International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Index Copernicus Value (2013): 6.14 | Impact Factor (2014): 5.611

Study of Dataset Feature Filtering of OpCode for Malware Detection Using SVM Training Phase

Bhushan Kinholkar

PG Student, Department of Computer Science Engineering, SSBT"s College of Engineering and Technology, Jalgaon, India.

Abstract: ***Malware can be defined as any type of malicious code that has the potential to harm a computer or network. To detect unknown malware families, the frequency of the appearance of Opcode (Operation Code) sequences are used through dynamic analysis. Opcode n-gram analysis used to extract features from the inspected files. Opcode n-grams are used as features during the classification process with the aim of identifying unknown malicious code. A support vector machine (SVM) is used to create a reference model, which is used to evaluate two methods of feature reduction, which are "area of intersect" and "subspace analysis using eigenvectors." The SVM is configured to traverse through the dataset searching for Opcodes that have a positive impact on the classification of benign and malicious software. The dataset is constructed by representing each executable file as a set of Opcode density histograms. Classification tasks involve separating dataset into training and test data. The training sets are classified into benign and malicious software. In area of interest the characteristics of benign and malicious Opcodes are plotted as normal distributions. They are grouped into density curves of a single Opcode. The key feature to note is the overlapping area of the two density curves. In Subspace analysis the importance of individual OpCodes, are investigated by the eigenvalues and eigenvectors in subspace.PCA is used for data compression and mapping. The eigenvector filter Opcodes coincides with the SVM classify the malware Opcodes feature.***

Keywords: SVM, N-gram analysis, obfuscation, area of intersect.

1. Introduction

The recent growth in high-speed Internet connections enable malware to propagate and infect hosts very quickly, therefore it is essential to detect and eliminate new (unknown) malware. OpCode sequence is used to detect the malware in runtime environment.

N-gram analysis in feature extraction increases the computational overhead. The computation processing overhead is reduced by the filtering the less or irrelevant feature. Two types of filtering techniques are used. Area of interest is used to investigate the feature of the dataset by obtaining the overlapping area of the density curves between malicious and benign software. In subspace analysis the feature extraction for dataset is based on the eigen values and eigenvectors In the subspace. PCA technique is used to map the data in the subspace, which provides original data.

Signature-based detection is based on investigating suspicious code and gathering information in order to characterize any malicious intent of the malware. The main objective of this approach is to extract specific byte sequences of code as signatures and to look for a signature in suspicious files. For large datasets, or costly (computation) distance functions, the training process associated with learning machines can become immense. Thus, the feature explosion that occurs with N-grams for large values of N needs to be addressed.

Anti-virus vendors are facing huge quantities (thousands) of suspicious files everyday. These files are collected from various sources including dedicated honeypots, third party providers and files reported by customers either automatically or explicitly. The large amount of files makes efficient and effective inspection of files particularly challenging.

Several analysis techniques for detecting malware, which commonly distinguished between dynamic and static, have been studied. In dynamic analysis (also known as behavioral analysis) the detection of malware consists of information that is collected from the operating system at runtime (i.e., during the execution of the program) such as system calls, network access and files and memory modifications. For large datasets, or costly (computation) distance functions, the training process associated with learning machines can become immense. Thus, the feature explosion that occurs with

N-grams for large values of N needs to be addressed.

This paper investigates approaches to filtering out irrelevant features and in Section 1, with a discussion on related research. In Section 2, related work is discussed. Overall system overview is discussed in Section 3. Section 4 how dataset is crate is discussed. SVM for classification is discussed in Section 5. Section 6 Proposed approach is discussed area of intersect, subspace analysis of eigenvalue and eigenvector. Conclusion is discussed in Section 7.

2. Related Work

Lakhotia *et al.* [1] presented a state machine method to detect obfuscated calls relating to *push* , *pop* and *ret* opcodes that are mapped to stack operations. However, their approach did not model situations where the *push* and *pop* instructions are decomposed into multiple instructions, such as directly manipulating the stack pointer using *mov* commands.

Bilar [2] used static analysis to obtain opcode distributions from PE files that could be used to identify polymorphic and metaphoric malware. Bilar"s findings show that many prevalent opcodes (*mov, push, call*, etc.) did not make good indicators of malware. However, lesser frequent opcodes such *ja, adc, sub,inc* and *add* proved to be better indicators of malware. In other research, Bilar [3] compared the statically generated CFG of benign and malicious code. Their findings showed a difference in the basic block count for benign and malicious code. Bilar concluded that malicious code has a lower basic block count, implying a simpler structure: Less interaction, fewer branches and less functionality.

N-grams are based on a signature approach that relies on small sequences of strings or byte codes that are used to detect malware. Santos *et al.* [4] demonstrated that n-gram signatures could be used to detect unknown malware. The experiment extracted code and text fragments from a large database of program executions to form signatures that are classified using machine learning methods.

Asaf Shabtail, Robert Moskovitch,[5]Classification algorithms are employed for the detection of unknown malicious code. Byte n-gram patterns are used in to represent the inspected files. The inspected files are used as patterns for OpCode n-gram patterns which are extracted from the files after disassembly. The OpCode n-gram patterns are used as features for the classification process. The classification process main goal is to detect unknown malware within a set of suspected files and used in antivirus software as signatures. A problem of this domain is the imbalance problem in which the distribution of the classes varies. For detecting malware, dynamic and static anlysis is used. In dynamic the detection of malware consists of information that is collected from the operating system at runtime. In static, the information is collected from explicit and implicit observations in its binary/source code. Classification algorithms uses the binary code of a file (i.e., byte n-grams), and classifiers are used to learn patterns in the code in order to classify new (unknown) files as malicious or benign. Text categorization technique is used for Malware categorization which is based on OpCode n-gram patterns, generated by disassembling the inspected executable files, to represent the files. OpCode expressions, extracted from the executable file, are expected to provide a more meaningful representation of the code rather than byte sequence. Binary classifiers for the detection of unknown malicious code introduce the imbalance problem. The imbalance problem refers to scenarios in which the proportions of the classes are not equal. Imbalance problem leads to misclassification of datasets.

Xu Chen, John Andersen, Z. Morley Mao[6] Malware is becoming more advanced.A detailed taxonomy of malware defender fingerprinting techniques should be developed.A novel fingerprinting method assistsmalware propagation, and creates an effective new techniqueto protect production systems.Systems should be divided as production systems and monitoring systems.Taxonomy is used to capture essential techniques fordistinguishing between productions and monitoring systems**.** A remote network based reconnaissance is used todifferentiate between VMs and real machines.A new paradigm is used for protecting production systemsmaking them appear to be monitoring systems. Both VMs and debuggers make hardware detectable changes when malware are present. Debuggers communicate with the rest of the system. The execution environment of a process is altered when it's running in a VM or under a debugger.

R. Sekar, Bendre D. , Dhurjati P., Bollineni.[7] Intrusion detection approach identifies anomalous sequences of system calls executed by programs. A natural way for learning sequences is to use a finite-state automaton (FSA). FSA-learning is computationally expensive, and requires much space usage. The algorithm proposed in this project approach builds a compact FSA in a fully automatic and efficient, without requiring access to source code for programs. The space requirements are also reduced. The FSA uses only a constant time per system call during the learning as well as detection period. This leads to low overheads for intrusion detection. More accurate detection is performed. The training periods needed for our FSA based approach are shorter. Moreover, false positives rates are reduced.

Clemens Kolbitsch, Paolo Milani Comparetti, Christopher Kruegel,[8]Host-based detection approaches suffer from *ineffective* detection models. Host-based detection models concentrate on the features of a specific malware instance, and are often easily evadable by obfuscation or polymorphism. In order to address the shortcomings of ineffective models, several dynamic detection approaches have been proposed that aim to identify the behavior exhibited by a malware family. These approaches are unfortunately too slowto be used as real-time detectors on the end host. In this project, a malware program is analyzed in a controlled environment to build a model that characterizes its behavior and also describe the information flows between the system calls essential to the malware‟s mission. The program slices are responsible for each information flows. For detection, these slices are matched against the runtime behavior of an unknown program. The behavior is then automatically translated into detection models that operate at the host level. Rapid detection and elimination of novel Malware is made. The developed fine-grained model is used to monitor and observe the interactions of Malware with operating system. Using dynamic detection is much efficient compared to the conventional static model.

Ulrich Bayer, Imam Habibi, Davide Balzarotti, Engin Kirda, [9]Malicious software (or malware) is one of the major security threats facing the Internet today. To develop effective malware countermeasures and mitigation techniques understanding of malware behavior is important. To detect the behavior of Malware, malicious code samples that were collected by Anubis. Anubis is a dynamic malware analysis platform that executes submitted binaries in a controlled environment. The analysis is performed by the system monitors by invocating the important Windows API calls and system services, it records the network traffic, and it tracks data flows. The reports are generated, while submitting the binaries. Anubis receives Malware samples through a public web interface and a number of feeds from security organizations and anti-malware companies. When compiling statistics about the behaviors of malicious code, certain Malware families make use of polymorphism. To address this problem, analysis of malware behavior are also based on malware families (clusters). The influence of code polymorphism on malware statistics is also addressed. Anubis submitters are categorized as following: large, medium, small, single. The behavioral information with respect to the number of malware families is approximated as clusters of samples that exhibit similar behaviors. Several activities are performed to detect the behavior of Malware. Several Malware activities are detected using following activities. File system activity, Registry activity, Network activity, GUI windows, Botnet activity, and Sandbox detection. Each of the activities detects the common behavior and clusters it to detect the similar group.

3. System Overview

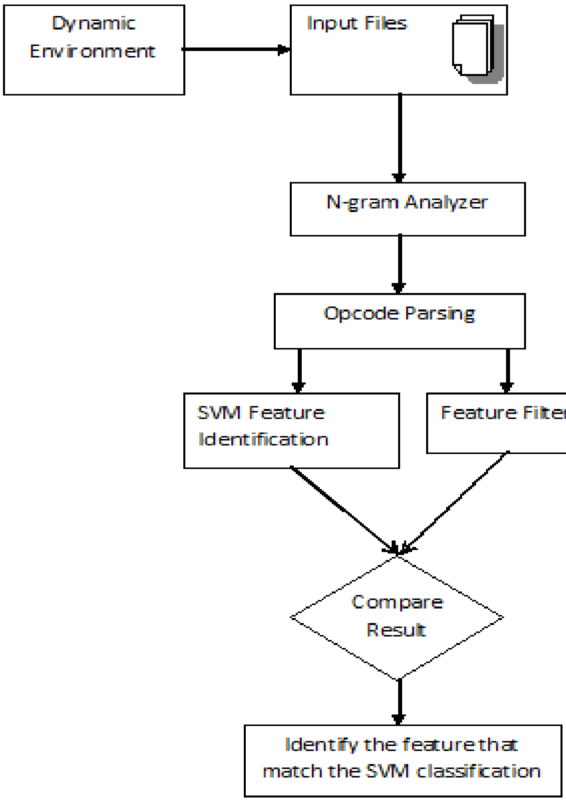

Figure1: System Overview

The motivation for this research is to reduce the computational overhead required when N-gram analysis is performed on low-level fine grain data. Therefore, developing a lightweight filter that will reduce the number of features to be processed will in turn reduce the computational overhead; thus making the training phase of the SVM approach a viable solution for N-gram analysis where large feature sets are generated. Fig. 1 illustrates an overview of the experimental approach taken in this paper. The programs under investigation are run in a test environment with a debug tool monitoring the runtime opcodes. After completion, the data is parsed into opcode histograms and after some conditioning the dataset is passed to the SVM to construct a reference model. The reference model is constructed by configuring the SVM to perform an exhaustive search by traversing through all the features, searching for those opcodes that have a positive impact on the classification of benign and malicious software. To evaluate the various filtering algorithms, each filter processes the original dataset in an attempt to reproduce the same reference model produced by the SVM.

4. Dataset Creation

Operational Codes (Opcodes) are machine language instructions that perform CPU operations on operands such as arithmetic, memory/data manipulation, logical operations and program flow control. created a dataset of malicious and benign executables for the Windows operating system, the system most commonly used and attacked today. This malicious and benign file collection was previously used. Acquired some malicious files from the VX Heaven website. To identify the files, used the Kaspersky antivirus. Benign files, including executable and DLL (Dynamic Linked Library) files, were gathered from machines running the Windows XP operating system on our campus. The benign set contained some files.

To ensure that Ollydbg tool correctly unpacked and ran the malware, samples were restricted to programs that ollydbg correctly identified as packed or encrypted. The malware samples were run for 3 minutes ensuring that not only the loading and unpacking phases were recorded but also that malicious activity occurred, i.e., pop-up, writing to the disk or registry files. While there are 344 Intel opcodes, only 149 different opcodes are recorded during the captured datasets for all programs traced during this experiment. The dataset is normalized by calculating the percentage density of opcodes rather than the absolute opcode count to remove time variance introduced by different run lengths of the various programs. The dataset is sorted into most commonly occurring opcodes as illustrated in Fig. 2. An initial assessment of the data shows two key properties a) The distribution of the various opcodes does not conform to any consistent distribution shape; rather opcode distribution varies greatly as illustrated by the difference between the *mov* and *ret* opcodes, described later in VI: "Area of Intersect". Therefore, no one data shape could be assumed and hence a nonparametric method should be used. b) The data values are a percentage of the opcodes within a particular program. For example, 0 means that the opcode does not occur within that program trace or 0.25 means that 25% of the program trace comprises of that opcode. To improve the performance of the SVM the data is linearly scaled.

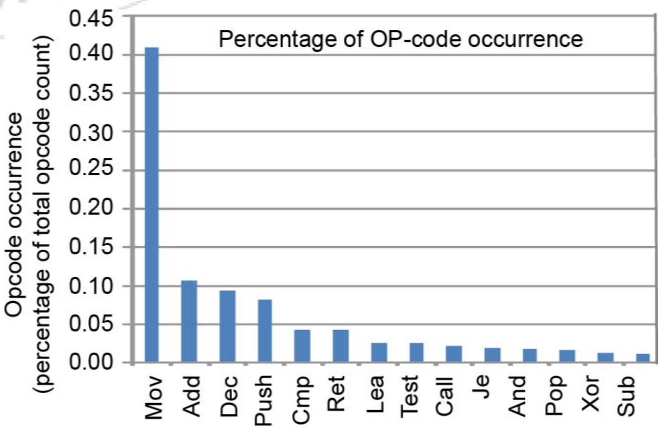

Figure 2: OPcode Percentage

5. Support Vector Machine

SVM classifiers consist of a hyperplane dividing a *n* dimensional space based representation of the data into two regions. This hyperplane is the one that maximizes the *margin* between the two regions or classes (in our case, malware or benign software). Maximal margin is defined by the largest distance between the examples of the two classes computed from the distance between the closest instances of both classes (called supporting vectors machine).Support Vector Machine (SVM) is a technique used for data classification and was introduced by Boser *et al.* in 1992 [10]and is categorized as a kernel method. The kernel method algorithm depends on dot-products function, which can be replaced by other kernel functions that map the data into a higher dimensional feature space.

This has two advantages: Firstly, the ability to generate a nonlinear decision plane and secondly, allows theuser to apply a classification to data that does not have an intuitive approach i.e., SVM training when the data has a nonregularor unknown distribution.

The dataset consists of 149 different opcodes, each having their own unique distribution characteristics and therefore a SVM is an appropriate choice. As mentioned earlier, the data is linearly scaled to improve the performance of the SVM. The main advantages of scaling are it avoids attributes with greater numeric ranges dominating those with smaller numeric ranges and it avoids numerical difficulties during the calculation as kernel values usually depend on the inner products of feature vectors, e.g., in the case of the linear kernel and the polynomial kernel, large attribute values might cause numerical problems. SVM is used to create a reference model to validate the filter experiments that are presented in the subsequence sections. The SVM is configured to traverse through the dataset searching for opcodes that have a positive impact on the classification of benign and malicious software. The search starts with six opcodes scanning across the complete data sequence for all unique permutations for that number of opcodes. The search is repeated for five opcodes and then four opcodes. An average of these results is sorted by most occurrences as illustrated in Fig. 3, which show the most important opcodes as chosen by the SVM. Only unique opcodes are selected for eachSVM classification test and no duplicates of repeated opcode patterns are processed.

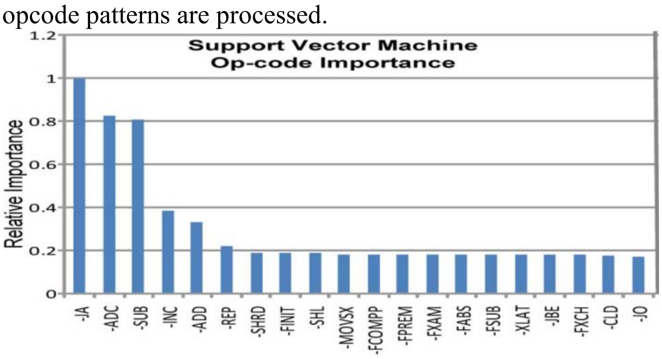

Figure 3: SVM OPcode Sensitvity

6. Reduction Approach

N-gram analysis presents a dimensionality problem in terms of the number of raw features produced and if left unfiltered would result in a high computation cost during the SVM training phase. To reduce this effort and narrow the area of search, this research aims to identify filters that can select the optimum features prior to feeding them to a SVM. The hypothesis is: Malware that employs evasion techniques will exhibit telltale signs in terms of run-time opcodes; such as a higher density of instructions that are commonly used in malware to evade detection and carry out malicious activity. Therefore filtering out less relevant opcodes and allowing the SVM to focus on a subset will result in a fast training phase. This section investigates two approaches to filtering irrelevant opcodes. Starting with an investigation into the "area of intersect" between benign and malicious distributions using Linear programming techniques and then concludes with an investigation into subspace analysis using Principle Component Analysis (PCA) and Eigenvectors.

6.1. Area of Intersect

Consider the simplistic characteristics of benign and malicious opcodes with a normal distribution as shown in Fig. 4. The plots are grouped into density curves for benign and malicious software of a single opcode. The horizontal axis relates to the percentage of a given program that is made up of a particular opcode and the vertical axis indicates the number of programs with that percentage of opcode. The key feature to note is the overlapping area of the two density curves. The greater the difference between the mean of the curves and narrower the standard deviation reduces the overlapping area and therefore reduces the interference and corresponding misclassification of the benign and malicious software. This implies that a simple analysis of low order statistics, such as calculating the product of the mean and the inverse of the standard deviation to determine the overlapping area might yield the best indicators (opcodes) of benign and malicious software. Hence, calculating the overlapping area for the density curves provides a numerical value and is shown in Fig. 4. These results need to be placed in a context that provides meaning in term of relative importance. Those opcodes chosen by the SVM as the reference model are highlighted.

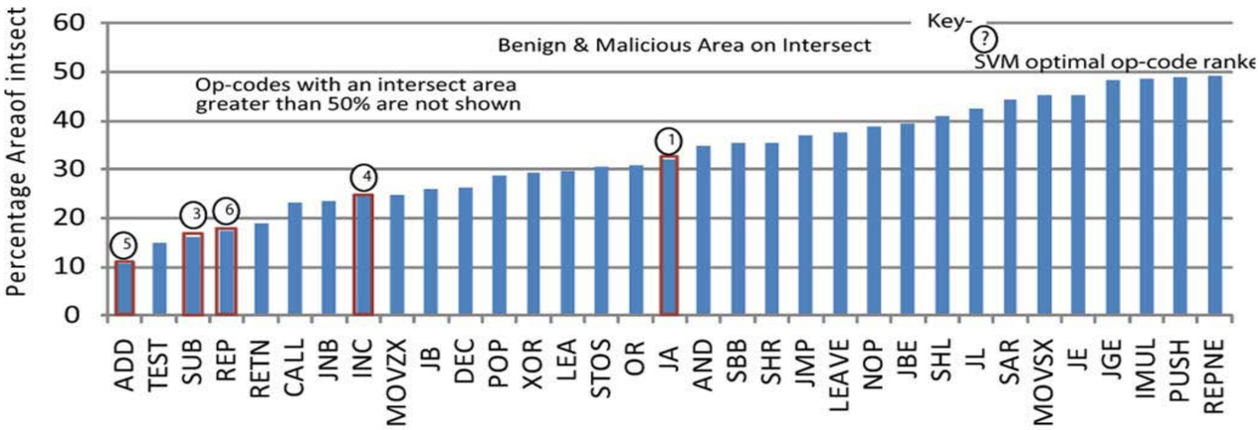

Figure 4: Area of Intersect

It can be seen that the opcodes with the least area of intersect correlate in part to the reference model. While this approach removes 75% of opcodes that provide no values an important opcode *(adc)* is removed therefore the "area of intersect" cannot be considered a useful tool for removing irrelevant opcodes. The SVM selected: *ja*, *adc*, *sub*, *inc, rep* and *add* as the reference model and as *adc* is filtered out by the "area of intersect" filter, which contradicts the hypothesis that opcodes with the least area of intersect make the best indicator of benign and malicious software. Two further points need to be considered. Firstly, the overall density of a particular opcode needs to be considered in the context of their area of intersect and its population as it needs to be significantly important to be considered as an indicator of benign and malicious software. Taking *ja* and *rep* opcodes (SVM selected range) as reference points, it can see from the data presented in Table II that the other opcodes relating to population and area of intersect fall within the characteristics of *ja* and *rep*. Therefore the area of intersect does not tell the full story as many other opcodes such as *ret, call*, etc. have lower area of intersect than *ja* and a population that lays between both *rep* and *ja*. In addition the „area intersect" filter removes the *adc* opcode. Low dimensional analysis does not consider covariance i.e., the relationship between the distributions of one opcode with that of another opcode. As shown in Fig. 4, it is not always the case that opcodes with a low area of intersect produce the best indicators of benign and malicious software. This requires a closer inspection of the opcode distribution curve to understand the characteristics that make the best indicators chosen by the SVM over the other opcodes that have similar area of intersect and population. Therefore further investigation is required and is carried out using Linear Programming (LP) to understand how the area under each curve is interpreted when a decision plane is applied. Linear programming is a technique that is applied to optimize a linear function when subject to linear equality and inequality constraints. LP can be applied to the classification of benign and malicious software. The data is in the form of a probability density curve. The horizontal axis represents the makeup of a program i.e., the opcode percentage that makes up a program and the vertical axis, representing the number of programs that have that percentage of opcodes. The probability density is based on a percentage of opcode counts obtained from traces during the execution of a program. The minimum value is 0 and the maximum is the percentage of the most occurring opcode within the captured dataset *(mov)*. Thus the maximum value is 0.4 (40%).

6.2 Subspace

An alternative approach to determine the importance of the individual opcodes, thereby ranking their usefulness as classification features, is to investigate the eigenvalues and eigenvectors in subspace. Principal Component Analysis (PCA) [11] is a transformation of the covariance matrix and it is defined as this is a technique used to compress data by mapping the data into a subspace while retaining most of the information/ variation in the data. It reduces the dimensionality by mapping the data into a subspace and finding a new set of variables (fewer variables) that represent the original data. These new variables are called principal components (PCs) and are uncorrelated and are ordered by their contribution (usefulness/eigenvalue) to the total information that each contain.

7. Conclusion

This paper is study of area intersecting to using of SVM training phase as a means of identifying malware. It shows that malware, that is packed/encrypted, can be detected using SVMs and by using the opcodes chosen by the SVM as a benchmark, determined a prefilter stage using eigenvectors that can reduce the feature set and therefore reduce the training effort. In this study first the identification of a high population opcode: *mov* that is not only is a poor indicator of benign/malicious software,but inhibits the ability to correctly classify software when usedit other opcodes such as *ja, adc, sub, inc, add* and *rep*. Secondly, a subset of opcodes can be used to detect malware. However, the SVM analysis demonstrates that *ja*, *adc* and *sub* are strong indicators of malware as they are four times more likely to be used in the correct classification of malware than the next most significant opcodes *(inc)*. Several opcodes have been identified as potential indicators of malware. Finally, using the principal component analysis of calculating eigenvector and eigenvalue from the dataset can safely remove irrelevant features.

References

[1] A. Lakhotia, E. U. Kumar, and M. Venable, "A method for detecting obfuscated calls in malicious binaries," *IEEE Trans. Software Eng.*, vol. 31, no. 11, pp. 955–968, Nov. 2005.
[2] D. Bilar, "Opcodes as predictor for malware," *Int. J. Electron. Security Digital Forensics*, vol. 1, no. 2, pp. 156–168, 2007.
[3] D. Bilar, "Callgraph properties of executables and generative mechanisms," *AI Commun., Special Issue on Network Anal. in Natural Sci.and Eng.*, vol. 20, no. 4, pp. 231–243, 2007.
[4] I. Santos, Y. K. Penya, J. Devesa, and P. G. Garcia, "N-grams-based file signatures for malware detection," *S3Lab, Deusto TechnologicalFound.*, 2009
[5] A. Shabtai, R. Moskovitch, C. Feher, S. Dolev, and Y. Elovici, "Detecting unknown malicious code by applying classification techniques on opcode patterns," *Security Informatics*, vol. 1, pp. 1–22, 2012.
[6] X. Chen, "Towards an understanding of anti-virtualization and antidebugging behavior in modern malware," *ICDSN Proc.*, pp. 177–186, 2008.
[7] R. Sekar, M. Bendre, D. Bollineni, and Bollineni, R. Needham and M. Abadi, Eds., "A fast automaton-based method for detecting anomalous program behaviors," in *Proc. 2001 IEEE Symp. Security and Privacy,IEEE Comput. Soc.*, Los Alamitos, CA, USA, 2001, pp. 144–155.
[8] C. Kolbitsch, P. M. Comparetti, C. Kruegel, E. Kirda, X. Zhou, and X. Wang, "Effective and efficient malware detection at the end host," in *Proc. 18th Usenix Security Symp.*, 2009, pp. 351–366.
[9] U. Bayer, I. Habibi, D. Balzarotti, E. Kirda, and C. Kruegel, "A view on current malware behaviors," in *Proc. 2nd USENIX Conf. on Large-Scale Exploits and Emergent Threats: Botnets, Spyware, Worms, andMore. USENIX Association*, Berkeley, CA, USA, 2009.
[10] B. E. Bernhard, G. M. Isabelle, and V. N. Vladimir, H. Haussler, Ed., "A training algorithm for optimal margin classifiers," in *Proc. 5th Ann.ACM Workshop on COLT ACM Press*, Pittsburgh, PA, USA, 1992, pp. 144–152.
[11] B. Schölkopf, A. Smola, and K.-R. Müller, "Kernel principal component analysis," *Artificial Neural Networks—ICANN'97 Lecture Notesin Comput. Sci.*, vol. 1327, pp. :583–588, 1997.