Idea: Opcode-sequence-based Malware Detection

Abstract. Malware is every malicious code that has the potential to harm any computer or network. The amount of malware is increasing faster every year and poses a serious security threat. Hence, malware detection has become a critical topic in computer security. Currently, signature-based detection is the most extended method within commercial antivirus. Although this method is still used on most popular commercial computer antivirus software, it can only achieve detection once the virus has already caused damage and it is registered. Therefore, it fails to detect new variations of known malware. In this paper, we propose a new method to detect variants of known malware families. This method is based on the frequency of appearance of opcode sequences. Furthermore, we describe a method to mine the relevance of each opcode and, thereby, weigh each opcode sequence frequency. We show that this method provides an effective way to detect variants of known malware families.

Key words: malware detection, computer security, machine learning

1 Introduction

Malware (or malicious software) is every computer software that has harmful intentions, such as viruses, Trojan horses, spyware or Internet worms. The amount, power and variety of malware increases every year as well as its ability to avoid all kind of security barriers [1] due to, among other reasons, the growth of Internet.

Furthermore, malware writers use code obfuscation techniques to disguise an already known security threat from classic syntactic malware detectors. These facts have led to a situation in which malware writers develop new viruses and different ways for hiding their code, while researchers design new tools and strategies to detect them [2].

Generally, the classic method to detect malware relies on a signature database [3] (i.e. list of signatures). An example of a signature is a sequence of bytes that is always present in a concrete malware file and within the files already infected by that malware. In order to determine a file signature for a new malware executable and to finally find a proper solution for it, specialists have to wait until that

new malware instance has damaged several computers or networks. In this way, malware is detected by comparing its bytes with that list of signatures. When a match is found the tested file will be identified as the malware instance it matches with. This approach has proved to be effective when the threats are known in beforehand, and it is the most extended solution within antivirus software.

Still, upon a new malware appearance and until the corresponding file signature is obtained, mutations (i.e. aforementioned obfuscated variants) of the original malware may be released in the meanwhile. Therefore, already mentioned classic signature-based malware detectors fail to detect those new variants [2].

Against this background we advance the state of art in two main ways. First, we address here a new method that is able to mine the relevance of an opcode (operational code) for detecting malicious behaviour. Specifically, we compute the frequency with which the opcode appears in a collection of malware and in a collection of benign software and, hereafter, we calculate a discrimination ratio based on statistics. In this way, we finally acquire a weight for each opcode. Second, we propose a new method to compute similarity between two executable files that relies on opcode sequence frequency. We weigh this opcode sequence frequency with the obtained opcode relevance to balance each sequence in the way how discriminant the composing opcodes are.

2 Mining opcode relevance

Opcodes (or operational codes) can act as a predictor for detecting obfuscated or metamorphic malware [4]. Some of the opcodes (i.e. `mov` or `push`), however, have a high frequency of appearance within malware and benign executables, therefore the resultant similarity degree (if based on opcode frequency) between two files can be somehow distorted. Hence, we propose a way to avoid this phenomenon and to give each opcode the relevance that it really has.

In this way, we have collected malware from the *VxHeavens* website [5] forming a malware dataset of 13189 malware executables. This dataset contains only PE executable files, and, more accurately, it is made up of different kind of malicious software (e.g. computer viruses, Trojan horses, spyware, etc). For the benign software dataset, we have collected 13000 executables from our computers. This benign dataset includes, for instance, word-processors, drawing tools, windows games, internet browsers, pdf viewers and so on.

We accomplish the following steps for computing the relevance of each opcode. First, we disassemble the executables. In this step, we have used *The New-Basic Assembler* [6] as the main tool for obtaining the assembly files. Second, using the generated assembly files, we have built an opcode profile file. Specifically, this file contains a list with the operational code and the un-normalized frequency within both datasets (i.e. benign software dataset and malicious software dataset). Finally, we compute the relevance of each opcode based on the frequency with which it appears in both datasets. To this extent, we use *Mutual Information* [7], $I(X;Y) = \sum_{y \epsilon Y} \sum_{x \epsilon X} p(x,y) \log \left(\frac{p(x,y)}{p(x) \cdot p(y)} \right)$. Mutual information is a measure that indicates how statistically dependant two variables are.

In our particular case, we define the two variables as each opcode frequency and whether the instance is malware. In this way, X is the opcode frequency and Y is the class of the file (i.e. malware or benign software), $p(x, y)$ is the joint probability distribution function of X and Y, $p(x)$ and $p(y)$ are the marginal probability distribution functions of X and Y.

Furthermore, once we computed the mutual information between each opcode and the executable class (malware of benign software) and we sorted them, we created an opcode relevance file. Thereby, this list of opcode relevance can help us to achieve a more accurate detection of malware variations since we are able to weigh the similarity function using these calculated opcode relevance and reducing the noise that irrelevant opcodes can produce.

3 Malware detection method

In order to detect both malware variants we extract the *opcode sequences* and their frequency of appearance. More accurately, we define a program ρ as a sequence of instructions I where $\rho = (I_1, I_2, ..., I_{n-1}, I_n)$. An instruction is composed by an operational code (*opcode*) and a parameter or list of parameters. In this way, we assume that a program is made up of opcodes. These opcodes can gather into several blocks that we call *opcode sequences*.

More accurately, we assume a program ρ as a set of ordered opcodes o, $\rho = (o_1, o_2, o_3, o_4, ..., o_{n-1}, o_n)$, where n is the number of instructions I of the program ρ. A subgroup of opcodes is defined as an opcode sequence os where $os \subseteq \rho$, and it is made up of opcodes o, $os = (o_1, o_2, o_3, ..., o_{m-1}, o_m)$, where m is the length of the sequence of opcodes os.

First of all, we choose the length of *opcode sequences*. Afterwards, we compute the frequency of appearance of each opcode sequence. Specifically, we use *term frequency* [8], $tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}}$, that is a weight widely used in information retrieval. More accurately, $n_{i,j}$ is the number of times the term $t_{i,j}$ (in our case opcode sequence) appears in a document d, and $\sum_k n_{k,j}$ is the total number of terms in the document d (in our case the total number of possible opcode sequences).

Further, we compute this measure for every possible opcode sequence of a fixed length n, acquiring by doing so, a vector $\overrightarrow{v}$ made up of frequencies of opcode sequences $S = (o_1, o_2, o_3, ..., o_{n-1}, o_n)$. We weigh the frequency of appearance of this opcode sequence using the weights described in section 2. To this extent, we define *weighted term frequency* (*wtf*) as the result of weighting the relevance of each opcode when calculating the *term frequency*. Specifically, we compute it as the result of multiplying *term frequency* by the calculated weight of every opcode in the sequence. In this way, $weight(o)$ is the calculated weight for the opcode o and $tf_{i,j}$ is the *term frequency measure* for the given opcode sequence, $wtf_{i,j} = tf_{i,j} \cdot \prod_{o \epsilon S} \frac{weight(o)}{100}$. Once we have calculated the *weighted term frequency*, we have the vector of *weighted opcode sequence frequencies*. $\overrightarrow{v} = (wtf_1, wtf_2, wtf_3, ..., wtf_{n-1}, wtf_n)$.

We have focused on detecting known malware variants in this method. In this way, what we want to provide is a similarity measure between two files. Once we extract the opcode sequences that will act as features, we have a proper representation of the files as two input vectors $\overrightarrow{v}$ and $\overrightarrow{u}$ of *opcode sequences*. Hereafter, we can calculate a similarity measure between those two vectors. In this way, we use *cosine similarity*, $sim(\overrightarrow{v}, \overrightarrow{u}) = \cos(\theta) = \frac{\overrightarrow{v} \cdot \overrightarrow{u}}{||\overrightarrow{v}|| \cdot ||\overrightarrow{u}||}$ [9]. Therefore, we think that this measure will give a high result when two versions of the same malware instance are compared.

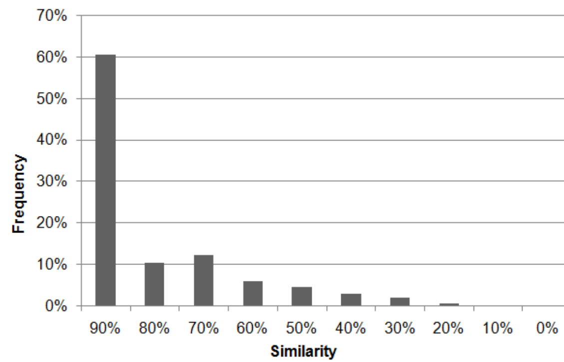

Fig. 1. A histogram showing the obtained results in terms of frequency of similarity ratio for the comparison of malware instances with their variants for n=1

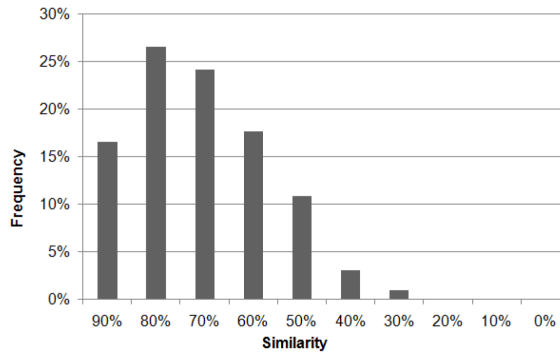

Fig. 2. A histogram showing the obtained results in terms of frequency of similarity ratio for the comparison of malware instances with benign executables for n=1

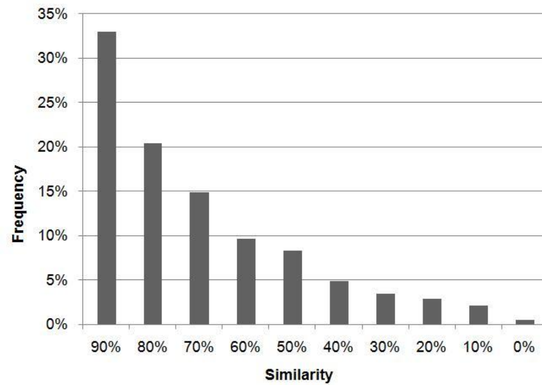

Fig. 3. A histogram showing the obtained results in terms of frequency of similarity ratio for the comparison of malware instances with their variants for n=2

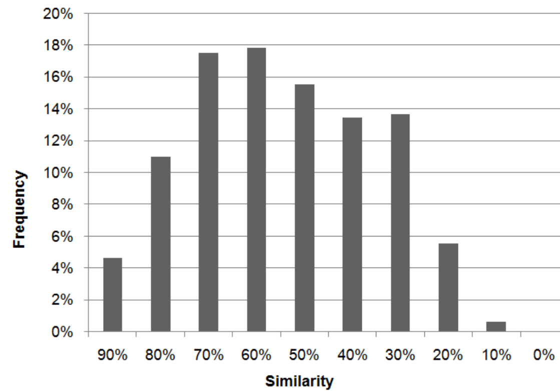

Fig. 4. A histogram showing the obtained results in terms of frequency of similarity ratio for the comparison of malware instances with benign executables for n=2

4 Experimental Results

For the following experiment, we have used two different datasets for testing the system: a malware dataset and a benign software one. First, we downloaded a big malware collection from VxHeavens [5] website conformed by different malicious code such as trojan horses, virus or worms. Specifically, we have used the next malware families: Agobot, Bifrose, Kelvir, Netsky, Opanki and Protoride.

We have extracted the *opcode sequences* of a fixed length (n) with $n = 1$ and $n = 2$ for each malware and some of its variants. Moreover, we have followed the same procedure for the benign software dataset. Hereafter, we have computed the *cosine similarity* between each malware and its set of variants. Further, we have computed the similarity of the malware instance with the whole benign software dataset.

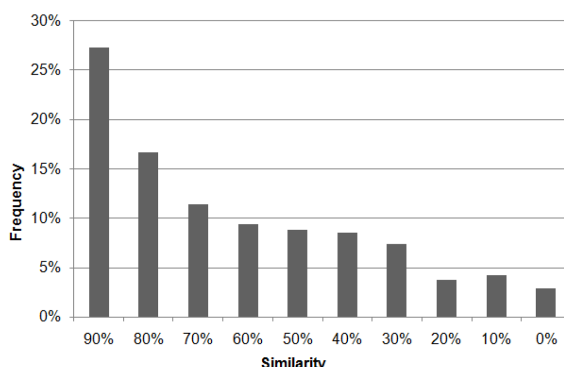

Fig. 5. A histogram showing the obtained results in terms of frequency of similarity ratio for the comparison of malware instances with their variants for n=1 and n=2 combined

Specifically, we have performed this process for every malware executable file within the dataset. For each malware family, we have randomly chosen one of its variants as the known instance and we have computed the cosine similarity between this variant and the other variants of that specific malware family. Moreover, we have performed the same procedure with a set of benign software in order to test the appearance of false positives.

Fig. 1 shows the obtained results of the comparison of malware families and their variants for an opcode sequence length n of 1. In this way, nearly every malware variant achieved a similarity degree between 90% and 100%. Still, the results obtained when comparing with the benign dataset (see Fig. 2) show that the similarity degree is too high, thus, this opcode sequence length seems to be not appropriate.

For an opcode sequence length of 2, the obtained results in terms of malware variant detection(see Fig. 3) show that the similarity degrees are more distributed in frequencies, however, the majority of the variants achieved a relatively high results. In addition, Fig. 4 shows the obtained results for the benign dataset. In this way, the results are better for this opcode sequence length, being more frequent the low similarity ratios.

Summarizing, on one hand, for the obtained results in terms of similarity degree for malware variant detection, the most frequent similarity degree is in the 90-100% interval. Moreover, the similarity degree frequency decreases and so the frequency does. Therefore, this method will be able to detect reliably a high number of malware variants after selecting the appropriate threshold of similarity ratio for declaring an executable as malware variant. Nevertheless, some of the executables were packed and, thereby, there are several malware variants that when computing the similarity degree did not achieve a high similarity degree.

Still, the similarity degrees between the two kind of sets (i.e. malware variants and benign software) are not different enough. Therefore, we decided to perform

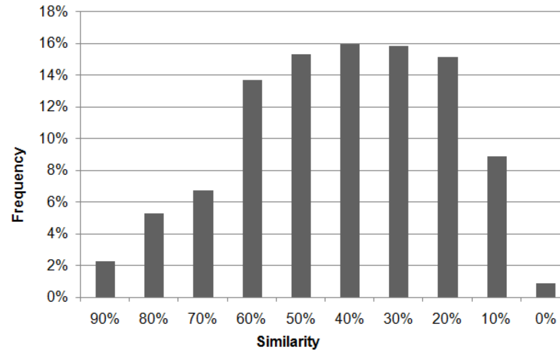

Fig. 6. A histogram showing the obtained results in terms of frequency of similarity ratio for the comparison of malware instances with benign executables for n=1 and n=2 combined

another experiment where the different opcode sequence lengths are combined ($n = 1$ and $n = 2$). Figures 5 and 6 show the obtained results. In this way, the malware variant similarity degrees remained quite high whilst the benign similarity degrees scrolled to lower results. On the other hand, for the obtained results in terms of similarity degree when comparing the malware instances with the benign dataset, as one may think in beforehand, the behaviour of the system yields to be nearly the opposite than when comparing it with its variants. In this way, the system achieved low similarity degrees in the majority of the cases of the benign dataset. Hence, if we select a threshold that allows us to detect the most number of malware variants as possible whilst the number of false positives is kept to 0, our method renders as a very useful tool to detect malware variants.

5 Related Work

There has been a great concern regarding malware detection in the last years. Generally, we can classify malware detection in *static* or *dynamic* approaches. Static detectors obtain features for further analysis without executing them since dynamic detectors execute malware in a contained environment.

In this way, static analysis for malware detection can be focused on the binary executables [10] or in source code [11] like the method proposed in this paper.

With regard to the binary analysis of the executables, there has been an hectic activity around the use of *machine-learning* techniques over byte-sequences. The first attempt of using non-overlapping sequence of bytes of a given length n as features to train a *machine-learning* classifier was proposed by Schulz et al. [12]. In that approach the authors proposed a method using the printable ASCII strings of the binary, tri-grams of bytes, the list of imported dynamically linked libraries (DLL), the list of DLL functions imported and the number of functions for each DLL. They applied multiple learning algorithms showing that multi-

Naïve Bayes perform the best. Kolter et al. [13] improved the results obtained by Schulz et al. using n-grams (overlapping byte sequences) instead of non-overlapping sequences. Their method used several algorithms and the best results were achieved by a boosted decision tree. In a similar vein, a lot of work has been made over n-gram distributions of byte sequences and machine-learning [14]. Still, most of the features they used for the training of the classifiers can be changed easily by simply changing the compiler since they focus on byte distributions.

Moreover, several approaches have been based in the so-called *Control Flow Graph Analysis*. In this way, it is worth to mention the work of Christodorescu and Jha [2] that proposed a method based of *Control Flow Analysis* to handle obfuscations in malicious software. Lately, Christodorescu et. at. [15] improved the previous work including semantic-templates of malicious specifications. Nevertheless, the time resources they consume render them as not already full prepared to be adopted for antivirus vendors, although *Control Flow Analysis* techniques have proved to obtain some very valuable information of malicious behaviours.

Dynamic analysis for malware detection, as aforementioned, runs a program in a contained environment and collects information about it. Despite they are limited by one execution flow, they can overcome the main issue of static analysis: be sure that the code that will be executed is the one that is being analysed [16]. Therefore, these methods do not have to face obfuscations or in-memory mutation [17]. In this way, the safe environment can be based on a virtual machine [18] or based on DLL Injection and API Hooking [19].

6 Conclusions and future work

Malware detection has risen to become a topic of research and concern due to its increasing growth in past years. The classic signature methods that antivirus vendors have been using are no longer effective since the increasing number of new malware renders them unuseful. Therefore, this technique has to be complemented with more complex methods that provide detection of malware variants, in an effort of detecting more malware instances with a single *signature*.

In this paper, we proposed a method detecting malware variants that relied in opcodes sequences in order to construct a vector representation of the executables. In this way, based upon some length sequences, the system was able to detect the malicious behaviour of malware variants. Specifically, experiments have shown the following abilities of the system. First, the system was able to identify malware variants. Second, it was able to distinguish benign executables.

The future development of this malware detection system is oriented in three main directions. First, we will focus on facing packed executables using a hybrid dynamic-static approach. Second, we will expand the used features using even longer sequences and more information like system calls. Finally, we will perform experiments with a larger malware dataset.