Copyright © IEEE.
Citation for the published paper:

This material is posted here with permission of the IEEE. Such permission of the IEEE does not in any way imply IEEE endorsement of any of BTH's products or services. Internal or personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution must be obtained from the IEEE by sending a blank email message to pubs-permissions@ieee.org.

By choosing to view this document, you agree to all provisions of the copyright laws protecting it.

Accurate Adware Detection using Opcode Sequence Extraction

Raja Khurram Shahzad

School of Computing
Blekinge Institute of Technology
SE-371 79 Karlskrona, Sweden
rks@bth.se

Niklas Lavesson

School of Computing
Blekinge Institute of Technology
SE-371 79 Karlskrona, Sweden
niklas.lavesson@bth.se

Henric Johnson

School of Computing
Blekinge Institute of Technology
SE-371 79 Karlskrona, Sweden
henric.johnson@bth.se

Abstract — Adware represents a possible threat to the security and privacy of computer users. Traditional signature-based and heuristic-based methods have not been proven to be successful at detecting this type of software. This paper presents an adware detection approach based on the application of data mining on disassembled code. The main contributions of the paper is a large publicly available adware data set, an accurate adware detection algorithm, and an extensive empirical evaluation of several candidate machine learning techniques that can be used in conjunction with the algorithm. We have extracted sequences of opcodes from adware and benign software and we have then applied feature selection, using different configurations, to obtain 63 data sets. Six data mining algorithms have been evaluated on these data sets in order to find an efficient and accurate detector. Our experimental results show that the proposed approach can be used to accurately detect both novel and known adware instances even though the binary difference between adware and legitimate software is usually small.

Keywords—Data Mining; Adware Detection; Binary Classification; Static Analysis; Disassembly; Instruction Sequences

I. INTRODUCTION

The aim of this study is to investigate adware detection and to develop an algorithm that accurately detects known and unknown adware instances. Adware may be defined as software that is installed on the client machine with the objective of displaying ads for the user of that machine [1]. Basic adware may also be bundled with extra functionality or software to invade the privacy of a user by monitoring his or her surfing activities or preferences in order to display related pop-up or pop-under advertisements. However, advanced adware may, for example: read data from locally stored files, collect surfing or chat related information, and even create remote connections for transferring and installing software in the future by making a system vulnerable and compromised [1]. These and similar capabilities may even turn the adware into a spyware or some other type of malicious software (malware). Arguably, adware compromises the confidentiality, and in some cases also the integrity and availability of computer systems. Analogously to computer viruses, which infect executable computer files, adware may be installed automatically when the user visits infected websites, installs freeware or shareware or when the user tries to open infected E-mail attachments [2]. The presence of adware is often mentioned in the End User License Agreement (EULA) but in a manner, which makes it

difficult for the average users to comprehend or even notice [3]. As a consequence, the users' informed consent is thus not obtained. Because of its presence in the EULA, adware vendors often claim that their software should be regarded as benign [3]. Such claims along with the differences in policies and regulations decided upon by different countries place adware in a grey zone in terms of legal status.

The adware problem is growing continuously due to the profound monetary gains for adware developers [1][4]. The user's awareness about adware and its potential consequences is generally considered to be low [2]. Currently, the major commercial antivirus tools try to detect instances of adware by relying on static or dynamic analysis such as signature-based and heuristic approaches (which were developed for detection of viruses). These techniques have a deficiency in detecting unknown or new instances and can be bypassed in different ways [5]. Two popular commercial tools for adware detection are SpyBot [6] and AdAware [7], which rely on signature based approach. Hence they require frequent update of their signature database and can detect only known instances.

Consequently, in this paper, we present a (static) detection method based on data mining. We have proposed an automated means for extracting instruction sequences (ISes) from adware and benign files in order to capture the behavior of the corresponding software. Our method extracts the operation code (opcode) from each instruction and then produces a data set in which each instance is described by sequences of opcodes. As the remainder of this paper will show, our approach is feasible for detecting adware despite the fact that the binary files of this type of software are sometimes quite similar to legitimate software.

A. Aim and Scope

In this paper, we present the results from an experimental study of adware detection. The aim is to determine the success of using data mining techniques for the detection of unseen and new instances of adware. Additionally, we investigate the relationship between opcode n -gram size and the number of features required to generate accurate detection models. Our hypothesis is that: it is possible to find a balance between the size of n -grams (that are used to represent opcode sequences) and the number of features (the number of n -grams) that yields a model of reasonable classification performance.

II. BACKGROUND

Adware is different from other malware since it may be installed with or without the consent of the user [8]. Users

may accept its presence knowingly for using freeware software or unknowingly when it is obfuscated in the EULA. The user may also be fooled into installing adware when trying to install other software or the installation of adware may be carried out as a background task without any human interaction at all [8]. Thus, it is important to be able to automatically detect adware. As mentioned earlier, traditional detection techniques, i.e., signature-based and heuristic methods have a deficiency in detecting novel instances of traditional malware, spyware and adware. In the signature-based technique, specific features or unique strings are extracted from binaries, which are later used for detection of malware. However, a copy of the malware is required to extract and develop a signature for detection purposes. Due to this fact, signature-based techniques are usually not capable of detecting novel (unseen) instances of malware. In the heuristic technique, human experts define rules for detecting behavioral patterns for malware detection. This technique is capable of detecting novel instances albeit with limited capacity and may be prone to false alarms.

A. Data Mining-based Detection

To overcome the aforementioned deficiency in detection techniques, Machine Learning (ML) methods, as an alternative approach, were applied for malware detection in 2001 [9]. Since then, different studies have been conducted for detection of traditional malware such as viruses, worms, and so forth, by applying ML and Data Mining (DM) technologies. DM helps in analyzing the data, with automated statistical analysis techniques, by identifying meaningful patterns or correlations. The results from this analysis can be summarized into useful information and can be used for prediction [10]. ML algorithms are used for detecting patterns or relations in data, which are further used to develop a classifier or a regression function [10].

For DM purposes, researchers have prepared their experimental data sets either by using different representations of binary files or by extracting a certain type of features that is present in the files. A binary file may be converted into hexadecimal code, binary code or ISeS as a means for representation. Moreover, these representations may be further used to create n -grams, which are fixed-size strings. Other features that are commonly present in files are printable text strings or calls to an application-programming interface. The use of opcodes as an alternative representation has also been suggested in [11]. An opcode is a part of the instruction for an operation in machine language. It may or may not include one or more operands for performing an operation such as an arithmetical operation or transferring program control.

When the data set is prepared for machine learning classification tasks, a class imbalance problem may arise. Typically, the imbalance problem occurs in a data set when one class has significantly more instances in comparison to another class or other classes. Due to this problem, the generated classifier tends to misclassify instances of the least represented class(es) and thus the problem may result in degradation of classification performance. Therefore, it is necessary to address the imbalance problem during data set

preparation. One approach is of course to try to ensure that all classes are equally represented. This approach, however, turns out to be practically impossible to adopt in many real-world problems since there usually is a great shortage of data instances of certain classes.

B. Feature Selection

Another important task when preparing the data set is to reduce the data set complexity while maintaining or improving performance of the classification model. For this purpose, the most common approach is to apply a feature selection algorithm. The objective of feature selection is basically to apply a feature quality measure to prioritize the available features and then keep only the best features from the prioritized list. In the information retrieval domain, the bag-of-words model (in which the logical order of words has no importance) performs better than other models in representing text documents [12].

Different feature selection measures, such as Document Frequency, Gain Ratio, and Fisher Score, are commonly used for obtaining reduced data sets [10]. Categorical Proportional Difference (CPD) is a relatively new addition in the feature selection algorithm family for text classification tasks [13]. The experiments have shown that CPD outperforms common feature selection methods such as chi-square and information gain. CPD represents a measure of the degree to which a word contributes to differentiating a specific class from others [13]. The possible value of CPD is within the interval of -1 and 1. A CPD value close to -1 indicates that a word to large extent occurs in an equal number of instances in all classes and a value in proximity of 1 indicates that a word occurs only in one class. Given that A is the number of times word w and class c occur together and B is the number of times word w occurs without class c , then we may define CPD for a particular word w and class c as follows:

$$CPD(w, c) = \frac{A - B}{A + B} \quad (1)$$

The reduced feature sets can then be used for data mining purposes and can be used as input to learning algorithms. Many types of learning algorithms are available. Therefore, it is important to choose suitable algorithms with respect to the problem at hand.

III. RELATED WORK

Due to legal issues and lawsuits from adware vendors, anti-virus vendors are hesitant to classify any software as adware [4]. Therefore, we have not been able to find any specific approaches for detecting adware. However, we argue that it is important to detect adware to let users exercise the right to make an informed choice about the software they install. In previous work, opcodes have been used for detection of different variants of worms and some types of spyware [14]. From the original malware, opcodes were extracted and paired with labels. With these pairs, researchers developed signatures, which were matched with pairs of variants of malware. A three-stage scanning was performed, which was successful in detecting the different

variants. In another study, an attempt to detect unknown malware was made by extracting opcodes from malware and then converting them into sequences of opcodes [5]. In their experiment, the researchers applied three classifiers out of which two were boosted and achieved 93 per cent accuracy. In yet another study, variable length instruction sequences were used as a representation for the detection of worms. This time, researchers applied Bagging and were successful in achieving 96 per cent accuracy [15]. In an attempt to detect spyware, n -grams of hexadecimal representation were used as features [16]. This attempt was successful in obtaining 90.5 per cent accuracy. Most of the reviewed detection experiments on traditional malware were performed using hexadecimal n -grams as features. Only a few researchers seem to have considered opcodes as features and then only from the code segment of the studied files [5][15]. The files in these experiments were disassembled using commercial disassemblers to obtain the IS. Moreover, most of these studies have not considered the class imbalance problem, which may lead to unnecessarily high rates of misclassification. In conclusion, most of the work concerning malware detection focuses on viruses, worms, and trojans. It is not clear whether the same type of detection methods would be successful when dealing with adware, which is more similar to legitimate software than such types of malware. Nevertheless, adware represents a serious threat to privacy and, as such, the research on adware is important, especially in terms of detection approaches.

IV. METHOD

We propose a static DM-based analysis method, which includes disassembling the adware and benign files during preprocessing. We aim to evaluate our proposed method for detecting unknown and new instances as well as existing instances of adware.

A. Overview

The focus of our analysis is Windows-based executable files, since the Windows operating system has been considered to be more vulnerable to adware as opposed to, say, Unix-based operating systems. When any software is disassembled, the generated output contains text, which may represent hexadecimal dumps, binary dumps or ISes. We argue that text categorization techniques can therefore be applied on disassembled output to distinguish between adware and benign software. Thus, we disassemble executable files to obtain ISes and then extract opcodes from those instruction sequences. The extracted opcodes are converted into a vocabulary data set. Each word in the vocabulary data set is an n -gram of a specific size, which represents a feature. Although the size of each word in a particular vocabulary set is fixed, the length is variable. For example, if we observe a data set where the n -gram size is 4 then each word is constructed by joining four opcodes, where each opcode may have a different length. We use Term Frequency - Inverse Document Frequency (*tf-idf*) to measure the significance of every word in order to extract significant features. The generated data is converted into the Attribute-Relation File Format (ARFF) data set file format.

The ARFF files are further processed with CPD to obtain feature-reduced data sets, which are used as input to the Waikato Environment for Knowledge Analysis (Weka) [10] to perform the classification experiments. Weka is a suite that includes a large set of machine learning algorithms as well as analysis tools for solving data mining problems.

B. Data Set Generation

No public data set is available for use in adware detection experiments as opposed to what is available for, e.g., virus and intrusion detection. Therefore, we have created a data set with 600 files out of which 300 files are adware and 300 files are considered as benign. All files represent executable binaries for the Windows operating system. The benign files stem from two sources: a copy of the Windows XP operating system was installed on a clean computer to obtain benign files, e.g., small programs such as notepad, paint, clock, and so forth. Second, to represent files available on the Internet, programs were downloaded from download.com [17]. This website claims to provide spyware free software; however, when downloaded data set was scanned with a commercial version of the F-Secure Client Security software [18], some instances were found infected by so-called riskware. The infected instances were replaced by other benign files. Adware files were obtained from a malware database [7].

C. File Size and Data Size Analysis

File size and data size analysis has to be performed to investigate potential imbalance problems. When adware and benign programs were collected, it was observed that the mean file size of the two software program groups was quite different. Therefore, it was necessary to avoid an unbalanced number of instructions since different file sizes may produce a varying number of ISes. This may further lead to a class-based difference in the generated vocabulary, which in turn may lead to an imbalance problem. Therefore, we decided to restrict the maximum file size to 512 KB for this particular study. It was also considered that the total number of files and the total size of these files should be approximately equal in both data sets.

D. Disassembly and Opcode Extraction

The collected programs were disassembled to get instruction sequences in assembly Language. This step was performed using the Netwide disassembler (Ndisasm), which is commonly available for UNIX/Linux operating systems [19]. Ndisasm disassembles binary files without understanding (correctly processing) object file formats. The generated output contained the memory address of the instruction, the byte-based location in the file and the instruction itself, i.e., the combination of opcode and operands. An application was further developed to extract the opcodes from the disassembled file. We did not just include the opcodes from the code segment of the files but instead used opcodes extracted from any segment in the file.

E. Parsing and n -Gram Size

The extracted opcode data were processed further with a parser that tokenized the data to produce vocabulary/words

as per a selected n -gram size. In a previous research study, an n -gram size of 4 or 5 yielded promising results for the hexadecimal representation [20][21]. In another study, an n -gram size of 2 for opcode representation yielded the best performance [5]. Therefore, we decided to use n -grams of sizes ranging from 2 to 8 while considering 4 and 5 as intermediary values. The purpose of selecting this range was to evaluate n -gram sizes in proximity of what has been considered adequate settings in previous research. We created seven master data sets using these n -gram sizes. Each row in these data sets represented one word, which is an n -gram of a specific size. Thus we obtained features of n -grams with seven different n -gram sizes. These data sets contain the features with different number of occurrences in each class. We also calculated the number of unique features in one class. Table I presents the vocabulary statistics for each class and data set.

TABLE I. VOCABULARY STATISTICS

n	Adware		Benign		Final ($tf-idf$)
	Total	Unique	Total	Unique	
2	4497344	35666	4381315	25921	1236
3	2998173	452915	2920818	228780	1340
4	2248586	876451	2190581	440580	1413
5	1798843	881768	1752439	565536	1518
6	1499012	804138	1460335	630851	1630
7	1284845	727570	1251705	656345	1676
8	1124215	660092	1095219	643148	1753

a. The n column shows the n -gram size
b. Final column presents the vocabulary obtained on the basis of $tf-idf$

F. Feature Selection

The main objective of our particular feature selection step was to obtain sets of features with a different amount of data that represents both adware and benign programs. The output obtained from the previous steps, contains huge vocabularies, which may lead to two problems i.e. ML algorithms may not process this huge vocabulary and all words in vocabulary do not provide valuable information for classification. We used $tf-idf$ for initial feature selection. The frequency n is number of times a word (or n -gram in our case) appears in single document, d_j . It is not feasible to use this frequency as basis for selection of words as documents may be of different length so that some words will be more frequent regardless of their actual importance. For normalization purposes, we use *Term Frequency* (tf), which gives a measure of the importance of a word (also known as term) in document d_j . This measure is obtained by dividing the frequency of a word, $n_{i,j}$, with the sum of all frequencies of all words in the document d_j . For obtaining the general importance of word in a document set, D , we use *Inverse Document Frequency*, idf . For obtaining idf , D is divided by the number of documents that include that particular word and then the logarithm of that value is taken. To get the final measure of a word and filter out common words, we use *Term Frequency - Inverse Document Frequency*. The $tf-idf$ of a word is obtained by multiplying tf and idf of that particular word. By using $tf-idf$, we obtained the final data sets. The total number of final words in every data set varies. In information retrieval, it is common to use a predefined

number of words obtained from $tf-idf$ (such as the top 1,000 words for both classes or each class). We argue that our problem is different from normal text classification in terms that when the n -gram size increases, the number of unique words in each class is also increased (see Table I). Therefore, we let the number of selected words depend on the data set in question instead of a predefined number. There is an additional benefit derived from this feature selection step: suppose a file in the benign data set may be infected with a zero-day threat or that the features extracted from some files are really part of the data segment instead of the code or images. For these cases, the corresponding features will be ignored due to their absence in other files of the same class.

Moreover, the output from the previous step was further processed using the CPD algorithm to create the final data sets. CPD has shown promising results in text classification, but has not been used previously for malware classification. We expected that the use of CPD would lead to better detection performance than other common feature selection methods. As the exact percentage of features to keep in order to yield optimal performance is not known beforehand, we chose to discard features in increments of 10 per cent for every generated data set. Nine final data sets for each n -gram size were created. These data sets can be downloaded from <http://www.bth.se/com/rks>

G. Data mining algorithms

Previous studies on similar problems are not conclusive regarding which learning algorithm generates the most accurate classifiers. In a number of studies of malware detection, Ripper (JRip), C4.5 Decision Tree (J48), Support Vector Machines (SMO), and Naive Bayes (NB) performed better than other algorithms. In a previous study of text categorization [23], k -nearest neighbor (IBk) outperformed NB and other algorithms. Based on previous research, we selected these algorithms as candidates and compared them against ZeroR as a baseline.

1) ZeroR

ZeroR is a simple, deterministic rule-based algorithm. ZeroR resembles as a random guesser, which could be used to model a user that makes an uninformed decision about software by always predicting the majority class [10]. This algorithm is frequently used as a baseline to measure the performance gain of other algorithms in classification against chance.

2) JRip

JRip is an implementation of the Ripper algorithm [22], which tries to generate an optimized rule set for classification. Rules are added on the basis of coverage (that is, how many data instances that are matched) and accuracy. Ripper includes intermediate and post pruning techniques to get increase the accuracy of the final rule set.

3) J48

J48 is a decision tree induction algorithm, extended from the ID3 algorithm, which uses the concept of information entropy [23]. Decision trees recursively partition instances from the root node to some leaf node and a tree is constructed.

TABLE II. AREA UNDER ROC CURVE VALUE FOR N-GRAM SIZE OF 4

n -Size = 4	Naive Bayes	SMO	IBk	J48	JRip
10%	0.820(0.059)	0.815(0.058)	0.884(0.047)	0.824(0.057)	0.817(0.061)
20%	0.752(0.066)	0.848(0.043)	0.886(0.043)	0.832(0.058)	0.818(0.064)
30%	0.879(0.046)	0.906(0.036)	0.920(0.032)	0.884(0.045)	0.889(0.048)
40%	0.901(0.039)	0.927(0.033)	0.926(0.033)	0.896(0.053)	0.906(0.036)
50%	0.892(0.036)	0.945(0.031)	0.934(0.031)	0.886(0.049)	0.903(0.045)
60%	0.863(0.042)	0.942(0.031)	0.945(0.024)	0.888(0.049)	0.906(0.041)
70%	0.838(0.042)	0.939(0.031)	0.949(0.024)	0.885(0.045)	0.901(0.036)
80%	0.828(0.044)	0.945(0.026)	0.935(0.029)	0.884(0.044)	0.898(0.043)
90%	0.822(0.046)	0.944(0.028)	0.934(0.031)	0.884(0.046)	0.911(0.035)

4) SMO

SMO is an implementation of the support vector machines (SVM) algorithm using Platt's sequential minimization optimization. During classification, SMO tries to find the optimal hyperplane, which maximizes the distance/margin between two classes thus defining the decision boundaries. It is used for classification and regression [24]. SMO has been generalized in order to be applicable for problems in which there are more classes than two.

5) Naive Bayes

Naive Bayes is based on Bayes' theorem and generates a probabilistic classifier with independence assumptions, i.e., the different features in the data set are assumed not to be dependent of each other [25]. Clearly, such an assumption is violated in most real-world data sets. Nevertheless, the Naive Bayes algorithm has proven to generate quite accurate classifiers for many problems.

6) IBk

IBk is an implementation of the k -nearest neighbor (k NN) algorithm, which computes the Euclidean distance between the instance to be classified and the instances included in the training set. Predictions from the neighbors is obtained and weighted according to their distance from the test instance. The majority class of the closest k neighbors is assigned to the new instance [26].

V. EVALUATION METRICS

We evaluated each learning algorithm by performing cross-validation tests. Confusion matrices were generated by using the responses from classifiers. The following four estimates defined the elements of such a matrix: True Positives (TP) represent the correctly identified adware programs. False Positives (FP) represent the incorrectly classified benign programs. True Negatives (TN) represent the correctly identified benign programs and False Negatives (FN) represent the incorrectly identified adware programs.

The performance of each classifier was evaluated using Detection Rate (DR), which is the percentage of correctly identified adware. False Alarm Rate (FAR), which is the percentage of wrongly identified benign programs and Accuracy (ACC), the percentage of correctly identified programs. We argue that, for our problem, False Negative Rate, which is the percentage of incorrectly identified adware programs, is more important than FAR. The last evaluation parameter was Area Under Receiver Operating

Characteristic Curve (AUC). AUC is essentially a single-point value derived from a ROC curve, which is commonly used when the performance of a classifier needs to be evaluated for the selection of a high proportion of positive instances in the data set [10]. Therefore it plots the DR on the x-axis in function of the FAR on the y-axis at different points. In many situations, ACC can be a reasonable estimator of predictive performance. However, AUC has the benefits of being independent of class distribution and cost [27]. In many real-world problems, the classes are not equally distributed and the cost of misclassifying one class may be different to that of misclassifying another. For such problems, the ACC metric is not a good measure of performance but it may be used as a complementary metric.

VI. EXPERIMENTAL PROCEDURE

To investigate our hypothesis that it is possible to find a suitable combination of n -gram size and the number of features to yield a model of reasonable classification performance, a comprehensive set of evaluation runs was designed. Our experiment used seven different sizes of n -grams to create data sets and for each specific n there were nine sub sets ranging from 10% features to 90% features. In total, we conducted 630 10-fold cross-validation (CV) tests for each classifier, which resulted in 3,780 runs in total. Default configurations were used for all algorithms. We used corrected paired t-test (confidence 0.05, two tailed) to compare each classifier with the base line classifier ZeroR.

VII. RESULTS

Most of the algorithms performed well when using an n -gram size of 4 and the 70% features data set. The results of all algorithms were compared with the results of ZeroR, which achieves an AUC score of 0.50 (random guessing). Figure 1 shows the comparison of all algorithms in terms of AUC score for n -gram size of 4 with 70% features data set. The AUC scores for IBk for aforementioned data set are presented in Table II. Considering AUC as the primary performance metric, the results clearly show that our proposed methodology is successful in detecting novel (unseen) instances of adware. IBk achieved the best result (AUC = 0.949, FNR = 0.022 and FAR = 0.115 with $n=4$ and 70% attributes kept). In terms of FNR on the 70% data set for different n -gram sizes, most of the algorithms achieved the highest FNR at n -gram size of 2. NB has shown high variance among all n -gram sizes for FNR and FAR with highest FNR value of 0.475 for n -gram size 2 and highest

FAR value of 0.335 for n -gram size of 3. All other algorithms gave their highest FNR and FAR for n -gram size of 2. The IBk achieved highest FAR i.e. 0.462 for n -gram size of 8.

Figure 1. AUC of 70% data set for all n -gram sizes

VIII. ANALYSIS

The results clearly show the possibility to detect adware using data mining on disassembled code and thus strengthen the validity of our hypothesis. The aim of this study is two-fold. Firstly, we need to evaluate our methodology for detection and secondly, we need to find a suitable combination of n -gram size with percentage of features. Results have shown that adware could be detected using n -grams of opcodes. We have used opcode sizes ranging from 2 to 8. For our experiments, we have not considered an n -gram size of 1 since it has been concluded in a previous study that sequences of opcodes are more suitable than a single opcode for representation [5]. We have considered the false negative rate (adware classified as benign) because this is more important for a user than the false alarm rate (benign classified as adware). We argue that, if a benign file is classified as adware it may not affect the system as much as if an adware application is classified as benign and thus installed on the system.

1) Algorithm Performance Analysis

The classifier generated from the IBk algorithm has shown the most promising results in terms of AUC and accuracy for an n -gram size of 4 especially for higher percentages of kept features. k NN and SVM are effective when the data are noisy. k NN has an advantage that its classification performance is refined incrementally when new training samples are introduced. J48 also has shown variance in results for smaller percentages of data. This may be attributed to the fact that for small data sets or in presence of noise, J48 is prone to overfitting the training data.

NB has not been successful in classification as compared with other classifiers. It is evident that, as n -gram size and percentage of data are increased, the performance of NB classifier is varying significantly. This may be because, as the n -gram size increases, the number of unique combinations of opcodes in each data set increases as shown in Table I. NB assigns probability to each feature. These unique combinations may be present in only a few instances and so the probability of occurrence is determined to be low in one class. However, this is not the case when using an n -gram size of 2 since the occurrence of any combination can be high.

For the studied problem, we may draw the conclusion that an n -gram size of 4 seems to be reasonable for good

detection. The reason for this could be that at this size each n -gram is representing combination of four instructions sequences which may be referring a function or interesting feature in the file. This is also easy to track this combination in the malware or benign files for further analysis.

2) State-of-the-Art

In a previous study on opcode-based malware detection [5], $n=2$ yielded the best results but we argue that such short combinations of opcodes may not indicate any important function or set of instructions in the files. Due to these reasons it may be difficult to perform analysis. In another study [15], Bagging was used in conjunction with the Random Forests algorithm. However the basis for this selection was not reported. These experiments were performed on worms and viruses and have shown promising results for detection but worms and viruses are quite different from adware in that they may exhibit clearly malicious routines; these routines can then easily be identified by human experts. But in the case of adware, the resemblance to benign software is greater. Normal characteristics of adware (such as: the displaying of ads in popup windows or the transferring of information over the network) are also present in several instances of legitimate software. Therefore, it is difficult for human experts to classify a piece of software as adware on the basis of such characteristics.

3) Opcode Analysis

We decided to use ISes rather than other common representations, such as: hexadecimal n -gram representations, printable strings, API calls, or messages because ISes include program control flow information. Moreover, a group of ISes may indicate an interesting function, which can be easily tracked back in the program for deeper analysis. In order to find interesting functions we analyzed models generated by SMO, JRip, NB and J48 for n -gram size of 4 with 70% features. We found that most of the features that were linked to adware by other models were not considered by J48 (e.g.: *pushcallsbbinc* and *incaddandinc*). This may be because J48 is considered unstable as a small variation in data set results in selection of different attributes, which affect the descendent sub trees.

4) Practical Considerations

DM techniques have performed well in detecting adware. But in the case of advanced adware with encrypted functionalities, the static analysis method used in this paper may not be successful albeit the presence of an encrypted segment can potentially be considered as an indication. It could be the case that a dynamic analysis approach has to be applied to detect these instances of advanced adware.

In terms of converting our approach into a practical solution for general users or experts, we argue that IBk represents a good choice of algorithm. IBk is the simplest algorithm with respect to working as it classifies an instance on basis of majority vote of its k nearest neighbors. The k is a small positive integer due to which the duration for training and building classifier from IBk was less than tree based and rule based algorithms where new trees or rules are required to generate or update previous. JRip algorithm was most expensive algorithm in terms of time consumed to train and

generate the model, due to which this may not be considered feasible option for users. J48 algorithm was better than JRip in terms of results and time consumed for training but still it was expensive than other classifiers, due to which it is also not suitable candidate. SMO was nearest to IBk, so it may be used as an alternative or to complement the results of IBk. Another alternative for adware detection may be combining DM techniques with EULA analyzer as many adware vendors mention the presence of their adware in EULA to avoid legal consequence. In this way, we argue that advanced adware with encrypted routines can also be detected.

IX. CONCLUSION AND FUTURE WORK

Many papers have been devoted to the study of detection approaches for malware such as viruses, worms, and trojans. However, less work has been done in the area of adware detection. We argue that this has little to do with the fact that adware is considered less harmful. Rather, it seems that the area of adware is avoided due to the fact that this type of software resides in a legal grey zone: some people regard adware as legitimate and others perceive adware as harmful. This paper considers the latter perception. We have presented a static file analysis method, based on operation code mining, for adware detection. A series of experiments with data sets generated using different n -gram sizes were performed. The experiments show promising results in terms of the area under the ROC curve (AUC) for the detection of novel instances of adware based on previously unseen examples, while maintaining a low false negative rate. The highest classification performance (AUC score of 0.949) was achieved by the k -nearest neighbor algorithm. Another conclusion inferred from these experiments is that, as the size of n -grams and the percentage of features are increased, the detection performance also increases. However, an n -gram size of 4 seems to represent a local optimum, at least for the studied algorithms. For future work, we plan to perform experiments on a larger collection of adware and benign files by introducing a hybrid identification method, which uses the combination of n -grams of opcodes and features extracted from EULAs. We plan to combine dynamic and static analysis techniques to be able to detect basic as well as advanced adware.

REFERENCES

- [1] G. Shaw, "Spyware & Adware: The Risks Facing Businesses," *Network Security*, vol. 2003, no. 9, pp. 12-14, Sep. 2003.
- [2] E. E. Schultz, "Pandora's Box: Spyware, Adware, Autoexecution, and NGSCB," *Computers & Security*, vol. 22, no. 5, pp. 366-367, Jul. 2003.
- [3] N. Lavesson, M. Boldt, P. Davidsson, and A. Jacobsson, "Learning to Detect Spyware Using End User License Agreements," *Knowledge and Information Systems*, vol. 26, no. 2, pp. 285-307, 2011.
- [4] J. Malcho, "Is There A Lawyer in the Lab?" in *Proceedings of the 19th Virus Bulletin International Conference*, 2009.
- [5] R. Moskovitch et al., "Unknown Malcode Detection using OPCODE Representation," in *Proceedings of the 1st European Conference on Intelligence and Security Informatics (EuroISI 2008)*, 2008, pp. 204-215.
- [6] "Spybot Search & Destroy." [Online]. Available: <http://www.safer-networking.org/en/home/index.html>. [Accessed: 29-May-2011].
- [7] Lavasoft AB, "Lavasoft." [Online]. Available: <http://www.lavasoft.com/>. [Accessed: 13-Jul-2010].
- [8] S. Gordon, "Fighting Spyware and Adware in the Enterprise," *Information Security Journal: A Global Perspective*, vol. 14, no. 3, pp. 14-17, 2005.
- [9] M. G. Schultz, E. Eskin, F. Zadok, and S. J. Stolfo, "Data Mining Methods for Detection of New Malicious Executables," in *Proceedings of the IEEE Symposium on Security and Privacy (S&P 2001)*, 2001, pp. 38-49.
- [10] I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques*, 2nd ed. Morgan Kaufmann, 2005.
- [11] S. Dolev and N. Tzachar, "Malware Signature Builder and Detection for Executable Code", Patent No. EP2189920. [Online]. Available: <http://www.freepatentsonline.com/EP2189920.html>. [Accessed: 21-Jan-2011].
- [12] S. Scott and S. Matwin, "Feature Engineering for Text Classification," in *Proceedings of the Sixteenth International Conference on Machine Learning (ICML 99)*, 1999, pp. 379-388.
- [13] M. Simeon and R. Hilderman, "Categorical Proportional Difference: A Feature Selection Method for Text Categorization," in *Proceedings of the Seventh Australasian Data Mining Conference (AusDM 2008)*, 2008, vol. 87, pp. 201-208.
- [14] A. Sulaiman, K. Ramamoorthy, S. Mukkamala, and A. H. Sung, "Disassembled Code Analyzer for Malware (DCAM)," in *Proceedings of the IEEE International Conference on Information Reuse and Integration (IRI-2005)*, 2005, pp. 398-403.
- [15] M. Siddiqui, W. Wang, and J. Lee, "Detecting Internet worms Using Data Mining Techniques," *Journal of Systemics, Cybernetics and Informatics*, vol. 6, no. 6, pp. 48-53.
- [16] R. K. Shahzad, S. I. Haider, and N. Lavesson, "Detection of Spyware by Mining Executable Files," in *Proceedings of the International Conference on Availability, Reliability, and Security (ARES 10)*, 2010, pp. 295-302.
- [17] CNET, "Free Software Downloads." [Online]. Available: <http://download.cnet.com/>. [Accessed: 02-Jan-2010].
- [18] F-Secure Corporation, "F-Secure - A Global IT Security & Antivirus Provider." [Online]. Available: <http://www.f-secure.com/>. [Accessed: 02-Oct-2010].
- [19] "The Netwide Assembler: NASM." [Online]. Available: <http://www.nasm.us/>. [Accessed: 13-Jul-2010].
- [20] O. Henchiri and N. Japkowicz, "A Feature Selection and Evaluation Scheme for Computer Virus Detection," in *Proceedings of the Sixth International Conference on Data Mining (ICDM 06)*, 2006, pp. 891-895.
- [21] Y. Elovici, A. Shabtai, R. Moskovitch, G. Tahan, and C. Glezer, "Applying Machine Learning Techniques for Detection of Malicious Code in Network Traffic," in *Proceedings of the 30th Annual German Conference on AI: Advances in Artificial Intelligence (KI 2007)*, 2007, pp. 44-50.
- [22] W. W. Cohen, "Learning Trees and Rules with Set-valued Features," in *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, 1996, pp. 709-716.
- [23] J. R. Quinlan, *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., 1993.
- [24] J. Platt, "Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines," *Technical Report MST-TR-98-14. Microsoft Research*, 1998.
- [25] C. Feng and D. Michie, "Machine Learning of Rules and Trees," in *Machine learning, neural and statistical classification*, Ellis Horwood, 1994, pp. 50-83.
- [26] T. Cover and P. Hart, "Nearest Neighbor Pattern Classification," *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 21-27, Jan. 1967.
- [27] F. J. Provost, T. Fawcett, and R. Kohavi, "The Case against Accuracy Estimation for Comparing Induction Algorithms," in *Proceedings of the Fifteenth International Conference on Machine Learning*, 1998, pp. 445-453.