL method selects examples from the pool (of unlabeled examples) which reduces the number of hypotheses. Calculating the VS is complex and impractical where large datasets are concerned and therefore, this method is based on simple heuristics that depend on the relationship between the VS and SVM with the maximal margin. Practically speaking, examples that lie closest to the separating hyperplane (inside the margin), are more likely to be informative and new to the classifier. Consequently, these examples will be selected for labeling and acquisition.

This method, contrary to ours, selects the examples according to their distance from the separating hyperplane only to explore and acquire the informative files without relation to their classified labels. Thus it will not necessarily focus on selecting and acquiring malware instances. The SVM-Margin AL method is very fast and can be applied to real problems, yet, as its authors indicate (Tong & Koller, 2000–2001), this agility is achieved due to the fact that it is based on a rough approximation and relies on the assumption that the VS is fairly symmetric and that the hyperplane's normal (W) is centrally placed. It has been demonstrated, both in theory and practice, that these assumptions can fail significantly (Herbrich, Graepel, & Campbell, 2001). Indeed, the method may actually query instances whose hyperplane does not intersect the VS and therefore may not be at all informative. Moskovitch, Nissim, et al. (2009) used the SVM-Margin method for detecting instances of PC malware and according to their preliminary results, the method also assisted in updating the detection model but not the anti-virus application itself. In Moskovitch, Nissim, et al. (2009) the method was used for only one day-long trial and not for a period of several consecutive days as actually happens in reality. Accordingly, we thought it would be interesting to compare its performance to our proposed AL methods in a daily process of detection and acquisition experiments. Therefore, SVM-Margin is the base line AL method we expect to improve.

4.2.3. Exploitation: our proposed active learning method

Our method, which we term "Exploitation", is designed and based on SVM classifier principles. It is oriented towards selecting the examples that are probably the most malicious. More specifically, it selects the examples that lie furthest from the separating hyperplane. In our domain, detection of PC malware, this indicates that only the files that are most likely to be malware will be acquired. Our motivation for this set of actions is the desire to enhance the signature repository of the anti-virus tool with as many new malwares as possible. Thus for every file X that is suspected of being malicious, Exploitation rates its distance from the separating hyperplane using Eq. (15) based on the Normal of the separating hyperplane of the SVM classifier that serves as the detection model. As explained above, the separating hyperplane of the SVM is represented by W, which is the Normal of the separating hyperplane and actually a linear combination of the most important examples (supporting vectors), multiplied by LaGrange multipliers (alphas) and by the kernel function K that assists in achieving linear separation in higher dimensions. Accordingly, the distance calculation in Eq. (15) is simply done between example X and the Normal (W) that is presented in Eq. (5).

In Fig. 2, it can be observed that the files that were acquired (marked with a red circle) are those files that were classified as malicious and have maximum distance from the separating hyperplane. Acquiring several new malicious files that are quite similar and belong to the same virus family is considered a waste of manual analysis resources since these files will probably be detected by the same signature. Thus, acquiring one representative file for this set of new malicious files will serve the goal of efficiently updating the signatures repository. In order to adhere to the goal of enhancing the signature repository as much as possible, we also check the similarity between the selected files using the kernel farthest-first (KFF) method suggested by Baram, El-Yaniv, and Luz (2004). By using this method, we avoid acquiring examples that are quite

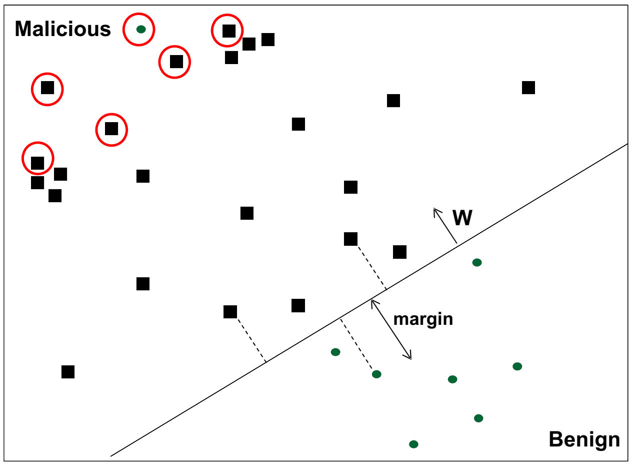

Fig. 2. The criteria by which Exploitation acquires new unknown malicious files. These files lie the farthest from the hyperplane and are regarded as representative files.

similar (the similarity is checked according to its representation in the SVM kernel space). Consequently, only the representative files that are most probably malicious are being selected. In case the representative file is detected as malware as a result of the manual analysis, then all its variants that were not acquired will be detected the moment the anti-virus is updated. And in case these files are not actually variants of the same malware, they will be acquired the following day as long as they are still most likely to be malware after the detection model has been updated. In Fig. 2, it can be observed that there are sets of relatively similar files (according to their distance in the kernel space). However, only the representative files that are most likely to be malwares are being acquired.

As is well-known, the SVM classifier defines the class margins using a small set of supporting vectors (i.e., PC files).While the usual goal is to improve the classification by uncovering (labeling) files from the margin area, in our case the primary goal is to acquire malwares for updating the AV. Actually the same number of files are acquired each day, but with Exploitation we attempt to better explore the "malicious side" of the incoming files), resulting sometimes in the discovery of also benign files (these files will probably become support vectors and update the classifier). In Fig. 2 we can observe an example of a benign file lying deep inside the malicious side. Contrary to SVM-Margin that explores examples that lie inside the SVM Margin, Exploitation explores the "malicious side" more efficiently as part of an effort to discover new and unknown malicious files that are essential for the frequent update of the antivirus signature repository.

The distance calculation required for each instance in this method is quite fast and equal to the time it takes to classify an instance in a SVM classifier. Consequently, it is a very practical and fast method that can provide the ranking for acquisition in a short time frame. It is therefore applicable for products working in real-time.

$$Dist(X) = \left(\sum_{1}^{n} \alpha_i y_i K(x_i x)\right) \tag{15}$$

4.2.4. Combination: a combined active learning method

The combination method lies between SVM-Margin and Exploitation. On the one hand, the combination method will start with a phase in which it will acquire examples based on SVM-Margin criteria in order to acquire the most informative files. Consequently, both malicious and benign files will be acquired. This Exploration-type phase is important in order to enable the detection model to discriminate between malicious and benign files. On the other hand, the combination method will then try to maximally update the signature repository in an Exploitation-type phase. This means that in the early acquisition period, in the first part of the day, SVM-Margin predominates over Exploitation. As the day progresses, Exploitation becomes predominant. The combination was also applied in the course of the ten-day experiment and not only on a specific day. Consequently, as the day progresses, the combination will perform more Exploitation than SVM-Margin. This means that on the *i*th day there is more Exploitation than in the $(i-1)$th day.

We defined and tracked several configurations over the course of several days. We found that in the relation between SVM-Margin and Exploitation a balanced division performs better than other divisions (i.e., for 50% of the days, the method will conduct more SVM-Margin with Exploitation being implemented for the remaining time). In short, this method tries to take the best from both of the previous methods.

4.3. A Framework for improving detection capabilities

Fig. 3 illustrates the framework and the process of detecting and acquiring new malicious files through preserving the updatability of the anti-virus and detection model. In order to receive maximal contribution from the suggested framework, it should be deployed in strategic nodes over the Internet network in an attempt to expand its exposure to as many new files as possible. This wide deployment will result in a scenario in which almost every new file will go through the framework. If the file is informative enough or is grasped as probably malicious, then it will be acquired for manual analysis. Examples of strategic nodes can be ISPs and gateways of large organizations. As Fig. 3 shows, the packets of files transported over the Internet through our framework are constructed into files {1}. These files are transformed into vector form {2}. This means that *n*-grams are extracted from the binary code of the files; frequencies are being calculated; and the files are now represented as vectors of *n*-gram frequencies (as explained above). Then, the known files are filtered by the "known files module" that actually filters all the known benign and malicious files {3} (according to a white list, reputation systems (Jnanamurthy, Warty, & Singh, 2013) and signature repository).

The remaining files, which are unknown, are then introduced to the detection model based on SVM and AL. The detection model scrutinizes the files and provides two values for each file: a classification decision using Eq. (4) and a distance calculation from the separating hyperplane using Eq. (15) {4}. A file that the AL method recognizes as informative and which it has indicated should be acquired, is sent to an expert who manually analyzes and labels it {5}. By acquiring these informative files, we aim to frequently update the anti-virus software by focusing the expert's efforts on labeling files that are most likely to be malware or on benign files that are expected to improve the detection model. Note that informative files are those that when added to the training set improve the detection model's predictive capabilities and enrich the anti-virus signature repository. Accordingly, in our context, there are two types of files that may be more informative. The first type includes files for which the classifier is not confident as to their classification (the probability of that they are malicious is very close to the probability that they may be benign). Acquiring them as labeled examples will probably improve the model's detection capabilities. In practical terms, these files will have new *n*-grams or special combinations of existing *n*-grams that represent their execution code (inside the binary code of the executable). Therefore these files will probably lie inside the SVM Margin and consequently will be acquired by the SVM-Margin strategy that selects informative files both malicious and benign that are a small distance from the separating hyperplane.

The second type of informative files includes those that lie deep inside the malicious side of the SVM Margin and that are maximal distance from the separating hyperplane according to Eq. (15). These files will be acquired by the Exploitation strategy and are also a large distance from the labeled files. This distance is measured by the KFF calculation that was explained in the Exploitation AL method section. These informative files are then added to the training set {6} for updating and retraining the detection model {8}. The files that were labeled as malicious are also added to the anti-virus signature repository for enriching and preserving its updatability {7}. This updating of the signature repository also requires a distribution of an update for clients implementing the anti-virus application.

The framework integrates two main phases training and detection/updating.

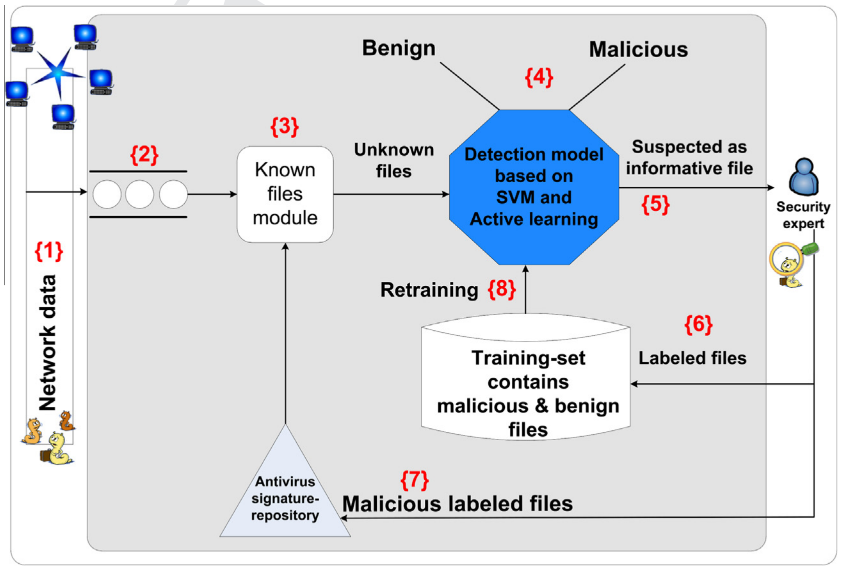
Benign
Malicious
{4}
{3}
{2}
Known files module
Unknown files
Detection model based on SVM and Active learning
Suspected as informative file
{5}
Security expert
{1}
Network data
Retraining {8}
{6}
Training-set contains malicious & benign files
Labeled files
{7}
Antivirus signature-repository
Malicious labeled files

Fig. 3. The process of preserving the updatability of the anti-virus tool using AL methods.

4.3.1. Training

A detection model is trained over an initial training set that includes 10% malicious files (MFP). After the model is tested over a stream that only consists of unknown files that were presented to it on the first day, the initial accuracy of the detection model is evaluated.

4.3.2. Detection and updating

For every unknown file in the stream, the detection model provides a classification while the AL method provides a rank representing how informative the file is. Thus the framework will consider acquiring them. After being selected and receiving their true labels from the expert, the informative files are acquired by the training set and the signature repository is updated as well, just in case the files are malicious. The detection model is retrained over the updated and extended training set which now also includes the acquired examples that are regarded as being very informative. At the end of the day, the updated model receives a new stream of unknown files on which the updated model is once again tested and from which the updated model again acquires informative files. Note that the motive is to acquire as many malicious files as possible since such information will maximally update the anti-virus tool.

5. Measurements and evaluation

The objective of the first set of experiments was to determine the optimal settings for: the feature representation (TF or TFIDF); the *n*-gram representation (3, 4, 5 or 6); the best global range (top 5500 or top 1000–6500); feature selection method (DF, FS or GR); and the top number of selected features (50, 100, 200, 300, 1000, 1500 or 2000). After determining the optimal settings, we performed a second set of experiments to evaluate our proposed acquisition process using the active learning methods.

5.1. Evaluation measures

To evaluate the capability of the framework and the AL methods for efficiently acquiring new files and update the detection model, we relied upon a set of standard, widely used measures. We measured the true positive rate (TPR) which is the percentage of correctly classified positive instances as shown in Eq. (10). The false positive rate (FPR), which is the percentage of misclassified negative instances, is also shown in Eq. (10). The total accuracy, which measures the number of absolutely correctly classified instances, either positive or negative, divided by the entire number of instances, is shown in Eq. (11). Q5

$$TPR = \frac{|TP|}{|TP| + |FN|} \qquad FPR = \frac{|FP|}{|FP| + |TN|} \tag{16}$$

$$\text{Total Accuracy} = \frac{|TP| + |TN|}{|TP| + |FP| + |TN| + |FN|} \tag{17}$$

In addition to the abovementioned measures, we present the core measure of this study which is the number of new malwares acquired each day, that is to say, the malwares that were discovered and acquired daily and added into the training set and signature repository of the anti-virus software.

5.2. Experiment design

5.2.1. Experiment 1: determining the best configuration of the dataset and kernel

To determine the best settings of the file representation and the feature selection, we used the results and insights from previous work that we conducted on the same dataset (Moskovitch, Stopel, et al., 2009). In that study, we performed a comprehensive set of evaluation runs including all combinations of the optional settings for each of the aspects listed above. The number of runs amounted to 1536 in a 5-fold cross-validation format for all three kernels. It should be noted that the files in the test set were not in the training set that presented unknown files to the classifier. Elaboration and analysis of the results of this experiment can be found in Moskovitch, Stopel, et al. (2009). Here, however, we will briefly illustrate the best configuration and detection accuracy rate. The best configuration included the dataset represented by: 5-grams; global top 5500; TF representation; Fischer score feature selection method; and Top300 which is the number of features considered. The RBF kernel out-performed the others with a 93.9% detection accuracy and a low false positive rate of less than 0.03%.

5.2.2. Experiment 2: acquisition of new malwares through active learning

The objective in this main experiment was to evaluate and compare the performance of our new AL methods to the existing selection method, SVM-Margin in regard to two tasks:

- Acquiring as many new unknown malicious files as possible in order to efficiently enrich on a daily basis the signature repository of the anti-virus.
- Updating the predictive capabilities of the detection model that serves as the knowledge store of the AL methods in efficiently identifying the most informative new malwares.

As assumed in a previous work (Moskovitch, Stopel, et al., 2009), malwares on the Internet amount to approximately 10% of the traffic (which actually represents the test set). In this previous work (Moskovitch, Stopel, et al., 2009), it was found that the percentage of malicious files in the training set that leads to the highest detection accuracy is also 10%. In order to adhere as closely as possible to real-life conditions in our experiment, we used the guidelines proposed by Rossow et al. (2012). Consequently, we used 25,260 executables (22,734 benign, 2526 malicious), from a total of 30,423 executables of which 10% were malicious and 90% benign. One should note that in reality the detection model also encounters known and unknown files, but since there is no need to scrutinize known files, we filtered them out since they would be detected in any case by the signature repository of the anti-virus or the white list of the training set. We conducted this experiment using the insights from experiment 1, the dataset configuration specifics and the RBF kernel of the SVM.

Over a ten-day period, we compared file acquisition based on active-learning methods and random selection based on the performance of the detection model.

We took the 25,260 executables (10% MFP) and created ten randomly selected datasets with each dataset containing ten sub-sets of 2521 files representing each day's stream of new files. The 50 remaining files were used by the initial training set to induce the initial model. Note that each day's stream contained 2521 files with 10% MFP. At first, we induced the initial model by training it over the 50 known files. We then tested it on the first day's stream. Next, from this same stream, the selective sampling method selected the most informative files according to that method's criteria. The informative files were sent to a human expert who labeled them. The files were later acquired by the training set which was enriched with an additional *X* new informative files. Of course, when a file was found to be malicious, it was immediately used for updating the signature repository of the antivirus. An update was also distributed to clients. The process was repeated over the next 9 days. The performance of the detection model was averaged for10 runs over the 10 different datasets that were created. Each selective sampling method was checked separately on 43 different amounts of files that had been acquired. This means that for every acquisition amount, the methods were restricted to acquiring a number of files equal to the acquisition amount that followed, denoted as *X*: 10 files, 20 files and so on until 350 files had been acquired (with gaps of 10 files), 450, 550, 600, 650, 700, 750, 800. We also considered the acquisition of all the files in the daily stream (referred to as the ALL method), which represents an ideal but not a feasible situation for acquiring all the new files.

The experiment's various step are as follows:

1. Inducing the initial detection model from the 50 files in the training set.
2. The detection model is tested on the stream of the first day to measure its initial performance.
3. The stream of the first day is introduced to the selective sampling method, which chooses the *X* most informative files according to its criteria and sends them to the virus expert who manually analyzes them to determine their true labels.
4. These informative files are added to the training set, where the malwares are also used for updating the signature repository of the anti-virus tool.
5. Inducing an updated detection model from the extended training-set for the next day.
6. The second day begins with a test of the updated detection model over the stream of the second day to measure its performance and improvement relative to the previous day.
7. The stream of the second day is introduced to the selective sampling method which chooses the *X* most informative files according to its criteria and sends them to the virus expert.
8. Those informative files are added to the training set, where the malwares are also used for updating the signature repository of the anti-virus.
9. Inducing an updated detection model from the extended training set for the next day.

The process repeats itself until the 10th day.

6. Results

In order to appropriately evaluate the efficiency and effectiveness of our framework, we compared four selective sampling methods: an existing AL method; SVM-Simple-Margin (SVM-Margin); our method (Exploitation); a combination of the two previous methods (Combination) and random-selection (Random). Each method was checked for all the forty-three acquisition amounts where the results, in order to neutralize the variance, were the average of 10 different folds. As previously mentioned, we also took into consideration the acquisition of the whole files in the stream (the ALL method) in order to compare the performance of the methods in relation to an ideal case. Obviously ALL is not a feasible method since anti-virus vendors cannot deal with the daily amount of new files requiring manual analysis and inspection. We depicted the results of the most representative acquisition amounts: batches of 50, 250 and 800 files.

We now present the results of the core measure in this study, the number of new unknown malicious files that were discovered and finally acquired into the training set and signature repository of the anti-virus software. As explained above, each day the framework deals with 2521 new files, a 10% MFP of about 250 new unknown malicious files. Statistically, the more files that are selected daily, the more the malicious files that will be acquired daily. Using AL methods, we tried to improve the number of malicious files acquired by means of existing solutions. More specifically, using our methods (Exploitation, Combination) we also sought to improve the number of files that are acquired by the current AL method (SVM-Margin).

Figs. 4–6 present the number of malicious files obtained by acquiring batches of 50, 250 and 800 files respectively, by each of the four methods during the course of the ten-day experiment. Note that in these three Figs. 4–6, the graph of Combination falls behind Exploitation and it is therefore difficult to observe Combination's behavior. As can be seen in Fig. 4, Exploitation and Combination outperformed the other selection methods and showed an increasing trend from the second day. These methods succeeded in acquiring the maximal number of malwares from the 50 files acquired daily.

On the second day, all the AL methods acquired the fewest number of new malwares, even fewer than random selection. This can be explained by the fact that the initial detection model was trained on an initial set of only *X* labeled files that consisted of only 6 malwares. Thus the detection model was not accurate enough to provide the AL methods with the knowledge needed to select new unknown malwares out of the daily amount of 50 files.

On the 10th day, using Exploitation, 88.5% of the acquired files were malicious (44.1 files out of 50); implementing SVM-Margin, only 59.8% of the acquired files were malicious (29.9 files out of 50). This is a significant improvement of almost 30% in unknown malware acquisition. Note that on the 10th day, using Random, only 11.8% of the acquired files were malicious. This is far less than the malware acquisition rates that Exploitation and Combination achieved. The trend is very clear: each day Exploitation and

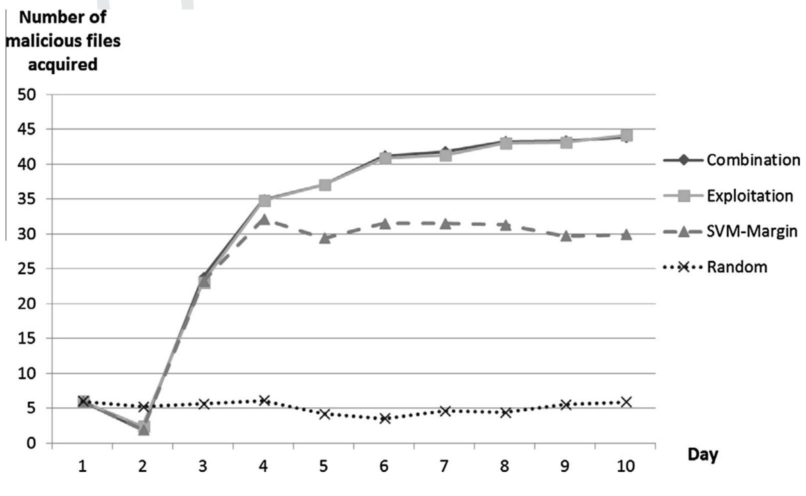

Fig. 4. The number of malicious files acquired by the framework for the different methods through the acquisition of 50 files daily.

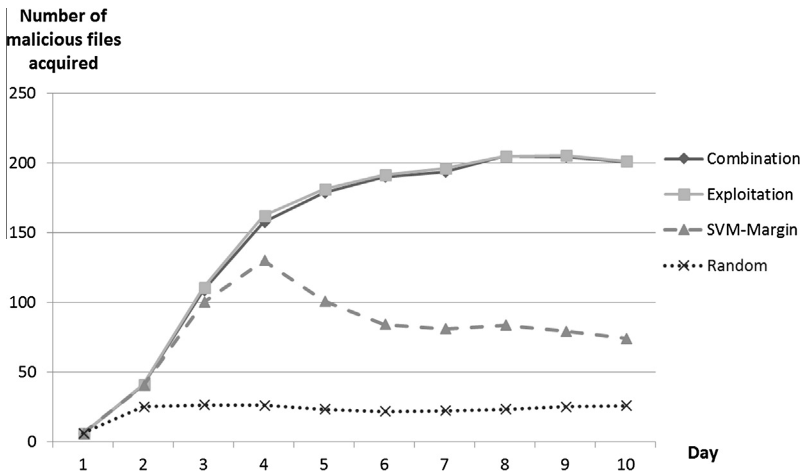

Fig. 5. The amount of malicious files acquired by the framework for the different methods through the acquisition of 250 files daily.

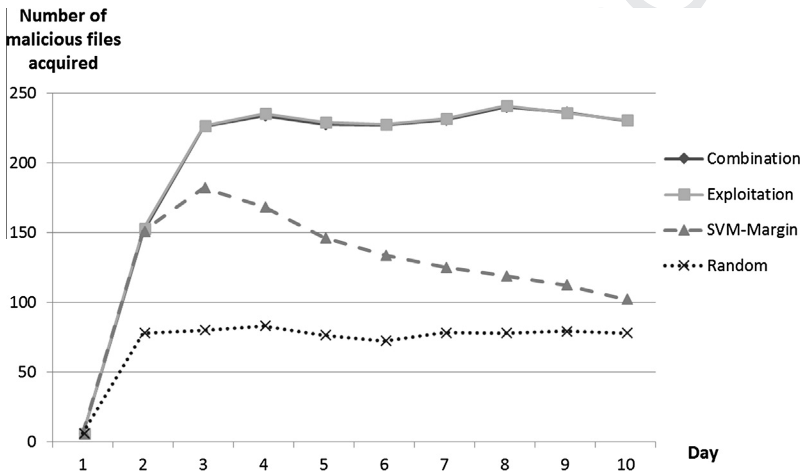

Fig. 6. The amount of malicious files acquired by the framework for the different methods through the acquisition of 800 files daily.

Combination acquired more malicious files than the day before – a feature that supports the daily improvement in the capability for daily updating the detection model, identifying new malwares and for enriching the signatures repository of the anti-virus. As far as we could observe, the random selection trend was constant; there was no improvement in acquisition capabilities over the 10-day period.

As can be seen in Fig. 5, Exploitation outperformed the other selection methods and showed an increasing trend. It succeeded in acquiring the maximal number of malwares from the 250 files acquired daily.

In tracking the performances of the various methods, we observed an interesting phenomenon. Until the 4th day, all the AL methods showed improvement and an increasing number of acquired files. However, after the 4th day, the SVM-Margin AL method showed a decrease in the number of malwares acquired, while our methods, Exploitation and Combination, continued to show an increase and improvement in their acquisition capabilities. This phenomenon can be explained by the way the methods work. The SVM-Margin acquires examples about which the detection model is less confident. Consequently, they are considered to be more informative but not necessarily malicious. As was explained above, SVM-Margin selects new informative files inside the margin of the SVM. Over time and with the improvement of the detection model towards more malicious files, it seems that the malicious files are less informative (due to the fact that malware writers frequently try to use upgraded variants of previous malwares). Since these new malwares might not lie inside the margin, SVM-Margin may actually be acquiring informative benign rather than malicious files. However, our methods, especially Exploitation, are more oriented towards acquiring the most informative and probable malwares by acquiring informative files from the malicious side of the SVM Margin. As a result, an increasing number of new malwares are being acquired. And if the acquired benign file lies deep within the malicious side, it is still a very informative file that can be used for learning purposes and for improving the detection capabilities for the next day.

This observation could not have been made from the results of a previous study (Moskovitch, Nissim, et al., 2009) in which there was only one active learning trial. In our experiment, which sought to represent reality, there were several days of detection and acquisition. Consequently, we see that SVM-Margin is less efficient in acquiring malwares on a continuous basis. The Exploitation method outperformed the SVM-Margin method throughout the ten-day period displaying on the 9th day the largest gap between the two methods in acquiring malwares. At this point, SVM-Margin acquired 79.2 malwares with Exploitation acquiring 205.3 or 2.6 times more malwares than SVM-Margin. We can see that

Exploitation acquired 7.8 times more malwares than Random. The advantage of Exploitation over SVM-Margin and Random is very clear. Random fails to improve over time and it fails to select new and informative malwares. Where SVM-Margin fails in acquiring more malwares daily in the course of the 10 days. These results actually emphasize the efficiency of our framework and methods. Combination performed almost as well as Exploitation, yet was the second best performer.

Table 1 and Fig. 6 present the results of the selection methods for acquiring 800 files daily. The bold red numbers represent the highest quantities acquired by each of the selective sampling methods. Almost the same trend that appeared in dealing with 250 files appears here, with the AL methods performing better than random selection; Exploitation outperforming the other methods; and SVM-Margin AL displaying a decreasing trend.

These results are not significantly better than those achieved with the acquisition of 250 files. Here the number of acquired files daily was 800 which is much more than the 250 malwares in the daily stream (due to the 10% MFP). We expected that acquiring 800 files would probably identify almost all of the 250 malwares in each day's stream. However it seems that the identifying and acquiring all the new malwares is not simple, even when more files are being acquired daily. The cost of acquiring (and manually analyzing) more files daily should be considered in light of the benefits to be obtained by acquiring additional files in an attempt to discover more malicious files.

For example, if we compare the results of Exploitation we can see that on the 10th day, with the acquisition amount at 250 as presented in Fig. 5, Exploitation acquired 201.1 out of 250 malwares (80.44%). On the same day, when the acquisition mount stood at 800, as presented in Table 1 and Fig. 6, Exploitation acquired 230 out of 250 malwares (92.2%). This is equivalent to the cost of acquiring and analyzing 550 more files daily which would have to be achieved by an increased manual effort that would be 3.2 times greater than when the acquisition mount stood at 250 files. Those remaining benign files that were acquired as thought to be malicious will be discussed later.

We have shown here that our AL methods outperformed the SVM-Margin AL method and improved the capabilities for acquiring new malwares. This improvement enriched the signatures repository of the anti-virus software which is the main goal of this study.

We now show that our methods, compared to SVM-Margin, also preserve and even improve the predictive performance of the detection model who plays the knowledge store in the acquisition process. We will show the Accuracy, TPR and FPR levels and their trends in the course of the 10 days.

The same trends were observed when measuring the accuracy rates of the three acquisition amounts (50, 250 and 800) by each of the four selection methods over the ten-day experimental setup. First and foremost, the main and significant observation is that our AL methods performed as well as the baseline existing AL method, SVM-Margin. The accuracy rates of the methods were very similar with negligible difference between them. When 50 files were being acquired daily, SVM-Margin performed a bit better than our methods for several days within the ten-day period. But when dealing with larger acquisition amounts of 250 and 800, our methods (Exploitation and Combination) performed a bit better than SVM-Margin during the whole ten-day period. Secondly as was expected, the acquisition each day of additional files indeed contributes to the accuracy of the detection model because the accuracy usually increase over time; the larger the amount of files acquired each day, the higher the accuracy. Lastly, the three AL methods outperformed random selection in such a way that the gap in the performance between the AL methods and the random selection become larger over the ten-day period.

Since the same trend was observed for the three acquisition amounts, we presented in Table 2, only the results achieved when 800 files were acquired daily. As can be seen, the three AL methods performed almost the same, indicating that different informative files contributed similarly to detection model's accuracy. Exploitation and Combination outperformed all the selection methods during the 10 days with Exploitation outperforming Combination, and specifically presented a bit better accuracy than baseline SVM-Margin we aim to improve. In dealing with large acquisition amounts, every improvement in accuracy is significant since this helps reduce the extent of the manual analysis the experts must carry out.

Table 2 also shows that the difference between our AL methods (97.83%) and random selection (95.85%) amounts to a 2% detection accuracy rate by the 10th day. This rate is especially significant when the detection model is encountering each day dozens of files from which it should detect the newest malicious files. Note that since the dataset is imbalanced and consists of 90% benign files, it is not a hard to achieve 90% accuracy, whereas every other percentage above 90% accuracy is a challenge. Thus, those differences in the accuracy that were achieved are very significant.

While the accuracy measure provides a means for determining the overall efficiency and effectiveness of the framework, the primary task is to detect the files that are most likely to be malicious in order to use them for updating the signature repository. Accordingly, the TPR and FPR measures shed significant light on the four methods that we are examining in this paper.

Fig. 7 represents the TPR levels that were achieved as each of the four methods acquired 50,250,800 files in the final day of the experiment – the tenth day, including also the acquisition of all the 2521 files in the daily stream which is the unfeasible scenario. It can be observed that the three AL methods performed almost the same (again Combination's graph falls behind Exploitation's). SVM-Margin outperformed other selection methods in the 50 and 250 acquisition amounts. While our AL methods (Exploitation and Combination) outperforming in the acquisition amount of 800 files

Table 1
The quantities of malicious files that the framework has acquired for its different methods through the acquisition of 800 files daily.

Day	Random	SVM-Margin	Exploitation	Combination
1	6	6	6	6
2	78	150.7	152.6	150.7
3	79.9	181.9	226.7	225.7
4	83.1	168.3	235.1	233.9
5	76.4	146.1	229.1	227.6
6	72.1	133.7	227.7	227.1
7	78.2	125.1	231.4	230.5
8	77.8	118.8	240.8	239.8
9	79.3	112.5	235.6	236.1
10	77.9	102	230.7	230.1

Table 2
The accuracy of the framework for different methods through the daily acquisition of 800 files.

Day	Random (%)	SVM-Margin (%)	Exploitation (%)	Combination (%)
1	90.05	90.05	90.05	90.05
2	91.97	92.98	93.05	92.98
3	93.43	94.89	94.93	94.97
4	94.30	95.96	96.03	96.06
5	94.94	96.50	96.55	96.58
6	95.21	96.78	96.89	96.87
7	95.51	97.16	97.31	97.28
8	95.56	97.28	97.42	97.40
9	96.04	97.58	97.69	97.69
10	95.85	97.68	97.83	97.83

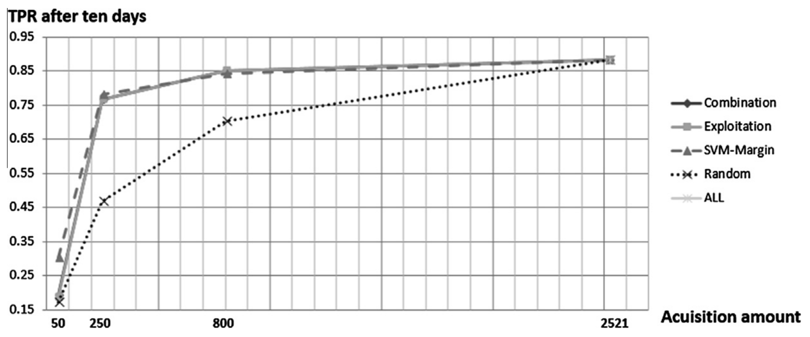

Fig. 7. The TPR of the framework on the tenth day for different methods through the acquisition of 50, 250, 800 and 2521 (ALL) files daily.

daily. In addition, the performance of the detection model improves as more files are acquired daily, so that in the acquisition amount of 800 files daily the results indicate that by acquiring a small and well selected set of informative files only (31% of the stream), the detection model can achieve TPR levels (85.12%) that are quite close to those achieved by acquiring the whole stream (88.14%) – represented by the single point of 2521 files. As can be observed, the trend indicates that the difference between the AL methods and ALL becomes smaller, a trend that supports the efficiency of the framework and the AL method. This approach indicates viability in terms of time and money since it dramatically reduces the amount of files sent to virus experts. There is no doubt that these achievements in acquiring minimal acquisition amounts have implications in terms of time and money for the efficiency of the framework in preserving the updatability of the detection model and ultimately of the anti-virus tool. These factors indicate the benefits to be obtained from performing this process on a daily basis. Note that when the acquisition amount was 50 files, the TPR levels were quite low because the detection model is induced from a small number of files: 50 files on the first day a (only 5–6 malicious files and 44–45 benign files) resulting in only 550 files by the tenth day.

We can see that the difference between the AL methods and random selection becomes greater than 30% in the acquisition amount of 250 files. This difference becomes smaller (15%) yet remains significant when the daily acquisition amount is 800 files. Acquiring files depends upon sending them to human experts for final labeling. This is carried out manually and requires time and money. High TPR rates, achieved by acquiring a small set of files, indicate a capability for preserving the anti-virus updatability and achieving a significant savings of time and money.

Fig. 8 presents the FPR levels of the four acquisition methods for a batch of 800 files. As can be observed, the FPR rates were low and quite similar among the AL methods. A similar decreasing FPR trend began to emerge on the 4th day. This decrease indicates an improvement in the detection capabilities and the contribution of the AL methods contrary to the increase in FPR rates for Random. Random had the lowest FPR until the 4th day – it can be explained by a random selection of informative files that actually improved the initial detection model that was not very accurate in the beginning of the process due to relatively small initial training set. However in the long run, from the 5th day and on, Random had the highest FPR levels what indicates on selection of not very informative files that should have update of the detection model through the days.

Most of the time, from the 5th day and on, Exploitation and Combination achieved the lowest FPR rates, a bit lower than SVM-Margin. This indicates that our methods (Exploitation and Combination) performed as well as SVM-Margin method in regard to predictive capabilities (Accuracy, TPR and FPR) but better than SVM-Margin in acquiring larger number of new malwares daily and in enriching the signature repository of the anti-virus. The FPR is presented for the ten-day period due to the setup of the evaluation. Thus, on each day of the acquisition iteration, we evaluated the learnt classifier. Since for each day, a set of new unknown applications (malwares and benign) are presented to the classifier, the FPR is not constantly decreasing as would have happened if the classifier was tested on the same files daily. Again we can see a general trend in which the difference between the AL methods and Random becomes larger in the course of the ten-day period. This trend supports the efficiency of the AL framework.

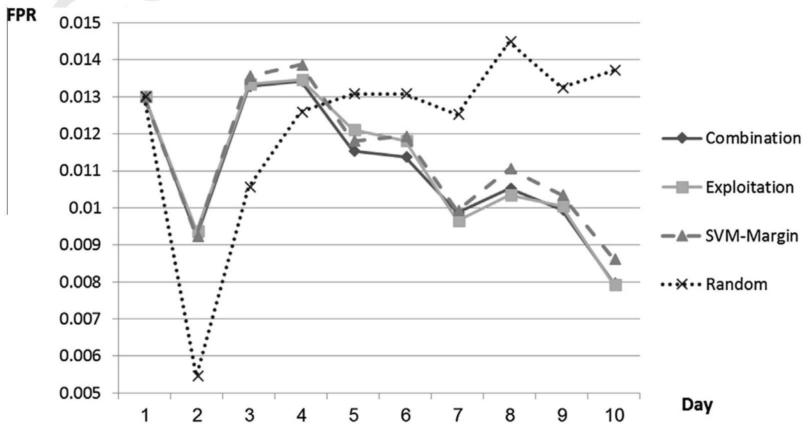

Fig. 8. The FPR of the trends of the framework for different methods based on acquiring 800 files daily.

7. Coping with possible attacks

Zhao, Long, Yin, Cai, and Xia (2012) recently discussed two possible attack scenarios on an AL methods. In these attacks, referred to as adding and deleting, the attacker can actually pollute the unlabeled data before it is sampled by the active learner module. The results of their experiments on an intrusion detection dataset showed that these attacks disrupt the performance of the AL process and significantly decreased detection accuracy: a decrease of 16–30% due to the adding attack and a 15–34% decrease due to deleting. In our context, an adaptive attacker might, "guide" the AL process and poison the classifiers by producing many malwares that contain specifically chosen *n*-grams by design. Consequently, the AL would acquire these files since they would contain new and interesting *n*-grams which did not exist before. Attacking such a biased classifier then becomes easy. The attacker simply leaves out the chosen *n*-grams and creates a malicious file that can evade the detection model.

The way to meet this attack is quite simple. First the AL process is not based on a specific node in the Internet, but is sustained by many sources of information and files. Thus such an attack must flood significant parts of the Internet network in order to poison the presented framework in a way that will bias the classifier. Not only is such a flood by an attacker not feasible but it is also time-consuming and therefore anti-virus vendors have enough time to distribute a patch against it. Secondly, since our framework tries to select the most informative files and attempts to enlarge the signature repository, it is not choosing files that are similar to previous ones that were acquired before. Our AL methods would not acquire a full set of malicious files that are similar in specific *n*-grams but only a few representative ones. Thus the framework is resilient to such attacks and its detection capabilities remain unaffected.

Whenever one uses machine learning methods based on static analysis (especially *n*-grams) for detecting unknown malicious code files, a question is raised about the capability of the suggested framework to detect obfuscated (Moser, Kruegel, & Kirda, 2007b) (including encrypted, compressed and packed), malicious files. In PC executables (contrary to Android mobile applications), most of the files, benign and malicious, are not obfuscated in the first instance. When they are obfuscated and encrypted, it may be due to an attempt to evade security mechanisms such as anti-virus packages (Carey Nachenberg, 1997) that analyze the static data of files. Thus, obfuscated files are more likely to be malicious rather than benign and therefore become more suspicious. Such files are automatically sent to the lab for deeper scrutiny. Additionally, as was already Q6 reported by Zhao et al. (2012), obfuscation is usually performed by automated tools that generate distinctive properties inside the binary code of the obfuscated file, properties that distinguishes them from unobfuscated files. The FalckonEncrypter is an example of an obfuscation library that was developed by hackers and there is no reason for genuine software companies to use an untrustworthy library developed by hackers. Consequently, the obfuscation actually helps in detecting malicious files. Moreover, as was presented in Newsome and Song (0000) andNewsome, Karp, and Song (2005), one of the desirable properties of a malicious file is its self-propagation capability. Since the malicious file is likely to have self-decompression or self-decryption commands inside the code of the malicious file that is being represented by fixed binary sequences, these commands can be used for detection. Accordingly, these sequences are taken into account in the learning process and assist in discriminating between malicious and benign files.

Two other independent studies have shown that ML methods based on static analysis of files can detect obfuscated malicious files even better than detecting unobfuscated malicious files. Kolter & Maloof (2006) used *n*-grams features, as we did in our study, and reported that the detection accuracy on obfuscated malicious files was higher when using the *n*-grams features rather than when using payload functions as a feature extraction methodology. A recent study by Zhao et al. (2012), reported on the same trend, that the TPR values among obfuscated files are higher than those among unobfuscated files for the same abovementioned reasons.

It should be noted that packed files also contain a portion of code that is responsible for the unpacking operation, where those portions of code can be used for identifying different and informative packed files that might contain malicious code, especially in case those files were not packed by a popular packer. For instance Polyunpack (Royal et al., 2006), mentioned above, also conducted a static analysis phase of packed files in which the sequences of packed or hidden code in a malware instance can be made self-identifying.

There are also different malware families that use popular packers such as UPX/PKlite in which a portion of the unpacking operation will be similar to those benign files that have been packed with the same packer. However, the other part which is packed will contain patterns that differ from the benign ones. These patterns can be discovered with high probability when a suitable feature extraction methodology is used.

In relation to handling obfuscation files, one may ask how the framework will react if the reality is altered and many benign files are obfuscated? Likewise, what if an adaptive attacker obfuscates multiple benign programs and floods the network with them? Our answer is that this attack will probably ruin the discrimination in the binary sequences which currently exists between obfuscated malicious and benign files. This is a subject for further exploration and we have recently begun working on this idea. Our work is based on building two complementary detection models. One will be trained on an obfuscated dataset and will discriminate between obfuscated files (benign and malicious); the other will be trained on a non-obfuscated dataset and will discriminate between non-obfuscated files (benign and malicious).

8. Discussion and conclusion

The main goal of this paper was to present a framework for efficiently updating anti-virus tools with new unknown malwares. With an updated classifier, we can detect new malwares that can be utilized for sustaining an anti-virus tool. Both the anti-virus and the detection model (classifier) must be updated with new and labeled files. This labeling is done manually by experts, thus the goal of the classification is to focus expert efforts on labeling files that are more likely to be malware or on files that might add new information about benign files.

In this paper we proposed a framework based on new active learning methods (Exploitation and Combination) designed for acquiring unknown malware. The framework seeks to acquire the most important files, benign and malwares, in order to improve classifier performance, enabling it to frequently discover and enrich the signature repository of anti-virus tools with new unknown malwares.

Adopting ideas from text categorization, we presented a static analysis methodology for representing malicious and benign executables in detecting unknown malicious codes. Two experiments were conducted. In the first and most basic experiment, we tried to find the optimal configuration of dataset and SVM kernel that would yield the best capability for detecting unknown malwares. In the second and most important experiment, we evaluated the proposed framework, comparing the performance of our two AL methods to SVM-Margin (an existing AL method) and random selection in acquiring new malicious files and updating both the signatures repository of the anti-virus and the detection model.

In general, three of the AL methods performed very well, with our methods, Combination and Exploitation, outperforming SVM-Margin. This fact is one of the contributions of our study: the development of new AL methods that are more suitable than current ones in acquiring unknown malwares. The evaluation of the classifier before and after the daily acquisition showed an improvement in the detection rate and subsequently more new malwares were acquired – a fact that actually justified the acquisition process done by our framework.

In the 10th day of the 50 file acquisition amount, Exploitation acquired 44 new malwares, which is almost 30% more malwares than the number acquired by SVM-Margin (~30 malwares) and 77% more malwares than those acquired by the random selection method (~6 malwares). On the 10th day of acquisition, with the 250 batch, Exploitation acquired 201 malwares, which is almost eight times better than the amount acquired by random sampling (~26 malwares), the base line selection method. It also acquired 2.6 times more malwares than the existing AL method, SVM-Margin (74 malwares). In both the 50 and 250 acquisition amounts, the trend in the course of the ten-day period was very clear. Each day Exploitation acquired more malicious files than the day before. This is an important feature that enables the detection model to update itself daily and to identify new malwares for enriching the signature repository of the anti-virus tool. In both the 250 and 800 acquisition amounts, we observed an interesting phenomenon. In the first few days of all the AL methods, there was an increase in the number of acquired files. However for the rest of the time, the SVM-Margin AL method showed a decrease in the number of malwares acquired, while our AL methods continued to increase and improve acquisition capabilities. Therefore, we conclude that the SVM-Margin method is less efficient in continuously acquiring new malwares and that our methods provide an essential feature for continuously acquiring new files and updating anti-virus tools. The larger acquisition amounts of 250/800 were also checked in order to measure the capability of the framework for efficiently acquiring informative and malicious files in large scale scenarios. Basically this step demonstrated that, the more the anti-virus vendor can invest in labeling, the better our framework will update the detection model and enlarge the signature repository of the anti-virus.

We can explain the better acquisition performance of Exploitation according to the way it actually functions. Exploitation tries to acquire the files that are most likely malicious. In fact, it also acquires benign files that are thought to be malicious. Although these benign files are indeed confusing, they are very informative for the detection model. As a consequence, their acquisition improved the performance of the detection model better than the SVM-Margin method that acquires files which are known to be confusing and thus contribute less to improving the detection model. We understood from this phenomenon that there are noisy benign files lying deep within a side class of the malicious files. As noted they are perhaps confusing but nevertheless very informative and valuable to the detection model (these files will probably become a support vectors). Additionally, they help in acquiring malicious files that eventually update and enrich the signature repository of the anti-virus. It should be noted that these files seem to be more informative than malwares since they embody relevant information that was hitherto hidden since the classifier first regarded them as malwares. (The classifier thought them to be malwares and it finally were discovered as benign). Compared to domains in which noisy data is not designed in the first place, in the malware detection domain there is a significant rate of noisy files that make detection much more complicated (such as malwares that are purposely designed as benign files). This was demonstrated in a recent and comprehensive study on worm detection (Nissim et al., 2012) in 2012. It seems that our method (Exploitation) for acquiring files that mostly seem malicious induces a better detection model that will eventually acquire also confusing but valuable and informative benign files.

In future work, we are interested in implementing this framework also on Android applications where it is not very feasible to apply advanced detection techniques over the device itself due to its resource limitations (CPU, battery, etc.). Therefore these mobile devices are very dependent on the anti-virus solutions that should be frequently and efficiently updated. Very possibly our suggested AL framework could address this problem. An additional research direction lies in developing AL methods for non-executable files such as PDF files. Those malicious PDF files were found to exploit much vulnerability in Adobe-Reader versions. This follows from the recent phenomenon of PDF files being used to perform malicious activities, especially as part of APT attacks against organizations. The AL methods will help in detecting and identifying the newest malicious PDF files that utilize zero-day exploits found on the Adobe-Reader.

9. Uncited references

(Chen et al. (2008), Fawcett et al. (2003), Hansman and Hunt (2003); McAfee study finds 4% of search results malicious (2007), Roy and McCallum (2001) and Shin et al. (2006). Q7

Acknowledgements

This research was partly supported by the National Cyber Bureau of the Israeli Ministry of Science, Technology and space.

References

Abou-Assaleh, T., Cercone, N., Keselj, V., & Sweidan, R. (2004). N-gram-based detection of new malicious code. In *Computer software and applications conference, 2004. COMPSAC 2004. Proceedings of the 28th annual, international*, September 28–30 (Vol. 2, pp. 41–42).
Angluin, D. (1988). Queries and concept learning. *Machine Learning, 2*(4), 319–342.
Baram, Y., El-Yaniv, R., & Luz, K. (2004). Online choice of active learning algorithms. *The Journal of Machine Learning Research, 5*, 255–291.
Bayer, U., Comparetti, P. M., Hlauschek, C., Kruegel, C., & Kirda, E. (2009). Scalable, behavior-based malware clustering. In *NDSS* (Vol. 9, pp. 8–11).
Bi, J., Bennett, K., Embrechts, M., Breneman, C., & Song, M. (2003). Dimensionality reduction via sparse support vector machines. *The Journal of Machine Learning Research, 3*, 1229–1243.
Burges, C. J. (1998). A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery, 2*(2), 121–167.
CERT (1999). Trojan horse version of tcp wrappers. <http://www.cert.org/advisories/CA-1999-01.html>.
Chang, C. C., & Lin, C. J. (2011). LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST), 2*(3), 27.
Chen, Z., Roussopoulos, M., Liang, Z., Zhang, Y., Chen, Z., & Delis, A. (2012). Malware characteristics and threats on the internet ecosystem. *Journal of Systems and Software, 85*(7), 1650–1672. July, ISSN 0164-1212..
Chen, Z., Wei, P., & Delis, A. (2008). Catching remote administration Trojans (RATs). *Software: Practice and Experience, 38*(7), 667–703. June.
Fawcett, T. (2003). ROC graphs: Notes and practical considerations for researchers, Technical Report, HPL-2003-4, HP Laboratories.
Golub, T., Slonim, D., Tamaya, P., Huard, C., Gaasenbeek, M., Mesirov, J., et al. (1999). Molecular classification of cancer: Class discovery and class prediction by gene expression monitoring. *Science, 286*, 531–537.
Hansman, S., & Hunt, R. (2003). A taxonomy of network and computer attack methodologies. *Department of computer science and software engineering* (Vol. 7), University of Canterbury, Christchurch, New Zealand.
Henchiri, O., & Japkowicz, N. (2006). A feature selection and evaluation scheme for computer virus detection. In *Sixth international conference on data mining, 2006 ICDM '06* 18–22, December (Vol., pp. 891–895). Q8
Herbrich, R., Graepel, T., & Campbell, C. (2001). Bayes point machines. *The Journal of Machine Learning Research, 1*, 245–279.
Jacob, G., Debar, H., & Filiol, E. (2009). Malware behavioral detection by attribute-automata using abstraction from platform and language. In *Recent advances in intrusion detection* (pp. 81–100). Berlin Heidelberg: Springer.
Jang, J., Brumley, D., & Venkataraman, S. (2011). BitShred: Feature hashing malware for scalable triage and semantic analysis. In *Proceedings of the 18th ACM conference on computer and communications security (CCS '11)* (pp. 309–320). New York, NY, USA: ACM.

Jnanamurthy, H. K., Warty, C., & Singh, S. (2013). Threat analysis and malicious user detection in reputation systems using mean bisector analysis and cosine similarity (MBACS).

Q9 Joachims, T. (1999). Making large scale SVM learning practical.

Kiem, H., Thuy, N. T., & Quang, T. M. N. (2004). A machine learning approach to anti-virus system. In *Joint workshop of Vietnamese society of AI, SIGKBS-JSAI, ICS-IPSJ and IEICE-SIGAI on active mining*, 4–7 December (pp. 61–65), Hanoi-Vietnam.

Kolbitsch, C., Comparetti, P. M., Kruegel, C., Kirda, E., Zhou, X. Y., & Wang, X. (2009). Effective and efficient malware detection at the end host. In *USENIX security symposium* (pp. 351–366).

Kolter, J. Z., & Maloof, M. A. (2004). August. Learning to detect malicious executables in the wild. In *Proceedings of the tenth ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 470–478). ACM.

Kolter, J. Z., & Maloof, M. A. (2006). Learning to detect and classify malicious executables in the wild. *The Journal of Machine Learning Research, 7*, 2721–2744.

Lanzi, A., Sharif, M. I., & Lee, W. (2009). K-tracer: A system for extracting kernel malware behavior. In *NDSS*.

Lewis, D. D., & Gale, W. A. (1994). A sequential algorithm for training text classifiers. In *Proceedings of the 17th annual international ACM SIGIR conference on research and development in information retrieval* (pp. 3–12). Springer-Verlag New York Inc..

Masud, M. M., Khan, L., & Thuraisingham, B. (2007). Feature based techniques for auto-detection of novel email worms. In *Advances in knowledge discovery and data mining* (pp. 205–216). Berlin Heidelberg: Springer.

McAfee study finds 4% of Search results malicious (2007). By Frederick lane, 4th June [<http://www.newsfactor.com/story.xhtml?story_id=010000CEUEQO>].

Menahem, E., Shabtai, A., Rokach, L., & Elovici, Y. (2009). Improving malware detection by applying multi-inducer ensemble. *Computational Statistics and Data Analysis, 53*(4), 1483–1494.

Mitchell, T. M. (1997). *Machine learning*. Burr Ridge, IL: McGraw Hill. 45.

Mokube, I., & Adams, M. (2007). Honeypots: Concepts, approaches, and challenges. In *Proceedings of the 45th annual southeast regional conference (ACM-SE 45)* (pp. 321–326). New York, NY, USA: ACM.

Moore, D., Paxson, V., Savage, S., Shannon, C., Staniford, S., & Weaver, N. (2003). Inside the slammer worm. *IEEE Security and Privacy*.

Moore, D., Shannon, C., & Claffy, K. (2002). Code-Red: A case study on the spread and victims of an internet worm. In *Proceedings of the 2nd ACM SIGCOMM workshop on internet measurement (IMW '02)* (pp. 273–284). New York, NY, USA: ACM.

Moser, A., Kruegel, C., & Kirda, E., 2007a. Exploring multiple execution paths for malware analysis. In *IEEE symposium on security and privacy, SP '07*, 20–23 May (Vol., pp. 231–245).

Moser, A., Kruegel, C., & Kirda, E., 2007b. Limits of static analysis for malware detection. In *Twenty-third annual computer security applications conference, ACSAC 2007*, 10–14 December (Vol., pp. 421–430).

Moskovitch, R., Elovici, Y., & Rokach, L. (2008). Detection of unknown computer worms based on behavioral classification of the host. *Computational Statistics and Data Analysis, 52*(9), 4544–4566.

Moskovitch, R., Gus, I., Pluderman, S., Stopel, D., Glezer, C., Shahar Y., & Elovici, Y. (2007). Detection of unknown computer worms activity based on computer behavior using data mining. In *IEEE symposium on computational intelligence in security and defense applications, CISDA 2007*, 1–5 April (Vol., pp. 169–177). http://dx.doi.org/10.1109/CISDA.2007.368150.

Moskovitch, R., Stopel, D., Feher, C., Nissim, N., & Elovici, Y. (2008). Unknown malcode detection via text categorization and the imbalance proble. In *IEEE international conference on intelligence and security informatics, 2008 ISI 2008*, 17–20 June (Vol., pp. 156–161).

Moskovitch, R., Nissim, N., Englert, R., & Elovici, Y. (2008). Detection of unknown computer worms activity using active learning, In *The 11th international conference on information fusion*, Cologne, Germany, June 30–July 03.

Moskovitch, R., Feher, C., Tzachar, N., Berger, E., Gitelman, M., Dolev, S., et al. (2008). Unknown malcode detection using OPCODE representation. In *Intelligence and security informatics* (pp. 204–215). Berlin Heidelberg: Springer.

Moskovitch, R., Nissim, N., & Elovici, Y. (2009). Malicious code detection using active learning. In *Privacy, security, and trust in KDD* (pp. 74–91). Berlin Heidelberg: Springer.

Moskovitch, R., Stopel, D., Feher, C., Nissim, N., Japkowicz, N., & Elovici, Y. (2009). Unknown malcode detection and the imbalance problem. *Journal in Computer Virology, 5*(4), 295–308.

Nachenberg, C. (1997). Computer virus-antivirus coevolution. *Communications of the ACM, 40*(1), 46–51.

Nataraj, L., Yegneswaran, V., Porras, P., & Zhang, J. (2011). A comparative assessment of malware classification using binary texture analysis and dynamic analysis. In *Proceedings of the 4th ACM workshop on security and artificial intelligence* (pp. 21–30). ACM.

Nataraj, L., Karthikeyan, S., Jacob, G., & Manjunath, B. S. (2011). Malware images: Visualization and automatic classification. In *Proceedings of the 8th international symposium on visualization for cyber security* (p. 4). ACM.

Newsome, & Song, D. Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software. In *Proceedings of the network and distributed system security symposium (NDSS)*. Q10

Newsome, J., Karp, B., & Song, D. (2005). Polygraph: Automatically generating signatures for polymorphic worms. In *2005 IEEE symposium on security and privacy*, 8–11 May (Vol., pp. 226–241).

Nissim, N., Moskovitch, R., Rokach, L., & Elovici, Y. (2012). Detecting unknown computer worm activity via support vector machines and active learning. *Pattern Analysis and Applications, 15*(4), 459–475.

Perdisci, R., Lanzi, A., & Lee, W. (2008). McBoost: Boosting scalability in malware collection and analysis using statistical classification of executables. In *Annual computer security applications conference, 2008. ACSAC 2008*, 8–12 December (Vol., pp. 301–310).

Provos, N., & Holz, T. (2008). *Virtual honeypots: From botnet tracking to intrusion detection*. Addison-Wesley, pp. 231–272.

Rieck, K., Holz, T., Willems, C., Düssel, P., & Laskov, P. (2008). Learning and classification of malware behavior. In *Detection of intrusions and malware, and vulnerability assessment* (pp. 108–125). Berlin Heidelberg: Springer.

Rieck, K., Trinius, P., Willems, C., & Holz, T. (2011). Automatic analysis of malware behavior using machine learning. *Journal of Computer Security, 19*(4), 639–668.

Rossow, C., Dietrich, C. J., Grier, C., Kreibich, C., Paxson, V., Pohlmann, N., et al. (2012). Prudent practices for designing malware experiments: Status quo and outlook. In *IEEE symposium on security and privacy (SP)*, 20–23 May (Vol., pp. 65–79).

Roy, N., & McCallum, A. (2001). Toward optimal active learning through Monte Carlo estimation of error reduction. ICML, Williamstown.

Royal, P., Halpin, M., Dagon, D., Edmonds, R., & Lee, W. (2006). PolyUnpack: Automating the hidden-code extraction of unpack-executing malware, In *22nd Annual computer security applications conference, 2006. ACSAC '06*, December (Vol., pp. 289–300).

Salton, G., Wong, A., & Yang, C. S. (1975). A vector space model for automatic indexing. *Communications of the ACM, 18*(11), 613–620.

Schultz, M.G., Eskin, E., Zadok, E., Stolfo, & S.J. (2001). Data mining methods for detection of new malicious executables. In *Proceedings 2001 IEEE symposium on security and privacy S&P 2001* (Vol., pp. 38–49).

Shabtai, A., Moskovitch, R., Elovici, Y., & Glezer, C. (2009). Detection of malicious code by applying machine learning classifiers on static features: A state-of-the-art survey, information security technical report 14 (pp. 16–29).

Sharif, M., Lanzi, A., Giffin, J., & Lee, W. (2009). Automatic reverse engineering of malware emulators. In *30th IEEE Symposium on Security and Privacy, 2009*, 17–20 May (Vol., pp. 94–109).

Shin, S., Jung, J., & Balakrishnan, H. (2006). Malware prevalence in the KaZaA file-sharing network. In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement* (pp. 333–338). ACM.

Song, D., Brumley, D., Yin, H., Caballero, J., Jager, I., Kang, M. G., et al. (2008). BitBlaze: A new approach to computer security via binary analysis. In *Information systems security* (pp. 1–25). Berlin Heidelberg: Springer.

Stopel, D., Boger, Z., Moskovitch, R., Shahar, Y., & Elovici, Y. (2006). Improving worm detection with artificial neural networks through feature selection and temporal analysis techniques. In *Proceedings of third international conference on neural networks*, Barcelona.

Tahan, G., Rokach, L., & Shahar, Y. (2012). Mal-id: Automatic malware detection using common segment analysis and meta-features. *The Journal of Machine Learning Research, 13*(1), 949–979.

Tong, S., & Koller, D. (2000–2001). Support vector machine active learning with applications to text classification. *Journal of Machine Learning Research, 2*, 45–66.

Vapnik, V. N. (1982). *Estimation of dependences based on empirical data* (Vol. 41). Berlin: Springer.

Vapnik, V. (1998). *Statistical learning theory*. New York: Springer.

Wang, X., Yu, W., Champion, A., Fu, X., & Xuan, X.D. (2007). Worms via mining dynamic program execution. In *Third international conference security and privacy in communication networks and the workshops, SecureComm* (pp. 412–421).

Willems, Carsten, Holz, Thorsten, & Freiling, Felix (2007). Toward automated dynamic malware analysis using CWSandbox. *IEEE Security and Privacy, 5*, 32–39. 2 (March 2007).

Yu, Z. H. U., Xin-cai, W., & Hai-bin, S. (2008). Detection method of computer worms based on SVM. *Mechanical and Electrical Engineering Magazine, 8*, 2.

Zhao, W., Long, J., Yin, J., Cai, Z., & Xia, G. (2012). Sampling attack against active learning in adversarial environment. In *Modeling decisions for artificial intelligence* (pp. 222–233). Berlin Heidelberg: Springer.